UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Analizador de redes CAN

Proyecto Final de Carrera

Ingeniería Técnica en Informática de Sistemas

Autor: Andrés Martínez Mas

Director: José Carlos Campelo Rivadulla

3 de Septiembre de 2012

Resumen

El presente proyecto es una continuación del proyecto previo Interfaz USB-CAN. Se va a realizar una herramienta para poder monitorizar y analizar redes CAN desde un PC, a través de un bus series como es el USB.

En el primer proyecto se programó el microcontrolador C8051F500, para lo cual se utilizó el lenguaje de programación C adaptado, para sistemas empotrados. Este dispositivo una vez programado, es capaz de recoger todo el tráfico en la red CAN y trasmitirlo al PC.

En el actual proyecto se realiza un driver con el cual el PC puede reconocer el microcontrolador y de esta manera puedan comunicarse entre si. También se ha diseñado y programado una Interfaz Gráfica de Usuario (IGU). Esta es la encargada de comunicarse con el microcontrolador. A través de esta se recoge y visualiza todo el tráfico en la red CAN, dando la posibilidad de filtrar la información de varias maneras. Una vez recogidos los datos, existe la opción de guardado para su posterior análisis. La información mostrada se visualiza en el formato que sea necesario para el usuario. También es posible realizar esto con los mensajes de error.

Otra herramienta más de la que dispone la Interfaz es el seguimiento en tiempo real del estado del microcontrolador. Se muestra en todo momento el número de errores recibidos, el número de errores trasmitidos y el total de errores. También se visualiza el estado en el que se encuentra internamente el microcontrolador.

El programa es capaz de comunicarse con otros PC's a través de Ethernet, de esta manera se monitoriza la red CAN sin necesidad de estar conectado físicamente al microcontrolador.

A través de la interfaz se configuran todos los parámetros del microcontrolador. De esta manera se conecta el analizador a cualquier red CAN, adaptándose sin ningún problema.

Existe la posibilidad de inyectar paquetes dentro de la red CAN, los cuales estarán configurados como el usuario desee, gracias a las opciones que proporciona la interfaz. Se pueden configurar envíos de paquetes periódicos, a la vez que se visualizan estos y las consiguientes respuestas.

Otra opción más de la que dispone la interfaz es la captura de tramas remotas, para su posterior análisis.

El programa está realizado en C# a con la herramienta de programación Visual Studio 2010. Se ha escogido este programa por la versatilidad y las herramientas de las que dispone.

Tabla de contenidos

1. Introducción

En el presente proyecto se realiza una interfaz gráfica para usuario, gracias a la cual se podrá gestionar, analizar y monitorizar cualquier tipo de red CAN. En los siguientes capítulos se detallará el funcionamiento de este tipo de redes.

A través del microcontrolador C8051F500 el usuario se podrá conectar a cualquier red CAN gracias a la diversidad de opciones que permite configurar dicho programa.

Se encuentra una introducción al mundo de las redes industriales, redes informáticas y a las herramientas de análisis de redes. También se definirá y explicará qué son los sistemas empotrados, como el utilizado en este proyecto.

Se realiza una especificación técnica del microcontrolador y del protocolo de comunicación usado, los cuales han sido desarrollados en la anterior parte del proyecto.

Se desarrollará con gran detalle el funcionamiento y opciones del programa y se hará un listado con todas las herramientas empleadas, finalizando con una conclusión global del proyecto.

1.1. Redes Industriales

El desarrollo del control distribuido en la industria va paralelo al de las comunicaciones. Cada vez es más necesario disponer de dispositivos inteligentes para realizar el control o la supervisión remota, tanto de procesos de fabricación, como de almacenamiento o distribución. Los sistemas o redes de comunicación empleados en entornos industriales se encuentran sometidos a una problemática específica que condiciona enormemente su diseño y los diferencia de las redes de datos o redes ofimáticas.

Hacia los años 70 se comenzaron a introducir los computadores en el control de procesos, fundamentalmente para realizar tareas de vigilancia. El computador se encargaba de supervisar las variables controladas para detectar niveles anómalos, generando entonces las alarmas pertinentes y generando informes sobre el estado del sistema. Posteriormente, se comenzó a incluir también en las labores de control, ya que debido a su capacidad de cálculo podía sustituir al panel de control y tener programados los bucles de control. El principal inconveniente que se planteaba era la mayor debilidad del sistema al existir un punto de fallo crítico, el propio ordenador. Una posible solución consiste en duplicar el equipo, disponiendo de un ordenador de respaldo, comunicado con el primero, y capaz de seguir con el control del proceso en caso de fallo.

El desarrollo de los microprocesadores, microcontroladores y los controladores lógicos programables (PLC's) dio lugar a la aparición del control distribuido. En este tipo de esquema, un PLC o un microprocesador controla una o más variables del sistema realizando un control directo de las mismas. Estos equipos de control local se comunican con otros elementos de su nivel y con el nivel superior de supervisión. El fallo de un elemento del nivel superior no compromete necesariamente el funcionamiento de los equipos de control local, minimizando su incidencia en el sistema.

Además, la aparición de sensores inteligentes y elementos programables (máquinas de control numérico, PLC's, robots, etc.) que favorecen la automatización y flexibilizan el proceso productivo, demanda la necesidad de permitir su programación y control de forma remota.

Desde el punto de vista empresarial, la necesidad de comunicación no se restringe sólo a la producción. Otros departamentos de la empresa también pueden participar en la red de comunicaciones para permitir un control global del sistema. De este modo, no sólo se controlaría el propio funcionamiento de la planta de fabricación, sino que en función de las decisiones tomadas en las capas administrativas de la empresa, podría actuarse directamente sobre la producción. Se establece un sistema de **control jerarquizado** como el que se representa en la figura 1.

Cada uno de los niveles, además de llevar a cabo labores específicas, realiza un tratamiento y filtrado de la información que es transmitida en sentido ascendente o descendente por la pirámide. Así se limitan los flujos de información a los estrictamente necesarios para cada nivel. También existe un tráfico en sentido horizontal dentro de cada nivel, con distintas condiciones en cada uno de ellos.

El nivel inferior realiza el **control digital directo** de las variables del sistema o el control de los elementos de fabricación (en sistemas de fabricación flexible). Se adquieren datos de los sensores y se actúa en función de los algoritmos de control y consignas seleccionadas por el nivel superior. Se ejecutan programas de mecanización o manipulación, se activan alarmas y se transmiten los mensajes e informaciones oportunas al nivel superior.

Niveles jerárquicos en la automatización industrial

El siguiente nivel es el de **supervisión** al nivel de célula de fabricación o de control. Elabora la información procedente del nivel inferior y se informa al operario de la situación de las variables y de las alarmas. Corrige algoritmos de control, consignas y programas.

El tercer nivel lleva a cabo labores de **coordinación de la planta**. Controla y organiza toda el área de producción tratando de optimizar balances de materias y energía (flujos entre almacén, planta, distribución e incluso proveedores). Para ello establece las condiciones de operación de cada proceso del área y las envía a cada control supervisor para que estos las adapten y distribuyan entre los controles directos.

En el siguiente nivel se realiza la **planificación** de la producción del conjunto de la factoría. Se encuentran en el también los elementos de oficina técnica que mediante herramientas CAD/CAM/CAE permiten el diseño de productos y la elaboración automática de programas para los elementos de fabricación.

En el nivel superior se establece la **política de la producción** del conjunto de la empresa en función de los recursos y costes del mercado. En él se incluyen labores de contabilidad y gestión empresarial.

La consecución de la implementación completa de todos estos niveles da lugar a la aparición del **CIM (Computeer Integrated Manufacturing)**. Uno de los principales inconvenientes para el logro de esta integración se encuentra en los problemas que presenta la intercomunicación de los elementos de la base de la pirámide.

1.1.1. Buses de Campo

Un bus de campo es un término genérico que describe un conjunto de redes de comunicación para uso industrial, cuyo objetivo es sustituir las conexiones punto a punto entre los elementos de campo y el equipo de control a través del tradicional bucle de corriente de 4-20mA. Típicamente son redes digitales, bidireccionales, multipunto, montadas sobre un bus serie, que conectan dispositivos de campo como PLC's, transductores, actuadores y sensores. Cada dispositivo de campo incorpora cierta capacidad de proceso, que lo convierte en un dispositivo inteligente, manteniendo siempre un coste bajo. Cada uno de estos elementos será capaz de ejecutar funciones simples de autodiagnóstico, control o mantenimiento, así como de comunicarse bidireccionalmente a través del bus.

El objetivo es reemplazar los sistemas de control centralizados por redes para control distribuido con las que mejorar la calidad del producto, reducir costes y mejorar la eficiencia. Para ello se basa en que la información que envían y/o reciben los dispositivos de campo es digital, lo que resulta mucho más preciso que si se recurre a métodos analógicos. Además, cada dispositivo de campo es un dispositivo inteligente y puede llevar a cabo funciones propias de control, mantenimiento y diagnóstico. De esta forma, cada nodo de la red puede informar en caso de fallo del dispositivo asociado, y en general sobre cualquier anomalía asociada al dispositivo. Esta monitorización permite aumentar la eficiencia del sistema y reducir la cantidad de horas de mantenimiento necesarias.

1.1.1.1. Ventajas de los Buses de Campo

La principal ventaja que ofrecen los buses de campo, y la que los hace más atractivos a los usuarios finales, es la **reducción de costes**. El ahorro proviene fundamentalmente de tres fuentes: ahorro en coste de instalación, ahorro en el coste de mantenimiento y ahorros derivados de la mejora del funcionamiento del sistema.

Una de las principales características de los buses de campo es una significativa reducción en el cableado necesario para el control de una instalación. Cada célula de proceso sólo requiere un cable para la conexión de los diversos nodos. Se estima que puede ofrecer una reducción de 5 a 1 en los costes de cableado. En comparación con otros tipos de redes, dispone de herramientas de administración del bus que permiten la reducción del número de horas necesarias para la instalación y puesta en marcha.

El hecho de que los buses de campo sean más sencillos que otras redes de uso industrial como por ejemplo MAP, hace que las necesidades de **mantenimiento de la red** sean menores, de modo que la fiabilidad del sistema a largo plazo aumenta. Además, los buses de campo

permiten a los operadores monitorizar todos los dispositivos que integran el sistema e interpretar fácilmente las interacciones entre ellos. De esta forma, la detección de las fuentes de problemas en la planta y su corrección resulta mucho más sencilla, reduciendo los costes de mantenimiento y el tiempo de parada de la planta.

Los buses de campo ofrecen mayor **flexibilidad** al usuario en el diseño del sistema. Algunos algoritmos y procedimientos de control que con sistemas de comunicación tradicionales debían incluirse en los propios algoritmos de control, radican ahora en los propios dispositivos de campo, simplificando el sistema de control y sus posibles ampliaciones.

También hay que tener en cuenta que las prestaciones del sistema mejoran con el uso de la tecnología de los buses de campo debido a la **simplificación** en la forma de obtener información de la planta desde los distintos sensores. Las mediciones de los distintos elementos de la red están disponibles para todos los demás dispositivos. La simplificación en la obtención de datos permitirá el diseño de sistemas de control más eficientes.

Con la tecnología de los buses de campo, se permite la comunicación bidireccional entre los dispositivos de campo y los sistemas de control, pero también entre los propios dispositivos de campo.

Otra ventaja de los buses de campo es que sólo incluyen 4 capas (Física, Enlace, Aplicación y Usuario), y un conjunto de **servicios de administración**. El usuario no tiene que preocuparse de las capas de enlace o de aplicación. Sólo necesita saber cual es funcionalidad. Al usuario sólo se le exige tener un conocimiento mínimo de los servicios de administración de la red, ya que parte de la información generada por dichos servicios puede ser necesaria para la reparación de averías en el sistema. De hecho, prácticamente, el usuario sólo debe preocuparse de la capa física y la capa de usuario.

1.1.1.2. Buses de Campo existentes

Debido a la falta de estándares, diferentes compañías han desarrollado diferentes soluciones, cada una de ellas con diferentes prestaciones y campos de aplicación. En una primera clasificación podríamos dividirlos en los siguientes grupos:

Buses de alta velocidad y baja funcionalidad:

Diseñados para integrar dispositivos simples como finales de carrera, fotocélulas, relés y actuadores simples, funcionando en aplicaciones de tiempo real, y agrupados en una pequeña zona de la planta, típicamente una máquina. Suelen especificar las capas física y de enlace del modelo OSI, es decir, señales físicas y patrones de bits de las tramas. Algunos ejemplos son:

- CAN: Diseñado originalmente para su aplicación en vehículos.
- SDS: Bus para la integración de sensores y actuadores, basado en CAN
- ASI: Bus serie diseñado por Siemens para la integración de sensores y actuadores.

Buses de alta velocidad y funcionalidad media

Se basan en el diseño de una capa de enlace para el envío eficiente de bloques de datos de tamaño medio. Estos mensajes permiten que el dispositivo tenga mayor funcionalidad de modo que permite incluir aspectos como la configuración, calibración o programación del dispositivo. Son buses capaces de controlar dispositivos de campo complejos, de forma eficiente y a bajo coste. Normalmente incluyen la especificación completa de la capa de aplicación, lo que significa que se dispone de funciones utilizables desde programas basados en PCs para acceder, cambiar y controlar los diversos dispositivos que constituyen el sistema. Algunos incluyen funciones estándar para distintos tipos de dispositivos (perfiles) que facilitan la interoperabilidad de dispositivos de distintos fabricantes. Algunos ejemplos son:

- DeviceNet: Desarrollado por Allen-Bradley, utiliza como base el bus CAN, e incorpora una capa de aplicación orientada a objetos.
- LONWorks Red desarrollada por Echelon.
- BitBus: Red desarrollada por INTEL.
- DIN MessBus: Estándar alemán de bus de instrumentación, basado en comunicación RS-232.
- InterBus-S: Bus de campo alemán de uso común en aplicaciones medias.

Buses de altas prestaciones

Son capaces de soportar comunicaciones a nivel de toda la factoría, en muy diversos tipos de aplicaciones. Aunque se basan en buses de alta velocidad, algunos presentan problemas debido a la sobrecarga necesaria para alcanzar las características funcionales y de seguridad que se les exigen. La capa de aplicación oferta un gran número de servicios a la capa de usuario, habitualmente un subconjunto del estándar MMS. Entre sus características incluyen:

- Redes multi-maestro con redundancia.
- Comunicación maestro-esclavo según el esquema pregunta-respuesta.
- Recuperación de datos desde el esclavo con un límite máximo de tiempo.
- Capacidad de direccionamiento unicast, multicast y broadcast.
- Petición de servicios a los esclavos basada en eventos.
- Comunicación de variables y bloques de datos orientada a objetos.
- Descarga y ejecución remota de programas.
- Altos niveles de seguridad de la red, opcionalmente con procedimientos de autentificación.
- Conjunto completo de funciones de administración de la red. Algunos ejemplos son:

- Profibus
- FIP
- Fieldbus Foundation

Buses para áreas de seguridad intrínseca

Incluyen modificaciones en la capa física para cumplir con los requisitos específicos de seguridad intrínseca en ambientes con atmósferas explosivas. La **seguridad intrínseca** es un tipo de protección por la que el aparato en cuestión no tiene posibilidad de provocar una explosión en la atmósfera circundante. Un circuito eléctrico o una parte de un circuito tienen seguridad intrínseca, cuando alguna chispa o efecto térmico en este circuito producidos en las condiciones de prueba establecidas por un estándar (dentro del cual figuran las condiciones de operación normal y de fallo específicas) no puede ocasionar una ignición. Algunos ejemplos son HART, Profibus PA o FIP.

1.1.1.3. Los Buses de Campo más utilizados

En este apartado realizaremos una breve introducción a los buses más utilizados en el mundo industrial, salvo el bus CAN, que esta explicado con detenimiento en un apartado 5.

Bus EIB:

El Bus de Instalación Europeo EIB es un completo sistema integrado de automatización y control de edificios y viviendas, destinado a la aplicación de soluciones gradualmente compatibles, flexibles y rentables. Debido a su versatilidad funcional, su uso no se reduce a las instalaciones simples y limitadas sino que también proporciona soluciones para el sector del edificio completo.

Las siglas EIB representan la tecnología de instalaciones de edificios más innovadora (“sistema bus”), promovida desde 1990 por el grupo de fabricantes que engloban la EIBA (Asociación EIB), con sede en Bruselas. EIBA está envuelta en la emisión de las marcas registradas relacionadas con el sistema, los estándares de comprobación y calidad de los productos, las actividades de marketing y estandarización,... El EIB también es distribuido bajo varias denominaciones diferentes, por ejemplo: instabus, ABB I-Bus, Tebis,…

Así, el EIB nació de las exigencias de mayor flexibilidad y comodidad en las instalaciones eléctricas, unidas al deseo de minimizar las necesidades de energía.

Las empresas participantes en EIBA garantizan que sus productos sean compatibles con el bus. Por ello se pueden emplear en una instalación EIB aparatos de distintos fabricantes con total interoperabilidad.

El bus de control (medio de transmisión por pares trenzados – "TwistedPair" – TP) se tiende paralelo al cableado de 230 V. Esto implica:

- Una reducción considerable de la cantidad total de cable instalada, en comparación con una instalación convencional (hasta un 60%)
- Un incremento del número de funciones posibles del sistema
- Una mejora en la claridad de la instalación

Este conductor:

- Conecta las cargas y los interruptores que las controlan
- Suministra alimentación a los componentes bus, en la mayoría de los casos.

Al disponer todos los componentes bus de su propia inteligencia, no resulta necesaria una unidad central de control (p. ej. un ordenador). Por lo tanto, el EIB puede ser utilizado tanto para pequeñas instalaciones (viviendas) como en proyectos mucho más grandes (hoteles, edificios administrativos,….).

Gracias a la flexibilidad de la tecnología EIB, cualquier instalación puede ser fácilmente adaptable a las necesidades cambiantes del usuario.

Igualmente, resulta posible implementar el sistema EIB en la red de fuerza de 230 V existente ("Medio de transmisión Powerline" – EIB PL) y en el futuro vía radio (Medio de transmisión por "Radio Frecuencia" – EIB RF).

VENTAJAS:

En las instalaciones tradicionales cada función requiere una línea eléctrica propia, y cada sistema de control precisa una red separada. Por el contrario, con el EIB se pueden controlar, comunicar y vigilar todas las funciones de servicio y su desarrollo, con una única línea común. Con esto se puede dirigir la línea de energía sin desvíos, directamente hasta el aparato consumidor.

Además del ahorro en el cableado se presentan adicionalmente otras ventajas: La instalación en un edificio se puede realizar de un modo más sencillo desde el principio, y después se puede ampliar y modificar sin problemas. Ante cambios de uso o reorganización del espacio, el EIB consigue una adaptación rápida y sin problemas, mediante una fácil ordenación (cambio de parametrización) de los componentes del bus, sin necesidad de un nuevo cableado.

Este cambio de parametrización se realiza con un PC, conectado al sistema EIB, que tenga instalado el software ETS (EIB Tool Software) para proyecto y puesta en servicio, que ya se emplea en la primera puesta en marcha. El EIB se puede conectar mediante las correspondientes interfaces con los centros de control de otros sistema de automatización de edificios o con una red digital de servicios integrados (RDSI). De este modo el uso del EIB en una vivienda unifamiliar resulta tan rentable como en hoteles, escuelas, bancos, oficinas o edificios del sector terciario.

LONWORKS:

LONWorks es un estándar propietario desarrollado por la empresa Echelon. El estándar ha sido ratificado por la organización ANSI como oficial en Octubre de 1999 (ANSI/EIA 709.1-A-1999).

El estándar LONWork se basa en el esquema propuesto por LON(Local Operating Network). Este consiste en un conjunto de dispositivos inteligentes, o nodos, que se conectan mediante uno o más medios físicos y que se comunican utilizando un protocolo común. Por inteligente se entiende que cada nodo es autónomo y proactivo, de forma que puede ser programado para enviar mensajes a cualquier otro nodo como resultado de cumplirse ciertas condiciones, o llevar a cabo ciertas acciones en respuesta a los mensajes recibidos.

Un nodo LON se puede ver como un objeto que responde a varias entradas y que produce unas salidas. El funcionamiento completo de la red surge de las distintas interconexiones entre cada uno de los nodos. Mientras que la función desarrollada por uno de los nodos puede ser muy simple, la interacción entre todos puede dar lugar a implementar aplicaciones complejas. Uno de los beneficios inmediatos de LON es que un pequeño número de nodos pueden realizar un gran número de distintas funciones dependiendo de cómo estén interconectados.

LONWorks utiliza para el intercambio de información (ya sea de control o de estado) el protocolo LonTalk. Este tiene que ser soportado por todos los nodos de la red. Toda la información del protocolo está disponible para cualquier fabricante.

PROTOCOLO LONTALK:

LonTalk ha sido creado dentro del marco del control industrial por lo que se enfoca a funciones de monitorización y control de dispositivos. Dentro de este marco se han potenciado una serie de características:

- Fiabilidad: El protocolo soporta acuso de recibo (acknowledgments) extremo a extremo con reintentos automáticos.
- Variedad de medios de comunicación: tanto cableado como radio. Entre los que están soportados: Par trenzado, red eléctrica, radio frecuencia, cable coaxial y fibra óptica.
- Tiempo de Respuesta: Se utiliza un algoritmo propietario para predicción de colisiones que consigue evitar la degradación de prestaciones que se produce por tener un medio de acceso compartido.

- Bajo coste de los productos: Muchos de los nodos LON son simples dispositivos como interruptores o sensores. El protocolo ha sido diseñado para poder ser implementado en un único chip de bajo coste.

Para simplificar el enrutamiento de mensajes, el protocolo define una jerarquía de direccionamiento que incluye dirección de dominio, subred y nodo. Cada nodo está conectado físicamente a un canal. Un dominio es una colección lógica de nodos que pertenecen a uno o más canales. Una subred es una colección lógica de hasta 127 nodos dentro de un dominio. Se pueden definir hasta 255 subredes dentro de un único dominio. Todos los nodos de una subred deben pertenecer al mismo canal, o los canales tienen que estar conectados por puentes (bridges). Cada nodo tiene un identificador de 48-bits único, asignado durante la fabricación, que se usa como dirección de red durante la instalación y configuración. La tabla siguiente resume la jerarquía de red:

Subredes por dominio:	255
Nodos por subred:	127
Nodos por dominio:	32,385
Grupos por dominio:	255
Nodos por grupo:	63
Número de dominios:	281,474,976,710,656

LonTalk es un estándar abierto que puede ser implementado por cualquier fabricante de circuitos integrados. En la realidad el chip que se utiliza es el denominado Neuron, fabricado por Cypress, Toshiba y Motorola.

Variables de Red (Network Variables):

La comunicación entre nodos se completa con las variables de red. Cada nodo define una serie de variables de red que pueden ser compartidas por los demás nodos. Cada nodo tiene variables de entrada y de salida, que son definidas por el desarrollador.

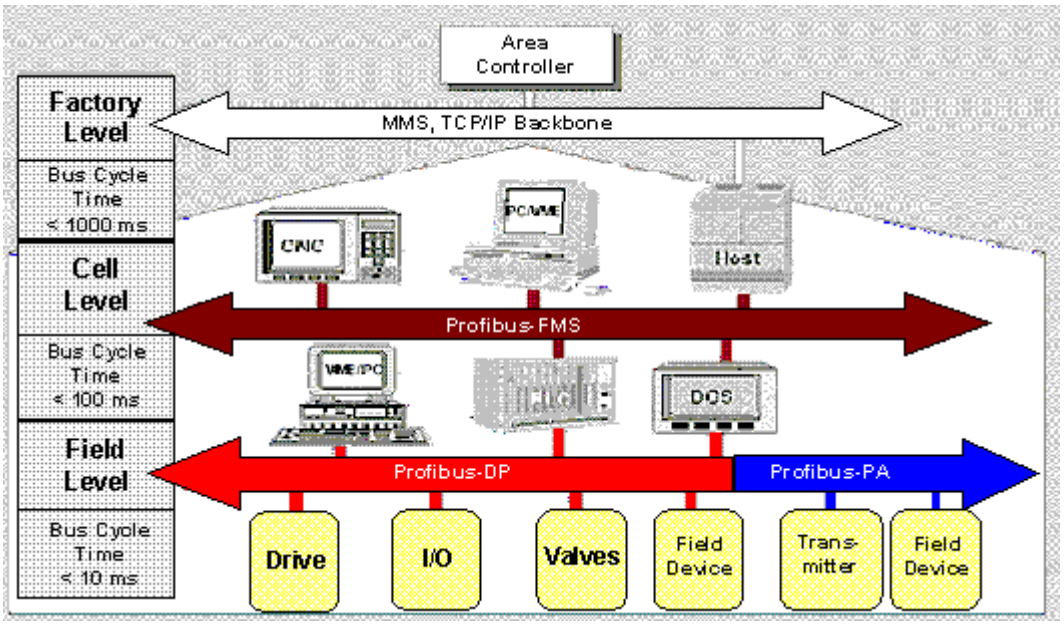
Siempre que el programa que se ejecuta en un nodo escribe un nuevo valor en una de sus variables de salida, este se propaga a través de la red a todos los nodos cuyas variables de entrada estén conectadas a esta variable de salida. Todas estas acciones están implementadas dentro del protocolo. Sólo se podrán ligar variables de red que sean del mismo tipo.

Esta forma de comunicación es orientada a datos (eventos), encontraste a la comunicación orientada a comandos.

Para guardar la interoperabilidad entre productos de distintos fabricantes, se definen las variables a partir de una definición de tipos estándar (Standard Network Variable Types). Echelon mantiene una lista de unos 100 tipos accesible a cualquier fabricante.

Profibus:

Es un estándar de red de campo abierto e independiente de proveedores, donde la interfaz de ellos permite amplia aplicación en procesos, fabricación y automatización predial. Este estándar es garantizado según los estándares EN 50170 y EN 50254. Desde enero de 2000, el PROFIBUS está fuertemente establecido con el IEC 61158, al lado de siete otros fieldbuses. El IEC 61158 se divide en siete partes, de números 61158-1 a 61158-6, con las especificaciones del modelo OSI. Esa versión, que fue ampliada, incluyó el DPV-2.



En todo el mundo, los usuarios pueden ahora tener como referencia un estándar internacional de protocolo, cuyo desarrollo busco y aún busca la reducción de costos, flexibilidad, confianza, orientación hasta el porvenir, posibilitar las más variadas aplicaciones, interoperabilidad y múltiples proveedores.

Actualmente, calculase por encima de 20 millones de nudos instalados con tecnología PROFIBUS y más de 1000 fábricas con tecnología PROFIGUS PA. Son 23 organizaciones regionales (RPA's) y 33 Centros de Capacidad en PROFIBUS (PCC's), ubicados estratégicamente en varios países, vueltos a proveer soporte a sus usuarios, inclusive en Brasil, en la Escuela de Ingeniería de São Carlos – USP, que tiene el único PCC de América Latina.

- Más de 1300 socios alrededor del mundo.
- Más de 20 millones de nudos instalados exitosamente.
- Más de 2800 productos y más de 2000 proveedores de las más variadas aplicaciones.

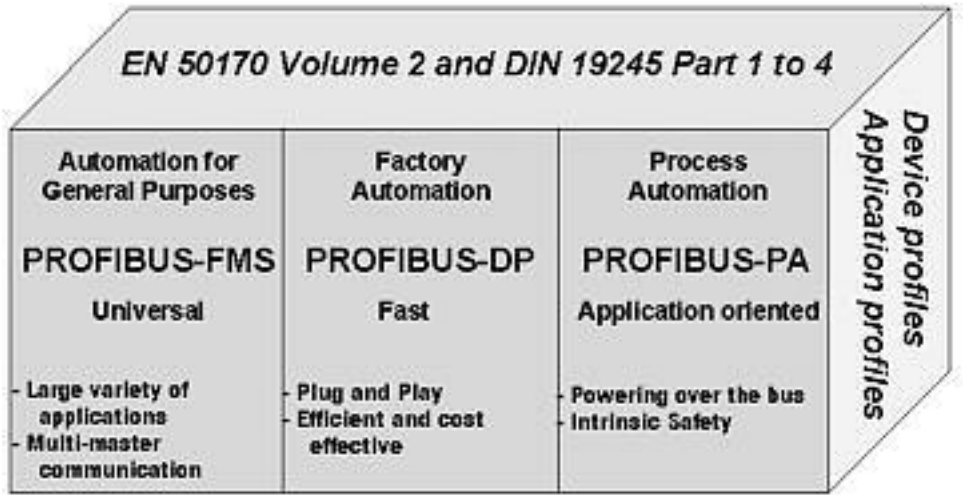
La tecnología de la información tuvo un papel decisivo en el desarrollo de la automoción, cambiando jerarquías y estructuras en el ambiente de la oficina, y llega ahora a los más

variados sectores del entorno industrial, de las industrias de proceso y manufactura hasta los edificios y sistemas logísticos. La capacidad de comunicación entre instrumentos y el uso de mecanismos estandarizados, abiertos y transparentes son componentes indispensables del moderno concepto de automatización.

La comunicación ampliase muy rápido en el sentido horizontal, en los niveles inferiores y aún en el sentido vertical, integrando los niveles jerárquicos de un sistema. Según las características de la aplicación e el costo máximo buscado, la combinación gradual de distintos sistemas de comunicación, tal como: Ethernet, PROFIBUS y AS-Interface, brinda las condiciones ideales de redes abiertas en procesos industriales.

La revolución de la comunicación industrial en la tecnología de la automatización demuestra mucho potencial en la optimización de sistemas de proceso e hizo una gran contribución a la mejoría del uso de los recursos. Las informaciones siguientes proveen explicación resumida del PROFIBUS como el vínculo central en el flujo de informaciones en la automatización.

La arquitectura del PROFIBUS se divide en tres tipos principales:



Tipos de Profibus

PROFIBUS DP:

Esta es la solución de alta velocidad del PROFIBUS. Su desarrollo fue perfeccionado principalmente para comunicación entre los sistemas de automatización y los equipos descentralizados. Es aplicable en los sistemas de control, donde se destaca el acceso a los dispositivos distribuidos de I/O. Es utilizado en sustitución a los sistemas convencionales 4 a 20 mA, HART o en transmisiones de 23 Volts, en medio físico RS-485 o fibra óptica.

Requiere menos de 2 ms para transmitir 1 Kbyte de entrada y salida y es muy usado en controles con tiempo crítico.

Actualmente, 90% de las aplicaciones relativas a esclavos Profibus utilizan el PROFIBUS DP. Esta variedad está disponible en tres versiones: DP-V0 (1993), DP-V1 (1997) e DP-V2

(2002). Cada versión tuvo su origen según el adelanto de la tecnología y la búsqueda de nuevas aplicaciones a lo largo del tiempo.

PROFIBUS-FMS:

El PROFIBUS-FMS brinda al usuario amplia selección de funciones cuando comparado con otras variedades. Es la solución estándar de comunicación universal usada para solucionar tareas complejas de comunicación entre CLP's y DCS's. Esa variedad soporta la comunicación entre sistemas de automatización, además del cambio de datos entre equipos inteligentes, y es usada, en general, a nivel de control. Debido a su función primaria establecer la comunicación maestro-a-maestro (peer-to-peer) viene siendo reemplazada por aplicaciones en la Ethernet.

PROFIBUS-PA:

El PROFIBUS-PA es la solución PROFIBUS que satisface las exigencias de la automatización de procesos, donde hay la conexión de sistemas de automatización y los sistemas de control de proceso con equipos de campo, tal como: transmisores de presión, temperatura, conversores, posicionadores, etc. Puede usarse para reemplazar el estándar 4 a 20mA.

Existen ventajas potenciales en utilizarse esta tecnología, que subrayan las ventajas funcionales (transmisión de informaciones confiables, tratamiento de estatus de las variables, sistema de seguridad en fallos, equipos con capacidad de auto-diagnosis, alcance de los equipos, alta resolución en mediciones, integración con el control discreto en alta velocidad, aplicaciones en cualquier sección, etc.). Además de los beneficios económicos pertinentes a las instalaciones (reducción hasta 25% en algunos casos en comparación con los sistemas convencionales), menos tiempo de puesta en marcha, ofrece un aumento sensible de funcionalidad y seguridad.

El PROFIBUS PA permite medición y control a través de línea de dos hilos simples. También permite accionar los equipos de campo en zonas con seguridad intrínseca. El PROFIBUS PA permite aún el mantenimiento y la conexión/desconexión de equipos mismo durante la operación, sin afectar otras estaciones en zonas de potencial explosivo. El PROFIBUS PA fue desarrollado en cooperación con los usuarios de la Industria de Control y Proceso (NAMUR), cumpliendo con las exigencias de esa zona de aplicación:

La conexión de los transmisores, conversores y posicionadores de red PROFIBUS DP se hace con un acoplador DP/PA. El par torcido de hilos es utilizado en la impulsión y la comunicación de datos de todos los equipos, resultando en la instalación más fácil y en el bajo costo de hardware, menos tiempo de iniciación, mantenimiento libre de problema, bajo costo de software de ingeniería y alta confianza en la operación.

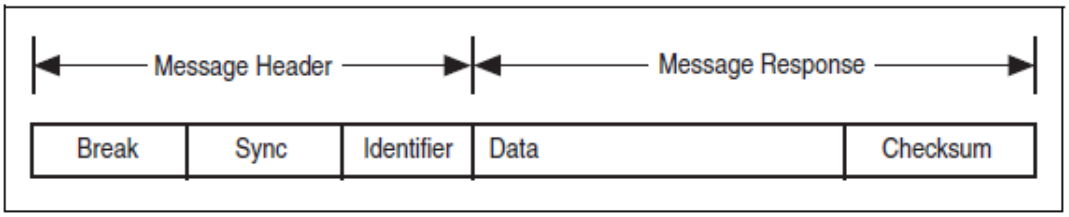
Todas las variedades del PROFIBUS se basan en el modelo de comunicación de redes OSI (Open System Interconnection), cumpliendo con el estándar mundial ISO 7498. Debido a las exigencias del campo, sólo los niveles 1 y 2, además del nivel 7 del FMS, son instalados, por razones de eficiencia.

Bus LIN:

El bus de Red Local de Interconexión (LIN) fue desarrollado para crear un estándar para comunicación multiplexada de bajo costo en redes automotrices. A pesar que CAN cubre la necesidad para alto ancho de banda, redes de manejo de error avanzado, los costos de hardware y software por la implementación de CAN se han vuelto prohibitivos para dispositivos de menor rendimiento como controladores de potencia de ventanas y asientos. LIN proporciona comunicación rentable en aplicaciones donde el ancho de banda y la versatilidad de CAN no son requeridos. Puede implementar LIN prácticamente a un menor precio usando el transmisor/receptor estándar serial universal asincrónico (UART) embebido en la mayoría de los microcontroladores modernos de bajo costo de 8 bits.

Las redes automotrices modernas usan una combinación de LIN para aplicaciones de bajo costo principalmente en electrónicos, CAN para comunicación de tren de potencia y carrocería y el bus FlexRay para comunicaciones de datos sincronizados de alta velocidad en sistemas avanzados como suspensión activa.

El bus LIN usa un enfoque maestro/esclavo que consta de un LIN maestro y uno o más LIN esclavos.



