UNIVERSIDAD POLITECNICA DE VALENCIA

Escola Tècnica Superior d'Enginyeria Informàtica

etsinf

UNIVERSIDAD POLITÉCNICA DE VALENCIA

ESCUELA TÉCNICA SUPERIOR DE INFORMÁTICA APLICADA

PROYECTO FINAL DE CARRERA

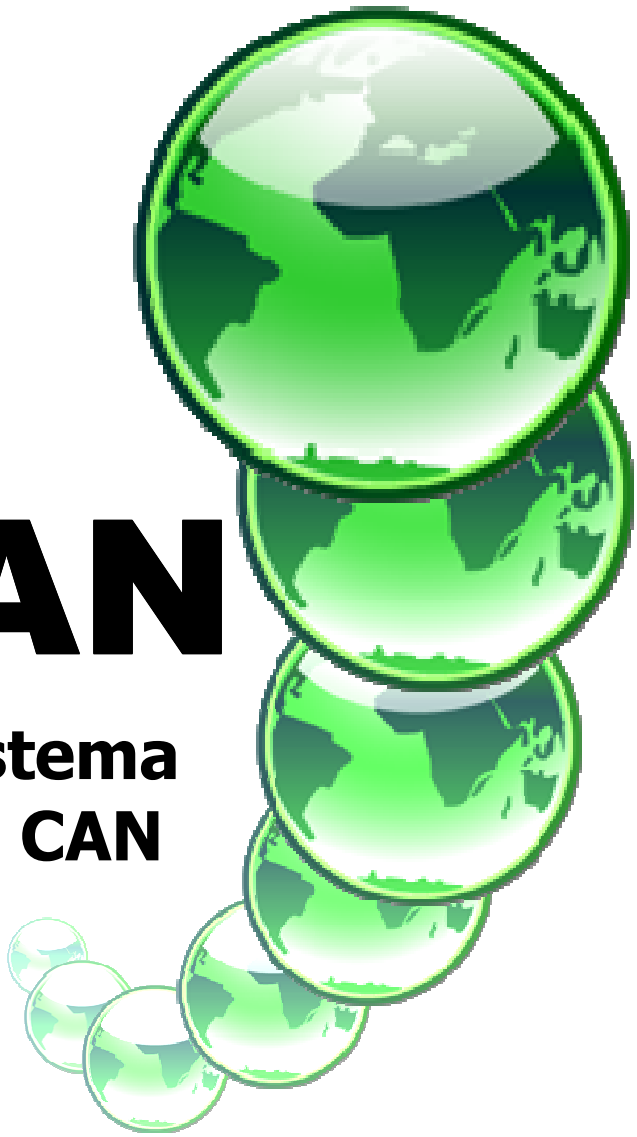
TÍTULO: Software para un sistema domótico basado en CAN

AUTOR: Salvador Arnal Julián

DIRECTOR: Lenin Guillermo Lemus Zúñiga

FECHA: Noviembre - 2010

TITULACIÓN: Ingeniería Informática



DOMCAN

Software para un sistema Domótico basado en CAN

Agradecimientos:

A *Ella*.

I ÍNDICE

II ÍNDICE DE FIGURAS

III ÍNDICE DE TABLAS

CAPÍTULO 1. INTRODUCCIÓN Y OBJETIVOS

1.1. Motivación

Diseñar e implementar el software para un sistema domótica basado sobre el protocolo CAN.

El principal motivo para realizar este proyecto es para completar el PFC realizado por mi compañero y amigo Pere Joan Antoni Chordá, el cual se centra en el diseño del hardware sobre el que se utilizar este programa.

1.2. Objetivos del PFC

Diseño e implementación del SW necesario para sacar el máximo rendimiento de las placas diseñadas para un sistema domótico basado en el protocolo CAN

En este contexto, los objetivos de este proyecto final de carrera son:

1. **Implementación del Software de administración**: Diseño e implementación una aplicación que permita la configuración y programación de los nodos sin necesidad de conocimientos informáticos por parte del usuario.

2. **Implementación del Software de los nodos**: Diseño e implementación del firmware que llevará cargado cada nodo

3. **Implementación del Software de pasarela**: Diseño e implementación del firmware que llevara un nodo especial que actuará como pasarela entre el equipo con la aplicación de administración y la red CAN

CAPÍTULO 2. PLANTEAMIENTO INICIAL

2.1. El Nodo Hardware

El nodo hardware fue desarrollado por Pere Joan Antoni Chordá en su Proyecto Final de Carrera [1].

Uno de los objetivos del diseño era que se pudiera instalar dentro de una caja estándar de conexiones de una vivienda, de ahí su reducido tamaño.

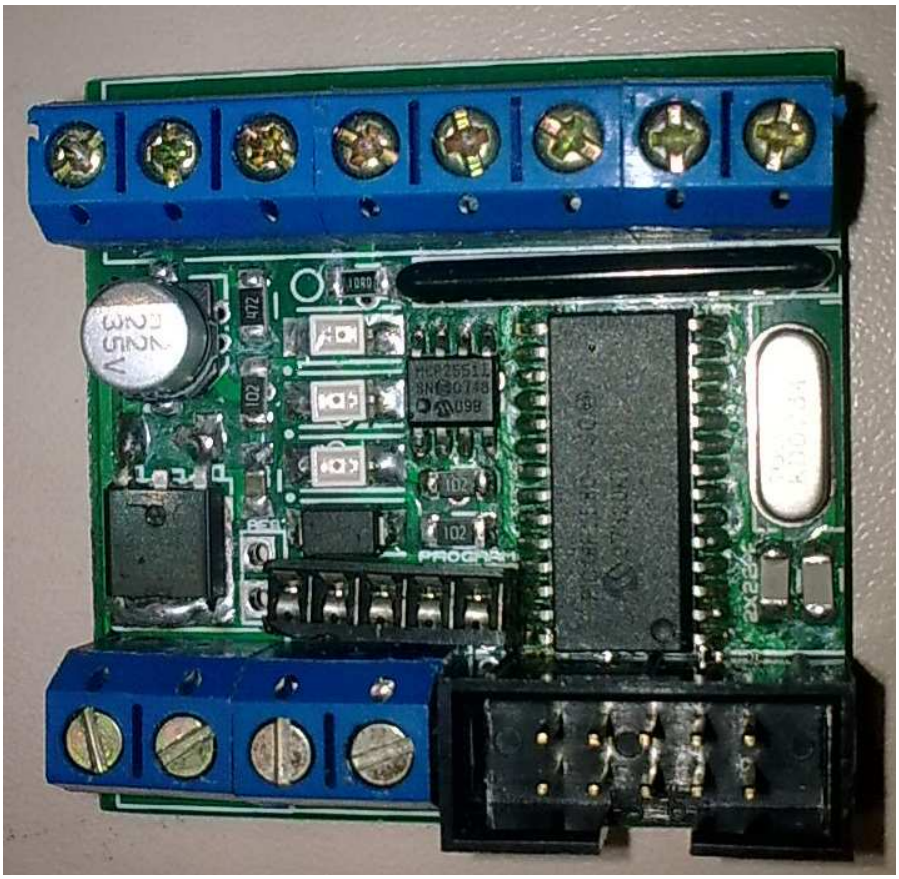


Figura 1: Nodo Hardware

El coste para una tirada de 1000 unidades seria de 10.42 € por nodo, muy por debajo de cualquier solución actual que se encuentre en el mercado.

2.1.1. Chipset

El chipset del nodo está formado por dos circuitos integrados:
Un microcontrolador PIC18F2580 con las siguientes características:

Power Managed Modes:

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

Flexible Oscillator Structure:

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

Special Microcontroller Features:

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

Peripheral Highlights:

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

ECAN Module Features:

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

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

Figura 2: Datasheet PIC18F2580

Un transceiver MCP2551:

Features

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

Package Types

PDIP/SOIC

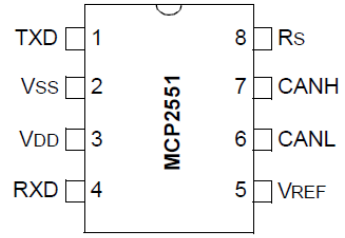


Figura 3: Datasheet MCP2551 (Parte 1)

MCP2551

TABLE 1-1: MODES OF OPERATION

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

TABLE 1-2: TRANSCEIVER TRUTH TABLE

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

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

Figura 4: Datasheet MCP2551 (Parte 2)

2.1.2. Características Generales:

- **Entradas analógicas**: 8 entradas para conectar sensores analógicos. Como por ejemplo sensores de iluminación y temperatura.
- **Entradas digitales**: 8 conexiones para señales de entrada con dos estados, como por ejemplo pulsadores o sensores con salida todo/nada.
- **Salidas digitales**: 8 conexiones para activar y desactivar actuadores, como por ejemplo relés, triacs o dimmers.
- **Salidas analógicas**: 8 conexiones para controlar actuadores, con entradas 0~10Vcc ó 4~20mA.
- **Puerto para bus CAN**: un puerto de comunicación entre nodos.

2.2. La Domótica

Según el diccionario de la Real Academia Española (RAE) el término domótica proviene de la unión de la palabra "domus" (que significa "casa" en latín) y el sufijo "-tica" (esta terminación remite a "automática", y hoy en general induce al significado de "gestión por medios informáticos").

Etimológicamente lo que se entiende por domótica es el conjunto de sistemas capaces de automatizar y controlar las diferentes instalaciones de una vivienda. Los conceptos de automatización y control se asocian al encendido, apagado, apertura, cierre y regulación de aparatos y sistemas de instalaciones eléctricas. En resumen se podría definir como la integración de la tecnología en el diseño inteligente de un recinto, ya sea una vivienda o no.

2.2.1. Dispositivos

La extensión de una solución domótica puede variar desde un único dispositivo que realiza una sola acción, hasta amplios sistemas que controlan prácticamente todas las instalaciones dentro de la vivienda. Los distintos dispositivos de un sistema domótico se clasifican en los siguientes grupos:

- **Controlador**: Gestiona el sistema según la programación y la información que recibe, como si de un microprocesador se tratara.

- **Actuador**: Es un dispositivo capaz de ejecutar y/o recibir una orden del controlador y realizar una acción sobre un aparato o sistema concreto (encendido/apagado, subida/bajada, apertura/cierre, etc.)

- **Sensor**: Es el dispositivo que monitoriza el entorno captando información que transmite al sistema (sensores de agua, gas, humo, temperatura, viento, humedad, lluvia, iluminación, etc.).

- **Bus**: Es el medio de transmisión que transporta la información entre los distintos dispositivos ya sea por un cableado propio, por la redes de otros sistemas (red eléctrica, red telefónica, etc.) o de forma inalámbrica.

- **Interfaz**: Se refiere a los dispositivos (interruptores, conectores, pantallas, etc.) en los que se muestra la información del sistema para los usuarios y/o donde los mismos pueden interactuar con el sistema.

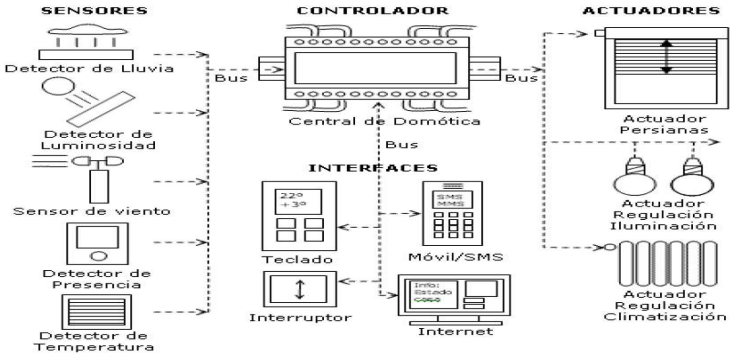


Figura 5: Dispositivos en una red domótica

Hay que señalar que todos los dispositivos de un sistema domótico no tienen porqué estar físicamente separados, sino que varios de ellos pueden estar combinados en un mismo nodo o controlador.

Por ejemplo, un nodo puede estar compuesto por un controlador, varios actuadores y sensores, y diferentes interfaces. Tampoco es estrictamente necesario que los cinco tipos de dispositivos se incluyan en una misma instalación domótica. Hay algunas arquitecturas que no requieren de un bus. Lo mismo se puede decir por el resto de tipos de dispositivos.

2.2.2. La Arquitectura

La arquitectura empleada es una arquitectura descentralizada: existen varios controladores (nodos a partir de ahora), conectados por un bus, que envían información entre ellos y a los actuadores e interfaces conectados a los controladores. Todo ello también según el programa, la configuración y la información que reciben de los sensores, sistemas interconectados y usuarios.

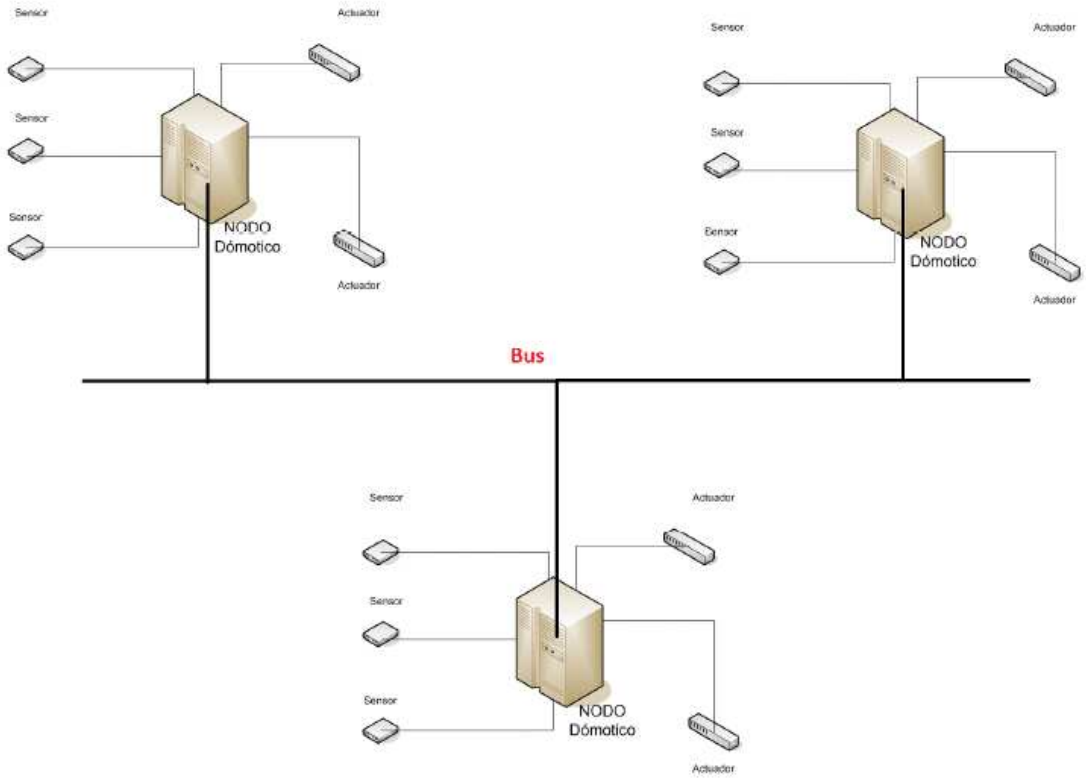


Figura 6: Arquitectura descentralizada

La instalación domótica que se propone estará compuesta por un número definido de nodos inteligentes. Cada nodo abarcará una zona distinta de la vivienda.

De cada nodo colgarán sus respectivos actuadores y sensores que permitirán realizar el control de la vivienda. Aunque es una instalación basada en una arquitectura descentralizada esto nos permitirá de cada uno de los nodos se autosuficiente,

proporcionando a la red domótica mas flexibilidad e independencia de actuación.

Lo más relevante de este diseño es que no existe nodo Maestro, todos los nodos son maestros y esclavos simultáneamente. La información fluirá por el bus y cada nodo tomara las decisiones que crea convenientes en función de los datos que escucha en el bus.

Este modelo de arquitectura seleccionado requiere de un bus para que los nodos se puedan comunicar entre ellos.

2.3. El Bus de Comunicaciones CAN

El protocolo de comunicación CAN fue originalmente desarrollado y especificado en los años 80 por la compañía alemana Robert Bosch en un intento por resolver los problemas de comunicación entre los diferentes sistemas de control de los coches. No es necesario decir que la idea no tardó en pasar de los vehículos al sector de la automatización y control.

CAN está estructurado de acuerdo con el modelo OSI en una arquitectura colapsada de dos capas, esto es, la capa física y la capa de enlace de datos. Como el resto de capas del modelo OSI (red, transporte, sesión, presentación y aplicación) no están especificadas por el protocolo, es necesario diseñarlas. Sobre todo es de vital importancia especificar un protocolo de alto nivel (HLP).

La especificación original de CAN, publicada por Robert BSCH GMBH, se llama Especificación de CAN 2.0-A. La razón por la cual existe una parte A es porque una especificación actualizada de CAN se publicó después por la misma compañía, y contenía tanto la parte A, como una nueva versión de CAN llamada parte B.

La especificación CAN 2.0-B es compatible con la Parte A. La diferencia se localiza básicamente en el formato del encabezado del mensaje del identificador. La especificación CAN 2.0-A define sistemas CAN con un estándar de 11 bits del identificador. En cambio, CAN 2.0-B especifica la trama extendida con 29 bits en el identificador.

La especificación de CAN 2.0 es de dominio público. La mayoría de las implementaciones actuales de HW CAN usan la nueva versión de la especificación, incluso aunque sólo sea de forma pasiva (solo transmite mensajes estándar, pero comprueba si recibe mensajes estándar y mensajes extendidos).

2.3.1. Características principales

A continuación se mostraran algunas de las propiedades fundamentales del CAN. Prácticamente todas ellas suponen una ventaja respecto otros tipos de buses convencionales usados en instalaciones domótica estándar:

1. La comunicación está basada en mensajes y no en direcciones. Un mensaje es diferenciado por el campo llamado identificador, que no indica el destino del mensaje, pero sí describe el contenido del mismo.

2. No hay un sistema de direccionamiento de los nodos en el sentido convencional. Los mensajes se envían según su prioridad.

3. La prioridad entre los mensajes la define el identificador. Se trata de una prioridad para el acceso al bus.

4. Es un sistema multimaestro. Cuando el bus está libre, cualquier nodo puede empezar la transmisión de un mensaje, y el mensaje con mayor prioridad gana la arbitración del bus. (Esta prioridad viene marcada por ID del mensaje).

5. Todos los nodos CAN son capaces de transmitir y recibir datos y varios pueden acceder al bus de datos simultáneamente.

6. Un nodo emisor envía el mensaje a todos los nodos de la red, cada nodo, según el identificador del mensaje, lo filtra y decide si debe procesarlo inmediatamente o descartarlo. Como consecuencia el sistema se convierte en multicast en el cual un mensaje puede estar dirigido a varios nodos al mismo tiempo.

7. Gran fiabilidad y robustez en la transmisión. Detecta errores, los señaliza, envía mensaje de error y reenviará el mensaje corrupto una vez que el bus vuelva a estar activo. Además puede operar en ambientes con condiciones extremas de ruido e interferencias gracias a que es un bus diferencial.

8. Es un protocolo de comunicaciones normalizado, con lo que se simplifica y economiza la tarea de comunicar subsistemas de diferentes fabricantes sobre una red común o bus.

9. Al ser una red multiplexada, reduce considerablemente el cableado y elimina las conexiones punto a punto.

10. El procesador principal delega la carga de comunicaciones a un periférico inteligente (controlador), por lo tanto el procesador principal dispone de mayor tiempo para ejecutar sus propias tareas.

