UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escuela Técnica Superior de Ingeniería Informática

Universitat Politècnica de València

Gestión de un interfaz USB-CAN (Configuración)

Proyecto Final de Carrera

Ingeniería Técnica en Informática de Sistemas

Autor: Alberto Ramis Fuambuena

Director: José Carlos Campelo Rivadulla

Julio de 2012

Resumen

Las redes industriales tienen un gran ámbito de uso, y son muy importantes en la época actual. En la sociedad en la que vivimos, muchos procesos de producción o máquinas que nos facilitan la vida hacen uso de redes que han de ser robustas, versátiles y fiables. CAN es una de ellas.

Ante el incremento del número de dispositivos electrónicos en los automóviles, aumentó el cableado en ellos y su complejidad. Pronto se vio la necesidad de conectar todos los dispositivos a un bus que cumpliera con las características de una red industrial. Además, debía permitir altas velocidades de transmisión en ambientes difíciles por la temperatura, las vibraciones y otros problemas derivados del medio de uso. CAN encuentra, aparte de en el automóvil, una gran utilidad como bus de campo en diversas aplicaciones industriales

Para poder establecer como trabajan los distintos dispositivos de una red, nacieron las herramientas de análisis (monitores). Mediante un monitor podemos observar la cantidad de tráfico de una red, el uso que de esta hacen los dispositivos, y que tipo de información es. En redes CAN, como en cualquier otra red, nos surge esta necesidad, y en ello se centra este proyecto.

Para ello, se ha creado un driver instalado en un hardware específico que, junto con una aplicación para el PC, se encarga de monitorizar una red CAN, es decir, de recoger la información que por ella circula y mostrárnosla. En esta memoria, nos encargaremos de explicar la parte de configuración, envío y recepción de tramas.

Tabla de contenidos

1. Introducción

1.1 Redes Industriales

Una red industrial es una red usada para comunicar a los llamados dispositivos de campo para actuar directamente con el proceso productivo. Las comunicaciones a este nivel deben poseer unas características especiales y para responder a las necesidades de intercomunicación en tiempo real que se deben producir y ser capaces de resistir un ambiente hostil donde existe gran capacidad de ruido electromagnético y condiciones ambientales duras. En el uso de comunicaciones industriales se pueden separar dos áreas principales, una comunicación a nivel de campo, y una comunicación hacia el SCADA, En ambos casos la transmisión de datos se realiza en tiempo real, o por lo menos con una demora que no es significativa respecto de los tiempos de proceso, pudiendo ser crítico para el nivel de campo.

Según el entorno en el que van a ser utilizadas, en un ámbito industrial existen varios tipos de redes:

- **Red de factoría**: Volumen de intercambio de información muy alto, y donde los tiempos de respuesta nos son críticos.
- **Red de planta**: Sirve para interconectar módulos y células de fabricación entre sí. Suele emplearse para el enlace entre las funciones de ingeniería y planificación con los de control y producción en planta y secuenciamiento de operaciones.
- **Red de célula**: Para conectar dispositivos de fabricación que operan en modo secuencial como robots, máquinas de control numérico, etc. Deben gestionar mensajes cortos eficientemente, manejar tráfico de eventos discretos, tener mecanismos de control de error, posibilidad de transmitir mensajes prioritarios, bajo coste de instalación y de conexión por nodo, recuperación rápida ante eventos anormales en la red y alta fiabilidad.
- **Bus de campo**: Para suministrar cableado entre sensores-actuadores, y los correspondientes elementos de control. Este tipo de buses debe ser de bajo coste, tiempo real, permitir la transmisión serie sobre un bus digital de datos con capacidad de interconectar controladores de todo tipo de dispositivos de entrada-salida sencillos., y permitir controladores esclavos inteligentes. También deben de poseer todas las características de una red de célula. Por regla general, tienen un tamaño pequeño (de 5 a 50 nodos), utilizan tráfico de mensajes cortos para control y sincronización entre dispositivos, y la transferencia de archivos es ocasional o inexistente. Según la cantidad de datos a transmitir, se dividen en buses de alto nivel, buses de dispositivo (unos

pocos bytes a transmitir), como es nuestro bus, el CAN, y buses actuador/sensor (se transmiten datos a nivel de bit), pero en ningún caso llegan a transmitir grandes cantidades de datos.

De manera general, especialmente para los buses de campo y célula, las ventajas principales que se obtienen en su utilización son: mejor calidad y cantidad de flujo de datos, ahorro en el coste de cableado e instalación, facilidad en la ampliación o reducción del número de elementos del sistema, reducción de errores en la instalación y número de terminales y cajas de conexión.

1.1.1 Redes sensor-actuador: ASi

El bus ASi nació en 1990 como un intento de eliminar el cableado existente entre los sensores y actuadores binarios (todo-nada) con la característica añadida de proporcionar la tensión de alimentación sobre el mismo cable (hasta 8A). Posteriormente, el bus ha evolucionado para comunicarse con elementos inteligentes y poder transmitir datos y parámetros además de las señales binarias. El bus ASi es considerado uno de los sistemas de comunicación más sencillos y con menos prestaciones, por lo que se emplea a nivel de campo en la parte más baja de la pirámide de automatización.

1.1.2 Buses orientados a dispositivos

Este tipo de redes no dispones de una alta tasa de transferencia de datos las transmisiones se hacen por bloques de tamaño reducido (unos pocos bytes). Se consideran dentro de los buses de campo de bajo nivel, aunque algunos de ellos, como el caso del CAN, no fue concedido para el sector industrial, pero dada su robustez y bajo coste se encuentra bastante popularizado en el sector industrial.

1.1.3 Buses de campo

Este tipo de buses es el que ha diversificado su oferta de manera más amplia, dado que han aparecido numerosos estándares para su implementación industrial. A pesar de tratarse de estándares abiertos, cada protocolo suele estar impulsado pos un fabricante diferente, por lo que existe una pequeña batalla enmascarada por el control del mercado a través de la filosofía de sistemas abiertos. Entre los diferentes protocolos existen ciertas diferencias, pero generalmente es posible realizar el mismo tipo de aplicaciones sobre cualquiera de ellos.

1.2 **Sistemas empotrados**

Un sistema empotrado es un sistema informático que se encuentra físicamente incluido es un sistema de ingeniería más amplio al que supervisa o controla. Los sistemas empotrados se encuentran en multitud de aplicaciones, desde la electrónica hasta el control de complejos procesos industriales.

En el diseño de un sistema empotrado se suelen implicar ingenieros y técnicos especializados tanto en el diseño electrónico hardware como el diseño software. A su vez también se requerirá la colaboración de los especialistas en el segmento de usuarios de tales dispositivos

1.2.1 Hardware

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

- Microprocesador
- Microcontrolador de 4, 8, 16 o 32 bits.
- DPS de punto fijo o punto flotante.
- Diseño de 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, y que pueden ser el tamaño, margen de temperatura, consumo de energía, robustez, coste, etc.

1.2.2 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, como es nuestro caso, la capacidad de cálculo y dispositivos externos será limitado, y otros posibles problemas derivados de no tener un sistema más amplio. Podemos hablar de las siguientes necesidades:

- Trabajar en tiempo real.
- Optimizar al máximo los recursos disponibles.
- Disponer de un sistema de desarrollo específico para cada familia de microprocesadores empleados.
- Programación a bajo nivel, 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, como ANSI C.

El empleo de un sistema operativo determinado o el no empleo de este dependerán del sistema a desarrollar y es una de las principales decisiones que habrá que tomar en la fase de diseño del sistema empotrado. En nuestro sistema empotrado, por ejemplo, no es necesario el uso de un sistema operativo, mientras que en un micro de tipo ARM, PowerPC, etc. sí que lo llevará por lo general. La decisión, no obstante, dependerá de los requisitos del sistema, tanto técnicos como económicos.

1.2.3 Visión general

Podemos concluir que un sistema empotrado consiste en un sistema basado en microprocesador cuyo hardware y software están específicamente diseñados y optimizados para resolver un sistema concreto de forma eficiente. Normalmente un sistema empotrado interactúa continuamente con el entorno a 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. Bajo el concepto amplio de sistemas empotrados se da cabida a toda una serie de técnicas y metodologías de diseño tanto en hardware y software.

1.3 Herramientas de análisis de redes

1.3.1 Monitorización de redes

La característica principal de un sistema de monitoreo es recolectar información de forma ordenada de cada uno de los dispositivos o agentes pertenecientes a una red, de modo que se pueda detectar por medio de informes gráficos o escritos el comportamiento de cada nodo.

Estos programas tienen la finalidad de facilitarle al administrador de una red, mediante el uso de todo tipo de información visual, todas las características posibles a la hora de obtener información acerca de los dispositivos que intervienen en la red.

Vamos a repasar dos de los programas de monitorización más utilizados para uso en redes informáticas, como son Nagios y Wireshark, para entender un poco mejor de que se trata el análisis de una red. No profundizaremos en su uso, sino más bien nos centraremos en aquellas características que definen a un programa de este tipo. Así, podremos determinar qué tipos de servicios son los que se requieren. Toda la información está sacada de las páginas web de las aplicaciones, y por ello pueden dar un enfoque más publicitario que didáctico, pero de todas maneras y para nuestro propósito, nos será útil.

1.3.2 Herramientas de monitorización TCP/IP

1.3.2.1 Nagios

Nagios es un potente sistema de monitorización que identificar y resolver problemas en infraestructuras de tecnologías de la información, antes de que afecten de manera crítica a una red.

Mediante el uso de Nagios, se puede:

- Planificar actualizaciones de la infraestructura antes de que sistemas desactualizados causen fallos.
- Responder ante problemas al primer signo de fallo.
- Arreglar problemas automáticamente cuando se detecten.
- Coordinar respuestas del equipo técnico.
- Asegurar que ante cualquier interrupción de la infraestructura de red, esta tiene el mínimo impacto.
- Monitorizar una infraestructura completamente.

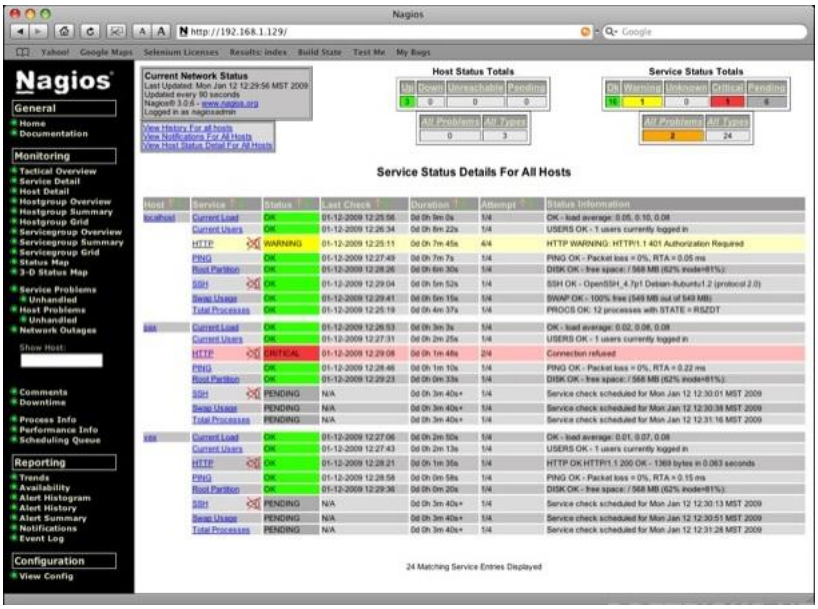


Fig. 1.1: Captura de Nagios

Funciona de la siguiente manera:

- Monitorización: El equipo de IT configura Nagios para monitorizar componentes críticos de la infraestructura IT, incluyendo medidas del sistema, protocolos de red, aplicaciones, servicios, servidores, e infraestructura.

- Alertar: Nagios envía alertas cuando componentes críticos del sistema fallan y se recuperan, proveyendo a los administradores la notificación de eventos importantes.

- Respuesta: El equipo IT puede reconocer alertas y comenzar a resolver cortes e investigar alertas de seguridad inmediatamente. Las alertas pueden ser escaladas a los diferentes grupos, si las alertas no se reconocen en el momento oportuno.

- Reportes: Los reportes proveen un registro histórico de cortes, eventos, notificaciones y respuesta a alertas para su posterior revisión.

- Mantenimiento: El tiempo de inactividad programado impide alertas durante el mantenimiento y actualización de ventanas.

1.3.2.2 Wireshark

Wireshark es un analizador de protocolos open-source diseñado por Gerald Combs y que actualmente está disponible para para plataformas Windows y Unix.

Conocido originalmente como Ethereal, su principal objetivo es el análisis de tráfico además ser una excelente aplicación didáctica para el estudio de las comunicaciones y para la resolución de problemas de red.

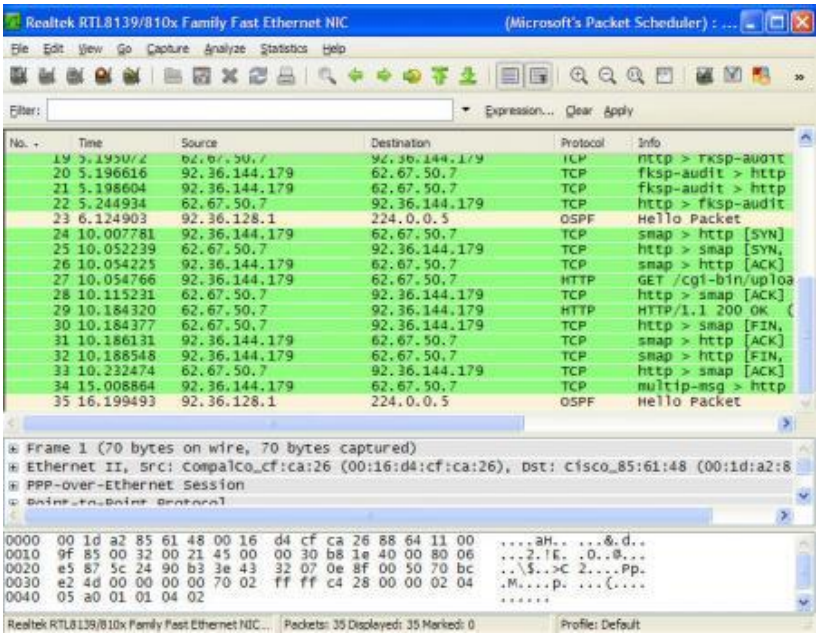


Fig. 1.2: Captura de Wireshark

Implementa una amplia gama de filtros que facilitan la definición de criterios de búsqueda para una cantidad ingente de protocolos (1100 en la versión 1.4.3); y todo ello por medio de una interfaz sencilla e intuitiva que permite desglosar por capas cada uno de los paquetes capturados. Gracias a que Wireshark 'entiende' la estructura de los protocolos, podemos visualizar los campos de cada una de las cabeceras y capas que componen los paquetes monitorizados, proporcionando un gran abanico de posibilidades al administrador de redes a la hora de abordar ciertas tareas en el análisis de tráfico.

1.3.3 Monitores propietarios

1.3.3.1 CAN AnalyzerAdvanced

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

Fig. 1.3: 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."

1.3.3.2 CAN BUS Analyzer Tool

Otro ejemplo es el **CAN BUS Analyzer Tool**, de la compañía Microchip, el cual, según el fabricante, "es un monitor para bus CAN de fácil uso y bajo precio mediante el cual se puede desarrollar y arreglar una red CAN de alta velocidad. La herramienta soporta CAN 2.0b y ISO11898-2 y un rango extenso de funciones que permiten usar varios segmentos de mercado cruzados, incluyendo automóvil, industria, medicina o marina. El kit de herramientas proporciona todo el hardware y el software requerido para conectar una red CAN al PC. La interfaz gráfica de usuario hace fácil la observación rápida del tráfico por el bus."

Fig. 1.4: CAN BUS Analyzer Tool

Según la página del fabricante, estas son sus características:

- Soporte para CAN 2.0b y ISO 11898-2
- Interfaz gráfica de usuario intuitiva para funciones tales como configuración, trazar, transmitir, filtros, log, etc.
- Características mejoradas en la IGU de PC para microcontroladores soportando cada la vista de registros ECAN en la IGU.
- Acceso directo tanto a señales CAN H y CAN L, como a CAN TX y CAN RX para depuración robusta.
- Opciones de interfaz bus CAN flexibles.
- Software de control de la resistencia de terminación y pantalla de LED de estado, error, tráfico…

1.3.4 Necesidad de monitorizar una red CAN

Las redes CAN, al igual que cualquier otro tipo de redes informáticas, están sujetas a errores en el envío o a una mala recepción por parte de los nodos que la compongan. Por ello, se plantea la necesidad de poder observar cual es el tráfico de la red, y saber si la información que circula por ella es la correcta.

Dada la naturaleza automovilística de este bus, esta necesidad se convierte en crítica, ya que un fallo en la comunicación de la red puede significar un problema grave de seguridad para las personas que están utilizando un vehículo el cual haga uso de esta tecnología.

Pero no solo los coches utilizan CAN, su robustez ha permitido que sean utilizados en redes industriales, por lo que también el funcionamiento sin testeo puede provocar un peligro para los operarios que hagan uso de máquinas en las cuales se utiliza.

Por ello, un programa de este tipo serviría para testear y perfeccionar el funcionamiento de los nodos de la red, y evitar el peligro que representarían para personas o posibles daños materiales.

1.4 Objetivos del PFC

Con todo lo anteriormente comentado, nuestro objetivo se centrará en realizar una herramienta de monitorización de redes CAN, red de uso principalmente industrial, mediante un driver programado específicamente para el microcontrolador, y una aplicación para el PC. En nuestro caso, el microcontrolador será uno de la marca Silabs, modelo C8051F500, uno de los más usados, que es un sistema empotrado. En el microcontrolador, se ejecutara el pequeño driver que realizará la escucha de todo el tráfico que circule por la red, y que podrá ser configurado para funcionar a distintas velocidades, en dos modos de funcionamiento, podrá establecer filtros de recepción de todo tipo y, además, podrá enviar tráfico por la red.

Para manejar todas estas configuraciones de manera lo más agradable y clara al posible usuario, en la aplicación del PC se podrá configurar todo lo que se implemente en el driver, además de una ventana en la que se puedan visualizar el tráfico y los errores de la red.

2. CAN (Controller Area Network)

Ante el incremento del número de dispositivos electrónicos en los automóviles, aumentaron las necesidades de cableado y su complejidad. Pronto se vio la posibilidad de conectar todos los dispositivos a un bus que debía ser fiable, robusto, con alta inmunidad al ruido, etc... Además, el bus debía permitir altas velocidades de transmisión en entornos difíciles por la temperatura, vibraciones y otros problemas derivados del medio de uso. Este bus encuentra, aparte de en el automóvil, una gran utilidad como bus de campo en diversas aplicaciones industriales.

Revisando su historia, el bus CAN, acrónimo de *Controller Area Network*, fue inventado por la compañía Robert Bosch en el año 1982. Inicialmente se pensó en el bus como en el bus de campo, pero donde realmente encontró utilidad fue en el sector del automóvil, para interconectar el bus de confort, seguridad... El primer coche en incorporarlo fue el Mercedes Clase E en 1992, 10 años después de patentarse. Fue diseñado para permitir la comunicación fiable entre centralitas electrónicas basadas en microprocesador, ECUs (*Electronic Control Unit*) y reducir cableado. En Europa se ha convertido en un estándar *de facto*, con carácter internacional y documentado por las normas ISO (ISO-11898). Su principal ventaja es la reducción de costes y la mejora de flexibilidad.

El bus CAN es un protocolo serie asíncrono del tipo CSMA/CD (*Carrier Sense Multiple Acces With Collision Detectio*n).Al ser CSMA, cada nodo de la red debe monitorizar el bus y si detecta que no hay actividad, puede enviar un mensaje. Al ser CD, si dos nodos de la red comienzan a transmitir a la vez un mensaje, ambos detectan la colisión. Un método arbitrario basado en prioridades resuelve el conflicto. Este bus en un medio compartido mediante multiplicación. Se trata de un protocolo "Multicast", es decir, todo el mundo puede hablar (de uno en uno) y escuchar.

Se utilizan un par de cables trenzados (bus diferencial) para conseguir alta inmunidad a las interferencias electromagnéticas, y en algunos casos va apantallado. La impedancia característica de la línea es del orden de 120 ohmios, por lo que se emplean impedancias (resistencias) de este valor en ambos extremos del bus para evitar ondas reflejadas y que el bus se convierta en una antena. Este bus tiene una longitud máxima de 1000m a 40 Kbps, con una velocidad máxima de 1Mbps a 40 metros, utilizándose en los coches a 125 Kb/s y a 500 Kb/s.

2.1 Capa física

El protocolo CAN define la capa de enlace de datos y parte de la capa física en el modelo OSI, consistente de siete capas. La ISO (*Internacional Standars Organization*, en español, Organización Internacional de Estándares) define un estándar, que incorpora las especificaciones CAN así como una parte de la capa física: la señalización física, comprendiendo la codificación y decodificación de bit (*No return Zero*, NRZ, en español, no retorno a cero), así como los tiempos de bit y sincronización.

2.1.1 Codificación de bit

En la codificación de bit escogida NRZ, el nivel de señal permanece constante durante el tiempo de bit y por lo tanto solo un tiempo es requerido para la representación de un bit (otro método de codificación es, por ejemplo Manchester).

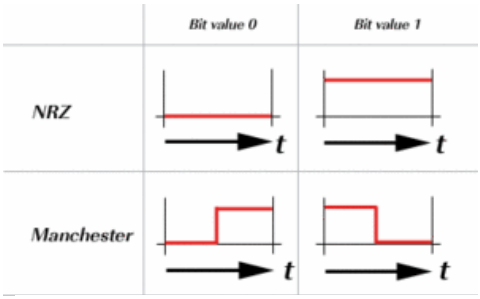


Fig. 2.1: Codificación NRZ vs. Manchester

El nivel de señal puede permanecer constante un periodo de tiempo largo; Por lo tanto, las medidas se deben tomar asegurándose que el intervalo máximo permisible entre dos límites de señal no se excede. Esto es importante para la sincronización. El bit *stuffing* (bit de relleno) se aplica para insertar un bit complementario después de cinco bits de igual valor. Por supuesto, el receptor no tiene en cuenta estos bits complementarios cuando procesa los datos.

2.1.2 Tiempo de bit y sincronización

En el nivel de bit (Capa 1 de OSI, capa física) CAN utiliza la sincronización para la transmisión de bit. Esto mejora la capacidad de transmisión pero también hace que se requiera un método sofisticado de sincronización de bit. Mientras que la sincronización de bit en una transmisión orientada al carácter

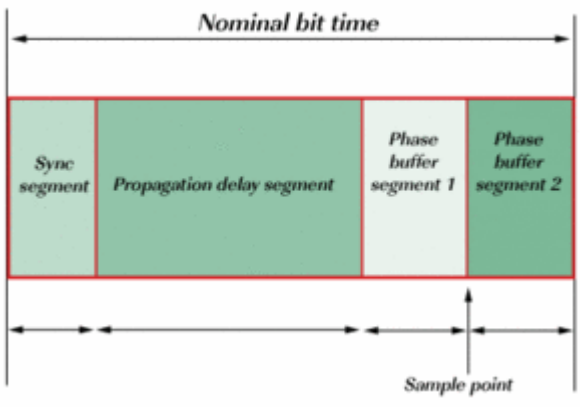


Fig. 2.2: Tiempo de bit nominal 1

(asíncrona) se re realiza sobre la recepción del bit de inicio disponible en cada carácter, un protocolo de transmisión síncrono tiene solo un bit de inicio disponible al principio del paquete. Para permitir que el receptor lea correctamente los mensajes, se debe hacer una continua resincronización. Segmentos de fase de amortiguamiento se insertan, por lo tanto, antes y después del punto de muestreo nominal dentro de un intervalo de bit.