Mensaje en un bus LIN

El encabezado de mensaje consiste de una interrupción usada para identificar el inicio del marco y el campo de sincronización usado por el nodo esclavo para sincronización de reloj. El identificador (ID) consta de un ID de mensaje de seis bits y un campo paridad de dos bits. El ID indica una dirección específica de mensaje pero no el destino. Después de la recepción e interrupción del ID, un esclavo comienza la respuesta del mensaje, la cual consiste de uno a ocho bytes de datos y una suma de verificación de ocho bits.

El maestro controla la secuencia de los marcos del mensaje, la cual es fija en un programa. Puede cambiar el programa según se necesite.

Hay varias versiones de LIN estándar. La versión 1.3 finalizó la comunicación byte-layer. Las versiones 2.0 y 2.1 añaden más especificaciones de mensajes y servicios pero son compatibles al nivel de byte con LIN 1.3.

FORMATO:

El bus LIN es un bus con un solo dispositivo maestro y uno o más dispositivos esclavos. El dispositivo maestro contiene una tarea de maestro y una tarea de esclavo. Cada dispositivo

esclavo tiene solamente una tarea de esclavo. La comunicación a través del bus LIN está controlada completamente por la tarea de maestro en el dispositivo maestro. La unidad básica de transferencia en el bus LIN es el marco, el cual está dividido en un encabezado y una respuesta. El encabezado siempre es transmitido por el nodo maestro y consiste de tres diferentes campos: la interrupción, la sincronización (symc) y el identificador (ID). La respuesta, la cual es transmitida por una tarea de esclavo y puede residir ya sea en el nodo maestro o un nodo esclavo, consiste de una carga útil de datos y una suma de verificación.

Normalmente, la tarea de maestro consulta cada tarea de esclavo en un ciclo al transmitir un encabezado, el cual consiste de una secuencia de interrupción-sincronización-ID. Antes de comenzar el LIN, cada tarea esclavo es configurada para publicar datos al bus o suscribir a datos en respuesta a cada ID de encabezado recibido. Después de recibir el encabezado, cada tarea de esclavo verifica la paridad de ID y después checa el ID para determinar si necesita publicar o suscribir. Si la tarea de esclavo necesita publicar una respuesta, transmite uno de los ochos bytes de datos al bus seguido de un byte de suma de verificación. Si la tarea de esclavo necesita suscribirse, lee la carga útil de los datos y el byte de la suma de verificación del bus y procede de manera apropiada.

Para comunicación estándar de esclavo a maestro, el maestro publica el identificador a la red y solamente un esclavo responde con una carga de datos.

La comunicación de maestro a esclavo se logra por una tarea de esclavo diferente en el nodo maestro. Esta tarea auto recibe todos los datos publicados al bus y responde como si fuera un nodo esclavo independiente. Para transmitir bytes de datos, el maestro debe primero actualizar su respuesta interna a la tarea de esclavo con los valores de datos que desea transmitir. El maestro entonces publica el encabezado de marco adecuado y la tarea interna de esclavo transmite su carga de datos al bus.

1.2. Análisis de redes

Los servicios de tecnologías de la información, la plataforma tecnológica, los departamentos de sistemas, la conectividad y las líneas de comunicaciones se están convirtiendo cada vez más en la base y columna vertebral sobre la que las empresas gestionan sus negocios y se comunican y realizan transacciones con sus clientes, empleados y proveedores. Dicha infraestructura crece y se vuelve más compleja a medida que se introducen nuevos servicios con el objeto de llegar a nuevos mercados o de mejorar la calidad y fidelizar a nuestros clientes.

Por todo ello, el nivel de servicio, de profesionalidad y de disponibilidad que se exige a los empleados de tecnologías de la información y a las plataformas que mantienen es cada vez mayor, llegando a ser del 100% en la mayoría de los casos, 24 horas al día y 365 días al año. Sin embargo, el gran número de dispositivos, tecnologías, plataformas, lenguajes e interfaces con las que deben enfrentarse en el día a día aumenta proporcionalmente. Además, no basta con actuar reactivamente o como respuesta a las quejas por falta de servicio de usuarios o clientes, sino rápida y pro-activamente ya que cualquier tiempo de parada de la plataforma tecnológica se convierte en tiempo de parada del negocio y, por lo tanto, repercute negativamente en la facturación de la empresa.

En este escenario, cualquier acción o solución encaminada a monitorizar la totalidad de la plataforma tecnológica supone una inversión de rápido retorno. Ya no se trata de reaccionar ante las llamadas por parte de los usuarios denunciando problemas con el correo electrónico o la salida a Internet, o ante las quejas de clientes y directivos que no pueden realizar transacciones comerciales a través de los sistemas de gestión, sino ante un sistema de control y monitorización que escala los problemas detectados de manera instantánea. De este modo, los tiempos de parada se reducen drásticamente y se mejora la calidad de servicio ofrecida desde T.I.

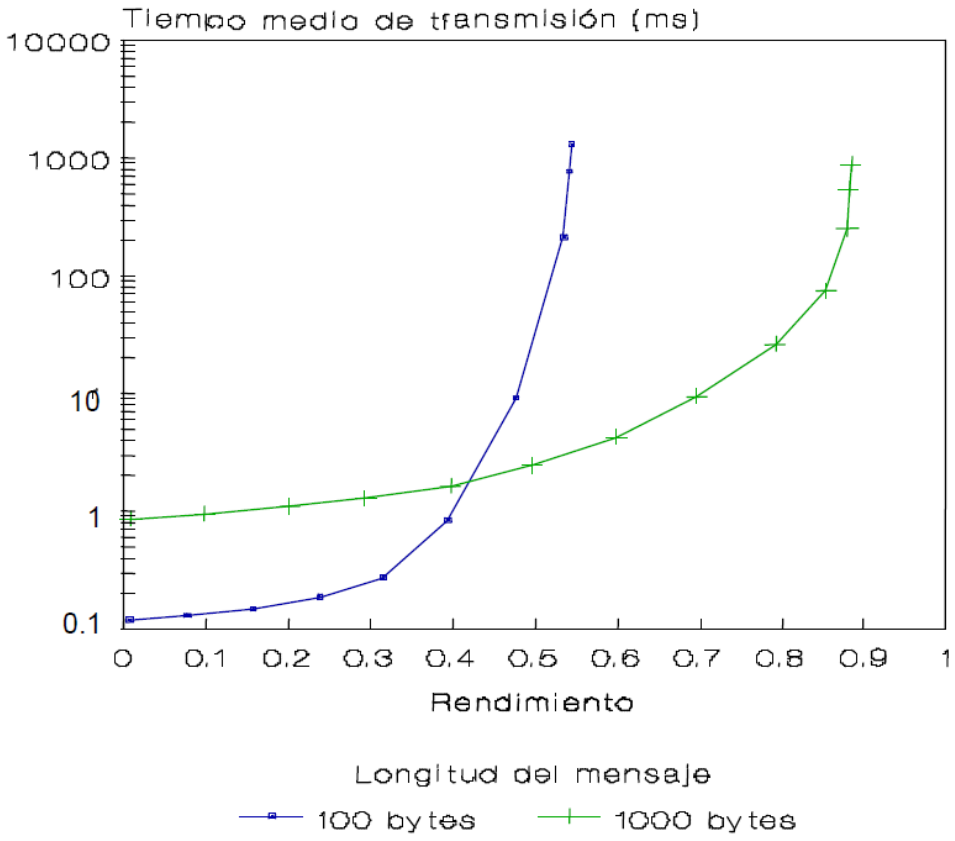
La solución que cumpla con los requisitos especificados debe ser de arquitectura abierta y debe respaldarse en protocolos estándares. De esta manera, podrán monitorizarse plataformas de cualquier tipo y de cualquier fabricante: dispositivos de comunicaciones, estaciones de trabajo, servidores y hosts críticos, impresoras de red, dispositivos de seguridad, PLC’s, aplicaciones críticas, etc.

1.2.1. Redes de Control y redes de Datos

En el esquema piramidal que se presentó en el apartado anterior, existían diferentes niveles de comunicación, cada uno de ellos con diferentes necesidades. Podemos hablar en realidad de dos tipos de redes: redes de control y redes de datos. Las redes de control están ligadas a la parte

baja de la pirámide, mientras que las redes de datos (o de ofimática) están más ligadas a las partes altas de la jerarquía.

En general, las redes de datos están orientadas al transporte de grandes paquetes de datos, que aparecen de forma esporádica (baja carga), y con un gran ancho de banda para permitir el envío rápido de una gran cantidad de datos. En contraste, las redes de control se enfrentan a un tráfico formado por un gran número de pequeños paquetes, intercambiados con frecuencia entre un alto número de estaciones que forman la red y que muchas veces trabajan en tiempo real.

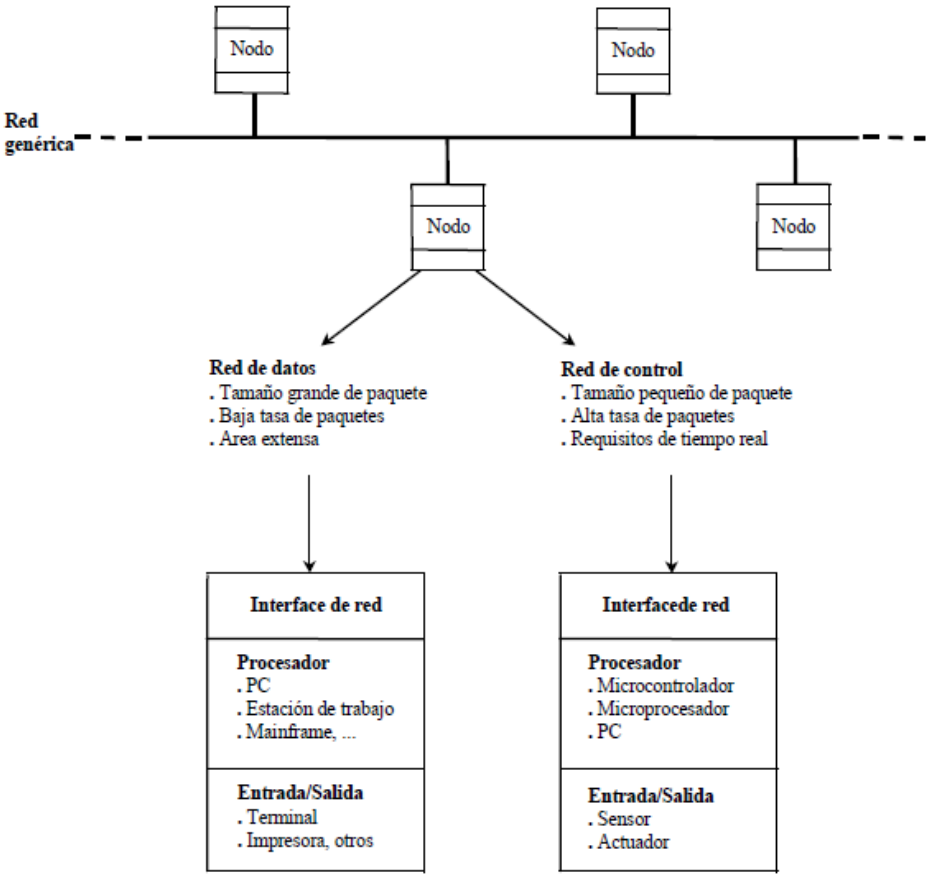


Comportamiento de la red ante variaciones del tamaño de los paquetes

En principio, las redes de datos estudiadas hasta ahora podrían emplearse para su uso como redes de control, sin embargo, es evidente que no resultan adecuadas para las necesidades de este tipo de aplicaciones. Por ejemplo, es sabido que la red Ethernet tiene una gran eficiencia, hasta el 90-95% de la capacidad del canal cuando los mensajes son largos y suficientemente espaciados. Sin embargo, la cantidad de información que una red Ethernet es capaz de transportar cae bruscamente cuando se utiliza por encima del 35% de la capacidad del canal, si el tamaño de los mensajes es pequeño. En las redes de control es habitual encontrar este tipo de carga, porque el tráfico de la red depende directamente de eventos externos que están siendo controlados (o monitorizados) por los diferentes nodos que la componen. A menudo, varios nodos necesitan enviar información simultáneamente en función de uno o más eventos externos. Este hecho, junto con el gran número de nodos que suelen estar presentes, implica la existencia de frecuentes periodos en los que muchas estaciones envían pequeños paquetes de información.

Por todas estas razones, es necesario diseñar una arquitectura de red adaptada a las características particulares de este tipo de tráfico. En el diseño se deberán tener en cuenta aspectos como los tipos de protocolos utilizados, la interoperabilidad, la topología y la facilidad de administración.

Deben usarse protocolos abiertos, disponibles por toda la comunidad de fabricantes y usuarios. Este aspecto es básico para conseguir que equipos de diferentes fabricantes puedan trabajar en conjunto en una misma red. También juega un papel fundamental determinar el tipo de información que viajará por la red. En las redes ofimáticas, esta información consiste básicamente en datos de usuario y en algunas ocasiones información para la administración y el mantenimiento de la propia red. En una red de control, esta elección es menos clara ya que el correcto funcionamiento de la red es vital.



Comparación entre redes de datos y redes de control

Pueden distinguirse dos tipos de redes según la información que transporten: redes basadas en comandos y redes basadas en estado. En las redes basadas en comandos, la información consiste en una orden con la que un nodo controla el funcionamiento de otro. El principal problema radica en que si se dispone de un amplio conjunto de tipos de nodos, habrá un aumento exponencial del número de posibles comandos y de la sobrecarga que supone su procesamiento. En las redes orientadas a estado las cosas son más sencillas. En este caso, la funcionalidad de un nodo no depende de ningún otro. Cada nodo enviará mensajes en los que

indicará a los demás el estado en que se encuentra. Los nodos que reciban estos mensajes modificarán su estado en función de la nueva información. Existen implementaciones que combinan ambos métodos.

Por lo que se refiere al tipo de topología que deben adoptar las redes de control, cabe destacar que cualquiera de las topologías clásicas de las redes de datos es válida. Cada una de ellas con sus propias ventajas y limitaciones. Cualquiera puede satisfacer las necesidades de cableado, prestaciones y coste de algún tipo de aplicación. La elección está determinada fundamentalmente por el control de acceso al medio y el tipo de medio que se emplea. El conjunto formado por el medio, el control de acceso y la topología, afecta prácticamente a cualquier otro aspecto de la red de control: coste, facilidad de instalación, fiabilidad, prestaciones, facilidad de mantenimiento y expansión.

La selección de la topología suele hacerse basándose en los requisitos específicos de cada sistema en cuanto a coste de instalación y tolerancia a fallos. Muchas redes de control permiten el uso de distintas topologías.

El control de acceso al medio es vital. Elegida una topología, hay que definir como accederá cada nodo a la red. El objetivo es reducir las colisiones (idealmente eliminarlas) entre los paquetes de datos y reducir el tiempo que tarda un nodo en ganar el acceso al medio y comenzar a transmitir el paquete. En otras palabras, maximizar la eficiencia de la red y reducir el retardo de acceso al medio. Este último parámetro es el factor principal a la hora de determinar si una red sirve para aplicaciones en tiempo real o no.

El direccionamiento de los nodos es otro de los aspectos claves. En una red de control, la información puede ser originada y/o recibida por cualquier nodo. La forma en que se direccionen los paquetes de información afectará de forma importante a la eficiencia y la fiabilidad global de la red. Se pueden distinguir tres tipos de direccionamiento:

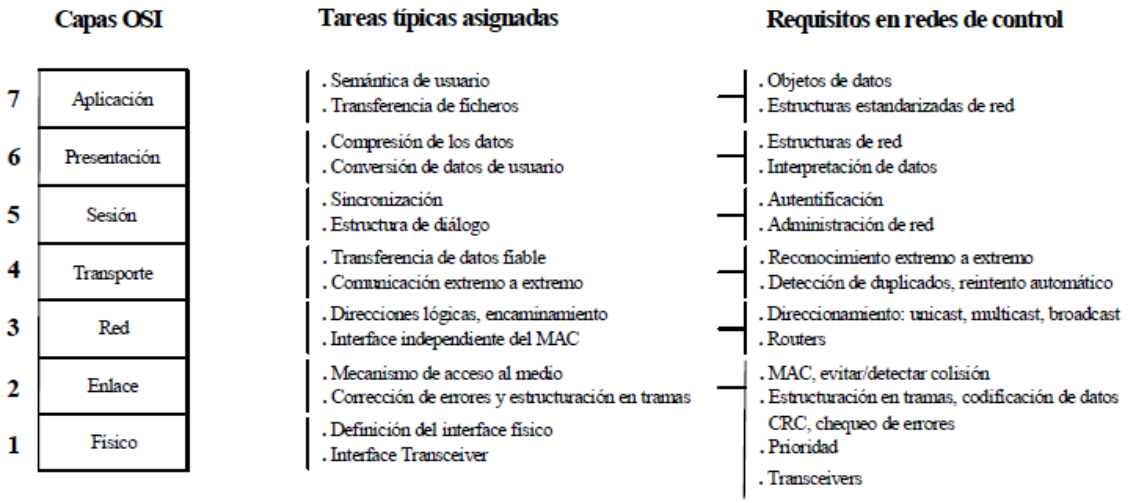
- **Unicast:** El paquete es enviado a un único nodo de destino.
- **Multicast:** El paquete es enviado a un grupo de nodos simultáneamente.
- **Broadcast:** El paquete es enviado a todos los nodos de la red simultáneamente.

El direccionamiento broadcast presenta la ventaja de su sencillez. Es adecuado para redes basadas en información de estado. Cada nodo informa a todos los demás de cuál es estado actual. El principal inconveniente es que los nodos pueden tener que procesar paquetes que no les afecten directamente. Los esquemas de direccionamiento unicast y multicast son más eficientes, y facilitan operaciones como el acuse de recibo y el reenvío, características que aumentan la fiabilidad del sistema. En redes de control, es muy habitual encontrar esquemas de direccionamiento del tipo maestro-esclavo. Este tipo de esquemas permite plasmar ciertos aspectos jerárquicos del control de forma sencilla, a la vez que simplifica el funcionamiento de la red y por tanto abarata los costes de la interfaz física.

La elección del medio físico afecta a aspectos tales como la velocidad de transmisión, distancia entre nodos y fiabilidad. En muchas redes de control se recurre a una mezcla de distintos medios físicos para cumplir con los requisitos de diferentes secciones al menor coste

posible. Se incorporarán los encaminadores, puentes o repetidores necesarios para asegurar el objetivo de una comunicación extremo a extremo transparente, al menor coste posible, y sin que la integración conlleve una disminución de las prestaciones.

El control en tiempo real demanda de las redes de control buenos tiempos de respuesta. Por ejemplo, el retardo entre la detección de un objeto en una línea de montaje de alta velocidad y el arranque de una máquina de pintado puede ser del orden de decenas de milisegundos. En general, las redes de datos no necesitan una respuesta en tiempo real cuando envían grandes conjuntos de datos a través de la red. El control de acceso al medio y el número de capas implementadas en la arquitectura de red resultan determinantes a la hora de fijar la velocidad de respuesta de la red. La implementación de las siete capas del modelo OSI implica una mayor potencia de proceso por la sobrecarga que conlleva con respecto a un sistema más sencillo que por ejemplo sólo implementase las dos primeras capas. En ocasiones, los beneficios que aportan las capas adicionales compensan la sobrecarga adicional (que implica un mayor coste), sobre todo a medida que aumenta la funcionalidad demandada de la red y mejora la tecnología disponible. Cuando la velocidad es el factor esencial, como ocurre con muchos buses de campo, el modelo puede aligerarse ya que en la mayor parte de este tipo de aplicaciones las capas de red, transporte, sesión y presentación no son necesarias.



Capas del modelo OSI/ISO y su relación con las redes de control

Otra forma de favorecer un tiempo de respuesta pequeño es la capacidad para establecer mensajes con diferentes prioridades, de forma que mensajes de alta prioridad (como por ejemplo una alarma) tengan más facilidad para acceder al medio.

Por último, hay que destacar el papel que juega la seguridad de la red. Podemos destacar dos niveles diferentes de seguridad. Por una parte la protección frente a accesos no autorizados a la red, y por otra parte la protección frente a fallos del sistema y averías.

El primer problema es el menos grave, ya que la mayor parte de las redes de control no están conectadas a redes externas a la fábrica. Además, en la práctica, la mayor parte de las veces, las redes pertenecientes a los escalones más bajos de la pirámide no están conectadas siquiera con las redes de nivel superior dentro de la propia fábrica. En cualquier caso, los mecanismos de

protección son similares a los empleados en las redes de datos: claves de usuario y autentificación de los nodos de la red.

La protección frente a fallos juega un papel mucho más importante, debido a que se debe evitar a toda costa, que este hecho afecte negativamente a la planta. Por ejemplo, los sistemas de refrigeración de una central nuclear no pueden bloquearse porque la interfaz de comunicaciones de un nodo de la red se averíe. Para ello es fundamental que los nodos puedan detectar si la red está funcionando correctamente o no, y en caso de avería puedan pasar a un algoritmo de control que mantenga la planta en un punto seguro. Si el sistema es crítico, se deben incluir equipos redundantes, que reemplacen al averiado de forma automática en caso de avería. La monitorización de la red y la capacidad de diagnóstico representan por tanto dos puntos básicos de cualquier red de control.

La necesidad de buenas herramientas de mantenimiento y administración de la red son vitales. No sólo por lo dicho anteriormente sino que también porque en las redes de control las operaciones de reconfiguración y actualización de la red son frecuentes.

1.2.2. Ejemplos de Monitorización

En este apartado vamos a ver algunos ejemplos de monitorización en los buses industriales anteriormente mencionados, incluyendo el bus CAN.

Bus EIB

El sistema permite registrar y obtener información del estado en que se encuentran los distintos elementos de la instalación. A través del bus se pueden, por ejemplo, enviar mediciones de temperatura, avisos indicaciones de alarma y señales de detección de movimiento para la vigilancia, así como recibir información del estado de apertura/cierre o de conexión/desconexión de distintos componentes del sistema.

Todos esos valores pueden ser recogidos, modificados y supervisados mediante sistemas de visualización que se conectan al bus a través de interfaces serie RS-232, como, por ejemplo, el software de visualización EIB de Siemens o, en el caso de edificios residenciales, el sistema de gestión para funciones de la casa Home Assistant.

Lonworks

Cada nodo LonWorks está basado en un microcontrolador llamado Neuron Chip, del cual podemos destacar:

Tiene un identificador único, el Neuron ID, que permite direccionar cualquier nodo de forma unívoca dentro de una red de control Lonworks.. Tiene un modelo de comunicaciones que es independiente del medio físico sobre el que funciona. El firmware que

implementa el protocolo LonTalk, proporciona servicios de transporte y routing extremo-a-extremo.

Estos circuitos se comunican entre sí enviándose telegramas que contienen la dirección de destino, información para el routing, datos de control así como los datos de la aplicación del usuario y un checksum como código detector de errores. Todos los intercambios de datos se inician en un Neuron Chip y se supervisan en el resto de los circuitos de la red. Un telegrama puede tener hasta 229 octetos de información neta para la aplicación distribuida.

Los datos pueden tener dos formatos, desde un mensaje explícito o una variable de red. Los mensajes explícitos son la forma más sencilla de intercambiar datos entre dos aplicaciones residentes en dos Neuron Chips del mismo segmento Lonworks. Por el contrario, las variables de red proporcionan un modelo estructurado para el intercambio automático de datos distribuidos en un segmento Lonworks. Aunque son menos flexibles que los mensajes explícitos, las variables de red evitan que el programador de la aplicación distribuida esté pendiente de los detalles de las comunicaciones.

Por lo tanto, deberemos comunicarnos con los distintos Neuron Chip, para poder obtener toda la información de control que se reciba en los distintos mensajes, ya que aquí los paquetes van dirigidos.

Profibus

Este bus cuenta con un tipo de dispositivo diseñado para monitorizar, son los AMO DP CLASE 2 (DPM2), que son estaciones de ingeniería para configuración, monitoreo o sistemas de supervisión.

El monitoreo es implementado tanto en el amo DP cuanto en los esclavos, cual monitoreo de tiempo especificado en la configuración. El Amo DPM1 monitorea la transmisión de datos de los esclavos con el Data Control Timer. Utilizase un contador de tiempo para cada dispositivo. El contador expira cuando ocurre una transmisión de datos incorrecta durante el monitoreo y el usuario es informado a respecto. Si la reacción automática a un error (Auto Clear = true) esté habilitada, el amo DPM1 encierra su estado de OPERACIÓN, protegiendo todas las salidas de esclavos y pasando al estado “CLEAR” (apagar o limpiar). El esclavo usa el “watchdogtimer” (contador vigilante) para detectar fallos en el amo o en la línea de transmisión. Si no ocurrir ninguna comunicación dentro de este periodo, el esclavo pondrá sus salidas automáticamente en el estado de seguridad (failsafestate).

En este bus podemos utilizar sistemas modulares, como ComBriks, donde según su página web:

Sistema ComBricks

"COMbricks es un sistema modular que permite mezclar distintos componentes de automatización en un mismo backplane. Los módulos repetidores se pueden insertar junto a un esclavo Profibus y, al mismo tiempo, en un navegador web sobre Ethernet podemos monitorizar remotamente las condiciones de la instalación gracias a ProfiTrace OE".

Y lo hace, de la siguiente manera: "La actual tendencia en la arquitecturas Profibus es la segmentar mediante repetidores, fibra óptica y ProfiHubs para solventar los problemas típicos de la dificultad en el cableado. COMbricks añade otro importante elemento: la creación de hubs repetidores modulares que pueden ser gestionados remotamente con un ProfiTrace embebido.

COMbricks está basado en un backplane en el que podemos insertar hasta 10 módulos repetidores hot swap de 2 canales cada uno (20 segmentos transparentes aislados galvánicamente). Cada canal puede manejar hasta 31 dispositivos y un máximo de 1200 m de cable (dependiendo de la velocidad).

La tecnología de redundancia de bus de los módulos repetidores es muy avanzada. Podemos crear un sistema redundante con hasta 10 cables de red paralelos. Esta arquitectura permite una muy alta disponibilidad de la red."

Aparte de estos módulos es necesario el navegador web de Profitrace (ProfiTrace OE), La monitorización y el almacenamiento de datos la llevan a cabo los módulos repetidores insertados en el backplane.

Bus CAN

Al igual que otros buses domóticos, para monitorizar el bus CAN necesitaremos un hardware adicional. Utilizaremos a modo de ejemplo, el proyecto fin de carrera el cual estamos haciendo dos de los tres componentes del trabajo.

En él, tenemos una placa Silabs con dos microcontroladores, que son capaces de enviar y recibir paquetes CAN. Además, en uno de los controladores, tenemos una unidad UART,

mediante la cuál nos comunicamos con el PC por un puerto RS-232. Así, con un módulo retransmitimos -o desde o otro módulo CAN conectado a la placa- y con el otro módulo, el cual también tiene también la UART, recogemos los datos. Esto es posible poniendo el módulo en modo escucha.

Estando en modo escucha, programamos el receptor para que automáticamente pase la información al módulo UART, y de ahí al PC, donde con un programa específico recogemos esta información y la analizamos, trama a trama, para ver cuánto tráfico hay.

Existen soluciones hardware propietarias para este tipo de menesteres. Por ejemplo, el analizador CAN AnalyzerAdvanced, de la compañía Ditecom, que según la compañía "es un medio para verificar y desarrollar con buses CAN-CANOpen. Permite el estudio y la configuración de redes CANopen, y un acceso fácil a los dispositivos y a los objetos. El CAN Analyzer está optoacoplado y preparado para montaje en carril DIN. Permite filtrar las tramas entrantes."

CAN AnalyzerAdvanced

Por supuesto, esto necesita una solución software mediante la cual realizamos el monitoreo. En este caso, se tiene el gestor de red, que "permite escanear todos los nodos de la red o un único nodo así como leer/escribir el registro CAN Y cambiar el estado del ID del nodo (pre-operativo, operativo)."

El monitor CAN, que "monitoriza todas las tramas que circulan por la red. Muestra tanto el COB-ID como los datos, tanto en formato estándar como extendido. Por ejemplo en automoción se pueden capturar los datos que circulan por el bus, datos del inmovilizador, del sensor de volante,... para verificar que éstos están funcionando correctamente."

Por último, el CAN Sender, que "permite enviar tramas (en modo estándar o extendido) al bus CAN directamente. Permite transmitir de forma cíclica o en modo debug (sólo se transmite la trama que se ha seleccionado al bus CAN). Esta función puede ser interesante para simular un sistema, ideal para reparaciones de equipos CAN. Por ejemplo en automoción se pueden generar las tramas CAN para la activación de una radio para poder repararla de forma práctica."

Bus LIN

Este bus tiene un parecido al bus CAN, por lo tanto se puede utilizar el método manual explicado en el punto anterior. LIN es una red barata y lenta utilizada en sub-sistemas CAN.

También hay soluciones hardware propietarias como la interfaz USB LIN 847x de National Instruments, que, entre otras cosas, permite monitorizar y registrar redes portátiles, monitorear la carga del bus, validar de dispositivos con adquisición de datos sincronizada, desarrollar y probar dispositivos LIN y correlacionar datos LIN con medidas externas.”

USB LIN 847x

Incluye software de monitorización, pero al ser compatible con librerías USB, puede crearse un software propio de monitorización.

1.3. Objetivo del Proyecto Final de Carrera

Debido al gran auge que está teniendo el bus CAN en el mundo industrial, y en particular en la automoción, se ha estudiado la realización de una herramienta que permita interactuar a modo de administrador en una red CAN. Este programa se ha basado en el estudio de las herramientas disponibles hoy en día para poder gestionar, visualizar y monitorizar una red. Los objetivos del presente trabajo son:

- Crear una herramienta de fácil manejo para que el usuario pueda interactuar con la red CAN.
- Gestionar y configurar un nodo CAN.
- Conectarse en tiempo real, a través de Ethernet, a cualquier nodo CAN.

2. Sistemas Empotrados

Un sistema embebido (SE) o sistema empotrado lo vamos a definir como un sistema electrónico diseñado específicamente para realizar unas determinadas funciones, habitualmente formando parte de un sistema de mayor entidad. La característica principal es que emplea para ello uno o varios procesadores digitales (CPUs) en formato microprocesador, microcontrolador o DSP lo que le permite aportar 'inteligencia' al sistema anfitrión al que ayuda a gobernar y del que forma parte.

En el diseño de un sistema embebido se suelen implicar ingenieros y técnicos especializados tanto en el diseño electrónico hardware como el diseño del software. A su vez también se requerirá la colaboración de los especialistas en el segmento de usuarios de tales dispositivos, si hubiese lugar a ello.

2.1. Partes y Resumen

Hardware

Normalmente un sistema embebido se trata de un modulo electrónico alojado dentro de un sistema de mayor entidad ('host' o anfitrión) al que ayuda en la realización tareas tales como el procesamiento de información generada por sensores, el control de determinados actuadores, etc.. El núcleo de dicho módulo lo forma al menos una CPU en cualquiera de los formatos conocidos:

- Microprocesador.
- Microcontrolador de 4, 8, 16 o 32 bits.
- DSP de punto fijo o punto flotante.
- Diseño a medida 'custom' tales como los dispositivos FPGA

El módulo o tarjeta, además puede haber sido desarrollado para satisfacer una serie de requisitos específicos de la aplicación a la que está dirigido. Entre éstos, podemos citar:

- Tamaño: por lo general deberá ser reducido, aunque también es posible que se desee que adopte un formato estándar: PC-104, Eurocard, etc.
- Margen de temperatura especifico del ámbito de aplicación:
  - Gran consumo (0ºC hasta 70ºC)
  - Industrial y automoción. Márgenes de temperatura hasta 125ºC

  - Aeroespacial
  - Militar
  - Electromedicina

- Consumo de energía: En aplicaciones en las que es necesario el empleo de baterías, se buscará minimizar éste.
- Robustez mecánica: Existen aplicaciones donde los dispositivos sufren un alto nivel de vibraciones, golpes bruscos, etc. En el diseño se deberá tener en cuenta dicha posibilidad.
- Coste: No es lo mismo diseñar un producto a medida con pocas unidades que diseñar un producto para el competitivo mercado del gran consumo. La calibración de los costes es esencial y es tarea de los ingenieros de diseño.
- Etc.

Software

En lo que se refiere al software, se tendrán requisitos específicos según la aplicación. En general para el diseño de un SE no se dispone de recursos ilimitados sino que la cantidad de memoria será escasa, la capacidad de cálculo y dispositivos externos será limitada, etc. Podemos hablar de las siguientes necesidades:

- Trabajo en tiempo real.
- Optimizar al máximo los recursos disponibles.
- Disponer de un sistema de desarrollo específico para cada familia de microprocesadores empleados.
- Programación en ensamblador, aunque en los últimos años, los fabricantes o empresas externas han mejorado la oferta de compiladores que nos permiten trabajar en lenguajes de alto nivel, tales como C.
- etc.

El empleo de un sistema operativo determinado o el no empleo de éste dependerá del sistema a desarrollar y es una de las principales decisiones que habrá que tomar en la fase de diseño del SE. Así, en el caso de decidirse por el empleo de microcontroladores y DSP, por lo general no se usará sistema operativo mientras que si se emplea algún micro del tipo ARM, PowerPC, Intel X86, etc. si que lo llevará. La decisión dependerá de los requisitos del sistema, tanto técnicos como económicos.

Resumen

Podemos concluir finalmente que un SE consiste en un sistema basado en microprocesador cuyo hardware y software están específicamente diseñados y optimizados para resolver un problema concreto de forma eficiente. Normalmente un SE interactúa continuamente con el

entorno para vigilar o controlar algún proceso mediante una serie de sensores. Su hardware se diseña normalmente a nivel de chips (SoC, System on Chip) o de tarjeta PCB, buscando minimizar el tamaño, el coste y maximizar el rendimiento y la fiabilidad para una aplicación particular.

También comentar que bajo el concepto amplio de sistemas embebidos se da cabida a toda una serie de técnicas y metodologías de diseño tanto hardware como software. Tratarlas todas ellas con un mínimo de profundidad en una única asignatura cuatrimestral es una tarea inabordable. En esta primera fase se ha optado por dar un enfoque volcado hacia el mundo de los microcontroladores, dado el amplio peso que éstos tienen en las aplicaciones de carácter industrial y consumo, la relativa facilidad con que es posible manejar un sistema de desarrollo y la posibilidad de abordar proyectos prácticos no muy complejos en poco tiempo.

2.2. Aplicaciones

Las aplicaciones más numerosas y habituales de los SSEE suelen ser del tipo industrial y gran consumo. Existen en el mercado de semiconductores una amplia variedad de familias de microprocesadores, microcontroladores y DSP's dirigidos a este sector.

En la práctica totalidad de las áreas de nuestra vida nos encontramos con sistemas embebidos que prácticamente nos pasan desapercibidos. Sirva como ejemplo el sector del automóvil, que en pocos años ha introducido notables avances en lo referente a la seguridad, confort, infomovilidad, etc.

Pero, en general, podemos enumerar los siguientes campos de aplicación:

- Equipos industriales de instrumentación, automatización, producción, etc.
- Equipos de comunicaciones.
- En vehículos para transporte terrestre, marítimo y aéreo
- En dispositivos dedicados al sector de consumo tales como electrodomésticos, equipamiento multimedia, juguetes, etc.
- En bioingeniería y electromedicina.
- Sector aeroespacial y de defensa.
- Equipos para domótica.
- Etc.

