Universidad de Valladolid

Escuela Técnica Superior
de Ingenieros de Telecomunicación

Proyecto Fin de Carrera
Ingeniero en Electrónica

Diseño y realización de un analizador del protocolo de comunicaciones CAN bus mediante microcontroladores de la familia PIC18FXX8

Autores:
Miguel Ángel Pérez Herrero
Carlos Gobernado Tejedor
Tutor:
Jesús M. Hernández Mangas

Valladolid, 14 de Octubre de 2004

TÍTULO:	**Diseño y realización de un analizador del protocolo de comunicaciones CAN bus mediante microcontroladores de la familia PIC18FXX8**
AUTORES:	**Miguel Ángel Pérez Herrero** **Carlos Gobernado Tejedor**
TUTOR:	**Jesús M. Hernández Mangas**
DEPARTAMENTO:	**Electricidad y Electrónica**

Miembros del tribunal

PRESIDENTE:	Jesús Arias Álvarez
VOCAL:	Luis Quintanilla Sierra
SECRETARIO:	Jesús M. Hernández Mangas

FECHA DE LECTURA:

CALIFICACIÓN:

Resumen del proyecto

La modernización de distintas aplicaciones industriales tales como la automatización de procesos, la electrónica del automóvil, equipamientos médicos o el control de equipos exige sistemas de comunicaciones cada vez más sofisticados, que presenten una fiabilidad y robustez máximas. El protocolo de comunicaciones bus *CAN*, desarrollado por Bosch, es una solución que reúne estos requisitos y que ha ido ganándose un reconocimiento en el sector ante el alto rendimiento prestado.

El objetivo de este proyecto es el diseño y realización de un analizador del protocolo de comunicaciones bus *CAN* con la finalidad de constituir una herramienta para el diagnóstico de posibles errores de redes *CAN*. Este analizador es capaz de detectar todos los tipos de tramas en distintas configuraciones posibles, y para distintos valores de la tasa de transmisión.

Palabras clave

Analizador, Bus *CAN*, Microcontrolador, PIC18FXX8, PIC18F258, Tarjeta de Memoria, MMC, Multimedia Card, SD, Secure Digital, C18.

Abstract

The modernization of different industrial applications such as the automatization of processes, electronics of the automobile, medical equipment or the control of equipment demands more reliability, sophistication and robustness from communication systems. The communication protocol *CAN* bus, developed by Bosch, is a solution that meets these requirements. It has received recognition in the sector for its high performance.

The objectives of the project are the design and realisation of an analyzer of the communication protocol *CAN* bus. It will be used as a tool for the diagnosis of possible errors in a *CAN* network. This analyzer can detect all types of frames in different configurations and for different values of the rate of transmission.

Key words

Analyzer, Bus *CAN*, Microcontroller, PIC18FXX8, PIC18F258, Memory Card, MMC, Multimedia Card, SD, Secure Digital, C18.

"El futuro está oculto detrás de los hombres que lo hacen"

Anatole France

Agradecimientos

En estas líneas tan personales quiero agradecer de una manera sincera a todas aquellas personas que de una forma u otra han colaborado en la elaboración de este proyecto. En especial me gustaría destacar:

A nuestro tutor Jesús M. Hernández Mangas, por su ayuda y disposición en todo momento. Sus consejos y comentarios han contribuido mucho en la elaboración de este proyecto.

A mi compañero Miguel, por ser un estupendo compañero. Con compañeros así siempre es fácil trabajar en grupo.

A mi novia Irma por su apoyo, comprensión y paciencia en estos últimos meses.

A mi hermano Álvaro, por facilitarme en todo lo posible la elaboración de este proyecto.

A mi amigo Oscar por la gran ayuda que nos ha prestado a la hora de imprimir este documento.

A mis Padres...

Carlos Gobernado Tejedor

En primer lugar, quiero mostrar mi más sincero agradecimiento a Jesús M. Hernández Mangas, por su gran predisposición al aceptar el tutelaje de nuestro proyecto y por los buenos e interesantes consejos recibidos para el desarrollo del mismo.

A Carlos, mi compañero de fatigas en el proyecto, también le tengo que dar las gracias porque gran parte de los conocimientos y experiencia adquiridos se los debo a él. Buenos han sido los momentos disfrutados en el laboratorio, peleándonos con la electrónica y discutiendo sobre distintos temas.

En otro plano, tengo que incluir en estas líneas el apoyo prestado por mi familia: mi hermana Áurea y mi cuñado Jorge, que siempre han estado ahí, mi padre y mis dos abuelas. Los amigos que estuvieron en mi lugar de estudios: Gustavo, Carlitos, Elena, Fernando, Eduardo, María, Cris, Joaquín, Domingo. Los que estuvieron en mi segundo hogar, la R.U. Alfonso VIII: Raúl, Javi B., Javi G., Fer, Kike, Pablo, Ina, Rober, Juan, Toñín, Jazin, Diego, Yeyo, Alberto F., Stef, Mariajo, Ele, Antonio, Alberto M. Y con los que hemos dado mucha guerra: Agus, Nach, Ignacio, José, Elena, Merche, Carlos M. y el resto del clan Titán. Todos ellos, aunque algunos sea ahora desde la lejanía, te proporcionan ese aliento extra que siempre te da el empujón final en un sprint, y que sin él, dudo si habría llegado a estar en estos momentos escribiendo esto.

Finalmente, no me olvido de Susana, que allende mar no ha dejado de animarme, mostrarme todo su cariño y hacer que siga ilusionándome en la vida. Esta memoria te va dedicada.

Miguel Ángel Pérez Herrero

Índice General

Índice de Figuras

Índice de Tablas

Índice de listados

Capítulo 1

Introducción

1.1 Motivación y descripción general

La invención del transistor por Bardeen, Brattain, y Shockley dota a los equipos de transmisión y recepción de señales de un tamaño reducido y una mayor potencia, convirtiendo en socios ya inseparables a los sistemas de comunicaciones y la electrónica. La evolución que ambos campos han sufrido en los últimos 50 años ha tenido dimensiones enormes.

Hoy día navegamos por internet, vemos TV por satélite, somos capaces de establecer una vídeoconferencia entre distintos puntos del planeta, podemos conocer nuestra situación con una precisión extraordinaria gracias al GPS, la industria maneja robots interconectados que precisan de protocolos de comunicación especiales, muchos de los aparatos y dispositivos que manejamos constan de una serie de módulos que realizan tareas dependientes de los demás y que necesitan intercambiar información (los automóviles constituyen el mejor ejemplo). En resumen, hemos adoptado una conducta de vida íntimamente ligada a los sistemas de comunicaciones, y como no podía ser de otra forma, el desarrollo e investigación militar e industrial han sido el motor de este nuevo concepto.

De todos los tipos de sistemas de comunicación, los sistemas de bus presentan una serie de ventajas que les han hecho ser aceptados en sectores relacionados con la automatización industrial y la ingeniería de automoción. El coste de las instalaciones y el cableado es reducido, permiten rápidos diagnósticos y resoluciones de problemas de los equipos conectados, nuevos nodos pueden ser integrados con mayor facilidad pudiéndose calibrar y configurar vía red. La estructura modular de los buses de campo permite construir sistemas estandarizados y facilita la labor de reemplazar equipos, mantenimiento y configuración. El problema que aparece aquí es la existencia de distintos protocolos de comunicación por bus, desarrollados por distintas empresas fuertes en el sector, que generan problemas de compatibilidad.

Destacamos por ser los más establecidos en el mercado, el sistema *Profibus* [1], resultado de la unión de proyectos relacionados con el tema de buses de campo de varias compañías e instituciones de investigación de Alemania, *Bitbus* [2], de Intel, *Interbus-S*, estandarizado por Phoenix Contact [3], y *CAN* bus, desarrollado por Bosch [4] (ver tabla 1.1). Este último parece ser por el que están apostando finalmente distintos sectores, influenciados por la gran y competitiva industria de la automoción [5, 6], ante el alto rendimiento prestado. Entre otras, las principales razones para ello son sus eficientes mecanismos de detección y tratamiento de errores, su gran simplicidad, adecuado para sistemas que requieran respuestas rápidas, y su fiabilidad y robustez máximas que le otorgan su alta inmunidad a interferencias electromagnéticas, permitiendo la comunicación en ambientes muy ruidosos.

Característica	**CAN bus**	**Profibus**	**Bitbus**	**Interbus-S**
Tasa de transmisión	Hasta 1Mbit/s	Hasta 1,5 Mbit/s	Hasta 1,5 Mbit/s	500 Kbit/s
Acceso al medio	CSMA/CD	Paso de testigo	Ninguno	Ninguno
Número máximo de nodos	Limitado por el *transceiver*	127 ó 256 por red	249 por red	256 por red
Técnica de comunicación	Orientado a mensajes	Maestro-Esclavo	Maestro-Esclavo	Maestro-Esclavo
Estándar soportado	ISO 11898	DIN19245	IEEE 1118	DIN E 19258

Tabla 1.1: Comparativa entre los estándares de buses de campo más establecidos.

CAN es un sistema de bus serie multimaestro, orientado a mensajes, esto es, la información no es direccionada, si no que los mensajes enviados contienen un identificador que marca su prioridad en la red. Todos los mensajes son leídos por todos y cada uno de los nodos del sistema *CAN*, que son los responsables de aceptar o descartar cada mensaje. Esto permite una gran flexibilidad para incorporar y retirar nodos de la red, así como implementar fácilmente la difusión *broadcast* y *multicast* [7, 8]. Como ya hemos comentado se aplica principalmente en la industria del automóvil, como elemento principal en la comunicación entre los distintos nodos que conforman la red electrónica de control [9, 10, 11]. No obstante, otros campos como la automatización de procesos industriales, equipamiento médico, control de equipos o domótica, entre otros, están incorporando este estándar como sistema de comunicación [12, 13, 14].

El objetivo principal de este proyecto ha consistido en el estudio y comprensión del sistema bus *CAN*, y el planteamiento, desarrollo y diseño final de un analizador de este protocolo de comunicaciones, con la finalidad de constituir una herramienta para el diagnóstico de posibles errores en el diseño de una red de nodos *CAN*. Este analizador tiene la capacidad de detectar todos los tipos de tramas en distintas configuraciones posibles del sistema *CAN*. Asimismo puede ser programado para que funcione correctamente con distintos valores de la tasa de transmisión.

La información del tráfico analizado es almacenada en una tarjeta de memoria MMC o SD (*Multimedia Card* o *Secure Digital*) [15, 16], con la finalidad de hacer más cómoda la utilización del analizador al evitar estar conectados a un PC u ordenador portátil durante la captura. Los datos acerca de la configuración del analizador que precisa el módulo *CAN*, tales como las máscaras y filtros, parámetros del *bit-timing* o tipos de tramas que se van a aceptar, son introducidos por el programador en la tarjeta para luego ser leídos directamente por el microcontrolador en las primeras fases del programa. La tarjeta contiene un sistema de archivos FAT16 [17]. Los datos deben ser guardados respetando la estructura de este sistema. De esta forma, cuando introduzcamos la tarjeta de memoria en el lector de tarjetas conectado en el PC, donde vamos a hacer el tratamiento de la información, será interpretada como un disco extraíble, evitando la necesidad de programar un *driver* con toda la complejidad que ello supone.

El controlador utilizado para gestionar el análisis del tráfico *CAN* así como todos los periféricos acoplados y el programa interno, es el PIC18F258, de la familia PIC18FXX8 de Microchip [18, 19]. El módulo *CAN* incorporado y las altas prestaciones ofrecidas han sido elementos claves en la elección de este microcontrolador. Presenta una memoria de programa *flash* de 16 Kbytes, una memoria de datos de 1.536 bytes y es capaz de operar con un reloj de hasta 40 MHz. Destacamos el amplio abanico de periféricos disponibles (temporizadores/contadores, comparadores, convertidores A/D, módulos de comunicación, etc.), muchos de ellos no incluidos en microcontroladores de familias inferiores, posibilitando al usuario desarrollar muchas aplicaciones. Además tiene un sistema de interrupciones basado en prioridades, con una gran variedad de fuentes, que permiten tener bajo control los distintos módulos.

La última parte del diseño final del analizador se refiere al tratamiento de los datos almacenados en la tarjeta de memoria. En el PC se mostrarán en una ventana, dispuestos en columnas, los distintos campos de interés de cada trama *CAN* así como el instante de tiempo de su captura. A través de este mismo programa se grabará en la tarjeta el fichero con los parámetros de configuración fijados para una captura en concreto.

Finalmente, hemos desarrollado como aplicación práctica, para la comprobación del correcto funcionamiento del analizador, un módulo *CAN* universal. La idea de la que partimos es la de disponer de un módulo estándar en el que podamos conectar distintos periféricos, tanto digitales, de entrada y de salida, como analógicos, sólo de entrada, aprovechando las prestaciones que ofrece el microcontrolador. Tanto la activación o desactivación de estos periféricos, como el envío o petición de datos, será realizada por un nodo controlador, desde un PC u otro microcontrolador, mediante mensajes a través del bus *CAN*. De esta forma, los nodos *CAN* universales no deberán ser programados cada vez que queramos incorporar un sensor o actuador. Además, aprovecharemos la ventaja de *CAN* bus para la difusión *multicast* para la petición o envío de información a un conjunto de nodos con características idénticas o similares.

1.2 Fases de desarrollo del proyecto

Para la realización del proyecto se han planteado una serie de fases divididas en seis bloques:

1. Documentación: recopilación bibliográfica sobre el protocolo *CAN* bus y trabajos relacionados con él. Documentación sobre la familia de microcontroladores PIC18FXX8 y manuales y *datasheets* relacionados con el compilador C18 de Microchip. Para la consulta de todo el material bibliográfico se ha contado tanto con los fondos propios de la Universidad de Valladolid como los de la Biblioteca Pública, así como con distintas fuentes accedidas a través de la red. Todas ellas son citadas de forma pormenorizada en el capítulo de Bibliografía (página 403).

2. Diseño hardware: estudio, evaluación y especificación de los componentes necesarios. División del diseño en cuatro bloques principales: etapa de alimentación, LCD y botonera, bloque del microcontrolador y *transceiver CAN* y bloque de la tarjeta de memoria.

3. Diseño software: desarrollo del programa del microcontrolador que gestiona la captura del tráfico *CAN*, los distintos periféricos asociados y la lectura y escritura de datos en la tarjeta de memoria. También incluimos en este bloque la implementación de la interfaz gráfica para el usuario en el PC.

4. Depuración del analizador: establecimiento de un test de ensayo con una serie de pruebas a las que someter al analizador, con el fin de comprobar el correcto funcionamiento del programa y del hardware asociado ante una serie de situaciones planteadas y diferentes sistemas *CAN*.

5. Diseño del prototipo: elaboración de la placa PCB final con todos los componentes integrados.

6. Planteamiento, diseño y desarrollo de la aplicación práctica: un módulo universal en el que poder conectar distintos periféricos, controlado mediante *CAN* bus.

1.3 Medios empleados

- Las herramientas informáticas empleadas son:
  - Para la programación y depuración MPLAB (versión 6.4), de Microchip [19][1], Proteus (versión 6.3), de Labcenter Electronics [20], y el compilador MPLAB C18, de Microchip [21, 22].
  - Para el diseño de las placas finales, Proteus (el módulo ARES).
  - La interfaz gráfica con el PC ha sido programada en Visual Basic [23].

[1]En esta web es posible la descarga tanto del programa como de los manuales de usuario.

- Material y componentes electrónicos:
  - Los microcontroladores, *transceivers*, adaptadores de nivel y conversores de alimentación han sido obtenidos gratuitamente de las empresas de electrónica Microchip [19] y Maxim [24] [2].
  - La tarjeta de memoria, el LCD, conectores, cable UTP y otros elementos discretos (resistencias, condensadores, LEDs, relojes,...) han sido suministrados por el Departamento de Electrónica o bien adquiridos por los proyectantes.
- Espacio físico: para el trabajo de programación y montaje experimental hemos dispuesto del Laboratorio de Proyectos del Departamento de Electrónica de la Universidad de Valladolid, dotado de los ordenadores e instrumental electrónico necesarios, así como la maquina fresadora utilizada en la realización de las placas finales.

1.4 Estructura de la memoria

La memoria tiene una división lógica en dos bloques diferenciados. En el primero agrupamos todos los conceptos teóricos que consideramos necesarios para entender capítulos posteriores. De esta forma incluimos aquí la descripción del protocolo de comunicaciones *CAN* bus, capítulo 2 (página 31), el estudio de la familia de microcontroladores PIC18FXX8, capítulo 3 (página 45) y una exposición sobre tarjetas de memoria, tipos y características, y la estructura interna de ficheros FAT16 que utilizamos para almacenar los datos, capítulo 4 (página 61).

El segundo bloque comienza con el capítulo 5 (página 75), el más extenso e importante, en el que describimos los distintos aspectos del analizador desarrollado. Tras un primera introducción sobre las generalidades sobre el analizador, pasamos a los sub-apartados relacionados con el esquema hardware y todo el software programado. El siguiente capítulo, 6 (página 125), hace referencia a la aplicación práctica propuesta, de nuevo con sub-apartados relacionados con el hardware y software.

Finalmente tenemos los capítulos relacionados con los costes y presupuestos, capítulo 7 (página 147), y las conclusiones 8 (página 153). A continuación aparecen los distintos apéndices relacionados con nuestro proyecto.

[2] En el apartado de *Samples.*

Capítulo 2

CAN bus

2.1 Introducción

El continuo aumento de las prestaciones y confort en los automóviles ha traído consigo un incremento de los dispositivos electrónicos, así como la necesidad de interconectar dichos dispositivos de una forma sencilla y eficaz. Para dar solución a dichos problemas los ingenieros de Robert Bosch desarrollaron a mediados de los años ochenta el bus *CAN*(*Controller Area Network*). En la actualidad, gracias a su sencillez y robustez, el bus *CAN* no sólo se aplica en la industria del automóvil, sino que se ha extendido a diversos campos. No es extraño encontrar aplicaciones de este bus en equipos médicos, aviones, ascensores, sistemas de control en plantas industriales o en electrónica de consumo.

El bus *CAN* es un sistema de bus serie multimaestro [4, 7, 8], es decir, cualquier nodo en un momento dado puede iniciar una comunicación. Es un protocolo orientado a mensajes, al contrario de otros protocolos de comunicaciones (como *Ethernet*) que están orientados a direcciones. Su principal característica consiste en que los nodos del bus no tienen una dirección específica. En lugar de direccionar la información, se etiqueta los mensajes transmitidos con un identificador, el cual determina la prioridad del mensaje e informa de su contenido. Todos los nodos recibirán el mensaje, y en función del identificador decidirán si ese mensaje tiene algún sentido para ellos. De esta forma es sencillo hacer llegar un mensaje a varios nodos.

El número de nodos no está limitado por el protocolo, por lo que se pueden cambiar dinámicamente sin interferir en la comunicación. Esto permite conectar y desconectar nodos del bus para la incorporación de nuevas funciones. Además gracias al sistema de identificación de mensajes es muy fácil implementar comunicaciones *multicast* (difusión a un conjunto de nodos) y *broadcast* (difusión a todos los nodos).

La especificación del bus *CAN* sigue el modelo de referencia OSI [25], y en particular las norma ISO 11898 para comunicaciones de alta velocidad (entre 125 kbit/s hasta 1Mbit/s) y la norma ISO11519 para comunicaciones de baja velocidad. En la figura 2.1

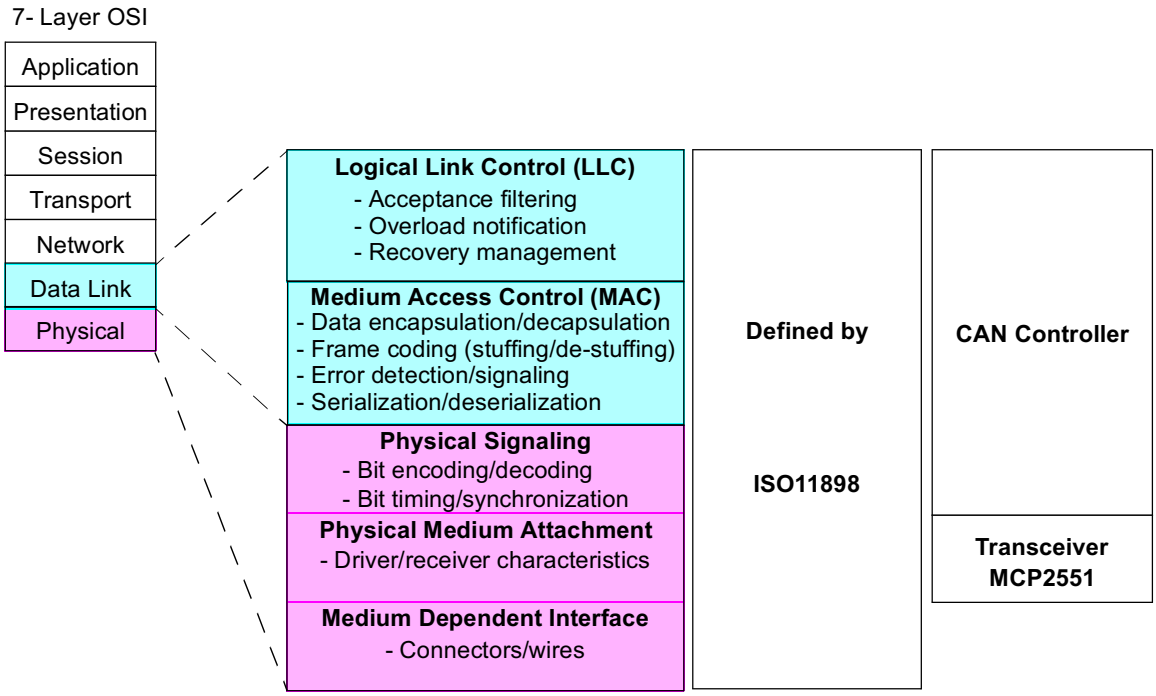


Figura 2.1: Comparación del modelo de referencia OSI con la especificación *CAN*.

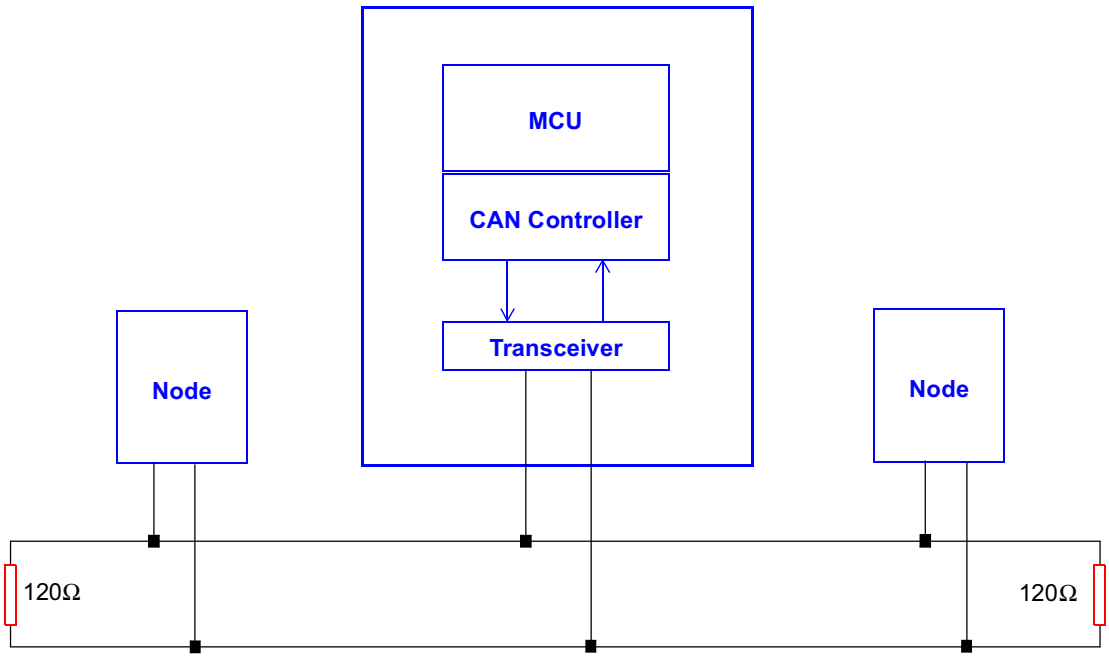


Figura 2.2: Esquema de tres nodos conectados por bus *CAN*.

se puede apreciar las capas del modelo OSI con la correspondiente equivalencia *CAN*. La especificación de Bosch sólo implementa la capa de enlace y parte de la capa física, dejando las capas superiores sin especificar. El resto de las capas superiores han sido implementadas por diferentes consorcios o agrupaciones de empresas para resolver problemas concretos.

La especificación *CAN* define completamente la capa de enlace [4], mientras que para la capa física sólo define la parte destinada a la señalización dejando las subcapas inferiores sin especificar. La norma ISO-11898 [26] incorporó la especificación *CAN* y añadió a la capa física la subcapa *Physical Medium Attachment*, para asegurar una completa compatibilidad entre *transceivers* de diferentes compañías. La parte acerca de tipos de conectores y cables ha quedado sin definir, aunque deben cumplir las especificaciones eléctricas.

Normalmente la capa de aplicación es desarrollada a través de un microcontrolador o DSP. La capa de enlace está implementada por un controlador *CAN*, que puede estar incluido o no en el microcontrolador. Actualmente, la mayoría de los fabricantes de microcontroladores tienen modelos que incorporan dichos controladores. La capa física está controlada por un *transceiver*, cuya misión es la de garantizar que se cumplan todas las especificaciones eléctricas del protocolo. En particular Microchip tiene desarrollado el *transceiver* MCP2551[1].

2.2 Capa física del bus CAN

El protocolo *CAN* usa la codificación de bit no retorno a cero (NRZ), realizando la inserción de bits adicionales (*bit stuffing*) para evitar la perdida de sincronismo en cadenas largas de bits iguales [4, 25]. En lugar de voltajes TTL, en el bus *CAN* están presentes la señales diferenciales CAN_L, CAN_H (ver figura 2.3).

El *transceiver* es el encargado de convertir los niveles TTL del controlador *CAN* en señales diferenciales. Hay dos estados lógicos para el bus: 'fuerte' (*dominant*), y 'débil' (*recessive*). En un bit débil, las dos líneas del bus están a un nivel de 2,5V, y proporcionan una tensión diferencial cercana a 0V. En un bit fuerte la línea CAN_H está a 3,5V y CAN_L a 1,5V, lo que origina una tensión diferencial entorno a 2V.

Generalmente el estado fuerte se corresponde con un 'cero lógico' y el débil con el 'uno lógico'. Los nodos se conectan al bus con una estructura AND cableada, de modo que el bit fuerte de un nodo siempre prevalecerá sobre el bit débil. Sólo si todos los nodos transmiten bits débiles, el estado del bus será débil.

El acceso al bus se basa en el protocolo CSMA/CD (*Carrier Sense Multiple Access / Colision Detect*, acceso múltiple por detección de portadora con detección de colisión).

[1] Consultar el apéndice H (página 277).

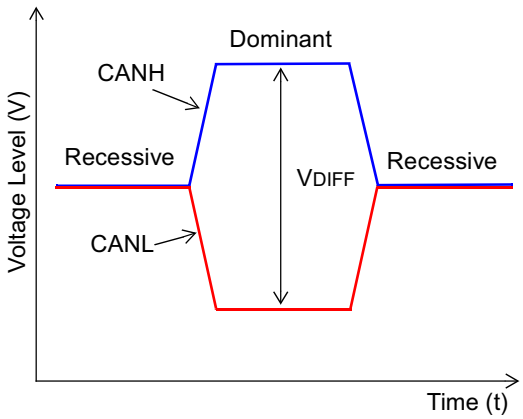


Figura 2.3: Niveles para los bits fuertes y débiles dentro del bus.

El arbitraje es no destructivo, lo que significa que la detección de una colisión no destruye todos los mensajes sino que prevalece uno de ellos (*bit wise arbitration*), lo que evita las pérdidas de tiempo en la comunicación.

Los conflictos en el bus se resuelven comparando bit a bit los identificadores de los mensajes. Un nodo pierde el arbitraje cuando el bit del identificador del mensaje que está trasmitiendo está formado por un bit débil (*recessive*) y al menos uno de sus otros contrincantes presenten un bit fuerte (*dominant*). Cuando un nodo pierde el arbitraje deja de transmitir. Por lo tanto, con los chequeos bit a bit del identificador, aquel identificador con menor número binario es el que prevalecerá en el bus y por lo tanto es el de mayor prioridad. Este último punto es muy importante a la hora de diseñar una red *CAN*. Aquellos mensajes que lleven información urgente e importante deberán tener identificadores bajos, para que la información llegue lo antes posible sin importar los estados de los demás nodos que compongan la red.

El medio físico para este protocolo podría ser cualquier medio capaz de transmitir dos posibles estados (fuerte y débil). El medio más barato es el par de cable trenzado, pero sería posible un medio óptico donde el estado fuerte (*dominant*) sea la presencia de luz y el débil (*recessive*) la ausencia de ella.

Según [26], el bus está formado por un par de cables trenzados que deben ser terminados con una impedancia de 120Ω. Existen varias formas de terminar el bus que las presentamos de forma resumida en la figura 2.4. Cada cable conduce una señal diferencial (CAN_H, CAN_L). La naturaleza diferencial de la trasmisión *CAN*, la hace insensible a algunas interferencias electromagnéticas. Blindando el par trenzado con una malla podemos reducir las propias emisiones electromagnéticas del bus, especialmente a altas velocidades. Por lo general se suele emplear par trenzado de tipo UTP o FTP.

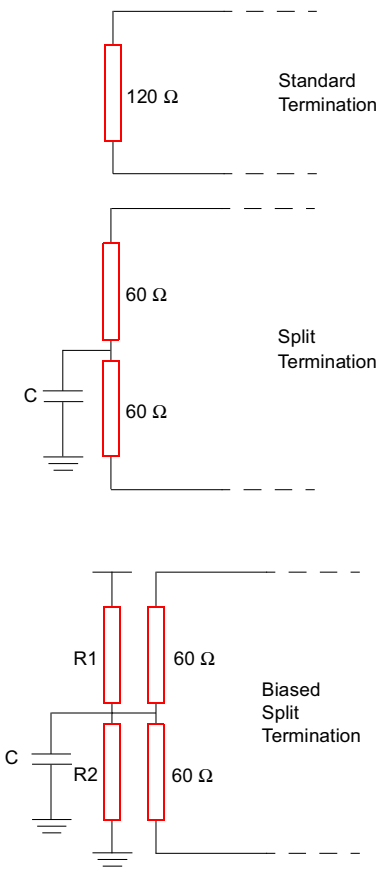


Figura 2.4: Múltiples formas de terminar el bus *CAN*.

Pin	Signal	Description
1	CAN_H	CAN_H bus line (dominant high)
2	CAN_L	CAN_L bus line (dominant low)
3	CAN_GND	Ground / 0 V / V-
4	-	Reserved
5	-	Reserved
6	(CAN_SHLD)	Optional CAN Shield
7	CAN_GND	Ground / 0 V / V-
8	(CAN_V+)	Optional CAN external positive supply (dedicated for supply of transceiver and opto-couplers, if galvanic isolation of the bus node applies)

Figura 2.5: Recomendación para los conectores RJ45 para el protocolo CANOpen.

No existe una estandarización acerca del tipo de conectores a usar, pero si existen recomendaciones [27] acerca de multitud de conectores a emplear y la asignación de pines para estos. En la figura 2.5 se puede observar la recomendación en el caso de emplear conectores RJ45 dentro del estándar de CANOpen.

2.2.1 Tiempo de bit y propagación de la señal

Cada tiempo de bit en el bus *CAN* está formado por 4 segmentos de tiempo no solapables. Cada segmento es múltiplo entero de un cuanto de tiempo denominado T_q. Este cuanto de tiempo es la menor resolución temporal usada en el nodo *CAN*, en otras palabras, es la unidad de tiempo mínima. Un bit puede estar comprendido entre 8 y 25 T_q. El tiempo de bit (y por lo tanto la tasa de transmisión) se selecciona programando el ancho de T_q y el número de T_q contenidos en los diferentes segmentos que forman el bit.

Los segmentos de que consta un bit son (ver figura 2.6):

- **Segmento de sincronización**: es el primer segmento. Se utiliza para sincronizar todos los nodos. La duración de este segmento es de 1 T_q.
- **Segmento de tiempo de propagación**: se usa para compensar los retardos de propagación de la señal a través de la red. Este segmento puede puede tener una longitud entre 1 y 8 T_q.
- **Segmento de fase 1**: este segmento junto al siguiente se utilizan para compensar los errores de fase debido a las tolerancias de los osciladores de los nodos. El punto de muestreo del bit se sitúa al final de este segmento. Este segmento puede variar entre 1 y 8 T_q. En el caso de una pérdida de sincronismo el segmento de fase 1 puede ser alargado o acortado cuando haya una pérdida de sincronismo.
- **Segmento de fase 2**: es similar al anterior y su tamaño varia entre 1 y 8 T_q.

El segmento de fase 1 se puede alargar o acortar el segmento de fase 2 cuando es necesaria una resincronización. Para ello existe un parámetro denominado salto de sincronización (SJW). Su valor debe de estar entre 1 y 4 T_q y permite alargar o acortar uno de los segmentos. No puede ser nunca mayor que el segmento de fase 2. Esta capacidad de alargar o acortar el tiempo de bit reduce la sensibilidad a las tolerancias de los osciladores, pudiendo reducir el coste de los mismos.

Respecto las velocidades de transmisión y la longitud del bus, la especificación ISO11898 [26] fija la tasa máxima de transmisión a 1Mbit/s para una longitud de 40m. A medida que la longitud del bus aumente, es necesario reducir la tasa de transmisión debido al tiempo que tarda en propagarse la señal por el bus (ver tabla 2.1).

El tiempo de propagación es muy importante debido a la forma que se ha elegido en la especificación para el arbitraje del bus [28, 29]. Cada nodo involucrado en un arbitraje

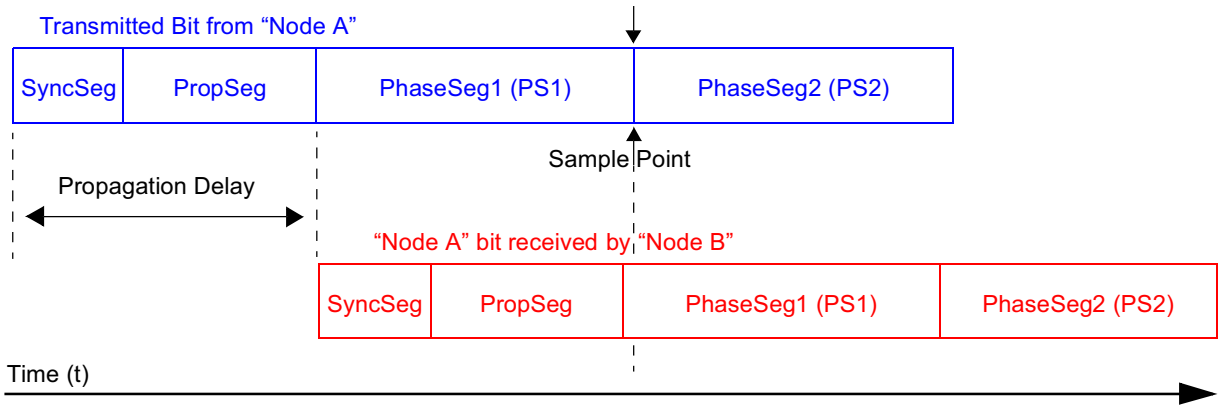


Figura 2.6: Efecto del tiempo de propagación en el proceso de muestreo de bit.

debe de ser capaz de muestrear cada bit dentro del mismo tiempo de bit. Supongamos el caso extremo de dos nodos, nodo 'A' y nodo 'B', situados cada uno al final de cada lado del bus. Si ambos nodos empiezan a transmitir al mismo tiempo y el tiempo de propagación de la señal es muy alto (ver figura 2.6), entonces ambos nodos no podrán muestrear un bit determinado dentro de la zona de muestreo, lo que provocará que falle el mecanismo de arbitraje.

El tiempo de propagación máximo de la señal en un sistema *CAN* vendrá determinado por la ecuación 2.1:

$$t_{prop} = 2(t_{bus} + t_{drv} + t_{cmp}) \tag{2.1}$$

Donde:

- t_{bus} es el tiempo que tarda en propagarse la señal de un extremo a otro del bus.
- t_{drv} es el retardo introducido por el driver de salida del *transceiver*.
- t_{cmp} es el retado introducido por el comparador de entrada del *transceiver*.

Velocidad de transmisión (kbits/s)	**Longitud del bus (m)**
1000	40
800	50
500	100
250	250
125	500
62,5	1000
20	2500
10	5000

Tabla 2.1: Relación de la longitud máxima del bus con la velocidad de transmisión.

2.3 Capa de enlace del bus CAN

La versión más reciente de *CAN* es la 2.0, donde se definen dos formatos de tramas distintas (tramas estándar y tramas extendidas). La diferencia entre unas y otras está en la longitud del identificador. En las tramas de tipo estándar el identificador tiene una longitud de 11 bits, lo que permite tener hasta 2048 mensajes distintos en una misma red. Este tipo de trama es la que se definió en el estándar anterior (versión 1.2). Para evitar que se agotasen los identificadores se decidió ampliar el tamaño del identificador a 29 bits (tramas extendidas), dando lugar a la versión 2.0 de este protocolo. Con un identificador de 29 bits podemos asignar hasta 2^{29} mensajes distintos. Cualquier controlador *CAN* que cumpla la especificación 2.0 debe de ser capaz de aceptar tanto tramas estándar como extendidas.

2.3.1 Tipos de tramas

En el protocolo *CAN* existen cuatro tipos de tramas distintas:

- **Tramas de datos** (*Data Frame*).
- **Tramas remotas** (*Remote Frame*).
- **Tramas de error** (*Error Frame*).
- **Tramas de sobrecarga** (*Overload Frame*).

Tanto las tramas de datos como las remotas pueden ser tanto estándares como extendidas. Por no aburrir al lector, no vamos a entrar en mucho detalle en cada uno de los campos de las tramas. Para una lectura más profunda del tema se puede consultar la especificación de Bosch [4].

Trama de datos, *Data frame*

La misión de estas tramas es la transmitir datos a los posibles receptores interesados. La trama de datos se compone de los siguientes campos (ver figura 2.7):

- Comienzo de trama (*Start of frame*): consiste en un bit fuerte (*dominant*). Un nodo sólo puede comenzar una transmisión cuando el bus está libre y dicho bit fuerza a todos los nodos a sincronizarse.
- Campo de arbitraje (*Arbitration field*): es diferente para el formato estándar y para el formato extendido. Dentro de este campo se encuentra el identificador de la trama (11 bits para el estándar, 29 para el extendido), y otra serie de bits que permiten diferenciar entre una trama remota y una trama de datos.
- Campo de control (*Control field*): en esta parte de la trama se indica el número de bytes de datos que se han mandado y que puede variar entre 0 y 8.

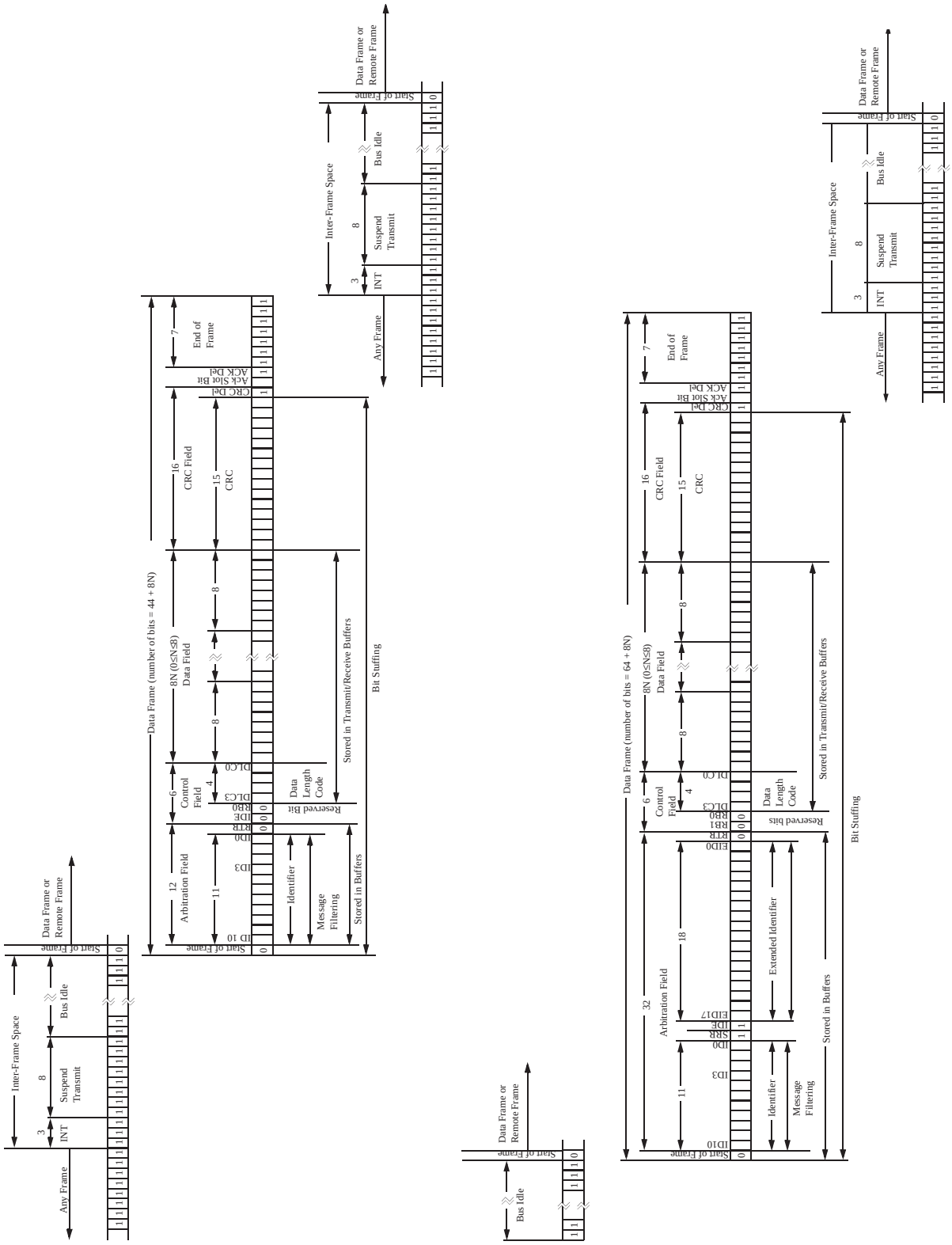


Figura 2.7: Formato de las tramas de datos con identificador estándar y extendido.

- Campo de datos (*Data field*): esta zona esta destinada a los datos a enviar, y puede variar entre 0 y 8 bytes.

- Campo de control de errores (*CRC field*): contiene la secuencia de control de errores. El polinomio generador que se emplea es:

$$X^{15} + X^{14} + X^{10} + X^{8} + X^{7} + X^{4} + X^{3} + 1 \tag{2.2}$$

- Campo de confirmación (*ACK field*): el campo ACK consiste en dos bits. El transmisor cuando envía la trama manda estos dos bits como débiles. El receptor si recibe el mensaje correctamente manda un bit fuerte durante el primer bit. Este proceso lo realizan todos los nodos que reciben el mensaje y cuya secuencia CRC concuerda.

- Fin de trama (*End of frame*). Consiste en una secuencia de 7 bits débiles (*recessive*).

Trama remota, *Remote frame*

Las tramas remotas son enviadas por los nodos para solicitar la transmisión de una trama de datos con el mismo identificador que posee la trama remota. Las tramas remotas son muy parecidas a las tramas de datos (ver figura 2.7). La principal diferencia es que no tienen campo de datos. Para distinguir entre una trama de datos o una trama remota existe el bit RTR (*Remote Transmission Request*) en el campo de arbitraje. En el caso de que la trama sea de datos, este bit es fuerte; mientras que en el caso de una trama remota, este bit es débil. Por lo tanto si dos nodos transmiten al mismo tiempo una trama de datos y una trama remota con el mismo identificador, entonces, la trama de datos ganará el arbitraje.

Trama de sobrecarga, *Overload frame*

Este tipo de trama se usa para obtener retardos extra entre tramas de datos y tramas remotas. Son generadas por el controlador *CAN* cuando necesita más tiempo para procesar los mensajes recibidos (2.8).

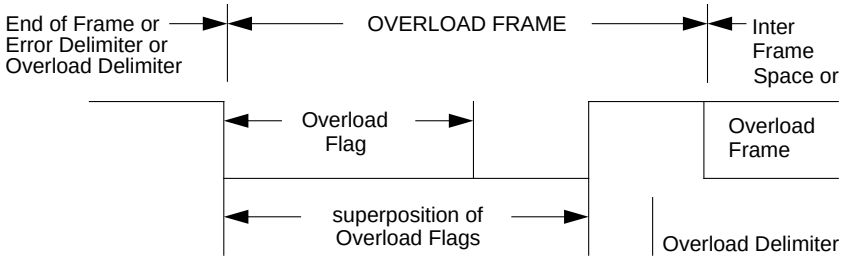


Figura 2.8: Trama de sobrecarga.

Trama de error, *Error frame*

Como su propio nombre indica, esta trama es enviada por cualquiera de los nodos cuando se detecta un error en el bus. En el protocolo *CAN* existen cinco fuentes de error:

- **Error CRC**: se produce si un nodo detecta que el código de redundancia no concuerda con el mensaje. En ese caso se enviará una trama de error.
- **Error de asentimiento**: el nodo que transmite la trama comprueba si el bit ACK*slot bit* ha pasado a estado fuerte (este bit es enviado por el transmisor como débil). Este bit fuerte es colocado por al menos un nodo que haya recibido correctamente el mensaje. Si no se ha producido un asentimiento el transmisor genera una trama de error y a continuación repite la transmisión del mensaje original.
- **Error de formato**: si un nodo detecta un bit fuerte al final de una trama, en el espacio de tramas, en el delimitador ACK, o en el delimitador CRC (esto es una violación del protocolo, ver figura 2.7), entonces se genera un trama de error.
- **Error de bit**: ocurre cuando el nodo transmisor envía un bit fuerte y detecta ese bit como débil o al contrario. En ese caso se envía una trama de error y se vuelve a trasmitir la trama.
- **Error de *stuff***: si un nodo detecta seis bits consecutivos con la misma polaridad, entonces se ha violado el protocolo. En ese caso se envía una trama de error y se repite el mensaje.

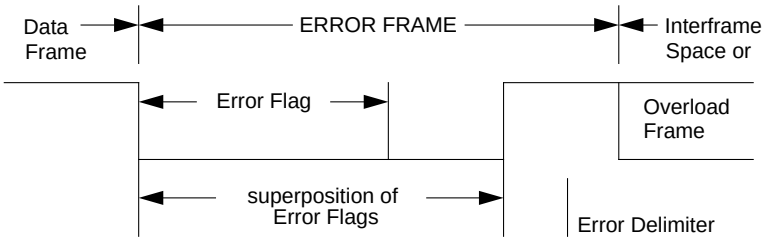


Figura 2.9: Trama de error.

Un nodo puede estar en tres modos o estados de error. Todo dependerá del valor de los contadores de error de recepción y de transmisión. Estos contadores se incrementan cuando se produce un error en la recepción o en la transmisión y se decrementan cuando la comunicación es exitosa. Los tres modos de error son:

- **Error activo**: si un nodo detecta un error incrementará el contador de error de transmisión o de recepción y procederá a mandar una señalización de error activo. La señalización (*error flag*) de error activo consiste en seis bits fuertes consecutivos. Esto provoca una violación del protocolo, por lo que el resto de los nodos responderán con su propia señalización de error (ver figura 2.9).

- **Error pasivo**: un nodo alcanza el estado de error pasivo cuando los contadores de error de transmisión o recepción son mayores de 127. En ese caso el nodo envía una señalización de seis bits débiles. Esto viola el protocolo y los demás nodos responderán con su propia señalización de error dependiendo si están en el modo de error activo o pasivo.

- **Bus *off***: un nodo entra en este estado cuando el contador de error de transmisión es mayor de 255. En ese instante el nodo se desconecta del la red y no puede enviar ningún tipo de trama o asentimiento.

2.3.2 Máscaras y filtros

En el protocolo *CAN* no existen direcciones. Los mensajes son etiquetados con un identificador y son los nodos *CAN* los que deciden, analizando dicho identificador, si el mensaje lo aceptan o no. Para realizar ese proceso, los controladores *CAN* poseen una serie de máscaras y filtros asociados a esas máscaras. La tabla 2.2 muestra la forma de comparar los bits de las máscaras, filtros e identificadores. Las máscaras se emplean para determinar que bits del identificador serán comparados con los filtros. Si la máscara se fija a cero se aceptarán todos los mensajes ignorando cualquier filtro. Si comparando bit a bit, el bit del filtro coincide con el bit del identificador, entonces el mensaje será aceptado por el nodo.

Bit máscara	Bit filtro	Bit identificador	Acción
0	x	x	Aceptado
1	0	0	Aceptado
1	0	1	Rechazado
1	1	0	Rechazado
1	1	1	Aceptado

Tabla 2.2: Tabla de verdad para el análisis del n-ésimo bit de un identificador.

2.4 Protocolos de capas superiores

Como ya se ha comentado, la especificación de Bosch [4] sólo estandariza las dos primeras capas del modelo OSI. Para las restantes capas han surgido una serie de protocolos. Algunos de ellos son para aplicaciones específicas mientras que otros son de uso más general. Los ejemplos más importantes son:

- **CANOpen** fue pensado como un protocolo que permitía una alta flexibilidad a la hora de configurar la red. Los campos de actuación de este protocolo son: electrónica para automóviles, electrónica para barcos, ascensores, etc. La organización encargada de mantener la especificación de CANOpen es CiA (*CAN in Automation*) [5].

- **CANKingdom** es un protocolo que da a los diseñadores la máxima libertad a la hora de diseñar un sistema. CANKingdom ofrece la posibilidad de realizar diseños genéricos y modulares que pueden ser luego integrados en cualquier sistema. Este protocolo ha sido desarrollado por CKI (*CANKingdom International*) [30].

- **DeviceNet** se emplea principalmente en aplicaciones industriales, y más concretamente en tareas de automatización de plantas. Es un protocolo de bajo coste, que comunica dispositivos industriales como sensores, *starters* para motores, electroválvulas, variadores de frecuencia, etc. Es un protocolo pensado para minimizar el cableado en la planta y facilitar el mantenimiento de la red. DeviceNet es una solución rápida desarrollada por ODVA (*Open DeviceNet Vendor Association*) [9].

- **J1939** es un protocolo muy especifico. El ámbito de aplicación de este protocolo son los vehículos pesados como *trailers*, y maquinaria agrícola. Su misión es la de comunicar los dispositivos electrónicos de la cabeza tractora y el remolque. Este estándar está desarrollado por SAE (*Society of Automotive Engineers*) [6].

- **TTCAN** (*Time-Triggered Communication on CAN*). Todos los eventos que se producen dentro de la red sólo pueden enviarse en segmentos de tiempo bien definidos. Este tipo de sistema es ideal cuando el tráfico es de naturaleza periódica. Este protocolo facilita la comunicación en sistemas de control de bucle cerrado y sistemas que trabajan en tiempo real. Existe un mensaje de referencia a través del cual todos los nodos se sincronizan. Todos los nodos de la red tienen asignado una ventana de tiempo. En esa ventana sólo el nodo propietario de esa ventana de tiempo puede transmitir datos.

Capítulo 3

PIC18FXX8

3.1 Características generales

Los controladores utilizados en el analizador y en los distintos ejemplos prácticos pertenecen a la familia PIC18FXX8 de Microchip, formada por cuatro microcontroladores: PIC18F248, PIC18F258, PIC18F448 y PIC18F458. Por una parte hemos elegido los controladores de la compañía Microchip por el envío de muestras gratuitas, que nos ha permitido disponer del número suficiente de ellas para realizar las distintas pruebas planteadas. Por otro lado disponen de una extensa y detallada documentación sobre sus productos y además ponen a disposición del usuario de todas las herramientas de desarrollo necesarias [19]. La decisión en concreto por esta familia se debe a que tiene integrado el módulo de comunicación *CAN*, con lo que evitamos tener que utilizar un controlador *CAN* externo, que dificultaría el diseño y la programación, aparte que la placa de montaje final tendría un tamaño mayor. Además, los cuatro modelos tienen versiones con encapsulado PDIP, para montar en placas de prototipos.

En la tabla 3.1 mostramos las especificaciones y prestaciones básicas de esta familia de microcontroladores, que serán repasadas en las siguientes secciones, con especial atención a los apartados de interrupciones y módulo *CAN*, por su mayor implicación en el presente proyecto. Las figuras 3.1 y 3.2 nos presentan el esquema de pines de estos microcontroladores.

Característica	PIC18F248	PIC18F258	PIC18F448	PIC18F458
Frecuencia máxima de operación	DC - 40 Mhz	DC - 40 Mhz	DC - 40 Mhz	DC - 40 Mhz
Memoria FLASH interna de programa (bytes)	16K	32K	16K	32K
Memoria de datos SRAM (bytes)	768	1536	768	1536
Memoria de datos EEPROM (bytes)	256	256	256	256
Fuentes de interrupción	17	17	21	21
Puertos I/O	Puertos A, B y C	Puertos A, B y C	Puertos A, B, C, D y E	Puertos A, B, C, D y E
Temporizadores/Contadores	4	4	4	4
Módulo de captura/comparadores/PWM	1	1	1	1
Módulo avanzado de captura/comparadores/PWM	-	-	1	1
Comunicaciones serie	MSSP, USART	MSSP, USART	MSSP, USART	MSSP, USART
Comunicaciones paralelo (PSP)	No	No	Sí	Sí
Módulo *CAN*	Sí	Sí	Sí	Sí
Convertidor A/D de 10 bits	5 canales de entrada	5 canales de entrada	8 canales de entrada	8 canales de entrada
Comparadores analógicos	No	No	Sí	Sí
Programación serie *In-Circuit*	Sí	Sí	Sí	Sí
Conjunto de instrucciones	75	75	75	75
Encapsulados	SPDIP y SOIC, de 28 pines	SPDIP y SOIC, de 28 pines	PDIP, de 40 pines PLCC y TQFP de 44 pines	PDIP, de 40 pines PLCC y TQFP de 44 pines

Tabla 3.1: Características principales de la familia PIC18FXX8.

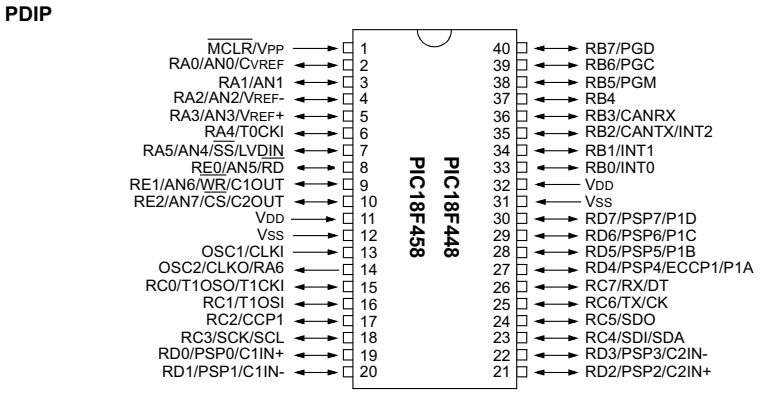


Figura 3.1: Diagrama de patillas del encapsulado PDIP de los PIC18F4X8.

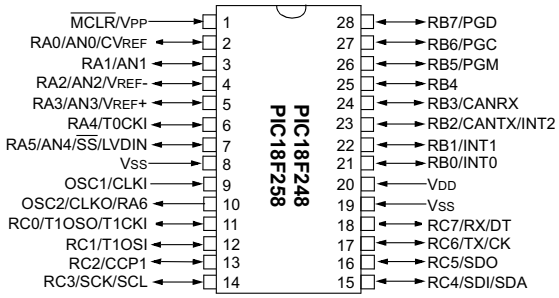


Figura 3.2: Diagrama de patillas del encapsulado PDIP de los PIC18F2X8.

3.2 Prestaciones principales

Los PIC18FXX8 incluyen las prestaciones habituales de muchos otros microcontroladores de Microchip, como por ejemplo el PIC16F84A y los PIC16F87XA:

- Interrupciones ocasionadas por distintos eventos (haremos un énfasis especial en ellas en el siguiente apartado).
- Memoria EEPROM de 256 bytes.
- Protección de código programable.
- *Resets* especiales: *Power-on reset*, POR, que se trata de un *reset* generado durante el funcionamiento del microcontrolador cuando la tensión de alimentación se sitúa entre 1,2 y 1,7V; *Power-up timer*, PWRT, que proporciona un *reset* de unos 72ns aproximadamente con el fin de que la tensión de alimentación llegue a un nivel aceptable; *Oscillator start-up timer*, OST, que activa un *reset* adicional de 1024 ciclos de reloj para asegurar que la oscilación del cristal o resonador se ha estabilizado, encontrándose en los niveles adecuados.
- Perro guardián (WDT, *Watchdog timer*): se trata de un mecanismo que, activado, evita que el microcontrolador entre en un estado indeterminado como consecuencia de un error de programación o un posible fallo de hardware.
- Modo de bajo consumo o SLEEP, que mantiene al microcontrolador con los elementos imprescindibles para su mantenimiento, a la espera de un evento que produzca la salida de este estado. Estos sucesos son un *reset* externo, el WDT que despierta o por alguna interrupción (externa, cambio de estado en algún pin del puerto B o algún módulo del controlador).
- Programación en serie *In-Circuit*, que permiten programar el microcontrolador sin necesidad de extraerlo del sistema final en el que se encuentre.
- Posibilidad de operación para distintos tipos de osciladores.

Otras utilidades son ya propias de esta familia. Destacamos el multiplicador hardware de 8 bits por 8 bits, que realiza la operación en un único ciclo de instrucción, y la posibilidad de multiplicar por 4 la frecuencia de oscilación del reloj de cuarzo, para relojes de hasta 10 MHz, por medio de un PLL integrado (Sistema de enganche de fase). Esto nos permite la posibilidad de operar a velocidades de hasta 40 MHz.

Los módulos integrados en el microcontrolador abarcan, entre otros, los campos de comunicaciones, temporizadores/contadores y diversas operaciones analógicas, proporcionando una gran versatilidad al programador. Existen diferencias entre los PIC18F2X8 y los PIC18F4X8, debido a que estos últimos disponen de un mayor número de pines para utilizar como entrada y/o salida. A continuación describimos someramente los módulos principales:

1. Módulos de comunicaciones integrados:

   - Puerto Maestro Serie Síncrono, MSSP, que puede operar en dos modos: Comunicación Serie SPI e I_2C, tanto en modo multimaestro como en modo esclavo. Este módulo sirve como interfaz serie entre el microcontrolador y distintos dispositivos externos que se puedan acoplar, como memorias serie EEPROMs, registros de desplazamiento, conversores A/D, *displays*, etc.
   - Receptor-Transmisor Universal Direccionable Síncrono-Asíncrono, USART. Puede ser configurado como un sistema asíncrono *full-duplex* para comunicarse por ejemplo con terminales CRT o PCs, o bien como un sistema síncrono *half-duplex* para comunicación con memorias serie EEPROMs, circuitos integrados de distintos tipos, como conversores A/D y D/A, etc.
   - Sistema de comunicación *CAN*, del que entraremos en más detalles en siguientes secciones.

2. Dispone de 4 módulos temporizadores, que también pueden ser configurados como contadores. Tres de ellos son de 16 bits y uno de 8 bits. Al 'temporizador 0' se le puede asignar una preescala programable de 8 bits, lo que significa que puede llegar a incrementarse en un factor de 256 la capacidad de dicho módulo. Cada uno de los temporizadores/contadores tiene asignada su correspondiente interrupción.

3. Módulos CCP, de captura, comparación y PWM:

   - Tanto los PIC18F2X8 como los PIC18F4X8 utilizan un registro de 16 bits para las tres tareas mencionadas. Ante un evento ocurrido en el pin *RC2/CCP1* (ver figuras 3.1 y 3.2) el 'temporizador 1' o el 'temporizador 3' es capturado en dicho registro o comparado con él, según esté configurado el módulo. La tercera opción es que a través del pin *RC2/CCP1* tengamos una salida Modulada por Anchura de Pulso (*Pulse Width Modulation*), valiéndose del 'temporizador 2'.
   - Los PIC18F4X8 disponen exclusivamente de un módulo CCP especial, que además de las características del CCP básico, ofrece una serie de prestaciones muy útiles para el control avanzado de motores como son el tener 1, 2 o 4 salidas PWM, la posibilidad de seleccionar la polaridad y la programación del *deadband delay* para mejorar la eficiencia y aumentar la seguridad para los posibles motores conectados.

4. Dispositivos analógicos avanzados:

   - Conversor Analógico-Digital de 10 bits, que nos da una resolución de $\frac{Vref^+ - Vref^-}{2^{10}}$, donde $Vref^+$ y $Vref^-$ corresponden a la tensión de alimentación del microcontrolador y a tierra respectivamente, a no ser que sea programado para que se tomen los valores que haya en los pines *AN3* y *AN2*. El conversor tiene 5 canales en los PIC18F2X8, por 8 de los PIC18F4X8.

- El módulo comparador analógico contiene 2 comparadores y sólo está disponible en los PIC18F4X8. Existen hasta 8 modos de operación para comparar distintas señales.
- Detección de bajo voltaje. Esta opción es muy útil para distintas aplicaciones que deben tener una tensión de alimentación bien definida. Cuando el módulo detecta que el voltaje de alimentación desciende de un valor determinado (programado por nosotros) genera una interrupción que nos avisa de tal suceso.
- Programable *Brown-out Reset* (BOR). El microcontrolador genera una señal de *reset* cuando la señal de alimentación desciende por debajo de un nivel programable.

Cualquier información sobre el funcionamiento, especificaciones y estructura de estos módulos puede consultarse en el *datasheet* de Microchip [18].

3.3 Interrupciones

3.3.1 Descripción general

La familia PIC18FXX8 presenta múltiples fuentes posibles de interrupción, como era de esperar en unos microcontroladores con una complejidad considerable y que contienen muchos módulos integrados. La principal característica que destacamos, respecto a versiones más antiguas o inferiores, es la posibilidad de priorizar las interrupciones en dos niveles, alto y bajo, de modo que cualquier fuente de interrupción asignada al nivel alto invalida cualquier interrupción con prioridad baja que se esté atendiendo. Esta habilidad es programable, y estando activada, deberemos asignar uno de los dos niveles a cada interrupción habilitada.

Existen 13 registros asociados al control de las interrupciones: *RCON*, *INTCON*, *INTCON2*, *INTCON3*, *PIR1*, *PIR2*, *PIR3*, *PIE1*, *PIE2*, *PIE3*, *IPR1*, *IPR2*, *IPR3*. La figura 3.3 muestra los más importantes en nuestro proyecto por su utilización.

Cada fuente de interrupción tiene tres bits de control asociados con tres funciones definidas:

- Bit de *flag* o de aviso, que se activa (se pone a '1') cuando ha ocurrido el suceso asociado a la interrupción[1].
- Bit de habilitación de la interrupción, que al activarse permite ejecutarse la rutina de atención a interrupciones cuando el bit de *flag* correspondiente se pone a '1', es decir, ha ocurrido el evento asociado.

[1]Es un bit pegajoso (*sticky bit*), es decir, es responsabilidad del programador ponerlo de nuevo a '0' para poder detectar un nuevo evento.

- Bit de prioridad, con el que clasificamos la interrupción en alta o baja prioridad.

RCON REGISTER

R/W-0	U-0	U-0	R/W-1	R-1	R-1	R/W-0	R/W-0
IPEN	—	—	$\overline{RI}$	$\overline{TO}$	$\overline{PD}$	$\overline{POR}$	$\overline{BOR}$
bit 7							bit 0

INTCON REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF
bit 7							bit 0

INTCON2 REGISTER

R/W-1	R/W-1	R/W-1	U-0	U-0	R/W-1	U-0	R/W-1
$\overline{RBPU}$	INTEDG0	INTEDG1	—	—	TMR0IP	—	RBIP
bit 7							bit 0

PIR3: PERIPHERAL INTERRUPT REQUEST (FLAG) REGISTER 3

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
IRXIF	WAKIF	ERRIF	TXB2IF	TXB1IF	TXB0IF	RXB1IF	RXB0IF
bit 7							bit 0

PIE3: PERIPHERAL INTERRUPT ENABLE REGISTER 3

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
IRXIE	WAKIE	ERRIE	TXB2IE	TXB1IE	TXB0IE	RXB1IE	RXB0IE
bit 7							bit 0

IPR3: PERIPHERAL INTERRUPT PRIORITY REGISTER 3

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
IRXIP	WAKIP	ERRIP	TXB2IP	TXB1IP	TXB0IP	RXB1IP	RXB0IP
bit 7							bit 0

Figura 3.3: Registros más significativos asociados a interrupciones.

El sistema de prioridades en las interrupciones es habilitado mediante la activación (poner a '1') del bit *IPEN*, en el registro *RCON* (*RCON.IPEN*). En este caso existen dos bits para la habilitación global de interrupciones:

- *INTCON.GIEH*, que activa todas las interrupciones.
- *INTCON.GIEL*, que activa todas las interrupciones de bajo nivel.

Si están activados el bit de habilitación de una fuente de interrupción en concreto, su correspondiente bit de *flag*, y el bit de activación global apropiado, inmediatamente la dirección de retorno es almacenada en la pila y el programa saltará a la dirección de memoria 000008h si la interrupción es de alta prioridad y a la 000018h si pertenece al grupo de baja prioridad[2]. A la vez el bit de habilitación global de interrupciones es puesto a 0: el *INTCON.GIE* si no tenemos sistema de prioridades o en el caso contrario, el *INTCON.GIEH* o el *INTCON.GIEL*, según corresponda. En la posición de memoria 000008h y/o 000018h deberemos tener la llamada a la función de atención de interrupciones, y será tarea del programador detectar la fuente que ha ocasionado la interrupción mediante el sondeo de los bits de *flags*. Luego se realizarán las instrucciones que se consideren oportunas. El retorno de la interrupción se realiza por medio de la instrucción

[2]Si el sistema de prioridades no está habilitado, el programa saltará a la posición de memoria 000008h, independientemente del origen de la interrupción.

RETFIE que sale de la rutina de atención a interrupciones. El programa recupera de la pila la posición de memoria en la que se encontraba cuando aconteció la interrupción y activa el bit *INTCON.GIE* (*INTCON.GIEH* o *INTCON.GIEH* si estamos trabajando con prioridades) de habilitación global.

3.3.2 Fuentes de interrupción

Las interrupciones asociadas al módulo *CAN* son controladas mediante 3 registros: el *PIR3*, que contiene los *flags* asociados a distintos eventos relacionados con este módulo, el *PIE3*, que permite la habilitación-deshabilitación de dichas interrupciones, y el *IPR3* que clasifica las mismas en alta o baja prioridad si procede. En el siguiente apartado analizaremos con más detalle este tipo de interrupciones.

El 'temporizador 0' es utilizado para tener en todo momento conocimiento del tiempo transcurrido desde el comienzo del análisis de tráfico *CAN*, para guardar el instante exacto de cada captura. El flag *INTCON.TMR0IF* se activa cuando el registro que va almacenando el tiempo se desborda, al ser utilizado en modo 16 bits, FFFFh→0000h. La interrupción es habilitada mediante el bit *INTCON.TMR0IE* y la prioridad se define en el bit *INTCON2.TMR0IP*.

Otra interrupción importante es la asociada externamente al pin *RB0/INT0*[3]. Al ser detectado un cambio de estado en ese pin, con un flanco de subida o de bajada según esté configurado en el registro *INTCON2*, el *flag INTCON.INT0IF* es puesto a '1'. Se puede desactivar esta prestación mediante el bit *INTCON.INT0IE*. Esta interrupción siempre está considerada como de alta prioridad[4].

Otras interrupciones están asociadas a distintos módulos del microcontrolador, como la que nos comunica que la conversión A/D en curso ha finalizado, las asociadas a los temporizadores '1 y 2', al módulo de captura, la relacionada con la memoria EEPROM o los distintos módulos de comunicación. Un cambio de estado en alguno de los pines de la parte alta del puerto *B* (*RB4-RB7*) también puede ser fuente de una interrupción. Para obtener información precisa de cómo funciona alguna de estas prestaciones se debe consultar el manual de Microchip [18].

[3]Existen análogas interrupciones asociadas a los pines *RB1/INT1* y *RB2/INT2*, con idéntico funcionamiento.

[4]En el caso de las *RB1* y *RB2* sí existen bits asociados para determinar su prioridad.

3.4 Módulo CAN-bus

3.4.1 Características principales

El módulo de comunicación *CAN* que soporta la familia PIC18FXX8 de Microchip, implementa las versiones del protocolo *CAN* 1.2, *CAN* 2.0A, *CAN* 2.0 Activo y *CAN* 2.0 Pasivo de la especificación definida por Bosch [4]. En dicha implementación se incluye la capa de enlace completa (ver sección 2.3 (página 38)). El módulo es capaz de generar todos los tipos de tramas posibles: tramas de datos con identificadores estándar y extendido, tramar remotas, mensajes de error, espacio entre tramas y la recepción de mensajes de aviso de sobrecarga.

El microcontrolador tiene un receptor de doble-*buffer*, es decir, dos *buffers* con distinta prioridad en los que se almacenan las tramas *CAN* que se van recibiendo. Para controlar los mensajes que queremos aceptar, el módulo dispone de 2 máscaras de aceptación completa (tanto identificadores estándar como extendidos), uno para cada *buffer* de recepción. El *buffer* de alta prioridad tiene asociado 2 filtros de aceptación completa, por 4 del *buffer* de baja prioridad. Cada mensaje detectado por el módulo es revisado para detectar algún posible error y a continuación se comprueba el identificador del mensaje con el objeto de decidir si el mensaje debe de ser procesado y almacenado en alguno de los dos registros de recepción. Para la transmisión de mensajes se vale de 3 *buffers* con prioridades programables y con capacidad de abortar mensajes pendientes.

También se pone a disposición del programador contadores de error, que permitirán realizar un adecuado control de flujo y de errores. Una serie de interrupciones programables de distintos eventos relacionados con el módulo *CAN* flexibilizan su uso. El microcontrolador puede operar en modo *SLEEP* de bajo consumo con el módulo *CAN* activo, a la espera de detectar actividad en el bus que "despertará" al microcontrolador. En los siguientes apartados analizaremos los aspectos más destacados del módulo *CAN*. A continuación mostramos los registros de control asociados:

CANCON: CAN CONTROL REGISTER

R/W-1	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0
REQOP2	REQOP1	REQOP0	ABAT	WIN2	WIN1	WIN0	—
bit 7							bit 0

CANSTAT: CAN STATUS REGISTER

R-1	R-0	R-0	U-0	R-0	R-0	R-0	U-0
OPMODE2	OPMODE1	OPMODE0	—	ICODE2	ICODE1	ICODE0	—
bit 7							bit 0

COMSTAT: COMMUNICATION STATUS REGISTER

R/C-0	R/C-0	R-0	R-0	R-0	R-0	R-0	R-0
RXB0OVFL	RXB1OVFL	TXBO	TXBP	RXBP	TXWARN	RXWARN	EWARN
bit 7							bit 0

Figura 3.4: Registros más significativos relacionados con el módulo *CAN*.

3.4.2 Modos de operación

El módulo *CAN* de los PIC18FXX8 puede trabajar en seis modos de operación: de configuración, desactivado, operación normal, modo de sólo escucha, modo de *loopback* y de reconocimiento de errores. Salvo el último, los modos de operación son accedidos mediante la activación de una combinación determinada de los bits *REQOP*, situados en el registro *CANCON*. La entrada en un modo es comprobada previamente mediante la comparación de los bits *OPMODE.CANSTAT* con los ya mencionados (consultar [18], página 226 y ss.).

El cambio de modo no se llevará a cabo hasta que todos los mensajes pendientes sean transmitidos, siendo tarea del programador asegurarse de abortar estos mensajes o verificar el modo en el que se encuentra el módulo si el microcontrolador va a realizar instrucciones que dependen de ello. A continuación vamos a describir sucintamente los distintos modos.

Modo de Configuración

La inicialización y posteriores cambios en la configuración del módulo *CAN* sólo se pueden realizar en este modo. Una vez solicitado el modo mediante la puesta a '1' del bit *CANCON.REQOP2* y verificado que el bit *CANSTAT.OPMODE2* también lo está, los registros de configuración, los asociados al *bit-timing*, las máscaras y los filtros pueden ser modificados. Para evitar violaciones accidentales del protocolo *CAN* por errores de programación mientras el módulo esté activo, éste protege todos estos registros, de modo que no podremos acceder al modo configuración mientras alguna transmisión o recepción tenga lugar.

Una vez accedido a este modo, se puede modificar todos los registros relacionados con la configuración, pero no transmitir o recibir tramas. Los distintos contadores de error son inicializados y los bit de *flags* de las interrupciones permanecen sin cambios.

Modo desactivado

En este modo no se pueden recibir ni trasmitir mensajes *CAN*. La petición de acceso se hace mediante los bits *CANCON.REQOP*. Entonces el módulo en estado activo, cuando detecte 11 bits recesivos (*recessives*) consecutivos en el bus *CAN*, interpreta la condición de bus *off* y aceptará el comando de entrada al modo desactivado.

La única interrupción del módulo que permanece activa en este modo es la *WAKIF*, que si se encuentra habilitada, al detectarse actividad en el bus, esto es, se ha detectado un bit dominante (*dominant*), se genera una interrupción, y el programador puede utilizarlo para entrar en un modo distinto. Cualquier interrupción que quedase pendiente se resolverá y los contadores de error mantendrán sus valores.

Hay que tener cuidado si en el modo desactivado hemos "dormido" el microcontrolador, es decir, hemos ejecutado la instrucción *SLEEP*, pues al "despertarse" por medio de la interrupción *WAKIF* varios mensajes *CAN* podrían llegar a perderse. El programador deberá tener presente esto al desarrollar el software. Si el microcontrolador está trabajando en estado normal, sólo llegaríamos quizás a perder un mensaje.

Modo normal

Se trata del modo de operación estándar del módulo *CAN*, en el que el microcontrolador monitoriza activamente todos los mensajes presentes en el bus y genera los correspondientes bits de confirmación o tramas de error según corresponda. Es el único modo en el que los PIC18FXX8 pueden transmitir mensajes a través del bus *CAN*.

Modo de sólo escucha

Este modo proporciona una manera de poder recibir todos los mensajes, incluidas las tramas de error. Sin embargo, es un modo silencioso, es decir, en este estado el módulo no puede transmitir ningún tipo de mensaje, tanto de datos o *remote frame*, como mensajes de error o bits de confirmación. También es importante señalar que los contadores de error son inicializados y desactivados en este modo.

Todas estas características convierten este estado en una herramienta muy útil para distintas aplicaciones que monitorizan el bus y para la auto-detección de la tasa de transmisión. Ésta se utiliza cuando un nodo quiere entrar en una red *CAN* formada al menos por otros dos nodos que se comunican entre sí, con una tasa de transmisión desconocida. El nuevo nodo irá probando distintas configuraciones del tiempo de bit en el modo de escucha hasta recibir mensajes válidos, sin que mientras haya colapsado el bus con tramas de error que perturbarían la comunicación en la red *CAN* original.

Modo de *loopback*

Este modo permite la transmisión interna de mensajes, desde los *buffers* de transmisión directamente a los *buffers* de recepción, sin que sean puestos físicamente en el bus. Al ser ignorados los bits de confirmación, las tramas son recibidas como si procedieran de otro nodo. Se trata, al igual que el anterior, de un modo silencioso en el que no son permitidas la transmisión de tramas de ningún tipo, incluyendo bits de confirmación y de error.

El sistema de máscaras y filtros se mantiene, por lo que dicho modo facilita el desarrollo y prueba del sistema *CAN*.

Modo de reconocimiento de errores

El módulo no rechaza ninguna trama, recibiendo y copiando en el *buffer* de recepción todos los mensajes, tanto válidos como inválidos.

3.4.3 Configuración del *bit-timing*

Como ya vimos en el apartado 2.2.1 (página 36) el protocolo *CAN* usa la codificación NRZ, que no incluye el reloj en la secuencia de datos, por lo que debe ser recuperado por los nodos receptores y sincronizado con el reloj del nodo transmisor. Como los nodos pueden tener distintos osciladores y los instantes de transmisión varían, es preciso que los receptores tengan el mismo tipo de PLL sincronizado con los flancos de la señal de transmisión de datos, para así poder recuperar el reloj.

En los PIC18FXX8 se usa un DPLL (Sistema de enganche de fase digital) para implementar el *bit-timing*, configurado para sincronizarse con la señal de entrada, proporcionando a su vez la base de tiempos para la transmisión. El DPLL divide el tiempo de bit en varios segmentos, múltiplos del periodo de tiempo mínimo T_q, denominado "cuanto de tiempo".

Todos los nodos de la red *CAN* deben de tener la misma tasa de transmisión para que la comunicación sea compatible. Sin embargo los nodos no tienen porque tener la misma frecuencia de oscilación en el reloj que regula el microcontrolador. Mediante el ajuste de la preescala de la tasa de transmisión y del número de T_q en cada segmento, conseguiremos obtener una misma tasa de transmisión. Dicha preescala así como los distintos segmentos que conforman el tiempo de bit, ver apartado 2.2.1 (página 36), se configuran con los siguientes registros:

BRGCON1: BAUD RATE CONTROL REGISTER 1

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SJW1	SJW0	BRP5	BRP4	BRP3	BRP2	BRP1	BRP0
bit 7							bit 0

BRGCON2: BAUD RATE CONTROL REGISTER 2

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SEG2PHTS	SAM	SEG1PH2	SEG1PH1	SEG1PH0	PRSEG2	PRSEG1	PRSEG0
bit 7							bit 0

BRGCON3: BAUD RATE CONTROL REGISTER 3

U-0	R/W-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0
—	WAKFIL	—	—	—	SEG2PH2(1)	SEG2PH1(1)	SEG2PH0(1)
bit 7							bit 0

Figura 3.5: Registros que configuran el *bit-timing* del módulo *CAN*.

La codificación NRZ requiere el *bit stuffing*, que tiene lugar cuando se transmiten 5 bits con la misma polaridad y consiste en introducir un bit adicional de polaridad opuesta, necesario para mantener la sincronización del DPLL. El microcontrolador introduce este bit extra automáticamente.

3.4.4 Recepción de mensajes

La familia PIC18FXX8 presenta dos *buffers* de aceptación completa con múltiples filtros para cada uno de ellos. En la figura 3.6 podemos ver como existe un tercer *buffer*, el BEM (*Buffer* de Ensamblaje de Mensajes). Este *buffer* está continuamente obligado a recibir el siguiente mensaje del bus. Si el identificador concuerda con el criterio establecido por

máscaras y filtros para aceptar mensajes, el mensaje del BEM es transferido a uno de los otros dos *buffers*, denominados *RXB0* y *RXB1*. El procesador podrá entonces acceder al *buffer* correspondiente para hacer el tratamiento oportuno en cada caso. El procesador es capaz de acceder a uno de los *buffers* mientras el otro está preparado para una nueva recepción o mantiene un mensaje recibido con anterioridad.

Cuando un mensaje es movido del BEM a uno de los *buffers RXBn*, el bit de *flag* correspondiente, *RXBnIF*, es puesto a '1'. Mientras esto suceda, la escritura de un nuevo mensaje no está permitida por lo que el programador debe poner este bit a '0' al finalizar el tratamiento de la trama recibida. Podemos habilitar una interrupción que nos avise de la activación de este bit de *flag*.

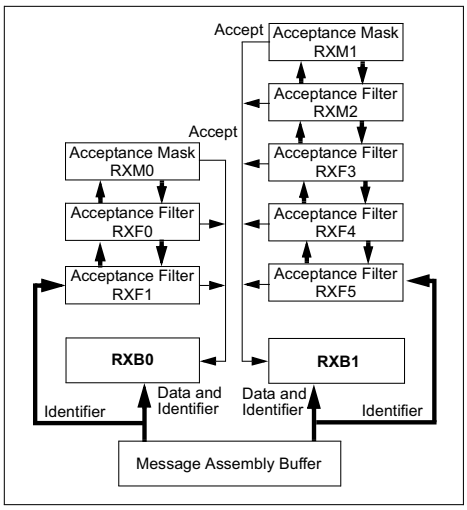


Figura 3.6: Diagrama de bloques del módulo de recepción *CAN*.

Cada *buffer* de recepción, *RXB0* y *RXB1*, tiene asociado una máscara de aceptación de mensajes programable. *RXB0* es el *buffer* de mayor prioridad y tiene asociado dos filtros, mientras que *RXB1* es el de baja prioridad y presenta cuatro filtros. Este mayor número de filtros hacen a este *buffer* menos restrictivo a la hora de comparar identificadores, razón por la cual se le concede una prioridad menor respecto *RXB0*. No obstante puede configurarse el módulo receptor de tal forma que si una nueva trama válida es recibida y *RXB0* contiene un mensaje válido, no se produce error de sobrecarga y el nuevo mensaje es desplazado a *RXB1*, sin tener en cuenta los criterios de filtros de éste. Mediante la configuración oportuna podremos seleccionar que tipo de mensajes queremos que sean analizados: todo tipo de tramas, todos los mensajes válidos o bien sólamente mensajes con indentificador estándar o con identificador extendido (ver *datasheet*, página 230 y ss. [18]).

Como vimos en el apartado 2.3.2 (página 42) el criterio de selección de mensajes se basa en un sistema formado por máscaras y filtros. Cuando un mensaje válido es recibido en el BEM, el campo correspondiente al identificador es comparado con los valores de los filtros de ambos *buffers*. Si existe coincidencia, el mensaje es copiado en el *buffer* correspondiente. Las máscaras de cada *buffer* nos indican qué bits del identificador deben ser comparados (ver tabla 2.2). Estas máscaras y filtros sólo pueden ser modificadas cuando el módulo está operando en modo configuración.

3.4.5 Transmisión de mensajes

El PIC18FXX8 tiene tres *buffers* de transmisión, cada uno de los cuales ocupa 14 bytes de la memoria SRAM, el tamaño que se necesita para almacenar una trama *CAN* lista para ser enviada. Los registros asociados a la transmisión pueden ser consultados en el *datasheet*, página 199 y ss. [18]. En la figura 3.7 mostramos el diagrama de bloques de los *buffers* de transmisión.

La familia PIC18FXX8 permite establecer un sistema de prioridades sobre el orden en el que deben enviarse los mensajes pendientes que tenemos almacenados en los *buffers* de transmisión. Este orden jerárquico es interno, siendo independiente de cualquier prioridad inherente al esquema de arbitrio de mensajes que presenta el protocolo *CAN*. Con anterioridad al envío del primer campo de la trama *CAN* se compara la prioridad de los *buffers* que están esperando para transmitir. Aquel que tenga la prioridad más alta enviará primero. En caso de igualdad el *buffer* con el número asignado de *buffer* mayor será el que comience a enviar. Existen cuatro niveles de prioridad que pueden ser configurados por el programador.

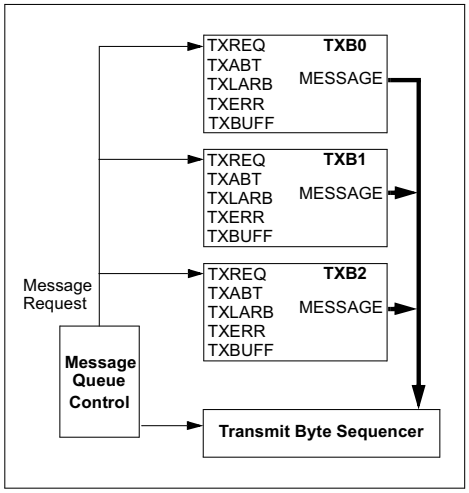


Figura 3.7: Diagrama de bloques del módulo de transmisión *CAN*.

El proceso de transmisión comenzará cuando el módulo detecte que el bus está libre, seleccionando el mensaje con mayor prioridad listo para ser enviado. Cuando la

transmisión ha finalizado, el módulo informa al procesador de tal evento. Si ha ocurrido algún fallo se indicará que el mensaje aún está pendiente de transmisión mediante una interrupción. Igualmente si el mensaje está siendo transmitido pero es detectada una condición de error, otra interrupción, de mensaje no válido, es generada.

Tenemos la posibilidad de abortar todos los mensajes pendientes de enviarse, salvo en el caso de que un mensaje haya comenzado a transmitirse, el cual sí se permite. Si ocurre un error o pierde el arbitrio por el bus dicho mensaje, no será retransmitido; tanto en este caso como si ha habido éxito, el procesador será informado oportunamente.

3.4.6 Detección de errores

El protocolo *CAN* proporciona distintos mecanismos de detección de errores, que son reconocidos por el módulo *CAN* de los PIC18FXX8. Los siguientes errores pueden ser detectados:

1. **Error CRC**: éste se produce cuando el cálculo de Chequeo de Redundancia Cíclica (CRC) por parte del nodo receptor a partir del mensaje recibido no coincide con el campo CRC incluido en la trama generada en el nodo transmisor. El mensaje se vuelve a enviar.

2. **Error de asentimiento**: si en el intervalo de trama de confirmación, el transmisor detecta un bit dominante (en lugar de un bit recesivo) entonces es que ha habido un error en la recepción en algunos de los otros nodos otros nodos. Se genera una trama de error y el mensaje se vuelve a enviar.

3. **Error de formato**: un bit dominante es detectado en alguno de los segmentos en los que nunca debería de haberlo si todo funciona correctamente. Estos segmentos son el Final de Trama, el Espacio entre Tramas (*Interframe Space*), el campo CRC o el Intervalo de Confirmación (ACK). Se genera una trama de error y el mensaje se vuelve a enviar.

4. **Error de *stuff***: si entre el comienzo de la trama y el campo CRC son detectados seis bits consecutivos con la misma polaridad se está violando la regla de *stuff*. Se genera una trama de error y el mensaje se vuelve a enviar.

Los errores que detecta un nodo son comunicados al resto de elementos de la red *CAN* mediante las tramas de error. La transmisión de un mensaje erróneo es automáticamente abortada y se intenta enviar de nuevo tan pronto como sea posible.

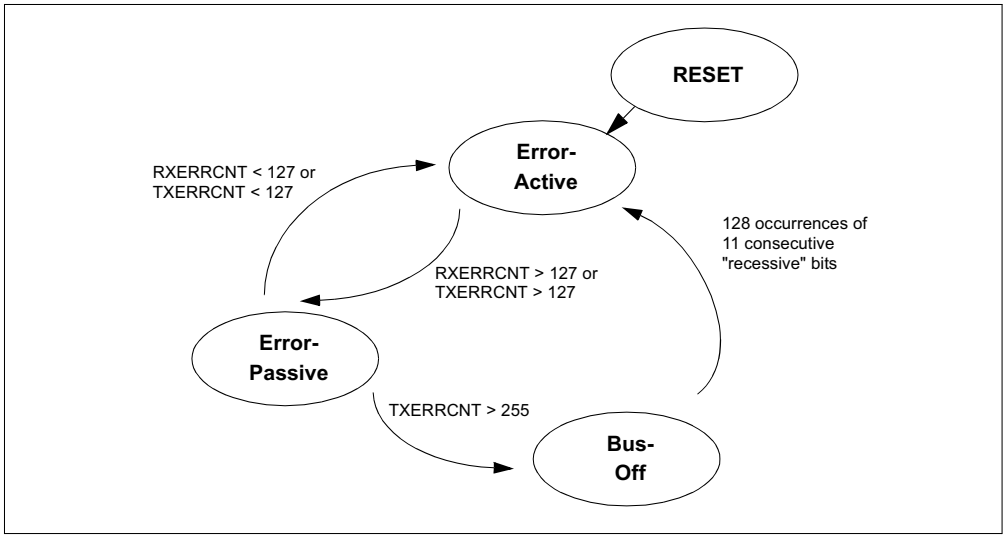


Figura 3.8: Diagrama de estados de error del módulo *CAN*.

Los PIC18FXX8 tienen dos registros que sirven como contadores de error: Contador de Error de Recepción, el registro *RXERRCNT*, y el Contador de Error de Transmisión, el registro *TXERRCNT*. Ambos contadores son incrementados y reducidos en función de si se reciben o transmiten mensajes erróneamente o de una forma correcta, todo ello de acuerdo con la especificación del *CAN* bus. En función del valor de estos contadores, el módulo *CAN* se puede encontrar en uno de los siguientes estados (ver figura 3.8):

- **Error Activo**, cuando ambos contadores de error están por debajo de 128. Es el estado normal si no hay complicaciones. En él, el nodo transmite mensajes y tramas de error activas (formadas por bits dominantes), sin ningún tipo de restricción.

- **Error Pasivo**, cuando al menos alguno de los contadores es igual o superior a 128. Se pueden transmitir mensajes y tramas de error pasivas (formadas por bits recesivos, de forma que son ignoradas si coinciden con una trama de error activa).

- **Bus-*off***, cuando el Contador de Trasmisión es igual o excede del límite de 256. Este estado imposibilita temporalmente la participación del nodo en la comunicación en el bus, no pudiendo recibir ni trasmitir mensajes. Cuando tiene lugar 128 recepciones de 11 bits consecutivos recesivos, el módulo se recupera de esta situación, pasando al estado de Error Activo, sin intervención del procesador.

En el registro *COMSTAT* (figura 3.4) existen una serie de bits que nos informan del estado de los contadores de error, que facilitan la elaboración de un control de flujo de tráfico por la red *CAN*.

3.4.7 Interrupciones asociadas al módulo CAN

Hemos visto a lo largo de este subapartado dedicado al módulo *CAN* que existen un número elevado de eventos que llevan asociada una interrupción, mediante las cuales podemos tener un control adecuado sobre la actividad del módulo. Cada una de estas interrupciones puede ser habilitada o deshabilitada individualmente. Los registros que controlan estas interrupciones son el *PIE3*, *PIR3*, *IPR3* y el general *INTCON* (ver figura 3.3).

Todas las interrupciones tienen una única fuente de origen, con excepción de la interrupción de Error, que puede presentar varias. Las dividimos en dos bloques:

1. Relacionadas con la recepción:
   - Interrupciones de Recepción, cuando tiene lugar una recepción con éxito de un mensaje y su posterior copia en uno de los dos *buffers RXB0* o *RXB1*.
   - De Sobrecarga, cuando se ha intentado transferir una trama a *RXB0* o *RXB1* pero una de estos *buffers* o ambos no están disponibles.
   - De Advertencia en la Recepción, cuando el Contador de Error de Recepción alcanza el valor de 96.
   - De Error Pasivo en la Recepción, cuando el Contador de Error de Recepción excede del límite 127, pasando el módulo a estado de Error Pasivo.
   - De *Wake-up* o Detección de Actividad en el bus, que tiene lugar cuando estando el microcontrolador en modo SLEEP se detecta un bit dominante en el bus, sinónimo de existencia de actividad. Esta interrupción "despierta" al PIC18FXX8, saliendo del modo SLEEP.
2. Relacionadas con la transmisión:
   - Interrupciones de Transmisión, que nos avisan cuando el correspondiente *buffer* de transmisión asociado está vacío y preparado para aceptar un nuevo mensaje que se quiera enviar.
   - De Advertencia en la Transmisión, cuando el Contador de Error de Transmisión excede del valor de 96.
   - De Error Pasivo en la Transmisión, cuando el Contador de Error de Transmisión alcanza el valor límite de 127 y el módulo pasa a estado de Error Pasivo.
   - De Bus-*off*, el Contador de Error de Transmisión supera del valor 255 y el dispositivo pasa a estado de Bus-*off*.

Capítulo 4

Tarjetas Multimedia Card y Secure Digital

4.1 Introducción a las tarjetas de memoria flash

Desde la década de los noventa, la sociedad ha venido experimentado una revolución tecnológica en el tratamiento de la información. Cada vez manejamos más cantidad de información en dispositivos móviles cada vez más pequeños, lo que ha obligado a las empresas del sector a desarrollar una nueva familia de dispositivos de almacenamiento cada vez más pequeños, ligeros y con una capacidad de almacenamiento elevada.

En estos años han surgido una serie de estándares que podríamos denominar tarjetas de memoria *flash*. Estos dispositivos se han convertido en la solución más óptima para todas aquellas aplicaciones móviles como reproductores MP3, ordenadores portátiles, teléfonos móviles, cámaras digitales y PDAs. Esta tecnología ofrece a cambio de un elevado coste por mega, una alta capacidad de almacenamiento en un espacio mínimo, lo que las convierte en elementos imprescindibles de la informática de bolsillo.

En esta introducción vamos a repasar las diferentes tecnologías existentes. En la tabla 4.1 presentamos a modo resumen una comparativa de todas ellas.

Compact Flash

Las tarjetas de memoria *Compact Flash* o CF son las tarjetas más voluminosas y también las más económicas [31]. Presentan dos versiones: el tipo I y el tipo II. Se comunican con el dispositivo mediante dos hileras de contactos situados en un borde que suman 50 conexiones. Contienen un controlador IDE/ATA integrado que permite, si el aparato lo aprovecha, una elevada tasa de transferencia. Las CF de tipo I pueden emplearse en ranuras de tipo II.

Las CF tipo II son simplemente una evolución de las de tipo I, pensadas para admitir mayores capacidades. Al tener mayor espesor una tarjeta tipo II no cabe dentro de una ranura tipo I. La mayor parte de tarjetas CF tipo II carecen de partes móviles, como casi

todas las tarjetas de memoria; la excepción es el *Microdrive*, un disco duro en miniatura desarrollado por IBM.

SmartMedia

Las tarjetas *SmartMedia* reciben algunas veces el nombre de tarjetas SSFDC (acrónimo de *Solid-State Flash Digital Card*) [32, 33, 34, 35]. Este estándar ha sido desarrollado por *Toshiba Corporation*. De dimensiones similares a las CF, son sin embargo mucho más delgadas. Se distinguen rápidamente por su superficie de contactos dorados que cubre la mitad de una sus caras. Inicialmente se fabricaron dos variedades similares pero a menudo incompatibles entre sí (de 3,3V y de 5V), pero hoy en día ya sólo se utilizan las de 3,3V.

Multimedia Card

El aspecto externo *MultiMedia Card* o MMC [15] recuerda al de las *SmartMedia*, pero su tamaño es inferior (como un sello de correo grande). Muestran siete contactos dorados en el extremo de una de las caras. En la actualidad es junto a *SmartMedia* uno de los estándares con menos capacidad de almacenamiento. Su particular ventaja es que puede trabajar en dos modos distintos. El primero es el protocolo *Multimedia Card*, pero además soporta como modo secundario el protocolo SPI. Esta característica permite desde el punto de vista de una aplicación, poder desarrollarla con un bajo coste y un mínimo esfuerzo.

Secure Digital

Las tarjeta *Secure Digital* o SD son tarjetas MMC de segunda generación desarrolladas por Panasonic, Toshiba y SanDisk. Son ligeramente más gruesas e incorporan circuitería adicional para gestión de derechos digitales, pensada para frenar la copia no autorizada de archivos. Un aparato que acepta tarjetas SD habitualmente admite también MMC, aunque no siempre. Las tarjetas SD también soportan el protocolo SPI.

xD-Picture Card

xD-Picture Card son las últimas en aparecer. Estas tarjetas han sido introducidas por Fuji y Olympus en el año 2002, y son las más reducidas: apenas el tamaño de un sello de correo pequeño. No obstante, su precio es de los más elevados junto al del *Memory Stick*.

Memory Stick

Sony desarrolló su propia tecnología de memoria *flash*: el *Memory Stick*. Su forma alargada es muy característica. Algunos dispositivos (en particular reproductores MP3) requieren un tipo especial de *Memory Stick* con características de gestión digital de derechos: éstos reciben el nombre de *Magic Gate Memory Stick*.

	Tamaño (mm)	Peso (g)	Pines	Capacidad max.[1]	Transferencia max.[2]	Ancho de bus	CRC[3]	PW[4]
Compact Flash	42,8 x 36,4 x 0,76	11	50	4 GB	6MB/s	8 o 16	Si	No
Smart Media	37 x 45 x 0,76	2	22	128 MB	-	8	No	No
Multimedia Card	24 x 32 x 1.4	1,5	7	128 MB	1MB/s 200KB/s	1	Si	No
Secure Digital	24 x 32 x 2,1	2	9	1 GB	10MB/s 2MB/s	4	Si	Si
xD-Picture Card	25 x 20 x 1,7	2	18	512 MB	5MB/s 3MB/s	8	No	No
Memory Stick	50 x 21,5 x 2,8	4	10	128 MB	2,45MB/s 1,8MB/s	1	No	Si
Memory Stick PRO	50 x 21,5 x 2,8	4	10	1 GB	20MB/s	4	Si	Si

Tabla 4.1: Comparativa de los diferentes estándares de tarjetas de memoria *flash*.

4.2 Multimedia Card y SD Card en modo SPI

Las tarjetas MMC y SD soportan el protocolo de comunicaciones SPI. El protocolo SPI es un protocolo serie asíncrono desarrollado originalmente por Motorola y que actualmente muchos fabricantes implementan en sus familias de microcontroladores. Por lo tanto con muy poco esfuerzo en el desarrollo se pueden generar aplicaciones que incorporen dichas tarjetas.

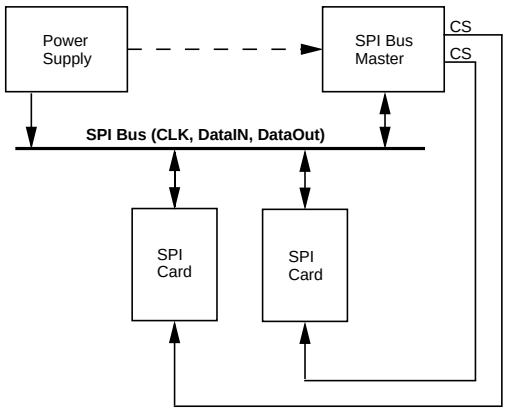
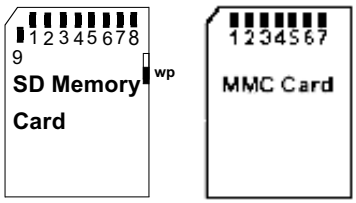


Figura 4.1: Interfaz SPI para tarjetas MMC y SD.

[1]Año 2004.
[2]Lectura / Escritura.
[3]Corrección de errores.
[4]Protección contra escritura.

La figura 4.1 muestra el esquema general del protocolo SPI, en el que podemos observar la existencia de un nodo maestro que mediante la señal CS (*Chip Select*) selecciona el nodo esclavo con el que quiere comunicarse (en nuestro ejemplo una tarjeta SD). La señal CS debe de ser puesta a nivel bajo, debiendo permanecer así durante toda la comunicación. El bus SPI consta de tres señales, una correspondiente al reloj, CLK, y dos de datos, DI (*Data In*) y DO (*Data Out*). En la tabla 4.2 presentamos el esquema de pines para ambos tipos de tarjeta para el protocolo SPI.



	SD Card			Multimedia Card		
Pin	Nombre	Tipo	Descripción	Nombre	Tipo	Descripción
1	CS	I	Selección de Chip	CS	I	Selección de Chip
2	DI	I	Entrada datos	DI	I	Entrada datos
3	VSS	S	Tierra	VSS	S	Tierra
4	VDD	S	Alimentación	VDD	S	Alimentación
5	CLK	I	Señal de reloj	CLK	I	Señal de reloj
6	VSS2	S	Tierra	VSS2	S	Tierra
7	DO	O	Salida de datos	DO	O	Salida de datos
8	RSV		Reservado			No existe
9	RSV		Reservado			No existe

I: Entrada, O: Salida, S: Alimentación

Tabla 4.2: Esquema de pines para el modo SPI.

4.2.1 Comandos en modo SPI

La comunicación entre el microcontrolador y la tarjeta se establece con una serie de comandos, todos ellos con una longitud de 6 bytes. Cada comando está formado por un primer byte donde se manda el identificador del comando, los cuatro siguientes son el argumento del comando y por último se manda una secuencia CRC de control de errores (ver figura 4.2). El primer bit a enviar es el más significativo.

Cada comando tiene asociado seis bits dentro del primer byte para identificarlo. El código binario de cada comando corresponde con el nombre nemónico del mismo. Por ejemplo al comando CMD0 le corresponde el código binario '000000', o lo que es lo mismo, el primer byte sería el código 0x40 (en hexadecimal); el comando CMD1 en binario sería '000001' o 0x41 para el primer byte y así sucesivamente. Estos comandos generan respuestas de dos tipos: R1 y R2. En las figuras 4.3 y 4.4 presentamos el formato de las mismas. La identificación de cada tipo de respuesta se realiza poniendo a '1' el bit correspondiente.

BYTE 1	7	6	5	4	3	2	1	0
FIELD	0	1	Command					

BYTES 2 - 5	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
FIELD	Argument															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FIELD	Argument															

BYTE 6	7	6	5	4	3	2	1	0
FIELD	CRC						1	x

Figura 4.2: Estructura de los comandos en modo SPI.

BIT	7	6	5	4	3	2	1	0
FIELD	0	Parameter Error	Address Error	Erase Seq Error	Com CRC Error	Illegal Command	Erase Reset	In Idle State

Figura 4.3: Estructura de la respuesta del tipo R1.

BITS	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FIELD	0	Parameter Error	Address Error	Erase Seq Error	Com CRC Error	Illegal Command	Erase Reset	In Idle State	Out of Range	Erase Param	WP Violation	Card ECC Failed	CC Error	Error	WP Erase Skip	0

Figura 4.4: Estructura de la respuesta del tipo R2.

En la tabla 4.3 se encuentran los comandos más comunes, así como el tipo de respuesta que generan. Toda la serie de comandos disponibles se pueden encontrar en las especificaciones de las tarjetas MMC y SD [15, 16, 36, 37]. Hemos incluido el apéndice I (página 381) con el capítulo quinto del manual de las tarjetas SD [37], que contiene las especificaciones para la comunicación SPI de este tipo de tarjetas.

Podemos considerar que las tarjetas de memoria SD son una evolución de las MMC. Las tarjetas SD, además de utilizar todos los comandos de las MMC, incluyen una serie de comandos nuevos denominados *Application Specific Commands* (ACMD). Su estructura es idéntica a los mostrados en la figura 4.2. La manera que tienen las tarjetas SD de distinguir ambos tipos de comandos es por la recepción previa del comando CMD55[5]. Por ejemplo, si queremos enviar el comando ACMD41 habrá que enviar primero el comando CMD55 y a continuación el CMD41.

[5]El comando CMD55 (*Application Command*) es exclusivo de las tarjetas SD.

Comandos MMC y SD			
Comando	Argumento	Respuesta	Descripción
CMD0	Ninguno	R1	Reset
CMD1	Ninguno	R1	Inicialización MMC.
CMD13	Ninguno	R2	Envío del registro de estado
CMD17	[31:0] dirección	R1	Lectura de un bloque
CMD24	[31:0] dirección	R1	Escritura de un bloque
CMD32	[31:0] dirección	R1	Comienzo de borrado
CMD33	[31:0] dirección	R1	Fin de borrado
CMD34	[31:0] cualquiera	R1	Anular la zona marcada
CMD38	[31:0] cualquiera	R1	Borrar la zona marcada
Comandos SD			
CMD55	[31:0] cualquiera	R1	El siguiente comando es ACMD
ACMD41	[31:0] cualquiera	R1	Inicialización SD card

Tabla 4.3: Comandos más importantes para las tarjetas MMC y SD.

4.2.2 Selección de modo en tarjetas MMC y SD

Las tarjetas MMC y SD arrancan por defecto en los modos *Multimedia Card* y *SD Card* respectivamente. Para conmutar al modo SPI, el que vamos a utilizar nosotros, la señal CS debe de ser puesta a nivel bajo durante la recepción del CMD0. En el modo SPI, la comprobación CRC está deshabilitada por defecto, pero como las tarjetas arrancan en los modos *Multimedia Card* y *SD Card*, el CMD0 debe de ser enviado con el código CRC correspondiente. Una vez que la tarjeta cambie de modo, la comprobación CRC se deshabilitará.