El protocolo CAN regula el acceso el arbitraje *bit-wise (*bit "sabio"*)*. La señal se propaga desde el emisor al receptor y vuelve al receptor, debiendo de ser completada con un tiempo de bit. Para propósitos de sincronización, se complementa de un segmento de tiempo más, el segmento de retraso de la propagación, que es necesario añadir al tiempo reservado para la sincronización, el segmento de amortiguamiento de fase. El retraso de propagación se tiene en cuenta la propagación de la señal en el bus así como los retrasos de señal causados por los nodos de trasmisión y recepción.

Se pueden distinguir dos tipos de sincronización: la sincronización fuerte al principio del paquete y la resincronización con el paquete.

- Antes de una sincronización fuerte el tiempo de bit se reinicia al final del segmento de sincronización. Por lo tanto, el límite, que causó la sincronización fuerte, se encuentra dentro del segmento de sincronización del tiempo de bit reiniciado.
- La resincronización acorta o alarga el tiempo de bit asique el punto de muestra es desplazado de acuerdo al límite detectado.

El diseñador del dispositivo de programar los parámetros de tiempo de bit en el controlador CAN por medio de los registros correspondientes.

2.1.3 Interdependencia entre el ratio de datos y el tamaño del bus

Dependiendo del tamaño del segmento de retraso de propagación se puede determinar el máximo posible de tamaño de bus para un ratio específico (o el ratio de datos máximo posible para una longitud de bus determinada). La propagación de señal viene determinada por los dos nodos dentro del sistema que están más distantes el uno del otro. Este es el tiempo que tarda una señal a viajar desde un nodo a su nodo más lejano (teniendo en cuenta el retraso causado por los nodos de recepción y transmisión), sincronización y que la señal desde el segundo nodo vuelva al principio. Solo entonces el primer nodo sabrá cuál es su propio nivel de señal (recesivo en este caso) y su nivel actual en el bus o si ha sido reemplazado por el nivel dominante por otro nodo. Este hecho es importante para el arbitraje del bus.

Algunos transmisores modernos soportan tasas de datos no bajas. Sea como fuere, en la adquisición de los transmisores, la longitud máxima de la red es un dato a considerar.

2.1.4 Medios físicos

La base de la transmisión de los mensajes CAN y para competir por el acceso del bus es la capacidad de representar un valor de bit recesivo y dominante. Esto es posible por los medios eléctricos y ópticos hasta ahora. También la transmisión por la red eléctrica y la conexión inalámbrica son posibles. Para los medios eléctricos los voltajes de salida diferencial del bus se definen en ISO 11898-2 y ISO 11898-3, en SAE J2411, y ISO 11992.

Para medios ópticos el nivel recesivo es representado por la "oscuridad" y el dominante por la "luz". Los medios físicos más utilizados comúnmente para implementar redes CAN son un par diferencial de cables con retorno común. Para el cuerpo electrónico del vehículo se utiliza también un solo cable. Se han hecho algunos esfuerzos para desarrollar una solución para la transmisión de señales CAN en la misma línea de alimentación.

Los parámetros de los medios eléctricos se vuelven más importantes conforme la longitud del cable aumenta. La propagación de señal, la resistencia de la línea y las secciones en las que se cruzan cables son factores cuando dimensionamos una red. Para lograr la tasa de bits más alta posible a una determinada longitud, se requiere una señal de alta velocidad. Para líneas de bus grandes el voltaje cae más a mayor distancia. En las secciones con cruce de cables es necesario calcular la caída máxima permitida de voltaje del nivel de señal entre dos nodos lo más alejados en el sistema, y la resistencia final de salida de todos los receptores conectados. La caída de tensión permitida debe ser de tal manera que el nivel de señal puede ser interpretado de forma fiable en cualquier nodo receptor.

Tiempo de propagación de señal (ns/m)	Tasa de bits máxima (Kbits/s)
5.0	80
5.5	73
6.0	67
6.5	62
7.0	58

Tabla 2.1: Asumimos longitud de línea de 100 metros

2.1.5 Topología de red

Las señales eléctricas en el bus son reflejadas al final de la línea a menos que se hayan tomado medidas para evitarlo. Para el nodo que lee el nivel del bus correctamente es importante que las señales reflejadas del bus se eviten. Está bien que en la terminación de cada extremo del bus haya una resistencia de terminación para evitar líneas de relleno innecesarias en él. El factor más alto posible de tasa de transmisión y longitud de línea se consigue manteniendo lo más cerca posible una estructura de una única línea y por la terminación de ambos extremos de esta. Las recomendaciones específicas para esto se encuentran en los estándar (ISO 11898-2 y 3).

Se puede hacer uso para superar las limitaciones de la topología de línea básica repetidores, puentes o puertas de enlace. Un repetidor transfiere una señal eléctrica desde un segmento de bus físico a otro segmento. La señal solo es refrescada y el repetidor puede ser considerado como un componente pasivo comparable al cable. El repetidor divide la red en dos segmentos físicamente independientes. Esto causa un tiempo adicional de propagación de la señal. Por lo demás, se considera un solo segmento de red lógico.

Un puente conecta dos redes separadas lógicamente en la capa de enlace de datos (capa 2 OSI). Esto es para que los identificadores CAN puedan ser únicos en cada uno de los dos sistemas de buses. Los puentes implementan una función de almacenamiento y pueden reenviar mensajes o parte de ellos en una organización independiente con retardo de tiempo de transmisión. Los puentes se diferencian de los repetidores ya que pueden reenviar los mensajes a plazos, que no son locales, mientras que los repetidores lanzan todas las señales eléctricas incluyendo el identificador CAN.

Una puerta de enlace ofrece la conexión de redes con diferentes protocolos de capas altas. Por lo tanto, realiza la traducción de dates de protocolo entre dos sistemas de comunicación. Esta traducción se lleva a cabo en la capa de aplicación (capa 7 OSI).

2.1.6 Acceso al bus

La interfaz de conexión entre un controlador CAN y un par diferencial de cables, consiste básicamente de un amplificador de transmisión y un amplificador de recepción (transceptor = transmisión y recepción). Aparte de la adaptación de la representación de señal entre el chip y el bus, el transceptor tiene que cumplir una serie de requisitos adicionales. Debe cumplir que el transmisor proporciona la suficiente capacidad de salida del controlador y protege el controlador *on-chip* contra sobrecargas. Esto reduce radiaciones electromagnéticas. Y que el receptor del transceptor CAN ofrece un nivel

determinado de señal recesiva y protege el controlador de chip comparador de entrada contra sobretensiones en las líneas de bus. Además, amplía el rango de modo común del comparador de entrada en el controlador CAN y proporciona una sensibilidad de entrada suficiente. Además, detecta errores de bus, como rotura de cable, cortocircuitos, etc. Otra función del transceptor también puede ser el aislamiento galvánico de un nodo CAN y la línea.

2.2 Nivel de enlace CAN

El protocolo CAN es un estándar internacional definido en la ISO 11898. Además del propio protocolo, se define en la norma ISO 16845 el test de conformidad, que garantiza la intercambiabilidad de los chips CAN.

2.2.1 Principios del intercambio de datos

CAN se basa en el "mecanismo de comunicación multidifusión", que está basado en el protocolo de trasmisión orientado al mensaje. Se define el contenido del mensaje en lugar de los nodos y las direcciones de los nodos. Cada mensaje tiene un identificador, que es único en toda la red, ya que define el contenido y también la prioridad del mensaje. Esto es importante cuando varios nodos tienen que competir por el acceso al bus (arbitraje de bus).

Como resultado del direccionamiento orientado al contenido, se obtiene un alto grado de flexibilidad en la configuración del sistema. Es fácil añadir nodos a una red existente y se puede hacer sin necesidad de modificar ni el hardware ni el software de los nodos existentes, siempre y cuando los nodos añadidos sean puramente receptores. Esto permite un diseño modular, la recepción múltiple de datos y sincronizar procesos distribuidos. Además, la trasmisión de datos no se basa en la disponibilidad de tipos específicos de nodos, con lo que se simplifica y mejora la red.

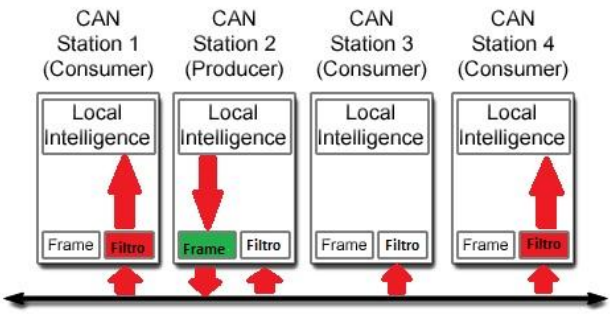


Fig. 2.3: Ejemplo de broadcast en CAN

2.2.2 Transmisión de datos en tiempo real

En el procesamiento en tiempo real la urgencia con la cual los mensajes deben de ser intercambiados a través de la red puede diferir mucho: una dimensión que cambia rápidamente, por ejemplo, carga de un motor, tiene que ser transmitida con mayor frecuencia y por lo tanto con menos retrasos que otras dimensiones, como pudiera ser la temperatura de un motor. En nuestro caso, el bus CAN hace el esfuerzo por conseguir un comportamiento en tiempo real, pero este no está garantizado.

La prioridad a la cual un mensaje se transmite comparado a otro mensaje menos urgente, se especifica mediante el identificador de cada mensaje. Las prioridades se establecen durante el diseño del sistema en forma de valores de correspondencia binaria y no se pueden cambiar dinámicamente. El identificador con el menor número binario tiene la prioridad más alta.

Los conflictos de acceso al bus se resuelven por el arbitraje de bits de los identificadores que participan en cada nodo observando el nivel bit a bit del bus. Esto ocurre de acuerdo con el mecanismo y el cable, mediante el cual el estado dominante sobrescribe el recesivo. Todos los nodos con transmisión recesiva y observación dominante pierden el acceso al bus. Los perdedores se convierten automáticamente en los receptores del mensaje con la más alta prioridad y no intenta la transmisión hasta que es bus está disponible de nuevo.

Las solicitudes de transmisión se manejan en orden de importancia para el sistema como un todo. Esto resulta especialmente ventajoso en situaciones de sobrecarga. Dado que en el acceso al bus se prioriza en base a los mensajes, es posible garantizar una baja latencia individual en sistemas de tiempo real.

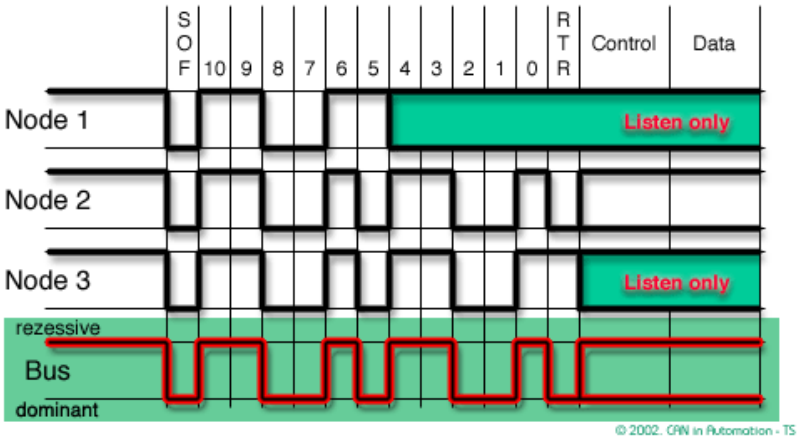


Fig. 2.4: Ejemplo de arbitraje

2.2.3 Formato de trama

El protocolo CAN soporta dos formatos de trama, con la única diferencia de la longitud de sus identificadores. La trama "básica" de CAN soporta una longitud de 11 bits para el identificador, y la trama "extendida" de CAN soporta una longitud de 29 bits para el identificador.

Fig. 2.5: Trama CAN de datos

2.2.4 Formato de trama básica

Un formato de trama básica empieza con un bit de inicio llamado "principio de trama", en inglés, "*Start of Frame* (SOF)", seguido del "campo de arbitraje" que consiste en el identificador y el bit de "Solicitud de transmisión remota", en inglés, "*Remote Transmission Request* (RTR)", utilizado para distinguir entre un trama de datos normal y una trama de petición de datos llamada trama remota. El siguiente "campo de control" contiene el "Extensión de identificador, o *IDentifier Extension* (IDE)", bit que distingue entre trama básica o extendida, seguido del "código de longitud de datos, o *Data Length Code* (DLC)", utilizado para indicar el número de bytes de datos en el "campo de datos". Si el mensaje es una trama remota, el campo DLC contendrá el número de bytes de datos solicitados. El "campo de datos" que sigue puede contener hasta 8 bytes. LA integridad de la trama está garantizada a través del siguiente campo, la suma de "comprobación de redundancia cíclica, o *Cyclic Redundant Check* (CRC)". El "campo de reconocimiento, *ACKnowdledge* (ACK)" comprende la ranura ACK y el delimitador ACK. El bit en la ranura ACK se envía como recesivo y se sobrescribe como dominante por dichos receptores, que tienen en este momento los datos recibidos correctamente. Los mensajes incorrectos son reconocidos por los receptores, independientemente del resultado de la prueba de aceptación. El final del mensaje se con el "fin de trama, *End Of Frame* (EOF)". El "espacio entre tramas, *Space Frame Intermission* (IFS)", es el número mínimo de bits que separan mensajes consecutivos. A menos que otra estación se ponga a transmitir, el bus permanece inactivo después de esto.

2.2.5 Formato de trama extendida

La diferencia entre una trama extendida y una trama básica es la longitud de identificador utilizado. El identificador de 29 bits se compone de 11 bits ("identificador básico") y una extensión de 18 bits ("extensión del

identificador"). La distinción entre el formato de trama extendida y la básica se realiza mediante el bit IDE, que se transmite como dominante en el caso de una trama de 11 bits, y se transmite como recesivo en una de 29. Como los formatos tienen que coexistir en el bus, se establece cuál es el de mayor prioridad, en el caso de colisión entre tramas de diferente formato y el mismo identificador básico / identificador. El mensaje de 11 bits siempre tiene prioridad sobre el de 29 bits.

El formato extendido tiene algunas ventajas y desventajas. El tiempo de latencia en el bus es más largo (en un mínimo de 20 tiempos de bit), los mensajes en formato extendido requieren más ancho de banda (aproximadamente un 20%), y el rendimiento de detección de error es menor (debido a que el polinomio elegido para el CRC de 15 bits esta optimizado para tramas de hasta 112 bits).

Los controladores CAN que soportan formatos de trama extendida, también son capaces de enviar y recibir tramas básicas. Los controladores CAN que solo admiten tramas básicas, no pueden reconocer tramas extendidas correctamente. Sin embargo, existen controladores CAN que solo soportan tramas básicas pero reconocen las los mensajes extendidos y los ignoran.

2.2.6 Detección y señalización de errores

A diferencia de otros sistemas de bus, el protocolo CAN no utiliza mensajes de reconocimiento, sino que señala los errores de inmediato a medida que ocurren. Para la detección de errores el protocolo CAN implementa tres mecanismos a nivel de mensaje (capa de enlace de datos, capa OSI 2):

- Comprobación de Redundancia Cíclica (CRC): EL CRC salvaguarda la información en la trama añadiendo una trama de comprobación de secuencia (FCS) al final de la transmisión. En el receptor este FCS se recalcula y prueba contra el FCS recibido. Si no coinciden, se ha producido un error CRC.
- Comprobación de trama: Este mecanismo verifica la estructura de la trama transmitida mediante la comprobación de los campos de bits contra el formato fijo y el tamaño de la trama. Los errores detectados por los controles de trama se denominan "errores de formato".
- Errores ACK: Los receptores de un mensaje responden las tramas recibidas con un ACK. Si el transmisor no recibe el ACK, ocurre un error de ACK.

El protocolo CAN también implementa dos mecanismos para la detección de errores a nivel de bit (capa física, capa OSI 1):

- Monitorización: La capacidad del transmisor para detectar errores se basa en la monitorización de señales del bus. Cada nodo que transmite observa el nivel del bus y así detecta diferencias entre el bit enviado y el bit recibido. Esto permite la detección segura de errores globales y errores locales en el transmisor.
- Bit de relleno (*Bit Stuffing*): La codificación de cada bit individual es probada a nivel de bit. La representación de bits utilizada en CAN es, como hemos comentado anteriormente, la NRZ. La sincronización de extremos se genera mediante el relleno de bits. Esto se hace después de que hayan cinco bits consecutivos, introduciendo un bit dentro del flujo. Su valor es el complementario al de los otros cinco., y es borrado por los receptores.

Si uno o más errores se descubren por lo menos en un nodo utilizando los métodos anteriores, la trasmisión de alimentación se interrumpe mediante una trama de error. Esto evita que otros nodos acepten el mensaje y así garantizar la coherencia de los datos en toda la red. Después de la transmisión de un mensaje erróneo que se ha anulado, el remitente de forma automática reintenta la transmisión (retransmisión automática). Los nodos pueden competir de nuevo por el bus.

Sin embargo, el eficaz y eficiente método explicado anteriormente puede ser, en caso de una estación defectuosa, que pudiera conducir a todos los mensajes (incluidos los correctos) a anularse. Si no se toman medidas para el autocontrol, el sistema será bloqueado por ello. El protocolo CAN, por ello, proporciona un mecanismo para distinguir los errores esporádicos de los errores permanentes y los fallos locales en la estación. Esto se hace mediante una evaluación estadística de situaciones de error con el fin de reconocer los errores propios de un nodo y, a poder ser, entrando en funcionamiento de una manera que no afecte al resto de los nodos. Esto puede continuar hasta que el nodo se desactive a sí mismo para evitar que los mensajes de los otros nodos erróneamente sean reconocidos como incorrectos.

2.3 Estandarización CAN

La especificación del protocolo CAN esta estandarizada por la Organización Internacional de la Estandarización (*International Organization of for Standarization,* ISO), con base en Génova (Suiza). El documento ISO 11898 está

bajo revisión y se publicará en distintas partes. La parte 1 definirá el protocolo , la parte 2 especificará la capa de velocidad de alta velocidad (no tolerante a fallos), y la parte 3 describe la capa física tolerante a fallos. La parte 4 está bajo desarrollo, y especifica el protocolo de comunicación TTCAN (*time-triggered CAN protocol*).

Además de estas especificaciones básicas, hay varias actividades internacionales en organismos de normalización oficiales y asociaciones sin ánimo de lucro con respecto a capas superiores para las redes CAN. Algunas de estas especificaciones son muy específicas de la aplicación, mientras que otros son más genéricos.

2.4 Especificación del protocolo CAN

La especificación CAN 2.0 describe el formato de la trama básica (con identificador de 11 bits) y el formato de trama extendida (con 29 bits de identificador).

A fin de distinguir la trama básica y la extendida, el bit IDE (extensión del identificador) en el campo de control de un mensaje CAN, se define en la especificación CAN 2.0 parte A. Esto se hace de tal manera que el formato de la trama en la especificación CAN 2.0 A es equivalente al formato de trama básico, y por lo tanto, sigue siendo válida. Además, el formato de trama extendida se ha definido para que los mensajes en formato básico y en formato extendido puedan coexistir en la misma red.

El formato de trama extendida tiene algunas ventajas y desventajas: el tiempo de latencia de bus es más largo (en un mínimo de 20 tiempos de bit), los mensajes en formato extendido requieren más ancho de banda (aproximadamente un 20%), y el rendimiento de detección de error es menor (debido a que el CRC de 15 bits está optimizado para una longitud de trama de aproximadamente 80 bits).

Esta especificación CAN 2.0 consta de dos partes, con:

- CAN 2.0 Parte A describe el formato de trama básico.
- CAN 2.0 Parte B, que describe las tramas básicas y las extendidas. Con el fin de ser compatible con la especificación 2.0, se requiere que una aplicación CAN sea compatible con cualquiera de las dos partes, A o B.

CAN 2.0 A -> http://www.can-cia.org/fileadmin/cia/specifications/CAN20A.pdf

CAN 2.0 B -> http://www.can-cia.org/fileadmin/cia/specifications/CAN20B.pdf

2.4.1 Guía de implementación del protocolo CAN

El documento de especificación del protocolo de CAN describe la función de la red en su conjunto. Además, Bosch ofrece un modelo de referencia CAN para los

licenciatarios, apoyando las implementaciones del protocolo dentro de los nodos de los licenciatarios CAN.

NOTA: La especificación CAN oficial ha sido liberada por la ISO como 11989-1 (Protocolo de la capa de enlace de datos). Hay dos capas físicas CAN estandarizadas por la ISO (11989-2: transmisión de CAN a alta velocidad, e ISO 11898-3: transmisión de CAN tolerante a fallos).

2.5 Certificación CAN

El plan de pruebas de conformidad de CAN se especifica en la norma ISO 16845. Este documento especifica la metodología y el conjunto de pruebas abstractas necesarias para comprobar el cumplimiento de cualquier implementación en la especificación CAN (ISO 11898-1). El entorno de plan de prueba consiste en un probador inferior, la implementación bajo prueba (*Implementation Under Test*, IUT), y el probador superior. Dado que la capacidad de almacenamiento de mensajes, el filtro de hardware de aceptación, y la interfaz de la CPU no están estandarizados, el único vínculo entre el probador inferior y el UIT son la PLS-Data.indicate y las interfaces PLS-DATA.REQUEST.

Los conjuntos de pruebas abstractos son independientes uno de otro. Cada suite de prueba verifica el comportamiento de la UIT de un parámetro en particular de la especificación CAN. Los casos de prueba se agrupan para los casos de tipos de trama recibidos, para los tipos de trama de transmisión, y para los casos de tramas bidireccionales. Cada tipo de prueba se divide en seis clases de prueba:

- Clase de formato de trama válido.
- Clase de detección de errores.
- Clase de manejo de errores activos de la trama.
- Clase de sobrecarga de trama.
- Clase de estado de error pasivo y *bus-off*.
- Clase de tiempo de bit.

Un laboratorio autorizado de pruebas oficiales, que certifique las implementaciones CAN, no existe.

3. C_CAN: Manual del usuario de Bosch

En esta sección de la memoria, explicaremos que hemos utilizado del manual de usuario de Bosch, Revisión 2.1, del 6 de junio de 2000. Se recomienda que ante cualquier duda que no esté resuelta en este apartado, se recurra a él. Este apartado se centrará en explicar cómo poner el modo silencioso, explicar los registros utilizados, y el mecanismo de transmisión y recepción de datos. Lo que se había llamado *trama **básica*** (trama con identificador de 11 bits) en el apartado anterior, en el manual C_CAN de Bosch se le llama *trama **estándar***, y es así como se le ha llamado en la aplicación. Todos los nombres en inglés han sido traducidos para mejor comprensión.

3.1 Visión general

C_CAN es un módulo CAN que puede ser integrado en un dispositivo autónomo o en una parte de un ASIC (circuito integrado para aplicaciones específicas). Está descrito en VHDL en nivel RTL. Consiste en los componentes núcleo CAN (*CAN Core*), RAM de mensajes (*Message RAM*), manejador de mensajes (Message Handler), registros de control (*Control Register*) e interfaz del módulo (*Module Interface*).

El núcleo CAN ejecuta las comunicaciones de acuerdo al protocolo CAN versión 2.0 parte A y parte B. La tasa de bits puede ser programada a valores por encima de 1Mbit/s dependiendo de la tecnología usada. Para la conexión a la capa física se requiere un transceptor adicional.

Para la comunicación a una red CAN, se tienen que configurar los objetos de mensaje individuales. Los objetos de mensaje y las máscaras de identificador para filtros de aceptación de mensajes recibidos son almacenados en la RAM de mensajes.

Todas las funciones concernientes al manejo de mensajes están implementadas en el manejador de mensajes. Estas funciones son los filtros de aceptación, la transferencia de mensajes entre el núcleo CAN y la RAM de mensajes, y el manejo de peticiones de transmisión así como la generación del módulo de interrupción.

El registro activo del C_CAN puede ser accedido directamente por una CPU vía la interfaz del módulo. Estos registros se utilizan para controlar y configurar el núcleo CAN y el manejador de mensajes y acceder a la RAM de mensajes.

El módulo interfaz entregado con el módulo C_CAN puede ser fácilmente reemplazado por un módulo interfaz adaptado a las necesidades del usuario.