En la actualidad, todos los fabricantes de semiconductores ofrecen su gama de productos relacionándolos con el amplio rango de aplicaciones a los que van dirigidos. A modo de ejemplo, se reproduce la clasificación que hace Texas Instrument, uno de los líderes mundiales en la fabricación de semiconductores:

- Audio
- Automotive
- Broadband
- Communications & Telecom
- Computers & Peripherals
- Consumer Electronics
- Industrial
- Medical
- Security
- Space, Avionics, & Defense
- Video and Imaging
- Wireless

Un usuario no técnico de un sistema embebido puede no ser consciente de que está usando un sistema computador. En algunos hogares las personas, que no tienen por qué ser usuarias de un ordenador personal estándar (PC), utilizan del orden de diez o más sistemas embebidos cada día: TV, móvil, cámara de fotos, frigorífico, lavadora, coche, etc.

2.3. Arquitectura

En el diseño de SSEE basados en microcontroladores, en general no se requiere una gran potencia de procesado, ni dispositivos de presentación con gran resolución gráfica ni sistema operativo y si en cambio el trabajo en tiempo real. Tampoco se suelen contemplar las posibilidades de ampliación hardware con nuevos módulos ya que el sistema anfitrión se diseña en su totalidad para unos requisitos específicos, de forma tal que si el sistema anfitrión se queda obsoleto lo será no sólo por la CPU embebida sino también por el resto de los elementos que lo integran, con lo que la única alternativa consistirá en el rediseño del sistema completo, en la mayoría de los casos. Requisitos tales como tamaño, margen de temperatura, consumo e inmunidad ante interferencias electromagnéticas suelen ser de gran importancia.

Sin ánimo de ser exhaustivos, en la siguiente figura se muestra un diagrama de bloques de lo que puede ser un modelo general de un sistema embebido, de los aquí considerados.

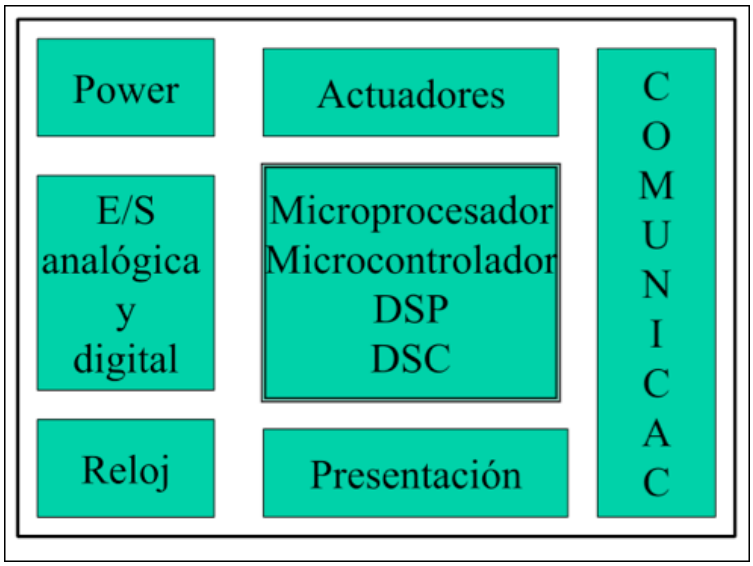


Diagrama de bloques simplificado de un modulo típico de un SE

Se comenta cada uno de sus módulos a continuación.

2.3.1. Formato CPU

Se entiende que en nuestra definición de SE, éste siempre alberga una o más CPUs ya que son el elemento encargado de aportar la 'inteligencia' al sistema. El formato en el que la CPU se encuentra puede ser el de microprocesador, microcontrolador (μC), DSP, etc. Según este formato, la memoria necesaria puede ir integrada dentro del chip que contiene la CPU, de forma externa a éste o un bajo ambas posibilidades. La oferta en el mercado de semiconductores tanto de microprocesadores como microcontroladores y DSP es elevada y se requiere de una cuidada fase de estudio inicial para seleccionar el más adecuado a cada aplicación.

Básicamente, en el diseño de SSEE haremos uso de cualquiera de los siguientes conceptos:

Microprocesador: Es un chip que incluye básicamente la CPU y circuitería relacionadas con los buses de datos y memoria. Para poder realizar su tarea se necesitan otros chips adicionales (Sistema mínimo) tales como memoria, circuitos de entrada salida E/S (I/O) y reloj.

Microprocesador

Microcontrolador (MCU): Es un dispositivo que alberga el sistema mínimo dentro de un único chip, esto es, incluye CPU, buses, reloj, memoria ROM, memoria RAM, E/S, otros periféricos tales como conversores A/D, temporizadores (timers), etc.

Microcontroladores

Procesador Digital de Señal (DSP): Son microcontroladores o microprocesadores diseñados específicamente, tanto en arquitectura hardware como conjunto de instrucciones, para realizar tareas típicas de procesamiento digital de señales en tiempo real.

Procesador Digital de Señal (DSP)

DSC: Dispositivos mixtos microcontrolador/DSP que algunos fabricante ofrecen dentro de su catálogo de productos.

Probablemente, el microcontrolador 8051 (8 bits) desarrollado por Intel 8051 en 1980, marca el inicio en la carrera hacia el desarrollo de productos específicos para aplicaciones embebidas. Éste es probablemente el microcontrolador más popular, pues aunque se lleva hablando mucho tiempo de que estaba condenado a la desaparición, sus continuas mejoras le auguran una larga vida. En este aspecto, comentar que los núcleos 8051 se usan en más de 100 microcontroladores de más de 20 fabricantes independientes como Atmel, Dallas Semiconductor, Philips, Winbond, entre otros. La denominación oficial de Intel para familia de µC's 8051 es MCS 51.

Pero a lo largo de estos años, la gran mayoría de empresas fabricantes de semiconductores, han ido lanzando productos que han invadido el mercado y que hace extremadamente difícil para los ingenieros la labor de seleccionar el µC mas adecuado para cada aplicación, pues se dispone de una tremenda oferta de micros de 8 bits, de 16 bits y en la actualidad de 32 bits. A su vez con arquitecturas más completas, capacidades de cálculo más elevadas y menores consumos de energía.

2.3.2. Comunicaciones

Los sistemas de comunicaciones adquieren, en el diseño de sistemas embebidos, cada vez mayor importancia. Lo normal es que el SE pueda comunicarse mediante interfaces estándar de comunicaciones por cable o inalámbricas. Así un SE normalmente incorpora puertos de comunicaciones bajo los estándares más extendidos, bien aquellos que necesitan de un cableado físico o se trate de comunicaciones inalámbricas. Podemos citar:

- RS-232
- RS-485
- SPI
- CAN
- USB
- Ethernet
- Fibra óptica.
- Comunicaciones inalámbricas (WiFi, WiMax, Bluetooth, GSM, GPRS, UMTS, DSRC, RFID, etc.)

2.3.3. Presentación

El subsistema presentación típico suele ser una pantalla gráfica, táctil, LCD alfanumérico, diodos LED, etc. Por lo general forma parte del interfaz hombre máquina del sistema, si es que lo lleva. El uso de pantallas gráficas del tipo táctil suele ser una solución muy aceptada, aunque conlleva mayor complejidad en el software a desarrollar y mayor potencia de cálculo de la CPU seleccionada. A continuación se muestra un típico display LCD alfanumérico de dos líneas por 20 caracteres por línea.

Display LCD alfanumérico

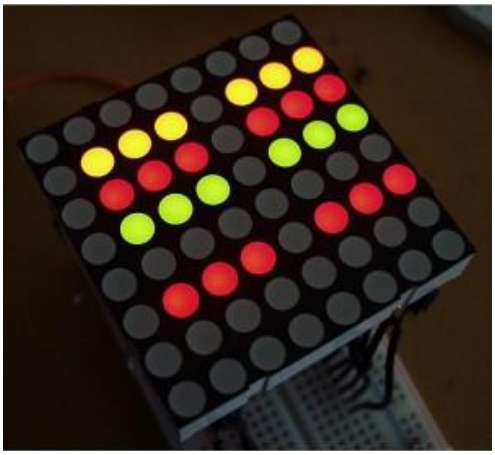
2.3.4. Actuadores

Denominamos actuadores a los posibles elementos encargados de llevar a cabo las acciones indicadas por la CPU. Entre éstos disponemos de drivers de corriente, controladores de motores eléctricos, conmutadores, relés, etc.

Actuadores de un robot

2.3.5. Entrada/Salida

El módulo de Entrada/Salida (I/O) se encarga de hacer llegar o enviar las señales analógicas y digitales a los diferentes circuitos encargados de su generación y procesamiento. Tal es el caso de la conversión A/D para el procesamiento digital de señales analógicas procedentes de sensores, activación de actuadores mediante circuitos 'driver', reconocimiento del estado abierto cerrado de un conmutador o pulsador, encendido de diodos LED, etc.

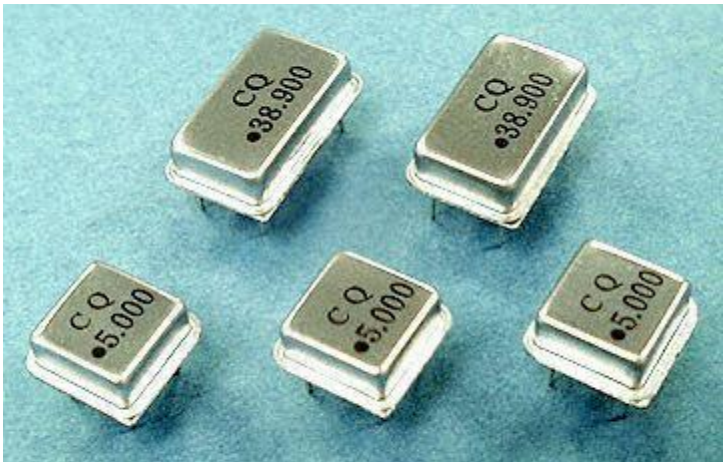


Matriz de diodos LED para iluminación

2.3.6. Reloj

El modulo de reloj es el encargado de generar las diferentes señales de reloj necesarias para la temporización de los circuitos digitales. Habitualmente se parte de un único oscilador principal, cuyas características son de vital importancia en determinadas aplicaciones. Aspectos a tener en cuenta en la selección del tipo de oscilador son:

- Frecuencia necesaria y la posible selección de ésta de forma automática.
- Estabilidad y precisión de la frecuencia con la temperatura, envejecimiento, vibraciones, etc.
- Consumo de corriente requerido y su complejidad hardware.
- El coste del resonador empleado para construirlo.



Módulos osciladores de cristal de cuarzo

Existen varios tipos de osciladores basados en el empleo de resonadores de cristal de cuarzo, cerámicos, SAW, LC y RC.

Cada uno de ellos presenta características específicas de margen de frecuencia, estabilidad y consumo. De entre todos, el oscilador con mejores características en cuanto a estabilidad de frecuencia y coste son los basados en resonador de cristal de cuarzo, mientras que los que requieren menor consumo son los RC. Además estos últimos es posible integrarlos dentro de un chip, lo que contribuye a minimizar el tamaño.

El uso de sintetizadores de frecuencia basados en el empleo de lazos enganchados en fase (PLL) permite disponer de un conjunto discreto de frecuencias con gran precisión y estabilidad, donde la selección de la frecuencia se realiza digitalmente.

NOTA: La frecuencia del oscilador normalmente no es la frecuencia a la que se ejecutan las instrucciones. Por ejemplo, en la familia PIC, cada instrucción necesita para ejecutarse 4 ciclos de reloj.

2.3.7. Módulo de alimentación

El módulo de energía (power) se encarga de generar las diferentes tensiones y corrientes necesarias para alimentar los componentes activos que forman el SE. Lo normal es el empleo de baterías para los dispositivos portátiles y fuentes de alimentación (conversor AC/DC) para los sistemas que disponen de acceso a la red de energía eléctrica.

Cuando son necesarias dos o más tensiones de valor especifico, mediante el empleo de conversores DC/DC se pueden obtener ésta a partir de una única tensión de entrada generada por una fuente o batería. Los valores típicos más empleados para alimentar los sistemas embebidos son 5, 9, 12 y 24 Vdc.

Dado que la conexión de la energía es clave en muchas aplicaciones y ésta a su vez es fuente de ruido e interferencias, a veces es necesario dotarla de filtros, circuitos integrados supervisores de alimentación, protectores de sobretensión, etc.

El consumo de energía puede ser determinante en el desarrollo de algunos SSEE, que necesariamente se alimentan con baterías y a las que no es posible recargar de forma continua. En aplicaciones donde es imposible su sustitución, la vida del SE suele estar limitada por la de las baterías, tal es el caso de los satélites artificiales, dotados de paneles solares y baterías.

2.4. Generalidades

La arquitectura de los dispositivos empleados en los SSEE ha evolucionado tremendamente en los últimos años, con el objetivo de conseguir mayor capacidad de procesamiento, de manejo de dispositivos, menor consumo, etc. Por otro lado, si bien hace unos años existía una diferencia nítida entre el concepto de microprocesador y de microcontrolador, en la actualidad, los sistemas fabricados en un solo chip han adquirido tal grado de complejidad que es muy difícil separar ambos conceptos.

Ya se ha comentado que un microprocesador lo definimos como la implementación en forma de circuito integrado (IC) de una Unidad Central de Proceso (CPU) junto con los buses de interconexión (el bus de control, el bus de direcciones y el bus de datos) mientras que un microcontrolador lo definimos como la implementación, dentro de un único chip, del microprocesador, la memoria y subsistemas de E/S.

También se han desarrollado los procesadores digitales de señal (DSP) que habitualmente se fabrican bajo un formato semejante al de un microcontrolador (CPU + Memoria + Periféricos de E/S en un solo chip) pero con una arquitectura especialmente diseñada para realizar la tareas más habituales en procesamiento digital de señales, de forma rápida.

Sin embargo existe una amplia gama de dispositivos pertenecientes a estos tres subconjuntos, cada vez con la inclusión de más funcionalidades, que hace a esta división cada vez más difusa.

La idea de los microcontroladores fue realizar dispositivos muy sencillos con muy poco hardware adicional, dedicados a tareas de control de dispositivos. Es por esto que los subsistemas de E/S debían ser muy completos mientras que la necesidades de memoria eran muy escasas. Se buscaba, a su vez, bajo coste y trabajo en tiempo real.

El diseño del SE consistirá en un modulo (PCB) que contiene varios circuitos integrados (chip’s) interconectados entre si y con el restos de componentes electrónicos pasivos y activos que se definen en el circuito eléctrico (esquemático) del SE. En general, un SE simple contará con un microprocesador, memoria, unos pocos periféricos de E/S y un programa dedicado a una aplicación concreta almacenado permanentemente en la memoria.

La arquitectura en la que se basan la inmensa mayoría de micros es del tipo Harvard con juegos de instrucciones reducido (RISC), pero existen notables diferencias entre cada familia de éstos.

Podemos resumir como principales características de las actuales familias de microprocesadores las siguientes:

- Uso de CPUs con arquitecturas de 8, 16, 32 y 64 bits.
- Gran cantidad de periféricos de E/S integrados.
- Incluyen memoria dentro del chip y necesidad de manejo de gran cantidad de memoria externa.
- Empleo de sistema operativo.
- Sistemas de desarrollo basados en leguajes de alto nivel tal como el C.
- Amplia gama de chips y gran variación en el coste.
- Se insertan en placas base, de tamaño cada vez más compacto, que incorporan una amplia gama de chips dedicados a las distintas funciones previstas.

También, las principales características actuales de los microcontroladores las podemos resumir en:

- Uso de CPUs con arquitecturas harvard de 2, 4, 8, 16 o 32 bits.
- Gran cantidad de periféricos de E/S integrados.
- Necesidad de poca memoria y generalmente no posibilidad de manejar memoria externa.
- No empleo de sistema operativo.
- Sistemas de desarrollo típicos basados en ANSI C.
- Bajo coste del chip.
- La placa (PCB) en la que se insertan suele ser sencilla en comparación con un formato PC, pues a veces solo contiene este único chip.

En el caso de los DSP’s, las podemos resumir en:

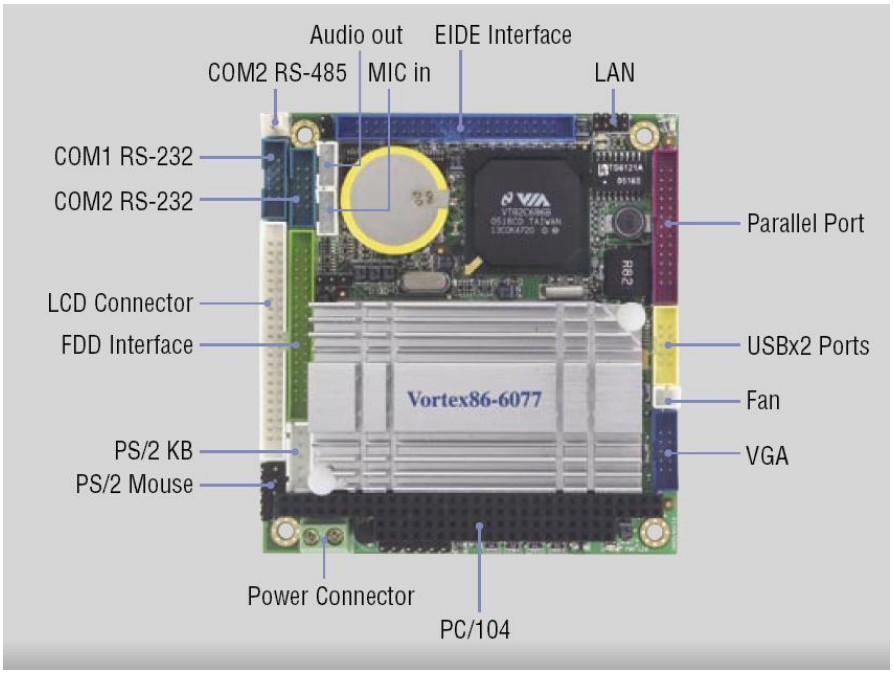
- Uso de CPUs con arquitecturas harvard modificadas de 16 y 32 bits. Versiones de coma fija y coma flotante.
- Gran cantidad de periféricos de E/S integrados, especializados en la transferencia en tiempo real de grandes volúmenes de datos.
- Incluyen gran cantidad de memoria dentro del chip.
- No empleo de sistema operativo.
- Sistemas de desarrollo típicos basados en ANSI C.
- Coste del chip medio.

- La placa base en la que se incorporan (PCB) suele tener una complejidad media y a veces forma parte de un sistema superior, como por ejemplo, la tarjeta de sonido de un PC.

2.5. Empleo de un PC embebido

Las opciones de diseño para un SE cada vez son mayores debido a los imparables avances tecnológicos en el sector de la electrónica y microelectrónica.

Así, cada vez el grado de integración de los dispositivos semiconductores, pasivos y electromecánicos es mayor y lo que hasta no hace mucho tiempo era impensable hoy en día es una realidad: Es posible incorporar como SE dentro de un determinado host a todo un modulo tipo PC en formato compacto, tipo Single Board Computer (SBC) dentro de determinadas aplicaciones que necesitan de una mayor complejidad. Existen en el mercado una amplia oferta de SBC's basados en la familia de procesadores de Intel: i386, i486, Pentium, etc. a un coste razonable para determinadas aplicaciones. La familia de procesadores ARM también surge como una opción especifica dirigida al sector de los SSEE portables, tales como las cámaras de fotos, móviles, PDA's, etc.



Aspecto de un PC embebido bajo formato estándar PC-104

Un PC embebido posee una arquitectura basada en éstos elementos básicos:

- **Microprocesador:** Encargado de realizar las operaciones de cálculo principales de sistema. Ejecuta código para realizar una determinada tarea y dirige el funcionamiento de los demás elementos que le rodean.

- **Memoria RAM:** Almacena el código de los programas que el sistema puede ejecutar así como los datos. Su característica principal es que debe tener un acceso de lectura y escritura lo más rápido posible para que el microprocesador no pierda tiempo en tareas que no son meramente de cálculo. Al ser volátil el sistema requiere de un soporte donde se almacenen los datos incluso sin disponer de alimentación o energía.

- **Memoria Caché:** Más rápida que la principal en la que se almacenan los datos y el código accedido últimamente. Dado que el sistema realiza micro tareas, muchas veces repetitivas, la caché consigue ahorrar tiempo ya que no hará falta ir a memoria principal si el dato o la instrucción ya se encuentra en la caché.

- **Memoria No volátil.** Habitualmente conocida como 'Disco duro', en él la información no es volátil y además puede conseguir capacidades muy elevadas. A diferencia de la memoria RAM que es de estado sólido éste suele ser magnético en aplicaciones tipo PC pero su excesivo tamaño y falta de robustez mecánica lo suele hacer inviable para PCs embebidos. Los avances tecnológicos, una vez mas, han conseguido resolver el problema desarrollando discos de estado sólido. Existen en el mercado varias soluciones de esta clase (DiskOnChip, CompactFlash, IDE Flash Drive, etc.) con capacidades suficientes para la mayoría de sistemas embebidos (hasta más de 20 GB).

- **BIOS-ROM:** Basic Input & Output System, sistema básico de entrada y salida) es código que es necesario para inicializar el ordenador y para poner en comunicación los distintos elementos de la placa madre. La ROM (Read Only Memory, memoria de sólo lectura no volátil) suele ser un chip donde se encuentra el código BIOS.

- **CMOS-RAM:** Es un chip de memoria de lectura y escritura alimentado con una pila donde se almacena el tipo y ubicación de los dispositivos conectados a la placa madre (disco duro, puertos de entrada y salida, etc.). Además contiene un reloj en permanente funcionamiento que ofrece al sistema la fecha y la hora.

- **Chip Set:** Es un chip que se encarga de controlar las interrupciones dirigidas al microprocesador, el acceso directo a memoria (DMA) y al bus ISA, además de ofrecer temporizadores, etc. Es frecuente encontrar la CMOS-RAM y el reloj de tiempo real en el interior del Chip Set.

- **Entradas al sistema:** pueden existir puertos para ratón, teclado, vídeo en formato digital, comunicaciones serie o paralelo, etc.

- **Salidas del sistema:** puertos de vídeo para monitor, pantallas de cristal líquido, altavoces, comunicaciones serie o paralelo, etc.

- **Ranuras de expansión** para tarjetas de tareas específicas que pueden no venir incorporadas en la placa madre, como pueden ser más puertos de comunicaciones, acceso a red de ordenadores vía LAN (Local Area Network, red de área local) o vía red telefónica: básica, RDSI (Red Digital de Servicios Integrados), ADSL (Asynchronous Digital Subscriber Loop, Lazo Digital Asíncrono del Abonado), etc. Un PC estándar suele tener muchas más ranuras de expansión que un PC embebido. Las ranuras de expansión están asociadas a distintos tipos de bus: VESA, ISA, PCI, NLX (ISA + PCI), etc.

2.6. Sistemas en un chip

Hoy en día existen en el mercado fabricantes que integran un microprocesador y los elementos controladores de los dispositivos fundamentales de entrada y salida en un mismo chip, pensando en las necesidades de los sistemas embebidos (bajo coste, pequeño tamaño, entradas y salidas específicas,...). Su capacidad de proceso suele ser inferior a los procesadores de propósito general pero cumplen con su cometido ya que los sistemas donde se ubican no requieren tanta potencia. Los principales fabricantes son ST Microelectronics (familia de chips STPC), National (familia Geode), Motorola (familia ColdFire) e Intel.

En cuanto a los sistemas operativos necesarios para que un sistema basado en microprocesador pueda funcionar y ejecutar programas suelen ser específicos para los sistemas embebidos. Así nos encontramos con sistemas operativos de bajos requisitos de memoria, posibilidad de ejecución de aplicaciones de tiempo real, modulares (inclusión sólo de los elementos necesarios del sistema operativo para el sistema embebido concreto), etc. Los más conocidos en la actualidad son Windows CE, QNX y VxWorks de WindRiver.

3. Redes CAN

El sistema de bus serie CAN, siglas cuyo significado en castellano es “Control de Área de Red”, nació en febrero de 1986, cuando el grupo “Robert Bosch GmbH” (más conocido como “Bosch” a secas) lo presentó en el congreso de la “Sociedad de Ingeniería de la Automoción”. Desde entonces, CAN se ha convertido en uno de los protocolos líderes en la utilización del bus serie.

3.1. Introducción

A comienzos de los 80, los ingenieros de Bosch evaluaron el posible uso de los sistemas de bus serie existentes en los coches de pasajeros, pero ninguno de los protocolos de red disponibles satisfacía los requisitos de estos.

El ingeniero Uwe Kiencke fue quien inició el desarrollo del nuevo sistema en 1983.

Este protocolo de bus serie se ideó principalmente para aportar mayor funcionalidad, seguridad y fiabilidad, junto a una mayor eficiencia en el gasto del combustible, ya que la reducción del peso y la complejidad de los automóviles a través de la reducción del cableado iban a favorecer este hecho.

Ingenieros de “Mercedes-Benz” pronto se involucraron en las primeras fases de creación del nuevo sistema, y pronto lo hizo también “Intel” como potencial vendedor de semiconductores. El profesor Wolfhard Lawrenz de la universidad de ciencias aplicadas de Brunswick-Wolfenbüttel, Alemania, fue quién dio al nuevo protocolo de red el nombre de “Controller Area Network” (CAN).

A mediados de 1987, “Intel” presentó el primer chip de controlador CAN: el 82526. Poco tiempo después, “Semiconductores Philips” presentaría el 82C200. Estos dos antepasados primigenios de los controladores CAN actuales eran completamente distintos en cuanto a filtros de aceptación y control de mensajes:

Intel adoptó el concepto de FullCAN; este requería menos carga de la CPU del microcontrolador que la implementación BasicCAN elegida por Philips. Pero por otra parte, el dispositivo de FullCAN era limitado con respecto al número de los mensajes que podían ser recibidos. Además, el controlador de BasicCAN requería menos silicio, lo que abarataba aun más su coste. Actualmente, los términos 'BasicCAN' y 'FullCAN' han quedado obsoletos.

Estandarización

La especificación CAN (versión 2.0) de "Bosch" fue sometida a la estandarización internacional a comienzos de los 90.

Concretamente en Noviembre de 1993, después de diversos conflictos políticos, se publicó el estándar ISO 11898, que definía además una capa física para velocidades de hasta 1 Mbit/s.

Paralelamente, un formato de CAN 'tolerante a fallos' se incluyó en la ISO 11519-2. En 1995, el estándar se amplió con la descripción del identificador CAN de 29 bits.

Desafortunadamente, todas las especificaciones y estandarizaciones publicadas acerca de CAN contenían errores o estaban incompletas. Para evitar incompatibilidades, "Bosch" se cercioró, y sigue haciéndolo, de que todos los micros CAN cumplen con el modelo de referencia que ellos definieron.

De todas formas, las especificaciones CAN han sido revisadas y estandarizadas con el tiempo en diferentes secciones: la norma ISO 11898-1 describe la 'capa de transmisión de datos CAN'; la ISO 11898-2 la 'capa física CAN no tolerante a fallos'; y la ISO 11898-3 la 'capa física CAN tolerante a fallos'. Los estándares de ISO 11992 (referente a la interfaz para camiones y remolques) e ISO 11783 (referente a la maquinaria agrícola y forestal) definen los perfiles del uso de CAN basados en el US-protocol J1939.

Pioneros

A pesar de que CAN surgió con la idea de ser utilizado en coches de pasajeros, un gran número de las primeras aplicaciones llegarían desde otros segmentos de mercado distintos:

Entre otros, el fabricante finlandés de ascensores "Kone", fue uno de los primeros en aprovecharse de sus ventajas. La oficina de ingeniería Suiza "Kyaser" lo introdujo a los fabricantes de maquinaria textil del país, que acabaron fundando el 'Grupo de usuarios textiles CAN', bajo el liderazgo de 'Lars-Berno Fredriksson'. En Holanda, "Sistemas médicos Philips" lo utilizó para las redes internas de sus máquinas de Rayos X. La 'especificación para mensajes de Philips' (PMS), desarrollada principalmente por Tom Suters, representó la primera capa de aplicación para redes CAN.

También se sucedieron todo tipo de propuestas académicas, como por ejemplo, la creación a finales de los 80 de un sistema de bus basado en CAN, para vehículos agrícolas (LBS).

Ya en 1992, Mercedes-Benz implantó CAN en sus automóviles de clase alta. Y aunque en un principio el bus se limitaba a interconectar las unidades de control electrónico que cuidaban del

control del motor, pronto conectarían además las unidades de control necesarias para los componentes electrónicos. Para ello, se implementaron dos sistemas de bus CAN separados físicamente, y conectados a través de 'gateways'. Actualmente, muchos otros fabricantes de coches como "BMW", "Renault", "Fiat", "Saab", "Volkswagen" o "Volvo", han seguido su ejemplo y suelen implementar dos redes CAN en sus coches.

En Marzo de ese mismo año, usuarios internacionales y grupos de fabricantes fundaron oficialmente la organización 'CAN en la Automoción' (CiA).

La primera publicación técnica, que trataba acerca de la capa física, fue emitida solo unas semanas después: 'CiA' recomendaba utilizar solamente transceptores CAN que cumplieran la normativa ISO 11898. A día de hoy, son los transceptores RS485 los utilizados más comúnmente.

Otra de las primeras tareas de la CiA fue la especificación de la 'capa de aplicación CAN' (CAL), que se desarrolló a partir del material existente de los "Sistemas médicos de Philips" y de "STZP".

Capas de aplicación

Desde 1993, dentro del alcance del proyecto 'ASPIC', un consorcio europeo liderado por "BOSCH", desarrolló un prototipo conocido como CANopen para las redes internas de las celdas de producción. En 1995, se publicó el perfil totalmente revisado de las comunicaciones de CANopen, que en el plazo de cinco años ya se había convertido en la red integrada estandarizada más importante de Europa, puesto que ofrecía una gran flexibilidad y un gran número de opciones de configuración, así como diversos perfiles de dispositivo, interfaz y uso.

Actualmente, CANopen se sigue utilizando especialmente en Europa: máquinas de moldeado por inyección en Italia, máquinas expendedoras en Inglaterra, grúas en Francia, control de maquinaria en Austria, o maquinaria de fabricación de relojes en Suiza... Mientras que en Estados Unidos CANopen se abre camino en el ámbito de los toros mecánicos y las máquinas clasificadoras.

Poco después de aquello, a comienzos de 1994, "Allen-Bradley" desarrolló la tecnología DeviceNet, enfocada esencialmente a la automatización de las fábricas. Pronto ganaría adeptos en Estados Unidos debido sobre todo a su funcionamiento "plug & play" (enchufar y listo).

Pero no fueron las únicas. Desde el momento de la creación de CAL, se fue sucediendo la creación de diferentes estándares que definían la capa de aplicación; algunos son muy específicos y están relacionados casi exclusivamente con sus campos de aplicación. Dejando de lado las 3 mencionadas anteriormente, entre los protocolos de alto nivel y las capas de aplicación más utilizadas cabe mencionar: SDS, OSEK y CANKingdom.

Por una parte, SDS (Smart Distributed System), fue creado por Honeywell durante la gestación de DeviceNet, y de hecho era muy similar a este, así que no entraremos en detalle. OSEK, que estaba basado además en TCP/IP y algoritmos DSP, rápidamente enfocaría su aplicación a una gran cantidad de usos en los turismos. Mientras que CANKingdom se enfocó a

aplicaciones con una necesidad alta de aplicaciones en tiempo real, como pueda ser por ejemplo la electrónica marítima.

Ya por último, en el año 2000, ISO formó un grupo de trabajo que agrupaba empleados de "Bosch", académicos y expertos de la industria de los semiconductores, que definió un nuevo protocolo basado en CAN, el llamado TTCAN (Time-tiggered communication on CAN). Esta extensión permitía tanto la transmisión de mensajes simultáneos, como la implementación de un bucle cerrado para el control del bus. Y gracias a que el protocolo del bus no fue modificado, este tipo de mensajes podían seguir siendo enviados con el mismo sistema físico de bus.

3.1.1. Características Básicas

- **Económico y sencillo:** Dos de las razones que motivaron su desarrollo fueron precisamente la necesidad de economizar el coste monetario y el de minimizar la complejidad del cableado, por parte del sector automovilístico.

- **Estandarizado:** Se trata de un estándar definido en las normas ISO (Internacional Organization for Standardization), concretamente la ISO11898, que se divide a su vez en varias partes, cada una de las cuales aborda diferentes aspectos de CAN.

- **Medio de transmisión adaptable:** El cableado, como ya hemos dicho, es muy reducido a comparación de otros sistemas. Además, a pesar de que por diversas razones el estándar de hardware de transmisión sea un par trenzado de cables, el sistema de bus CAN también es capaz de trabajar con un solo cable. Esta particularidad es empleada en diversos tipos de enlaces, como los enlaces ópticos o los enlaces de radio.

- **Estructura definida:** La información que circula entre las unidades a través de los dos cables (bus) son paquetes de bits (0's y 1's) con una longitud limitada y con una estructura definida de campos que conforman el mensaje.

- **Programación sencilla.**

- **Número de nodos:** Es posible conectar hasta 110 dispositivos en una sola red CAN.

- **Garantía de tiempos de latencia:** CAN aporta la seguridad de que se transmitirá cierta cantidad de datos en un tiempo concreto, es decir, que la latencia de extremo a extremo no excederá un nivel específico de tiempo. Además, la transmisión siempre será en tiempo real.

- **Optimización del ancho de banda:** Los métodos utilizados para distribuir los mensajes en la red, como el envío de estos según su prioridad, contribuyen a un mejor empleo del ancho de banda disponible.

- **Desconexión autónoma de nodos defectuosos:** Si un nodo de red cae, sea cual sea la causa, la red puede seguir funcionado, ya que es capaz de desconectarlo o aislarlo del resto. De forma contraria, también se pueden añadir nodos al bus sin afectar al resto del sistema, y sin necesidad de reprogramación.

- **Velocidad flexible:** ISO define dos tipos de redes CAN: una red de alta velocidad (de hasta 1 Mbps) definida por la ISO 11898-2, y una red de baja velocidad tolerante a fallos (menor o igual a 125 Kbps) definida por la ISO 11898-3.

- **Relación velocidad-distancia:** Al punto anterior habría que añadir que la velocidad también depende de la distancia hasta un máximo de 1000 metros (aunque podemos aumentar la distancia con bridges o repetidores), como podemos corroborar en la siguiente tabla comparativa:

Velocidad (Kbps)	Tiempo de bit (µS)	Longitud máxima de bus (m)
1000	1	30
800	1.25	50
500	2	100
250	4	250
125	8	500
50	20	1000
20	50	2500
10	100	5000

- **Orientado a mensajes:** Se trata de un protocolo orientado a mensajes, y no a direcciones, es decir, la información que se va a intercambiar se descompone en mensajes, a los cuales se les asigna un identificador y son encapsulados en tramas para su transmisión. Cada mensaje tiene un identificador único dentro de la red, a partir del cual los nodos deciden aceptar o no dicho mensaje. Además están priorizados.

- **Multidifusión (multicast):** Permite que todos los nodos puedan acceder al bus de forma simultánea con sincronización de tiempos. Medio compartido (broadcasting): La información es enviada en la red a todos los destinos de forma simultánea. Así que los destinos habrán de saber si la información les concierne o deben rechazarla.