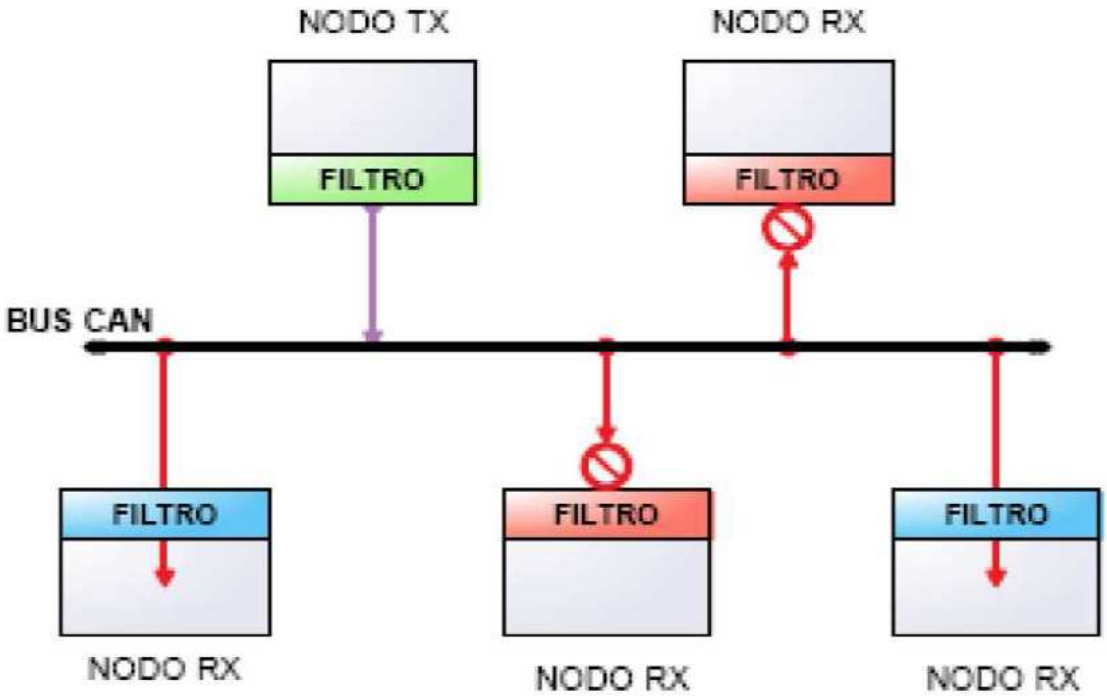


Figura 7: Sistema multicast

Algunas limitaciones de CAN:

1. Velocidad limitada por la longitud de la red. Cada red puede alcanzar como máximo los 5000m de longitud (10 kbps) o una velocidad máxima de 1 Mbps (40 m). La longitud del bus también esta limitada en función del transceiver de can que utilicemos. EL HW de Microchip admite como velocidad más baja 16kbps que equivale a un bus máximo de 3300m. El transceiver de Panasonic admite velocidad de transmisión desde 0 hasta 1Mb. Esto implica que con un buen cableado podríamos cubrir distancias de 10Km sin problemas.

2. La cantidad de nodos abonados a la red es variable pero limitada. Esta limitación viene marcada según el HW (transceiver de can utilizado). Para el caso del HW Microchip el número máximo de nodos es 121.

2.3.2. Máscaras y Filtros

Hay que recordar que CAN es un protocolo que se basa en tipos de mensajes, no en direcciones. Esto significa que los mensajes no son enviados de un nodo a otro según una dirección destino, sino que todos los nodos reciben todos los mensajes que se transmiten al bus. Por tanto, no hay forma de enviar un mensaje exclusivamente a un nodo en concreto, ya que todos los nodos recogen el tráfico de la red. Pero una vez recibido el mensaje correctamente, los nodos realizan un test de aceptación para determinar si lo procesan o no. Este proceso se denomina filtrado de mensajes. El campo de la trama que se "testea o filtra" es el campo de arbitraje (identificador).

Bit n de la máscara	Bit n del filtro	Bit n del identificador	¿Aceptado o rechazado?
0	X	X	Aceptado
1	0	0	Aceptado
1	0	1	Rechazado
1	1	0	Rechazado
1	1	1	Aceptado

Tabla 1: Ejemplo de Mascara y filtros en CAN

El funcionamiento de este proceso es bien sencillo. Los filtros guardan el valor del identificador, el cual se ha tenido que configurar previamente, de los mensajes de interés para el nodo en cuestión. Las máscaras se utilizan para indicar cuales bits de los 29 o 11 del identificador serán revisados al ejecutar el proceso de filtrado, es decir, cuales serán comparados. La máscara permite realizar el proceso de filtrado de forma más rápida. Si un nodo tiene distintos tipos de mensajes de interés, con la máscara podrá aceptarlos más rápido sin necesidad de revisar todo el identificador.

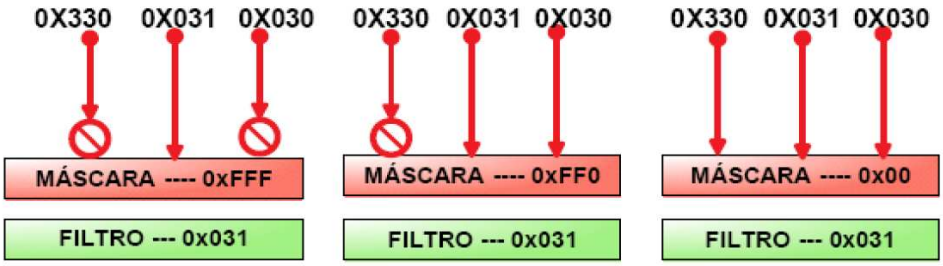


Figura 8: Ejemplo de Mascara y filtros en CAN

El principal inconveniente de este sistema de mascaras y filtros es que si se utiliza una máscara toda a 1 para que cada nodo reciba únicamente los mensajes que van dirigidos a él, se pierde la funcionalidad de broadcast, por lo que para el firmware que se va a implementar, se ha decidido que no se utilizará el sistema de mascaras y filtros que trae el protocolo CAN, se definirá una mascará toda a 0 para los nodos y será el propio nodo el que decidirá si manejar o no los mensajes que reciba.

De esta forma, el nodo comprobará si el mensaje va dirigido a él comparando su dirección MAC con el identificador del mensaje, y descartará aquellos mensajes en los que coincidan. Salvo en el caso del identificador con todos los bits a 0, que se ha acordado que será una dirección MAC de broadcast, por lo que todos los nodos manejarán estos mensajes.

CAPÍTULO 3. HERRAMIENTAS EMPLEADAS

3.1. Entrono de programación MikroC

Uno de los principales motivos por se eligió los PIC como microcontrolador fue por las potentes herramientas que ya existen desarrolladas para su programación, en concreto sus entornos de desarrollo.

La principal alternativa a MikroC es CCS, ya que las dos herramientas son muy potentes y permiten programación en C y Debug en tiempo de ejecución (que no es del todo cierto para el caso de CAN bus). El principal motivo para decantarse por MikroC fue por la cantidad de placas de HW que tiene el fabricante implementado que se pueden interconectar con sus placas de desarrollo.

El entorno es muy intuitivo del estilo de Visual Studio, con muchas herramientas que facilitan la programación. Desde un editor para la EEPROM del PIC, generador GLCD bitmap, Terminal USART, HID Terminal para USB….

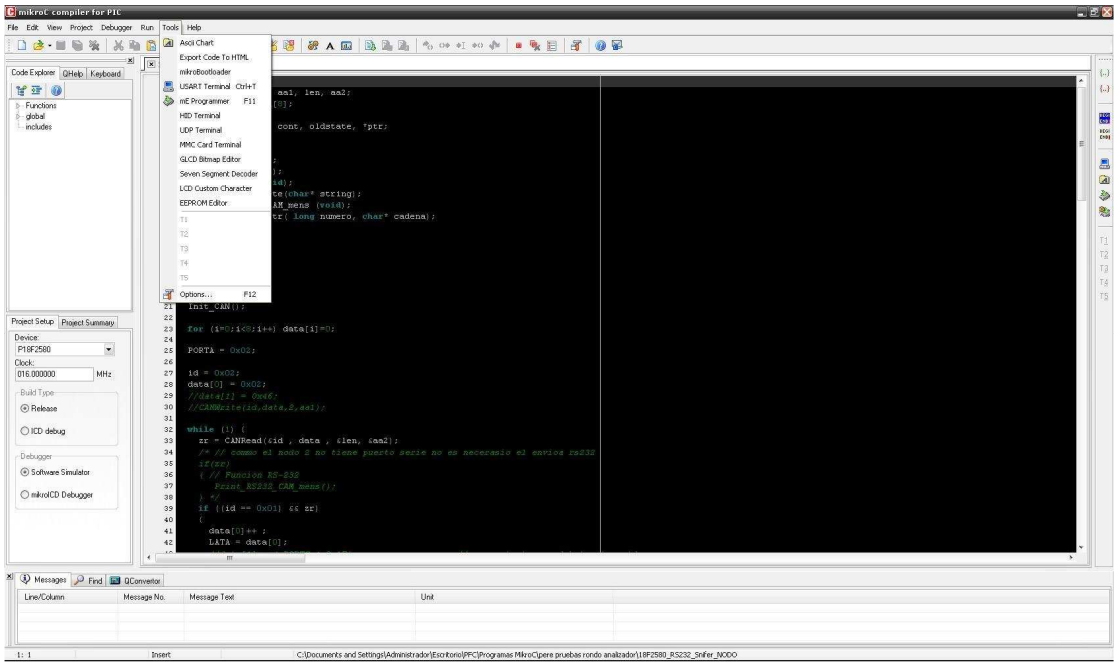


Figura 9: Interfaz MikroC

Pero lo realmente potente del compilador es la cantidad de las funciones y librerías que ya están implementadas listas para ser invocadas y proceder a su uso. Por si esto no fuese suficiente dispone de una ayuda estupenda donde te explican cada una de las funciones con ejemplos.

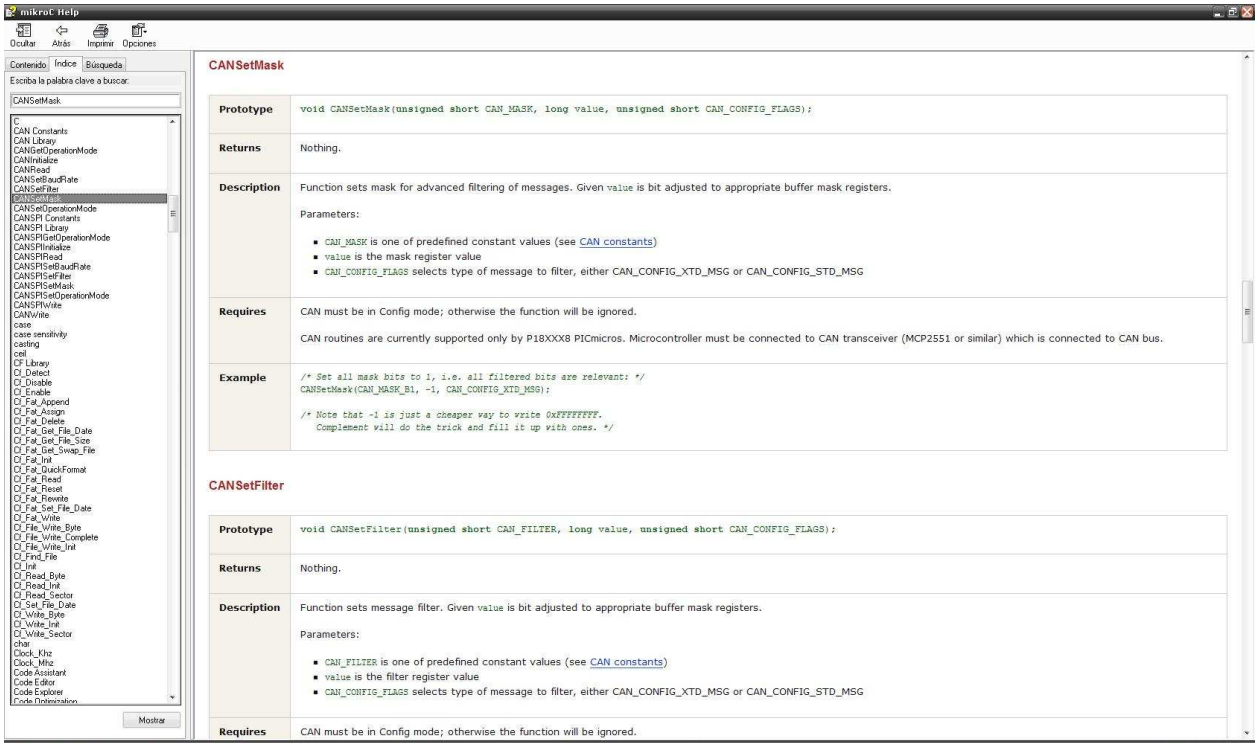


Figura 10: Ayuda MikroC

Evidentemente el entorno también da soporte para las placas de desarrollo permitiendo la depuración del programa mediante la ejecución paso a paso o con puntos de interrupción, al igual que la programación del PIC desde la propia placa de desarrollo.

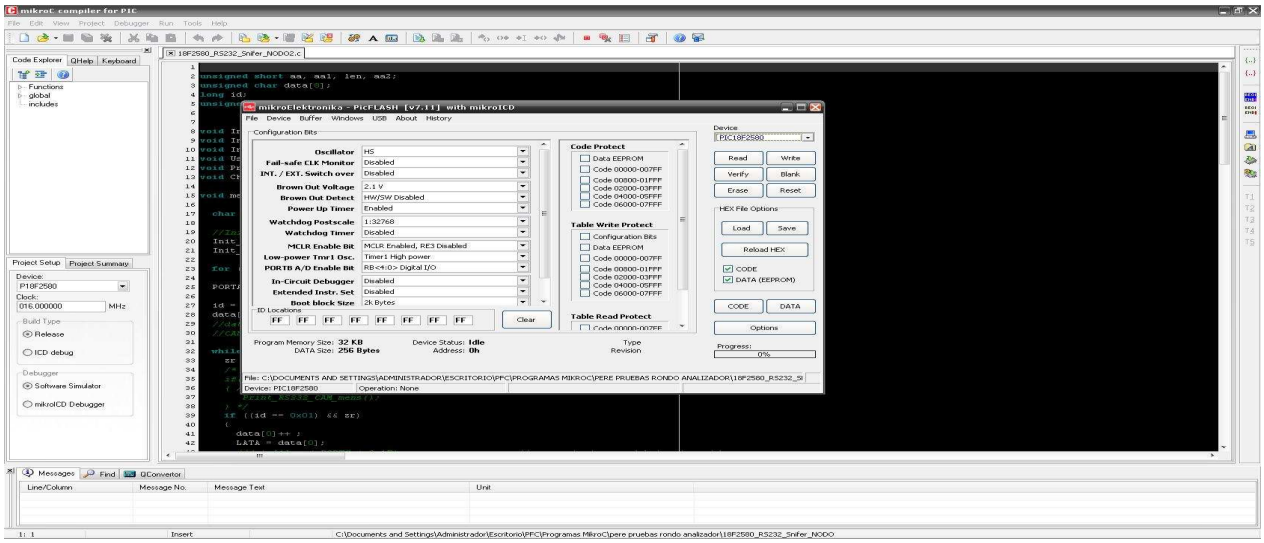


Figura 11: Programador MikroC

3.2. Entorno de programación Visual Studio 2005 Express

Microsoft Visual Studio es un entorno de desarrollo integrado (IDE, por sus siglas en inglés) para sistemas operativos Windows. Soporta varios lenguajes de programación tales como Visual C++, Visual C#, Visual J#, ASP.NET y Visual Basic .NET, aunque actualmente se han desarrollado las extensiones necesarias para muchos otros.

Visual Studio permite a los desarrolladores crear aplicaciones, sitios y aplicaciones Web, así como servicios Web en cualquier entorno que soporte la plataforma .NET (a partir de la versión Net 2002). Así se pueden crear aplicaciones que se intercomuniquen entre estaciones de trabajo, páginas Web y dispositivos móviles.

La actualización más importante que recibieron los lenguajes de programación fue la inclusión de tipos genéricos, similares en muchos aspectos a las plantillas de C#. Con esto se consigue encontrar muchos más errores en la compilación en vez de en tiempo de ejecución, incitando a usar comprobaciones estrictas en áreas donde antes no era posible. C++ tiene una actualización similar con la adición de C++/CLI como sustituto de C# manejado.

Se incluye un diseñador de implantación, que permite que el diseño de la aplicación sea validado antes de su implantación. También se incluye un entorno para publicación Web y pruebas de carga para comprobar el rendimiento de los programas bajo varias condiciones de carga.

Visual Studio 2005 también añade soporte de 64-bit. Aunque el entorno de

desarrollo sigue siendo una aplicación de 32 bits Visual C++ 2005 soporta compilación para x86-64 (AMD64 e Intel 64) e IA-64 (Itanium). El SDK incluye compiladores de 64 bits así como versiones de 64 bits de las librerías.

Las ediciones Express se han diseñado para principiantes, aficionados y pequeños negocios, todas disponibles gratuitamente a través de la página de Microsoft se incluye una edición independiente para cada lenguaje: Visual Basic, Visual C++, Visual C#, Visual J# para programación .NET en Windows, y Visual Web Developer para la creación de sitios Web ASP.NET. Las ediciones Express carecen de algunas herramientas avanzadas de programación así como de opciones de extensibilidad.

El lenguaje elegido para el desarrollo ha sido el C#, por la experiencia personal en este leguaje.

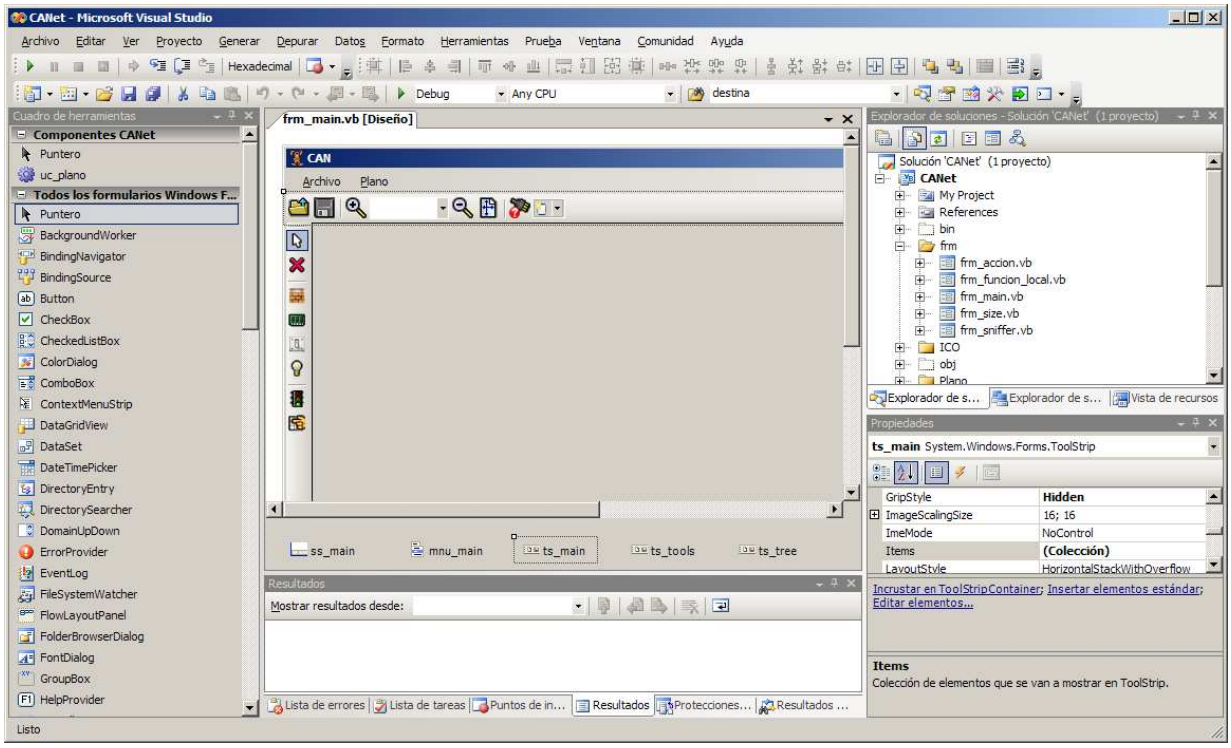


Figura 12: IDE Visual Studio 2005 Express

3.3. Programador PIC's

Necesario para cargar el firmware en los nodos, se ha empleado un programador GTPLite y el programa WinPic80. Es un programador realmente rápido y compatible

con todos SO de Windows, además dan soporte para actualizar el FW del programador y actualizan constantemente todos los micros que soporta. La interfaz es intuitiva y fácil de usar.

Este programador nos permitirá programar los chips sin necesidad de extraerlos de la placa.

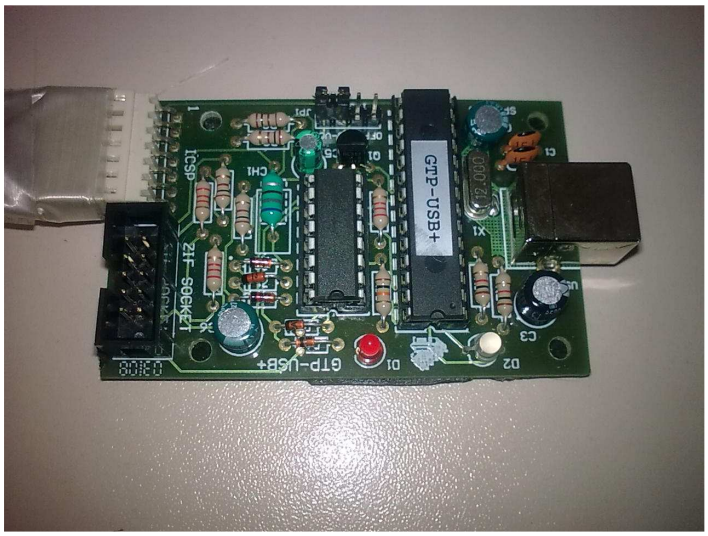


Figura 13: Programador GTPLite

3.4. Pasarela RS232/CAN

El nodo pasarela fue diseñado para poder comunicar la red CAN bus con el PC. El nodo utilizado es un prototipo creado, del que únicamente existen dos unidades, pero el nodo final mantendrá sus características, si bien se espera que sus dimensiones sean más reducidas.