C_CAN implementa las siguientes características:

Soporte de protocolo CAN versión 2.0 A y B.

- Tasa de bits por encima de 1Mbit/s.
- 32 objetos de mensaje
- Cada objeto de mensaje tiene su propia mascara de identificación.
- Modo FIFO programable (concatenación de objetos de mensaje).
- Interrupciones enmascarables.
- Modo “Desactivar Retransmisión Automática” para aplicaciones TTCAN.
- Modo loop-back programable y operación auto prueba.
- Módulo interfaz compatible Motorola HC08 de 8 bit sin multiplexar.
- Dos módulos interfaz de 16 bits al bus AMBA AMP para ARM.

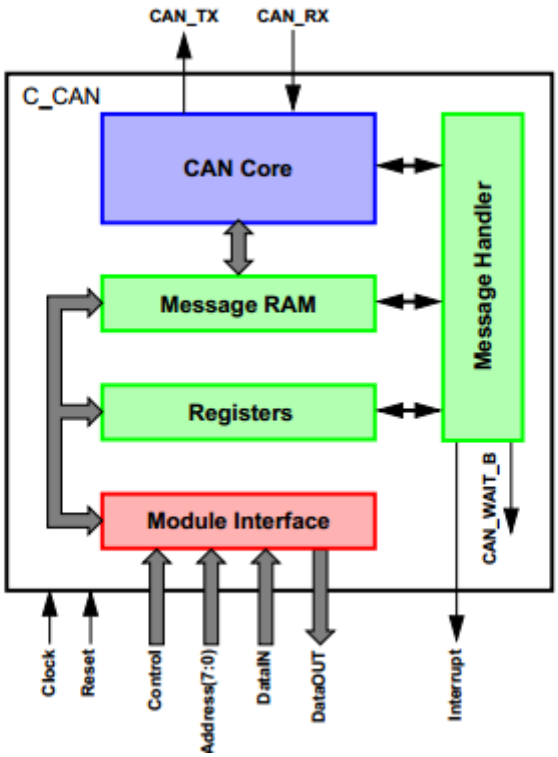


Fig. 3.1: Diagrama de bloques del C_CAN

3.2 Modos de operación: Modo Test

El modo test se introduce mediante el bit Test en el registro de control CAN a uno. En el modo test los bits Tx1, Tx0, LBack, Silent y Basic en el registro Test son modificables. El bit Rx monitoriza el estado del pin CAN_RX y por lo tanto, es solo de lectura. Todas las funciones del registro Test se desactivan cuando el bit Test se resetea a cero.

3.2.1 Modo silencioso

El núcleo CAN puede ser activado a modo silencioso programando en el registro Test el bit Silent a uno. En modo silencioso, el C_CAN se habilita para recibir tramas de datos válidas y tramas remotas válidas. Pero solo envía bits recesivos al bus CAN y no puede iniciar una transmisión. Si el núcleo CAN requiere enviar un bit dominante (bit de ACK, flag de sobrecarga, flag de error activo), el bit se enruta internamente para que el núcleo CAN monitoree este bit dominante, aunque el bus CAN puede mantenerse en modo recesivo. El modo silencioso puede usarse para analizar el tráfico de un bus CAN sin afectarse por la transmisión de bits dominantes (bits de ACK, tramas de error).

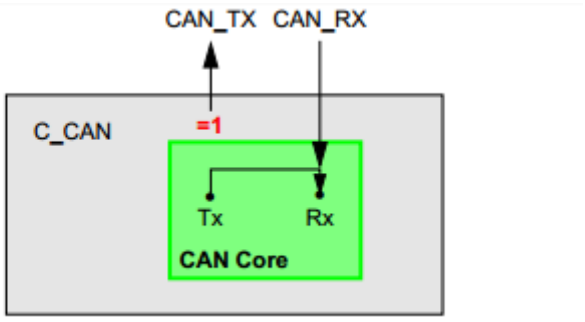


Fig. 3.2: Núcleo CAN en modo silencioso

Para nosotros este comportamiento es interesante debido a la naturaleza de la aplicación, ya que nos interesa que el nodo monitor sea absolutamente transparente para el resto, y así poder capturar el tráfico que existe en la red sin interponernos en las comunicaciones (si no hay nadie escuchando, por ejemplo, se notará que el nodo emisor no deja de mandar la trama).

3.3 Modelo de programador

El módulo C_CAN alberga un espacio de direccionamiento de 256 bytes. Los registros se organizan como registros de 16 bits, con el byte alto en la dirección par y el byte bajo en las impares.

Los dos conjuntos de registros de interfaz (IF1 e IF2) controlan el acceso de la CPU a la RAM de mensajes. Regulan los datos que deben transferirse a y desde la RAM, evitando conflictos entre los accesos a la CPU y la recepción y transmisión de mensajes.

Dirección	Nombre	Reset	Nombre Driver
CAN Base + 0x00	Registro de Control CAN	0x0001	CAN0CN
CAN Base + 0x02	Registro de estado	0x0000	CAN0STAT
CAN Base + 0x04	Contador de errores	0x0000	CAN0ERR
CAN Base + 0x06	Registro de *bit timing*	0x2301	CAN0BT
CAN Base + 0x08	Registro de interrup.	0x 0000	CAN0IID
CAN Base + 0x0A	Registro Test	0x 0000	CAN0TST
CAN Base + 0x0C	Registro de BRP Ext.	0x 0000	CAN0BRPE
CAN Base + 0x10	Orden de petición IF1	0x 0001	CAN0IF1CR
CAN Base + 0x12	Orden de mascara IF1	0x 0000	CAN0IF1CM
CAN Base + 0x14	Mascara 1 IF1	0x FFFF	CAN0IF1M1
CAN Base + 0x16	Mascara 2 IF1	0x FFFF	CAN0IF1M2
CAN Base + 0x18	Arbitraje 1 IF1	0x 0000	CAN0IF1A1
CAN Base + 0x1A	Arbitraje 2 IF1	0x 0000	CAN0IF1A2
CAN Base + 0x1C	Control de mensaje IF1	0x 0000	CAN0IF1MC
CAN Base + 0x1E	Datos A 1 IF1	0x 0000	CAN0IF1DA1
CAN Base + 0x20	Datos A 2 IF1	0x 0000	CAN0IF1DA2
CAN Base + 0x22	Datos B 1 IF1	0x 0000	CAN0IF1DB1
CAN Base + 0x24	Datos B 2 IF1	0x 0000	CAN0IF1DB2
CAN Base + 0x80	Petición de transmisión 1	0x 0000	CAN0TR1
CAN Base + 0x82	Petición de transmisión 2	0x 0000	CAN0TR2
CAN Base + 0x90	Datos nuevos 1	0x 0000	CAN0ND1
CAN Base + 0x92	Datos nuevos 2	0x 0000	CAN0ND2
CAN Base + 0xA0	Interrupción pendiente 1	0x 0000	CAN0IP1
CAN Base + 0xA2	Interrupción pendiente 2	0x 0000	CAN0IP2
CAN Base + 0xB0	Mensaje válido 1	0x 0000	CAN0MV1
CAN Base + 0xB2	Mensaje válido2	0x 0000	CAN0MV2

Tabla 3.1: Registros C_CAN (solo la interfaz 1)

3.3.1 Registros relacionados con el protocolo CAN

3.3.1.1 Registro de control CAN (CAN0CN)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
res	res	res	res	res	res	res	res	Test	CCE	DAR	res	EIE	SIE	IE	Init
r	r	r	r	r	r	r	r	rw	rw	rw	r	rw	rw	rw	rw

Test -> Modo test activado

- **Uno**: Modo Test.
- **Cero**: Funcionamiento normal.

CCE -> Activar cambio de configuración.

- **Uno**: La CPU tiene acceso para escribir en el registro *Bit-Timing*.
- **Cero**: La CPU no tiene acceso para escribir en el registro *Bit-Timing*.

DAR -> Retransmisión automática desactivada .

- **Uno**: Retransmisión automática desactivada.
- **Cero**: Retransmisión automática de mensajes perturbados activada.

EIE -> Activar interrupciones de error.

- **Uno**: Activado – Un cambio en los bits BOff o EWarn en el registro de estado generará una interrupción.
- **Cero**: Desactivado – No se generarán interrupciones de error.

SIE -> Activar interrupciones de cambio de estado.

- **Uno**: Activado – Una interrupción será generada cuando un transferencia sea realizada con éxito, o se detecte un error en el bus CAN.
- **Cero**: Desactivado – No se generarán interrupciones de cambio de estado.

IE -> Activar interrupciones de módulo.

- **Uno**: Activado – Interrupciones activaran IRQ_B a bajo. IRQ_B permanecerá bajo hasta que todas las interrupciones pendientes sean procesadas.
- **Cero**: Desactivado – La interrupción de módulo IRQ_B siempre está en alto.

Init -> Inicialización

- **Uno**: Activado – Comienza la inicialización.
- **Cero**: Desactivado – Funcionamiento normal.

3.3.1.2 Registro de estado (CAN0STAT)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
res	res	res	res	res	res	res	res	BOff	EWarn	EPass	RxOk	TxOk	LEC		
r	r	r	r	r	r	r	r	r	r	r	rw	rw	rw		

Boff -> Estado de Busoff

- **Uno**: El módulo CAN está en modo busoff.
- **Cero**: El módulo CAN no está en modo busoff.

EWarn -> Estado de alerta

- **Uno**: El último de los errores del contador en el EML ha superado el límite de errores de alerta de 96.
- **Cero**: Ambos contadores de error está por debajo del límite de alarma de errores de 96.

EPass -> Error pasivo

- **Uno**: El núcleo CAN está en el estado de error pasivo definido en la especificación de CAN.
- **Cero**: El núcleo CAN está en error activo.

RxOk -> Recibir un mensaje correctamente.

- **Uno**: Desde que este bit fue reseteado la última vez (a cero) por la CPU, un mensaje fue correctamente recibido (independientemente de los filtros de aceptación).
- **Cero**: Desde que este bit fue reseteado la última vez por la CPU, no se han recibido correctamente mensajes. Este bit nunca se resetea por el núcleo CAN.

TxOk -> Transmitir un mensaje correctamente.

- **Uno**: Desde que este bit fue reseteado la última vez (a cero) por la CPU, un mensaje fue correctamente transmitido (libre de errores y con el ACK final de otro nodo).
- **Cero**: Desde que este bit fue reseteado la última vez por la CPU, no se han transmitido correctamente mensajes. Este bit nunca se resetea por el núcleo CAN.

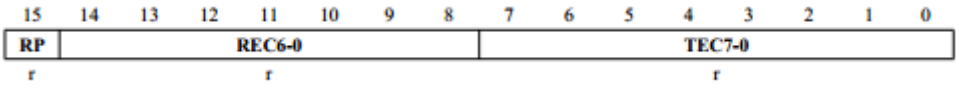
LEC -> Último código de error (Tipo del último error ocurrido en el bus CAN)

- **0**: Sin error
- **1**: Error *Stuff*: Más de cinco bits iguales en una secuencia dentro de un mensaje recibido, cuando esto no está permitido.
- **2**: Error de forma: Una parte de un formato fijo de un paquete recibido tiene un formato incorrecto.
- **3**: Error ACK: El mensaje que este núcleo CAN ha enviado no recibe el ACK de otro nodo.
- **4**: Error de bit1: Durante la transmisión de un mensaje (con la excepción del campo de arbitraje), el dispositivo quiere enviar un bit recesivo (bit de valor lógico a uno), pero el valor monitorizado en el bus era dominante.
- **5**: Error de bit0: Durante la transmisión de un mensaje (o el bit de ACK, o el flag de error activo, o el flag de sobrecarga), el dispositivo quiere enviar un bit dominante (en bits de datos o identificador tiene valor lógico 0), pero el bus monitoriza que el valor estaba en recesivo. Durante el busoff recupera este estatus se activa cada vez que la secuencia de 11 bits recesivos ha sido monitorizada. Esto habilita a la CPU a monitorizar la acción de la secuencia de recuperación de busoff (lo que indica que el bus no se ha quedado atascado o alterado de forma continua)

- **6**: Error CRC: La suma de CRC fue incorrecta en la recepción del mensaje, el CRC recibido para un mensaje entrante no coincide con el CRC calculado para los datos recibidos.
- **7**: Sin uso: Cuando el LEC muestra un 7, no ha habido evento en el bus detectado desde que la CPU escribió este valor en el LEC.

El campo LEC se limpia a 0 cuando un mensaje que ha sido transferido, recibido o enviado, sin error. El valor 7 lo puede escribir la CPU para comprobar actualizaciones.

3.3.1.3 Contador de errores (CAN0ERR)



RP -> Error Pasivo de recepción.

- **Uno**: El contador de errores de recepción ha alcanzado el nivel de error pasivo, definido en la especificación CAN.
- **Cero**: El Contador de errores de recepción está por debajo del nivel de error pasivo.

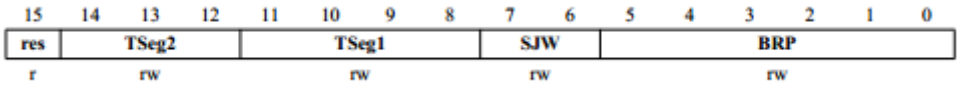
REC6-0 -> Contador de errores de recepción.

El estado actual del contador de errores de recepción. Su valor está entre 0 y 127.

TEC6-0 -> Contador de errores de transmisión.

El estado actual del contador de errores de transmisión. Su valor está entre 0 y 255.

3.3.1.4 Registro de bit Timing (CAN0BT)



TSeg2 -> El segmento de tiempo antes del punto de muestra.

0x00-0x07: Los valores válidos para TSeg2 son entre 0 y 7. La interpretación actual por el hardware de este valor es tal que se usa uno más que el valor programado.

SJW -> (Re) sincronización del ancho de salto

0x00-0x03: Los valores válidos para SJW son entre 0 y 3. La interpretación actual por el hardware de este valor es tal que se usa uno más que el valor programado.

BRP -> Tasa de baudios pre-escalada.

0x01-0x3F: El valor por el cual la frecuencia del oscilador se divide para generar el quantum del tiempo de bit. El tiempo de bit se construye desde un múltiplo de este quantum. Los valores válidos para la tasa de baudios pre-escalada son de 0 a 63. La interpretación actual por el hardware de este valor es tal que se usa uno más que el valor programado.

Con un módulo reloj CAN_CLK de 8Mhz, el valor de reinicio de 0x2301 configura el C_CAN para una tasa de bits de 500 Kbit/s. Los registros son solo modificables si el bit CCE e Init en el Registro de control CAN están activados.

3.3.1.5 Registro Test (CAN0TST)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
res	res	res	res	res	res	res	res	Rx	Tx1	Tx0	LBack	Silent	Basic	res	res
r	r	r	r	r	r	r	r	r	rw	rw	rw	rw	rw	r	r

Rx -> Monitoriza el valor actual del pin CAN_RX.

- **Uno**: El bus está en recesivo (CAN_RX=1).
- **Cero**: El bus está en dominante (CAN_RX=0).

Tx 1-0-> Controla el pin CAN_TX

- **00**: Valor de reinicio, CAN_TX se controla por el núcleo CAN
- **01**: Punto de muestreo puede ser monitorizado por el pin CAN_TX.
- **10**: CAN_TX maneja un valor dominante (0).
- **11**: CAN_TX maneja un valor dominante (1).

LBack -> Modo Loop Back

- **Uno**: Modo Loop back activo
- **Cero**: Modo Loop Back desactivado

Silent -> Modo Silencioso.

- **Uno**: El módulo está en modo silencioso.
- **Cero**: Funcionamiento normal

Basic -> Modo básico

- **Uno**: Los registros IF1 se usan como buffer de transmisión, los registros IF2 como buffer de recepción.
- **Cero**: Modo básico desactivado

El acceso al registro test está habilitado activando el bit Test en el registro de control CAN. Las diferentes funciones del registro pueden ser combinadas, pero Tx1-0 distinto del valor '00' corromperá la transferencia de mensajes.

3.3.1.6 Registro de Extensión de BRP (CAN0BRPE)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
res	res	res	res	res	res	res	res	res	res	res	res	BRPE			
r	r	r	r	r	r	r	r	r	r	r	r	rw			

BRPE -> Extensión del preescalador de la tasa de baudios

0x00-0x0F: Por programación del BRPE, el preescalador de la tasa de baudios se puede extender a valores por encima de 1023. LA interpretación actual por parte del hardware es que se toma un valor más que el valor programado en el BRPE (bits más significativos) y el BRP (bits menos significativos).

3.3.2 Set de registros de la interfaz de mensajes

3.3.2.1 Registros de orden de petición IFx (CAN0IFxCR)

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IF1 Command Request Register (addresses 0x11 & 0x10)	Busy	res	res	res	res	res	res	res	res	res	Message Number					
IF2 Command Request Register (addresses 0x41 & 0x40)	Busy	res	res	res	res	res	res	res	res	res	Message Number					
	r	r	r	r	r	r	r	r	r	r	rw					

Un mensaje se transfiere tan pronto como la CPU escribe un número de mensaje en el registro de orden de petición. Con esta operación de escritura, el bit Busy (ocupado) se activa automáticamente a 1 y la señal CAN_WAIT_B se pone a BAJO (LOW) para notificar a la CPU que hay una transferencia en proceso. Después de un tiempo de 3 o 6 periodos de CAN_CLK, la transferencia entre el Registro de interfaz y la RAM de mensajes se completa. El bit Busy vuelve a ponerse a 0 y CAN_WAIT_B vuelve a ALTO (HIGH). Los valores no válidos puestos en Message Number se cambian automáticamente a valores válidos.

Busy -> Flag de ocupado.

- **Uno**: Activo a uno cuando se escribe en el registro de orden de petición
- **Cero**: Se resetea a 0, cuando la operación de lectura/escritura termina.

Message Number -> Número de mensaje

0x01-0x20: Número de mensaje válido, el objeto de mensaje en la memoria RAM de mensajes se selecciona para la transferencia de datos.

0x00: No es un número de mensaje válido, interpretado como 0x20 (32_{10}).

0x21-0x3F: Número de mensaje no válido, interpretado como 0x01-0x1F.

3.3.2.2 Registro de orden de máscara. (CAN0IFxCM)

	15 14 13 12 11 10 9 8	7	6	5	4	3	2	1	0
IF2 Command Mask Register (addresses 0x13 & 0x12)	res	WR/RD	Mask	Arb	Control	ClrIntPnd	TxRqst/ NewDat	Data A	Data B
IF2 Command Mask Register (addresses 0x43 & 0x42)	res	WR/RD	Mask	Arb	Control	ClrIntPnd	TxRqst/ NewDat	Data A	Data B
	r r r r r r r r	rw	rw	rw	rw	rw	rw	rw	rw

Los bits de control de este registro especifican la dirección de transferencia, y seleccionan que registros de buffer de mensajes IF serán la fuente o el objetivo de los datos a transferir.

WR/RD -> Escritura/Lectura.

- **Uno**: Write, transfiere los datos desde el registro de buffer de mensajes al objeto de mensaje seleccionado en el registro de orden de petición.
- **Cero**: Read, transfiere los datos desde el objeto de mensaje seleccionado en el registro de orden de petición al registro de buffer de mensajes seleccionado.

Dirección = Escritura (Write)

Mask -> Acceso a bits de máscara.

- **Uno**: Transferir Identidier Mask + MDir + MXtd al objeto de mensaje.
- **Cero**: No cambian los bits de la máscara.

Arb -> Acceso a bits de arbitraje.

- **Uno**: Transmitir Identifier + Dir + XTD + MsgVal al objeto de mensaje.
- **Cero**: No cambian los bits de arbitraje.

ClrIntPnd -> Limpiar bit de interrupción pendiente.

Nota: Cuando se escribe a un objeto de mensaje, se ignora este bit.

TxRqst/NewDat -> Acceso al bit de petición de transmisión

- **Uno**: activa el bit TxRqst.
- **Cero**: No cambian el bit TxRqst.

Nota: Si una transmisión es reclamada por la programación del bit TxRqst/NewDat en el registro de orden de máscara IFx, el bit TxRqst en el registro de control de mensaje se ignorará.

Data A -> Acceso a los bytes de datos 0-3.

- **Uno**: Transfiere los bytes de datos 0-3 al objeto de mensaje.
- **Cero**: No cambian los bytes de mensaje 0-3.

Data B -> Acceso a los bytes de datos 4-7.

- **Uno**: Transfiere los bytes de datos 4-7 al objeto de mensaje.
- **Cero**: No cambian los bytes de mensaje 4-7.

Dirección = Lectura (Read)

Mask -> Acceso a bits de máscara.

- **Uno**: Transferir Identidier Mask + MDir + MXtd al registro de buffer de mensajes.
- **Cero**: No cambian los bits de la máscara.

Arb -> Acceso a bits de arbitraje.

- **Uno**: Transmitir Identifier + Dir + XTD + MsgVal al registro de buffer de mensajes.
- **Cero**: No cambian los bits de arbitraje.

ClrIntPnd -> Limpiar bit de interrupción pendiente.

- **Uno**: Limpia el bit IntPnd en el objeto de mensaje.
- **Cero**: El bit IntPnd permanece sin cambiar.

TxRqst/NewDat -> Acceso al bit de petición de transmisión

- **Uno**: impia el bit NewDat en el objeto de mensaje.
- **Cero**: El bit NewDat permanece sin cambios.

Nota: Un acceso de lectura al objeto de mensaje puede combinarse con el reinicio de los bits de control IntPnd y NewDat. Los valores de estos bits se

transfieren al registro de control de mensaje siempre reflejando el estado antes de reiniciar estos bits.

Data A -> Acceso a los bytes de datos 0-3.

- **Uno**: Transfiere los bytes de datos 0-3 al registro de buffer de mensajes.
- **Cero**: No cambian los bytes de mensaje 0-3.

Data B -> Acceso a los bytes de datos 4-7.

- **Uno**: Transfiere los bytes de datos 4-7 al registro de buffer de mensajes.
- **Cero**: No cambian los bytes de mensaje 4-7.

3.3.3 Registros del buffer de mensajes IFx

Los bits de los registros de buffer de mensajes se copian a los objetos de mensaje en la RAM de mensajes.

3.3.3.1 Registro de mascara IFx (CAN0IFxMx)

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IF1 Mask 1 Register (addresses 0x15 & 0x14)	Msk15-0															
IF1 Mask 2 Register (addresses 0x17 & 0x16)	MXtd	MDir	res	Msk28-16												
IF2 Mask 1 Register (addresses 0x45 & 0x44)	Msk15-0															
IF2 Mask 2 Register (addresses 0x47 & 0x46)	MXtd	MDir	res	Msk28-16												
	rw	rw	r	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Msk28-0 -> Mascara de identificación

- **Uno**: Los bits correspondientes a los identificadores se utilizan para el filtro de aceptación.
- **Cero**: Los bits correspondientes en el identificador del objeto de mensaje no se utilizarán en el filtro de aceptación.

MXtd -> Mascara de identificador extendido.

- **Uno**: El bit de identificador extendido (IDE) se utiliza para el filtro de aceptación.
- **Cero**: El bit de identificador extendido (IDE) no se utiliza para el filtro de aceptación.

Nota: Cuando los identificadores de 11 bits (“estándar”) se utilizan para un objeto de mensaje, los identificadores de una trama de datos recibida se

escribe entre los bits ID28 e ID 18. Para el filtro de aceptación, solo los bits entre MSK28 y MSK 18 son los que se tienen en cuenta.

Mdir -> Mascara de dirección de mensaje.

- **Uno**: El bit de dirección del mensaje (Dir) se utilizan para el filtro de aceptación.
- **Cero**: El bit de dirección del mensaje (Dir) no se utilizarán en el filtro de aceptación.