CMD0 siempre genera el mismo código CRC de siete bits (0x4A). Añadiendo el '1' correspondiente (ver figura 4.2) se genera el byte 0x95. Por lo tanto la secuencia de seis bytes para enviar el CMD0 será 0x40 0x00 0x00 0x00 0x00 0x95. La secuencia completa para inicializar una tarjeta MMC o SD se muestra en la figura 4.5.

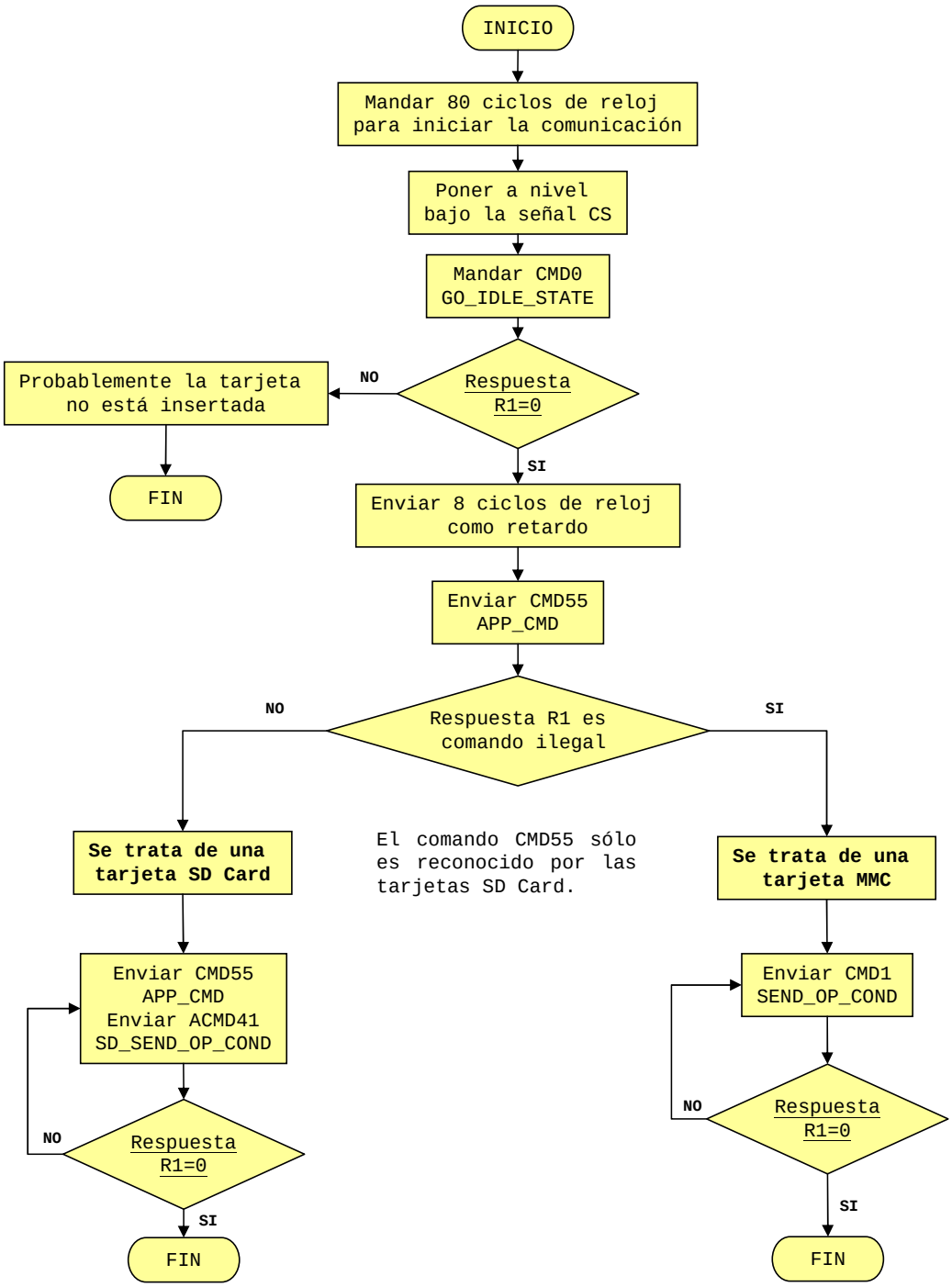


Figura 4.5: Secuencia de inicialización en modo SPI para tarjetas MMC y SD.

4.2.3 Lectura y escritura de datos

En el modo SPI, los procesos de lectura y escritura se realizan en bloques de 512 bytes. La memoria de estas tarjetas está organizada en sectores de 512 bytes y el bloque a leer o escribir debe de estar alineado con el comienzo de alguno de los sectores de la tarjeta.

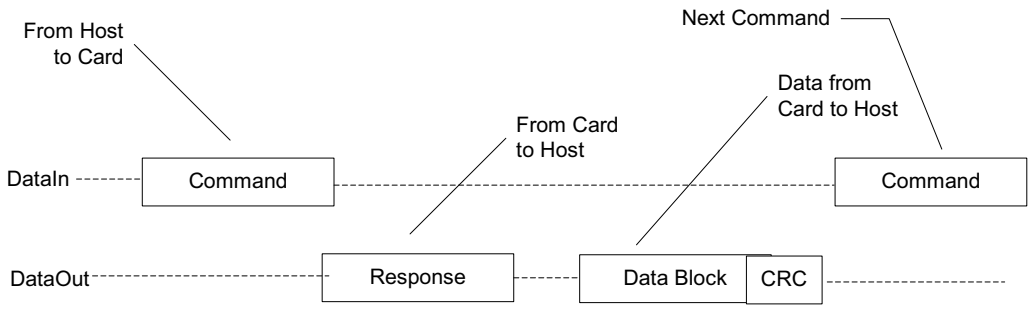


Figura 4.6: Operación de lectura de un bloque de 512 bytes.

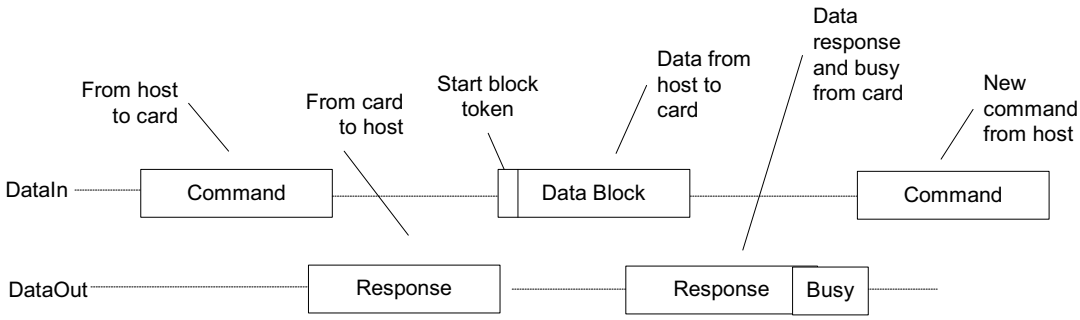


Figura 4.7: Operación de escritura de un bloque de 512 bytes.

En la secuencia de lectura (figura 4.6), después de enviar el comando de lectura CMD17, el microcontrolador debe de esperar a la llegada del byte 0xFE. Posteriormente la tarjeta enviará los 512 bytes correspondientes.

Respecto a la escritura (figura 4.7), el microcontrolador debe mandar el comando de escritura CMD24 y a continuación enviar los 512 bytes de datos a escribir.

4.3 El sistema de archivos FAT

El sistema de archivos FAT (*File Allocation Table*) ha ido evolucionando a lo largo de los años a medida que el tamaño de los dispositivos de almacenamiento masivo han ido creciendo. Todas las versiones de FAT fueron desarrolladas originalmente para la arquitectura IBM-PC, lo que implica que toda la estructura es del tipo *little endian* [17, 38, 39].

En la actualidad existen tres versiones del sistema FAT: FAT12, FAT16, FAT32. El sufijo 12, 16 y 32 hace referencia al número de bits empleado para numerar los *cluster* (aunque la versión FAT32 sólo emplea 28 bits, ya que 4 de ellos están reservados). En función del número reservado de *cluster* podremos tener una capacidad máxima de almacenamiento por volumen o disco lógico (ver tabla 4.4).

	Discos primitivos	**Discos pequeños**	**Discos medianos/grandes**
Tamaño de *cluster*	12 bit	16 bits	32 bits
Tamaño máximo del volumen	16 MB	1 GB	2.048 GB

Tabla 4.4: Tamaño máximo de un volumen en función del tamaño de *cluster*.

Los medios de almacenamiento están divididos en sectores lógicos. Generalmente estos sectores ocupan 512 bytes. En el sistema de archivos FAT, dichos sectores se agrupan en *clusters* y el tamaño del *cluster* dependerá del tamaño de la partición. Se considera al *cluster* como la unidad mínima en el sistema de archivos, y siempre ocuparemos *clusters* enteros a medida que escribimos en un disco. El tamaño de *cluster* es fijado por el sistema operativo cuando formateamos los discos y se basa en el tamaño del disco a formatear. En la tabla 4.5 podemos ver los diferentes tamaños de *cluster*.

Tamaño de la partición	**Tamaño cluster FAT16**	**Tamaño cluster FAT32**
7 MB a 16 MB	2 KB	No soportado
17 MB a 32 MB	512 bytes	No soportado
33 MB a 64 MB	1 KB	512 bytes
65 MB a 128 MB	2 KB	1 KB
129 MB a 256 MB	4 KB	2 KB
257 MB a 512 MB	8 KB	4 KB
513 MB a 1 GB	16 KB	4 KB

Tabla 4.5: Tamaño del *cluster* en función del tamaño del disco.

4.3.1 Sector de Arranque Maestro (*Master Boot Record*)

El sector de arranque maestro (*Master Boot Record* o MBR) está localizado siempre en el primer sector del disco. Es la primera pieza de código que el sistema operativo ejecuta al arrancar. También contiene información de las diferentes particiones que puede contener el disco. El número máximo de particiones primarias es de cuatro. En la tabla 4.6 se pueden ver los diferentes campos que contiene este sector. Cada partición tiene destinados 16 bytes para almacenar datos acerca de dicha partición (ver tabla 4.7).

Offset	Descripción	Tamaño (bytes)
000h	Código ejecutable	446
1BEh	1º Partición (ver tabla 4.7)	16
1CEh	2º Partición	16
1DEh	3º Partición	16
1EEh	4º Partición	16
1FEh	Marca ejecutable (55h AAh)	2

Tabla 4.6: Sector de arranque.

Offset	Descripción	Tamaño (bytes)
00h	Estado de la partición (00h=Inactiva, 80h=Activa)	1
01h	Comienzo de la partición - Cabeza	1
02h	Comienzo de la partición - Cilindro/Sector	2
04h	Tipo de partición (ver tabla 4.8)	1
05h	Final de la partición - Cabeza	1
06h	Final de la partición - Cilindro/Sector	2
08h	Número de sectores entre MBR y el primer sector de la partición	4
0Ch	Número de sectores de la partición	4

Tabla 4.7: Datos de cada partición.

Valor	Descripción
00h	Desconocida o ninguna
01h	*FAT12*
04h	FAT16
05h	MS-DOS Partición extendida 1
06h	FAT16 (Partición mayor de 32 MB)
0Bh	FAT32 (Partición hasta 2048 GB)

Tabla 4.8: Tipos de particiones más comunes.

4.3.2 Estructura del sistema de archivos FAT16

El sistema de archivos FAT tiene cuatro zonas bien diferenciadas. Estas zonas son el sector de arranque, la tabla FAT, el directorio raíz y la zona de datos (figura 4.8). El sector de arranque contiene código ejecutable que utiliza el sistema operativo para el arranque del PC. Además posee información del número de particiones que contiene ese disco.

La zona de datos está dividida en bloques denominados *clusters*. Cada uno de estos *clusters* están numerados en una región llamada tabla FAT(*File Allocation Table*). Cada *cluster* en la tabla FAT apunta al siguiente *cluster* del fichero. El último *cluster* del fichero tiene una marca especial que indica que es el último *cluster* del fichero. El directorio raíz y los subdirectorios contienen la información acerca del nombre del fichero, fechas, atributos y el *cluster* donde comienza el fichero.

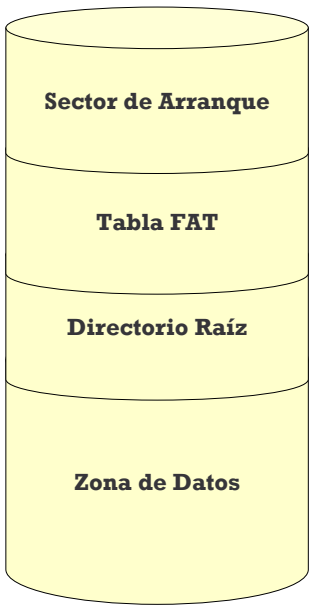


Figura 4.8: Estructura del sistema de archivos FAT16.

Sector de arranque FAT16 *(Boot Record)*

En el primer sector de la partición se encuentra el sector de arranque, donde se encuentra la información que presentamos en la tabla 4.9. Este primer sector permite determinar la posición de la tabla FAT, el directorio raíz, número de sectores por *cluster*, etc.

Offset	Descripción	Tamaño
00h	Instrucción de salto + NOP	3 Bytes
03h	Nombre *OEM*	8 Bytes
0Bh	Bytes por sector	1 Word
0Dh	Sectores por *cluster*	1 Byte
0Eh	Sectores reservados	1 Word
10h	Número de copias de la FAT (generalmente dos)	1 Byte
11h	Número de entradas máximas del directorio raíz	1 Word
13h	Número de sectores en particiones menores de 32MB	1 Word
15h	Tipo de medio (F8h para discos duros)	1 Byte
16h	Sectores por FAT	1 Word
18h	Sectores por *track*	1 Word
1Ah	Número de cabezas	1 Word
1Ch	Número de sectores ocultos en la partición	1 Double Word
20h	Número de sectores en la partición	1 Double Word
24h	Número de unidad lógica de la partición	1 Word
26h	*Extended Signature* (29h)	1 Byte
27h	Número de serie de la partición	1 Double Word
2Bh	Nombre del volumen	11 Bytes
36h	Nombre de la FAT(FAT16)	8 Bytes
3Eh	Código ejecutable	448 Bytes
1FEh	Marca ejecutable (55h AAh)	2 Bytes

Tabla 4.9: Estructura del sector de arranque de una partición FAT16.

Entradas de la tabla FAT

Un *cluster* es un grupo de sectores dentro de la unidad de almacenamiento. Es la unidad mínima de almacenamiento en la zona de datos. Cada *cluster* de la zona de datos está mapeado en la tabla FAT. Cada entrada en la tabla FAT consta de dos bytes. Este número tiene asociado un significado que presentamos en la tabla 4.10. Los *clusters* de la zona de datos empiezan en el número dos.

Directorio raíz

En el directorio raíz se almacena la información acerca de los ficheros, nombre, fecha, tamaño, etc. En la tabla 4.11 se puede observar el formato tradicional de almacenamiento (8 caracteres para el nombre y 3 para la extensión) del sistema DOS 8.3.

Código FAT	Descripción
0000h	*Cluster* libre
0002h a FFEFh	*Cluster* usados, indica siguiente *cluster* del fichero
FFF0h a FFF6h	*Cluster* reservados
FFF7h	*Cluster* dañados
FFF8h a FFFFh	*Cluster* usados, marca fin fichero

Tabla 4.10: Identificación de *clusters* en FAT16.

Offset	Descripción	Tamaño
00h	Nombre	8 Bytes
08h	Extensión	3 Bytes
0Bh	Atributos	1 Byte
16h	Tiempo	1 Word
18h	Fecha	1 Word
1Ah	*Cluster* de inicio	1 Word
1Ch	Tamaño del fichero en bytes	1 Dword

Tabla 4.11: Estructura del directorio raíz.

Determinación de cada una de las regiones

¿Cómo podemos calcular donde empieza cada una de las zonas de la partición? En discos con una única partición, ésta empieza en el sector 0. Toda la información necesaria se encuentra en el sector de arranque de la partición. En la tabla 4.12 presentamos en forma resumida todas las fórmulas necesarias.

¿Qué vamos a calcular?	¿Cómo lo calculamos?
Comienzo tabla FAT	Comienzo partición + Sectores reservados
Comienzo dir. raíz	tabla FAT+ (Nº de copias de la FAT× Sectores por FAT)
Comienzo zona de datos	dir. raíz +((Nº entradas dir. raíz × 32)/Bytes por sector)

Tabla 4.12: Fórmulas para la determinación de las distintas zonas.

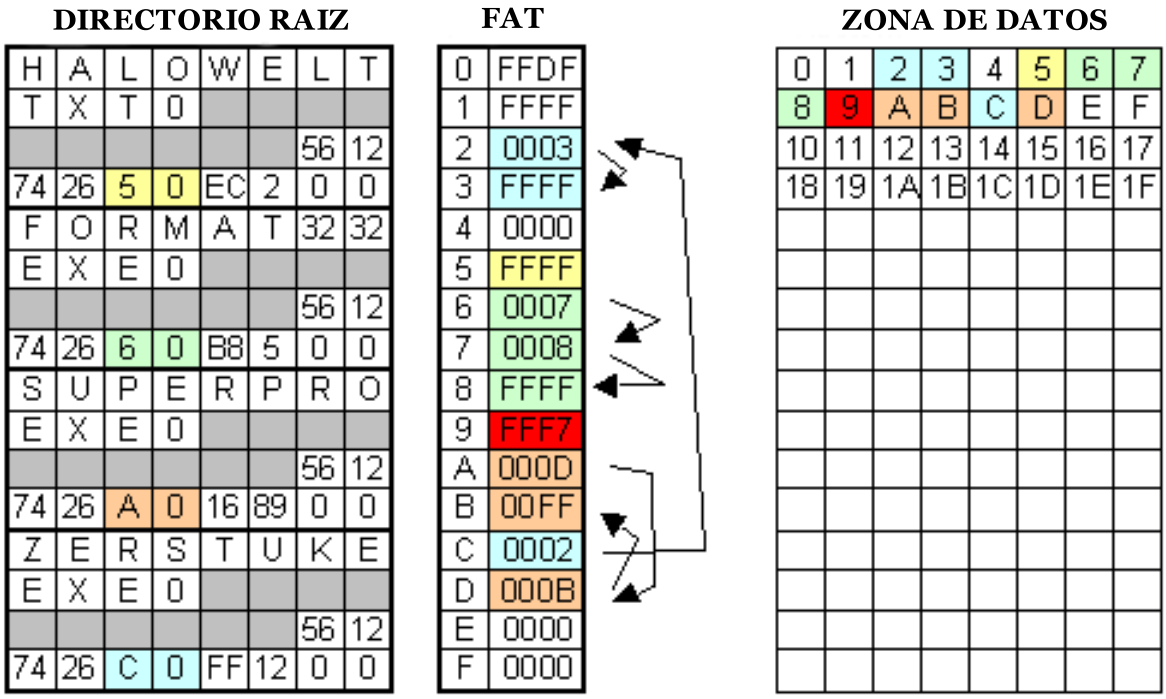


Figura 4.9: Sistema de almacenamiento de ficheros en FAT.

Proceso de lectura de un fichero

Para terminar de comprender el funcionamiento del sistema de archivos FAT, vamos a explicar cómo accederíamos a un fichero, apoyados en la figura 4.9. Supongamos que necesitamos acceder a un fichero llamado "`SUPERPRO.EXE`". El primer paso, como es lógico, sería acceder al directorio raíz y buscar el fichero en cuestión. Analizando la entrada observamos que el *cluster* de inicio es el número 10. Con esta información podríamos leer el primer *cluster* del fichero. Para saber los *clusters* que se encuentran a continuación necesitamos acceder a la tabla FAT. Leyendo la entrada número 10 de la tabla FAT sabemos que el siguiente *cluster* del fichero "`SUPERPRO.EXE`" es el número 13. El siguiente al *cluster* 13 es el 11, que tiene la marca de fin de fichero y que por tanto se trata del último *cluster* del fichero.

Capítulo 5

Desarrollo del analizador

5.1 Descripción general

El analizador desarrollado tiene la capacidad de capturar, seleccionar y tratar tráfico de un sistema de comunicación por bus *CAN*. Este dispositivo tiene autonomía propia, contando con un sistema de alimentación formado por una batería. Su transportabilidad se ve favorecida por la inclusión de un zócalo para tarjetas de memoria del tipo *Multimedia Card* y *Secure Digital*, que serán utilizadas como dispositivo de almacenamiento de la información capturada. Esto pospone el uso del PC para el tratamiento de los datos, proporcionando una evidente ventaja en cuanto a versatilidad y comodidad de manejo.

Podemos dividir el esquema general del analizador en cuatro bloques diferenciados: etapa de alimentación, bloque del microcontrolador y *transceiver CAN* , los periféricos añadidos y la parte correspondiente a la tarjeta de memoria. Los distintos módulos son regidos por un microcontrolador PIC18F258. Conectado directamente a él hay un *display* LCD que mantiene al usuario informado constantemente de la etapa de funcionamiento en la que se encuentra el analizador: la fase de configuración, la espera de la pulsación del botón de inicio de captura, captura de tráfico y el proceso de finalización de análisis de mensajes y cierre del fichero de datos. El aparato dispone de un botón de arranque y parada de captura de tramas, que actúa vía interrupción a través del controlador.

Las tarjetas de memoria utilizadas precisan de una tensión de alimentación diferente que el resto de componentes, por lo que es necesario crear dos líneas de alimentación independientes. La tarjeta de memoria utiliza el protocolo SPI para comunicarse con el microcontrolador, a través de una serie de comandos definidos. Los datos del tráfico *CAN* deben ser enviados en bloques de 512 bytes por requerimiento expreso de la tarjeta. Por esta razón hemos reestructurado la memoria SRAM del microcontrolador para disponer de 1024 bytes consecutivos, organizados en 2 *buffers* de almacenamiento de datos. Cuando uno de ellos esté completo se transferirá íntegro a la tarjeta de memoria.

Los parámetros de configuración para una captura de tráfico *CAN* en concreto están almacenados en un fichero de configuración[1] dentro de la tarjeta de memoria. En la tabla 5.1 presentamos la estructura de dicho fichero, que deberá ser leído por el analizador para configurar correctamente el módulo *CAN*.

Posición de memoria	**Parámetro**	**Posición de memoria**	**Parámetro**
byte [0]	Preescala BRP	byte [22]	Filtro 1 del *buffer* 2
byte [1]	Salto de sincronización SJW	byte [23]	
byte [2]	Segmento de propagación	byte [24]	
byte [3]	Segmento de fase 1	byte [25]	
byte [4]	Segmento de fase 2	byte [26]	Filtro 2 del *buffer* 2
byte [5]	Tipo de trama	byte [27]	
byte [6]	Máscara del *buffer* 1	byte [28]	
byte [7]		byte [29]	
byte [8]		byte [30]	Filtro 3 del *buffer* 2
byte [9]		byte [31]	
byte [10]	Filtro 1 del *buffer* 1	byte [32]	
byte [11]		byte [33]	
byte [12]		byte [34]	Filtro 4 del *buffer* 2
byte [13]		byte [35]	
byte [14]	Filtro 2 del *buffer* 1	byte [36]	
byte [15]		byte [37]	
byte [16]		byte [38]	Modo de operación
byte [17]			
byte [18]	Máscara del *buffer* 1		
byte [19]			
byte [20]			
byte [21]			

Tabla 5.1: Esquema de los parámetros de configuración del módulo *CAN* en memoria del microcontrolador.

Este analizador distingue tramas de datos, remotas y de error. Por otra parte, podemos analizar mensajes con identificador tanto estándar como extendido. La información que precisamos de cada trama analizada es la siguiente:

- Instante de la captura del mensaje.
- Tipo de trama que se trata.
- Identificador del mensaje.
- En caso de ser una trama de datos, el tamaño y el contenido del campo de datos.

En la tabla 5.1 mostramos cómo se organiza en la memoria todos los campos relativos a un mensaje capturado.

[1] El nombre de este fichero es "config.txt".

<table>
<tr><td colspan="8">Byte 15</td><td>B14</td><td>B13</td><td>B12</td><td>B11</td><td>B10</td><td>B9</td><td>B8</td></tr>
<tr><td>15.7</td><td>15.6</td><td>15.5</td><td>15.4</td><td>15.3</td><td>15.2</td><td>15.1</td><td>15.0</td><td colspan="3"></td><td>D7</td><td>D6</td><td>D5</td><td>D4</td></tr>
<tr><td colspan="3">Tipo de trama</td><td colspan="8">Identificador de la trama</td><td colspan="4">Campo de datos</td></tr>
<tr><td>B7</td><td>B6</td><td>B5</td><td>B4</td><td colspan="8">Byte 3</td><td>B2</td><td>B1</td><td>B0</td></tr>
<tr><td>D3</td><td>D2</td><td>D1</td><td>D0</td><td>3.7</td><td>3.6</td><td>3.5</td><td>3.4</td><td>3.3</td><td>3.2</td><td>3.1</td><td>3.0</td><td colspan="3"></td></tr>
<tr><td colspan="4">Campo de datos</td><td colspan="4">Tamaño de la trama</td><td colspan="8">Instante de la captura</td></tr>
</table>

Figura 5.1: Organización de los datos almacenados en una entrada del *buffer*, correspondientes a una trama *CAN*.

Cada entrada en alguno de los dos *buffers* de memoria del microcontrolador, reservados para almacenar las tramas, ocupa 16 bytes. Este valor es divisor exacto de 512 bytes, el tamaño total de cada *buffer*. De esta forma garantizamos 32 entradas por *buffer*. Para el identificador precisamos 29 bits[2], 3 bits para distinguir el tipo de trama, 4 bits para el tamaño y 8 bits para el campo de datos. Esto nos deja 28 bits para almacenar el tiempo de captura. Considerando que el temporizador interno se incrementa en una unidad cada 51,2 μs y podemos almacenar hasta 2^{28} incrementos, el tiempo máximo de captura del que podemos dar cuenta es de unos 229 minutos aproximadamente[3].

El otro factor que nos limita el tiempo de captura es la capacidad de almacenamiento de la tarjeta de memoria. Conociendo dicha capacidad y el intervalo de tiempo medio en el que llega una trama y la siguiente, podemos estimar con la ecuación 5.1 el tiempo total de captura.

$$Tiempo\,de\,captura(ms) = \frac{Capacidad(Mbytes) \times 2^{20}}{16\,bytes} \times Intervalo\,captura(ms) \quad (5.1)$$

El factor 2^{20} se utiliza para convertir a bytes la capacidad de memoria de la tarjeta.

Procesar la recepción de una trama, desde que es capturada hasta que es registrada en uno de los dos *buffers* de almacenamiento de datos, lleva un tiempo de 200 μs. Suponiendo el caso límite de recibir tramas en este intervalo de tiempo, en la tabla 5.2 mostramos los tiempos máximos de captura para distintos tamaños de la tarjeta de memoria.

Tamaño de la tarjeta (MBytes)	Tiempo máximo de captura (s)
32	419
64	839
128	1.678
512	6.711

Tabla 5.2: Tiempos máximos de captura.

[2]Considerando que podemos analizar tramas con identificador extendido.
[3]Cuando se desborda el contador interno del tiempo, el analizador finaliza la captura de tráfico.

5.2 Diseño hardware

En esta sección vamos a describir el hardware del analizador *CAN*. En la figura 5.2 se puede observar un esquema general del analizador. En él se distinguen una serie de bloques:

- Bloque de alimentación.
- Microcontrolador y *transceiver CAN*.
- Bloque de la tarjeta memoria con la adaptación a lógica de 3,3V.
- Botonera y LCD.

En el apéndice H (página 277) se pueden consultar las hojas de especificaciones de los distintos dispositivos utilizados en este diseño[4].

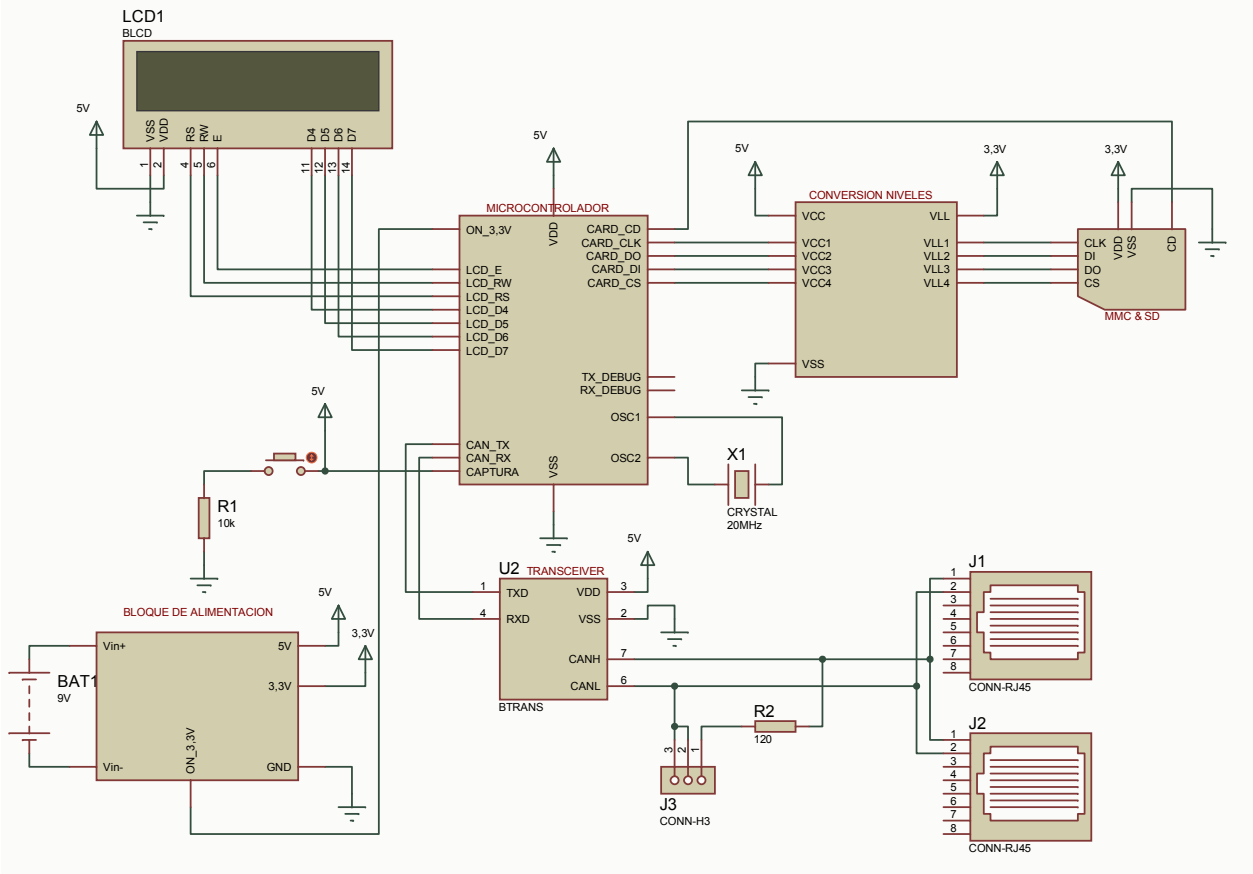


Figura 5.2: Esquema general del analizador *CAN*.

[4]El único *datasheet* que no se ha incluido es el referente al microcontrolador. En el CD adjunto de la memoria se puede encontrar dicha hoja de especifiaciones.

Etapa de alimentación

Este bloque se encarga de obtener 3,3V regulados para alimentar la tarjeta MMC o SD y 5V para el resto del circuito (ver figura 5.3). Se compone de dos reguladores lineales pensados para aplicaciones con baterías y que suministran como máximo 200mA. Al regulador de 3,3V, MAX884, le llega una señal del microcontrolador para activar la salida de 3,3V. La misión de las resistencias R1 y R2 es la de establecer un nivel de tensión a partir del cual se encenderá un diodo LED indicando batería baja. Esta condición se producirá cuando el voltaje de la batería llegue a 4,5V[5]. Este valor es la mínima tensión de alimentación que admite el *trasnceiver* MCP2551.

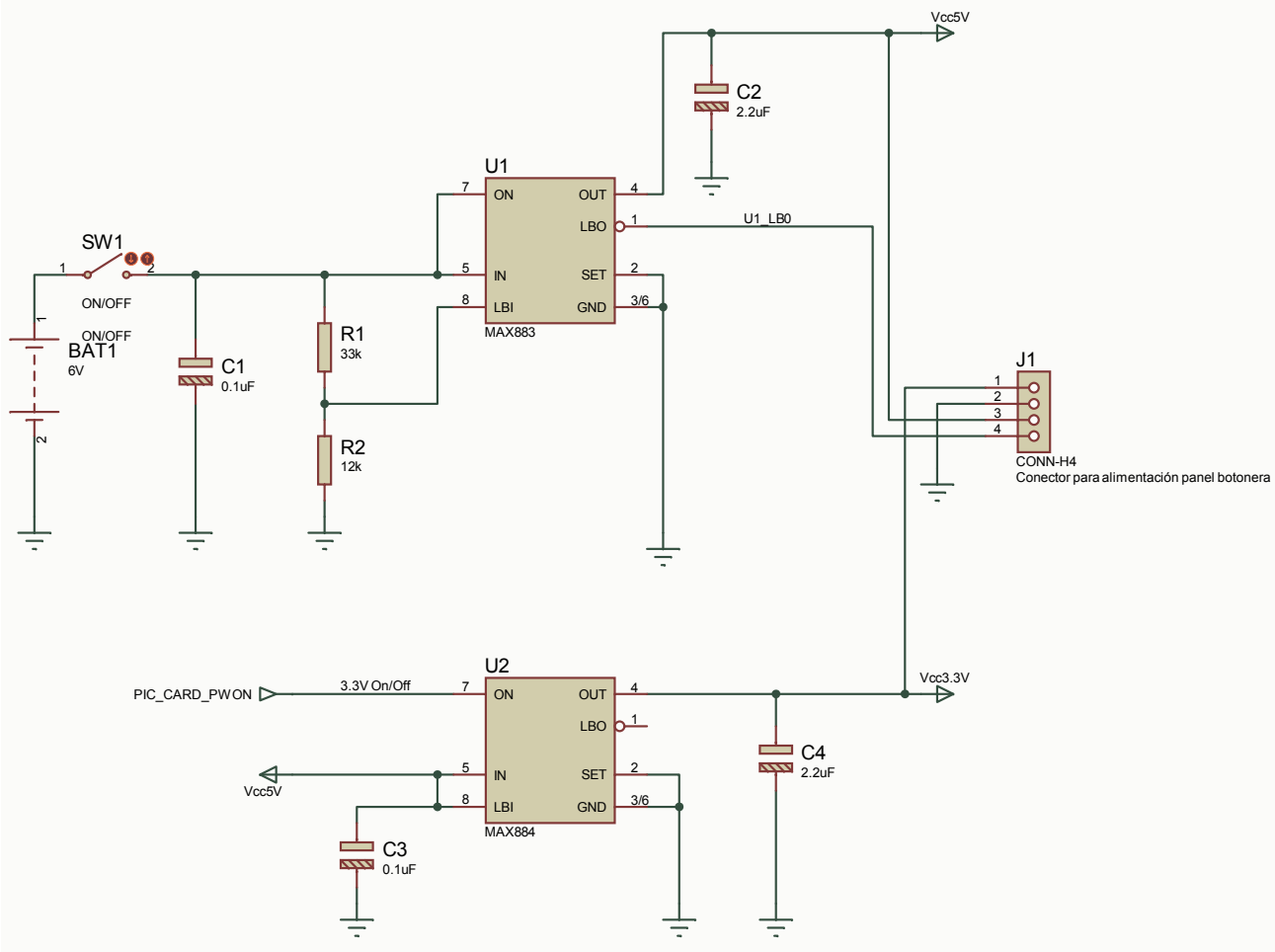


Figura 5.3: Bloque de alimentación.

[5]La entrada LBI se compara internamente con una referencia de tensión de 1,20V. Cuando el nivel de LBI es inferior a 1,20V activa la salida LBO.

LCD y botonera

El analizador posee un LCD de 16 caracteres y 2 líneas para poder visualizar los mensajes informativos. Además posee tres diodos LEDs para indicar las condiciones de bajo voltaje de batería, alimentación de la tarjeta y recepción de tráfico *CAN*. En la figura 5.4 presentamos el esquema de estos componentes.

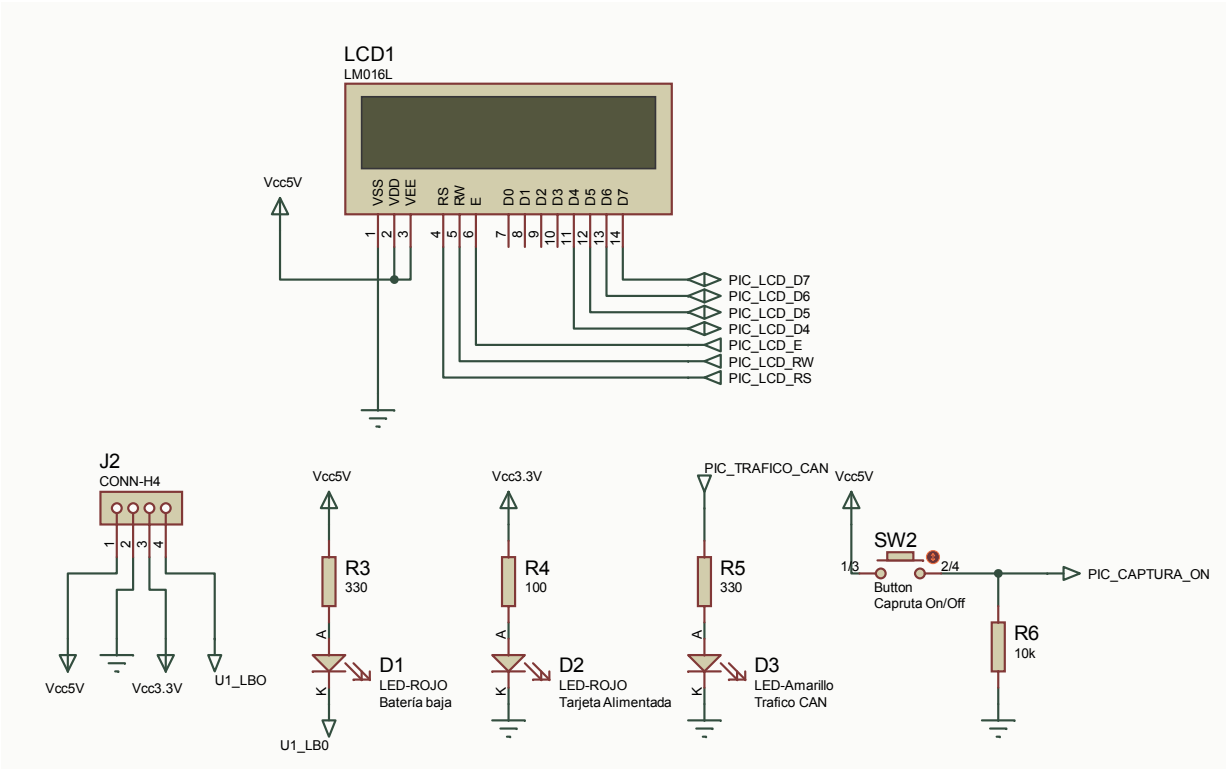


Figura 5.4: Bloque LCD y botonera.

Microcontrolador y comunicación *CAN*

Este bloque es el núcleo del analizador. El microcontrolador PIC18F258 posee un controlador *CAN* que nos permite analizar el tráfico capturado. Además, es necesario un *transceiver CAN* (MCP2551) para adaptar las señales al bus. El analizador posee dos conectores RJ45, de manera que el tráfico que llegue a un conector es derivado al otro. El esquema de pines utilizado en los RJ45 es el propuesto por el estándar CANOpen citado en la sección 2.2 (página 33). De esta forma garantizamos la estructura lineal del bus. Se ha pensado en la posibilidad de que el analizador sea el último nodo de la red. En ese caso, con el *jumper* J6 se activa la resistencia de 120 Ω de terminación[6]. Además, el microcontrolador posee una comunicación serie asíncrona que se ha empleado para tareas de depuración del analizador en la fase de desarrollo. En la realización de la placa se ha mantenido la conexión RS232 para facilitar posibles mejoras[7].

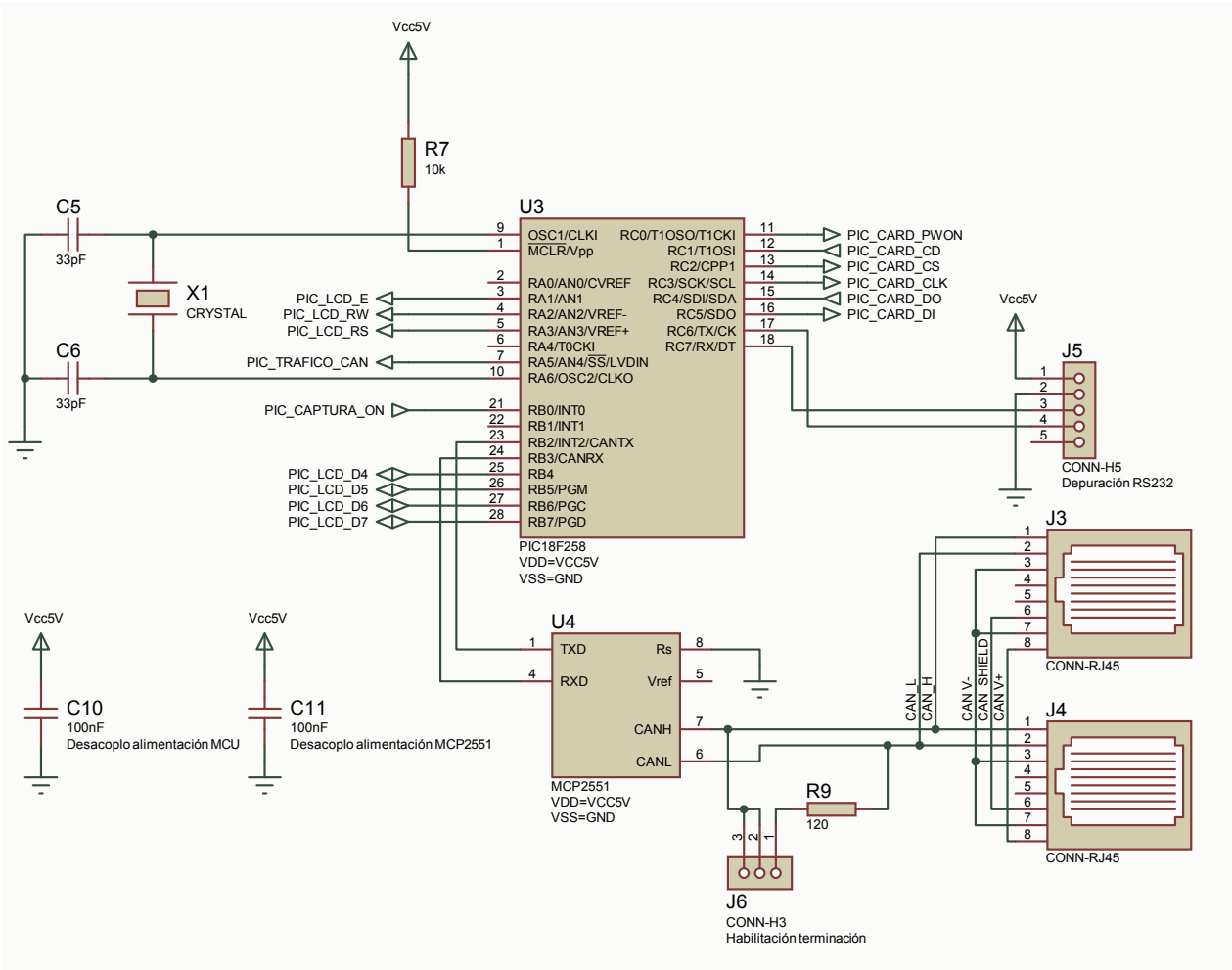


Figura 5.5: Bloque PIC y *transceiver CAN*.

[6]También se pueden usar los terminadores RJ45, ver figura 6.4 en la sección 6.3 (página 131).
[7]No debe de conectarse directamente a un PC sin adaptar las señales. Ver apéndice E (página 253).

Tarjeta de memoria

Las tarjetas de memoria MMC y SD tienen un rango de alimentación entre 2V y 3,6V. Debido a que el resto del circuito está alimentado con 5V, ha sido necesario intercalar entre el microcontrolador y la tarjeta un conversor de niveles (ver figura 5.6). El producto elegido es el conversor bidireccional MAX3002, que permite una trasferencia de 20Mbps.

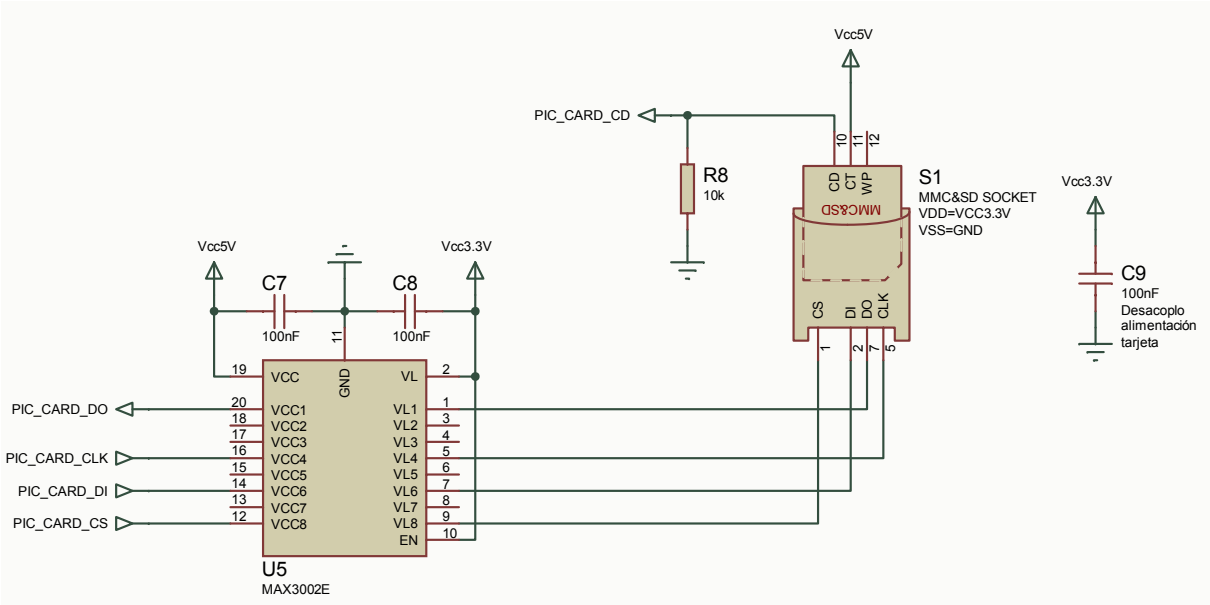


Figura 5.6: Bloque de memoria.

5.3 Diseño software

5.3.1 Comentarios generales

En primer lugar el programa entra en la función *Setup()*, donde se configuran los puertos de entrada/salida, el temporizador y el LCD. Posteriormente la función *LecturaCard()* se encarga de inicializar la comunicación con la tarjeta *flash* y cargar en memoria los parámetros de configuración del controlador *CAN*. Estos datos se configuran a través de la interfaz gráfica del PC, que será comentada con mayor detalle en la sección 5.5 (página 121).

Tres son las posibles causas para que una captura de tráfico *CAN* finalice:

1. Que sea pulsado el botón de OFF, y por consiguiente se haya activado la interrupción de *INT0*.
2. Que el tiempo total de captura haya expirado.
3. Que se haya desbordado la capacidad de almacenamiento total de la tarjeta.

5.3.2 Definiciones de las variables principales y tipos de datos

Se utilizan cuatro tipos de datos enteros:

- **char**: *signed char*, entero de 1 byte con signo.
- **uchar**: *unsigned char*, entero de 1 byte sin signo.
- **uint**: *unsigned int*, entero de 2 bytes sin signo.
- **ulong**: *unsigned long*, entero largo, de 4 bytes, sin signo.

Además se utiliza el tipo de variable *enum*, constante enumerada que crea una lista de elementos afines, *arrays* de variables, y estructuras, *struct*, para agrupar tipos de datos relacionados entre sí.

A continuación enumeramos las variables principales utilizadas, su tipo y su función. Las agrupamos en tres bloques:

1. Relacionadas con la recepción de mensajes *CAN*:
    - ulong NewMessage: contiene el identificador de la última trama capturada.
    - uchar NewMessagedata [8]: almacena el campo de datos de la última trama capturada.
    - uchar NewMessageLen: contiene el tamaño de la última trama capturada.
    - enum CAN_RX_MSG_FLAGS NewMessageFlags: enumeración con los distintos casos que se pueden presentar en la captura de una trama.

- char bufferCard1[512]: es el *buffer*1 donde almacenamos los datos correspondientes a las tramas capturadas. Es un *array* de 512 posiciones.
- char bufferCard2[512]: es el *buffer*2 donde almacenamos los datos correspondientes a las tramas capturadas. Es un *array* de 512 posiciones.
- uint tiempoH: almacenamos aquí la parte baja del instante de tiempo transcurrido desde el inicio de la captura.
- uint tiempoL: almacenamos aquí la parte alta del instante de tiempo transcurrido desde el inicio de la captura.
- uchar tipoTrama: guardamos el tipo de trama según una tabla de valores interna.
- uchar tamanioTrama: contiene la longitud de la trama capturada.

2. Relacionadas con la tarjeta de memoria:

- char nFichero[12]: contiene el nombre del fichero de configuración.
- char nFichero2[12]: contiene el nombre del fichero de captura, donde se guardará toda la información del tráfico capturado.
- extern struct structFAT FAT:
- extern struct structFICHERO FICHERO:
- extern ulong res , res1 , arg1 , arg2
- uint clusterInicio : almacena la entrada en la tabla FAT del primer *cluster* disponible.
- uint cluster : almacena la entrada en la tabla FAT del siguiente *cluster* libre.
- ulong tamanioFAT: guardamos aquí el número total de *clusters* libres inicialmente.

3. Variables de control del programa:

- uchar bufferLleno: si está a '1' nos indica que uno de los dos *buffers*, bufferCard1 o bufferCard2 están completos y que debemos comenzar el correspondiente volcado a la tarjeta de memoria. Al finalizar se pone de nuevo a '0'.
- uchar buffer: cuando vale '0', estamos guardando las tramas en el *buffer* bufferCard1. Si está a '1', escribimos en el *buffer* bufferCard2.
- uint i: es la variable índice que recorre los *buffers* para almacenar los distintos campos del mensaje *CAN*.
- uchar finCaptura: a '1' indica la condición de finalización de análisis de tráfico, e inicia la secuencia de cierre del fichero de captura y de la tarjeta de memoria para poder ser extraída.
- ulong nCaptura: es un contador que se incrementa con cada captura y que nos permite determinar el tamaño del fichero.

5.3.3 Descripción de Funciones

A continuación describimos brevemente las funciones utilizadas, con su prototipo y cometido fundamental. Para una mayor aclaración, se recomienda la consulta de los diagramas de flujo incluidos.

Main

- **Prototipo:** void main(void)
- **Descripción:** comienza con la llamada a la función de configuración de los distintos módulos del microcontrolador. Inicializa la tarjeta de memoria. A continuación espera la secuencia de pulsación del botón de inicio de captura, llama a la función de configuración del módulo *CAN* y activa las interrupciones particularmente y a nivel global. Evalúa continuamente si alguno de los dos *buffers* de memoria está completo. En caso afirmativo se procede a la copia de los datos a la tarjeta de memoria. Si detecta condición de finalización de captura rellena con '0' las entradas de los *buffers* que no hayan sido utilizadas y se realiza el último volcado a la tarjeta. Se procede al cierre del fichero de datos y se realizan los cálculos sobre su tamaño. Esperamos una nueva posible captura.
- **Diagrama de flujo:** hoja1, figura 5.7; hoja2, figura 5.8.

SetupCAN

- **Prototipo:** void SetupCAN(void)
- **Descripción:** configura el módulo *CAN*; fija las máscaras, filtros, el *baud rate*, tipo de tramas que se van a capturar, modo de operación y las interrupciones asociadas (incluyendo su prioridad).
- **Diagrama de flujo:** figura 5.9.

Setup

- **Prototipo:** void Setup(void)
- **Descripción:** configura puertos del PIC18F258, el depurador del programa por puerto serie, el display LCD y el temporizador *TIMER0*. Habilitamos el sistema de prioridades para las interrupciones y clasificamos como de alta prioridad las correspondientes al *TIMER0* y a *INT0*.
- **Diagrama de flujo:** figura 5.10.

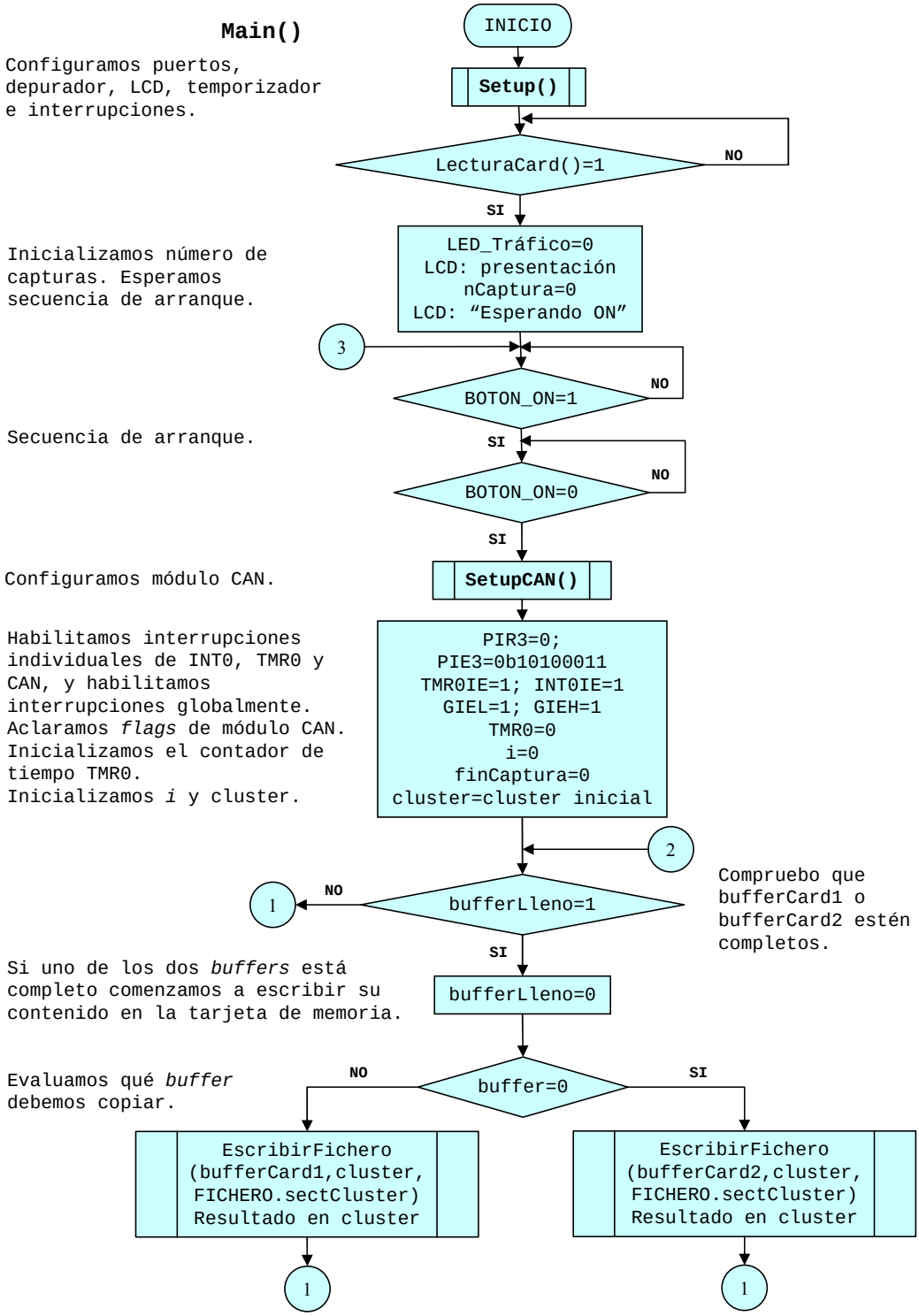


Figura 5.7: Diagrama de flujo de la función Main (hoja1).

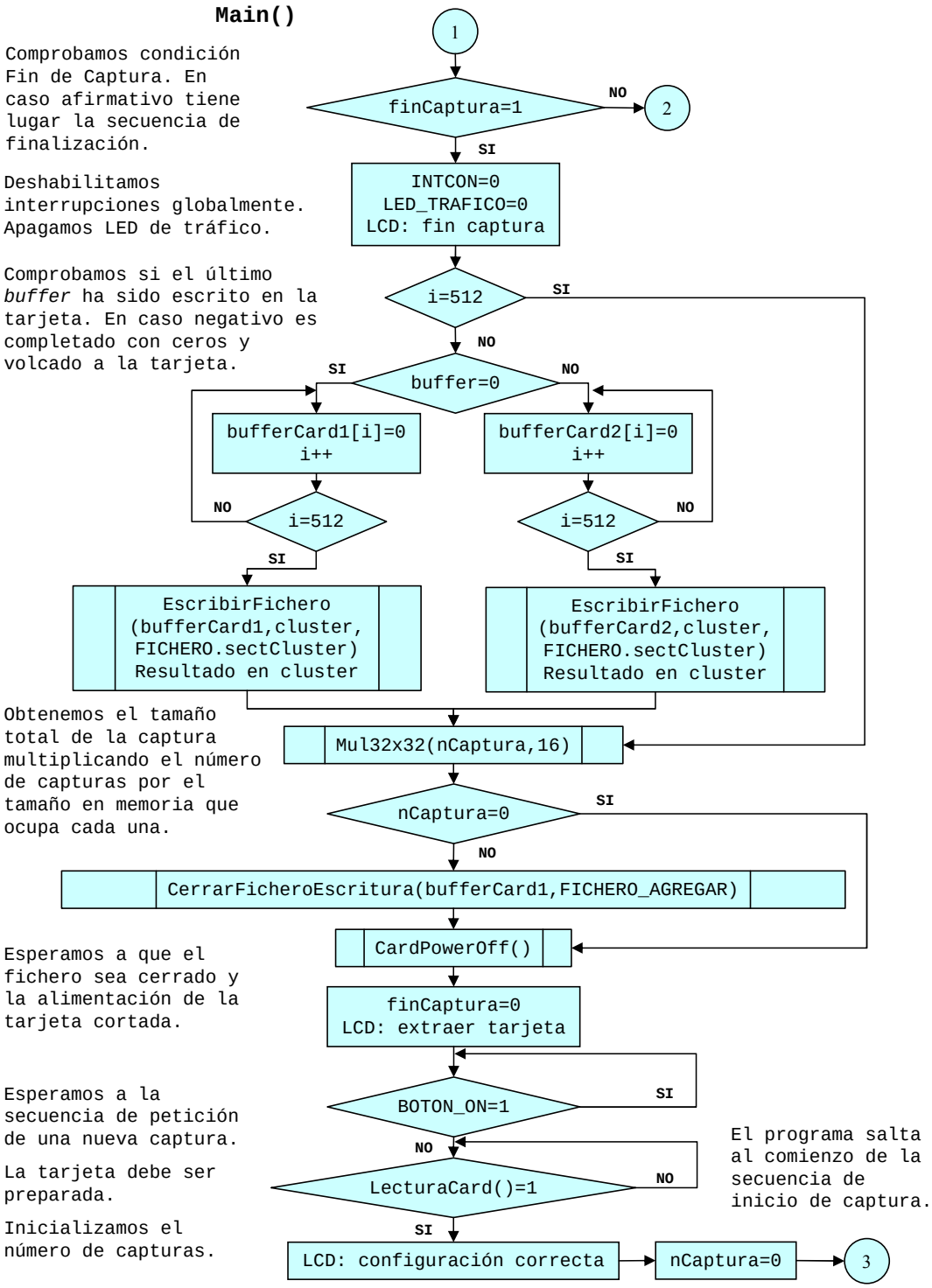


Figura 5.8: Diagrama de flujo de la función Main (hoja2).

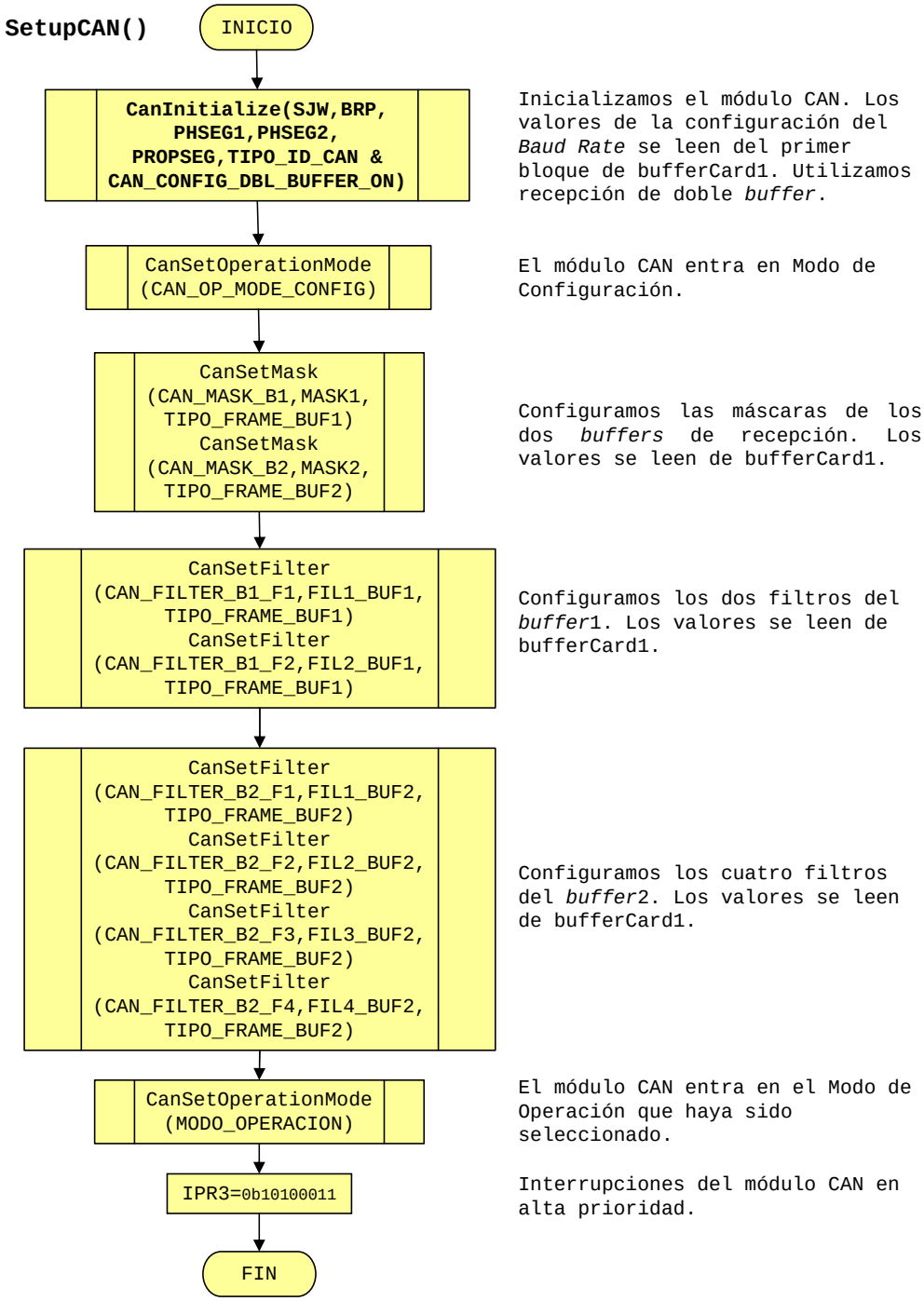


Figura 5.9: Diagrama de flujo de la función SetupCAN.

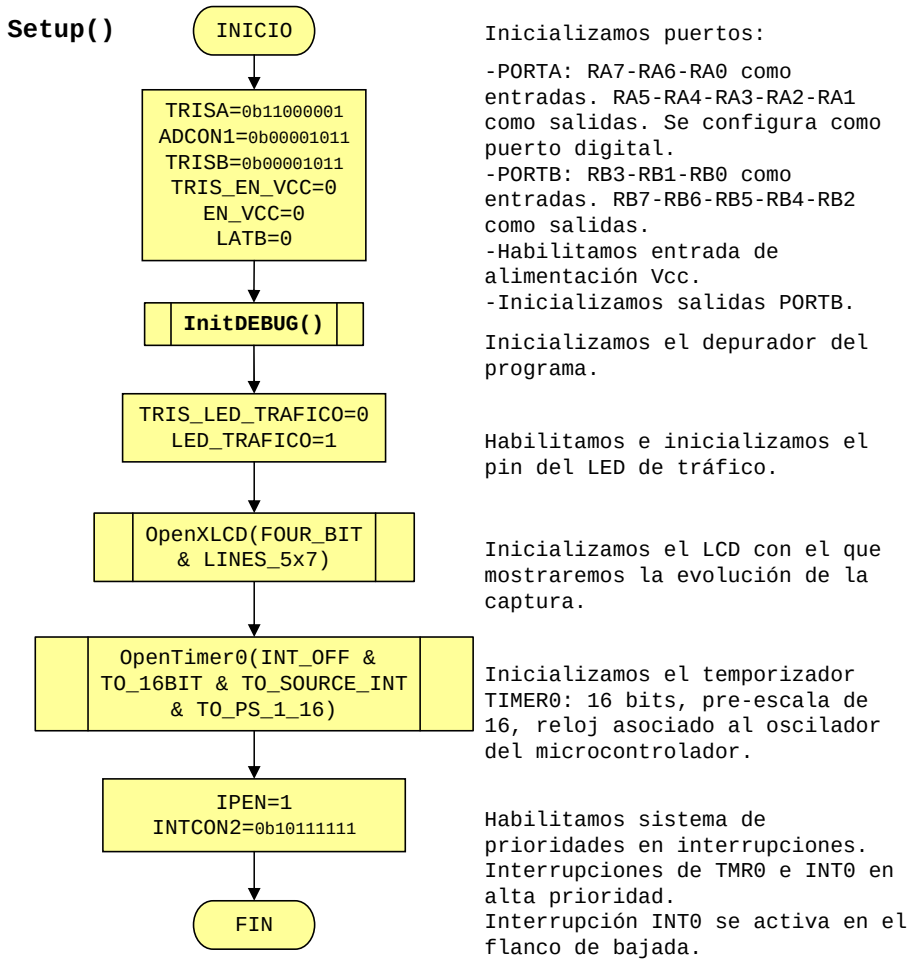


Figura 5.10: Diagrama de flujo de la función Setup.

LecturaCard

- **Prototipo:** unsigned char LecturaCard(void)
- **Descripción:** esta rutina inicializa la comunicación con la tarjeta y carga en memoria los parámetros de configuración del controlador *CAN*.
- **Valor devuelto:** devuelve '1' en el caso que todo haya sido configurado e inicializado correctamente. En caso contrario, devuelve un '0'.
- **Diagrama de flujo:** hoja1, figura 5.11; hoja2, figura 5.12.

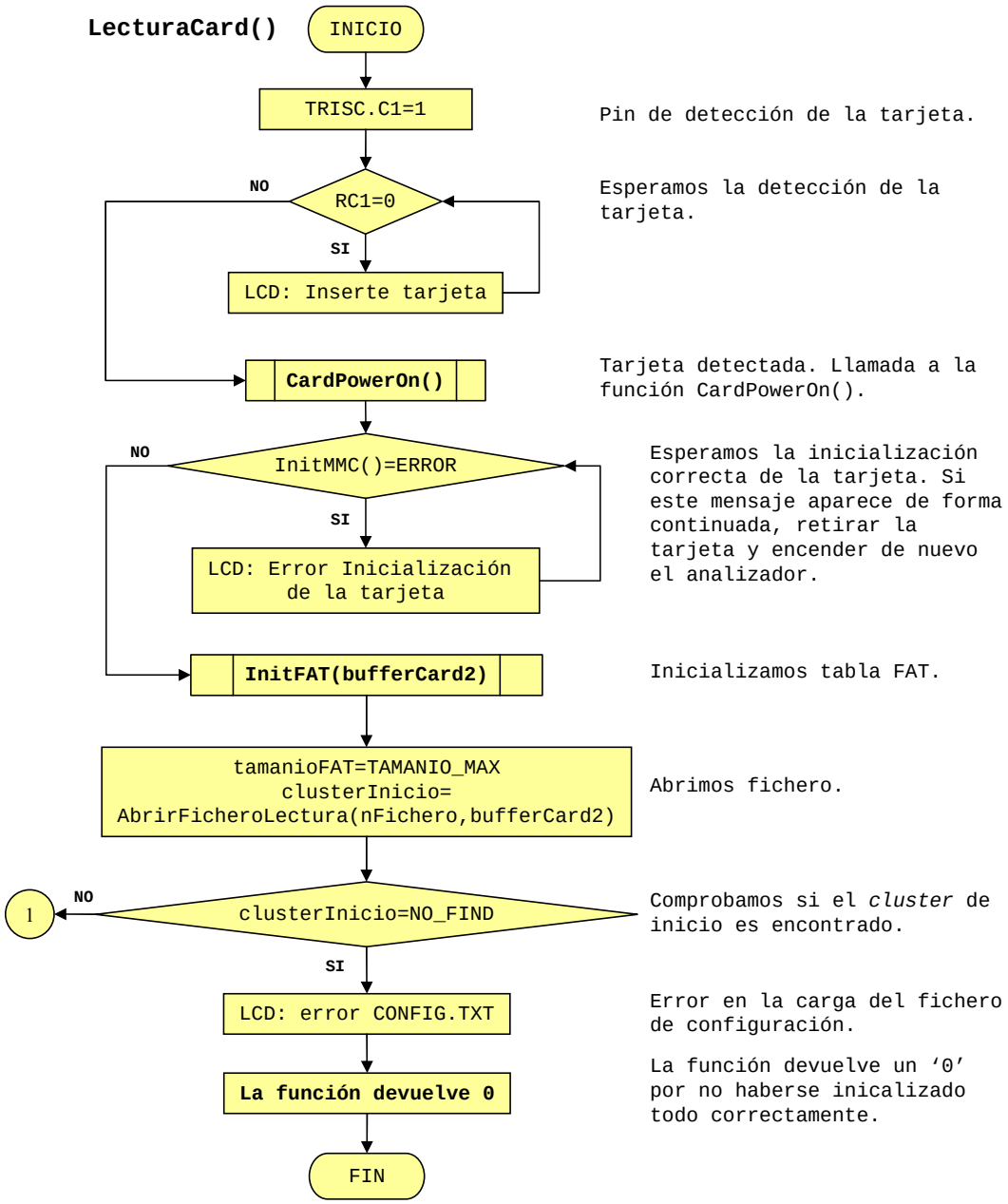


Figura 5.11: Diagrama de flujo de la función LecturaCard (hoja1).

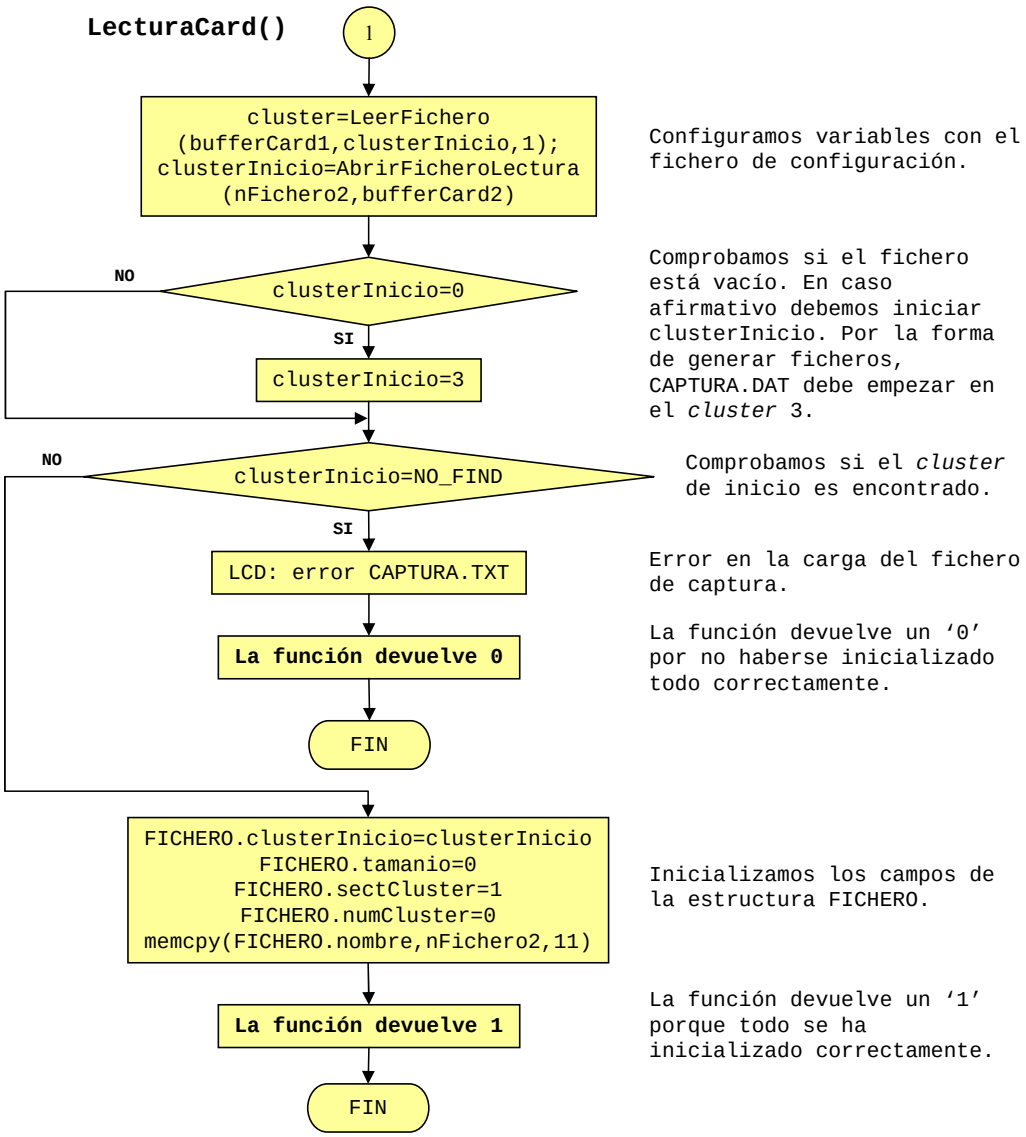


Figura 5.12: Diagrama de flujo de la función LecturaCard (hoja2).

IntCanRx

- **Prototipo:** void IntCanRx(void)
- **Descripción:** es la rutina de atención interrupciones de alta prioridad, en las que se incluyen todas las empleadas: *TIMER0*, *INT0* y las del módulo *CAN*. Analizando los distintos bits de *flag*, deduciremos el origen de la interrupción y se realizará el tratamiento correspondiente.
- **Diagrama de flujo:** hoja1, figura 5.13; hoja2, figura 5.14; hoja3, figura 5.15; hoja4, figura 5.16.

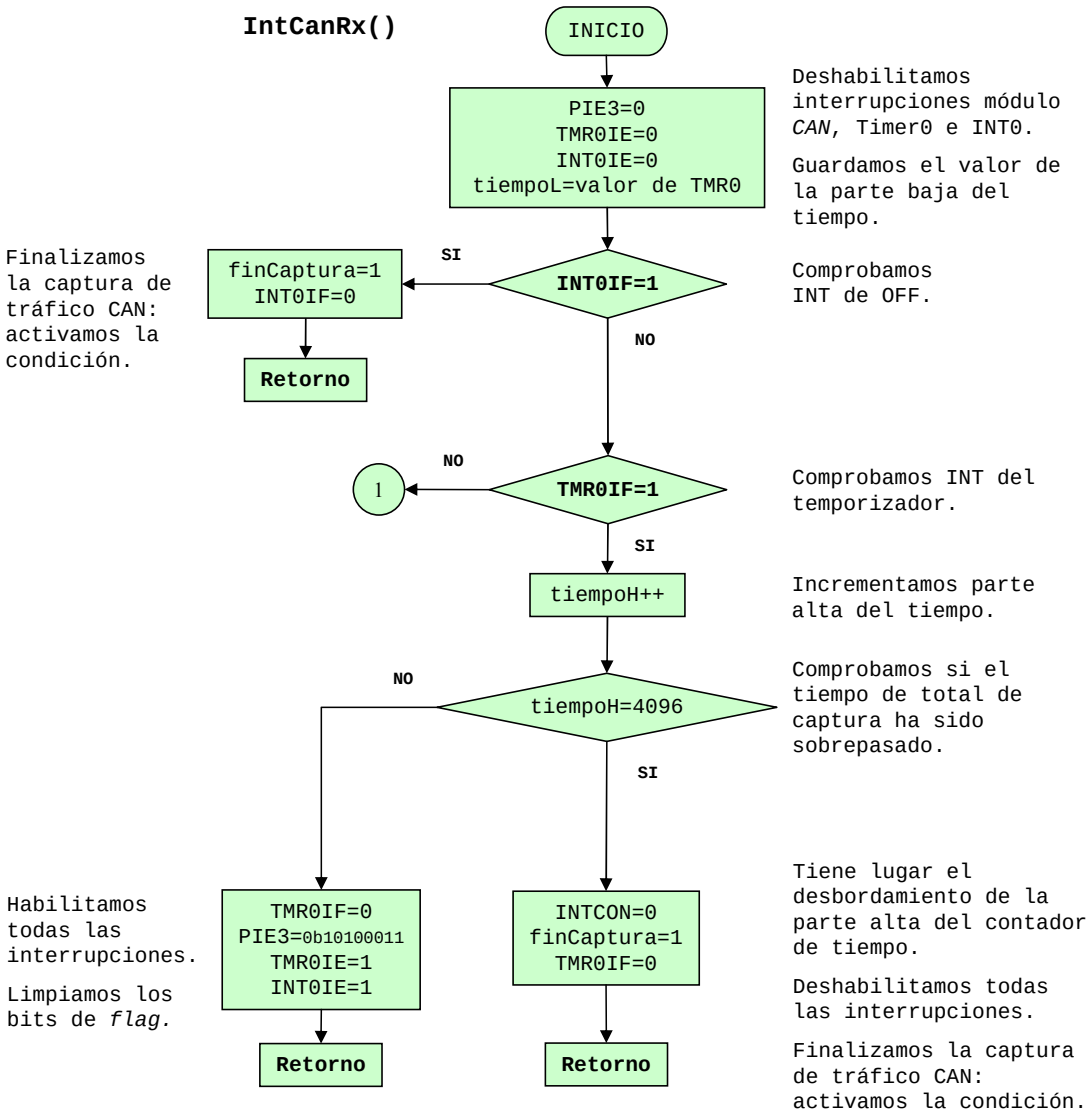


Figura 5.13: Diagrama de flujo de la función IntCanRx (hoja1).

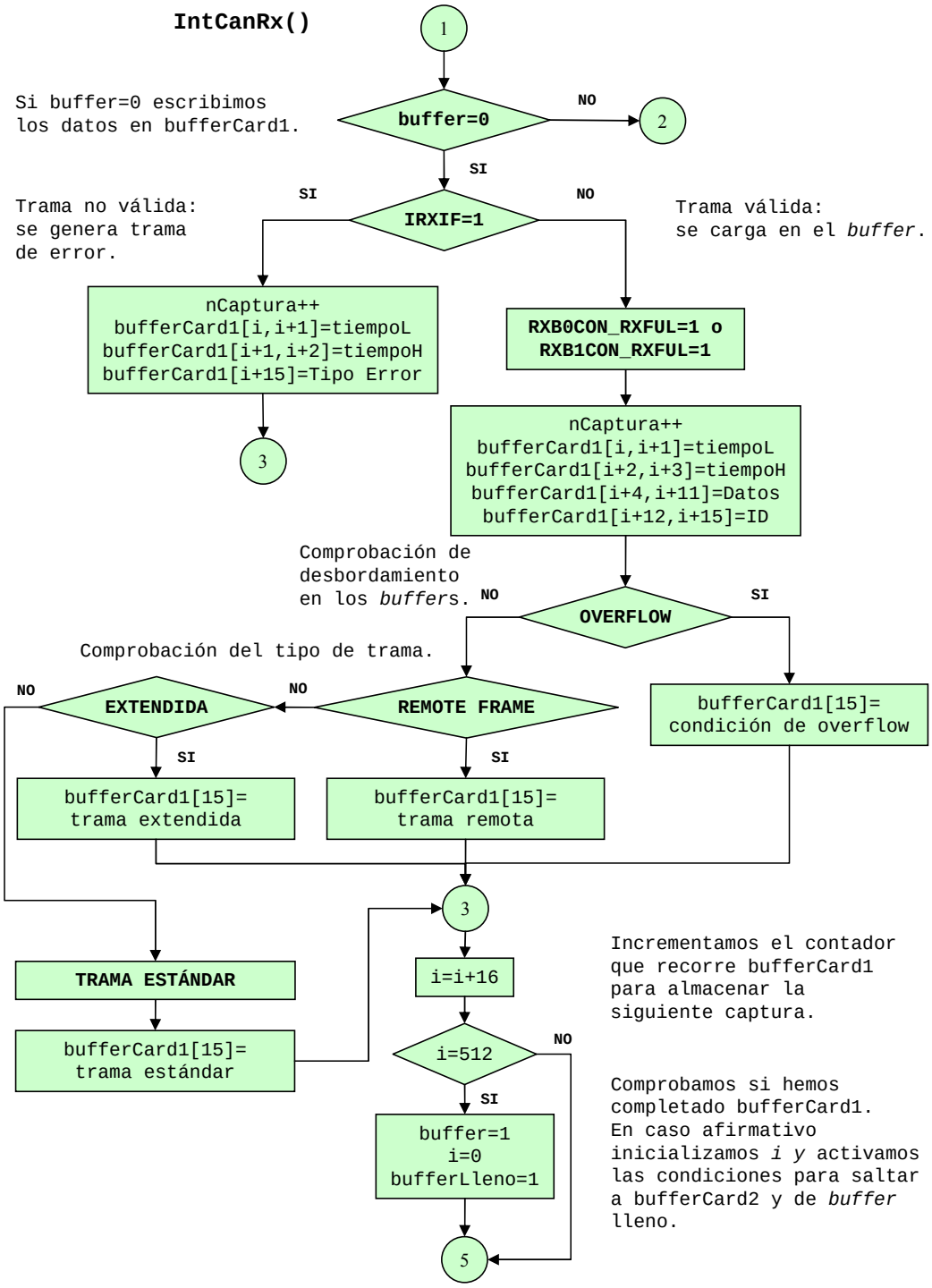


Figura 5.14: Diagrama de flujo de la función IntCanRx (hoja2).

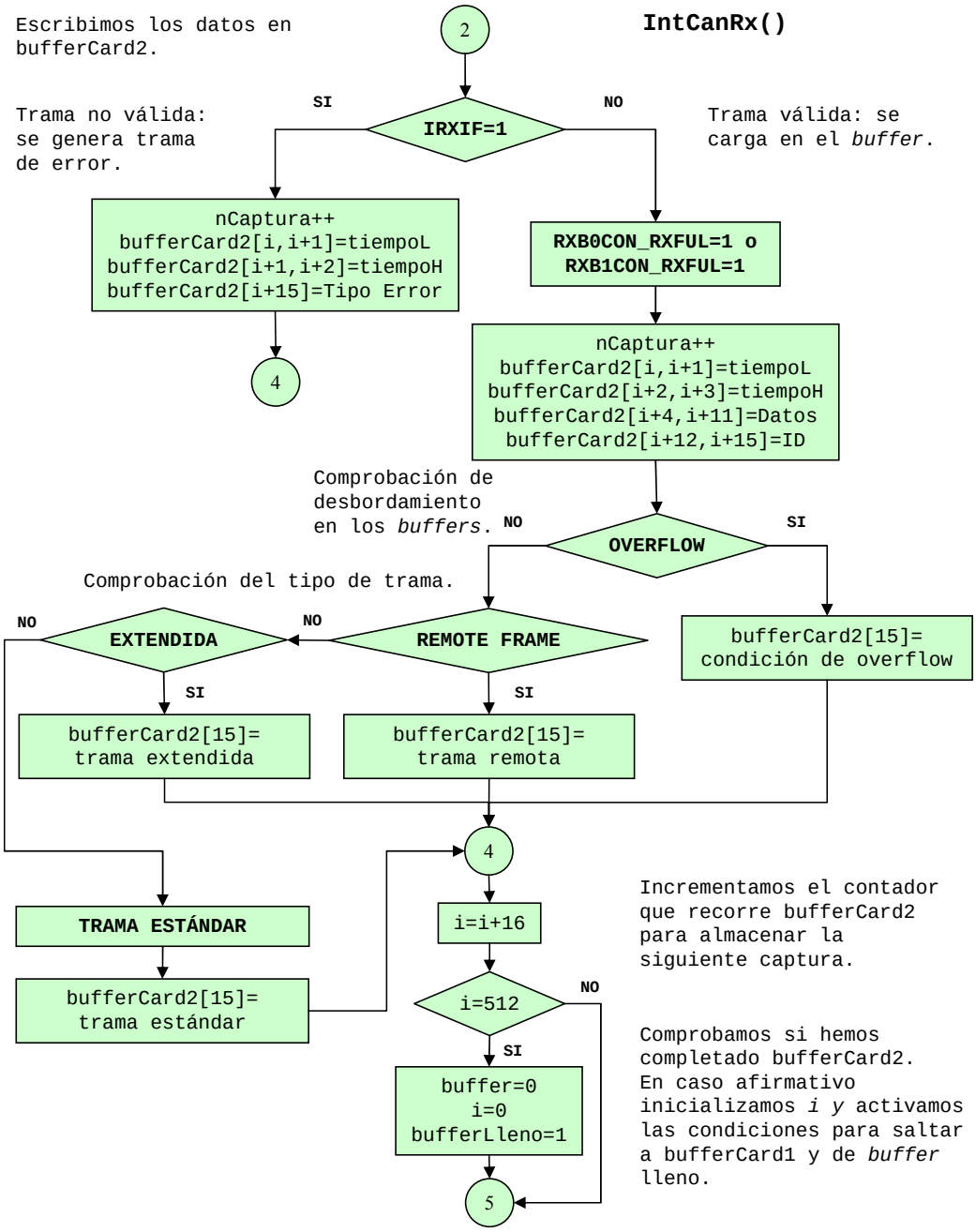


Figura 5.15: Diagrama de flujo de la función IntCanRx (hoja3).

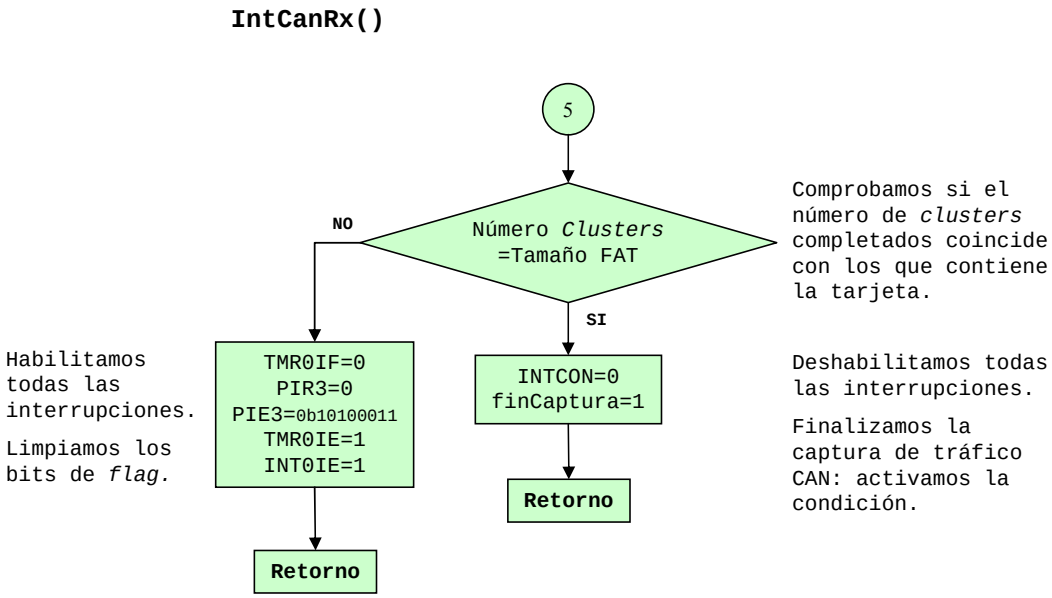


Figura 5.16: Diagrama de flujo de la función IntCanRx (hoja4).

5.3.4 Biblioteca de funciones MMC

Este módulo se encarga de la comunicación entre una tarjeta MMC o SD y el microcontrolador. Este código está desarrollado para la familia 18FXXX, ya que empleamos el puerto SPI implementado en estos microcontroladores[8]. Además es necesario configurar otros pines, la señal de habilitación de la tarjeta (CS) y la señal de alimentación de la tarjeta. Estos dos pines pueden modificarse en el fichero mmc.h. A continuación mostramos la zona de código de este fichero.

Listado 5.1: Fichero mmc.h: configuración de las señales CS y habilitación de alimentación.

```
#define SPI_CSPORTCbits.RC2
#define EN_VCC PORTCbits.RC0
#define TRIS_EN_VCC TRISCbits.TRISC0
#define TRIS_CS TRISCbits.TRISC2
```

Esta bibloteca de funciones debe de usarse con un *buffer* de memoria de 512 bytes. Como el tamaño del *buffer* ocupa más de un banco de memoria del microcontrolador, es necesario modificar el fichero de linkado [21]. Este fichero se puede consultar en el apéndice A.2 (página 156).

Durante el desarrollo de este código hemos probado con distintos tipos de tarjetas y fabricantes (figura 5.17). En las pruebas realizas no hemos tenido problemas con las tarjetas MMC probadas, pero sí hemos notado un comportamiento poco estable con las tarjetas SD.

[8]El módulo SPI emplea ciertos pines del microcontrolador y por lo tanto deberán reservarse.

Figura 5.17: Tarjetas MMC y SD empleadas en el desarrollo del analizador.

En concreto hemos utilizado:

- Tarjeta MMC de 32Mb de tipo OEM (obtenida de un móvil Nokia).
- Tarjeta MMC de 64Mb de SanDisk.
- Tarjeta SD de 128Mb de Kingston.
- Tarjeta SD de 128Mb de Lexar.

El proceso de uso es simple. La primera función llamada será *InitMMC*, que se encargará de configurar el puerto SPI, configurar el resto de pines e inicializar la secuencia de arranque de la tarjeta. A partir de aquí la tarjeta está lista para usarse. Usaremos los siguientes tres comandos estándar: *Lectura512*, *Escritura512*, *Borrar512*.

InitMMC

- **Prototipo:** char InitMMC(void)
- **Descripción:** esta función se encarga de inicializar las pines del microcontrolador concernientes, configurar los parámetros de la comunicación SPI y realizar el proceso de arranque de la tarjeta.
- **Valor devuelto:** devuelve '1' si todo ha ido bien y '0' si se ha producido un error.
- **Diagrama de flujo:** hoja1, figura 5.18; hoja2, figura 5.19.

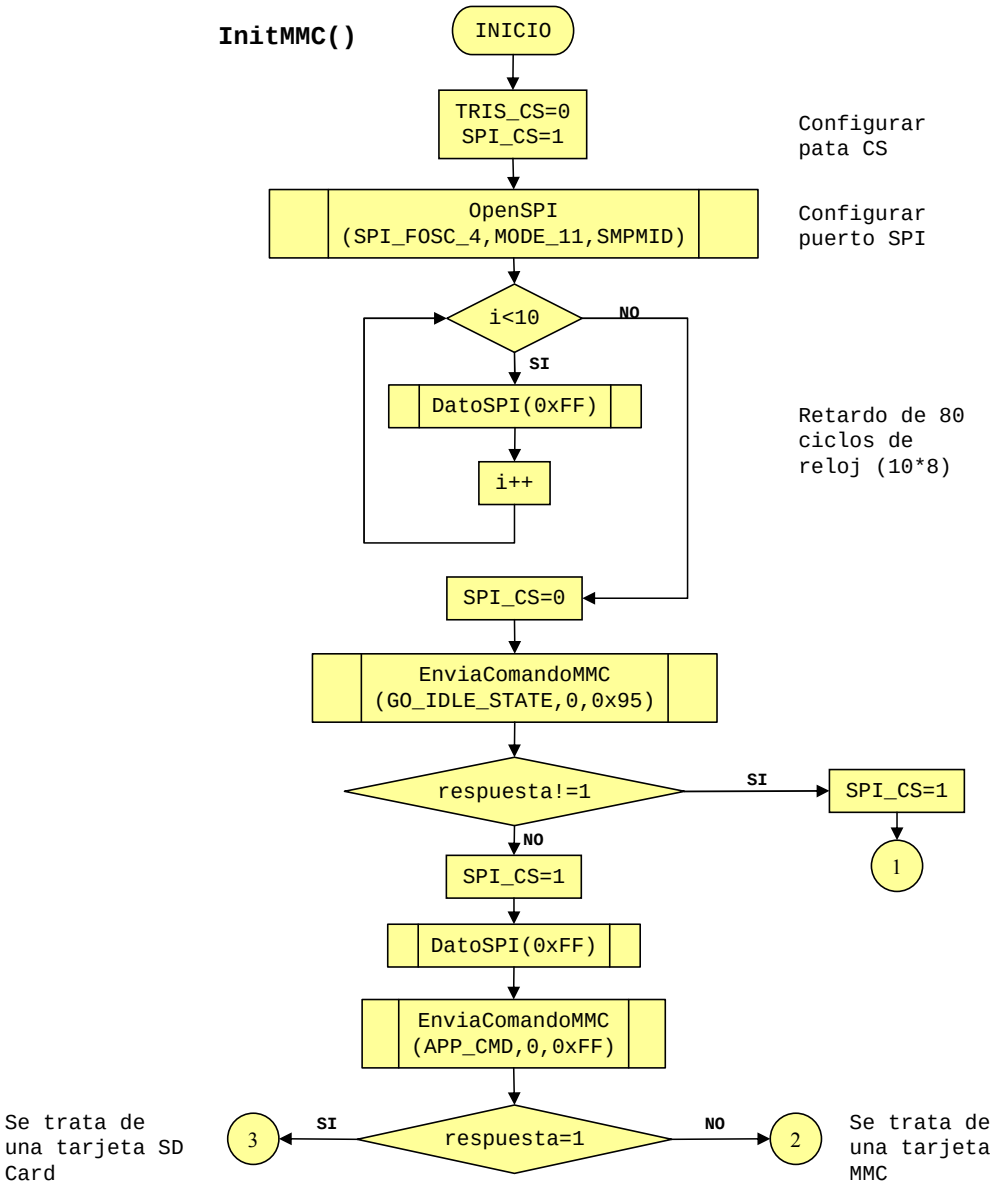


Figura 5.18: Diagrama de flujo de la función InitMMC (hoja1).

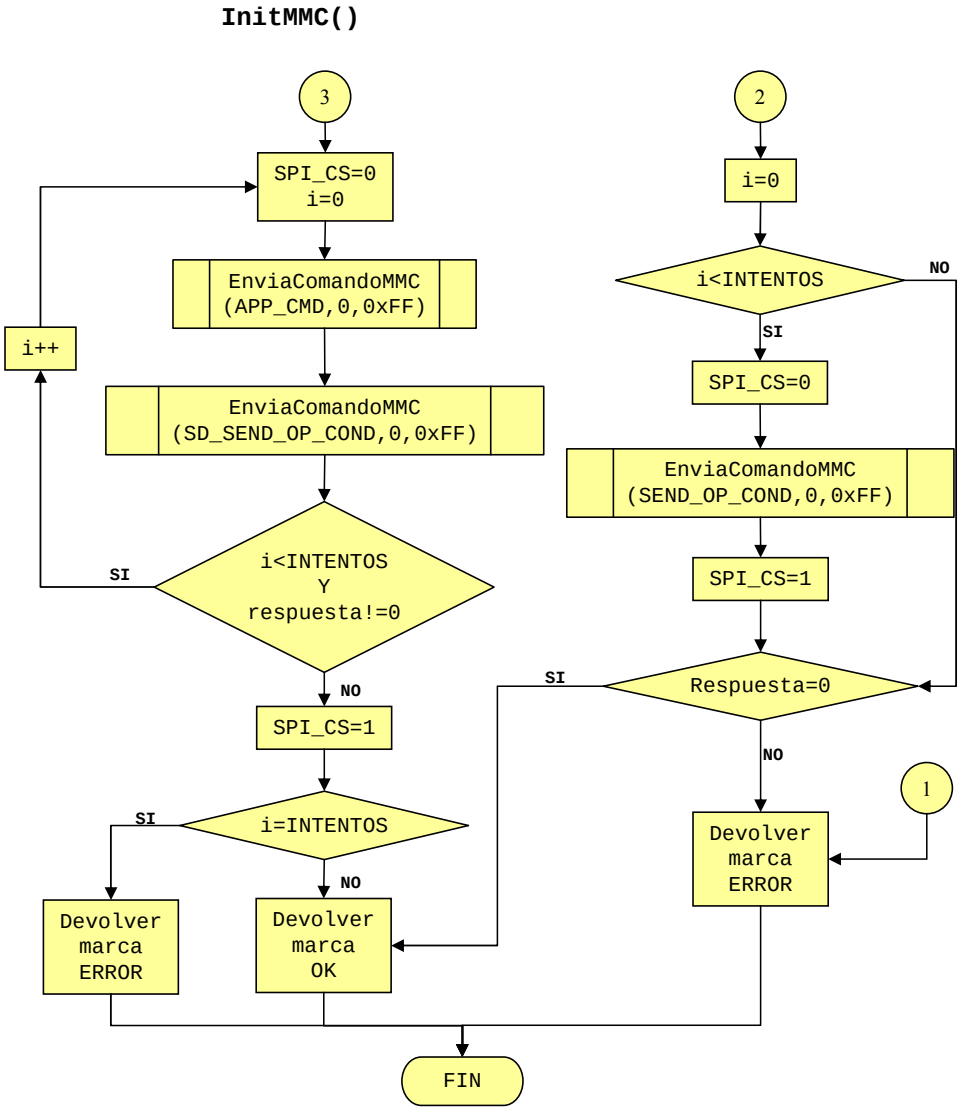


Figura 5.19: Diagrama de flujo de la función InitMMC (hoja2).

CardPowerOn & CardPowerOff

- **Prototipo:** void CardPowerOn(void)
- **Descripción:** esta función coloca a nivel alto el pin EN_VCC e introduce un retardo de espera.
- **Prototipo:** void CardPowerOff(void)
- **Descripción:** esta función coloca a nivel bajo el pin EN_VCC.
- **Diagramas de flujos:** figura 5.20.

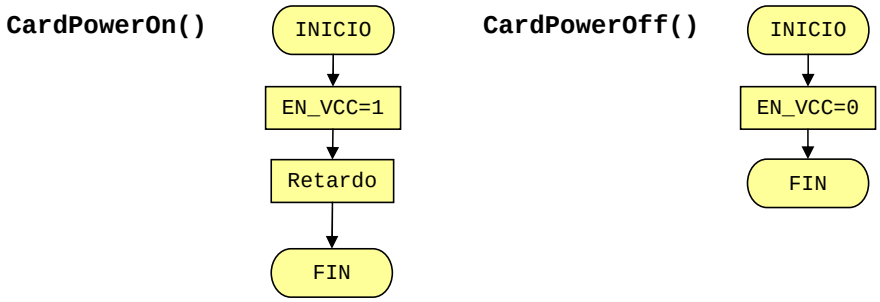


Figura 5.20: Diagrama de flujo de las funciones CardPowerOn y CardPowerOff.

DatoSPI

- **Prototipo:** char DatoSPI(char)
- **Descripción:** esta función envía un dato por el puerto SPI.
- **Valor devuelto:** el dato recibido en *Data In*.
- **Diagrama de flujo:** figura 5.21.

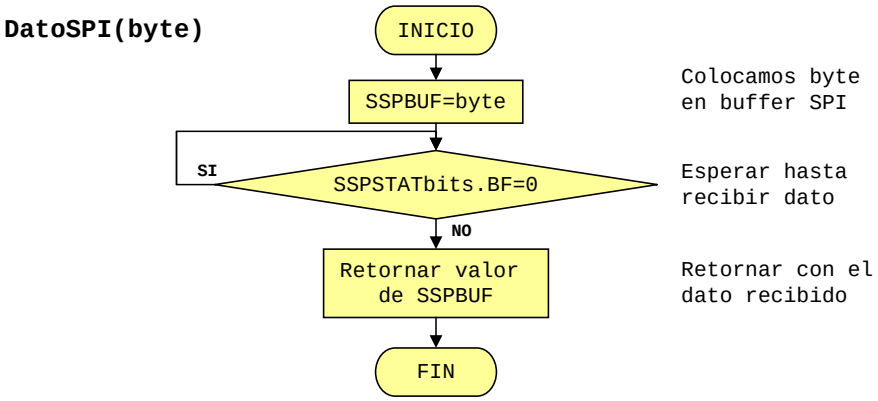


Figura 5.21: Diagrama de flujo de la función DatoSPI.

EnviaComandoMMC

- **Prototipo:** char EnviaComandoMMC(char cmd, ulong adr, char crc)

- **Descripción:** esta función envía una comando especificado por el parámetro *cmd*. El argumento del comando es el argumento *adr*, que suele ser una dirección de memoria de la tarjeta en los comandos de lectura y escritura. El último argumento es el código de corrección de errores. Por lo general, este campo siempre será 0xFF ya que en el modo SPI la comprobación CRC está deshabilitada por defecto.

- **Valor devuelto:** devuelve una respuesta de tipo R1.

- **Diagrama de flujo:** figura 5.22.

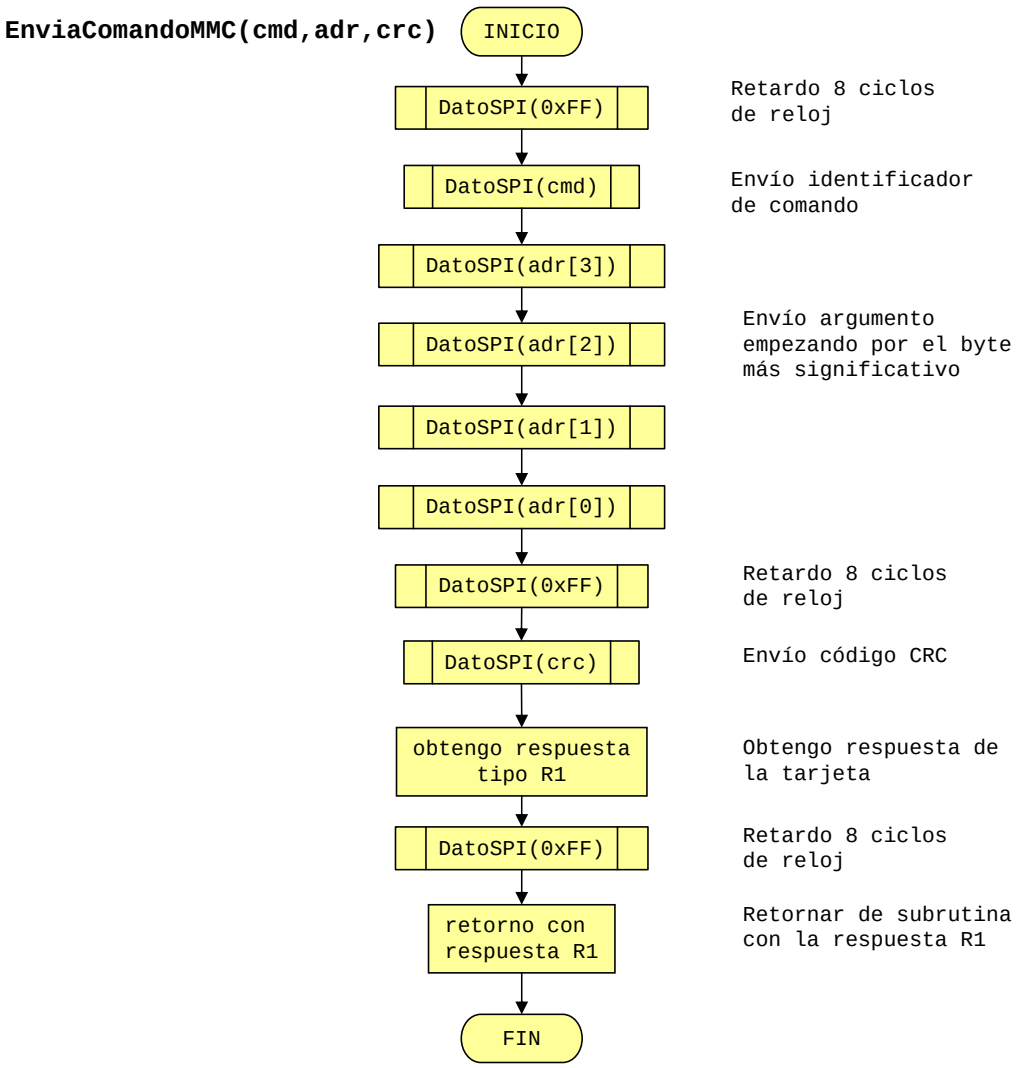


Figura 5.22: Diagrama de flujo de la función EnviaComandoMMC.