Esta conexión nos ofrece varias utilidades:

- Herramienta de depuración de programas.
- Interactuar con los nodos de la red CAN bus.
- Creación de una pasarela mediante un PC CAN <-> Internet

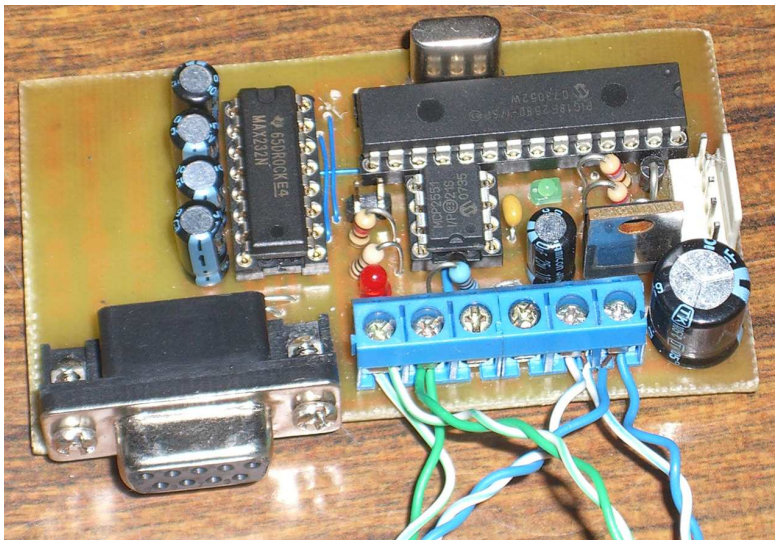


Figura 1: Hardware pasarela RS232/CAN

CAPÍTULO 4. Diagramas UML

4.1. Diagrama de dominio

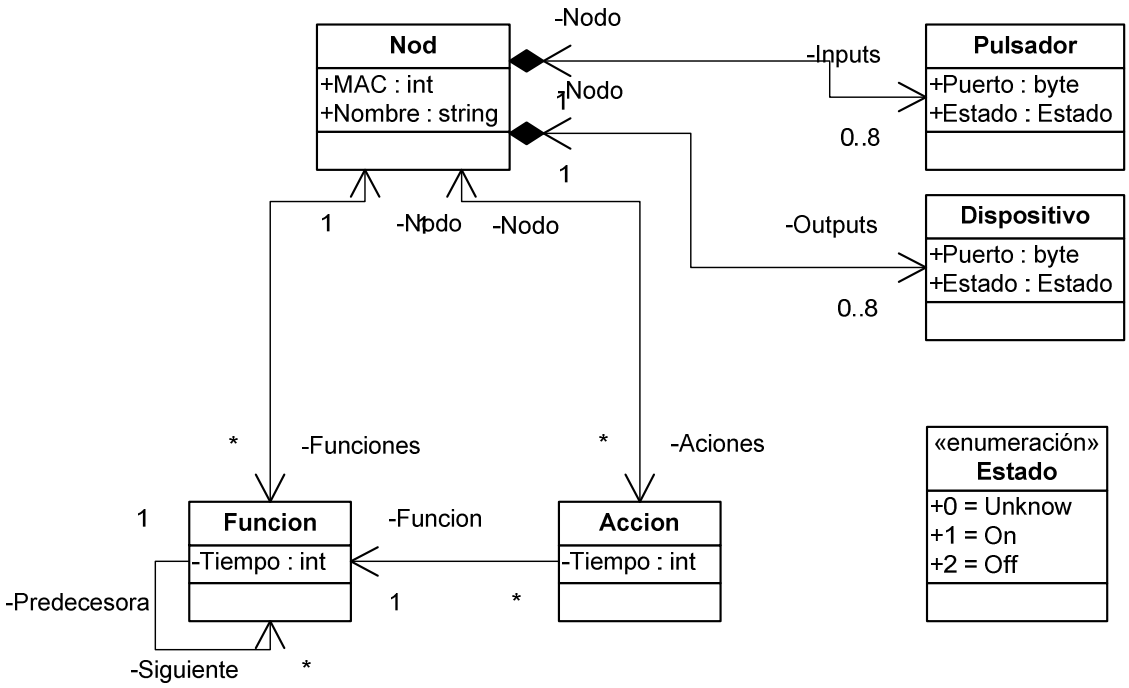


Figura 14: Diagrama de dominio

4.2. Diagrama de despliegue

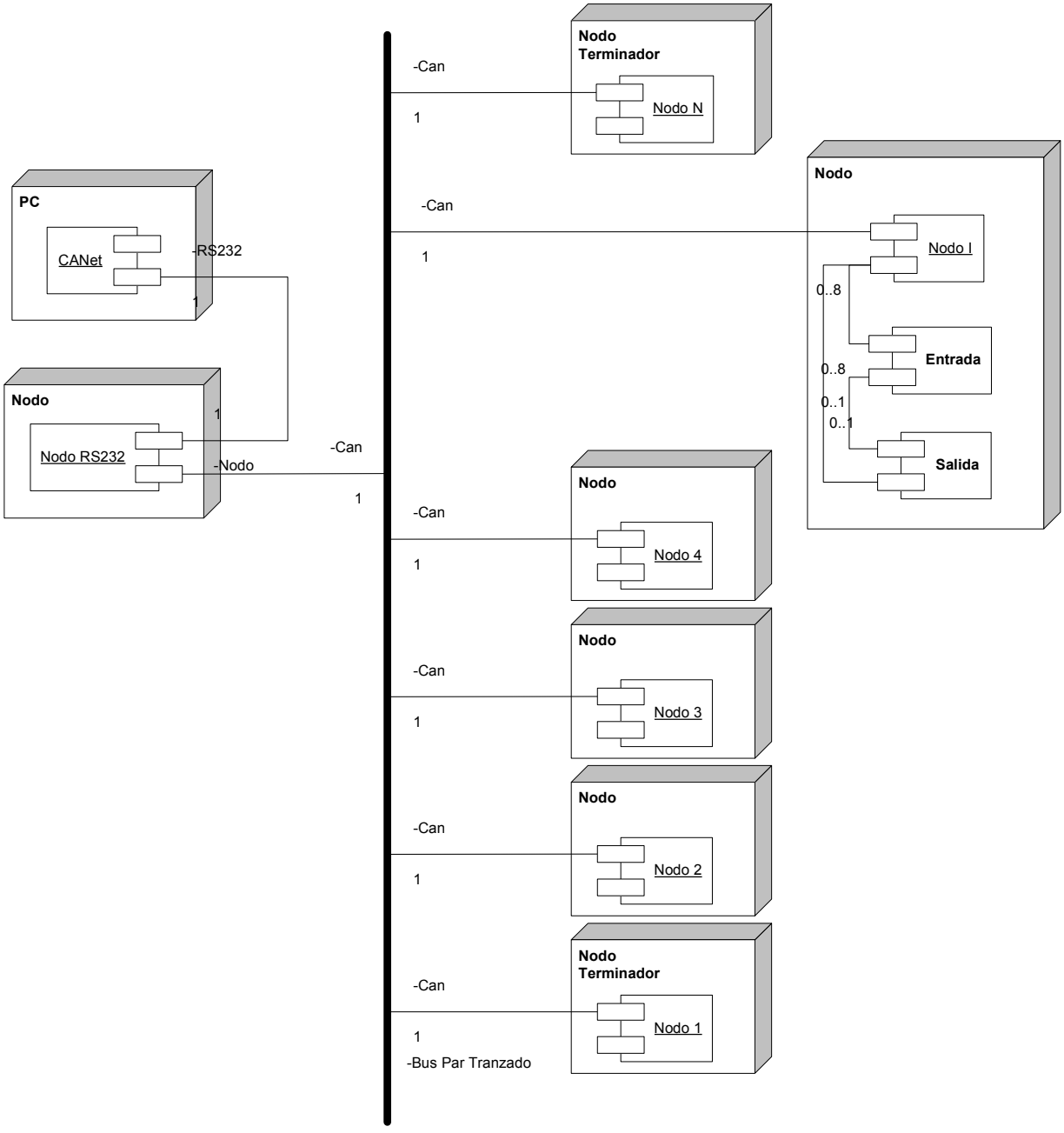


Figura 15: Diagrama de despliegue

4.3. Diagrama de secuencia

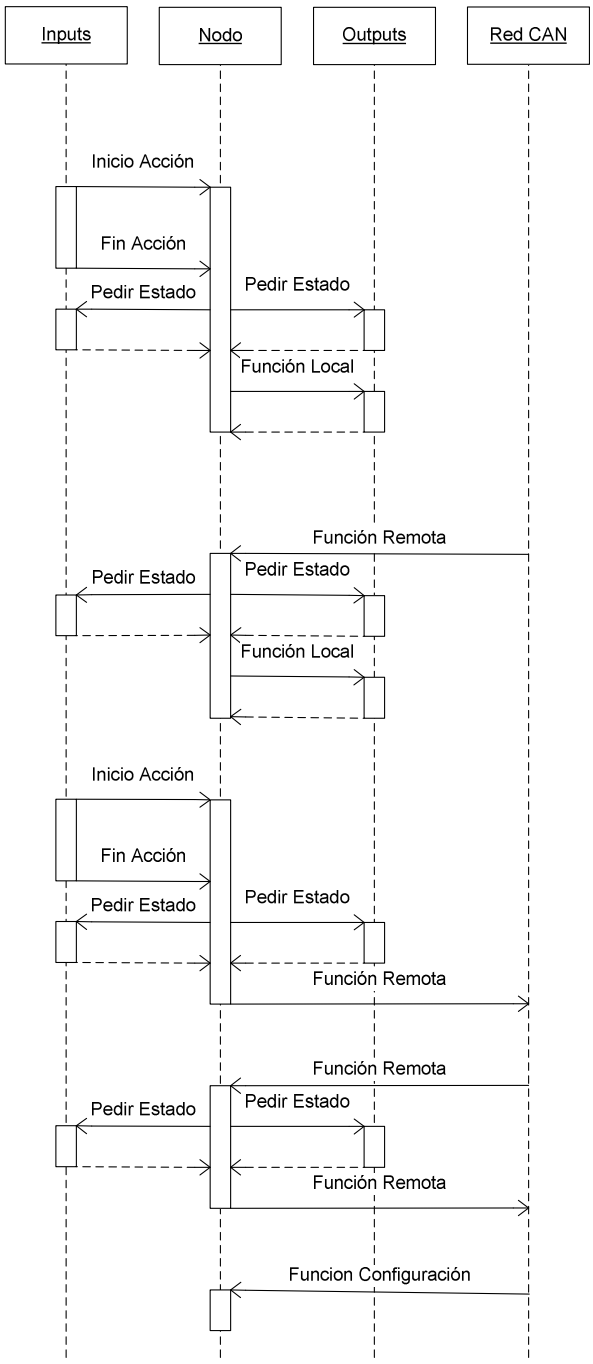


Figura 16: Diagrama de secuencia

4.4. Diagramas de Estados

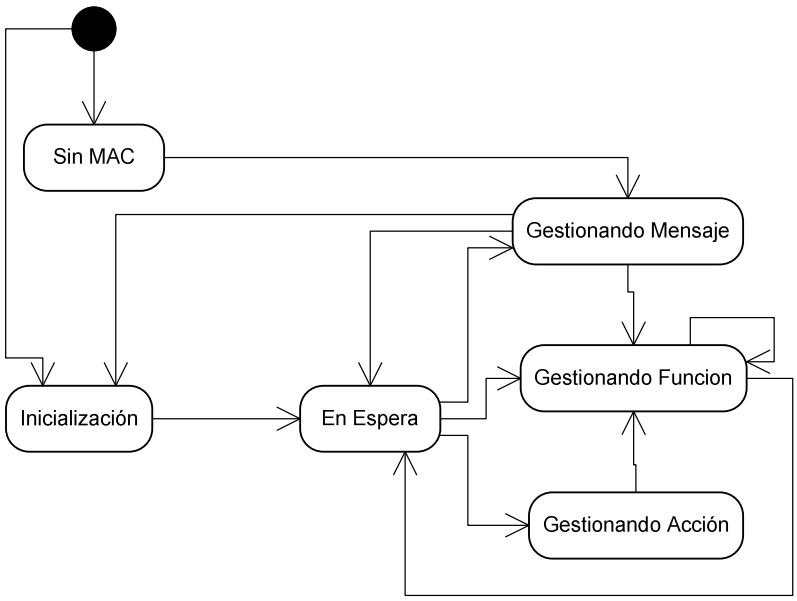


Figura 17: Diagrama de estados del nodo

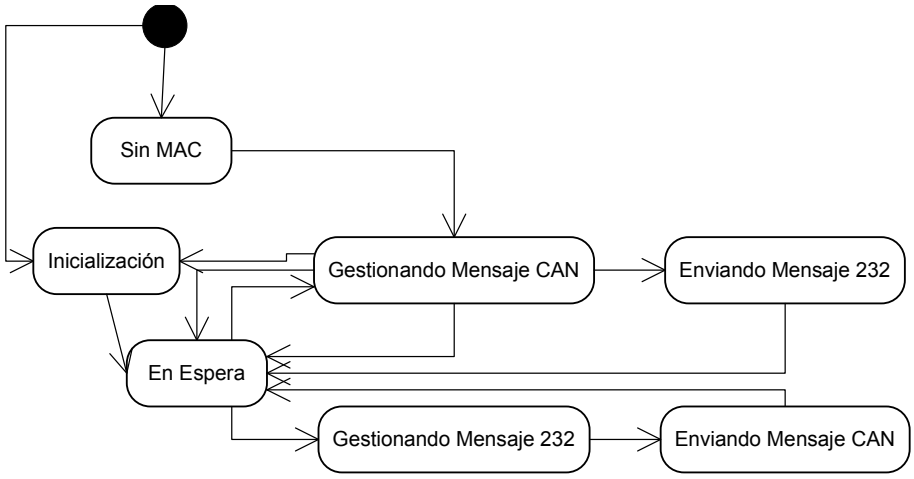


Figura 18: Diagrama de estados de la pasarela

CAPÍTULO 5. APLICACIÓN DE ADMINISTRACIÓN

El principal objetivo a la hora de desarrollar la aplicación de administración era poder exprimir al máximo la potencia del nodo, sin obligar al usuario a adquirir conocimientos técnicos.

Por ello se ha optado por una interfaz grafica con una filosofía de "Arrastrar y Soltar", en la que la configuración de los nodos se hace de una forma muy visual, empleando mayoritariamente el ratón.

5.1. Estructura de la interfaz

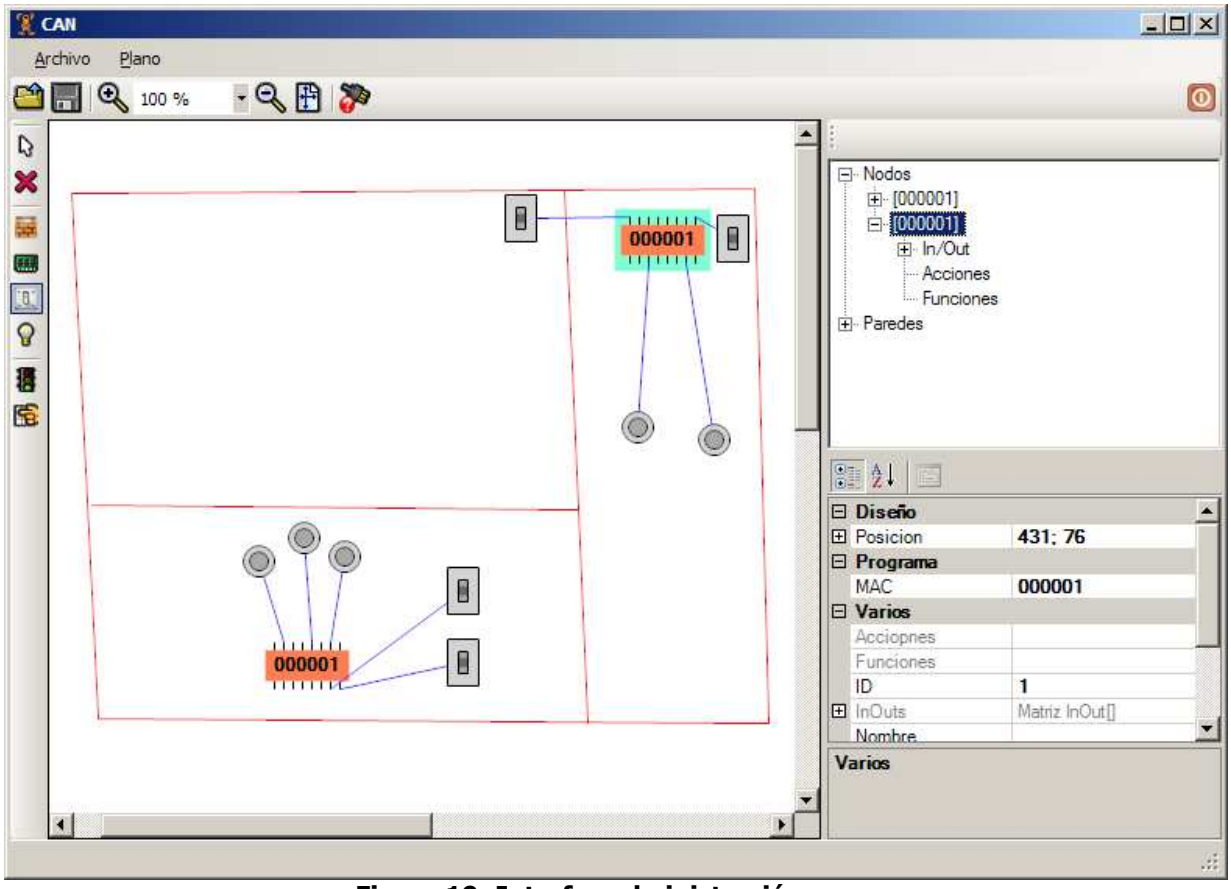


Figura 19: Interfaz administración

La interfaz se ha dividido en 5 zonas:

5.1.1. Barra de Menús

La barra de menús permite al usuario abrir, cerrar y guardar ficheros, cambiar el nivel del zoom, el tamaño del dibujo o enviar la configuración de los nodos.

Figura 20: La Barra de Menús

Para facilitar el uso, además de los menús contextuales, se ha añadido una barra de botones que permite acceder a los principales comandos del menú.

5.1.2. Barra de herramientas

Permite seleccionar la herramienta que se va a utilizar para modificar el dibujo de la configuración.

Figura 21: Interfaz administración

Los botones disponibles, por orden de arriba abajo son:

- **Seleccionar**: Permite seleccionar un elemento del dibujo para modificarlo o borrarlo.
- **Borrar**: Elimina el elemento seleccionado.
- **Pared**: Dibuja una pared. Las paredes no tienen otra utilidad que ayudar al usuario a la hora de identificar la posición de los elementos del dibujo.
- **Nodo**: Añade un nodo de control.
- **Pulsador**: Añade un dispositivo de entrada a un nodo.
- **Dispositivo**: Añade un dispositivo de salida a un nodo.
- **Función**: Crea una función en un nodo.

- **Acción**: Crea una acción en un nodo.

5.1.3. Área de dibujo

Representa la distribución de nodos y dispositivos de entrada y salida conectados a cada uno.

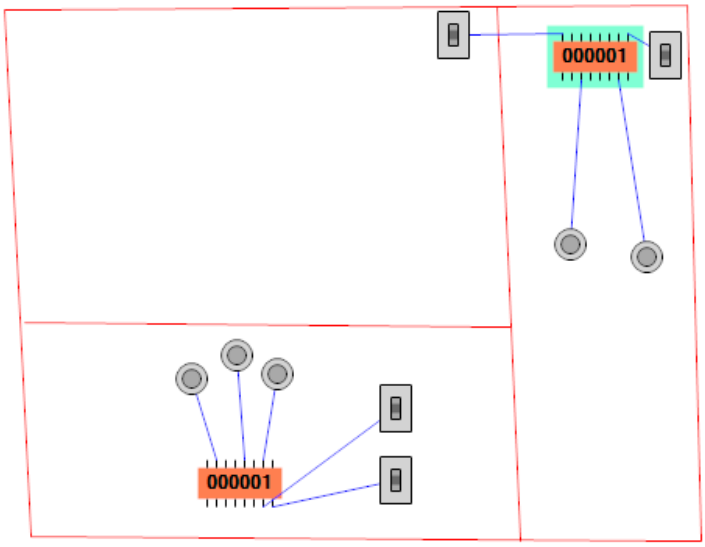


Figura 22: Área de dibujo

Cuando se están creando funciones o acciones también se refleja en el área de dibujo el estado de las entradas y salidas que configuran la función o acción.