3.3.3.2 Registros de arbitraje IFx (CAN0IFxAx)

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IF1 Arbitration 1 Register (addresses 0x19 & 0x18)	ID5-0															
IF1 Arbitration 2 Register (addresses 0x1B & 0x1A)	MsgVal	Xtd	Dir	ID28-16												
IF2 Arbitration 1 Register (addresses 0x49 & 0x48)	ID15-0															
IF2 Arbitration 2 Register (addresses 0x4B & 0x4A)	MsgVal	Xtd	Dir	ID28-16												
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

ID28-0 -> Identificación de mensaje.

- **ID28-0**: Identificador de 29 bits (“Trama extendida”).
- **ID28-18**: Identificador de 11 bits (“Trama estándar”).

MsgVal -> Mensaje válido

- **Uno**: Este objeto de mensaje se ha configurado y deberá ser considerado por el manejador de mensajes.
- **Cero**: El objeto de mensaje se ignorará por el manejador de mensajes.

Nota: La CPU debe resetear el bit MsgVal para todos los mensajes que no se utilizen durante la inicialización antes de reiniciar el bit Init en el registro de control CAN.Este bit debe ser reiniciado también antes de modificar id28-0, los bits de control Xtd,Dir o el código de longitud de los datos DLC3-0, o si la longitud de los mensajes no se necesita.

Xtd -> Identificador extendido.

- **Uno**: El identificador de 29 bits (“extendido”) se utilizará para este objeto de mensaje.
- **Cero**: El identificador de 11 bits (“estándar”) se utilizará para este objeto de mensaje.

Dir -> Mascara de dirección de mensaje.

- **Uno**: Dirección = transmitir. En TxRqst, el objeto de mensaje se transmite como trama de datos. En la recepción de una trama remota con identificador coincidente, el bit TxRqst de un objeto de mensaje está activado (si RmtEn = uno).
- **Cero**: Dirección = recibir. En TxRqst, una trama remota con el identificador de este objeto de mensaje se transmite. Al recibir un trama de datos con coincidencia de identificador, este mensaje se almacena en este objeto de mensaje.

3.3.3.3 Registros de control de mensajes IFx (CAN0IFxMC)

	15	14	13	12	11	10	9	8	7	6	5	4	3 2 1 0
IF1 Message Control Register (addresses 0x1D & 0x1C)	NewDat	MsgLst	IntPnd	UMask	TxIE	RxIE	RmtEn	TxRqst	EoB	res	res	res	DLC3-0
IF2 Message Control Register (addresses 0x4D & 0x4C)	NewDat	MsgLst	IntPnd	UMask	TxIE	RxIE	RmtEn	TxRqst	EoB	res	res	res	DLC3-0
	rw	rw	rw	rw	rw	rw	rw	rw	rw	r	r	r	rw

UMask -> Uso de mascara de aceptación.

- **Uno**: Utiliza la máscara (Msk28-0, Mxtd y MDir) para filtro de aceptación.
- **Cero**: Mascara ignorada.

Nota: Si el bit UMask está a uno, los bits de la máscara del mensaje tienen que ser programados durante la inicialización del objeto de mensaje antes de que MsgVal esté a uno.

EoB -> Fin del buffer

- **Uno**: Objeto de mensaje único o el último mensaje de la cola.
- **Cero**: Objeto de mensaje perteneciente a la cola y no es el último objeto de mensaje de este buffer.

Nota: Este bit se utiliza para concatenar dos o más objetos de mensaje (por encima de 32) para construir una cola. Para objetos de mensajes solos (no pertecientes a la cola) es bit debe estar siempre activado a uno.

NewDat -> Nuevos datos.

- **Uno**: El manejador de mensaje o la CPU ha escrito nuevos datos en la parte de datos de este objeto de mensaje.
- **Cero**: No hay datos nuevos escritos dentro de la parte de datos de este objeto de mensajes por el manejador de mensajes desde la última vez que este flag fue puesto a 0 por la CPU.

MsgLst -> Mensaje perdido (solo válido para los objetos de mensaje con dirección = recibir)

- **Uno**: El manejador de mensajes almacena un nuevo mensaje dentro de este objeto cuando NewDat permanece activo, la CPU ha perdido un mensaje.
- **Cero**: No hay mensajes perdidos desde la última vez que este bit fue reiniciado por la CPU.

RxIE -> Habilitar interrupciones de recepción.

- **Uno**: IntPnd se activará después de la recepción correcta de una trama.
- **Cero**: IntPnd permanecerá sin cambiar antes de una recepción correcta de una trama.

TxIE -> Habilitar interrupciones de transmisión.

- **Uno**: IntPnd se activará después de la transmisión correcta de una trama.
- **Cero**: IntPnd permanecerá sin cambiar antes de una transmisión correcta de una trama.

IntPnd -> Interrupción pendiente.

- **Uno**: El objeto de mensaje es la fuente de la interrupción. El identificador de interrupción en el registro de interrupción apuntará a este objeto de mensaje si no hay otra fuente con prioridad mayor.
- **Cero**: Este objeto de mensaje no es fuente de una interrupción.

RmtEn -> Habilitar Trama Remota.

- **Uno**: Cuando se recibe una trama remota, se activa TxRqst.
- **Cero**: Cuando se recibe una trama remota, el bit TxRqst no cambia.

TxRqst -> Petición de transmisión.

- **Uno**: Se pide la transmisión de un objeto de mensaje y esta todavía no se ha realizado.
- **Cero**: El objeto de mensaje no está esperando para ser transmitido.

DLC3-0 -> Nuevos datos.

- **0-8**: La trama de datos tiene 0-8 bytes.
- **9-15**: La trama de datos tiene 8 bytes.

3.3.3.4 Registros de datos A y Datos B IFx (CAN0Dxx)

	15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0
IF1 Message Data A1 (addresses 0x1F & 0x1E)	Data(1)	Data(0)
IF1 Message Data A2 (addresses 0x21 & 0x20)	Data(3)	Data(2)
IF1 Message Data B1 (addresses 0x23 & 0x22)	Data(5)	Data(4)
IF1 Message Data B2 (addresses 0x25 & 0x24)	Data(7)	Data(6)
IF2 Message Data A1 (addresses 0x4F & 0x4E)	Data(1)	Data(0)
IF2 Message Data A2 (addresses 0x51 & 0x50)	Data(3)	Data(2)
IF2 Message Data B1 (addresses 0x53 & 0x52)	Data(5)	Data(4)
IF2 Message Data B2 (addresses 0x55 & 0x54)	Data(7)	Data(6)
	rw	rw

Data 0: Primer byte de datos de una trama de datos CAN.

Data 1: Segundo byte de datos de una trama de datos CAN.

Data 2: Tercero byte de datos de una trama de datos CAN.

Data 3: Cuarto byte de datos de una trama de datos CAN.

Data 4: Quinto byte de datos de una trama de datos CAN.

Data 5: Sexto byte de datos de una trama de datos CAN.

Data 6: Séptimo byte de datos de una trama de datos CAN.

Data 7: Octavo byte de datos de una trama de datos CAN.

Nota: El byte Data 0 es el primer byte de datos en desplazarse en el registro de desplazamiento del núcleo CAN durante una recepción. Cuando el manejador de mensajes almacena una trama de datos, escribirá todos los bytes dentro del objeto de mensaje. Si el Código de longitud de datos es menor que 8, los bytes restantes del objeto de mensaje se sobrescribirán con valores no especificados.

3.3.4 Objetos de mensaje en la memoria RAM

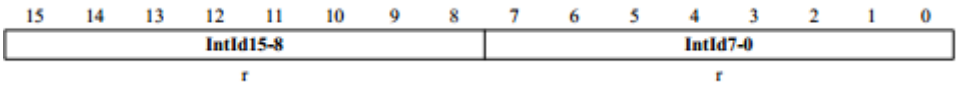
Message Object												
UMask	Msk28-0	MXtd	MDir	EoB	NewDat		MsgLst	RxIE	TxIE	IntPnd	RmtEn	TxRqst
MsgVal	ID28-0	Xtd	Dir	DLC3-0	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6	Data 7

Fig. 3.3: Estructura de un objeto de mensaje en la memoria de mensajes

Hay 32 objetos de mensaje en la RAM de mensajes. Para evitar conflictos entre el acceso a la CPU a la RAM de mensajes y la recepción y transmisión de mensajes CAN, la CPU no puede acceder directamente a la memoria de mensajes, estos accesos están manejados por los registros de interfaz IFx.

3.3.5 Registros del manejador de mensajes

3.3.5.1 Registro de interrupciones (CAN0IID)



- **IntId15-0**: Identificador de interrupción (El número indica la fuente de la interrupción).
    - 0x0000: No hay interrupciones pendientes.
    - 0x0001-0x0020: Numero de objeto de mensaje que ha provocado la interrupción.
    - 0x0021-0x7FFF: Sin uso.
    - Ox8000: Estado de interrupción.
    - 0x8001-0xFFFF: Sin uso

Una interrupción de mensaje se limpia cuando el bit del objeto de mensajes Intld. El estado de interrupción se limpia leyendo del registro de estado.

3.3.5.2 Registros de petición de transmisión (CAN0TRx)

	15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0
Transmission Request 1 Register (addresses 0x81 & 0x80)	TxRqst16-9	TxRqst8-1
Transmission Request 2 Register (addresses 0x83 & 0x82)	TxRqst32-25	TxRqst24-17
	r	r

TxRqst32-1 -> Bits de petición de transmisión (para todos los objetos de mensaje.

- **uno**: La transmisión de estos objetos de mensajes ha sido pedida y todavía no se ha finalizado.
- **cero**: Este objeto de mensaje no espera para transmitirse.

Estos registros almacenan el bit TxRqst de cada uno de los 32 objetos de mensaje. Con la lectura de estos bits, la CPU puede saber que objetos de mensaje tienen peticiones de transmisión.

3.3.5.3 Registro de nuevos datos (CAN0NDx)

	15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0
New Data 1 Register (addresses 0x91 & 0x90)	NewDat16-9	NewDat8-1
New Data 2 Register (addresses 0x93 & 0x92)	NewDat32-25	NewDat24-17
	r	r

NewDat32-1 Bits de datos nuevos (para todos los objetos de mensaje).

- **uno**: El manejador de mensajes o la CPU tiene escritos datos nuevos en la parte de datos de este objeto de mensaje.
- **cero**: No hay datos nuevos escritos en la parte de datos de este objeto de mensaje por el manejador de mensajes desde la ultima vez que este flag se puso a cero por la CPU.

Estos registros cogen los bits NewDat de los 32 objetos de mensaje. Leyendo los bits NewDat, la CPU puede saber que objetos de mensaje han sido actualizados con datos nuevos. Se puede resetear por la CPU mediente los registros de interfaz IFx o por el manejador de mensajes después de la recepción de una trama de datos o después de una transmisión correcta.

3.3.5.4 Registro de interrupciones pendientes (CAN0IPx)

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Interrupt Pending 1 Register (addresses 0xA1 & 0xA0)	**IntPnd16-9**								**IntPnd8-1**							
Interrupt Pending 2 Register (addresses 0xA3 & 0xA2)	**IntPnd32-25**								**IntPnd24-17**							
	r								r							

IntPnd32-1 Bits de interrupciones pendientes (para todos los objetos de mensaje).

- **uno**: Este objeto de mensaje es fuente de una interrupción.
- **cero**: Este objeto de mensaje no es fuente de una interrupción.

Estos registros cogen los bits IntPnd de los 32 objetos de mensaje. Leyendo los bits IntPnd, la CPU puede saber que objetos de mensaje han sido actualizados con datos nuevos. Se puede resetear por la CPU mediente los registros de interfaz IFx o por el manejador de mensajes después de la recepción de una trama de datos o después de una transmisión correcta. Esto también afectará al valor de Intld en el registro de interrupciones.

3.3.5.5 Registro de mensajes válidos (CAN0MVx)

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Message Valid 1 Register (addresses 0xB1 & 0xB0)	**MsgVal16-9**								**MsgVal8-1**							
Message Valid 2 Register (addresses 0xB3 & 0xB2)	**MsgVal32-25**								**MsgVal24-17**							
	r								r							

MsgVal32-1 Bits de mensajes válidos (para todos los objetos de mensaje).

- **uno**: Este objeto de mensaje está configurado y debería ser considerado por el manejador de mensajes.
- **cero**: Este objeto de mensaje se ignora por el manejador de mensajes.

Estos registros cogen los bits MsgVal de los 32 objetos de mensaje. Leyendo los bits MsgVal, la CPU puede saber que objetos de mensaje han sido actualizados con datos nuevos. Se puede resetear por la CPU mediente los registros de interfaz IFx.

3.4 Aplicación CAN

3.4.1 Manejo de los objetos de mensaje

La configuración de los objetos de mensaje en la RAM de mensajes no será afectado (con la excepción de los bits MsgVal, NewDat, IntPnd y TxRqst) al reinicio del chip. Todos los objetos de mensaje se inicializarán por la CPU o deberán ser no válidos (MsgVal = 0) y el tiempo de bit debe ser configurado antes de que la CPU limpie el bit de Init en el registro de control CAN.

La configuración de los objetos de mensaje se hace mediante la programación de los campos de máscara, Arbitraje, control y datos de uno de los dos sets de registros de interfaz con los valores deseados. Para escribir en los correspondientes Registros de orden de petición, los registros de buffer de mensajes IFx estarán cargados en las objetos de mensaje direccionados en la RAM de mensajes.

Cuando el bit Init en el registro de control CAN se limpie, el estado de la máquina del controlador de protocolo CAN del núcleo CAN y el estado de la máquina del manejador de mensajes controla el flujo de datos interno. Los mensajes recibidos que pasen el filtro de aceptación se almacenaran dentro de la RAM de mensajes, y los mensajes pendientes con petición de transición se cargarán dentro del registro de desplazamiento del núcleo CAN y serán transmitidos mediante el bus CAN.

La CPU lee los mensajes recibidos y los mensajes actualizados para transmitirse por los registros de interfaz IFx. Dependiendo de la configuración, la CPU interrumpe ciertos mensajes CAN y ciertos eventos de error.

3.4.2 Transferencia de datos desde/a RAM de mensajes

Cuando la CPU inicializa una transferencia de datos entre los registros IFx y la RAM de mensajes, el manejador de mensajes activa el bit Busy es el respectivo registro de

orden de máscara a 1. Después de que se complete la transferencia, el bit Busy vuelve a 0.

El registro de orden de máscara especifica si se transferirá un objeto de mensaje completo o solo una parte de él. Debido a la estructura de la RAM de mensajes no es posible escribir un solo bit/byte de un objeto de mensaje, siempre es necesario escribir un objeto de mensaje entero en la RAM de mensajes. Por lo tanto, los datos transferidos desde los registros IFx a la RAM de mensajes requieren un ciclo de lectura/modificación/escritura. Primero se leen desde la RAM de mensajes las partes del objeto de mensaje no van a cambiarse y entonces el contenido completo de los registros de buffer de mensajes se introducen en el objeto de mensaje.

Después de la escritura parcial de un objeto de mensaje, los registros de buffer de mensajes que no se hayan seleccionado en el registro de orden de máscara activaran al contenido actual de los objetos de mensaje seleccionados.

Después de una lectura parcial de un objeto de mensaje, los registros del buffer de mensajes, que no se seleccionen permanecerán sin cambiar.

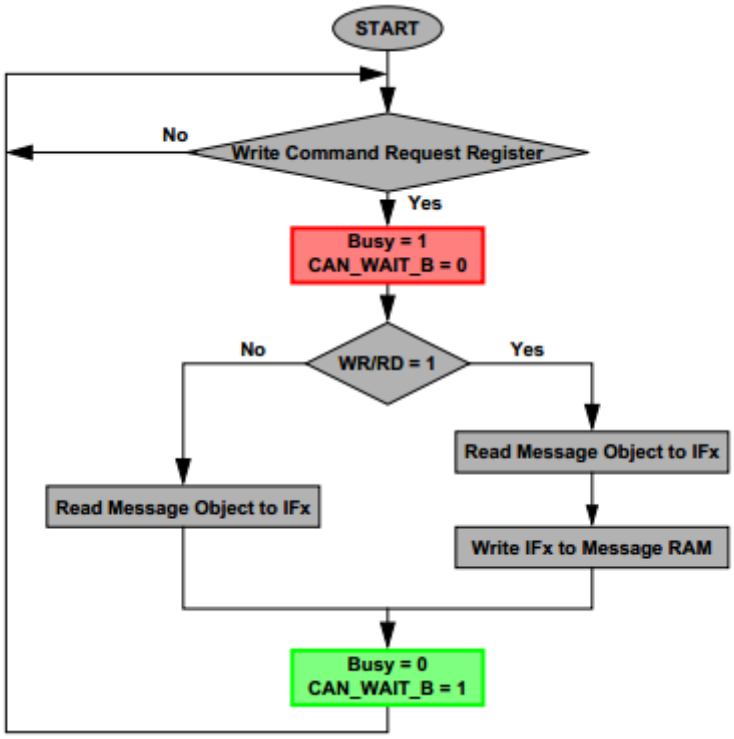


Fig. 3.4: Transferencia de datos entre los registros IFx y la RAM de mensaje

3.4.3 Transmisión de mensajes

Si el registro de desplazamiento del núcleo CAN está preparado para cargar y no hay transferencia de datos entre los registros IFx y la RAM de mensajes, se evaluará el bit MsgVal del registro de mensajes válidos y el bit TxRqst en el registro de petición de transmisión. Los objetos válidos con la mayor prioridad y pendientes de transmitir se

cargan en el registro de desplazamiento por el manejador de mensajes y la transmisión comienza. El bit NewDat del objeto de mensaje se activa.

Después de una transmisión correcta y si no se han escrito datos nuevos en el objeto de mensaje (NewDat = 0) desde el comienzo de una transmisión, el bit TxRqst se reinicia. Si TxIE está activo, IntPnd se activará después de una transmisión correcta. Si el C_CAN ha perdido el arbitraje o si durante la transmisión ha ocurrido un error, el mensaje será transmitido tan pronto como el bus CAN esté libre de nuevo. Si, mientras tanto, se ha solicitado la transmisión de un mensaje de prioridad mayor, se transferirán los mensajes dependiendo de su prioridad.

3.4.4 Filtros de aceptación de mensajes recibidos

Cuando el campo de control y el de arbitraje (Identificador + IDE + RTR + DLC) de un mensaje entrante está completamente desplazado en el registro de desplazamiento Rx/Tx del núcleo CAN, el manejador de mensajes FSM comienza el escaneo del mensaje de la RAM de mensajes para buscar una objeto de mensaje valido.

Para escanear la RAM de mensajes para un objeto de mensaje deseado, la unidad filtro de aceptación se carga con los bits de arbitraje desde el registro de desplazamiento del núcleo CAN. Entonces, los campos de arbitraje y máscara (incluyendo MsgVal, UMask, NewDat y EoB) del objeto de mensaje 1 se cargan en la unidad de filtro de aceptación y se comparan con los campos de arbitraje desde el registro de desplazamiento. Esto se repite con cada objeto de mensaje siguiente hasta que se encuentra un objeto de mensaje coincidente, o hasta el final de la RAM de mensajes.

Si ocurre una coincidencia, se detiene el escaneo y el manejador de mensajes FSM procede dependiendo del tipo de trama (trama de datos o trama remota) recibido.

3.4.4.1 Recepción de una trama de datos

El manejador de mensajes FSM almacena el mensaje desde el registro de desplazamiento del núcleo CAN al respectivo objeto de mensajes en la RAM de mensajes. No solo los bytes de datos, sino todos los bits de arbitraje y el código de longitud de datos se almacenan en su correspondiente objeto de mensaje. Esto se implementa para mantener los bytes de datos conectados con el identificador incluso si se usa máscara de arbitraje.

El bit NewDat se activa para indicar que los datos nuevos (no vistos todavía por la CPU) se han recibido. La CPU debe reiniciar NewDat cuando el objeto de mensaje esté listo. Si durante el tiempo en el que dura la recepción el bit NewDat se define como activado, MsgLst se activa para indicar que los datos previos (supuestamente no vistos por la CPU) se pierden. Si el bit RxIE se activa, el bit IntPnd se activa, causando que el registro de interrupción apunte a este objeto de mensaje.

El bit TxRqst de este objeto de mensaje se reinicia para prevenir la transmisión de una trama remota, mientras la trama de datos recibida, está siendo enviada.

3.4.4.2 Recepción de una trama remota

Cuando se recibe una trama remota, se consideran tres configuraciones del objeto de mensaje coincidente:

1) **Dir** = '1' (dirección = transmitir), **RmtEn** = '1', **UMask** = '0' o '1'.
   Si se recibe una trama remota, el bit TxRqst del objeto de mensaje se activa. El resto del objeto de mensaje permanece sin cambios.
2) **Dir** = '1' (dirección = transmitir), **RmtEn** = '0', **UMask** = '0'.
   Si se recibe una trama remota, el bit TxRqst del objeto de mensaje permanece sin cambiar; la trama remota se ignora.
3) **Dir** = '1' (dirección = transmitir), **RmtEn** = '0', **UMask** = '1'.
   Si se recibe una trama remota, el bit TxRqst del objeto de mensaje se reinicia. El campo de control y el de arbitraje (Identificador + IDE + RTR + DLC) del registro de desplazamiento se almacena dentro del objeto de mensaje en la RAM de mensajes y el bit NewDat del objeto de mensaje se activa. El campo de datos del objeto de mensajes permanece sin cambios; la trama remota se trata de manera similar a una trama de datos recibida.

3.4.5 Prioridad entre recepción/transmisión

La prioridad entre mensajes recibidos y transmitidos para los objetos de mensaje está relacionada con el número de mensaje. El objeto de mensaje 1 es el de mayor prioridad, mientras el objeto de mensaje 32 es el que menos prioridad tiene. Si hay más de una petición de transmisión pendiente, se servirán dependiendo de la prioridad del mensaje de objeto correspondiente.

3.4.6 Configuración de un objeto de transmisión

MsgVal	Arb	Data	Mask	EoB	Dir	NewDat	MsgLst	RxIE	TxIE	IntPnd	RmtEn	TxRqst
1	appl.	appl.	appl.	1	1	0	0	0	appl.	0	appl.	0

Fig. 3.4: Inicialización de un objeto de transmisión

Los registros de arbitraje (bits ID28-0 y Xtd) vienen dados por la aplicación. Definen el identificador y el tipo de mensaje saliente. Si se utiliza un identificador de 11 bits ("trama estándar"), se programarán del bit ID28 al bit ID18, siendo ignorados los bits desde ID17 a ID0.

Si el bit TxIE se activa, el bit IntPnd deberá estar activado después de una transmisión correcta del objeto de mensaje.

Si el bit RmtEn está activado, una recepción de trama remota activará el bit TxRqst; la trama remota se contestará de manera autónoma por una trama de datos.

Los registros de datos (DLC3-0, Data0-7) son dados por la aplicación, TxRqst y RmtEn no se pueden activar antes de validar los datos.

Los registros de mascara (bits Msk28-0, UMask, MXtd y MDir) pueden usarse (Umask = '1') para permitir que grupos de tramas remotas con el mismo identificador activen el bit TxRqst. El bit Dir debe no enmascararse.

3.4.7 Actualizar un objeto de transmisión

La CPU puede actualizar los bytes de datos de un objeto de transmisión cuando quiera mediante los registros de interfaz IFx, pero ni MsgVal ni TxRqst tienen que reiniciarse antes de la actualización.

Incluso si solo una parte de los bytes de datos se van a actualizar, los cuatro bytes correspondientes del registro IFx Datos A o del registro IFx Datos B tienen que ser válidos antes de que el contenido del registro se transfiera al objeto de mensaje. Ya sea la CPU escribiendo los cuatro bytes en el registro Datos del IFx o el objeto de mensaje transfiriendo a los registros de Datos del IFx antes que la CPU escriba nuevos bytes de datos.

Cuando solo los (ocho) bytes de datos son los que se actualizan, primero se escribe 0x0087 en el registro de orden de máscara, y después el número del objeto de mensaje se escribe en el registro de orden de petición, actualizando concurrentemente los bytes de datos y activando TxRqst.

Para prevenir el reinicio de TxRqst al final de la transmisión, que puede estar en proiceso mientras los datos se actualizan, NewDat tiene que estar activo junto con TxRqst.