Lectura512

- **Prototipo:** char Lectura512(char∗ buffer, ulong adr)
- **Descripción:** esta función lee un bloque de 512 bytes especificados por la dirección *adr* y lo almacena en el *buffer*. Si se produce un error al mandar el comando de lectura reintenta la comunicación n-veces. Este valor, llamado INTENTOS, está definido en el fichero mmc.h.
- **Valor devuelto:** devuelve '1' si todo ha ido bien y '0' si se ha producido un error.
- **Diagrama de flujo:** figura 5.23.

Escritura512

- **Prototipo:** char Escritura512(char∗ buffer, ulong adr)
- **Descripción:** esta función escribe en la tarjeta un bloque de 512 bytes especificados por la dirección *adr*. Si se produce un error al mandar el comando de lectura reintenta la comunicación n-veces. Este valor, llamado INTENTOS, está definido en el fichero mmc.h.
- **Valor devuelto:** devuelve '1' si todo ha ido bien y '0' si se ha producido un error.
- **Diagrama de flujo:** figura 5.24.

Borrar512

- **Prototipo:** char Borrar512(ulong adrInicio, ulong adrFin)
- **Descripción:** esta función borra los bloques de memoria comprendidos entre *adrInicio* y *adrFin*.
- **Valor devuelto:** devuelve '1' si todo ha ido bien y '0' si se ha producido un error.
- **Diagrama de flujo:** figura 5.25.

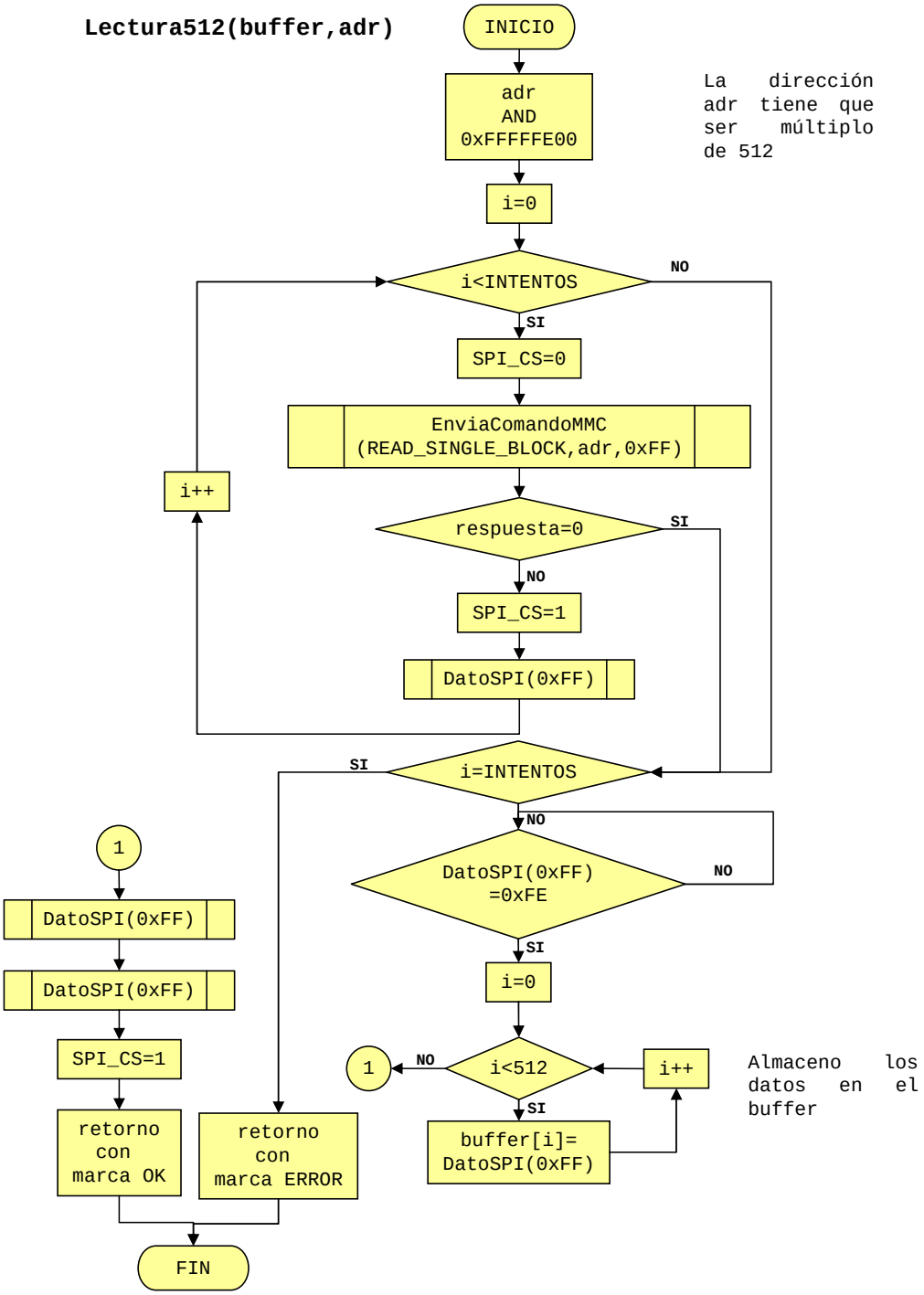


Figura 5.23: Diagrama de flujo de la función Lectura512.

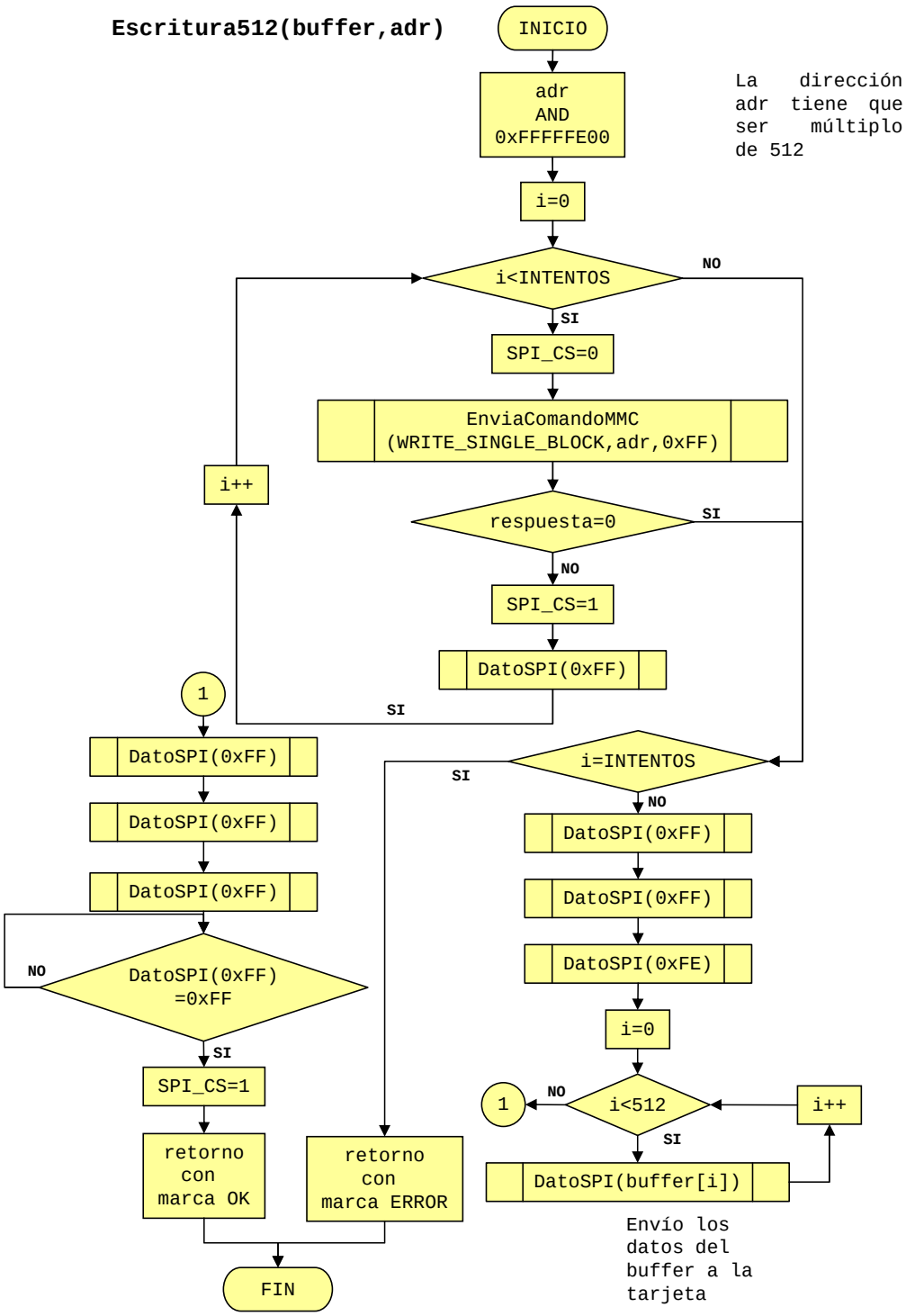


Figura 5.24: Diagrama de flujo de la función Escritura512.

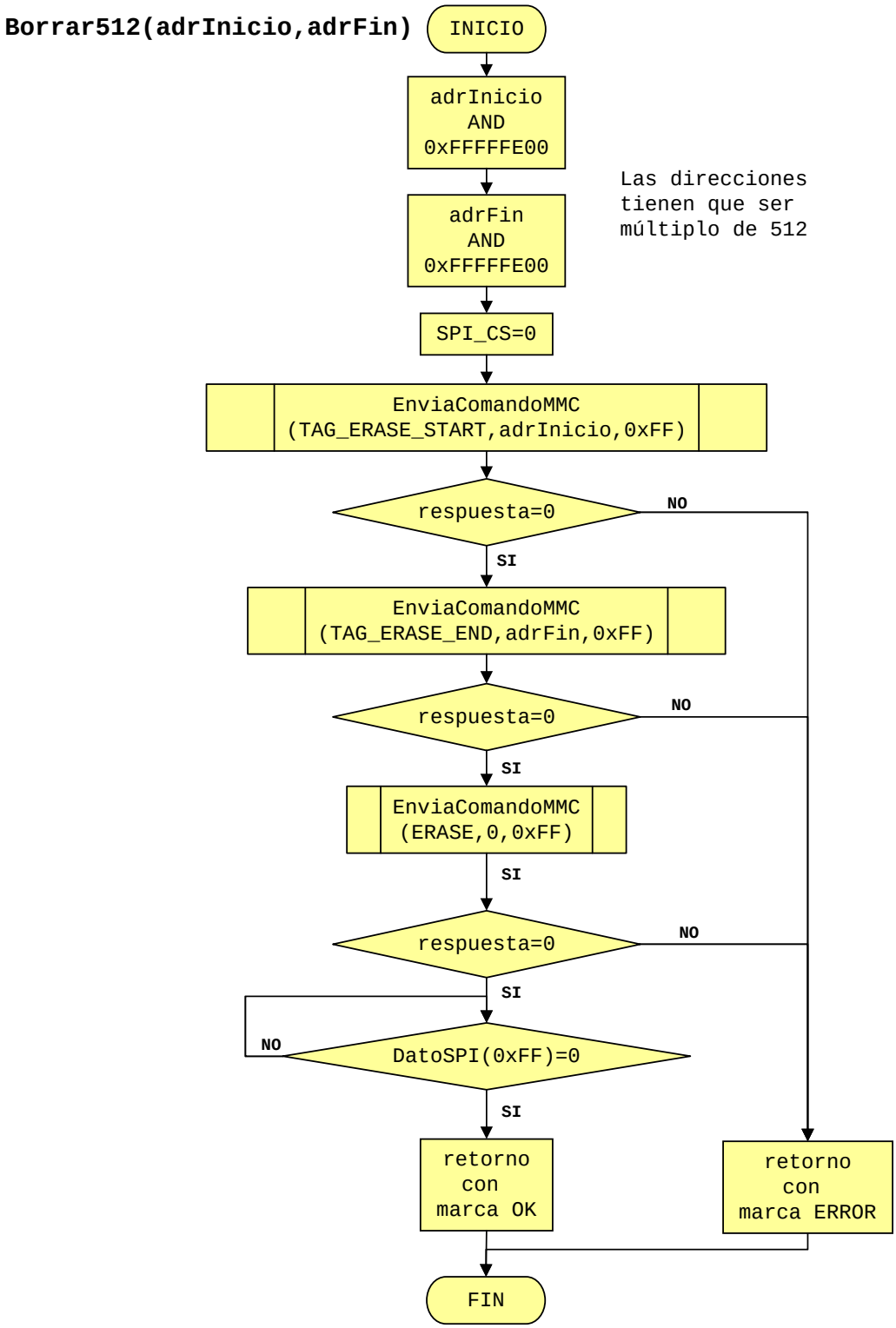


Figura 5.25: Diagrama de flujo de la función Borrar512.

5.3.5 Biblioteca de funciones FAT

Estructuras de datos empleadas

Para el manejo de esta librería existen dos estructuras de datos definidas en el fichero fat.h.

Listado 5.2: Fichero fat.h, estructuras FAT.

```
struct structFAT{
  ulong tablaFAT;        // direccion que apunta a la FAT
  uint  sectoresFAT;     // numero de sectores por FAT
  uint  entradasRaiz;    //
  ulong datos;           // direccion que apunta a la zona de datos
                         // esta zona empieza con el segundo cluster
  ulong dirRaiz;         // direccion al directorio raiz
  char  sectorCluster;   // numero de sectores por cluster
};

struct structFichero{
    char  nombre[11];      // nombre del fichero.
    uint  clusterInicio;   // primer cluster del fichero.
    uint  numCluster;      // numero de cluster que tiene el fichero
    uchar sectCluster;     // ultimo sector por cluster escrito
    ulong tamanio;         // tamanio del fichero en bytes
};
```

La primera de ellas, `structFAT`, se encarga de agrupar todos los datos concernientes a la FAT, como posición de la tabla FAT, del directorio raíz, etc. La segunda estructura, `structFichero`, agrupa toda la información acerca del fichero como es el caso del nombre, *cluster* de inicio, tamaño, etc. Estas estructuras están declaradas en el fichero fat.c y deben de ser declaradas como externas en los posibles ficheros a usar.

LeeInt

- **Prototipo:** uint LeeInt(char * buffer , char dir)
- **Descripción:** devuelve el entero que se encuentra en la posición *dir* del *buffer*.
- **Valor devuelto:** el entero apuntado por *dir*.
- **Diagrama de flujo:** figura 5.26.

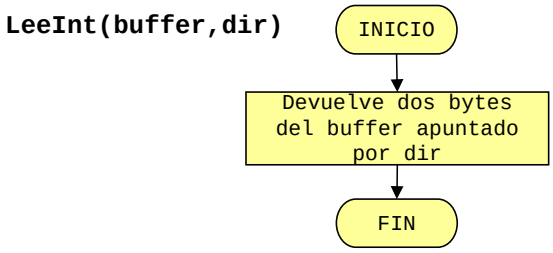


Figura 5.26: Diagrama de flujo de la función LeeInt.

InitFAT

- **Prototipo:** void InitFAT(char *buffer)
- **Descripción:** Esta función se encarga de inicializar la estructura `structFAT` con los valores pertinentes. Esta función debe de ser llamada antes de poder usar cualquier fichero. Se apoya en un *buffer* de 512 bytes para poder realizar las operaciones de lectura.
- **Diagrama de flujo:** figura 5.27.

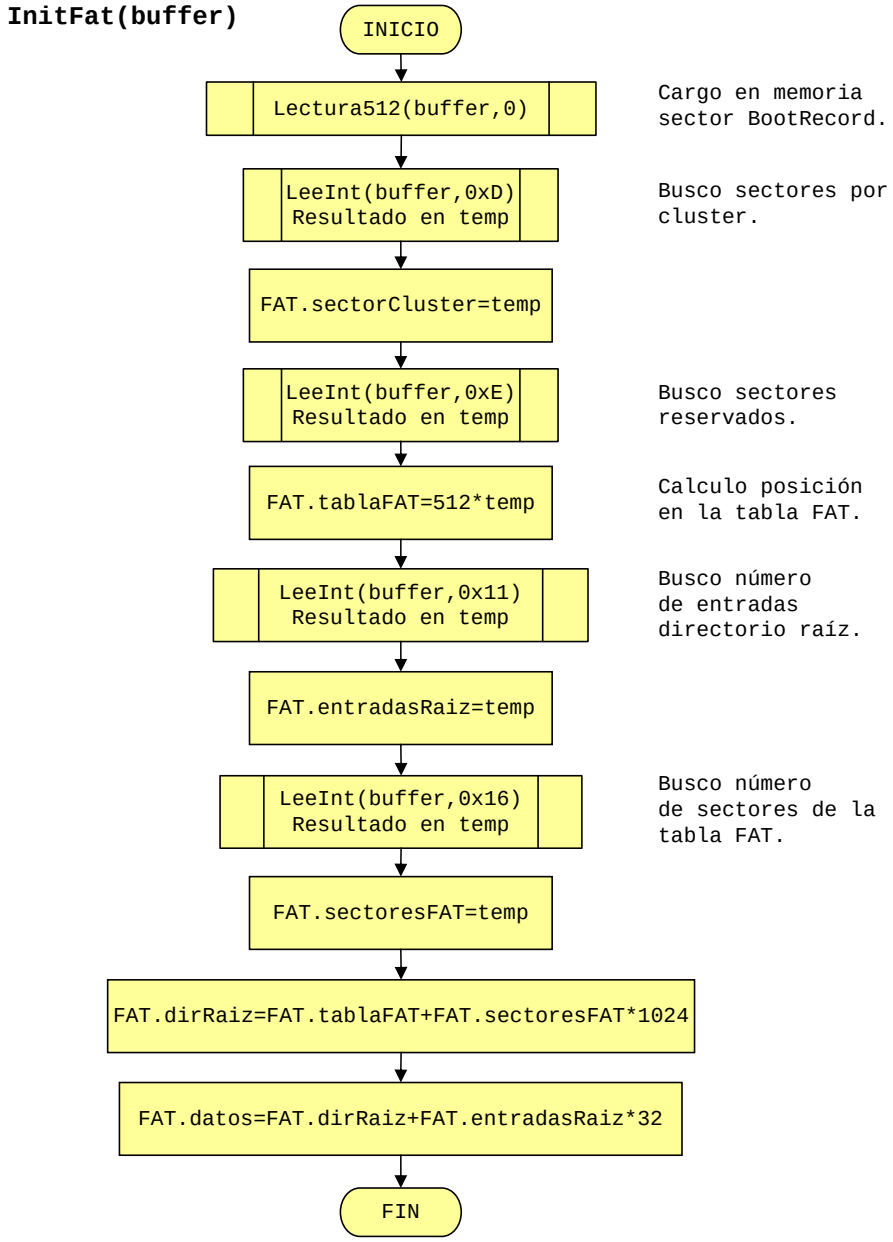


Figura 5.27: Diagrama de flujo de la función InitFat.

AbrirFicheroLectura

- **Prototipo:** uint AbrirFicheroLectura(char∗ nombre, char∗ buffer)
- **Descripción:** Busca el nombre de un fichero dentro de la estructura de directorio raíz.
- **Valor devuelto:** El *cluster* de inicio del fichero, en caso de no encontrar el fichero devuelve la marca NO_FIND.
- **Diagrama de flujo:** figura 5.28.

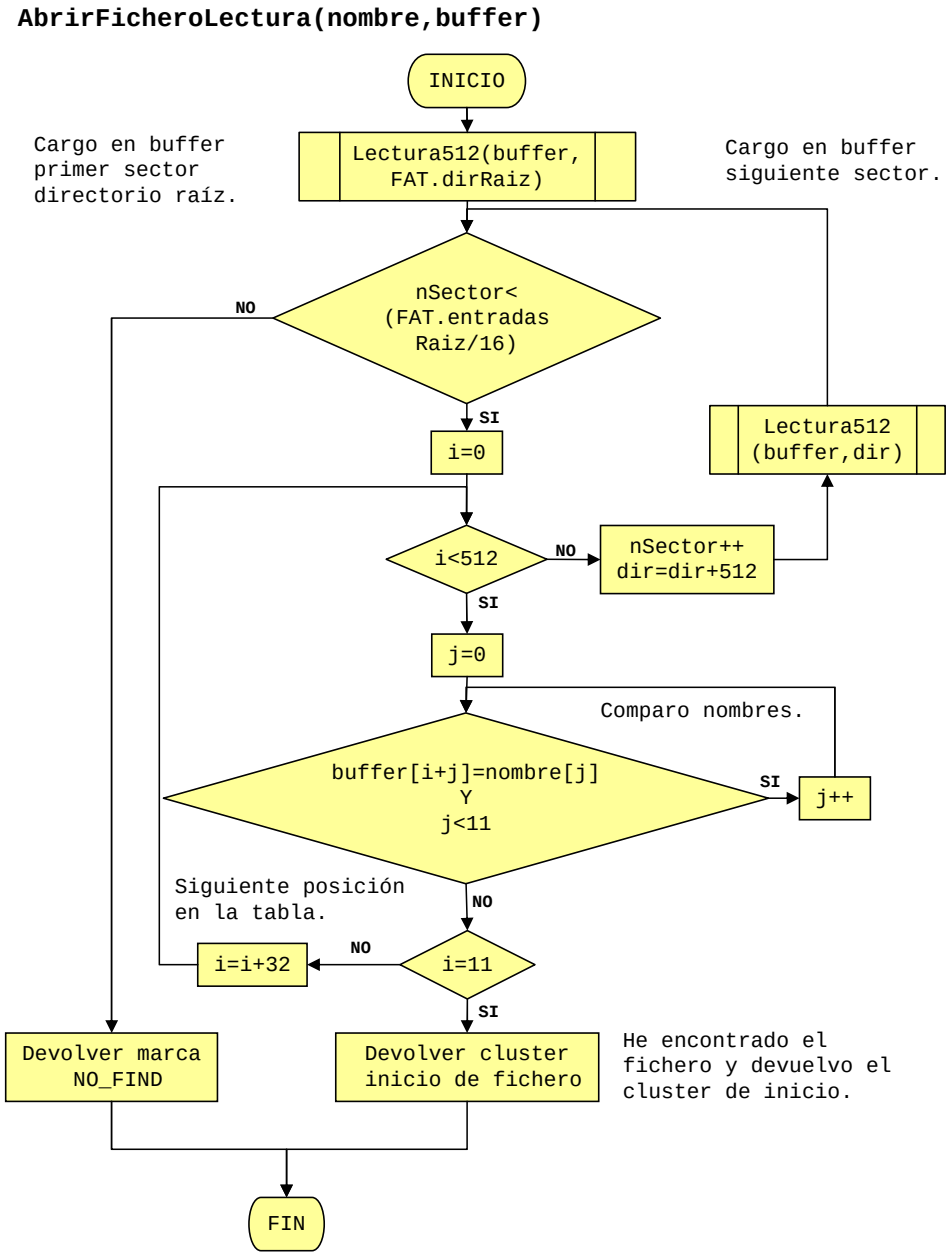


Figura 5.28: Diagrama de flujo de la función AbrirFicheroLectura.

LeerFichero

- **Prototipo:** uint LeerFichero(char* buffer, uint clusterInicio , char sector)
- **Descripción:** Esta función escribe los datos apuntados por *clusterInicio* y *sector* en el *buffer*.
- **Valor devuelto:** El *cluster* de inicio del fichero, en caso de no encontrar el fichero devuelve la marca NO_FIND.
- **Diagrama de flujo:** figura 5.29.

EscribirFichero

- **Prototipo:** uint EscribirFichero(char* buffer , uint cluster , char sector)
- **Descripción:** Esta función escribe los datos del *buffer* en la zona apuntada por *clusterInicio* y *sector*.
- **Valor devuelto:** El *cluster* del siguiente sector a escribir.
- **Diagrama de flujo:** figura 5.30.

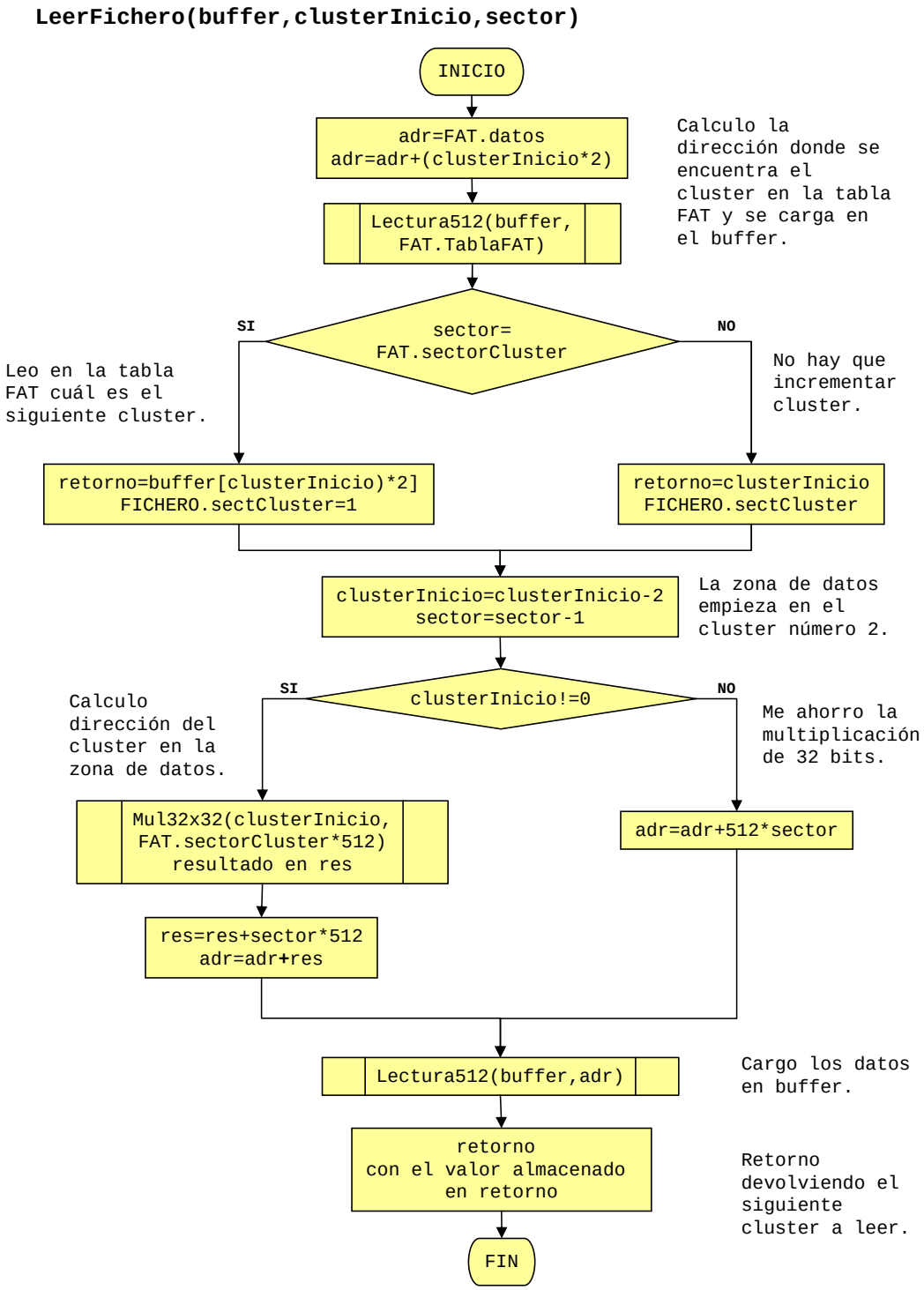


Figura 5.29: Diagrama de flujo de la función LeerFichero.

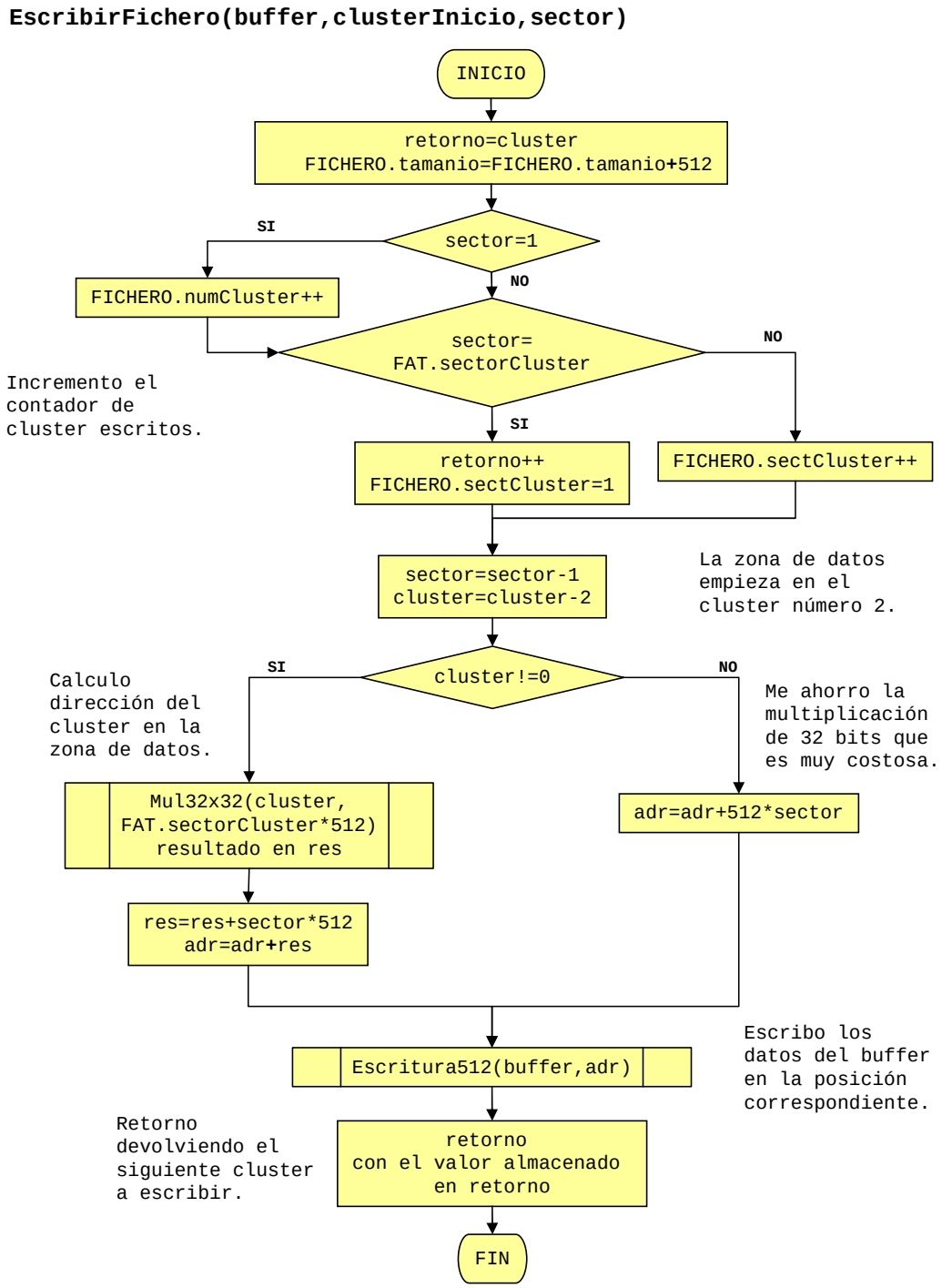


Figura 5.30: Diagrama de flujo de la función EscribirFichero.

CerrarFicheroEscritura

- **Prototipo:** void CerrarFicheroEscritura(char *buffer,char tipo)
- **Descripción:** Esta función actualiza la tabla FAT y la estructura del directorio raíz para cerrar un fichero. Esta función está pensada para el caso en que el fichero se encuentre en *cluster* consecutivos. En *tipo* se especifica si el fichero es FICHERO_NUEVO o FICHERO_AGREGAR.
- **Diagrama de flujo:** hoja1, figura 5.31; hoja2, figura 5.32.

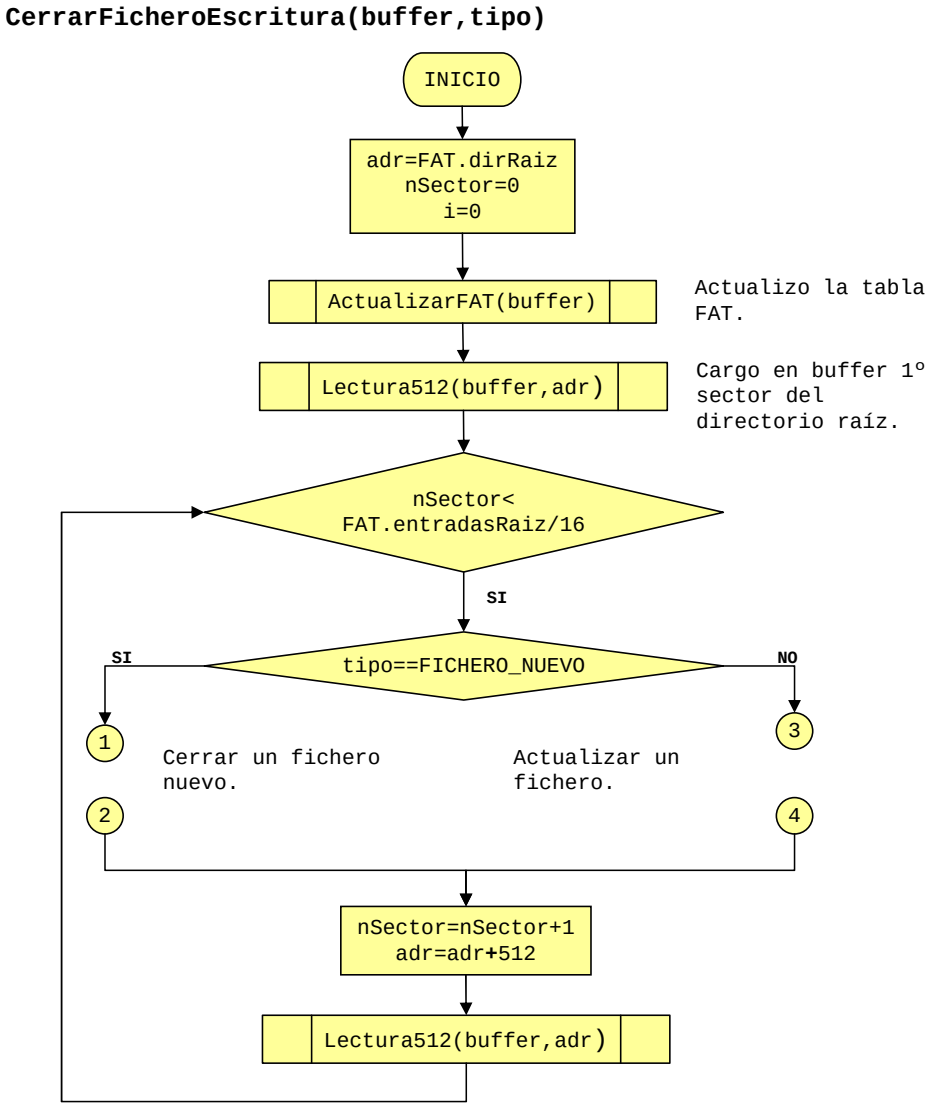


Figura 5.31: Diagrama de flujo de la función CerrarFicheroEscritura (hoja1).

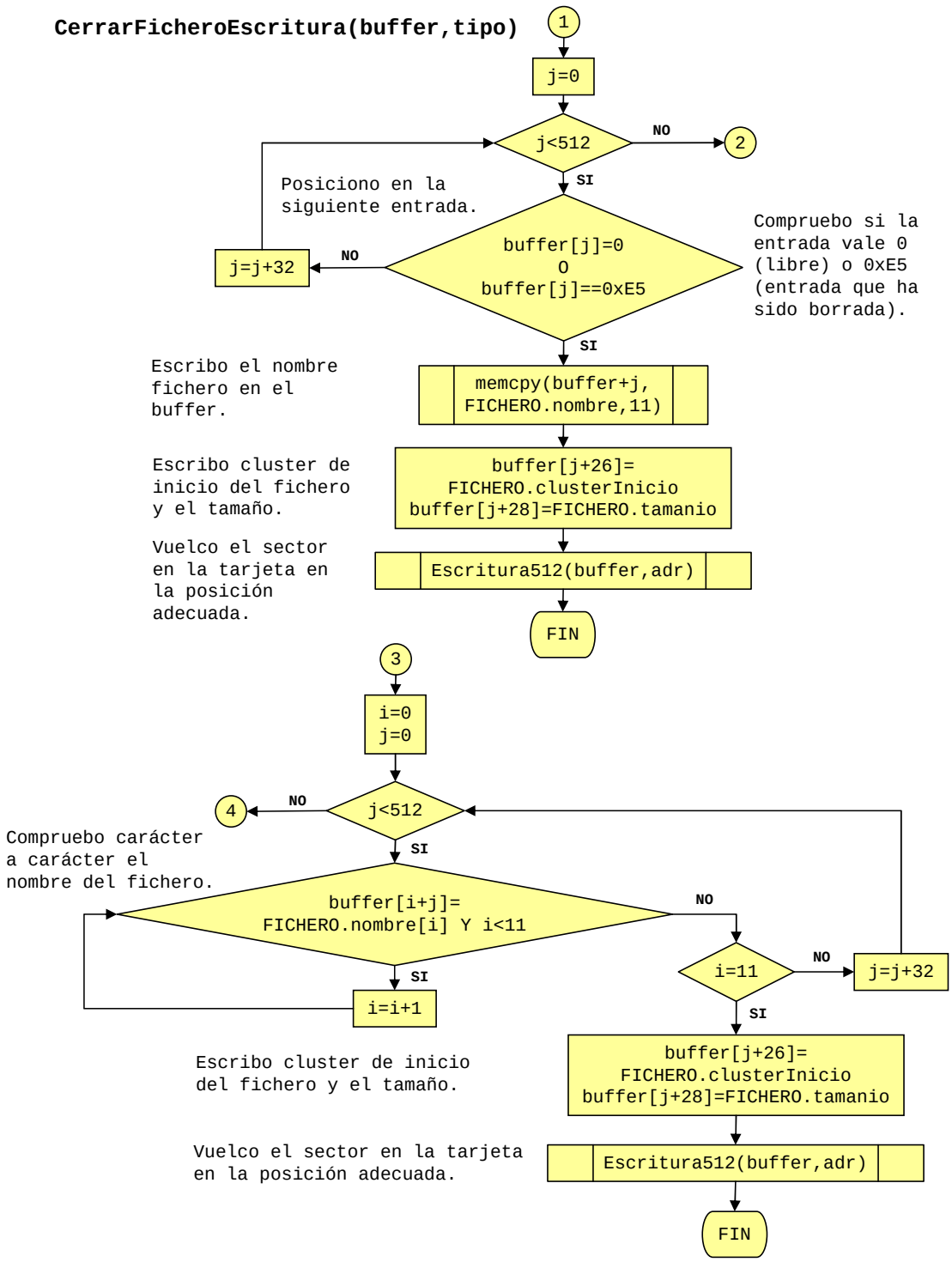


Figura 5.32: Diagrama de flujo de la función CerrarFicheroEscritura (hoja2).

ActualizarFAT

- **Prototipo:** void ActualizarFAT(char* buffer)
- **Descripción:** Actualiza la tabla FAT del archivo. Esta función es llamada por CerrarFicheroEscritura.
- **Diagrama de flujo:** figura 5.33.

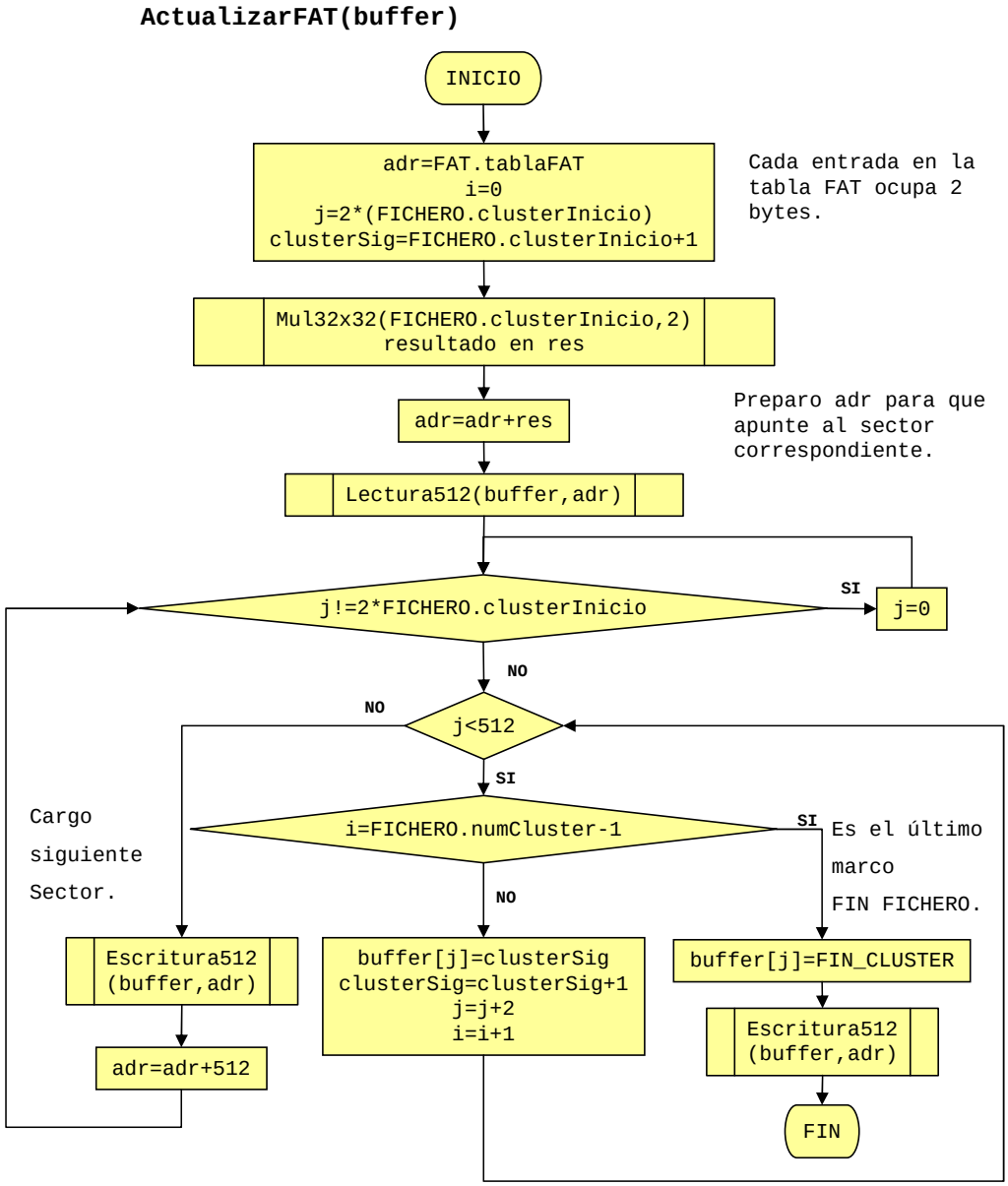


Figura 5.33: Diagrama de flujo de la función ActualizarFAT.

5.3.6 Otras bibliotecas de funciones

El compilador C18, destinado a la familia de microcontroladores 18, posee una serie de bibliotecas de funciones para poder configurar todos los módulos del microcontrolador. Como es lógico, han sido empleadas en el diseño del código fuente. En el caso de que surgiera alguna duda a la hora de interpretar el código fuente del analizador, recomendamos la consulta del manual de referencia del compilador [40].

Esta librería posee un módulo para manejar LCD compatibles con el controlador de LCD Hitachi HD44780. Para poder conectar dicho LCD a cualquier pin de nuestro microcontrolador, hemos tenido que modificar el fichero fuente xlcd.h (ver listado de código fuente 5.3).

Para programar el controlador *CAN* interno que posee el microcontrolador, nos hemos apoyado en la nota de aplicación [22]. Esta nota de aplicación propone un código fuente para simplificar la programación del controlador *CAN*[9].

Listado 5.3: Parte del fichero xlcd.h modificado.

```
/* CODIGO FUENTE MODIFICADO*/
#ifndef __XLCD_H
#define __XLCD_H

/* PIC 17Cxxx and 18Cxxx XLCD peripheral routines.
 *
 *   Notes:
 *      - These libraries routines are written to support the
 *        Hitachi HD44780 LCD controller.
 *      - The user must define the following items:
 *          - The LCD interface type (4- or 8-bits)
 *          - If 4-bit mode
 *              - whether using the upper or lower nibble
 *          - The data port
 *              - The tris register for data port
 *              - The control signal ports and pins
 *              - The control signal port tris and pins
 *          - The user must provide three delay routines:
 *              - DelayFor18TCY() provides a 18 Tcy delay
 *              - DelayPORXLCD() provides at least 15ms delay
 *              - DelayXLCD() provides at least 5ms delay
 */

/* Interface type 8-bit or 4-bit
 * For 8-bit operation uncomment the #define BIT8
 */
/* #define BIT8 */

/* When in 4-bit interface define if the data is in the upper
 * or lower nibble.  For lower nibble, comment the #define UPPER
```

[9]Dicho código fuente corresponde a los fichero can18xx8.c y can18xx8.h.

```
 */
#define UPPER

/* DATA_PORT defines the port to which the LCD data lines are
connected */
#if    __18CXX
#define DATA_PORT      PORTB
#define TRIS_DATA_PORT TRISB
#else /* 17CXX */
#define DATA_PORT      PORTC
#define TRIS_DATA_PORT DDRC
#endif

/* CTRL_PORT defines the port where the control lines are connected.
 * These are just samples, change to match your application.
 */
#if    __18CXX
#define RW_PIN   PORTAbits.RA2   /* Port for RW */
#define TRIS_RW  DDRAbits.RA2    /* TRIS for RW */
#define RS_PIN   PORTAbits.RA3   /* Port for RS */
#define TRIS_RS  DDRAbits.RA3    /* TRIS for RS */
#define E_PIN    PORTAbits.RA1   /* PORT for E  */
#define TRIS_E   DDRAbits.RA1    /* TRIS for E  */
#else /* 17CXX */
#define RW_PIN   PORTCbits.RC5   /* Port for RW */
#define TRIS_RW  DDRCbits.RC5    /* TRIS for RW */
#define RS_PIN   PORTCbits.RC4   /* Port for RS */
#define TRIS_RS  DDRCbits.RC4    /* TRIS for RS */
#define E_PIN    PORTCbits.RC6   /* PORT for E  */
#define TRIS_E   DDRCbits.RC6    /* TRIS for E  */
#endif
```

5.4 Guía de usuario

5.4.1 Especificaciones funcionales

Características generales

- Dispositivo transportable con autonomía propia.
- Reloj de operación de 20 Mhz.
- Admite tarjetas de memoria *Multimedia Card* y *Secure Digital*, con estructura de ficheros FAT16 [10].
- Tiempo total máximo de captura (independiente del tamaño de la tarjeta de memoria): 229 minutos[11,12].
- Rango de temperaturas de operación: -40 ºC - +85 ºC.
- Detección de nivel bajo de alimentación.

Módulo CAN

- 2 *buffers* de recepción de mensajes.
- Cumple con el estándar *CAN*2.0B ACTIVE [4]:
  - Admite campo de Identificador Estándar (11 bits) y Extendido (29 bits).
  - Longitud máxima del campo de datos de 8 bytes.
  - Establecimiento de prioridades para los dos *buffers* de recepción.
  - 2 máscaras de aceptación de tramas (una para cada *buffer* de recepción).
  - 6 filtros de aceptación de tramas (2 asociadas al *buffer* de alta prioridad y 4 al de baja).
  - Posibilidad de configuración de doble *buffer*, para minimizar problemas de sobrecarga (*overflow*).
  - Tratamiento de errores en la transmisión de mensajes.
- Puede ser integrado en una red *CAN* de hasta 112 nodos.
- Soporta hasta velocidades de 1Mbit/s.
- Para velocidades de 1Mbit/s no soporta tráfico continuo para un tiempo superior a 10 ms, por sobrecarga en los *buffers* de recepción.

[10] Necesidad de formatear las tarjetas en FAT16.

[11] Tiempo máximo exacto de captura: 13.743,895 segundos.

[12] Este límite está impuesto por el número de bits destinados a almacenar el instante de tiempo de cada captura. Para las tarjetas existentes en el mercado, nuestro límite va a estar en el tamaño de las mismas, y no en este factor.

- Tiene capacidad para detectar 2 tramas consecutivas en un intervalo mínimo de tiempo de 200 μs.
- Implementa para la capa física el estándar ISO 11898.

Características eléctricas

- Alimentación: pila de 9 V.
- Rango de tensiones de alimentación: 11.5 V (máxima) - 4.5 V (mínima).
- LED de advertencia de alimentación baja.
- Protección de alimentación de la tarjeta de memoria.
- Consumo máximo: 40 mA[13].

5.4.2 Configuración del tiempo de bit

Como ya hemos señalado en capítulos anteriores, capítulos 2.2.1 (página 36) y 3.4.3 (página 55), el tiempo de bit debe ser común para todos los nodos que integran la red *CAN*. La introducción del analizador en dicha red por tanto ha de cumplir el mismo requisito. El tiempo de bit se divide en los periodos mínimos de tiempo T_q, que pueden variar en número entre 8 y 25. Por otra parte el tiempo de bit está formado por cuatro segmentos: de sincronización, tiempo de propagación, segmento de *buffer* 1 y segmento de *buffer* 2, formados por unidades de T_q.

Para el cálculo del tiempo de bit debemos manejar dos variables:

- El valor del cuanto de tiempo T_q, que depende del periodo del oscilador y de un factor de preescala, BRP:

$$T_q(\mu s) = \frac{2 \times (BRP + 1)}{F_{OSC}(MHz)} \tag{5.2}$$

- El número de total de T_q que conforman el tiempo de bit, repartidos entre los cuatro segmentos comentados:

$$Tiempo_{bit} = T_q \times (Segm_{Sincron.} + Segm_{Prop.} + Segm_{Fase1} + Segm_{Fase2}) \tag{5.3}$$

$$número\, T_q = \frac{1000}{Tasa_{tx}(kbit/s) \times T_q(\mu s)} \quad (8 \leq número\ T_q \leq 25) \tag{5.4}$$

El usuario debe conocer previamente la tasa de transmisión de la red *CAN* que desea analizar. En función del valor del periodo de oscilación del reloj con el que funciona el microcontrolador, debe encontrar un valor de BRP, que nos da el tiempo que dura un cuanto T_q, y un número total de cuantos, tales que aplicando las fórmulas obtengamos el mismo tiempo de bit.

[13]El consumo depende de la intensidad de tráfico capturado.

A continuación hay que repartir el número total de cuantos seleccionado entre los cuatro segmentos mencionados de la forma que se prefiera, siempre y cuando se verifiquen las siguientes dos condiciones:

$$Segm_{Prop.} + Segm_{Fase1} \geq Segm_{Fase2} \tag{5.5}$$

$$Segm_{Fase2} > SJW \tag{5.6}$$

SJW es el parámetro de salto de sincronización, ver sección 2.2.1 (página 36), cuyo valor puede variar entre 1 T_q y 4 T_q. Este último valor es recomendable cuando la señal de reloj de los distintos nodos es inestable o poco precisa. En caso contrario, un SJW de 1 T_q es suficiente (*datasheet*, página 235 [18]).

5.4.3 Modos de operación

Existen dos modos de operación, sección 3.4.2 (página 53), en los que puede trabajar el analizador durante la captura de tráfico *CAN*:

- **Modo normal:** las tramas no válidas son detectadas por el módulo *CAN*, pero no se cargan en los *buffers* de recepción. En este modo podemos seleccionar el tipo de tramas, según el tamaño de su identificador, que queremos capturar: sólamente tramas estándar, únicamente tramas extendidas, o ambos tipos simultáneamente.

- **Modo de escucha:** además de ser detectadas, las tramas no válidas son almacenadas en alguno de los dos *buffers* de recepción. El sistema de máscaras y filtros para discriminar mensajes está desactivado, siendo detectado todo el tráfico *CAN* del sistema. Por último, señalamos que en este modo no tiene lugar el envío del bit de ACK, por lo que el modo de escucha sólo es aplicable cuando la red *CAN* al menos consta de 2 nodos[14].

El modo de escucha es en principio el más idóneo para analizar un sistema *CAN*, ya que es invisible a la red al no enviar el módulo *CAN* el bit de ACK ni generar tramas de error. Pero este modo tiene el inconveniente que en el caso de tráfico intenso y velocidades elevadas aumente el riesgo de sobrecarga en los *buffers* de recepción, porque no es posible filtrar los mensajes. Además, en el caso de que quisiéramos probar un único nodo, estamos obligados a operar en modo normal.

[14]Si tenemos un único nodo además del analizador, no recibirá la confirmación ACK de las tramas que envía. Por lo tanto desconoce si sus mensajes son correctamente recibidos.

5.4.4 Otros aspectos

En caso de operación en modo normal podemos seleccionar, mediante el uso de máscaras y filtros, mensajes con identificadores concretos. En las secciones 2.3.2 (página 42) y 3.4.4 (página 55) se explica cómo funciona este mecanismo de filtrado.

El bus utilizado en la transmisión *CAN* debe terminarse con 2 resistencias de 120 Ω, para realizar el acoplo de impedancias y evitar "rebotes". Generalmente el analizador será empleado para estudiar redes *CAN* ya montadas, por lo que no tendremos que preocuparnos de este problema. No obstante, está previsto el caso en el que el analizador fuera el último nodo del sistema *CAN*, y hubiera necesidad de terminar el bus en este punto. La placa *PCB* del analizador posee un *jumper* que, cambiándolo de posición, conecta el bus con una resistencia de 120 Ω (ver figura 5.34).

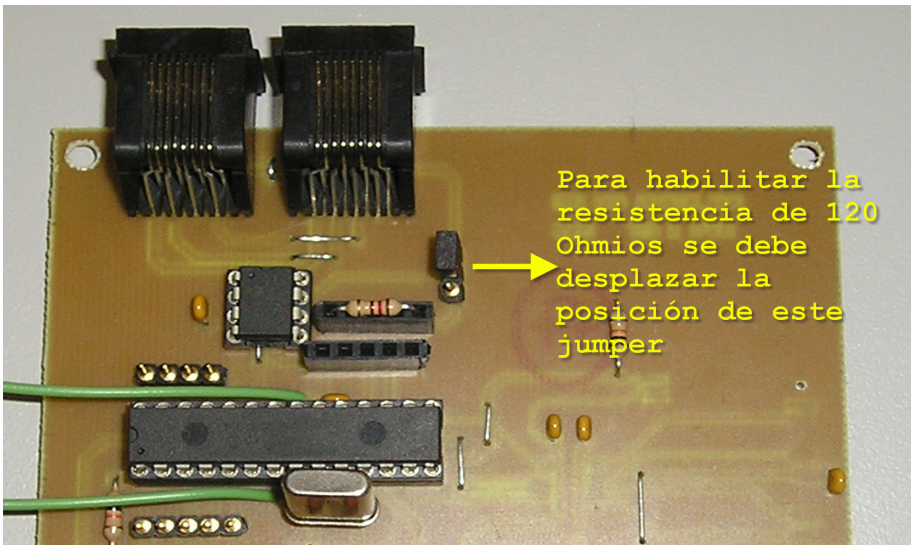


Figura 5.34: *Jumper* que habilita la resistencia de 120 Ω.

5.4.5 Visualización del analizador

A continuación vamos a mostrar algunas imágenes muy ilustrativas del diseño final del prototipo del analizador. En la imagen de la izquierda de la figura 5.35, tenemos el aparato boca abajo, con la tapa levantada. Podemos observar el compartimento donde debemos alojar la batería de 9V que proporciona la energía al sistema. La tapa va atornillada a la caja de forma que la placa PCB que tiene integrado el microcontrolador, el zócalo para la tarjeta de memoria y los conectores RJ45, esté firmemente sujeta. La imagen de la derecha, de la misma figura, muestra la orientación correcta para introducir la tarjeta MMC o SD donde se van a almacenar los datos relativos a la captura de tráfico *CAN*.

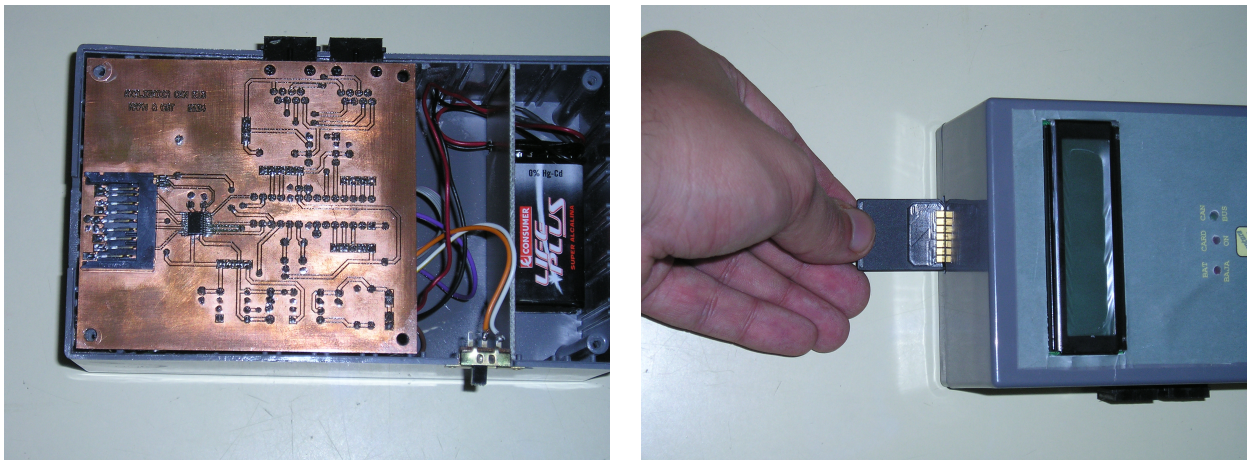


Figura 5.35: Vistas del prototipo final del analizador.

La figura 5.36 presenta dos vistas frontales del analizador. Podemos observar que, marcado con un recuadro amarillo, se encuentra el botón de inicio y finalización de captura de tráfico. Por encima de él se sitúan tres LEDs indicadores. De izquierda a derecha:

- *BAT BAJA*: cuando se ilumina este LED significa que el nivel de alimentación está por debajo de 4,5V, y es conveniente sustituir la pila de 9V de alimentación.
- *CARD ON*: la tarjeta no debe ser introducida ni retirada del analizador hasta que este LED esté apagado.
- *CAN BUS*: este LED parpadea con cada trama aceptada por el analizador.

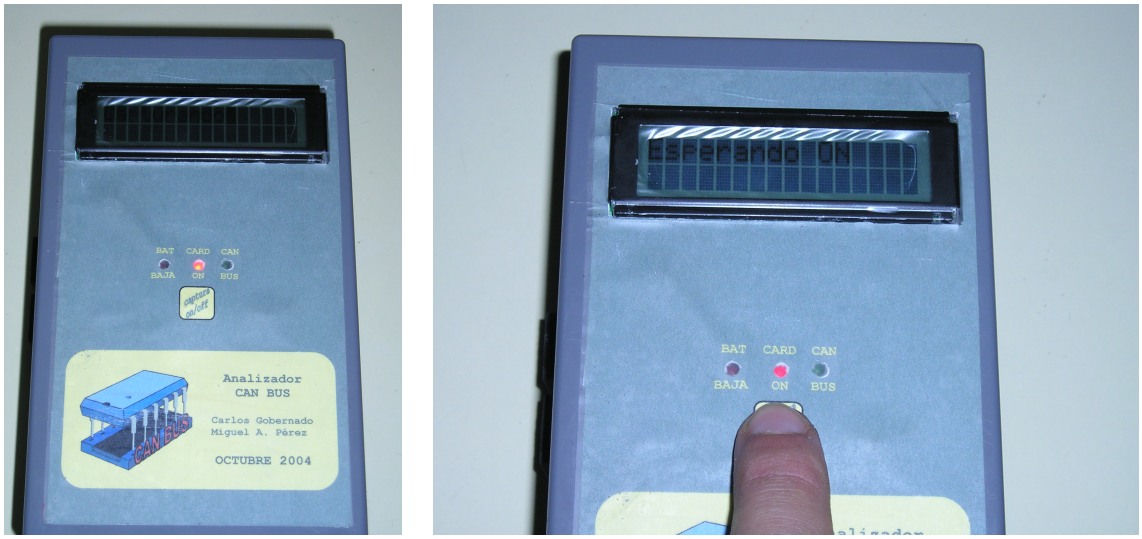


Figura 5.36: Vistas del prototipo final del analizador.

Para finalizar tenemos el *display* LCD, que nos informa con mensajes de los distintos estados en los que se encuentra el analizador.

En los distintos laterales del aparato, tenemos la entrada de la tarjeta de memoria, los conectores para el cable de comunicación y el botón de encendido, que habilita la corriente suministrada por la batería a toda la electrónica integrada en el analizador.

5.5 Interfaz gráfica

5.5.1 Introducción

Los objetivos a la hora de diseñar la interfaz gráfica son:

1. Facilitar el proceso de configuración de las máscaras, filtros y parámetros temporales.

2. Proporcionar una interfaz gráfica sencilla para poder interpretar los datos capturados. Esta interfaz debería de contar con una funcionalidad básica, pero a la vez su implementación tendría que facilitar la escalabilidad de la aplicación, facilitando las posibles mejoras.

5.5.2 Grabación de los parámetros en la tarjeta

El cuadro de diálogo de la figura 5.37 permite configurar tanto los parámetros temporales como los filtros y máscaras. Todos los cálculos se realizan para un reloj de 20Mhz. Tanto las máscaras como los filtros se introducen en formato hexadecimal.

El primer paso que debe de realizar el usuario es elegir una tasa de transmisión y un valor para BRP. En el momento que alguna de las dos cajas de texto pierda el foco, se procederá a calcular el número de T_q que le corresponde a esta combinación. Estos cálculos se realizan con las ecuaciones 5.2 y 5.4, verificando también que se cumple la condición de que el número total de T_q esté entre 8 y 25, mostrando un mensaje de error en caso contrario.

Una vez fijado el número de T_q que tiene el tiempo de bit, es necesario determinar el resto de los parámetros temporales. Recordamos que tanto el segmento de sincronización, propagación, segmento de fase 1 y 2 deben sumar todos ellos el número total de T_q calculado. El programa también nos advertirá con un mensaje de error cuando se viole alguna de las condiciones 5.5 y 5.6.

El usuario también puede configurar el modo de operación del analizador, según el caso al que nos enfrentemos (apartado 5.4.3).

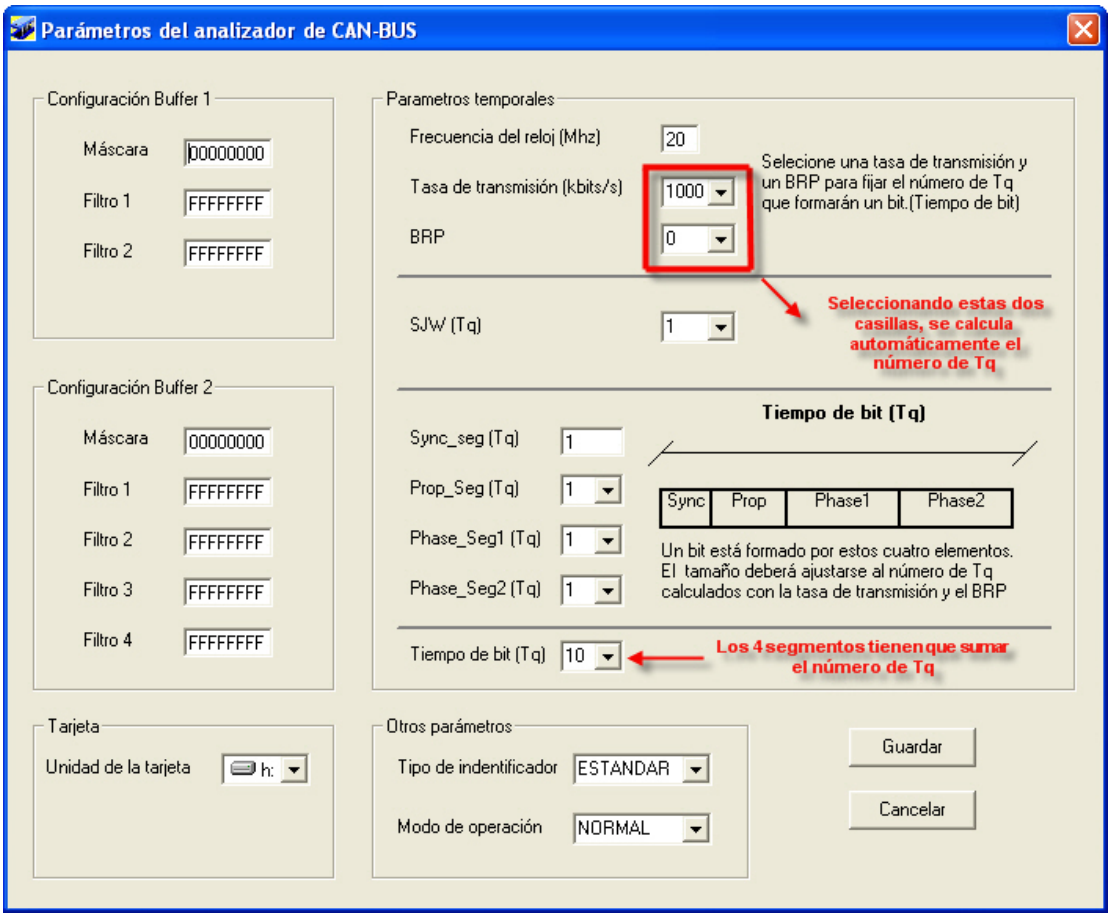


Figura 5.37: Ventana de grabación de los parámetros del analizador.

5.5.3 Carga de los datos copiados en la tarjeta

El usuario a través del comando abrir puede cargar el fichero de captura que posee la extensión .dat. Este fichero es seleccionado a través de una ventana de diálogo estándar como la de la figura 5.38.

La figura 5.39 es la ventana principal del programa. Para facilitar la interpretación de los datos, el identificador de la trama siempre se muestra en hexadecimal. Los datos a través del botón de la barra de tareas se pueden cambiar entre la base decimal y la hexadecimal. En la barra de tareas también se incorpora un control para cambiar la escala de tiempo. Las escalas de tiempo disponibles son milisegundos, segundos y minutos.

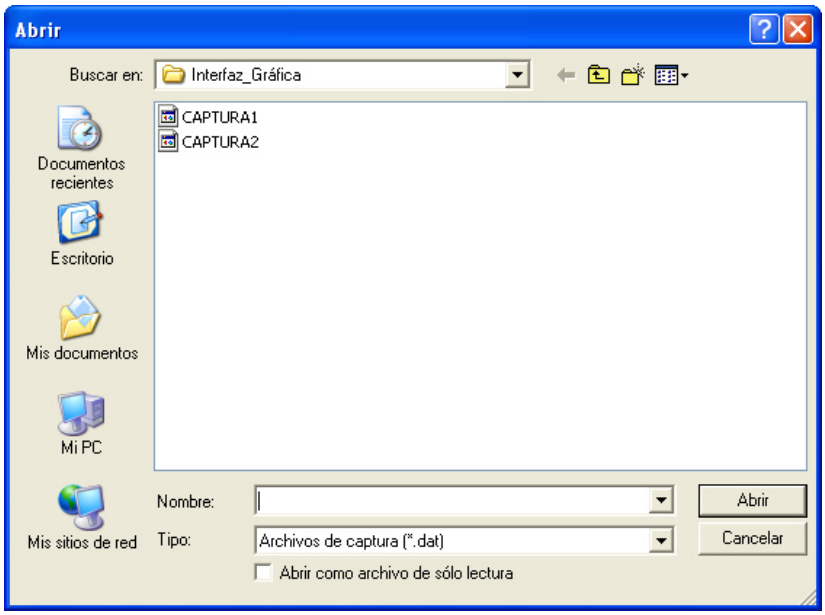


Figura 5.38: Formulario de la carga del fichero de captura.

CANSoft

Archivo Ver Edición Ayuda

1 ms

Captura	Instante de tiempo	Tipo trama	Identificador	Dato[0]	Dato[1]	Dato[2]	Dato[3]	Dato[4]	Dato[5]	Dato[6]	Dato[7]
0	626,28	Estándar	0xFF	10							
1	10653,13	Estándar	0x1	11	48	48	0				
2	12213,4	Estándar	0x2	32	48						
3	20621,41	Estándar	0xFF	10							
4	28213,61	Estándar	0x2	33	48						
5	30616,83	Error									
6	30746,83	Estándar	0x1	11	48	48	0				
7	36290,15	Error									
8	36313,45	Error									
9	36412,36	Estándar	0xFF	10							
10	44217,8	Estándar	0x2	34	48						
11	46345,63	Error									
12	46405,63	Error									
13	46531,89	Estándar	0x1	11	48	48	0				
14	56344,32	Error									
15	56440,52	Estándar	0xFF	10							
16	60224,31	Estándar	0x2	35	48						
17	66391,96	Estándar	0x1	11	48	48	0				
18	76228,15	Estándar	0x2	36	48						
19	76361,27	Estándar	0xFF	10							
20	86388,17	Estándar	0x1	11	48	48	0				
21	92233,57	Estándar	0x2	37	48						
22	96356,4	Estándar	0xFF	10							

Figura 5.39: Ventana de datos capturados por el analizador.

Capítulo 6

Aplicación práctica

6.1 Descripción general

En este capítulo presentamos el planteamiento y desarrollo de una aplicación práctica para probar el correcto funcionamiento del analizador. Dicha aplicación ha consistido en el diseño de un módulo *CAN* universal, al que se pueda conectar un periférico analógico o digital en cualquier momento. El control de estos nodos se realiza vía bus *CAN* por medio de un controlador, que emplea un protocolo de comunicación basado en una serie de comandos. La idea básica que se defiende es la de no necesitar reprogramar el microcontrolador integrado en el nodo universal cada vez que un sensor o un actuador es incorporado, si no que su configuración y posterior utilización se realice a través del bus *CAN*. Esto proporciona a los usuarios comodidad, sencillez y flexibilidad, al descargar en el controlador la tarea de coordinar los distintos nodos universales, tanto para enviar datos como solicitarlos.

Las funciones del nodo universal están reguladas por un microcontrolador PIC18F258[1], el mismo que ha sido utilizado en el diseño del analizador. Disponemos de un botón para la conexión y desconexión del nodo a la red *CAN*. Además, la placa del nodo incluye el *transceiver* necesario para adaptar la señal *CAN*, y dos conectores RJ45 para introducir el cable de comunicación.

El nodo universal ofrece aprovechando las prestaciones del microcontrolador escogido:

- 5 entradas analógicas con conversión analógico-digital de 10 bits de resolución (8 entradas para el PIC18F458).
- 12 entradas o salidas digitales programables, con posibilidad de configurar grupos de 4 y/o de 8 entradas o salidas conjuntas (20 para el PIC18F458).
- 1 salida PWM o de modulación por anchura de pulso (4 para el PIC18F458).

[1] El programa interno está preparado para la ampliación a un microcontrolador PIC18F458, que posee un mayor número de pines para poder conectar más periféricos.

6.2 Protocolo de comunicación entre el nodo controlador y los nodos universales

6.2.1 Comandos

La comunicación entre el nodo controlador y los nodos universales se realiza a través de un protocolo de comunicación basado en comandos. Éstos difieren unos de otros en los valores que toman los distintos bytes del campo de datos. A continuación mostramos el formato de estos comandos.

1. Comando de Petición de Identificador

Origen: nodo universal. **Destino:** nodo controlador. **Ident.:** ID_CONTROLADOR.

Campo de datos de la trama							
Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
PETICION_ID	MAC			Sin utilizar			

2. Comando de Asignación de Identificador

Origen: nodo controlador. **Destino:** nodo universal. **Identificador:** ID_CONEXION.

Campo de datos de la trama							
Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
ID Asignada	MAC			Sin utilizar			

3. Comando de Asignación de Dato

Origen: nodo controlador. **Destino:** nodo universal. **Identificador:** ID nodo.

Campo de datos de la trama							
Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
ENVIO_DATOS	Módulo seleccionado	Dato	MAC			Sin utilizar	

4. Comando de Petición de Dato

Origen: nodo controlador. **Destino:** nodo universal. **Identificador:** ID nodo.

Campo de datos de la trama							
Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
RECEP_DATOS	Módulo seleccionado	0	MAC			Sin utilizar	

5. Comando de Respuesta a la Petición de Dato Digital

Origen: nodo universal. **Destino:** nodo controlador. **Ident.:** ID_CONTROLADOR.

Campo de datos de la trama							
Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
ACEPTACION_DATO	Módulo seleccionado	Dato	0	ID nodo	MAC		

6. Comando de Respuesta a la Petición de Dato Analógico

Origen: nodo universal. **Destino:** nodo controlador. **Ident.:** ID_CONTROLADOR.

Campo de datos de la trama							
Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
ACEPTACION_DATO	Módulo seleccionado	Dato		ID nodo	MAC		

7. Comando de Configuración del Módulo PWM

Origen: nodo controlador. **Destino:** nodo universal. **Identificador:** ID nodo.

Campo de datos de la trama							
Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
RECEP_ DATOS	Módulo seleccionado	Preescala del 'temporizador 2'	Ciclo de trabajo		PR2: ciclo total	Sin utilizar	

8. Comando de Modificación del Módulo PWM

Origen: nodo controlador. **Destino:** nodo universal. **Identificador:** ID nodo.

Campo de datos de la trama							
Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
RECEP_DATOS	Módulo seleccionado	0	Ciclo de trabajo		Sin utilizar		

9. Comando de Petición de Desconexión

Origen: nodo universal. **Destino:** nodo controlador. **Ident.:** ID_CONTROLADOR.

Campo de datos de la trama							
Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
PETICION_ DESCONEXION	ID nodo	MAC			Sin utilizar		

10. Comando de Confirmación de Desconexión

Origen: nodo controlador. **Destino:** nodo universal. **Identificador:** ID nodo.

Campo de datos de la trama							
Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
DESCONEXION	MAC			Sin utilizar			

6.2.2 Campo de control

El primer byte del campo de datos de cada comando contiene información relativa sobre el tipo del mismo[2]. Será analizado tanto en el nodo controlador como en los nodos universales para establecer la actuación a seguir. En la tabla 6.1 mostramos la asignación de valores para cada situación.

Campo de control	**Código**
Comandos enviados por el universal	
PETICION_ID	0
ACEPTACION_DATO	1
PETICION_DESCONEXION	2
Comandos enviados por el controlador	
ENVIO_DATOS	0
RECEP_DATOS	1
DESCONEXION	2

Tabla 6.1: Código numérico para el campo de control de los comandos del protocolo.

6.2.3 Identificadores

El nodo controlador es el encargado de asignar identificadores a los distintos nodos universales que se van conectando a nuestro sistema *CAN*, los cuales envían un comando de Petición de Identificador. Existen dos identificadores reservados, el ID_CONTROLADOR y el ID_CONEXION. El primero se corresponde al utilizado para todos los mensajes destinados al controlador, mientras que el segundo es el empleado cuando el nodo controlador debe enviar el comando de Asignación de Identificador a un nodo que lo ha solicitado, pero que aún no lo tiene. Como vimos en la sección 2.2 (página 33), los identificadores más bajos son los que prevalecen en el bus. Por esta razón, se asigna el valor '0' al ID_CONTROLADOR, pues consideramos que los mensajes más prioritarios son los destinados al controlador, y el valor '1' al ID_CONEXION. Cada nodo universal está programado para aceptar inicialmente mensajes sólo con este último identificador, hasta que reciba el comando y reajuste su módulo *CAN* con el nuevo identificador. La política de

[2]Exceptuando en el Comando de Asignación de Identificador, que contiene el identificador asignado al nodo universal.

asignación de éstos es responsabilidad del usuario que maneje el nodo controlador.

Los nodos universales incluyen su identificador en todos los mensajes que envían al nodo controlador, para que éste pueda distinguir su procedencia.

6.2.4 MAC

Una posibilidad interesante del bus *CAN* es la difusión *multicast*, esto es, enviar el mismo mensaje a un conjunto de nodos con características similares. Esto se consigue asignando el mismo identificador. Pero en un momento dado podemos necesitar discriminar un nodo entre todos ellos, por lo que necesitaríamos un parámetro que sea único para cada nodo. Otro caso a tener en cuenta es el supuesto en el que dos nodos acceden a la red *CAN* simultáneamente. Ambos serán contestados por el controlador por sendos comandos con ID_CONEXION, asignando dos identificadores, uno para cada nodo. Pero el primero de ellos será leído por los dos, y como ambos están esperando un comando de este tipo, considerarán que es su identificador asignado. Cuando llegue el segundo comando, ambos nodos ya tendrán asignado su identificador, el mismo para los dos, estando uno de ellos erróneamente configurado. Éstos son los motivos por los cuales, además del identificador, cada nodo tiene asignado una dirección única MAC. Esta dirección va incluida en todos los comandos y permiten, por una parte, a cada nodo universal confirmar si el mensaje va dirigido a él[3], y por otro lado proporciona información adicional al nodo controlador sobre el origen del mensaje recibido.

6.2.5 Módulos

El nodo universal ofrece una serie de pines a los que conectar los periféricos. Cada uno de ellos, incluyendo las distintas posibilidades y combinaciones, debe tener asignado un código numérico único y común a todos los nodos, incluyendo al controlador. De esta forma podremos distinguir para qué pin nos han solicitado o enviado el dato, y la naturaleza del mismo. Para los posibles microcontroladores empleados, se muestran en la tabla 6.2 los puertos disponibles, con su función y código correspondiente.

6.2.6 Ejemplo

Supongamos una situación muy simple, en la que un nuevo nodo universal se incorpora al sistema *CAN*, le solicitan un dato, lo envía y a continuación pide ser desconectado.

En primer lugar, al incorporarse a la red *CAN*, el universal envía el comando de Petición de Identificador, con el identificador ID_CONTROLADOR. El controlador recibe el mensaje y después de consultar su algoritmo interno de asignación de identificadores, envía el comando de Asignación de Identificador, con identificador ID_CONEXION. El nodo universal lo acepta, pues su sistema de filtros está configurado inicialmente para

[3]Para realizar una difusión *multicast* se puede introducir una MAC igual a '0', que será reconocida por todos los nodos como propia.

Módulo	Descripción	Código	Módulo	Descripción	Código
AN_IN0	Entrada analóg. Pin RA0	0	PIN_RC3	E/S digital Pin RC3	15
AN_IN1	Entrada analóg. Pin RA1	1	PIN_RC4	E/S digital Pin RC4	16
AN_IN2	Entrada analóg. Pin RA2	2	PIN_RC5	E/S digital Pin RC5	17
AN_IN3	Entrada analóg. Pin RA3	3	PIN_RC6	E/S digital Pin RC6	18
AN_IN4	Entrada analóg. Pin RA5	4	PIN_RC7	E/S digital Pin RC7	19
AN_IN5[5]	Entrada analóg. Pin RE0	5	ALL_PORTC	E/S dig. Pines RC0-RC7	29
AN_IN6[5]	Entrada analóg. Pin RE1	6	PIN_RD0[5]	E/S digital Pin RD0	20
AN_IN7[5]	Entrada analóg. Pin RE2	7	PIN_RD1[5]	E/S digital Pin RD1	21
PIN_RB4	E/S digital Pin RB4	8	PIN_RD2[5]	E/S digital Pin RD2	22
PIN_RB5	E/S digital Pin RB5	9	PIN_RD3[5]	E/S digital Pin RD3	23
PIN_RB6	E/S digital Pin RB6	10	PIN_RD4[5]	E/S digital Pin RD4	24
PIN_RB7	E/S digital Pin RB7	11	PIN_RD5[5]	E/S digital Pin RD5	25
ALL_PORTB_ALTO	E/S dig. Pines RB4-RB7	28	PIN_RD6[5]	E/S digital Pin RD6	26
PIN_RC0	E/S digital Pin RC0	12	PIN_RD7[5]	E/S digital Pin RD7	27
PIN_RC1	E/S digital Pin RC1	13	ALL_PORTD[5]	E/S dig. Pines RD0-RD7	30
PIN_RC2	E/S digital Pin RC2	14	PWM_IN	Salida PWM Pin RC2	31

Tabla 6.2: Código numérico para los pines de conexión del nodo universal.

aceptar tramas con ese único identificador. Comprueba si coincide el campo del comando que contiene la MAC con la propia. En caso afirmativo, el módulo *CAN* entra en modo de configuración, y los filtros se modifican para aceptar mensajes también con el identificador asignado por el controlador.

Una vez adaptado al sistema *CAN*, el nodo universal recibe un comando de Petición de Dato, que ha reconocido al sondear el primer byte del campo de datos[4]. Selecciona el pin, o conjunto de pines, al que se ha solicitado la información, y envía el comando de Respuesta a la Petición de Dato Digital o el de Respuesta a la Petición de Dato Analógico (según la naturaleza del dato pedido), con identificador ID_CONTROLADOR. El nodo controlador reconoce el comando y almacena el dato para realizar la tarea que considere oportuna con él.

El usuario del nodo universal decide su retirada de la red *CAN*, y pulsa el botón de finalización. El nodo genera el comando de Petición de Desconexión, con identificador ID_CONTROLADOR. El controlador recibe dicho comando, y realiza las tareas necesarias en su programa interno para tramitar esta solicitud. Si está conforme, el nodo controlador envía el comando de Confirmación de Desconexión, que es recibido por el nodo universal, que comienza la secuencia de desconexión.

[4]Previamente habrá comprobado que la dirección MAC coincide con la propia.
[5]Sólo disponible para el microcontrolador PIC18F458.

6.3 Diseño hardware

En esta sección vamos a describir el hardware del nodo universal *CAN*. El nodo universal se compone básicamente de un microcontrolador PIC18F228, un *transceiver CAN* y un regulador de 5V para asegurar la correcta alimentación del circuito. En la figura 6.1 se puede observar el bloque de alimentación y las conexiones de los terminales RJ45 para el *CAN*. El esquema de pines utilizados para los conectores RJ45 es el propuestos por el estándar CANOpen citado en la sección 2.2 (página 33).

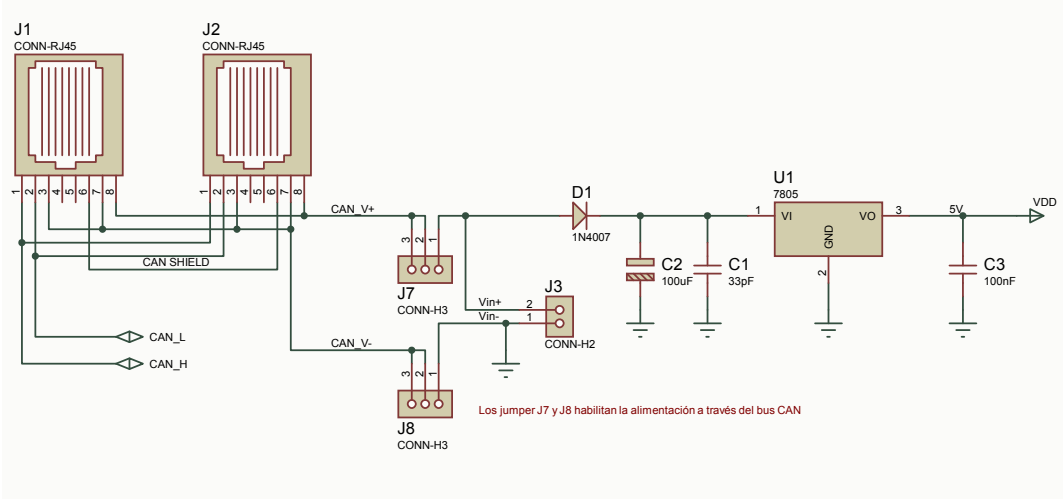


Figura 6.1: Conexiones *CAN* y etapa de alimentación.

El nodo se puede alimentar de dos formas:

1. **A través del bus CAN**. Para ello habrá que habilitar los *jumpers* J7 y J8 (ver figura 6.3). En ese caso, el terminal J3 puede servir para poder extraer la alimentación del bus *CAN* hacia otros dispositivos. La tensión máxima de alimentación que podría soportar este nodo es de 24V[6]. Lo normal en cualquier aplicación es que el bus *CAN* esté alimentado a tensiones de 12V o 24V.

2. **A través del terminal J3**. Si el bus *CAN* no está alimentado, podemos introducir alimentación a través de este terminal. La tensión mínima de entrada estaría entorno a los 8V[7]. Este terminal también puede ser utilizado para alimentar todos los dispositivos del bus.

[6]La tensión máxima que soporta el condensador electrolítico C2 es de 25V.

[7]La tensión mínima de entrada del regulador 7805 es de 6,7V; mas la caída de tensión del diodo obtenemos 7,9V.

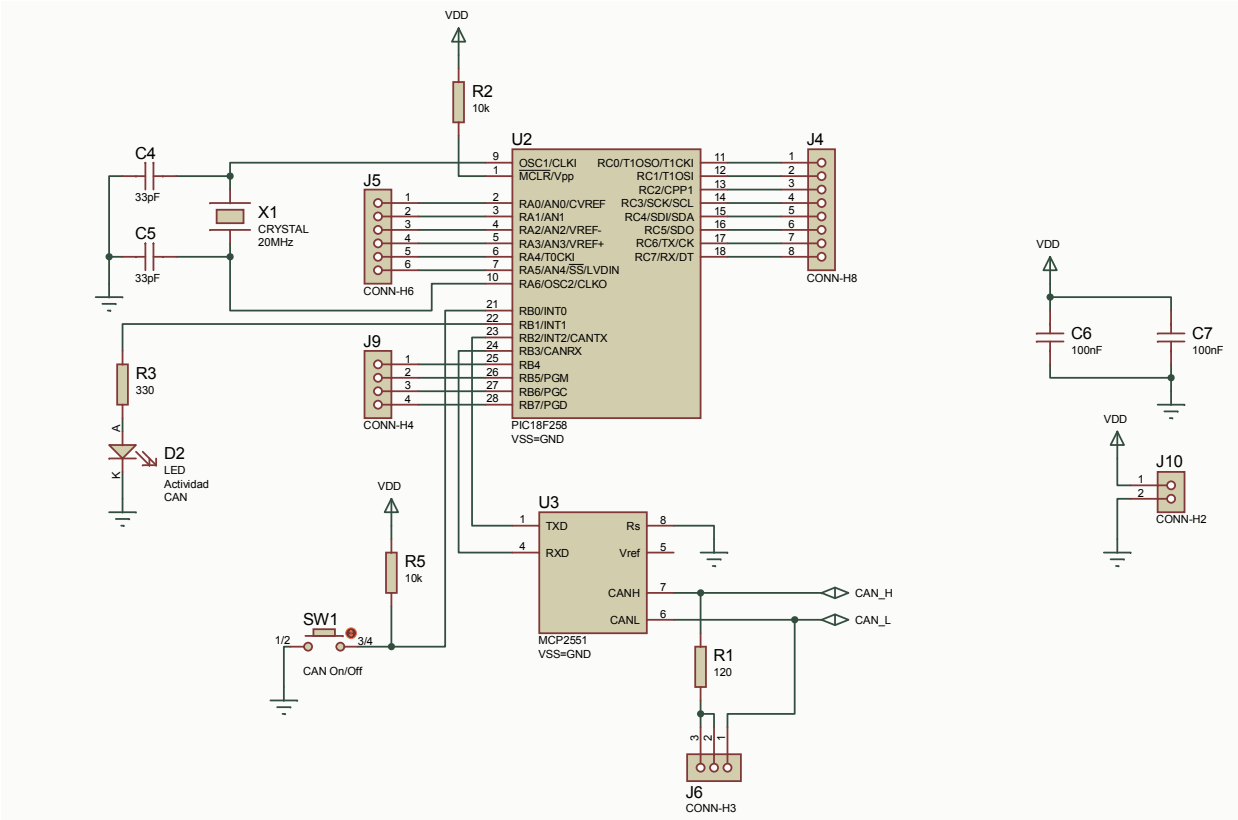


Figura 6.2: Microcontrolador y *transceiver CAN*.

En la figura 6.2 se puede observar el microcontrolador y el *transceiver CAN*. Se ha añadido un botón de conexión/desconexión a la red *CAN* y un diodo LED cuya misión es la de parpadear cada vez que el nodo transmita o reciba tramas. El terminal J10 está destinado a proporcionar tensión de alimentación de 5V a circuitos externos que lo requieran.

El nodo posee un *jumper* J6 para habilitar la resistencia de terminación del bus, si se da el caso (ver figura 6.3). Para evitar el engorro de tener que modificar los *jumpers* J6 en los nodos finales, hemos ideado unos terminadores de bus con conectores RJ45 machos (ver figura 6.4). Estos terminadores habrá que colocarlos en los conectores RJ45 libres de los nodos finales.

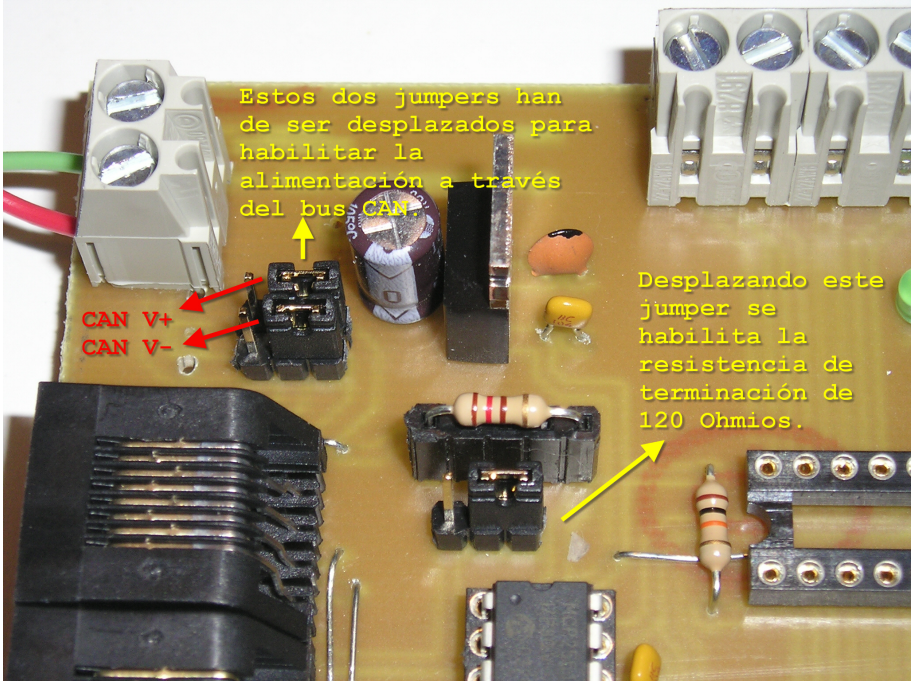


Figura 6.3: Vista detallada de la posición de los *jumpers* de configuración.

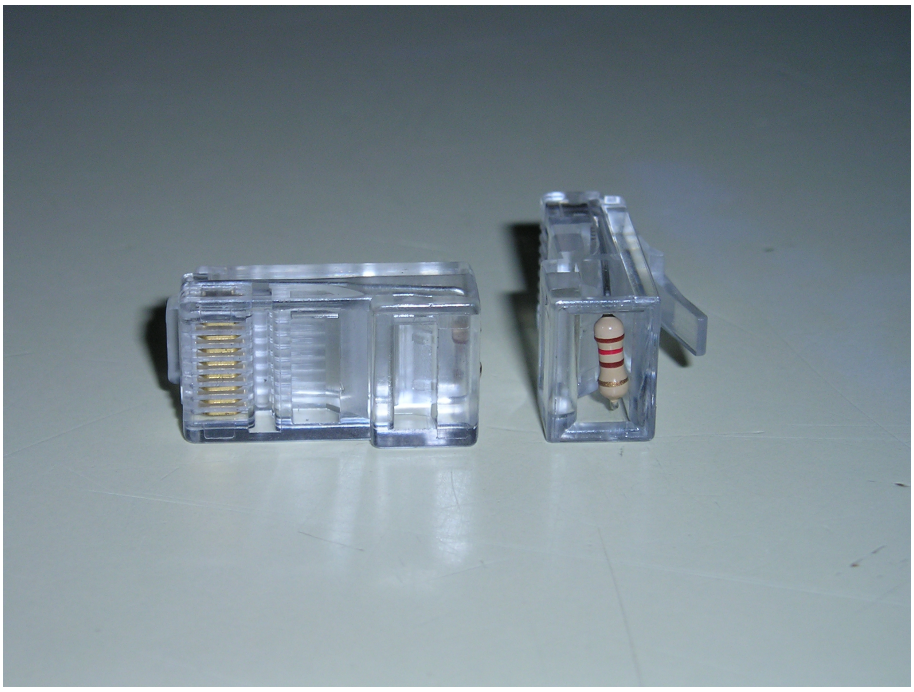


Figura 6.4: Terminadores RJ45.

6.4 Diseño software

6.4.1 Comentarios generales

El programa del nodo universal comienza configurando los distintos módulos que se van a utilizar, incluyendo el sistema *CAN*, para a continuación enviar al nodo controlador un mensaje solicitando asignación de identificador. La carga de trabajo se encuentra situada principalmente en la rutina de atención a interrupciones. Cuando se recibe una trama, se analizan los distintos bytes del campo de datos, y en función de su valor y del protocolo establecido se establece la actuación que corresponda.

La única tarea que se realiza en el bucle principal del programa, fuera de la zona de interrupciones, es el sondeo continuo del botón de finalización. Cuando detecta que ha sido pulsado, envía un mensaje al controlador solicitando la desconexión del nodo universal de la red *CAN*. En caso de confirmación, se enciende un LED durante 5 segundos para informar al usuario de que ese nodo puede ser definitivamente retirado.

6.4.2 Definiciones de las variables principales y tipos de datos del programa del nodo universal

Utilizamos los siguientes cuatro tipos de datos enteros:

- **uchar**: *unsigned char*, entero de 1 byte sin signo.
- **uint**: *unsigned int*, entero de 2 bytes sin signo.
- **uslong**: *unsigned short long*, entero de 3 bytes, sin signo.
- **ulong**: *unsigned long*, entero largo, de 4 bytes, sin signo.

También se emplean *arrays* de variables y el tipo de variable *enum*. El programa utiliza las variables que mostramos a continuación:

- ulong NewMessage: contiene el identificador de la última trama capturada.
- uchar NewMessagedata [8]: almacena el campo de datos de la última trama capturada.
- uchar NewMessageLen: contiene el tamaño de la última trama capturada.
- enum CAN_RX_MSG_FLAGS NewMessageFlags: enumeración con los distintos casos que se pueden presentar en la captura de una trama.
- ulong ID_nodo: registra el identificador con el que asociaremos a este nodo en concreto. Por defecto vale '1', que corresponde al identificador usado por el controlador para el mensaje de asignación de ID.
- uchar i: selecciona el canal de entrada/salida correspondiente.

- uint temp1: variable temporal utilizada para realizar cálculos.
- uchar envio_dato[8]: almacenamos los distintos campos de datos de las tramas que el nodo universal envía en distintas fases de su funcionamiento.

6.4.3 Descripción de funciones y diagramas de flujo del programa del nodo universal

main

- **Prototipo:** void main(void)
- **Descripción:** llama a las funciones de configuración de los distintos módulos del microcontrolador, incluyendo el módulo *CAN*. Tramita la solicitud de asignación de un identificador al nodo controlador. Activa las interrupciones globales y entra en un bucle infinito en el que no realiza ninguna actividad, hasta que la llegada de un mensaje con su identificador asignado ocasiona un salto a la rutina de interrupción.
- **Diagrama de flujo:** figura 6.5.

Setup

- **Prototipo:** void Setup(void)
- **Descripción:** Configura los puertos por defecto del microcontrolador de la familia PIC18FXX8 utilizado para el nodo universal, así como el conversor A/D. Habilitamos el sistema de prioridades para las interrupciones y clasificamos como de alta prioridad la a INT0.
- **Diagrama de flujo:** figura 6.6.

SetupCAN

- **Prototipo:** void SetupCAN(void)
- **Descripción:** Configura el módulo *CAN*; fija las máscaras, filtros, el *baud rate*, tipo de tramas que se van a capturar, modo de operación y las interrupciones asociadas (incluyendo su prioridad).
- **Diagrama de flujo:** figura 6.7.

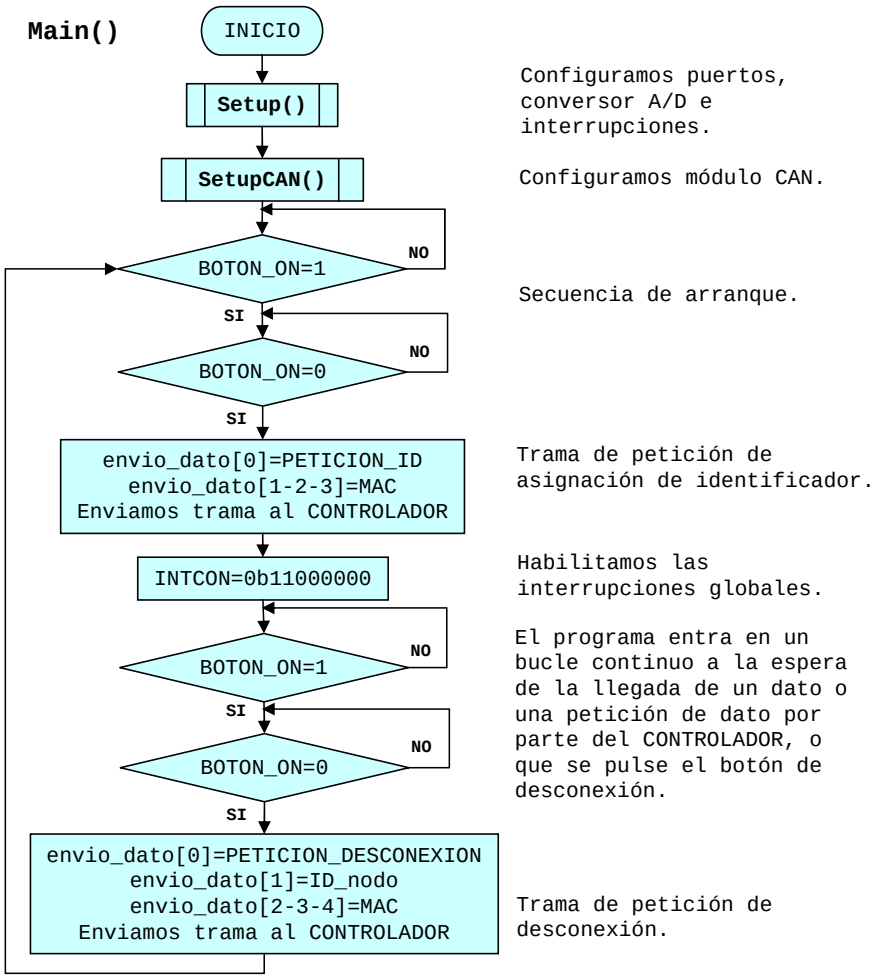


Figura 6.5: Diagrama de flujo de la función Main.

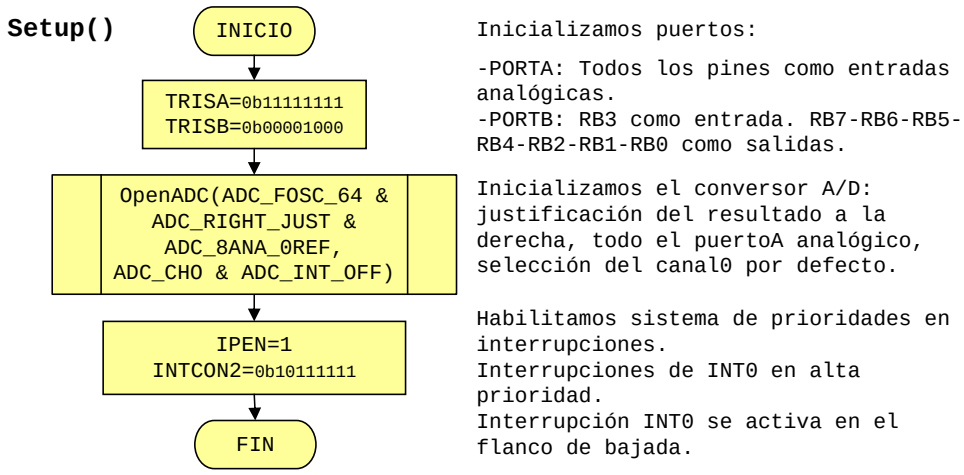


Figura 6.6: Diagrama de flujo de la función Setup.

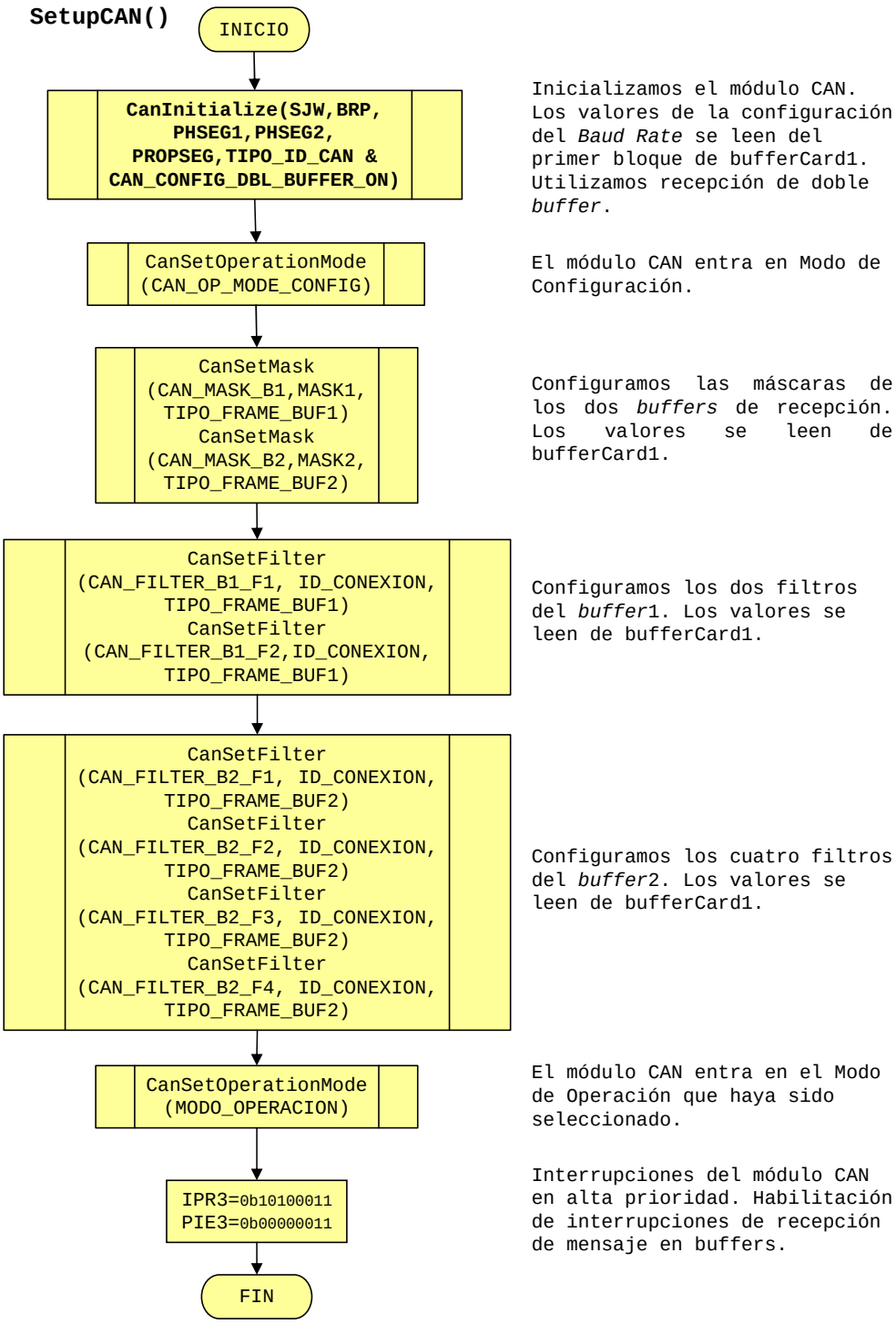


Figura 6.7: Diagrama de flujo de la función SetupCAN.