- **Detección y señalización de errores:** CAN posee una gran capacidad de detección de errores, tanto temporales, como permanentes, lograda a través de cinco mecanismos de

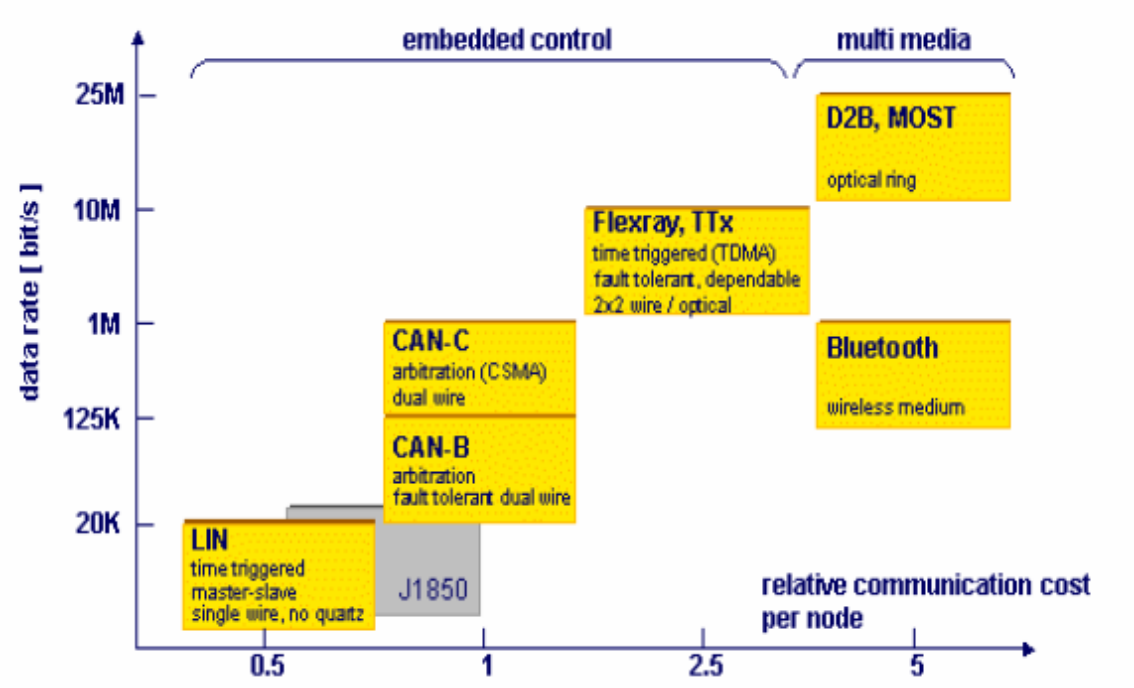
detección, 3 al nivel de mensaje y 2 a nivel de bit. Los errores además pueden ser señalizados.

- **Retransmisión automática de tramas erróneas:** Junto a la detección y señalización de errores la retransmisión automática de tramas erróneas aporta la integridad de los datos. Además ambos procesos son transparentes al usuario.

- **Jerarquía multimaestro:** CAN es un sistema multimaestro en el cual puede haber más de un maestro (o master) al mismo tiempo y sobre la misma red, es decir, todos los nodos son capaces de transmitir, hecho que permite construir sistemas inteligentes y redundantes.

3.1.2. Presente y Futuro

Realmente CAN se encuentra en una posición privilegiada en el mercado. El siguiente gráfico traza una comparativa entre diferentes tecnologías respecto al coste por nodo.



Se observa que J1850, pero sobretodo LIN, que tiene una mayor presencia en la industria, son opciones más económicas, ¿pero hasta qué punto son estas tecnologías competidoras de CAN?

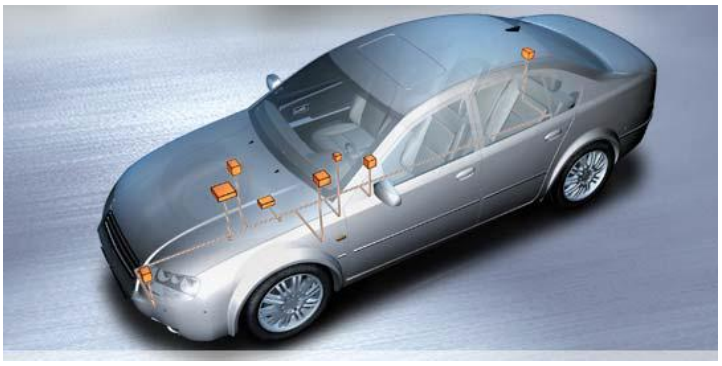
En la siguiente tabla, se exponen las diferentes características de LIN y CAN (y de cómo extra el I2C de "Philips"), para detallar la similitud entre ellas:

	CAN	**LIN**	**I2C**
Compañía que lo desarrolló	Bosh	Open source	Philips
Velocidad	1Mb/s	20 Kb/s	0.1Mb/s / 0.4Mb/s
Tamaño de datos	64bits	8bits	8bits
Prioridad de mensajes	Si	No	No
Garantía de latencia	Si	***	No
Flexibilidad en la configuración	Si	***	Si
Sistema Multimaestro	Si	No	Si
Detección y señalización de errores	Si	Si	Si
Retransmisión de tramas	Automática	No	Programable

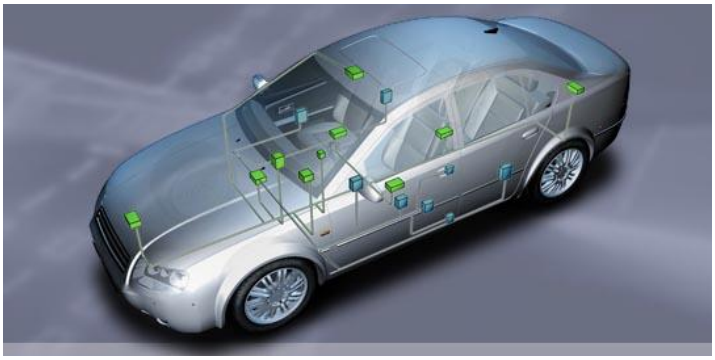
Se puede llegar a la conclusión fácilmente, de que cada una va a ser útil en un ámbito de manufactura distinto, ya que sus características son distintas. La diferencia principal, además de la económica, es la velocidad.

Dentro de la industria automovilística por ejemplo, CAN es empleado como bus de comunicaciones para los elementos más importantes de este, que van a requerir una seguridad y velocidad mayores. Para otro tipo de vicisitudes, como los elevalunas eléctricos por ejemplo, que tienen mucha menos importancia que la que puedan tener los frenos o el estado del motor, se emplea el bus LIN.

Uso general del bus CAN en los coches:



Uso general del bus LIN en los coches:



En definitiva, más que competir, lo que hacen es repartirse el pastel, ya que al tener diferentes características, el fabricante va a poder adaptarse en todo momento de la mejor forma a las exigencias del mercado, eligiendo uno u otro para cada necesidad.

Ya que otro grupo de la clase ha optado por hacer un trabajo acerca de Ethernet, vamos a nombrar algunas de las diferencias más importantes que hay entre estos, aunque no sean competidores directos:

CAN – Ethernet

Aun cuando las similitudes entre CAN y Ethernet son obvias, CAN posee algún beneficio clave cuando se compara con el protocolo Ethernet. El esquema de arbitraje que usa el protocolo CAN es de bit inteligente no destructivo. Esto significa que se comparan los mensajes con cada bit en un momento determinado, pero el mensaje con la prioridad más alta no se destruye y se retransmite; sólo el mensaje que no gana el arbitraje de bus se detiene y se retransmite. Éste es un punto importante que ayuda a minimizar el tiempo de fuera de servicio del bus y aumentan al máximo uso eficaz del ancho de banda disponible.

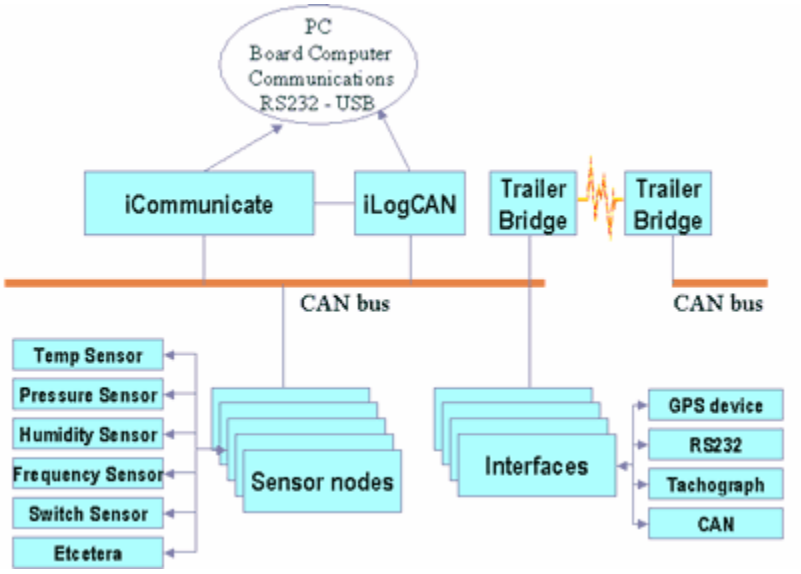
Otra ventaja importante del arbitraje del bit inteligente no destructivo, que usa CAN, es el hecho que esto da al bus características muy predecibles. Con Ethernet, ambos transmisores se detienen cuando se detecta una colisión y una cantidad de tiempo aleatorio se permite pasar antes de que ambos prueben de la retransmisión. Con este elemento aleatorio de tiempo eliminado de la función del bus, es posible lograr casi el 100% de eficacia en términos de utilización del ancho de banda.

Futuro

A día de hoy, el futuro de CAN se prevé esperanzador, ya que incluso las estimaciones más conservadoras coinciden en que la presencia de CAN en el mercado y en diversos campos de la industria, va a seguir en aumento durante los próximos diez o quince años.

3.1.3. Campos de aplicación

El 80% de las aplicaciones modernas del bus CAN se pueden encontrar en la ingeniería del automóvil y otro tipo de vehículos como autobuses, trenes y aviones. En el caso de los automóviles por ejemplo, CAN es el encargado de la comunicación y automatización del sistema de freno, los faros, el ABS, o el ordenador de a bordo, del cual vemos su esquema en el siguiente gráfico:



Esquema bus CAN en un vehículo

Sin embargo, también se puede encontrar el bus CAN en aplicaciones de diversa índole debido a su naturaleza, que le aporta robustez, economía y un altísimo grado de seguridad y fiabilidad; entre las más comunes:

Control y automatización industrial:

- Redes entre diversas máquinas y elementos de las mismas.
- Redes de supervisión.
- Redes de seguridad.

Control y automatización de edificios:

- Control de ascensores, puertas mecánicas, aspersores y diversos elementos mecánicos.
- Control de iluminación.

Aplicaciones específicas:

- Control de máquinas expendedoras (en Inglaterra está muy extendido su uso).
- Control de equipamiento médico.

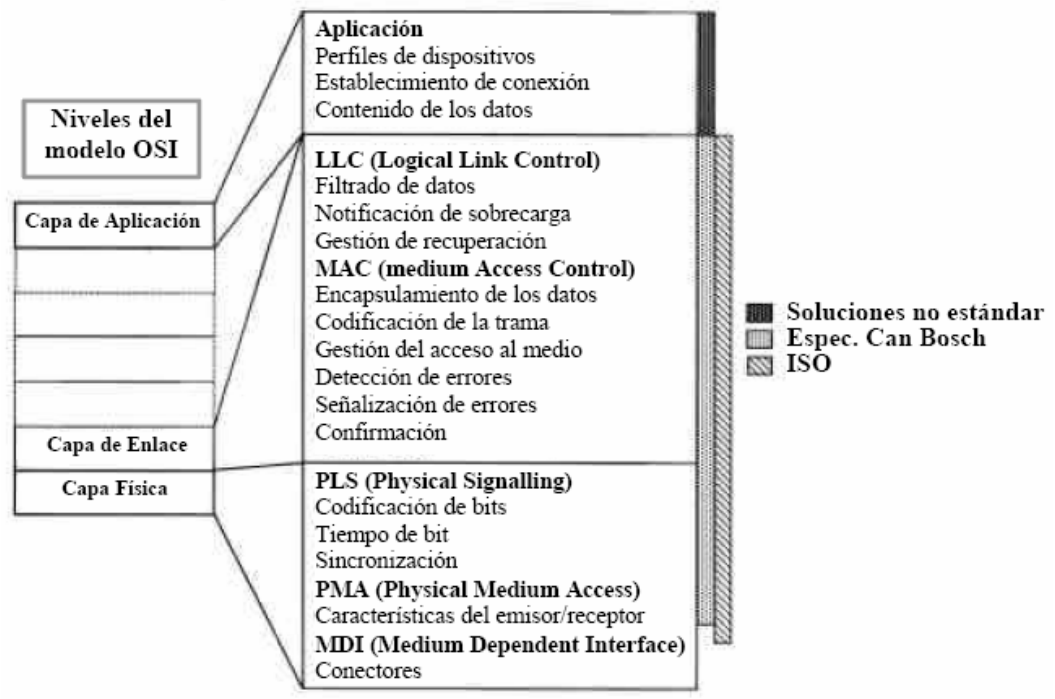
- Control de sistemas automáticos de almacenaje.
- Control de electrodomésticos.

3.2. Trasmisión de datos

3.2.1. Arquitectura de capas

Las implementaciones hardware de CAN cubren de forma estandarizada las capas físicas y de enlace del modelo de comunicaciones OSI (Open Systems Interconection), mientras diversas soluciones de software no estandarizadas cubren la capa de aplicación.

Las estandarizaciones ISO (International Standard Organization), a diferencia de las normas BOSCH, especifican también el medio de comunicación. Por lo tanto una implementación CAN a partir de las especificaciones de BOSCH no siempre será compatible con las normas ISO.



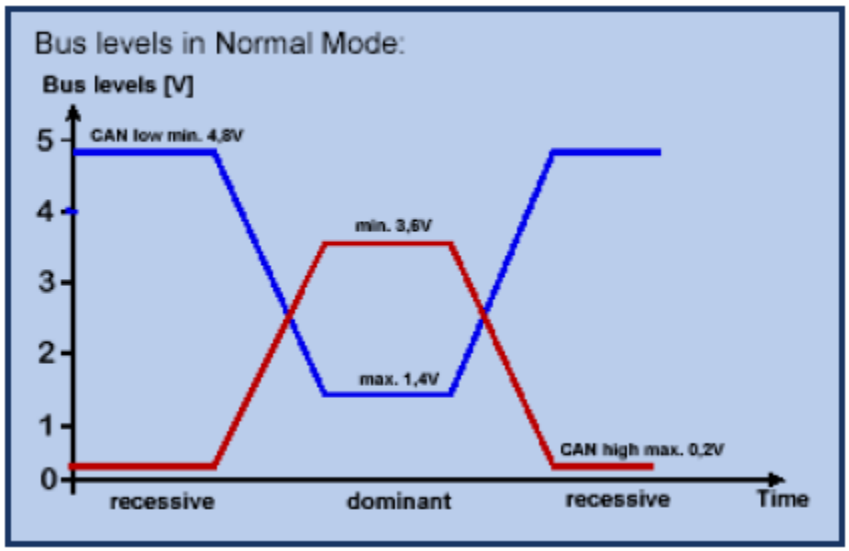
Niveles de modelo OSI

3.2.1.1. Capa Física

La capa física de CAN, es responsable de la transferencia de bits entre los distintos nodos que componen la red. Define aspectos como niveles de señal, codificación, sincronización y tiempos en que los bits se transfieren al bus.

En la especificación original de CAN, la capa física no fue definida, permitiendo diferentes opciones para la elección del medio y niveles eléctricos de transmisión. Las características de las señales eléctricas en el bus, fueron establecidas más tarde por el ISO 11898 para las aplicaciones de alta velocidad y, por el estándar ISO 11519 para las aplicaciones de baja velocidad.

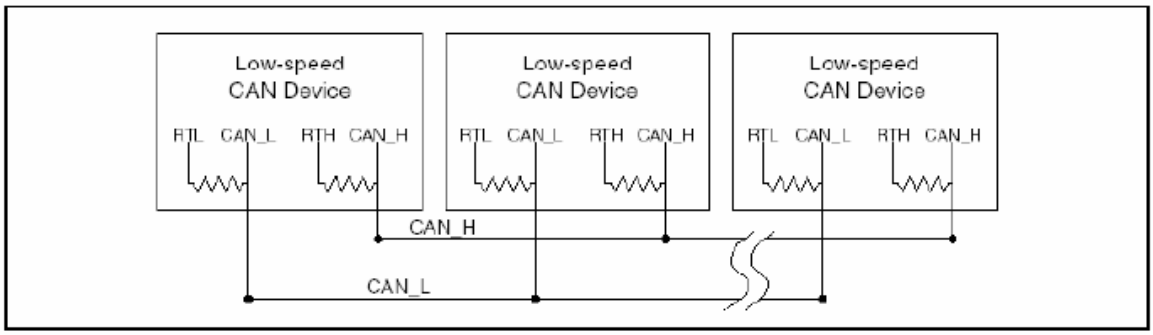
Estándar 11519



Los nodos conectados en este bus interpretan dos niveles lógicos denominados: Dominante y Recesivo.

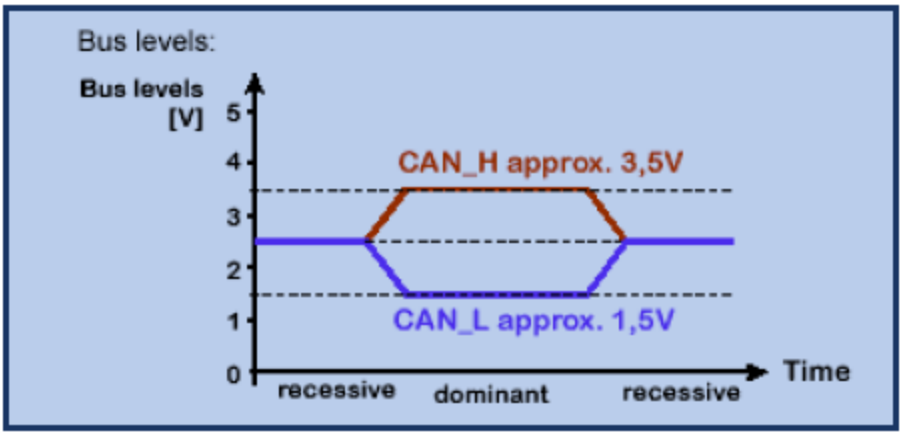
- Dominante: la tensión diferencial (CAN_H - CAN_L) es del orden de 2.0 V con CAN_H = 3.5V y CAN_L = 1.5V (nominales).
- Recesivo: la tensión diferencial (CAN_H - CAN_L) es del orden de 5V con CAN_H = 0V y CAN_L = 5V (nominales).

A diferencia del bus de alta velocidad, el bus de baja velocidad requiere dos resistencias en cada transceptor: RTH para la señal CAN_H y RTL para la señal CAN_L. Esta configuración permite al transceptor de bus de baja velocidad (fault-tolerant) detectar fallas en la red. La suma de todas las resistencias en paralelo, debe estar en el rango de 100-500.



Red Bus CAN de Baja Velocidad (Fault Tolerant)

Estándar 11898



Los nodos conectados en este bus interpretan los siguientes niveles lógicos:

- Dominante: la tensión diferencial (CAN_H - CAN_L) es del orden de 2.0 V con CAN_H = 3.5V y CAN_L = 1.5V (nominales).
- Recesivo: la tensión diferencial (CAN_H - CAN_L) es del orden de 0V con CAN_H = CAN_L = 2.5V (nominales).

El par de cables trenzados (CAN_H y CAN_L) constituyen una transmisión de línea. Si dicha transmisión de línea no está configurada con los valores correctos, cada trama transferida causa una reflexión que puede originar fallos de comunicación. Como la comunicación en el bus

CAN fluye en ambos sentidos, ambos extremos de red deben de estar cerrados mediante una resistencia de 120. Ambas resistencias deberían poder disipar 0.25 w. de potencia.

3.2.1.2. Capa de Enlace

La capa de enlace de datos es responsable del acceso al medio y el control lógico y está dividida a su vez en dos niveles.

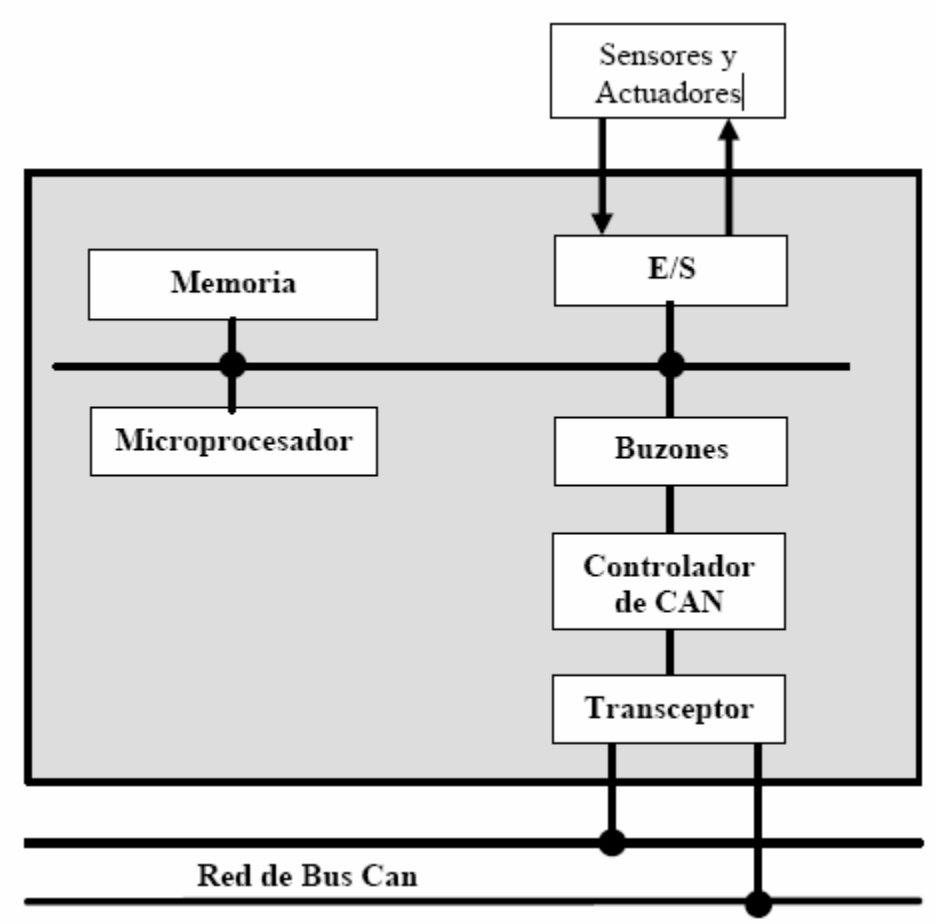
- **El subnivel LLC (Logical Link Control):** Gestiona el filtrado de los mensajes, las notificaciones de sobrecarga y la administración de la recuperación.

- **El subnivel MAC (Medium Acces Control):** Es el núcleo del protocolo CAN y gestiona el tramado y desentramado de los mensajes, el arbitraje a la hora de acceder al bus y el reconocimiento de los mensajes, así como el chequeo de posibles errores y su señalización, el aislamiento de fallos en unidades de control y la identificación del estado libre del bus para iniciar una transmisión o recepción de un nuevo mensaje.

3.2.2. Estructura de un Nodo CAN

Dentro de un nodo CAN, se pueden distinguir una serie de módulos interconectados entre ellos: un bus de direcciones, datos y un control (paralelo) enlazando el controlador central, la memoria de los datos y el programa (donde está almacenado el software de aplicación y el controlador de red de alto nivel), los dispositivos de entrada y salida y, la interfaz de comunicación.

Desde el punto de vista del controlador, la interfaz de comunicación se puede ver como un conjunto de buzones, donde cada uno de estos sirve como registro lógico de interfaz entre el controlador local y los nodos remotos. Si un nodo quiere comunicarse, tiene que dar de alta los correspondientes buzones de recepción y transmisión antes de hacer ninguna operación.



Esquema de un Nodo CAN

Teniendo en cuenta esta arquitectura, el funcionamiento sigue los siguientes pasos:

- Para inicializar, el programador especifica los parámetros de los registros de control de interfaz de comunicación, como las características del controlador de red o la velocidad de transmisión.
- A continuación, se actualizan todos los buzones. En cada buzón se especifica si es receptor o transmisor y, su estado inicial, inicializando su parte de datos del búfer.
- Posteriormente, para transmitir un mensaje es necesario poner los datos en el búfer de datos correspondiente al buzón de transmisión y activar el flanco de transmisión.
- Por último, la interfaz de red intenta comunicar los datos a través de la red. El estado de la transferencia se puede comprobar en el estatus de estado de cada buzón.

Una vez hecha la configuración inicial a partir del software de la capa de aplicación de esta manera, los nodos CAN funcionarán de forma autónoma.

3.2.3. Codificación y Sincronización

La codificación de bits se realiza por el método NRZ (Non-Return-to Zero) que se caracteriza por que el nivel de señal puede permanecer constante durante largos periodos de tiempo y habrá que tomar medidas para asegurarse de que el intervalo máximo permitido entre dos señales no es superado. Esto es importante para la sincronización (Bit Timing).

Este tipo de codificación requiere poco ancho de banda para transmitir, pero en cambio, no puede garantizar la sincronización de la trama transmitida. Para resolver esta falta de sincronismo se emplea la técnica del "bit stuffing": cada 5 bits consecutivos con el mismo estado lógico en una trama (excepto del delimitador de final de trama y el espacio entre tramas), se inserta un bit de diferente polaridad, no perdiéndose así la sincronización. Por otro lado este bit extra debe ser eliminado por el receptor de la trama, que sólo lo utilizará para sincronizar la transmisión.

No hay flanco de subida ni de bajada para cada bit, durante el tiempo de bit hay bits dominantes ("0") y recesivos ("1") y disminuye la frecuencia de señal respecto a otras codificaciones.

3.2.4. Tramas

3.2.4.1. Tipos de Tramas

El protocolo CAN está basado en mensajes, no en direcciones. El nodo emisor transmite el mensaje a todos los nodos de la red sin especificar un destino y todos ellos escuchan el mensaje para luego filtrarlo según le interese o no.

Existen distintos tipos de tramas predefinidas por CAN para la gestión de la transferencia de mensajes:

- **Trama de datos:** Se utiliza normalmente para poner información en el bus y la pueden recibir algunos o todos los nodos.

- **Trama de información remota:** Puede ser utilizada por un nodo para solicitar la transmisión de una trama de datos con la información asociada a un identificador

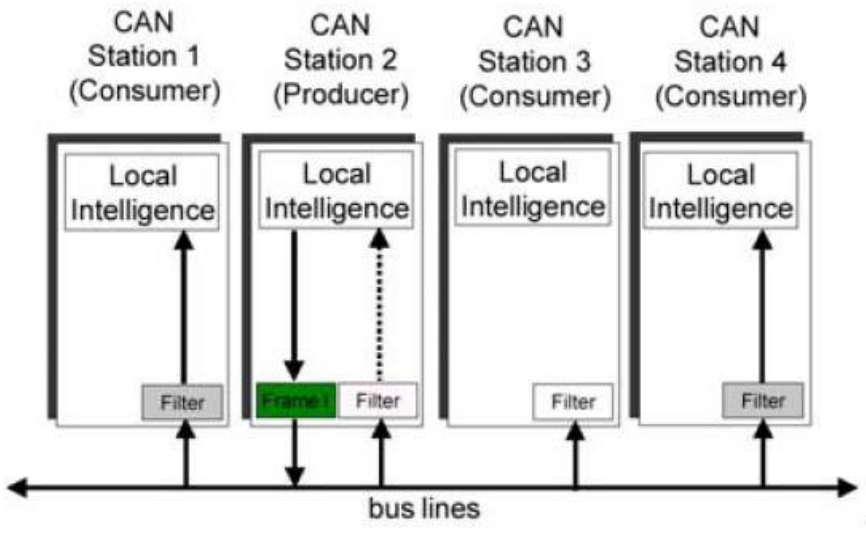
dado. El nodo que disponga de la información definida por el identificador la transmitirá en una trama de datos

- **Trama de error:** Se generan cuando algún nodo detecta algún error definido.
- **Trama de sobrecarga:** Se generan cuando algún nodo necesita más tiempo para procesar los mensajes recibidos.
- **Espaciado entre tramas:** Las tramas de datos (y de interrogación remota) se separan entre sí por una secuencia predefinida que se denomina espaciado inter-trama.
- **Bus en reposo:** En los intervalos de inactividad se mantiene constantemente el nivel recesivo del bus.

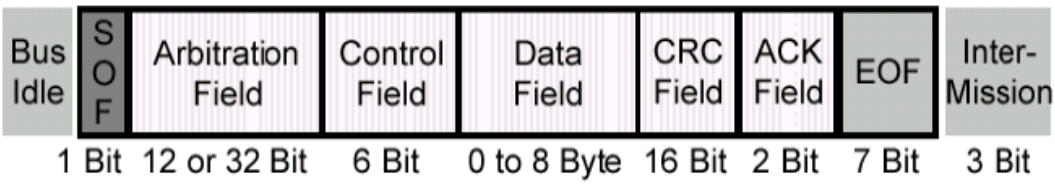
En un bus CAN los nodos transmiten la información espontáneamente con tramas de datos, bien sea por un proceso cíclico o activado ante eventos en el nodo. La trama de interrogación remota sólo se suele utilizar para detección de presencia de nodos o para puesta al día de información en un nodo recién incorporado a la red. Los mensajes pueden entrar en colisión en el bus, el de identificador de mayor prioridad sobrevivirá y los demás son retransmitidos lo antes posible.

3.2.4.2. Trama de datos

Es la utilizada por un nodo normalmente para poner información en el bus. Puede incluir entre 0 y 8 bytes de información útil.

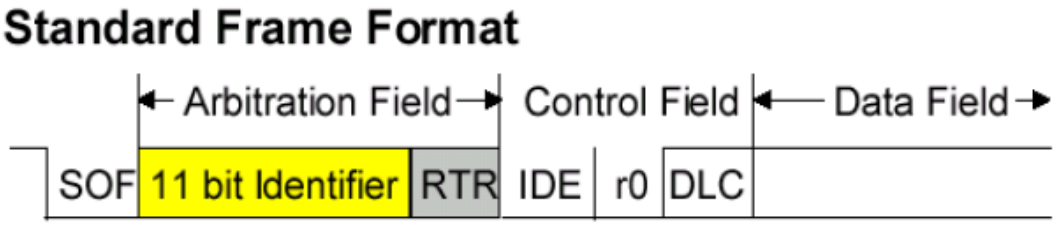


Los mensajes de datos consisten en celdas que envían datos y añaden información definida por las especificaciones CAN:

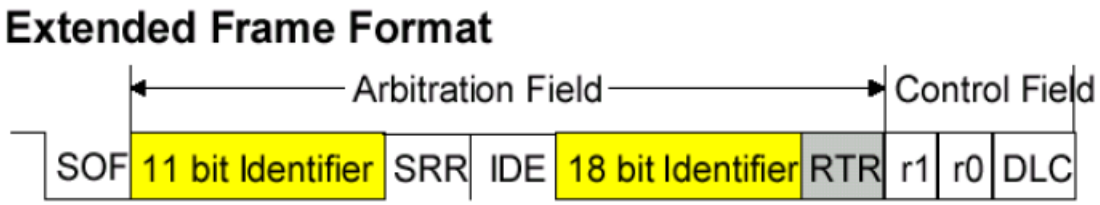


- Inicio de trama (SOF): El inicio de trama es una celda de un sólo bit siempre dominante que indica el inicio del mensaje, sirve para la sincronización con otros nodos.
- Celda de Arbitraje (Arbitration Field): Es la celda que concede prioridad a unos mensajes o a otros:

En formato estándar tendrá 11 bits seguidos del bit RTR (Remote Transmisión Request) que en este caso será dominante.



En formato extendido serán 11 bits de identificador base y 18 de extendido. El bit SRR substituye al RTR y será recesivo.



NOTA: La trama en formato estándar prevalece sobre la extendida

- Celda de control (Control Field): El campo de control está formado por dos bits reservados para uso futuro y cuatro bits adicionales que indican el número de bytes de datos. En realidad el primero de estos bits (IDE) se utiliza para indicar si la trama es de CAN Estándar (IDE dominante) o Extendido (IDE recesivo). El segundo bit (RB0) es siempre recesivo. Los cuatro bits de código de longitud (DLC) indican en binario el número de bytes de datos en el mensaje (0 a 8).

- Celda de Datos (Data Field): Es el campo de datos de 0 a 8 bytes.

- CRC: Código de redundancia cíclica: Tras comprobar este código se podrá comprobar si se han producido errores.

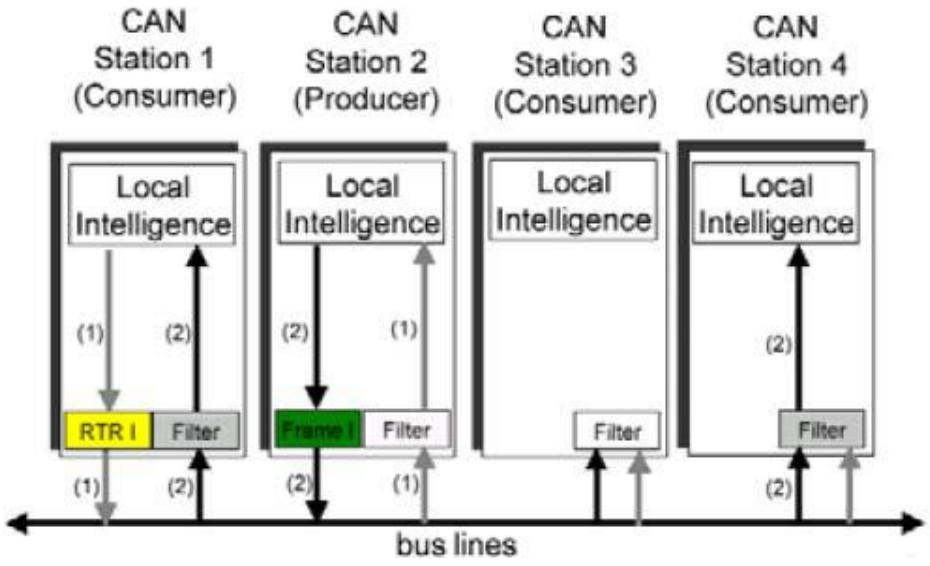
- Celda de reconocimiento (ACK): es un campo de 2 bits que indica si el mensaje ha sido recibido correctamente. El nodo transmisor pone este bit como recesivo y cualquier nodo que reciba el mensaje lo pone como dominante para indicar que el mensaje ha sido recibido.

- Fin de trama (EOF): Consiste en 7 bits recesivos sucesivos e indica el final de la trama.

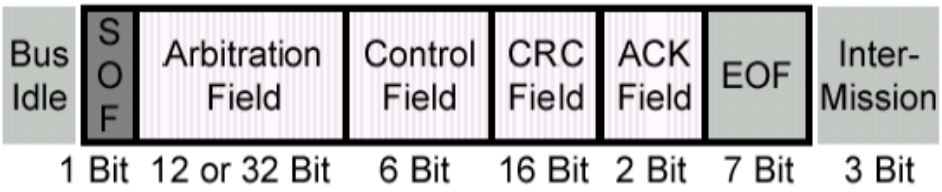
- Espaciado entre tramas (IFS): Consta de un mínimo de 3 bits recesivos.