Cuando NewDat se activa junto a TxRqst, NewDat se reiniciará tan pronto como la nueva transmisión se inicie.

3.4.8 Configurar un objeto de recepción

MsgVal	Arb	Data	Mask	EoB	Dir	NewDat	MsgLst	RxIE	TxIE	IntPnd	RmtEn	TxRqst
1	appl.	appl.	appl.	1	0	0	0	appl.	0	0	0	0

Fig. 3.5: Inicialización de un objeto de recepción

Los registros de arbitraje (bits ID28-0 y Xtd) vienen dados por la aplicación. Definen el identificador y el tipo de mensaje entrante aceptado. Si se utiliza un identificador de

11 bits (“trama estándar”), se programarán del bit ID28 al bit ID18, siendo ignorados los bits desde ID17 a ID0. Cuando una trama de datos con identificador de 11 bits se reciba, ID17-0 se pondrán a ‘0’.

El código de longitud de datos (DLC3-0) es dado por la aplicación. Cuando el manejador de mensajes almacena una trama de datos en el objeto de mensaje, se almacenará el código de longitud de datos y los ocho bytes de datos. Si el DLC es menor de 8, los bytes restantes del objeto de mensaje se sobrescribirán por valores no especificados.

Los registros de mascara (bits Msk28-0, UMask, MXtd y MDir) pueden usarse (Umask = ‘1’) para permitir que grupos de tramas de datos con el mismo identificador se acepten. El bit Dir debe no enmascararse en aplicaciones típicas.

3.4.9 Manejador de objetos de recepción.

La CPU puede leer un mensaje recibido cuando quiera mediante los registros de interfaz IFx, la consistencia de los datos está garantizada por la máquina de estado del manejador de mensajes.

Normalmente, la CPU escribirá primero 0x007F en el registro de orden de máscara y después el número del objeto de mensaje en el registro de orden de petición. Esta combinación transmitirá todo el mensaje recibido desde la RAM de mensajes al registro de buffer de mensajes. Además, los bits NewDat y IntPnd se limpiaran en la RAM de memoria (no en el buffer de mensajes).

Si el objeto de mensaje utiliza mascara para filtros de aceptación, los bits de arbitraje muestran cuales de los mensajes coincidentes se recibirán.

El valor actual de NewDat muestra si un mensaje nuevo se ha recibido desde la última vez que se leyó este objeto de mensaje. MsgLst no se reiniciará automáticamente.

Por medio de una trama remota, la CPU puede requerir a otro nodo CAN que le provea datos nuevos para un objeto de mensaje. Activando el bit TxRqst de un objeto de recepción, se causa una interrupción que causa que la transmisión de una trama remota con el identificador del objeto de recepción. Esta trama remota provoca al otro nodo CAN a iniciar la transmisión de la trama de datos coincidente. Si la trama de datos coincidente se recibe antes de que la trama remota pueda enviarse, el bit TxRqst se reinicia automáticamente.

3.4.10 Manejo de interrupciones

Si hay interrupciones pendientes, el registro de interrupciones CAN apuntará a las interrupciones pendientes de mayor prioridad, sin tener en cuenta el orden cronológico. Una interrupción permanece pendiente hasta que la CPU la limpia.

El estado de la interrupción tiene la mayor prioridad. Entre las interrupciones de mensaje, la prioridad de las interrupciones en objetos de mensaje, decrece conforme aumenta el número de mensaje.

Una interrupción de mensaje se limpia mediante la limpieza del bit IntPnd del objeto de mensaje. El estado de la interrupción se limpia tras leer el registro de estado.

El identificador de interrupción Intld en el registro de interrupción indica la causa de la interrupción. Cuando no hay interrupciones pendientes, el registro mantendrá el valor

Cero. Si el valor del registro de interrupción es diferente de cero, entonces hay una interrupción pendiente y, si IE está activo, la línea de interrupción a la CPU, IRQ_B, está activa. La línea de interrupción permanece activa hasta que el valor del registro de interrupción vuelve a cero (la causa de la interrupción se reinicia) o hasta que IE se reinicie.

El valor 0x8000 indica que una interrupción está pendiente porque el núcleo CAN ha actualizado (no necesariamente cambiando) el registro de estado (Interrupción de error o interrupción de estado). Esta interrupción tiene la prioridad mayor. La CPU puede actualizar (reiniciar) los bits de estado RxOk, TxOk y LEC, pero un acceso de escritura al registro de estrado nunca puede generar o reiniciar una interrupción.

Todos los otros valores indican que la fuente de las interrupciones es uno de los objetos de mensaje, Intld apunta a la interrupción del mensaje pendiente con la mayor prioridad de interrupción.

La CPU controla si un cambio en el registro de estado ha podido causar una interrupción (los bits EIE y SIE en el registro de control CAN) y si la línea de interrupción se activa cuando el registro de interrupción es distinto de cero (bit IE en el registro de control CAN). El registro de interrupción se actualizará incluso cuando se reinicie el bit IE.

La CPU tiene dos posibilidades para seguir la fuente de la interrupción de mensaje. Primero puede seguir el bit Intld en el registro de interrupciones y, segundo, puede consultar el registro de interrupciones pendientes.

Una interrupción de rutina de servicio leyendo el mensaje que es fuente de interrupción puede leer el mensaje y reiniciar el bit IntPnd del objeto de mensaje a la vez. (el bit ClrIntPnd en el registro de orden de máscara). Cuando IntPnd está limpio, el registro de interrupción apuntará al siguiente objeto de mensaje con una interrupción pendiente.

3.4.11 Tiempo de bit y tasa de bit

CAN soporta tasas de bits en el rango desde 1Kbit/s hasta 1000 Kbit/s. Cada miembro de una red CAN tiene su propio generador de reloj, normalmente un oscilador de cuarzo. El parámetro temporal de tiempo de bit (*bit time*, es decir, lo reciproco a la tasa de bit) puede configurarse individualmente para cada nodo CAN, creando una tasa de bits común incluso si los periodos de oscilación de los nodos CAN es diferente.

Las frecuencias de estos osciladores no es absolutamente estable, con pequeñas variaciones causadas por cambios en la temperatura o el voltaje o el deterioro de los componentes. Tanto como las variaciones permanentes dentro del rango de tolerancia de los osciladores específicos (df), los nodos CAN permiten compensar las diferentes tasas de datos mediante la resincronización del flujo de bit.

De acuerdo a las especificaciones CAN, el tiempo de bit se divide en cuatro segmentos. El segmento de sincronización, el segmento de tiempo de propagación, el segmento de buffer de fase 1, y el segmento de buffer de fase 2. Cada segmento consiste en un número específico y programable del quantum de tiempo. La longitud del quantum (tq), que es la unidad de tiempo básica de tiempo de bit, se define por el reloj de sistema del controlador CAN fsys, y la preescalador de la tasa de baudios (BRP), tq = BRP /fsys. El reloj del sistema C_CAN fsys es la frecuencia de salida de la salida de CAN_CLK.

El segmento de sincronización Sync_Seg es la parte del tiempo de bit cuando se llega al final del nivel del bus CAN que se espera que ocurra; la distancia entre que un límite ocurre fuera del Sync_Seg y el Sync_Seg se llama error de fase o de límite. El segmento tiempo de propagación se destina a compensar los tiempos de retraso físico en la red CAN. Los segmentos de buffer de fase Phase_Seg1 y Phase_Seg2 envuelven al punto de muestra. La (re) sincronización del ancho de salto (SJW) define cuánto tardará una resincronización en mover el punto de muestra dentro de los límites definidos por el segmento de buffer de fase para compensar los errores de fase de límites.

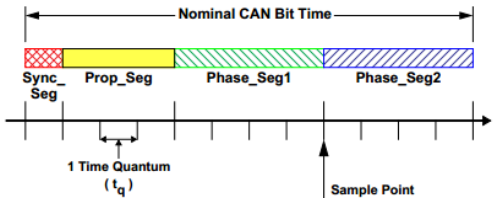


Fig. 3.5: Temporización de bit

Parámetro	Rango	Observaciones
BRP	[1…32]	Define la longitud del quantum de tiempo tq
Sync_Sec	1 tq	Fija la longitud, sincroniza la salida del bus del reloj del sistema
Prop_Seg	[1…8] tq	Compensación para los tiempos de retraso físicos
Phase_Seg1	[1…8] tq	Puede alargarse temporalmente por sincronización
Phase_Seg2	[1…8] tq	Puede acortarse temporalmente por sincronización
SJW	[1…4] tq	Puede no ser mas largo que el segmento de buffer de registro.
Esta tabla describe los rangos mínimos programables requeridos por el protocolo CAN		

Tabla 3.2 Parámetros del tiempo de bit CAN

Una tasa de bits puede ser configurada con distintas configuraciones de tiempo de bit, pero para la función propia de la red CAN, tienen que considerarse los tiempos de retraso físicos y el rango de tolerancia del oscilador.

4. Herramientas utilizadas

En este apartado, trataremos las dos herramientas que se han utilizado para la edición de código: Microsoft Visual Studio 2010, en su versión Ultimate, en la parte del monitor de PC, en el cual se ha desarrollado un proyecto en Windows Forms, es decir, en las clásicas ventanas de los sistemas operativos convencionales.

En la parte del monitor en el microcontrolador, se ha utilizado la herramienta que Silabs nos proporciona, Silabs Laboratories IDE, una herramienta más tosca, con menos facilidades y de aspecto antiguo con el cual se introduce el código en el C8051F500, el microcontrolador que hará de monitor.

4.1 Microsoft Visual Studio 2010 Ultimate

Microsoft Visual Studio Ultimate es un paquete completo de herramientas de administración del ciclo de vida de las aplicaciones para equipos. Con este paquete se pueden realizar desde el diseño hasta la implementación. Tanto como si se crean soluciones nuevas como mejorar aplicaciones ya existentes, Visual Studio 2010 Ultimate es útil gracias a que admite un número considerable de plataformas y tecnologías.

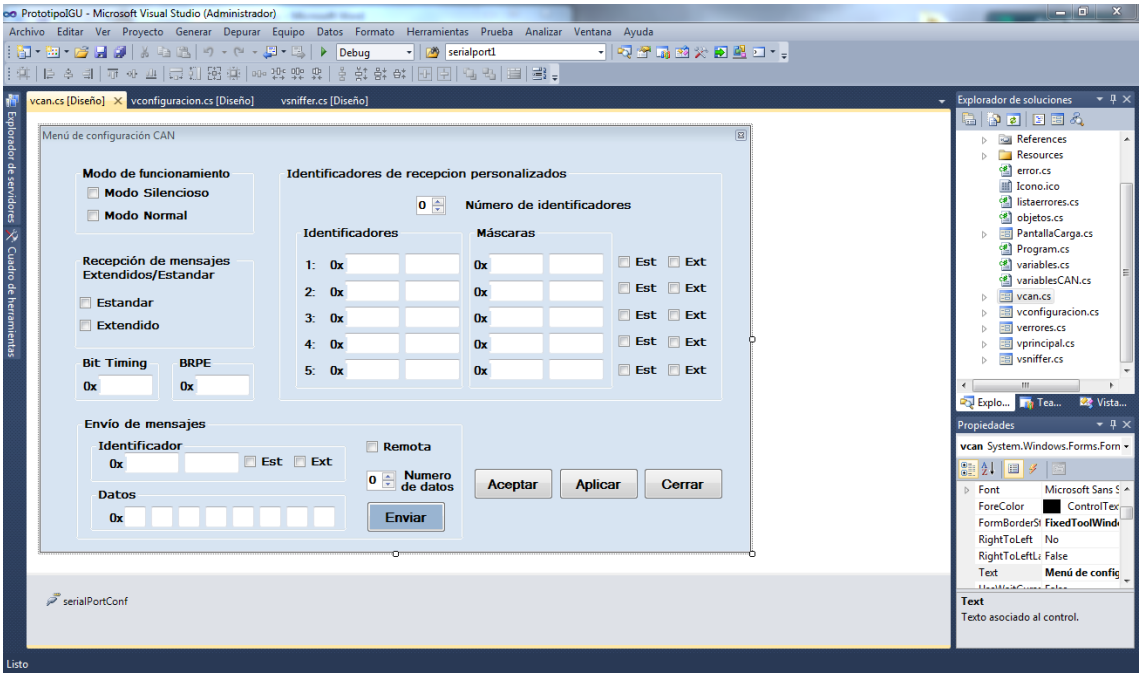


Fig. 4.1: Ventana de edición de formularios Windows de VS 2010 Ultimate

En esta herramienta se ha desarrollado toda la interfaz gráfica, además de la lógica de comunicación con el puerto serie. Se escogió este programa por la gran ayuda que proporciona al desarrollador a la hora de editar código, y por su simplicidad a la hora de editar los formularios, sin ser problema para permitir introducir gran cantidad de elementos para la interacción con el usuario.

4.1.1 C#

C#, también escrito algunas veces como C Sharp, es el lenguaje de programación utilizado en la parte del PC. Es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET

La sintaxis de C# es muy expresiva, aunque cuenta con menos de 90 palabras clave; también es sencilla y fácil de aprender. La sintaxis de C# basada en signos de llave podrá ser reconocida inmediatamente por cualquier persona familiarizada con C, C++ o Java. Los desarrolladores que conocen cualquiera de estos lenguajes pueden empezar a trabajar de forma productiva en C# en un plazo muy breve. La sintaxis de C# simplifica muchas de las complejidades de C++ y, a la vez, ofrece funciones eficaces tales como tipos de valores que aceptan valores NULL, enumeraciones, delegados, métodos anónimos y acceso directo a memoria, que no se encuentran en Java. C# también admite métodos y tipos genéricos, que proporcionan mayor rendimiento y seguridad de tipos, e iteradores, que permiten a los implementadores de clases de colección definir comportamientos de iteración personalizados que el código de cliente puede utilizar fácilmente.

Como lenguaje orientado a objetos, C# admite los conceptos de encapsulación, herencia y polimorfismo. Todas las variables y métodos, incluido el método Main que es el punto de entrada de la aplicación, se encapsulan dentro de definiciones de clase. Una clase puede heredar directamente de una clase primaria, pero puede implementar cualquier número de interfaces. Los métodos que reemplazan a los métodos virtuales en una clase primaria requieren la palabra clave **override** como medio para evitar redefiniciones accidentales. En C#, una estructura es como una clase sencilla; es un tipo asignado en la pila que puede implementar interfaces pero que no admite la herencia.

4.2 Silicon Laboratories IDE y compilador KEIL.

Para la programación del microcontrolador se ha hecho uso del entorno de desarrollo integrado (IDE), el cual viene incluido como un elemento más del kit de desarrollo. Desde este entorno dispondremos de un gestor de proyectos, un editor de código y una herramienta de depuración entre otras utilidades. No obstante, dicho IDE no cuenta con un compilador propio, sino que deberá usarse un compilador específico desarrollado por Keil (compañía asociada a ARM), el cual se integra en el entorno, formando en conjunto la herramienta necesaria para el desarrollo de aplicaciones para nuestro sistema empotrado.

Fig. 4.2: Kit de desarrollo C8051F500 de Silabs (Hardware)

Nos da a elegir dos lenguajes de programación a la hora de desarrollar código. Nosotros usaremos el lenguaje C, lenguaje mucho más cómodo, simple y limpio que la otra opción, que es desarrollar directamente en ensamblador. Además, las funciones extras que hemos creado, como la función buffer, pueden ser reutilizables en otros proyectos fácilmente.

No se han utilizado librerías adicionales, aparte de las que definen los registros, definidas por el fabricante del microcontrolador, para tener un mejor manejo en todo momento de la comunicación con el puerto serie.

También hacemos uso de la herramienta debugger, proporcionado con el kit de desarrollo, que nos permite descargar el código al microcontrolador, y además, ejecutarlo instrucción a instrucción para poder visualizar el desarrollo del programa instrucción a instrucción. También nos permite comprobar el estado de las variables y los registros en tiempo de ejecución. El debugger es una herramienta más que necesaria para desarrollar la aplicación, ya que nos permite analizar el código de una forma fácil e intuitiva.

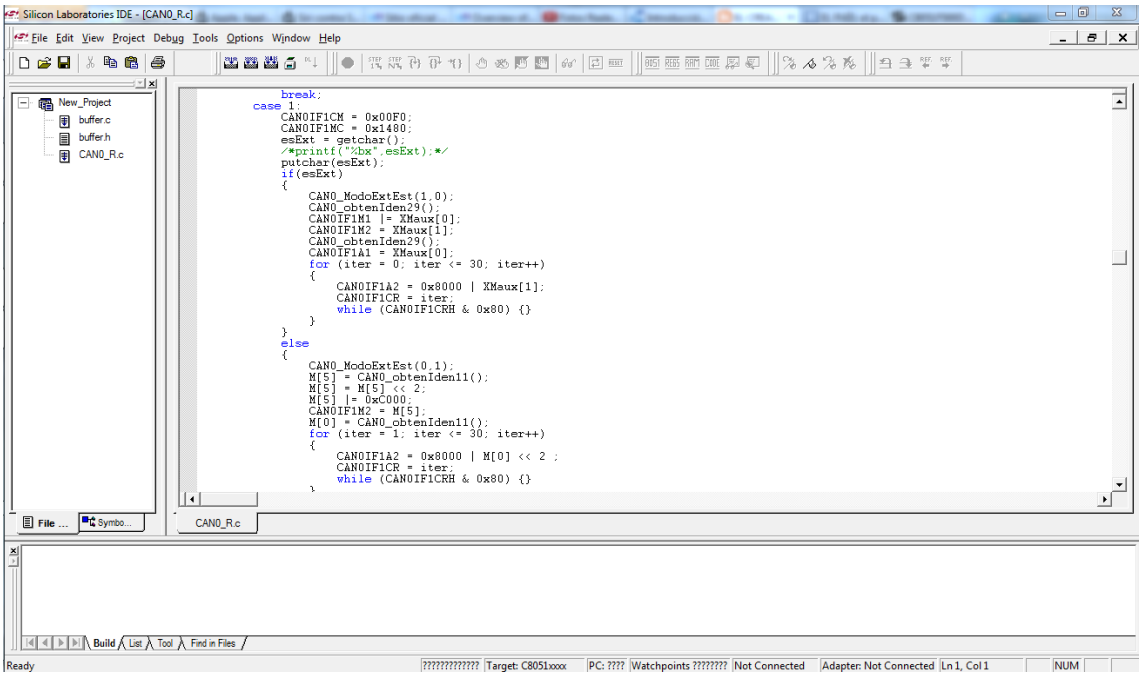


Fig. 4.3: Ventana del IDE de Silicon Labs

4.3 Placa de desarrollo Silabs C8051F500

Manual del C8051F50x/F51x de Silicon Labs.

Este dispositivo es un sistema MCU integrado de señal mixta en un chip (*integrated mixed-signal System-on-a-chip*). Tiene las siguientes características, genéricas a toda la familia F50x y F51x:

- Reloj del sistema de 50 Mhz y un rendimiento pico de 50 MIPS.
- 4352 bytes de RAM (256 bytes internos y 4096 bytes de XRAM).
- SMBus/I^2C, SPI mejorado, UART mejorada.
- Cuatro temporizadores
- Seis canales de contadores programables en array.
- Oscilador interno de 24 Mhz.
- Regulador de tensión interno.
- 12 bits, 200 ksps ADC.
- Referencia interna de voltaje y sensor de temperatura.
- Dos comparadadores analógicos.

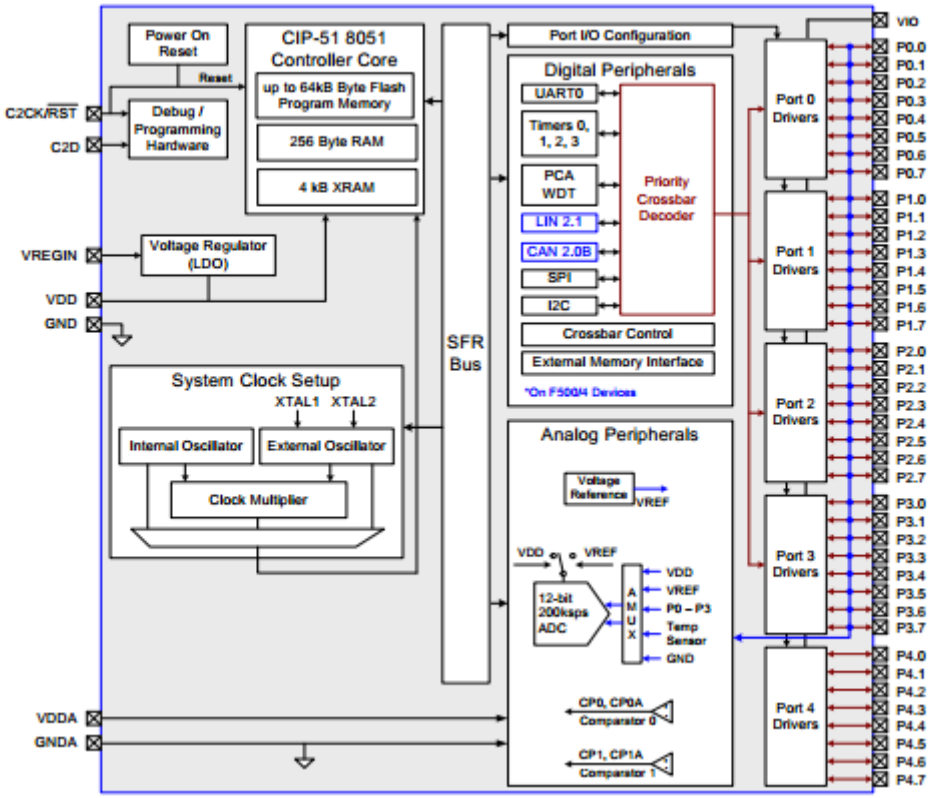


Fig. 4.4: Diagrama de bloque del C8051F500/1/4/5

El corazón de este sistema es un integrado de la familia 8051 de Intel, que data de los años 80, para uso en sistemas empotrados. Es un sistema obsoleto, con popularidad en la década de los 80 y 90, pero para nuestro uso es más que suficiente. La '*C*' del nombre es indicativo del uso de tecnología CMOS.

De todo el sistema, nosotros usaremos la UART, para comunicarnos con el PC mediante el puerto serie, y el módulo CAN, donde con los registros y las metodologías explicadas en el punto 3, y de la manera que explicaremos en el punto 6, se ha realizado toda la parte de configuración y envío de tramas.

Este dispositivo es, al conectarlo a una red CAN mediante su puerto lateral, el que hará de monitor y analizador de la red. El monitor, que se ha programado especialmente para él, deberá ser descargado antes de poder ser utilizado para ese menester.

5. Monitor en el microcontrolador

Con todo lo comentado anteriormente, vamos a analizar la parte de configuración instalada en el microcontrolador. Dividiremos en tres partes la funcionalidad del driver: bucle de recogida de opciones y lógica de configuración, la comunicación con el PC mediante la UART y la lógica de envío y recepción de tramas CAN.

El monitor dispone de una inicialización, en la cual se activan todas las opciones por defecto antes de entrar al bucle, y son estas:

- Modo de funcionamiento normal.
- Registro *Bit Timing* = 0x1402.
- Registro BRPE = 0x0000.
- Admisión de tramas con identificador estándar y extendido.
- Cualquier número de identificador.

Por lo tanto, una vez instalado el software de monitorización en el microcontrolador, obtendremos todos los paquetes de la red, siendo visible además a los distintos nodos de la red, por lo que cada trama que recibamos, tendrá su respuesta con su ACK correspondiente. La visibilidad la podremos modificar mediante la puesta en funcionamiento en modo silencioso, que como hemos explicado antes, puede ser útil para monitorizar sin interferir en el tráfico de la red. Las otras inicializaciones corresponden al oscilador, el cual configuramos de modo que la variable SYSCLK deriva desde el oscilador interno dividido entre 1 (OSCICN = 0x87). El registro RSTSRC, nos indicará si un desbordamiento del temporizador ha sido la que ha causado el último reset.