IntCanRx

- **Prototipo:** void IntCanRx(void)
- **Descripción:** Es la rutina de atención interrupciones de alta prioridad, en la que tratamos los eventos de captura de trama *CAN* y el asociado a INT0. Analizando los distintos bits de *flag*, deduciremos el origen de la interrupción y se realizará el tratamiento correspondiente.
- **Diagrama de flujo:** hoja 1, figura 6.8; hoja 2, figura 6.9; hoja 3, figura 6.10; hoja 4, figura 6.11; hoja 5, figura 6.12.

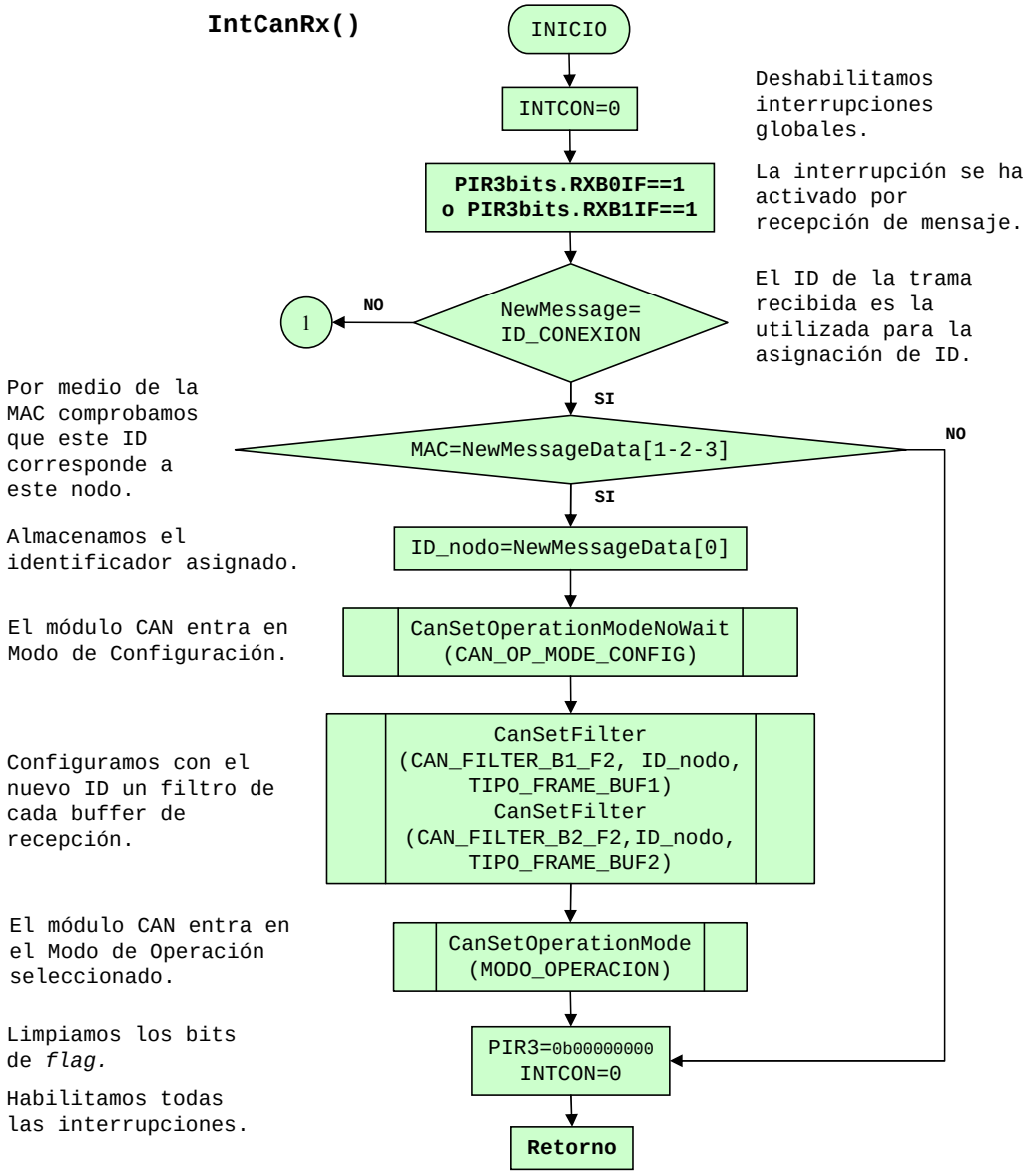


Figura 6.8: Diagrama de flujo de la función IntCanRx (hoja1).

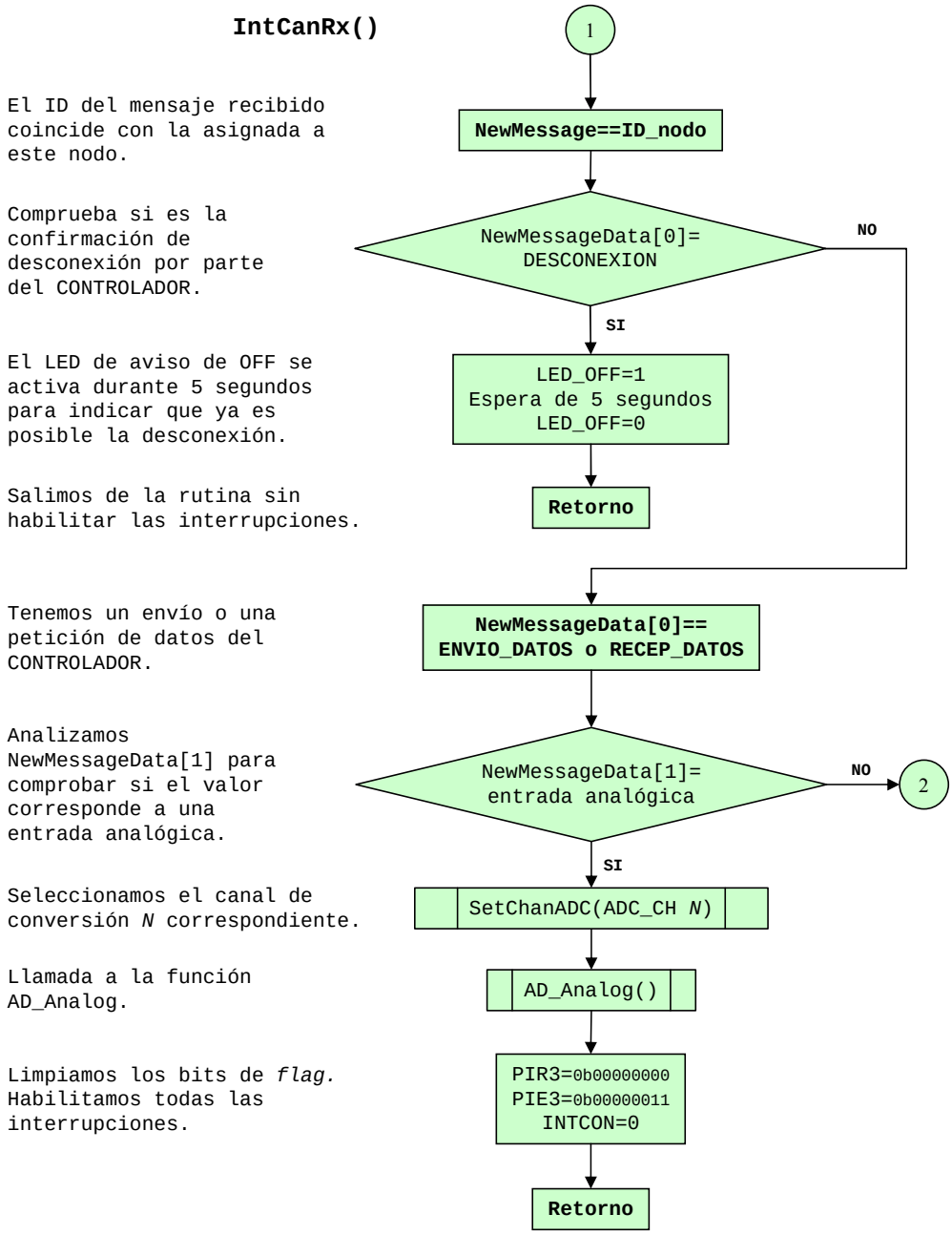


Figura 6.9: Diagrama de flujo de la función IntCanRx (hoja2).

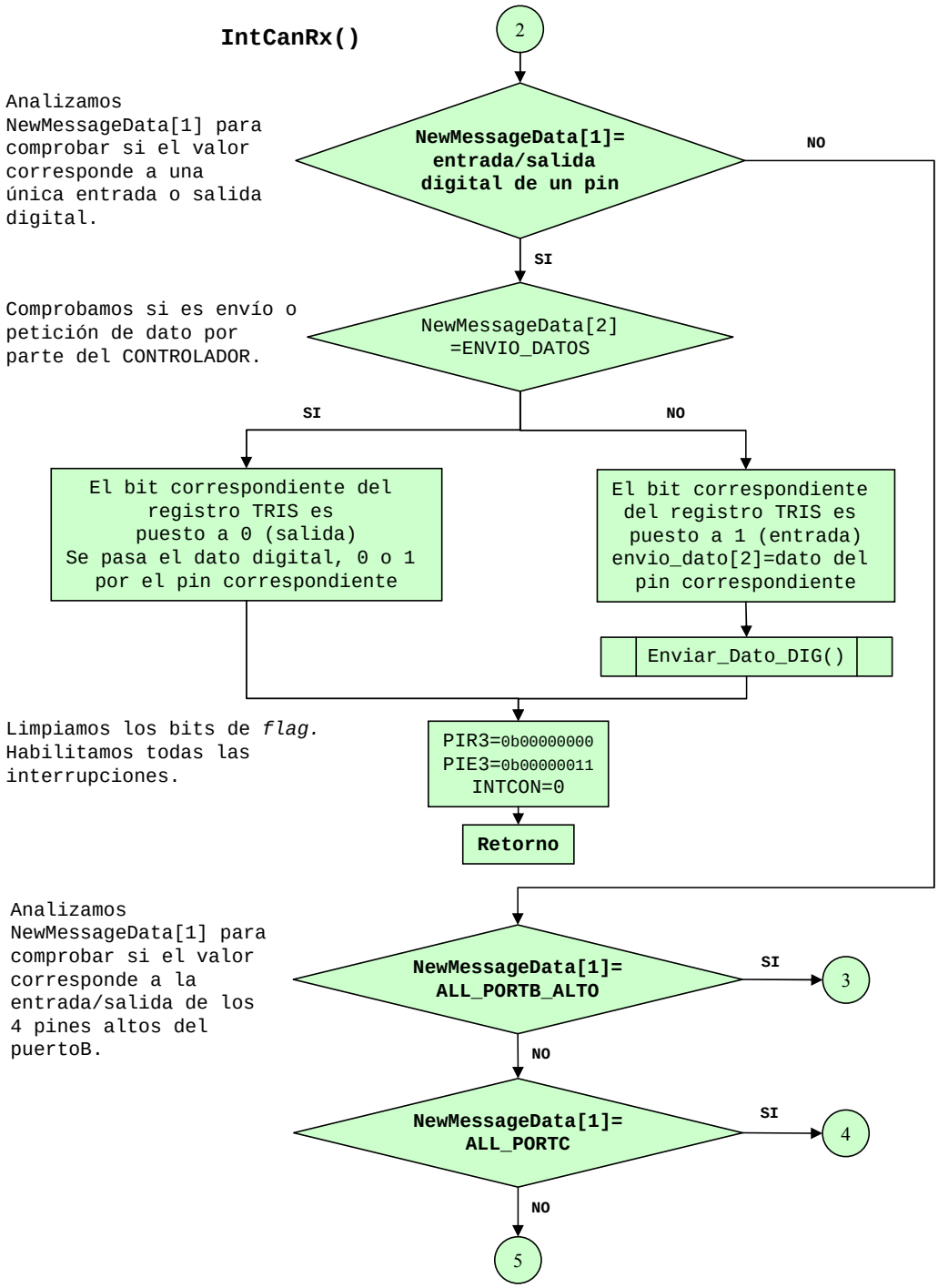


Figura 6.10: Diagrama de flujo de la función IntCanRx (hoja3).

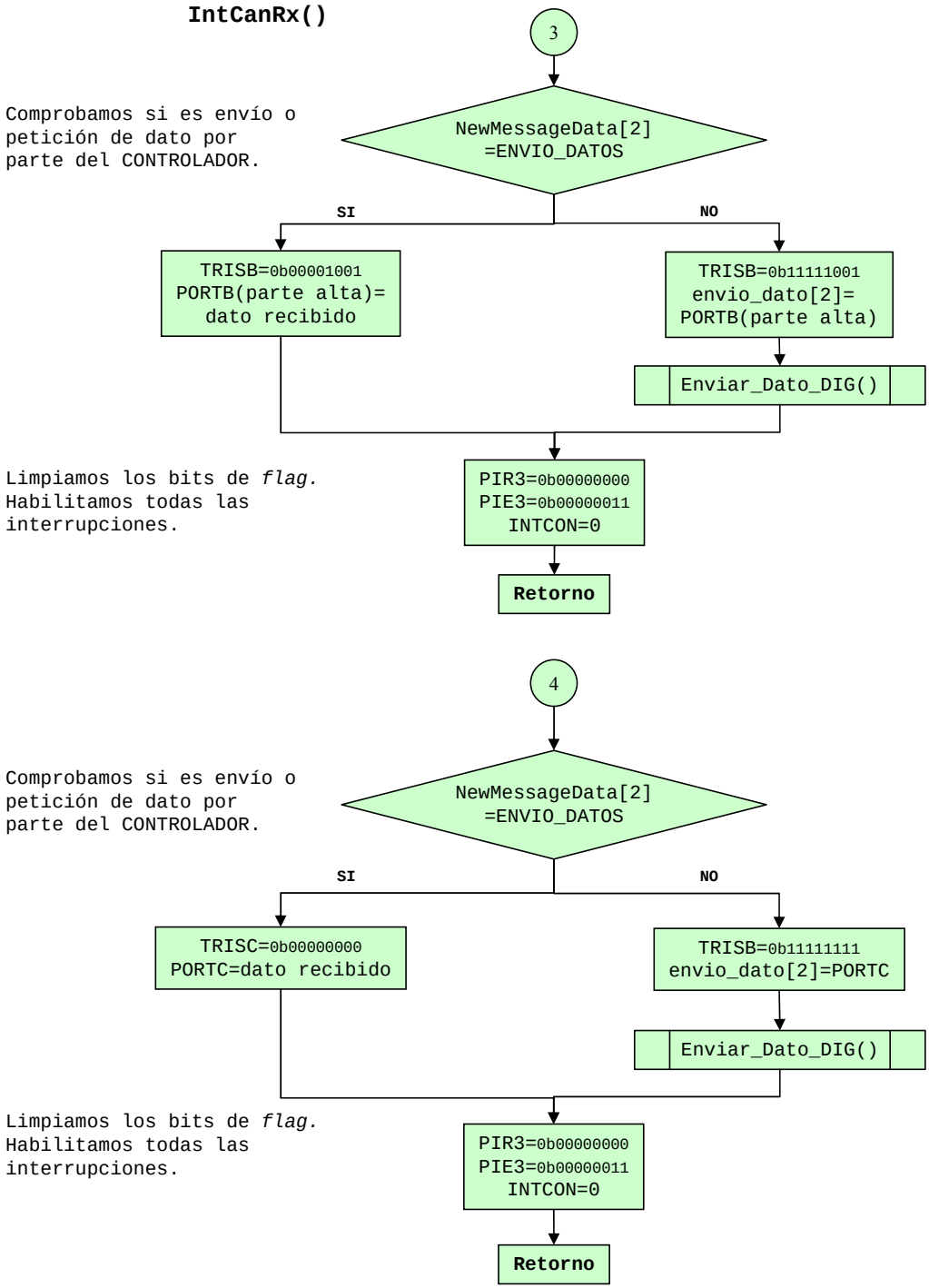


Figura 6.11: Diagrama de flujo de la función IntCanRx (hoja4).

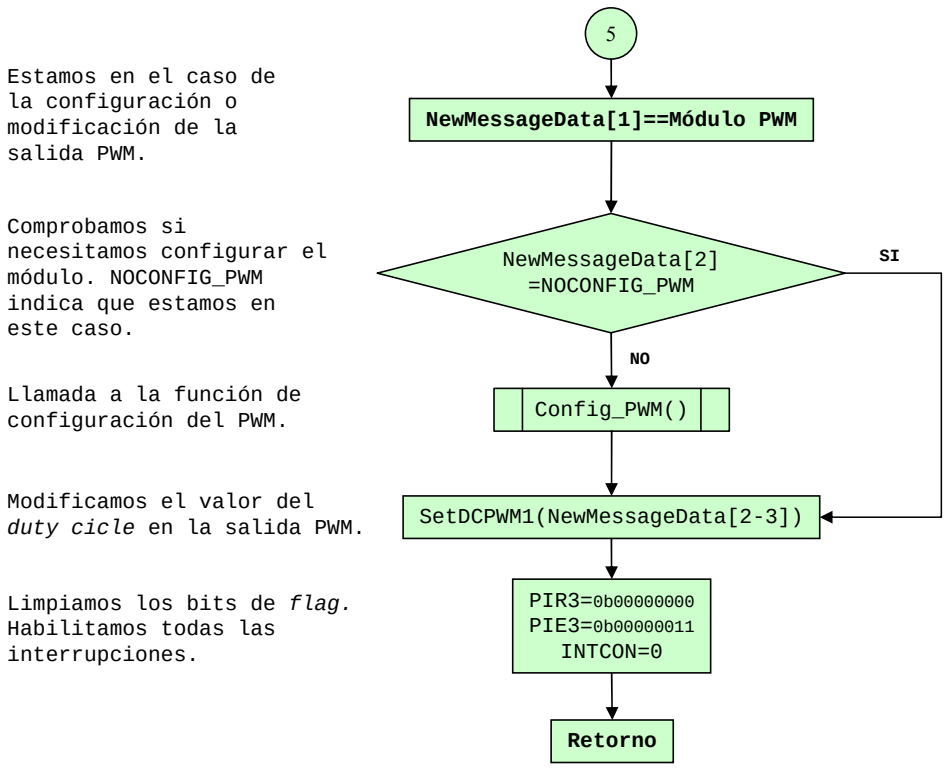


Figura 6.12: Diagrama de flujo de la función IntCanRx (hoja5).

Enviar_Dato_DIG

- **Prototipo:** void Enviar_Dato_DIG(void)
- **Descripción:** Lee el dato de la entrada digital seleccionada y prepara la trama *CAN* con la información necesaria para ser enviada al CONTROLADOR.
- **Diagrama de flujo:** figura 6.14.

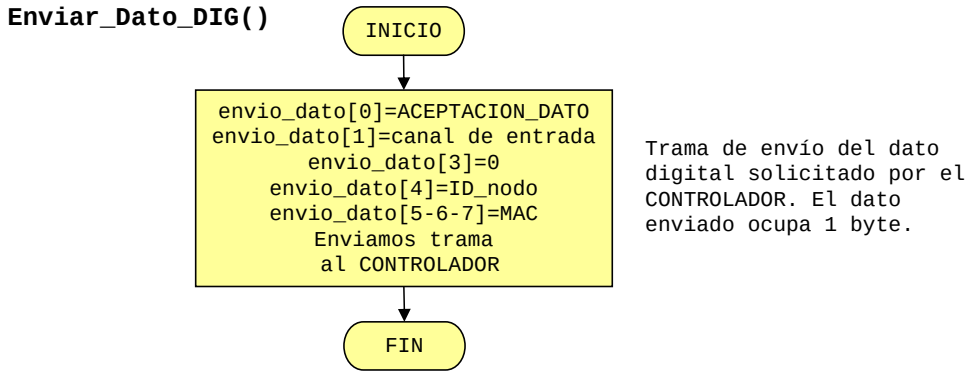


Figura 6.13: Diagrama de flujo de la función Enviar_dato_dig.

AD_Analog

- **Prototipo:** void AD_Analog(void)
- **Descripción:** Realiza la conversión analógica-digital de la entrada analógica seleccionada y guarda el dato en una variable interna del tipo int. Prepara la trama *CAN* con la información necesaria y la envía al CONTROLADOR.
- **Diagrama de flujo:** figura 6.14.

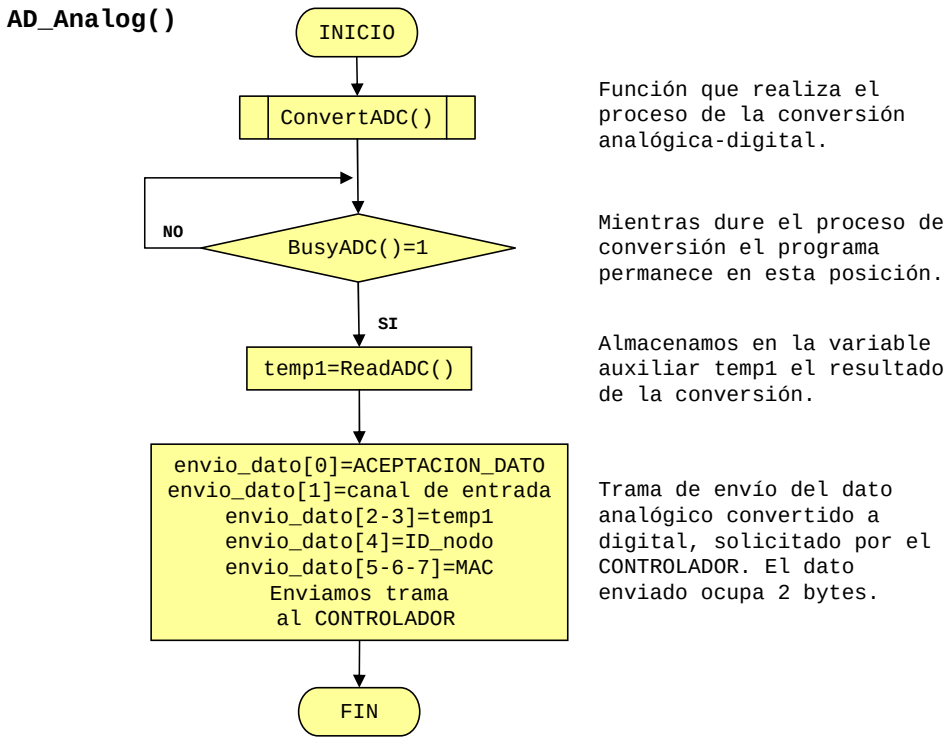


Figura 6.14: Diagrama de flujo de la función AD_analog.

Config_PWM

- **Prototipo:** void Config_PWM(void)
- **Descripción:** Configura el módulo de Modulación por Anchura de Pulso o PWM. Primero configura el 'temporizador 2', con una post-escala de 1:1, con las interrupciones deshabilitadas y con una pre-escala cuyo valor toma del comando recibido. Fija el valor del registro PR2, que determina el periodo total de la señal PWM.
- **Diagrama de flujo:** figura 6.15.

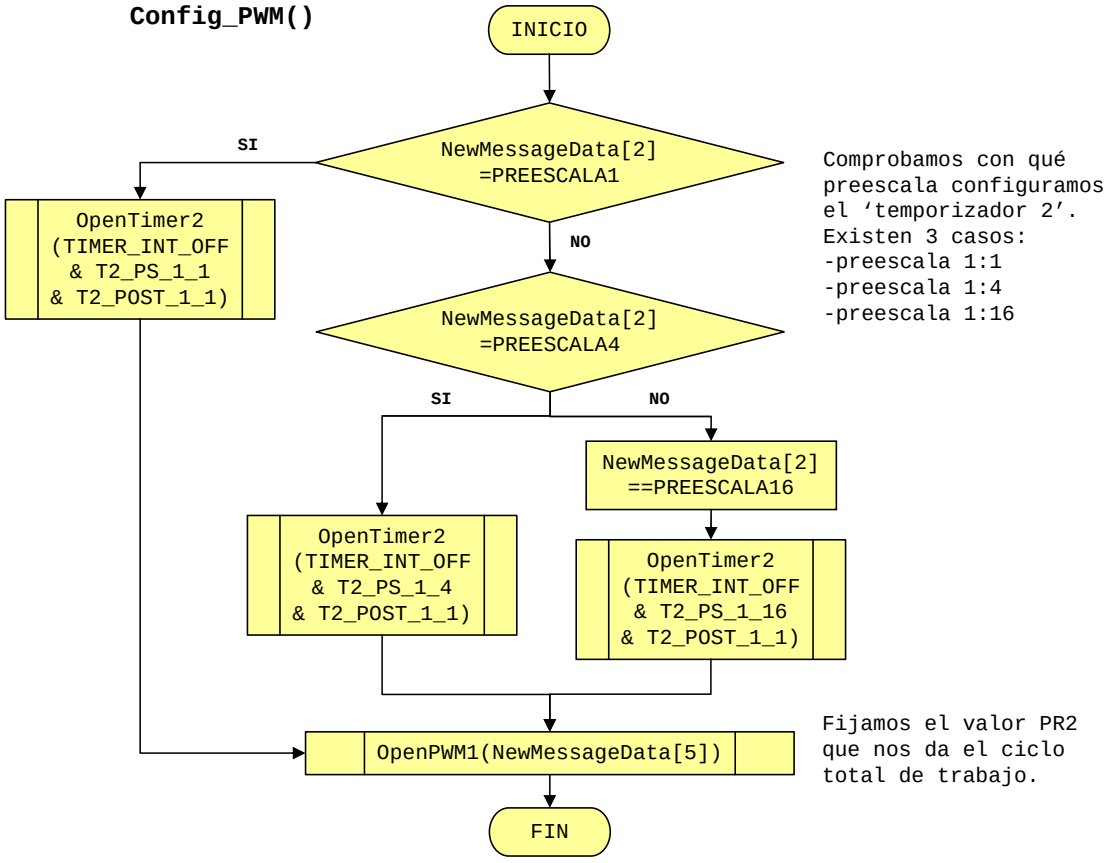


Figura 6.15: Diagrama de flujo de la función Config_PWM.

6.5 Ejemplo práctico propuesto

El sistema *CAN* de nuestra aplicación práctica está integrado por dos nodos universales y el nodo controlador. El nodo universal '1' tiene conectado un sensor de temperatura, que se está escaneando periódicamente. El dato correspondiente es mostrado por un *display* LCD en el controlador. Por otra parte, un puerto completo de 8 bits del universal '1' se utiliza para mostrar un valor en un *display* de 7 segmentos. El controlador a su vez regula un motor de continua en el nodo universal '2', a través del módulo PWM, pudiendo variar su velocidad. El nodo universal '2' también presenta un pin de entrada digital, que mandará información al controlador. En las figuras 6.16 y 6.17 mostramos la imagen de un nodo universal y el esquema de conexión de la aplicación montada respectivamente.

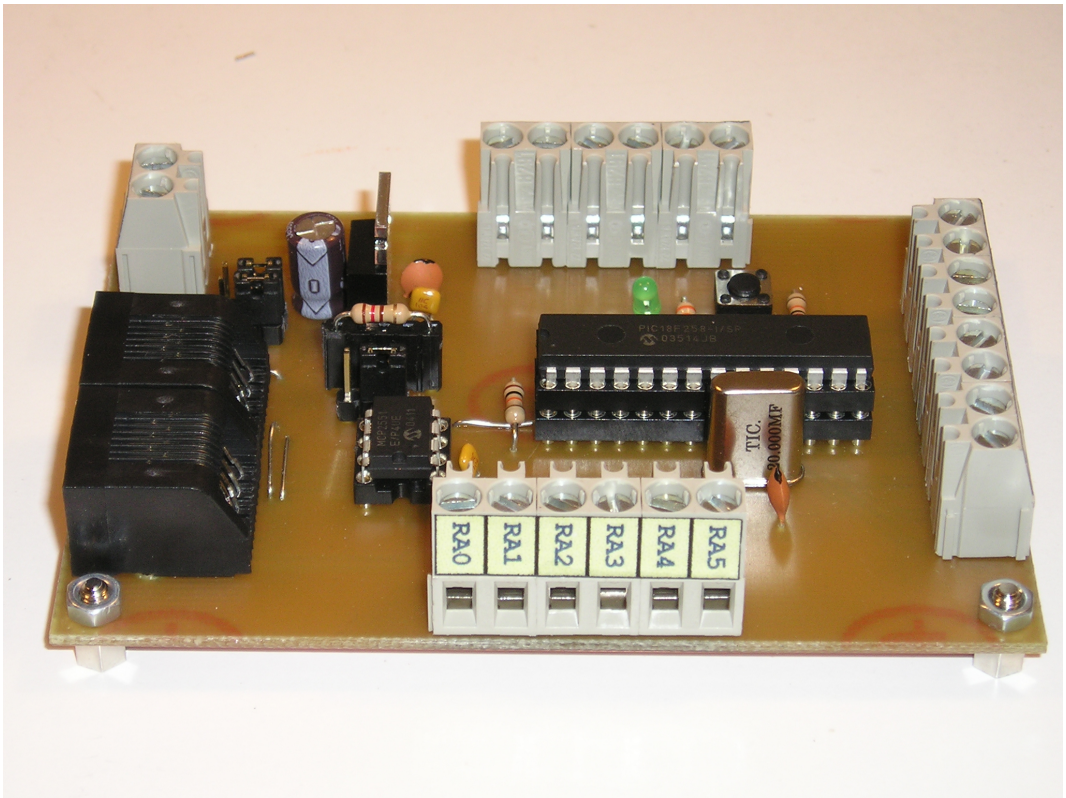


Figura 6.16: Imagen de un nodo universal.

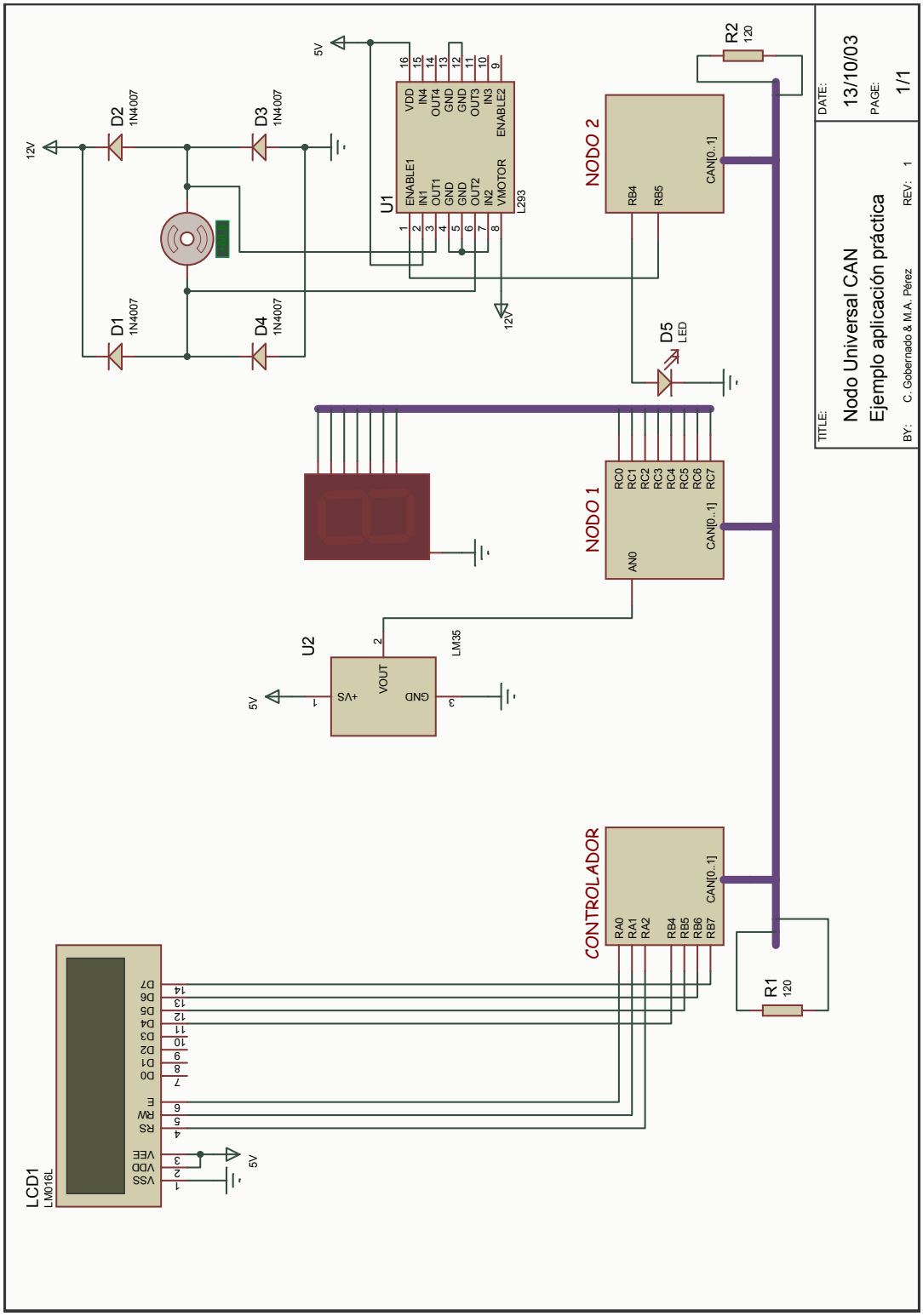


Figura 6.17: Diseño de la aplicación práctica propuesta.

Capítulo 7

Costes y presupuesto

En este capítulo vamos a evaluar el presupuesto tanto del analizador como del nodo universal. Para ello vamos primero a estimar el coste de ambos prototipos diseñados, considerando tanto coste del material como el capital humano invertido. Si los proyectos se decidiesen como viables, hay que tener en cuenta nuevos aspectos como son los costes correspondientes al montaje de la cadena de producción, los gastos de la misma, los salarios del personal necesario, el marketing y la publicidad, etc. Como ingenieros de prototipos, nos limitaremos al cálculo del coste para una producción variable de unidades, dejando los otros conceptos mencionados como un factor de gasto extra que se incluye para poder estimar el precio de salida al mercado de los dos productos.

7.1 Coste de los prototipos

7.1.1 Material y componentes del analizador

En la tabla 7.1 mostramos el coste de los distintos componentes utilizados en el diseño del prototipo del analizador. Los precios son susceptibles de cambios por parte de los proveedores. El microcontrolador y los integrados deben obtenerse de las compañías Microchip [19] y MAXIM [24].

7.1.2 Coste humano del analizador

En este apartado hacemos referencia al trabajo realizado por los dos ingenieros de prototipos. El tiempo necesario para la realización del analizador fue de 22 semanas (5 meses), con una media de 40 horas semanales. Consideramos que el salario de un ingeniero de estas características en una empresa mediana, se sitúa sobre los 18 €/hora. Con estos datos podemos calcular el coste humano total:

Número de horas = 22 semanas × 40 horas/semana = 880 horas.
Coste por ingeniero = 880 horas × 18 €/hora = 15.840 €.
Coste humano total = 15.840 € × 2 ingenieros = 31.680 €.

Componente	**Coste/unidad (€)**	**Cantidad**	**Importe Total (€)**
Microcontrolador PIC18F258-I/SP	7,50	1	7,50
Integrado MCP2551-I/P	0,99	1	0,99
Integrado MAX3002EUP[1]	2,00	1	2,00
Integrado MAX883EPA[1]	2,61	1	2,61
Integrado MAX884EPA[1]	2,61	1	2,61
Zócalo FPS009-3200	8,99	1	8,99
Tarjeta de memoria[2]	18,00	1	18,00
Display LCD 2x16	11,30	1	11,30
Conector RJ45	0,52	2	1,04
LED circular 3 mm rojo	0,15	2	0,30
LED circular 3 mm verde	0,16	1	0,16
Zócalo pin torneado de 8	0,10	3	0,30
Zócalo pin torneado de 14	0,18	2	0,36
Cable plano	0,28/metro	0,5	0,14
Conector hembra de 36 contactos	1,63	1	1,63
Conector macho de 36 contactos	1,30	1	1,30
Cristal de 20 MHz	0,25	1	0,25
Condensador de disco de 33 pF	0,06	2	0,12
Condensador multicapa de 100 nF	0,06	7	0,42
Condensador electrolítico de 2,2μF	0,07	2	0,14
Resistencia 0,25 W del 5%	0,02	8	0,16
Conmutador de 2 posiciones	0,55	1	0,55
Jumper	0,03	1	0,03
Pulsador	0,09	1	0,09
Conector de pilas de 9 V	0,12	1	0,12
Caja de soporte	6,96	1	6,96
COSTE DEL MATERIAL DEL PROTOTIPO:			**72,07**

Tabla 7.1: Coste de los componentes empleados en el diseño del analizador.

7.1.3 Material y componentes del nodo universal

La tabla 7.2 presenta el coste de los distintos componentes utilizados en el diseño del prototipo del nodo universal.

[1]Estos componentes se venden en dólares estadounidenses. Hemos aplicado el factor de cambio de divisa correspondiente al 1 de Octubre del 2004: 1 dólar USA ($) = 0,8057 euros (€).

[2]Existe una gran variedad de marcas de tarjetas, cada una con sus propios precios. Éstos varían también en función de la capacidad de memoria y del tipo, MMC o SD. El valor introducido se corresponde con una MMC de 64 MB de una marca conocida.

Componente	**Coste/unidad (€)**	**Cantidad**	**Importe Total (€)**
Microcontrolador PIC18F258-I/SP	7,50	1	7,50
Integrado MCP2551-I/P	0,99	1	0,99
Regulador 7805	0,20	1	0,20
Diodo 1N4007	0,05	1	0,05
Conector RJ45	0,52	2	1,04
LED circular 3 mm verde	0,16	1	0,16
Zócalo pin torneado de 8	0,10	1	0,10
Zócalo pin torneado de 14	0,18	2	0,36
Terminales 2polos Paso 5,08	0,28	11	3,08
Conector macho de 36 contactos	1,30	1	1,30
Cristal de 20 MHz	0,25	1	0,25
Condensador de disco de 33 pF	0,06	3	0,18
Condensador multicapa de 100 nF	0,06	3	0,18
Condensador electrolítico de 100 μF	0,08	1	0,08
Resistencia 0,25 W del 5%	0,02	3	0,06
Jumper	0,03	3	0,09
Pulsador	0,09	1	0,09
COSTE DEL MATERIAL DEL PROTOTIPO:			**15,71**

Tabla 7.2: Coste de los componentes empleados en el diseño del nodo universal.

7.1.4 Coste humano del nodo universal

El tiempo necesario para la realización del nodo universal fue de 4 semanas (1 mes), con una media de 40 horas semanales. Seguimos considerando el mismo salario para los dos ingenieros de desarrollo:

Número de horas = 4 semanas × 40 horas/semana = 160 horas.
Coste por ingeniero = 160 horas × 18 €/hora = 2.880 €.
Coste humano total = 2.880 € × 2 ingenieros = 5.760 €.

7.2 Costes de producción del analizador y del nodo universal

En esta sección evaluamos los posibles costes en el caso de que el analizador y el nodo universal fueran comercializados. Existen una serie de gastos que no podemos cuantificar desde el punto de perspectiva de un ingeniero de prototipos. Como ejemplos tenemos el coste de la sala de producción (mantenimiento, posible alquiler, maquinaria, etc.), el transporte de las unidades fabricadas, gastos de marketing y publicidad (impresión en los catálogos de productos, visita a distribuidores y empresas directamente, publicidad en revistas del sector, etc.), entre otros.

Nosotros nos limitaremos al coste de cada unidad fabricada, aplicando posteriormente un porcentaje sobre el coste base para incluir estos factores. Hay que considerar por una parte el material empleado, incluyendo los descuentos que los proveedores realizan ante la venta de un número elevado de unidades. En segundo lugar se sitúan el coste de la mano de obra, esto es, el salario de los operarios necesarios para el montaje de las unidades que van a salir al mercado.

7.2.1 Costes del material

Consultando los catálogos de las distintas empresas proveedoras, consideramos conveniente establecer tres niveles de producción, cada uno de ellos con una media de descuento en el precio de los componentes:

- de 20 a 100 unidades: 20% de descuento[3].
- de 101 a 1000 unidades: 25% de descuento[3].
- por encima de 1000 unidades: 35% de descuento[3].

Con estos datos podemos estimar el coste por unidad del analizador en los tres casos comentados:

ANALIZADOR			
		Coste del prototipo:	**72,07 €**
Nivel de producción	20-100	101-1.000	Más de 1.000
Descuento	20%	30%	35%
Coste de material/unidad (€)	57,66	50,45	46,84
NODO UNIVERSAL			
		Coste del prototipo:	**15,71 €**
Nivel de producción	20-100	101-1.000	Más de 1.000
Descuento	20%	30%	35%
Coste de material/unidad (€)	12,57	11,00	10,21

Figura 7.1: Coste por unidad del material para el analizador y el nodo universal, para los tres niveles de producción estudiados.

[3]Sobre el coste calculado en la tabla 7.1.

7.2.2 Gasto de mano de obra

En este apartado vamos a contabilizar el coste humano por unidad producida, relativo a los operarios de montaje que trabajan en la cadena de producción. Estimamos que el tiempo necesario para montar el analizador o el nodo universal es de unos 45 minutos, en los que se incluyen el control de la soldadura de los distintos componentes y su integración en el soporte físico. Consideramos el salario adecuado para un operador cualificado en 9€/hora:

$$\text{Coste operarios por unidad} = 9\ €/\text{hora} \times 0{,}75 \text{ horas (45 minutos)} = 6{,}75\ €/\text{unidad}.$$

7.2.3 Coste total

Agrupemos ahora los distintos costes analizados para obtener el gasto total. Para aquellos factores que no hemos considerado (referidos algunos como ejemplos en la introducción de esta sección), suponemos una ampliación del 40% del coste total base. Además, hay que incluir el coste del desarrollo del prototipo, que está previsto amortizarlo en 2 años. Hemos supuesto 3 niveles de producción anuales: 100 unidades, 1.000 unidades y 10.000 unidades (que entran dentro de cada una de las categorías de descuentos en el material). A la cifra final, introducimos un 10% de beneficio, cantidad que estimamos oportuna en la fase de lanzamiento del producto. La tabla-resumen 7.2 muestra los resultados finales.

Nivel de producción anual (unidades)	100	1.000	10.000
ANALIZADOR			
Coste de material/unidad (€)	57,66	50,45	46,84
Coste operarios/unidad (€)	6,75	6,75	6,75
Coste total base/unidad (€)	**64,41**	**57,20**	**53,59**
Gastos extras (40% del coste base)/unidad (€)	25,76	22,88	21,44
Coste desarrollo/unidad (€)	158,40	15,84	1,59
COSTE TOTAL/unidad (€)	**248,57**	**95,92**	**76,61**
Beneficio impuesto: 10%			
PRECIO SIN I.V.A./unidad (€)	**273,43**	**105,51**	**84,27**
P.V.P.(16% de I.V.A.)/unidad (€)	**317,18**	**122,39**	**97,75**
NODO UNIVERSAL			
Coste de material/unidad (€)	12,57	11,00	10,21
Coste operarios/unidad (€)	6,75	6,75	6,75
Coste total base/unidad (€)	**19,32**	**17,75**	**16,96**
Gastos extras (40% del coste base)/unidad (€)	7,73	7,10	6,78
Coste desarrollo/unidad (€)	28,80	2,88	0,29
COSTE TOTAL/unidad (€)	**55,84**	**27,73**	**24,03**
Beneficio impuesto: 10%			
PRECIO SIN I.V.A./unidad (€)	**61,43**	**30,50**	**26,43**
P.V.P.(16% de I.V.A.)/unidad (€)	**71,26**	**35,38**	**30,66**

Figura 7.2: Evaluación del coste total por unidad y del precio de salida al mercado, para los tres niveles de producción estudiados.

Capítulo 8

Conclusiones

En este último capítulo de la memoria (a falta de los apéndices y la bibliografía) resaltamos las principales conclusiones obtenidas en nuestro Proyecto Fin de carrera.

- La multitud de bibliografía reciente encontrada sobre *CAN* bus demuestra que se trata de un tema tecnológico de interés actual. Además, hay sobradas pruebas de su apuesta en nuevos campos industriales, así como una mayor potenciación en su cuna de origen, el sector del automóvil.

- El interés en el desarrollo de un analizador de este protocolo es importante. Los diferentes sistemas *CAN* bus necesitan disponer de herramientas de análisis del tráfico, para detectar posibles anomalías en su funcionamiento.

- El analizador permite ser configurado de manera que sólamente admita tramas con identificador estándar, sólamente tramas con identificador extendido, o ambos tipos a la vez. También es posible discriminar mensajes gracias al sistema de máscaras y filtros. Por último posee dos modos de operación, normal o de sólo escucha.

- La captura de tráfico es almacenada en tarjetas de memoria MMC o SD. Estos dispositivos portátiles presentan una gran capacidad de memoria en un tamaño reducido, por lo que actualmente se están añadiendo a multitud de sistemas del tipo *data-logger*. Al usuario este sistema le proporciona evidentes ventajas, al evitarle estar conectado a un PC durante la captura.

- Las capturas son almacenadas en un archivo con estructura FAT16. Esto nos permite leer la información de la tarjeta en el PC de manera transparente, como un disco más. Una posible mejora es utilizar el sistema de archivos FAT32 para tarjetas por encima de los 32 MB, ya que se aprovecharía mejor la memoria física disponible al emplear *clusters* de menor tamaño.

- El analizador desarrollado presenta un comportamiento óptimo para sistemas *CAN* de intensidad de tráfico baja y media. Para estados de tráfico continuo puede presentar problemas, al no disponer el microcontrolador de tiempo suficiente para transferir los datos a la tarjeta.

- Para evitar el problema comentado en la conclusión anterior, está previsto en el diseño del analizador el cambio de reloj, con vistas a aumentar la velocidad de transferencia de datos a la tarjeta. No obstante, los distintos módulos así como ciertos cálculos realizados dependen de la frecuencia de oscilación del reloj del microcontrolador, por lo que el programa de éste debe ser modificado.

- Para poder configurar correctamente los parámetros temporales de captura del analizador se debe conocer la tasa de transmisión del sistema *CAN* objeto de estudio. Además, se debe tener un conocimiento básico de cómo genera *CAN* el tiempo de bit y la manera de sincronización entre nodos. Una mejora de este producto sería implementar un algoritmo de búsqueda automática de los parámetros de configuración temporal, valiéndonos del modo de sólo escucha del microcontrolador.

- El nodo universal es una forma de conectar a distancia una serie de periféricos sin necesidad de disponer de un software específico. La configuración de los distintos módulos, así como la solicitud o envío de un dato, es realizado por un nodo controlador mediante el bus de comunicación *CAN*. Para ello se ha diseñado un protocolo específico.

- El software del nodo universal está preparado para admitir microcontroladores PIC18F258 y PIC18F458. Este último tiene un mayor número de pines a los que poder conectar periféricos.

- El nodo universal se diseñó como una demostración práctica del sistema *CAN* bus. Se trata de una primera propuesta, con aspectos sensiblemente mejorables. Para futuros desarrollos, se considera conveniente mejorar los siguientes aspectos: optoacoplar las entradas/salidas del nodo, proporcionar la dirección MAC de manera externa, aprovechar mejor todas las prestaciones que ofrece el microcontrolador, o crear, igual que en el caso del analizador, un algoritmo para autoconfigurar los parámetros temporales.

- La amplia variedad de temas tratados en este proyecto ha supuesto para los autores una valiosa experiencia, así como un buena oportunidad de aplicar los conocimientos adquiridos durante el estudio de esta ingeniería.

Apéndice A

Código fuente del analizador

A.1 Fichero makefile

Listado A.1: Fichero makefile.mak.

```
proyecto=D:\PFC-CANBUS-NOBORRAR\pfc\Analizador
mcc18=D:\Archivos de programa\mcc18
cc=$(mcc18)\bin\c18demo.exe
linker=$(mcc18)\bin\mplink.exe
lib=$(mcc18)\lib
lkr=$(mcc18)\lkr
include=$(mcc18)\h
micro=18f258

all : \
  clean \
  link

link : analizador.o can18xx8.o xlcd.o mmc.o fat.o
  $(linker) /l$(lib) /k$(lkr) $(proyecto)\18f258i.lkr
  $(proyecto)\analizador.o $(proyecto)\can18xx8.o
  $(proyecto)\xlcd.o $(proyecto)\fat.o
  $(proyecto)\mmc.o /o analizador.cof

analizador.o :
  $(cc) -p=$(micro) $(proyecto)\analizador.c
  -fo=analizador.o /i
  $(include) -Ou+ -Ot- -Ob- -Op- -Or- -Od- -Opa+

can18xx8.o:
  $(cc) -p=$(micro) $(proyecto)\can18xx8.c
  -fo=can18xx8.o /i
  $(proyecto) -Ou+ -Ot- -Ob- -Op- -Or- -Od- -Opa+

xlcd.o:
  $(cc) -p=$(micro) $(proyecto)\xlcd.c
  -fo=xlcd.o /i
  $(proyecto) -Ou+ -Ot- -Ob- -Op- -Or- -Od- -Opa+

mmc.o:
  $(cc) -p=$(micro) $(proyecto)\mmc.c
  -fo=mmc.o /i
  $(proyecto) -Ou+ -Ot- -Ob- -Op- -Or- -Od- -Opa+

fat.o:
```

```
  $(cc) -p=$(micro) $(proyecto)\fat.c
  -fo=fat.o /i
  (proyecto) -Ou+ -Ot- -Ob- -Op- -Or- -Od- -Opa+

clean :
  del *.o
  del *.hex
  del *.cof
  del *.cod
  del *.lst
  del *.err
```

A.2 Fichero de linkado

Listado A.2: Fichero 18f258i.lkr.

```
// $Id: 18f258i.lkr,v 1.4 2003/03/13 05:02:22 sealep Exp $
// File: 18f258i.lkr
// Sample linker script for the PIC18F258 processor
// Fichero modificado para almacenar en ram dos buffers de 512 bytes
// La pila original se almacenaba desde 0x400 a 0x4FF, pero la hemos
// reducido un poco pasandola al banco gpr5 y pasando de size=0x100 (256)
// a size=0xF3 (243)


LIBPATH .

FILES c018i.o
FILES clib.lib
FILES p18f258.lib

CODEPAGE   NAME=vectors    START=0x0          END=0x29          PROTECTED
CODEPAGE   NAME=page       START=0x2A         END=0x7DBF
CODEPAGE   NAME=debug      START=0x7DC0       END=0x7FFF        PROTECTED
CODEPAGE   NAME=idlocs     START=0x200000     END=0x200007      PROTECTED
CODEPAGE   NAME=config     START=0x300000     END=0x30000D      PROTECTED
CODEPAGE   NAME=devid      START=0x3FFFFE     END=0x3FFFFF      PROTECTED
CODEPAGE   NAME=eedata     START=0xF00000     END=0xF000FF      PROTECTED

ACCESSBANK NAME=accessram  START=0x0          END=0x5F
DATABANK   NAME=gpr0       START=0x60         END=0xFF
DATABANK   NAME=BUFFERMMC  START=0x100        END=0x4FF
DATABANK   NAME=gpr5       START=0x500        END=0x5F3
DATABANK   NAME=dbgspr     START=0x5F4        END=0x5FF         PROTECTED
DATABANK   NAME=bankedsfr  START=0xF00        END=0xF5F         PROTECTED
ACCESSBANK NAME=accesssfr  START=0xF60        END=0xFFF         PROTECTED

SECTION    NAME=CONFIG     ROM=config

STACK     SIZE=0xF3   RAM=gpr5
```

A.3 Código fuente analizador

Listado A.3: Fichero analizador.c.

```
//-----------------------------------------------------------------------
//
//                            DESCRIPCION
//
// Analizador del protocolo CAN
//
//-----------------------------------------------------------------------
//
//    Archivo:       analizador.c
//
//    Autores:      Miguel Angel Perez
//                                  Carlos Gobernado
//
//      Version: 2.10   1-9-2004
//
//          - Comprobar inicializacion del indice i que se emplea en int
//          - Terminar el proceso de setup CAN desde la tarjeta
//      Version 2.10:
//          - Se ha modificado la óatencin a la óinterrupcin. Controla
//            si se produce un ocerflow en los buffers y controla mejor
//            los mensajes de error.
//      Version 2.00:
//          - Se ha eliminado el ADC, el control de bateria se realiza
//            por hardware.
//          - Esta version necesita mmc v2.00 y fat v1.11
//          - Se finaliza captura cuando sobrepasemos la capacidad de la tarjeta
//          - Se ha introducido la variable nCaptura que se incrementa cuando
//            recibimos una captura.
//          - Se ha perfecionado para que admita cualquier tamanio de cluster.
//          - Se finaliza la captura si desbordamos el campo destinado a
//            almacenar el tiempo.
//          - Se ha unificado el boton de on/off
//
//      Version 1.11:  6-Julio-2004
//          - Se ha incroporado un LED para indicar átrfico en el CAN.
//          - Se han corregido problemas del tiempo y del fichero.
//
//      Version 1.10   28 Junio 2004
//          - Las escritura solo esta hecha para cuando un cluster es un
//              sector. Hay que mejorarla.
//          - Hacer comprobacion de espacio en disco antes de capturar
//
//-----------------------------------------------------------------------

#include <p18cxxx.h>
#include <stdlib.h>
#include <timers.h>
#include <delays.h>
#include <usart.h>

#include "CAN18XX8.h"
#include "xlcd.h"
#include "string.h"
#include "mmc.h"
#include "fat.h"


//-----------------------------------------------------------------------
// Bits de configuracion del microcontrolador
//
// Proteccion de codigo: off
// Switch del oscilador (_OSCS_OFF_1H): OFF Tipo: HS
```

```
// Power-up timer: off
// Brown-out reset: off (_BORV_42_2L fija el valor de Brown-out=4,2)
// Watch-dog:off
// Lov Level programming: off
// Debug: off
// Reset si hay overflow in la pila (STVR_ON_4L): on
//-----------------------------------------------------------------------

#pragma romdata CONFIG
_CONFIG_DECL (  _OSCS_OFF_1H & _OSC_HS_1H,
        _PWRT_OFF_2L & _BOR_OFF_2L,
        _WDT_OFF_2H,
        _STVR_ON_4L & _LVP_OFF_4L & _DEBUG_OFF_4L,
        _CONFIG5L_DEFAULT,
        _CONFIG5H_DEFAULT,
        _CONFIG6L_DEFAULT,
        _CONFIG6H_DEFAULT,
        _CONFIG7L_DEFAULT,
        _CONFIG7H_DEFAULT);
#pragma romdata

//-----------------------------------------------------------------------
// Definicion de constantes
//-----------------------------------------------------------------------

#define VERSION "Version␣2.1" // Para el mensaje LCD

/* La estructura del fichero de configuracion es la siguiente:

    BRP(1byte),SJW(1byte),PropSeg(1byte),PhaseSeg1(1byte),
    PhaseSeg2(1byte),TIPO_FRAME_BUF1(1byte),TIPO_FRAME_BUF2(1byte),
    Mascara1(4bytes),F1_B1(4bytes),F2_B1(4bytes),Mascara2(4bytes),
    F1_B2(4bytes),F2_B2(4bytes),F3_B2(4bytes),F4_B2(4bytes),
    MODO_OPERACION(1byte),TIPO_ID(1byte)
*/

// La carga de datos se realizara en bufferCard1
#define BRP                         bufferCard1[0]
#define SJW                         bufferCard1[1]
#define PROPSEG                 bufferCard1[2]
#define PHSEG1                  bufferCard1[3]
#define PHSEG2                  bufferCard1[4]
#define TIPO_FRAME_BUF1 bufferCard1[5]
#define TIPO_FRAME_BUF2 bufferCard1[6]
#define MASK1                   *((ulong*)(bufferCard1+7))
#define FIL1_BUF1           *((ulong*)(bufferCard1+11))
#define FIL2_BUF1           *((ulong*)(bufferCard1+15))
#define MASK2                   *((ulong*)(bufferCard1+19))
#define FIL1_BUF2           *((ulong*)(bufferCard1+23))
#define FIL2_BUF2           *((ulong*)(bufferCard1+27))
#define FIL3_BUF2           *((ulong*)(bufferCard1+31))
#define FIL4_BUF2           *((ulong*)(bufferCard1+35))
#define MODO_OPERACION  bufferCard1[39]
#define TIPO_ID                 bufferCard1[40]

#define tamanioMask         0b00001111      // mascara para mezclar longitud e
                                                                    // instante de la trama
#define tipoMask                0b00011111      // mascara para mezclar tipo e ID
                                                                    // alta de la trama

/* Macros definidas para obtener el tipo de trama recibida */
#define CAN_STD                         0b00000000      // Trama estandar
#define CAN_EXT                         0b00100000      // Trama extendida
#define CAN_RTF                         0b01000000      // Trama remota
#define CAN_ERROR                       0b01100000      // Error en recepcion
                                                                    // Casos de overflow junto
```

```
#define CAN_OVERFLOW_STD        0b10000000      // con trama STD
#define CAN_OVERFLOW_EXT        0b10100000      // con trama XTD
#define CAN_OVERFLOW_RTF        0b11000000      // con trama remota
#define CAN_OVERFLOW_ERROR  0b11100000      // con órecepcin de un error

/* Nota: El analizador indica si se produce un overflow en un buffer y
         posteriormente extrae un mensaje STD o EXT o RTF o error
*/

#define COMIENZO_CONTADOR 0                 // Para inicializar contador de tiempo
#define LED_TRAFICO PORTAbits.RA5   // Inidicador LED para trafico CAN
#define TRIS_LED_TRAFICO TRISAbits.TRISA5
#define BOTON_OFF PORTBbits.RB0
#define BOTON_ON    PORTBbits.RB0

//Numero de cluster que posee la tarjeta susceptibles de llenar
#define TAMANIO_MAX (256*FAT.sectoresFAT)-2


//-----------------------------------------------------------------------
// Declaracion de variables globales
//-----------------------------------------------------------------------

typedef unsigned char BYTE;
typedef unsigned long ulong;
typedef unsigned int uint;
typedef unsigned char uchar;

// Variables tramas CAN
ulong NewMessage;                   //Campo ID de la trama CAN
BYTE NewMessageData[8];     //Campo de Datos de la trama CAN
BYTE NewMessageLen;             //Campo ñTamao de la trama CAN
enum CAN_RX_MSG_FLAGS NewMessageFlags;

// Buffers para usar con la tarjeta
#pragma udata BUFFERMMC=0x100
char bufferCard1[512];
char bufferCard2[512];
#pragma udata

// Ficheros de la tarjeta MMC
char nFichero[12]="CONFIG␣␣TXT";        // Nombre de fichero,
                      // "12345678ext"           // incluido extension
char nFichero2[12]="CAPTURA␣DAT";
extern struct structFAT FAT;
extern struct structFichero FICHERO;    // definidas en fat.h
extern near unsigned long    res,res1,arg1,arg2;

uint clusterInicio;     // Para abrir el fichero configuracion

uint cluster;

uchar bufferLleno=0;
uint tiempoH=0;         // Tiempo H y L se emplean para almacenar el
uint tiempoL=0;             // tiempo transcurrido en cada órecepcin.
uchar buffer=0;             // buffer=0 estamos escribiendo en el
                                        // bufferCard1
                                        // buffer=1 estamos escribiendo en el bufferCard2
uint i=0;                           // i es la variable que recorre los bufferCards
uchar tipoTrama=0;
uchar tamanioTrama=0;
uchar finCaptura=0;
ulong nCaptura=0;           // Es un contador que se incrementa con cada
                                        // captura y que nos permitira determinar el
                                        // ñtamao del fichero.
ulong tamanioFAT=0;
```

```
//-----------------------------------------------------------------------
// Prototipos de funciones
//-----------------------------------------------------------------------

// Configura el micro e inicializa variables
void Setup(void);
// Interrupcion de recepcion del CAN
void IntCanRx(void);
// Carga datos de configuracion de la tarjeta
uchar LecturaCard(void);
// Configura el ómdulo CAN
void SetupCAN(void);


//-----------------------------------------------------------------------
// Codigo de interrupciones
//-----------------------------------------------------------------------

#pragma code high_vector=0x08
void high_interrupt(void)
{
_asm
    goto IntCanRx
_endasm
}

#pragma code
#pragma interrupt IntCanRx save=PROD,CANCON,cluster

void IntCanRx(){

    PIE3=0;                                 // Deshabilitamos INT CAN
    INTCONbits.TMR0IE=0;        // Deshabilitamos INT de temporizador
    INTCONbits.INT0IE=0;        // y de RB0
    tiempoL=ReadTimer0();       // Leemos parte baja de la interrupcion

//**** INT debida a la ópulsacin de OFF ****
    if(INTCONbits.INT0IF){
        finCaptura=1;
        INTCONbits.INT0IF=0;
        return;
    }
//**** INT debida al desbordamiento del temporizador ****

    else if(INTCONbits.TMR0IF){
        tiempoH++;
        if (tiempoH==4096){         // Si se agota el tiempo total de captura
                                                        // finalizamos la misma
            INTCON=0;
            finCaptura=1;
            INTCONbits.TMR0IF=0;
            return;
        }
        else{
            INTCONbits.TMR0IF=0;
        }                                               // trama CAN en buffer0 o buffer1
    }

//**** INT debida a la órecepcin de una trama CAN ****

    else if(PIR3bits.RXB0IF || PIR3bits.RXB1IF||
            PIR3bits.IRXIF  || PIR3bits.ERRIF){

        /* Nota: He introducido el contador de captura, la variable i y el
```

```
            parpadeo del led dentro de cada caso porque se ípodra dar el caso
            de generar un error ERRIF=1 debido a que algun contador de error
            se disparara dando lugar a capturas fantasmas
        */
        if(buffer==0){  // buffer=0, entonces se escribe en el bufferCard1

            if(PIR3==0b10000000){
                //  Caso de error en la órecepcin
                LED_TRAFICO=!LED_TRAFICO;
                nCaptura++;                 // Incrementamos el contador capturas
                *((uint*)(bufferCard1+i))=tiempoL;      // Guardamos parte baja
                *((uint*)(bufferCard1+i+2))=tiempoH;    // y parte alta tiempo
                bufferCard1[i+15]=CAN_ERROR;                    // Guardamos tipo

                /* Nota: No ísera mala cosa poner en el buffer lo demas a 0 */

                i=i+16;                             // Incrementamos contador buffer
            }
            else if(CANIsRxReady()){
                // Estamos ante EXT STD o RTF
                nCaptura++;                     // Incrementamos el contador de capturas
                LED_TRAFICO=!LED_TRAFICO;
                CANReceiveMessage(&NewMessage,
                                  NewMessageData,
                                  &NewMessageLen,
                                  &NewMessageFlags);
                *((uint*)(bufferCard1+i))=tiempoL;      // Guardamos parte baja
                *((uint*)(bufferCard1+i+2))=tiempoH;    // y parte alta  tiempo

                bufferCard1[i+4]=NewMessageData[0];     // Leemos los datos de
                bufferCard1[i+5]=NewMessageData[1];     // trama recibida
                bufferCard1[i+6]=NewMessageData[2];
                bufferCard1[i+7]=NewMessageData[3];
                bufferCard1[i+8]=NewMessageData[4];
                bufferCard1[i+9]=NewMessageData[5];
                bufferCard1[i+10]=NewMessageData[6];
                bufferCard1[i+11]=NewMessageData[7];

                *((ulong*)(bufferCard1+i+12))=NewMessage;

                tamanioTrama=NewMessageLen;                                 // xxxxtamanio
                tamanioTrama=tamanioTrama<<4;                                       // tamanio0000
                bufferCard1[i+3]=bufferCard1[i+3]&tamanioMask;  // 0000tiempo
                bufferCard1[i+3]=bufferCard1[i+3]|tamanioTrama; // tamanio|tiempo

                bufferCard1[i+15] &= tipoMask;                                  // 000 || YYYYY
                if(NewMessageFlags&CAN_RX_OVERFLOW){
                    //  Caso de overflow en los buffers

                    if(NewMessageFlags&CAN_RX_RTR_FRAME)
                        bufferCard1[i+15] |= CAN_OVERFLOW_RTF;

                    else if(NewMessageFlags&CAN_RX_XTD_FRAME)
                        bufferCard1[i+15] |= CAN_OVERFLOW_EXT;

                    else if(NewMessageFlags&CAN_RX_INVALID_MSG)
                        bufferCard1[i+15] |= CAN_OVERFLOW_ERROR;
                    else
                        bufferCard1[i+15] |= CAN_OVERFLOW_STD;

                }
                else{
                    if(NewMessageFlags&CAN_RX_RTR_FRAME)
                        bufferCard1[i+15] |= CAN_RTF;

                    else if(NewMessageFlags&CAN_RX_XTD_FRAME)
```

```
                        bufferCard1[i+15] |= CAN_EXT;

                    else
                        bufferCard1[i+15] |= CAN_STD;
                }

                /* Mensajes para ódepuracin */

                i=i+16;

            } // Fin else de mesajes de datos recibidos

            //**** Evaluamos condicion de cambio de buffer ****
            if (i==512){
                    buffer=1;                   // Saltamos al buffer2
                    i=0;                            // Inicializamos contador buffer
                    bufferLleno=1;
            }
        }// fin de if(buffer==0)

        else{               // buffer=1, entonces se escribe en el bufferCard2

            if(PIR3==0b10000000){
                            //  Caso de error en la órecepcin
                nCaptura++;             // Incrementamos el contador de capturas
                LED_TRAFICO=!LED_TRAFICO;

                *((uint*)(bufferCard2+i))=tiempoL;      // Guardamos parte baja
                *((uint*)(bufferCard2+i+2))=tiempoH;    // y parte alta del tiempo
                bufferCard2[i+15]=CAN_ERROR;                    // Guardamos tipo
                /* Nota: No ísera mala cosa porner lo demas a 0 */

                i=i+16;                     // Incrementamos contador buffer

            }
            else if(CANIsRxReady()){
                // Estamos ante EXT STD o RTF
                nCaptura++;             // Incrementamos el contador de capturas
                LED_TRAFICO=!LED_TRAFICO;
                CANReceiveMessage(&NewMessage,
                                    NewMessageData,
                                    &NewMessageLen,
                                    &NewMessageFlags);
                *((uint*)(bufferCard2+i))=tiempoL;      // Guardamos parte baja
                *((uint*)(bufferCard2+i+2))=tiempoH;    // y parte alta del tiempo

                bufferCard2[i+4]=NewMessageData[0];     // Leemos los datos de la
                bufferCard2[i+5]=NewMessageData[1];     // trama recibida
                bufferCard2[i+6]=NewMessageData[2];
                bufferCard2[i+7]=NewMessageData[3];
                bufferCard2[i+8]=NewMessageData[4];
                bufferCard2[i+9]=NewMessageData[5];
                bufferCard2[i+10]=NewMessageData[6];
                bufferCard2[i+11]=NewMessageData[7];

                *((ulong*)(bufferCard2+i+12))=NewMessage;

                tamanioTrama=NewMessageLen;                             // xxxxtamanio
                tamanioTrama=tamanioTrama<<4;                                   // tamanio0000
                bufferCard2[i+3]=bufferCard2[i+3]&tamanioMask;  // 0000tiempo
                bufferCard2[i+3]=bufferCard2[i+3]|tamanioTrama; // tamanio|tiempo

                bufferCard2[i+15] &= tipoMask;                              // 000 || YYYYY
                if(NewMessageFlags&CAN_RX_OVERFLOW){
                    //  Caso de overflow en los buffers
```

```
                    //Printrs("Overflow");
                    if(NewMessageFlags&CAN_RX_RTR_FRAME)
                    bufferCard2[i+15] |= CAN_OVERFLOW_RTF;

                    else if(NewMessageFlags&CAN_RX_XTD_FRAME)
                        bufferCard2[i+15] |= CAN_OVERFLOW_EXT;

                    else if(NewMessageFlags&CAN_RX_INVALID_MSG)
                        bufferCard2[i+15] |= CAN_OVERFLOW_ERROR;
                    else
                        bufferCard2[i+15] |= CAN_OVERFLOW_STD;

                }
                else{
                    if(NewMessageFlags&CAN_RX_RTR_FRAME)
                        bufferCard2[i+15] |= CAN_RTF;

                    else if(NewMessageFlags&CAN_RX_XTD_FRAME)
                        bufferCard2[i+15] |= CAN_EXT;

                    else
                        bufferCard2[i+15] |= CAN_STD;
                }


                i=i+16;                                     // Incrementamos contador buffer
            }

            //**** Evaluamos ócondicin de cambio de buffer ****
            if (i==512){
                    buffer=0;                               // Saltamos al buffer1
                    i=0;                                        // Inicializamos contador de buffer
                    bufferLleno=1;                  // para empezar a escribir en la MMC
            }
        }

        if(FICHERO.numCluster==tamanioFAT){
                                        // Inserto aqui porque necesito un sitio donde
                                        // no puedan aparecer interrupciones, sino
                                        // daba problemas puntuales.
            INTCON=0;
            finCaptura=1;
            return;
        }
        //LED_TRAFICO=!LED_TRAFICO;
    }


    PIR3=0b00000000;            // Limpiamos flags rx CAN en buffer0 o buffer1
    PIE3=0b10100011;            // Activamos INT CAN
    INTCONbits.TMR0IE=1;    // Habilitamos INT de temporizador y RB0
    INTCONbits.INT0IE=1;

} // Fin IntCanRx

//-----------------------------------------------------------------------
// Implementacion de funciones
//-----------------------------------------------------------------------

//-----------------------------------------------------------------------
// Setup()
// La mision de esta funcion es la de configurar el micro e inizializar
// las variables declaradas
//-----------------------------------------------------------------------

void Setup(void)
```

```
{
// ***** PUERTOS DEL PIC *****

    TRISA=0b11000001;       // RA1-RA2-RA3 Control del LCD SALIDAS
                                            // RA5 LED de control de átrfico CAN
    ADCON1=0b00000111;  // Configuro PORTA como digital
    TRISB=0b00001011;       // RB0 Boton de ON/OFF ENTRADA
                                            // RB2 CANTX          SALIDA
                                            // RB3 CANRX          ENTRADA
                                            // RB4-RB5-RB6-RB7 Entrada de datos del LCD
                                            // (datos altos) SALIDAS

    TRIS_EN_VCC=0;          // Patilla de habilitacion de la alimentacion
                                            // de la tarjeta
    EN_VCC=0;
    LATB=0;

    TRIS_LED_TRAFICO=0; // INDICADOR LED
    LED_TRAFICO=1;

// ***** LCD *****
    Delay10KTCYx(250);
    OpenXLCD(FOUR_BIT&LINES_5X7);
    while ( BusyXLCD() );
    WriteCmdXLCD(BLINK_OFF);
    while ( BusyXLCD() );
    WriteCmdXLCD(CURSOR_OFF);


// ***** Temporizador *****
    OpenTimer0( TIMER_INT_OFF&
                    T0_16BIT&
                    T0_SOURCE_INT&
                    T0_PS_1_16);
// ***** INTERRUPCIONES *****

    RCONbits.IPEN=1;        // Habilito INT de dos niveles
    INTCON2=0b10111111; // INT TMR0 en alta prioridad.  INT0 flanco bajada
                                            // La INT externa INT0 va asociada  alta prioridad

}

void SetupCAN(void){

// ***** Inicializamos ómdulo CAN *****

    CANInitialize(SJW, BRP, PHSEG1, PHSEG2, PROPSEG,
                    TIPO_ID&CAN_CONFIG_DBL_BUFFER_ON);



    // Set CAN module into configuration mode
    CANSetOperationMode(CAN_OP_MODE_CONFIG);
    // ÁMSCARAS Y FILTROS
    // Buffer 1 Valor ámscara
    CANSetMask(CAN_MASK_B1,MASK1,TIPO_FRAME_BUF1);
    // Buffer 2 Valor ámscara
    CANSetMask(CAN_MASK_B2,MASK2,TIPO_FRAME_BUF2);
    // Filtro1 Buffer1
    CANSetFilter(CAN_FILTER_B1_F1,FIL1_BUF1,TIPO_FRAME_BUF1);
    // Filtro2 Buffer1
    CANSetFilter(CAN_FILTER_B1_F2,FIL2_BUF1,TIPO_FRAME_BUF1);
    // Filtro1 Buffer2
    CANSetFilter(CAN_FILTER_B2_F1,FIL1_BUF2,TIPO_FRAME_BUF2);
    // Filtro2 Buffer2
    CANSetFilter(CAN_FILTER_B2_F2,FIL2_BUF2,TIPO_FRAME_BUF2);
```

```
    // Filtro3 Buffer2
    CANSetFilter(CAN_FILTER_B2_F3,FIL3_BUF2,TIPO_FRAME_BUF2);
    // Filtro4 Buffer2
    CANSetFilter(CAN_FILTER_B2_F4,FIL4_BUF2,TIPO_FRAME_BUF2);

    // CAN en modo de escucha, no realiza ACK, las tramas con error son
    // cargadas en el buffer y no funcionan los filtros
    // CAN en modo normal funcionan los filtros pero realiza ACK

    CANSetOperationMode(MODO_OPERACION);

    // ***** INTERRUPCIONES CAN *****


    IPR3=0b10100011;        // INT órecepcin trama CAN en alta prioridad
}

//-----------------------------------------------------------------------
// implementacion de retardos en funcion del cristal para xlcd
// definidas en xlcd.h
//-----------------------------------------------------------------------

void DelayFor18TCY( void )                  //segun datasheet microchip C18
{
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
}

void DelayXLCD( void )
{
    Delay1KTCYx(25);            //Delay of 5ms para un reloj 20Mhz
    return;
}
void DelayPORXLCD( void )
{
    Delay1KTCYx(75);            //Delay of 15ms para un reloj de 20Mhz
    return;
}

//-----------------------------------------------------------------------
// uchar LecturaCard()
// La mision de esta funcion es la de leer el fichero de configuracion
// CONFIG.TXT y cargar los datos necesarios.
//-----------------------------------------------------------------------

uchar LecturaCard(void){
    // *** Lectura del fichero CONFIG de la tarjeta MMC ***
    TRISCbits.TRISC1=1;     // Patilla de deteccion de tarjeta
    while(PORTCbits.RC1==0){
        while ( BusyXLCD() );
        WriteCmdXLCD(CLEAR);
        while ( BusyXLCD() );
        putrsXLCD("Inserte␣tarjeta");
        Delay10KTCYx(250);
```

```
        Delay10KTCYx(250);  // retardo un segundo
    }
    Delay10KTCYx(250);
    Delay10KTCYx(250);
    CardPowerOn();

    // Inicializo tarjeta en modo SPI
    while(InitMMC()==ERROR){
        while ( BusyXLCD() );
        WriteCmdXLCD(CLEAR);
        while ( BusyXLCD() );
        putrsXLCD("Error:␣initCARD");
        Delay10KTCYx(250);
        Delay10KTCYx(250);
    }

    // Inicializo estructura FAT
    InitFAT(bufferCard2);

    tamanioFAT=TAMANIO_MAX;
    // Lectura del fichero de configuracion
    clusterInicio=AbrirFicheroLectura(nFichero,bufferCard2);
    if(clusterInicio!=NO_FIND){
            cluster=LeerFichero(bufferCard1,clusterInicio,1);
            // En bufferCard1 tengo las variables para el CAN

            // Compruebo que esta el fichero de captura CAPTURA.dat
            clusterInicio=AbrirFicheroLectura(nFichero2,bufferCard2);
            if(clusterInicio==0)    // Fichero totalmente vacio
                clusterInicio==3;       // Por la forma de generar los ficheros
                                        // Captura.dat siempre ídebera empezar
                                        // en el cluster numero 3
            if(clusterInicio!=NO_FIND){
                // Inicializo los campos de la estructura FICHERO
                FICHERO.clusterInicio=clusterInicio;
                FICHERO.tamanio=0;
                FICHERO.sectCluster=1;
                FICHERO.numCluster=0;
                memcpy(FICHERO.nombre,nFichero2,11);
                RETORNO_CARRO();
            }
            else{
                while ( BusyXLCD() );
                WriteCmdXLCD(CLEAR);
                while ( BusyXLCD() );
                putrsXLCD("Error:");
                while ( BusyXLCD() );
                SetDDRamAddr(LINE2);
                while ( BusyXLCD() );
                putrsXLCD("CAPTURA.TXT");
                Delay10KTCYx(250);
                Delay10KTCYx(250);
                Delay10KTCYx(250);
                Delay10KTCYx(250);  // retardo 2 segundos aprox
                return 0;
            }
        }
        else{
            while ( BusyXLCD() );
            WriteCmdXLCD(CLEAR);
            while ( BusyXLCD() );
            putrsXLCD("Error:");
            while ( BusyXLCD() );
            SetDDRamAddr(LINE2);
            while ( BusyXLCD() );
            putrsXLCD("CONFIG.TXT");
```

```
            Delay10KTCYx(250);
            Delay10KTCYx(250);
            Delay10KTCYx(250);
            Delay10KTCYx(250);  // retardo 2 segundos aprox
            return 0;
        }
}

//-----------------------------------------------------------------------
// main()
//-----------------------------------------------------------------------

void main(void)
{
// *** Configuracion del LCD, Timer, Puertos, INT ***
    Setup();
    LED_TRAFICO=0;
    while ( BusyXLCD() );
    WriteCmdXLCD(CLEAR);
    while ( BusyXLCD() );
    putrsXLCD("Analizador␣CAN");
    while ( BusyXLCD() );
    SetDDRamAddr(LINE2);            //Coloco en la segunda linea 64=0x40
    while ( BusyXLCD() );
    putrsXLCD( VERSION );
    Delay10KTCYx(250);
    Delay10KTCYx(250);
    Delay10KTCYx(250);
    Delay10KTCYx(250);
    Delay10KTCYx(250);
    Delay10KTCYx(250);          //Obtenemos un retraso de 3s para 20MHz
    while ( BusyXLCD() );
    SetDDRamAddr(LINE2);
    putrsXLCD( "Gober-Perez" );
    Delay10KTCYx(250);
    Delay10KTCYx(250);
    Delay10KTCYx(250);
    Delay10KTCYx(250);
    Delay10KTCYx(250);
    Delay10KTCYx(250);          //Obtenemos un retraso de 3s para 20MHz

    //Cargamos los datos del fichero CONFIG.TXT
    while(!LecturaCard());

    while ( BusyXLCD() );
    WriteCmdXLCD(CLEAR);
    while ( BusyXLCD() );
    putrsXLCD("Configuracion");
    while ( BusyXLCD() );
    SetDDRamAddr(LINE2);
    putrsXLCD("correcta");
    Delay10KTCYx(250);
    Delay10KTCYx(250);
    Delay10KTCYx(250);
    Delay10KTCYx(250);
    Delay10KTCYx(250);
    Delay10KTCYx(250);          //Obtenemos un retraso de 3s para 20MHz


    while(1){
    // *** Condicion de inicio del proceso analizador ***
        nCaptura=0;
        while ( BusyXLCD() );
        WriteCmdXLCD(CLEAR);
        while ( BusyXLCD() );
        putrsXLCD("Esperando␣ON");
```

```
        while(BOTON_ON==1);
        while(BOTON_ON==0);

        // *** Configuracion Nodo CAN ***
        SetupCAN();

        while ( BusyXLCD() );
730     WriteCmdXLCD(CLEAR);
        while ( BusyXLCD() );
        putrsXLCD("Capturando");

        i=0;
        PIR3=0b00000000;        // Limpiamos flags recepcion CAN en buffer0
                                                    // o buffer1
        PIE3=0b10100011;            // Habilito INT CAN
                                                    // Inicializamos temporizador
        WriteTimer0( COMIENZO_CONTADOR );
740     INTCONbits.INT0IF=0;    // Limpio el flag para que no salte INT
        INTCON=0b11110000;      // Habilito INT temporizador TMR0 y
                                                    // la INT externa
                                                    // INT0 (pin RB0) y las INT totales

    // *** BUCLE PRINCIPAL ***
        finCaptura=0;
        cluster=FICHERO.clusterInicio;
        while(1){
            if (bufferLleno==1){  // Evaluacion de escritura si algun buffer
750                                                 // áest lleno
                if(buffer==0){          // Selecionado el buffer 1 y lleno el 2
                    bufferLleno=0;
                    cluster=EscribirFichero(bufferCard2,cluster,
                                                    FICHERO.sectCluster);
                }
                else{
                    bufferLleno=0;
                    cluster=EscribirFichero(bufferCard1,cluster,
                                                  FICHERO.sectCluster);
760             }
            }
            if(finCaptura==1){
                INTCON=0;
                LED_TRAFICO=0;
                while ( BusyXLCD() );
                WriteCmdXLCD(CLEAR);
                while ( BusyXLCD() );
                putrsXLCD("Fin␣captura");
                while ( BusyXLCD() );
770             SetDDRamAddr(LINE2);
                while ( BusyXLCD() );   //Coloco en la segunda linea 64=0x40
                putrsXLCD("Cerrando␣archivo");
                if(i!=512){
                    for(i;i<512;i++){
                        if(buffer==0)
                            bufferCard1[i]=0;
                        else
                            bufferCard2[i]=0;
                    }
780                 if(buffer==0){          // Esta selecionado el buffer 1
                        cluster=EscribirFichero(bufferCard1,cluster,
                                                    FICHERO.sectCluster);
                    }
                    else{
                        PrintInt("Escribo␣cluster",cluster);
                        cluster=EscribirFichero(bufferCard2,cluster,
                                                    FICHERO.sectCluster);
                    }
```

```
                    }
790                 // Empleo Mul32x32 para evitar posibles desbordamientos
                    Mul32x32(nCaptura,16);  // Multiplico cada captura por 16bytes
                                                // resultado en res
                    FICHERO.tamanio=res;
                    if(nCaptura!=0)
                        CerrarFicheroEscritura(bufferCard1,FICHERO_AGREGAR);
                    CardPowerOff();
                    while ( BusyXLCD() );
                    WriteCmdXLCD(CLEAR);
                    while ( BusyXLCD() );
800                 putrsXLCD("Fichero␣cerrado");
                    while ( BusyXLCD() );
                    SetDDRamAddr(LINE2);
                    while ( BusyXLCD() );           //Coloco en la segunda linea 64=0x40
                    putrsXLCD("Extraiga␣tarjeta");
                    finCaptura=0;
                    Delay10KTCYx(250);
                    Delay10KTCYx(250);
                    Printrs("FIN");
                    while(BOTON_ON==1);
810                     break;
                }
            }
            //Cargamos los datos del fichero CONFIG.TXT
            while(!LecturaCard());

            while ( BusyXLCD() );
            WriteCmdXLCD(CLEAR);
            while ( BusyXLCD() );
            putrsXLCD("Configuracion");
820         while ( BusyXLCD() );
            SetDDRamAddr(LINE2);
            putrsXLCD("correcta");
            Delay10KTCYx(250);
            Delay10KTCYx(250);
            Delay10KTCYx(250);
            Delay10KTCYx(250);
            Delay10KTCYx(250);
            Delay10KTCYx(250);          //Obtenemos un retraso de 3s para 20MHz
        }
830 }
```

A.4 Código fuente librería MMC

Listado A.4: Fichero mmc.h.

```
//-----------------------------------------------------------------------
//
//                          DESCRIPCION
//
// Libreria para utilidar tarjetas MMC card con el protocolo SPI
//
//-----------------------------------------------------------------------
//
//    Archivo:              mmc.h
//
//    Autores:      Miguel Angel Perez
//                                  Carlos Gobernado
//
//    Version: 2.11 1-9-2004
//
//
// Las tarjetas MMC trabajan en el modo CKP=1 CKE=0 luego MODO 11
// y se muestrea en la mitad del bit
//
// Version 2.11:
//      Se ha procedido a una limpieza y revision de codigo.
//
// Version 2.10:
//      Se ha introducido las modificaciones necesarias para trabajar
//      con SD-CARDs.
//
//  Version 2.00:
//      Ante los problemas puntuales con distintos modelos de tarjetas
//      se ha decidido hacer una revision en profundidad de este codigo.
//      Se ha introducido pin de habilitacion de alimentacion de tarjeta y
//      funciones correspondientes.
//      EnviaComandoMMC se ha introducido un retardo de 8 ciclos antes de
//      devolver la respuesta.
//      Por las pruebas realizadas este es un codigo mas robusto.
//
//  Version 1.11:
//      Se ha introducido las macros SPI_CS_ON() y SPI_CS_OFF()
//      Se han introducido casi todos los comandos para una posible
//      ampliacion.
//      La habilitacion CS se levanta y se baja en los comandos
//      La funcion Borrar512 ya funciona.
//
//  Version 1.10:
//      La funcion Borrar512 no he conseguido hacerla funcionar y no se pq
//      No funcionan con tarjetas SD CARD
//
//-----------------------------------------------------------------------

#ifndef __MMC_H
#define __MMC_H

//-----------------------------------------------------------------------
// Definicion de constantes
//-----------------------------------------------------------------------

#define ERROR       0
#define OK          1

//-----------------------------------------------------------------------
//  Pines utilizados para habilitacion y senial de alimentacion
//-----------------------------------------------------------------------
```

```
#define SPI_CS              PORTCbits.RC2
#define EN_VCC              PORTCbits.RC0
#define TRIS_EN_VCC         TRISCbits.TRISC0
#define TRIS_CS             TRISCbits.TRISC2

//-----------------------------------------------------------------------
// Definicion de comandos MMC/SD
//-----------------------------------------------------------------------

#define GO_IDLE_STATE               0x40
#define SEND_OP_COND                0x41
#define SEND_CSD                    0x49
#define SEND_CID                    0x4A
#define STOP_TRANSMISSION           0x4C
#define SEND_STATUS                 0X4D
#define SET_BLOCKLEN                0x50
#define READ_SINGLE_BLOCK           0x51
#define READ_MULTIPLE_BLOCK         0x52
#define SET_BLOCK_COUNT             0x57
#define WRITE_SINGLE_BLOCK          0x58
#define WRITE_MULTIPLE_BLOCK        0x59
#define PROGRAM_CSD                 0x5B
#define SET_WRITE_PORT              0x5C
#define CLR_WRITE_PORT              0x5D
#define SEND_WRITE_PROT             0x5E
#define TAG_ERASE_START             0x63
#define TAG_ERASE_END               0x64
#define ERASE                       0x66
#define LOCKUNLOCK                  0x6A
#define APP_CMD                     0x77
#define GEN_CMD                     0x78
#define READ_OCR                    0x7A
#define CRC_ON_OFF                  0x7B
#define SD_SEND_OP_COND             0x41

//-----------------------------------------------------------------------
// Definicion de respuestas a comandos
//-----------------------------------------------------------------------
#define R1_OK                       0b00000000
#define R1_IDLE_STATE               0b00000001
#define R1_ERASE_RESET              0b00000010
#define R1_ILLEGAL_COMMAND          0b00000100
#define R1_CRC_ERROR                0b00001000
#define R1_ERARSE_SEQUENCE_ERROR    0b00010000
#define R1_ADDRESS_ERROR            0b00100000
#define R1_PARAMETER_ERROR          0b01000000

//-----------------------------------------------------------------------
// Definicion de macros
//-----------------------------------------------------------------------
#define SPI_CS_ON()         SPI_CS=0
#define SPI_CS_OFF()        SPI_CS=1
#define CardPowerOff()      EN_VCC=0;
#define INTENTOS            50          // Es el numero de intentos para escritura
                                        // y lectura

//-----------------------------------------------------------------------
//Estructuras de datos
//-----------------------------------------------------------------------
typedef unsigned char BYTE;
typedef unsigned char byte;
typedef unsigned char uchar;
typedef unsigned long ulong;

//-----------------------------------------------------------------------
// Prototipos de funciones
```

```
//-----------------------------------------------------------------------
130
char InitMMC(void);
void CardPowerOn(void);
/*
void CardPowerOff(); es una macro ya que solo modifica una pata
*/
char EnviaComandoMMC(char,ulong,char);
char DatoSPI(char);
char Lectura512(char*,ulong);
char Escritura512(char*,ulong);
140 char Borrar512(ulong,ulong);
void CardDelay(uchar);

//char IdMMC(char*);
//char CsdMMC(char*);
//int  CapacidadMMC();
//char ModeloMMC(char *);
```

Listado A.5: Fichero mmc.c.

```
//-----------------------------------------------------------------------
//
//                            DESCRIPCION
//
// Libreria para utilidar tarjetas MMC card con el protocolo SPI
//
//-----------------------------------------------------------------------
//
//    Archivo:              mmc.c
//
//    Autores:      Miguel Angel Perez
//                                  Carlos Gobernado
//
//    Version: 2.11 1-9-2004
//
//
// Las tarjetas MMC trabajan en el modo CKP=1 CKE=0 luego MODO 11
// y se muestrea en la mitad del bit
//
// Version 2.11:
//      Se ha procedido a una limpieza y revision de codigo.
//
// Version 2.10:
//      Se han introducido las modificaciones necesarias para trabajar
//      con SD-CARDs.
//
//  Version 2.00:
//      Ante los problemas puntuales con distintos modelos de tarjetas
//      se ha decidido hacer una órevisin en profundidad de este ócdigo.
//      Se ha ñaadido pin de óhabilitacin de óalimentacin de tarjeta y
//      funciones correspondientes.
//      EnviaComandoMMC se ha ñaadido un retardo de 8 ciclos antes de
//      devolver la respuesta.
//      Por las pruebas realizadas este es un ócdigo áms robusto.
//
//  Version 1.11:
//      Se ha introducido las macros SPI_CS_ON() y SPI_CS_OFF()
//      Se han introducido casi todos los comandos para una posible
// ó     ampliacin.
//      La habilitacion CS se levanta y se baja en los comandos
//      La funcion Borrar512 ya funciona.
//
//  Version 1.10:
//      La funcion Borrar512 no he conseguido hacerla funcionar y no se pq
//      No funcionan con tarjetas SD CARD
//
//-----------------------------------------------------------------------

#include <spi.h>
#include <p18cxxx.h>
#include <stdlib.h>
#include "mmc.h"
#include "debug.h"

//-----------------------------------------------------------------------
// void CardPowerOn()
//
// Esta funcion se encargar de habilitar la alimentacion de la tarjeta
// y introducir un ñpequeo retardo para dar tiempo a que se estabilice
// los 3,3V.
//
// Nota: Esta funcion esta pensada para cuando EN_VCC=1 se alimenta
// la tarjeta. Esta pensado para usarse con un regulador de 3.3V (MAX884)
// que tiene una patilla de habilitacion de la alimentacion.
// Esta funcion como CardPowerOff deben de ser utilizadas por el
```

```
// usuario.
// NOTA2: Previamente hay que configurar como salida la pata
// correspondiente TRIS_EN_VCC=0;
//
//-----------------------------------------------------------------------

void CardPowerOn(void){
    int i;
    EN_VCC=1;
    // introduzco un retardo por seguridad
    for(i=0;i<1000;i++);
}

//-----------------------------------------------------------------------
// void CardPowerOff()
//
// Esta funcion interrumpe la óalimentacin de la tarjeta
//
// Nota: Esta definida como una macro por eso áest comentada
//
//-----------------------------------------------------------------------

/*
void CardPowerOff(){
    EN_VCC=0;
}
*/

//-----------------------------------------------------------------------
// char InitMMC()
//
// Esta funcion se encargar de:
// º 1 Inicializar patillas correspondientes
// º 2 Inicializar el puerto SPI del micro
// º 3 Realizar la secuencia de comandos para inicializar la
//     tarjeta en modo SPI
//
// Retorno:
//  1 Si todo ha ido bien
//      0 Error
//-----------------------------------------------------------------------

char InitMMC(void){

    unsigned char i;    // indice bucle
                            // (char para ocupar menos en pila)
    TRIS_CS=0;          // Patilla CS como salida
    SPI_CS=1;           // MMC-Deshabilitada SPI_CS_OFF
    TRIS_CS=0;          // Configuramos la patilla seleccionada
                            // como salida

    // Inicializacion del puerto SPI
    OpenSPI(SPI_FOSC_4,MODE_11,SMPMID);

    for(i=0;i<10;i++)
        DatoSPI(0xFF);  // 10*8=80 ciclos de reloj
    SPI_CS_ON();        // Habilitamos la tarjeta

    // Mandamos GO_IDLE_STATE haber si esta la tarjeta
    if(EnviaComandoMMC(GO_IDLE_STATE,0,0x95)!=1){
        SPI_CS_OFF();
        return ERROR;   // Algo ha ido mal
    }
    SPI_CS_OFF();
    DatoSPI(0xFF);      // Retardo de 8 ciclos
```

```
    // Puede que sea una SD CARD en ese caso necesito mandar el
    // comando ACMD41 (mandar CMD55 y luego el CMD41)
    // para saber si es SD debe de reconocer CMD55

    SPI_CS_ON();
    temp=EnviaComandoMMC(APP_CMD,0,0xFF);
    SPI_CS_OFF();
    if(temp==1){
        // Caso SD-card
        Printrs("Tarjeta␣tipo␣SD");
        SPI_CS_ON();
        i=0;
        do{
            temp=EnviaComandoMMC(APP_CMD,0,0xFF);
            PrintChar("Respuesta␣APP_CMD",temp);
            temp=EnviaComandoMMC(SD_SEND_OP_COND,0,0xFF);
            PrintChar("Respuesta␣SD_SEND_OP_COND",temp);
            i++;
        }
        while(temp!=0&&i<255);
        SPI_CS_OFF();
        if(i==INTENTOS)
            return ERROR;   // Algo ha ido mal
        else
            return OK;
    }
    else{
        Printrs("Tarjeta␣tipo␣MMC");
        for(i=0;i<255;i++){
        SPI_CS_ON();
        if(EnviaComandoMMC(SEND_OP_COND,0,0xFF)==0){
            SPI_CS_OFF();
            return OK;      // OK MMC Inicializado
        }
        SPI_CS_OFF();
        DatoSPI(0xFF);      // Retardo de 8 ciclos
    }
    return ERROR;
    }
}

//-----------------------------------------------------------------------
// EnviaComandoMMC(char cmd, int adr,char crc)
//
// Esta funcion se encargar de enviar comandos y esperar una respuesta
// de tipo R1
//
//
//
//-----------------------------------------------------------------------

char EnviaComandoMMC(char cmd, ulong adr,char crc){

// Este ócdigo funciona bien
    char *ps;
    char retorno;
    ps=(char*)&adr;
    DatoSPI(0xFF);
    DatoSPI(cmd);
    DatoSPI(*(ps+3));   //Mando 4byte de adr
    DatoSPI(*(ps+2));   //Mando 3byte de adr
    DatoSPI(*(ps+1));   //Parte 2byte de adr
    DatoSPI(*ps);       //Parte 1byte de adr el menos significativo
    DatoSPI(crc);
    DatoSPI(0xFF);
    retorno=DatoSPI(0xFF);
```

```
    DatoSPI(0xFF);      // retardo de 8 ciclos de reloj
                        // introducido version 2.00
    return retorno;
}

//-----------------------------------------------------------------------
// char DatoSPI(char byte)
//
// Esta funcion se encargar de enviar un dato por el puerto SPI y devuelve
// el dato recibido
//
// Retorno:
//  char dato recibido
//-----------------------------------------------------------------------
char DatoSPI(char byte){
    SSPBUF=byte;        // coloco dato de salida
    while(!SSPSTATbits.BF); // espero a tener dato en la entrada
    return SSPBUF;
}


//-----------------------------------------------------------------------
// char Lectura512(char* buffer,ulong adr)
//
// Realiza una lectura de 512 y lo almacena en el buffer.Si se
// produce un error al mandar el commando reintenta el valor definido en
// INTENTOS y produce un error llegado el caso.
//
// Nota: No se controla el token que devuelve la tarjeta al final de
// recibir todos los datos.
//
// Retorno:
//  1 Si todo ha ido bien
//      0 Error
//-----------------------------------------------------------------------
char Lectura512(char* buffer,ulong adr){
    uchar i;                        // indice de intentos
    //char temp;                    // variable necesaria para depurar
    adr&=0xFFFFFE00;                // La direcciones tiene que ser multiplo de 512
    for(i=0;i<INTENTOS;i++){
        SPI_CS_ON();
        //temp=EnviaComandoMMC(READ_SINGLE_BLOCK,adr,0xFF);
        //PrintChar("Lectura512->Respuesta READ_SINGLE_BLOCK",temp);
        if(EnviaComandoMMC(READ_SINGLE_BLOCK,adr,0xFF)==R1_OK)
            break;
        else{
            SPI_CS_OFF();
            DatoSPI(0xFF);  // Retardo de 8 ciclos
        }
    }
    if(i==INTENTOS)
        return ERROR;

    while(DatoSPI(0xFF)!=0xFE);
    for (i=0;i<512;i++)
        *(buffer+i)=DatoSPI(0xFF);
    DatoSPI(0xFF);
    DatoSPI(0xFF);
    SPI_CS_OFF();
    return OK;
}

//-----------------------------------------------------------------------
// char Escritura512(char* buffer,ulong adr)
//
// Realiza una escritura de 512 y lo almacena en el buffer. Si se
```

```
// produce un error al mandar el commando reintenta el valor definido en
// INTENTOS y produce un error llegado el caso.
//
// Nota: No se controla el token que devuelve la tarjeta al final de
// enviar todos los datos.
//
// Retorno:
//  1 Si todo ha ido bien
//  0 Error
//-----------------------------------------------------------------------

char Escritura512(char* buffer,ulong adr){
    uchar i;
    //char temp;                    // variable necesaria para depurar

    adr&=0xFFFFFE00; // La direcciones tiene que ser multiplo de 512
    for(i=0;i<INTENTOS;i++){
        SPI_CS_ON();
        //temp=EnviaComandoMMC(WRITE_SINGLE_BLOCK,adr,0xFF);
        //PrintChar("Escritura512->Respuesta WRITE_SINGLE_BLOCK",temp);
        if(EnviaComandoMMC(WRITE_SINGLE_BLOCK,adr,0xFF)==R1_OK)
            break;
        else{
            SPI_CS_OFF();
            DatoSPI(0xFF);  // Retardo de 8 ciclos
        }
    }
    if(i==INTENTOS)
        return ERROR;

    DatoSPI(0xFF);
    DatoSPI(0xFF);
    DatoSPI(0xFE);
    for (i=0;i<512;i++)
        DatoSPI(buffer[i]);
    DatoSPI(0xFF);
    DatoSPI(0xFF);
    DatoSPI(0xFF);
    while(DatoSPI(0xFF) !=0xFF);
    SPI_CS_OFF();
    return OK;
}

//-----------------------------------------------------------------------
// char Borrar512(ulong adrInicio,ulong adrFin)
//
// Borra  un bloque de 512 bytes especificado entre las direcciones
// adr1 y adr2
//
// Retorno:
//  1 Si todo ha ido bien
//      0 Error
//-----------------------------------------------------------------------
char Borrar512(ulong adrInicio,ulong adrFin){
    /* Este comando no se simula con el modelo de Proteus */

    adrInicio&=0xFFFFFE00; // La direcciones tiene que ser multiplo de 512
    adrFin&=0xFFFFFE00;
    SPI_CS_ON();
    //Envia direccion de comienzo para borrar
    if(EnviaComandoMMC(TAG_ERASE_START,adrInicio,0xFF)!=0){
        SPI_CS_OFF();
        return ERROR;
    }
    //Envia direccion de final
    if(EnviaComandoMMC(TAG_ERASE_END,adrFin,0xFF)!=0){
```

```
330         SPI_CS_OFF();
            return ERROR;
        }
        //Envia orden de borrar
        if(EnviaComandoMMC(ERASE,0,0xFF)!=0){
            SPI_CS_OFF();
            return ERROR;
        }
        while(DatoSPI(0xFF)==0);
        SPI_CS_OFF();
340     return OK;
    }

    //-----------------------------------------------------------------------
    // void CardDelay(char retraso)
    //
    // Genera una espera de 8*retraso ciclos de reloj
    //
    // Nota: Debe de ser usada despues de hacer un SPI_CS_ON()
    ///----------------------------------------------------------------------
350
    void CardDelay(uchar retraso){
        uchar i=0;
        for(i=0;i<retraso;i++)
            DatoSPI(0xFF);
    }

    //-----------------------------------------------------------------------
    // char IdMMC(char *id)
    //
360 // Almancena en id el ID de la tarjeta
    //
    // Retorno:
    //  1 Si todo ha ido bien
    //      0 Error
    //-----------------------------------------------------------------------

    /*
    char IdMMC(char *id){
        Futura implementacion
370 }
    */

    //-----------------------------------------------------------------------
    // int CapacidadMMC()
    //
    // Devuelve la capacidad de la tarjeta en MB, si el valor devuelto es
    // cero se ha producido un error.
    //
    //-----------------------------------------------------------------------
380 /*
    int CapacidadMMC(){
         Futura implementacion
    }
    */
    //-----------------------------------------------------------------------
    // char ModeloMMC(char* modelo)
    //
    // Almacena en modelo el modelo de tarjeta. El string modelo debe de ser
    // de longitud 7 caracteres (6 del modelo +\0)
390 //
    // Retorno:
    //  1 Si todo ha ido bien
    //  0 Error
    //-----------------------------------------------------------------------
    /*
```

```
char ModeloMMC(char * modelo){
    Futura implementacion
}
*/

//-----------------------------------------------------------------------
// CsdMMC(char *csd)
//
// Almancena en csd el CSD de la tarjeta
// Este registro tiene una longitud de 16 bytes
//
// Retorno:
//  1 Si todo ha ido bien
//      0 Error
//-----------------------------------------------------------------------
/*
char CsdMMC(char *csd){
    Futura implementacion
}
*/
```

A.5 Código fuente librería FAT

Listado A.6: Fichero fat.h.

```
//-------------------------------------------------------------------
//
//                          DESCRIPCION
//
// Libreria para acceder a estructuras de archivos fat con MMC&SD
//
//-------------------------------------------------------------------
//
//    Archivo:              fat.h
//
//    Autores:      Miguel Angel Perez
//                                  Carlos Gobernado
//
//    Version: 1.12  1-9-2004
//
//      Version:    1.12:
//      Se ha procedido ha hacer una limpieza y revision de codigo.
//      LeeByte se ha cambiado por LeeInt
//
//      Version:    1.11:
//      Se ha introducido structFichero->sectCluster para cuando un
//      cluster ocupa áms de un sector.
//      Se ha introducido structFichero->tamanio para reflejar el
// ñ     tamao del fichero en bytes (multiplos de 512).
//      Se han corregido LeerFichero y EscribirFichero.
//
//      Version: 1.10:
//      Se ha corregido un bug en ActualizarFAT() y
//      CerrarFicheroEscritura se ha modificado para que diferencia entre
//      fichero nuevo y fichero existente
//
// ó     Versin 1.00:
//      La funcion FXM3232 hace una multiplicacion 32x32=64, para esta
//      caso solo no es necesario un resultado de 32bits. Por lo que
//      cuando tenga tiempo la modificare para que sea mas rapida.
//      La escritura de ficheros esta solo pensada para escribir en
//      CLUSTER CONSECUTIVOS!!!
//
//-------------------------------------------------------------------

//-------------------------------------------------------------------
// Definicion de constantes
//-------------------------------------------------------------------

#ifndef __FAT_H
#define __FAT_H

#define ERROR                   0
#define OK                      1
#define NO_FIND                 0xFFF7  // Uso la marca de sector defectuoso
#define NULL                    0
#define FIN_CLUSTER             0xFFFF  // Marca deú ltimo cluster.
#define CLUSTER_LIBRE           0
#define FICHERO_NUEVO           0
#define FICHERO_AGREGAR         1

#define PRODH   (PRODL + 1)
#define Mul32x32(d1,d2) arg1=d1; arg2=d2; FXM3232();

typedef unsigned char BYTE;
typedef unsigned char byte;
typedef unsigned char uchar;
```

```
typedef unsigned int  uint;
typedef unsigned long ulong;

//-------------------------------------------------------------------
//Estructuras de datos
//-------------------------------------------------------------------

struct structFAT{
  ulong tablaFAT;              // direccion que apunta a la FAT
  uint  sectoresFAT;           // numero de sectores por FAT
  uint  entradasRaiz;          //
  ulong datos;                 // direccion que apunta a la zona de datos
                               // esta zona empieza con el segundo cluster
  ulong dirRaiz;               // direccion al directorio raiz
  char sectorCluster;          // numero de sectores por cluster
};

struct structFichero{
    char    nombre[11];        // nombre del fichero.
    uint    clusterInicio;     // primer cluster del fichero.
    uint    numCluster;        // numero de cluster que tiene el fichero
    uchar sectCluster;         // ultimo sector por cluster que se ha escrito
    ulong tamanio;             // tamanio del fichero en bytes;
};


//-------------------------------------------------------------------
// Prototipos de funciones
//-------------------------------------------------------------------

uint LeeInt(char *,char);
void InitFAT(char *);
uint AbrirFicheroLectura(char *,char *);
uint LeerFichero(char *,uint,char);
ulong EspacioLibre(char *);
uint BuscaClusterLibre(char *);
uint EscribirFichero(char*,uint,char);
void CerrarFicheroEscritura(char *,char);
void ActualizarFAT(char *);
void FXM3232 (void);
```