3.2.4.3. Trama remota

Los nodos tienen habilidad para requerir información a otros nodos. Un nodo pide una información a los otros y el nodo que tiene dicha información envía una comunicación con la respuesta que puede ser recibida además por otros nodos si están interesados.



Un mensaje de petición remota tiene la siguiente forma:



En este tipo de mensajes se envía una trama con el identificador del nodo requerido, a diferencia con los mensajes de datos, el bit RTR toma valor recesivo y no hay campo de datos.

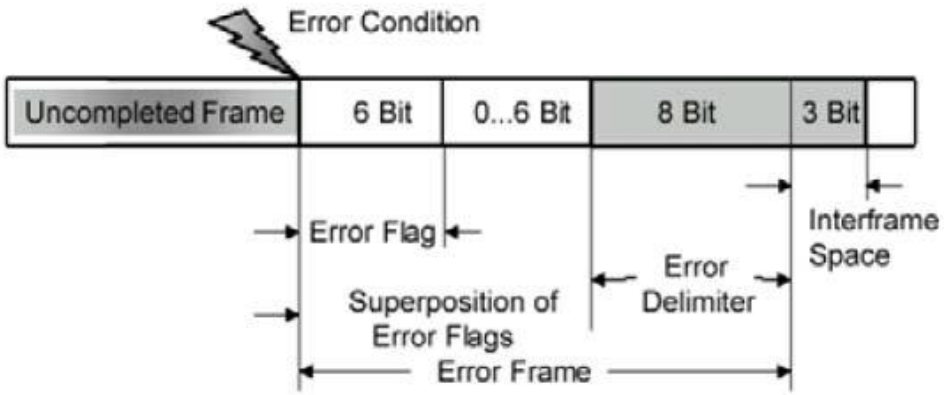
En caso de que se envíe un mensaje de datos y de petición remota con el mismo identificador, el de datos ganará el acceso al bus puesto que el RTR lleva valor dominante.

3.2.4.4. Trama de error

Las tramas de error son generadas por cualquier nodo que detecta un error. Consiste en dos campos: Indicador de error ("Error Flag") y Delimitador de error (“Error Delimeter”).

El delimitador de error consta de 8 bits recesivos consecutivos y permite a los nodos reiniciar la comunicación limpiamente tras el error.

El Indicador de error es distinto según el estado de error del nodo que detecta el error:

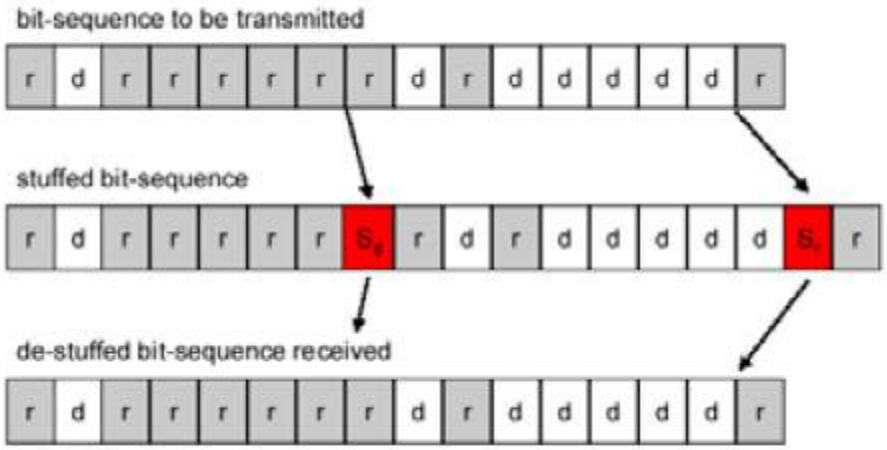


Si un nodo en estado de error "Activo" detecta un error en el bus interrumpe la comunicación del mensaje en proceso generando un "Indicador de error activo" que consiste en una secuencia de 6 bits dominantes sucesivos. Esta secuencia rompe la regla de relleno de bits y provocará la generación de tramas de error en otros nodos. Por tanto el indicador de error puede extenderse entre 6 y 12 bits dominantes sucesivos. Finalmente se recibe el campo de delimitación de error formado por los 8 bits recesivos. Entonces la comunicación se reinicia y el nodo que había sido interrumpido reintenta la transmisión del mensaje.

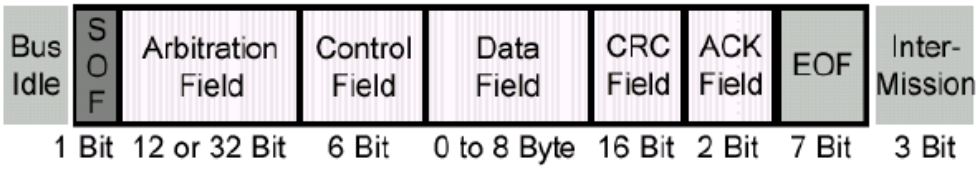
Si un nodo en estado de error "Pasivo" detecta un error, el nodo transmite un "Indicador de error pasivo" seguido, de nuevo, por el campo delimitador de error. El indicador de error de tipo pasivo consiste en 6 bits recesivos seguidos y, por tanto, la trama de error para un nodo pasivo es una secuencia de 14 bits recesivos. De aquí se deduce que la transmisión de una trama de error de tipo pasivo no afectará a ningún nodo en la red, excepto cuando el error es detectado por el propio nodo que está transmitiendo. En ese caso los demás nodos detectarán una violación de las reglas de relleno y transmitirán a su vez tramas de error.

Tras señalar un error por medio de la trama de error apropiada cada nodo transmite bits recesivos hasta que recibe un bit también recesivo, luego transmite 7 bits recesivos consecutivos antes de finalizar el tratamiento de error.

La regla de relleno de bits que aparece líneas superiores, consiste en que cada cinco bits de igual valor se introduce uno de valor inverso tal y como se ve en la figura siguiente:



Otro método para la detección de errores es el chequeo de la trama:



El campo CRC contiene información adicional a la trama, éste se calcula con un polinomio generador de igual manera en el receptor y en el emisor. Esto permite detectar errores aleatorios en hasta 5 bits o una secuencia seguida de 15 bits corruptos.

El campo ACK, en el caso del emisor será recesivo y el receptor deberá sobrescribirlo como dominante y el primero comprobará, mediante la monitorización, que el mensaje ha sido escuchado. Si no sucede así, la trama se considerará corrupta.

3.2.4.5. Trama de sobrecarga

Una trama de sobrecarga tiene el mismo formato que una trama de error activo. Sin embargo, la trama de sobrecarga sólo puede generarse durante el espacio entre tramas. De esta forma se diferencia de una trama de error, que sólo puede ser transmitida durante la transmisión de un mensaje. La trama de sobrecarga consta de dos campos, el Indicador de Sobrecarga, y el delimitador. El indicador de sobrecarga consta de 6 bits dominantes que pueden ser seguidos por los generados por otros nodos, dando lugar a un máximo de 12 bits dominantes. El delimitador es de 8 bits recesivos.

Una trama de sobrecarga puede ser generada por cualquier nodo que debido a sus condiciones internas no está en condiciones de iniciar la recepción de un nuevo mensaje. De esta forma retrasa el inicio de transmisión de un nuevo mensaje. Un nodo puede generar como máximo 2 tramas de sobrecarga consecutivas para retrasar un mensaje. Otra razón para iniciar la transmisión de una trama de sobrecarga es la detección por cualquier nodo de un bit dominante en los 3 bits de "intermission". Por todo ello una trama de sobrecarga de 5 generada por un nodo dará normalmente lugar a la generación de tramas de sobrecarga por los demás nodos dando lugar, como se ha indicado, a un máximo de 12 bits dominantes de indicador de sobrecarga.

3.2.4.6. Espacio entre tramas

El espacio entre tramas separa una trama (de cualquier tipo) de la siguiente trama de datos o interrogación remota. El espacio entre tramas ha de constar de, al menos, 3 bits recesivos. Esta secuencia de bits se denomina "íntermission". Una vez transcurrida esta secuencia un nodo en estado de error activo puede iniciar una nueva transmisión o el bus permanecerá en reposo. Para un nodo en estado error pasivo la situación es diferente, deberá espera una secuencia adicional de 8 bits recesivos antes de poder iniciar una transmisión. De esta forma se asegura una ventaja en inicio de transmisión a los nodos en estado activo frente a los nodos en estado pasivo.

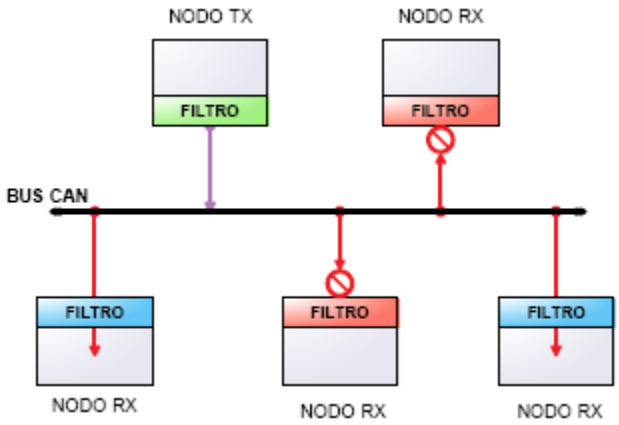
3.2.5. Acceso Múltiple y Arbitraje

Unas de las características que distingue a CAN con respecto a otras normas, es su técnica de acceso al medio denominada como CSMA/CD+CR o "Carrier Sense, Multiple Access/Collision Detection + Collision Resolution" (Acceso Múltiple con detección de portadora, detección de colisión más Resolución de colisión). Cada nodo debe vigilar el bus en un periodo sin actividad antes de enviar un mensaje (Carrier Sense) y además, una vez que ocurre el periodo sin actividad cada nodo tiene la misma oportunidad de enviar un mensaje (Multiple Access). En caso de que dos nodos comiencen a transmitir al unísono se detectará la colisión.

El método de acceso al medio utilizado en bus CAN añade una característica adicional: la resolución de colisión. En la técnica CSMA/CD utilizada en redes Ethernet ante colisión de varias tramas, todas se pierden. CAN resuelve la colisión con la supervivencia de una de las tramas que chocan en el bus. Además la trama superviviente es aquella a la que se ha identificado como de mayor prioridad.

La resolución de colisión se basa en una topología eléctrica que aplica una función lógica determinista a cada bit, que se resuelve con la prioridad del nivel definido como bit de tipo dominante. Definiendo el bit dominante como equivalente al valor lógico '0' y bit recesivo al nivel lógico '1' se trata de una función AND de todos los bits transmitidos simultáneamente. Cada transmisor escucha continuamente el valor presente en el bus, y se retira cuando ese valor no coincide con el que dicho transmisor ha forzado. Mientras hay coincidencia la transmisión continua, finalmente el mensaje con identificador de máxima prioridad sobrevive. Los demás nodos reintentarán la transmisión lo antes posible.

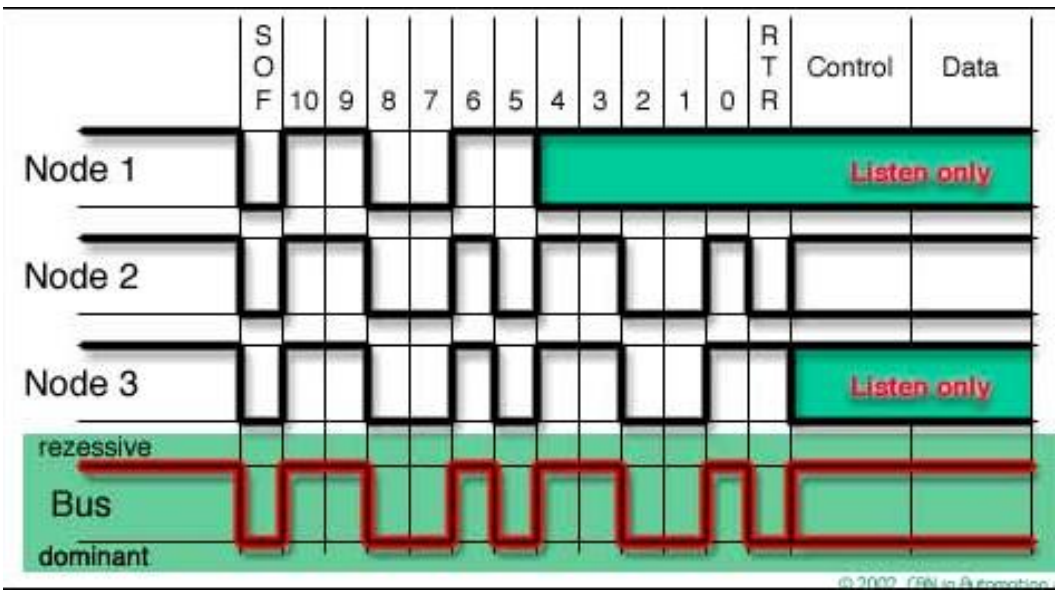
Se ha de tener en cuenta que la especificación CAN de Bosh no establece cómo se ha de traducir cada nivel de bit (dominante o recesivo) a variable física. Cuando se utiliza par trenzado según ISO 11898 el nivel dominante es una tensión diferencial positiva en el bus, el nivel recesivo es ausencia de tensión, o cierto valor negativo, (los transceptores no generan corriente sobre las resistencias de carga del bus). Esta técnica aporta la combinación de dos factores muy deseados en aplicaciones industriales distribuidas: la posibilidad de fijar con determinismo la latencia en la transmisión de mensajes entre nodos y el funcionamiento en modo multimaestro sin necesidad de gestión del arbitraje, es decir control de acceso al medio, desde las capas de software de protocolo. La prioridad queda así determinada por el contenido del mensaje, en CAN es un campo determinado, el identificador de mensaje, el que determina la prioridad.



Sistema multicast

En un bus único, un identificador de mensaje ha de ser asignado a un solo nodo concreto, es decir, se ha de evitar que dos nodos puedan iniciar la transmisión simultánea de mensajes con el mismo identificador y datos diferentes. El protocolo CAN establece que cada mensaje es único en el sistema, de manera que por ejemplo, si en un automóvil existe la variable "presión de aceite", esta variable ha de ser transmitida por un nodo concreto, con un identificador concreto, con una longitud fija concreta y coherente con la codificación de la información en el campo de datos.

En la siguiente figura se ve un ejemplo de arbitraje en un bus CAN.



3.2.6. Detección de errores

Una de las características más importantes y útiles de CAN es su alta fiabilidad, incluso en entornos de ruido extremos, el protocolo CAN proporciona una gran variedad de mecanismos para detectar errores en las tramas. Esta detección de error es utilizada para retransmitir la trama hasta que sea recibida con éxito. Otro tipo de error es el de aislamiento, y se produce cuando un dispositivo no funciona correctamente y un alto por ciento de sus tramas son erróneas. Este error de aislamiento impide que el mal funcionamiento de un dispositivo condicione el funcionamiento del resto de nodos implicados en la red.

3.2.6.1. Tipos de error

En el momento en que un dispositivo detecta un error en una trama, este dispositivo transmite una secuencia especial de bits, "el error flag". Cuando el dispositivo que ha transmitido la trama errónea detecta el error flan, transmite la trama de nuevo. Los dispositivos CAN detectan los errores siguientes:

- **Error de bit:** Durante la transmisión de una trama, el nodo que transmite monitoriza el bus. Cualquier bit que reciba con polaridad inversa a la que ha transmitido se considera un error de bit, excepto cuando se recibe durante el campo de arbitraje o en el bit de reconocimiento. Además, no se considera error de bit la detección de bit dominante por un nodo en estado de error pasivo que retransmite una trama de error pasivo.

- **Error de relleno (Stuff Error):** Se considera error de relleno la detección de 6 bits consecutivos del mismo signo, en cualquier campo que siga la técnica de relleno de bits, donde por cada 5 bits iguales se añade uno diferente.

- **Error de CRC:** Se produce cuando el cálculo de CRC realizado por un receptor no coincide con el recibido en la trama. El campo CRC (Cyclic Redundant Code) contiene 15 bits y una distancia de Hamming de 6, lo que asegura la detección de 5 bits erróneos por mensaje. Estas medidas sirven para detectar errores de transmisión debido a posibles incidencias en el medio físico como por ejemplo el ruido.

- **Error de forma (Form Error):** Se produce cuando un campo de formato fijo se recibe alterado como bit.

- **Error de reconocimiento (Acknowledgement Error):** Se produce cuando ningún nodo cambia a dominante el bit de reconocimiento.

Si un nodo advierte alguno de los fallos mencionados, iniciará la transmisión de una trama de error. El protocolo CAN especifica diversos fallos en la línea física de comunicación, como por ejemplo línea desconectada, problemas con la terminación de los cables o líneas cortocircuitadas. Sin embargo, no especificará como reaccionar en caso de que se produzca alguno de estos errores.

3.2.6.2. Aislamiento de Nodos defectuosos

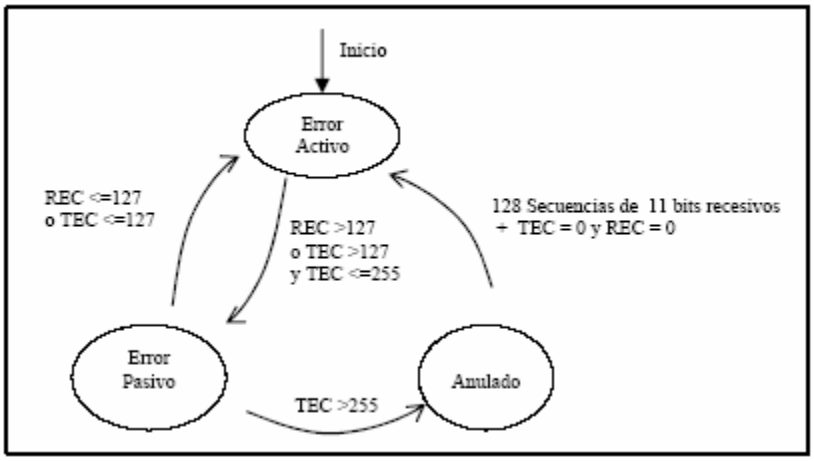
Para evitar que un nodo en problemas condicione el funcionamiento del resto de la red, se han incorporado a la especificación CAN medidas de aislamiento de nodos defectuosos que son gestionadas por los controladores. Un nodo pude encontrarse en uno de los tres estados siguientes en relación a la gestión de errores:

- **Error Activo (Error Active):** Es el estado normal de un nodo. Participa en la comunicación y en caso de detección de error envía una trama de error activa.

- **Error pasivo (Error Passive):** Un nodo en estado de error pasivo participa en la comunicación, sin embargo ha de esperar una secuencia adicional de bits recesivos antes de transmitir y sólo puede señalar errores con una trama de error pasiva.

- **Anulado (Bus Off):** En este estado, deshabilitará su transceptor y no participará en la comunicación.

La evolución entre estos estados se basa en dos contadores incluidos en el controlador de comunicaciones. Contador de errores de transmisión (TEC) y Contador de errores de recepción (REC).



Evolución entre estados de error

3.2.7. Especificaciones

CAN tiene dos formatos diferentes para transmitir datos: el formato de trama estándar (Standard Frame), según la especificación CAN 2.0A y, el formato de trama extendida (Extended Frame) según la especificación CAN 2.0B. La principal diferencia entre ambos es la longitud del identificador del mensaje (ID), que en el caso de la trama estándar es de 11 bits (2032 identificadores, ya que los 16 bits identificadores de menor prioridad están reservados) y en el caso de la extendida es de 29 bits (más de 536 millones de identificadores):

- **Controladores 2.0A:** únicamente transmiten y reciben mensajes en formato estándar. Si el formato es extendido se producirá un error.

- **Controladores 2.0B pasivos:** únicamente transmiten y reciben mensajes en formato estándar, pero admiten la recepción de mensajes en formato extendido, aunque los ignora posteriormente.

- **Controladores 2.0B activos:** transmiten y reciben mensajes en ambos formatos.

3.2.8. Implementaciones

Existen tres tipos de arquitecturas en microcontroladores: Stand-Alone CAN controller, Integrated CAN Controller y Single-Chip CAN Node.

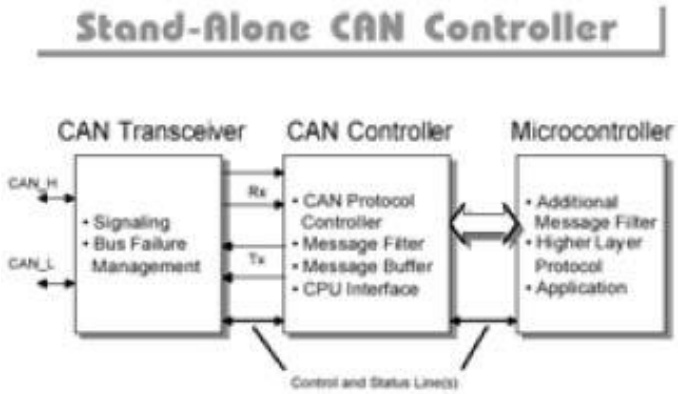
Stand-Alone CAN Controller

Es la arquitectura más simple, para llevar a cabo una comunicación en una red CAN será necesario:

- Un microcontrolador como puede ser el 16F877, con el que se ha venido trabajando a lo largo de todo este proyecto.

- Un controlador CAN que introduzca el protocolo CAN, filtro de mensajes,… y todo el interface necesario para las comunicaciones. Un ejemplo de controlador CAN es el MCP2510 cuya DATA SHEET se puede encontrar en la página web de microchip.

- Un transceiver CAN, esto es, un transmisor/receptor que puede ser por ejemplo el MCP2551 desarrollado por microchip.

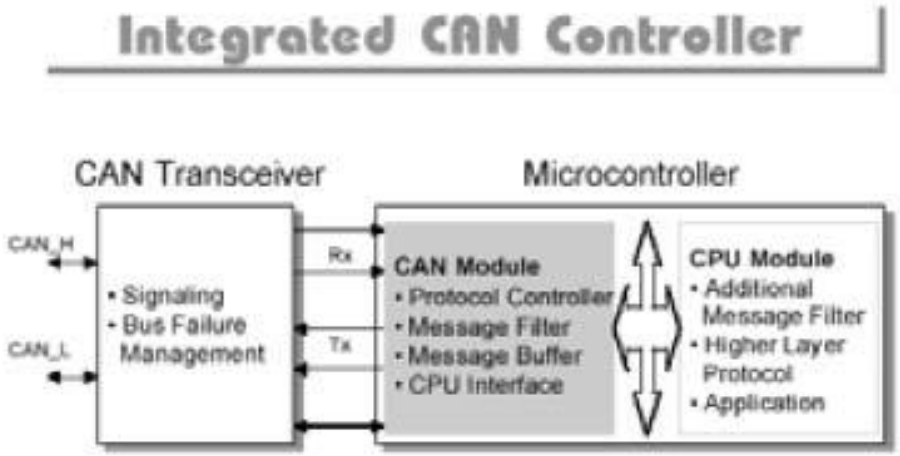


Así pues, si de alguna manera se pretende trabajar en una red CAN con una placa de pruebas, en un principio no sería factible a no ser que de alguna manera se le acoplasen el controlador y el transceptor CAN correspondiente.

Pero Microchip ha creado una placa de pruebas específica para este tipo de comunicaciones, que desarrolla además este tipo de arquitectura y que se puede conseguir también a través de la página web de microchip.

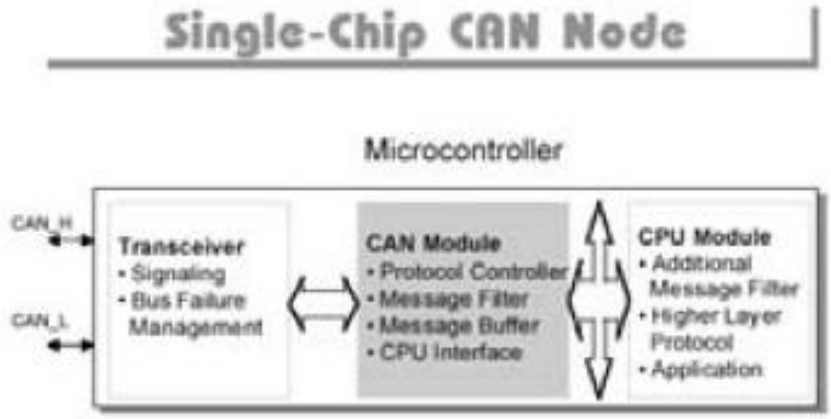
Integrated CAN Controller

Este tipo de arquitectura consiste en un microcontrolador que incluya, no sólo sus características propias sino además un módulo CAN con las características de un microcontrolador CAN. El transceiver se sitúa de manera separada. Este es el caso de nuestro microcontrolador dSPIC30F4013. El módulo del transceptor actúa como un controlador de línea, balanceando la señal, esto es, adecuando los niveles de tensión de esta al exterior.



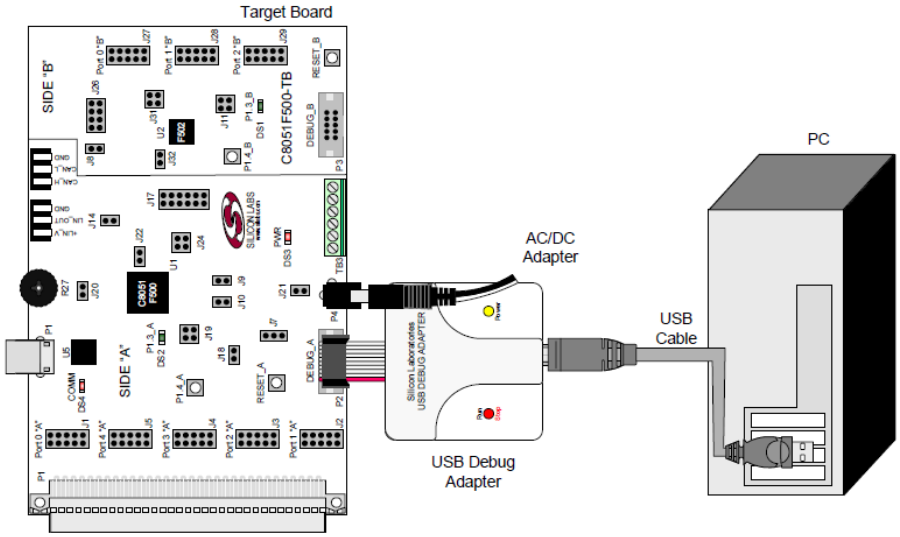
Single-chip CAN Node

Se trata de un chip que incluye en su interior los tres elementos necesarios para llevar a cabo las comunicaciones en un entorno CAN.



4. Microcontrolador

En este apartado realizaremos una introducción y un breve resumen de la parte del microcontrolador en el proyecto. Esta es la parte que se entrego en el primer proyecto.



En el primer apartado se muestran las especificaciones técnicas del microcontrolador y en el segundo apartado las especificaciones técnicas de la placa en la que está el microcontrolador. Continuaremos con una introducción a el lenguaje de programación C y como se a programado el microcontrolador pero sin llegar a introducirse en el código.

4.1. Especificaciones Microcontrolador

Resumen de las principales características del Microcontrolador C8051F500:

Analog Peripherals

- 12-Bit ADC
  - Up to 200 ksps
  - Up to 32 external single-ended inputs
  - VREF from on-chip VREF, external pin or VDD
  - Internal or external start of

Memory

- 4352 bytes internal data RAM (256 + 4096 XRAM)
- 64 or 32 kB Flash; In-system programmable in 512-byte Sectors

Digital Peripherals

conversion source
    - Built-in temperature sensor
- Two Comparators
    - Programmable hysteresis and response time
    - Configurable as interrupt or reset source
    - Low current

On-Chip Debug

- On-chip debug circuitry facilitates full speed, non-intrusive in-system debug (no emulator required)
- Provides breakpoints, single stepping, inspect/modify memory and registers
- Superior performance to emulation systems using ICE-chips, target pods, and sockets
- Low cost, complete development kit

Supply Voltage 1.8 to 5.25 V

- Typical operating current: 18 mA at 50 MHz or 20 µA at 32 kHz
- Typical stop mode current:3 µA

High-Speed 8051 µC Core

- Pipelined instruction architecture; executes 70% of instructions in 1 or 2 system clocks
- Up to 50 MIPS throughput with 50 MHz clock
- Expanded interrupt handler
- 40 or 25 Port I/O; All 5 V tolerant with high sink current
- CAN 2.0 Controller—no crystal required
- LIN 2.0 Controller (Master and Slave capable); no crystal required
- Hardware enhanced UART, SMBus™, and enhanced SPI™ serial ports
- Four general purpose 16-bit counter/timers
- 16-Bit programmable counter array (PCA) with six capture/compare modules and enhanced PWM functionality

Clock Sources

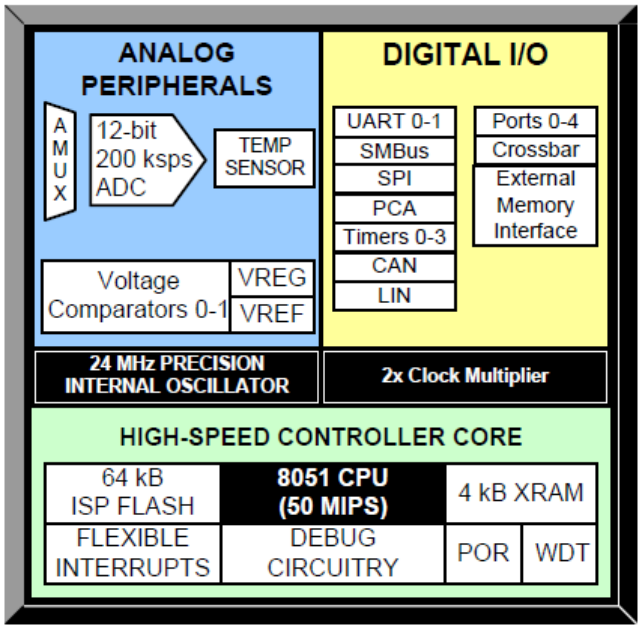
- Internal 24 MHz with ±0.5% accuracy for CAN and master LIN operation (C8051F500/2/4/6)
- External oscillator: Crystal, RC, C, or clock (1 or 2 pin modes)
- Can switch between clock sources on-the-fly; useful in power saving modes

Packages

- 48-Pin QFP/QFN (C8051F500/1/4/5)
- 32-Pin QFP/QFN (C8051F502/3/6/7)

Automotive Qualified

- Temperature Range: –40 to +125 °C
- Compliant to AEC-Q100



4.2. Especificaciones Placa

En este apartado se muestra las especificaciones técnicas de la placa C8051F500.

50 MIPS, 64 kB Flash, 12-Bit ADC, 48-Pin Automotive MCU

Analog Peripherals

- 12-Bit ADC, 5 V input signal; up to 32 external inputs
  - ±1 LSB INL; guaranteed monotonic
  - Programmable throughput up to 200 ksps
  - Data-dependent windowed interrupt generator
  - Programmable gain maximizes input signal span
- Built-in Temperature Sensor (±3 °C)
- Two Comparators
- Precision Internal Voltage Reference
- V_{DD} Monitor/Brown-out Detector
- On-Chip Debug
  - On-chip debug circuitry facilitates full speed, non-intrusive in system debug (no emulator required)
  - Provides breakpoints, single stepping, watch-points
  - Inspect/modify memory, registers, and stack
  - Superior performance to emulation systems using ICE-chips, target pods, and sockets
  - Multiple power saving sleep and shutdown modes

Temperature Range: –40 to +125 °C
Operating Voltage: 1.8 to 5.25 V

High-Speed 8051 µC Core

- Pipelined instruction architecture; executes 70% of instructions in 1 or 2 system clocks
- Up to 50 MIPS throughput

Memory

- 64 kB Flash; in-system programmable; flexible security features
- 4352 bytes data RAM (256 + 4 kB)

CAN 2.0B

- 32 message objects

LIN 2.1

- Master or slave operation using dedicated hardware

Digital Peripherals

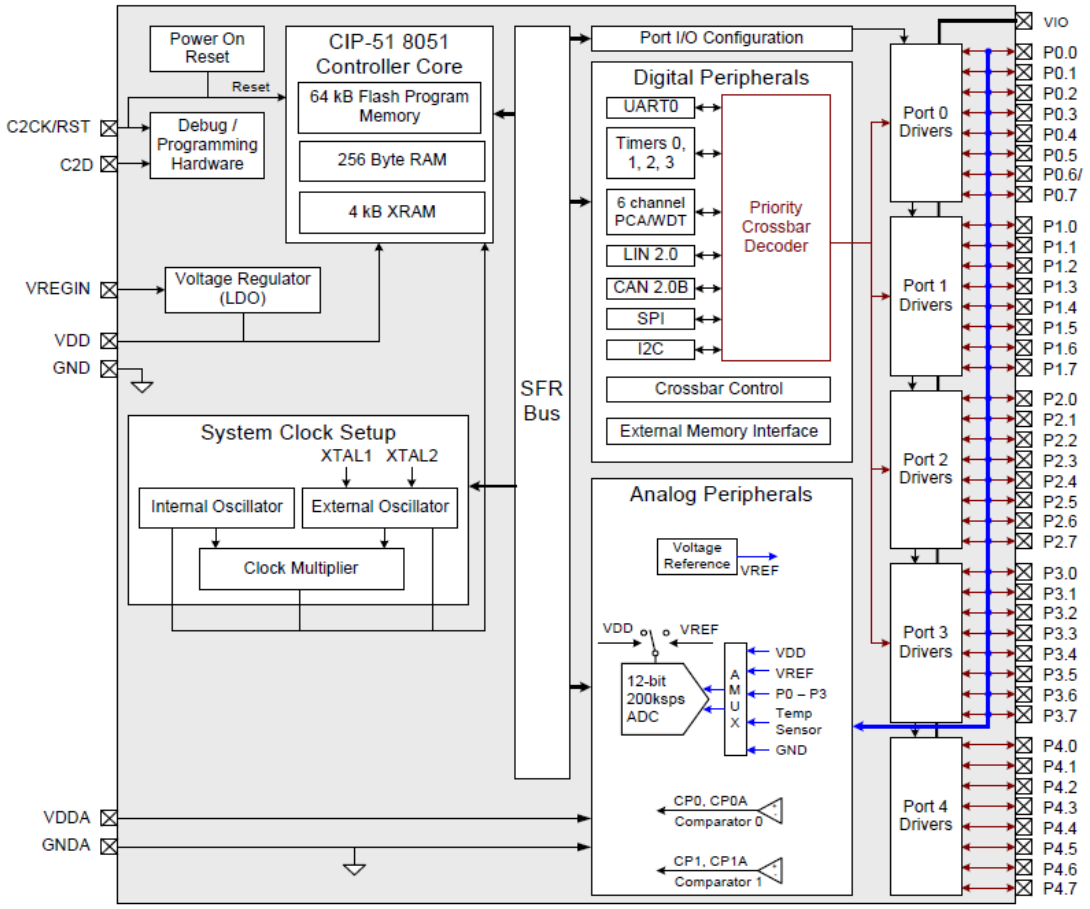
- Up to 40 digital I/O; all are 5 V push-pull
- Hardware I2C, SPI™, and UART serial ports available concurrently
- Programmable 16-bit counter array with 6 capture/compare modules
- 4 general-purpose 16-bit counter/timers
- External Memory Interface (EMIF)

Clock Sources

- Internal programmable 0.5% oscillator: up to 50 MHz
- External oscillator: Crystal, RC, C, or CMOS Clock

Ordering Part Numbers

- C8051F500-IM, 48-Pin QFN (RoHS-compliant), 7x7 mm^2
- C8051F500-IQ, 48-Pin QFP (RoHS-compliant), 9x9 mm^2



4.3. Programación en C

C es un lenguaje bastante conciso y en ocasiones desconcertante. Considerado ampliamente como un lenguaje de alto nivel, posee muchas características importantes, tales como: programación estructurada, un método definido para llamada a funciones y para paso de parámetros, potentes estructuras de control, etc.