Respecto a la inicialización del puerto, activamos el pin P0.7 (CAN_RX) para entrada/salida digital (para configurar una salida digital, el pin tiene que configurarse como digital y como push-pull(fuerza la tensión de salida independientemente de donde esté conectada la carga, y cuya lógica es: ‘1’ activado, ‘0’ desactivado). A las salidas P1.4 (botón) y P1.3 (LED) se las configura en push-pull (P0MDOUT |= 0x40 y P1MDOUT |= 0x18). Después, activamos las salidas de CAN0 (pines P0.6 para CAN_TX y P0.7 para CAN_RX) y UART0 (pines P0.4 para TX0 y P0.5 para RX0).

Por último, en la inicialización de la UART, borramos TI0 y RI0, por si había quedado algún dato eliminarlo, se calcula el valor del generador de tasa de baudios, dependiendo de SYSCLK, BAUDRATE (Ratio de baudios = [Reloj BRG / (65536 - (SBRLH0:SBRLL0))] x 1/2 x 1/preescalador) y el divisor que hayamos puesto. Por último, activamos TI0, generando una interrupción para poder seguir transmitiendo.

Por último se activan las interrupciones CAN, y las interrupciones generales (EIE2 |= 0x02 y EA = 1), y activamos el LED para notificar que el dispositivo ya está en condiciones de empezar a realizar su función. Hay varias variables globales, pero la más importante es cola, un buffer donde se almacenaran las tramas con su identificador, número de bytes y los datos, con capacidad para 64 tramas distintas. Su ubicación es la memoria xdata, ya que la parte de memoria reservada a datos es demasiado pequeña. El funcionamiento de esta cola lo explicaremos más adelante.

Como método de seguridad en la transferencia mediante la UART con el PC, cuando se recibe un byte, este se devuelve al PC. Esto se hace como método de sincronización. Es decir, cuando recibimos un byte, el PC se queda esperando a que le llegue la respuesta, deteniendo la ejecución del programa. Cuando recibe el byte, se continúa con la ejecución del programa, ya que sabemos que el microcontrolador estará preparado para seguir ejecutando su código. Cuando tiene que recibir un dato, el microcontrolador se espera a que este le llegue, y mientras no se realice esta operación, se queda esperando.

No analizaremos en mayor profundidad estas tres funciones, porque no es el cometido de este trabajo, pero si se necesita saber mejor cómo funcionan, se recomienda la lectura del manual de microcontrolador de Silicon Labs.

5.1 Bucle de recogida de opciones y lógica de configuración

Una vez se ha inicializado todo lo que nos interesa, entramos a un bucle que se repite indefinidamente. Este bucle infinito, se encarga de llamar a dos funciones: comprueba_opción(), la cual se hace siempre para comprobar si ha llegado una opción nueva, y después, si la cola donde se almacenarán las tramas que llegan no está vacía, se saca una trama, mediante la función trans_byts(). Si llegan varias tramas a la vez, se irán sacando poco a poco tras cada iteración.

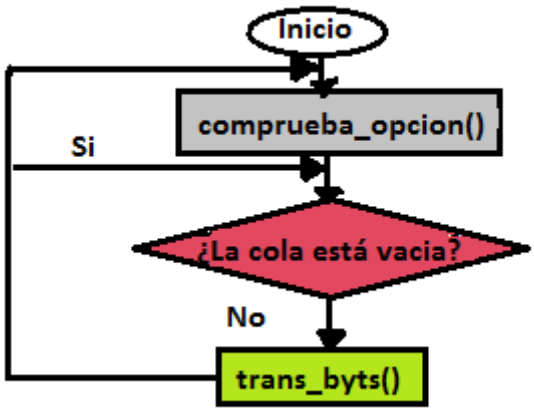


Fig. 5.1: Bucle infinito de la rutina principal

5.1.1 Comprobación de opción (*comprueba_opcion()*)

Esta rutina comprueba, primeramente, si hay algo en SBUF0 para recibir mediante la comprobación del bit RI0, en el registro SCON0 (registro de configuración de la UART, ver figura 5.2). Este bit se activa por hardware cuando ha llegado un dato para lectura. Una vez leído el dato, se pone a 0 manualmente.

Bit	7	6	5	4	3	2	1	0
Name	OVR0	PERR0	THRE0	REN0	TBX0	RBX0	TI0	RI0
Type	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W
Reset	0	0	1	0	0	0	0	0

SFR Address = 0x98; Bit-Addressable; SFR Page = All Pages

Fig. 5.2: Contenido registro SCON0

Si hay un dato, entonces se pasa a comprobar el contenido de ese byte, ya que la lógica del programa del PC, envía siempre como primer dato el byte de opciones. El byte de opciones tiene el siguiente formato:

Bit	7	6	5	4	3	2	1	0
Nombre	R/T 2	R/T 1	R/T 0	EST	BTR	BRPE	EXT	MODO
Tipo	R	R	R	R	R	R	R	R
Reset	0	0	0	1	1	0	1	1

Tabla 5.1: Contenido del byte de opciones

Se ha intentado que el tamaño del byte de opciones sea el más pequeño posible, para que no se genere más tráfico del necesario a la hora de especificar que parámetros se tienen que modificar. El valor de reinicio es el mismo que toma en su inicialización, esto es, 0x1B.

Es importante reseñar que, debido a la naturaleza de la modificación de los temporizadores, cada vez que se vaya a cambiar cualquiera de los dos registros (*Bit Timing o BRPE*), se tienen que establecer valores en todos ellos, incluido también el modo de funcionamiento. Esto es así porque antes de cambiar los tiempos, hay que reiniciar el microcontrolador, perdiendo la configuración de los registros que se tenía anteriormente. Es un dato que hay que tener en cuenta a la hora de realizar la aplicación de PC. Sin embargo, esto no afecta a la configuración de las máscaras e identificadores, pues quedan almacenados en su objeto de mensaje correspondiente.

En la página siguiente, tenemos una tabla con lo que significan los valores del byte de opción, y un diagrama explicativo sobre que se hace cada vez que se entra en la función comprueba_opcion ().

Bit	Nombre	Función
7-5	R/T	**Configurar Recepción/Transmisión de Trama** **000**: Dependiente de los bits EST y EXT. **001**: Se va a configurar una máscara/identificador para su recepción. **010**: Se van a configurar dos máscaras/identificadores para su recepción. **011**: Se van a configurar tres máscaras/identificadores para su recepción. **100**: Se van a configurar cuatro máscaras/identificadores para su recepción. **101**: Se van a configurar cinco máscaras/identificadores para su recepción. **110**: Se va a configurar y proceder a hacer un envío de trama de datos. **111**: Se va a configurar y proceder a hacer un envío de trama remota.
4	EST	**Activar Recepción Trama Estándar** **0**: No se tienen en cuenta las tramas estándar. **1**: Se activa la recepción de tramas estándar.
3	BTR	**Editar Registro Bit *Timing*** **0**: El registro Bit Timing no se modifica. **1**: Se va a modificar el registro Bit Timing.
2	BRPE	**Editar Registro BRPE** **0**: El registro BRPE no se modifica. **1**: Se va a modificar el registro BRPE.
1	EXT	**Activar Recepción Trama Extendida** **0**: No se tienen en cuanta las tramas extendidas. **1**: Se activa la recepción de tramas extendidas.
0	MODO	**Modo de Funcionamiento** **0**: Modo Silencioso. No se interfiere en la red. **1**: Modo de funcionamiento normal. Se comportará como un nodo más.

Tabla 5.2: Explicación de los bits del byte de opción

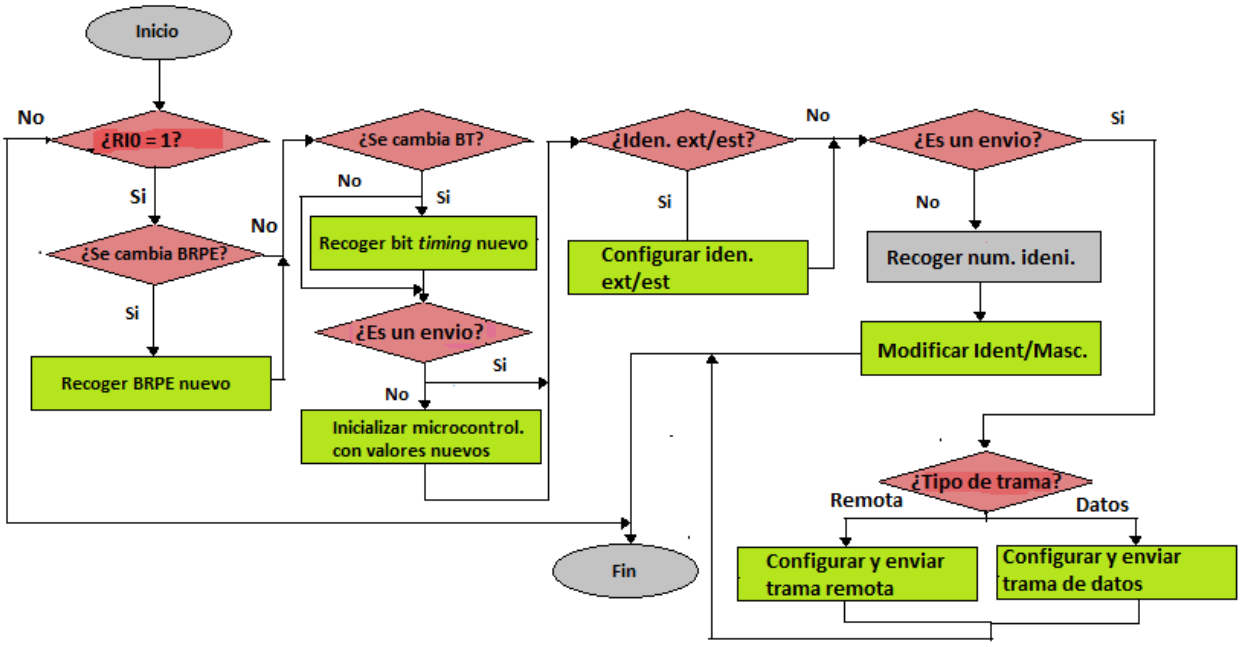


Fig. 5.3: Lógica del método escoger opción

5.1.2 Lógica de configuración

En este método, se recogerán los datos (si es necesario) del valor que se quiera poner en los registros BRPE y BT. Se recibirán los datos tal cual se deben escribir, por simplicidad a la hora de poder escribirlos. Esto se ha hecho así para simplificar la interfaz gráfica, y evitar que quede recargado. Por lo tanto, el usuario tendrá que tener calculado ya la velocidad que quiere (Se puede consultar que valores poner en los apartados **3.3.1.4** y **3.3.1.6** y **3.4.11**).

Para saber en qué modo está, y si se activará el uso de recepción de tramas extendidas o estándar, simplemente se consulta el bit que identifica si se realizan esos cambios, y se llama a las funciones correspondientes, que realizan las modificaciones pertinentes en los registros CAN0IF1Ax y CAN0IFMx. Para cambiar el modo, se tiene en cuenta si el registro test está activado, y si el bit de modo silencioso dentro de ese registro también lo está. Si queremos pasar de un modo a otro, solo tenemos que desactivar o activar estos bits (registros CAN0CN y CAN0TST, apartados **3.3.1.1** y **3.3.1.5**).

Para configurar mensajes o envíos, en esta rutina solo se decodifican los bits R/T, para saber, si es una recepción, cuantas máscaras/identificadores vamos a configurar para recibir, y si es un envío, que tipo de envío es. En otro apartado veremos cómo se interpreta esta información.

5.2 Comunicación con el PC

Como se ha explicado anteriormente, la comunicación entre microcontrolador y PC se realiza mediante un puerto USB, el cuál se convierte en serie para el PC con un convertidor serie-USB de Silabs (CP2102), que genera un puerto virtual COMx en el PC. En el microcontrolador, se utiliza la UART para enviar y recibir bytes de datos. La configuración de este puerto se hace al inicio. Es desde el PC donde se tienen que establecer los parámetros de comunicación con el micro (puerto, velocidad y paridad).

Se tienen dos funciones para enviar y recibir bytes. La función ***putchar (char output)***, que comprueba mediante la consulta del byte THRE0, si es posible enviar datos. Si se puede acceder a mandar un byte, este se escribe en el registro SBUF0, y el byte se envía al otro extremo del bus serie. La función ***U8 getchar ()***, que devuelve un dato del registro SBUF0, si ha llegado algo (para consultar como es el registro de configuración de la UART, se puede ver en la figura **5.2** en el apartado **5.1.1**). U8 es un tipo de dato definido en la librería compilers_defs.h, que no es más que una variable de tipo carácter (*char*), cuya longitud es de 8 bits. Las dos figuras siguientes muestran el funcionamiento de estas dos funciones:

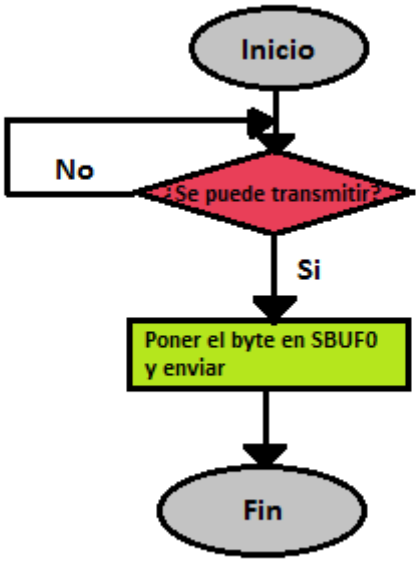


Fig. 5.4: Diagrama del putchar (char output)

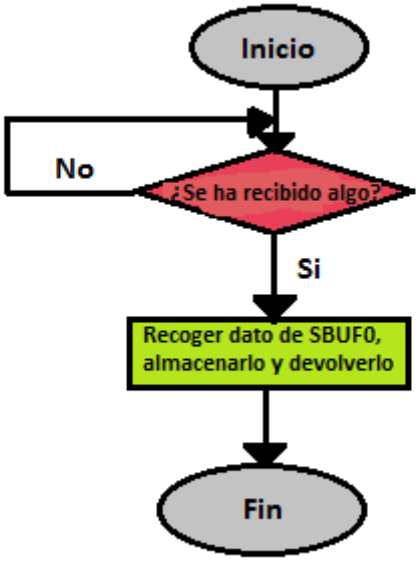


Fig. 5.5: Diagrama del U8 getchar ()

Como se puede comprobar, son dos rutinas sencillas, pero que aseguran que el flujo de información sea el correcto.

5.2.1 Comprobación del dato recibido

Como ya adelantamos, siempre que vayamos a recibir un byte, este será devuelto como método de seguridad para sincronizar los dos programas, ya que el programa de PC se quedará detenido esperando a la respuesta, antes de volver a mandar otro byte, eliminando la posibilidad de que se pueda perder información. La respuesta será la misma que el byte enviado. Así pues, para comprobar si ha habido error en la comunicación o no, el programa en el PC solo tiene que comprobar que este byte es el mismo que se envió.

5.3 Lógica de envío y recepción de tramas CAN

La configuración de la recepción o el envío de tramas CAN se realiza mediante la modificación de los registros de interfaz, y su escritura en objetos de mensaje. Para entender qué hacen todos estos registros, se recomienda la lectura de los apartados **3.3.2** y **3.4.**

5.3.1 Configuración de recepción de tramas

Tenemos 6 posibilidades a la hora de configurar la recepción. Estas posibilidades tienen que ver con el número del par identificador/mascara personalizados que vayamos a utilizar.

Cuantos más identificadores vayamos a personalizar, menos objetos de mensajes se reservarán para cada uno (lógicamente, ya que podremos reservar menos objetos de mensaje de los que podríamos con menos identificadores). Con 0 identificadores, se recogen los bits del byte de opciones EXT y EST, que permiten recibir tramas de un tipo, o de los dos. Con más de un identificador personalizado, estos dos bits siempre están a 0, y se procede a la lectura, primero cuantos id's vamos a poner, y después para cada uno, si va a ser extendido o no, y que id's/mascaras queremos.

Como máximo para un identificador personalizado, tendremos 30 objetos de mensaje donde almacenar los objetos de mensaje. Con 2, esta cifra se disminuye a 15, con 3 identificadores 10, con 4 tendremos 7 objetos de mensaje para cada uno, y por último, con 5, tendremos reservados 6 objetos de mensaje para cada identificador/mascara personalizado. Esto hace que tengamos que recoger los datos lo antes posible. Para nuestro caso, no hay problema, pues se hace durante la interrupción de recepción que se genera al recibir un mensaje.

Ya que los identificadores y las máscaras pueden ser de un tamaño grande, se han realizado dos funciones para reconstruir los bytes que se reciben. Así, disponemos de dos funciones que nos devuelven los valores deseados en un U16 (o dos, dependiendo si es extendida o estándar). El tipo de datos U16 es un tipo de dato renombrado que en realidad es un entero sin signo (16 bits).

Los dos últimos, el 31 y el 32 (que en el microcontrolador lo almacena en el objeto de mensaje 0), se reservan. El 31 para enviar, y el 32 para la recepción de tramas remotas, que se quedará configurado para siempre, se elija la configuración que sea.

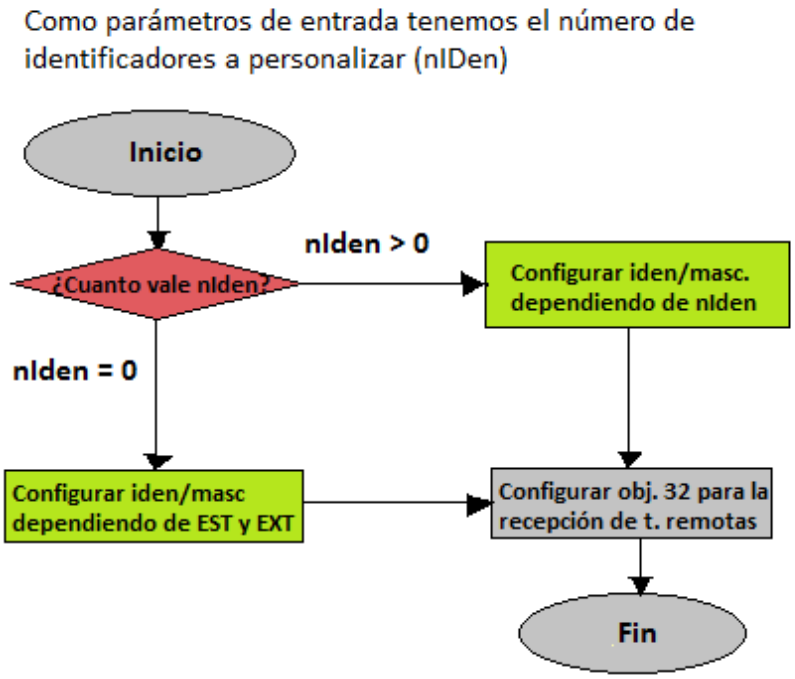


Fig. 5.6: Diagrama de CAN0_cambiaIden (U8 nIden)

Para modificar los objetos de mensaje, se sigue el siguiente planteamiento:

3. Configurar estándar o extendido.
4. Recogen la máscara.
5. Escribir máscara en los registros CAN0IF1M1 (ver apartado **3.3.3.1**)
6. Recoger el identificador.
7. Escribir identificador en los registros CAN0IF1A1 (ver apartado **3.3.3.2**).
8. Almacenar, mediante la configuración con el objeto de mensaje deseado en el registro CAN0IF1CR (apartado **3.3.2.1**).

5.3.2 Envío de tramas (Trama de datos y trama remota)

Cuando se va a realizar un envío de trama, no se hace nada más que configurar y realizar el envío de la trama (ver figura 5.3). En la configuración del envío, se tiene en cuenta que tipo de trama es. Una vez obtenemos la información de si es remota o no, activamos las interrupciones, escribimos los valores en los registros implicados (CAN0IF1Ax, CAN0IF1Mx, CAN0IF1CM y CAN0IF1CR). Si es trama remota activamos el bit Dir, y el RmtEn. Si es de datos, antes de realizar la transmisión se escriben los datos recibidos en los registros de datos (dependiendo del número de datos que se quiera enviar, se escribirá en unos registros o en otros). Para ver como se realiza un envío en profundidad, se consulta la lectura de los apartados **3.4.3** y **3.4.8**.

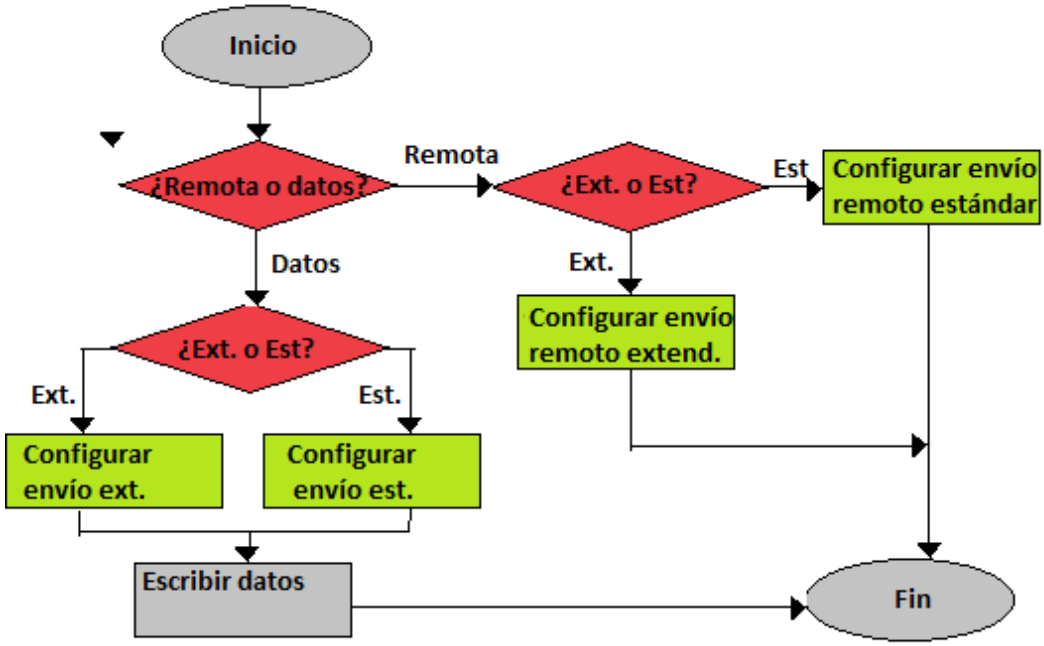


Fig. 5.7: Diagrama de conf. y envío de trama

En la trama remota, el campo DLC también se rellena porque algunas marcas lo utilizan para decir cuántos datos se quieren. Sin embargo, no se escribe nada en ellos, ya que no se envían.

5.3.3 Manejo de interrupciones

En el driver, el uso de interrupciones se restringe al tráfico CAN. Esto se hace así para otorgarle prioridad en el caso de la recepción de una trama CAN. Sin embargo, estas se desactivan cuando se está recibiendo información a través del puerto serie, y una vez realizada la comunicación y la recepción, éstas se vuelven a activar. Esta desactivación responde al hecho de que la comunicación entre el PC y el microcontrolador no se vea afectada. También se activa, lógicamente, mientras se está configurando un envío para realizarlo.

Una vez se va a proceder a enviar, o se va a recibir alguna trama, lo primero que recogemos es el estado para ver si hemos hecho la comunicación correctamente, y que objeto de mensaje es el artífice de la interrupción. Después dependiendo de si es un envío o una recepción, hacemos cosas distintas.

Si es un envío, limpiamos el bit que indica que ese objeto de mensaje tiene una interrupción, para después proceder a limpiar el bit de que la transmisión se ha realizado.

Si es una recepción, recogemos los datos, activamos la recepción de todos los bits de control, mascara y arbitraje al objeto de mensaje, e indicamos que queremos limpiar el bit de interrupción pendiente. Activamos el bit de datos nuevos y de que se quieren que estos se transmitan al objeto de mensaje. Decimos a que objeto de mensaje nos referimos, y si toda la información se ha recibido correctamente la almacenamos. Todo el control de recepción/envío se realiza mediante el uso de los registros CAN0IF1CR y CAN0IF1CM (ver apartados **3.3.2.1** y **3.3.2.2**). Una vez recogidos los datos, se almacenaran en la cola, para que, cuando sea posible, se envíen mediante la UART al PC.