5.1.4. Árbol de Elementos

Permite ver en una estructura en árbol todos los elementos del dibujo, así como seleccionar un elemento en concreto.

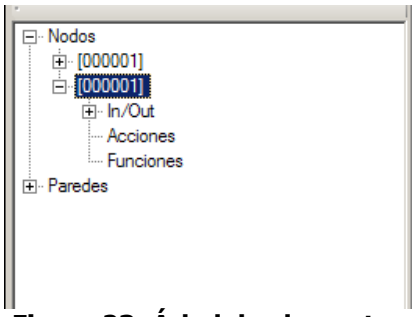


Figura 23: Árbol de elementos

5.1.5. Tabla de propiedades

Cuando se selecciona un elemento, ya sea en el "Área de dibujo" o en el "Árbol de elementos", en esta área se reflejan las propiedades de dicho elemento.

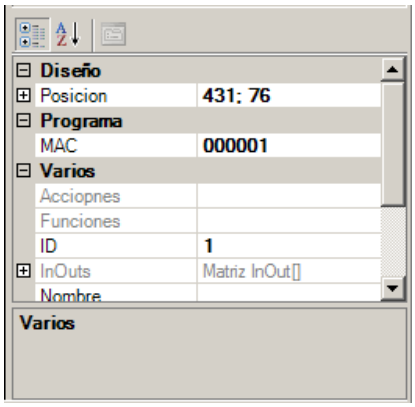


Figura 24: Tabla de propiedades

5.2. Uso de la aplicación

5.2.1. Crear Muros

Los muros sirven como guía a la hora de añadir más adelante los nodos y los dispositivos, ya que si hacemos previamente un plano de la casa, luego nos será más fácil identificar que representa cada dispositivo en función de la posición que ocupa en cada habitación.

Para dibujar un muro, hay que tener la herramienta "Muro" seleccionado, y entonces pulsar en dos puntos del "Área de dibujo", y un muro desde el primer punto al segundo se añadirá al dibujo.

Figura 25: Dibujando un Muro

Se han añadido unas cuantas ayudas para mejorar la experiencia del usuario:

Si se tiene pulsada la tecla "Control" cuando se pulsa en el punto de destino del muro, se empezará de forma automática un nuevo muro en ese punto.

Figura 26: Muros consecutivos

Si se aproxima el ratón al extremo de otro muro, aparece un círculo alrededor de ese extremo del muro para indicarnos que cuando pulsemos el ratón el punto que se tomará será el del extremo del muro.

Figura 27: Muros encadenados.

5.2.2. Crear Nodos

Con la herramienta "Nodo" seleccionada al pulsar sobre un punto del "Área de dibujo", un nuevo nodo aparecerá, y lo podemos arrastrar a la posición que queramos hasta que se vuelva a pulsar el ratón.

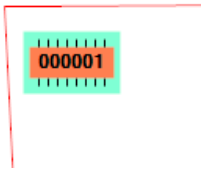


Figura 28: Creando un nodo.

Una vez creado el nodo debemos seleccionarlo cambiarle la dirección MAC en la tabla de propiedades.

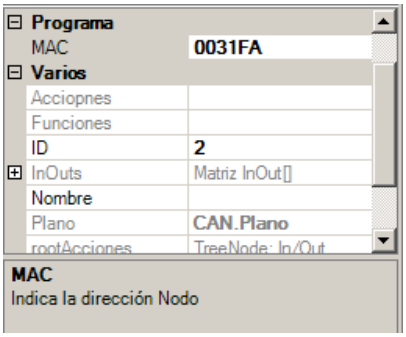


Figura 29: Cambiar la MAC de un nodo.

También se le pude asignar un nombre al nodo (por ejemplo "Nodo de pasillo", o "Control de persianas") para que sea más fácil identificarlo.

5.2.3. Crear Pulsadores

Con la herramienta "Pulsador" seleccionada hay que pulsar sobre una de las patillas de la representación de un nodo en el "Área de dibujo" para enlazar el pulsador a un nodo en concreto, y luego volver a pulsar en otra posición para situar el pulsador.

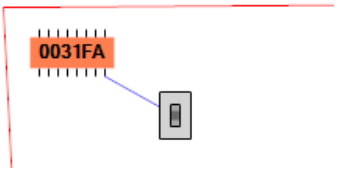


Figura 30: Añadiendo un pulsador.

5.2.4. Crear Dispositivos

Con la herramienta "Dispositivo" seleccionada hay que pulsar sobre una de las patillas de la representación de un nodo en el "Área de dibujo" para enlazar el dispositivo a un nodo en concreto, y luego volver a pulsar en otra posición para situar el dispositivo.

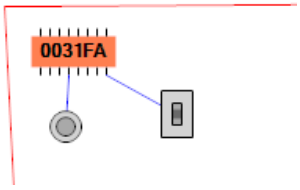


Figura 31: Añadiendo un dispositivo.

5.2.5. Crear Funciones

Con la herramienta "Función" seleccionada hay que pulsar sobre los dispositivos de un nodo para cambiar el estado hasta obtener la configuración deseada que se corresponda a la función que queremos crear.

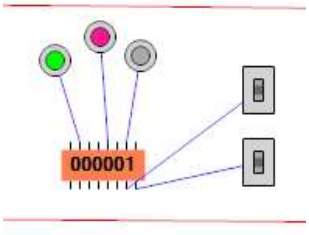


Figura 32: Creando una función.

Los dispositivos disponen de tres estados, identificados con un código de color:

- **Verde**: Dispositivo encendido.
- **Rojo**: Dispositivo apagado.
- **Gris**: Estado indiferente. Este estado se usa para indicar que el estado de este dispositivo no es controlado por esta función, y que cuando se active la función que estamos diseñando, el estado de dispositivo no se modificará.

Una vez tengamos la configuración deseada, pulsamos sobre el nodo para terminar de configurar la función.

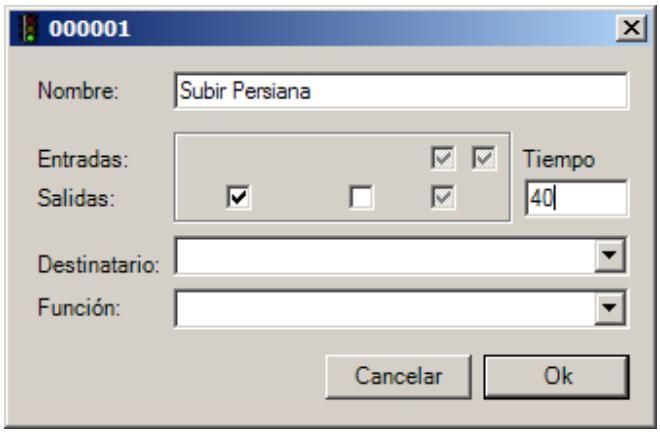


Figura 33: Configurar una función.

Ahora se puede nombrar la función con un nombre acorde con la acción que producirá.

Se debe añadir una duración (en cuartos de segundo), si no se indica, el estado de las salidas solo se mantendrá durante 250 milisegundos, suficiente para activar un teleruptor.

Por último, para definir funciones más complejas, se pueden encadenar varias (con los campos "Destinatario" y "Función", permitiendo que cuando una función termine invoque a otra función en el mismo nodo (seleccionando "Local" en el desplegable de "Destinatario") o en otro remoto.

5.2.6. Crear Acciones

Con la herramienta "Acción" seleccionada hay que pulsar sobre los pulsadores y los dispositivos de un nodo para cambiar el estado hasta obtener la configuración deseada que se corresponda a la acción que queremos crear.

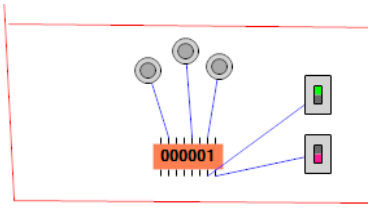


Figura 34: Creando una acción.

Los pulsadores, al igual que los dispositivos, disponen de tres estados, identificados con un código de color:

- **Verde**: Pulsador encendido.
- **Rojo**: Pulsador apagado.
- **Gris**: Estado indiferente. Este estado se usa para indicar que el estado de este pulsador no es tenido en cuenta para activar esta acción.

Una vez tengamos la configuración deseada, pulsamos sobre el nodo para terminar de configurar la acción.

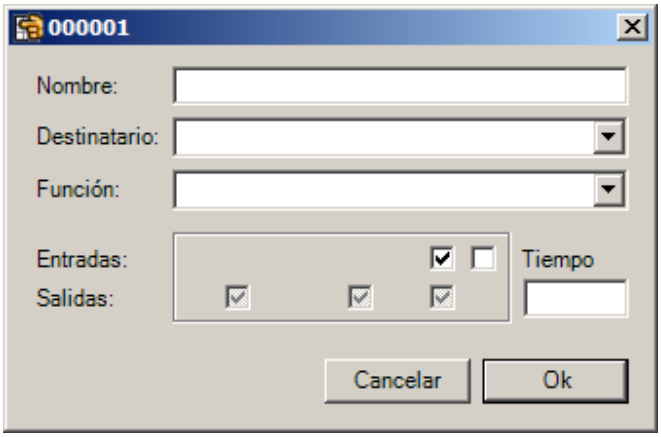


Figura 35: Configurar una acción.

Ahora se puede nombrar la acción con un nombre acorde con la acción que producirá.

Se debe añadir una duración (en cuartos de segundo). Este tiempo indica cuanto tiempo tiene que mantenerse la configuración de Entradas/Salidas para que se active esta acción.

Por último, se debe definir el "Destinatario" y la "Función" que se debe ejecutar.

5.3. Programación de los Nodos

Una vez se ha terminado de configurar todos los nodos con sus funciones y acciones, se puede mandar la programación a los nodos, a través del puerto serie al nodo pasarela, y este enviará a cada nodo su configuración.

Para ello se tiene que pulsar el botón de "Programar" que hay en la barra de menús.

El tiempo de programación depende del número de información que hay que enviar, y hay que tener en cuenta cada vez se borra toda la información en los nodos y se manda de nuevo toda la configuración.

CAPÍTULO 6. FIRMWARE DE LA PASARELA

El firmware de la pasarela es realmente simple, ya que simplemente actúa como traductor entre los mensajes de 232 y los de CAN.

Para evitar problemas con la tabla de caracteres en 232, los mensajes se mandan como una cadena de texto de los bytes en su representación en hexadecimal, terminando siempre con un retorno de carro.

Los mensajes de 232 tendrán siempre una longitud de 24 caracteres, siendo los 8 primeros para indicar la dirección MAC del destinatario y los 16 siguientes para indicar el contenido del mensaje de CAN.

Por ejemplo, si se desea decir al nodo con MAC 0x10A2036B que tiene que realizarla función 0x12, el texto que se enviaría por 232 seria:

10A2036B6012000000000000

Por último, el nodo pasarela actúa como un sniffer, leyendo todos los mensajes que haya en el bus CAN y enviándolos como texto por 232.

Debido a que el buffer de lectura de 232 del chipset de la pasarela tiene solo una longitud de 2 caracteres, no se puede asegurar que cada vez que se lea un mensaje por 232 sea un mensaje completo, por lo que se ha optado por ir almacenando los datos que se leen por 232 hasta que se obtiene un mensaje completo, y entonces procesarlo.

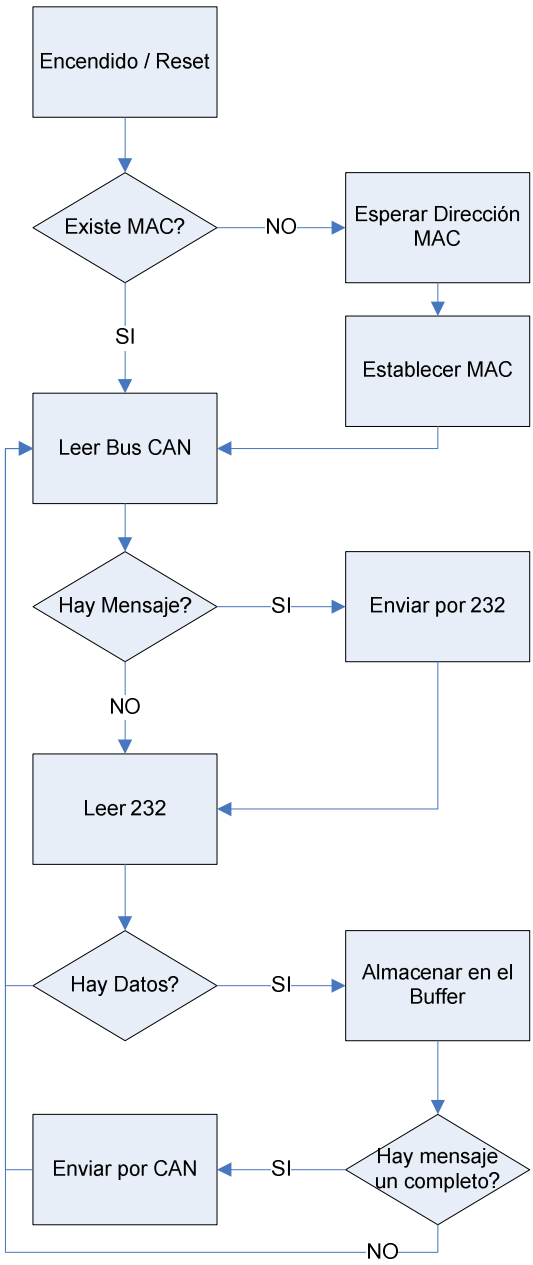


Figura 36: Diagrama de flujo de la pasarela

CAPÍTULO 7. FIRMWARE DEL NODO

7.1. Diagrama de flujo

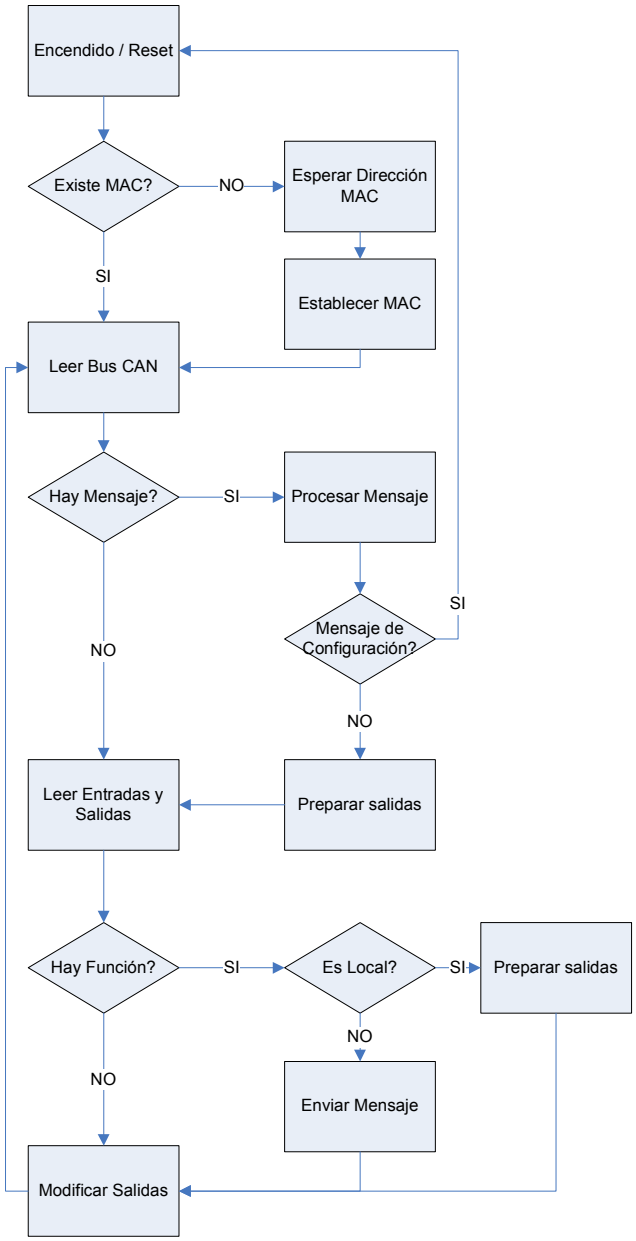


Figura 37: Diagrama de flujo del nodo

7.2. Direccionamiento MAC

Cada nodo, cuando se le carga el firmware por primera vez, tiene asignada una MAC igual a 0 y no funcionará correctamente hasta que se la asigne una.

En principio, cada MAC identifica de forma única a cada nodo, pero no hay nada que impida que dos nodos compartan la misma MAC, y de hecho, en algunas configuraciones puede ser interesante que esto ocurra.

Los nodos están atentos a todos los mensajes que circulen por el bus CAN, pero solo harán caso a aquellos que estén dirigidos a su dirección MAC, con la excepción de un mensaje enviado a la MAC 0x0000 que será tratado por todos los nodos.

7.3. Funciones y Acciones

Los nodos funcionan por lo que hemos llamado Funciones y Acciones.

Una función es cualquier combinación de estados de entradas y salidas que se mantiene constante durante una determinada duración de tiempo.
Las acciones se han subdividido en dos tipos:
Acciones locales: Cambian el estado de las salidas de un nodo.
Acciones remotas: Ejecutan una acción en otro nodo

Un Ejemplo práctico sería el siguiente:

Un nodo dispone de cuatro actuadores (salidas). Los actuadores están conectados a dos motores que controlan la subida y bajada de dos persianas.

Las posibles acciones locales que se definirían en el nodo serian:

Acción	Mascara	Valor	Duración
Subir Persiana A	00001100	00000100	5
Bajar Persiana A	00001100	00001000	5
Parar Persiana A	00001100	00000000	5
Subir Persiana B	00000011	00000001	5
Bajar Persiana B	00000011	00000010	5
Parar Persiana B	00000011	00000000	5
Subir Persianas	00001111	00000101	5
Bajar Persianas	00001111	00001010	5
Parar Persianas	00001111	00000000	5

Tabla 2: Acciones Persianas

Por ejemplo, la acción "Subir Persiana B" está actuando sobre las salidas 6 y 7 y está activando la salida 7 y desactivando la 6 durante 5 segundos, pasado este tiempo, desactivará todas.

Para las funciones, hay que especificar tanto el estado de las entradas como de las salidas, ya que el funcionamiento de un pulsador puede depender del estado actual, siguiendo con el ejemplo anterior, si añadimos 4 pulsadores al nodo, podríamos definir estas siguientes funciones:

ENTRADAS		SALIDAS			
Máscara	**Valor**	**Máscara**	**Valor**	**Duración**	**Acción**
00000011	00000001	00000010	00000000	1	Subir Persiana B
00000011	00000001	00000010	00000010	1	Parar Persiana B

Tabla 3: Funciones Persianas

Estas funciones se activarán que cuando accionamos el pulsador de la entrada 7 y no esté activo el pulsador el de la entrada 6, asumiendo que etiquetamos el pulsador 7 para subir la persiana B y el 6 para bajarla, el nodo no haría nada si pulsamos los dos botones a la vez.

Sin embargo el botón de subir actuará de distinta forma dependiendo de cual sea el estado de las salidas, ya que si la salida que controla el bajado de la persiana está activa, se parara, pero en caso contrario se iniciará la subida de la persiana

El campo duración indica cuanto tiempo (en ¼ de segundo) debe estar activa la configuración de entradas y salidas para que se ejecute la acción definida. De esta forma, podríamos definir una acción de subir persiana que dure todo el recorrido y otra que dure la mitad, y con el mismo pulsador de subir la persiana, podríamos ejecutar la acción de subirla completa o subirla hasta la mitad en función de cuanto lo mantengamos pulsado

Por último, podríamos añadir un nodo o en la otra punta de habitación, o en otra habitación, y si queremos controlar las persianas desde los pulsadores conectados a ese nodo, las acciones que definiríamos serian acciones remotas, y en lugar almacenar la configuración de salidas para el propio nodo, tendríamos la dirección MAC del otro nodo y la acción que queremos ejecutar en ese nodo

Un ejemplo en el que se utilizarían mensajes de broadcast seria el siguiente:

En todos los nodos que controlan la iluminación se define una acción que sea

"apagar todo"

Desde un nodo situado en la entrada de la vivienda, se programa una acción remota sobre la MAC de broadcast con la acción de "apagar todo"

De esta forma, al salir de la vivienda, pulsaríamos un botón que lanzaría el mensaje y todas las luces se apagarían.

7.4. Distribución de la memoria

Se utiliza la el propio espacio de memoria reservado para el programa para almacenar la configuración del nodo, por ello se ha dividido el espacio total 32Kb en 4 secciones:

- **Área de programa**: Almacena el código ejecutable, al reducir el espacio disponible para el programa, se ha tenido mucho cuidado al programar no sobrepasar el límite establecido.
- **Área de configuración**: Almacenas datos de configuración como la dirección MAC y las funciones y acciones que están disponibles
- **Área de Acciones**: Almacena las configuraciones de las Acciones
- **Área de Funciones**: Almacena las configuraciones de las Funciones

7.4.1. Área de programa

Empieza en la dirección 0x0000.