Sin embargo gran parte de la potencia de C reside en su habilidad para combinar comandos simples de bajo nivel, en complicadas funciones de alto nivel, y en permitir el acceso a los bytes y words del procesador. En cierto modo, C puede considerarse como una clase de lenguaje ensamblador universal. La mayor parte de los programadores familiarizados con C, lo han utilizado para programar grandes máquinas que corren Unix, MS-DOS, e incluso Windows (programación de drivers). En estas máquinas el tamaño del programa no es importante, y el interface con el mundo real se realiza a través de llamadas a funciones o mediante interrupciones DOS. Así el programador en C sólo debe preocuparse en la manipulación de variables, cadenas, matrices, etc.

Con los modernos microcontroladores de 8 bits, la situación es algo distinta. Tomando como ejemplo el 8051, el tamaño total del programa debe ser inferior a los 4 u 8K (dependiendo del

tamaño de la EEPROM), y debe usarse menos de 128 o 256 bytes de RAM. Idealmente, los dispositivos reales y los registros de funciones especiales deben ser direccionados desde C. Las interrupciones, que requieren vectores en direcciones absolutas también deben ser atendidas desde C. Además, se debe tener un cuidado especial con las rutinas de ubicación de datos para evitar la sobre escritura de datos existentes.

Uno de los fundamentos de C es que los parámetros (variables de entrada) se pasan a las funciones (subrutinas) en la pila, y los resultados se devuelven también en la pila. Así las funciones pueden ser llamadas desde las interrupciones y desde el programa principal sin temor a que las variables locales sean sobre escritas.

Una seria restricción de la familia 8051 es la carencia de una verdadera pila. En un procesador como el 8086, el apuntador de la pila tiene al menos 16 bits. Además del apuntador de pila, hay otros registros que pueden actuar como apuntadores a datos en la pila, tal como el BP (Base Pointer). En C, la habilidad para acceder a los datos en la pila es crucial. Como ya ha sido indicado, la familia 8051 está dotada de una pila que realmente sólo es capaz de manejar direcciones de retorno. Con 256 bytes disponibles, como máximo, para la pila no se pueden pasar muchos parámetros y realizar llamadas a muchas funciones.

De todo ello, puede pensarse que la implementación de un lenguaje que como C haga un uso intensivo de la pila, es imposible en un 8051. Hasta hace poco así ha sido. El 8051, hace tiempo que dispone de compiladores C, que en su mayor parte han sido adaptados de micros más potentes, tal como el 68000. Por ello la aproximación al problema de la pila se ha realizado creando pilas artificiales por software. Típicamente se ha apartado un área de RAM externa para que funcione como una pila, con la ayuda de rutinas que manejan la pila cada vez que se realizan llamadas a funciones. Este método funciona y proporciona capacidad de repetir variables locales a distintos niveles sin sobre escritura, pero a costa de hacer los programas muy lentos. Por lo tanto, con la familia 8051, la programación en lenguaje ensamblador ha sido la única alternativa real para el desarrollo de pequeños sistemas en los que el tiempo es un factor crítico.

Sin embargo, en 1980, Intel proporcionó una solución parcial al problema al permitir la programación del 8051 en un lenguaje de alto nivel llamado PLM51. Este compilador no era perfecto, había sido adaptado del PLM85 (8085), pero Intel fue lo suficientemente realista para evitar el uso de un lenguaje totalmente dependiente del uso de la pila.

La solución adoptada fue sencillamente pasar los parámetros en áreas definidas de memoria. Así cada función o procedure tenía su propia área de memoria en la que recibía los parámetros, y devolvía los resultados. Si se utilizaba la RAM interna para el paso de parámetros, la sobrecarga de las llamadas a funciones era muy pequeña. Incluso utilizando RAM externa, siempre más lenta que la RAM interna, se consigue mayor velocidad que con una pila artificial.

El problema que tiene esta especie de "pila compilada" es que la sobrecarga de variables no es posible. Esta aparentemente seria omisión, en la práctica no tiende a causar problemas con los típicos programas del 8051. Sin embargo las últimas versiones de C51 permiten la sobrecarga de variables selectiva, es decir permiten que unas pocas funciones críticas tengan sobrecarga, sin comprometer la eficiencia de todo el programa.

Otras consideraciones dignas de destacar para el C en un microcontrolador son:

- Control de los periféricos internos y externos del chip.
- Servicio de las interrupciones.
- Hacer el mejor uso de los limitados conjuntos de instrucciones.
- Soportar diferentes configuraciones de ROM/RAM.
- Un alto nivel de optimización para conservar el espacio de código.
- Control de la conmutación de registros.
- Soporte para los derivados de la familia (87C751, 80C517 etc.).

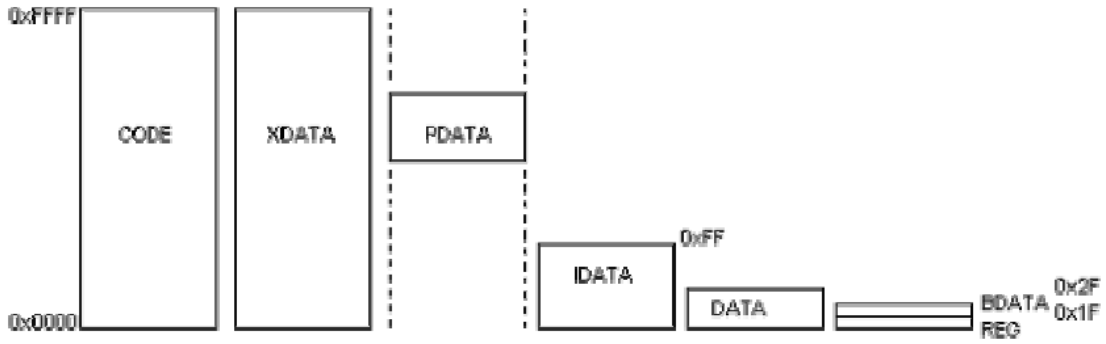
El compilador Keil C51 contiene todas las extensiones para el uso del lenguaje C con microcontroladores. Este compilador C utiliza todas las técnicas apuntadas por Intel con su PLM51, pero añade características propias tales como la aritmética en coma flotante, la entrada/salida (I/O) con formato, etc. Se trata de la implementación del estándar ANSI C específico para los procesadores 8051.

4.3.1. Configuración de memoria

Inicialmente la cosa más confusa sobre el 8051 es quizás la existencia de varios espacios de memoria, que comienzan en la misma dirección.

Otros μC, tales como el 68HC11, tienen una configuración de memoria mucho más sencilla, en la que solo existe un área de memoria de tipo Von Neuman, residente en uno o en varios chips.

Dentro del 8051 hay un espacio de RAM llamado DATA. Este espacio comienza en la dirección D:00 (el prefijo 'D:' indica segmento DATA) y termina en la dirección 0x7F (127 en decimal). Este área RAM puede utilizarse para almacenar las variables del programa. Se trata de una región direccionable directamente en la que pueden utilizarse instrucciones como 'MOV A,direcc'. Por encima de la dirección 0x7F se encuentran los registros de funciones especiales (SFR), que también son accesibles mediante direccionamiento directo. Sin embargo, en algunos derivados del 8051 existe otro espacio de memoria entre las direcciones 0x80 y 0xFF, llamado IDATA (Indirect DATA), que sólo resulta accesible por direccionamiento indirecto (MOV A,@Ri). Para esta región que se solapa con los SFR se utiliza el prefijo 'I:'. El 8051 carece de estos 128 bytes del espacio IDATA, que se añadieron cuando apareció el 8052. Esta región resulta adecuada para la pila a la que siempre se accede indirectamente a través del apuntador de pila SP (Stack Pointer). Y para hacer las cosas más confusas, resulta que los 128 bytes de RAM comprendidos en las direcciones 0..0x7F también pueden ser accedidos indirectamente con la instrucción MOV A,@Ri.



Los espacios de memoria del 8051

Un tercer espacio de memoria, el segmento CODE, también comienza en la dirección cero, pero está reservado para el programa o para almacenar valores constantes, ya que el µC no puede escribir en esta región. Se extiende desde C:0000 hasta C:0xFFFF (65536 bytes); parte de este área puede residir dentro del µC y la otra parte, si es necesaria, debe residir en chips externos de memoria EPROM o FLASH. El acceso al contenido del segmento CODE se realiza mediante el contador de programa PC (Program Counter) en los ciclos de búsqueda de instrucciones y con el DPTR (Data Pointer) para la lectura de los datos constantes.

La cuarta región de memoria, llamada XDATA reside en una RAM externa al µC. Comienza en X:0000 y se extiende hasta X:0xFFFF (65536 bytes). El acceso a esta región se realiza mediante el registro de 16 bits DPTR (Data Pointer). En esta región se puede seleccionar un pequeño espacio de 256 bytes, llamado PDATA (acceso paginado), al que se accede con un apuntador de 8 bits (registros R0 o R1). La dirección inicial de esta región PDATA viene dada por 16 bits, siendo los 8 bits de mayor peso los contenidos en el SFR P2, estando a 0 los 8 bits restantes.

Una pregunta inmediata es: "¿Cómo evita el 8051 que el acceso a un dato en D:00, realice también un acceso al dato X:0000?

La respuesta reside en el hardware del 8051: Cuando la CPU intenta acceder a D:00, habilita la RAM interna mediante una señal READ interna, que no provoca cambios en la patilla /RD, utilizada para leer la RAM externa.

- MOV A,40H ; Lleva el valor de la dirección D:0x40 al acumulador.

Este modo de direccionamiento (directo) se utiliza en el modelo de memoria SMALL.

- MOV R0,#0A0H
- MOV A,@R0 ; Lleva el valor de la dirección I:0xA0 al acumulador

Este modo de direccionamiento se utiliza para acceder a la región de RAM interna de acceso indirecto situada por encima de la dirección 0x7F, aunque también es una forma alternativa de acceso a los 128 bytes situados debajo de la dirección 0x80.

Una variación de DATA es BDATA (bit data). Esta es un área de 16 bytes (128 bits) que se extiende desde D:0x20 hasta D:0x2F. Se trata de una región muy útil que permite el acceso normal a los bytes con las instrucciones MOV, y también permite el acceso a los bits mediante instrucciones especialmente orientadas al bit, como las siguientes:

- SETB 20H.0 ; Pone a uno el bit 0 de la dirección D:0x20
- CLR 20H.2 ; Pone a cero el bit 2 de la dirección D:0x20

El dispositivo externo EPROM o FLASH de memoria CODE no se habilita durante los accesos a la RAM. De hecho, la memoria externa de tipo CODE sólo se habilita cuando una patilla del 8051 llamada PSEN (Program Store Enable) se pone a nivel bajo. El nombre CODE indica que la principal función de la EPROM o FLASH es almacenar el programa.

No existe colisión entre los accesos a la RAM XDATA y la EPROM-FLASH CODE, ya que el dispositivo externo XDATA sólo se activa a petición de dos patillas del 8051 llamadas /RD y /WR (Read & Write), mientras que el dispositivo externo CODE se activa exclusivamente cuando la patilla PSEN se pone a nivel bajo. Lógicamente el 8051 nunca activa a la vez las patillas PSEN y /RD o /WR.

Para acceder a la RAM XDATA existen instrucciones especiales (MOVX)

- MOV DPTR,#08000H
- MOVX A,@DPTR ; "Lleva al acumulador el contenido de la posición de RAM externa cuya dirección está en DPTR (8000H)".

Este modo de direccionamiento se utiliza en el modelo de memoria LARGE.

- MOV R0,#080H ;
- MOVX A,@R0 ;

Este es un modo de acceso alternativo a la RAM externa que se utiliza en el modelo de memoria COMPACT. La dirección realmente accedida es 0xYY80 siendo YY el contenido del SFR P2.

Un punto importante a recordar es que la patilla PSEN se activa durante la fase de búsqueda de instrucción, y con las instrucciones MOVC... (Move Code) usadas para la lectura de los datos constantes residentes en memoria de código. Por otro lado, las instrucciones MOVX... (Move eXternal) activan las patillas /RD o /WR, según que el movimiento sea hacia la CPU (lectura), o desde la CPU (escritura). La 'X' indica que la dirección no está dentro del 8051, sino que está contenida en un dispositivo eXterno que se habilita con las patillas /RD y /WR.

4.3.1.1. Modelos de memoria

Así como el programador de PC tiene que elegir entre los modelos de memoria tiny, small, medium, compact, large y huge para definir la segmentación de la RAM, el programador de 8051 tiene que decidir el modelo de memoria a utilizar, el cual determina el lugar de residencia del programa y de los datos.

C51 soporta actualmente las siguientes configuraciones de memoria:

- **ROM:** el tamaño máximo del programa que permite generar C51 es de 64K, sin embargo es posible aumentarlo hasta 1MB mediante el modelo BANKED descrito más abajo. Todos los elementos que se deseen almacenar en la EPROM/ROM, tales como constantes, tablas etc., deben declararse como code.

- **RAM:** Se admiten tres modelos de memoria, SMALL, COMPACT y LARGE

    - **SMALL:** todas las variables y segmentos de paso de parámetros se guardan en RAM interna.

    - **COMPACT:** las variables se almacenan en una región de RAM externa de acceso paginado y de tamaño máximo 256 bytes. El acceso a las mismas se hace indirectamente por medio de instrucciones "MOVX A,@R0". Los registros internos del 8051 se usan para variables locales y parámetros de función.

    - **LARGE:** las variables se almacenan en memoria externa, y se accede a ellas con instrucciones "MOVX A,@DPTR", lo que permite disponer de un espacio máximo de 64Kbytes para las variables. Los registros internos del 8051 se usan para variables locales y parámetros de función.

Modelo BANKED: Con este modelo el código puede ocupar hasta 1MB utilizando algunas patillas de un puerto del 8051, o un latch mapeado en memoria. Estos hilos se utilizan para paginar la memoria por encima de la dirección 0xFFFF. Dentro de cada bloque de 64KB debe existir un bloque común (COMMON) para hacer posibles las llamadas a funciones que se encuentren en bancos distintos.

Además de la elección del modelo de RAM, es posible utilizar un modelo globalmente y forzar que ciertas variables y objetos residan en otros espacios de memoria.

Cada tipo de memoria tiene sus pros y sus contras. Aquí se ofrecen algunas recomendaciones para hacer el mejor uso de los mismos.

- **DATA:** 128 bytes; área utilizada por el modelo SMALL

    - El mejor para: Datos a los que se accede con frecuencia, variables usadas por rutinas de interrupción, variables de rutinas re-entrantes.

    - El peor para: Arrays y estructuras de tamaño de medio a grande.

- **IDATA:** No dependiente del modelo utilizado

    - El mejor para: Acceso rápido a arrays y estructuras de tamaño medio (unos 32 bytes cada uno, sin sobrepasar los 64 bytes). Ya que el acceso a este tipo de datos suele realizarse indirectamente, mediante punteros, éste tipo de memoria es el mejor para los mismos. También es un buen lugar para la pila, a la que siempre se accede indirectamente.

    - El peor para: Grandes arrays de datos.

- **CODE:** 64K bytes

    - El mejor para: Constantes y grandes tablas, además de para el código del programa, ¡No faltaba más!

    - El peor para: ¡Variables!

- **PDATA:** 256 bytes; área utilizada por el modelo COMPACT

    - El mejor para: Accesos a datos que requieran una velocidad intermedia, así como a matrices y estructuras de tamaño moderado.

    - El peor para: Arrays y estructuras cuyo tamaño supere los 256 bytes. Datos a los que se accede con mucha frecuencia, etc..

- **XDATA:** Área utilizada por el modelo LARGE

    - El mejor para: Arrays y estructuras cuyo tamaño supere los 256 bytes. Variables a las que se accede con poca frecuencia.

    - El peor para: Datos a los que se accede con mucha frecuencia, etc..

4.3.1.2. Selección de Modelos

La selección del tipo de memoria global se realiza incluyendo la línea #pragma SMALL (o COMPACT o LARGE) como primera línea de un fichero C. El compilador C51 utiliza por omisión de la directiva #pragma, el modelo SMALL. Este modelo puede utilizarse en casi el 100% de las aplicaciones, si se tiene la precaución de forzar las variables de tamaño grande, y las variables a las que se accede rara vez, en las áreas PDATA y XDATA.

El modelo COMPACT hace ciertas suposiciones sobre el estado del puerto P2. El espacio XDATA se direcciona mediante instrucciones MOVX que ponen los 16 bits del registro DPTR en los puertos P2 y P0. El modelo COMPACT utiliza el registro R0 como un apuntador de 8 bits, cuyo contenido se pone en el puerto P0 cuando se ejecuta la instrucción MOVX A,@R0. El puerto P2 queda bajo control del usuario para el acceso paginado a la RAM externa. El compilador no tiene información sobre P2, y a menos que se le asigne explícitamente un valor, su contenido será indefinido, aunque generalmente será 0xFF. El linker tiene la tarea de combinar las variables XDATA y PDATA, y si no se le informa adecuadamente, coloca el área PDATA en dirección 0. Por lo tanto el programa COMPACT no funcionará.

Es por tanto esencial asignar a PPAGE en el fichero "startup.a51" el valor adecuado para P2, y poner PPAGEENABLE a 1 para habilitar el modo paginado. La asignación del valor de PPAGE también puede hacerse por medio del control PDATA(ADDR) al realizar la llamada al linker como en:

L51 module1.obj, module2.obj to exec.abs PDATA(0)XDATA(100H)

Notar que el área normal XDATA comienza ahora en 0x100, por encima de la página cero usada para PDATA.

Especificación local del modelo de memoria.

C51 permite asignar modelos de memoria a funciones individuales. Dentro de un mismo módulo, las funciones pueden declararse como SMALL, COMPACT o LARGE así:

```
#pragma COMPACT

/* Una función con modelo SMALL */
void fsmall(void) small
{
   printf("HELLO") ;
}

/* Una función con modelo LARGE */
void flarge(void) large
{
printf("HELLO") ;
}

main()
{
fsmall() ; // Llamada a función small.
flarge() ; // Llamada a función large.
}
```

Un aspecto a vigilar en proyectos multi-modelo

Supongamos que un programa C51 utiliza el modelo COMPACT para todas sus funciones excepto para las funciones de interrupción, para las que se desea utilizar el modelo SMALL que resulta más rápido. En estos casos, de no actuar correctamente, el linker puede emitir mensajes del tipo MULTIPLE PUBLIC DEFINITION refiriéndose por ejemplo a la función putchar().

Ello se debe a que en los módulos compilados como COMPACT, C51 crea referencias a funciones de la librería COMPACT, mientras que los módulos SMALL acceden a las funciones de la librería SMALL. De esta forma el linker L51 puede encontrarse con dos putchars() de distintas librerías.

La solución en este caso consiste en utilizar globalmente el modelo de memoria COMPACT y seleccionar localmente un modelo diferente para ciertas funciones.

```
#pragma COMPACT

void fast_func(void) small
{
/* código */
}
```

4.3.2. Tareas

Aplicaciones con el 8051

Muchas personas han empezado a programar utilizando el lenguaje BASIC en un PC o máquina similar. Los programas que realizan inicialmente no suelen ser muy complicados. Empiezan a correr cuando se teclea "RUN" y terminan en un END o STOP. Mientras tanto, el PC se dedica totalmente a la ejecución del típico programa "HELLO WORLD". Cuando el programa termina, se retorna al editor BASIC, o al entorno de trabajo utilizado.

La experiencia es muy buena y el nuevo programador cree que ya sabe programar. Sin embargo, cuando se escribe un programa para un microcontrolador como el 8051, el problema del comienzo y final del programa se presenta rápidamente. Típicamente, el software de un sistema basado en el 8051 está formado por múltiples programas individuales, que ejecutados conjuntamente, contribuyen a la consecución del objetivo final. Entonces, un problema fundamental es asegurarse de que todas las partes del programa se ejecutan.

Sistemas sencillos con el 8051

El enfoque más sencillo es llamar a cada una de las sub-funciones del programa de una forma secuencial, de tal forma que después de un cierto tiempo, cada parte se habrá ejecutado el mismo número de veces. Esto constituye el bucle de fondo (background loop), o programa principal. En primer plano (foreground) aparecen las funciones de interrupción, iniciadas por sucesos producidos en tiempo real, tales como señales de entrada o desbordamiento de temporizadores.

El intercambio de datos entre las interrupciones y el programa principal se realiza usualmente a través de variables globales y flags. Este tipo de programas puede funcionar correctamente si se tiene cuidado en el orden y frecuencia de ejecución de cada sección.

Las funciones llamadas por el programa principal deben escribirse de tal forma que en cada llamada se ejecute una sección particular de su código. Así al entrar en esta función, se toma la decisión de la tarea que le toca ejecutar, se ejecuta esa parte y se sale de la función, cambiando posiblemente algunos flags que indicarán la tarea a realizar en la siguiente llamada. De esta forma, cada bloque funcional debe mantener su propio sistema de control que asegure la ejecución del código adecuado en cada entrada al mismo.

En un sistema de este tipo, todos los bloques funcionales tienen la misma importancia, y no se entra en un nuevo bloque hasta que no le toque su turno dentro del programa principal. Sólo las rutinas de interrupción pueden romper este orden, aunque también están sujetas a un sistema de prioridades. Si un bloque necesita una cierta señal de entrada, o se queda a la espera de la misma impidiendo que otros bloques puedan ejecutarse, o cede el control hasta que le vuelva a tocar su turno dentro del bucle principal. En este último caso se corre el riesgo de que el suceso esperado se produzca y no tenga una respuesta, o que la respuesta llegue demasiado tarde. No obstante, los sistemas de este tipo funcionan bien en programas en las que no hay secciones que tengan exigencias críticas en los tiempos de respuesta.

Los llamados sistemas en tiempo real no son de este tipo. Típicamente contienen código cuya ejecución origina o es originada por sucesos del mundo real, que trabajan con datos procedentes de otras partes del sistema, cuyas entradas pueden cambiar lenta o rápidamente.

El código que contribuye en mayor medida a la funcionalidad del sistema, debe tener precedencia sobre las secciones cuyos objetivos no son críticos respecto al objetivo final. Sin embargo, muchas de las aplicaciones con el 8051 tienen partes con grandes exigencias de tiempo de respuesta, que suelen estar asociadas a interrupciones. La necesidad de atender a las interrupciones tan rápido como sea posible, requiere que el tiempo de ejecución de las mismas sea muy corto, ya que el sistema dejará de funcionar si el tiempo de respuesta a cada interrupción supera al intervalo de tiempo en la que las interrupciones se producen.

Por regla general se consigue un nivel aceptable de prestaciones si las funciones complejas se llevan al programa principal, y se reservan las interrupciones para las secciones con exigencias de tiempo críticas. En cualquier caso, aparece el problema de la comunicación entre el programa principal y las rutinas de interrupción.

Este sencillo sistema de reparto de tareas trata a todas ellas de igual forma. Cuando la CPU se encuentra muy cargada de trabajo por tener que atender a entradas que cambian con rapidez, puede ocurrir que el programa principal no corra con la suficiente frecuencia y la respuesta transitoria del mismo se degrade.

Reparto de tareas simple

Los problemas de los sistemas de bucle simple pueden solucionarse parcialmente, controlando el orden y la frecuencia de las llamadas a funciones. Una posible solución pasa por asignar una prioridad a cada función y establecer un mecanismo que permita a cada función especificar la siguiente función a ejecutar. Las funciones de interrupción no deben seguir este orden y deben ser atendidas con rapidez. Los sistemas de este tipo pueden resultar útiles, si el tiempo de ejecución de las funciones no es excesivo.

Una solución alternativa consiste en utilizar un temporizador, que asigne un tiempo de ejecución a cada trabajo a realizar. Cada vez que el tiempo asignado se supere, la tarea en curso se suspende y comienza otra tarea.

Desafortunadamente todas estas posibilidades suelen intentarse tarde, cuando los tiempos de respuesta se alargan en exceso. En estos casos, lo que había sido un programa bien estructurado termina degenerando en código spaghetti, plagado de ajustes y modos especiales, tendentes a superar las diferencias entre las demandas de los sucesos del mundo real, y la respuesta del sistema. En muchos casos, los mecanismos de control de las funciones llamadas generan una sobrecarga que acentúa aún más el desfase entre las exigencias y la respuesta.

La realidad es que los sucesos del mundo real no siguen ningún orden ni pueden predecirse. Algunos trabajos son más importantes que otros, pero el mundo real produce sucesos a los que hay que responder inmediatamente.

Un enfoque práctico

Si no se recurre a un sistema operativo en tiempo real como RTX51, ¿Qué se puede hacer?

Un mecanismo sencillo para controlar el programa principal, puede ser una sentencia switch, con la variable switch controlada por algún suceso externo en tiempo real. Idealmente, éste debe ser la rutina de interrupción de mayor prioridad. Las tareas del programa principal con mayor prioridad se colocan en los case de menor valor numérico, y las de menor prioridad en los case con mayor valor numérico. Cada vez que se ejecuta una tarea, se incrementa la variable switch, permitiendo que se ejecuten las tareas de menor prioridad. Si se produce la interrupción, se asigna a la variable switch el valor correspondiente a la tarea de mayor prioridad. Pero si la interrupción tarda bastante en producirse nuevamente, la variable switch permitirá la ejecución de la tarea de menor prioridad, para después comenzar automáticamente con la de mayor prioridad.

Si la interrupción se produce en el case de nivel 2, la variable switch se pone a 0 y así sucesivamente. En este caso las tareas de menor prioridad se ignoran. El sistema no es obviamente ideal, ya que solo la tarea de mayor nivel se ejecuta con la suficiente frecuencia. Sin embargo, bajo condiciones normales puede ser una forma útil de asegurar que las tareas de baja prioridad no se ejecuten con mucha frecuencia. Por ejemplo, no tiene mucho sentido medir la temperatura ambiente más de una vez cada segundo. En un sistema de este tipo, la tarea encargada de medir la temperatura ambiente estaría situada a nivel 100, en el sistema de reparto de tareas.

Este método falla cuando una tarea de baja prioridad tiene un tiempo de ejecución muy largo. Incluso si se produce la interrupción que exige que el bucle regrese a la tarea de mayor prioridad, el salto a la misma no se producirá mientras no termine la tarea en curso. Para que se produzca la situación deseada se necesita de un mecanismo de rebanado de tiempo (time-slice).

Un truco útil consiste en utilizar una interrupción libre para garantizar que la tarea de alta prioridad se ejecute a tiempo. Para ello se asigna la tarea de alta prioridad a la rutina de servicio de la interrupción libre o sobrante. Así cuando se produzca la interrupción en tiempo real, y justo antes de salir de la rutina de servicio de la misma, se pondrá a uno el flag de petición de la interrupción libre, con lo cual, tras el RETI comenzará la ejecución de la tarea de alta prioridad. Por supuesto, la interrupción libre debe ser de baja prioridad.

Hay que tener en cuenta que en estos casos el factor más importante es conseguir el menor tiempo de ejecución posible, y particularmente en las rutinas de interrupción. Ello significa que se debe hacer un uso completo de las extensiones de C51, tales como los punteros específicos, los bits de funciones especiales y las variables locales de tipo register.

5. Interfaz Gráfica de Usuario (IGU)

El apartado “Interfaz Gráfica de Usuario” es una continuación del proyecto previo Interfaz USB-CAN. Primero se realiza una presentación visual de todas las ventanas y el camino para llegar a ellas. Posteriormente se explica el funcionamiento de todas y cada una de las herramientas que contiene el programa.

Se introducirá brevemente el funcionamiento interno del programa además de mostrar y explicar todos los mensajes de error que puede producirse durante su uso.

5.1. Ventanas

En este apartado se realiza una presentación de todas las ventanas de la Interfaz y como llegar a ellas. También se nombraran las zonas más importantes de la ventana, para así en futuras explicaciones poder hacer referencia a ellas, sin problemas. La forma de uso viene explicada en el siguiente apartado.

5.1.1. Ventana de Presentación y Principal

La ventana que hay a continuación es la **Ventana de Carga** o **Presentación**. Esta ventana se muestra mientras arranca el programa.

Ventana de Presentación

En ella visualizar los nombres de los estudiantes que han realizado el proyecto, el tutor que los ha supervisado y las entidades involucradas.

A continuación de esta ventana se nos mostrara la **Ventana Principal**. De la cual detallaremos su funcionamiento más adelante.

Ventana Principal

5.1.2. Ventana de Configuración del Puerto Serie

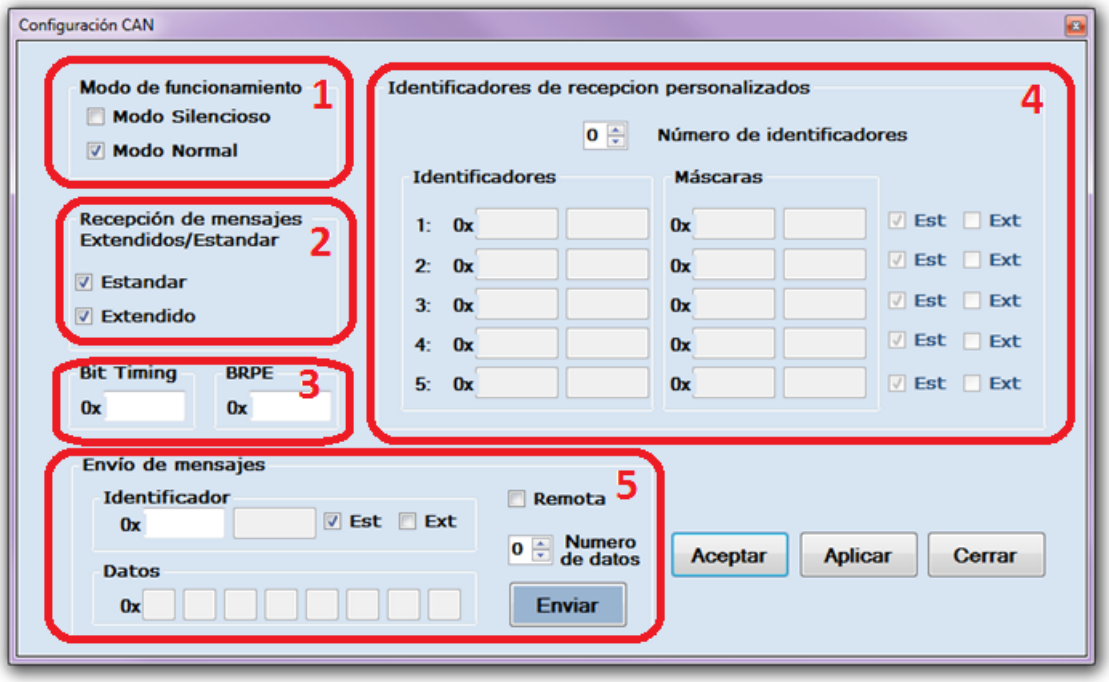
La ventana que hay a continuación es la ventana de configuración del puerto serie para poder conectarnos con la tarjeta. Para acceder a esta ventana hay dos caminos, desde **Configuración** en la **Ventana Principal** o en la de **Sniffer**.

Ventana de Configuración del Puerto Serie

En esta ventana se puede apreciar tres campos a seleccionar, el **Nombre** del puerto (1), B**audios** (2) a los que se va trasmitir y la **Paridad** (3).

5.1.3. Ventana de Configuración CAN

Para llegar a la ventana de **Configuración CAN**, hay que pulsar en la **Ventana Principal**, en la opción de **Configuración** y se desplegara un menú con varias opciones. Pulsaremos sobre la opción que deseamos, que es **Configuración CAN**. Y no mostrara la siguiente ventana.



Ventana de Configuración CAN

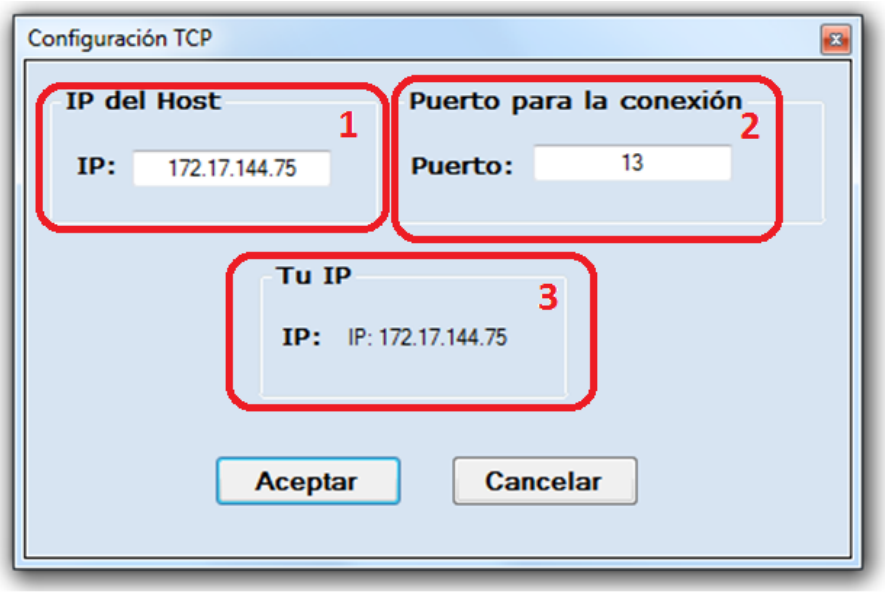
Es ventana se puede separa en varias secciones:

- **Primera:** Modo de funcionamiento del Nodo CAN.
- **Segunda:** Tipo de Identificador, si es Estándar o Extendido.
- **Tercera**: Configuración del Bit Timing y de el Byte BRPE.
- **Cuarta:** En esta sección se configuran los filtros de Identificar.
- **Quinta:** Aquí es donde se configura el mensaje a enviar.

Cada una de las secciones indicadas está explicada en el apartado de **Configuración del bus CAN**, salvo la sección de enviar mensaje, que se explicara en el apartado de **Enviar de mensajes**.