Listado A.7: Fichero fat.c.

```
//-------------------------------------------------------------------
//
//                          DESCRIPCION
//
// Libreria para acceder a estructuras de archivos fat con MMC&SD
//
//-------------------------------------------------------------------
//
//    Archivo:             fat.c
//
//    Autores:      Miguel Angel Perez
//                                  Carlos Gobernado
//
//    Version: 1.12 1-9-2004
//
//      Version:    1.12:
//      Se ha procedido ha hacer una limpieza y revision de codigo
//
//      Version:    1.11:
//      Se ha introducido structFichero->sectCluster para cuando un
//      cluster ocupa áms de un sector.
//      Se ha introducido structFichero->tamanio para reflejar el
// ñ     tamao del fichero en bytes (multiplos de 512).
//      Se han corregido LeerFichero y EscribirFichero.
//
//      Version: 1.10:
//      Se ha corregido un bug en ActualizarFAT() y
//      CerrarFicheroEscritura se ha modificado para que diferencia entre
//      fichero nuevo y fichero existente
//
//      Version 1.00:
//      La funcion FXM3232 hace una multiplicacion 32x32=64, para esta
//      caso solo no es necesario un resultado de 32bits. Por lo que
//      cuando tenga tiempo la modificare para que sea mas rapida.
//      La escritura de ficheros esta solo pensada para escribir en
//      CLUSTER CONSECUTIVOS!!!
//
//-------------------------------------------------------------------

#include <p18cxxx.h>
#include <stdlib.h>
#include <string.h>
#include <usart.h>
#include "fat.h"
#include "debug.h"
#include "mmc.h"



struct structFAT FAT;
struct structFichero FICHERO;

// Estas variables deben de ser definidas para poder usar con FXM3232
// implementado en este mismo fichero
// reservo espacio para poder hacer los productos;

#pragma udata access accessram              // Estas funciones deben de
near unsigned long   res,res1,arg1,arg2;    // ser near y se emplean para
                                            // productos de 32bitsx32bits
                                            // en fat.c
#pragma udata

//-------------------------------------------------------------------
// uint LeeInt(char * buffer ,char dir)
//
```

```
// Esta funcion se encargar de devolver 2bytes apuntados por la
// dir dentro de un buffer de 512bytes. Logicamente el buffer debe de
// tener una longitud de 512bytes y la direccion menor de 512
//
// Retorno:
//  El valor apuntado por dir
//
//-------------------------------------------------------------------

uint LeeInt(char * buffer ,char dir){
    return *((uint*)(buffer+dir));
}


//-------------------------------------------------------------------
// void InitFAT(char *buffer)
//
// Esta funcion se encarga  de obtener todos los punteros y cargarlos
// en la estructura FAT (definida en FAT.h). Para ello lee de un buffer
// de 512 bytes.
//
//
//-------------------------------------------------------------------

void InitFAT(char *buffer){
    ulong adr=0;
    ulong temp=0;
    char *p=&temp;
/*
    Lectura512(buffer,adr);             // Leo el MBR de la tarjeta
    adr=(ulong)LeeInt(buffer,0x1C8);    // Busco la° 1Particion en MBR
    //PrintLong("adr 0x1C6",adr);

    // De las pruebas que he hecho con la tarjeta he visto que no tiene
    // MBR y en el primer sector de la tarjeta esta BootRecord de la
    // primera particion directamente. Por eso estan comentadas estas lineas
*/
    Lectura512(buffer,adr);             // Leo particion 1 de la tarjeta


    temp=LeeInt(buffer,0xD);            // En el° 1byte de temp tengo los
                                        // sectores por cluster
    FAT.sectorCluster=*p;               // Almaceno el primer byte

    temp=LeeInt(buffer,0xE);            // Busco la entrada a la FAT
    FAT.tablaFAT=adr+(512*temp);        // Almaceno en la estructura

    // Leo el °n de entradas del directorio raiz
    FAT.entradasRaiz=LeeInt(buffer,0x11);

    temp=LeeInt(buffer,0x16);           // Leo el únmero de sect. por FAT
    FAT.sectoresFAT=temp;

    // dirRaiz=Start + # of Reserved + (# of Sectors Per FAT * 2*512bytes)


    Mul32x32(temp,1024);                // resultado en res

    FAT.dirRaiz=FAT.tablaFAT+res;       // Almaceno directorio raiz

    // datos= dirRaiz +(((Max Root Diry Entries * 32) / Bytes per Sector))*512
    Mul32x32(FAT.entradasRaiz,32);      // resultado en res
    FAT.datos=FAT.dirRaiz+res;

}
```

```
//-------------------------------------------------------------------
// uint AbrirFicheroLectura(char *nombre,char * buffer)
//
// Busca el nombre de un fichero dentro de la estructura de directorio
// raiz. La funcion strstr no me vale porque estas cadenas no áestn
// terminadas con \0.
//
// Retorno:
//
// ú         Nmero del primer cluster del fichero.
//
//-------------------------------------------------------------------
uint AbrirFicheroLectura(char *nombre,char * buffer){
    // Busco el nombre del fichero dentro de la estructura de directorio
    // raiz
    // nombre debe de ser un vector de 11 posiciones.

    // El °n de entradas por sector=512/32=16 entradas
    // El °n de sectores del directorio raiz=(num de entradasRaiz)/16

    uint i=0;               // Indice de entradas en el directorio raiz
    char j;                 // j-> indice para la cadena del nombre del fichero
    uint nSector=0;         // °n de sector del directorio raiz
    ulong dir=FAT.dirRaiz;
    char *p;                //Puntero buffer
    uint temp;
    //° 1 Cargo en el buffer la estrucutra de directorio raiz
    Lectura512(buffer,dir);
    //° 2 Busco el nombre del fichero , p apunta al fichero
    while(nSector<(FAT.entradasRaiz/16)){
        i=0;
        while(i<512){
            j=0;
            while((buffer[i+j]==nombre[j])&&(j<11)){
                j++;
            }
            if(j==11){
                // fichero encontrado devuelvo el cluster
                return *(buffer+i+26);
            }
            else{
                i+=32;
            }
        }
        nSector++;
        dir+=512;                   // Cargo siguiente direccion
        Lectura512(buffer,dir);     // Leo siguiente sector
    }
    return NO_FIND;                 // Fichero no encontrado devuelvo marca de cluster
                                    // defectuoso

}

//-------------------------------------------------------------------
// uint LeerFichero(char * buffer,uint clusterInicio, char sector)
//
// Esta funcion deja en buffer los datos del cluster de inicio y
// devuelve el numero del siguiente cluster y actualiza la variable
// FICHERO.sectCluster. Esta funcion esta preparada para cluster que
// ocupen áms de un sector.
//
//  buffer: Buffer de 512 bytes de datos,
// clusterInicio: numero de cluster a leer
// sector: numero de sector dentro del cluster a leer (previsto
//         para FAT donde un cluster ocupe áms de un sector de 512bytes
//
```

```
// Retorno:
//
//      Valor del siguiente cluster, si este valor es 0xFFFF entonces
//      es elú ltimo cluster.
//
//-------------------------------------------------------------------

uint LeerFichero(char * buffer,uint clusterInicio, char sector){
    uint retorno;
    ulong adr=FAT.datos;
    //º 1 Cargo sector FAT que corresponde a ese cluster

    adr+=(clusterInicio*2);         // Cada cluster en la FAT ocupa 2byte
    // La lecutra me devolvera el sector correspondiente a esa direccion

    Lectura512(buffer,FAT.tablaFAT);

    if(sector==FAT.sectorCluster){
        //º 2 Busco eL siguiente enlace.
        retorno=*(buffer+(clusterInicio)*2);            // obtengo sig cluster
        FICHERO.sectCluster=1;
    }
    else{
        // íTodava no necesito siguiente cluster
        retorno=clusterInicio;
        FICHERO.sectCluster++;
    }

    // Calculo direccion del clusterInicio
    clusterInicio-=2;   // La zona de datos empieza en elº 2 cluster
    sector--;
    if(clusterInicio!=0){
        Mul32x32(clusterInicio,(FAT.sectorCluster)*512);
        res+=sector*512;
        // resultado en res

        adr+=res;
    }
    else
        adr=adr+512*sector;         // Si no me ahorro la ómultiplicacin
                                    //con el gasto que implica

    Lectura512(buffer,adr);
    return retorno; // Devuelvo siguiente cluster.

}

//-------------------------------------------------------------------
// uint EscribirFichero(char* buffer,uint cluster,char sector)
//
// Esta ófuncin vuelca los datos del buffer en la zona de datos apuntda
// por el cluster y sector correspondiente
// El sector hay que emplearlo cuando un cluster ocupa mas de un sector
//
//  Retorno:
//
//      Valor del siguiente cluster.
//-------------------------------------------------------------------

uint EscribirFichero(char* buffer,uint cluster,char sector){
    ulong adr=FAT.datos;
    uint retorno;
                                    // El primer cluster en la zona de
                                    // datos es el 2
   retorno=cluster;
   FICHERO.tamanio+=512;            // Aumenta en 512 bytes
```

```
    //PrintInt("EscribirFichero->sector",sector);
    if(sector==1){
        FICHERO.numCluster++;
        //PrintInt("EscribirFichero->FICHERO.numCluster",FICHERO.numCluster);
    }
    if(sector==FAT.sectorCluster){
        retorno++;                  // obtengo sig cluster
        FICHERO.sectCluster=1;
    }
    else{
        // retorno=cluster;
        FICHERO.sectCluster++;
    }
    sector--;
    cluster-=2;   // La zona de datos empieza en elº 2 cluster
    if(cluster!=0){
        Mul32x32(cluster,(FAT.sectorCluster)*512);
        res+=sector*512;
        // resultado en res.
        adr+=res;
    }
    else{
                                    // Cluster únmero 2
        adr=adr+512*sector;         // Si no me ahorro la ómultiplicacin
                                    // con el gasto que implica
    }
    // tengo la direccion del sector

    // íSera interesante controlar si ha habido error en Escritura512
    if(Escritura512(buffer,adr)==ERROR){
        return ERROR;
    }
    return retorno;
}

//-----------------------------------------------------------------------
// ulong EspacioLibre(char * buffer)
//
// Esta funcion recorre toda la tabla FAT y calcula cuanto espacio libre
// hay en la particion.
//
//  Retorno:
//      Devuelve el ºn de sectores de 512 bytes libres.
//
//
//-----------------------------------------------------------------------

ulong EspacioLibre(char * buffer){
    ulong nClusterLibres=0;
    uint i,j;
    ulong adr=FAT.tablaFAT;
    for(i=0;i<FAT.sectoresFAT;i++){
        Lectura512(buffer,adr);             // Leo ese sector de FAT
        j=0;
        while(j<512){
            if(*((uint*)(buffer+j))==CLUSTER_LIBRE)
                nClusterLibres++;           // Cuento cluster libre

            j+=2;
        }
        adr+=512;                           // Siguiente sector de tabla FAT
    }
    if(FAT.sectorCluster!=1){
        Mul32x32(nClusterLibres,FAT.sectorCluster); //resultado en res

        return res;         // res contiene el numero de sectores libres
```

```
330     }
        else{
            // Si un Cluster es un sector me ahorro la multiplicacion
            //PrintLong("EspacioLibre-> nClusterLibres",nClusterLibres);
            return nClusterLibres;
        }
    }

    //-----------------------------------------------------------------
    // uint BuscaClusterLibre(char *buffer)
340 //
    // Esta ófuncin busca el primer cluster libre en la tabla FAT para
    // luego poder escribir en CLUSTER CONSECUTIVOS. El cluster libre se
    // almacena en la estructura FICHERO declarada arriba de este fichero.
    // Tambien incializa FICHERO.numCluster y FICHERO.tamanio
    //
    //  Retorno:
    //      Devuelve el °n de cluster.
    //    Si devuelve NO_FIND no ha encontrado cluster libre.
    //
350 //
    //-----------------------------------------------------------------

    uint BuscaClusterLibre(char *buffer){
        ulong adr=FAT.tablaFAT;
        uint i,j;
        uint retorno=0;
        for(i=0;i<FAT.sectoresFAT;i++){
            j=0;
            Lectura512((char*)buffer,adr);
360         while(j<512){
                if(*((uint*)(buffer+j))==CLUSTER_LIBRE){
                    FICHERO.clusterInicio=retorno;
                    FICHERO.numCluster=0;
                    FICHERO.tamanio=0;
                    return retorno;
                }
                j+=2;
                retorno++;
            }
370         adr+=512;
        }
        return NO_FIND;
    }


    //-----------------------------------------------------------------
    // void CerrarFicheroEscritura(char *buffer,char tipo)
    //
    // Esta ófuncin actualiza la tabla FAT y la estructura del directorio
380 // raiz para cerrar un fichero que se ha escrito en CLUSTER
    // CONSECUTIVOS.
    // En tipo se especifica si el fichero ya existe o es nuevo
    //
    //-----------------------------------------------------------------

    void CerrarFicheroEscritura(char *buffer,char tipo){
        // 1- Actualizar tabla FAT
        //> 2-  Copia FAT?-> He descubierto que no hace falta actualizarla
        // 3- Actualizar directorio raiz.
390     ulong adr=FAT.dirRaiz;
        uint nSector=0;      // °n de sector del directorio raiz
        uint j;              // indice entradas del directorio raiz
        uint i=0;
        //° 1 Actualizar tabla FAT
        ActualizarFAT(buffer);
```

```
    //º 3 Actualizar directorio raiz

    Lectura512(buffer,adr);
    while(nSector<(FAT.entradasRaiz/16)){
400
        if(tipo==FICHERO_NUEVO){
            j=0;
            while(j<512){
                if(buffer[j]==0 || buffer[j]==0xE5){
                    // Posicion libre
                    // 0xE5 es la marca que pone cuando se borra

                    memcpy((void *)(buffer+j),(void *) FICHERO.nombre,11);
                    buffer[j+26]=FICHERO.clusterInicio;
410                 // No hago nada por ahora acerca de la fecha
                    //Escribo ñtamao fichero
                    *((ulong*)(buffer+j+28))=FICHERO.tamanio;
                    Escritura512(buffer,adr);
                    return;
                }
                else
                    j+=32;
            }
        }
420     else if (tipo==FICHERO_AGREGAR){
            // el fichero ya existe
            j=0;
            i=0;
            while(j<512){
                //PrintInt("Contador i antes while",i);
                //Busco en el directorio raiz el fichero correspondiente
                while((buffer[i+j]==FICHERO.nombre[i])&&(i<11)){
                    //PrintInt("CerrarFicheroEscritura->i",i);
                    i++;
430             }
                if(i==11){
                    //PrintInt("Cluster inicio",FICHERO.clusterInicio);
                    // fichero encontrado devuelvo el cluster
                    buffer[j+26]=FICHERO.clusterInicio;
                    // No hago nada por ahora acerca de la fecha
                    //Escribo ñtamao fichero
                    *((ulong*)(buffer+j+28))=FICHERO.tamanio;
                    Escritura512(buffer,adr);
                    return;
440             }
                else{
                    j+=32;
                }
            }
        }
        nSector++;
        adr+=512;                   // Cargo siguiente direccion
        Lectura512(buffer,adr);     // Leo siguiente sector
    }
450
}

//-----------------------------------------------------------------------
// void ActualizarFAT(char * buffer)
//
// Actualiza la tabla FAT del archivo. Esta funcion es llamada por
// CerrarFicheroEscritura(char *buffer)
//-----------------------------------------------------------------------

460 void ActualizarFAT(char * buffer){
```

```
    ulong adr=FAT.tablaFAT;
    uint i=0;
    uint j=2*(FICHERO.clusterInicio);
    uint clusterSig=FICHERO.clusterInicio+1;

    if(FICHERO.clusterInicio<32767) // FFFF/2
        adr+=(FICHERO.clusterInicio*2);
    else{
        //Solo multiplica 16x16 Se desborda, necesito ...
        Mul32x32(FICHERO.clusterInicio,2);
        adr+=res;
    }


    Lectura512(buffer,adr);
    // Tengo la zona de la FAT correspondiente.
    i=0;        // Contador de cluster
    while(1){
        if(j!=2*FICHERO.clusterInicio)
            j=0;
        while(j<512){
            if(i==(FICHERO.numCluster-1)){
                // Marca de fin de fichero
                *((uint*)(buffer+j))=FIN_CLUSTER;
                Escritura512(buffer,adr);
                return;
            }
            *((uint*)(buffer+j))=clusterSig;
            clusterSig++;
            j+=2;
            i++;
        }
        //PrintLong("ActualizarFAT-> Escribiendo sector en ",adr);
        Escritura512(buffer,adr);
        adr+=512;
    }
    return;
}

//-----------------------------------------------------------------------
// void FXM3232 (void)
//
// Multiplica dos numeros de 32 bits sin signo y obtiene el resultado
// de 64 bits
// raiz
//
//  arg1 registro con dato1
//      arg2 registro con dato2
//      res  resultado
//
//  Nota: Esta funcion puede ser mejorada porque solo necesito
//  32x32=32bits.
//
//
//-----------------------------------------------------------------------

void FXM3232 (void)
{
  _asm
    movf arg1, 0, 0
    mulwf arg2, 0
    // low byte of both operands, so result adds into the low order
    // result bytes
    movff PRODL, res
    movff PRODH, res + 1
    // W still contains arg1
```

```
mulwf arg2 + 1, 0
// arg2[1], so result adds into res[1,2]
movf PRODL, 0, 0
addwf res + 1, 1, 0
movlw 0
addwfc PRODH, 0, 0
movwf res + 2, 0
// reload arg1 to continue
movf arg1, 0, 0
mulwf arg2 + 2, 0
// arg2[2], so result adds into res[2,3]
movf PRODL, 0, 0
addwf res + 2, 1, 0
movlw 0
addwfc PRODH, 0, 0
movwf res + 3, 0
// reload arg1 to continue
movf arg1, 0, 0
mulwf arg2 + 3, 0
// arg2[3], so result adds into res[3].
// we don't care about result bytes above res[3], ignore prodh here.
movf PRODL, 0, 0
addwf res + 3, 1, 0
// that's the end of all terms involving arg1[0].
// load arg1[1] to continue
movf arg1 + 1, 0, 0
mulwf arg2, 0
// arg1[1], so result adds into res[1,2]
movf PRODL, 0, 0
addwf res + 1, 1, 0
movf PRODH, 0, 0
addwfc res + 2, 1, 0
movlw 0
addwfc res + 3, 1, 0
// reload arg1[1] to continue
movf arg1 + 1, 0, 0
mulwf arg2 + 1, 0
// arg1[1] and arg2[1], so result adds into res[2,3]
movf PRODL, 0, 0
addwf res + 2, 1, 0
movf PRODH, 0, 0
addwfc res + 3, 1, 0
// reload arg1[1] to continue
movf arg1 + 1, 0, 0
mulwf arg2 + 2, 0
// arg1[1] and arg2[2], so result adds into res[3]
// we don't care about result bytes above res[3], ignore prodh here.
movf PRODL, 0, 0
addwf res + 3, 1, 0
// all bytes of the term from the product of arg1[1] and arg2[3] are
// above our 32-bit result, don't even need to bother calculating
// that term.
// load arg1[2] to continue
movf arg1 + 2, 0, 0
mulwf arg2, 0
// arg1[2] and arg2[0], so result adds into res[2,3]
movf PRODL, 0, 0
addwf res + 2, 1, 0
movf PRODH, 0, 0
addwfc res + 3, 1, 0
// reload arg1[2] to continue
movf arg1 + 2, 0, 0
mulwf arg2 + 1, 0
// arg1[2] and arg2[1], so result adds into res[3]
// we don't care about result bytes above res[3], ignore prodh here.
movf PRODL, 0, 0
```

```
    addwf res + 3, 1, 0
    // all bytes of the termsfrom the products of arg1[2] and arg2[2,3] are
    // above our 32-bit result, don't even need to bother calculating
    // those terms.
    // load arg1[3] to continue
    movf arg1 + 3, 0, 0
    mulwf arg2, 0
    // arg1[3] and arg2[0], so result adds into res[3]
    movf PRODL, 0, 0
    addwf res + 3, 1, 0

    _endasm;
}
```

A.6 Código fuente librería controlador *CAN*

Listado A.8: Fichero CAN18XX8.h.

```
/*********************************************************************
 *              PIC18CXX8 CAN C Library Header File
 *********************************************************************
 * FileName:        CAN18CXX8.h
 * Dependencies:    None
 * Date:            09/06/00
 * Processor:       PIC18CXX8
 * Complier:        MPLAB 5.11.00
 * Company:         Microchip Technology, Inc.
 *
 *
 * Software License Agreement
 *
 * The software supplied herewith by Microchip Technology Incorporated
 * (the Company) for its PICmicro  Microcontroller is intended and
 * supplied to you, the Company  customer, for use solely and
 * exclusively on Microchip PICmicro Microcontroller products. The
 * software is owned by the Company and/or its supplier, and is
 * protected under applicable copyright laws. All rights are reserved.
 *  Any use in violation of the foregoing restrictions may subject the
 * user to criminal sanctions under applicable laws, as well as to
 * civil liability for the breach of the terms and conditions of this
 * license.
 *
 * THIS SOFTWARE IS PROVIDED IN AN AS IS CONDITION. NO WARRANTIES,
 * WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
 * TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
 * PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
 * IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
 * CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
 *
 * Author               Date    Comment
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * Nilesh Rajbharti     9/6/00  Original        (Rev 1.0)
 * Nilesh Rajbharti     12/1/00 Fixed bugs.
 *                              (CANRegsToID, CANReceiveMessage,
 *                               CANSetMask + Added
 *                               CAN_CONFIG_DBL_BUFFER_ON/OFF option)
 * Nilesh Rajbharti     6/8/01  Renamed CAN_CONFIG_DBL_BUFFERED to
 *                              CAN_RX_DBL_BUFFERED
 * Nilesh Rajbharti     10/11/01 Added support for HITECH compiler
 *                              Modified CAN_MESSAGE_ID def to be
 *                              compatible with HITECH Compiler
 *                              Fixed CAN_CONFIG_SAMPLE_BIT enum.
 *                              (Rev 1.1)
 *
 *********************************************************************/
#ifndef CAN18XX8_H          // To avoid duplicate inclusion
#define CAN18XX8_H

#if defined(HI_TECH_C)
    #define HITECH_C18
#else
    #define MCHP_C18
#endif

#if defined(MCHP_C18) && defined(HITECH_C18)
#error "Invalid Compiler selection."
#endif

#if !defined(MCHP_C18) && !defined(HITECH_C18)
#error "Compiler not supported."
```

```
#endif

#if defined(MCHP_C18)
    #define COMSTAT_TXB0        COMSTATbits.TXB0
    #define COMSTAT_TXBP        COMSTATbits.TXBP
    #define COMSTAT_RXBP        COMSTATbits.RXBP
    #define CANCON_ABAT         CANCONbits.ABAT
    #define RXB0CON_RXFUL       RXB0CONbits.RXFUL
    #define RXB1CON_RXFUL       RXB1CONbits.RXFUL
    #define TXB0CON_TXREQ       TXB0CONbits.TXREQ
    #define TXB1CON_TXREQ       TXB1CONbits.TXREQ
    #define TXB2CON_TXREQ       TXB2CONbits.TXREQ
#endif

#if defined(HITECH_C18)
    #define COMSTAT_TXB0        TXB0
    #define COMSTAT_TXBP        TXBP
    #define COMSTAT_RXBP        RXBP
    #define CANCON_ABAT         ABAT
    #define RXB0CON_RXFUL       RXB0RXFUL
    #define TXB0CON_TXREQ       TXB0REQ

    /*
     * Following are special defs to overcome compiler problem
     * at the time of this re-write.
     * Set following line to "#if 0" after verifiying correct
     * compiler behavior.
     */
#if 1
    static struct
    {
        unsigned : 7;
        unsigned RXFUL : 1;
    } RXB1CONbits @ 0xf50;
    #define RXB1CON_RXFUL       RXB1CONbits.RXFUL

    static struct
    {
        unsigned TXPRI0:1;
        unsigned TXPRI1:1;
        unsigned :1;
        unsigned TXREQ:1;
        unsigned TXERR:1;
        unsigned TXLARB:1;
        unsigned TXABT:1;
    } TXB1CONbits @ 0xf30;
    #define TXB1CON_TXREQ       TXB1CONbits.TXREQ

    static struct
    {
        unsigned TXPRI0:1;
        unsigned TXPRI1:1;
        unsigned :1;
        unsigned TXREQ:1;
        unsigned TXERR:1;
        unsigned TXLARB:1;
        unsigned TXABT:1;
    } TXB2CONbits @ 0xf20;
    #define TXB2CON_TXREQ       TXB2CONbits.TXREQ
#else
    #define RXB1CON_RXFUL       RXB1RXFUL
    #define TXB1CON_TXREQ       TXB1REQ
    #define TXB2CON_TXREQ       TXB2REQ
#endif
```

```
#endif


/*********************************************************************
 *
 * General purpose typedef's used by PICCAN libray.
 *
 * Remove these definations if they are already defined in one of your
 * application files.  Make sure that corresponding header file is
 * included before including this header file.
 *********************************************************************/
typedef enum _BOOL { FALSE = 0, TRUE } BOOL;
typedef unsigned char BYTE;
typedef unsigned char byte;

//////////////////////////////////////////////////////////////////////

/*********************************************************************
 *
 * union CAN_MESSAGE_ID
 *
 * This union provides abstract data type for CAN message id.
 * It is used for both 11-bit and 29-bit message identifiers.
 * There are multiple union members to be able to access individual
 * parts of it.
 *
 *********************************************************************/
 // Parse-out 29-bit or 11-bit (saved in 32-bit number)
typedef union _CAN_MESSAGE_ID
{
    unsigned long ID;

    struct
    {
        struct
        {
            unsigned SIDL:3;        // SIDL<5:7>
            unsigned SIDH:5;        // SIDH<0:4>
        } BYTE1;
        struct
        {
            unsigned SIDHU:3;       // SIDH<5:7>
            unsigned EIDL_LN:5;     // EIDL<0:4>
        } BYTE2;
        struct
        {
            unsigned EIDL_UN:3;     // EIDL<5:7>
            unsigned EIDH_LN:5;     // EIDH<0:4>
        } BYTE3;
        struct
        {
            unsigned EIDH_UN:3;     // EIDH<5:7>
            unsigned EIDHU:2;       // SIDL<0:1>
            unsigned :3;
        } BYTE4;
    } ID_VALS;

    // This is to allow individual byte access within message id.
    struct
    {
        BYTE BYTE_1;
        BYTE BYTE_2;
        BYTE BYTE_3;
```

```
        BYTE BYTE_4;
    } BYTES;
} CAN_MESSAGE_ID;

//////////////////////////////////////////////////////////////////////
200
/*******************************************************************
 *
 * enum CAN_TX_MSG_FLAGS
 *
 * This enumeration values define flags related to transmission of a
 * CAN message.  There could be more than one this flag
 * ANDed together to form multiple flags.
 *
 *******************************************************************/
210 enum CAN_TX_MSG_FLAGS
 {
    CAN_TX_PRIORITY_BITS= 0b00000011,
    CAN_TX_PRIORITY_0   = 0b11111100,   // XXXXXX00
    CAN_TX_PRIORITY_1   = 0b11111101,   // XXXXXX01
    CAN_TX_PRIORITY_2   = 0b11111110,   // XXXXXX10
    CAN_TX_PRIORITY_3   = 0b11111111,   // XXXXXX11

    CAN_TX_FRAME_BIT    = 0b00001000,
    CAN_TX_STD_FRAME    = 0b11111111,   // XXXXX1XX
220 CAN_TX_XTD_FRAME    = 0b11110111,   // XXXXX0XX

    CAN_TX_RTR_BIT      = 0b01000000,
    CAN_TX_NO_RTR_FRAME = 0b11111111,   // X1XXXXXX
    CAN_TX_RTR_FRAME    = 0b10111111    // X0XXXXXX
};

//////////////////////////////////////////////////////////////////////

/*******************************************************************
230 *
 * enum CAN_RX_MSG_FLAGS
 *
 * This enumeration values define flags related to reception of a CAN
 * message.  There could be more than one this flag
 * ANDed together to form multiple flags.
 * If a particular bit is set, corresponding meaning is TRUE or else
 * it will be FALSE.
 *
 * e.g.
240 *      if (MsgFlag & CAN_RX_OVERFLOW)
 *      {
 *          // Receiver overflow has occured.  We have lost previous
 *          // message.
 *          ...
 *      }
 *
 *******************************************************************/
enum CAN_RX_MSG_FLAGS
{
250 CAN_RX_FILTER_BITS  = 0b00000111,   // Use this to access filter
                                        // bits
    CAN_RX_FILTER_1     = 0b00000000,
    CAN_RX_FILTER_2     = 0b00000001,
    CAN_RX_FILTER_3     = 0b00000010,
    CAN_RX_FILTER_4     = 0b00000011,
    CAN_RX_FILTER_5     = 0b00000100,
    CAN_RX_FILTER_6     = 0b00000101,

    CAN_RX_OVERFLOW     = 0b00001000,   // Set if Overflowed else
260                                     // cleared
```

```
    CAN_RX_INVALID_MSG  = 0b00010000,   // Set if invalid else
                                        // cleared

    CAN_RX_XTD_FRAME    = 0b00100000,   // Set if XTD message else
                                        // cleared

    CAN_RX_RTR_FRAME    = 0b01000000,   // Set if RTR message else
                                        // cleared

    CAN_RX_DBL_BUFFERED = 0b10000000    // Set if this message was
                                        // hardware double-buffered
};

//////////////////////////////////////////////////////////////////////


/*********************************************************************
 *
 * enum CAN_MASK
 *
 * This enumeration values define mask codes.  Routine CANSetMask()
 * requires this code as one of its arguments.  These enumerations
 * must be used by itself i.e. it cannot be ANDed to form multiple
 * values.
 *
 *********************************************************************/
enum CAN_MASK
{
     CAN_MASK_B1,
     CAN_MASK_B2
};

//////////////////////////////////////////////////////////////////////

/*********************************************************************
 *
 * enum CAN_FILTER
 *
 * This enumeration values define filter codes.  Routine CANSetFilter
 * requires this code as one of its arguments.  These enumerations
 * must be used by itself
 * i.e. it cannot be ANDed to form multiple values.
 *
 *********************************************************************/
enum CAN_FILTER
{
    CAN_FILTER_B1_F1,
    CAN_FILTER_B1_F2,
    CAN_FILTER_B2_F1,
    CAN_FILTER_B2_F2,
    CAN_FILTER_B2_F3,
    CAN_FILTER_B2_F4
};

//////////////////////////////////////////////////////////////////////

/*********************************************************************
 *
 * enum CAN_OP_MODE
 *
 * This enumeration values define codes related to CAN module
 * operation mode. CANSetOperationMode() routine requires this code.
 * These values must be used by itself
 * i.e. it cannot be ANDed to form * multiple values.
 *
```

```
 ******************************************************************/
enum CAN_OP_MODE
{
    CAN_OP_MODE_BITS     = 0b11100000,   // Use this to access opmode
                                         // bits
    CAN_OP_MODE_NORMAL   = 0b00000000,
    CAN_OP_MODE_SLEEP    = 0b00100000,
    CAN_OP_MODE_LOOP     = 0b01000000,
    CAN_OP_MODE_LISTEN   = 0b01100000,
    CAN_OP_MODE_CONFIG   = 0b10000000
};

////////////////////////////////////////////////////////////////////

/*******************************************************************
 *
 * enum CAN_CONFIG_FLAGS
 *
 * This enumeration values define flags related to configuring CAN
 * module. Routines CANInitialize() and CANSetBaudRate() use these
 * codes. One or more these values may be ANDed to form multiple
 * flags.
 *
 ******************************************************************/
enum CAN_CONFIG_FLAGS
{
    CAN_CONFIG_DEFAULT           = 0b11111111,   // 11111111

    CAN_CONFIG_PHSEG2_PRG_BIT    = 0b00000001,
    CAN_CONFIG_PHSEG2_PRG_ON     = 0b11111111,   // XXXXXXX1
    CAN_CONFIG_PHSEG2_PRG_OFF    = 0b11111110,   // XXXXXXX0

    CAN_CONFIG_LINE_FILTER_BIT   = 0b00000010,
    CAN_CONFIG_LINE_FILTER_ON    = 0b11111111,   // XXXXXX1X
    CAN_CONFIG_LINE_FILTER_OFF   = 0b11111101,   // XXXXXX0X

    CAN_CONFIG_SAMPLE_BIT        = 0b00000100,
    CAN_CONFIG_SAMPLE_ONCE       = 0b11111111,   // XXXXX1XX
    CAN_CONFIG_SAMPLE_THRICE     = 0b11111011,   // XXXXX0XX

    CAN_CONFIG_MSG_TYPE_BIT      = 0b00001000,
    CAN_CONFIG_STD_MSG           = 0b11111111,   // XXXX1XXX
    CAN_CONFIG_XTD_MSG           = 0b11110111,   // XXXX0XXX

    CAN_CONFIG_DBL_BUFFER_BIT    = 0b00010000,
    CAN_CONFIG_DBL_BUFFER_ON     = 0b11111111,   // XXX1XXXX
    CAN_CONFIG_DBL_BUFFER_OFF    = 0b11101111,   // XXX0XXXX

    CAN_CONFIG_MSG_BITS          = 0b01100000,
    CAN_CONFIG_ALL_MSG           = 0b11111111,   // X11XXXXX
    CAN_CONFIG_VALID_XTD_MSG     = 0b11011111,   // X10XXXXX
    CAN_CONFIG_VALID_STD_MSG     = 0b10111111,   // X01XXXXX
    CAN_CONFIG_ALL_VALID_MSG     = 0b10011111    // X00XXXXX
};

////////////////////////////////////////////////////////////////////

/*******************************************************************
 * Function:        void CANInitialize( BYTE SJW,
 *                                      BYTE BRP,
 *                                      BYTE PHSEG1,
 *                                      BYTE PHSEG2,
 *                                      BYTE PROPSEG,
 *                                      enum CAN_CONFIG_FLAGS flags)
 *
 * PreCondition:    MCU must be in Configuration mode or else these
```

```
 *                  values will be ignored.
 *
 * Input:           SJW     - SJW value as defined in 18CXX8 datasheet
 *                            (Must be between 1 thru 4)
 *                  BRP     - BRP value as defined in 18CXX8 datasheet
 *                            (Must be between 1 thru 64)
 *                  PHSEG1  - PHSEG1 value as defined in 18CXX8
 *                            datasheet
 *                            (Must be between 1 thru 8)
 *                  PHSEG2  - PHSEG2 value as defined in 18CXX8
 *                            datasheet
 *                            (Must be between 1 thru 8)
 *                  PROPSEG - PROPSEG value as defined in 18CXX8
 *                            datasheet
 *                            (Must be between 1 thru 8)
 *                  flags   - Value of type enum CAN_CONFIG_FLAGS
 *
 * Output:          CAN bit rate is set.
 *                  All masks registers are set '0'
 *                  to allow all messages.
 *                  Filter registers are set according to flag value.
 *                  If (config & CAN_CONFIG_VALID_XTD_MSG)
 *                      Set all filters to XTD_MSG
 *                  Else if (config & CONFIG_VALID_STD_MSG)
 *                      Set all filters to STD_MSG
 *                  Else
 *                      Set half of the filters to STD while rests to
 *                      XTD_MSG.
 *
 * Side Effects:    All pending transmissions are aborted.
 ***************************************************************/
void CANInitialize(BYTE SJW,
                   BYTE BRP,
                   BYTE PHSEG1,
                   BYTE PHSEG2,
                   BYTE PROPSEG,
                   enum CAN_CONFIG_FLAGS config);

//////////////////////////////////////////////////////////////////


/***************************************************************
 * Function:        void CANSetOperationMode(CAN_OP_MODE mode)
 *
 * PreCondition:    None
 *
 * Input:           mode    - Operation mode code
 *                            must be of type enum CAN_OP_MODES
 *
 * Output:          MCU is set to requested mode
 *
 * Side Effects:    None
 *
 * Note:            This is a blocking call.  It will not return until
 *                  requested mode is set.
 ***************************************************************/
void CANSetOperationMode(enum CAN_OP_MODE mode);

//////////////////////////////////////////////////////////////////


/***************************************************************
 * Macro:           void CANSetOperationModeNoWait(CAN_OP_MODE mode)
 *
 * PreCondition:    None
```

```
 *
460 * Input:           mode    - Operation mode code
 *                             must be of type enum CAN_OP_MODES
 *
 * Output:            MCU is set to requested mode
 *
 * Side Effects:      None
 *
 * Note:              This is a non-blocking call.
 *                    It does not verify that
 *                    CAN module is switched to requested mode or not.
470 *                  Caller must use CANGetOperationMode() to verify
 *                    correct operation mode before performing mode
 *                    specific operation.
 *
 ********************************************************************/
#define CANSetOperationModeNoWait(mode) (CANCON = mode)

////////////////////////////////////////////////////////////////////

/********************************************************************
480 * Macro:             CAN_OP_MODE CANSetOperationMode()
 *
 * PreCondition:      None
 *
 * Input:             None
 *
 * Output:            Current operational mode of CAN module is returned
 *
 * Side Effects:      None
 *
490 ********************************************************************/
#define CANGetOperationMode() (CANCON & CAN_OP_MODE_BITS)

////////////////////////////////////////////////////////////////////

/********************************************************************
 * Function:          void CANSetBaudRate(BYTE SJW,
 *                                        BYTE BRP,
 *                                        BYTE PHSEG1,
 *                                        BYTE PHSEG2,
500 *                                        BYTE PROPSEG,
 *                                        enum CAN_CONFIG_FLAGS flags)
 *
 * PreCondition:      MCU must be in Configuration mode or else these
 *                    values will be ignored.
 *
 * Input:             SJW     - SJW value as defined in 18CXX8 datasheet
 *                                (Must be between 1 thru 4)
 *                    BRP     - BRP value as defined in 18CXX8 datasheet
 *                                (Must be between 1 thru 64)
510 *                    PHSEG1  - PHSEG1 value as defined in 18CXX8
 *                              datasheet
 *                                (Must be between 1 thru 8)
 *                    PHSEG2  - PHSEG2 value as defined in 18CXX8
 *                              datasheet
 *                                (Must be between 1 thru 8)
 *                    PROPSEG - PROPSEG value as defined in 18CXX8
 *                              datasheet
 *                                (Must be between 1 thru 8)
 *                    flags   - Value of type enum CAN_CONFIG_FLAGS
520 *
 * Output:            CAN bit rate is set as per given values.
 *
 * Side Effects:      None
 *
```

```
 ********************************************************************/
void CANSetBaudRate(BYTE SJW,
                    BYTE BRP,
                    BYTE PHSEG1,
                    BYTE PHSEG2,
                    BYTE PROPSEG,
                    enum CAN_CONFIG_FLAGS);

////////////////////////////////////////////////////////////////////////

/********************************************************************
 * Function:        void CANSetMask(enum CAN_MASK code,
 *                                  unsigned long val,
 *                                  enum CAN_CONFIG_FLAGS type)
 *
 * PreCondition:    MCU must be in Configuration mode.  If not, all
 *                  values
 *                  will be ignored.
 *
 * Input:           code    - One of CAN_MASK value
 *                  val     - Actual mask register value.
 *                  type    - Type of message to filter either
 *                            CAN_CONFIG_XTD_MSG or CAN_CONFIG_STD_MSG
 *
 * Output:          Given value is bit adjusted to appropriate buffer
 *                  mask registers.
 *
 * Side Effects:    None
 ********************************************************************/
void CANSetMask(enum CAN_MASK code,
                unsigned long Value,
                enum CAN_CONFIG_FLAGS type);

////////////////////////////////////////////////////////////////////////

/********************************************************************
 * Function:        void CANSetFilter(enum CAN_FILTER code,
 *                                    unsigned long val,
 *                                    enum CAN_CONFIG type)
 *
 * PreCondition:    MCU must be in Configuration mode.  If not, all
 *                  values will be ignored.
 *
 * Input:           code    - One of CAN_FILTER value
 *                  val     - Actual filter register value.
 *                  type    - Type of message to filter either
 *                            CAN_CONFIG_XTD_MSG or CAN_CONFIG_STD_MSG
 *
 * Output:          Given value is bit adjusted to appropriate buffer
 *                  filter registers.
 *
 * Side Effects:    None
 ********************************************************************/
void CANSetFilter( enum CAN_FILTER code,
                   unsigned long Value,
                   enum CAN_CONFIG_FLAGS);

////////////////////////////////////////////////////////////////////////

/********************************************************************
 * Function:        BOOL CANSendMessage(unsigned long id,
 *                                      BYTE *Data,
 *                                      BYTE DataLen,
 *                                      enum CAN_TX_MSG_FLAGS MsgFlags)
 *
 * PreCondition:    None
```

```
 *
 * Input:           id          - CAN message identifier.
 *                                Only 11 or 29 bits may be used
 *                                depending on standard or extended
 *                                message type.
 *                  Data        - Data bytes of upto 8 bytes in length
 *                  DataLen     - Data length from 1 thru 8.
 *                  MsgFlags    - One or CAN_TX_MSG_FLAGS values ANDed
 *                                together
 *
 * Output:          If at least one empty transmit buffer is found,
 *                  given message is queued to be transmitted. If none
 *                  found FALSE value is returned.
 *
 * Side Effects:    None
 *
 ***************************************************************/
BOOL CANSendMessage(unsigned long id,
                    BYTE *Data,
                    BYTE DataLen,
                    enum CAN_TX_MSG_FLAGS MsgFlags);

////////////////////////////////////////////////////////////////

/***************************************************************
 * Function:        BOOL CANReceiveMessage(unsigned long *id,
 *                                         BYTE *Data,
 *                                         BYTE *DataLen,
 *                                         enum CAN_RX_MSG_FLAGS *MsgFlags)
 *
 * PreCondition:    None
 *
 * Input:           None
 *
 * Output:          id          - CAN message identifier.
 *                  Data        - Data bytes of upto 8 bytes in length
 *                  DataLen     - Data length from 1 thru 8.
 *                  MsgFlags    - One or CAN_RX_MSG_FLAGS values ANDed
 *                                together
 *
 * Output:          If at least one full receive buffer is found,
 *                  it is extrated and returned. If none found FALSE
 *                  value is returned.
 *
 * Side Effects:    None
 *
 ***************************************************************/
BOOL CANReceiveMessage(unsigned long* id,
                       BYTE *Data,
                       BYTE *DataLen,
                       enum CAN_RX_MSG_FLAGS *MsgFlags);

////////////////////////////////////////////////////////////////

/***************************************************************
 * Macro:           BYTE CANGetTxErrorCount()
 *
 * PreCondition:    None
 *
 * Input:           None
 *
 * Output:          Current transmit error count as defined by
 *                  CAN specifications.
 *
 * Side Effects:    None
 *
```

```
 ********************************************************************/
#define CANGetTxErrorCount()    (TXERRCNT)

////////////////////////////////////////////////////////////////////////

/********************************************************************
 * Macro:           BYTE CANGetRxErrorCount()
 *
 * PreCondition:    None
 *
 * Input:           None
 *
 * Output:          Current receive error count as defined by
 *                  CAN specifications.
 *
 * Side Effects:    None
 *
 ********************************************************************/
#define CANGetRxErrorCount()    (RXERRCNT)

////////////////////////////////////////////////////////////////////////

/********************************************************************
 * Macro:           BOOL CANIsBusOff()
 *
 * PreCondition:    None
 *
 * Input:           None
 *
 * Output:          TRUE if CAN Module is off due to excessive error
 *                  FALSE is it is not off.
 *
 * Side Effects:    None
 *
 ********************************************************************/
#define CANIsBusOff()           (COMSTAT_TXB0)

////////////////////////////////////////////////////////////////////////

/********************************************************************
 * Macro:           BOOL CANIsTxPassive()
 *
 * PreCondition:    None
 *
 * Input:           None
 *
 * Output:          TRUE if CAN transmit module is error passive as
 *                  defined by CAN specifications.
 *
 * Side Effects:    None
 *
 ********************************************************************/
#define CANIsTxPassive()        (COMSTAT_TXBP)

////////////////////////////////////////////////////////////////////////

/********************************************************************
 * Macro:           BYTE CANIsRxPassive()
 *
 * PreCondition:    None
 *
 * Input:           None
 *
 * Output:          TRUE if CAN receive module is error active as
 *                  defined by CAN specifications.
 *
```

```
 * Side Effects:    None
 *
 ********************************************************************/
#define CANIsRxPassive()        (COMSTAT_RXBP)

////////////////////////////////////////////////////////////////////

/********************************************************************
 * Macro:           void CANAbortAll()
 *
 * PreCondition:    None
 *
 * Input:           None
 *
 * Output:          None
 *
 * Side Effects:    None
 *
 ********************************************************************/
#define CANAbortAll()           (CANCON_ABAT = 1)

////////////////////////////////////////////////////////////////////

/********************************************************************
 * Macro:           BOOL  CANIsRxReady()
 *
 * PreCondition:    None
 *
 * Input:           None
 *
 * Output:          TRUE if at least one of the CAN receive buffer is
 *                  empty FALSE if none receive buffers are empty.
 *
 * Side Effects:    None
 *
 ********************************************************************/
#define CANIsRxReady()          (RXB0CON_RXFUL || RXB1CON_RXFUL)

////////////////////////////////////////////////////////////////////

/********************************************************************
 * Macro:           BOOL CANIsTxReady()
 *
 * PreCondition:    None
 *
 * Input:           None
 *
 * Output:          TRUE if at least one CAN transmit buffer is empty
 *                  FALSE if all CAN transmit buffers are full
 *
 * Side Effects:    None
 *
 ********************************************************************/
#define CANIsTxReady()          (!TXB0CON_TXREQ || \
                                 !TXB1CON_TXREQ || \
                                 !TXB2CON_TXREQ )

#endif      // CAN18XX8_H
```

Listado A.9: Fichero CAN18XX8.c.

```
/*********************************************************************
 *
 *              PIC18CXX8 CAN C Library Source Code
 *
 *********************************************************************
 * FileName:        CAN18CXX8.C
 * Dependencies:    CAN18CXX8.h
 * Date:            09/06/00
 * Processor:       PIC18CXX8
 * Complier:        MPLAB 5.11.00
 * Company:         Microchip Technology, Inc.
 *
 * Software License Agreement
 *
 * The software supplied herewith by Microchip Technology Incorporated
 * (the Company) for its PICmicro Microcontroller is intended and
 * supplied to you, the Company s customer, for use solely and
 * exclusively on Microchip PICmicro Microcontroller products. The
 * software is owned by the Company and/or its supplier, and is
 * protected under applicable copyright laws. All rights are reserved.
 * Any use in violation of the foregoing restrictions may subject the
 * user to criminal sanctions under applicable laws, as well as to
 * civil liability for the breach of the terms and conditions of this
 * license.
 *
 * THIS SOFTWARE IS PROVIDED IN AN, AS IS, CONDITION. NO WARRANTIES,
 * WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
 * TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
 * PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
 * IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
 * CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
 *
 * Author                Date    Comment
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * Nilesh Rajbharti     9/6/00  Original        (Rev 1.0)
 * Nilesh Rajbharti     12/1/00 Fixed bugs.
 *                              (CANRegsToID, CANReceiveMessage,
 *                               CANSetMask + Added
 *                               CAN_CONFIG_DBL_BUFFER_ON/OFF option)
 * Nilesh Rajbharti     6/8/01  Renamed CAN_CONFIG_DBL_BUFFERED to
 *                              CAN_RX_DBL_BUFFERED
 * Nilesh Rajbharti     10/11/01 Added support for HITECH compiler
 *                              Modified CAN_MESSAGE_ID def to be
 *                              compatible with HITECH Compiler
 *                              (Rev 1.1)
 * Nilesh Rajbharti     6/5/02  Modified RXB0DLC_RTR def to fix compiler
 *                              missing def problem.
 *                              Fixed CANIDToRegs() where "static"
 *                              mode would not compile.  (Rev 1.2)
 *
 *********************************************************************/
#include <usart.h>
#include "can18xx8.h"
#include "debug.h"

#if defined(MCHP_C18)
    #include <p18cxxx.h>    // p18cxxx.h must have current processor
                            // defined.
#endif

#if defined(HITECH_C18)
    #include <pic18.h>
#endif
```

```
#if defined(MCHP_C18)
    #define RXB0CON_RX0DBEN     RXB0CONbits.RXB0DBEN
    #define BRGCON2_SAM         BRGCON2bits.SAM
    #define BRGCON2_SEG2PHTS    BRGCON2bits.SEG2PHTS
    #define BRGCON3_WAKFIL      BRGCON3bits.WAKFIL
    #define PIR3_RXB0IF         PIR3bits.RXB0IF
    #define COMSTAT_RX0OVFL     COMSTATbits.RXB0OVFL
    #define PIR3_RXB1IF         PIR3bits.RXB1IF
    #define COMSTAT_RX1OVFL     COMSTATbits.RXB1OVFL
    #define RXB0DLC_RTR         (RXB0DLC & 0x40) // RXB0DLCbits.RTR
    #define RXB0SIDL_EXID       RXB0SIDLbits.EXID
    #define PIR3_IRXIF          PIR3bits.IRXIF
#endif

#if defined(HITECH_C18)
    #define RXB0CON_RX0DBEN     RXB0DBEN
    #define BRGCON2_SAM         SAM
    #define BRGCON2_SEG2PHTS    SEG2PHTS
    #define BRGCON3_WAKFIL      WAKFIL
    #define PIR3_RXB0IF         RXB0IF
    #define COMSTAT_RX0OVFL     RXB0OVFL
    #define PIR3_RXB1IF         RXB1IF
    #define COMSTAT_RX1OVFL     RXB1OVFL
    #define RXB0DLC_RTR         RXB0RXRTR
    #define RXB0SIDL_EXID       RXB0EXID
    #define PIR3_IRXIF          IRXIF
#endif



/*
 * Private helper functions to convert 32-bit CAN ID value into
 * corresponding PIC18CXX8 registers and vice-versa.
 */

static void CANIDToRegs(BYTE* ptr,
                        unsigned long val,
                        enum CAN_CONFIG_FLAGS type);
static void RegsToCANID(BYTE* ptr,
                        unsigned long *val,
                        enum CAN_CONFIG_FLAGS type);


/*******************************************************************
 * Function:        void CANInitialize(BYTE SJW,
 *                                     BYTE BRP,
 *                                     BYTE PHSEG1,
 *                                     BYTE PHSEG2,
 *                                     BYTE PROPSEG,
 *                                     enum CAN_CONFIG_FLAGS flags)
 *
 * PreCondition:    MCU must be in Configuration mode or else these
 *                  values will be ignored.
 *
 * Input:           SJW     - SJW value as defined in 18CXX8 datasheet
 *                              (Must be between 1 thru 4)
 *                  BRP     - BRP value as defined in 18CXX8 datasheet
 *                              (Must be between 1 thru 64)
 *                  PHSEG1  - PHSEG1 value as defined in 18CXX8
 *                            datasheet
 *                              (Must be between 1 thru 8)
 *                  PHSEG2  - PHSEG2 value as defined in 18CXX8
 *                            datasheet
 *                              (Must be between 1 thru 8)
```

```
*                   PROPSEG - PROPSEG value as defined in 18CXX8
*                             datasheet
*                               (Must be between 1 thru 8)
*                   flags   - Value of type enum CAN_CONFIG_FLAGS
*
* Output:           CAN bit rate is set. All masks registers are set
*                   '0' to allow all messages.
*                   Filter registers are set according to flag value.
*                   If (config & CAN_CONFIG_VALID_XTD_MSG)
*                       Set all filters to XTD_MSG
*                   Else if (config & CONFIG_VALID_STD_MSG)
*                       Set all filters to STD_MSG
*                   Else
*                       Set half of the filters to STD while rests to
*                       XTD_MSG.
*
* Side Effects:     All pending transmissions are aborted.
*
*******************************************************************/
void CANInitialize(BYTE SJW,
                   BYTE BRP,
                   BYTE PHSEG1,
                   BYTE PHSEG2,
                   BYTE PROPSEG,
                   enum CAN_CONFIG_FLAGS config)
{
    BYTE FilterConfig1;
    BYTE FilterConfig2;

   // In order to setup necessary config parameters of CAN module,
   // it must be in CONFIG mode.
   CANSetOperationMode(CAN_OP_MODE_CONFIG);

   // Now set the baud rate.
   CANSetBaudRate(SJW,
                  BRP,
                  PHSEG1,
                  PHSEG2,
                  PROPSEG,
                  config);

   RXB0CON = config & CAN_CONFIG_MSG_BITS;
   if ( (config & CAN_CONFIG_DBL_BUFFER_BIT)
             == CAN_CONFIG_DBL_BUFFER_ON )
       RXB0CON_RX0DBEN = 1;


   RXB1CON = RXB0CON;

   // Set default filter and mask registers for all receive buffers.
   CANSetMask(CAN_MASK_B1, 0, CAN_CONFIG_XTD_MSG);
   CANSetMask(CAN_MASK_B2, 0, CAN_CONFIG_XTD_MSG);

   switch( (config & CAN_CONFIG_MSG_BITS) | ~CAN_CONFIG_MSG_BITS )
   {
   case CAN_CONFIG_VALID_XTD_MSG:
       FilterConfig1 = CAN_CONFIG_XTD_MSG;
       FilterConfig2 = CAN_CONFIG_XTD_MSG;
       break;

   case CAN_CONFIG_VALID_STD_MSG:
       FilterConfig1 = CAN_CONFIG_STD_MSG;
       FilterConfig2 = CAN_CONFIG_STD_MSG;
       break;
   default:
       FilterConfig1 = CAN_CONFIG_STD_MSG;
```

```
            FilterConfig2 = CAN_CONFIG_XTD_MSG;
            break;
200     }

        // By default, there will be no mask on any receive filters,
        // hence filter value of '0' will be ignored.
        CANSetFilter(CAN_FILTER_B1_F1, 0, FilterConfig1);
        CANSetFilter(CAN_FILTER_B1_F2, 0, FilterConfig1);
        CANSetFilter(CAN_FILTER_B2_F1, 0, FilterConfig2);
        CANSetFilter(CAN_FILTER_B2_F2, 0, FilterConfig2);
        CANSetFilter(CAN_FILTER_B2_F3, 0, FilterConfig2);
        CANSetFilter(CAN_FILTER_B2_F4, 0, FilterConfig2);
210
        // Restore to Normal mode.
        CANSetOperationMode(CAN_OP_MODE_NORMAL);
    }

    ////////////////////////////////////////////////////////////////////

    /*******************************************************************
     * Function:        void CANSetOperationMode(CAN_OP_MODE mode)
     *
220  * PreCondition:    None
     *
     * Input:           mode    - Operation mode code
     *                            must be of type enum CAN_OP_MODES
     *
     * Output:          MCU is set to requested mode
     *
     * Side Effects:    None
     *
     * Overview:        Given mode byte is copied to CANSTAT and made
230  *                  sure that requested mode is set.
     *
     * Note:            This is a blocking call.  It will not return until
     *                  requested mode is set.
     *******************************************************************/
    void CANSetOperationMode(enum CAN_OP_MODE mode)
    {
         // Request desired mode.
         CANCON = mode;

240     // Wait till desired mode is set.
        while( (CANSTAT & CAN_OP_MODE_BITS) != mode );
    }

    ////////////////////////////////////////////////////////////////////

    /*******************************************************************
     * Function:        void CANSetBaudRate(BYTE SJW,
     *                                      BYTE BRP,
     *                                      BYTE PHSEG1,
250  *                                      BYTE PHSEG2,
     *                                      BYTE PROPSEG,
     *                                      enum CAN_CONFIG_FLAGS flags)
     *
     * PreCondition:    MCU must be in Configuration mode or else these
     *                  values will be ignored.
     *
     * Input:           SJW     - SJW value as defined in 18CXX8 datasheet
     *                              (Must be between 1 thru 4)
     *                  BRP     - BRP value as defined in 18CXX8 datasheet
260  *                              (Must be between 1 thru 64)
     *                  PHSEG1  - PHSEG1 value as defined in 18CXX8
     *                            datasheet
     *                              (Must be between 1 thru 8)
```

```
 *                  PHSEG2  - PHSEG2 value as defined in 18CXX8
 *                            datasheet
 *                              (Must be between 1 thru 8)
 *                  PROPSEG - PROPSEG value as defined in 18CXX8
 *                            datasheet
 *                              (Must be between 1 thru 8)
 *                  flags   - Value of type enum CAN_CONFIG_FLAGS
 *
 * Output:          CAN bit rate is set as per given values.
 *
 * Side Effects:    None
 *
 * Overview:        Given values are bit adjusted to fit in 18CXX8
 *                  BRGCONx registers and copied.
 *
 ********************************************************************/
void CANSetBaudRate(BYTE SJW,
                    BYTE BRP,
                    BYTE PHSEG1,
                    BYTE PHSEG2,
                    BYTE PROPSEG,
                    enum CAN_CONFIG_FLAGS flags)
{
    // Actual values are offset '0'.
    // Hence map given values from offset '1' to offset '0'
    SJW--;
    BRP--;
    PHSEG1--;
    PHSEG2--;
    PROPSEG--;

   // Bit adjust given values into their appropriate registers.
   BRGCON1 = SJW << 6;
   BRGCON1 |= BRP;

   BRGCON2 = PHSEG1 << 3;
   BRGCON2 |= PROPSEG;

   if ( !(flags & CAN_CONFIG_SAMPLE_BIT) )
       BRGCON2_SAM = 1;

   if ( flags & CAN_CONFIG_PHSEG2_PRG_BIT )
       BRGCON2_SEG2PHTS = 1;


   BRGCON3 = PHSEG2;
   if ( flags & CAN_CONFIG_LINE_FILTER_BIT )
       BRGCON3_WAKFIL = 1;

}

////////////////////////////////////////////////////////////////////////
/********************************************************************
 * Function:        void CANIDToRegs(BYTE* ptr,
 *                                   unsigned long val,
 *                                   enum CAN_CONFIG_FLAGS type)
 *
 * PreCondition:    None
 *
 * Input:           ptr     - Starting address of a buffer to be updated
 *                  val     - 32-bit value to be converted
 *                  type    - Type of message - either
 *                            CAN_CONFIG_XTD_MSG or CAN_CONFIG_STD_MSG
 *
 * Output:          Given CAN id value 'val' is bit adjusted and copied
 *                  into corresponding PIC18CXX8 CAN registers
```

```
330  *
     * Side Effects:    None
     *
     * Overview:        If given id is of type standard identifier,
     *                  only SIDH and SIDL are updated
     *                  If given id is of type extended identifier,
     *                  bits val<17:0> is copied to EIDH, EIDL and SIDH<1:0>
     *                  bits val<28:18> is copied to SIDH and SIDL
     *
     ******************************************************************/
340 static void CANIDToRegs(BYTE* ptr,
                           unsigned long val,
                           enum CAN_CONFIG_FLAGS type)
    {
        CAN_MESSAGE_ID *Value;
        Value =  (CAN_MESSAGE_ID*)&val;

        if ( type & CAN_CONFIG_MSG_TYPE_BIT )
        {
            // Standard Identifier
350         *ptr = Value->BYTES.BYTE_1 >> 3;        // Copy SID<7:3> to SIDH<4:0>
            *ptr |= (Value->BYTES.BYTE_2 << 5);     // Copy SID<10:8> to SIDH<7:5>
            ptr++;                                  // Point to SIDL
            *ptr = Value->BYTES.BYTE_1 << 5;        // Copy SID<2:0> to SIDL<7:5>
        }
        else
        {
            // Extended Identifier
            *ptr = Value->BYTES.BYTE_3 >> 5;        // Copy EID<23:21> to SIDH<2:0>
            *ptr |= Value->BYTES.BYTE_4 << 3;       // Copy EID<28:24> to SIDH<7:3>
360         ptr++;                                  // Point to SIDL
            *ptr = (Value->BYTES.BYTE_3 << 3) & 0xE0; // Copy EID<20:18> to SIDL<7:5>
                                                    // mask out EID<17:16> bits
            *ptr |= 0b00001000;                     // Set EXIDEN bit  to SIDL<3>
            *ptr |= Value->BYTES.BYTE_3 & 0x03;     // Copy EID<17:16> to SIDL<1:0>
            ptr++;                                  // Point to EIDH
            *ptr = Value->BYTES.BYTE_2;             // Copy EID<15:8> to EIDH<7:0>
            ptr++;                                  // Point to EIDL
            *ptr = Value->BYTES.BYTE_1;             // Copy EID<7:0> to EIDL<7:0>
        }
370 }

    ////////////////////////////////////////////////////////////////////
    /******************************************************************
     * Function:        void RegsToCANID(BYTE *ptr,
     *                                   unsigned long *val,
     *                                   enum CAN_CONFIG_FLAGS type)
     *
     * PreCondition:    None
     *
380  * Input:           ptr     - Starting address of a buffer to be updated
     *                  val     - 32-bit buffer to hold value
     *                  type    - Type of message - either
     *                            CAN_CONFIG_XTD_MSG or CAN_CONFIG_STD_MSG
     *
     * Output:          CAN registers starting at given address are bit
     *                  adjusted and copied into 'val'
     *
     * Side Effects:    None
     *
390  * Overview:        If given id is of type standard identifier,
     *                  only SIDH and SIDL are used
     *                  If given id is of type extended identifier,
     *                  bits EIDH, EIDL and SIDL<1:0> is copied to val<17:0>
     *                  bits SIDH and SIDL is copied to val<28:18>
     *
```

```
 ********************************************************************/
static void RegsToCANID(BYTE* ptr,
                        unsigned long *val,
                        enum CAN_CONFIG_FLAGS type)
{
    CAN_MESSAGE_ID *Value;

    Value = (CAN_MESSAGE_ID*)val;

    if ( type & CAN_CONFIG_MSG_TYPE_BIT )
    {
        // Standard Identifier
        Value->BYTES.BYTE_1 = (*ptr << 3);      // Copy SIDH<4:0> to SID<7:3>
        Value->BYTES.BYTE_2 = *ptr >> 5;        // Copy SIDH<7:5> to SID<10:8>
        ptr++;                                  // Point to SIDL
        Value->BYTES.BYTE_1 |= (*ptr >> 5);     // Copy SIDL<7:6> to SID<2:0>
        Value->BYTES.BYTE_3 = 0x00;
        Value->BYTES.BYTE_4 = 0x00;
    }
    else
    {
        // Extended Identifier
        Value->BYTES.BYTE_3 = (*ptr << 5);      // Copy SIDH<2:0> to EID<23:21>
        Value->BYTES.BYTE_4 = (*ptr >> 3);      // Copy SIDH<7:3> to EID<29:25>
        ptr++;                          // Point to SIDL
        Value->BYTES.BYTE_3 |= (*ptr & 0x03);   // Copy SIDH<1:0> to EID<17:16>
        // Bug-Fix NKR 11/20/00
        Value->BYTES.BYTE_3 |= ((*ptr & 0xe0) >> 3);  // Copy SIDL<7:6> to EID<20:18>
        ptr++;                          // Point to EIDH
        Value->BYTES.BYTE_2 = *ptr;             // Copy EIDH<15:8> to EID<15:8>
        ptr++;                                  // Point to EIDL
        Value->BYTES.BYTE_1 = *ptr;             // Copy EIDH<7:0> to EID<7:0>
    }
}


/*******************************************************************
 * Function:        void CANSetMask(enum CAN_MASK code,
 *                                  unsigned long val,
 *                                  enum CAN_CONFIG_FLAGS type)
 *
 * PreCondition:    MCU must be in Configuration mode.  If not, all
 *                  values will be ignored.
 *
 * Input:           code    - One of CAN_MASK value
 *                  val     - Actual mask register value.
 *                  type    - Type of message to filter either
 *                            CAN_CONFIG_XTD_MSG or CAN_CONFIG_STD_MSG
 *
 * Output:          Given value is bit adjusted to appropriate buffer
 *                  mask registers.
 *
 * Side Effects:    None
 *
 ********************************************************************/
 void CANSetMask(enum CAN_MASK code,
                 unsigned long val,
                 enum CAN_CONFIG_FLAGS type)
 {
     BYTE *ptr;

    // Select appropriate starting address based on given CAN_MASK
    // value.
    if ( code == CAN_MASK_B1 )
        ptr = (BYTE*)&RXM0SIDH;
```

```
    else
        ptr = (BYTE*)&RXM1SIDH;

    // Convert given 32-bit id value into corresponding register values.
    CANIDToRegs(ptr, val, type);
}

////////////////////////////////////////////////////////////////////
/*******************************************************************
 * Function:        void CANSetFilter(enum CAN_FILTER code,
 *                                    unsigned long val,
 *                                    enum CAN_CONFIG type)
 *
 * PreCondition:    MCU must be in Configuration mode.  If not, all
 *                  values will be ignored.
 *
 *
 Input:             code     - One of CAN_FILTER value
 *                  val      - Actual filter register value.
 *                  type     - Type of message to filter either
 *                             CAN_CONFIG_XTD_MSG or CAN_CONFIG_STD_MSG
 *
 * Output:          Given value is bit adjusted to appropriate buffer
 *                  filter registers.
 *
 * Side Effects:    None
 *******************************************************************/
 void CANSetFilter(enum CAN_FILTER code,
                   unsigned long val,
                   enum CAN_CONFIG_FLAGS type)
 {
    BYTE *ptr;

    // Select appropriate starting address based on given CAN_FILTER
    // code.
    switch(code)
    {
    case CAN_FILTER_B1_F1:
        ptr = (BYTE*)&RXF0SIDH;
        break;

    case CAN_FILTER_B1_F2:
        ptr = (BYTE*)&RXF1SIDH;
        break;

    case CAN_FILTER_B2_F1:
        ptr = (BYTE*)&RXF2SIDH;
        break;

    case CAN_FILTER_B2_F2:
        ptr = (BYTE*)&RXF3SIDH;
        break;

    case CAN_FILTER_B2_F3:
        ptr = (BYTE*)&RXF4SIDH;
        break;

    default:
        ptr = (BYTE*)&RXF5SIDH;
        break;
    }

    // Convert 32-bit value into register values.
    CANIDToRegs(ptr, val, type);
```

```
}
530 ////////////////////////////////////////////////////////////////////

/*******************************************************************
 * Function:        BOOL CANSendMessage(unsigned long id,
 *                                      BYTE *Data,
 *                                      BYTE DataLen,
 *                                      enum CAN_TX_MSG_FLAGS MsgFlags)
 *
 * PreCondition:    None
 *
540 * Input:          id          - CAN message identifier.
 *                                Only 11 or 29 bits may be used
 *                                depending on standard or extended
 *                                message type.
 *                  Data        - Data bytes of upto 8 bytes in length
 *                  DataLen     - Data length from 1 thru 8.
 *                  MsgFlags    - One or CAN_TX_MSG_FLAGS values ANDed
 *                                together
 *
 * Output:          If at least one empty transmit buffer is found,
550 *                  given message is queued to be transmitted. If none
 *                  found FALSE value is returned.
 *
 * Side Effects:    None
 *
 *******************************************************************/
BOOL CANSendMessage(unsigned long id,
                     BYTE* Data,
                     BYTE DataLen,
                     enum CAN_TX_MSG_FLAGS MsgFlags)
560 {
    BYTE i;
    BYTE *ptr;

    // Find the first empty transmitter.
    if ( TXB0CON_TXREQ == 0 )
    {
        // TxBuffer0 is empty.  Set WIN bits to point to TXB0
        CANCON &= 0b11110001;
        CANCON |= 0b00001000;
570 }
    else if ( TXB1CON_TXREQ == 0 )
    {
        // TxBuffer1 is empty. Set WIN bits to point to TXB1
        CANCON &= 0b11110001;
        CANCON |= 0b00000110;
    }
    else if ( TXB2CON_TXREQ == 0 )
    {
        // TxBuffer2 is empty. Set WIN bits to point to TXB2
580     CANCON &= 0b11110001;
        CANCON |= 0b00000100;
    }
    else
        // None of the transmit buffers were empty.
        return FALSE;

    /*
     * Now that WIN has remapped RXB0 to empty buffer, simply
     * populate RXB0 buffer
590  */

    // Set transmit priority.
    RXB0CON = MsgFlags & CAN_TX_PRIORITY_BITS;
```

```
    // Populate Extended identifier information only if it is
    // desired.
    if ( !(MsgFlags & CAN_TX_FRAME_BIT) )
        CANIDToRegs((BYTE*)&RXB0SIDH, id, CAN_CONFIG_XTD_MSG);
    else
        CANIDToRegs((BYTE*)&RXB0SIDH, id, CAN_CONFIG_STD_MSG);

    RXB0DLC = DataLen;

    if ( !(MsgFlags & CAN_TX_RTR_BIT) )
        RXB0DLC |= 0b01000000;

    // Populate data values.
    ptr = (BYTE*)&RXB0D0;
    for ( i = 0; i < DataLen; i++ )
        ptr[i] = Data[i];

    /*
     * Mark this buffer as ready to start transmit.
     * We are not using C bit field structure because RXB0 registers
     * are remapped to one of the empty transmit buffers and their
     * bit3 is not same as RXB0CON bit3.  To avoid confusion, in-line
     * assembly is used to directly set bit 3 of corresponding TXBnCON
     * register.
     */
#if defined(MCHP_C18)
     _asm
     bsf RXB0CON, 3, 0
    _endasm
#endif
#if defined(HITECH_C18)
       asm("bsf␣_RXB0CON,3");
#endif

    /*
     * Restore CAN buffer mapping so that subsequent access to RXB0
     * buffers are to the real RXB0 buffer.
     */
    CANCON &= 0b11110001;

    return TRUE;
}

////////////////////////////////////////////////////////////////////////

/*********************************************************************
 * Function:        BOOL CANReceiveMessage(unsigned long *id,
 *                                         BYTE *Data,
 *                                         BYTE DataLen,
 *                                         enum CAN_RX_MSG_FLAGS MsgFlags)
 *
 * PreCondition:    None
 *
 * Input:           None
 *
 * Output:          id          - CAN message identifier.
 *                  Data        - Data bytes of upto 8 bytes in length
 *                  DataLen     - Data length from 1 thru 8.
 *                  MsgFlags    - One or CAN_RX_MSG_FLAGS values ANDed
 *                                together
 *
 * Output:          If at least one full receive buffer is found,
 *                  it is extrated and returned.
 *                  If none found FALSE value is returned.
 *
```

```
660  * Side Effects:    None
     *
     ********************************************************************/
BOOL CANReceiveMessage(unsigned long *id,
                       BYTE *Data,
                       BYTE *DataLen,
                       enum CAN_RX_MSG_FLAGS *MsgFlags)
{
    BYTE i;
    BYTE *ptr;
670 BOOL lbIsItBuffer0;

    // Start with no error or flags set.
    *MsgFlags = 0x0;

    // Find which buffer is ready.
    if ( RXB0CON_RXFUL )
    {
        // RXBuffer0 is full.
        CANCON &= 0b11110001;
680
        lbIsItBuffer0 = TRUE;

        // Clear the received flag.
        PIR3_RXB0IF = 0;

        // Record and forget any previous overflow
        if ( COMSTAT_RX0OVFL )
        {
            *MsgFlags |= CAN_RX_OVERFLOW;
690         COMSTAT_RX0OVFL = 0;
        }

        if ( RXB0CON_RX0DBEN )
        {
            *MsgFlags |= RXB0CON & CAN_RX_FILTER_BITS;
            *MsgFlags &= 0x01;
        }
    }
    else if ( RXB1CON_RXFUL )
700 {
        // RXBuffer1 is full
        CANCON &= 0b11110001;
        CANCON |= 0b00001010;

        lbIsItBuffer0 = FALSE;

        // Clear the received flag.
        PIR3_RXB1IF = 0;

710     // Record and forget any previous overflow
        if ( COMSTAT_RX1OVFL )
        {
            *MsgFlags |= CAN_RX_OVERFLOW;
            COMSTAT_RX1OVFL = 0;
        }

        *MsgFlags |= RXB1CON & CAN_RX_FILTER_BITS;
        if ( *MsgFlags < 0x02 )
            *MsgFlags |= CAN_RX_DBL_BUFFERED;
720 }
    else
        return FALSE;

    // Retrieve message length.
    *DataLen = RXB0DLC & 0b00001111;
```

```
    // Determine whether this was RTR or not.
    if ( RXB0DLC_RTR )
        *MsgFlags |= CAN_RX_RTR_FRAME;
730
    // Retrieve EIDX bytes only if this is extended message
    if ( RXB0SIDL_EXID )
    {
        *MsgFlags |= CAN_RX_XTD_FRAME;

        RegsToCANID((BYTE*)&RXB0SIDH, id, CAN_CONFIG_XTD_MSG);
    }
    else
        RegsToCANID((BYTE*)&RXB0SIDH, id, CAN_CONFIG_STD_MSG);
740
    // Get message data itself
    ptr = (BYTE*)&RXB0D0;
    for ( i = 0; i < *DataLen; i++ )
        Data[i] = ptr[i];

    // Restore default RXB0 mapping.
    CANCON &= 0b11110001;

    // Record and Clear any previous invalid message bit flag.
750 if ( PIR3_IRXIF )
    {
        *MsgFlags |= CAN_RX_INVALID_MSG;
        PIR3_IRXIF = 0;
    }

    if ( lbIsItBuffer0 )
        RXB0CON_RXFUL = 0;
    else
        RXB1CON_RXFUL = 0;
760
    return TRUE;
}
```

A.7 Código fuente librería XLCD

Listado A.10: Fichero xlcd.h.

```
//-----------------------------------------------------------------------
//
//                          DESCRIPCION
//
// Libreria para utilidar tarjetas MMC card con el protocolo SPI
//
//-----------------------------------------------------------------------
//
//    Archivo:              mmc.h
//
//    Autores:      Miguel Angel Perez
//                                  Carlos Gobernado
//
//    Version: 2.11 1-9-2004
//
//
// Las tarjetas MMC trabajan en el modo CKP=1 CKE=0 luego MODO 11
// y se muestrea en la mitad del bit
//
// Version 2.11:
//      Se ha procedido a una limpieza y revision de codigo.
//
// Version 2.10:
//      Se ha introducido las modificaciones necesarias para trabajar
//      con SD-CARDs.
//
//  Version 2.00:
//      Ante los problemas puntuales con distintos modelos de tarjetas
//      se ha decidido hacer una revision en profundidad de este codigo.
//      Se ha introducido pin de habilitacion de alimentacion de tarjeta y
//      funciones correspondientes.
//      EnviaComandoMMC se ha introducido un retardo de 8 ciclos antes de
//      devolver la respuesta.
//      Por las pruebas realizadas este es un codigo mas robusto.
//
//  Version 1.11:
//      Se ha introducido las macros SPI_CS_ON() y SPI_CS_OFF()
//      Se han introducido casi todos los comandos para una posible
//      ampliacion.
//      La habilitacion CS se levanta y se baja en los comandos
//      La funcion Borrar512 ya funciona.
//
//  Version 1.10:
//      La funcion Borrar512 no he conseguido hacerla funcionar y no se pq
//      No funcionan con tarjetas SD CARD
//
//-----------------------------------------------------------------------

#ifndef __MMC_H
#define __MMC_H

//-----------------------------------------------------------------------
// Definicion de constantes
//-----------------------------------------------------------------------

#define ERROR       0
#define OK          1

//-----------------------------------------------------------------------
//  Pines utilizados para habilitacion y senial de alimentacion
//-----------------------------------------------------------------------
```

```
#define SPI_CS              PORTCbits.RC2
#define EN_VCC              PORTCbits.RC0
#define TRIS_EN_VCC         TRISCbits.TRISC0
#define TRIS_CS             TRISCbits.TRISC2

//-----------------------------------------------------------------------
// Definicion de comandos MMC/SD
//-----------------------------------------------------------------------

#define GO_IDLE_STATE               0x40
#define SEND_OP_COND                0x41
#define SEND_CSD                    0x49
#define SEND_CID                    0x4A
#define STOP_TRANSMISSION           0x4C
#define SEND_STATUS                 0X4D
#define SET_BLOCKLEN                0x50
#define READ_SINGLE_BLOCK           0x51
#define READ_MULTIPLE_BLOCK         0x52
#define SET_BLOCK_COUNT             0x57
#define WRITE_SINGLE_BLOCK          0x58
#define WRITE_MULTIPLE_BLOCK        0x59
#define PROGRAM_CSD                 0x5B
#define SET_WRITE_PORT              0x5C
#define CLR_WRITE_PORT              0x5D
#define SEND_WRITE_PROT             0x5E
#define TAG_ERASE_START             0x63
#define TAG_ERASE_END               0x64
#define ERASE                       0x66
#define LOCKUNLOCK                  0x6A
#define APP_CMD                     0x77
#define GEN_CMD                     0x78
#define READ_OCR                    0x7A
#define CRC_ON_OFF                  0x7B
#define SD_SEND_OP_COND             0x41

//-----------------------------------------------------------------------
// Definicion de respuestas a comandos
//-----------------------------------------------------------------------
#define R1_OK                       0b00000000
#define R1_IDLE_STATE               0b00000001
#define R1_ERASE_RESET              0b00000010
#define R1_ILLEGAL_COMMAND          0b00000100
#define R1_CRC_ERROR                0b00001000
#define R1_ERARSE_SEQUENCE_ERROR    0b00010000
#define R1_ADDRESS_ERROR            0b00100000
#define R1_PARAMETER_ERROR          0b01000000

//-----------------------------------------------------------------------
// Definicion de macros
//-----------------------------------------------------------------------
#define SPI_CS_ON()         SPI_CS=0
#define SPI_CS_OFF()        SPI_CS=1
#define CardPowerOff()      EN_VCC=0;
#define INTENTOS            50          // Es el numero de intentos para escritura
                                        // y lectura

//-----------------------------------------------------------------------
//Estructuras de datos
//-----------------------------------------------------------------------
typedef unsigned char BYTE;
typedef unsigned char byte;
typedef unsigned char uchar;
typedef unsigned long ulong;

//-----------------------------------------------------------------------
// Prototipos de funciones
```

```
//-----------------------------------------------------------------------
130
char InitMMC(void);
void CardPowerOn(void);
/*
void CardPowerOff(); es una macro ya que solo modifica una pata
*/
char EnviaComandoMMC(char,ulong,char);
char DatoSPI(char);
char Lectura512(char*,ulong);
char Escritura512(char*,ulong);
140 char Borrar512(ulong,ulong);
void CardDelay(uchar);

//char IdMMC(char*);
//char CsdMMC(char*);
//int  CapacidadMMC();
//char ModeloMMC(char *);
```

Listado A.11: Fichero xlcd.c.

```
/* $Id: busyxlcd.c,v 1.3 2000/02/07 23:26:34 ConnerJ Exp $ */
#include <p18cxxx.h>
#include "xlcd.h"

/*******************************************************************
*       Function Name:  BusyXLCD                                   *
*       Return Value:   char: busy status of LCD controller        *
*       Parameters:     void                                       *
*       Description:    This routine reads the busy status of the  *
*                       Hitachi HD44780 LCD controller.            *
*******************************************************************/
unsigned char BusyXLCD(void)
{
        RW_PIN = 1;                     // Set the control bits for read
        RS_PIN = 0;
        DelayFor18TCY();
        E_PIN = 1;                      // Clock in the command
        DelayFor18TCY();
#ifdef BIT8                             // 8-bit interface
        if(DATA_PORT&0x80)                      // Read bit 7 (busy bit)
        {                               // If high
                E_PIN = 0;              // Reset clock line
                RW_PIN = 0;             // Reset control line
                return 1;               // Return TRUE
        }
        else                            // Bit 7 low
        {
                E_PIN = 0;              // Reset clock line
                RW_PIN = 0;             // Reset control line
                return 0;               // Return FALSE
        }
#else                                   // 4-bit interface
#ifdef UPPER                            // Upper nibble interface
        if(DATA_PORT&0x80)
#else                                   // Lower nibble interface
        if(DATA_PORT&0x08)
#endif
        {
                E_PIN = 0;              // Reset clock line
                DelayFor18TCY();
                E_PIN = 1;              // Clock out other nibble
                DelayFor18TCY();
                E_PIN = 0;
                RW_PIN = 0;             // Reset control line
                return 1;               // Return TRUE
        }
        else                            // Busy bit is low
        {
                E_PIN = 0;              // Reset clock line
                DelayFor18TCY();
                E_PIN = 1;              // Clock out other nibble
                DelayFor18TCY();
                E_PIN = 0;
                RW_PIN = 0;             // Reset control line
                return 0;               // Return FALSE
        }
#endif
}

/*******************************************************************
*       Function Name:  OpenXLCD                                   *
*       Return Value:   void                                       *
*       Parameters:     lcdtype: sets the type of LCD (lines)      *
*       Description:    This routine configures the LCD. Based on  *
*                       the Hitachi HD44780 LCD controller. The    *
```

```
*                       routine will configure the I/O pins of the  *
*                       microcontroller, setup the LCD for 4- or    *
*                       8-bit mode and clear the display. The user  *
*                       must provide three delay routines:          *
*                       DelayFor18TCY() provides a 18 Tcy delay     *
*                       DelayPORXLCD() provides at least 15ms delay *
*                       DelayXLCD() provides at least 5ms delay     *
*********************************************************************/
void OpenXLCD(unsigned char lcdtype)
{
        // The data bits must be either a 8-bit port or the upper or
        // lower 4-bits of a port. These pins are made into inputs
#ifdef BIT8                             // 8-bit mode, use whole port
        DATA_PORT = 0;
        TRIS_DATA_PORT = 0xff;
#else                                   // 4-bit mode
#ifdef UPPER                            // Upper 4-bits of the port
        DATA_PORT &= 0x0f;
        TRIS_DATA_PORT |= 0xf0;
#else                                   // Lower 4-bits of the port
        DATA_PORT &= 0xf0;
        TRIS_DATA_PORT |= 0x0f;
#endif
#endif
        TRIS_RW = 0;                    // All control signals made outputs
        TRIS_RS = 0;
        TRIS_E = 0;
        RW_PIN = 0;                     // R/W pin made low
        RS_PIN = 0;                     // Register select pin made low
        E_PIN = 0;                      // Clock pin made low

        // Delay for 15ms to allow for LCD Power on reset
        DelayPORXLCD();

        // Setup interface to LCD
#ifdef BIT8                             // 8-bit mode interface
        TRIS_DATA_PORT = 0;             // Data port output
        DATA_PORT = 0b00110000;         // Function set cmd(8-bit interface)
#else                                   // 4-bit mode interface
#ifdef UPPER                            // Upper nibble interface
        TRIS_DATA_PORT &= 0x0f;
        DATA_PORT &= 0x0f;
        DATA_PORT |= 0b00100000;        // Function set cmd(4-bit interface)
#else                                   // Lower nibble interface
        TRIS_DATA_PORT &= 0xf0;
        DATA_PORT &= 0xf0;
        DATA_PORT |= 0b00000010;        // Function set cmd(4-bit interface)
#endif
#endif
        E_PIN = 1;                      // Clock the cmd in
        DelayFor18TCY();
        E_PIN = 0;

        // Delay for at least 4.1ms
        DelayXLCD();

        // Setup interface to LCD
#ifdef BIT8                             // 8-bit interface
        DATA_PORT = 0b00110000;         // Function set cmd(8-bit interface)
#else                                   // 4-bit interface
#ifdef UPPER                            // Upper nibble interface
        DATA_PORT &= 0x0f;              // Function set cmd(4-bit interface)
        DATA_PORT |= 0b00100000;
#else                                   // Lower nibble interface
        DATA_PORT &= 0xf0;              // Function set cmd(4-bit interface)
        DATA_PORT |= 0b00000010;
```

```
#endif
#endif
        E_PIN = 1;                      // Clock the cmd in
        DelayFor18TCY();
        E_PIN = 0;

        // Delay for at least 100us
        DelayXLCD();

        // Setup interface to LCD
#ifdef BIT8                             // 8-bit interface
        DATA_PORT = 0b00110000;         // Function set cmd(8-bit interface)
#else                                   // 4-bit interface
#ifdef UPPER                            // Upper nibble interface
        DATA_PORT &= 0x0f;              // Function set cmd(4-bit interface)
        DATA_PORT |= 0b00100000;
#else                                   // Lower nibble interface
        DATA_PORT &= 0xf0;              // Function set cmd(4-bit interface)
        DATA_PORT |= 0b00000010;
#endif
#endif
        E_PIN = 1;                      // Clock cmd in
        DelayFor18TCY();
        E_PIN = 0;

#ifdef BIT8                             // 8-bit interface
        TRIS_DATA_PORT = 0xff;          // Make data port input
#else                                   // 4-bit interface
#ifdef UPPER                            // Upper nibble interface
        TRIS_DATA_PORT |= 0xf0;         // Make data nibble input
#else                                   // Lower nibble interface
        TRIS_DATA_PORT |= 0x0f;         // Make data nibble input
#endif
#endif

        // Set data interface width, # lines, font
        while(BusyXLCD());              // Wait if LCD busy
        WriteCmdXLCD(lcdtype);          // Function set cmd

        // Turn the display on then off
        while(BusyXLCD());              // Wait if LCD busy
        WriteCmdXLCD(DOFF&CURSOR_OFF&BLINK_OFF);        // Display OFF/Blink OFF
        while(BusyXLCD());              // Wait if LCD busy
        WriteCmdXLCD(DON&CURSOR_ON&BLINK_ON);           // Display ON/Blink ON

        // Clear display
        while(BusyXLCD());              // Wait if LCD busy
        WriteCmdXLCD(0x01);             // Clear display

        // Set entry mode inc, no shift
        while(BusyXLCD());              // Wait if LCD busy
        WriteCmdXLCD(SHIFT_CUR_LEFT);   // Entry Mode

        // Set DD Ram address to 0
        while(BusyXLCD());              // Wait if LCD busy
        SetDDRamAddr(0);                // Set Display data ram address to 0

        return;
}

/*****************************************************************
*       Function Name:  putrsXLCD
*       Return Value:   void
*       Parameters:     buffer: pointer to string
*       Description:    This routine writes a string of bytes to the
*                       Hitachi HD44780 LCD controller. The user
```

```
*                       must check to see if the LCD controller is
*                       busy before calling this routine. The data
*                       is written to the character generator RAM or
*                       the display data RAM depending on what the
*                       previous SetxxRamAddr routine was called.
*******************************************************************/

void putrsXLCD(PARAM_SCLASS const MEM_MODEL rom char *buffer)
{
        while(*buffer)                  // Write data to LCD up to null
        {
                while(BusyXLCD());      // Wait while LCD is busy
                WriteDataXLCD(*buffer); // Write character to LCD
                buffer++;               // Increment buffer
        }
        return;
}

/*******************************************************************
*       Function Name:  putsXLCD
*       Return Value:   void
*       Parameters:     buffer: pointer to string
*       Description:    This routine writes a string of bytes to the
*                       Hitachi HD44780 LCD controller. The user
*                       must check to see if the LCD controller is
*                       busy before calling this routine. The data
*                       is written to the character generator RAM or
*                       the display data RAM depending on what the
*                       previous SetxxRamAddr routine was called.
*******************************************************************/
void putsXLCD(char *buffer)
{
        while(*buffer)                  // Write data to LCD up to null
        {
                while(BusyXLCD());      // Wait while LCD is busy
                WriteDataXLCD(*buffer); // Write character to LCD
                buffer++;               // Increment buffer
        }
        return;
}

/*******************************************************************
*       Function Name:  ReadAddrXLCD                                *
*       Return Value:   char: address from LCD controller           *
*       Parameters:     void                                        *
*       Description:    This routine reads an address byte from the *
*                       Hitachi HD44780 LCD controller. The user    *
*                       must check to see if the LCD controller is  *
*                       busy before calling this routine. The address*
*                       is read from the character generator RAM or *
*                       the display data RAM depending on what the  *
*                       previous SetxxRamAddr routine was called.   *
*******************************************************************/
unsigned char ReadAddrXLCD(void)
{
        char data;                      // Holds the data retrieved from the LCD

#ifdef BIT8                             // 8-bit interface
        RW_PIN = 1;                     // Set control bits for the read
        RS_PIN = 0;
        DelayFor18TCY();
        E_PIN = 1;                      // Clock data out of the LCD controller
        DelayFor18TCY();
        data = DATA_PORT;               // Save the data in the register
        E_PIN = 0;
        RW_PIN = 0;                     // Reset the control bits
```

```
#else                                   // 4-bit interface
        RW_PIN = 1;                     // Set control bits for the read
        RS_PIN = 0;
        DelayFor18TCY();
        E_PIN = 1;                      // Clock data out of the LCD controller
        DelayFor18TCY();
#ifdef UPPER                            // Upper nibble interface
        data = DATA_PORT&0xf0;          // Read the nibble into the upper nibble of data
#else                                   // Lower nibble interface
        data = (DATA_PORT<<4)&0xf0;     // Read the nibble into the upper nibble of data
#endif
        E_PIN = 0;                      // Reset the clock
        DelayFor18TCY();
        E_PIN = 1;                      // Clock out the lower nibble
        DelayFor18TCY();
#ifdef UPPER                            // Upper nibble interface
        data |= (DATA_PORT>>4)&0x0f;    // Read the nibble into the lower nibble of data
#else                                   // Lower nibble interface
        data |= DATA_PORT&0x0f;         // Read the nibble into the lower nibble of data
#endif
        E_PIN = 0;
        RW_PIN = 0;                     // Reset the control lines
#endif
        return (data&0x7f);             // Return the address, Mask off the busy bit
}

/*******************************************************************
*       Function Name:  ReadDataXLCD                                *
*       Return Value:   char: data byte from LCD controller         *
*       Parameters:     void                                        *
*       Description:    This routine reads a data byte from the     *
*                       Hitachi HD44780 LCD controller. The user    *
*                       must check to see if the LCD controller is  *
*                       busy before calling this routine. The data  *
*                       is read from the character generator RAM or *
*                       the display data RAM depending on what the  *
*                       previous SetxxRamAddr routine was called.   *
*******************************************************************/
char ReadDataXLCD(void)
{
        char data;

#ifdef BIT8                             // 8-bit interface
        RS_PIN = 1;                     // Set the control bits
        RW_PIN = 1;
        DelayFor18TCY();
        E_PIN = 1;                      // Clock the data out of the LCD
        DelayFor18TCY();
        data = DATA_PORT;               // Read the data
        E_PIN = 0;
        RS_PIN = 0;                     // Reset the control bits
        RW_PIN = 0;
#else                                   // 4-bit interface
        RW_PIN = 1;
        RS_PIN = 1;
        DelayFor18TCY();
        E_PIN = 1;                      // Clock the data out of the LCD
        DelayFor18TCY();
#ifdef UPPER                            // Upper nibble interface
        data = DATA_PORT&0xf0;          // Read the upper nibble of data
#else                                   // Lower nibble interface
        data = (DATA_PORT<<4)&0xf0;     // read the upper nibble of data
#endif
        E_PIN = 0;                      // Reset the clock line
        DelayFor18TCY();
        E_PIN = 1;                      // Clock the next nibble out of the LCD
```

```
330         DelayFor18TCY();
    #ifdef UPPER                            // Upper nibble interface
            data |= (DATA_PORT>>4)&0x0f;    // Read the lower nibble of data
    #else                                   // Lower nibble interface
            data |= DATA_PORT&0x0f;         // Read the lower nibble of data
    #endif
            E_PIN = 0;
            RS_PIN = 0;                     // Reset the control bits
            RW_PIN = 0;
    #endif
340         return(data);                   // Return the data byte
    }

    /*******************************************************************
    *       Function Name:  SetCGRamAddr                                *
    *       Return Value:   void                                        *
    *       Parameters:     CGaddr: character generator ram address     *
    *       Description:    This routine sets the character generator   *
    *                       address of the Hitachi HD44780 LCD          *
    *                       controller. The user must check to see if   *
350 *                       the LCD controller is busy before calling   *
    *                       this routine.                               *
    *******************************************************************/
    void SetCGRamAddr(unsigned char CGaddr)
    {
    #ifdef BIT8                                     // 8-bit interface
            TRIS_DATA_PORT = 0;                     // Make data port ouput
            DATA_PORT = CGaddr | 0b01000000;        // Write cmd and address to port
            RW_PIN = 0;                             // Set control signals
            RS_PIN = 0;
360         DelayFor18TCY();
            E_PIN = 1;                              // Clock cmd and address in
            DelayFor18TCY();
            E_PIN = 0;
            DelayFor18TCY();
            TRIS_DATA_PORT = 0xff;                  // Make data port inputs
    #else                                           // 4-bit interface
    #ifdef UPPER                                    // Upper nibble interface
            TRIS_DATA_PORT &= 0x0f;                 // Make nibble input
            DATA_PORT &= 0x0f;                      // and write upper nibble
370         DATA_PORT |= ((CGaddr | 0b01000000) & 0xf0);
    #else                                           // Lower nibble interface
            TRIS_DATA_PORT &= 0xf0;                 // Make nibble input
            DATA_PORT &= 0xf0;                      // and write upper nibble
            DATA_PORT |= (((CGaddr |0b01000000)>>4) & 0x0f);
    #endif
            RW_PIN = 0;                             // Set control signals
            RS_PIN = 0;
            DelayFor18TCY();
            E_PIN = 1;                              // Clock cmd and address in
380         DelayFor18TCY();
            E_PIN = 0;
    #ifdef UPPER                                    // Upper nibble interface
            DATA_PORT &= 0x0f;                      // Write lower nibble
            DATA_PORT |= ((CGaddr<<4)&0xf0);
    #else                                           // Lower nibble interface
            DATA_PORT &= 0xf0;                      // Write lower nibble
            DATA_PORT |= (CGaddr&0x0f);
    #endif
            DelayFor18TCY();
390         E_PIN = 1;                              // Clock cmd and address in
            DelayFor18TCY();
            E_PIN = 0;
    #ifdef UPPER                                    // Upper nibble interface
            TRIS_DATA_PORT |= 0xf0;                 // Make inputs
    #else                                           // Lower nibble interface
```

```
        TRIS_DATA_PORT |= 0x0f;                 // Make inputs
#endif
#endif
        return;
}

/*****************************************************************
*       Function Name:  SetDDRamAddr                            *
*       Return Value:   void                                    *
*       Parameters:     CGaddr: display data address            *
*       Description:    This routine sets the display data address  *
*                       of the Hitachi HD44780 LCD controller. The  *
*                       user must check to see if the LCD controller*
*                       is busy before calling this routine.        *
*****************************************************************/
void SetDDRamAddr(unsigned char DDaddr)
{
#ifdef BIT8                                     // 8-bit interface
        TRIS_DATA_PORT = 0;                     // Make port output
        DATA_PORT = DDaddr | 0b10000000;        // Write cmd and address to port
        RW_PIN = 0;                             // Set the control bits
        RS_PIN = 0;
        DelayFor18TCY();
        E_PIN = 1;                              // Clock the cmd and address in
        DelayFor18TCY();
        E_PIN = 0;
        DelayFor18TCY();
        TRIS_DATA_PORT = 0xff;                  // Make port input
#else                                           // 4-bit interface
#ifdef UPPER                                    // Upper nibble  interface
        TRIS_DATA_PORT &= 0x0f;                 // Make port output
        DATA_PORT &= 0x0f;                      // and write upper nibble
        DATA_PORT |= ((DDaddr | 0b10000000) & 0xf0);
#else                                           // Lower nibble interface
        TRIS_DATA_PORT &= 0xf0;                 // Make port output
        DATA_PORT &= 0xf0;                      // and write upper nibble
        DATA_PORT |= (((DDaddr | 0b10000000)>>4) & 0x0f);
#endif
        RW_PIN = 0;                             // Set control bits
        RS_PIN = 0;
        DelayFor18TCY();
        E_PIN = 1;                              // Clock the cmd and address in
        DelayFor18TCY();
        E_PIN = 0;
#ifdef UPPER                                    // Upper nibble interface
        DATA_PORT &= 0x0f;                      // Write lower nibble
        DATA_PORT |= ((DDaddr<<4)&0xf0);
#else                                           // Lower nibble interface
        DATA_PORT &= 0xf0;                      // Write lower nibble
        DATA_PORT |= (DDaddr&0x0f);
#endif
        DelayFor18TCY();
        E_PIN = 1;                              // Clock the cmd and address in
        DelayFor18TCY();
        E_PIN = 0;
#ifdef UPPER                                    // Upper nibble interface
        TRIS_DATA_PORT |= 0xf0;                 // Make port input
#else                                           // Lower nibble interface
        TRIS_DATA_PORT |= 0x0f;                 // Make port input
#endif
#endif
        return;
}

/*****************************************************************
*       Function Name:  WriteCmdXLCD                            *
```

```
*       Return Value:   void                                    *
*       Parameters:     cmd: command to send to LCD             *
*       Description:    This routine writes a command to the Hitachi*
*                       HD44780 LCD controller. The user must check *
*                       to see if the LCD controller is busy before *
*                       calling this routine.                   *
*******************************************************************/
void WriteCmdXLCD(unsigned char cmd)
{
#ifdef BIT8                             // 8-bit interface
        TRIS_DATA_PORT = 0;             // Data port output
        DATA_PORT = cmd;                // Write command to data port
        RW_PIN = 0;                     // Set the control signals
        RS_PIN = 0;                     // for sending a command
        DelayFor18TCY();
        E_PIN = 1;                      // Clock the command in
        DelayFor18TCY();
        E_PIN = 0;
        DelayFor18TCY();
        TRIS_DATA_PORT = 0xff;          // Data port input
#else                                   // 4-bit interface
#ifdef UPPER                            // Upper nibble interface
        TRIS_DATA_PORT &= 0x0f;
        DATA_PORT &= 0x0f;
        DATA_PORT |= cmd&0xf0;
#else                                   // Lower nibble interface
        TRIS_DATA_PORT &= 0xf0;
        DATA_PORT &= 0xf0;
        DATA_PORT |= (cmd>>4)&0x0f;
#endif
        RW_PIN = 0;                     // Set control signals for command
        RS_PIN = 0;
        DelayFor18TCY();
        E_PIN = 1;                      // Clock command in
        DelayFor18TCY();
        E_PIN = 0;
#ifdef UPPER                            // Upper nibble interface
        DATA_PORT &= 0x0f;
        DATA_PORT |= (cmd<<4)&0xf0;
#else                                   // Lower nibble interface
        DATA_PORT &= 0xf0;
        DATA_PORT |= cmd&0x0f;
#endif
        DelayFor18TCY();
        E_PIN = 1;                      // Clock command in
        DelayFor18TCY();
        E_PIN = 0;
#ifdef UPPER                            // Make data nibble input
        TRIS_DATA_PORT |= 0xf0;
#else
        TRIS_DATA_PORT |= 0x0f;
#endif
#endif
        return;
}

/*******************************************************************
*       Function Name:  WriteDataXLCD                           *
*       Return Value:   void                                    *
*       Parameters:     data: data byte to be written to LCD    *
*       Description:    This routine writes a data byte to the  *
*                       Hitachi HD44780 LCD controller. The user *
*                       must check to see if the LCD controller is *
*                       busy before calling this routine. The data *
*                       is written to the character generator RAM or*
*                       the display data RAM depending on what the *
```

```
*                       previous SetxxRamAddr routine was called.   *
*********************************************************************/
void WriteDataXLCD(char data)
{
#ifdef BIT8                             // 8-bit interface
        TRIS_DATA_PORT = 0;             // Make port output
        DATA_PORT = data;               // Write data to port
        RS_PIN = 1;                     // Set control bits
        RW_PIN = 0;
        DelayFor18TCY();
        E_PIN = 1;                      // Clock data into LCD
        DelayFor18TCY();
        E_PIN = 0;
        RS_PIN = 0;                     // Reset control bits
        TRIS_DATA_PORT = 0xff;          // Make port input
#else                                   // 4-bit interface
#ifdef UPPER                            // Upper nibble interface
        TRIS_DATA_PORT &= 0x0f;
        DATA_PORT &= 0x0f;
        DATA_PORT |= data&0xf0;
#else                                   // Lower nibble interface
        TRIS_DATA_PORT &= 0xf0;
        DATA_PORT &= 0xf0;
        DATA_PORT |= ((data>>4)&0x0f);
#endif
        RS_PIN = 1;                     // Set control bits
        RW_PIN = 0;
        DelayFor18TCY();
        E_PIN = 1;                      // Clock nibble into LCD
        DelayFor18TCY();
        E_PIN = 0;
#ifdef UPPER                            // Upper nibble interface
        DATA_PORT &= 0x0f;
        DATA_PORT |= ((data<<4)&0xf0);
#else                                   // Lower nibble interface
        DATA_PORT &= 0xf0;
        DATA_PORT |= (data&0x0f);
#endif
        DelayFor18TCY();
        E_PIN = 1;                      // Clock nibble into LCD
        DelayFor18TCY();
        E_PIN = 0;
#ifdef UPPER                            // Upper nibble interface
        TRIS_DATA_PORT |= 0xf0;
#else                                   // Lower nibble interface
        TRIS_DATA_PORT |= 0x0f;
#endif
#endif
        return;
}
```

Apéndice B

Esquema del analizador

Este apéndice contiene:

- Esquema electrónico analizador *CAN* (4 hojas).
- *Layout* de la placa principal, cara superior de componentes.
- *Layout* de la placa principal, cara inferior de componentes.
- *Layout* de la placa principal, cara inferior de pistas.
- *Layout* de la botonera, cara superior de componentes.
- *Layout* de la botonera, cara de pistas.
- Imagen de la placa principal, cara inferior de componentes.
- Imagen de la placa principal, cara superior de componentes.