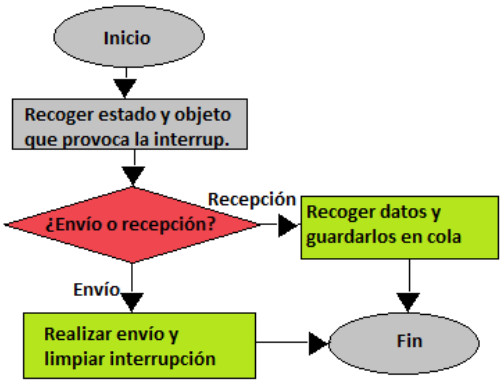


Fig. 5.7: Esquema de interrupción CAN

5.3.4 Cola para almacenar tramas y enviarlas por la UART

Para poder almacenar toda la información que nos llega se ha realizado un buffer de tramas. Su cometido es que, cuando haya una afluencia de datos grande, no se pierda ninguna trama, guardándola en esta cola, y vaciándola conforme se ha explicado anteriormente. Este buffer, situado en la memoria xdata (memoria *flash* externa al microcontrolador, con un espacio de direccionamiento grande de 16 bits, o lo que es lo mismo, 64 KBytes de espacio), almacena el identificador, los datos y la cantidad de estos de cada trama que llega. Para ello, una vez leídos los datos, se almacenan durante la misma interrupción. Su tamaño es de 64 tramas, o lo que es lo mismo, 832 bytes (4 de identificador, 8 de datos y un byte donde se dice la cantidad de datos). Esta estructura, viene definida en el archivo buffer.h, e implementada en el archivo buffer.c. Su estructura es una matriz. En la interrupción, los campos correspondientes al identificador se recogen de los registros CAN0IF1A1 y CAN0IF1A2. Los datos, en caso de no ser una trama remota, se recogen de los registros CAN0IF1DA1, CAN0IF1DA2, CAN0IF1DB1 y CAN0IF1DB2. El número de datos, se recoge del campo LEC en el registro CAN0IF1MC.

El envío de las tramas, se realiza mediante la función trans_byts(), donde con la función sacar, extraemos la primera trama y la enviamos por partes. Primero, el identificador, después el número de datos, y por último, si no es remota, se envían los datos. Esta transmisión se hace byte a byte. Si la trama es estándar se envía solo los dos primeros bytes del identificador, y si no es así, se envían todos. La comprobación para saber si es remota se hace mirando el bit DIR.

A la hora de enviar las tramas, si estas son estándar, se les somete a un tratamiento especial antes de enviarlas, que es el siguiente:

1. Se eliminan los dos primeros bits de la parte baja (la parte baja corresponde a CAN0IF1A2L).
2. Desplazamos la parte baja dos bits a la izquierda.
3. Ponemos a 0 los dos primeros bits de la parte baja.
4. Copiamos los dos primeros bits de la parte alta (que corresponde a CAN0IF1A2H), desplazándolos para que sean los bits 7 y 6.
5. Asignamos a una variable auxiliar el contenido de los tres últimos bits de la parte alta (correspondientes a Dir, Ext y MsgVal).
6. Desplazamos la parte alta dos posiciones a la izquierda, para eliminar los bits que han pasado a la parte baja.
7. Se elimina de la parte alta todo aquello que no sea identificador.
8. Se copia el contenido de aux en la parte alta, para tener los bits de ‘control’.

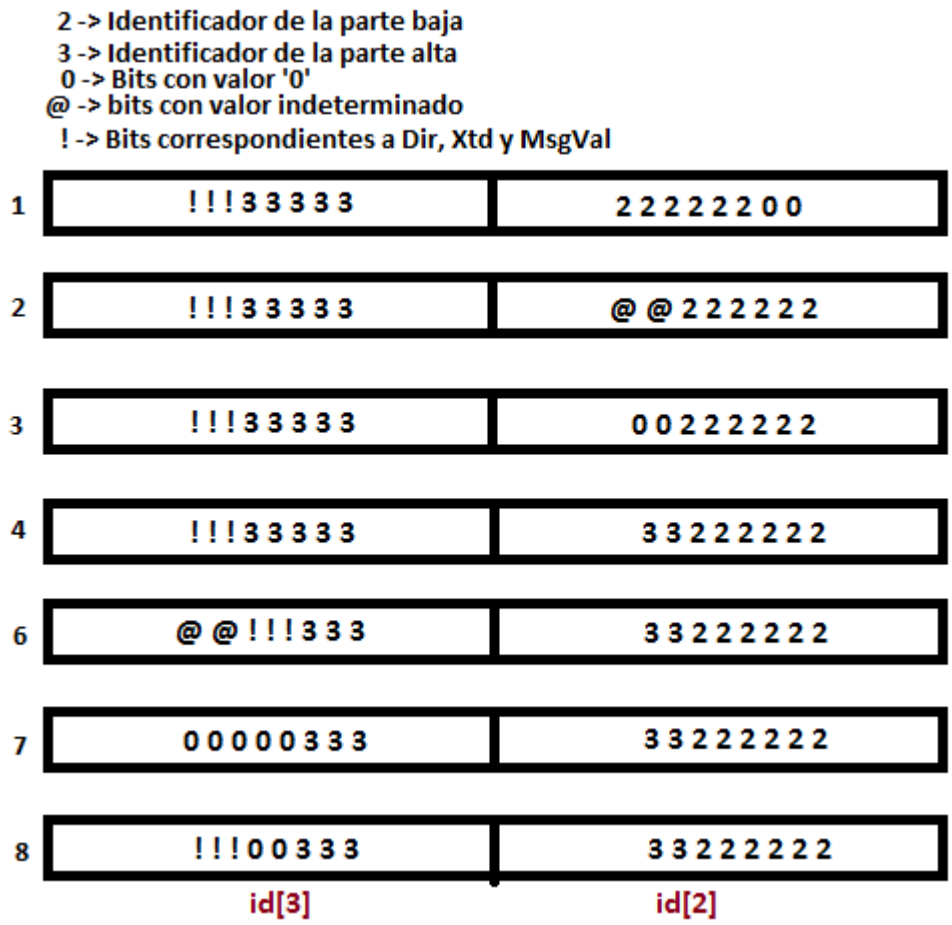


Fig. 5.8: Explicación gráfica de la trasformación en el identificador de trama estándar antes de enviarse al PC

Todo esto se hace para eliminar los dos bits inútiles que se han recogido en la interrupción (bits 0 y 1 de CAN0IF1A2), y preparar la trama para enviarla en un formato sencillo de tratar en el PC.

6. Monitor en el PC (Configuración CAN y envío y recepción)

Para poder acceder a todas las opciones de configuración que nos ofrece el monitor CAN, se ha desarrollado un programa en el PC que se comunica con el microcontrolador. Este programa es el que se comunicará mediante el puerto serie con el controlador, enviando y recibiendo bytes dependiendo de lo que se necesite.

En este apartado, describiremos la funcionalidad de la parte que nos interesa, explicando cada uno de los componentes que tenemos en los formularios (las clásicas ventanas de Windows), tanto visual como funcionalmente. Este punto se dividirá en cuatro partes, correspondientes a las ventanas que usamos para configurar: ventana principal, ventana de configuración del puerto, ventana de configuración CAN y envío en la ventana del *sniffer*.

6.1 Ventana principal

La ventana principal no tiene mucho misterio. Básicamente nos permite acceder a las distintas funcionalidades: configuración del puerto (en rojo), configuración CAN (en naranja) y monitor o *sniffer* (en verde), presentado en menús desplegables, y donde pinchado en los elementos, se nos abre el formulario correspondiente.

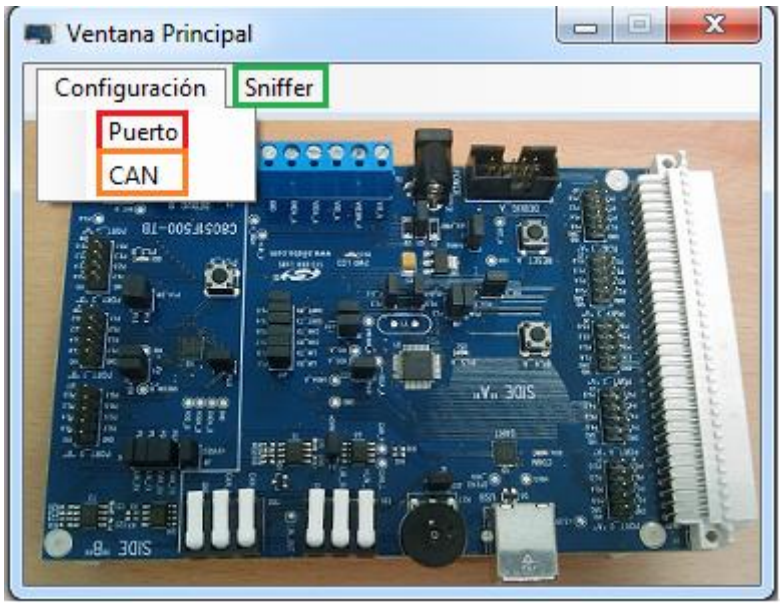


Fig. 6.1: Ventana principal

6.2 Ventana de configuración del puerto

La ventana que nos permitirá definir la comunicación con el microcontrolador, será esta:

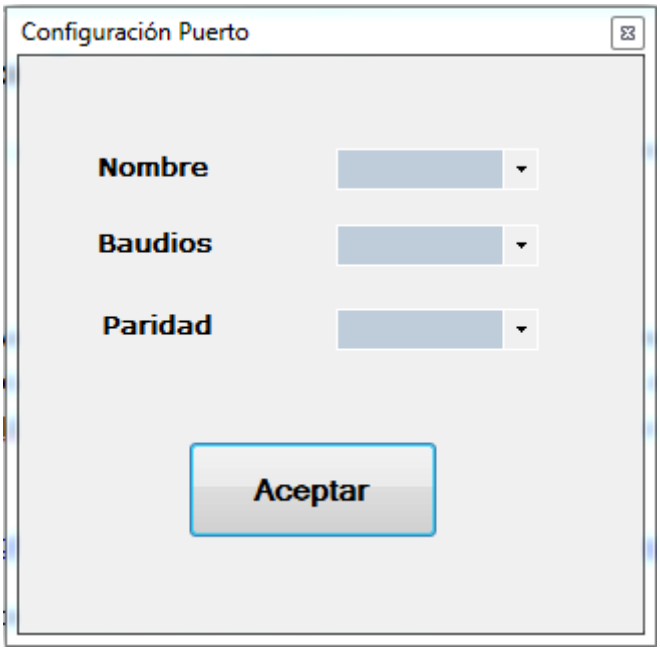


Fig. 6.2: Ventana de configuración del puerto

Son tres desplegables que nos ofrecen las posibilidades de configuración que deseemos. Al presionar el botón **Aceptar**, esta configuración se almacena internamente mediante la clase variables. Posteriormente, cuando se necesite abrir el puerto, se hará con los datos especificados aquí. Si no rellenamos todos los campos, saltará una excepción, la cual mostrará un mensaje con el motivo del error, y hasta que no rellenemos todos los o cerramos la ventana, no se nos deja avanzar.

Si al querer transmitir una configuración o un mensaje, o abrir el puerto para escuchar, aún no se ha hecho la configuración, una ventana emergente nos avisará de ello y nos mostrará esta ventana.

6.2.1 SerialPort()

Los datos que se envíen y reciban a y desde el programa, se harán mediante el puerto serie. Para comunicarnos mediante este puerto, utilizaremos el objeto SerialPort, definido en System.IO.Ports. Las funciones de esta clase, están descritas en el MSDN de Microsoft, en:

http://msdn.microsoft.com/es-es/library/system.io.ports.serialport.aspx

Nosotros utilizaremos básicamente cuatro funciones:

Abrir Puerto: Antes de ejecutar la función ***Open***, necesitaremos introducir tres campos, los cuales serán el nombre del puerto (COMx, donde x es el número del puerto), mediante la propiedad *PortName*, la velocidad en baudios, mediante la propiedad *BaudRate*, y la paridad, mediante la propiedad *Parity*. Después, se los asignaremos al objeto SerialPort, y entonces, se podrá abrir el puerto. Estos campos, como hemos visto anteriormente, los tendrá que dar el usuario en la ventana de configuración del puerto. En caso de no poder abrir el puerto especificado, se nos mostrará una ventana emergente avisándonos de ello. Esto significa que no se han introducido los datos correctos o que no hay nada conectado en ese puerto.

Cerrar puerto: La función ***Close***, cierra el puerto al cuál referencia el objeto.

Leer: Usaremos el método ***Read(byte [], Int32, Int32)*** para leer varios bytes del búfer de entrada de *SerialPort* y escribirlos en una matriz de bytes en la posición de desplazamiento especificada. Esta función se utilizará de distintas maneras según la ventana desde la que se llame, y la función que se especifique.

Escribir: El método ***Write(byte [], Int32, Int32)*** escribe un número especificado de bytes en el puerto serie utilizando los datos de un búfer. A diferencia del caso anterior, esta función siempre se usará para la configuración de recepción y para enviar tramas, y vendrá precedida siempre de la función *Read*, para la sincronización con el módulo CAN. Para monitorizar nunca se utiliza.

Otras propiedades menos importantes también se utilizan. Por ejemplo, **isOpen** nos indicará si el puerto está cerrado o no, cuando vayamos a abrir el puerto sin que el botón que lo hace lo especifique claramente, como los botones **Aceptar** y **Aplicar** de la ventana de configuración CAN.

6.3 Ventana de configuración CAN

La ventana de configuración CAN es la que nos permite acceder a todas las funcionalidades que han sido programadas en el monitor. Consta de varias *group box* en las que podemos variar el modo de funcionamiento, recibir tramas estándar, extendidas o de los dos tipos, modificar los registros Bit Timing y BRPE, recibir tramas con identificadores específicos (hasta 5), y enviar una trama a través de ella, ya sean de datos o remotas.

Al presionar el botón **Aceptar**, se enviará toda la información y se cerrara la ventana. Al presionar el botón **Aplicar**, se hará lo mismo que con el botón Aceptar, pero sin cerrar la ventana. Por último, el botón **Cancelar** cerrara la ventana sin más.

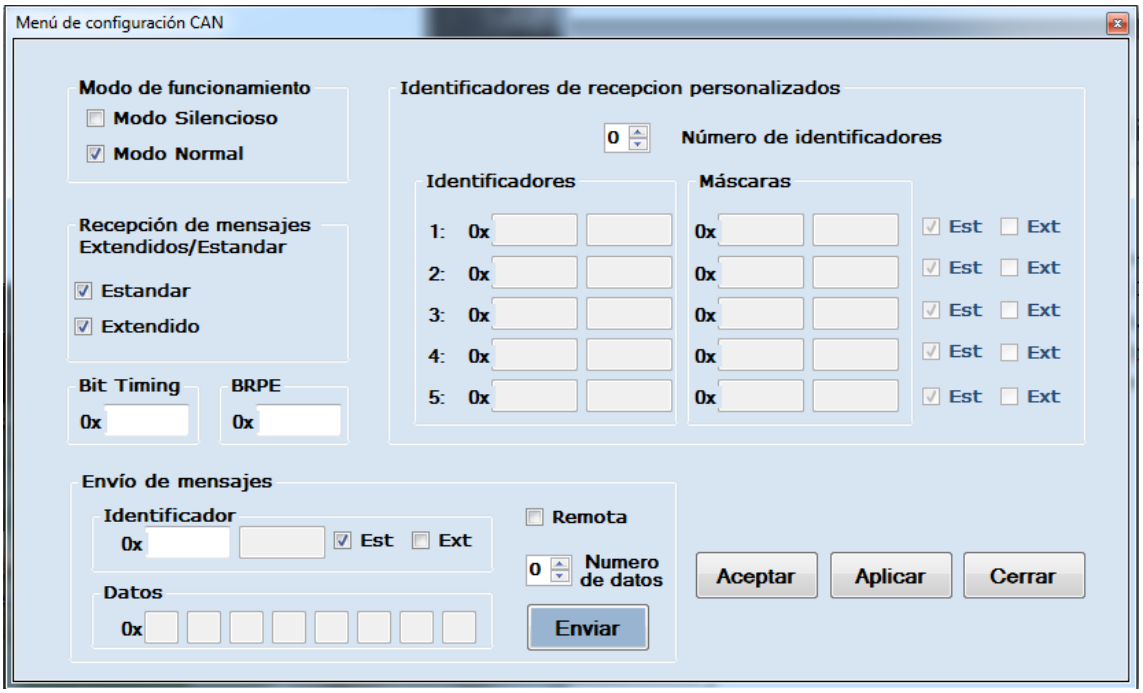


Fig. 6.3: Ventana de configuración CAN

6.3.1 Objeto variablesCAN

Cuando presionemos el botón Aceptar o Aplicar, para facilitar el uso al usuario y para mantener la consistencia del programa, todas las configuraciones que se han cambiado, se almacenan en un objeto llamado variablesCAN. Este objeto dispone de varios campos:

Nombre	Tipo de dato	Observaciones
Dato	Byte	Almacena el valor del byte de opciones al enviar
Ext	Bool	Almacena el estado del bit EXT del byte de opciones
Est	Bool	Almacena el estado del bit EST del byte de opciones
Timing	Byte []	Almacena el valor del registro Bit Timing
BRPE	Byte []	Almacena el valor del registro BRPE
numMasc	Byte	Almacena la cantidad de ident/masc personalizados
Iden	Byte []	Almacena el valor de los identificadores personalizados
Masc	Byte []	Almacena el valor de las máscaras personalizadas
EstExtMasc	Bool []	Almacena si los ident/masc personalizados son EST o EXT
Modificado	Bool	Almacena si los datos se han modificado desde el inicio

Tabla 6.1: Atributos del objeto variablesCAN

Todos los atributos son privados, y se acceden mediante sus correspondientes métodos get y set. Cuando abrimos otra vez la ventana de configuración CAN, todos sus atributos estarán inicializados como estaban anteriormente.

6.3.2 Modo de funcionamiento

En esta sección simplemente se activa uno de los dos modos de funcionamiento, no siendo posible activar los dos. Cuando pulsemos Aceptar o Aplicar, el byte de opciones se modificará dependiendo de que *check box* esté activado, y se almacenará en la estructura de variablesCAN el valor establecido (1 para modo normal, 0 para modo silencioso, o true o false en variablesCAN).

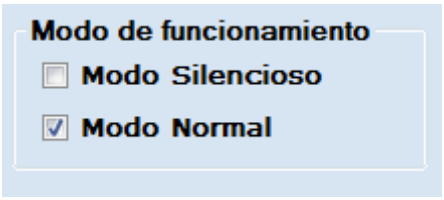


Fig. 6.4: *Group box* de "Modo de funcionamiento" con el modo normal activado

6.3.3 Recepción de mensajes Extendidos/Estandar

Tiene un funcionamiento similar al del "Modo de funcionamiento", solo que aquí sí que podemos activar los dos a la vez. Se desactiva cuando seleccionamos 1 o más identificadores personalizados. Cuando pulsamos Aceptar, modificará los bits EXT y EST del byte de datos y el estado del objeto variablesCAN dependiendo de los *check box* activados.

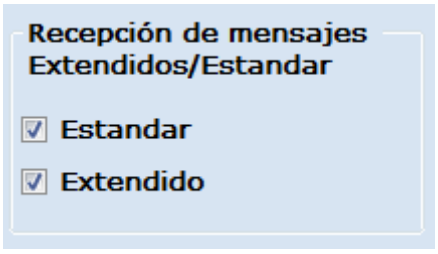


Fig. 6.5: Group box de Recepción de mensajes Extendidos/Estandar, con la recepción activada de los dos tipos de tramas

6.3.4 Registros *Bit Timing* y *BRPE*

En estos dos cajas de texto *(Text box)*, se introducirán directamente los valores deseados en estos registros. Para calcular los valores que queremos, se recomienda la lectura de los aparatados **3.3.1.4**, **3.3.1.6** y **3.4.11**). De esta manera, se facilita la transmisión de la información, y también, se hace más simple al usuario. El contenido ha de ser introducido en caracteres hexadecimales, y se deben escribir los cuatro caracteres. Si no es así, saltará una excepción avisando de que hemos hecho mal, y la información no se enviará al microcontrolador en el momento en el que pulsemos Aceptar o Aplicar.

En el caso de no introducir nada, se tomaran los valores por defecto, es decir, para bit Timing 0x1402, y para BRPE 0x0000.

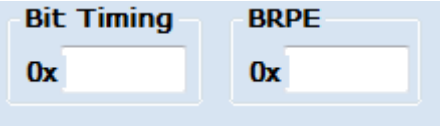


Fig. 6.6: Campos de Bit Timing y BRPE sin nada introducido.

6.3.5 Identificadores de recepción personalizados

Para poder recibir paquetes específicos, en la Interfaz Gráfica de Usuario (IGU a partir de ahora), se ha diseñado un conjunto de *Text box*, *Check box*, y un contador numérico.

El contador numérico, inicializado a 0, indicará cuantos identificadores personalizados queremos poner. Conforme añadamos o quitemos cantidad de este contador, se irán desbloqueando los *text box* oportunos y sus check box correspondientes, para evitar fallos en la introducción de los datos. Si este contador está a 0, se utilizará la información de la "Recepción de mensajes Estandar/Extendidos", mientras que si se indica un número mayor que 0, se bloquearán los campos de ese *Group box*, y los bits EST y EXT del byte de datos se pondrán a cero. El número máximo de este contador es 5, que corresponde con el valor máximo de identificadores personalizados que se han programado en el controlador del C8051F500.

Los *check box* tienen la función de especificar si el identificador personalizado es estándar o extendido, bloqueando los *text box* correspondientes en caso de indicar que son estándar para no dar lugar a posibles errores por parte del usuario.

Los *text Box* tienen un funcionamiento idéntico al de los correspondientes a temporizadores (BT y BRPE). Sin embargo, aquí se permite la introducción de 32 o 16 bits (según sea extendido o no), en cada identificador/máscara. Como sabemos, esto no es posible, pues los identificadores estándar son de 11 bits y los extendidos de 29. Por lo tanto, antes de enviar la información, y de manera absolutamente transparente al usuario, se eliminan los bits sobrantes (como se observa en los campos de máscara de la figura 6.6), que en el caso de los identificadores estándar son los 5 bits más altos, y en los extendidos los 3 bits más altos. La próxima vez que se abra esta ventana, aparecerán los valores corregidos, puesto que se almacenan corregidos en variables CAN.

Cuando se pulse el botón Aplicar o Aceptar, el valor del contador numérico modificará el campo R/T del byte de opciones dependiendo de su cantidad. Los valores de *los Text box* y de los *Check box* se enviarán tal y como está programado recibirlos en el programa del microcontrolador.

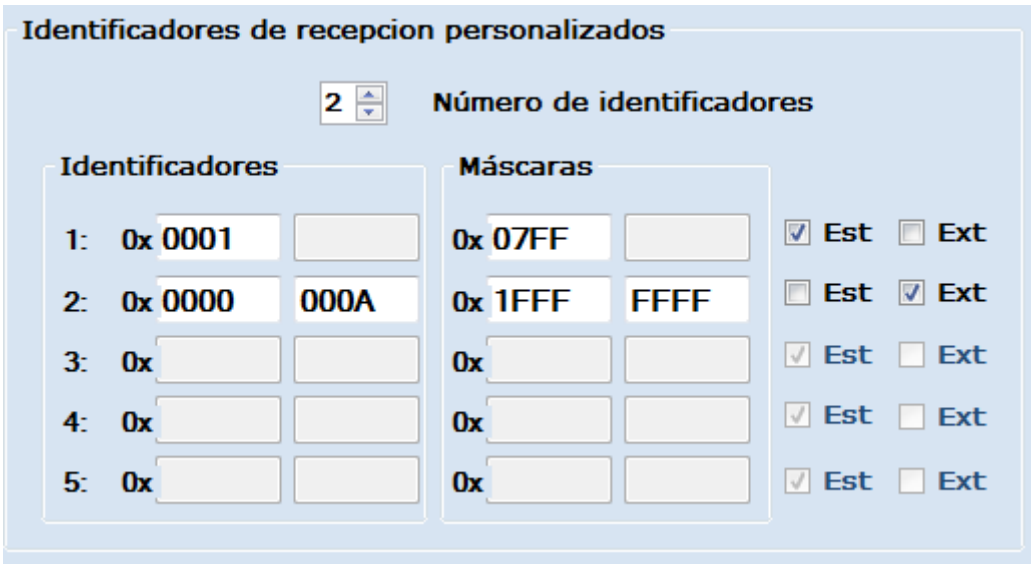


Fig. 6.7: Group box de Identificadores de recepción personalizados. Los valores mostrados son ejemplos de configuración, para mostrar cómo se comportan las distintas partes que la componen.