7.4.2. Área de configuración

Empieza en la dirección 0x2000 y ocupa 32 palabras.

Hay un primer bloque de 8 palabras, de las que se utilizan las dos primeras para almacenar la dirección MAC del nodo.

Luego se utiliza un bloque de 8 palabras en la que cada bit sirve apara indicar si existe o no una acción determinada, por lo que se pueden definir 128 acciones distintas.

Por último hay otro bloque de 8 palabras en la que cada bit sirve apara indicar si existe o no una función determinada, por lo que se pueden definir 128 funciones

distintas.

El motivo de utilizar 32 palabras aunque no se empleen todas es porque es el tamaño del bloque de escritura del chip.

7.4.3. Área de Acciones

Empieza en la dirección 0x2040.

Para el bloque de las acciones se utilizan cuatro palabras por cada acción, con las siguientes estructuras:

Dato	Long	Descripción
Tipo	8	0x21
Mascara Entradas	8	
Valor Entradas	8	
Mascara Salidas	8	
Valor Salidas	8	
Duración	8	En cuartos de segundo.
Siguiente Acción	8	Hace referencia a una acción del propio nodo.
Resto	8	

Tabla 4: Almacenamiento de una Acción local

Dato	Long	Descripción
Tipo	8	0x22
MAC Destinatario	32	
Función Remota	8	
Siguiente Acción	8	Hace referencia a una Acción del propio nodo.
Resto	8	

Tabla 5: Almacenamiento de una Acción remota

7.4.4. Área de Funciones

Empieza en la dirección 0x3000 y se utilizan cuatro palabras por cada acción, con la siguiente estructura:

Dato	Long	Descripción
Mascara Entradas	8	
Valor Entradas	8	
Mascara Salidas	8	
Valor Salidas	8	
Duración	8	En cuartos de segundo.
Acción	8	Hace referencia a una acción del propio nodo.
Resto	16	

Tabla 6: Almacenamiento de una función

CAPÍTULO 8. CATÁLOGO DE MENSAJES

Todos los mensajes DOMCAN tienen sentido en el contexto de un diálogo entre dos nodos. Este apartado recoge la lista de diálogos definidos al amparo del protocolo DOMCAN. Los diálogos se han clasificado en tres grupos atendiendo al contexto de utilización. Para cada diálogo se indican los nodos que los utilizan y el papel de cada una de ellos. También se describen detalladamente los mensajes y las reglas principales que gobiernan el diálogo.

8.1. Estructura general de los mensajes

La principal característica a tener en cuenta a la hora de definir el protocolo ha sido la longitud máxima del campo de datos de las tramas enviados a través de la red CAN: 8 bytes, por lo que se ha intentando optimizar al máximo la información que se manda en cada trama, de forma que cada mensaje tenga un significado completo y no haga falta segmentar un mensaje en varias tramas.

Como la mayoría de los mensajes son asíncronos y no requieren respuesta, en el campo del identificador de la trama se indica la dirección MAC del nodo de destino, de forma que todos nodos están abonados a todos los mensajes y es el firmware del nodo el que se encarga de decidir si debe leer el mensaje o ignorarlo. Para los mensajes broadcast que deben ser leídos por todos los nodos se utiliza la dirección MAC 0x0000000

El primer nibble del campo de datos de cada trama identifica el tipo de mensaje. Si es necesario, se utiliza el segundo nibble para indicar un subtipo.

El significado del resto de bytes depende del tipo de mensaje. Cuando un mensaje no requiere utilizar todos los bytes de la trama, se rellena el resto a 0x00

8.2. Diálogos de Configuración

Estos mensajes son generados por un nodo especial que es el que está conectado a la aplicación de control y son empleados para establecer el comportamiento de cada nodo. Por motivos de seguridad, no se permite utilizar estos mensajes con la dirección de broadcast (salvo el mensaje C_MAC)

8.2.1. Mensaje C_MAC_PETICION

Sirve para cambiar la dirección MAC de un nodo. Cuando se utiliza con la dirección de destino de broadcast (0x0000000) solo afecta a los nodos que tienen un firmware recién cargado, ya que su dirección MAC es 0x0000000 y están esperando a este mensaje para coger la dirección MAC que lo identificara. Estos nodos solo reaccionan a este mensaje, descartando el resto.

Por lo que cuando se quiere inicializar los nodos deben ser conectados y asignarles la MAC de uno en uno.

Es necesario incluir la dirección del remitente para enviar un ACK de respuesta, por este motivo, en este caso, es solo un seminibble el que indica el subtipo.

Dato	Long	Valor	Descripción
Tipo	4	0x1	C_MAC
Petición	2	00	PETICION
Remitente	29		Dirección MAC del remitente
Remitente	29		Nueva dirección MAC

Tabla 7: Mensaje C_MAC_PETICION

8.2.2. Mensaje C_MAC_ACK

Respuesta al mensaje C_MAC, al ser el cambio de MAC un punto crítico de la instalación de un nodo, el nodo ha de enviar la confirmación de que ha realizado el cambio correctamente.

Dato	Long	Valor	Descripción
Tipo	4	0x1	C_MAC
Subtipo	2	11	ACK
Remitente	29		Dirección MAC del remitente
MAC Vieja	29		Dirección MAC vieja

Tabla 8: Mensaje C_MAC_ACK

8.2.3. Mensaje C_ACCION_LOCAL

Sirve para configurar una acción local en un nodo. Los campos "Mascara OUT" y "Valor OUT" funcionan de la siguiente manera:

Los bits del campo "Mascara OUT" identifican las salidas, por ejemplo un valor de 00010001, significa que solo se va a actuar sobre las salidas 3 y 7.

Los bits del campo "Valor OUT" indican el valor que deben tomar las salidas identificadas en el campo "Mascara OUT", siguiendo con el ejemplo anterior, un valor de 00000001, significa que la salida 3 se debe apagar y la salida 7 encender.

Los bits del campo "Valor OUT" que no están afectados por el campo "Mascara OUT", podrían tomar cualquier valor al azar, pero por claridad en la lectura se recomienda que tomen valor 0.

Dato	Long	Valor	Descripción
Tipo	4	0x2	C_ACCION
Subtipo	4	0x1	LOCAL
ID Acción	8		Identificador único de la acción
Mascara IN	8		Entradas que se van a cambiar
Valor IN	8		Valores que deben tomar las entradas
Mascara OUT	8		Salidas que se van a cambiar
Valor OUT	8		Valores que deben tomar las salidas
Duración	8		Tiempo en ½ segundos que debe mantenerse activa la acción
Siguiente Acción	8		Identificador de la siguiente acción

Tabla 9: Mensaje C_ACCION_LOCAL

8.2.4. Mensaje C_ACCION_REMOTA

Sirve para configurar una acción remota en un nodo.

Dato	Long	Valor	Descripción
Tipo	4	0x2	C_ACCION
Subtipo	4	0x2	REMOTA
ID Acción	8		Identificador único de la acción
	3	000	Bits de relleno para la MAC
MAC Destino	29		Dirección MAC del nodo de destino
ID Acción Destino	8		Identificador de la acción en el nodo de destino
Siguiente Acción	8		Identificador de la siguiente acción

Tabla 10: Mensaje C_ACCION_REMOTA

8.2.5. Mensaje C_FUNCION

Sirve para configurar una función. Los campos "Mascara OUT", "Valor OUT", "Mascar IN" y "Valor IN" funcionan igual que en el mensaje C_ACCION_LOCAL.

El campo "Duración" indica el tiempo en fracciones de medio segundo, por lo que la duración mínima de una función sería 250 milisegundos (pulsaciones de menor duración se ignoran) y la duración máxima es de 64 segundos.

Dato	Long	Valor	Descripción
Tipo	4	0x3	C_FUNCION
Subtipo	4	0x0	
ID función	8		Identificador único de la función
Mascara OUT	8		Salidas que se van a comprobar
Valor OUT	8		Valores que deben tomar las Entradas
Mascara IN	8		Entradas que se van a comprobar
Valor IN	8		Valores que deben tomar las Entradas
Duración	8		Duración (en ¼ segundos) que deben mantenerse las entradas y salidas.
Acción	8		Identificador de la acción que debe ejecutarse

Tabla 11: Mensaje C_FUNCION

8.2.6. Mensaje C_BORRAR_FUNCIONES

Borra todas las funciones del nodo.

Dato	Long	Valor	Descripción
Tipo	4	0x4	C_BORRAR
Subtipo	4	0x1	FUNCIONES
Resto	56		

Tabla 12: Mensaje C_BORRAR_FUNCIONES

8.2.7. Mensaje C_BORRAR_ACCIONES

Borra todas las acciones del nodo, tanto las remotas como las locales.

Dato	Long	Valor	Descripción
Tipo	4	0x4	C_BORRAR
Subtipo	4	0x2	ACCIONES
Resto	56		

Tabla 13: Mensaje C_BORRAR_ACCIONES

8.3. Diálogos de Administración

Estos mensajes son generados por un nodo especial que es el que está conectado a la aplicación de control y son empleados para comprobar el estado de cada nodo.

8.3.1. Mensaje A_ESTADO_PETICION

Solicita a un Nodo que indique cual es el estado actual de sus entradas y salidas. Como el nodo ha de mandar una respuesta, se indica en el mensaje la dirección MAC a la que se debe dirigir esta respuesta. Este mensaje se puede utilizar con la dirección de broadcast para conocer el estado de todos los nodos.

Dato	**Long**	**Valor**	**Descripción**
Tipo	4	0x5	A_ESTADO
Subtipo	4	0x1	PETICION
	3	000	Bits de relleno para la MAC
MAC Respuesta	29		Dirección MAC a la que se debe enviar la respuesta.
Resto	24		

Tabla 14: Mensaje A_ESTADO_PETICION

8.3.2. Mensaje A_ESTADO_ACK

Respuesta al mensaje A_ESTADO_ PETICION

Dato	**Long**	**Valor**	**Descripción**
Tipo	4	0x5	A_ESTADO
Subtipo	4	0x2	ACK
	3	000	Bits de relleno para la MAC
MAC Remitente	29		Dirección MAC del remitente
Entradas	8		Estado de las entradas
Salidas	8		Estado de las salidas
Resto	8		

Tabla 15: Mensaje A_ESTADO_PETICION

8.3.3. Mensaje A_RESET

Solicita a un Nodo que se reinicie, se puede enviar a la dirección de broadcast para resetear todos los nodos de la red.

Dato	Long	Valor	Descripción
Tipo	4	0x4	A_RESET
Subtipo	4	0x0	
Resto	56		

Tabla 16: Mensaje A_RESET

8.3.4. Mensaje A_PING_PETICION

Solicita a un Nodo que indique que está activo. Como el nodo ha de mandar una respuesta, se indica en el mensaje la dirección MAC a la que se debe dirigir esta respuesta.

Dato	Long	Valor	Descripción
Tipo	4	0x5	A_PING
Subtipo	4	0x1	PETICION
	3	000	Bits de relleno para la MAC
MAC Respuesta	29		Dirección MAC a la que se debe enviar la respuesta.
Resto	24		

Tabla 17: Mensaje A_PING_PETICION

8.3.5. Mensaje A_PING_ACK

Respuesta al mensaje A_PING_PETICION

Dato	Long	Valor	Descripción
Tipo	4	0x5	A_PING
Subtipo	4	0x2	ACK
	3	000	Bits de relleno para la MAC
MAC Remitente	29		Dirección MAC del remitente
Resto	24		

Tabla 18: Mensaje A_PING_ACK

8.4. Diálogos de Funcionamiento

Estos mensajes pueden ser generados por cualquier nodo.

8.4.1. Mensaje F_FUNCION

Permite a un nodo ejecutar una función en otro nodo.

Dato	Long	Valor	Descripción
Tipo	4	0x6	F_FUNCION
Subtipo	4	0x0	
ID Función	8		Identificador de la Función
Resto	48		

Tabla 19: Mensaje F_FUNCION

CAPÍTULO 9. CONCLUSIONES

Las aplicaciones creadas han cubierto todos los objetivos propuestos al inicio del proyecto:

1. **Implementación del Software de administración**: Se ha desarrollado una aplicación de Windows que permite definir tanto los elementos del sistema (nodos y dispositivos de entrada/salida) como su comportamiento (funciones y acciones).

2. **Implementación del Software de los nodos**: Se ha creado un firmware común para todos los nodos, que dispone un sistema de configuración que permite variar el comportamiento del nodo en función de las necesidades. La configuración es gestionada por el propio firmware mediante un protocolo de mensajes que se ha definido expresamente para ello.

3. **Implementación del Software de pasarela**: Se ha creado el firmware que va en el nodo pasarela, y que es el responsable de establecer la comunicación entre la aplicación creada para la administración de los nodos con los nodos conectados al bus CAN.

9.1. Futuras líneas de trabajo

Aun le quedan muchas mejoras que implementar a la aplicación de administración y a los firmwares de los nodos, entre ellas las más importantes son:

- Poder utilizar los siguientes dispositivos de entrada/salida:
  - Dispositivos analógicos, como por ejemplo un dimmer para controlar la intensidad de la iluminación.
  - Sensores analógicos, como por ejemplo un sensor de temperatura.
  - Dispositivos digitales que requieran el uso de varias salidas, como por ejemplo un panel de leds.
  - Sensores digitales que requieran el uso de varias entradas, por ejemplo un teclado numérico.
- Optimizar la reprogramación de los nodos para poder reprogramar los nodos de forma individual.
- Se ha detectado una carencia en el dispositivo Hardware de los nodos, la ausencia de una memoria externa para almacenar la configuración. En la versión actual se utiliza la propia memoria flash del chip, lo que limita la vida

de este a 100.000 reprogramaciones. Por lo que se ha sugerido al creador que en futuras versiones incluya una memoria externa.
- Posibilidad de cargar ficheros DWG (Autocad) para definir el diseño del hogar.
- Crear una servidor Web en la aplicación de administración a fin de poder controlar el sistema domótico de forma remota.

Como objetivo a largo plazo, se estudiará la posibilidad de comercializar el sistema completo: los nodos se venderían ya programados con el firmware y la aplicación de administración se distribuiría gratuitamente.

9.2. Valoración Personal

Aunque el trabajo realizado haya sido extenso y se haya podido completar un sistema completo funcional, que en breve se pondrá en explotación en casa de Pere, estimo que se ha cubierto únicamente un 15% de todo el trabajo necesario para hacer una comercialización viable del producto, objetivo a largo plazo, pero que se quedaba fuera del objetivo del proyecto.

Por otro lado considero que se está desaprovechando en exceso el protocolo CAN, ya que se está utilizando para un proyecto que se aleja considerablemente del objetivo principal de este protocolo de comunicación. Se han tenido que descartar todos los tipos de mensaje que dispone, y se ha ignorado el sistema de mascaras y filtros. Esto no es una crítica a la elección de CAN, ya que las pruebas realizadas han demostrado la fiabilidad del protocolo. Lo que quiero decir, es que creo, que el protocolo CAN tiene suficiente versatilidad para ser utilizado en otros ámbitos, en lugar de quedar relegado a la industria de la automoción.

Por último, comentar el único inconveniente que veo a todo el sistema, la necesidad de cablear todo el hogar, tanto para el bus de datos como para alimentación, motivo por el que puede provocar rechazo a futuros usuarios y puede hacernos perder cuota de mercado frente a otras soluciones domótica que son más caras, pero que no requieren de instalación adicional. Por lo que considero, que aunque hemos querido hacer una solución destinada a un público sin conocimientos técnicos, la necesidad de realizar el cableado, aunque es una tarea sencilla, atraerá más a usuarios con unos conocimientos básicos de electrónica.

CAPÍTULO 10. BIBLIOGRAFIA

[1] Sistema domótico basado en CAN, SEP/2010, Pere Joan Antoni Chordá

[2] http://www.todopic.com/

[3] http://www.can-cia.org

[4] http://www.mikroe.com/

[5] http://www.microchip.com/

[6] MICROCONTROLADORES: FUNDAMENTOS Y APLICACIONES CON PIC Pallás, Ramón ; Valdés, Fernando MARCOMBO, EDICIONES TÉCNICAS

[7] http://www.micropic.es

[8] http://www.microcontroladorespic.com

[9] http://www.winpic800.com/

[10] http://www.todopic.com.ar/foros

[11] http://msdn.microsoft.com

[12] Microcontrolador PIC16F84. Desarrollo de proyectos. Enrique Palacios y otros. Ed.: Ra-Ma

[13] El Lenguaje unificado de Modelado. Manual de referencia.. James Rumbaugh, Ivar Jacobson y Grady Booch. Ed. Addison Wesley

[14] The complete reference C# 4.0, Herb Schildt, Ed. McGraw Hill.

[15] http://server-die.alc.upv.es/asignaturas/PAEEES/2005-06/A03-A04%20-%20Bus%20CAN.pdf

ANEXO A : BUS CAN

A.1 Orígenes y arquitectura de capas

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

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

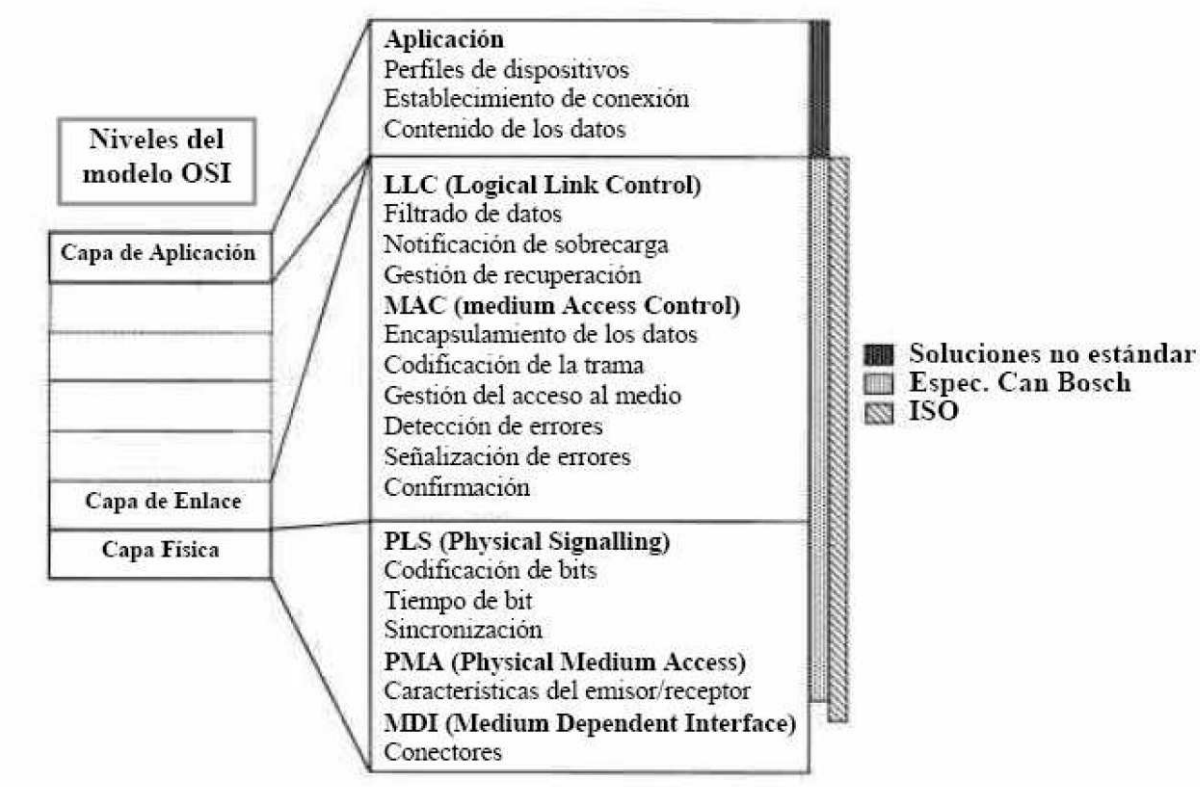


Figura 38: Capa física

A.1.1 Capa física

La capa física de CAN, es responsable de la transferencia de bits entre los distintos nodos que componen la red. Define aspectos como niveles de señal, codificación,

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

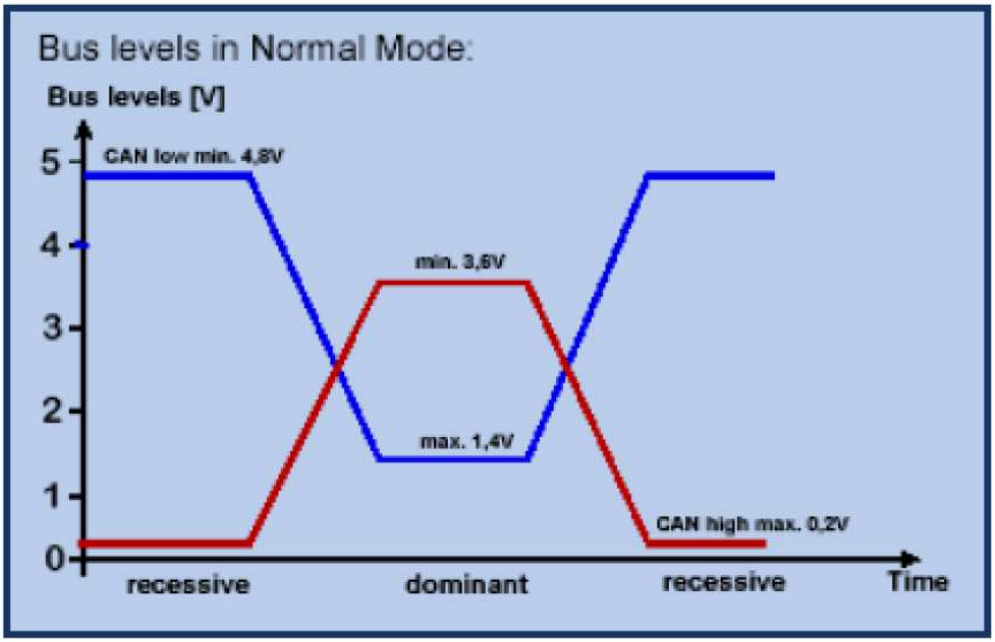


Figura 39: Estándar 11519

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

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

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

A diferencia del bus de alta velocidad, el bus de baja velocidad requiere dos resistencias en cada transceptor: RTH para la señal CAN_H y RTL para la señal CAN_L.