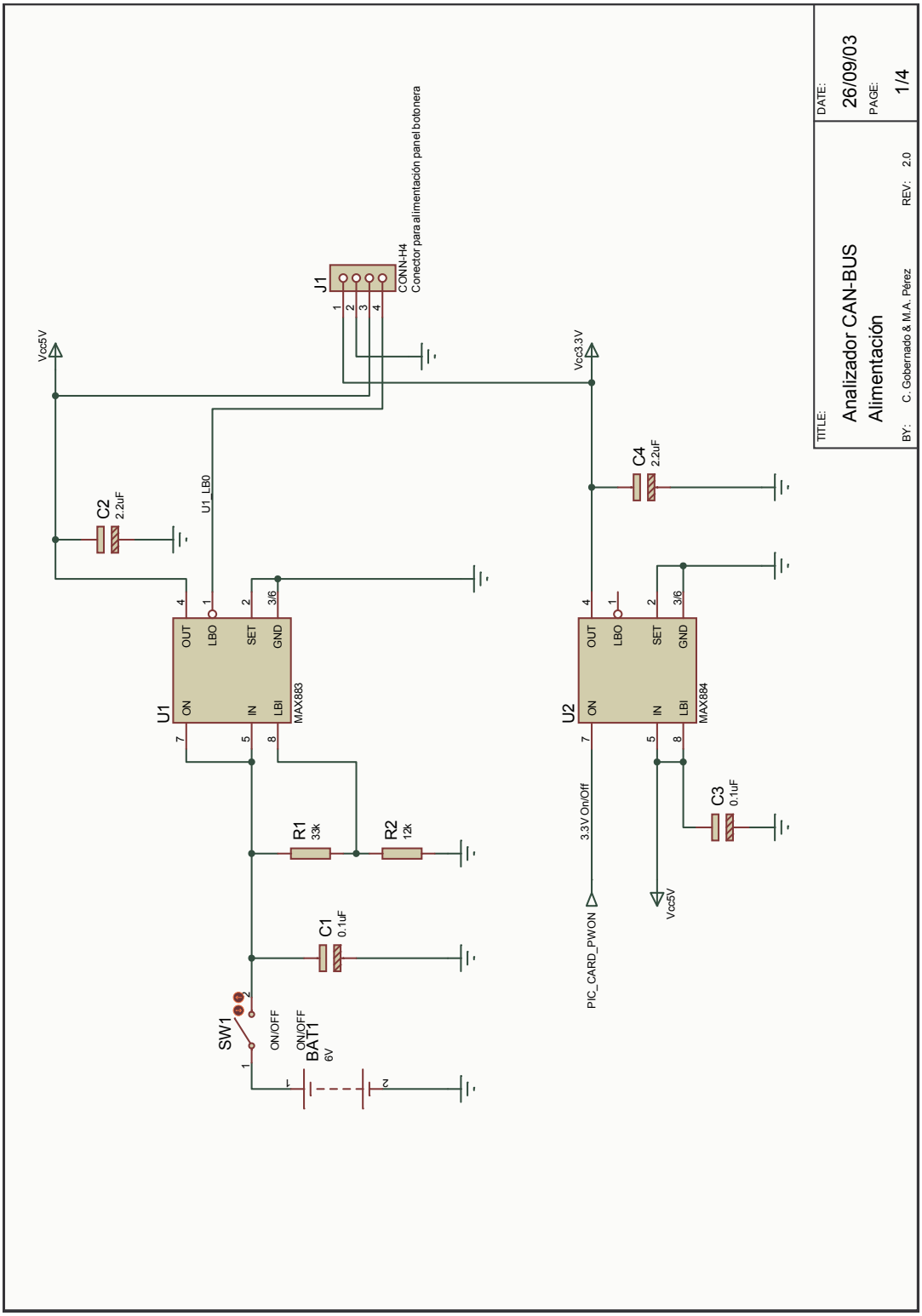


Figura B.1: Esquema analizador *CAN* (hoja1).

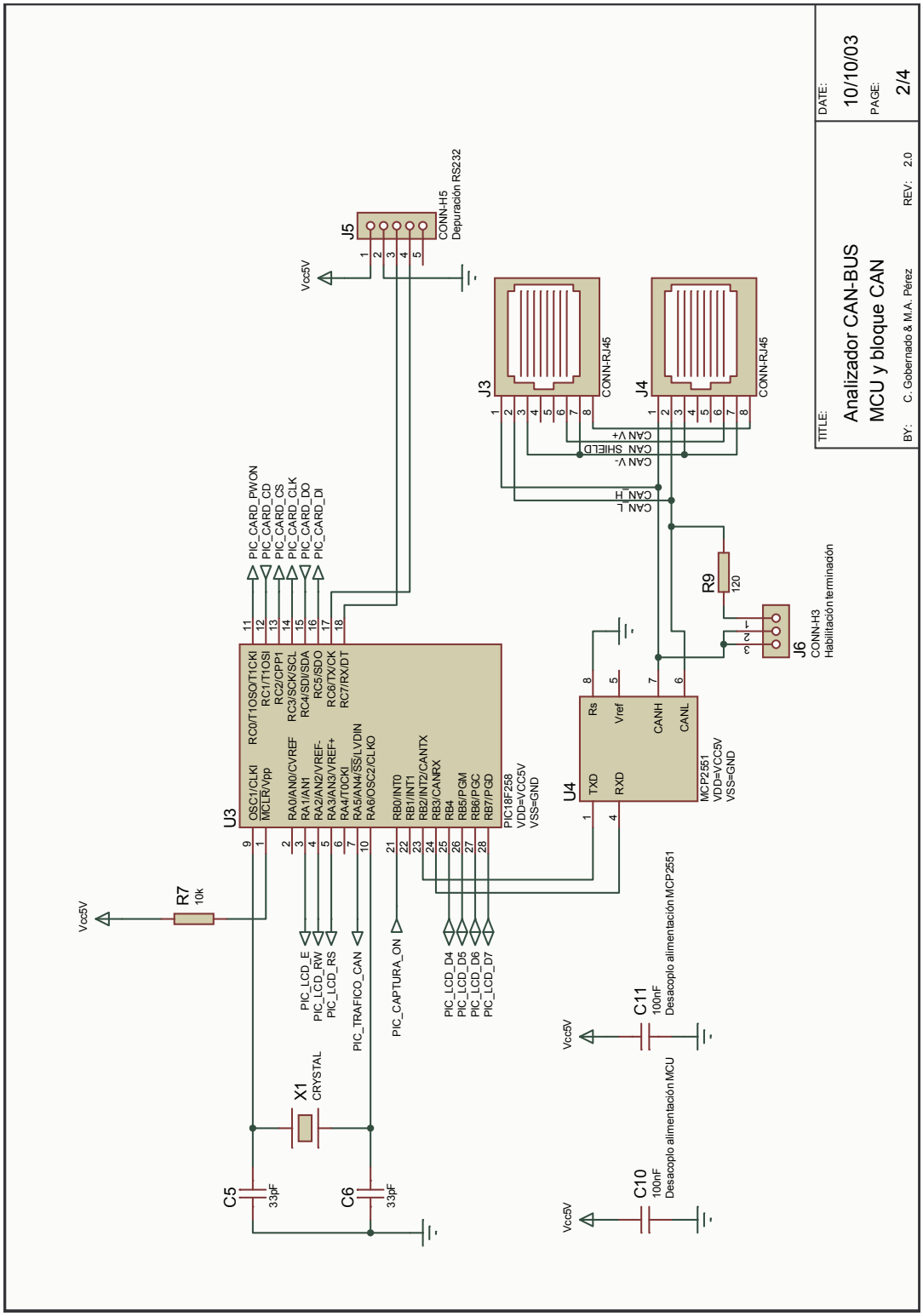


Figura B.2: Esquema analizador *CAN* (hoja2).

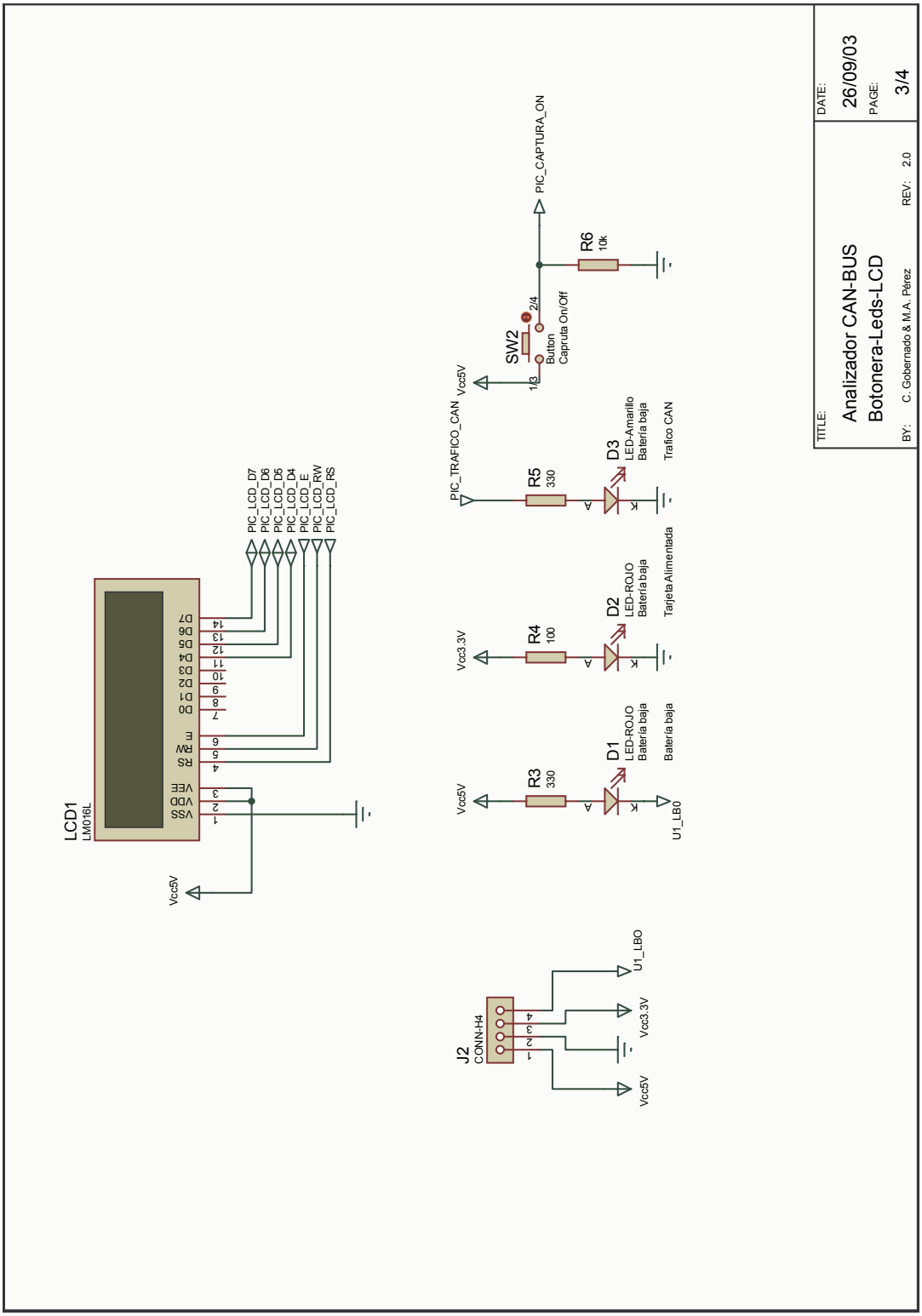


Figura B.3: Esquema analizador *CAN* (hoja3).

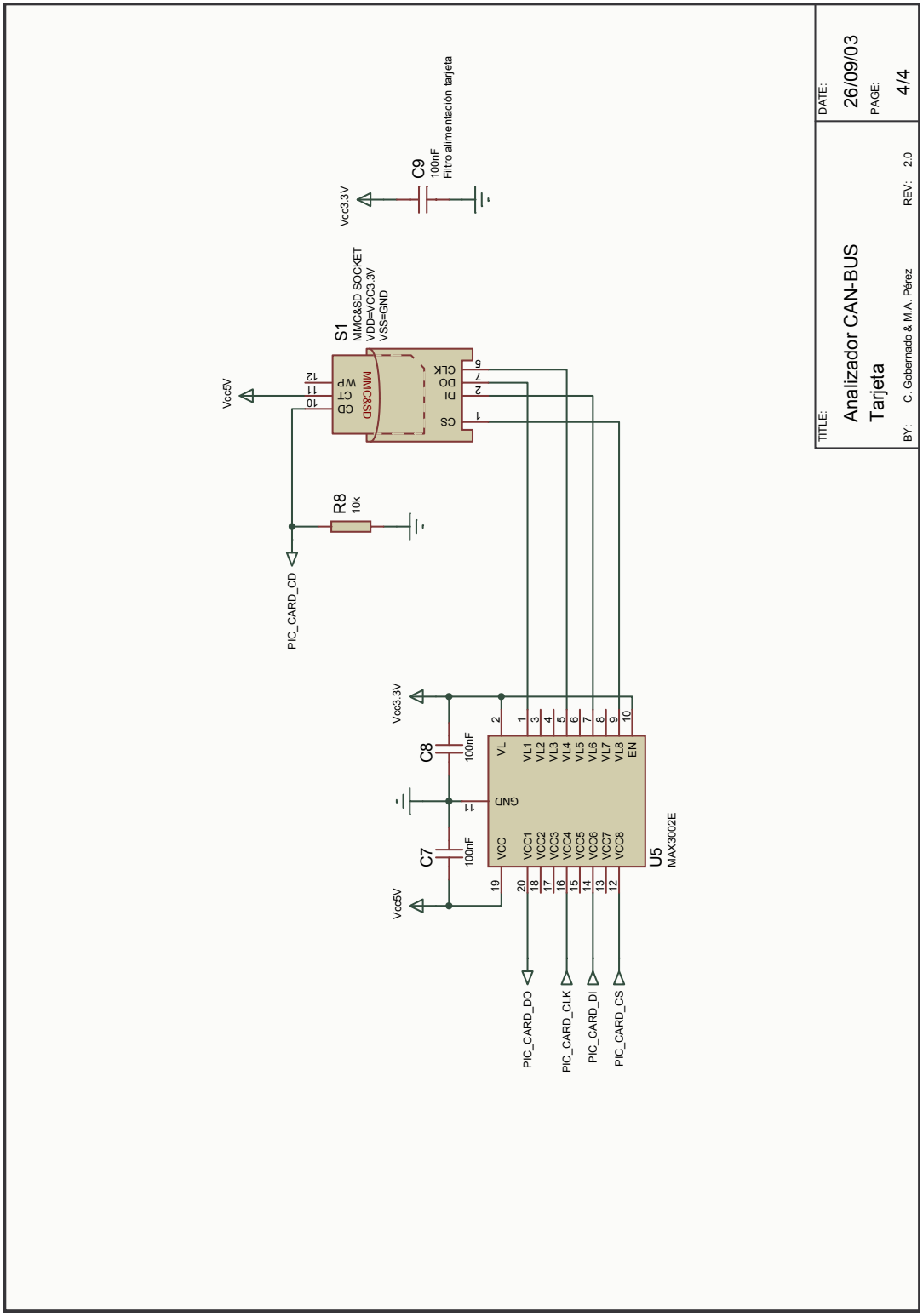


Figura B.4: Esquema analizador *CAN* (hoja4).

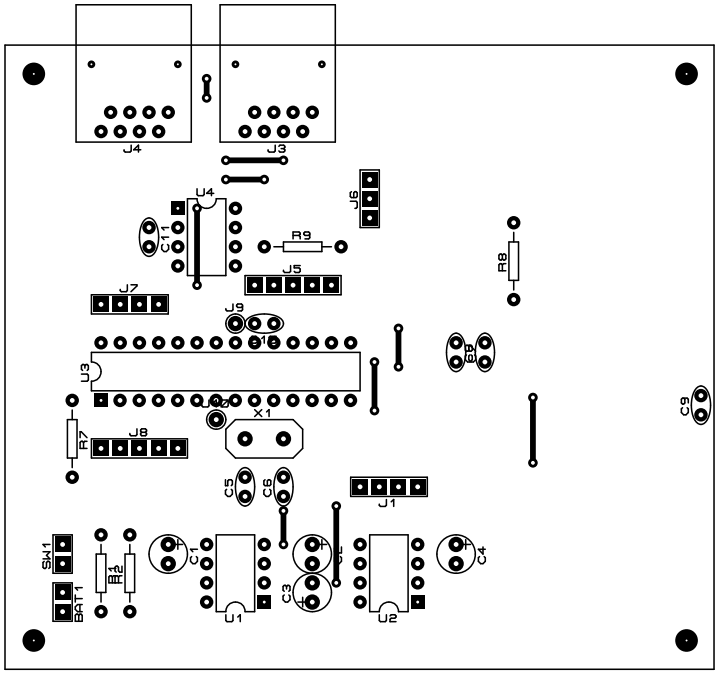


Figura B.5: Cara superior de componentes de la placa principal del analizador *CAN*.

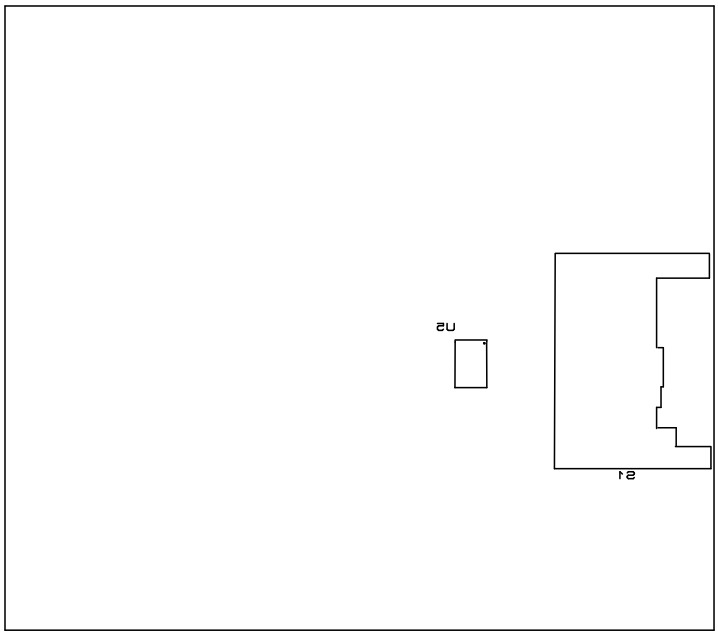


Figura B.6: Cara inferior de componentes de la placa principal del analizador *CAN*.

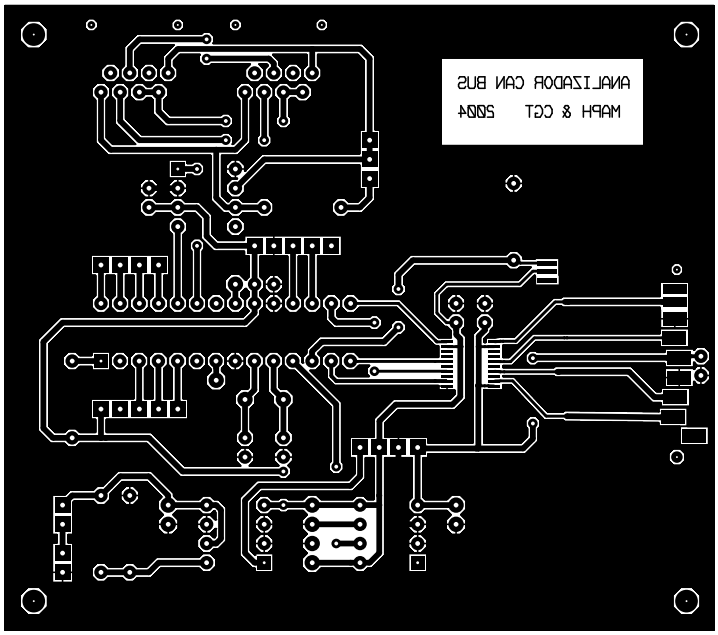


Figura B.7: Cara inferior de pistas de la placa principal del analizador *CAN*.

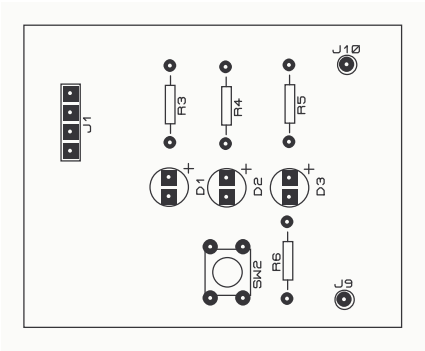


Figura B.8: Cara superior de componentes de la botonera del analizador *CAN*.

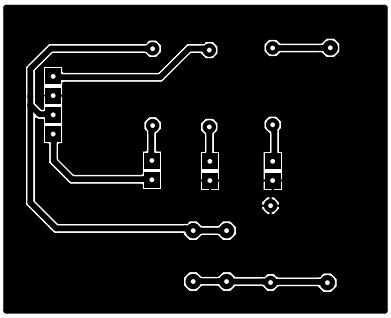


Figura B.9: Cara de cobre de la botonera del analizador *CAN*.

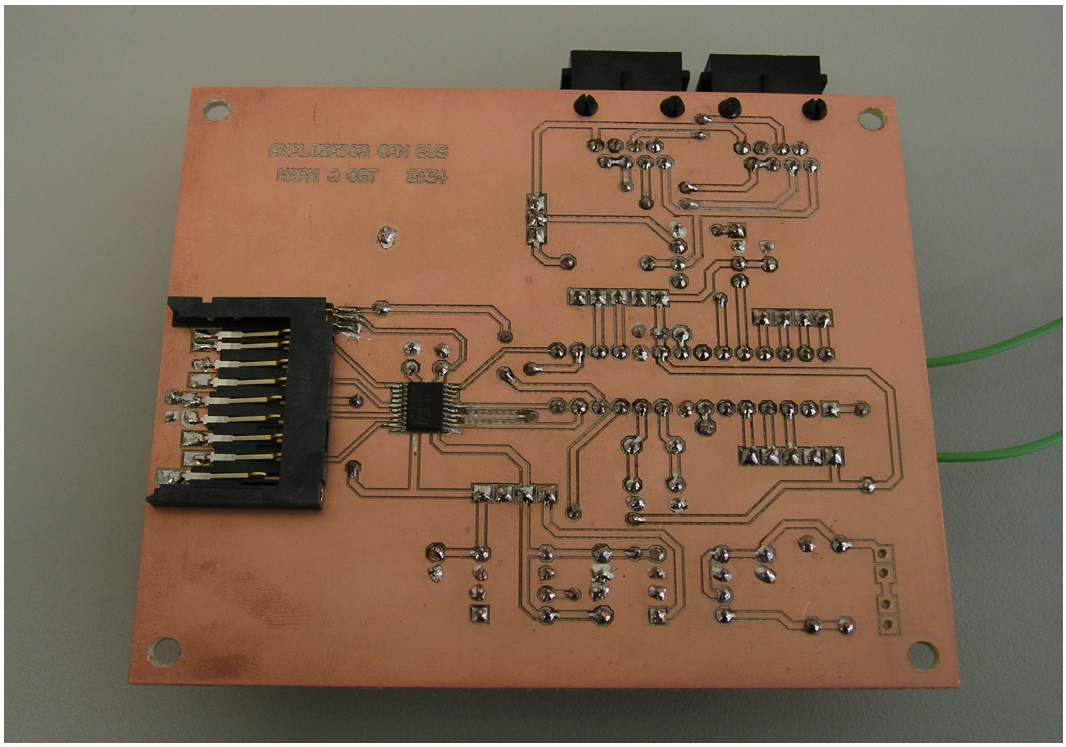


Figura B.10: Placa principal del analizador *CAN*, cara inferior de componentes.

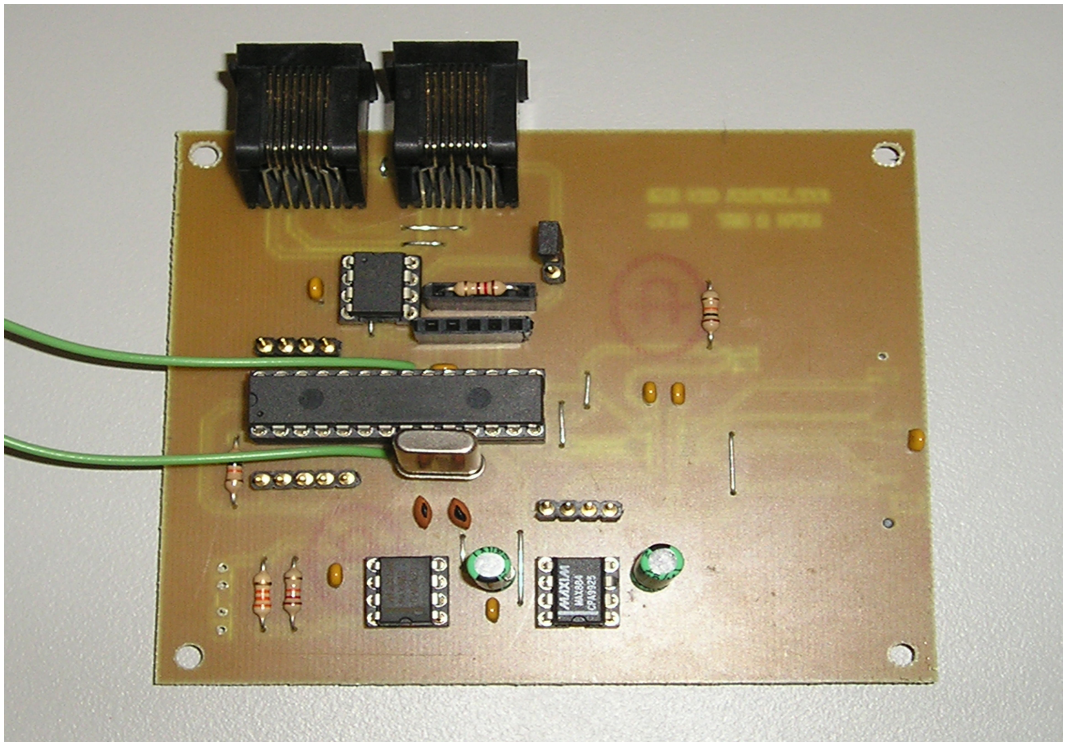


Figura B.11: Placa principal del analizador *CAN*, cara superior de componentes.

Apéndice C

Código fuente del nodo universal

C.1 Fichero makefile

Listado C.1: Fichero makefile.mak.

```
proyecto=D:\PFC-CANBUS-NOBORRAR\pfc\Nodo
mcc18=D:\Archivos de programa\mcc18
cc=$(mcc18)\bin\c18demo.exe
linker=$(mcc18)\bin\mplink.exe
lib=$(mcc18)\lib
lkr=$(mcc18)\lkr
include=$(mcc18)\h
micro=18f258

all : \
  clean \
  link

link : nodoUniv.o can18xx8.o
  $(linker) /l$(lib) /k$(lkr) $(proyecto)\18f258i.lkr
  $(proyecto)\nodoUniv.o $(proyecto)\can18xx8.o
  /o nodoUniv.cof

nodoUniv.o :
  $(cc) -p=$(micro) $(proyecto)\nodoUniv.c
  -fo=nodoUniv.o /i
  $(include) -Ou+ -Ot- -Ob- -Op- -Or- -Od- -Opa+

can18xx8.o :
  $(cc) -p=$(micro) $(proyecto)\can18xx8.c
  -fo=can18xx8.o /i
  $(proyecto) -Ou+ -Ot- -Ob- -Op- -Or- -Od- -Opa+

clean :
  del *.o
  del *.hex
  del *.cof
  del *.cod
  del *.lst
  del *.err
```

C.2 Código fuente del nodo universal

Listado C.2: Fichero nodoUniv.c.

```
//-----------------------------------------------------------------
//
//                          DESCRIPCION
//
// univ1.c es el programa del nodo UNIVERSAL1 de la aplicacion propuesta
// Este NODO debe comunicarse con el CONTROLADOR para fijar
// su ID y los modulos conectados a el
//
//-----------------------------------------------------------------
//
//    Archivo:              univ1.c
//
//    Autores:        CARLOS GOBERNADO TEJEDOR
//              MIGUEL ANGEL PEREZ HERRERO
//
//              VERSION:    2.01                  10/10/2004
//-----------------------------------------------------------------
//
//                    PARAMETROS DE CONFIGURACION
// Este codigo se ha compilado usando MPLAB-C18 ver. 2.20
//
// Frecuencia reloj: xxx
//
// Los siguientes archivos debe de ser incluidos en el proyecto MPLAB
//
//        univ1.c           -- Main source code file
//        pic18f458.lkr     -- Linker script file
//
//        Los siguientes archivos deben de incluirse en el linker script
//
//        c018i.o           -- C startup code
//        clib.lib          -- Math and function libraries
//        p18f458.lib       -- Processor library
//
//-----------------------------------------------------------------
#include <p18cxxx.h>
#include <stdlib.h>
#include "CAN18XX8.h"
#include <delays.h>
#include <adc.h>
#include <pwm.h>
#include <timers.h>

//-----------------------------------------------------------------
// Bits de configuracion del micro
//
// Proteccion de codigo: off
// Switch del oscilador (_OSCS_OFF_1H): OFF Tipo: HS
// Power-up timer: off
// Brown-out reset: off (_BORV_42_2L fija el valor de Brown-out=4,2)
// Watch-dog:off
// Lov Level programming: off
// Debug: off
// Reset si hay overflow in la pila (STVR_ON_4L): on
//-----------------------------------------------------------------

#pragma romdata CONFIG
_CONFIG_DECL (  _OSCS_OFF_1H & _OSC_HS_1H,
                _PWRT_OFF_2L & _BOR_OFF_2L,
                _WDT_OFF_2H,
                _STVR_ON_4L & _LVP_OFF_4L & _DEBUG_OFF_4L,
                _CONFIG5L_DEFAULT,
```

```
                _CONFIG5H_DEFAULT,
                _CONFIG6L_DEFAULT,
                _CONFIG6H_DEFAULT,
                _CONFIG7L_DEFAULT,
                _CONFIG7H_DEFAULT);
#pragma romdata

70 //-------------------------------------------------------------------
   // Definicion de constantes
   //-------------------------------------------------------------------

   // Macros para la configuracion de los tiempos en CAN
   // Tq=2(BRP+1)/Fosc(Mhz)

   //MAC
   #define MAC                 837  // MAC ejemplo

80 // Baund rate base 1Mbit/s
   #define BRP                      0   // Tq=2*(0+1)/20Mhz Tbit=10*Tq=1us
   #define SJW                      1   // Salto de sincronizacion de Tq
   #define PHSEG1                   4   // PHSEG1
   #define PHSEG2                   4   // PHSEG2
   #define PROPSEG                  1   // TPropagacion

   #define ID_CONTROLADOR           0
   #define ID_CONEXION              1
   #define TIPO_TRAMA               CAN_CONFIG_STD_MSG
90 #define MASK1                    0b11111111111
   #define MASK2                    0b11111111111
   #define TRIS_LED_OFF             TRISBbits.TRISB1
   #define LED_OFF                  PORTBbits.RB1
   #define BOTON_OFF                PORTBbits.RB0
   #define BOTON_ON                 PORTBbits.RB0
   #define MASCARA1_PUERTOB         0b11110000
   #define MASCARA2_PUERTOB         0b00001111

   // Valores asignados del campo de control
100 #define PETICION_ID             0
   #define ACEPTACION_DATO          1
   #define PETICION_DESCONEXION     2

   #define ENVIO_DATOS              0
   #define RECEP_DATOS              1
   #define DESCONEXION              2

   // Valores asignados a los distintos pines del nodo universal
   #define AN_IN0                   0
110 #define AN_IN1                  1
   #define AN_IN2                   2
   #define AN_IN3                   3
   #define AN_IN4                   4
   #define AN_IN5                   5
   #define AN_IN6                   6
   #define AN_IN7                   7
   #define PIN_RB4                  8
   #define PIN_RB5                  9
   #define PIN_RB6                  10
120 #define PIN_RB7                 11
   #define PIN_RC0                  12
   #define PIN_RC1                  13
   #define PIN_RC2                  14
   #define PIN_RC3                  15
   #define PIN_RC4                  16
   #define PIN_RC5                  17
   #define PIN_RC6                  18
   #define PIN_RC7                  19
```

```
#define PIN_RD0                    20
#define PIN_RD1                    21
#define PIN_RD2                    22
#define PIN_RD3                    23
#define PIN_RD4                    24
#define PIN_RD5                    25
#define PIN_RD6                    26
#define PIN_RD7                    27
#define ALL_PORTB_ALTO             28
#define ALL_PORTC                  29
#define ALL_PORTD                  30
#define PWM_IN                     31

#define NOCONFIG_PWM               0
#define PREESCALA1                 1
#define PREESCALA4                 4
#define PREESCALA16                16
//-------------------------------------------------------------------
// Prototipos de funciones

// Configura el micro e inicializa variables
void Setup(void);
// Configura el modulo CAN
void Setup_CAN(void);
// Lee y envia al CONTROLADOR un dato analogico convertido
void AD_Analog(void);
// Lee y envia al CONTROLADOR un dato digital
void Enviar_Dato_DIG(void);
// Configura modulo PWM
void Config_PWM(void);
// Interrupcion de recepcion del CAN
void IntCanRx();

//-------------------------------------------------------------------
// Declaracion de variables globales
//-------------------------------------------------------------------

typedef unsigned char BYTE;
typedef unsigned long ulong;
typedef unsigned int uint;
typedef unsigned char uchar;
typedef unsigned short long uslong;

// Variables relacionadas con el modulo CAN
ulong NewMessage;                          //ID de la trama
BYTE NewMessageData[8];
BYTE NewMessageLen;
BYTE RxFilterMatch;
enum CAN_RX_MSG_FLAGS NewMessageFlags;
ulong ID_nodo=1;                           // Almacenamos el ID asignado por el CONTROLADOR

BYTE controlador_CAN[8];                   // Trama de peticion de asignacion de ID
uchar i=0;                                 // Selecciona el canal correspondiente
uint temp1;                                // Dato temporal para calculos
uchar temp2;                               // Dato temporal para calculos
BYTE envio_dato[8];                        // Configuracion de los datos que se envian
                                           // ante una orden solicitada por el CONTROLADOR

//-------------------------------------------------------------------
// Codigo de interrupciones
//-------------------------------------------------------------------

//-------------------------------------------------------------------
// IntCanRx()
// Int que atiende los datos recibidos.
//-------------------------------------------------------------------
```

```
#pragma code high_vector=0x08
void high_interrupt(void)
{
_asm
        goto IntCanRx
_endasm
}
#pragma code
#pragma interrupt IntCanRx save=PROD,CANCON

void IntCanRx(){
  INTCON=0;          // Deshabilitamos las interrupciones
  PIE3=0b00000000;
  if(PIR3bits.RXB0IF==1 || PIR3bits.RXB1IF==1){
    if(CANIsRxReady()){
      CANReceiveMessage(&NewMessage,
                        NewMessageData,
                        &NewMessageLen,
                        &NewMessageFlags);
      if(NewMessage==ID_CONEXION){                          // Asignacion de ID
        if(MAC==*((uslong*)(NewMessageData+1))){
          ID_nodo=NewMessageData[0];
          CANSetOperationModeNoWait(CAN_OP_MODE_CONFIG);     // entramos en modo configuracion
          CANSetFilter(CAN_FILTER_B1_F2,ID_nodo,TIPO_TRAMA); // filtro con ID asignada
          CANSetFilter(CAN_FILTER_B2_F2,ID_nodo,TIPO_TRAMA); //  filtro con ID asignada
              CANSetOperationMode(CAN_OP_MODE_NORMAL);       // CAN en modo normal
        }
      }
      else if(NewMessage==ID_nodo){
        if(NewMessageData[0]==ENVIO_DATOS || NewMessageData[0]==RECEP_DATOS){
          switch (NewMessageData[1]){     // Envio-recepcion de Datos
            case AN_IN0:          // IN_analog0-AN0
              SetChanADC(ADC_CH0);
              AD_Analog();        // Llamada a la funcion de conversion
              break;
            case AN_IN1:          // IN_analog1-AN1
              SetChanADC(ADC_CH1);
              AD_Analog();        // Llamada a la funcion de conversion
              break;
            case AN_IN2:          // IN_analog2-AN2
              SetChanADC(ADC_CH2);
              AD_Analog();        // Llamada a la funcion de conversion
              break;
            case AN_IN3:          // IN_analog3-AN3
              SetChanADC(ADC_CH3);
              AD_Analog();        // Llamada a la funcion de conversion
              break;
            case AN_IN4:          // IN_analog4-AN4
              SetChanADC(ADC_CH4);
              AD_Analog();        // Llamada a la funcion de conversion
              break;
            case AN_IN5:          // IN_analog4-AN4
              SetChanADC(ADC_CH5);
              AD_Analog();        // Llamada a la funcion de conversion
              break;
            case AN_IN6:          // IN_analog4-AN4
              SetChanADC(ADC_CH6);
              AD_Analog();        // Llamada a la funcion de conversion
              break;
            case AN_IN7:          // IN_analog4-AN4
              SetChanADC(ADC_CH7);
              AD_Analog();        // Llamada a la funcion de conversion
              break;
            case PIN_RB4:               // IN_OUT digital RB4
              if(NewMessageData[0]==ENVIO_DATOS){   // Se saca el dato por este pin
```

```
            TRISBbits.TRISB4=0;
            PORTBbits.RB4=NewMessageData[2];
          }
          else{                  // CONTROLADOR quiere recibir dato de este pin
            TRISBbits.TRISB4=1;
            envio_dato[2]=PORTBbits.RB4;
            Enviar_Dato_DIG();          // Llamada a la funcion de lectura de entrada dig.
          }
          break;
270     case PIN_RB5:                // IN_OUT digital RB5
          if(NewMessageData[0]==ENVIO_DATOS){   // Se saca el dato por este pin
            TRISBbits.TRISB5=0;
            PORTBbits.RB5=NewMessageData[2];
          }
          else{                  // CONTROLADOR quiere recibir dato de este pin
            TRISBbits.TRISB5=1;
            envio_dato[2]=PORTBbits.RB5;
            Enviar_Dato_DIG();          // Llamada a la ófuncin de lectura de entrada dig.
          }
280       break;
        case PIN_RB6:                // IN_OUT digital RB6
          if(NewMessageData[0]==ENVIO_DATOS){   // Se saca el dato por este pin
            TRISBbits.TRISB6=0;
            PORTBbits.RB6=NewMessageData[2];
          }
          else{                  // CONTROLADOR quiere recibir dato de este pin
            TRISBbits.TRISB6=1;
            envio_dato[2]=PORTBbits.RB6;
            Enviar_Dato_DIG();          // Llamada a la funcion de lectura de entrada dig.
290       }
          break;
        case PIN_RB7:                // IN_OUT digital RB7
          if(NewMessageData[0]==ENVIO_DATOS){   // Se saca el dato por este pin
            TRISBbits.TRISB7=0;
            PORTBbits.RB7=NewMessageData[2];
          }
          else{                  // CONTROLADOR quiere recibir dato de este pin
            TRISBbits.TRISB7=1;
            envio_dato[2]=PORTBbits.RB7;
300         Enviar_Dato_DIG();          // Llamada a la funcion de lectura de entrada dig.
          }
          break;
        case ALL_PORTB_ALTO:            // IN_OUT digital PORTB_ALTO
          if(NewMessageData[0]==ENVIO_DATOS){   // Se saca el dato por este pin
            TRISB=0b00001001;
            NewMessageData[2]&=MASCARA1_PUERTOB;
            PORTB&=MASCARA2_PUERTOB;
            PORTB|=NewMessageData[2];
          }
310       else{                  // CONTROLADOR quiere recibir dato de este pin
            TRISB=0b11111001;
            envio_dato[2]=PORTB;
            envio_dato[2]&=MASCARA1_PUERTOB;
            Enviar_Dato_DIG();          // Llamada a la funcion de lectura de entrada dig.
          }
          break;
        case PIN_RC0:                // IN_OUT digital RC0
          if(NewMessageData[0]==ENVIO_DATOS){   // Se saca el dato por este pin
            TRISCbits.TRISC0=0;
320         PORTCbits.RC0=NewMessageData[2];
          }
          else{                  // CONTROLADOR quiere recibir dato de este pin
            TRISCbits.TRISC0=1;
            envio_dato[2]=PORTCbits.RC0;
            Enviar_Dato_DIG();          // Llamada a la funcion de lectura de entrada dig.
          }
```

```
            break;
          case PIN_RC1:                // IN_OUT digital RC1
            if(NewMessageData[0]==ENVIO_DATOS){   // Se saca el dato por este pin
330           TRISCbits.TRISC1=0;
              PORTCbits.RC1=NewMessageData[2];
            }
            else{                  // CONTROLADOR quiere recibir dato de este pin
              TRISCbits.TRISC1=1;
              envio_dato[2]=PORTCbits.RC1;
              Enviar_Dato_DIG();          // Llamada a la funcion de lectura de entrada dig.
            }
            break;
          case PIN_RC2:                // IN_OUT digital0 RC2
340         if(NewMessageData[0]==ENVIO_DATOS){   // Se saca el dato por este pin
              TRISCbits.TRISC2=0;
              PORTCbits.RC2=NewMessageData[2];
            }
            else{                  // CONTROLADOR quiere recibir dato de este pin
              TRISCbits.TRISC2=1;
              envio_dato[2]=PORTCbits.RC2;
              Enviar_Dato_DIG();          // Llamada a la funcion de lectura de entrada dig.
            }
            break;
350         case PIN_RC3:                // IN_OUT digital RC3
            if(NewMessageData[0]==ENVIO_DATOS){   // Se saca el dato por este pin
              TRISCbits.TRISC3=0;
              PORTCbits.RC3=NewMessageData[2];
            }
            else{                  // CONTROLADOR quiere recibir dato de este pin
              TRISCbits.TRISC3=1;
              envio_dato[2]=PORTCbits.RC3;
              Enviar_Dato_DIG();          // Llamada a la funcion de lectura de entrada dig.
            }
360         break;
          case PIN_RC4:                // IN_OUT digital RC4
            if(NewMessageData[0]==ENVIO_DATOS){   // Se saca el dato por este pin
              TRISCbits.TRISC4=0;
              PORTCbits.RC4=NewMessageData[2];
            }
            else{                  // CONTROLADOR quiere recibir dato de este pin
              TRISCbits.TRISC4=1;
              envio_dato[2]=PORTCbits.RC4;
              Enviar_Dato_DIG();          // Llamada a la funcion de lectura de entrada dig.
370         }
            break;
          case PIN_RC5:                // IN_OUT digital RC5
            if(NewMessageData[0]==ENVIO_DATOS){   // Se saca el dato por este pin
              TRISCbits.TRISC5=0;
              PORTCbits.RC5=NewMessageData[2];
            }
            else{                  // CONTROLADOR quiere recibir dato de este pin
              TRISCbits.TRISC5=1;
              envio_dato[2]=PORTCbits.RC5;
380           Enviar_Dato_DIG();          // Llamada a la funcion de lectura de entrada dig.
            }
            break;
          case PIN_RC6:                // IN_OUT digital RC6
            if(NewMessageData[0]==ENVIO_DATOS){   // Se saca el dato por este pin
              TRISCbits.TRISC6=0;
              PORTCbits.RC6=NewMessageData[2];
            }
            else{                  // CONTROLADOR quiere recibir dato de este pin
              TRISCbits.TRISC6=1;
390           envio_dato[2]=PORTCbits.RC6;
              Enviar_Dato_DIG();          // Llamada a la funcion de lectura de entrada dig.
            }
```

```
                break;
              case PIN_RC7:                // IN_OUT digital RC7
                if(NewMessageData[0]==ENVIO_DATOS){   // Se saca el dato por este pin
                  TRISCbits.TRISC7=0;
                  PORTCbits.RC7=NewMessageData[2];
                }
                else{                  // CONTROLADOR quiere recibir dato de este pin
                  TRISCbits.TRISC7=1;
                  envio_dato[2]=PORTCbits.RC7;
                  Enviar_Dato_DIG();          // Llamada a la funcion de lectura de entrada dig.
                }
                break;
              case ALL_PORTC:                // IN_OUT digital PORTD
                if(NewMessageData[0]==ENVIO_DATOS){   // Se saca el dato por este pin
                  TRISC=0b00000000;
                  PORTC=NewMessageData[2];
                }
                else{                  // CONTROLADOR quiere recibir dato de este pin
                  TRISC=0b11111111;
                  envio_dato[2]=PORTC;
                  Enviar_Dato_DIG();          // Llamada a la funcion de lectura de entrada dig.
                }
                break;
              case PWM_IN:                        // SALIDA PWM Pin RC2
                if(NewMessageData[2]!=0)          // Comprobamos si hay que configurar el modulo
                  Config_PWM();                   // Llamada a la funcion de configuracion de PWM
                SetDCPWM1(*((uint*)(NewMessageData+3)));// Modificamos la salida PWM
                break;
            }
          }
          else if(NewMessageData[0]==DESCONEXION){      // modulo OFF
            LED_OFF=1;                                  // LED de aviso de desconexion
            for(i=0;i<10;i++){
              Delay10KTCYx(250);
            }           // Cerramos todos los modulos
                      LED_OFF=0;              // Inicializamos los puertos
                      ID_nodo=1;
                      PORTB=0;
                      TRISB=0b00001001;
                      ClosePWM1();
                      PORTC=0;
                      TRISC=0;
                      CloseADC();
                      PORTA=0;
                      TRISA=0;
          }
        }
      }
    }
    PIR3=0b00000000;    // Limpiamos flags
    PIE3=0b00000011;    // Habilitamos las interrupciones
    INTCON=0b11000000;
  }

  // Configuramos el modulo PWM
  void Config_PWM( void )
  {                                           // Configuramos temporizador 2
    if(NewMessageData[2]==PREESCALA1){        // Preescala en el temporizador2 1:1
      OpenTimer2( TIMER_INT_OFF &
              T2_PS_1_1 &
              T2_POST_1_1 );
    }
    else if(NewMessageData[2]==PREESCALA4){   // Preescala en el temporizador2 1:4
      OpenTimer2( TIMER_INT_OFF &
              T2_PS_1_4 &
              T2_POST_1_1 );
```

```
    }
460 else if(NewMessageData[2]==PREESCALA16){  // Preescala en el temporizador2 1:16
      OpenTimer2( TIMER_INT_OFF &
              T2_PS_1_16 &
              T2_POST_1_1 );
    }
    OpenPWM1(NewMessageData[5]);              // Configuramos PR2
  }

  // Mandamos un dato digital
  void Enviar_Dato_DIG( void )
470 {
    envio_dato[0]=ACEPTACION_DATO;            // Preparamos la trama con el dato digital
    envio_dato[1]=NewMessageData[1];          // Guardamos canal seleccionado
    envio_dato[3]=0;                          // Este campo se deja como 0
    envio_dato[4]=ID_nodo;                    // ID del nodo que envia dato
    *((uslong*)(envio_dato+5))=MAC;           // MAC del nodo
    if(CANIsTxReady()){                       // Mandamos la trama al CONTROLADOR
      CANSendMessage(ID_CONTROLADOR,
               envio_dato,
               8,
480            CAN_TX_PRIORITY_0 &
               CAN_TX_STD_FRAME &
               CAN_TX_NO_RTR_FRAME);
    }
    return;
  }

  // Mandamos un dato analogico
  void AD_Analog( void )
  {
490 envio_dato[0]=ACEPTACION_DATO;          // Preparamos la trama con el dato analogico
    envio_dato[1]=NewMessageData[1];        // Guardamos canal seleccionado
    envio_dato[4]=ID_nodo;                  // ID del nodo que envia dato
    ConvertADC();                           // Empieza la conversion
    while(BusyADC());                       // Espera hasta que se complete
    temp1=ReadADC();                        // En temp1 esta el dato ya convertido
    *((uint*)(envio_dato+2))=temp1;         // Dato[2-3]=temp1: dato solicitado
    *((uslong*)(envio_dato+5))=MAC;         // MAC del nodo
    if(CANIsTxReady()){                     // Mandamos la trama al CONTROLADOR
      CANSendMessage(ID_CONTROLADOR,
500            envio_dato,
               8,
               CAN_TX_PRIORITY_0 &
               CAN_TX_STD_FRAME &
               CAN_TX_NO_RTR_FRAME);
    }
    return;
  }

  //-------------------------------------------------------------------
510 // Implementacion de funciones
  //-------------------------------------------------------------------

  //-------------------------------------------------------------------
  // Setup()
  // La mision de esta funcion es la de configurar el micro e inizializar
  // las variables declaradas
  //-------------------------------------------------------------------

  void Setup(void)
520 {

  // PUERTOS
          TRISA=0b11111111;          // RA0-RA3 RA5: Entradas analogicas (Conversiones A/D)
          TRISB=0b00001000;          // CANTX->Output CANRX->Input
```

```
        LATB=0;
        TRISC=0;                    // RC0-RC7: Inicialmente como salidas digitales
        LATC=0;
        TRIS_BOTON_ON=1;
        TRIS_LED_OFF=0;
530     LED_OFF=0;

// Configuracion ADC
        OpenADC(ADC_FOSC_64&        // Con un reloj de 20 MHz el mnimo es 32,
                ADC_RIGHT_JUST&     // introducimos un pequenyo margen.
                ADC_8ANA_OREF,      // AN0-AN7: entradas analogicas
                ADC_CH0&            // Activado el canal AN0 por defecto
                ADC_INT_OFF);
        Delay10TCYx(5);

540 // Habilito INT
        RCONbits.IPEN=1;            // Habilito int de dos niveles
        INTCON2=0b11111111;       // INT TMR0 en alta prioridad
}

void Setup_CAN(void)
{
      // Inicializacion del modulo CAN segun nota de aplicacion AN738

        CANInitialize(SJW, BRP, PHSEG1, PHSEG2, PROPSEG,CAN_CONFIG_VALID_STD_MSG &
550                                             CAN_CONFIG_DBL_BUFFER_ON);
      // Set CAN module into configuration mode
        CANSetOperationMode(CAN_OP_MODE_CONFIG);

      // Set Buffer 1 Mask value
        CANSetMask(CAN_MASK_B1,MASK1,TIPO_TRAMA);
      // Filtros buffer1: al principio solo se admiten tramas con ID=ID_CONEXION
        CANSetFilter(CAN_FILTER_B1_F1,ID_CONEXION,TIPO_TRAMA);
        CANSetFilter(CAN_FILTER_B1_F2,ID_CONEXION,TIPO_TRAMA);

560     // Set Buffer 2 Mask value
        CANSetMask(CAN_MASK_B2,MASK2,TIPO_TRAMA);
      // Filtros buffer2: al principio solo se admiten tramas con ID=ID_CONEXION
        CANSetFilter(CAN_FILTER_B2_F1,ID_CONEXION,TIPO_TRAMA);
        CANSetFilter(CAN_FILTER_B2_F2,ID_CONEXION,TIPO_TRAMA);

      // CAN en modo normal
        CANSetOperationMode(CAN_OP_MODE_NORMAL);

        PIE3=0b00000011;            // INT de CAN
570     IPR3=0b00000011;            // CAN en alta prioridad
}

//-----------------------------------------------------------------------
// main()
//-----------------------------------------------------------------------

void main(void)
{
   Setup_CAN();                             // Llamada a Setup_CAN()
580 while(1){
    Setup();                                // Llamada a Setup()
    while(BOTON_ON==0);                     // Condicion de inicio
    Delay1KTCYx(25);                        // Retraso para evitar rebotes
    while(BOTON_ON==1);
    Delay1KTCYx(25);                        // Retraso para evitar rebotes
    envio_dato[0]=PETICION_ID;              // Indica que es peticion de ID
    *((ulong*)(envio_dato+1))=MAC;          // Dato[1-3]=Direccion MAC
    if(CANIsTxReady()){                     // Solicitamos un identificador
      CANSendMessage(ID_CONTROLADOR,
590             envio_dato,
```

```
                    4,
                      CAN_TX_PRIORITY_0 &
                      CAN_TX_STD_FRAME &
                      CAN_TX_NO_RTR_FRAME);
      }
      INTCON=0b11000000;                        // Habilito INT de RB0 y las globales
      Delay10KTCYx(250);                        // Retraso para garantizar correcta inicializacion
        while(1){                               // BUCLE PRINCIPAL
        Nop();
        while(BOTON_OFF==0);                    // Secuencia de desconexion
        Delay1KTCYx(25);
        while(BOTON_OFF==1);
        Delay1KTCYx(25);                        // Retraso para evitar rebotes
        envio_dato[0]=PETICION_DESCONEXION;     // Indica solicitud de desconexion
        envio_dato[1]=ID_nodo;                  // ID del nodo que lo pide
        *((uslong*)(envio_dato+2))=MAC;         // MAC del nodo que lo pide
        if(CANIsTxReady()){                    // Se envia el comando de desconexion
          CANSendMessage(ID_CONTROLADOR,
                      envio_dato,
                       5,
                       CAN_TX_PRIORITY_0 &
                       CAN_TX_STD_FRAME &
                       CAN_TX_NO_RTR_FRAME);
        }
        break;                                // Salimos del bucle para esperar una nueva conexion
       }
    }
  }
```

Apéndice D

Esquema del nodo universal

Este apéndice contiene:

- Esquema electrónico nodo universal (2 hojas).
- *Layout* del nodo universal, cara de componentes.
- *Layout* del nodo universal, cara de pista.

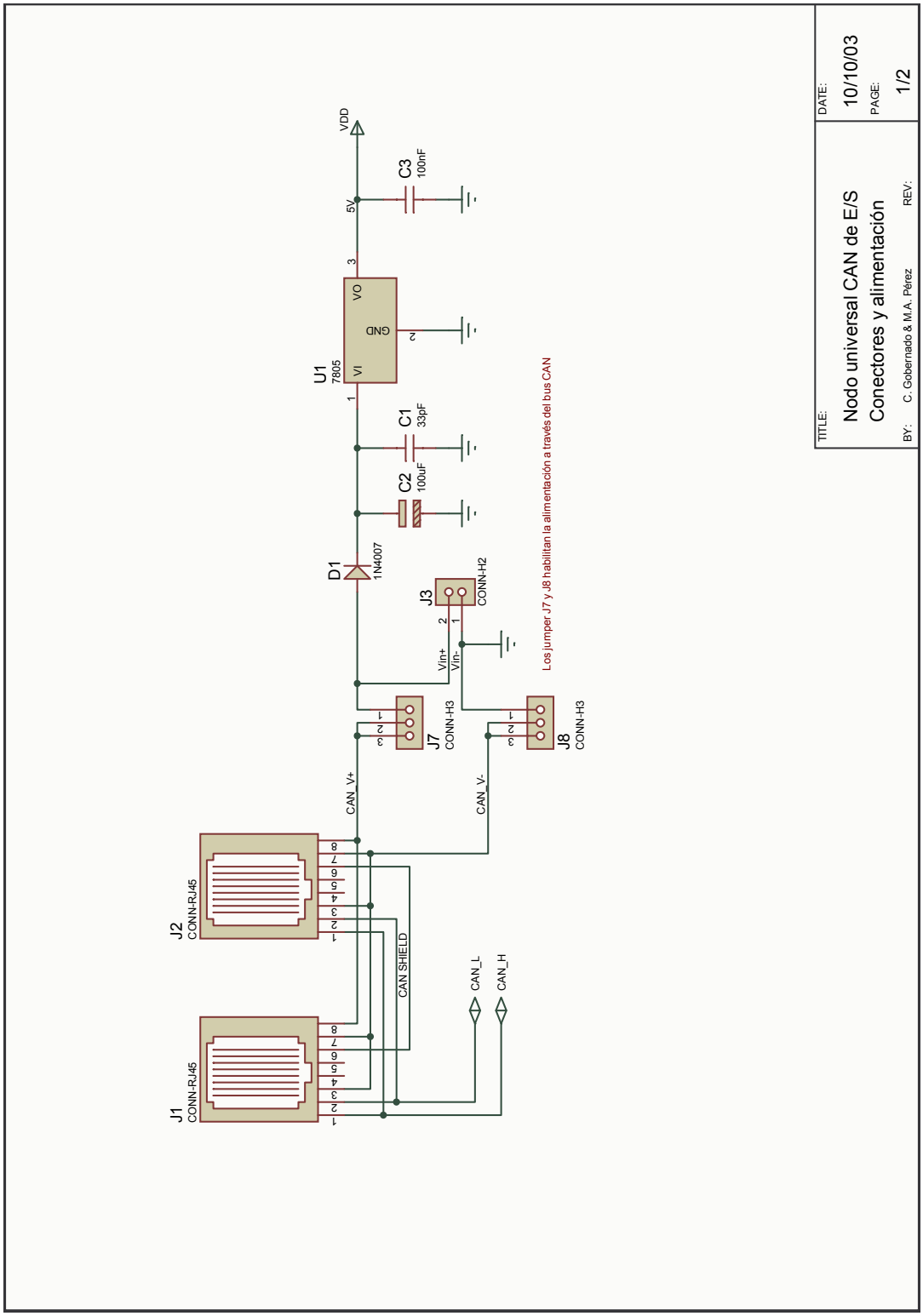


Figura D.1: Esquema nodo universal (hoja1).

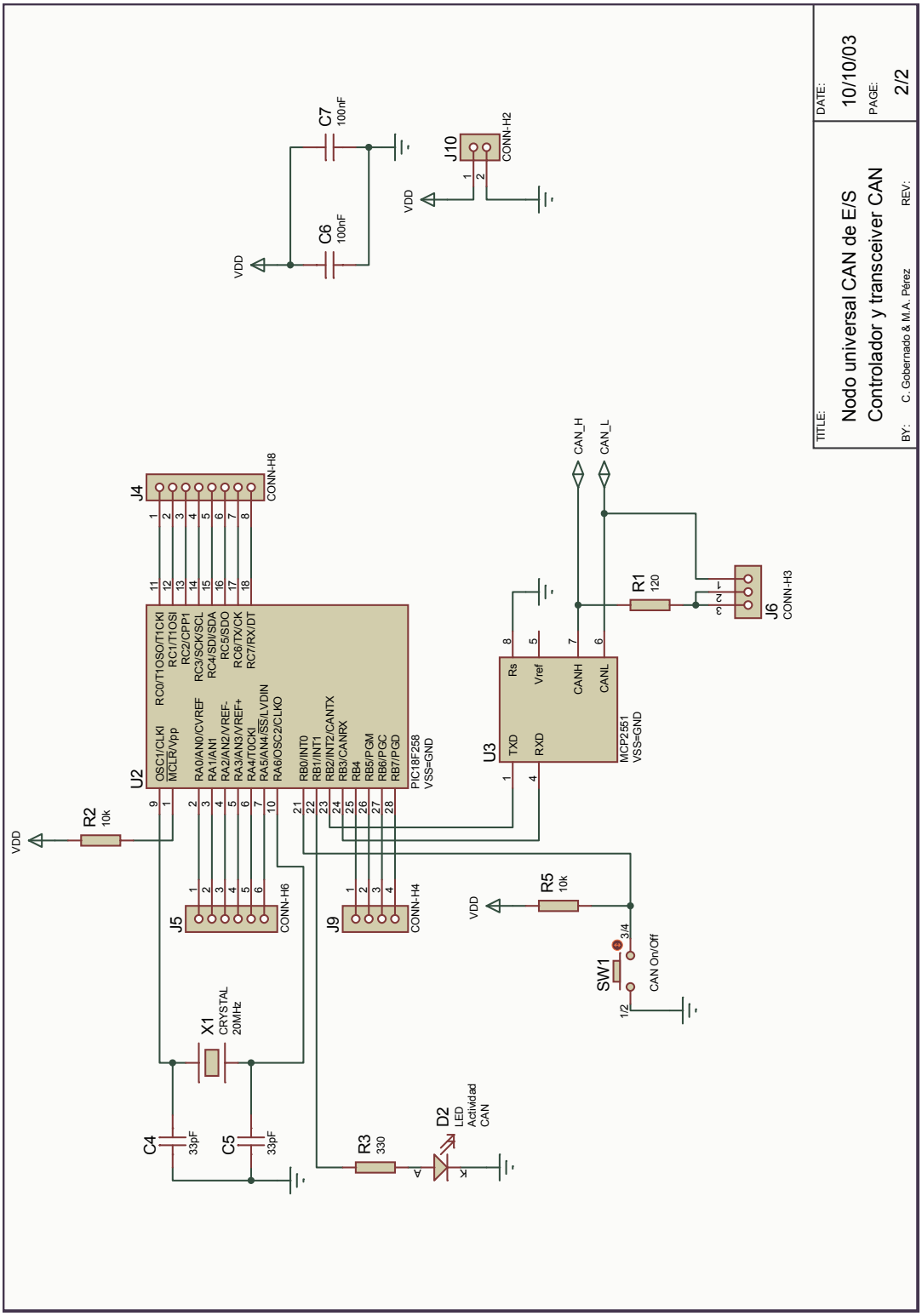


Figura D.2: Esquema nodo universal (hoja2).

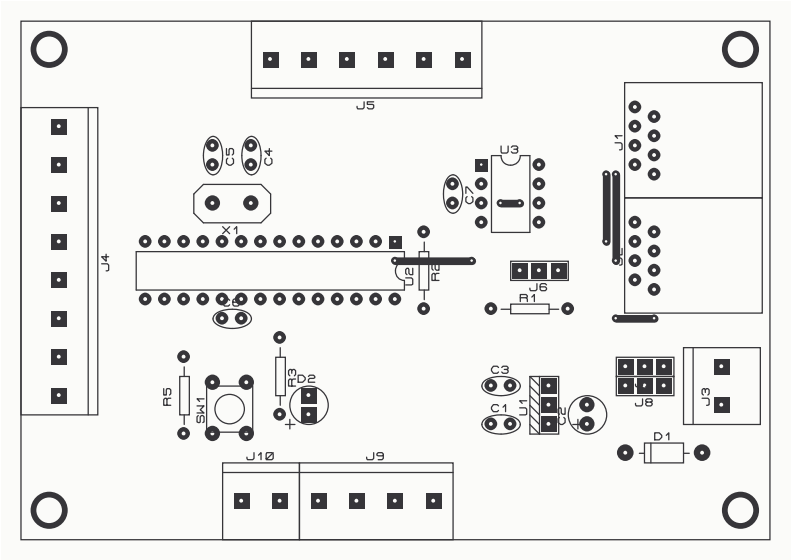


Figura D.3: Cara superior de componentes del nodo universal *CAN*.

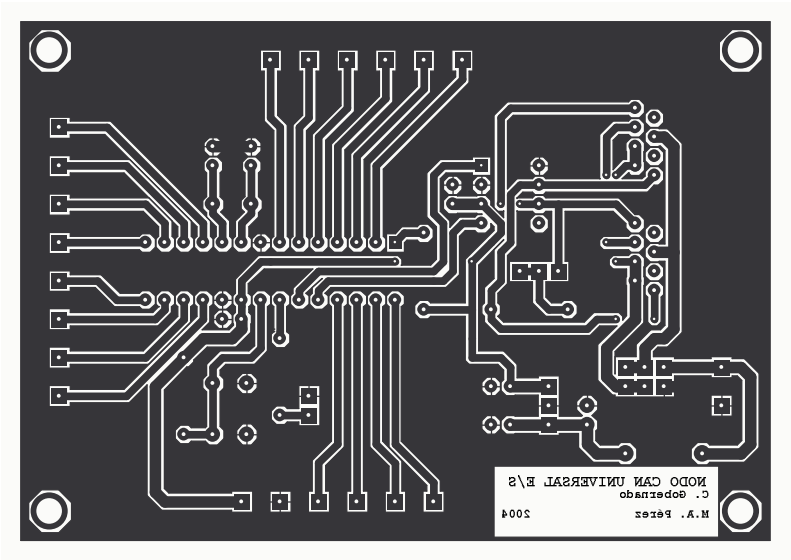


Figura D.4: Cara inferior de pistas del nodo universal *CAN*.

Apéndice E

Depurador RS232

Este apéndice contiene:

- Esquema electrónico de la conexión RS232.
- *Layout* de la cara de componentes.
- *Layout* de la cara de pistas.
- Código fuente de la biblioteca de funciones DEBUG.

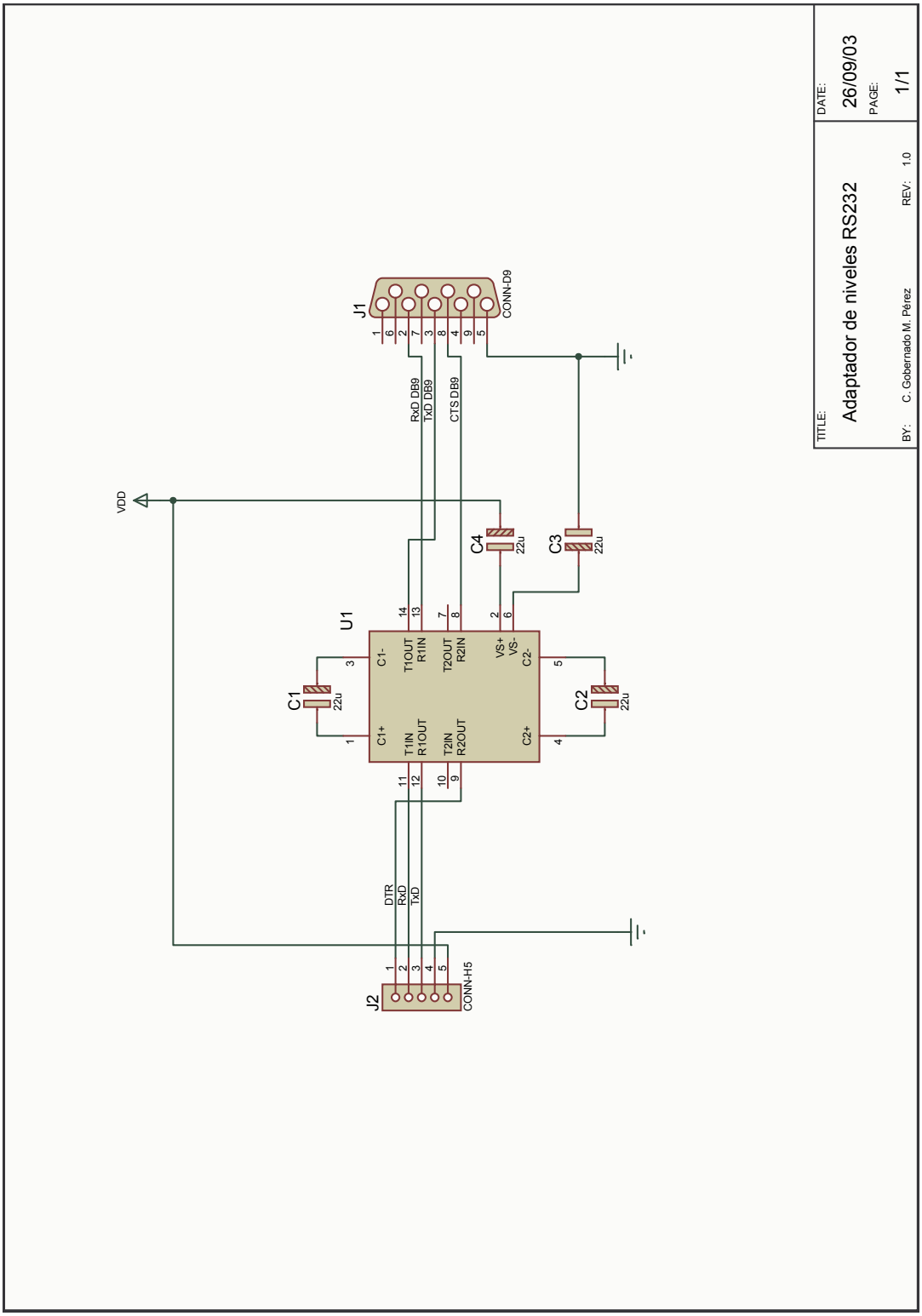


Figura E.1: Esquema electrónico conexión RS232.

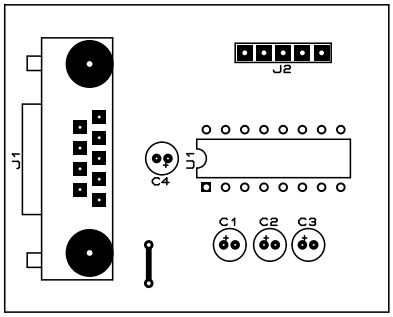


Figura E.2: Cara de componentes del depurador RS232.

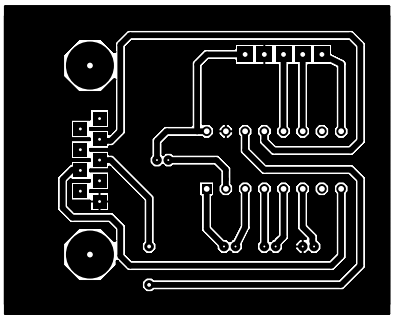


Figura E.3: Cara de pistas del depurador RS232.

Listado E.1: Fichero debug.h.

```
//-------------------------------------------------------------------
//
//                            DESCRIPCION
//
// Libreria para ayudar a depurar con un pc.
// Se encarga de transmitir bia RS232 informacion a un pc
//
//-------------------------------------------------------------------
//
//    Archivo:              debug.h
//
//    Autores:      Miguel Angel Perez
//                  Carlos Gobernado
//
//    Version:  1.10
//
//    Version 1.00 ->1.10
//    En la macro RETORNO_CARRO() ahora se hace \r\n
//    Se ha modificado Prints y se ha ñaaido un parametro para evitar
//    Mandar \0
//-------------------------------------------------------------------


//-------------------------------------------------------------------
// Definicion de constantes
//-------------------------------------------------------------------

#ifndef __DEBUG_H
#define __DEBUG_H

// El define siguiente hace \r\n
#define RETORNO_CARRO() while(BusyUSART()); putcUSART(0xD);
                        while(BusyUSART()); putcUSART(0xA)

#define ESPACIO()     while(BusyUSART()); putcUSART(0x20)



typedef unsigned char uchar;
typedef unsigned int uint;
typedef unsigned long ulong;

//-------------------------------------------------------------------
//Estructuras de datos
//-------------------------------------------------------------------


//-------------------------------------------------------------------
// Prototipos de funciones
//-------------------------------------------------------------------

void InitDEBUG(void);
void Printrs(const rom char *);
void Prints (char *,uint);
void PrintCharHex(char );
void PrintInt(const rom char*,uint);
void PrintChar(const rom char*,unsigned char);
void PrintLong(const rom char*,ulong);
void PrintSector512(char *);

//void RETORNO_CARRO();

#endif
```

Listado E.2: Fichero debug.c.

```
//-------------------------------------------------------------------
//
//                            DESCRIPCION
//
// Libreria para ayudar a depurar programas en el pc via RS232
//
//-------------------------------------------------------------------
//
//    Archivo:              debug.c
//
//    Autores:      Miguel Angel Perez
//                  Carlos Gobernado
//
//    Version:  1.10
//
//    Version 1.00 ->1.10
//    En la macro RETORNO_CARRO() ahora se hace \r\n
//    Se ha modificado Prints y se ha ñaaido un parametro para evitar
//    Mandar \0
//
//-------------------------------------------------------------------
#include <p18cxxx.h>
#include <usart.h>
#include <stdlib.h>
#include <ctype.h>
#include "debug.h"

char datoTemp=0;

//-------------------------------------------------------------------
//
// void InitDEBUG()
//
// Inicializa el puerto USART
//
//
//-------------------------------------------------------------------

void InitDEBUG(void){
  // Inicializacion del USART para ayudar a depurar
  // SPRBG 129  9600
  // SPRBG 21 56818
  // SPRBG 9  125000
  OpenUSART(USART_TX_INT_OFF&
      USART_RX_INT_OFF&
      USART_ASYNCH_MODE&
      USART_EIGHT_BIT&
      USART_SINGLE_RX&
      USART_BRGH_HIGH,
      21);
  putcUSART(13);
  RETORNO_CARRO();
  putrsUSART("Inicializado␣mensajes␣DEBUG");
  RETORNO_CARRO();
}

//-------------------------------------------------------------------
//
// Printrs( const rom char *data)
//
// Manda un string de la memoria de programa y un retorno de carro
//
//
//-------------------------------------------------------------------
void Printrs(const rom char *data){
```

```
  putrsUSART(data);
  RETORNO_CARRO();
}

//-----------------------------------------------------------------
//
// Prints( char *data, uint len)
//
// Escribe el buffer data al puertos USART.
// len especifica el ñtamao del buffer.
// No empleo la ófuncin putsUSART(char *data) para evitar mandar el fin
// de string (\0).
//
//
//-----------------------------------------------------------------
void Prints( char *data, uint len){
  int i;
  for(i=0;i<len;i++){
    while(BusyUSART());
    WriteUSART(data[i]);
  }
  //RETORNO_CARRO();
}

//-----------------------------------------------------------------
//
// PrintCharHex( char data)
//
// Manda dos char para representar numeros en hexadecimal
//
//
//-----------------------------------------------------------------
void PrintCharHex(char data){
  char c2=0x0F;          // Para enmascarar
  char c3=0xF0;          //
  datoTemp=data&c3;      // Swapf no vale con un dato de la pila
  Swapf(datoTemp,1,1);   // Roto el nible y lo guardo en registro
  if(datoTemp<10)
    datoTemp+=48;
  else
    datoTemp+=55;
  while(BusyUSART());
  WriteUSART(datoTemp);
  datoTemp=data&c2;
  if(datoTemp<10)
    datoTemp+=48;
  else
    datoTemp+=55;
  while(BusyUSART());
  WriteUSART(datoTemp);
}
//-----------------------------------------------------------------
//
// void PrintInt(const rom char* nomVariable,unsigned int dato)
//
// Escribe el nombre de la variable y acontinuacion en dato en Hex
//
//
//-----------------------------------------------------------------
void PrintInt(const rom char* nomVariable,unsigned int dato){
  char *p=&dato;
  putrsUSART(nomVariable);
  ESPACIO();                    // Macro definida en .h
  PrintCharHex(*(p+1));         // Primero el byte alto
  PrintCharHex(*p);             // Segundo byte bajo
  RETORNO_CARRO();
```

```
}
//-----------------------------------------------------------------
//
// void PrintChar(const rom char* nomVariable,unsigned char dato)
//
// Escribe el nombre de la variable y acontinuacion en dato en Hex
//
//
//-----------------------------------------------------------------
void PrintChar(const rom char* nomVariable,unsigned char dato){
  putrsUSART(nomVariable);
  ESPACIO();                // Macro definida en .h
  PrintCharHex(dato);
  RETORNO_CARRO();
}

//-----------------------------------------------------------------
//
// void PrintLong(const rom char* nomVariable,ulong dato)
//
// Escribe el nombre de la variable y acontinuacion en dato en Hex
//
//
//-----------------------------------------------------------------
void PrintLong(const rom char* nomVariable,ulong dato){
  char *p=&dato;
  putrsUSART(nomVariable);
  ESPACIO();                    // Macro definida en .h
  PrintCharHex(*(p+3));         //º 4byte
  PrintCharHex(*(p+2));
  PrintCharHex(*(p+1));
  PrintCharHex(*p);             //º 1byte
  RETORNO_CARRO();
}

//-----------------------------------------------------------------
//
// void PrintSector512(char * buffer)
//
// Esta funcion áest destinada a imprimir los sectores tanto en hex
// como en ASCII al igual que hace los editores de texto.
//
//
//-----------------------------------------------------------------
void PrintSector512(char *buffer){
  uchar i,j;
  uint  k=0;
  for(i=0;i<32;i++){
    // Imprimo en hexadecimal
    for(j=0;j<16;j++){
      while(BusyUSART());
      PrintCharHex(buffer[k]);
      k++;
    }
    ESPACIO();
    ESPACIO();
    // Imprimo lo mismo en formato texto
    k-=16;
    for(j=0;j<16;j++){
      while(BusyUSART());
      if(isalnum(buffer[k])!=0)
        WriteUSART(buffer[k]);
      else
        WriteUSART(0x2E); // El punto
      k++;
    }
```

```
    RETORNO_CARRO();
  }
}
/*
void RETORNO_CARRO(){
  while(BusyUSART());
  putcUSART(13);
} */
```

Apéndice F

Código fuente de CANSoft

F.1 Código fuente ficheros

Listado F.1: Fichero Dialog.frm

```
    morecomment
Option Explicit
Const FrecReloj As Double = 20
Dim Tq As Double
Dim nTq As Integer
Dim SyncSeg As Integer
Dim PropSeg As Integer
Dim PhaseSeg1 As Integer
Dim PhaseSeg2 As Integer
Dim SJW As Integer
Dim BRP As Integer

' Mensajes de error
Const MsgErrorNumeroTq As String = "El únmero de tq que forman un bit " _
                                 & "debe de estar comprendido entre " _
                                 & "8 Tq y 25 Tq"
Const MsgErrorSuma As String = "Los segmentos que forman el bit no " _
                             & "suman el únmero de Tq calculado"
Const MsgErrorCond1 As String = " Prop_Seg+Phase_Seg1 >= Phase_Seg2"
Const MsgErrorCond2 As String = " PhaseSeg2 > SJW "

Private Function ActualizarVariables()
    SyncSeg = Val(txtSyncSeg)
    PropSeg = Val(cboPropSeg)
    PhaseSeg1 = Val(cboPhaseSeg1)
    PhaseSeg2 = Val(cboPhaseSeg2)
    SJW = Val(cboSJW)
    BRP = Val(cboBRP)
End Function
Private Function CalcularTq()
    ' Esta ófuncin se encarga dado una tasa de ótransmisin y
    ' un BRP calcular tq y el únmero de tq que tiene un bit
    Tq = (2 * (cboBRP + 1)) / FrecReloj
    nTq = 1 / ((cboTasaTx / 1000) * Tq)
    If nTq < 8 Or nTq > 25 Then
        MsgBox "El únmero de tq que forma un bit con la " _
             & "ócombinacin de BRP y tasa de ótransmisin" _
               & " es de " & nTq & Chr(13) _
               & MsgErrorNumeroTq, vbExclamation, "Error"
    Else
```

```
        cboNumeroTq.Text = cboNumeroTq.List(nTq - 8)
morecomment    End If
End Function

Private Function ComprobarCombinaciones() As Boolean
    ' Esta ófuncin comprueba si las combinaciones elegidas
    ' son las correctas.
    '° 1 Comprobar que Sync+Prop+Phase1+Phase2=numeroTq
    If SyncSeg + PropSeg + PhaseSeg1 + PhaseSeg2 <> cboNumeroTq Then
       'Error
       MsgBox MsgErrorSuma, vbExclamation, "Error"
       ComprobarCombinaciones = False
    Else
        '° 2 Comprobar Prop_Seg+Phase_Seg1 >= Phase_Seg2
        If PropSeg + PhaseSeg1 >= PhaseSeg2 Then
            '° 3 Comprobar Phase_Seg2 > SJW
            If PhaseSeg2 > SJW Then
                ComprobarCombinaciones = True
            Else
                MsgBox MsgErrorCond2, vbExclamation, "Error"
                ComprobarCombinaciones = False
            End If
        Else
            MsgBox MsgErrorCond1, vbExclamation, "Error"
            ComprobarCombinaciones = False
        End If
    End If
End Function

Private Sub CancelButton_Click()
    ' No hacemos nada y nos ocultamos
    Dialog.Hide
End Sub

Private Sub cboBRP_Validate(Cancel As Boolean)
    CalcularTq
End Sub

Private Sub cboModoOperacion_Click()
    If cboModoOperacion = "ESCUCHA" Then
        ' Deshabilito los filtros y mascaras
        txtMascara1.Enabled = False
        txtF1_B1.Enabled = False
        txtF2_B1.Enabled = False
        txtMascara2.Enabled = False
        txtF1_B2.Enabled = False
        txtF2_B2.Enabled = False
        txtF3_B2.Enabled = False
        txtF4_B2.Enabled = False
    Else
        ' Habilito filtros y mascaras
        txtMascara1.Enabled = True
        txtF1_B1.Enabled = True
        txtF2_B1.Enabled = True
        txtMascara2.Enabled = True
        txtF1_B2.Enabled = True
        txtF2_B2.Enabled = True
        txtF3_B2.Enabled = True
        txtF4_B2.Enabled = True
    End If
End Sub

Private Sub cboTasaTx_Validate(Cancel As Boolean)
    CalcularTq
End Sub
```

```
Private Sub cmdGuardar_Click()
morecomment     ' Guardamos la configuracion en el fichero CONFIG.TXT
    ' Previamente ídeberamos comprobar que todo áest en úmaysculas
    On Error GoTo cmdGuardar_ClickError
    ' Actualizamos las variables
    ActualizarVariables
    'Comprobamos que las combinaciones elegidas sean las correctas
    If ComprobarCombinaciones() Then
    ' abrimos el ficehro en modo binario
        Open drvUnidad & "\CONFIG.TXT" For Binary As 1
            ' Parametros temporales
            ' Tipo ID
            Put #1, 1, BRP
            Put #1, 2, SJW
            Put #1, 3, PropSeg
            Put #1, 4, PhaseSeg1
            Put #1, 5, PhaseSeg2
            ' Tipo ID esto valores áestn sacados de can18xx8.h
            If cboTipoID = "ESTANDAR" Then
                Put #1, 6, &HFF ' tipo buffer 1
                Put #1, 7, &HFF ' tipo buffer 2
            ElseIf cboTipoID = "EXTENDIDO" Then
                Put #1, 6, &HF7
                Put #1, 7, &HF7
            Else
                Put #1, 6, &HFF
                Put #1, 7, &HF7
            End If
            ' Buffer1
            If cboModoOperacion = "NORMAL" Then
                ' En modo normal acepta filtros
                Put #1, 8, CLng("&h" & txtMascara1)
                Put #1, 12, CLng("&h" & txtF1_B1)
                Put #1, 16, CLng("&h" & txtF2_B1)
                ' Buffer2
                Put #1, 20, CLng("&h" & txtMascara2)
                Put #1, 24, CLng("&h" & txtF1_B2)
                Put #1, 28, CLng("&h" & txtF2_B2)
                Put #1, 32, CLng("&h" & txtF3_B2)
                Put #1, 36, CLng("&h" & txtF4_B2)
            ElseIf cboModoOperacion = "ESCUCHA" Then
                ' En modo escucha no funcionan los filtros
                ' pero de todas formas los pongo a 0
                Put #1, 8, CLng(0)
                Put #1, 12, CLng(0)
                Put #1, 16, CLng(0)
                ' Buffer2
                Put #1, 20, CLng(0)
                Put #1, 24, CLng(0)
                Put #1, 28, CLng(0)
                Put #1, 32, CLng(0)
                Put #1, 36, CLng(0)
            End If
            ' Modo de operacion, los valores de can18xx8.h
            If cboModoOperacion = "ESCUCHA" Then
                Put #1, 40, &H60 ' CAN_OP_MODE_LISTEN
            Else
                Put #1, 40, &H0  'Modo Normal CAN_OP_MODE_NORMAL
            End If
            ' Tipo de ID para configurar el ómdulo
            ' los valores corresponden con la enumeracion
            ' enum CAN_CONFIG_FLAGS

            If cboTipoID = "ESTANDAR" Then
                Put #1, 41, &HBF ' CAN_CONFIG_VALID_STD_MSG
            ElseIf cboTipoID = "EXTENDIDO" Then
```

```
                Put #1, 41, &HDF ' CAN_CONFIG_VALID_XTD_MSG
morecomment             Else
                Put #1, 41, &H9F ' CAN_CONFIG_ALL_VALID_MSG
            End If
        Close #1

        ' Preparo el fichero de captura y mando un caracter para que
        ' en la fat le asignen un cluster de partida
        Open drvUnidad & "\CAPTURA.DAT" For Output As 1
            Print #1, "a"
        Close #1
        MsgBox "Ficheros Generados", vbInformation, "Aviso"
        Me.Hide
    End If
    Exit Sub
cmdGuardar_ClickError:
    MsgBox Err.Description, vbExclamation
End Sub

Private Sub Form_Load()
    Dim i As Integer
    cboTipoID.AddItem "ESTANDAR", 0
    cboTipoID.AddItem "EXTENDIDO", 1
    cboTipoID.AddItem "AMBOS", 2
    cboModoOperacion.AddItem "NORMAL", 0
    cboModoOperacion.AddItem "ESCUCHA", 1
    For i = 0 To 63
        cboBRP.AddItem i
    Next
    For i = 8 To 25
        cboNumeroTq.AddItem i
    Next
    Call CalcularTq

End Sub
```

Listado F.2: Fichero frmMain.frm

```
    morecomment
Private Type dato
    indice      As Long
    tiempo      As Long
    longitud    As Byte
    tipo        As Long
    datos(8)    As Byte
    id          As Long
End Type
Dim captura() As dato
Private Declare Function OSWinHelp% Lib "user32" Alias "WinHelpA" _
(ByVal hwnd&, ByVal HelpFile$, ByVal wCommand%, dwData As Any)

Private Sub cmbTiempo_Change()
    datosFormulario (tbToolBar.Buttons(4).Value)
End Sub

Private Sub cmbTiempo_Click()
    On Error Resume Next
    datosFormulario (tbToolBar.Buttons(4).Value)
End Sub

Private Sub Form_Load()
    LoadResStrings Me
    Me.Left = GetSetting(App.Title, "Settings", "MainLeft", 1000)
    Me.Top = GetSetting(App.Title, "Settings", "MainTop", 1000)
    Me.Width = GetSetting(App.Title, "Settings", "MainWidth", 6500)
    Me.Height = GetSetting(App.Title, "Settings", "MainHeight", 6500)

    ' óInicializacin del control gridDatos
    s = "^Captura|^Instante de tiempo|^Tipo trama|^Identificador|" _
      & "^Dato[0] |^Dato[1] |^Dato[2] |^Dato[3] |^" _
      & "Dato[4] |^Dato[5] |^Dato[6] |^Dato[7] "
    gridDatos.FormatString = s

    ' Inicializo el combo Tiempo
   cmbTiempo.AddItem "1 ms", 0
   cmbTiempo.AddItem "1 s", 1
   cmbTiempo.AddItem "60 s", 2
End Sub

Private Sub Form_Resize()
    If Me.Width > 100 Then gridDatos.Width = Me.Width - 100
    If Me.Height > 1550 Then gridDatos.Height = Me.Height - 1550

   ' Codigo para colocar el combo en el barrra
   ' Button object using the Move method.
   With tbToolBar.Buttons(6) 'indice del combo de tiempo en la barra
      cmbTiempo.Move .Left, .Top, .Width
      cmbTiempo.ZOrder 0
   End With

   ' Codigo para colocar el progress Bar en la barra
   ' Button object using the Move method.
   With tbToolBar.Buttons(8) 'indice del combo de tiempo en la barra
      pgbCarga.Move .Left, .Top, .Width
      pgbCarga.ZOrder 0
      pgbCarga.Visible = False
   End With
End Sub

Private Sub Form_Unload(Cancel As Integer)
    Dim i As Integer
```

```
        'close all sub forms
    morecomment    For i = Forms.Count - 1 To 1 Step -1
            Unload Forms(i)
        Next
    70  If Me.WindowState <> vbMinimized Then
            SaveSetting App.Title, "Settings", "MainLeft", Me.Left
            SaveSetting App.Title, "Settings", "MainTop", Me.Top
            SaveSetting App.Title, "Settings", "MainWidth", Me.Width
            SaveSetting App.Title, "Settings", "MainHeight", Me.Height
        End If
    End Sub

    Private Sub tbToolBar_ButtonClick(ByVal Button As MSComCtlLib.Button)
        On Error Resume Next
    80  Select Case Button.Key
            Case "Abrir"
                mnuFileOpen_Click
            Case "Guardar"
                mnuFileSave_Click
            Case "Hex"
                'MsgBox "Agregar ócdigo de óbotn 'Hex'."
                datosFormulario (tbToolBar.Buttons(4).Value)
            'Case "Imprimir"
             '   mnuFilePrint_Click
    90      'Case "Buscar"
                'TareasPendientes: Agregar ócdigo de óbotn 'Buscar'.
             '   MsgBox "Agregar ócdigo de óbotn 'Buscar'."
            'Case "Tiempo"
                'Modifica la escala de tiempo de los datos.
            '    MsgBox "Agregar ócdigo de óbotn 'Tiempo'."
        End Select
    End Sub

    Private Sub mnuHelpAbout_Click()
    100 frmAbout.Show vbModal, Me
    End Sub

    Private Sub mnuHelpSearchForHelpOn_Click()
        Dim nRet As Integer


        'si no hay archivo de ayuda para este proyecto, mostrar un mensaje
        'al usuario puede establecer el archivo de Ayuda para su óaplicacin
        'en el cuadro de ádilogo Propiedades del proyecto
    110 If Len(App.HelpFile) = 0 Then
            MsgBox "No se puede mostrar el contenido de la Ayuda." _
            & " No hay Ayuda asociada a este proyecto.", _
            vbInformation, Me.Caption
        Else
            On Error Resume Next
            nRet = OSWinHelp(Me.hwnd, App.HelpFile, 261, 0)
            If Err Then
                MsgBox Err.Description
            End If
    120 End If

    End Sub

    Private Sub mnuHelpContents_Click()
        Dim nRet As Integer


        'si no hay archivo de ayuda para este proyecto, mostrar un mensaje
        'al usuario puede establecer el archivo de Ayuda para la óaplicacin
    130 'en el cuadro de ádilogo Propiedades del proyecto
        If Len(App.HelpFile) = 0 Then
```

```
        MsgBox "No se puede mostrar el contenido de la Ayuda." _
morecomment              & " No hay Ayuda asociada a este proyecto.", _
              vbInformation, Me.Caption
    Else
        On Error Resume Next
        nRet = OSWinHelp(Me.hwnd, App.HelpFile, 3, 0)
        If Err Then
            MsgBox Err.Description
        End If
    End If

End Sub

Private Sub mnuWindowArrangeIcons_Click()
    'TareasPendientes: Agregar ócdigo 'mnuWindowArrangeIcons_Click'.
    MsgBox "Agregar ócdigo 'mnuWindowArrangeIcons_Click'."
End Sub

Private Sub mnuWindowTileVertical_Click()
    'TareasPendientes: Agregar ócdigo 'mnuWindowTileVertical_Click'.
    MsgBox "Agregar ócdigo 'mnuWindowTileVertical_Click'."
End Sub

Private Sub mnuWindowTileHorizontal_Click()
    'TareasPendientes: Agregar ócdigo 'mnuWindowTileHorizontal_Click'.
    MsgBox "Agregar ócdigo 'mnuWindowTileHorizontal_Click'."
End Sub

Private Sub mnuWindowCascade_Click()
    'TareasPendientes: Agregar ócdigo 'mnuWindowCascade_Click'.
    MsgBox "Agregar ócdigo 'mnuWindowCascade_Click'."
End Sub

Private Sub mnuWindowNewWindow_Click()
    'TareasPendientes: Agregar ócdigo 'mnuWindowNewWindow_Click'.
    MsgBox "Agregar ócdigo 'mnuWindowNewWindow_Click'."
End Sub

Private Sub mnuEditBuscar_Click()
    'TareasPendientes: Agregar ócdigo 'mnuEditBuscar_Click'.
    MsgBox "Agregar ócdigo 'mnuEditBuscar_Click'."
End Sub

Private Sub mnuEditPasteSpecial_Click()
    'TareasPendientes: Agregar ócdigo 'mnuEditPasteSpecial_Click'.
    MsgBox "Agregar ócdigo 'mnuEditPasteSpecial_Click'."
End Sub

Private Sub mnuEditPaste_Click()
    'TareasPendientes: Agregar ócdigo 'mnuEditPaste_Click'.
    MsgBox "Agregar ócdigo 'mnuEditPaste_Click'."
End Sub

Private Sub mnuEditCopy_Click()
    'TareasPendientes: Agregar ócdigo 'mnuEditCopy_Click'.
    MsgBox "Agregar ócdigo 'mnuEditCopy_Click'."
End Sub

Private Sub mnuEditCut_Click()
    'TareasPendientes: Agregar ócdigo 'mnuEditCut_Click'.
    MsgBox "Agregar ócdigo 'mnuEditCut_Click'."
End Sub

Private Sub mnuEditUndo_Click()
    'TareasPendientes: Agregar ócdigo 'mnuEditUndo_Click'.
    MsgBox "Agregar ócdigo 'mnuEditUndo_Click'."
```

```
End Sub
morecomment
Private Sub mnuViewStatusBar_Click()
    mnuViewStatusBar.Checked = Not mnuViewStatusBar.Checked
    sbStatusBar.Visible = mnuViewStatusBar.Checked
End Sub

Private Sub mnuViewToolbar_Click()
    mnuViewToolbar.Checked = Not mnuViewToolbar.Checked
    tbToolBar.Visible = mnuViewToolbar.Checked
End Sub

Private Sub mnuFileExit_Click()
    'descargar el formulario
    Unload Me

End Sub

Private Sub mnuFilePrint_Click()
    'TareasPendientes: Agregar ócdigo 'mnuFilePrint_Click'.
    MsgBox "Agregar ócdigo 'mnuFilePrint_Click'."
End Sub

Private Sub mnuFilePrintPreview_Click()
    'TareasPendientes: Agregar ócdigo 'mnuFilePrintPreview_Click'.
    MsgBox "Agregar ócdigo 'mnuFilePrintPreview_Click'."
End Sub

Private Sub mnuFilePageSetup_Click()
    On Error Resume Next
    With dlgCommonDialog
        .DialogTitle = "Configurar ápgina"
        .CancelError = True
        .ShowPrinter
    End With

End Sub

Private Sub mnuFileProperties_Click()
    'TareasPendientes: Agregar ócdigo 'mnuFileProperties_Click'.
    MsgBox "Agregar ócdigo 'mnuFileProperties_Click'."
End Sub

Private Sub mnuFileSave_Click()
    'TareasPendientes: Agregar ócdigo 'mnuFileSave_Click'.
    'MsgBox "Agregar ócdigo 'mnuFileSave_Click'."
    Dialog.Show vbModal
End Sub

Private Sub mnuFileClose_Click()
    'TareasPendientes: Agregar ócdigo 'mnuFileClose_Click'.
    MsgBox "Agregar ócdigo 'mnuFileClose_Click'."
End Sub

Private Sub mnuFileOpen_Click()
    Dim sFile As String
    Dim longitud As Long
    Dim a As FileListBox

    With dlgCommonDialog
        .DialogTitle = "Abrir"
        .CancelError = True
        .Filter = "Archivos de captura (*.dat)|*.dat"
        .ShowOpen
        longitud = Len(.FileName)
        If Len(.FileName) = 0 Then
```

```
                Exit Sub
morecomment         End If
            sFile = .FileName
            On Error GoTo ErrorHandler
        End With
        'MsgBox "Valor:" & dlgCommonDialog.CancelError
        'Pendiente: agregar ócdigo para procesar el archivo abierto
        'If dlgCommonDialog.CancelError = False Then
            cargaDatos sFile
            datosFormulario (tbToolBar.Buttons(4).Value)
        'End If

    ErrorHandler:
    End Sub
    Private Sub cargaDatos(nombre As String)
        ' Esta ófuncin se encarga de abrir el fichero de captura y cargar
        ' los datos en la matriz captura() As dato. Esta matriz debe de ser
        ' dimensionada previamente.

        Dim i, j As Long        ' i=indice de capturas j=para carga de datos
        Dim longitud As Long    ' ñtamao del archivo a abrir para
        Dim tempTiempo As Long  ' dimensionar la matriz
        Dim tempId As Long
        Dim temp As Long
        On Error GoTo cargaDatosError
        Open nombre For Binary Access Read As 1
            longitud = LOF(1)
            ReDim captura(longitud / 16)    ' dimensiono matriz de datos
            pgbCarga.Max = (longitud / 16)
            pgbCarga.Min = 0
            pgbCarga.Visible = True
            For i = 0 To (longitud / 16) - 1
                pgbCarga.Value = i
                captura(i).indice = i
                Get #1, (1 + i * 16), tempTiempo  ' obtengo tiempo
                captura(i).tiempo = tempTiempo And &HFFFFFFFF
                ' para coger la longitud enmascaro elú ltimo byte
                ' y roto dividiendo por 2^29
                temp = ((tempTiempo And &HF0000000) / &H10000000)
                If temp < 0 Then
                   captura(i).longitud = 8
                Else
                    captura(i).longitud = temp
                End If
                For j = 0 To 7    ' leo el campo de datos
                    If j <= captura(i).longitud Then
                        Get #1, (5 + i * 16) + j, captura(i).datos(j)
                    Else
                        captura(i).datos(j) = 0
                    End If
                Next
                Get #1, (13 + i * 16), tempId
                                                ' obtengo el id enmascarando
                captura(i).id = tempId And &H1FFFFFFF

                ' para coger los tresú ltimos bits enmascaro
                captura(i).tipo = (tempId And &HE0000000)
            Next
        Close #1
        pgbCarga.Visible = False
        Exit Sub
    cargaDatosError:
        MsgBox Err.Description, vbExclamation, Error
        Resume Next
    End Sub
```

```
330 Private Sub datosFormulario(ByVal datos As Integer)
    morecomment     ' Esta ófuncin se encarga de cargar los datos de la matriz captura
        ' en el formulario con las opciones deseadas por el usuario.
        ' El parametros datos indica si debe de ser HEX o DEC el campo de datos

        Dim i, j As Long
        Dim temp As String
        Dim temp2 As String ' para obtener un valor numerico de la escala de tiempo
        gridDatos.Rows = 2
        gridDatos.Clear
340      ' óInicializacin del control gridDatos
        s = "^Captura|^Instante de tiempo|^Tipo trama|^Identificador|" _
          & "^Dato[0] |^Dato[1] |^Dato[2] |^Dato[3] |^" _
          & "Dato[4] |^Dato[5] |^Dato[6] |^Dato[7] "
        gridDatos.FormatString = s
        pgbCarga.Max = UBound(captura, 1)
        pgbCarga.Min = 0
        pgbCarga.Visible = True
        For i = 0 To UBound(captura, 1) - 1
            pgbCarga.Value = i
350         gridDatos.AddItem formateaDatos(i, datos), i + 1
        Next
        pgbCarga.Visible = False
    End Sub

    Private Function formateaDatos(ByVal indice As Long, _
                                   ByVal datos As Integer) As String
        ' Esta ófuncin se encarga de formatear el contendio de captura
        ' para hacerlo áms atractivo al usuario y devuelve el string
        ' necesario para el control gridDatos
360     Dim temp As String
        Dim temp2 As Double
        Dim j As Integer
        ' ñAado indice de captura
        temp = captura(indice).indice & vbTab
        ' ñAado el tiempo con el preformateo
        temp = temp & calculaTiempo(captura(indice).tiempo, cmbTiempo.Text) _
                    & vbTab
        Select Case captura(indice).tipo
            Case &H0
370             temp = temp & "áEstndar" & vbTab
            Case &H20000000
                temp = temp & "Extendida" & vbTab
            Case &H40000000
                temp = temp & "Remota" & vbTab
            Case &H60000000
                temp = temp & "Error" & vbTab
            Case &H80000000
                temp = temp & "áOvr_Estndar" & vbTab
            Case &HA0000000
380             temp = temp & "Ovr_Extendida" & vbTab
            Case &HC0000000
                temp = temp & "Ovr_Remota" & vbTab
            Case &HE0000000
                temp = temp & "Ovr_Error" & vbTab
        End Select
        If captura(indice).tipo <> &H60000000 Then
            temp = temp & "0x" & Hex(captura(indice).id)
        End If

390     If captura(indice).tipo < &H40000000 Then
            ' Es trama de datos y tendra datos
            j = 0
            While j < captura(indice).longitud
                If datos = tbrPressed Then
                    temp = temp & vbTab & "0x" & Hex(captura(indice).datos(j))
```

```
            Else
morecomment                 temp = temp & vbTab & captura(indice).datos(j)
            End If
            j = j + 1
        Wend
    End If
    formateaDatos = temp
End Function
Private Function calculaTiempo(ByVal tiempo As Long, _
                               ByVal escala As String) As String
    ' esta funcion fija la escala de tiempo, el micro muestrea cada 51,2 us
    Dim temp As Double
    Select Case escala
        Case "1 ms"
            temp = (tiempo * 51.2) / 1000
            calculaTiempo = Round(temp, 2)
        Case "1 s"
            temp = (tiempo * 51.2) / 1000000
            calculaTiempo = Round(temp, 5)
        Case "60 s"
            temp = (tiempo * 51.2) / 60000000
            calculaTiempo = Round(temp, 6)
    End Select

End Function
```

Apéndice G

Imágenes del desarrollo del proyecto

En este último apéndice mostramos algunas imágenes ilustrativas de distintas fases de nuestro proyecto. En la figura G.1 mostramos el montaje experimental en placa de prototipos de 2 nodos trasmitiendo mediante bus *CAN* (imagen de la izquierda) y el analizador capturando el tráfico (imagen de la derecha). La figura G.2 muestra con más detalle el montaje de prueba del analizador.

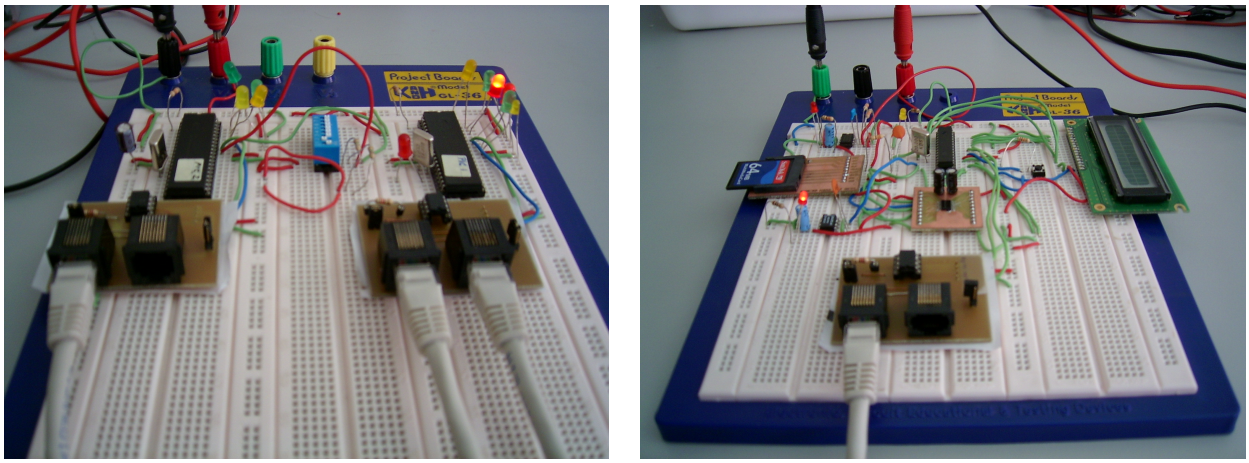


Figura G.1: Montaje de prueba experimental.

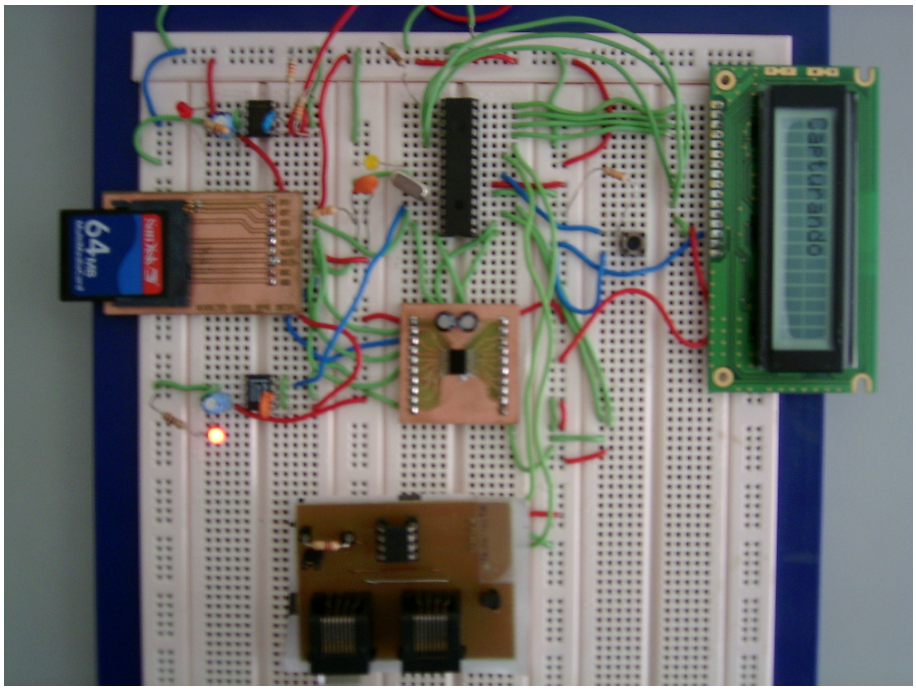


Figura G.2: Vista cenital del montaje en placa de prototipos del analizador.

La figura G.3 es otra imagen de otro montaje de prueba de una red *CAN*, esta vez ya probando los nodos universales.