5.1.4. Ventana de Configuración TCP

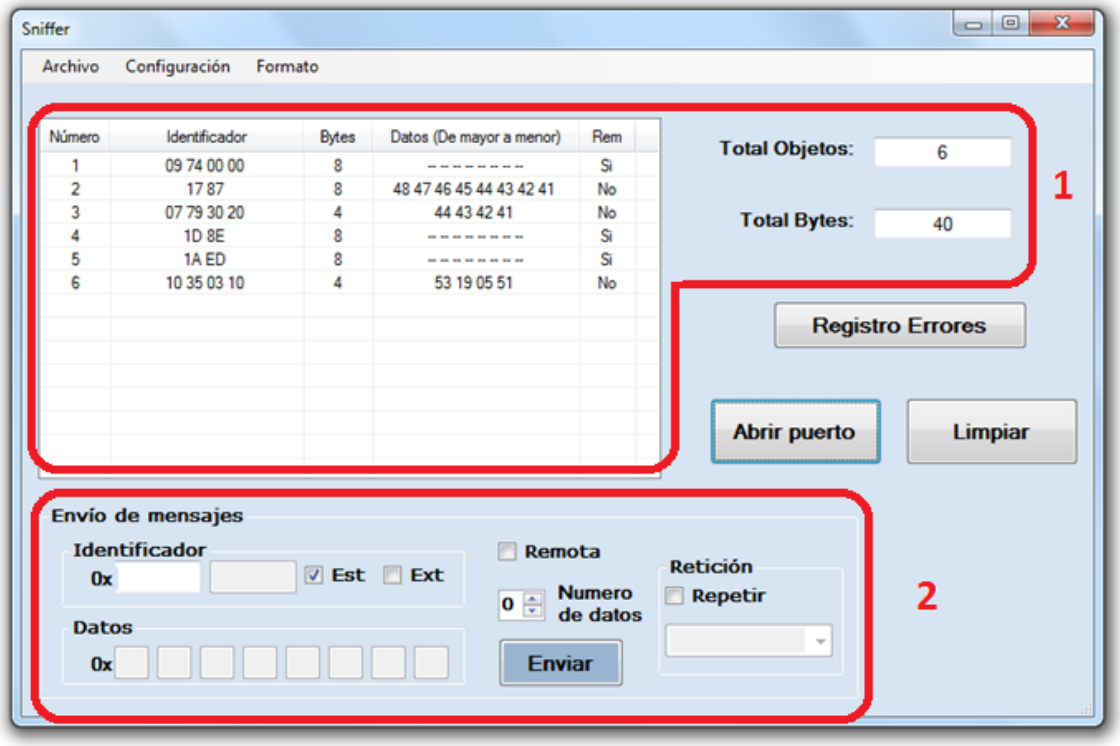
En la ventana de **Configuración de la Conexión TCP**, que se muestra a continuación, hay dos campos para rellenar, la **IP** del destino (1) y el **Puerto** (2). También muestra tu propia **IP** (3). Para acceder a esta ventana hay dos caminos, desde **Configuración** en la **Ventana Principal** o en la de **Sniffer**.



Ventana de Configuración TCP

5.1.5. Ventana de Sniffer

Esta ventana, junto con la de **Configuración CAN**, son las más importantes. Para acceder a esta ventana, hay que partir de la ventana principal. En esta pulsando sobre la opción de **Sniffer**, llegaremos a la venta de **Sniffer**. A continuación se muestra dicha ventana.

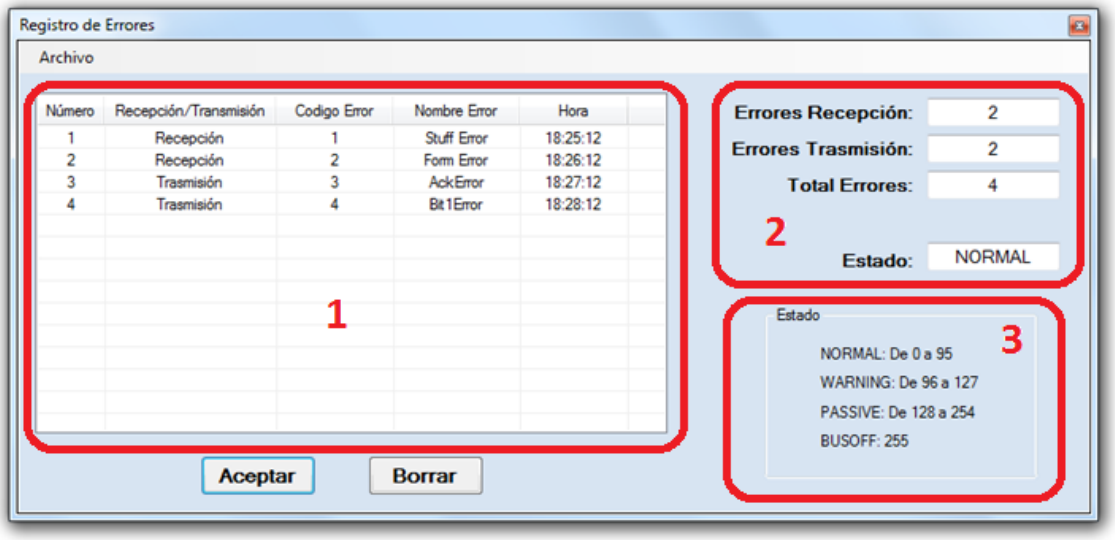


Ventana Sniffer

Esta ventana la podríamos separa en tres partes principalmente, la primera sería la de **Sniffer** (1), la segunda, la de **envío de mensajes** (2) y la tercera es el acceso al registro de errores. Más a delante se explicara el funcionamiento de cada una de las partes con detalle.

5.1.6. Ventana de Registro de Errores

Se accede a esta ventana desde la ventana de **Sniffer**. Esta ventana, que se muestra a continuación, tiene dos partes principalmente.

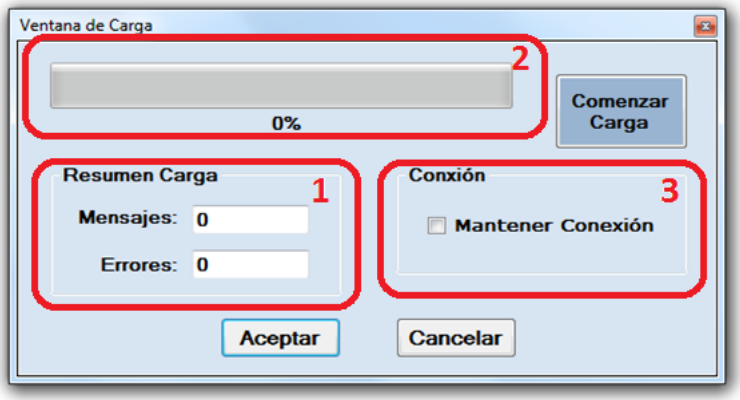


Ventana de Registro de Errores

La primera parte importante que se visualiza, es el **registro de errores** (1), donde se muestra una lista contos los errores. Y la segunda es el **contador de errores** (2) y el visualizador de estado. Por último hay una tercera sección en la que hay una breve **explicación del estado** (3) en que se encuentra según el número de errores.

5.1.7. Ventana de Carga

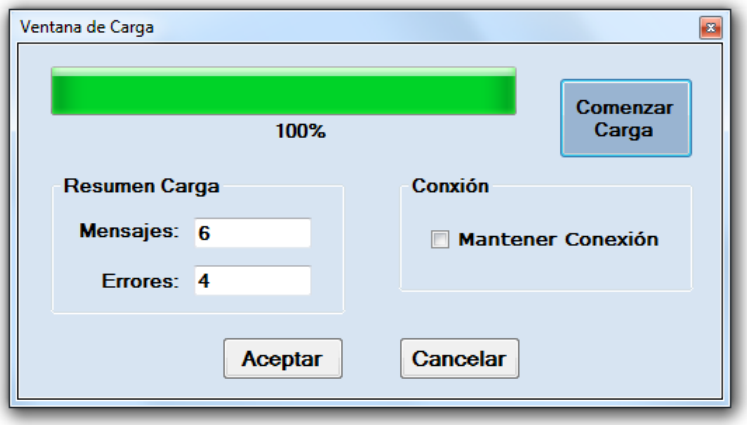
Esta es la ventana que se muestra para exportar los datos a través de una conexión TCP. A esta ventana solo se puede acceder a través de la ventana de **Sniffer>Importar>Desde Red**. Pero hay que tener previamente la **Configuración TCP** establecida.



Ventana de Carga

Esta ventana muestra dos cosas importantes, el **Resumen de Carga** (1), el cual nos muestra el total de mensajes y de errores trasmitidos; y la **barra de progreso** (2). Por ultima esta la opción de **Mantener la conexión** (3), para trasmisión en tiempo real.

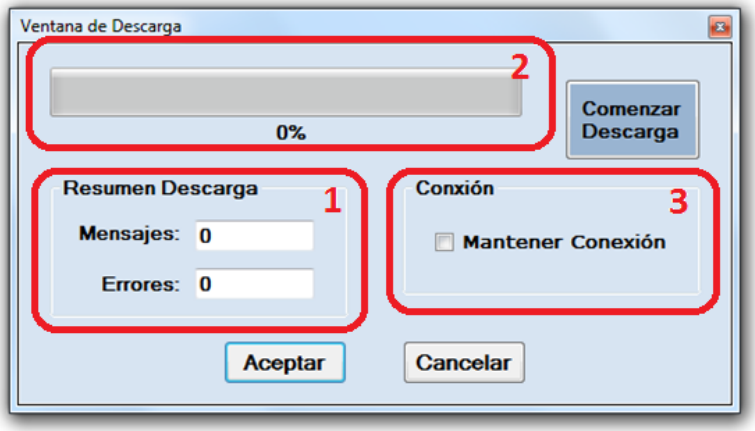
Al finalizar la carga te muestra el total se los mensajes y errores trasmitidos, como se muestra en la siguiente imagen.



Ventana de Carga Completa

5.1.7. Ventana de Descarga

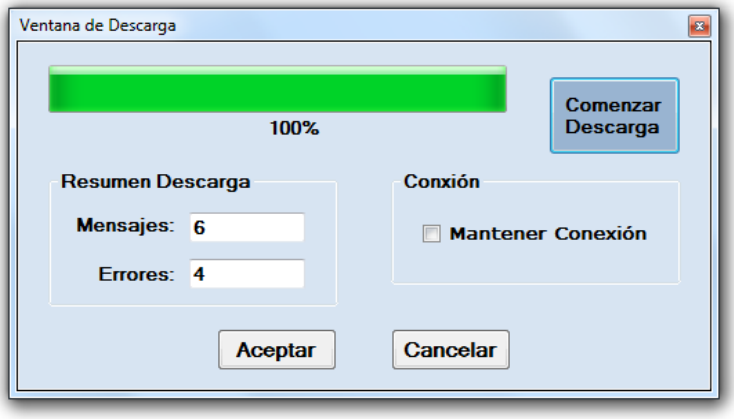
Esta es la ventana que se muestra para importar los datos a través de una conexión TCP. A esta ventana solo se puede acceder a través de la ventana de **Sniffer>Exportar>A Red**. Pero hay que tener previamente la **Configuración TCP** establecida.



Ventana de Descarga

Esta ventana muestra dos cosas importantes, el **Resumen de Descarga** (1), el cual nos muestra el total de mensajes y de errores trasmitidos; y la **barra de progreso** (2). Por ultima esta la opción de **Mantener la conexión** (3), para trasmisión en tiempo real.

Al finalizar la carga te muestra el total se los mensajes y errores trasmitidos, como se muestra en la siguiente imagen.



Ventana de Descarga Completa

5.2. Funcionamiento

En esta apartado se van a mostrar y explicar todas las funciones y opciones existentes en la Interfaz.

5.2.1. Configurar puerto Serie

En este apartado vamos a configurar de forma rápida y fácil, la comunicación serie entre el PC y la tarjeta.

Como se dijo con anteriormente, en la ventana de **Configuración Puerto Series**, solo hay tres parámetros para seleccionar.

Puerto: Hay que elegir entre los Puertos COM que hay, para saber en que puerto tenemos conectada la tarjeta, tendremos que acudir al Administrador de dispositivos de Windows. Ahí se puede ver en qué Puerto COM está.

Baudios: Con este parámetro indicamos a la cantidad de veces que cambia el estado de la señal en un periodo de tiempo, durante la comunicación entre el programa y la tarjeta. Es necesario para su correcto funcionamiento que en ambos extremos del bus series, se configuren las mismas velocidades. De lo contrario será imposible la comunicación.

Paridad: La paridad puede ser de varios tipos, aunque en este caso no usaremos ninguna.

- None: Ninguna.
- Even: Paridad par.
- Odd: Paridad impar.
- Mark: Paridad marca
- Space: Paridad espacio.

5.2.2. Configurar bus CAN

En el apartado de **Configurar el bus CAN**, vamos a explicar cómo y cuando se configuran cada una de las secciones que se han listado anteriormente. Salvo la sección de envío de mensajes, que se explica en su correspondiente apartado.

Modo de funcionamiento del Nodo CAN:

El modo de funcionamiento solo puede ser una de las dos opciones y nunca de forma simultánea, ya que son contradictorios.

- El modo Silencioso, significa que el Nodo no responderá a ningún en vio con un ACK. Solo vigila el bus. De esta manera evitamos incrementar el tráfico dentro del bus. Este es el modo de funcionamiento idóneo para cuando usemos el Sniffer, ya que no se capturara ningún mensaje de nuestro Nodo.

- El modo Normal, es el opuesto al Silencioso, es decir, que si algún mensaje va dirigido a él, responderá con su correspondiente ACK. Este modo se usa para establecer la comunicación con otro Nodo.

Tipo de Identificador:

El tipo de Identificador nos determinara el número de bits del mismo y por consiguiente limitara el número de Nodos. Por lo contrario que ocurre en el **Modo de Funcionamiento**, aquí si se pueden marcar ambas opciones, de esta manera podrá aceptar los dos tipos de Identificadores, ya que no son opuestos.

- Estándar: Este tipo indica que el número de bits es de 11, por lo que el número de Nodos no puede ser superior a 2048.

- Extendido; El Identificador Extendido tiene un espacio de 29 bits, que corresponde a más de 536 millones de Nodos.

Configuración del Bit Timing y el Byte BRPE:

Para entender estas opciones de configuración, realizaremos un breve introducción al tiempo de bit, donde se explica el significado y la utilidad de estas opciones.

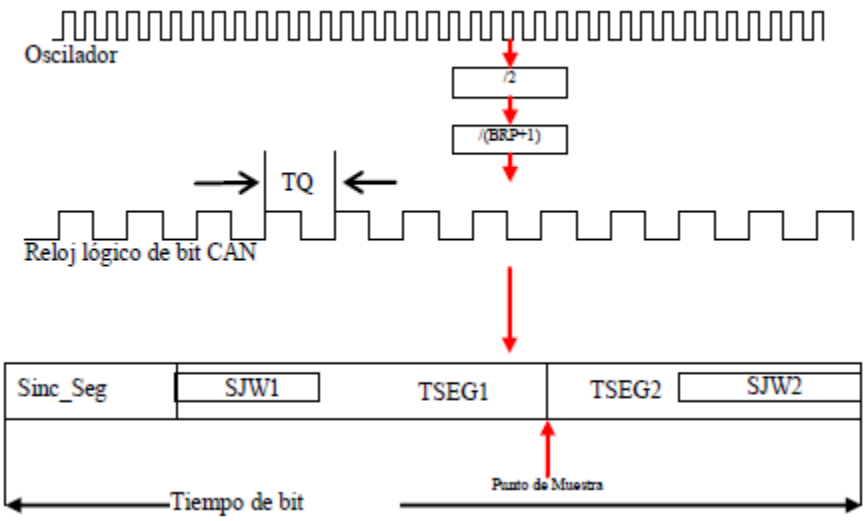
Nota: Significa lo mismo BRPE que BRP.

Las señales eléctricas reales en un bus CAN sufren las alteraciones propias de la distorsión que se produce por las características físicas de la línea, los retardos de propagación, y los producidos en los propios controladores y las posibles fuentes externas de interferencia electromagnética. Por otra parte, cada controlador CAN en un bus depende, normalmente, de un oscilador distinto, entre estos osciladores existen diferencias de frecuencia que pueden dar lugar a desfases en el muestreo de tramas en los distintos nodos. Los controladores CAN siguen un proceso de muestreo y resincronización orientado a evitar, los desajustes debidos a estos factores.

Estos desajustes se manifiestan especialmente en el arbitraje. Varios transmisores que inicien transmisión en forma simultánea, no estarán tan perfectamente sincronizados en el inicio, además su frecuencia de oscilador puede ser ligeramente distinta. Los receptores que se han sincronizado con el primer flanco de bajada detectado, deberán irse resincronizando sucesivamente a los flancos producidos por los o el transmisor que resulte vencedor en la contienda.

El bus CAN utiliza señalización asíncrona con codificación NRZ, es necesaria, por tanto, una operación de muestreo en los receptores y el muestreo se ha de resincronizar periódicamente.

Para asegurar la funcionalidad apropiada de una red CAN también es importante probar un bit a la posición correcta dentro de un tiempo del bit.

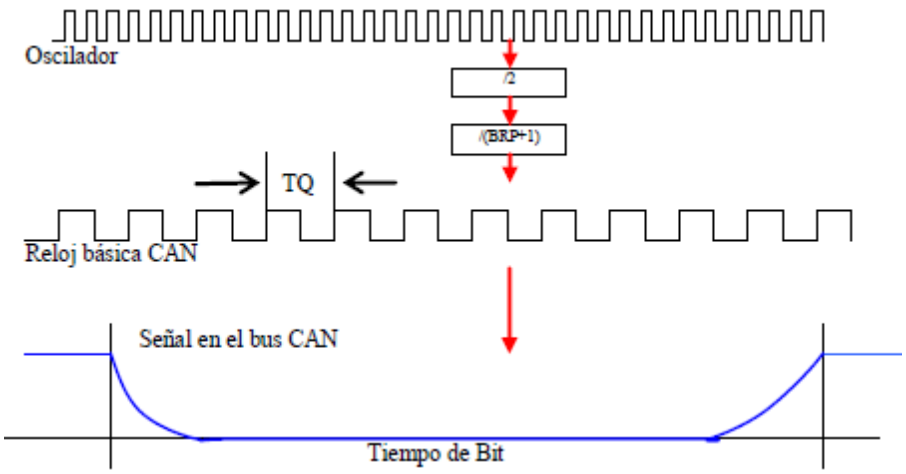


La tasa de transmisión también es un parámetro importante de un sistema de red de computadoras. Las tasas de la transmisión soportadas están desde menos de 1 kbps y sobre un máximo de 1Mbps.

Todos estos parámetros pueden programarse individualmente y pueden ser ejecutados por el reloj lógico CAN (Bit Timing Logic -BTL).

Según la especificación de la CAN, un tiempo del bit se subdivide en tres segmentos (vea Fig.N°9). Los InSync segmentan, el Time Segment 1 (TSEG1) y Time Segment 2 (TSEG2). Los segmentos están compuestos de un cierto (programable) número de ciclos de BTL. Los ciclos de BTL se definen del oscilador de cuarzo y un preescalar (Baud Rate Prescaler)

Para que el controlador, que es un sistema digital síncrono con frecuencia de reloj generada por un oscilador, pueda muestrear la señal asíncrona recibida con seguridad y precisión ha de utilizar una frecuencia de muestreo superior a la de la señal transmitida en el bus.



En la especificación CAN se contemplan las siguientes definiciones:

Frecuencia nominal de bit (Nbr, "'nominal bit rate"): se define como el número de bits por segundo transmitidos en ausencia de resincronizaciones por un transmisor ideal. Su inverso es el tiempo nominal de bit (Nbt, "nominal bit time")

El controlador CAN ha de utilizar una frecuencia de reloj múltiplo de la frecuencia nominal de bit, esa frecuencia se obtiene como cociente de la frecuencia del oscilador. Se define como cuánto de tiempo mínimo (Mtq o "minimum time quantum") es obtenido a partir de la frecuencia de oscilador (es la misma frecuencia o mitad), y como cuánto de tiempo básico (Tq) (su inverso es la frecuencia básica del controlador) es obtenido al dividir la frecuencia de esa señal por un divisor de frecuencia (Brp o "baud rate prescaler") que es normalmente configurable.

$$Tq = Brp \cdot Mtq$$

En la especificación CAN se establece que el Brp ha de poder configurarse al menos en el rango de valores 1 a 32 (suele ser un registro, BRP, en el que se pueden asignar los valores 0..31, por tanto la división es por BRP+1) La frecuencia básica del controlador es así varias veces superior a la de comunicación en el bus, el número de Tq en un bit, NTq ha de poder programarse en un rango de, al menos, 8 a 25.

El bit se muestrea en un punto dado del tiempo de bit (en algunos controladores y para bajas velocidades es posible muestreo múltiple con decisión por mayoría). La velocidad de comunicación a la que queda configurado el controlador, para un oscilador de frecuencia dada, queda definida por el valor del BRP, y por el número de TQ por bit. Este último valor se obtiene como suma de varios valores parciales, configurables también, que tienen que ver con la sincronización de bit y que se detallarán a continuación.

5.2.3. Configurar conexión TCP

La configuración de la conexión TCP es necesario realizarla antes de empezar a exportar o importar datos.

En dicha ventana solo hay que rellenar dos campos:

- La **IP** del destino: Aquí es donde pondremos la IP del destino o lo que es lo mismo, la Ip del ordenador con el cual queremos conectar. Al abrir la ventana, se muestra por defecto nuestra propia Ip, ya que es posible realizar conexiones TCP con el propio ordenador, pero para ello habría que tener en ejecución dos veces el este programa. Hasta que no se pulse el botón aceptar no memorizara ninguna configuración, aunque la este mostrando

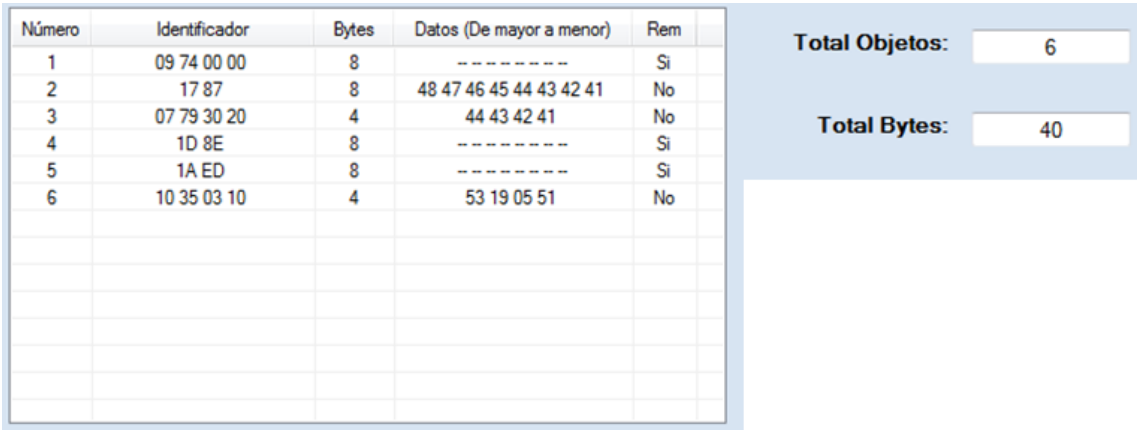
- El **Puerto**: Este campo debe ser igual en los dos PC's que se vayan a conectar entre si. Y aunque la conexión sea con nuestro propio ordenador, el puerto también debe coincidir en los dos programas en ejecución.

5.2.4. Sniffer

Como ya se ha mencionando anteriormente, esta es una de las herramientas más importante y en la que se basa el proyecto.

Anteriormente ya se explico que la ventana está dividida en tres partes, aunque en este apartado solo explicaremos la parte de Sniffer y del registro de errores, ya que la parte de envío de mensajes esta explicada en el apartado de **Enviar mensaje**.

Comenzaremos con la parte del Sniffer:



En la parte de Sniffer podemos diferenciar dos partes, la que está a la izquierda, que es el registro de mensajes y la de la derecha, que son los contadores de Bytes y objetos o mensajes.

En el registro se visualizan cinco campos:

- **Número:** Numeración de los mensajes. El valor del último digito de esta columna siempre es el mismo que el valor del contador de Objetos.

- **Identificador:** Muestra el identificador cada mensaje. Este puede tener dos longitudes, que variaran según sea el identificador de 11 o 29.

- **Bytes:** En este campo se indica la longitud en Bytes del campo Datos.

- **Datos:** Muestra los datos que contiene el mensaje. Si el mensaje es una trama remota, se nos mostrara una línea discontinua en su lugar.

- **Rem:** Nos indica si la trama es remota o no. De serlo, en el campo de Datos se mostrara una línea discontinua, como ya se ha dicho anteriormente.

Los campos de Identificador y de Datos se pueden visualizar tanto en Hexadecimal como en Decimal. Esto se puede cambiar desde el menú de **Formato**, aquí podremos elegir entre Hexadecimal o Decimal.

En cuanto a los dos contadores, el primero que es el número de objetos, este siempre debe coincidir con el número del último mensaje recibido. El segundo, que es el contador de Bytes, ira aumentando en cantidades que dependerá del tipo de mensaje, es decir, que dependiendo del tamaño de identificador o de los datos irá variando el incremento de este contador.

5.2.5. Tramas remotas

En apartado del Bus Can se explica detenidamente lo que es una trama remota, en este apartado explicaremos las dos partes en las que hay que tener en cuenta la trama remota.

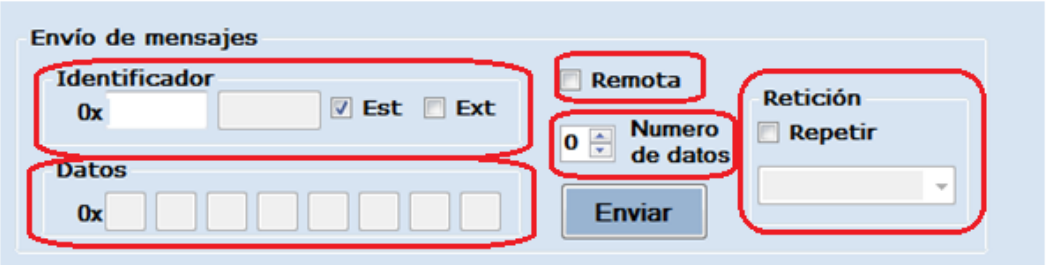
Una es a la hora de enviar mensajes, en dicha herramienta tenemos la opción de enviar tramas remotas, con solo marcarla en el CheckBox. Al hacerlo el campo de datos se bloqueara, ya que una trama remota no contiene datos y aunque los tuviese, el receptor los ignorara ya que también lo leerá como una trama remota.

La otra parte donde tenemos que tener en cuenta las tramas remostas es en el Sniffer, aunque este de forma automática nos indicara si es remota o no. De serlo el campo de datos estará en blanco.

5.2.6. Enviar mensaje

El envío de mensajes se puede realizar desde dos ventanas, una es la de **Sniffer** y otra es la de **Configuración CAN**.

La explicación se remitirá a la ventana de Sniffer, ya que es la más completa al tener la opción de envío repetido.



Como podemos ver en la imagen anterior, la herramienta de envío de mensajes se puede separa en varias partes. Explicaremos paso a paso como configurar el mensaje y enviarlo:

- **Paso uno:** Configuraremos el Identificador. Primero debemos elegir el tipo, es decir, si es Extendido (29 bits) o Estándar (11 bits). Si elegimos el Estándar, la segunda casilla para rellenar se bloqueara, ya que con la primera hay suficiente para 11 bits. En cambio, si elegimos la Extendida, permitirá rellenar ambas, aunque no sea necesario.

- **Paso dos:** Indicaremos si es Remota o no, ya que de serlo el campo de datos se bloqueara automáticamente. Por que como ya hemos explicado, las tramas remotas no contienen datos.

- **Paso tres:** Si no hemos marcado el mensaje como trama remota, debemos indicar el número de Bytes que contiene el campo de datos, no se permite más de 8.

- **Paso cuatro:** Introducir los datos, byte a byte. Tantos como hallamos indicado anteriormente. Al igual que en el paso anterior, este solo se podrá rellenar si no hemos marcado la opción de trama remota.

- **Paso quinto:** Aquí es donde indicamos si el mensaje se va a enviar de forma repetida. Para ello marcaremos la opción de **Repetir**. Al hacerlo se desbloque la pestaña que hay justo debajo. Ahí es donde podremos elegir el tiempo entre mensajes, nos da las opciones de 5, 10 y 30 segundos o un minuto.

Una vez realizados todos los pasos, podemos enviar el mensaje.

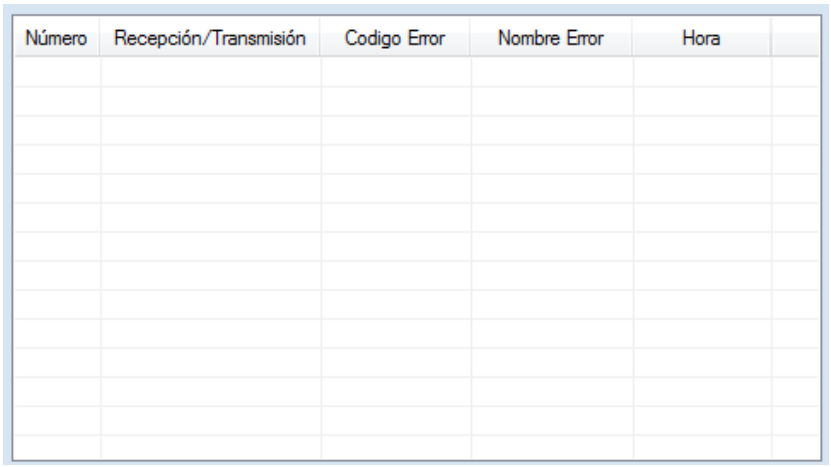
En la ventana de **Configuración CAN** se hace exactamente de la misma manera, salvo la parte de repetir, ya que no la tiene.

5.2.7. Registro de errores

Esta ventana, junto con la de Sniffer, son las ventanas donde se muestran los datos recibidos o importados, ya sea desde el propio ordenador o desde otro, gracias a la conexión TCP.

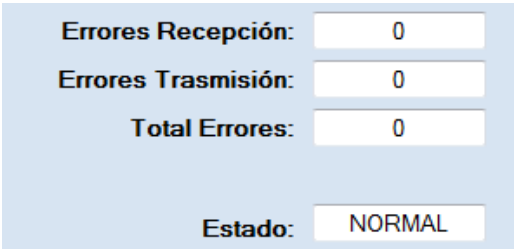
Como ya se ha mencionado con anterioridad, hay dos secciones en esta ventana.

La primera es el listado de errores, en el cual hay cinco campos:



- **Número:** Numeración de los errores, se ordenan según van llegando.
- **Recepción/Trasmisión:** Nos indica si el error es de Trasmisión o de Recepción.
- **Código Error:** Con una numeración de entre uno y cuatro, ambos incluidos, nos indica la causa del error.
- **Nombre Error:** Nos indica el nombre del error, correspondiente con su código.
- **Hora:** Nos indica el momento en el que ocurrió.

La segunda parte es el recuento de errores, en el que hay tres contadores:



- **Errores Recepción:** Contador de errores de Recepción.
- **Errores Trasmisión:** Contador de errores de Trasmisión.
- **Total Errores:** Cuenta total de errores, se halla sumando los errores de trasmisión con los de recepción.
- **Estado:** Nos indica en qué estado se encuentra internamente nuestro Nodo CAN

5.2.8. Importar datos

En el programa hay dos formas de importar datos:

Desde PC

Hay dos tipos de datos que se pueden importar, los datos del Sniffer y los datos del Registro de errores. Cada tipo de dato se importara desde su propia ventana, es decir por separado.

La extensión que reconoce este programa es TXT, además de que requiere un formato especial para que lo pueda leer. Por este motivo solo podrá leer archivos de texto que haya creado el mismo.

Dichos ficheros podrán ser abiertos y analizados sin necesidad del programa, pero su uso facilita mucho más su entendimiento.

El formato en el que se guardan los datos del Sniffer, es Hexadecimal. Al importar el fichero podremos visualizarlo tanto en Hexadecimal como en decimal. Por otro lado los errores se guardan tal y como se muestran en el programa

Para que el programa pueda encontrar el archivo debe de estar dentro de una carpeta y llamarse “Sniffer.txt”, la parte de los mensajes y “errores.txt”. Cuando lo exportamos también lo guarda con ese nombre.

En el apartado de exportar datos se realiza una breve explicación de cómo se guardan los datos en el fichero de texto y como se puede leer sin necesidad de importarlos.

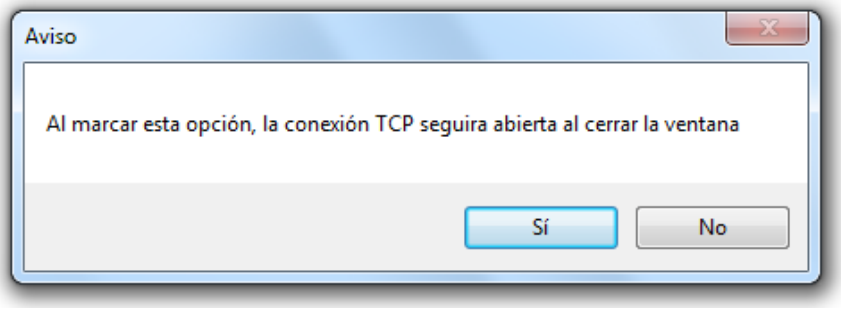
Desde Red

Al importar los datos desde la red no hay que realizarlo una vez para los mensajes y otra para los errores. Se importa todo lo que tenga el otro ordenador.

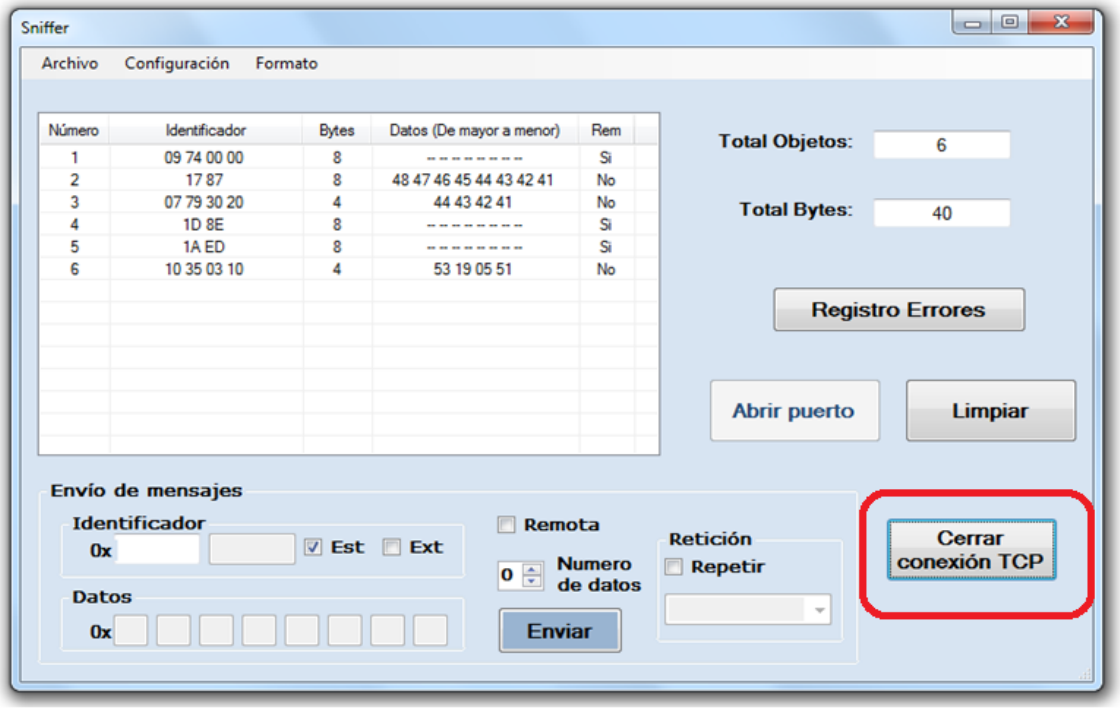
Nota: Para poder realizar la importación hay que realizar la configuración TCP primero, lo cual se explica en el apartado de cómo Configurar la Conexión TCP.