Esta configuración permite al transceptor de bus de baja velocidad (fault-tolerant) detectar fallas en la red. La suma de todas las resistencias en paralelo, debe estar en el rango de 100-500.

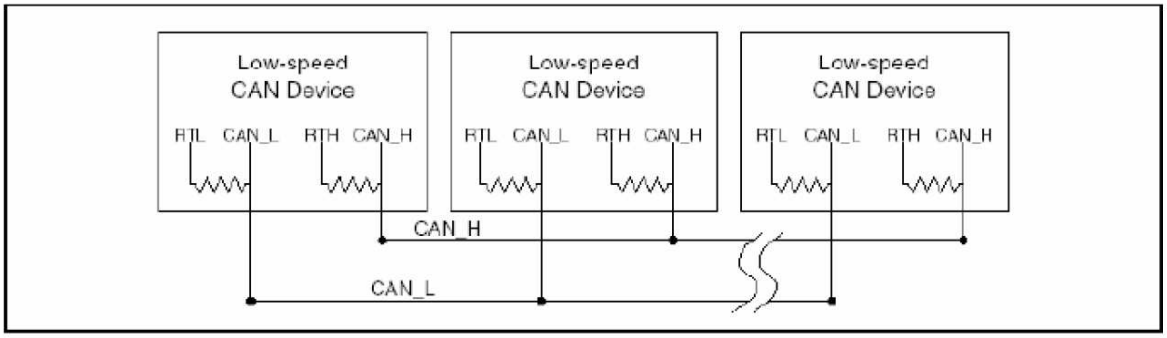


Figura 40: Red Bus CAN de Baja Velocidad (Fault Tolerant)

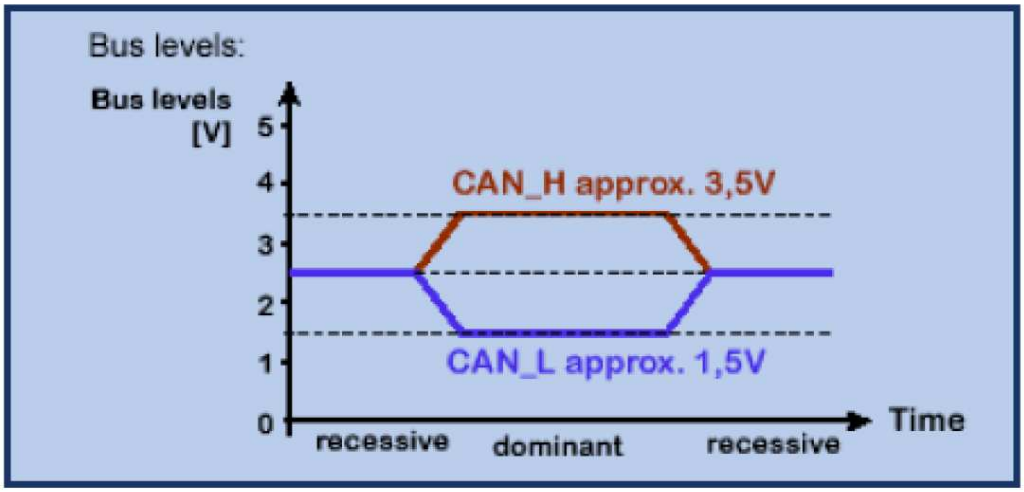


Figura 41: Estándar 11898

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

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

- **Recesivo**: la tensión diferencial (CAN_H - CAN_L) es del orden de 0V con CAN_H = CAN_L = 2.5V (nominales).

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

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

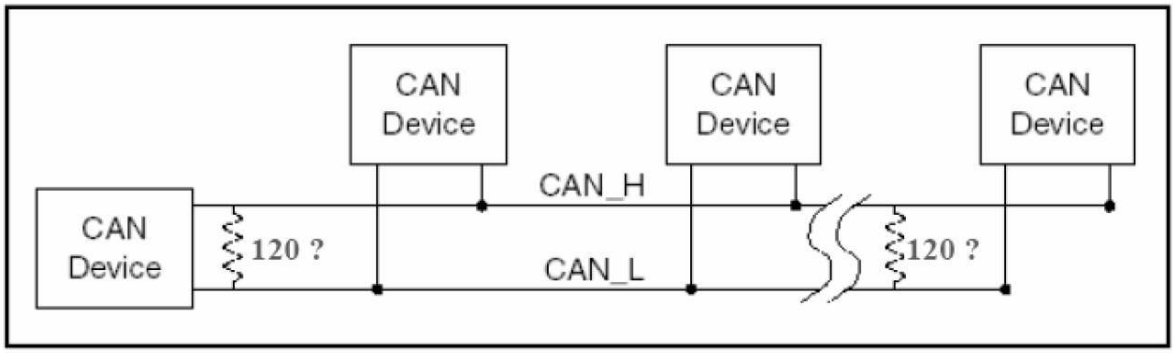


Figura 42: Red Bus CAN de Alta Velocidad

A.1.2 Capa de enlace de datos

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

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

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

A.2 Estructura de un nodo CAN

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

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

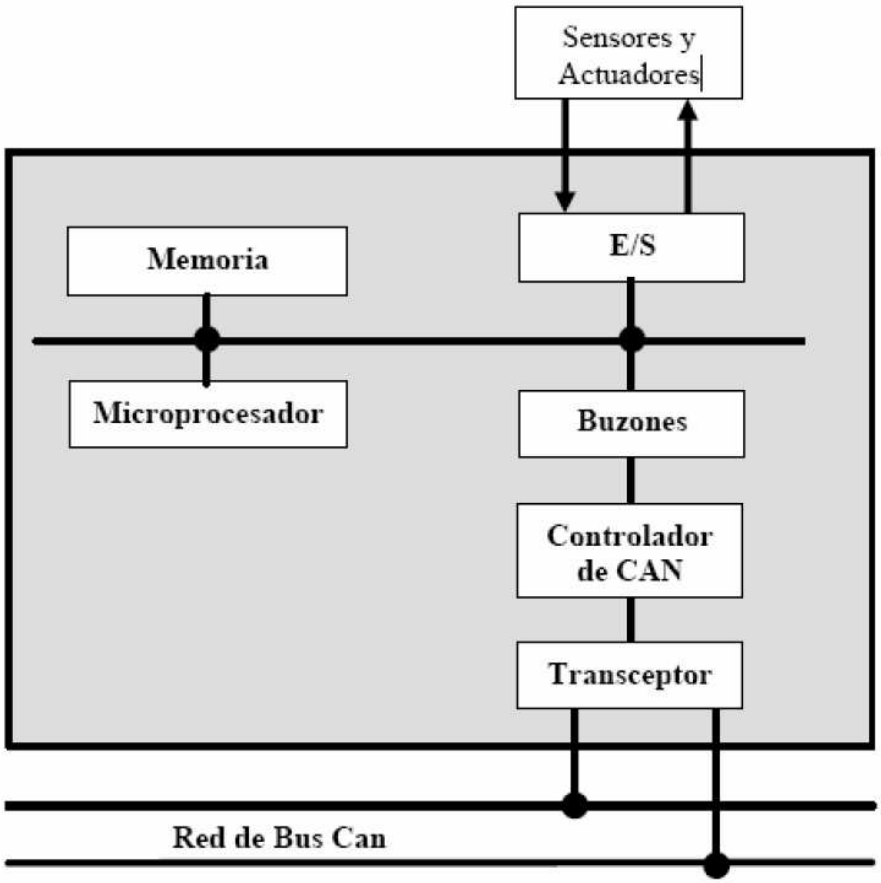


Figura 43: Estructura de un nodo CAN

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

- Para inicializar, el programador especifica los parámetros de los registros de control de interfaz de comunicación, como las características del controlador de red o la velocidad de transmisión.

- A continuación, se actualizan todos los buzones. En cada buzón se especifica si es receptor o transmisor y, su estado inicial, inicializando su parte de datos del búfer.

- Posteriormente, para transmitir un mensaje es necesario poner los datos en el búfer de datos correspondiente al buzón de transmisión y activar el flanco de transmisión.

- Por último, la interfaz de red intenta comunicar los datos a través de la red. El estado de la transferencia se puede comprobar en el estatus de estado de cada buzón.

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

A.3 Formato de codificación y sincronización de datos

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

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

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

A.4 Tramas

A.4.1 Tipo de tramas

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

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

- **Trama de datos**: Se utiliza normalmente para poner información en el bus y la pueden recibir algunos o todos los nodos.
- **Trama de información remota**: Puede ser utilizada por un nodo para solicitar la transmisión de una trama de datos con la información asociada a un identificador dado. El nodo que disponga de la información definida por el identificador la transmitirá en una trama de datos.
- **Trama de error**: Se generan cuando algún nodo detecta algún error definido.
- **Trama de sobrecarga**: Se generan cuando algún nodo necesita más tiempo para procesar los mensajes recibidos.
- **Espaciado entre tramas**: Las tramas de datos (y de interrogación remota) se separan entre sí por una secuencia predefinida que se denomina espaciado inter-trama.
- **Bus en reposo**: En los intervalos de inactividad se mantiene constantemente el nivel recesivo del bus.

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

A.4.2 Trama de datos

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

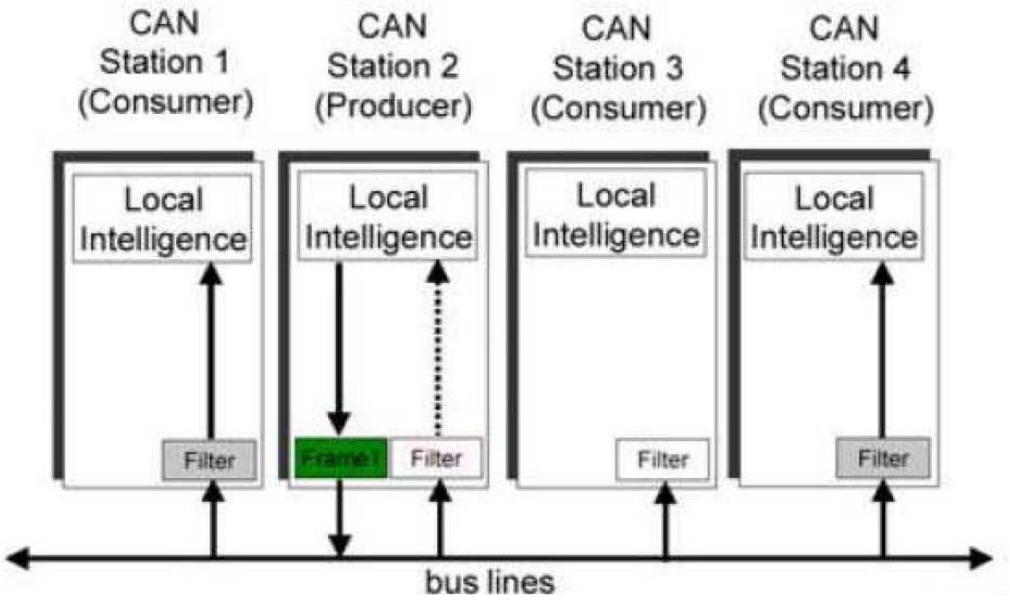


Figura 44: Ejemplo de red CAN

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

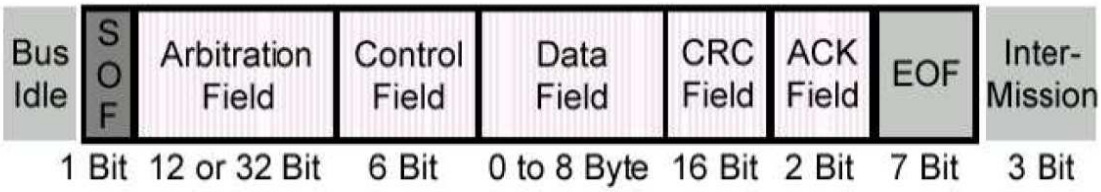


Figura 45: Trama de datos

- **Inicio de trama** (SOF): El inicio de trama es una celda de un sólo bit siempre dominante que indica el inicio del mensaje, sirve para la sincronización con otros nodos.

- **Celda de Arbitraje** (Arbitration Field): Es la celda que concede prioridad a unos mensajes o a otros: En formato estándar tendrá 11 bits seguidos del bit RTR (Remote Transmisión Request) que en este caso será dominante.

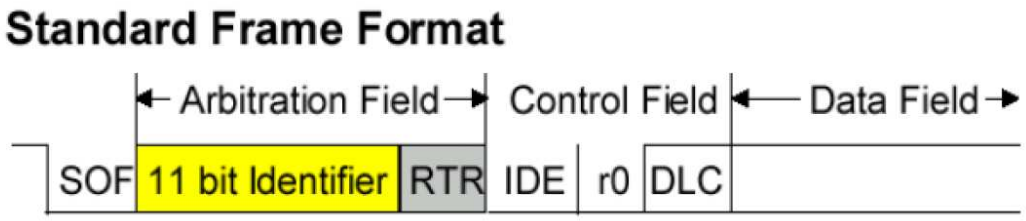


Figura 46: Formato de trama estándar

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

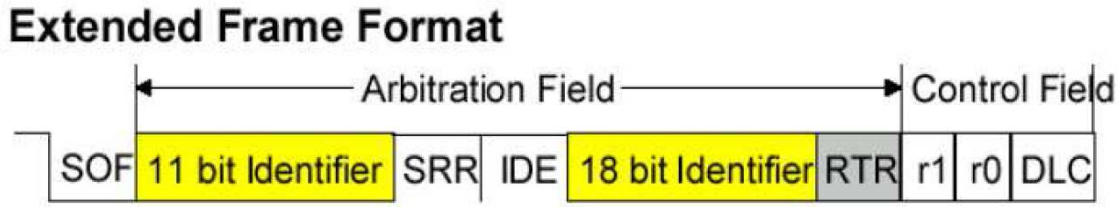


Figura 47: Formato de trama extendido

NOTA: La trama en formato estándar prevalece sobre la extendida

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

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

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

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

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

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

A.4.3 Trama remota (Remote Frame)

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

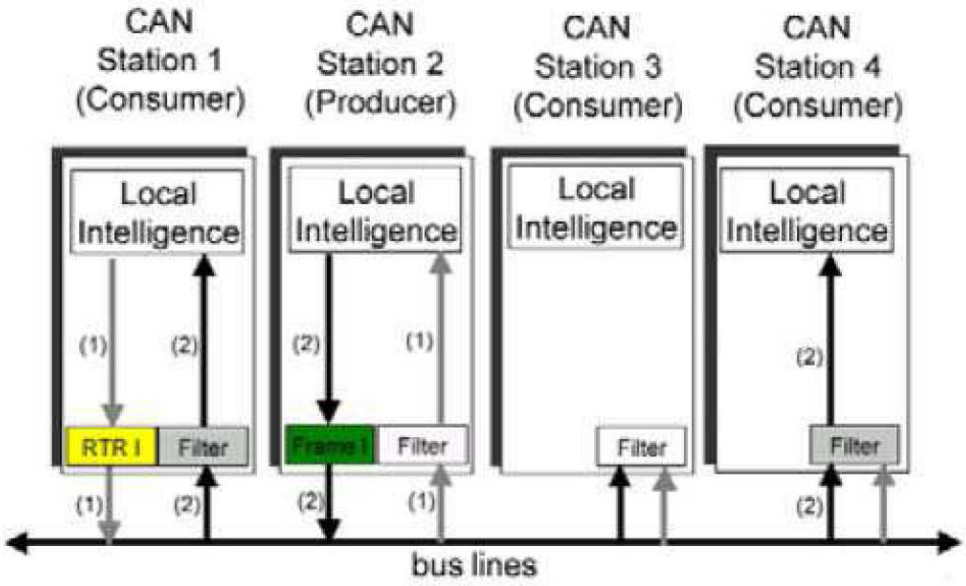


Figura 48: Bus trama remota

Un mensaje de petición remota tiene la siguiente forma:

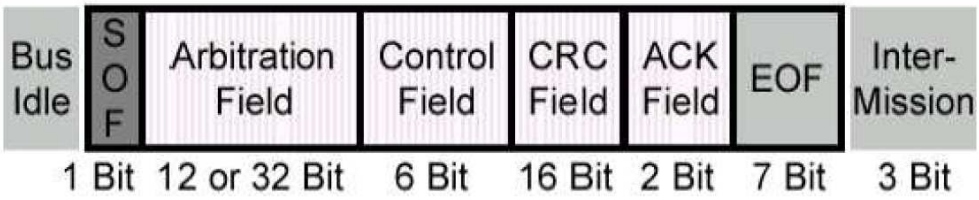


Figura 49: Formato trama remota

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

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

A.4.4 Trama de error

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

El delimitador de error consta de 8 bits recesivos consecutivos y permite a los nodos reiniciar la comunicación limpiamente tras el error. El Indicador de error es distinto según el estado de error del nodo que detecta el error:

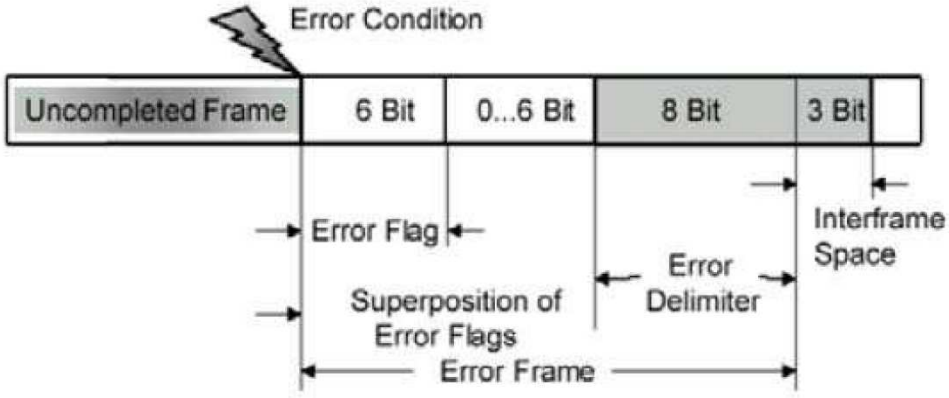


Figura 50: Trama error

Si un nodo en estado de error "Activo" detecta un error en el bus interrumpe la comunicación del mensaje en proceso generando un "Indicador de error activo" que consiste en una secuencia de 6 bits dominantes sucesivos. Esta secuencia rompe la regla de relleno de bits y provocará la generación de tramas de error en otros nodos. Por tanto el indicador de error puede extenderse entre 6 y 12 bits dominantes sucesivos.

Finalmente se recibe el campo de delimitación de error formado por los 8 bits recesivos. Entonces la comunicación se reinicia y el nodo que había sido interrumpido reintenta la transmisión del mensaje. Si un nodo en estado de error "Pasivo" detecta un error, el nodo transmite un "Indicador de error pasivo" seguido, de nuevo, por el campo delimitador de error.

El indicador de error de tipo pasivo consiste en 6 bits recesivos seguidos, por tanto, la trama de error para un nodo pasivo es una secuencia de 14 bits recesivos. De aquí se deduce que la transmisión de una trama de error de tipo pasivo no afectará a ningún nodo en la red, excepto cuando el error es detectado por el propio nodo que está transmitiendo. En ese caso los demás nodos detectarán una violación de las reglas de

relleno y transmitirán a su vez tramas de error.