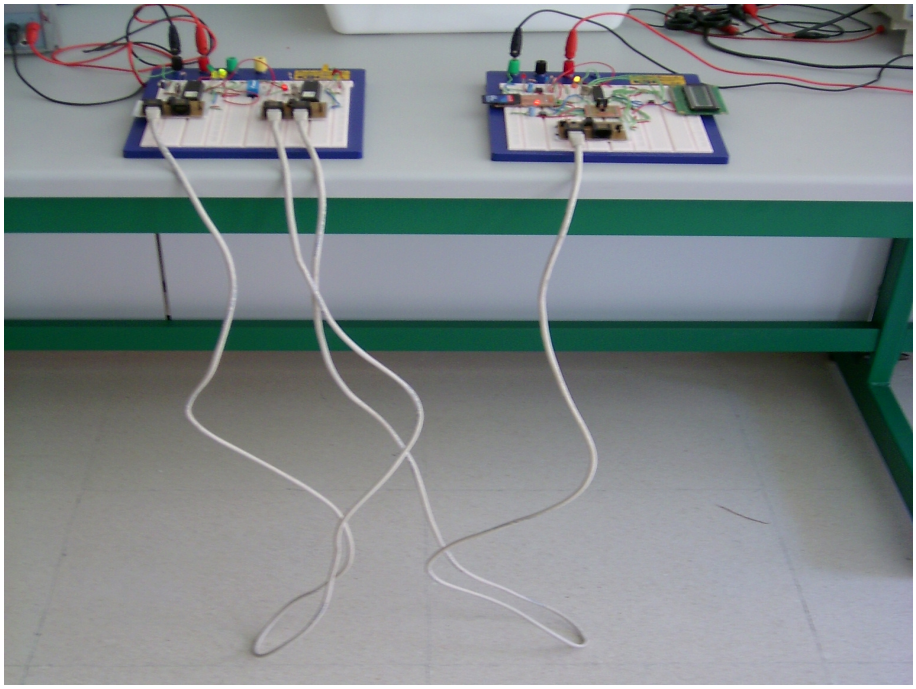


Figura G.3: Imagen de dos nodos universales conectados.

Las figuras G.4 y G.5 muestran un nodo universal desde dos ángulos distintos, uno mostrando el puerto A del microcontrolador de las entradas analógicas y el puerto digital C completo, y el otro, presentando los cuatro pines de la parte alta del puerto digital B.

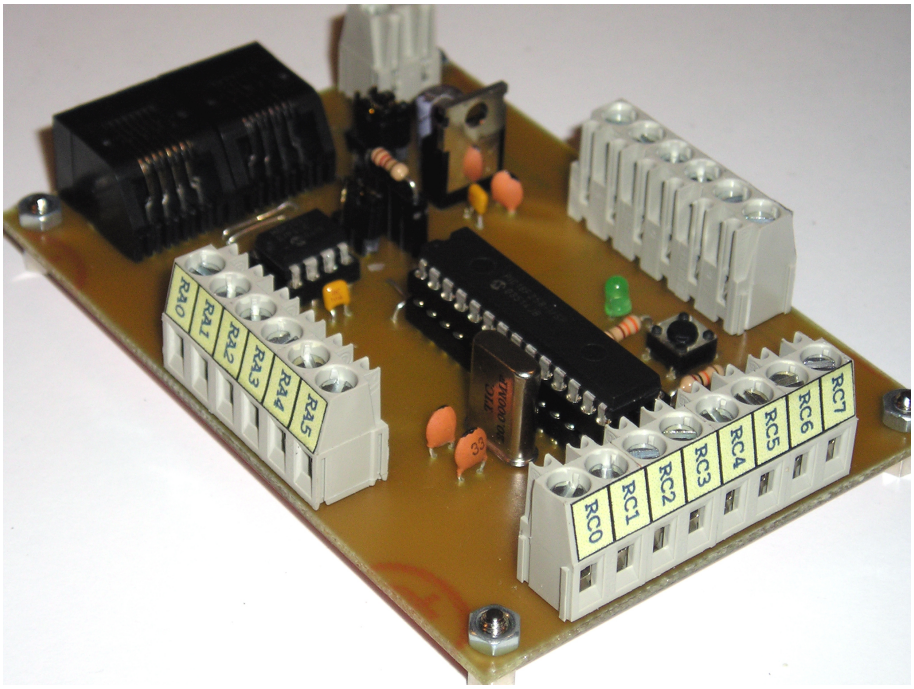


Figura G.4: Imagen del nodo universal (vista1).

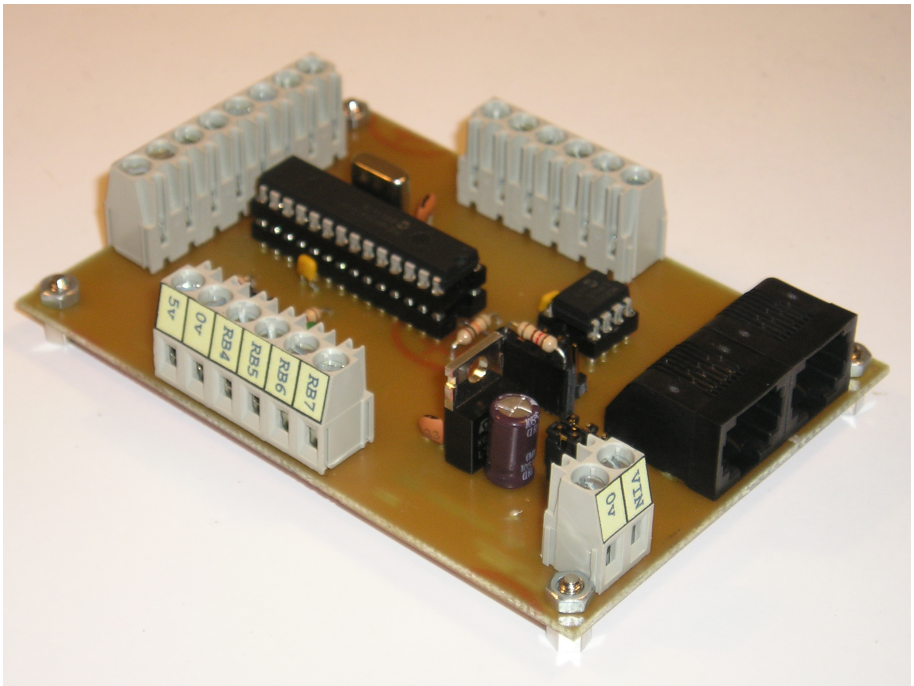


Figura G.5: Imagen del nodo universal (vista2).

Las placas PCB fueron realizadas con una máquina fresadora. En las figuras G.6 y G.7 nos enseñan distintas instantáneas del proceso.

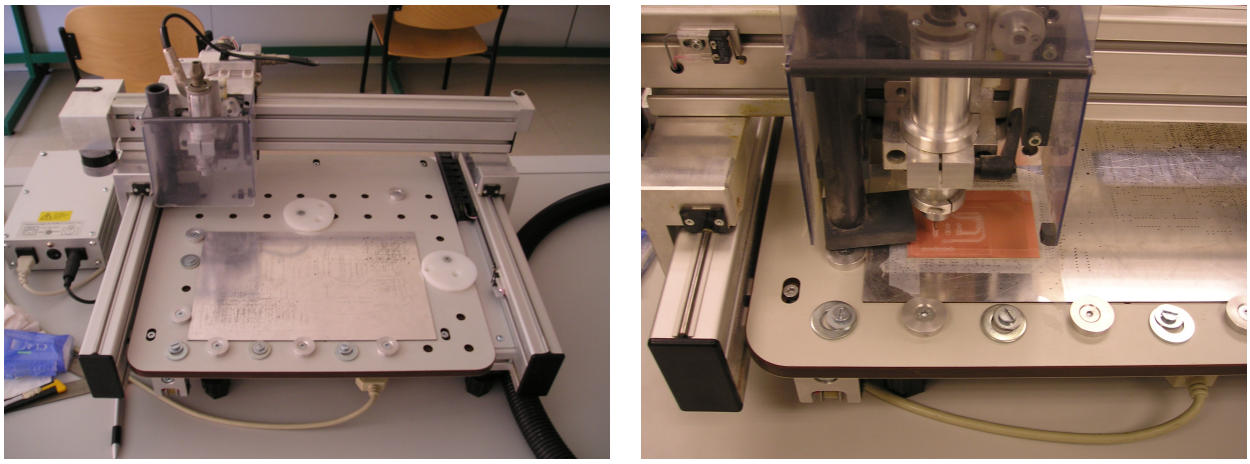


Figura G.6: Vistas de la máquina fresadora y de la realización de la placa.

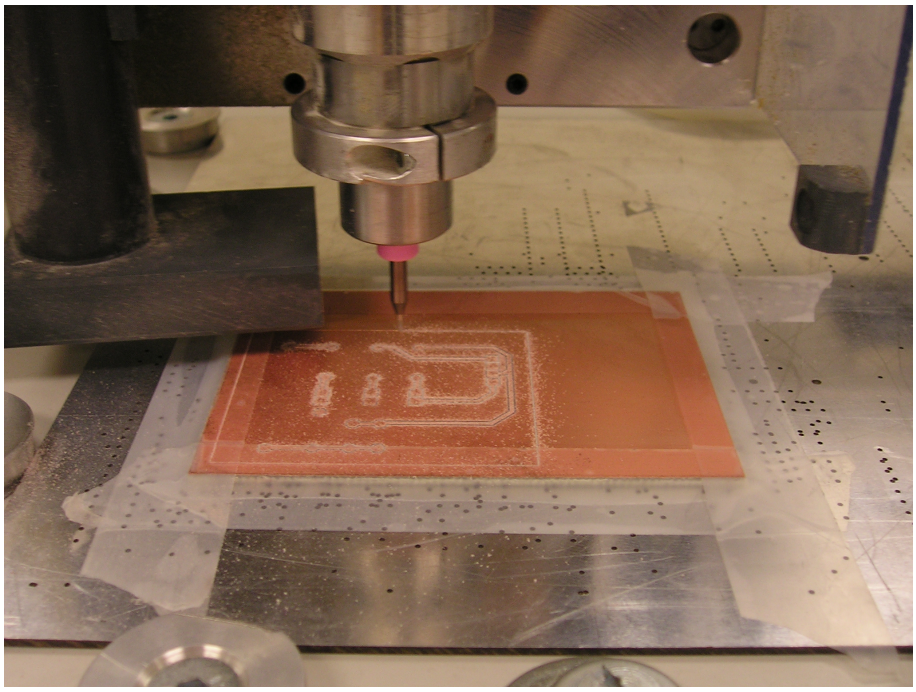


Figura G.7: Vista de la realización de la placa en detalle.

Apéndice H

Datasheets

Este apéndice contiene los siguientes *datasheet*:

- *Transceiver CAN* MCP2551 (20 páginas).
- Conversor de niveles MAX3002 (23 páginas).
- Reguladores lineales MAX883/MAX884 (15 páginas).
- Zócalo para tarjetas de memoria MMC y SD FPS009-3202 (2 páginas).
- LCD compatible con controlador HD44780 (2 páginas).
- Regulador 7805 (16 páginas).
- Sensor de temperatura LM35 (12 páginas).
- Puente en H cuádruple L239 (13 páginas).

MCP2551

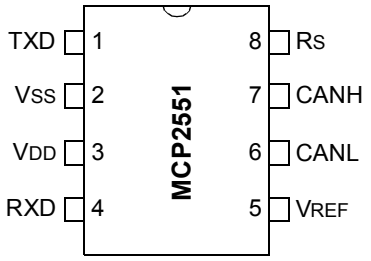
High-Speed CAN Transceiver

Features

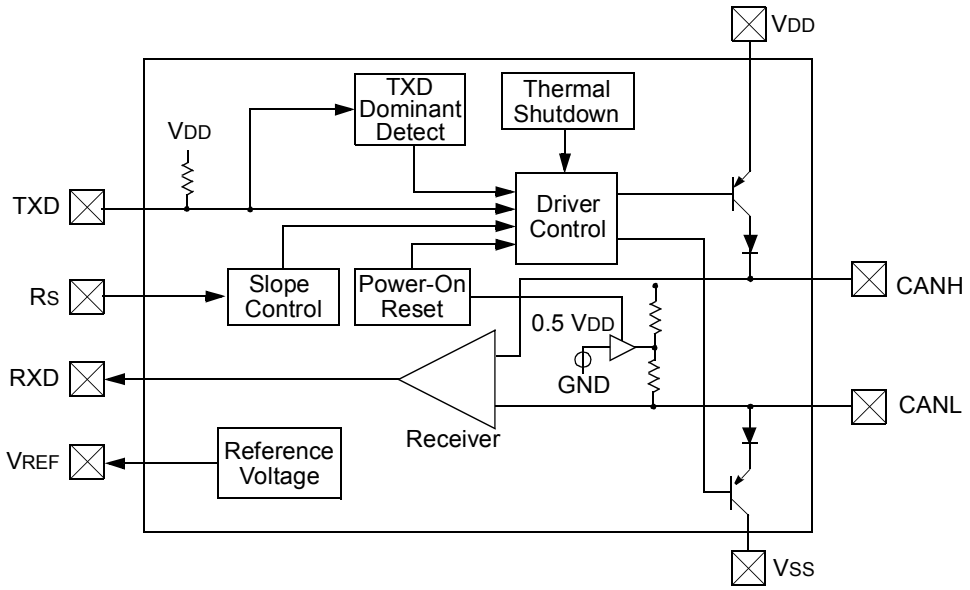
- Supports 1 Mb/s operation
- Implements ISO-11898 standard physical layer requirements
- Suitable for 12V and 24V systems
- Externally-controlled slope for reduced RFI emissions
- Detection of ground fault (permanent dominant) on TXD input
- Power-on reset and voltage brown-out protection
- An unpowered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

Package Types

PDIP/SOIC



Block Diagram



NOTES:

1.0 DEVICE OVERVIEW

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

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

1.1 Transmitter Function

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

1.1.1 MAXIMUM NUMBER OF NODES

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

1.2 Receiver Function

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

1.3 Internal Protection

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

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

1.4 Operating Modes

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

- High-Speed
- Slope-Control
- Standby

These modes are summarized in Table 1-1.

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

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

1.4.1 HIGH-SPEED

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

1.4.2 SLOPE-CONTROL

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

1.4.3 STANDBY MODE

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

TABLE 1-1: MODES OF OPERATION

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

TABLE 1-2: TRANSCEIVER TRUTH TABLE

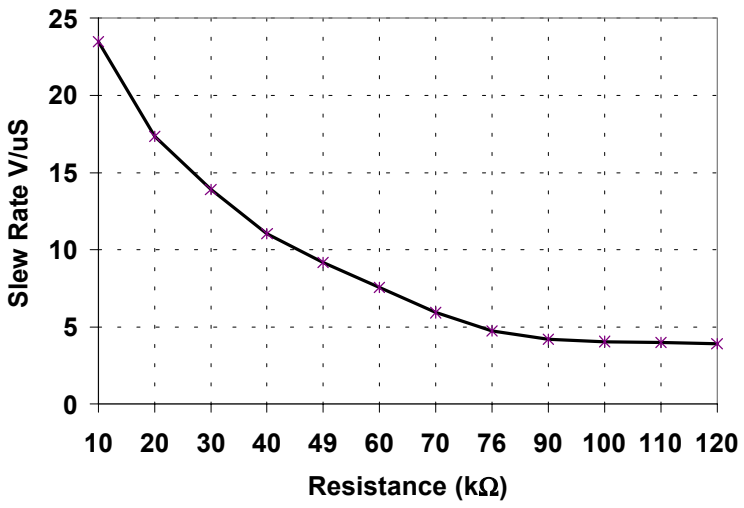
V_{DD}	V_{RS}	TXD	CANH	CANL	Bus State(1)	RXD(1)
$4.5V \le V_{DD} \le 5.5V$	$V_{RS} < 0.75\ V_{DD}$	0	HIGH	LOW	Dominant	0
		1 or floating	Not Driven	Not Driven	Recessive	1
	$V_{RS} > 0.75\ V_{DD}$	X	Not Driven	Not Driven	Recessive	1
$V_{POR} < V_{DD} < 4.5V$ (See **Note 3**)	$V_{RS} < 0.75\ V_{DD}$	0	HIGH	LOW	Dominant	0
		1 or floating	Not Driven	Not Driven	Recessive	1
	$V_{RS} > 0.75\ V_{DD}$	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 ensured to meet the ISO-11898 specification.

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



1.5 TXD Permanent Dominant Detection

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

1.6 Power-on Reset

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

1.7 Pin Descriptions

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

TABLE 1-3: MCP2551 PINOUT

Pin Number	Pin Name	Pin Function
1	TXD	Transmit Data Input
2	V_{SS}	Ground
3	V_{DD}	Supply Voltage
4	RXD	Receive Data Output
5	V_{REF}	Reference Output Voltage
6	CANL	CAN Low-Level Voltage I/O
7	CANH	CAN High-Level Voltage I/O
8	R_S	Slope-Control Input

1.7.1 TRANSMITTER DATA INPUT (TXD)

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

1.7.2 GROUND SUPPLY (V_{SS})

Ground supply pin.

1.7.3 SUPPLY VOLTAGE (V_{DD})

Positive supply voltage pin.

1.7.4 RECEIVER DATA OUTPUT (RXD)

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

1.7.5 REFERENCE VOLTAGE (V_{REF})

Reference Voltage Output (Defined as $V_{DD}/2$).

1.7.6 CAN LOW (CANL)

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

1.7.7 CAN HIGH (CANH)

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

1.7.8 SLOPE RESISTOR INPUT (R_S)

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

NOTES:

2.0 ELECTRICAL CHARACTERISTICS

2.1 Terms and Definitions

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

2.1.1 BUS VOLTAGE

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

2.1.2 COMMON MODE BUS VOLTAGE RANGE

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

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

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

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

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

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

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

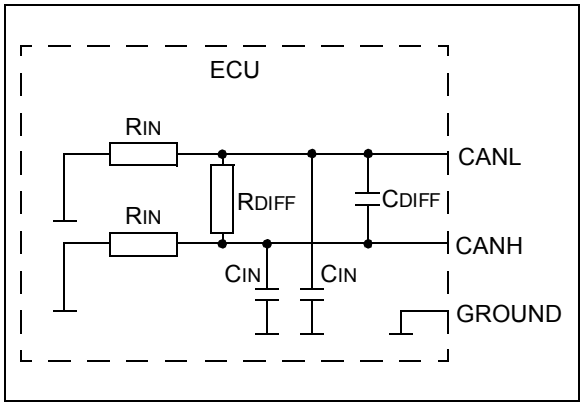
2.1.6 INTERNAL CAPACITANCE, C_{IN} (OF A CAN NODE)

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

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

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

FIGURE 2-1: PHYSICAL LAYER DEFINITIONS



Absolute Maximum Ratings†

V_{DD} 7.0V

DC Voltage at TXD, RXD, V_{REF} and V_S ..... -0.3V to V_{DD} + 0.3V

DC Voltage at CANH, CANL (**Note 1**) ..... -42V to +42V

Transient Voltage on Pins 6 and 7 (**Note 2**) ..... -250V to +250V

Storage temperature ..... -55°C to +150°C

Operating ambient temperature ..... -40°C to +125°C

Virtual Junction Temperature, T_{VJ} (**Note 3**) ..... -40°C to +150°C

Soldering temperature of leads (10 seconds) ..... +300°C

ESD protection on CANH and CANL pins (**Note 4**) ..... 6 kV

ESD protection on all other pins (**Note 4**) ..... 4 kV

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

2: In accordance with ISO-7637.

3: In accordance with IEC 60747-1.

4: Classification A: Human Body Model.

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

2.2 DC Characteristics

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

Note 1: This parameter is periodically sampled and not 100% tested.
2: I_{TXD} = I_{RXD} = I_{VREF} = 0 mA; 0V < V_{CANL} < V_{DD}; 0V < V_{CANH} < V_{DD}; V_{RS} = V_{DD}.
3: This is valid for the receiver in all modes; High-speed, Slope-control and Standby.

2.2 DC Characteristics (Continued)

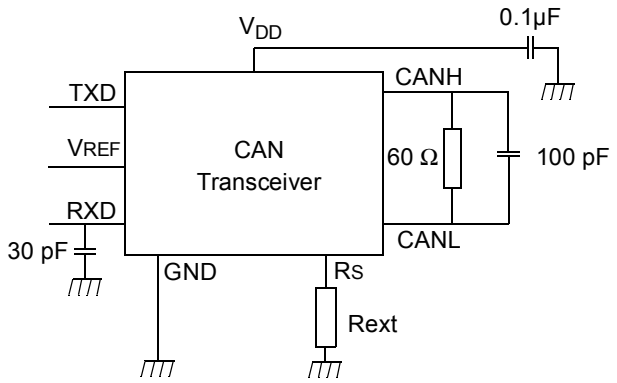
DC Specifications (Continued)			Electrical Characteristics: Industrial (I): T_{AMB} = -40°C to +85°C V_{DD} = 4.5V to 5.5V Extended (E): T_{AMB} = -40°C to +125°C V_{DD} = 4.5V to 5.5V			
Param No.	**Sym**	**Characteristic**	**Min**	**Max**	**Units**	**Conditions**
Bus Line (CANH; CANL) Receiver: [TXD = 2V; pins 6 and 7 externally driven]						
D22	R_{DIFF}	Differential input resistance	20	100	kΩ	
D24	I_{LI}	CANH, CANL input leakage current	—	150	µA	$V_{DD} < V_{POR}$; $V_{CANH} = V_{CANL}$ = +5V
Transmitter Data Input (TXD)						
D25	V_{IH}	High-level input voltage	2.0	V_{DD}	V	Output recessive
D26	V_{IL}	Low-level input voltage	V_{SS}	+0.8	V	Output dominant
D27	I_{IH}	High-level input current	-1	+1	µA	$V_{TXD} = V_{DD}$
D28	I_{IL}	Low-level input current	-100	-400	µA	V_{TXD} = 0V
Receiver Data Output (RXD)						
D31	V_{OH}	High-level output voltage	0.7 V_{DD}	—	V	I_{OH} = 8 mA
D32	V_{OL}	Low-level output voltage	—	0.8	V	I_{OL} = 8 mA
Voltage Reference Output (V_{REF})						
D33	V_{REF}	Reference output voltage	0.45 V_{DD}	0.55 V_{DD}	V	-50 µA < I_{VREF} < 50 µA
Standby/Slope-Control (Rs pin)						
D34	V_{STB}	Input voltage for standby mode	0.75 V_{DD}	—	V	
D35	I_{SLOPE}	Slope-control mode current	-10	-200	µA	
D36	V_{SLOPE}	Slope-control mode voltage	0.4 V_{DD}	0.6 V_{DD}	V	
Thermal Shutdown						
D37	$T_{J(sd)}$	Shutdown junction temperature	155	180	°C	**Note 1**
D38	$T_{J(h)}$	Shutdown temperature hysteresis	20	30	°C	-12V < $V_{(CANL, CANH)}$ < +12V **(Note 3)**

Note 1: This parameter is periodically sampled and not 100% tested.

2: $I_{TXD} = I_{RXD} = I_{VREF}$ = 0 mA; 0V < V_{CANL} < V_{DD}; 0V < V_{CANH} < V_{DD}; $V_{RS} = V_{DD}$.

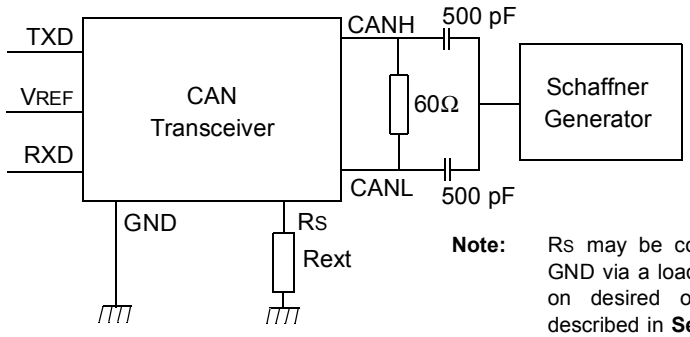
3: This is valid for the receiver in all modes; High-speed, Slope-control and Standby.

FIGURE 2-1: TEST CIRCUIT FOR ELECTRICAL CHARACTERISTICS



Note: R_S may be connected to V_{DD} or GND via a load resistor depending on desired operating mode as described in **Section 1.7.8, "Slope Resistor Input (Rs)"**.

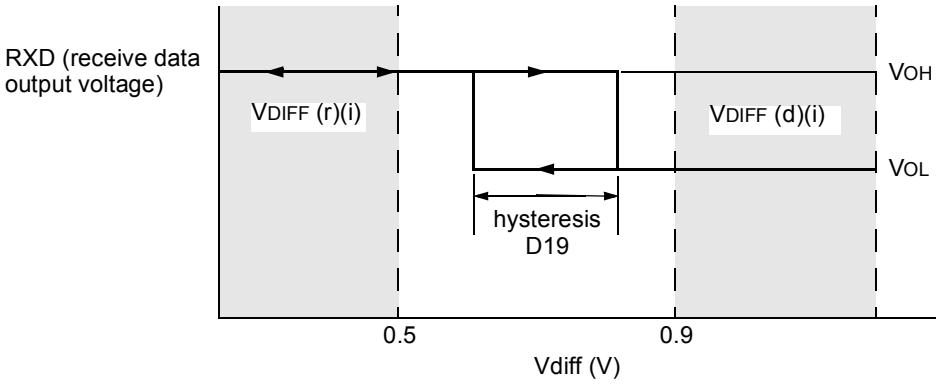
FIGURE 2-2: TEST CIRCUIT FOR AUTOMOTIVE TRANSIENTS



Note: RS may be connected to VDD or GND via a load resistor depending on desired operating mode as described in **Section 1.7.8, "Slope Resistor Input (Rs)"**

The wave forms of the applied transients shall be in accordance with "ISO-7637, Part 1", test pulses 1, 2, 3a and 3b.

FIGURE 2-3: HYSTERESIS OF THE RECEIVER



2.3 AC Characteristics

AC Specifications			Electrical Characteristics: Industrial (I): T_{AMB} = -40°C to +85°C V_{DD} = 4.5V to 5.5V Extended (E): T_{AMB} = -40°C to +125°C V_{DD} = 4.5V to 5.5V			
Param No.	**Sym**	**Characteristic**	**Min**	**Max**	**Units**	**Conditions**
1	t_{BIT}	Bit time	1	62.5	µs	V_{RS} = 0V
2	f_{BIT}	Bit frequency	16	1000	kHz	V_{RS} = 0V
3	TtxL2bus(d)	Delay TXD to bus active	—	70	ns	-40°C ≤ T_{AMB} ≤ +125°C, V_{RS} = 0V
4	TtxH2bus(r)	Delay TXD to bus inactive	—	125	ns	-40°C ≤ T_{AMB} ≤ +85°C, V_{RS} = 0V
			—	170	ns	-40°C ≤ T_{AMB} ≤ +125°C, V_{RS} = 0V
5	TtxL2rx(d)	Delay TXD to receive active	—	130	ns	-40°C ≤ T_{AMB} ≤ +125°C, V_{RS} = 0V
			—	250	ns	-40°C ≤ T_{AMB} ≤ +125°C, R_S = 47 kΩ
6	TtxH2rx(r)	Delay TXD to receiver inactive	—	175	ns	-40°C ≤ T_{AMB} ≤ +85°C, V_{RS} = 0V
			—	225	ns	-40°C ≤ T_{AMB} ≤ +85°C, R_S = 47 kΩ
			—	235	ns	-40°C ≤ T_{AMB} ≤ +125°C, V_{RS} = 0V
			—	400	ns	-40°C ≤ T_{AMB} ≤ +125°C, R_S = 47 kΩ
7	SR	CANH, CANL slew rate	5.5	8.5	V/µs	Refer to Figure 1-1; R_S = 47 kΩ, (**Note 1)**
10	t_{WAKE}	Wake-up time from standby (Rs pin)	—	5	µs	see Figure 2-5
11	TbusD2rx(s)	Bus dominant to RXD Low (Standby mode)	—	550	ns	V_{RS} = +4V; (see **Figure 2-2**)
12	$C_{IN(CANH)}$ $C_{IN(CANL)}$	CANH; CANL input capacitance	—	20 (typical)	pF	1 Mbit/s data rate; V_{TXD} = V_{DD}, **(Note 1)**
13	C_{DIFF}	Differential input capacitance	—	10 (typical)	pF	1 Mbit/s data rate **(Note 1)**
14	TtxL2busZ	TX Permanent Dominant Timer Disable Time	1.25	4	ms	
15	TtxR2pdt(res)	TX Permanent Dominant Timer Reset Time	—	1	µs	Rising edge on TXD while device is in permanent dominant state

Note 1: This parameter is periodically sampled and not 100% tested.

2.4 Timing Diagrams and Specifications

FIGURE 2-4: TIMING DIAGRAM FOR AC CHARACTERISTICS

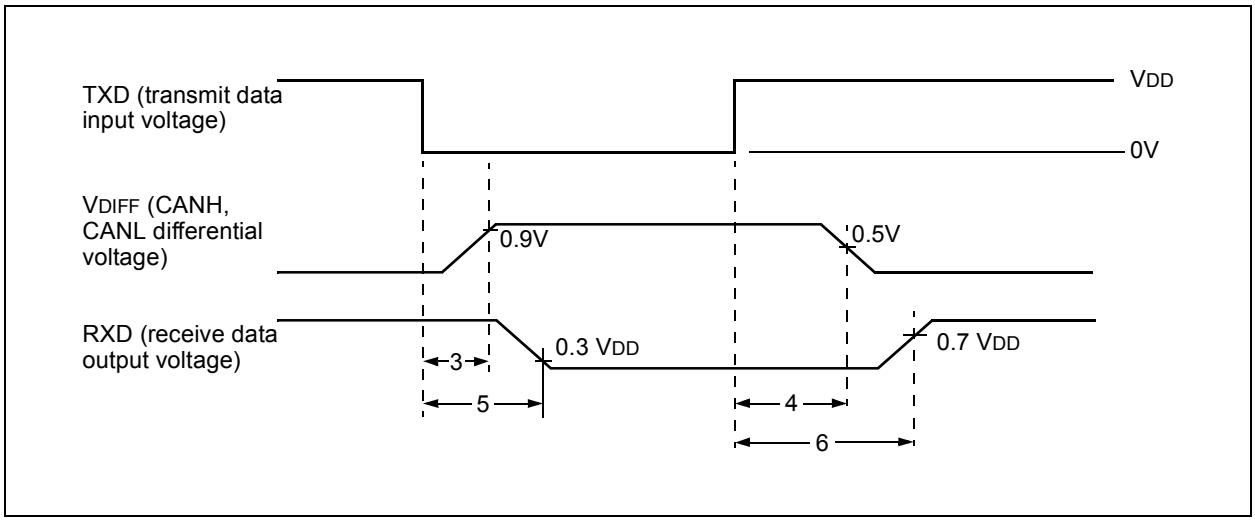


FIGURE 2-5: TIMING DIAGRAM FOR WAKE-UP FROM STANDBY

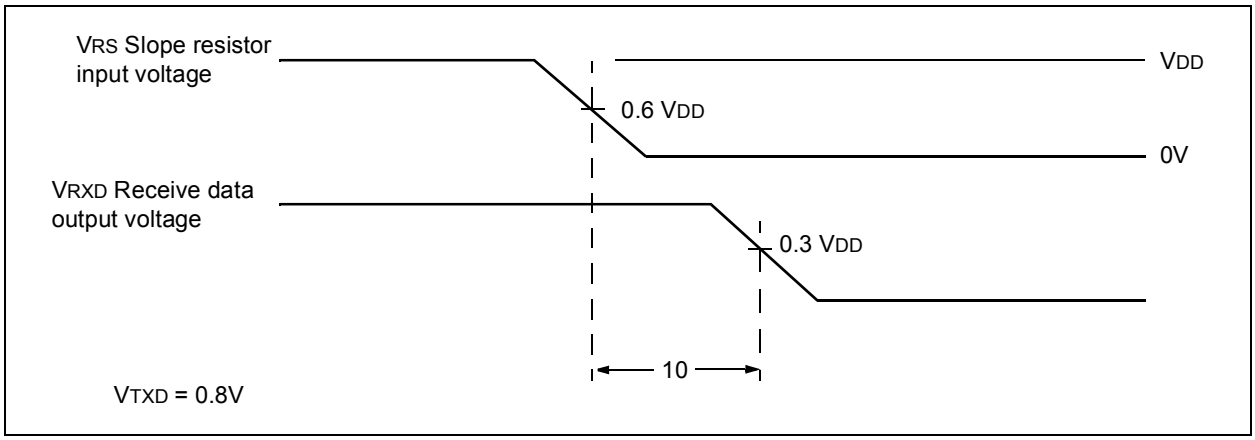
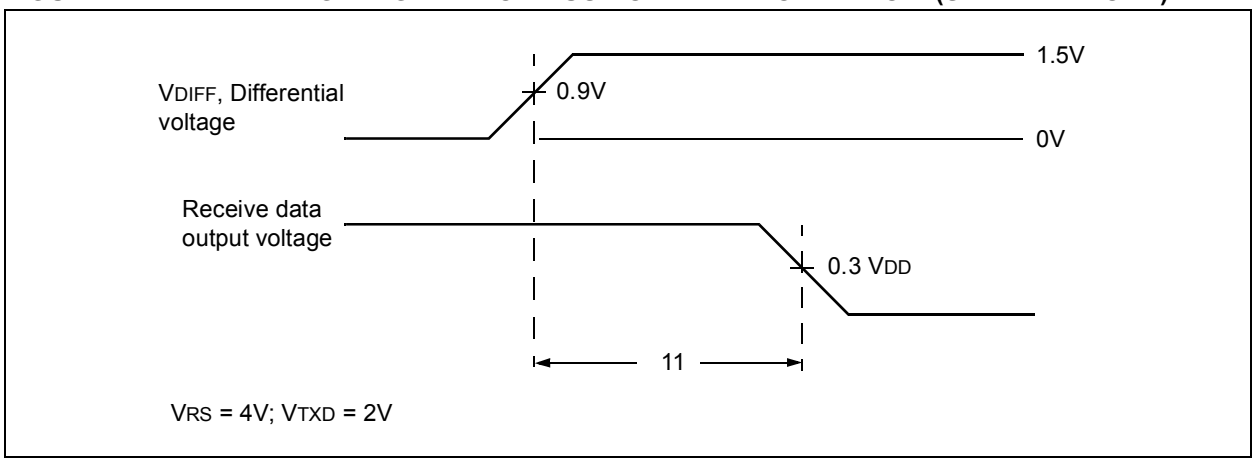


FIGURE 2-2: TIMING DIAGRAM FOR BUS DOMINANT TO RXD LOW (STANDBY MODE)

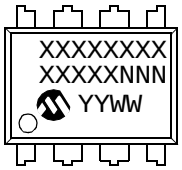


NOTES:

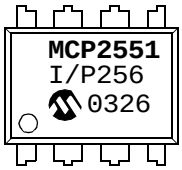
3.0 PACKAGING INFORMATION

3.1 Package Marking Information

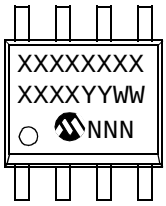
8-Lead PDIP (300 mil)



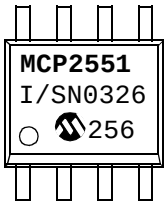
Example:



8-Lead SOIC (150 mil)



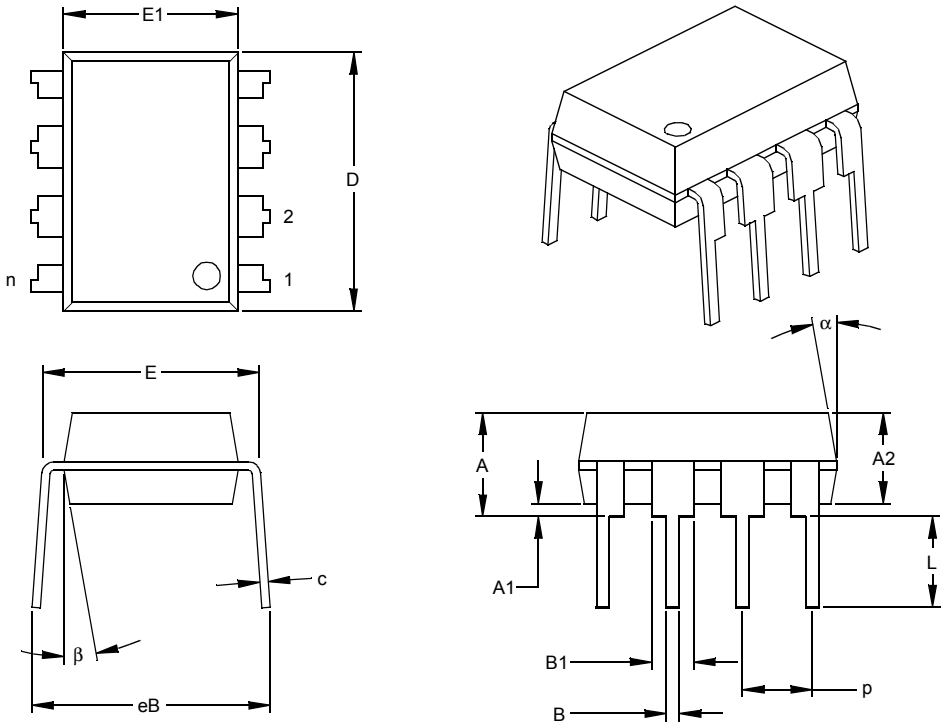
Example:



Legend:	XX...X	Customer specific information*
	YY	Year code (last 2 digits of calendar year)
	WW	Week code (week of January 1 is week ‘01’)
	NNN	Alphanumeric traceability code
Note:	In the event the full Microchip part number cannot be marked on one line, it will be carried over to the next line thus limiting the number of available characters for customer specific information.	

* Standard marking consists of Microchip part number, year code, week code, traceability code (facility code, mask rev#, and assembly code). For marking beyond this, certain price adders apply. Please check with your Microchip Sales Office.

8-Lead Plastic Dual In-line (P) – 300 mil (PDIP)

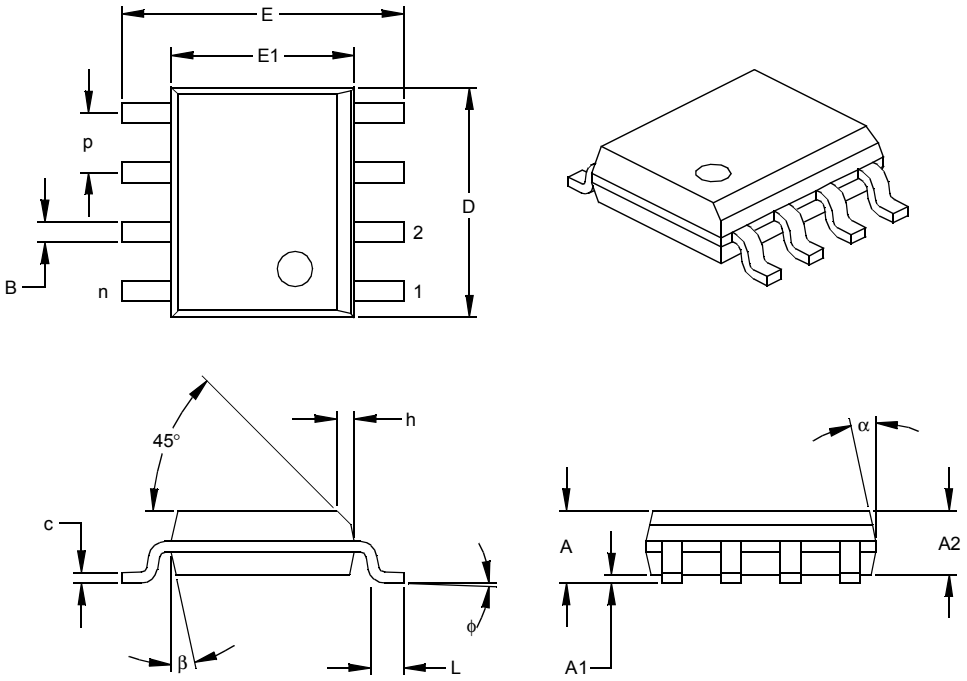


Units		INCHES*			MILLIMETERS		
Dimension Limits		MIN	NOM	MAX	MIN	NOM	MAX
Number of Pins	n		8			8	
Pitch	p		.100			2.54	
Top to Seating Plane	A	.140	.155	.170	3.56	3.94	4.32
Molded Package Thickness	A2	.115	.130	.145	2.92	3.30	3.68
Base to Seating Plane	A1	.015			0.38		
Shoulder to Shoulder Width	E	.300	.313	.325	7.62	7.94	8.26
Molded Package Width	E1	.240	.250	.260	6.10	6.35	6.60
Overall Length	D	.360	.373	.385	9.14	9.46	9.78
Tip to Seating Plane	L	.125	.130	.135	3.18	3.30	3.43
Lead Thickness	c	.008	.012	.015	0.20	0.29	0.38
Upper Lead Width	B1	.045	.058	.070	1.14	1.46	1.78
Lower Lead Width	B	.014	.018	.022	0.36	0.46	0.56
Overall Row Spacing §	eB	.310	.370	.430	7.87	9.40	10.92
Mold Draft Angle Top	α	5	10	15	5	10	15
Mold Draft Angle Bottom	β	5	10	15	5	10	15

* Controlling Parameter
§ Significant Characteristic

Notes:
Dimensions D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed .010” (0.254mm) per side.
JEDEC Equivalent: MS-001
Drawing No. C04-018

8-Lead Plastic Small Outline (SN) – Narrow, 150 mil (SOIC)



Units		INCHES*			MILLIMETERS		
Dimension Limits		MIN	NOM	MAX	MIN	NOM	MAX
Number of Pins	n		8			8	
Pitch	p		.050			1.27	
Overall Height	A	.053	.061	.069	1.35	1.55	1.75
Molded Package Thickness	A2	.052	.056	.061	1.32	1.42	1.55
Standoff §	A1	.004	.007	.010	0.10	0.18	0.25
Overall Width	E	.228	.237	.244	5.79	6.02	6.20
Molded Package Width	E1	.146	.154	.157	3.71	3.91	3.99
Overall Length	D	.189	.193	.197	4.80	4.90	5.00
Chamfer Distance	h	.010	.015	.020	0.25	0.38	0.51
Foot Length	L	.019	.025	.030	0.48	0.62	0.76
Foot Angle	φ	0	4	8	0	4	8
Lead Thickness	c	.008	.009	.010	0.20	0.23	0.25
Lead Width	B	.013	.017	.020	0.33	0.42	0.51
Mold Draft Angle Top	α	0	12	15	0	12	15
Mold Draft Angle Bottom	β	0	12	15	0	12	15

* Controlling Parameter
§ Significant Characteristic

Notes:
Dimensions D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed .010” (0.254mm) per side.
JEDEC Equivalent: MS-012
Drawing No. C04-057

NOTES:

PRODUCT IDENTIFICATION SYSTEM

To order or obtain information, e.g., on pricing or delivery, refer to the factory or the listed sales office.

PART NO.	X	/XX
Device	Temperature Range	Package

Device:	MCP2551	=	High-Speed CAN Transceiver
Temperature Range:	I	=	-40°C to +85°C
	E	=	-40°C to +125°C
Package:	P	=	Plastic DIP (300 mil Body) 8-lead
	SN	=	Plastic SOIC (150 mil Body) 8-lead

Examples:

a)	MCP2551-I/P:	Industrial temperature, PDIP package.
b)	MCP2551-E/P:	Extended temperature, PDIP package.
c)	MCP2551-I/SN:	Industrial temperature, SOIC package.
d)	MCP2551T-I/SN:	Tape and Reel, Industrial Temperature, SOIC package.
e)	MCP2551T-E/SN:	Tape and Reel, Extended Temperature, SOIC package.

Sales and Support

Data Sheets
Products supported by a preliminary Data Sheet may have an errata sheet describing minor operational differences and recommended workarounds. To determine if an errata sheet exists for a particular device, please contact one of the following:

1. Your local Microchip sales office
2. The Microchip Corporate Literature Center U.S. FAX: (480) 792-7277
3. The Microchip Worldwide Site (www.microchip.com)

Please specify which device, revision of silicon and Data Sheet (include Literature #) you are using.

Customer Notification System
Register on our web site (www.microchip.com/cn) to receive the most current information on our products.

NOTES:

19-2672; Rev 1; 10/03

+1.2V to +5.5V, ±15kV ESD-Protected, 0.1µA, 35Mbps, 8-Channel Level Translators

General Description

The MAX3000E/MAX3001E/MAX3002–MAX3012 8-channel level translators provide the level shifting necessary to allow data transfer in a multivoltage system. Externally applied voltages, V_{CC} and V_L, set the logic levels on either side of the device. Logic signals present on the V_L side of the device appear as a higher voltage logic signal on the V_{CC} side of the device, and vice-versa.

The MAX3000E/MAX3001E/MAX3002/MAX3004–MAX3012 feature an EN input that, when low, reduces the V_{CC} and V_L supply currents to <2µA. The MAX3000E/MAX3001E also have ±15kV ESD protection on the I/O V_{CC} side for greater protection in applications that route signals externally. The MAX3000E operates at a guaranteed data rate of 230kbps. The MAX3001E operates at a guaranteed data rate of 4Mbps. The MAX3002–MAX3012 operate at a guaranteed data rate of 20Mbps over the entire specified operating voltage range.

The MAX3000E/MAX3001E/MAX3002–MAX3012 accept V_L voltages from +1.2V to +5.5V and V_{CC} voltages from +1.65V to +5.5V, making them ideal for data transfer between low-voltage ASICs/PLDs and higher voltage systems. The MAX3000E/MAX3001E/MAX3002–MAX3012 are available in 20-pin UCSP™ and 20-pin TSSOP packages.

Applications

Cellphones

SPI™ and MICROWIRE™ Level Translation

Low-Voltage ASIC Level Translation

Smart Card Readers

Cellphone Cradles

Portable POS Systems

Portable Communication Devices

Low-Cost Serial Interfaces

GPS

Telecommunications Equipment

UCSP is a trademark of Maxim Integrated Products, Inc.

SPI is a trademark of Motorola, Inc.

MICROWIRE is a trademark of National Semiconductor.

Features

- **Guaranteed Data Rate Options**
  230kbps (MAX3000E)
  4Mbps (MAX3001E)
  20Mbps (MAX3002–MAX3012)
- **Bidirectional Level Translation (MAX3000E/MAX3001E/MAX3002/MAX3003)**
- **Unidirectional Level Translation (MAX3004–MAX3012)**
- **Operation Down to +1.2V on V_L**
- **±15kV ESD Protection on I/O V_{CC} Lines (MAX3000E/MAX3001E)**
- **Ultra-Low 0.1µA Supply Current in Shutdown**
- **Low Quiescent Current (<10µA)**
- **UCSP and TSSOP Packages**

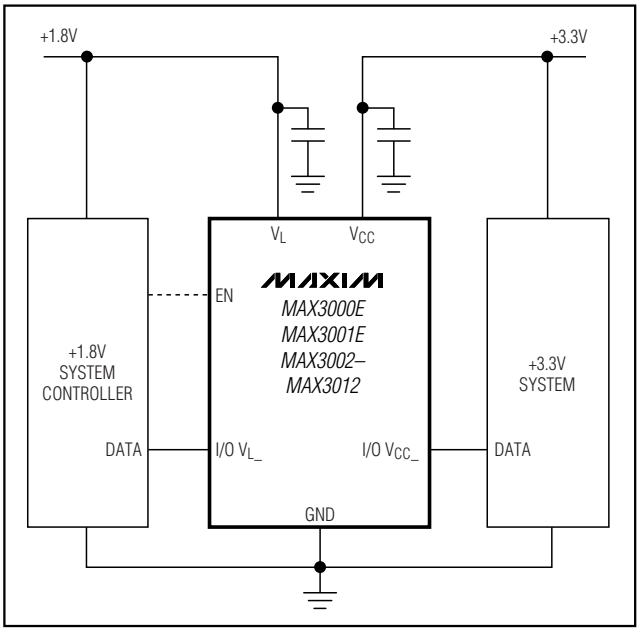
Ordering Information

PART	TEMP RANGE	PIN-PACKAGE
MAX3000EEUP	-40°C to +85°C	20 TSSOP
MAX3000EEBP-T*	-40°C to +85°C	4 x 5 UCSP

**Future product—contact factory for availability.*

Ordering Information continued at end of data sheet.

Typical Operating Circuit



Pin Configurations and Functional Diagrams appear at end of data sheet.

+1.2V to +5.5V, ±15kV ESD-Protected, 0.1µA, 35Mbps, 8-Channel Level Translators

ABSOLUTE MAXIMUM RATINGS

All Voltages Referenced to GND
V_{CC} .....-0.3V to +6V
V_L .....-0.3V to +6V
I/O $V_{CC\_}$ .....-0.3V to (V_{CC} + 0.3V)
I/O $V_{L\_}$ .....-0.3V to (V_L + 0.3V)
EN, EN A/B .....-0.3V to +6V
Short-Circuit Duration I/O $V_{L\_}$, I/O $V_{CC\_}$ to GND .....Continuous
Continuous Power Dissipation (T_A = +70°C)
20-Pin TSSOP (derate 7.0mW/°C above +70°C) 559mW
20-Pin UCSP (derate 10mW/°C above +70°C) 800mW

Operating Temperature Ranges
MAX3001EAUP .....-40°C to +125°C
MAX300_EE_P .....-40°C to +85°C
MAX30_ _E_P .....-40°C to +85°C
Junction Temperature .....+150°C
Storage Temperature Range .....-65°C to +150°C
Lead Temperature (soldering, 10s) .....+300°C

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of the specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

ELECTRICAL CHARACTERISTICS

(V_{CC} = +1.65V to +5.5V, V_L = +1.2V to V_{CC}, EN = V_L (MAX3000E/MAX3001E/MAX3002/MAX3004–MAX3012), EN A/B = V_L or 0 (MAX3003), T_A = T_{MIN} to T_{MAX}. Typical values are at V_{CC} = +1.65V, V_L = +1.2V, and T_A = +25°C.) (Notes 1, 2)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
POWER SUPPLIES						
V_L Supply Range	V_L		1.2		5.5	V
V_{CC} Supply Range	V_{CC}		1.65		5.50	V
Supply Current from V_{CC}	I_{QVCC}	I/O $V_{CC\_}$ = 0, I/O $V_{L\_}$ = 0 or I/O $V_{CC\_}$ = V_{CC}, I/O $V_{L\_}$ = V_L, MAX3000E/MAX3002–MAX3012		0.1	10	µA
		I/O $V_{CC\_}$ = 0, I/O $V_{L\_}$ = 0 or I/O $V_{CC\_}$ = V_{CC}, I/O $V_{L\_}$ = V_L, MAX3001E		0.1	50	
Supply Current from V_L	I_{QVL}	I/O $V_{CC\_}$ = 0, I/O $V_{L\_}$ = 0 or I/O $V_{CC\_}$ = V_{CC}, I/O $V_{L\_}$ = V_L, MAX3000E/MAX3002–MAX3012		0.1	10	µA
		I/O $V_{CC\_}$ = 0, I/O $V_{L\_}$ = 0 or I/O $V_{CC\_}$ = V_{CC}, I/O $V_{L\_}$ = V_L, MAX3001E		0.1	50	
V_{CC} Shutdown Supply Current	$I_{SHDN-VCC}$	T_A = +25°C, EN = 0, MAX3000E/MAX3001E/MAX3002/ MAX3004–MAX3012		0.1	2	µA
		T_A = +25°C, EN A/B = 0, MAX3003		0.1	2	
V_L Shutdown Supply Current	$I_{SHDN-VL}$	T_A = +25°C, EN = 0, MAX3000E/MAX3001E/MAX3002/ MAX3004–MAX3012		0.1	2	µA
		T_A = +25°C, EN A/B = 0, MAX3003		0.1	2	

ELECTRICAL CHARACTERISTICS (continued)

(V_{CC} = +1.65V to +5.5V, V_L = +1.2V to V_{CC}, EN = V_L (MAX3000E/MAX3001E/MAX3002/MAX3004–MAX3012), EN A/B = V_L or 0 (MAX3003), T_A = T_{MIN} to T_{MAX}. Typical values are at V_{CC} = +1.65V, V_L = +1.2V, and T_A = +25°C.) (Notes 1, 2)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
I/O $V_{CC\_}$ Three-State Output Leakage Current		T_A = +25°C, EN = 0, MAX3000E/MAX3001E/MAX3002/ MAX3004–MAX3012		0.1	2	µA
		T_A = +25°C, EN A/B = 0, MAX3003		0.1	2	
I/O $V_{L\_}$ Three-State Output Leakage Current		EN A/B = 0, MAX3003		0.1	2	µA
I/O $V_{L\_}$ Pulldown Resistance During Shutdown		EN = 0, MAX3000E/MAX3001E/MAX3002/ MAX3004–MAX3012	4.59		8.30	kΩ
EN or EN A/B Input Leakage Current		T_A = +25°C			1	µA
LOGIC-LEVEL THRESHOLDS						
I/O $V_{L\_}$ Input Voltage High Threshold	V_{IHL}				V_L - 0.4	V
I/O $V_{L\_}$ Input Voltage Low Threshold	V_{ILL}		0.4			V
I/O $V_{CC\_}$ Input Voltage High Threshold	V_{IHC}				V_{CC} - 0.4	V
I/O $V_{CC\_}$ Input Voltage Low Threshold	V_{ILC}		0.4			V
EN, EN A/B Input Voltage High Threshold	V_{IH}				V_L - 0.4	V
EN, EN A/B Input Voltage Low Threshold	V_{IL}		0.4			V
I/O $V_{L\_}$ Output Voltage High	V_{OHL}	I/O $V_{L\_}$ source current = 20µA, I/O $V_{CC\_}$ ≥ V_{CC} - 0.4V	V_L - 0.4			V
I/O $V_{L\_}$ Output Voltage Low	V_{OLL}	I/O $V_{L\_}$ sink current = 20µA, I/O $V_{CC\_}$ ≤ 0.4V			0.4	V
I/O $V_{CC\_}$ Output Voltage High	V_{OHC}	I/O $V_{CC\_}$ source current = 20µA, I/O $V_{L\_}$ ≥ V_L - 0.4V	V_{CC} - 0.4			V
I/O $V_{CC\_}$ Output Voltage Low	V_{OLC}	I/O V_{CC} sink current = 20µA, I/O $V_{L\_}$ ≤ 0.4V			0.4	V
ESD PROTECTION						
I/O $V_{CC\_}$		Human Body Model, MAX3000E/MAX3001E		±15		kV

+1.2V to +5.5V, ±15kV ESD-Protected, 0.1µA, 35Mbps, 8-Channel Level Translators

TIMING CHARACTERISTICS

(V_{CC} = +1.65V to +5.5V, V_L = +1.2V to V_{CC}, EN = V_L (MAX3000E/MAX3001E/MAX3002/MAX3004–MAX3012), EN A/B = V_L or 0 (MAX3003), T_A = T_{MIN} to T_{MAX}. Typical values are at V_{CC} = +1.65V, V_L = +1.2V, and T_A = +25°C.) (Notes 1, 2)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
I/O $V_{CC\_}$ Rise Time	t_{RVCC}	R_S = 50Ω, C_{VCC} = 50pF, MAX3000E, Figures 1a, 1b	400	800	1200	ns
		R_S = 50Ω, C_{VCC} = 50pF, MAX3001E, Figures 1a, 1b		25	50	
		R_S = 50Ω, C_{VCC} = 50pF, MAX3002–MAX3012, Figures 1a, 1b			15	
I/O $V_{CC\_}$ Fall Time	t_{FVCC}	R_S = 50Ω, C_{VCC} = 50pF, MAX3000E, Figures 1a, 1b	400	800	1200	ns
		R_S = 50Ω, C_{VCC} = 50pF, MAX3001E, Figures 1a, 1b		25	50	
		R_S = 50Ω, C_{VCC} = 50pF, MAX3002–MAX3012, Figures 1a, 1b			15	
I/O $V_{L\_}$ Rise Time	t_{RVL}	R_S = 50Ω, C_{VL} = 50pF, MAX3000E, Figures 2a, 2b	400	800	1200	ns
		R_S = 50Ω, C_{VL} = 50pF, MAX3001E, Figures 2a, 2b		25	50	
		R_S = 50Ω, C_{VL} = 15pF, MAX3002–MAX3012, Figures 2a, 2b			15	
I/O $V_{L\_}$ Fall Time	t_{FVL}	R_S = 50Ω, C_{VL} = 50pF, MAX3000E, Figures 2a, 2b	400	800	1200	ns
		R_S = 50Ω, C_{VL} = 50pF, MAX3001E, Figures 2a, 2b		25	65	
		R_S = 50Ω, C_{VL} = 15pF, MAX3002–MAX3012, Figures 2a, 2b			15	
Propagation Delay (Driving I/O $V_{L\_}$)	I/O_{VL-VCC}	R_S = 50Ω, C_{VCC} = 50pF, MAX3000E, Figures 1a, 1b			1000	ns
		R_S = 50Ω, C_{VCC} = 50pF, MAX3001E, Figures 1a, 1b			50	
		R_S = 50Ω, C_{VCC} = 50pF, MAX3002–MAX3012, Figures 1a, 1b			20	
Propagation Delay (Driving I/O $V_{CC\_}$)	I/O_{VCC-VL}	R_S = 50Ω, C_{VL} = 50pF, MAX3000E, Figures 2a, 2b			1000	ns
		R_S = 50Ω, C_{VL} = 50pF, MAX3001E, Figures 2a, 2b			50	
		R_S = 50Ω, C_{VL} = 15pF, MAX3002–MAX3012, Figures 2a, 2b			20	

Note 1: All units are 100% production tested at T_A = +25°C. Limits over the operating temperature range are guaranteed by design and not production tested.

Note 2: For normal operation, ensure that V_L < (V_{CC} + 0.3V). During power-up, V_L > (V_{CC} + 0.3V) does not damage the device.

TIMING CHARACTERISTICS (continued)

(V_{CC} = +1.65V to +5.5V, V_L = +1.2V to V_{CC}, EN = V_L (MAX3000E/MAX3001E/MAX3002/MAX3004–MAX3012), EN A/B = V_L or 0 (MAX3003), T_A = T_{MIN} to T_{MAX}. Typical values are at V_{CC} = +1.65V, V_L = +1.2V, and T_A = +25°C.) (Notes 1, 2)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
Channel-to-Channel Skew	t_{SKEW}	R_S = 50Ω, C_{VCC} = 50pF, C_{VL} = 50pF, MAX3000E			500	ns
		R_S = 50Ω, C_{VCC} = 50pF, C_{VL} = 50pF, MAX3001E			10	
		R_S = 50Ω, C_{VCC} = 50pF, C_{VL} = 15pF, MAX3002–MAX3012			5	
Part-to-Part Skew	t_{PPSKEW}	R_S = 50Ω, C_{VCC} = 50pF, C_{VL} = 50pF, ΔT_A = +20°C, MAX3000E (Note 3)			800	ns
		R_S = 50Ω, C_{VCC} = 50pF, C_{VL} = 50pF, ΔT_A = +20°C, MAX3001E (Note 3)			30	
		R_S = 50Ω, C_{VCC} = 50pF, C_{VL} = 15pF, ΔT_A = +20°C, MAX3002–MAX3012 (Note 3)			10	
Propagation Delay from I/O $V_{L\_}$ to I/O $V_{CC\_}$ after EN	t_{EN-VCC}	C_{VCC} = 50pF, MAX3000E/MAX3001E, MAX3002–MAX3012, Figure 3				µs
Propagation Delay from I/O $V_{CC\_}$ to I/O $V_{L\_}$ after EN	t_{EN-VL}	C_{VL} = 50pF, MAX3000E/MAX3001E/ MAX3002/MAX3004–MAX3012, Figure 4			2	µs
		C_{VL} = 15pF, MAX3003, Figure 4			2	
Maximum Data Rate		R_S = 50Ω, C_{VCC} = 50pF, C_{VL} = 50pF, MAX3000E	230			kbps
		R_S = 50Ω, C_{VCC} = 50pF, C_{VL} = 50pF, MAX3001E	4			Mbps
		R_S = 50Ω, C_{VCC} = 50pF, C_{VL} = 15pF, MAX3002–MAX3012	20			

Note 3: V_{CC} from device 1 must equal V_{CC} of device 2; V_L from device 1 must equal V_L of device 2.

+1.2V to +5.5V, ±15kV ESD-Protected, 0.1µA, 35Mbps, 8-Channel Level Translators

TIMING CHARACTERISTICS—MAX3002–MAX3012

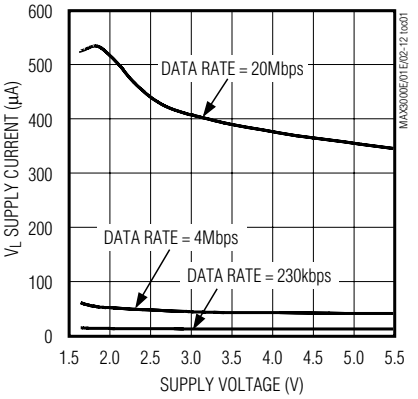
(V_{CC} = +1.65V to +5.5V, V_L = +1.2V to V_{CC}, EN = V_L (MAX3002/MAX3004–MAX3012), EN A/B = V_L or 0 (MAX3003), T_A = T_{MIN} to T_{MAX}.) (Notes 1, 2)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
+1.2V ≤ V_L ≤ V_{CC} ≤ +3.3V						
I/O $V_{CC\_}$ Rise Time	t_{RVCC}				15	ns
I/O $V_{CC\_}$ Fall Time	t_{FVCC}				15	ns
I/O $V_{L\_}$ Rise Time	t_{RVL}				15	ns
I/O $V_{L\_}$ Fall Time	t_{FVL}				15	ns
Propagation Delay	I/O_{VL-VCC}	Driving I/O $V_{L\_}$			15	ns
	I/O_{VCC-VL}	Driving I/O $V_{CC\_}$			15	
Channel-to-Channel Skew	t_{SKEW}	Each translator equally loaded			5	ns
Maximum Data Rate			20			Mbps
+2.5V ≤ V_L ≤ V_{CC} ≤ +3.3V						
I/O $V_{CC\_}$ Rise Time	t_{RVCC}				8.5	ns
I/O $V_{CC\_}$ Fall Time	t_{FVCC}				8.5	ns
I/O $V_{L\_}$ Rise Time	t_{RVL}				8.5	ns
I/O $V_{L\_}$ Fall Time	t_{FVL}				8.5	ns
Propagation Delay	I/O_{VL-VCC}	Driving I/O $V_{L\_}$			8.5	ns
	I/O_{VCC-VL}	Driving I/O $V_{CC\_}$			8.5	
Channel-to-Channel Skew	t_{SKEW}	Each translator equally loaded			10	ns
Maximum Data Rate			35			Mbps
+1.8V ≤ V_L ≤ V_{CC} ≤ +2.5V						
I/O $V_{CC\_}$ Rise Time	t_{RVCC}				10	ns
I/O $V_{CC\_}$ Fall Time	t_{FVCC}				10	ns
I/O $V_{L\_}$ Rise Time	t_{RVL}				10	ns
I/O $V_{L\_}$ Fall Time	t_{FVL}				10	ns
Propagation Delay	I/O_{VL-VCC}	Driving I/O $V_{L\_}$			15	ns
	I/O_{VCC-VL}	Driving I/O $V_{CC\_}$			10	
Channel-to-Channel Skew	t_{SKEW}	Each translator equally loaded			5	ns
Maximum Data Rate			30			Mbps

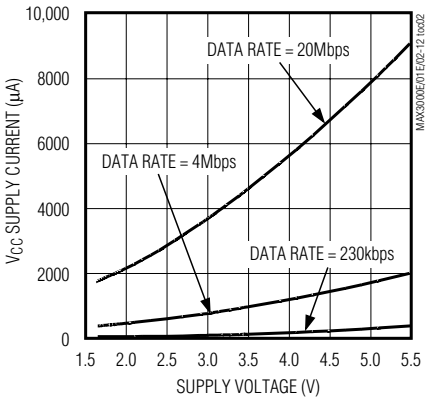
Typical Operating Characteristics

(T_A = +25°C, unless otherwise noted.)

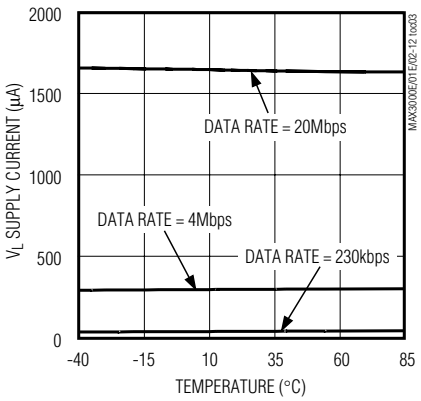
V_L SUPPLY CURRENT vs. SUPPLY VOLTAGE (DRIVING I/O V_L, V_L = 1.8V)



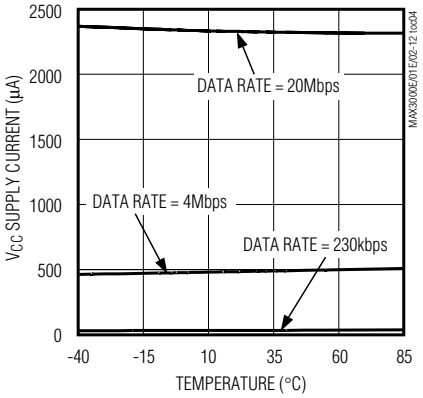
V_{CC} SUPPLY CURRENT vs. SUPPLY VOLTAGE (DRIVING I/O V_L, V_L = 1.8V)



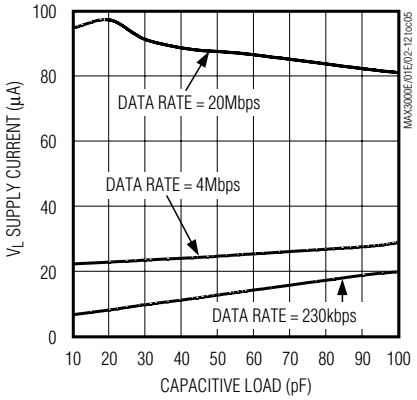
V_L SUPPLY CURRENT vs. TEMPERATURE (DRIVING I/O V_{CC}, V_{CC} = 3.3V, V_L = 1.8V)



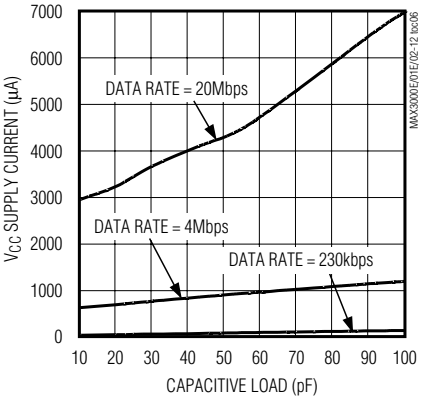
V_{CC} SUPPLY CURRENT vs. TEMPERATURE (DRIVING I/O V_{CC}, V_{CC} = 3.3V, V_L = 1.8V)



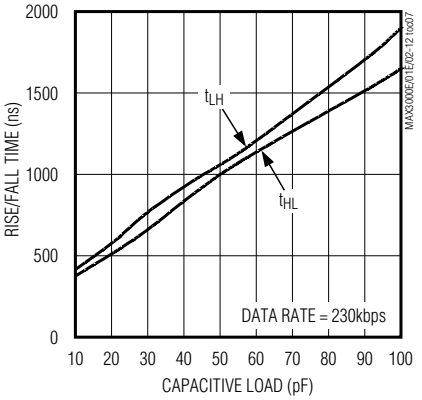
V_L SUPPLY CURRENT vs. CAPACITIVE LOAD ON I/O V_{CC} (DRIVING I/O V_L, V_{CC} = 3.3V, V_L = 1.8V)



V_{CC} SUPPLY CURRENT vs. CAPACITIVE LOAD ON I/O V_{CC} (DRIVING I/O V_L, V_{CC} = 3.3V, V_L = 1.8V)



MAX3000E RISE/FALL TIME vs. CAPACITIVE LOAD ON I/O V_{CC} (DRIVING I/O V_L, V_{CC} = 3.3V, V_L = 1.8V)

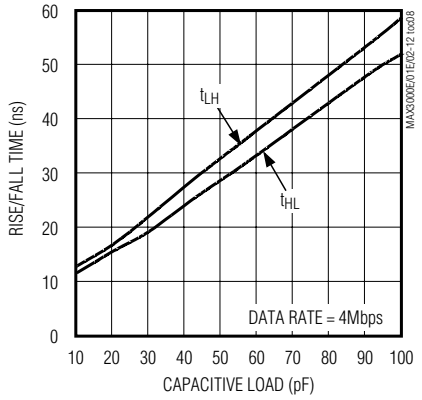


+1.2V to +5.5V, ±15kV ESD-Protected, 0.1µA, 35Mbps, 8-Channel Level Translators

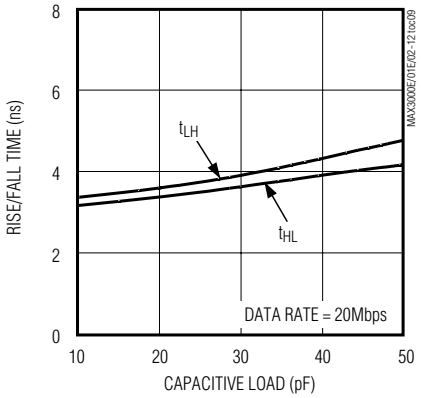
Typical Operating Characteristics (continued)

(T_A = +25°C, unless otherwise noted.)

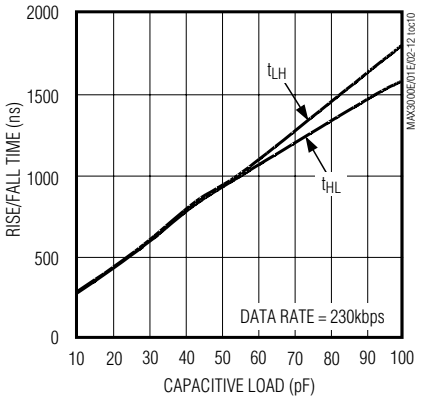
MAX3001E
RISE/FALL TIME vs. CAPACITIVE LOAD ON I/O V_{CC} (DRIVING I/O V_L, V_{CC} = 3.3V, V_L = 1.8V)



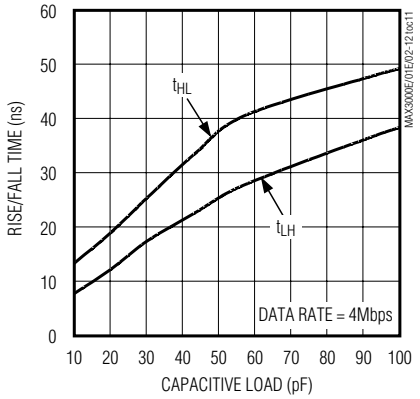
MAX3002–MAX3012
RISE/FALL TIME vs. CAPACITIVE LOAD ON I/O V_{CC} (DRIVING I/O V_L, V_{CC} = 3.3V, V_L = 1.8V)



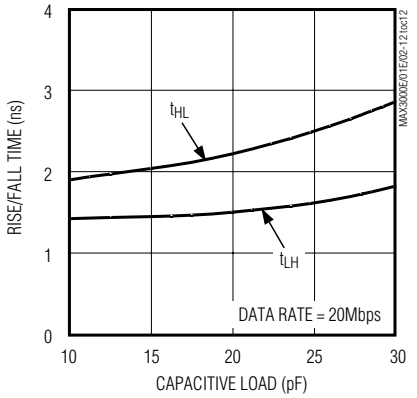
MAX3000E
RISE/FALL TIME vs. CAPACITIVE LOAD ON I/O V_L (DRIVING I/O V_{CC}, V_{CC} = 3.3V, V_L = 1.8V)



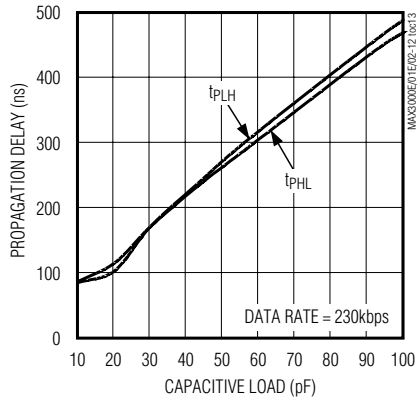
MAX3001E
RISE/FALL TIME vs. CAPACITIVE LOAD ON I/O V_L (DRIVING I/O V_{CC}, V_{CC} = 3.3V, V_L = 1.8V)



MAX3002–MAX3012
RISE/FALL TIME vs. CAPACITIVE LOAD ON I/O V_L (DRIVING I/O V_{CC}, V_{CC} = 3.3V, V_L = 1.8V)



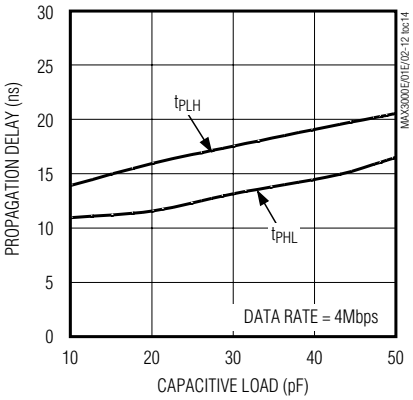
MAX3000E
PROPAGATION DELAY vs. CAPACITIVE LOAD ON I/O V_{CC} (DRIVING I/O V_L, V_{CC} = 3.3V, V_L = 1.8V)



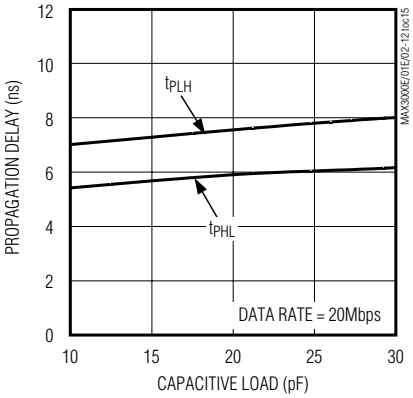
Typical Operating Characteristics (continued)

(T_A = +25°C, unless otherwise noted.)

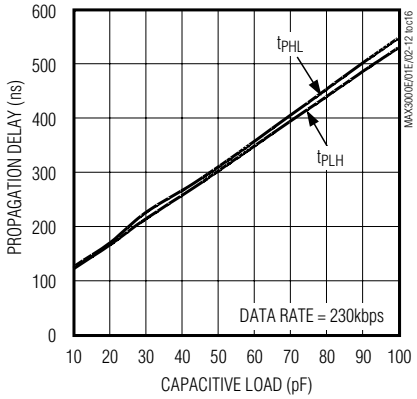
MAX3001E
PROPAGATION DELAY vs. CAPACITIVE LOAD ON I/O V_{CC} (DRIVING I/O V_L, V_{CC} = 3.3V, V_L = 1.8V)



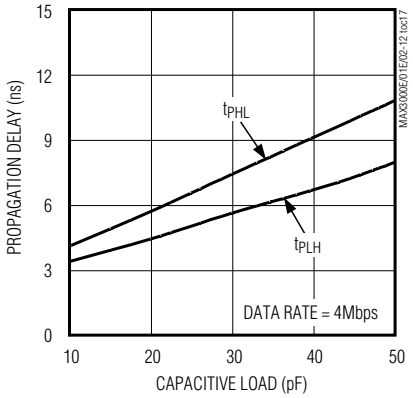
MAX3002–MAX3012
PROPAGATION DELAY vs. CAPACITIVE LOAD ON I/O V_{CC} (DRIVING I/O V_L, V_{CC} = 3.3V, V_L = 1.8V)



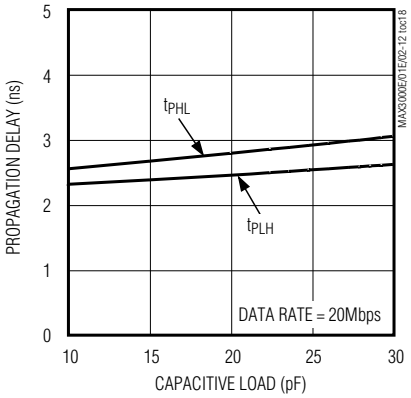
MAX3000E
PROPAGATION DELAY vs. CAPACITIVE LOAD ON I/O V_L (DRIVING I/O V_{CC}, V_{CC} = 3.3V, V_L = 1.8V)



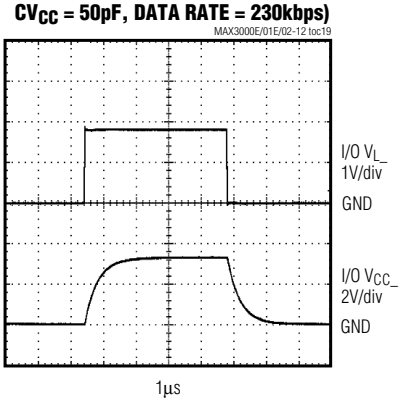
MAX3001E
PROPAGATION DELAY vs. CAPACITIVE LOAD ON I/O V_L (DRIVING I/O V_{CC}, V_{CC} = 3.3V, V_L = 1.8V)



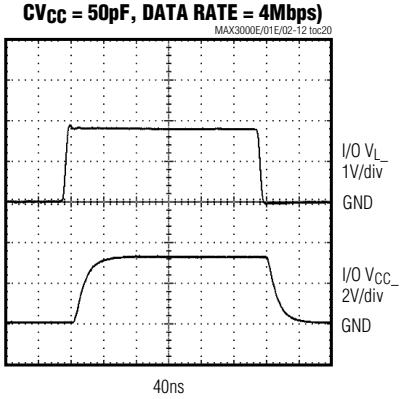
MAX3002–MAX3012
PROPAGATION DELAY vs. CAPACITIVE LOAD ON I/O V_L (DRIVING I/O V_{CC}, V_{CC} = 3.3V, V_L = 1.8V)



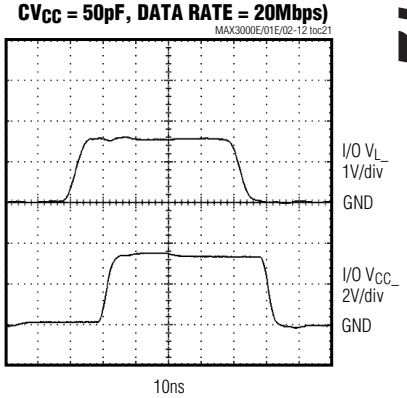
MAX3000E RAIL-TO-RAIL DRIVING (DRIVING I/O V_L, V_{CC} = 3.3V, V_L = 1.8V, CV_{CC} = 50pF, DATA RATE = 230kbps)



MAX3001E RAIL-TO-RAIL DRIVING (DRIVING I/O V_L, V_{CC} = 3.3V, V_L = 1.8V, CV_{CC} = 50pF, DATA RATE = 4Mbps)



MAX3002–MAX3012 RAIL-TO-RAIL DRIVING (DRIVING I/O V_L, V_{CC} = 3.3V, V_L = 1.8V, CV_{CC} = 50pF, DATA RATE = 20Mbps)



+1.2V to +5.5V, ±15kV ESD-Protected, 0.1µA, 35Mbps, 8-Channel Level Translators

Pin Description

MAX3000E/MAX3001E/MAX3002

PIN		NAME	FUNCTION
TSSOP	UCSP		
1	B1	I/O V_L1	Input/Output 1, Referenced to V_L
2	A1	V_L	Logic Input Voltage, $+1.2V \le V_L \le V_{CC}$. Bypass V_L to GND with a 0.1µF capacitor.
3	A2	I/O V_L2	Input/Output 2, Referenced to V_L
4	B2	I/O V_L3	Input/Output 3, Referenced to V_L
5	A3	I/O V_L4	Input/Output 4, Referenced to V_L
6	B3	I/O V_L5	Input/Output 5, Referenced to V_L
7	A4	I/O V_L6	Input/Output 6, Referenced to V_L
8	B4	I/O V_L7	Input/Output 7, Referenced to V_L
9	A5	I/O V_L8	Input/Output 8, Referenced to V_L
10	B5	EN	Enable Input. If EN is pulled low, I/O $V_{CC}1$ to I/O $V_{CC}8$ are in three-state, while I/O V_L1 to I/O V_L8 have internal 6kΩ pulldown resistors. Drive EN high (V_L) for normal operation.
11	C5	GND	Ground
12	D5	I/O $V_{CC}8$	Input/Output 8, Referenced to V_{CC}
13	C4	I/O $V_{CC}7$	Input/Output 7, Referenced to V_{CC}
14	D4	I/O $V_{CC}6$	Input/Output 6, Referenced to V_{CC}
15	C3	I/O $V_{CC}5$	Input/Output 5, Referenced to V_{CC}
16	D3	I/O $V_{CC}4$	Input/Output 4, Referenced to V_{CC}
17	C2	I/O $V_{CC}3$	Input/Output 3, Referenced to V_{CC}
18	D2	I/O $V_{CC}2$	Input/Output 2, Referenced to V_{CC}
19	D1	V_{CC}	V_{CC} Input Voltage, $+1.65V \le V_{CC} \le +5.5V$. Bypass V_{CC} to GND with a 0.1µF capacitor.
20	C1	I/O $V_{CC}1$	Input/Output 1, Referenced to V_{CC}

Pin Description (continued)

MAX3003

PIN		NAME	FUNCTION
TSSOP	UCSP		
1	B1	I/O V_L1A	Input/Output 1A, Referenced to V_L
2	A1	V_L	Logic Input Voltage, +1.2V ≤ V_L ≤ V_{CC}. Bypass V_L to GND with a 0.1µF capacitor.
3	A2	I/O V_L2A	Input/Output 2A, Referenced to V_L
4	B2	I/O V_L3A	Input/Output 3A, Referenced to V_L
5	A3	I/O V_L4A	Input/Output 4A, Referenced to V_L
6	B3	I/O V_L1B	Input/Output 1B, Referenced to V_L
7	A4	I/O V_L2B	Input/Output 2B, Referenced to V_L
8	B4	I/O V_L3B	Input/Output 3B, Referenced to V_L
9	A5	I/O V_L4B	Input/Output 4B, Referenced to V_L
10	B5	EN A/B	Enable Input. If EN A/B is pulled low, channels 1B through 4B are active, and channels 1A through 4A are in three-state. If EN A/B is driven high to V_L, channels 1A through 4A are active, and channels 1B through 4B are in three-state.
11	C5	GND	Ground
12	D5	I/O V_{CC}4B	Input/Output 4B, Referenced to V_{CC}
13	C4	I/O V_{CC}3B	Input/Output 3B, Referenced to V_{CC}
14	D4	I/O V_{CC}2B	Input/Output 2B, Referenced to V_{CC}
15	C3	I/O V_{CC}1B	Input/Output 1B, Referenced to V_{CC}
16	D3	I/O V_{CC}4A	Input/Output 4A, Referenced to V_{CC}
17	C2	I/O V_{CC}3A	Input/Output 3A, Referenced to V_{CC}
18	D2	I/O V_{CC}2A	Input/Output 2A, Referenced to V_{CC}
19	D1	V_{CC}	V_{CC} Input Voltage, +1.65V ≤ V_{CC} ≤ +5.5V. Bypass V_{CC} to GND with a 0.1µF capacitor.
20	C1	I/O V_{CC}1A	Input/Output 1A, Referenced to V_{CC}

+1.2V to +5.5V, ±15kV ESD-Protected, 0.1µA, 35Mbps, 8-Channel Level Translators

Pin Description (continued)

MAX3004–MAX3012

NAME	FUNCTION (Note 1)
V_{CC}	V_{CC} Input Voltage, +1.65V < V_{CC} < +5.5V. Bypass V_{CC} to GND with a 0.1µF capacitor.
V_L	Logic Input Voltage, +1.2V ≤ V_L ≤ V_{CC}. Bypass V_L to GND with a 0.1µF capacitor.
GND	Ground
EN (MAX3004)	Enable Input. If EN is pulled low, $OV_{CC}1$–$OV_{CC}8$ are in three-state, while IV_L1–IV_L8 have 6kΩ pulldown resistors. Drive EN high (V_L) for normal operation.
EN (MAX3005)	Enable Input. If EN is pulled low, $IV_{CC}1$ and $OV_{CC}2$–$OV_{CC}8$ are in three-state, while OV_L1 and IV_L2–IV_L8 have 6kΩ pulldown resistors. Drive EN high (V_L) for normal operation.
EN (MAX3006)	Enable Input. If EN is pulled low, $IV_{CC}1$, $IV_{CC}2$, and $OV_{CC}3$–$OV_{CC}8$ are in three-state, while OV_L1, OV_L2, and IV_L3–IV_L8 have 6kΩ pulldown resistors. Drive EN high (V_L) for normal operation.
EN (MAX3007)	Enable Input. If EN is pulled low, $IV_{CC}1$, $IV_{CC}2$, $IV_{CC}3$, and $OV_{CC}4$–$OV_{CC}8$ are in three-state, while OV_L1, OV_L2, OV_L3, and IV_L4–IV_L8 have 6kΩ pulldown resistors. Drive EN high (V_L) for normal operation.
EN (MAX3008)	Enable Input. If EN is pulled low, $IV_{CC}1$–$IV_{CC}4$ and $OV_{CC}5$–$OV_{CC}8$ are in three-state, while OV_L1–OV_L4 and IV_L5–IV_L8 have 6kΩ pulldown resistors. Drive EN high (V_L) for normal operation.
EN (MAX3009)	Enable Input. If EN is pulled low, $IV_{CC}1$–$IV_{CC}5$, $OV_{CC}6$, $OV_{CC}7$, and $OV_{CC}8$ are in three-state, while OV_L1–OV_L5, IV_L6, IV_L7, and IV_L8 have 6kΩ pulldown resistors. Drive EN high (V_L) for normal operation.
EN (MAX3010)	Enable Input. If EN is pulled low, $IV_{CC}1$–$IV_{CC}6$, $OV_{CC}7$, and $OV_{CC}8$ are in three-state, while OV_L1–OV_L6, IV_L7, and IV_L8 have 6kΩ pulldown resistors. Drive EN high (V_L) for normal operation.
EN (MAX3011)	Enable Input. If EN is pulled low, $IV_{CC}1$–$IV_{CC}7$ and $OV_{CC}8$ are in three-state, while OV_L1–OV_L7 and IV_L8 have 6kΩ pulldown resistors. Drive EN high (V_L) for normal operation.
EN (MAX3012)	Enable Input. If EN is pulled low, $IV_{CC}1$–$IV_{CC}8$ are in three-state, while OV_L1–OV_L8 have 6kΩ pulldown resistors. Drive EN high (V_L) for normal operation.
IV_L1–IV_L8	Inputs Referenced to V_L, Numbers 1 to 8
OV_L1–OV_L8	Outputs Referenced to V_L, Numbers 1 to 8
$IV_{CC}1$–$IV_{CC}8$	Inputs Referenced to V_{CC}, Numbers 1 to 8
$OV_{CC}1$–$OV_{CC}8$	Outputs Referenced to V_{CC}, Numbers 1 to 8

Note 1: For specific pin numbers, see the *Pin Configurations*.

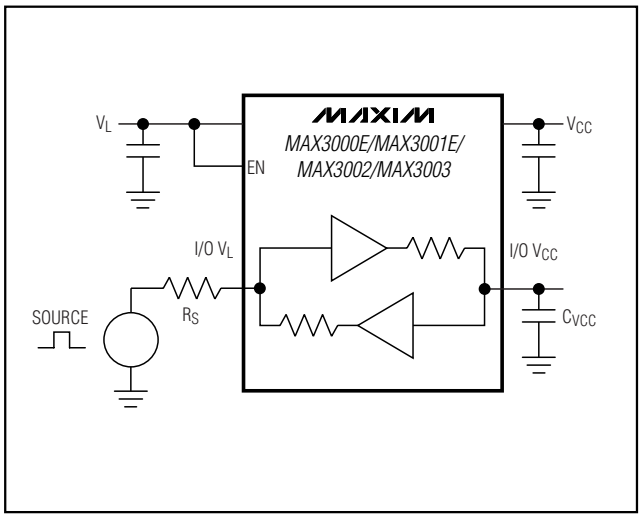


Figure 1a. Driving I/O V_L

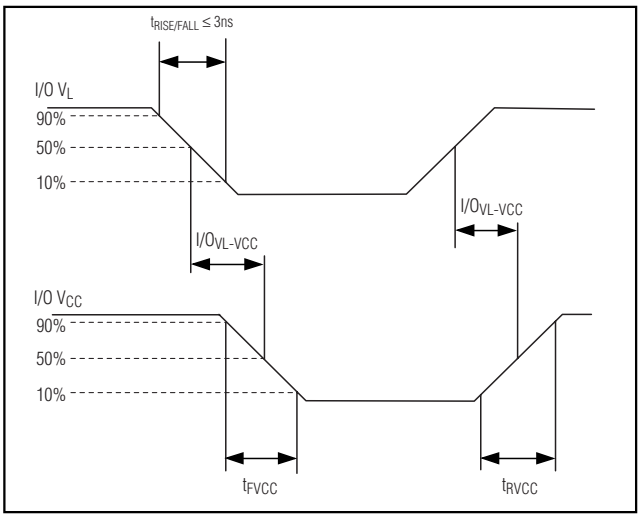


Figure 1b. Timing for Driving I/O V_L

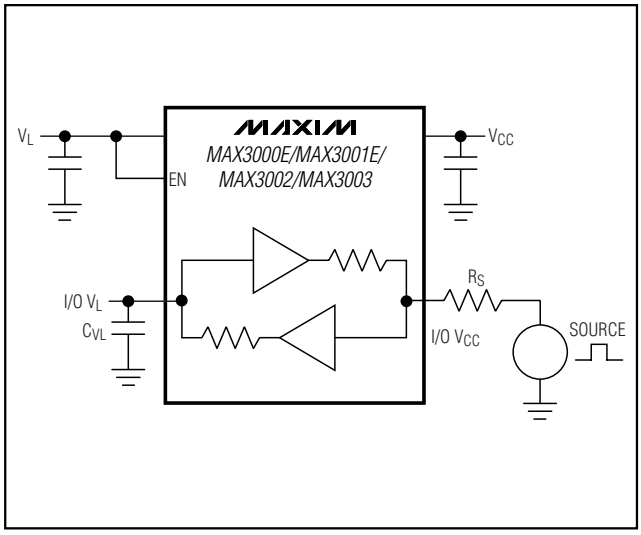


Figure 2a. Driving I/O V_{CC}

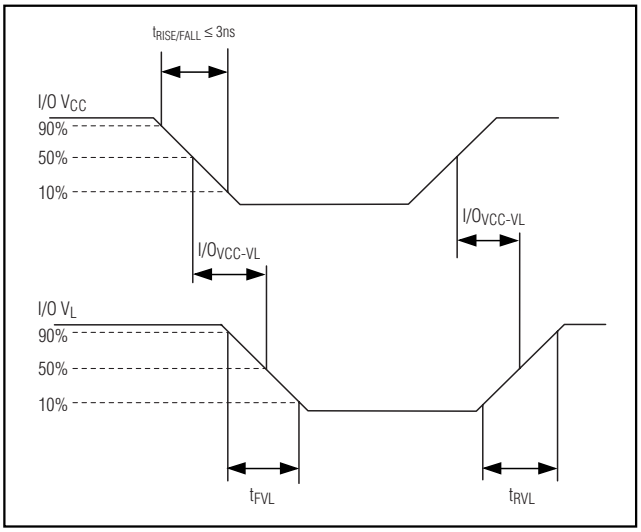


Figure 2b. Timing for Driving I/O V_{CC}

+1.2V to +5.5V, ±15kV ESD-Protected, 0.1µA, 35Mbps, 8-Channel Level Translators

Test Circuits/Timing Diagrams (continued)

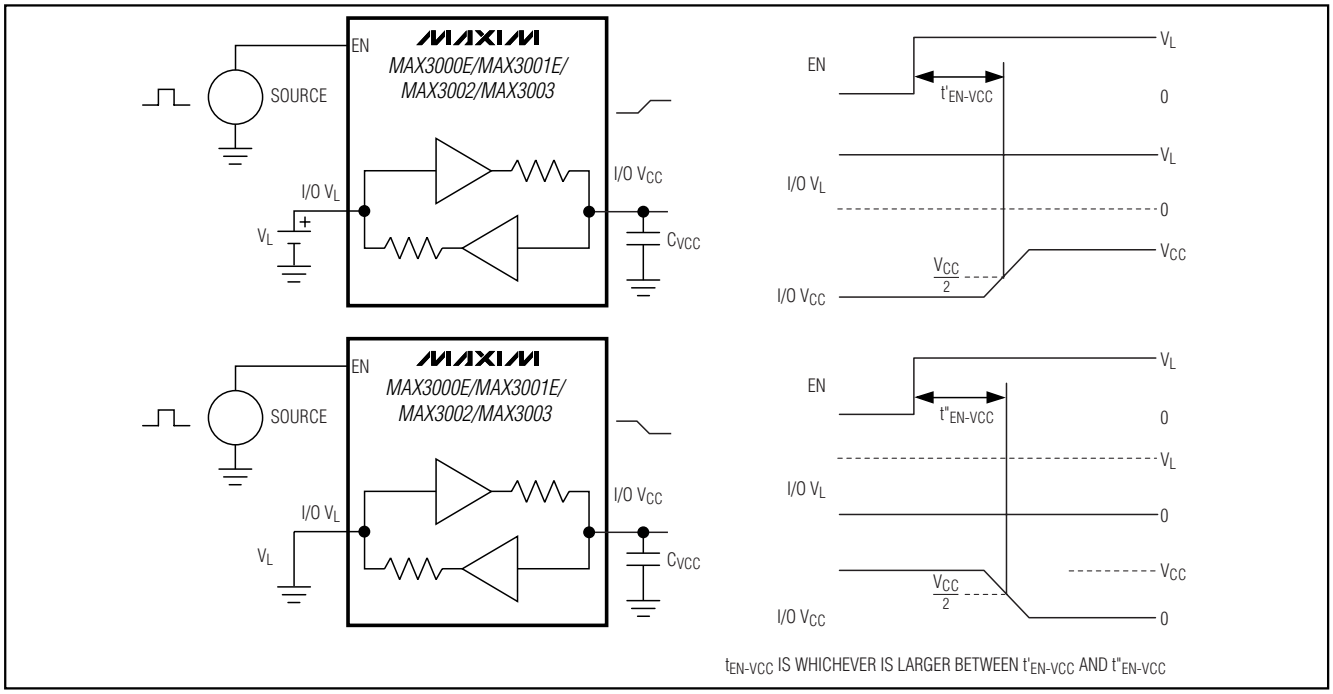


Figure 3. Propagation Delay from I/O V_L to I/O V_{CC} After EN

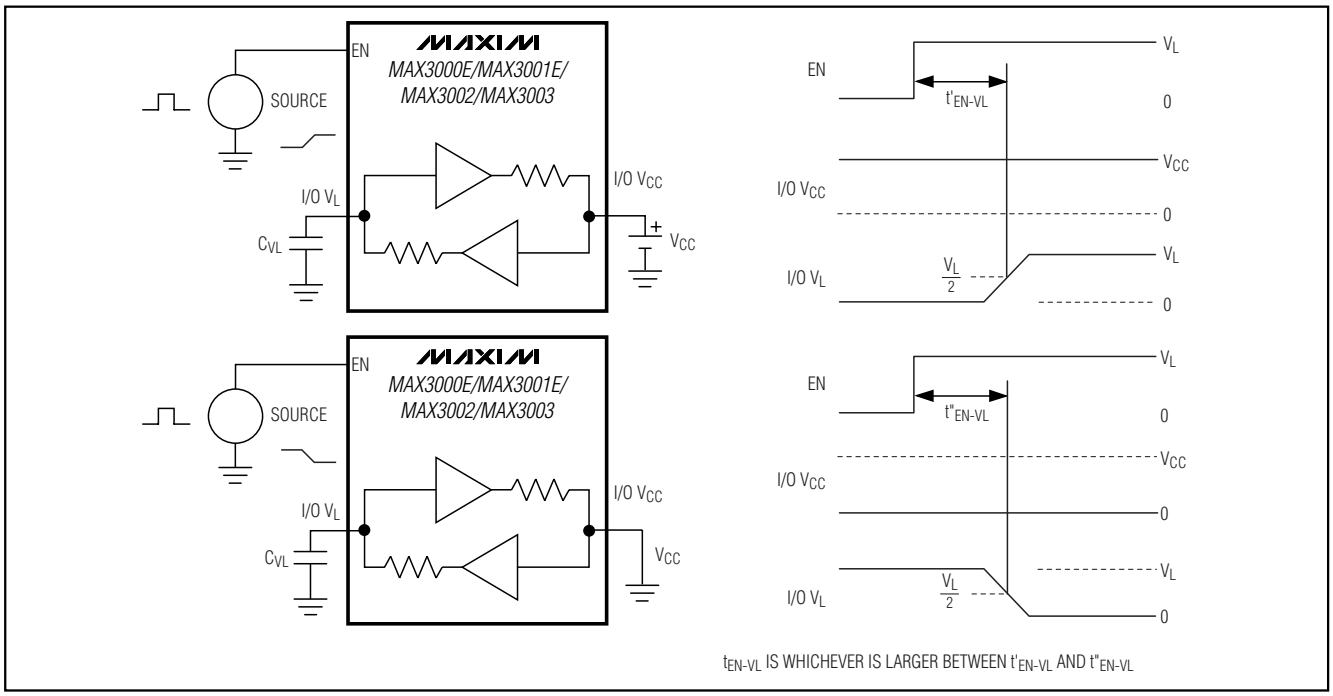


Figure 4. Propagation Delay from I/O V_{CC} to I/O V_L After EN

Detailed Description

The MAX3000E/MAX3001E/MAX3002–MAX3012 logic-level translators provide the level shifting necessary to allow data transfer in a multivoltage system. Externally applied voltages, V_{CC} and V_L, set the logic levels on either side of the device. Logic signals present on the V_L side of the device appear as a higher voltage logic signal on the V_{CC} side of the device, and vice-versa. The MAX3000E/MAX3001E/MAX3002/MAX3003 are bidirectional level translators allowing data translation in either direction ($V_L \leftrightarrow V_{CC}$) on any single data line. The MAX3004–MAX3012 unidirectional level translators level shift data in one direction ($V_L \rightarrow V_{CC}$ or $V_{CC} \rightarrow V_L$) on any single data line. The MAX3000E/MAX3001E/MAX3002–MAX3012 accept V_L from +1.2V to +5.5V. All devices have V_{CC} ranging from +1.65V to +5.5V, making them ideal for data transfer between low-voltage ASICs/PLDs and higher voltage systems.

The MAX3000E/MAX3001E/MAX3002/MAX3004–MAX3012 feature an output enable mode that reduces V_{CC} supply current to less than 2µA, and V_L supply current to less than 2µA when in shutdown. The MAX3000E/MAX3001E have ±15kV ESD protection on the V_{CC} side for greater protection in applications that route signals externally. The MAX3000E operates at a guaranteed data rate of 230kbps; the MAX3001E operates at a guaranteed data rate of 4Mbps and the MAX3002–MAX3012 are guaranteed with a data rate of 20Mbps of operation over the entire specified operating voltage range.

Level Translation

For proper operation, ensure that $+1.65V \leq V_{CC} \leq +5.5V$, $+1.2V \leq V_L \leq +5.5V$, and $V_L \leq V_{CC}$. During power-up sequencing, $V_L \geq V_{CC}$ does not damage the device. During power-supply sequencing, when V_{CC} is floating and V_L is powering up, up to 10mA current can be sourced to each load on the V_L side, yet the device does not latch up.

The maximum data rate also depends heavily on the load capacitance (see the *Typical Operating Characteristics*), output impedance of the driver, and the operational voltage range (see the *Timing Characteristics*).

Input Driver Requirements

The MAX3001E/MAX3002–MAX3012 architecture is based on a one-shot accelerator output stage. See Figure 5. Accelerator output stages are always in three-state except when there is a transition on any of the translators on the input side, either I/O V_L or I/O V_{CC}. Then, a short pulse is generated during which the accelerator output stages become active and charge/discharge the capacitances at the I/Os. Due to its bidirectional nature, both input stages become active during the one-shot pulse. This can lead to some current feeding into the external source that is driving the translator. However, this behavior helps to speed up the transition on the driven side.

For proper operation, the driver has to meet the following conditions: 50Ω maximum output impedance and 20mA minimum output current (for 20Mbps versions), 400Ω maximum output impedance and 4mA minimum output current (for 4Mbps versions), 1kΩ maximum output impedance and 1mA minimum output current (for 230kbps versions). Figure 6 shows a typical input current vs. input voltage.

Enable Output Mode (EN, EN A/B)

The MAX3000E/MAX3001E/MAX3002 and the MAX3004–MAX3012 feature an EN input, and the MAX3003 has an EN A/B input. Pull EN low to set the MAX3000E/MAX3001E/MAX3002/MAX3004–MAX3012s' I/O $V_{CC}1$ through I/O $V_{CC}8$ in three-state output mode, while I/O V_L1 through I/O V_L8 have internal 6kΩ pulldown resistors. Drive EN to logic high (V_L) for normal operation. For the MAX3003, pull EN A/B low to place channels 1B through 4B in active mode, while channels 1A through 4A are in three-state mode. Drive EN A/B to logic high (V_L) to enable channels 1A through 4A, while channels 1B through 4B remain in three-state mode.

±15kV ESD Protection

As with all Maxim devices, ESD-protection structures are incorporated on all pins to protect against electrostatic discharges encountered during handling and assembly. The I/O V_{CC} lines have extra protection against static discharge. Maxim's engineers have developed state-of-the-art structures to protect these pins against ESD of ±15kV without damage. The ESD structures withstand high ESD in all states: normal operation, three-state output mode, and powered down. After an ESD event, Maxim's E versions keep working without latchup, whereas competing products can latch and must be powered down to remove latchup.

ESD protection can be tested in various ways. The I/O V_{CC} lines of the MAX3000E/MAX3001E are characterized for protection to ±15kV using the Human Body Model.

+1.2V to +5.5V, ±15kV ESD-Protected, 0.1µA, 35Mbps, 8-Channel Level Translators

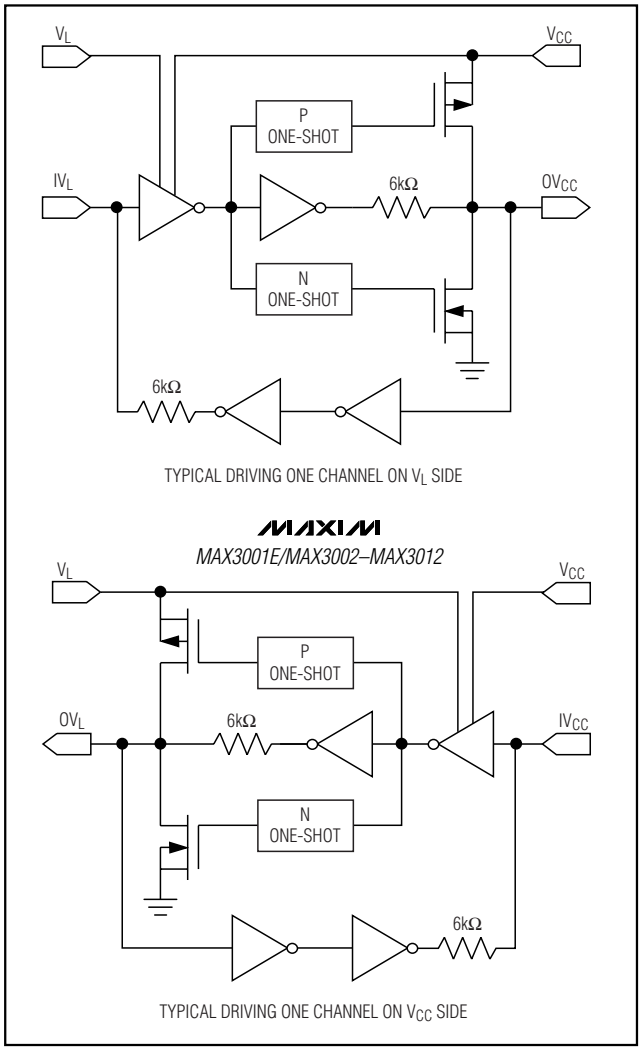


Figure 5. MAX3001E/MAX3002–MAX3012 Simplified Functional Diagram (1 I/O Line)

ESD Test Conditions

ESD performance depends on a variety of conditions. Contact Maxim for a reliability report that documents test setup, test methodology, and test results.

Human Body Model

Figure 7a shows the Human Body Model and Figure 7b shows the current waveform it generates when discharged into a low impedance. This model consists of a 100pF capacitor charged to the ESD voltage of interest, which is then discharged into the test device through a 1.5kΩ resistor.