En la ventana de descarga esta la opción de mantener la conexión, esta opción sirve para la recepción de datos en tiempo real, es decir al mismo tiempo que al ordenador al que estés conectado le van llegando información desde la tarjeta, la va mandando a través de la conexión TCP.

Al marcar dicha opción aparece la siguiente ventana:



En esta ventana nos explica que al dejar la opción marcada, lo que conseguimos es la trasmisión de datos en tiempo real. Por lo que al sincronizarse y pasarse todos los datos ya almacenados, se cerrara la ventana de descarga y en la ventana de Sniffer se irán mostrando todos los datos según vayan llegando.



Como se muestra en la ventana Sniffer, aparece un botón nuevo para poder cerrar la conexión TCP, cuando sea necesario. Al mismo tiempo está bloqueado el botón de **Abrir Puerto**, hasta que se cierre la conexión TCP.

Aunque no esté abierta la ventana de registro de errores, se irán almacenando para posteriormente poder mostrarlos.

5.2.9. Exportar datos

En el programa hay dos formas de exportar datos:

Desde PC

Hay dos tipos de datos que se pueden exportar, los datos del Sniffer y los datos del Registro de errores. Cada tipo de dato se exportara desde su propia ventana, es decir por separado.

La extensión en la que se guardan los datos es TXT, y con un formato especifico para que se pueda leer. Por este motivo solo podrá leer archivos de texto que haya creado el mismo.

Dichos ficheros podrán ser abiertos y analizados sin necesidad del programa, pero su uso facilita mucho más su entendimiento.

El formato en el que se guardan los datos del Sniffer, es Hexadecimal. Al exportar el fichero podremos visualizarlo tanto en Hexadecimal como en decimal. Por otro lado los errores se guardan tal y como se muestran en el programa

Para que el programa pueda encontrar el archivo debe de estar dentro de una carpeta y llamarse “Sniffer.txt”, la parte de los mensajes y “errores.txt”. Cuando lo exportamos también lo guarda con ese nombre.

A continuación pondremos un breve ejemplo de cómo leer el fichero resultante de exportar mensajes y posteriormente haremos lo mismo con el fichero resultante de la exportación de errores.

Fichero de Mensajes:

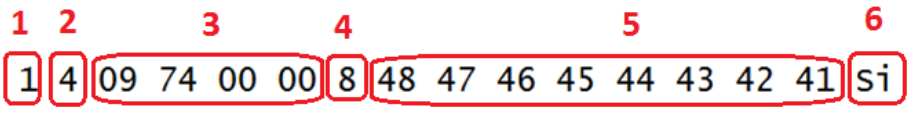
```
6        20/06/2012 18:25:12
1 4 09 74 00 00 8 48 47 46 45 44 43 42 41 Si
2 2 17 87 8 48 47 46 45 44 43 42 41 No
3 4 07 79 30 20 4 44 43 42 41 No
4 2 1D 8E 8 48 47 46 45 44 43 42 41 Si
5 2 1A ED 8 48 47 46 45 44 43 42 41 Si
6 4 10 35 03 10 4 53 19 05 51 No
```

Analizaremos las dos primeras líneas:

- Primera:

En esta línea hay tres partes importantes. De izquierda a derecha, el primer digito es el número de líneas o de mensajes que hay en el fichero, el segundo y el tercero, son la fecha y la hora respectivamente.

- Segunda: A partir de la segunda línea, cada una corresponde a una fila de la lista que se muestra en la ventana de Sniffer.



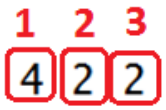
El primer campo es el número de fila, el segundo y el cuarto es el número de bytes del identificador y de datos correspondientemente. El tercero es el identificador y el quinto son los datos. Por último, el sexto campo indica si es una trama remota. Si fuese afirmativo, como es el caso, el programa a la hora de leer los datos o mostrarlos, ignoraría el campo de datos.

Fichero de Errores:

```
4 2 2
1 Recepción 1 18:25:12
2 Recepción 2 18:26:12
3 Trasmisión 3 18:27:12
4 Trasmisión 4 18:28:12
```

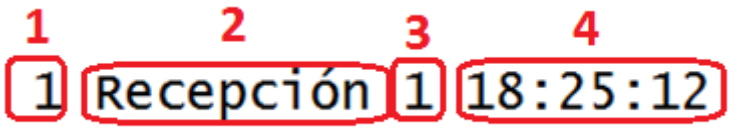
Analizaremos las dos primeras líneas:

- Primera:



En esta línea hay tres partes importantes. De izquierda a derecha, el primer digito es el número de líneas o el número de total de errores que hay en el fichero, el segundo y el tercero, son el número de errores de Trasmisión y de Recepción, respectivamente.

- Segunda: A partir de la segunda línea, cada una corresponde a una fila de la lista que se muestra en el registro de errores.



El primer campo es el número de fila, el segundo indica si el error es de trasmisión o de recepción. El tercer campo nos muestra el código del error y en último lugar, la hora a la que se produjo el error.

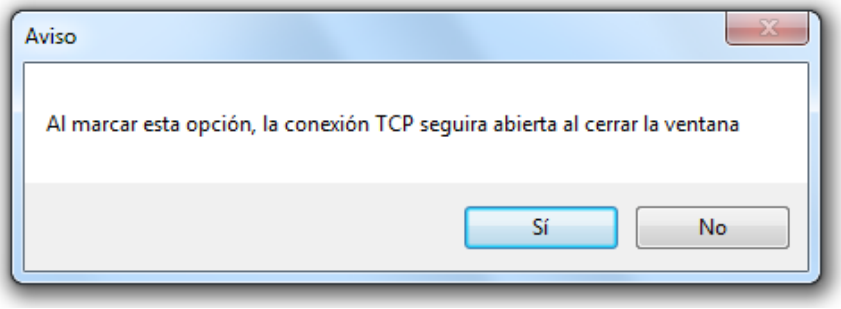
Desde Red

Al exportar los datos desde la red no hay que realizarlo una vez para los mensajes y otra para los errores. Se exportara todo lo que tengamos almacenado.

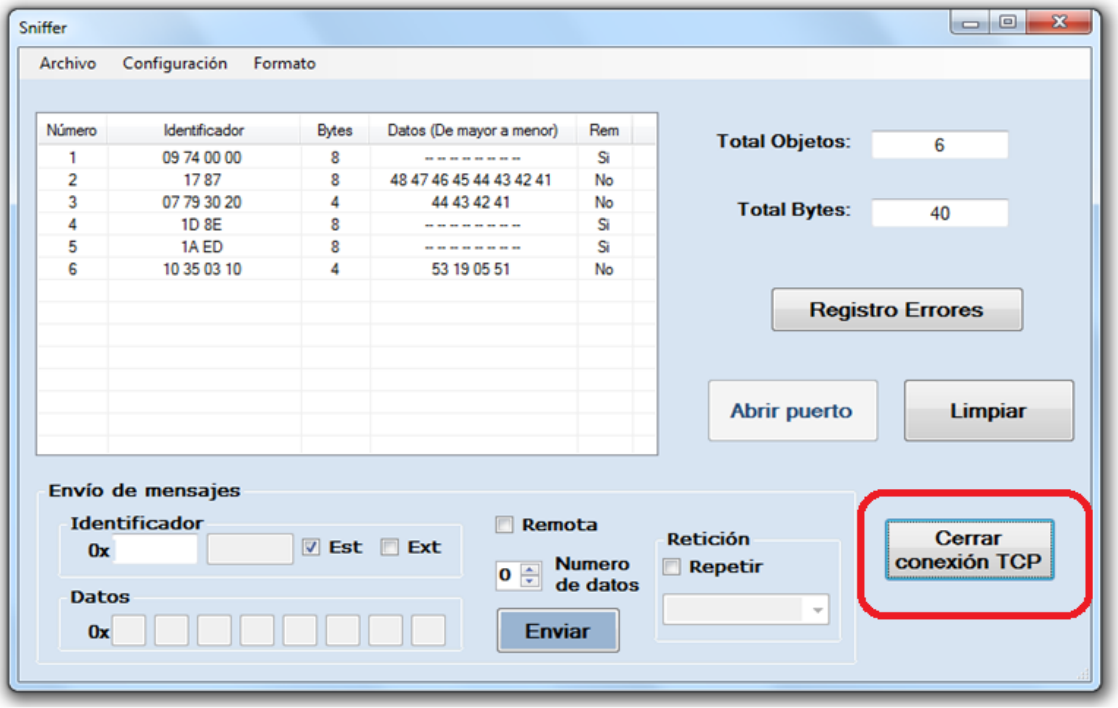
Nota: Para poder realizar la exportación hay que realizar la configuración TCP primero, lo cual se explica en el apartado de cómo Configurar la Conexión TCP.

En la ventana de carga esta la opción de mantener la conexión, esta opción sirve para la trasmisión de datos en tiempo real, es decir, al mismo tiempo que el ordenador esta recibiendo los datos desde la tarjeta a través del puerto serie. Los va mandando por medio de la conexión TCP.

Al marcar dicha opción aparece la siguiente ventana:



En esta ventana nos explica que al dejar la opción marcada, lo que conseguimos es la trasmisión de datos en tiempo real. Por lo que al sincronizarse y pasarse todos los datos ya almacenados, se cerrara la ventana de carga y en la ventana de Sniffer se irán mostrando todos los datos según vayan llegando por el puerto series y los ira trasmitiendo.



Como se muestra en la ventana Sniffer, aparece un botón nuevo para poder cerrar la conexión TCP, cuando sea necesario. Al mismo tiempo está bloqueado el botón de **Abrir Puerto**, hasta que se cierre la conexión TCP.

Aunque no esté abierta la ventana de registro de errores, se irán trasmitiendo los errores de la misma manera que los mensajes.

5.3. Funcionamiento Interno

Realizaremos un breve resumen sobre el funcionamiento interno de la Interfaz, sin llegar a mostrar el código, ya que no es necesario.

El lenguaje de programación elegido para realizare la Interfaz es C#. A continuación realizaremos una introducción a este lenguaje y a UML. Después de esto se realiza una explicación de cómo están estructuradas internamente las clases.

5.3.1. Introducción a C#

El lenguaje de programación orientado a objetos C parte de un lenguaje anterior, el lenguaje B, escrito por Ken Thompson en 1970 con el objetivo de recodificar el sistema operativo UNIX, que hasta el momento se había programado en ensamblador. A su vez B fue inspirado en el BCPL de Martin Richard, diseñado tres años antes.

En 1972 es Dennis Ritchie (de los Laboratorios Bell de AT&T) quien diseña finalmente C a partir del B de Thompson, aportando un diseño de tipos y estructuras de datos que consiguen una claridad y eficacia en el lenguaje muy superior. Es un lenguaje que permite realizar una programación estructurada economizando las expresiones, con abundancia de operadores y tipos de datos (aunque los básicos sean pocos), codificando en alto y bajo nivel simultáneamente, reemplazando ventajosamente la programación en ensamblador y permitiendo una utilización natural de las funciones primitivas del sistema.

Paralelamente, en 1980 surge C++ de la mano de Bjarne Stroustrup (también de Laboratorios Bell de AT&T). Diseña este lenguaje con el objetivo de añadir a Cnuevas características: clases y funciones virtuales (de SIMULA67), tipos genéricos y expresiones (de ADA), la posibilidad

de declarar variables en cualquier punto del programa (de ALGOL68), y sobre todo, un auténtico motor de objetos con herencia múltiple que permite combinar la programación imperativa de C con la programación orientada a objetos. Estas nuevas características mantienen siempre la esencia del lenguaje C: otorgan el control absoluto de la aplicación al programador, consiguiendo una velocidad muy superior a la ofrecida por otros lenguajes. El siguiente hecho fundamental en la evolución de C++ es sin duda la incorporación de la librería STL años más tarde, obra de Alexander Stepanov y Adrew Koening. Esta librería de clases con contenedores y algoritmos genéricos proporciona a C++ una potencia única entre los lenguajes de alto nivel.

Durante muchos años no existieron reglas estándar para el lenguaje, pero en 1983 se decide formar un comité con el objetivo de crear el estándar ANSI (Instituto Nacional Americano de Estándares). El proceso duro seis años y a principios de los90, el estándar es reconocido por la ISO (Organización Internacional de Estándares)y comienza a comercializarse con el nombre ANSI C.

Debido al éxito del lenguaje, en 1990 se reúnen las organizaciones ANSI e ISO para definir un estándar que formalice el lenguaje. El proceso culmina en 1998 con la aprobación del ANSI C++.

La última variante que ha surgido de C es el moderno C#. En el año 2000, Microsoft presenta su plataforma .NET junto con un nuevo lenguaje, C# (diseñado por Anders Hejlsberg), que servirá de lenguaje principal de la plataforma. C# es un híbrido de C++ y Java que fusiona, principalmente, la capacidad de combinar operadores propia del primero (no incorpora la herencia múltiple) con la plena orientación a objetos del segundo. La orientación a objetos es tal que el propio programa está encapsulado en una clase. Actualmente C# se encuentra entre los 10 lenguajes más utilizados. A pesar de su corta historia, ha recibido la aprobación del estándar de dos organizaciones: en el 2001 se aprueba el ECMA y en el 2003 el ISO.

5.3.2. Introducción a UML

Cualquier rama de ingeniería o arquitectura ha encontrado útil desde hace mucho tiempo la representación de los diseños de forma gráfica. Desde los inicios de la informática se han estado utilizando distintas formas de representar los diseños de una forma más bien personal o con algún modelo gráfico. La falta de estandarización en la manera de representar gráficamente un modelo impedía que los diseños gráficos realizados se pudieran compartir fácilmente entre distintos diseñadores.

Se necesitaba por tanto un lenguaje no sólo para comunicar las ideas a otros desarrolladores sino también para servir de apoyo en los procesos de análisis de un problema. Con este objetivo se creó el Lenguaje Unificado de Modelado (UML: Unified Modeling Language). UML se ha convertido en ese estándar tan ansiado para representar y modelar la información con la que se trabaja en las fases de análisis y, especialmente, de diseño.

El lenguaje UML tiene una notación gráfica muy expresiva que permite representar en mayor o menor medida todas las fases de un proyecto informático: desde el análisis con los

casos de uso, el diseño con los diagramas de clases, objetos, etc., hasta la implementación y configuración con los diagramas de despliegue.

Modelado Visual

Tal como indica su nombre, UML es un lenguaje de modelado. Un modelo es una simplificación de la realidad. El objetivo del modelado de un sistema es capturar las partes esenciales del sistema. Para facilitar este modelado, se realiza una abstracción y se plasma en una notación gráfica. Esto se conoce como modelado visual.

El modelado visual permite manejar la complejidad de los sistemas a analizar o diseñar. De la misma forma que para construir una choza no hace falta un modelo, cuando se intenta construir un sistema complejo como un rascacielos, es necesario abstraer la complejidad en modelos que el ser humano pueda entender.

UML sirve para el modelado completo de sistemas complejos, tanto en el diseño de los sistemas software como para la arquitectura hardware donde se ejecuten.

Otro objetivo de este modelado visual es que sea independiente del lenguaje de implementación, de tal forma que los diseños realizados usando UML se puedan implementar en cualquier lenguaje que soporte las posibilidades de UML (principalmente lenguajes orientados a objetos).

UML es además un método formal de modelado. Esto aporta las siguientes ventajas:

- Mayor rigor en la especificación.
- Permite realizar una verificación y validación del modelo realizado.
- Se pueden automatizar determinados procesos y permite generar código a partir de los modelos y a la inversa (a partir del código fuente generar los modelos). Esto permite que el modelo y el código estén actualizados, con lo que siempre se puede mantener la visión en el diseño, de más alto nivel, de la estructura de un proyecto.

¿Qué es UML?

UML es ante todo un lenguaje. Un lenguaje proporciona un vocabulario y unas reglas para permitir una comunicación. En este caso, este lenguaje se centra en la representación gráfica de un sistema.

Este lenguaje nos indica cómo crear y leer los modelos, pero no dice cómo crearlos. Esto último es el objetivo de las metodologías de desarrollo.

Los objetivos de UML son muchos, pero se pueden sintetizar sus funciones:

- Visualizar: UML permite expresar de una forma gráfica un sistema de forma que otro lo puede entender.

- Especificar: UML permite especificar cuáles son las características de un sistema antes de su construcción.

- Construir: A partir de los modelos especificados se pueden construir los sistemas diseñados.

- Documentar: Los propios elementos gráficos sirven como documentación del sistema desarrollado que pueden servir para su futura revisión.

Aunque UML está pensado para modelar sistemas complejos con gran cantidad de software, el lenguaje es los suficientemente expresivo como para modelar sistemas que no son informáticos, como flujos de trabajo (workflow) en una empresa, diseño de la estructura de una organización y por supuesto, en el diseño de hardware.

Un modelo UML está compuesto por tres clases de bloques de construcción:

- Elementos: Los elementos son abstracciones de cosas reales o ficticias (objetos, acciones, etc.)
- Relaciones: relacionan los elementos entre sí.
- Diagramas: Son colecciones de elementos con sus relaciones.

Diagramas UML

Un diagrama es la representación gráfica de un conjunto de elementos con sus relaciones. En concreto, un diagrama ofrece una vista del sistema a modelar. Para poder representar correctamente un sistema, UML ofrece una amplia variedad de diagramas para visualizar el sistema desde varias perspectivas. UML incluye los siguientes diagramas:

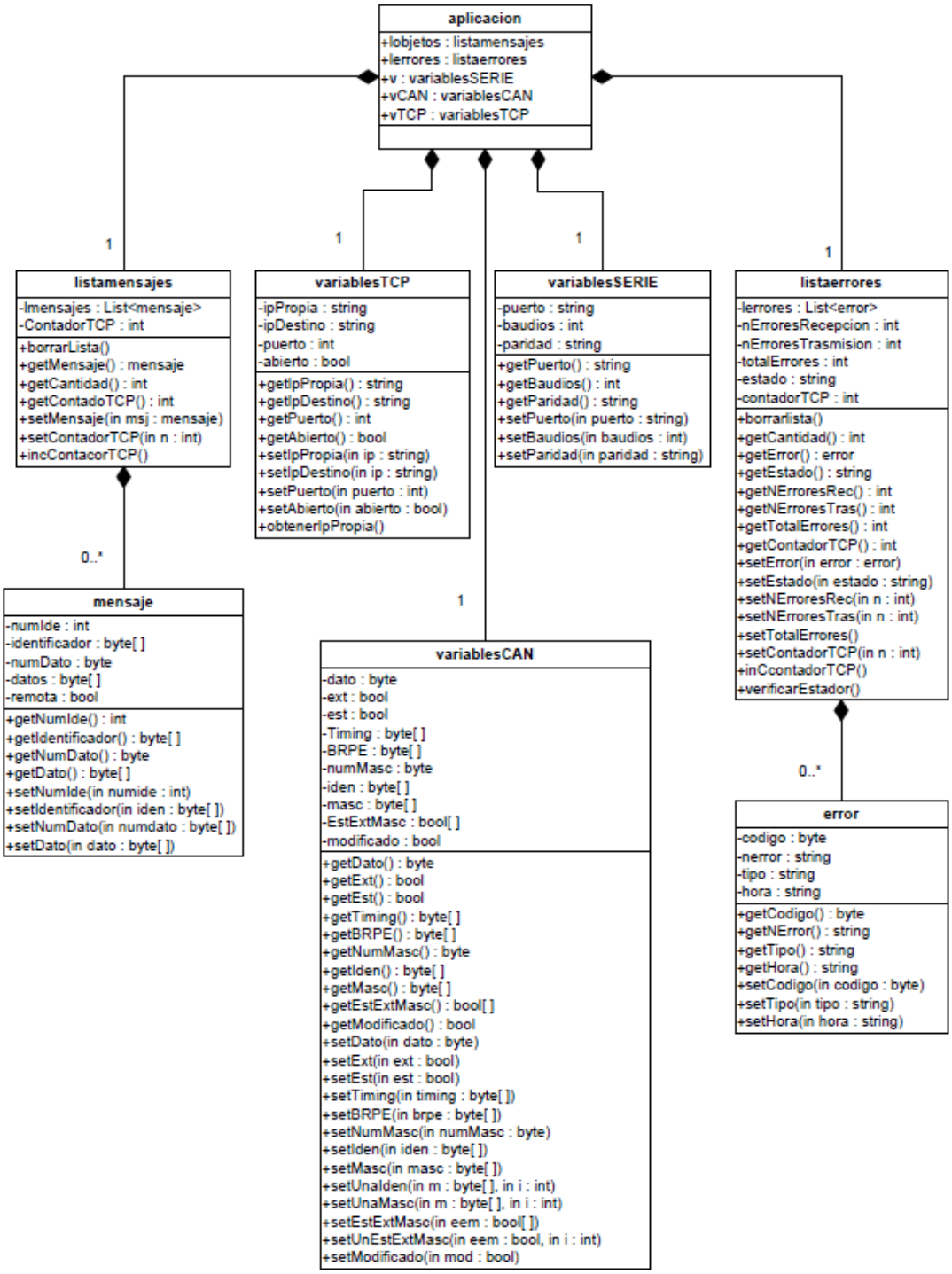
- Diagrama de casos de uso.
- Diagrama de clases.
- Diagrama de objetos.
- Diagrama de secuencia.

En este caso como solo vamos a usar el diagrama de clases, es el único que se explica.

El diagrama de clases muestra un conjunto de clases, interfaces y sus relaciones. Éste es el diagrama más común a la hora de describir el diseño de los sistemas orientados a objetos. En la figura 4 se muestran las clases globales, sus atributos y las relaciones de una posible solución al problema de la venta de entradas.

5.3.3. Estructura Interna

En este apartado se muestra un esquema UML de cómo están organizadas las clases internamente. De esta manera podemos ver como se pasan, modifican, insertan o eliminan datos.



Basándonos en el esquema mostrado, vamos a realizar una escueta descripción de cada una de las clases que constituyen la estructura interna del programa.

- **aplicacion:** Esta es la clase raíz, la que se pasara entre ventanas. A partir de aquí se inicializan el resto de clases y es de donde se pueden manejar.

- **listaerrores:** Es una clase que contiene una lista de objetos de tipo error, además de otras variables o atributos que son necesarios para manejar los errores.

- **error:** Es un objeto que contiene todos los atributos de un error, como el código, tipo y la hora.

- **listamensajes:** Es una clase que contiene una lista de objetos de tipo mensaje, además de otras variables o atributos que son necesarios para manejar los mensajes.

- **mensaje:** es un objeto de tipo mensaje que contiene todos los atributos de un mensaje, como el identificador, los datos o si es remota.

- **variablesSERIE:** Es un objeto en el cual sus atributos son los datos necesarios para la configuración del puerto serie, como el nombre del puerto, los baudios y la paridad.

- **variablesCAN:** Es un objeto en el cual sus atributos son los datos necesarios para la configuración del bus CAN, como el tipo de identificador, modo de funcionamiento, mascaras, etc.

- **variablesTCP:** Es un objeto en el cual sus atributos son los datos necesarios para la configuración de la conexión TCP, como la IP y el puerto.

La clase aplicación es la única que tiene sus atributos públicos, con lo que se puede acceder a ellos directamente sin necesidad de ningún método. Algo que agiliza bastante la programación.

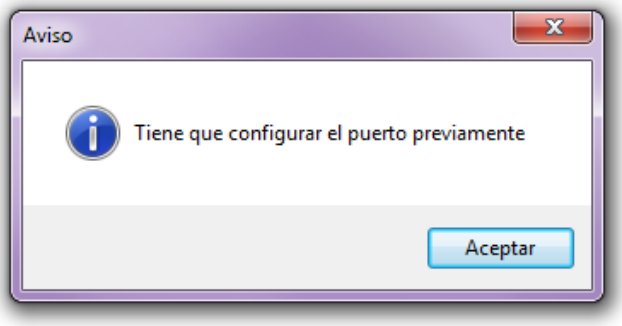
El resto de clases contiene todos sus atributos privados, es decir, que solo se puede acceder a ellos a través de sus respectivos métodos. Se a realizado de esta manera para asegurar una protección adecuada de los atributos.

5.4. Mensajes de error

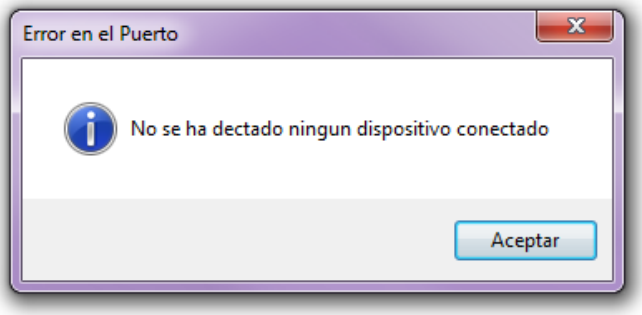
En este apartado están ordenados los mensajes de error según las acciones que se realizan.

Error al comunicarse con la tarjeta:

El primer mensaje que se muestra es el de que no se ha configurado el puerto series, algo indispensable para comunicarse con la tarjeta.

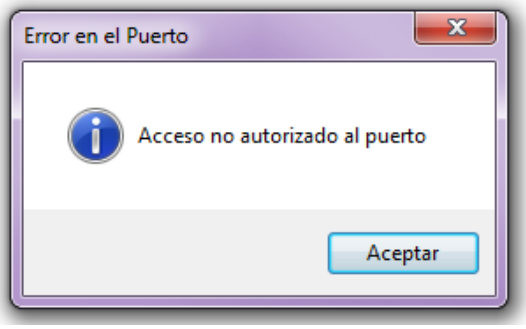


Los demás avisos o errores que pueden surgir al comunicarse con la tarjeta son:

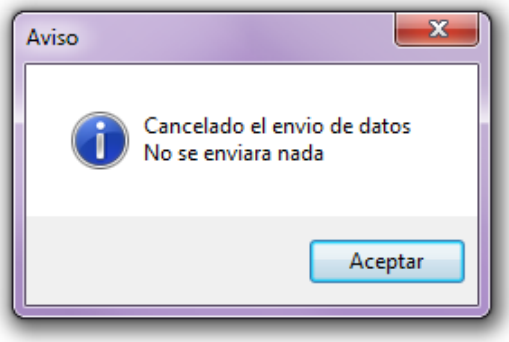


Que se muestra al pulsar **Abrir puerto** en la ventana de **Sniffer**, si no detecta ningún dispositivo.

Otro error que puede ocurrir es que el puerto este protegido o ya se esté usando, en este caso se no mostraría el siguiente mensaje:



Error al configurar el bus CAN:

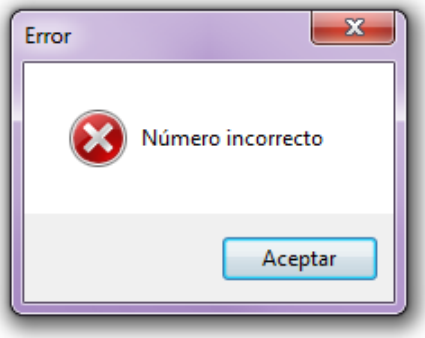


Este mensaje aparecerá cada vez que se haga una configuración incorrecta en la ventana de **Configuración CAN** e intentemos enviársela al Nodo CAN.

El envío se cancelara automáticamente, avisando de ello.

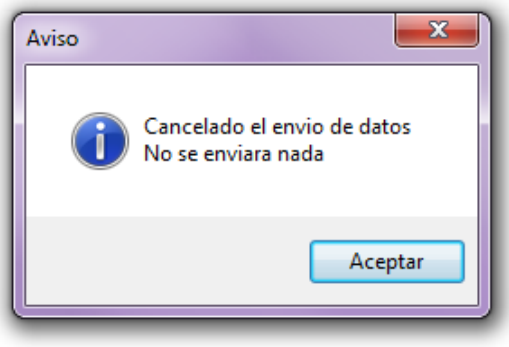
Error al enviar un mensaje:

Si rellenamos de forma incorrecta algún campo de envió de mensaje, nos mostrara el siguiente mensaje.



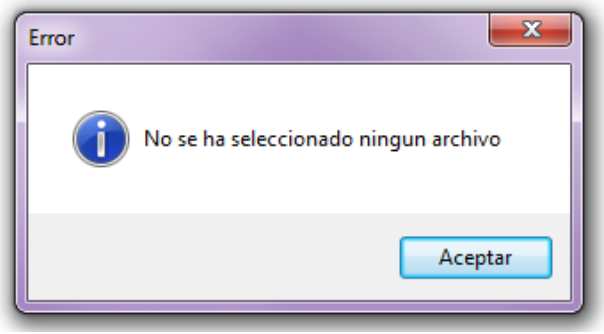
Esto sucede porque algún valor es incorrecto.

Además de que si estamos realizando el envío desde la ventana de **Configuración CAN** se nos mostrara también el siguiente mensaje.



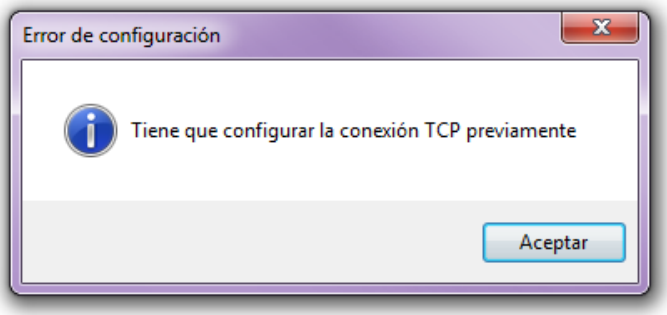
Error al importar desde PC:

Cuando intentamos importar un archivo, ya sea desde la ventana de **Sniffer** o desde la ventana de **Registro de errores**, si no seleccionamos una carpeta donde halla un fichero con el nombre "Sniffer.txt", nos mostrara el siguiente error.



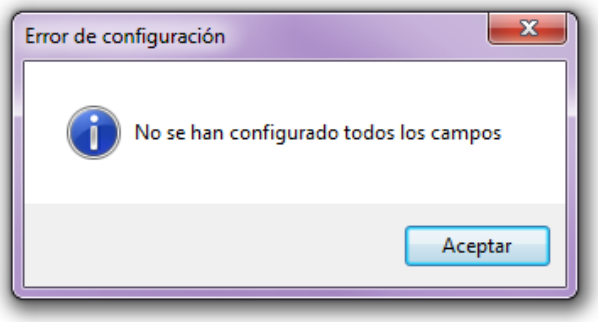
Error al comunicarse por TCP:

Cuando queremos importar o exportar los datos a través de TCP, debemos configurar la conexión TCP previamente. De no hacerlo e intentar cargar o descargar datos, se mostrara el siguiente mensaje.



Error al configurar la conexión TCP:

Aunque al abrir la ventana de **Configuración TCP**, ya aparezcan unos datos rellenado las respectivas casillas, no quiere decir que los tenga guardados. Estos datos que son la propia IP del PC y el puerto 13 por defecto; y solo son una ayuda, no quiere decir que sean los que ya están guardados. Para ello hay que pulsar aceptar o de lo contrario se nos mostrara el siguiente mensaje.



6. Herramientas usadas

En el presente apartado se muestra un listado con una breve explicación de los programas utilizados para la realización de este proyecto.

- **Microsoft Office Word 2007**
  Editor de texto utilizado para la redacción y edición del proyecto.

- **Hercules Setup Utility Versión 3.2.5**
  Herramienta utilizada para la visualización del puerto series.

- **Silicon Laboratories IDE Versión 4.40.00**
  Entorno de programación de C y drivers para el Microcontrolador C8051F500.

- **Microsoft Visual Studio 2010 Premium Versión 4.0.30319 RTMRel**
  Entorno de programación utilizado para crear la Interfaz Gráfica de Usuario en C#.

- **Config2 Versión 4.01**
  Herramienta de configuración del Microcontrolador C8051F500. De esta manera se puede configurar de forma gráfica algunos parámetros del microcontrolador.

- **Keil C51 Versión 9.06**
  Compilador de C ANSI para el Microcontrolador C8051F500.

7. Conclusión

Este proyecto, que comenzó a principios de a año, ha tenido una duración de ocho meses aproximadamente en los cuales se han encontrado varias dificultades ya solventadas.

Uno de los principales inconvenientes fue con la conexión serie en el PC y el microcontrolador. Llegados a este punto surgieron dos problemas, uno en el lado del PC y otro en el lado del microcontrolador.

El problema del PC era mantenerse siempre a la escucha del puerto series mientras se realizaban acciones con los datos recogidos. Esto podía llegar a provocar el desbordamiento en el buffer de entrada, si llegaban demasiados datos a la vez. Este problema fue solucionado gracias a las herramientas de programación concurrente que aporta el lenguaje usado.

Por otro lado, en la programación del microcontrolador surgió la dificultad de que la memoria de éste estaba separada en páginas, con lo cual surgían conflictos en el momento de leer determinadas variables en ciertos puntos del código.

La siguiente y última traba que se encontró fue el mantenimiento de la conexión TCP, ya que al intentar mantener la conexión activa bloqueaba el programa hasta el cierre de la conexión. Esto se pudo solucionar gracias a las técnicas de programación concurrente.

Finalmente se ha conseguido una herramienta muy útil de trabajo en el mundo de las redes industriales, especialmente como se ha explicado en el apartado “redes CAN”, en el mundo de la automoción. También gracias a la conexión TCP, se pueden transmitir los datos recibidos en tiempo real, lo cual es útil para la monitorización remota de las redes CAN.

8. Bibliografía

Periféricos e interfaces industriales.
José Carlos Campelo Rivadulla.
U. P. V. Facultad de Informática, 1997.

Diseño de un controlador industrial basado en el bus pc/104 para sistemas con red CAN.
Juan Antonio Donet Donet y Juan José Serrano Martín.
U. P. V. Facultad de Informática, 1999.

Sistema de monitorización y control de procesos basado en ethernet industrial.
Olimpo David Pérez Grau y Javier Sanchis Sáez.
U. P. V. Escuela Técnica Superior de Ingenieros Industriales, 2006.

Estudio de los buses utilizados en domótica.
José David García Del Rio, Angel Rodas Jordá y Vicente Montesinos Doménech.
U. P. V. Escuela Técnica Superior de Informática Aplicada, 2001.

Comunicaciones Industriales.
Pedro Morcillo Ruiz y Julian Cocera Rueda.
Paraninfo, 2000.

Programación de sistemas embebidos en C.
Gustavo Galeano.
Alfaomega, 2009.

Comunicaciones Industriales.
Aquilino Rodríguez Penin.
Marcombo, 2007.

www.can-cia.org/can/

www.can-cia.org/can/protocol/history/history.html

www.can-cia.org/applications/cankingdom/

www.serconet.com/usr/laureanog/HPV1B_31.HTM

www.semiconductors.bosch.de/de/20/can/index.asp

www.can.bosch.com

www.canbus.galeon.com/electronica/canbus.htm

www.ihs.com.es/

www.can-cia.org/products/pg2005/html/index.htm

www.canusb.com/index.htm

www.ixxat.de/

http://es.scribd.com/doc/14809760/REDES-INDUSTRIALES

http://neutron.ing.ucv.ve/revista-e/No4/RCI.html

http://es.kioskea.net/contents/internet/tcp.php3

http://webdiis.unizar.es/~joseluis/SE.pdf

www.uhu.es/raul.jimenez/EMPOTRADO/introduccion.pdf