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

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

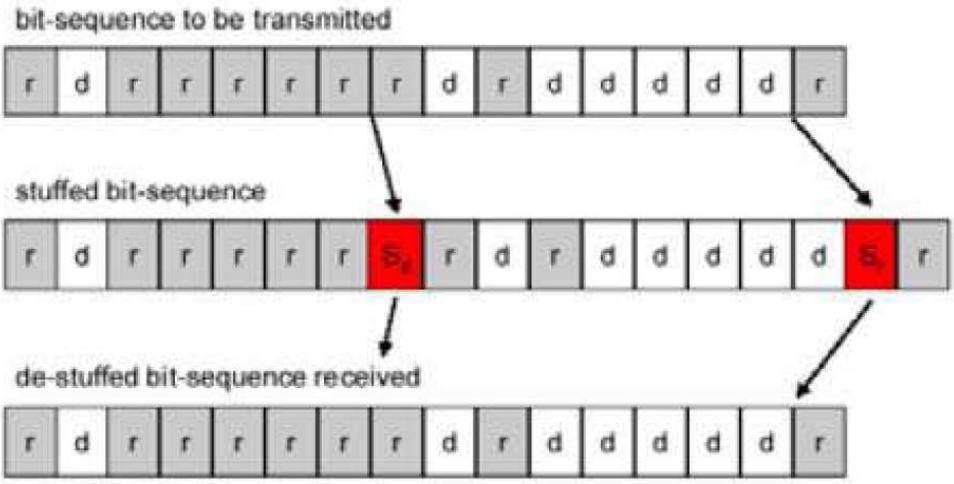


Figura 51: Relleno de bits en trama de error

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

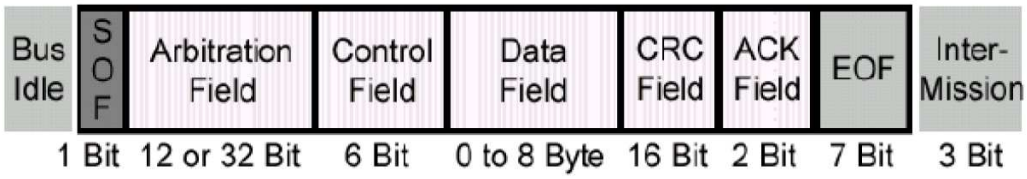


Figura 52: Trama detección de error

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

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

A.4.5 Trama de sobrecarga

Una trama de sobrecarga tiene el mismo formato que una trama de error activo. Sin embargo, la trama de sobrecarga sólo puede generarse durante el espacio entre tramas.

De esta forma se diferencia de una trama de error, que sólo puede ser transmitida durante la transmisión de un mensaje. La trama de sobrecarga consta de dos campos, el Indicador de Sobrecarga, y el delimitador.

El indicador de sobrecarga consta de 6 bits dominantes que pueden ser seguidos por los generados por otros nodos, dando lugar a un máximo de 12 bits dominantes. El delimitador es de 8 bits recesivos.

Una trama de sobrecarga puede ser generada por cualquier nodo que debido a sus condiciones internas no está en condiciones de iniciar la recepción de un nuevo mensaje. De esta forma retrasa el inicio de transmisión de un nuevo mensaje.

Un nodo puede generar como máximo 2 tramas de sobrecarga consecutivas para retrasar un mensaje. Otra razón para iniciar la transmisión de una trama de sobrecarga es la detección por cualquier nodo de un bit dominante en los 3 bits de "intermission".

Por todo ello una trama de sobrecarga de 5 generada por un nodo dará normalmente lugar a la generación de tramas de sobrecarga por los demás nodos dando lugar, como se ha indicado, a un máximo de 12 bits dominantes de indicador de sobrecarga.

A.4.6 Espaciado entre tramas

El espacio entre tramas separa una trama (de cualquier tipo) de la siguiente trama de datos o interrogación remota. El espacio entre tramas ha de constar de, al menos, 3 bits recesivos. Esta secuencia de bits se denomina "intermission".

Una vez transcurrida esta secuencia un nodo en estado de error activo puede iniciar una nueva transmisión o el bus permanecerá en reposo. Para un nodo en estado error pasivo la situación es diferente, deberá espera una secuencia adicional de 8 bits recesivos antes de poder iniciar una transmisión. De esta forma se asegura una ventaja en inicio de transmisión a los nodos en estado activo frente a los nodos en estado pasivo.

A.5 Acceso múltiple y arbitraje de acceso al bus

Unas de las características que distingue a CAN con respecto a otras normas, es su técnica de acceso al medio denominada como CSMA/CD+CR o "Carrier Sense, Multiple Access/Collision Detection + Collision Resolution" (Acceso Múltiple con detección de portadora, detección de colisión más Resolución de colisión).

Cada nodo debe vigilar el bus en un periodo sin actividad antes de enviar un mensaje (Carrier Sense) y además, una vez que ocurre el periodo sin actividad cada nodo tiene la misma oportunidad de enviar un mensaje (Multiple Access). En caso de que dos nodos comiencen a transmitir al unísono se detectará la colisión.

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

La resolución de colisión se basa en una topología eléctrica que aplica una función lógica determinista a cada bit, que se resuelve con la prioridad del nivel definido como bit de tipo dominante. Definiendo el bit dominante como equivalente al valor lógico '0' y bit recesivo al nivel lógico '1' se trata de una función AND de todos los bits transmitidos simultáneamente.

Cada transmisor escucha continuamente el valor presente en el bus, y se retira cuando ese valor no coincide con el que dicho transmisor ha forzado. Mientras hay coincidencia la transmisión continua, finalmente el mensaje con identificador de máxima prioridad sobrevive. Los demás nodos reintentarán la transmisión lo antes posible.

Se ha de tener en cuenta que la especificación CAN de Bosh no establece cómo se ha de traducir cada nivel de bit (dominante o recesivo) a variable física. Cuando se utiliza par trenzado según ISO 11898 el nivel dominante es una tensión diferencial positiva en el bus, el nivel recesivo es ausencia de tensión, o cierto valor negativo, (los transceptores no generan corriente sobre las resistencias de carga del bus).

Esta técnica aporta la combinación de dos factores muy deseados en aplicaciones industriales distribuidas: la posibilidad de fijar con determinismo la latencia en la transmisión de mensajes entre nodos y el funcionamiento en modo multimaestro sin necesidad de gestión del arbitraje, es decir control de acceso al medio, desde las capas de software de protocolo. La prioridad queda así determinada por el contenido del

mensaje, en CAN es un campo determinado, el identificador de mensaje, el que determina la prioridad.

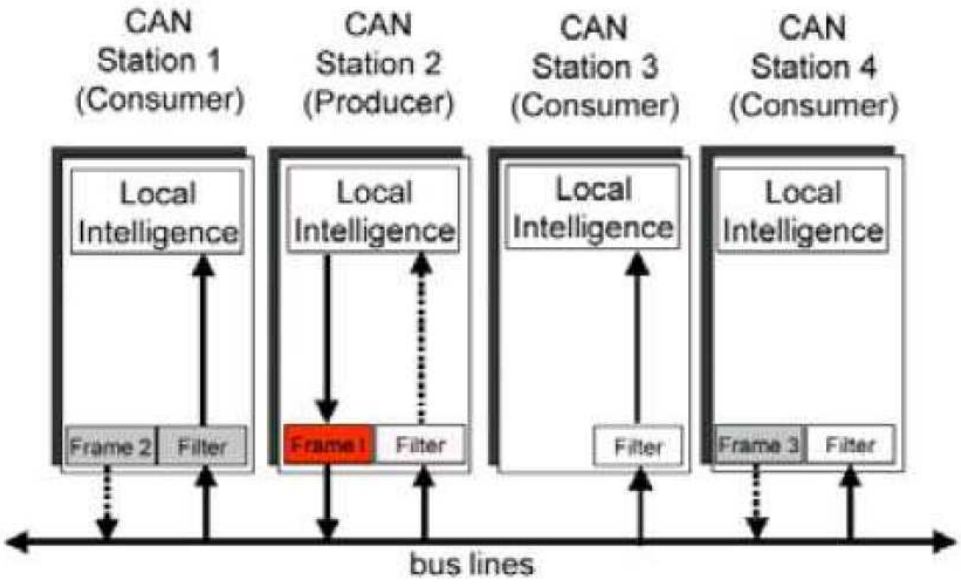


Figura 53: Arbitraje

En un bus único, un identificador de mensaje ha de ser asignado a un solo nodo concreto, es decir, se ha de evitar que dos nodos puedan iniciar la transmisión simultánea de mensajes con el mismo identificador y datos diferentes.

El protocolo CAN establece que cada mensaje es único en el sistema, de manera que por ejemplo, si en un automóvil existe la variable "presión de aceite", esta variable ha de ser transmitida por un nodo concreto, con un identificador concreto, con una longitud fija concreta y coherente con la codificación de la información en el campo de datos.

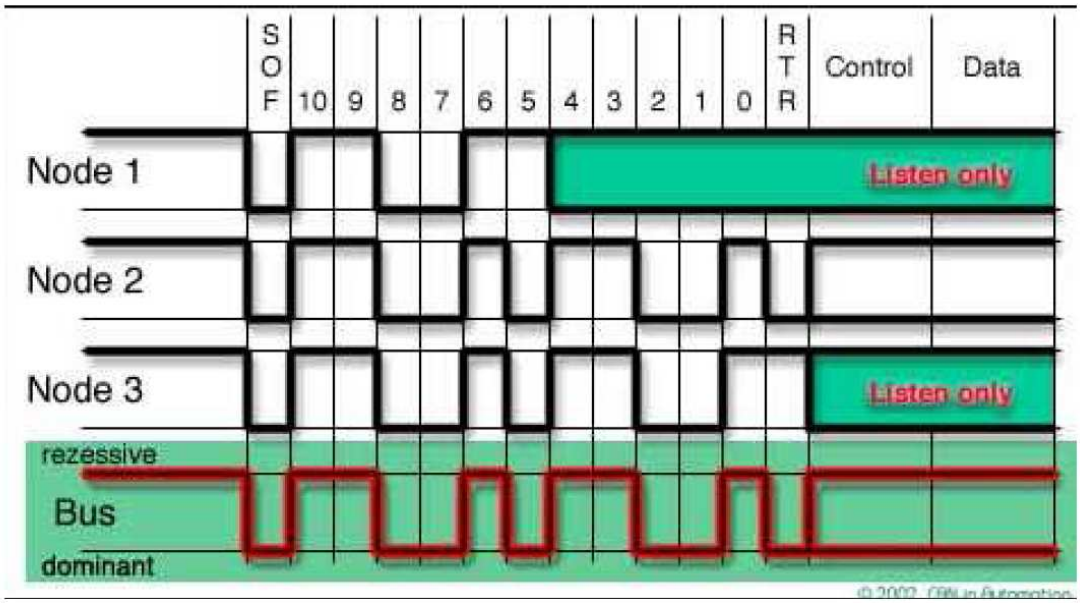


Figura 54: Ejemplo de arbitraje en un bus CAN

A.6 Detección de Errores

Una de las características más importantes y útiles de CAN es su alta fiabilidad, incluso en entornos de ruido extremos, el protocolo CAN proporciona una gran variedad de mecanismos para detectar errores en las tramas. Esta detección de error es utilizada para retransmitir la trama hasta que sea recibida con éxito.

Otro tipo de error es el de aislamiento, y se produce cuando un dispositivo no funciona correctamente y un alto por ciento de sus tramas son erróneas. Este error de aislamiento impide que el mal funcionamiento de un dispositivo condicione el funcionamiento del resto de nodos implicados en la red.

A.6.1 Detección de errores

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

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

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

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

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

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

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

A.6.2 Aislamiento de nodos defectuosos

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

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

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

- **Anulado** (Bus Off): En este estado, deshabilitará su transceptor y no participará en la comunicación. La evolución entre estos estados se basa en dos contadores incluidos en el controlador de comunicaciones. Contador de errores de transmisión (TEC) y Contador de errores de recepción (REC).

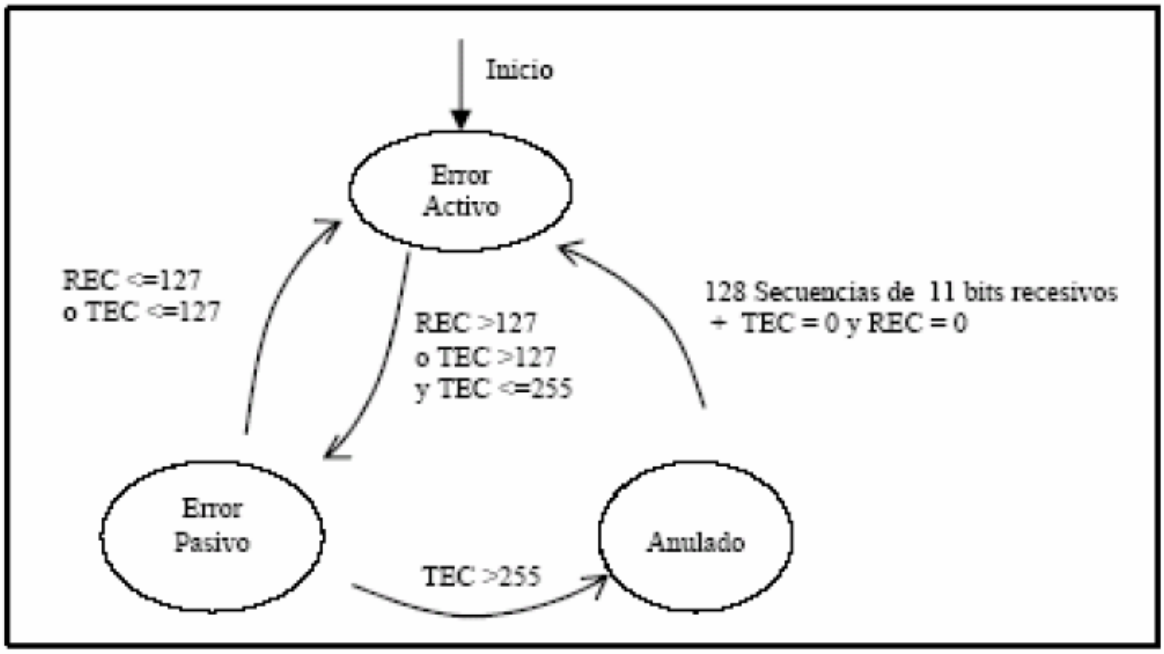


Figura 55: Evolución entre estados de error.

A.7 Acceso múltiple y arbitraje de acceso al bus

Como ya se comentó anteriormente en el apartado A.1, la mayor parte de las aplicaciones que posee el CAN están concentradas en la industria automovilística donde realiza el control del motor, de la mecánica del automóvil así como los sistemas de entretenimiento.

Esto es debido en parte a que el sistema CAN fue especialmente diseñado en sus orígenes con la idea de aplicarlo en el mundo de los automóviles para la transmisión de datos entre los sistemas electrónicos de control y regulación, como por ejemplo:

- Control del cambio.
- Control electrónico del motor o de la bomba de inyección.
- Sistema antibloqueo (ABS).
- Sistema de tracción antideslizante (ASR).
- Control de estabilidad (ESP).
- Regulación del momento de arrastre del motor (MSR).
- Inmovilizador.
- Ordenador de a bordo, etc.

Relacionado con la industria automovilística, el CAN se extiende al transporte público y otras máquinas móviles como aviones, helicópteros, trenes, barcos y los controles de tráfico y sistemas de información conductor/pasajero.

Pero en general el CAN se aplica en cualquier sistema de control industrial: sistemas de control de plantas y maquinaria, redes entre máquinas, maquinaria agrícola, instrumental médico, sistemas de supervisión, etc. y en la automatización de edificios: control de ascensores, aire acondicionado, sistemas de calefacción y refrigeración, control de iluminación, etc.

En resumen, en cualquier sistema que precise control en tiempo real distribuido y con escaso flujo de datos. Por lo tanto, esto augura un gran futuro para el CAN. Incluso actualmente ya se puede considerar que el CAN a alcanzado un nivel extraordinario de madurez e implantación ya que los fabricantes y procesadores digitales de señal están

incorporando controladores CAN de forma bastante generalizada (se habla de cientos de millones de nodos).

Volviendo al tema de las aplicaciones CAN en vehículos motorizados, cabe decir que existen tres campos de aplicación esenciales para estos sistemas:

- Acoplamiento de unidades de control.
- Electrónica de la carrocería y de confort.
- Comunicación móvil.

De estos tres campos se va a documentar un poco sobre el primero.

A.7.1 Acoplamiento de unidades de control

Se acoplan entre sí sistemas electrónicos como el control del motor o de bomba de inyección, sistema antibloqueo ABS, sistema de tracción antideslizante ASR o regulación de la dinámica de marcha ESP, control electrónico de cambio, etc. Las unidades de control están aquí unidas como estaciones con igualdad de derechos, mediante una estructura de bus lineal.

Esta estructura presenta la ventaja de que en caso de fallar una estación, el sistema bus continúa estando plenamente a disposición de las demás estaciones. En comparación con otras disposiciones lógicas (estructuras anulares o estructuras en estrella) se reduce así esencialmente la probabilidad de un fallo total. En el caso de estructuras anulares o en estrella, el fallo de una estación o de la unidad central, conduce a un fallo total.

Las velocidades de transmisión típicas están entre aprox. 125 Kbps y 1 Mbps (ejemplo: la unidad de control del motor y la unidad de control de bomba en la regulación electrónica diesel comunican entre sí a 500 Kbps). Las velocidades de transmisión deben ser tan altas para poder garantizar el comportamiento de tiempo real requerido. En las siguientes figuras vemos dos diferentes ejemplos:

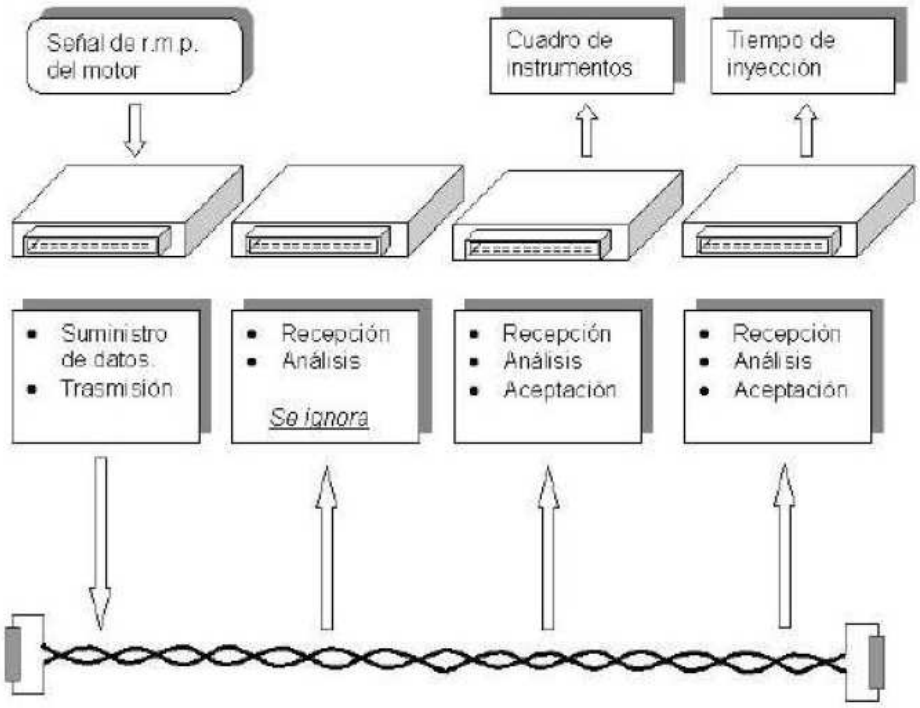


Figura 56: Ejemplo 1 sobre un sistema para automóvil basado en CAN

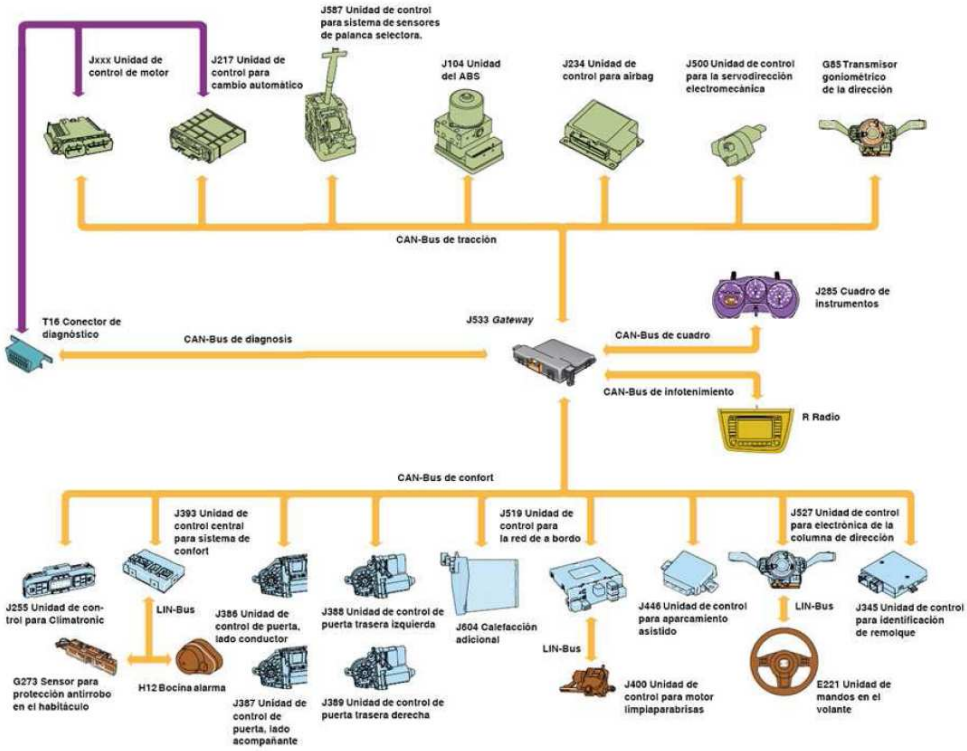


Figura 57: Ejemplo 2 sobre un sistema para automóvil basado en CAN

A.7.2 Diagnóstico

Tal como se ha explicado en este mismo anexo, el protocolo CAN dispone de una serie de mecanismos de control para el reconocimiento de anomalías. Si una estación registra una anomalía, emite entonces un "flag de error", que detiene la transmisión en curso. De esta forma se impide que otras estaciones reciban el mensaje erróneo.