6.3.6 Envío de mensajes (Ventana de configuración CAN)

Esta sección va absolutamente a parte respecto de las anteriores. Esto es así debido a que la información que contiene no se tiene en cuenta como las demás, cuando se presionan los botones Aceptar y Aplicar, sino que se tendrán en cuenta solo cuando se presiones el botón **Enviar**, que forma parte de esta sección.

Se compone de varios *Text box* (2 de identificador y 8 de datos), varios *Check box* (trama extendida o estándar, y otro para indicar si es una trama remota o no), y un contador numérico, que indica el número de bytes de datos que se van a enviar.

El contador numérico indica la cantidad de bytes de datos que contendrá la trama a enviar. Como sucedía con los identificadores personalizados, aquí limitamos el número de *Text box* dependiendo de la cantidad indicada, evitando fallos por parte del usuario. Este contador se enviará al microcontrolador como información de los bytes que tiene la trama para que se introduzcan en el campo DLC del objeto de mensaje.

Si se activa el *Check box* de trama remota, todos los Text box se desactivarán, puesto que en una trama remota no se envían datos, independientemente de lo que se diga en el contador numérico de "número de datos".

El campo identificador funciona exactamente igual que en los identificadores personalizados, solo que aquí no se define máscara. Todas las correcciones y la funcionalidad de los *Check box* "Est" y "Ext" es exactamente igual. Cuando se selecciona trama estándar, solo se pueden rellenar 16 bits de identificador (cuatro caracteres en hexadecimal), y si, por el contrario, se selecciona trama extendida, debemos de rellenar 32 bits (ocho caracteres en hexadecimal, divididos en cuatro y cuatro).

Los *Text box* de datos tienen pequeñas modificaciones respecto a todos los demás. En estos, la longitud máxima es de dos caracteres hexadecimales, que corresponden a un byte, ya que este es el tamaño que tiene un dato en una trama de datos CAN.

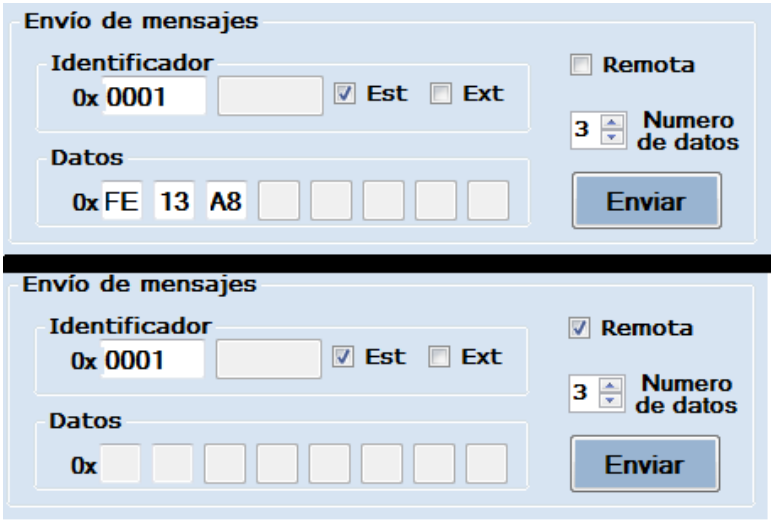


Fig. 6.8: Envío de trama de datos y envío de trama remota

6.3.6.1 Acciones al pulsar el botón Enviar

Cuando presionamos el botón Enviar, lo primero que se hace es, dependiendo de si lo que se envía es una trama remota o de datos, construir el byte de opciones (0xC0 para datos, 0xE0 para remota), y se introduce en un vector de bytes para enviar. Una vez recogido el byte de datos, se introduce en el vector si es una trama estándar (0x00) o extendida (0x01). A partir de ahí se recoge el identificador byte a byte, y se va introduciendo en el vector (teniendo en cuenta si es extendido o no). En este punto es donde se revisa el contenido de los *Text box*, provocando una excepción en el caso de que el identificador no sea correcto. Después, se introducen en el vector los bytes de datos, en el caso de que sea trama de datos, uno a uno y se montan en el vector.

Con toda la información recopilada, se abre el puerto (de manera transparente al usuario si los datos de conexión ya han sido introducidos), y mediante el sistema envío/repuesta, se van enviando los bytes del vector uno a uno, donde los recogerá el microcontrolador, los decodificará y realizará el envío. En caso de fallo en el envío, se cancelará este.

Por último, se cerrara el puerto y si todo ha ido bien, se mostrará una ventana advirtiéndolo.

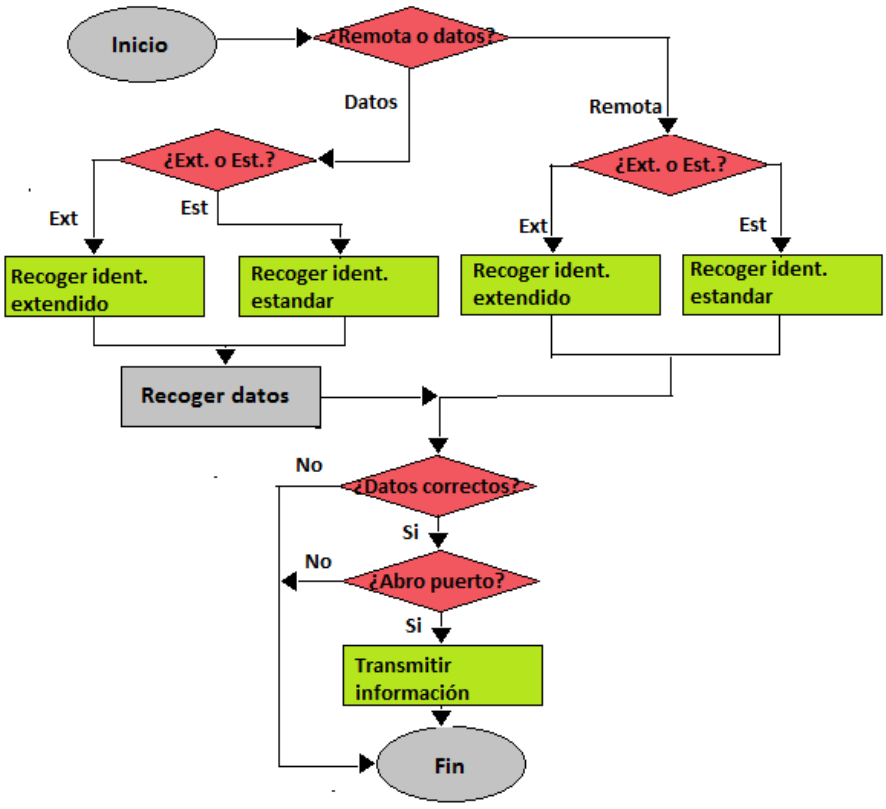


Fig. 6.9: Diagrama resumido de acción de enviar

6.3.7 Acciones al pulsar el botón Aceptar o Aplicar (enviarConf())

Una vez puesta la configuración deseada, es necesario para que se aplique presionar uno de estos dos botones. Cuando se hace, se llama a la función enviarConf(). El cambio entre Aplicar o Aceptar es que si se acepta, se cierra el formulario después de realizar la comunicación con el microcontrolador, mientras que si se acepta, solo se realiza la comunicación, y se deja la ventana abierta.

En el momento que comienza esta rutina, se va comprobando sección a sección las casillas marcadas y los *Text box* escritos. Primero se comprueba el modo de funcionamiento, modificando el byte de opciones según se escoja silencioso o normal (0 o 1). Después se comprueba si se tienen marcadas las casillas Estándar y Extendidas de recepción de mensajes Estándar/Extendidas, modificando el byte de opciones convenientemente.

Después, se comprueba si se ha escrito algo en el *Text box* de BRPE, y en caso afirmativo, se modifica el byte de opciones. Después se recogen los bytes y se introducen en un vector a enviar, de manera exacta a como hace el botón Enviar. Seguidamente, se comprueba el contenido del *Text box* de *Bit Timing*, y se hace lo mismo que con BRPE.

Por último, se comprueba el valor del contador numérico de la recepción personalizada de identificadores, y se van recogiendo los valores de los identificadores y las máscaras teniendo en cuenta si son extendidas o estándar. Se introducen en el vector de envío, primero 0x00 si es estándar y 0x01 si es extendido, luego los bytes corregidos del identificador y por último los bytes corregidos de la máscara. Se repite el proceso tantas veces como identificadores haya. Al finalizar, si hay más de 0 identificadores personalizados, se limpian los bits EST y EXT del byte de opciones y se modifica los bits R/T dependiendo de la cantidad.

Una vez hecho todo esto, y si no ha saltado ninguna excepción al ser los datos erróneos, se introduce el byte de opción en la primera posición del vector de bytes para enviar, y, como se hace al enviar una trama, se abre el puerto de manera transparente para el usuario en el caso de que este hubiera introducido los datos correspondientes y que estos sean correctos (si no tendrá que hacerlo antes de realizar la transmisión).

Después, se hace la trasmisión igual que en el evento del botón Enviar, y si todo ha sido correcto, se finaliza el método con el cierre del puerto, también de manera absolutamente transparente para el usuario.

Mientras se recogen los datos y tras ser validados como correctos en caso necesario, se almacena toda la información en el objeto vCAN, de tipo variablesCAN, para que en

posteriores cambios se muestre la configuración actual cuando se vuelva a abrir esta ventana.

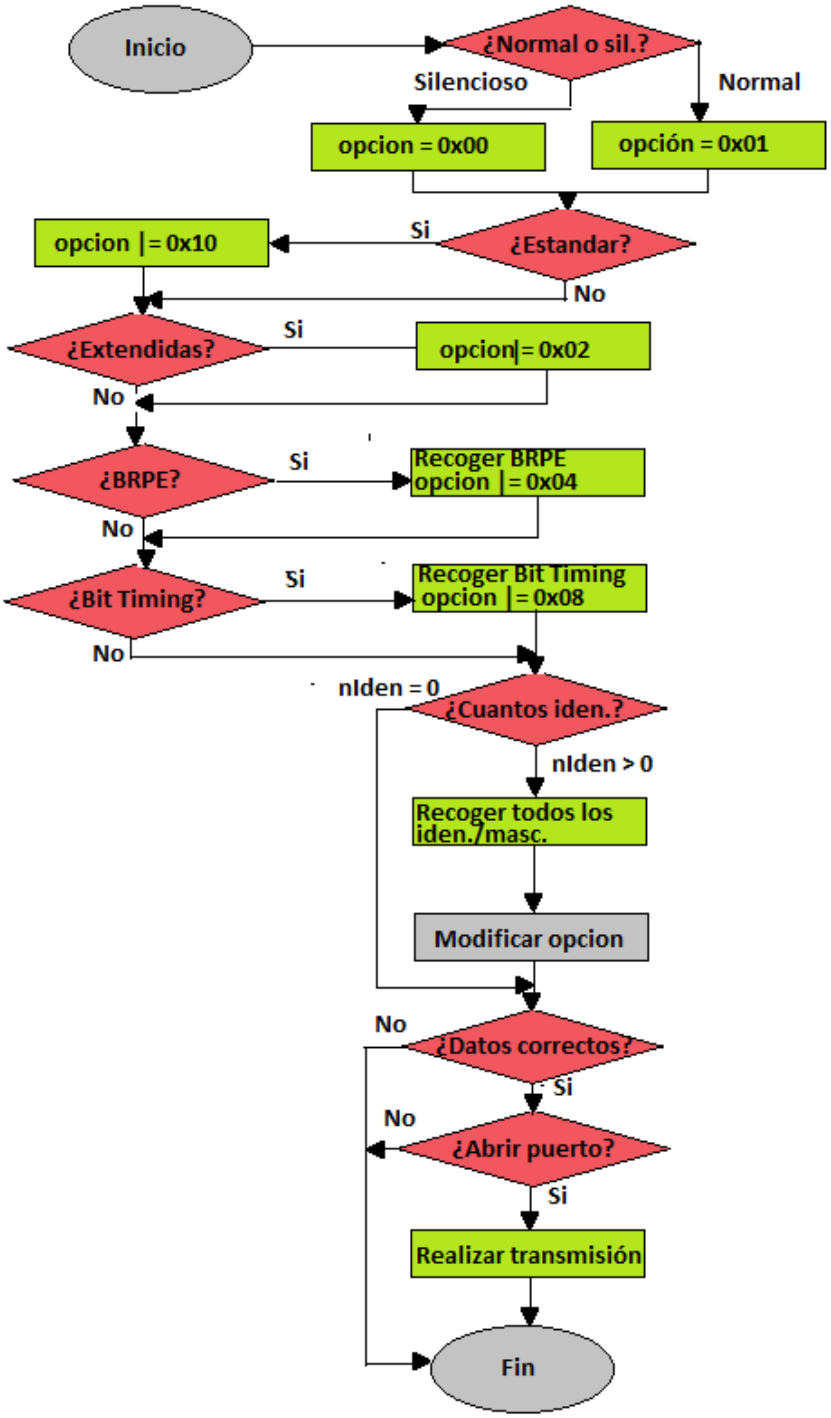


Fig. 6.10: Diagrama resumido de enviarConf()

6.4 Envío en la ventana de sniffer

En la ventana de sniffer se ha añadido la funcionalidad de enviar datos. Su funcionamiento es exacto al explicado en el punto **6.3.6** (de hecho, la apariencia es exactamente la misma). Sin embargo, este formulario tiene el inconveniente de que sirve para visualizar datos, y por lo tanto, sensible al modo de comunicación enviar/respuesta utilizado.

El problema radica en que esta ventana tiene programado un evento el cual se activa al recibir un byte por el puerto serie configurado en el objeto SerialPort, preparado para mostrar la información y que utiliza un protocolo distinto.

Para evitarlo, mientras hay comunicación en el puerto serie para enviar una trama CAN (de datos o remota), se desactiva el evento de recibir datos, y una vez terminado el envío, se vuelve a activar. De manera similar funciona, por ejemplo, el botón limpiar mientras llegan datos. El evento pulsar (click) de este botón no funciona mientras hay datos escribiéndose. De esta manera evitamos errores de forma preventiva.

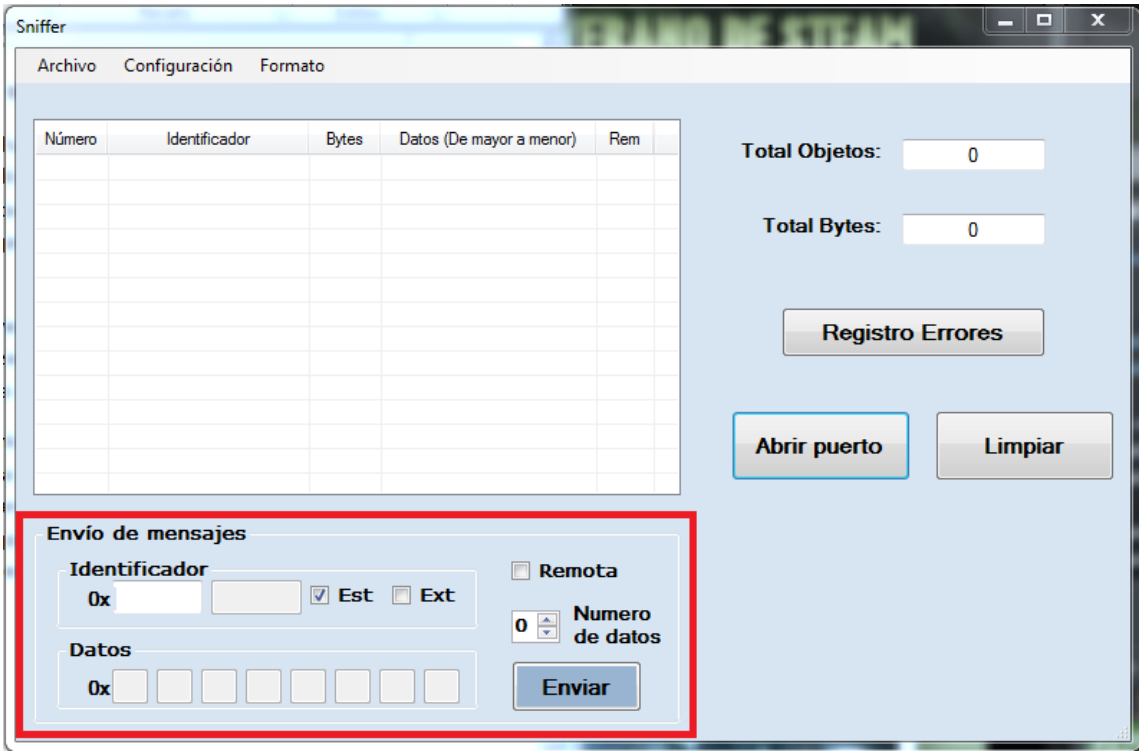


Fig. 6.11: Ventana de sniffer, con la parte dedicada al envío resaltada en un marco rojo

7. Conclusiones

Para la redacción de esta memoria, se ha intentado seguir un enfoque ascendente del problema. Es decir, en un principio hemos comentado que partes forman el problema y soluciones existentes. Luego, hemos explicado el bus de la manera más amplia pero resumida posible, aunque suene a contradicción, pues se ha intentado explicar todo lo que utilizamos y todo el protocolo, pero de manera sencilla. Después las herramientas que íbamos a utilizar, y por último, una explicación lo más completa posible de la solución propuesta.

Ver que es una red industrial nos ha ayudado a ver la difusión que puede tener este tipo de bus, y hacia donde está dirigido, además de sus problemáticas y las funciones que ha de cumplir en un entorno productivo real. La pequeña explicación de que es un sistema empotrado, nos ha ayudado a ver sobre qué sistema vamos a trabajar, y qué son los nodos CAN, ya que estos son sistemas empotrados.

Antes de meternos de lleno en el protocolo CAN, hemos visto qué es un monitor, o *sniffer*. Qué soluciones ofrecen las compañías tanto para monitorizar redes TCP/IP como redes CAN, nos ha ayudado a saber qué necesidades hemos de satisfacer, y después, con el material disponible, adoptar el número de funciones mayor posible.

En la explicación del protocolo, la información ha sido sacada y traducida por el autor de los organismos oficiales, dando una descripción de este dirigido a una persona que quiera diseñar una red CAN. La explicación del núcleo CAN has sido una traducción directa del manual del usuario de Bosch, explicando las partes que nos atañen, así como explicaciones más críticas hacia un posible usuario final de este monitor.

La explicación del monitor en las dos partes ha sido una explicación profunda de los algoritmos, del funcionamiento y del protocolo implementado, sin entrar en código más allá de pequeños apuntes, necesarios para el entendimiento del programa. Es más, se ha intentado ilustrar mediante diagramas todos los comportamientos más difíciles de entender, de manera sencilla, y capturas de pantalla para entender mejor como utilizar el programa de PC.

Este proyecto ha sido un trabajo conjunto junto con Andrés Martínez Mas, que lo continuará mejorando la parte del monitor. Por lo tanto, este proyecto no está finalizado a la entrega de esta memoria, detalle a tener en cuenta por si se quiere utilizar basando la información y el uso solo en esta memoria. Se ha intentado tener un producto usable en la configuración, pero para poder utilizar la parte de monitorización, se debe utilizar el proyecto final, que se está finalizando en estos momentos.

Durante la realización del proyecto, que ha llevado varios meses, se ha aprendido la programación de sistemas empotrados, y se han puesto en práctica todo lo aprendido en programación, tanto en lenguajes secuenciales como en orientados a objetos, poniendo en práctica todo lo aprendido. También se han aplicado técnicas como la modularización, el *refactoring* o el ciclo de vida de programación extrema, con ciclos de vida cortos y objetivos plausibles y claros a corto plazo.

Fig. 7.1: Programando el microcontrolador

Se ha echado en falta más tiempo a la hora de realizar este proyecto, pero las necesidades de acabarlo de una forma pronta han dejado sin implementar algunas funcionalidades interesantes. Sin embargo, todas las configuraciones críticas y que le confieren la posibilidad de ser usado en un entorno real han sido implementadas. En la IGU se ha intentado unir simplicidad de uso con claridad.

Para concluir, si en un futuro de desea continuar este proyecto, será necesario mejorar el sistema de seguridad en la transmisión PC-microcontrolador, para poder añadir un sistema que asegure que los datos introducidos son correctos, además de poder utilizar todos los registros disponibles (no solo los de la interfaz 1), de manera que se puedan personalizar más identificadores, y añadir otros sistemas de funcionamiento (loop-back, loop-back combinado con silencioso y modo básico). También un mecanismo de inicio-parada y un reseteo sería conveniente añadir, para poder manejar el controlador sin necesidad de acceder físicamente a él.

8. Bibliografía

Universidad de Valencia. Sistemas industriales Distribuidos. Tema 3: Redes de comunicación industriales. Alfredo Rosado. Consultado el 11 de Julio de 2012.

Universidad Politécnica de Madrid. Sistemas empotrados ubicuos y móviles. Master Universitario en Ingeniería Informática. Juan Zamorano. http://www.datsi.fi.upm.es/docencia/SEUM/ Consultado el 2 de Julio de 2012

Universidad de Murcia. Sistemas Embebidos. Tema 1: Introducción a los sistemas embebidos. Benito Úbeda Miñarro. http://ocw.um.es/ingenierias/sistemas-embebidos/material-de-clase-1/ssee-t01.pdf Consultado el 2 de Julio de 2012

Página oficial de Nagios. http://www.nagios.org/ Consultado el 4 de Julio de 2012.

Página oficial de Wireshark. http://www.wireshark.org/ Consultado el 4 de Julio de 2012.

Instituto Nacional de Tecnologías de la comunicación. Análisis de tráfico con Wireshark. Borja Merino Febrero. http://cert.inteco.es/extfrontinteco/img/File/intecocert/EstudiosInformes/cert_inf_seguridad_analisis_trafico_wireshark.pdf Consultado el 5 de Julio de 2012.

Página oficial de Microchip. CAN BUS Analyzer Tool. http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en546534 Consultado el 6 de Julio de 2012.

Página oficial de Ditecom Design. Analizador de Bus CAN/CAN OPEN. http://www.ditecom.com/analizadorCAN/analizador-CAN.shtml *Consultado el 6 de Julio de 2012.*

Página personal de la Universidad Politécnica de Valencia de Alberto Ramis Fuambuena. Bus CAN. http://personales.alumno.upv.es/alrafua/asignaturas/SES/Buses/CAN/can.html *Consultado el 8 de Julio de 2012.*

Página oficial de la organización CAN in Automation/CAN open. Controller Area Network (CAN). http://www.can-cia.org/index.php?id=systemdesign-can *Consultado el 9 de Julio de 2012.*

C_CAN: Manual del usuario. Revisión 1.2 6 de Junio de 2000. Robert Bosch GmbH. Automotive Equipment Divison B. Development of integrate Circuits (MOS).

Manual del C8051F50x/F51x: Mixed Signal ISP Flash MCU Family. Revisión 0.2. Julio de 2008. Silicon Laboratories Inc.

Anexo

a. LISTA DE FIGURAS

b. LISTA DE TABLAS

EOF