Gracias a este sistema de seguridad que incorpora el CAN se consigue que las probabilidades de fallo en el proceso de comunicación sean muy bajas, pero sigue siendo posible que cables, contactos y las propias unidades de mando presenten alguna disfunción.

Por ello, para el análisis de una avería, se debe tener presente que una unidad de mando averiada abonada al CAN en ningún caso impide que el sistema trabaje con normalidad. Lógicamente no será posible llevar a cabo las funciones que implican el uso de información que proporciona la unidad averiada, pero sí todas las demás.

Una alternativa posible para localizar fallos en el CAN es emplear el programa informático CANAlyzer (Vector Informatik GmbH) con el ordenador y con la conexión adecuada. Este programa permite visualizar el tráfico de datos en el bus CAN, indica el contenido de los mensajes y realiza la estadística de mensajes, rendimiento y fallos.

A.8 Especificaciones

CAN tiene dos formatos diferentes para transmitir datos: el formato de trama estándar (Standard Frame), según la especificación CAN 2.0A y, el formato de trama extendida (Extended Frame) según la especificación CAN 2.0B.

La principal diferencia entre ambos es la longitud del identificador del mensaje (ID), que en el caso de la trama estándar es de 11 bits (2032 identificadores, ya que los 16 bits identificadores de menor prioridad están reservados) y en el caso de la extendida es de 29 bits (más de 536 millones de identificadores):

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

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

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

A.9 Implementaciones

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

A.9.1 Stand-Alone CAN Controller

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

- **Un microcontrolador** como puede ser el 16F877, con el que se ha venido trabajando a lo largo de todo este proyecto.
- **Un controlador CAN** que introduzca el protocolo CAN, filtro de mensajes,… y todo el interface necesario para las comunicaciones. Un ejemplo de controlador CAN es el MCP2510 cuya DATA SHEET se puede encontrar en la página Web de microchip.
- **Un transceiver CAN**, esto es, un transmisor/receptor que puede ser por ejemplo el MCP2551 desarrollado por microchip.

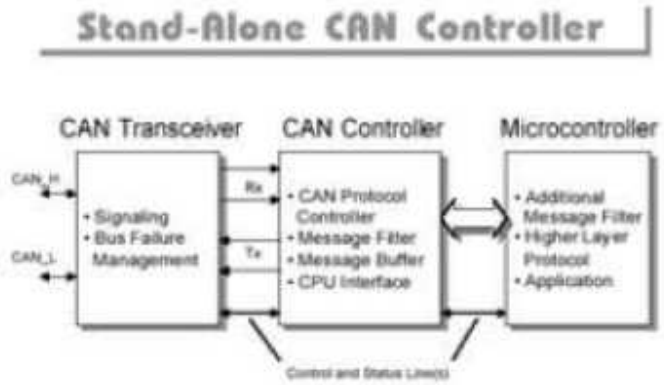


Figura 58: Stand-Alone CAN Controller

Así pues, si de alguna manera se pretende trabajar en una red CAN con una placa de pruebas, en un principio no sería factible a no ser que de alguna manera se le acoplasen el controlador y el transceptor CAN correspondiente. Pero Microchip ha creado una placa de pruebas específica para este tipo de comunicaciones, que desarrolla además este tipo de arquitectura y que se puede conseguir también a través de la página Web de microchip.

A.9.2 Integrated CAN Controller

Este tipo de arquitectura consiste en un microcontrolador que incluya, no sólo sus características propias sino además un módulo CAN con las características de un microcontrolador CAN.

El transceiver se sitúa de manera separada. Este es el caso de nuestro microcontrolador PIC18F2580. El módulo del transceptor actúa como un controlador de línea, balanceando la señal, esto es, adecuando los niveles de tensión de esta al exterior.

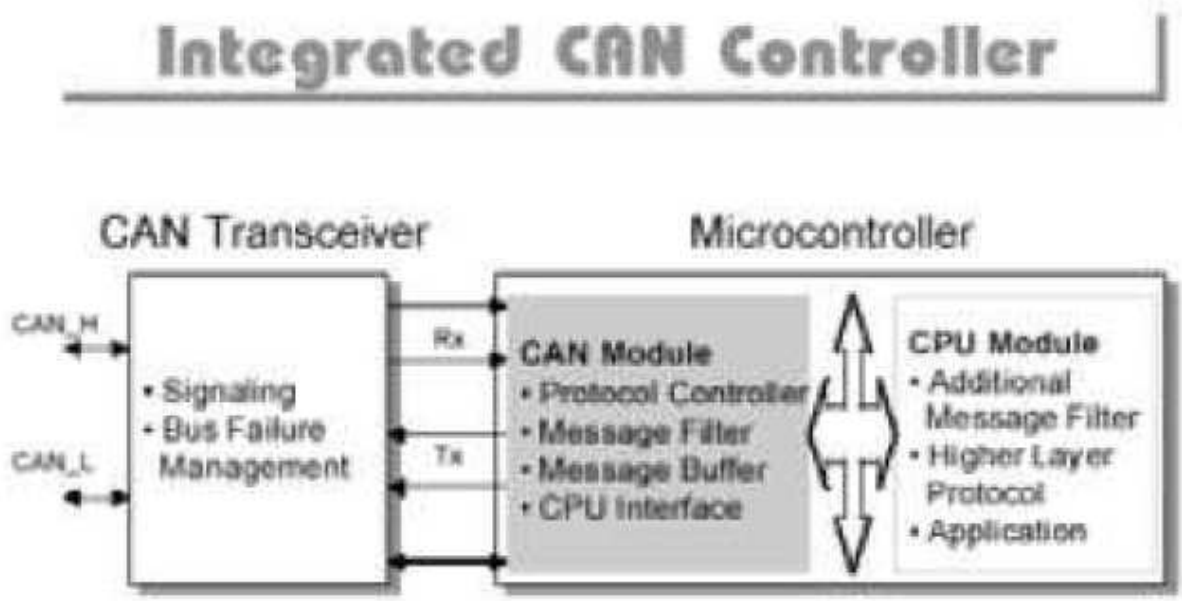


Figura 59: Integrated CAN Controller

A.9.3 Single-chip CAN Node

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

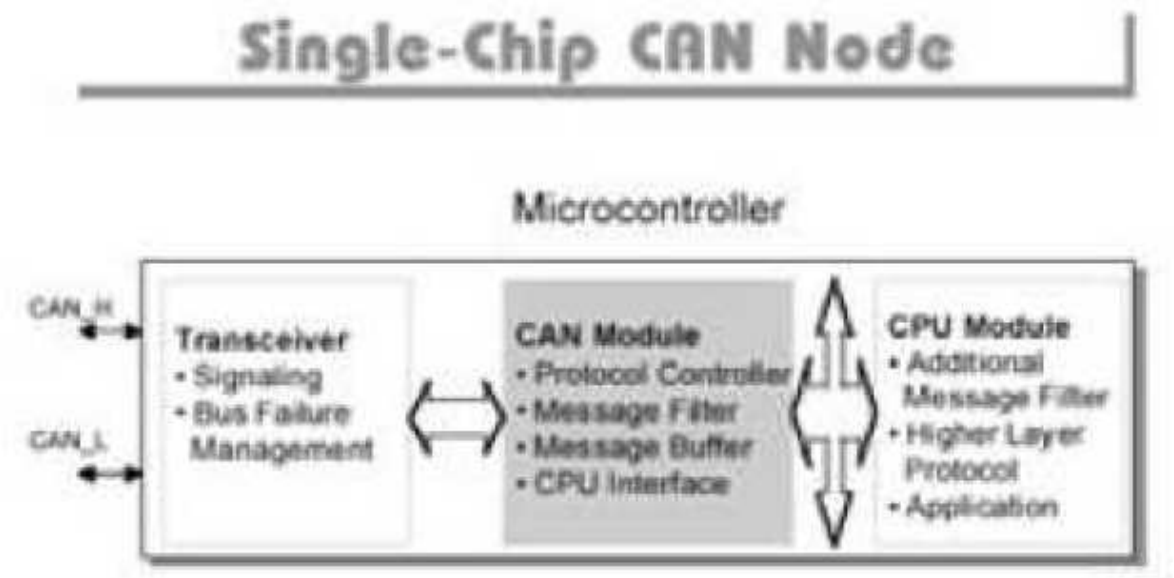


Figura 60: Single-Chip CAN Node

ANEXO B : CÓDIGO FUENTE

El objetivo de este anexo es mostrar parte del código generado para el PIC, a fin de comentar las secciones que se han considerado más significativas.

Se han definido 2 enumerados, boolean para simular el tipo boleano, y acción para no tener que recordar los códigos de cada tipo de mensaje.

```
enum boolean {
      FALSE = 0,
      TRUE = 1
};

enum accion{
     C_MAC = 0x10,
     C_MAC_ACK= 0x1C,
     C_ACCION_LOCAL = 0x21,
     C_ACCION_REMOTA = 0x22,
     C_FUNCION = 0x30,
     C_BORRAR_FUNCIONES = 0x41,
     C_BORRAR_ACCIONES = 0x42,
     A_ESTADO = 0x51,
     A_ESTADO_ACK = 0x52,
     A_RESET = 0x40,
     A_PING_PETICION = 0x51,
     A_PING_ACK = 0x52,
     F_FUNCION = 0x60
};
```

La estructura trama se utiliza para que sea más cómodo trabajar con el envío y recepción de mensajes por CAN

```
struct trama{
  unsigned char accion;

  unsigned long remitente;
  unsigned long destinatario;

  unsigned long nueva_mac;

  unsigned char layout;
  unsigned char valor;

  unsigned char rcv;
```

```
  unsigned char error;
  unsigned char flags;
  unsigned char len;
  unsigned char datos[8];
};
```

Se utilizan dos variables globales (datos_rec y datos_env) para el envío y recepción de mensajes de CAN

```
struct trama datos_rec;
struct trama datos_env;
```

El array buffer se utiliza para almacenar los caracteres recibidos por 232 hasta que son procesados. El código fuente es el mismo tanto para el nodo como para la pasarela, y es mediante la variable IS_232 como se indica como va a comportarse el nodo.

```
unsigned char buffer_232[64];
unsigned char IS_232 = TRUE;
```

Mediante estas variables se define donde comienzan los distintos bloques de memoria. De esta forma, si el tamaño del programa crece demasiado, se puede desplazar el inicio de los demás bloques sin necesidad de modificar el resto de código.

```
unsigned long MEM_MAC = 0x2000;
unsigned long MEM_ACCIONES = 0x2040;
unsigned long MEM_FUNCIONES = 0x3000;
```

El main, después de inicializar todos los datos, ejecuta un bucle infinito, que lo único que hace es leer el bus CAN y, si el código es para la pasarela, el bus 232.

```
void main() {

     Init_IO();
     Init_CAN();
     Init_DATOS();
```

```
        LATA = 0x00;

        while( TRUE ) {
               if(IS_232){
                    Leer_232();
               }
               Leer_Can();
        }
    }
```

El método Init_DATOS, lee la MAC del nodo, y si esta es 0x00000000, se queda en un bucle hasta que recibe el mensaje de cambiar la MAC.

```
    void Init_DATOS(void){
        unsigned char esperar;

        FLASH_Read_N_Bytes(MEM_MAC,buffer,4);

        MAC = 0x00000000;
        MAC |= buffer[0]<<8;
        MAC |= buffer[1];
        MAC |= MAC<<16;
        MAC |= buffer[2]<<8;
        MAC |= buffer[3];

        if( MAC == 0x00000000 ){
            esperar = TRUE;
            while( esperar ){
                   Leer_Can();
                   if( MAC != 0x00000000 ){
                       do_BORRAR();
                       esperar = FALSE;
                   }
            }
        }
    }
```

El método Leer_CAN, intenta leer un mensaje del bus de CAN, y si el mensaje es para el nodo, se llamará al método Parser_CAN.

```
    void Leer_CAN(void){
        unsigned char i;
```

```
          datos_rec.rcv  =  CANRead(&datos_rec.destinatario,  datos_rec.datos  ,
&datos_rec.len, &datos_rec.flags);
          datos_rec.accion = NADA;

          if( datos_rec.rcv ) {
              if( datos_rec.destinatario == MAC ){
                  for( i = datos_rec.len ; i < 8 ; i++){
                       datos_rec.datos[i]=0x00;
                  }
                  Parser_CAN();
              }
          }
     }
```

El método Parser_CAN, lo único que tiene es un switch con un case para cada uno de los tipos de mensaje que se pueden recibir, y llama al método que corresponda en cada caso.

Función para modifica la MAC. Como el procedimiento para escribir en la memoria flash trabaja en bloques de 64 bytes, primero hay que leer un bloque entero, modificar la parte que queremos y escribimos todo el bloque.

```
     void do_CAMBIAR_MAC(void){
          FLASH_Read_N_Bytes(MEM_MAC,buffer,64);
          MAC = datos_rec.nueva_mac;
          buffer[0] = MAC>>24 & 0xFF;
          buffer[1] = MAC>>16 & 0xFF;
          buffer[2] = MAC>> 8 & 0xFF;
          buffer[3] = MAC     & 0xFF;
          FLASH_Erase_Write_64(MEM_MAC, buffer);
     }
```

Función para añadir una nueva acción. En este caso, hay que modificar el bit correspondiente a la función que añadimos en la zona de configuración y calcular el bloque y la posición en el que hay que escribir los datos de la acción.

```
     void do_CONFIGURAR_ACCION(void){

          datos_rec.accion--;
          bloque = datos_rec.accion>>3;
          offset = bloque<<3 & 0xFF;
          offset = datos_rec.accion - offset;

          FLASH_Read_N_Bytes(MEM_MAC,buffer,64);
```

```
        buffer[bloque+16] |= 0x40>>offset;
        FLASH_Erase_Write_64(MEM_MAC, buffer);
        offset = offset<<3 & 0xFF;
        FLASH_Read_N_Bytes(MEM_ACCIONES + ( bloque * 64), buffer, 64 );
        buffer[offset] = datos_rec.tipo;
        if( datos_rec.tipo == C_ACCION_REMOTA){
            buffer[offset+1] = datos_rec.nueva_mac>>24 & 0xFF;;
            buffer[offset+2] = datos_rec.nueva_mac>>16 & 0xFF;
            buffer[offset+3] = datos_rec.nueva_mac>>8 & 0xFF;
            buffer[offset+4] = datos_rec.nueva_mac & 0xFF;
            buffer[offset+5] = datos_rec.funcion_remota;
        }else{
            buffer[offset+1] = datos_rec.entrada_mascara;
            buffer[offset+2] = datos_rec.entrada_valor;
            buffer[offset+3] = datos_rec.salida_mascara;
            buffer[offset+4] = datos_rec.salida_valor;
            buffer[offset+5] = datos_rec.tiempo;
        }
        buffer[offset+6] = datos_rec.funcion_next;
        buffer[offset+7] = 0x00;
        FLASH_Erase_Write_64(MEM_FUNCIONES + ( bloque * 64), buffer);
    }
```

Para borrar las funciones, solo hay que poner a 0 los bits de la parte de configuración.

```
    void do_BORRAR_FUNCIONES(void){
        unsigned char i;

        FLASH_Read_N_Bytes(MEM_MAC,buffer,64);
        for( i=16 ; i<32 ; i++ ){
            buffer[i]=0x00;
        }
        FLASH_Erase_Write_64(MEM_MAC, buffer);
    }
```

El método "do_EJECUTAR_ACCION", puede ser llamado como resultado de una "función" o por una llamada de ejecutar una función remota.

```
    void do_EJECUTAR_ACCION(unsigned char acc){
        unsigned char mascara;
        unsigned char valor;
        unsigned char tiempo;
        unsigned char i;

        acc--;
        bloque = acc>>3;
        offset = bloque<<3 & 0xFF;
        offset = acc - offset;
        FLASH_Read_N_Bytes(MEM_MAC,buffer,64);
```

```
        if( buffer[bloque+16] & 0x40>>offset ){
            offset = offset<<3 & 0xFF;
            FLASH_Read_N_Bytes(MEM_ACCIONES + ( bloque * 64), buffer, 64 );
            if( buffer[offset] == C_ACCION_REMOTA ){
                datos_env.tipo = F_FUNCION;
                datos_env.destinatario = 0x00000000;
                datos_env.destinatario |= buffer[offset+1]<<24;
                datos_env.destinatario |= buffer[offset+2]<<16;
                datos_env.destinatario |= buffer[offset+3]<<8;
                datos_env.destinatario |= buffer[offset+4];
                datos_env.funcion = buffer[offset+5];
                Enviar_CAN();
            }else{
                  tiempo = buffer[offset+5];
                  mascara = buffer[offset+1];
                  valor = buffer[offset+2];
                  for( i=0; i<8 ; i++ ){
                       if( mascara & 0x40>>i){
                           if( valor & 0x40>>i ){
                               Tiempos_Entradas[i]=tiempo;
                           }else{
                                 Tiempos_Entradas[i]=0x00;
                           }
                       }
                  }

                  mascara = buffer[offset+3];
                  valor = buffer[offset+4];
                  for( i=0; i<8 ; i++ ){
                       if( mascara & 0x40>>i){
                           if( valor & 0x40>>i ){
                               Tiempos_salidas[i]=tiempo;
                           }else{
                                 Tiempos_salidas[i]=0x00;
                           }
                       }
                  }
            }
        }
    }
```

El método "Comprobar_Salidas" es el que se encarga cambiar el estado de las salidas. Está asociado a una interrupción que se lanza cada 250ms.

```
    void Comprobar_Salidas(void){
       unsigned char N_LATA;
       unsigned char i;

       N_LATA = 0x00;

       for( i=0 ; i<8 ; i++){
```

```
            if( Tiempos_Salidas[i] > 0 ){
                N_LATA |= 0x40>>i;
                if( Tiempos_Salidas[i] < 0xFF ){
                    Tiempos_Salidas[i]--;
                }
            }
        }
        LATA = N_LATA;
    }
```

El método "Comprobar_Entradas" es el encargado de comprobar el estado las entradas, y de lanzar una acción cuando se cumple alguna función. Al igual que "Comprobar_Salidas" está asociado a una interrupción que se lanza cada 250ms

```
    void Comprobar_Entradas(void){
         unsigned char acc_act;
         unsigned char tiempo;


         entradas_actual = PORTC;
         if( entradas_anterior != entradas_actual  && ~entradas_actual ){

             estado_actual = pulso_XX(entradas_actual,2);
             tiempo = 2 + pulso_TM(entradas_actual);
             if( estado_actual && (NUM_ACCIONES > 0) ){

                 for( acc_act = 0 ; acc_act < NUM_FUNCIONES ; acc_act++){
                      FLASH_Read_N_Bytes(MEM_ACCIONES  +  (  acc_act  *  8),
buffer, 8 );
                      if( !( (estado_actual & buffer[0]) ^ buffer[1] ) ){
                          if( tiempo >= buffer[4] ){
                              do_EJECUTAR_ ACCION (buffer[5]);
                              entradas_anterior = entradas_actual;
                              return;
                          }
                      }
                 }
             }
         }
         entradas_anterior = entradas_actual;
    }
```

ANEXO C : Palabras clave

CAN, PIC, PIC18, PIC18F2580, RS232, MCP2551, DOMOTICA, MICROCHIP