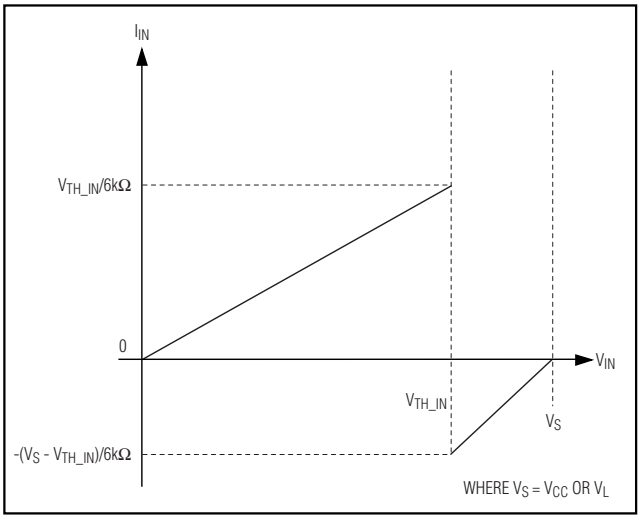


Figure 6. Typical I_{IN} vs. V_{IN}

Machine Model

The Machine Model for ESD tests all pins using a 200pF storage capacitor and zero discharge resistance. Its objective is to emulate the stress caused by contact that occurs with handling and assembly during manufacturing. Of course, all pins require this protection during manufacturing, not just inputs and outputs. Therefore, after PC board assembly, the Machine Model is less relevant to I/O ports.

Applications Information

Power-Supply Decoupling

To reduce ripple and the chance of transmitting incorrect data, bypass V_L and V_{CC} to ground with a 0.1µF capacitor. To ensure full ±15kV ESD protection, bypass V_{CC} to ground with a 1µF capacitor. Place all capacitors as close to the power-supply inputs as possible.

I²C Level Translation

For I²C level translation for I²C applications, please refer to the MAX3372E–MAX3379E/MAX3390E–MAX3393E datasheet.

Unidirectional vs. Bidirectional Level Translator

The MAX3000E/MAX3001E/MAX3002/MAX3003 can also be used to translate signals without inversion. These devices provide the smallest solution (UCSP package) for unidirectional level translation without inversion.

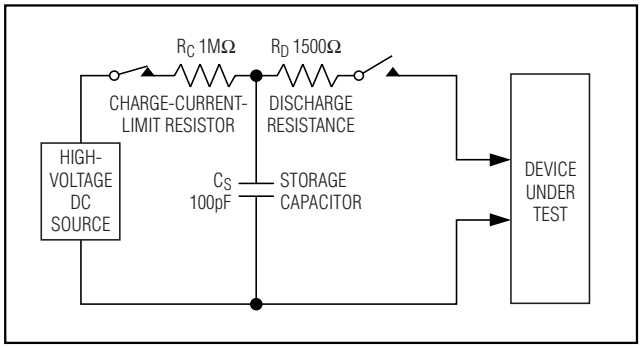


Figure 7a. Human Body ESD Test Model

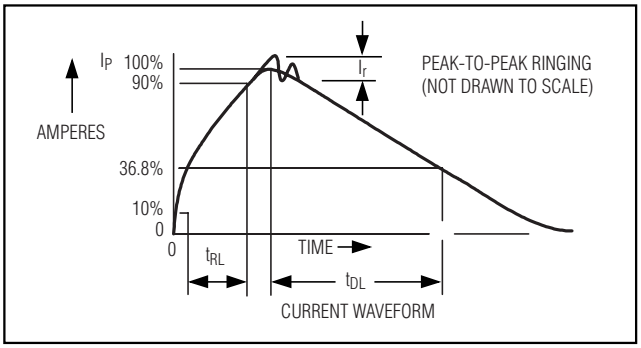


Figure 7b. Human Body Current Waveform

Selector Guide

PART	EN	EN A/B	Tx/Rx*	DATA RATE	ESD PROTECTION (kV)
MAX3000E	√	—	8/8	230kbps	±15
MAX3001E	√	—	8/8	4Mbps	±15
MAX3002	√	—	8/8	**	±2
MAX3003	—	√	8/8	**	±2
MAX3004	√	—	8/0	**	±2
MAX3005	√	—	7/1	**	±2
MAX3006	√	—	6/2	**	±2
MAX3007	√	—	5/3	**	±2
MAX3008	√	—	4/4	**	±2
MAX3009	√	—	3/5	**	±2
MAX3010	√	—	2/6	**	±2
MAX3011	√	—	1/7	**	±2
MAX3012	√	—	0/8	**	±2

*$Tx = V_L \rightarrow V_{CC}$; $Rx = V_{CC} \rightarrow V_L$

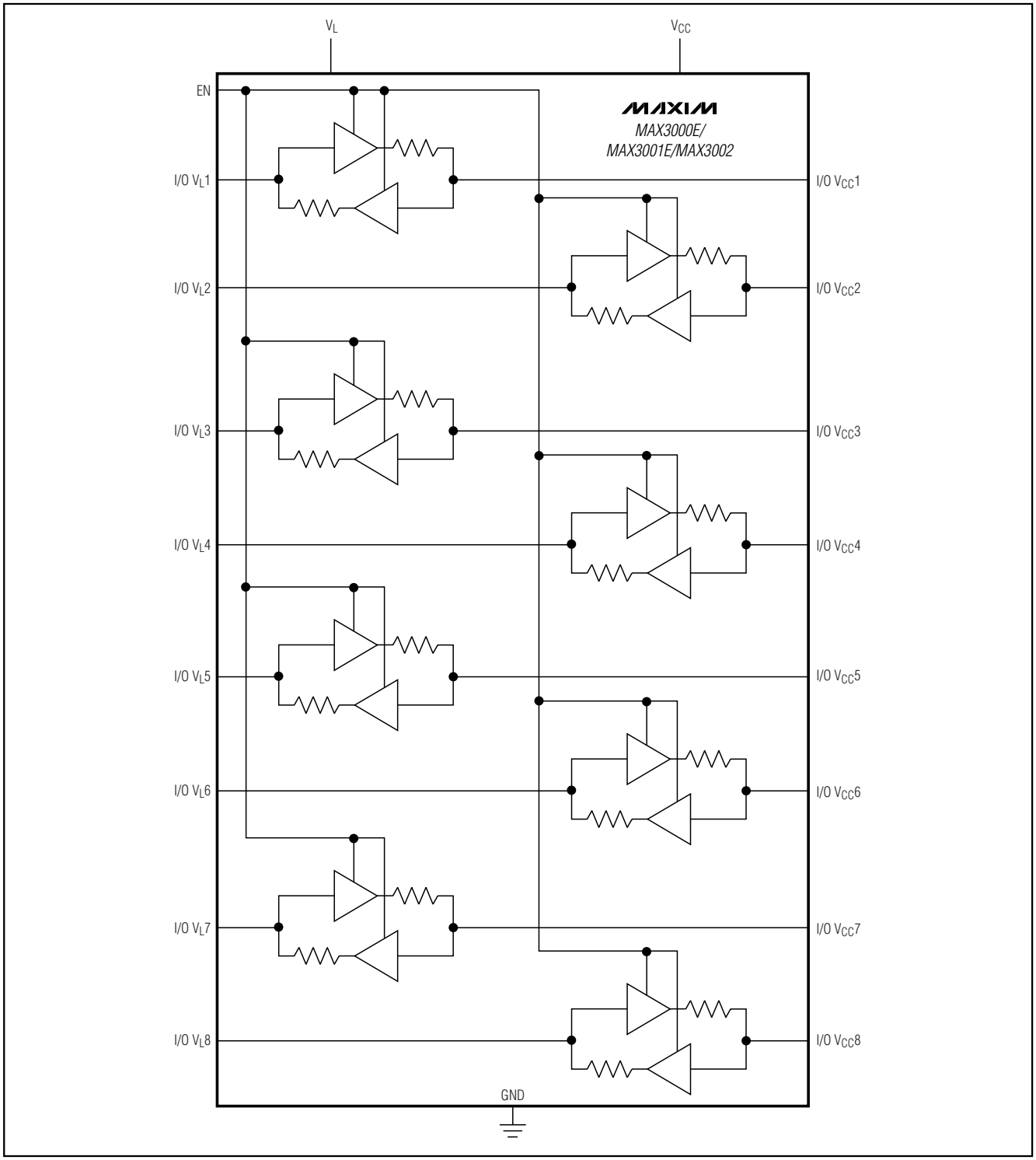
**See Table 1.

Table 1. Data Rate

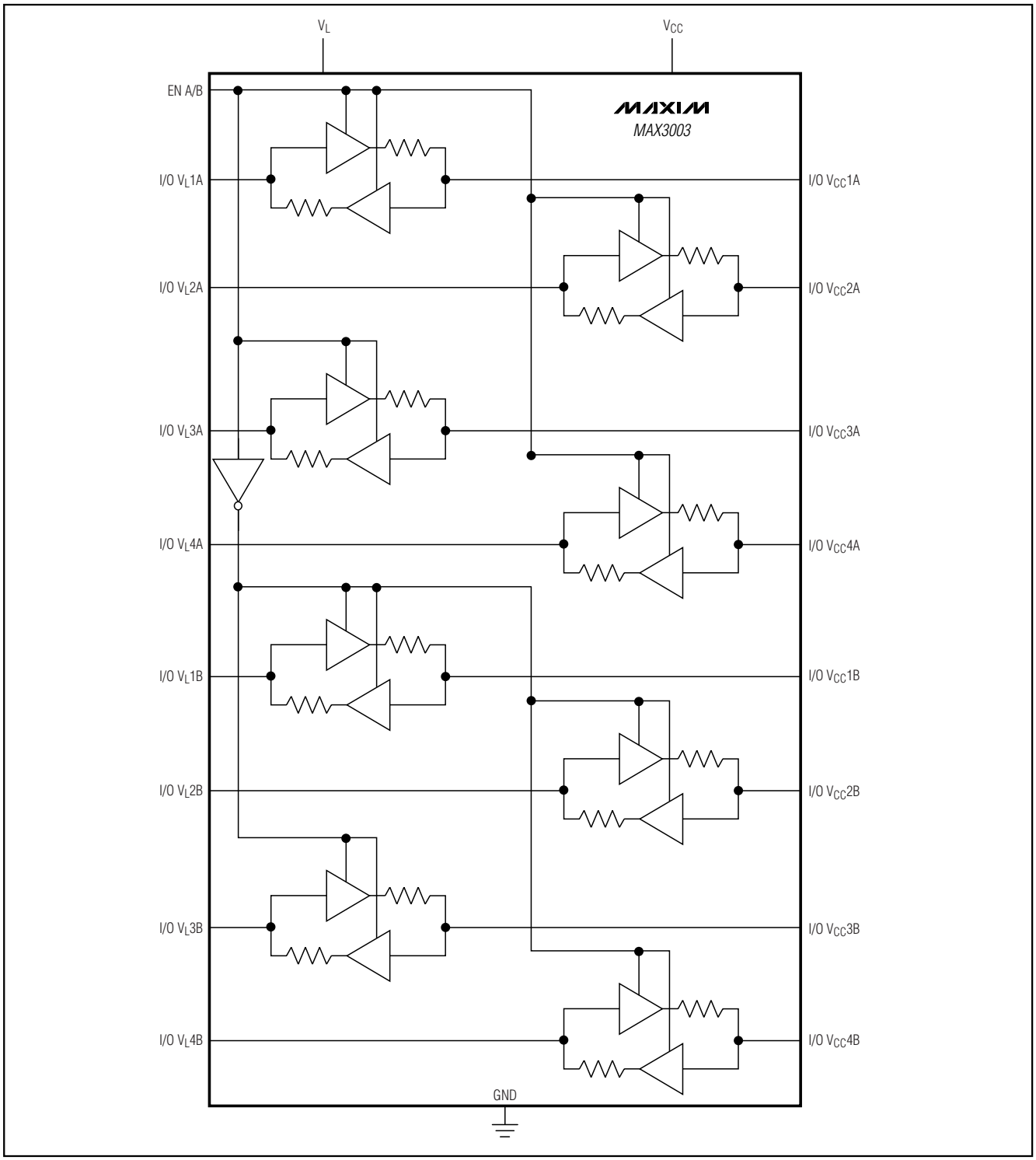
$V_L \leftrightarrow V_{CC}$ (V)	MAX3002–MAX3012 GUARANTEED DATA RATE (Mbps)
1.2 ↔ 5.5	40
1.2 ↔ 3.3	20
2.5 ↔ 3.3	35
1.8 ↔ 2.5	30
1.2 ↔ 2.5	20
1.2 ↔ 1.8	20

+1.2V to +5.5V, ±15kV ESD-Protected, 0.1µA, 35Mbps, 8-Channel Level Translators

MAX3000E/MAX3001E/MAX3002 Functional Diagram

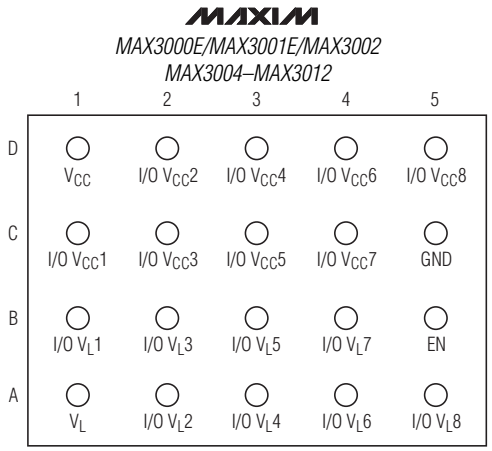


MAX3003 Functional Diagram

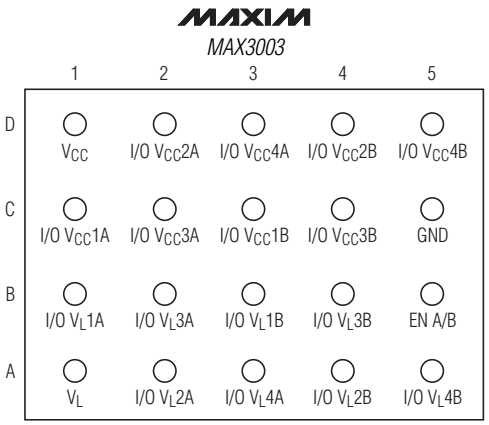


+1.2V to +5.5V, ±15kV ESD-Protected, 0.1µA, 35Mbps, 8-Channel Level Translators

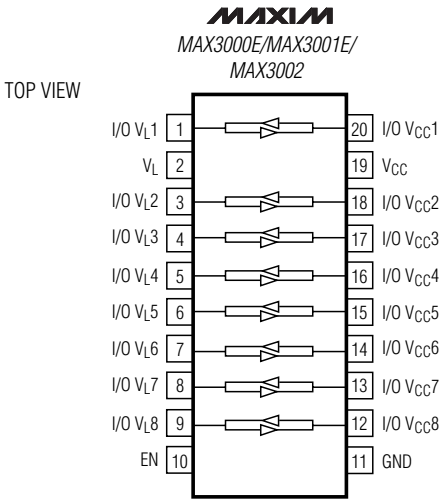
Pin Configurations



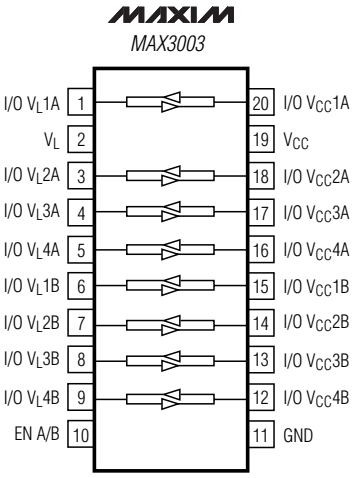
20 UCSP
(Bottom View)



20 UCSP
(Bottom View)



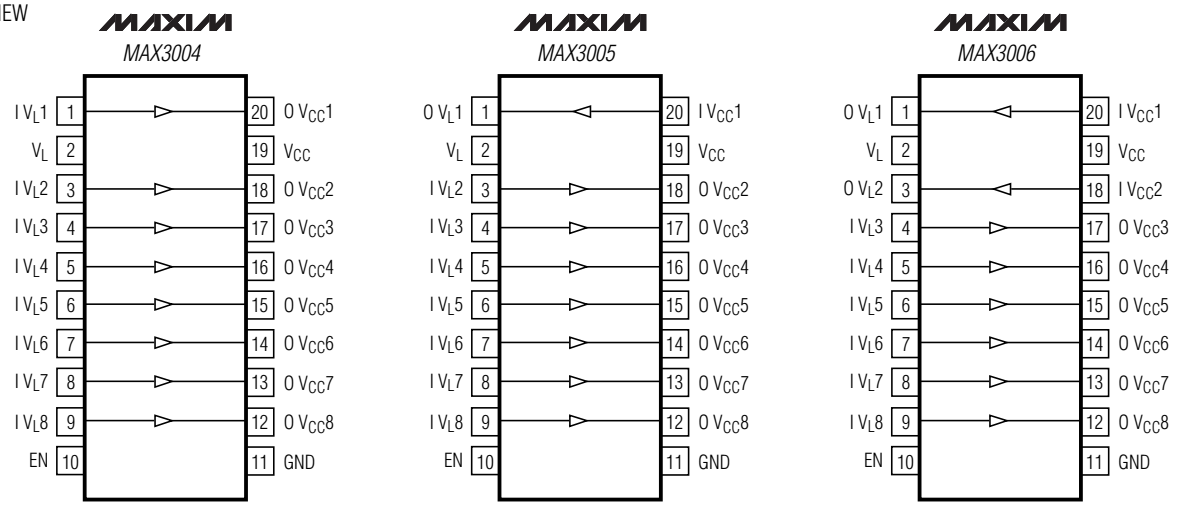
TSSOP



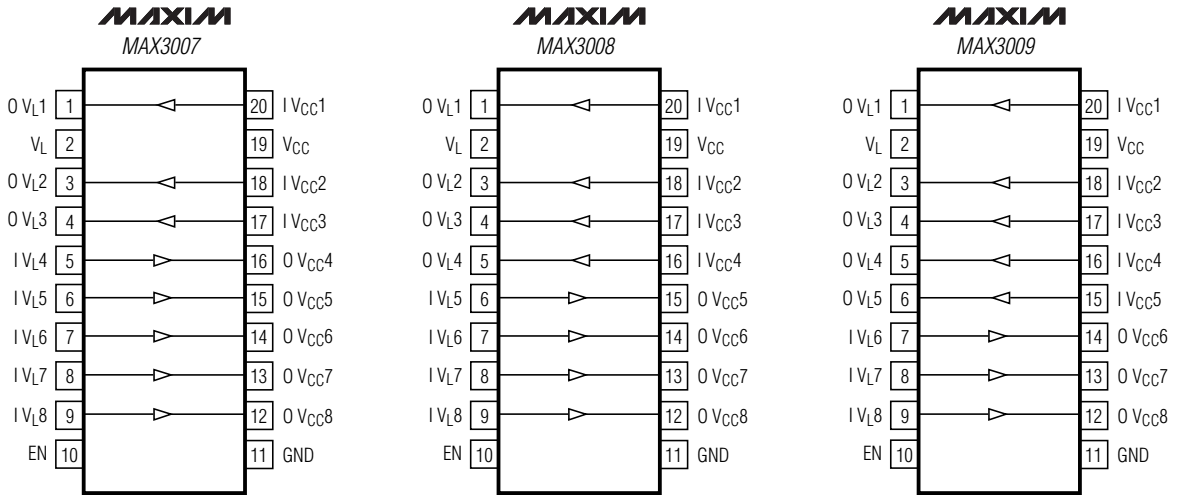
TSSOP

Pin Configurations (continued)

TOP VIEW



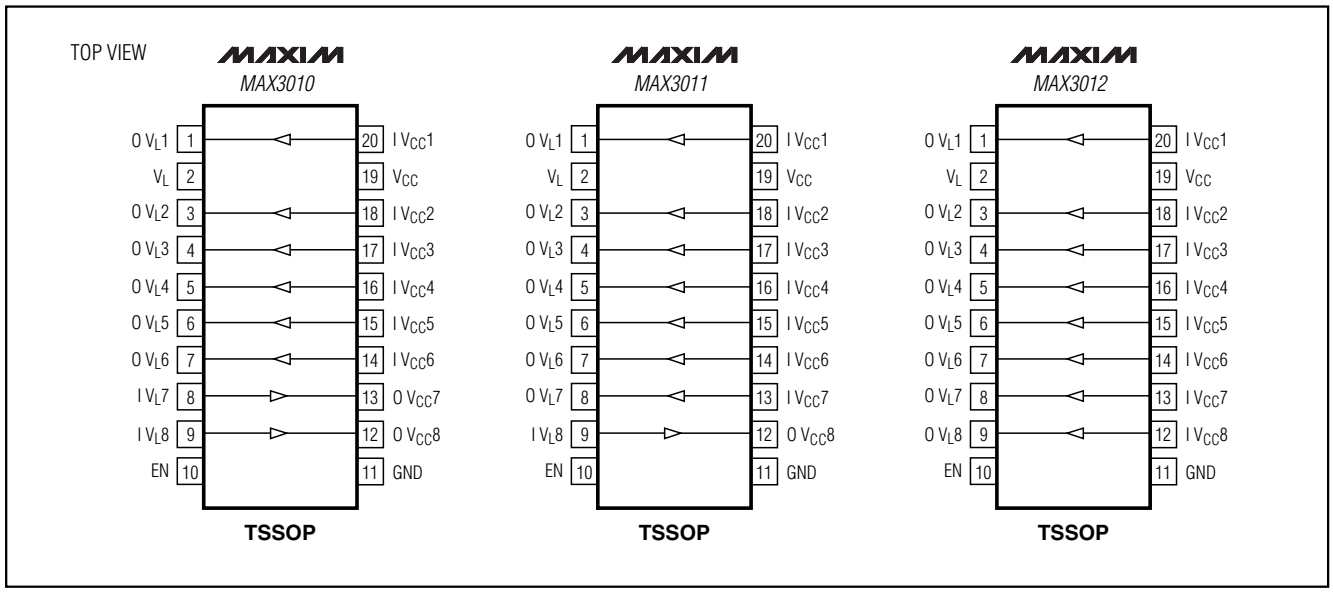
TSSOP TSSOP TSSOP



TSSOP TSSOP TSSOP

+1.2V to +5.5V, ±15kV ESD-Protected, 0.1µA, 35Mbps, 8-Channel Level Translators

Pin Configurations (continued)



Ordering Information (continued)

PART	TEMP RANGE	PIN-PACKAGE
MAX3001EEUP	-40°C to +85°C	20 TSSOP
MAX3001EEBP-T*	-40°C to +85°C	4 x 5 UCSP
MAX3001EAUP	-40°C to +125°C	20 TSSOP
MAX3002EUP	-40°C to +85°C	20 TSSOP
MAX3002EBP-T*	-40°C to +85°C	4 x 5 UCSP
MAX3003EUP	-40°C to +85°C	20 TSSOP
MAX3003EBP-T*	-40°C to +85°C	4 x 5 UCSP
MAX3004EUP*	-40°C to +85°C	20 TSSOP
MAX3004EBP-T*	-40°C to +85°C	4 x 5 UCSP
MAX3005EUP*	-40°C to +85°C	20 TSSOP
MAX3005EBP-T*	-40°C to +85°C	4 x 5 UCSP
MAX3006EUP*	-40°C to +85°C	20 TSSOP
MAX3006EBP-T*	-40°C to +85°C	4 x 5 UCSP

PART	TEMP RANGE	PIN-PACKAGE
MAX3007EUP*	-40°C to +85°C	20 TSSOP
MAX3007EBP-T*	-40°C to +85°C	4 x 5 UCSP
MAX3008EUP*	-40°C to +85°C	20 TSSOP
MAX3008EBP-T*	-40°C to +85°C	4 x 5 UCSP
MAX3009EUP*	-40°C to +85°C	20 TSSOP
MAX3009EBP-T*	-40°C to +85°C	4 x 5 UCSP
MAX3010EUP*	-40°C to +85°C	20 TSSOP
MAX3010EBP-T*	-40°C to +85°C	4 x 5 UCSP
MAX3011EUP*	-40°C to +85°C	20 TSSOP
MAX3011EBP-T*	-40°C to +85°C	4 x 5 UCSP
MAX3012EUP*	-40°C to +85°C	20 TSSOP
MAX3012EBP-T*	-40°C to +85°C	4 x 5 UCSP

**Future product—contact factory for availability.*

Chip Information

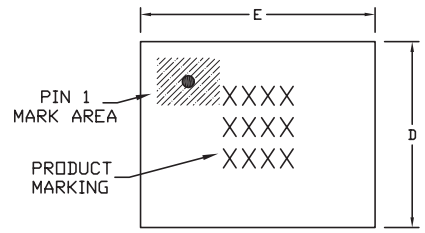
TRANSISTOR COUNT: 1184

PROCESS: BiCMOS

Package Information

(The package drawing(s) in this data sheet may not reflect the most current specifications. For the latest package outline information, go to **www.maxim-ic.com/packages**.)

5x4 UCSP.EPS

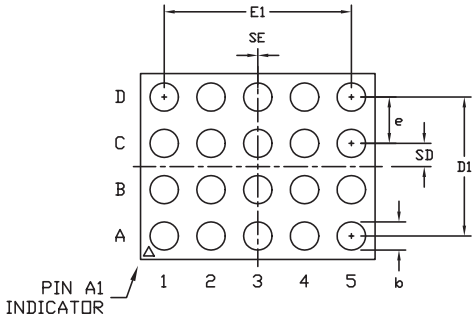


TOP VIEW

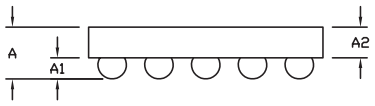
COMMON DIMENSIONS	
A	0.62+0.05-0.08
A1	0.29±0.02
A2	0.33 REF.
b	ø0.35±0.03
D1	1.50 BASIC
E1	2.00 BASIC
e	0.50 BASIC
SD	0.25 BASIC
SE	0.00 BASIC

PKG. CODE	VARIABLE DIMENSIONS		DEPOPULATED SOLDER BALLS
	D	E	
B20-1	2.03±0.08	2.54±0.08	NONE
B20-2	2.03±0.08	2.54±0.08	B2, B3, B4
B20-4	2.03±0.08	2.54±0.08	B2, B3, B4, C2, C4
B20-5	2.03±0.08	2.54±0.08	B2, B3, B4, C2, C3, C4
B20-6	2.16±0.08	2.54±0.08	B2, B3, B4

NOTES:
1. ALL DIMENSIONS ARE IN MILLIMETERS.
2. PRODUCT MARKING: NUMBER OF CHARACTERS AND LINES VARY PER PRODUCT.



BOTTOM VIEW



SIDE VIEW

DALLAS SEMICONDUCTOR MAXIM
PROPRIETARY INFORMATION

TITLE:
PACKAGE OUTLINE, 5x4 UCSP

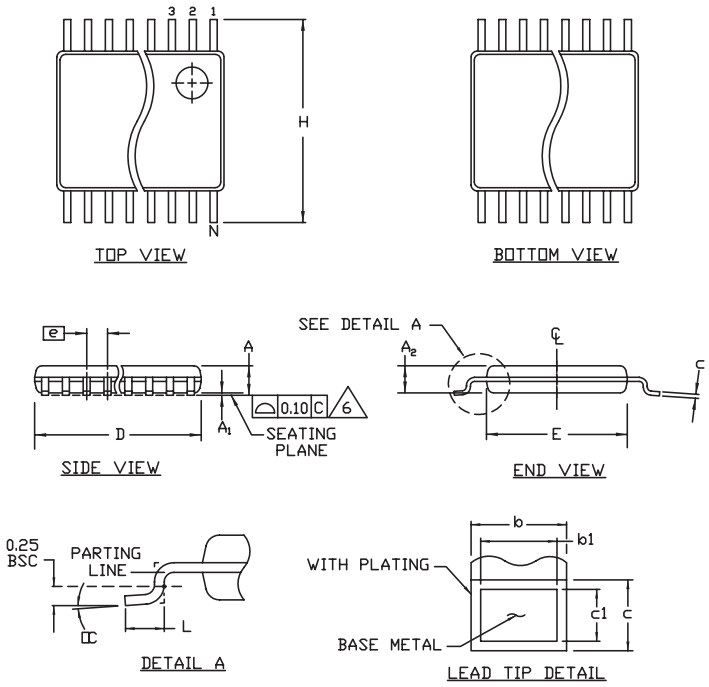
APPROVAL	DOCUMENT CONTROL NO.	REV.	
	21-0095	I	1/1

+1.2V to +5.5V, ±15kV ESD-Protected, 0.1µA, 35Mbps, 8-Channel Level Translators

Package Information (continued)

(The package drawing(s) in this data sheet may not reflect the most current specifications. For the latest package outline information, go to **www.maxim-ic.com/packages**.)

TSSOP4.40mm.EPS



SYMBOL	COMMON DIMENSIONS			
	MILLIMETERS		INCHES	
	MIN.	MAX.	MIN.	MAX.
A	——	1.10		.043
A_1	0.05	0.15	.002	.006
A_2	0.85	0.95	.033	.037
b	0.19	0.30	.007	.012
b_1	0.19	0.25	.007	.010
c	0.09	0.20	.004	.008
c_1	0.09	0.14	.004	.006
D	SEE VARIATIONS		SEE VARIATIONS	
E	4.30	4.50	.169	.177
e	0.65 BSC		.026 BSC	
H	6.25	6.55	.246	.258
L	0.50	0.70	.020	.028
N	SEE VARIATIONS		SEE VARIATIONS	
α	0°	8°	0°	8°

JEDEC			VARIATIONS			
MO-153	N		MILLIMETERS		INCHES	
			MIN.	MAX.	MIN.	MAX.
AB-1	14	D	4.90	5.10	.193	.201
AB	16	D	4.90	5.10	.193	.201
AC	20	D	6.40	6.60	.252	.260
AD	24	D	7.70	7.90	.303	.311
AE	28	D	9.60	9.80	.378	.386

NOTES:
1. DIMENSIONS D AND E DO NOT INCLUDE FLASH
2. MOLD FLASH OR PROTRUSIONS NOT TO EXCEED 0.15mm PER SIDE
3. CONTROLLING DIMENSION: MILLIMETER
4. MEETS JEDEC OUTLINE MO-153. SEE JEDEC VARIATIONS TABLE
5. "N" REFERS TO NUMBER OF LEADS
6. THE LEAD TIPS MUST LIE WITHIN A SPECIFIED ZONE. THIS TOLERANCE ZONE IS DEFINED BY TWO PARALLEL PLANES. ONE PLANE IS THE SEATING PLANE, DATUM [-C-]; THE OTHER PLANE IS AT THE SPECIFIED DISTANCE FROM [-C-] IN THE DIRECTION INDICATED

DALLAS SEMICONDUCTOR MAXIM
PROPRIETARY INFORMATION
TITLE: PACKAGE OUTLINE, TSSOP 4.40mm BODY

APPROVAL	DOCUMENT CONTROL NO.	REV.	
	21-0066	F	1/1

19-0275; Rev 1; 7/99

5V/3.3V or Adjustable, Low-Dropout, Low I_Q, 200mA Linear Regulators

General Description

The MAX882/MAX883/MAX884 linear regulators maximize battery life by combining ultra-low supply currents and low dropout voltages. They feature 200mA output current capability at up to +125°C junction temperature and come in a 1.5W SOIC package. The 1.5W package (compared to 0.47W for standard SOIC packages) allows a wider operating range for the input voltage and output current. The MAX882/MAX883/MAX884 use a P-channel MOSFET pass transistor to maintain a low 11µA (15µA max) supply current from no-load to the full 200mA output. Unlike earlier bipolar regulators, there are no PNP base current losses that increase with output current. In dropout, the MOSFET does not suffer from excessive base currents that occur when PNP transistors go into saturation. Typical dropout voltages are 220mV at 5V and 200mA, or 320mV at 3.3V and 200mA.

The MAX882 features a 7µA standby mode that disables the output but keeps the reference, low-battery comparator, and biasing circuitry alive. The MAX883/MAX884 feature a shutdown (OFF) mode that turns off all circuitry, reducing supply current to less than 1µA. All three devices include a low-battery-detection comparator, foldback current limiting, reverse-current protection, and thermal-overload protection.

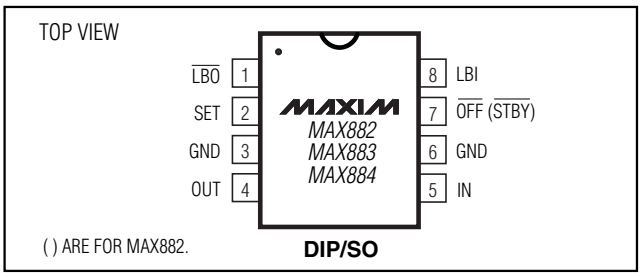
The output is preset at 3.3V for the MAX882/MAX884 and 5V for the MAX883. In addition, all devices employ Dual Mode™ operation, allowing user-adjustable outputs from 1.25V to 11V using external resistors. The input voltage supply range is 2.7V to 11.5V.

For low-dropout linear regulators with output currents up to 500mA, refer to the MAX603/MAX604 data sheet.

Applications

Pagers and Cellular Phones
3.3V and 5V Regulators
1.25V to 11V Adjustable Regulators
High-Efficiency Linear Regulators
Battery-Powered Devices
Portable Instruments
Solar-Powered Instruments

Pin Configuration



Dual Mode is a trademark of Maxim Integrated Products.

Features

- **Guaranteed 200mA Output Current at T_J = +125°C, with Foldback Current Limiting**
- **High-Power (1.5W) 8-Pin SO Package**
- **Dual Mode Operation: Fixed or Adjustable Output from 1.25V to 11V**
- **Large Input Range (2.7V to 11.5V)**
- **Internal 1.1Ω P-Channel Pass Transistor Draws No Base Current**
- **Low 220mV Dropout Voltage at 200mA Output Current**
- **11µA (typ) Quiescent Current**
- **1µA (max) Shutdown Mode or 7µA (typ) Standby Mode**
- **Low-Battery Detection Comparator**
- **Reverse-Current Protection**
- **Thermal-Overload Protection**

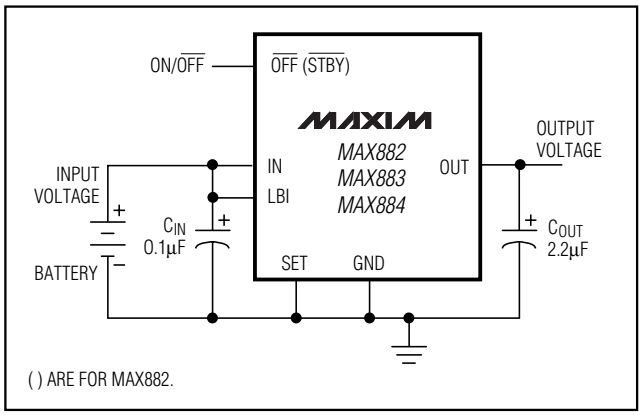
Ordering Information

PART	TEMP. RANGE	PIN-PACKAGE
MAX882CPA	0°C to +70°C	8 Plastic DIP
MAX882CSA	0°C to +70°C	8 SO
MAX882C/D	0°C to +70°C	Dice*
MAX882EPA	-40°C to +85°C	8 Plastic DIP
MAX882ESA	-40°C to +85°C	8 SO
MAX882MJA	-55°C to +125°C	8 CERDIP**

Ordering Information continued at end of data sheet.

** Dice are tested at T_J = +25°C, DC parameters only.*
*** Contact factory for availability.*

Typical Operating Circuit



5V/3.3V or Adjustable, Low-Dropout, Low I_Q, 200mA Linear Regulators

ABSOLUTE MAXIMUM RATINGS

Supply Voltage (IN or OUT to GND).....-0.3V to +12V
Output Short-Circuit Duration 1min
Continuous Output Current.....300mA
$\overline{LBO}$ Output Current.....50mA
$\overline{LBO}$ Output Voltage and LBI,
SET, STBY, OFF Input Voltages .....-0.3V to the greater of (IN + 0.3V) or (OUT + 0.3V)
Continuous Power Dissipation (T_J = +70°C)
Plastic DIP (derate 9.09mW/°C above +70°C) 727mW
High-Power SO (derate 18.75mW/°C above +70°C).....1.5W
CERDIP (derate 8.00mW/°C above +70°C).....640mW
Operating Temperature Ranges
MAX88_C_A.....0°C to +70°C
MAX88_E_A .....-40°C to +85°C
MAX88_MJA .....-55°C to +125°C
Junction Temperature .....+150°C
Storage Temperature Range .....-65°C to +160°C
Lead Temperature (soldering, 10s) .....+300°C

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of the specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

ELECTRICAL CHARACTERISTICS

(V_{IN} = 6V (MAX883) or V_{IN} = 4.3V (MAX882/MAX884), C_{OUT} = 2.2µF, STBY or OFF = V_{IN}, SET = GND, LBI = V_{IN}, T_J = T_{MIN} to T_{MAX}, unless otherwise noted. Typical values are at T_J = +25°C.) (Note 1)

PARAMETER	SYMBOL	CONDITIONS		MIN	TYP	MAX	UNITS
Input Voltage Range	V_{IN}	SET = OUT, R_L = 1kΩ	MAX88_C_A	2.7		11.5	V
			MAX88_E_A	2.9		11.5	
			MAX88_MJA	3.0		11.5	
Output Voltage (Note 2)	V_{OUT}	MAX883, 6.0V ≤ V_{IN} ≤ 11.5V	I_{OUT} = 10µA–200mA, T_J ≤ +125°C	4.75	5.00	5.25	V
			I_{OUT} = 10µA–250mA, T_J ≤ +85°C				
			I_{OUT} = 10µA–250mA, T_J ≤ +70°C				
		MAX882/MAX884, 4.3V ≤ V_{IN} ≤ 11.5V	I_{OUT} = 10µA–150mA, T_J ≤ +125°C	3.15	3.30	3.45	
			I_{OUT} = 10µA–200mA, T_J ≤ +85°C				
			I_{OUT} = 10µA–200mA, T_J ≤ +70°C				
Load Regulation	ΔV_{LDR}	I_{OUT} = 1mA to 200mA	MAX883C_A/E_A		60	100	mV
			MAX883MJA			150	
		I_{OUT} = 1mA to 150mA	MAX882, MAX884		30	100	
Line Regulation	ΔV_{LNR}	(V_{OUT} + 0.5V) < V_{IN} < 11.5V, I_{OUT} = 10mA			10	40	mV
Dropout Voltage (Note 3)	ΔV_{DO}	MAX883	I_{OUT} = 100mA		110	220	mV
			I_{OUT} = 200mA		220	440	
		MAX882/MAX884	I_{OUT} = 100mA		160	320	
			I_{OUT} = 200mA		320	640	
Quiescent Current	I_Q	SET = OUT, V_{IN} = 6V	MAX88_C_A/E_A		11	15	µA
			MAX88_MJA			30	
		V_{IN} = 11.5V	MAX88_C_A/E_A		15	25	
			MAX88_MJA			40	

ELECTRICAL CHARACTERISTICS (continued)

(V_{IN} = 6V (MAX883) or V_{IN} = 4.3V (MAX882/MAX884), C_{OUT} = 2.2µF, STBY or OFF = V_{IN}, SET = GND, LBI = V_{IN}, T_J = T_{MIN} to T_{MAX}, unless otherwise noted. Typical values are at T_J = +25°C.) (Note 1)

PARAMETER	SYMBOL	CONDITIONS		MIN	TYP	MAX	UNITS
STBY Quiescent Current (Note 4)	$I_{Q\ STBY}$	STBY = 0, V_{IN} = 6V, SET = OUT	MAX882C_A/E_A		7	15	µA
			MAX882MJA			30	
		STBY = 0, V_{IN} =11.5V, SET = OUT	MAX882C_A/E_A		10	25	
			MAX882MJA			40	
OFF Quiescent Current	$I_{Q\ OFF}$	OFF = 0, R_L = 1kΩ, V_{IN} = 11.5V (MAX883/MAX884)	MAX88_C_A		0.01	1	µA
			MAX88_E_A			5	
			MAX88_MJA			10	
Minimum Load Current	$I_{OUT(MIN)}$	V_{IN} = 11.5V, SET = OUT	MAX88_C_A			1	µA
			MAX88_E_A			3	
			MAX88_MJA			10	
Foldback Current Limit (Note 5)	I_{LIM}	V_{OUT} < 0.8V			170		mA
		V_{OUT} > 0.8V and V_{IN} - V_{OUT} > 0.7V			430		
Thermal Shutdown Temperature	T_{SD}				160		°C
Thermal Shutdown Hysteresis	ΔT_{SD}				10		°C
Reverse-Current-Protection Threshold (Note 6)	ΔV_{RTH}	V_{OUT} = 4.5V	MAX883_A		6	20	mV
		V_{OUT} = 3.0V	MAX882_A, MAX884_A		6	20	
Reverse Leakage Current	I_{RVL}	MAX882: V_{IN} = 0, STBY = 0, V_{OUT} = 3.0V			7		µA
		MAX883/MAX884: V_{IN} = 0, OFF = 0, V_{OUT} = 3.0V			0.01		
Startup Overshoot	V_{OSH}	R_L = 1kΩ, C_{OUT} = 2.2µF			1		% of V_{OUT}
Time Required to Exit OFF or STBY Modes	T_{START}	V_{IN} = 9V, R_L = 33Ω, OFF from 0 to V_{IN}, 0% to 95% of V_{OUT}			200		µs
Dual Mode SET Threshold	$V_{SET\ TH}$	For internal feedback			65	30	mV
		For external feedback		150	65		
SET Reference Voltage	V_{SET}	SET = OUT, R_L = 1kΩ		1.16	1.20	1.24	V
SET Input Leakage Current	I_{SET}	V_{SET} = 1.5V or 0			±0.01	±50	nA
LBI Threshold Voltage	V_{LBI}	LBI signal falling		1.15	1.20	1.25	V
LBI Hysteresis	ΔV_{LBI}				7		mV
LBI Input Leakage Current	I_{LBI}	V_{LBI} = 1.5V			±0.01	±50	nA
$\overline{LBO}$ Output Low Voltage	$V_{\overline{LBO}L}$	$I_{\overline{LBO}\ SINK}$ = 1.2mA, V_{LBI} =1V, 3V < V_{IN} < 11.5V, SET = OUT			90	250	mV
$\overline{LBO}$ Output Leakage Current	$I_{\overline{LBO}\ LKG}$	V_{LBI} = V_{IN}, $V_{\overline{LBO}}$ = V_{IN}			0.01	0.1	µA
OUT Leakage Current	$I_{OUT\ LKG}$	V_{IN} = 11.5V, V_{OUT} = 2V, SET = OUT	MAX88_C_A		0.01	1	µA
			MAX88_E_A			3	
			MAX88_MJA			10	

5V/3.3V or Adjustable, Low-Dropout, Low IQ, 200mA Linear Regulators

ELECTRICAL CHARACTERISTICS (continued)

(V_{IN} = 6V (MAX883) or V_{IN} = 4.3V (MAX882/MAX884), C_{OUT} = 2.2µF, STBY or OFF = V_{IN}, SET = GND, LBI = V_{IN}, T_J = T_{MIN} to T_{MAX}, unless otherwise noted. Typical values are at T_J = +25°C.) (Note 1)

PARAMETER	SYMBOL	CONDITIONS		MIN	TYP	MAX	UNITS
STBY Threshold Voltage	V_{STBY}	STBY signal falling	MAX882_A	1.15	1.20	1.25	V
STBY Hysteresis	ΔV_{STBY}		MAX882_A		7		mV
STBY Input Leakage Current	I_{STBY}	V_{STBY} = V_{IN} or 0	MAX882_A		±0.01	±50	nA
OFF Threshold Voltage	$V_{IL\ OFF}$	In off mode	MAX883_A, MAX884_A			0.4	V
	$V_{IH\ OFF}$	In on mode, SET = OUT, V_{IN} < 6V	MAX883_A, MAX884_A	2.0			
		In on mode, SET = OUT, 6V < V_{IN} < 11.5V	MAX883_A, MAX884_A	3.0			
OFF Input Leakage Current	I_{OFF}	V_{OFF} = V_{IN} or 0			±0.01	±50	nA
Output Noise (Note 7)	e_n	10Hz to 10kHz, SET = OUT, R_L = 1kΩ, C_{OUT} = 2.2µF			250		μV_{RMS}

Note 1: Electrical specifications are measured by pulse testing and are guaranteed for a junction temperature (T_J) within the operating temperature range, unless otherwise noted. When operating C- and E-grade parts up to a T_J of +125°C, expect performance similar to M-grade specifications. For T_J between +125°C and +150°C, the output voltage may drift more.

Note 2: (V_{IN} - V_{OUT}) is limited to keep the product (I_{OUT} x (V_{IN} - V_{OUT})) from exceeding the package power dissipation limits. See Figure 5. Therefore, the combination of high output current and high supply voltage is not tested. Output current at T_J = +125°C is guaranteed by guard banding tests at T_J = +85°C and +70°C.

Note 3: Dropout Voltage is (V_{IN} - V_{OUT}) when V_{OUT} falls to 100mV below its nominal value at V_{IN} = (V_{OUT} + 2V). For example, the MAX883 is tested by measuring the V_{OUT} at V_{IN} = 7V, then V_{IN} is lowered until V_{OUT} falls 100mV below the measured value. The difference (V_{IN} - V_{OUT}) is then measured and defined as ΔV_{DO}.

Note 4: Since standby mode inhibits the output but keeps all biasing circuitry alive, the Standby Quiescent Current is similar to the normal operating quiescent current.

Note 5: Foldback Current Limit was characterized by pulse testing to remain below the maximum junction temperature (not production tested).

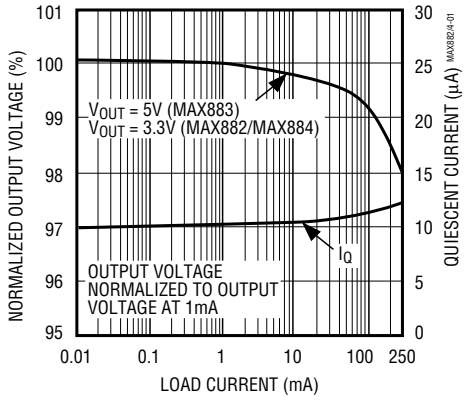
Note 6: The Reverse-Current Protection Threshold is the output/input differential voltage (V_{OUT} - V_{IN}) at which reverse-current protection switchover occurs and the pass transistor is turned off. See the section *Reverse-Current Protection* in the *Detailed Description.*

Note 7: Noise is tested using a bandpass amplifier with two poles at 10Hz and two poles at 10kHz.

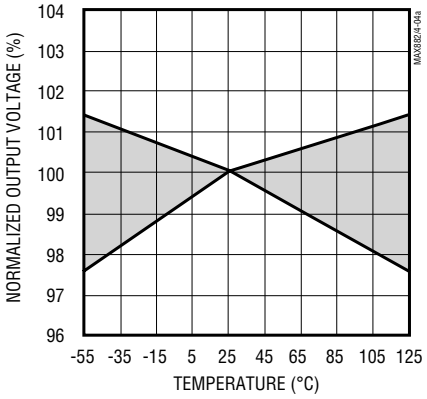
Typical Operating Characteristics

(V_{IN} = 7V for MAX883, VIN = 5.3V for MAX882/MAX884, OFF or STBY = VIN, SET = GND, LBI = V_{IN}, $\overline{LBO}$ = OPEN, C_{IN} = C_{OUT} = 2.2µF, R_L = 1kΩ, T_A = +25°C, unless otherwise noted.)

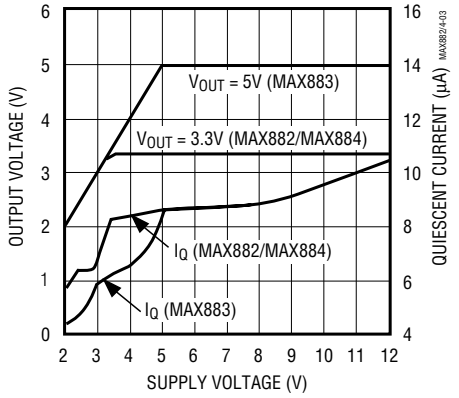
OUTPUT VOLTAGE AND QUIESCENT CURRENT vs. LOAD CURRENT



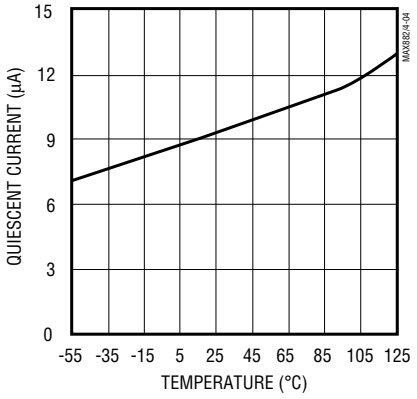
OUTPUT VOLTAGE vs. TEMPERATURE



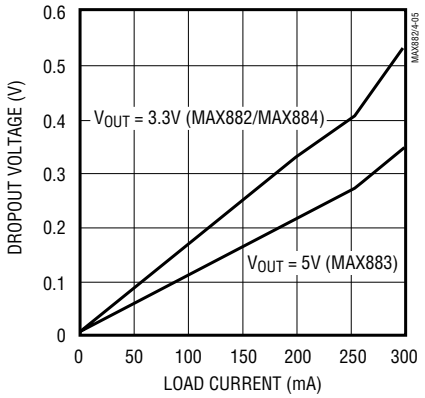
OUTPUT VOLTAGE AND QUIESCENT CURRENT vs. SUPPLY VOLTAGE



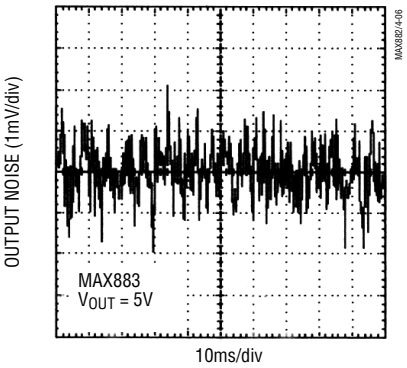
QUIESCENT CURRENT vs. TEMPERATURE



DROPOUT VOLTAGE vs. LOAD CURRENT



10Hz to 10kHz OUTPUT NOISE

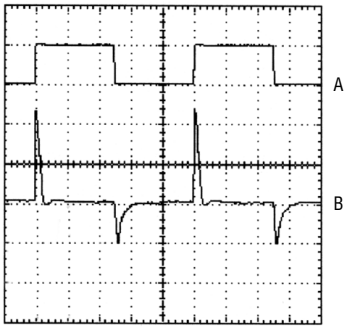


5V/3.3V or Adjustable, Low-Dropout, Low IQ, 200mA Linear Regulators

Typical Operating Characteristics (continued)

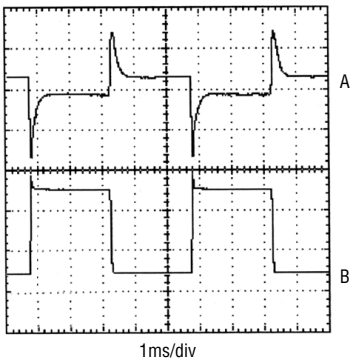
(V_{IN} = 7V for MAX883, V_{IN} = 5.3V for MAX882/MAX884, OFF or STBY = V_{IN}, SET = GND, LBI = V_{IN}, $\overline{LBO}$ = OPEN, C_{IN} = C_{OUT} = 2.2µF, R_L = 1kΩ, T_A = +25°C, unless otherwise noted.)

LINE-TRANSIENT RESPONSE



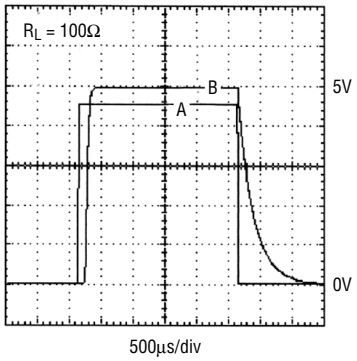
MAX883: V_{OUT} = 5V, C_{IN} = 0µF, t_R = 15µs, t_F = 13µs
A: V_{IN} = 8V (HIGH) / V_{IN} = 7V (LOW)
B: OUTPUT VOLTAGE (100mV/div)

LOAD-TRANSIENT RESPONSE



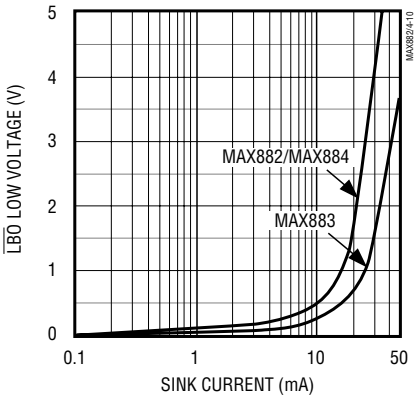
MAX883: V_{OUT} = 5V, t_R = 24µs, t_F = 44µs
A: OUTPUT VOLTAGE (100mV/div)
B: I_{OUT} = 250mA (HIGH) / I_{OUT} = 50mA (LOW)

OVERSHOOT AND TIME EXITING SHUTDOWN MODE



A: $\overline{OFF}$ PIN VOLTAGE (1V/div):
RISE TIME = 9µs
B: MAX883 OUTPUT VOLTAGE (1V/div):
DELAY = 135µs, RISE TIME = 67µs,
OVERSHOOT = 0%

$\overline{LBO}$ LOW VOLTAGE vs. SINK CURRENT



Pin Description

PIN		NAME	FUNCTION
MAX882	MAX883/ MAX884		
1	1	$\overline{LBO}$	Low-Battery Output is an open-drain output that goes low when LBI is less than 1.2V. Connect to IN or OUT through a pull-up resistor. $\overline{LBO}$ is undefined during shutdown mode (MAX883/MAX884).
2	2	SET	Feedback for setting the output voltage. Connect to GND to set the output voltage to the preselected 3.3V or 5V. Connect to an external resistor network for adjustable-output operation.
3, 6	3, 6	GND	Ground pins—also function as heatsinks in the SO package. All GND pins must be soldered to the PC board for proper power dissipation. Connect to large copper pads or planes to channel heat from the IC.
4	4	OUT	Regulator Output. Fixed or adjustable from 1.25V to 11.0V. Sources up to 200mA. Bypass with a 2.2µF capacitor.
5	5	IN	Regulator Input. Supply voltage can range from 2.7V to 11.5V.
7	—	STBY	Standby. Active-low comparator input. Connect to GND to disable the output or to IN for normal operation. A resistor network (from IN) can be used to set a standby mode threshold.
—	7	OFF	Shutdown. Active-low logic input. In OFF mode, supply current is reduced below 1µA and V_{OUT} = 0.
8	8	LBI	Low-Battery comparator Input. Tie to IN when not used.

Detailed Description

The MAX882/MAX883/MAX884 are micropower, low-dropout linear regulators designed primarily for battery-powered applications. They feature Dual Mode operation, allowing a fixed output of 5V for the MAX883 and 3.3V for the MAX882/MAX884, or an adjustable output from 1.25V to 11V. These devices supply up to 200mA while requiring less than 15µA quiescent current. As illustrated in Figure 1, they consist of a 1.20V reference, error amplifier, MOSFET driver, P-channel pass transistor, dual-mode comparator, and feedback voltage-divider.

The 1.20V reference is connected to the error amplifier's inverting input. The error amplifier compares this reference with the selected feedback voltage and amplifies the difference. The MOSFET driver reads the error signal and applies the appropriate drive to the P-channel pass transistor. If the feedback voltage is lower than the reference, the pass transistor's gate is pulled lower, allowing more current to pass and increasing the output voltage. If the feedback voltage is too high, the pass transistor gate is pulled up, allowing less current to pass to the output.

The output voltage is fed back through either an internal resistor voltage-divider connected to the OUT pin, or an external resistor network connected to the SET pin. The dual-mode comparator examines the SET pin voltage and selects the feedback path used. If the SET pin is below 65mV, internal feedback is used and the output voltage is regulated to 5V for the MAX883 or 3.3V for the MAX882/MAX884. Additional blocks include a foldback current limiter, reverse-current protection, a thermal sensor, shutdown or standby logic, and a low-battery-detection comparator.

Internal P-Channel Pass Transistor

The MAX882/MAX883/MAX884 feature a 200mA P-channel MOSFET pass transistor. This provides several advantages over similar designs using PNP pass transistors, including longer battery life.

The P-channel MOSFET requires no base drive, which reduces quiescent current considerably. PNP-based regulators waste large amounts of current in dropout when the pass transistor saturates. They also use high base-drive currents under large loads. The MAX882/MAX883/MAX884 do not suffer from these problems and consume only 11µA of quiescent current during light loads, heavy loads, and dropout.

Output Voltage Selection

The MAX882/MAX883/MAX884 feature Dual Mode operation. In preset voltage mode, the MAX883's output is set to 5V and the MAX882/MAX884's output is set to 3.3V, using internal trimmed feedback resistors. Select this mode by connecting SET to ground.

In preset voltage mode, impedances between SET and ground should be less than 100kΩ. Otherwise, spurious conditions could cause the voltage at SET to exceed the 65mV dual-mode threshold.

5V/3.3V or Adjustable, Low-Dropout, Low IQ, 200mA Linear Regulators

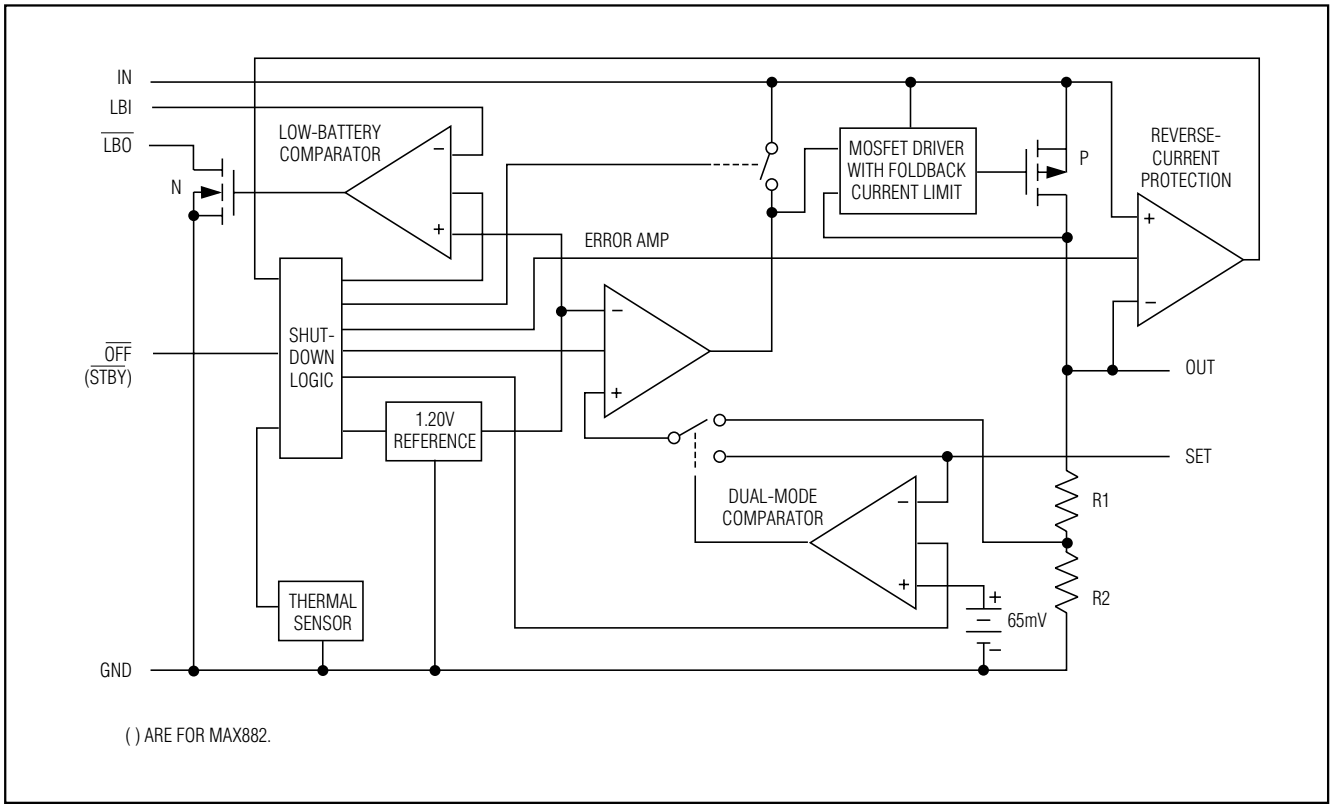


Figure 1. MAX882/MAX883/MAX884 Functional Diagram

In adjustable mode, the user selects an output voltage in the 1.25V to 11V range by connecting two external resistors, used as a voltage-divider, to the SET pin (Figure 2).

The output voltage is set by the following equation:

$$V_{OUT} = V_{SET}\left(1 + \frac{R1}{R2}\right)$$

where V_{SET} = 1.20V.

To simplify resistor selection:

$$R1 = R2\left(\frac{V_{OUT}}{V_{SET}} - 1\right)$$

Since the input bias current at SET is nominally zero, large resistance values can be used for R1 and R2 to minimize power consumption without losing accuracy. Up to 1.5MΩ is acceptable for R2. Since the V_{SET} tolerance is less than ±40mV, the output can be set using fixed resistors instead of trim pots.

Standby Mode (MAX882)

The MAX882 has a standby feature that disconnects the input from the output when STBY is brought low, but keeps all other circuitry awake. In this mode, V_{OUT} drops to 0, and the internal biasing circuitry (including the low-battery comparator) remains on. The maximum quiescent current during standby is 15µA. STBY is a comparator input with the other input internally tied to the reference voltage. Use a resistor network as shown in Figure 3 to set a standby-mode threshold voltage for undervoltage lockout. Connect STBY to IN for normal operation.

OFF Mode (MAX883/MAX884)

A low-logic input on the OFF pin shuts down the MAX883/MAX884. In this mode, the pass transistor, control circuit, reference, and all biases are turned off, and the supply current is reduced to less than 1µA. $\overline{LBO}$ is undefined in OFF mode. Connect OFF to IN for normal operation.

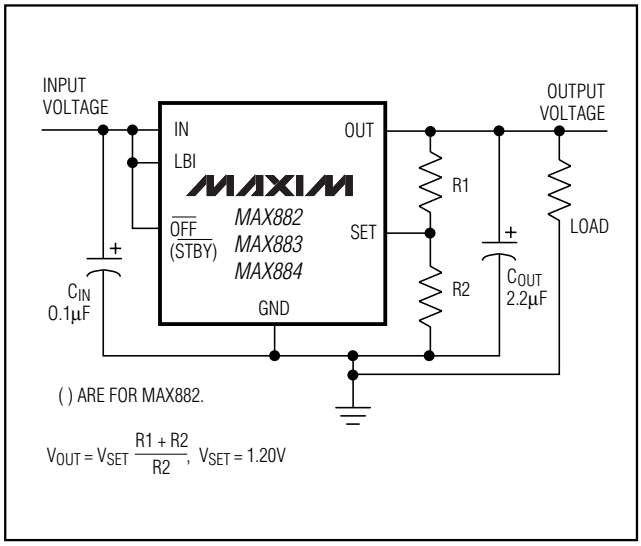


Figure 2. Adjustable Output Using External Feedback Resistors

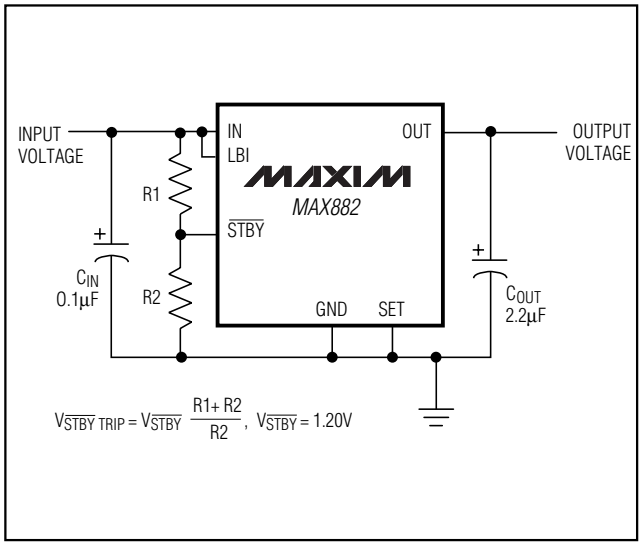


Figure 3. Setting an Undervoltage Lockout Threshold Using STBY

Foldback Current Limiting

The MAX882/MAX883/MAX884 also include a foldback current limiter. It monitors and controls the pass transistor's gate voltage, estimating the output current and limiting it to 430mA for output voltages above 0.8V and (V_{IN} - V_{OUT}) > 0.7V. If the output voltage drops below 0.8V, implying a short-circuit condition, the output current is limited to 170mA. The output can be shorted to ground for 1min without damaging the device if the package can dissipate (V_{IN} x 170mA) without exceeding T_J = +150°C. When the output is greater than 0.8V and (V_{IN} - V_{OUT}) < 0.7V (dropout operation), no current limiting is allowed, to provide maximum load drive.

Thermal Overload Protection

Thermal overload protection limits total power dissipation in the MAX882/MAX883/MAX884. When the junction temperature exceeds T_J =+160°C, the thermal sensor sends a signal to the shutdown logic, turning off the pass transistor and allowing the IC to cool. The thermal sensor turns the pass transistor on again after the IC's junction temperature cools by 10°C, resulting in a pulsed output during thermal overload conditions.

Thermal overload protection is designed to protect the MAX882/MAX883/MAX884 if fault conditions occur. It is not intended to be used as an operating mode. Prolonged operation in thermal-shutdown mode may reduce the IC's reliability. For continual operation, do not exceed the absolute maximum junction temperature rating of T_J = +150°C.

Power Dissipation and Operating Region

Maximum power dissipation of the MAX882/MAX883/MAX884 depends on the thermal resistance of the case and PC board, the temperature difference between the die junction and ambient air, and the rate of air flow. The power dissipation across the device is P = I_{OUT} (V_{IN} - V_{OUT}). The resulting power dissipation is as follows:

$$P = \frac{(T_J - T_A)}{(\theta_{JB} + \theta_{BA})}$$

where (T_J - T_A) is the temperature difference between the MAX882/MAX883/MAX884 die junction and the surrounding air, θ_{JB} (or θ_{JC}) is the thermal resistance of the package chosen, and θ_{BA} is the thermal resistance through the PC board, copper traces, and other materials to the surrounding air.

The 8-pin small-outline package for the MAX882/MAX883/MAX884 features a special lead frame with a lower thermal resistance and higher allowable power dissipation. This package's thermal resistance package is θ_{JB} = 53°C/W, compared with θ_{JB} = 110°C/W for an 8-pin plastic DIP package and θ_{JB} = 125°C/W for an 8-pin ceramic DIP package.

5V/3.3V or Adjustable, Low-Dropout, Low IQ, 200mA Linear Regulators

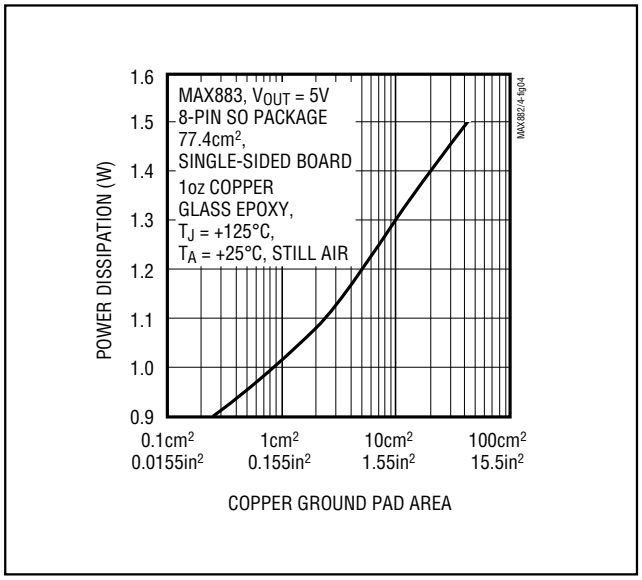


Figure 4. Typical Maximum Power Dissipation vs. Ground Pad Area

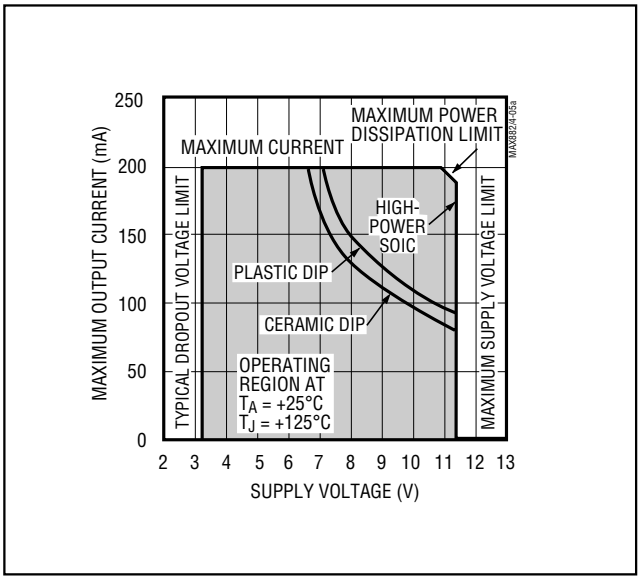


Figure 5a. Safe Operating Regions: MAX882/MAX884 Maximum Output Current vs. Supply Voltage

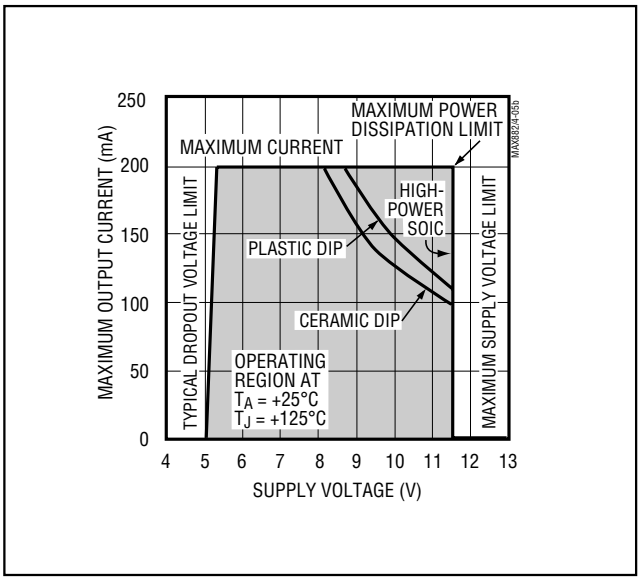


Figure 5b. Safe Operating Regions: MAX883 Maximum Output Current vs. Supply Voltage

The GND pins of the MAX882/MAX883/MAX884 SOIC package perform the dual function of providing an electrical connection to ground and channeling heat away. Connect all GND pins to ground using a large pad or ground plane. Where this is impossible, place a copper plane on an adjacent layer. For a given power dissipation, the pad should exceed the associated dimensions in Figure 4.

Figure 4 assumes the IC is in an 8-pin small-outline package that has a maximum junction temperature of +125°C and is soldered directly to the pad; it also has a +25°C ambient air temperature and no other heat sources. Use larger pad sizes for other packages, lower junction temperatures, higher ambient temperatures, or conditions where the IC is not soldered directly to the heat-sinking ground pad. When operating C- and E-grade parts up to a T_J of +125°C, expect performance similar to M-grade specifications. For T_J between +125°C and +150°C, the output voltage may drift more.

The MAX882/MAX883/MAX884 can regulate currents up to 250mA and operate with input voltages up to 11.5V, but not simultaneously. High output currents can only be sustained when input-output differential voltages are small, as shown in Figure 5. Maximum power dissipation depends on packaging, temperature, and air flow. The maximum output current is as follows:

$$I_{OUT(MAX)} = \frac{P(T_J - T_A)}{(V_{IN} - V_{OUT})100°C}$$

where P is derived from Figure 4.

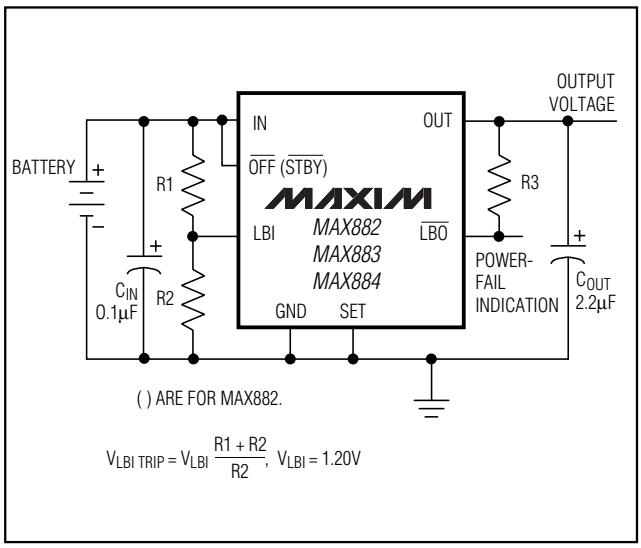


Figure 6. Using the Low-Battery Comparator to Monitor Battery Voltage

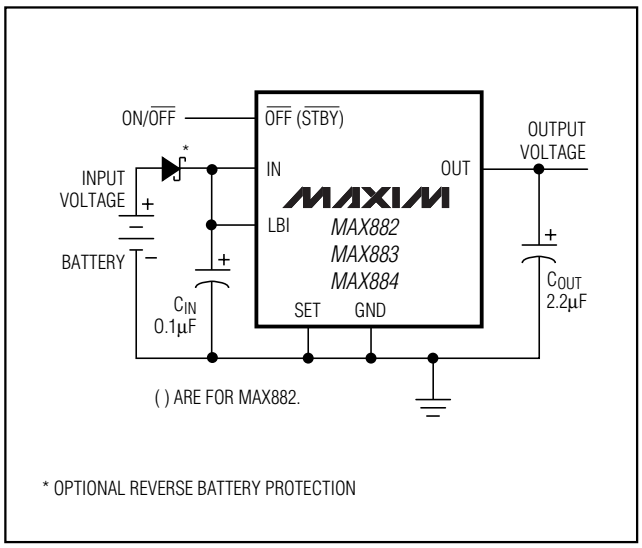


Figure 7. Typical 3.3V or 5V Linear Regulator Circuit

Reverse-Current Protection

The MAX882/MAX883/MAX884 have a unique protection scheme that limits reverse currents when the input voltage falls below the output. It monitors the voltages on IN and OUT and switches the IC's substrate and power bus to the more positive of the two. The control circuitry is then able to remain functioning and turn the pass transistor off, limiting reverse currents back through to the input of the device. In this mode, typical current into OUT to GND is 15µA at V_{OUT} = 3.3V and 50µA at V_{OUT} = 5V.

Reverse-current protection activates when the voltage on IN falls 6mV (or 20mV max) below the voltage on OUT. Before this happens, currents as high as several milliamperes can flow back through the device.

Low-Battery-Detection Comparator

The MAX882/MAX883/MAX884 provide a low-battery comparator that compares the voltage on the LBI pin to the 1.20V internal reference. $\overline{LBO}$, an open-drain output, goes low when LBI is below 1.20V. Hysteresis of 7mV has been added to the low-battery comparator to provide noise immunity during switching. $\overline{LBO}$ remains functional in standby mode for the MAX882, but is undefined in OFF mode for the MAX883 and MAX884. Tie LBI to IN when not used.

Use a resistor-divider network as shown in Figure 6 to set the low-battery trip voltage. Current into the LBI input is ±50nA (max), so R2 can be as large as 1MΩ. Add extra noise immunity by connecting a small capacitor from LBI to GND. Additional hysteresis can be added by connecting a high-value resistor from LBI to $\overline{LBO}$.

Applications Information

The MAX882/MAX883/MAX884 are series linear regulators designed primarily for battery-powered systems. Figure 7 shows a typical application.

Standby Mode vs. OFF Mode

STBY is a comparator input that allows the user to set the standby-mode threshold voltage, while OFF is a logic-level input. When in standby mode, the output is disconnected from the input, but the biasing circuitry (including the low-battery comparator) is kept alive, causing the device to draw approximately 7µA. Standby mode is useful in applications where a low-battery comparator function is still needed in shutdown.

A logic low at the $\overline{OFF}$ pin turns off all biasing circuitry, including the LBI/$\overline{LBO}$ comparator, and reduces supply current to less than 1µA. OFF mode is useful for maximizing battery life. There is little difference in the time it takes to exit standby mode or OFF mode.

Output Capacitor Selection and Regulator Stability

An output filter capacitor is required at the MAX882/MAX883/MAX884 OUT pin. The minimum output capacitance required for stability is 2.2µF.

5V/3.3V or Adjustable, Low-Dropout, Low IQ, 200mA Linear Regulators

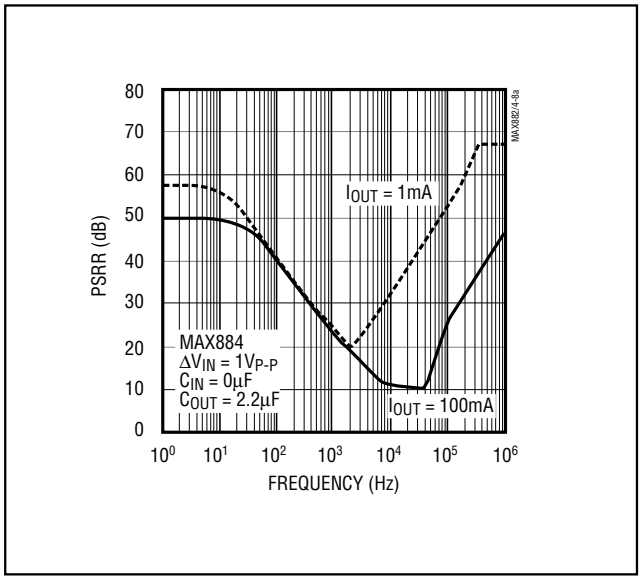


Figure 8a. Power-Supply Rejection Ratio vs. Ripple Frequency for Light and Heavy Loads

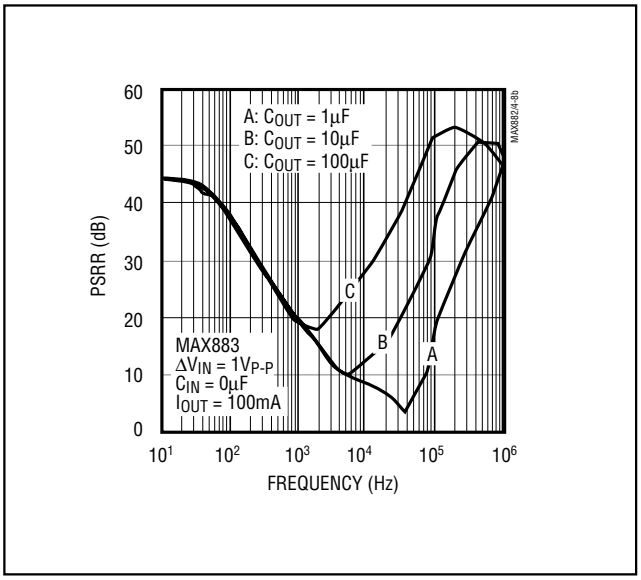


Figure 8b. Power-Supply Rejection Ratio vs. Ripple Frequency for Various Output Capacitances

The filter capacitor's size depends primarily on the desired power-up time and load-transient responses. Load-transient response is improved by using larger output capacitors.

The output capacitor's equivalent series resistance (ESR) will not affect stability as long as the minimum capacitance requirement is observed. The type of capacitor selected is not critical, but it must remain above the minimum value over the full operating temperature range.

Input Bypass Capacitor

Normally, use 0.1µF to 10µF capacitors on the MAX882/MAX883/MAX884 input. The best value depends primarily on the power-up slew rate of V_{IN}, and on load and line transients. Larger input capacitor values provide better supply-noise rejection and line-transient response, as well as improved performance, when the supply has a high AC impedance. The type of input bypass capacitor used is not critical.

Noise

The MAX882/MAX883/MAX884 exhibit up to $4mV_{p-p}$ of noise during normal operation. This is negligible in most applications. When using the MAX882/MAX883/MAX884 for applications that include analog-to-digital converters (ADCs) with resolutions greater than 12 bits, consider the ADC's power-supply rejection specifications. See the output noise plot in the *Typical Operating Characteristics* section.

PSRR and Operation from Sources Other than Batteries

The MAX882/MAX883/MAX884 are designed to achieve low dropout voltages and low quiescent currents in battery-powered systems. However, to gain these benefits, the devices must trade away power-supply noise rejection, as well as swift response to supply variations and load transients. For a 1mA load current, power-supply rejection ranges from 60dB down to 20dB at 2kHz. At higher frequencies, the circuit depends primarily on the characteristics of the output capacitor, and the PSRR increases (Figure 8).

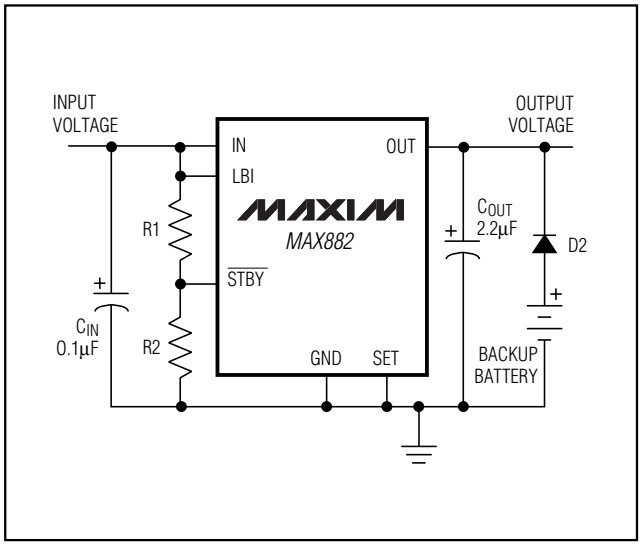


Figure 9. Short-Term Battery Backup Using the MAX882

When operating from sources other than batteries, supply-noise rejection and transient response can be improved by increasing the values of the input and output capacitors and employing passive filtering techniques. Do not use power supplies with ripple voltage exceeding 200mV at 100kHz.

Overshoot and Transient Considerations

The *Typical Operating Characteristics* section shows power-up, supply, and load-transient response graphs. On the load-transient graphs, two components of the output response can be observed: a DC shift from the output impedance due to the different load currents, and the transient response. Typical transients for step changes in the load current from 50mA to 250mA are 200mV. Increasing the output capacitor's value attenuates transient spikes.

During recovery from shutdown, overshoot is negligible if the output voltage has been given time to decay adequately. During power-up from V_{IN} = 0, overshoot is typically less than 1% of V_{OUT}.

Input-Output (Dropout) Voltage

A regulator's minimum input-output voltage differential (or dropout voltage) determines the lowest usable supply voltage. In battery-powered systems, this determines the useful end-of-life battery voltage. Because the MAX882/MAX883/MAX884 use a P-channel MOSFET pass transistor, their dropout voltage is a function of $R_{DS(ON)}$ multiplied by the load current (see *Electrical Characteristics*). Quickly stepping up the input voltage from the dropout voltage can result in overshoot.

Short-Term Battery Backup Using the MAX882

Figure 9 illustrates a scheme for implementing battery backup for 3.3V circuits using the MAX882. When the supply voltage drops below some user-specified value based on resistors R1 and R2, the standby function activates, turning off the MAX882's output. Under these conditions, the backup battery supplies power to the load. Reverse current protection prevents the battery from draining back through the regulator to the input.

This application is limited to short-term battery backup for 3.3V circuits. The current drawn by the MAX882's OUT pin at 3.3V during reverse-current protection is typically 8µA. It should not be used with the MAX883 and MAX884, since the OFF pin is a logic input, and indeterminate inputs can cause the regulator to turn on intermittently, draining the battery.

Reverse Battery Protection

Reverse battery protection can be added by including an inexpensive Schottky diode between the battery input and the regulator circuit, as shown in Figure 7. However, the dropout voltage of the regulator will be increased by the forward voltage drop of the diode. For example, the forward voltage of a standard 1N5817 Schottky diode is typically 0.29V at 200mA.

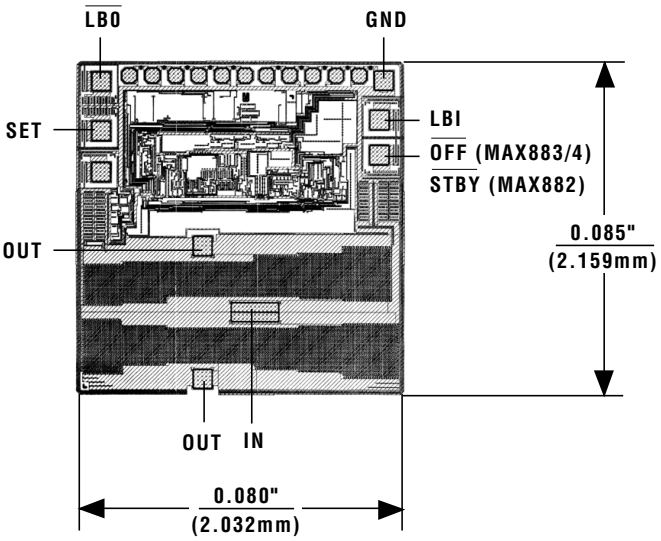
5V/3.3V or Adjustable, Low-Dropout, Low IQ, 200mA Linear Regulators

Ordering Information (continued)

PART	TEMP. RANGE	PIN-PACKAGE
MAX883CPA	0°C to +70°C	8 Plastic DIP
MAX883CSA	0°C to +70°C	8 SO
MAX883C/D	0°C to +70°C	Dice*
MAX883EPA	-40°C to +85°C	8 Plastic DIP
MAX883ESA	-40°C to +85°C	8 SO
MAX883MJA	-55°C to +125°C	8 CERDIP**
MAX884CPA	0°C to +70°C	8 Plastic DIP
MAX884CSA	0°C to +70°C	8 SO
MAX884C/D	0°C to +70°C	Dice*
MAX884EPA	-40°C to +85°C	8 Plastic DIP
MAX884ESA	-40°C to +85°C	8 SO
MAX884MJA	-55°C to +125°C	8 CERDIP**

** Dice are tested at T_J = +25°C, DC parameters only.*
*** Contact factory for availability.*

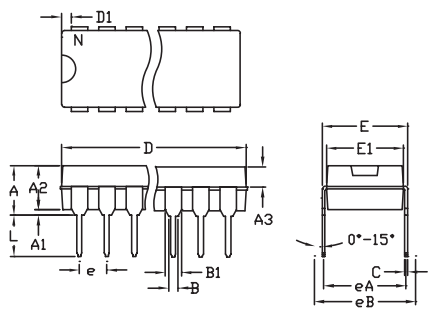
Chip Topography



TRANSISTOR COUNT: 151

NO DIRECT SUBSTRATE CONNECTION. THE N-SUBSTRATE IS INTERNALLY SWITCHED BETWEEN THE MORE POSITIVE OF IN OR OUT.

PDIPN.EPS

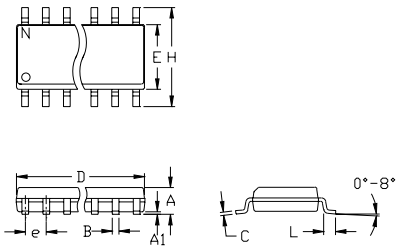


	INCHES		MILLIMETERS	
	MIN	MAX	MIN	MAX
A	---	0.180	---	4.572
A1	0.020	---	0.508	---
A2	0.125	0.175	3.18	4.45
A3	0.055	0.080	1.40	2.03
B	0.015	0.021	0.381	0.533
B1	0.045	0.060	1.14	1.524
C	0.009	0.014	0.229	0.355
D1	0.005	0.080	0.13	2.03
E	0.300	0.325	7.62	8.255
E1	0.275	0.295	6.985	7.493
e	0.100	---	2.54	---
eA	0.300	---	7.62	---
eB	---	0.400	---	10.16
L	0.115	0.150	2.921	3.81

	INCHES		MILLIMETERS			
	MIN	MAX	MIN	MAX	N	MS001
D	0.348	0.390	8.84	9.91	8	AB
D	0.735	0.765	18.67	19.43	14	AC
D	0.745	0.765	18.92	19.43	16	AA
D	0.885	0.915	22.48	23.24	18	AD
D	1.015	1.045	25.78	26.54	20	AE
D	1.14	1.265	28.96	32.13	24	AF
D	1.360	1.380	34.54	35.05	28	*5

NOTES:
1. D&E DO NOT INCLUDE MOLD FLASH
2. MOLD FLASH OR PROTRUSIONS NOT TO EXCEED .15mm (.006")
3. CONTROLLING DIMENSION: MILLIMETER
4. MEETS JEDEC MS001-XX AS SHOWN IN ABOVE TABLE
5. SIMILIAR TO JEDEC MS-095-AH
6. N = NUMBER OF PINS

MAXIM | PROPRIETARY INFORMATION | TITLE: PACKAGE FAMILY OUTLINE: PDIP .300" | 1/1 | DOCUMENT CONTROL NUMBER: 21-0043 | REV: B



	INCHES		MILLIMETERS	
	MIN	MAX	MIN	MAX
A	0.053	0.069	1.35	1.75
A1	0.004	0.010	0.10	0.25
B	0.014	0.019	0.35	0.49
C	0.007	0.010	0.19	0.25
e	0.050		1.27	
E	0.150	0.157	3.80	4.00
H	0.228	0.244	5.80	6.20
h	0.010	0.020	0.25	0.50
L	0.016	0.050	0.40	1.27

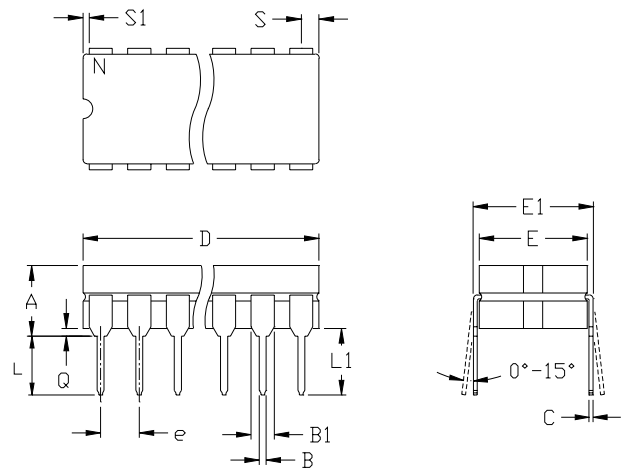
	INCHES		MILLIMETERS			
	MIN	MAX	MIN	MAX	N	MS012
D	0.189	0.197	4.80	5.00	8	A
D	0.337	0.344	8.55	8.75	14	B
D	0.386	0.394	9.80	10.00	16	C

NOTES:
1. D&E DO NOT INCLUDE MOLD FLASH
2. MOLD FLASH OR PROTRUSIONS NOT TO EXCEED .15mm (.006")
3. LEADS TO BE COPLANAR WITHIN .102mm (.004")
4. CONTROLLING DIMENSION: MILLIMETER
5. MEETS JEDEC MS012-XX AS SHOWN IN ABOVE TABLE
6. N = NUMBER OF PINS

MAXIM | PROPRIETARY INFORMATION | TITLE: PACKAGE FAMILY OUTLINE: SOIC .150" | 1/1 | DOCUMENT CONTROL NUMBER: 21-0041 | REV: A

5V/3.3V or Adjustable, Low-Dropout, Low IQ, 200mA Linear Regulators

Package Information (continued)



	INCHES		MILLIMETERS	
	MIN	MAX	MIN	MAX
A	---	0.200	---	5.08
B	0.014	0.023	0.36	0.58
B1	0.038	0.065	0.97	1.65
C	0.008	0.015	0.20	0.38
E	0.220	0.310	5.59	7.87
E1	0.290	0.320	7.37	8.13
e	0.100		2.54	
L	0.125	0.200	3.18	5.08
L1	0.150	---	0.00	---
Q	0.015	0.070	0.38	1.78
S	---	0.098	---	2.49
S1	0.005	---	0.13	---

	INCHES		MILLIMETERS			
	MIN	MAX	MIN	MAX	N	CASE
D	---	0.405	---	10.29	8	P:D4
D	---	0.785	---	19.94	14	C:D1
D	---	0.840	---	21.34	16	E:D2
D	---	0.960	---	24.38	18	V:D6
D	---	1.060	---	26.92	20	R:D8
D	---	1.280	---	32.51	24	L:D9

NOTES:
1. CONTROLLING DIMENSION: INCH
2. MEETS 1835 CASE OUTLINE CONFIGURATION #1 AS SHOWN IN ABOVE TABLE
3. N = NUMBER OF PINS

MAXIM 120 SAN GABRIEL DR SUNNYVALE CA 94086 FAX (408) 737 7194 PROPRIETARY INFORMATION | PACKAGE FAMILY OUTLINE: CDIP .300" TITLE | 1/1 | 21-0045 A DOCUMENT CONTROL NUMBER REV

FPS Series

SD Card and MMC Reader (Manual Insertion, Short Type)

Specifications

Insulation Resistance:	1,000MΩ min. at 500V DC
Withstanding Voltage:	500V AC rms for 1 minute
Contact Resistance:	100mΩ at 1mA / 20mV max.
Voltage Rating:	250Vrms AC
Current Rating:	0.5A
Operating Temp. Range:	–25°C to +85°C
Coplanarity:	≤ 0.1mm
Mating Cycles:	10,000 (office environment)

Part Number (Details)

FPS 009 - 320 *

- Series No.
- Number of Contacts
- Design No.
- 3 Combination Options (see function table below):

Materials and Finish

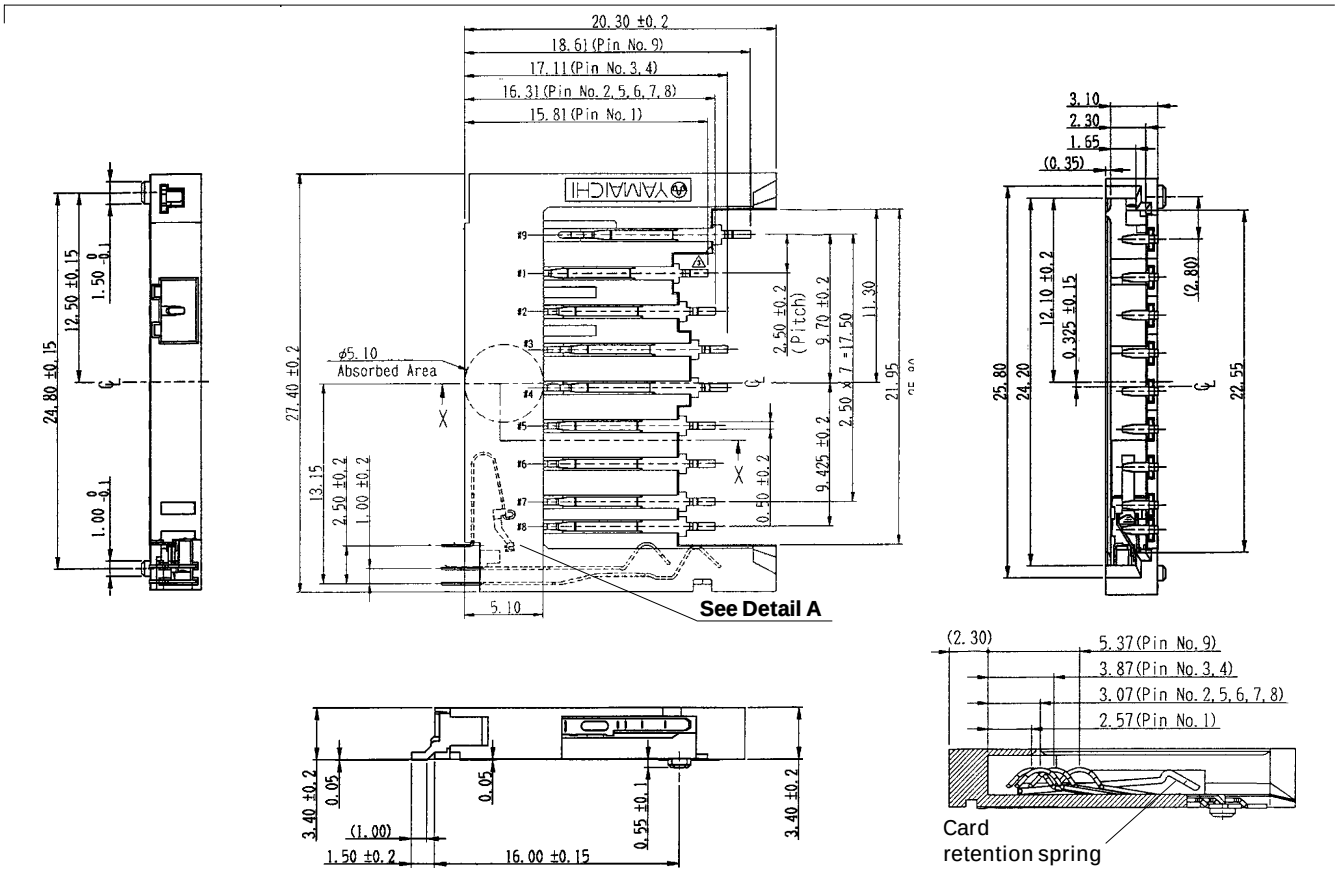
Insulator:	LCP glass filled
Contacts:	PB, t = 0.23
	Contact area = Ni-Au
	Soldertail = Ni-SnPb
Write Protection:	PB, t = 0.18 Ni-Au
Card Detector:	PB, t = 0.15, Ni-Au

Features

- Connector for both card types (SD and MMC card)
- Short housing (20.3mm instead of 23.3mm)
- Card detector and write protection (optional)
- Tape and Reel packaging
- Easy operation by manual insertion / extraction
- 3 step sequential contacting design
- With card retention mechanism

Applicable Dimensions (Reference Only)

Outline Dimensions for FPS009-3202



Section X - X

Function Table

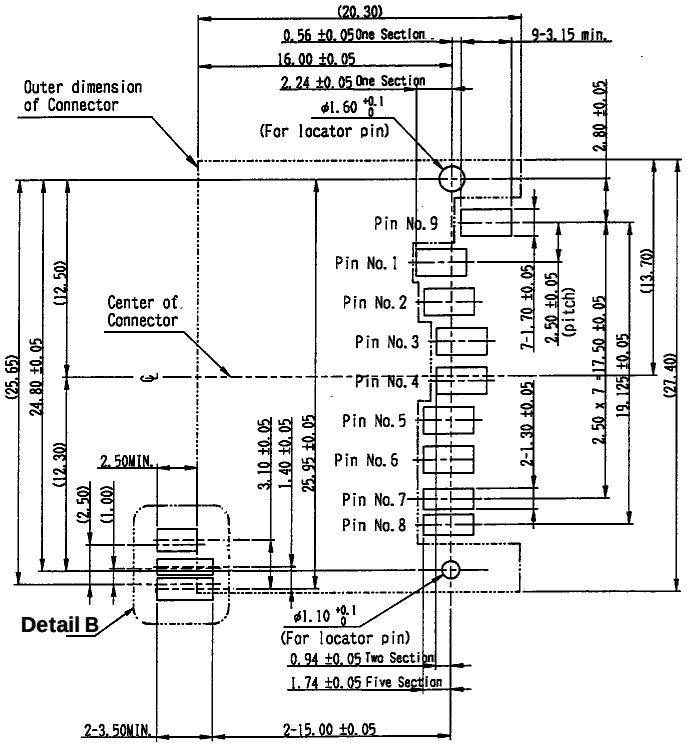
Part Number	Card Detector	Write Protection Switch
FPS009-3202	Yes	Yes
FPS009-3203	No	No
FPS009-3204	No	Yes

Applicable Dimensions (Reference Only)

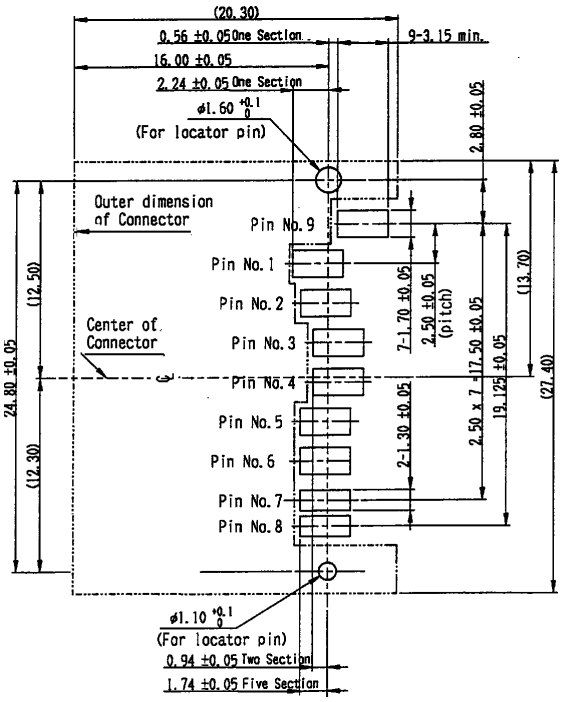
Recommended PCB Layouts

Top View

FPS009-3202 and FPS009-3204

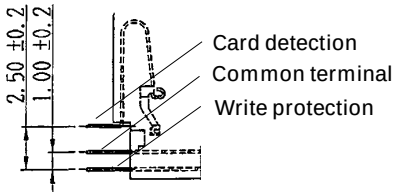


FPS009-3203



Detail A (Function Options)

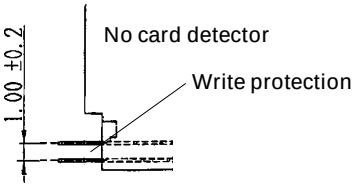
FPS009-3202



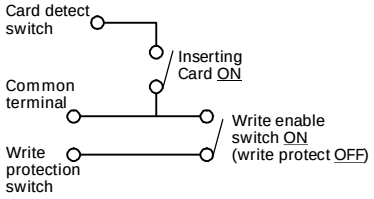
FPS009-3203

No card detector or write protect switch

FPS009-3204

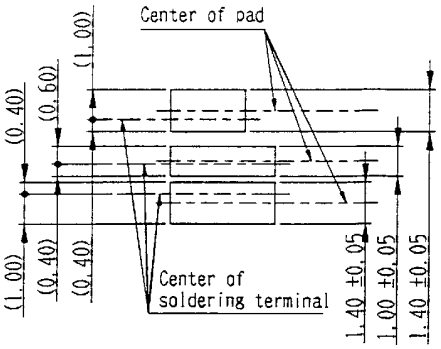


Circuit Diagram for Switch



Detail B (PCB Layout)

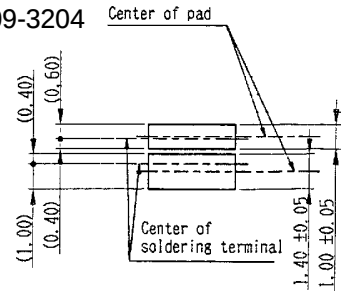
FPS009-3202



FPS009-3203

No soldering terminals with this type

FPS009-3204



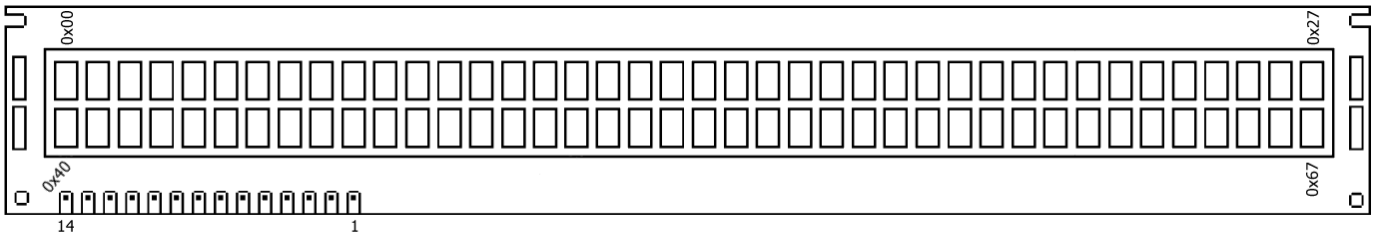
The *Extended Concise* LCD Data Sheet for HD44780

Version: 25.6.1999

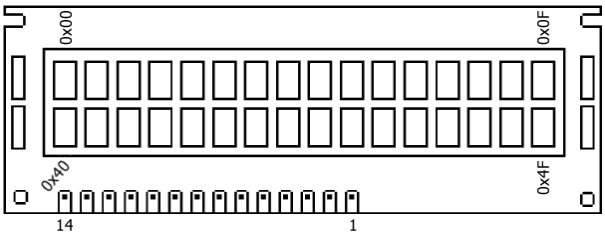
Instruction	RS	RW	D7	D6	D5	D4	D3	D2	D1	D0	Description	Clock-Cycles
NOP	0	0	0	0	0	0	0	0	0	0	No Operation	0
Clear Display	0	0	0	0	0	0	0	0	0	1	Clear display & set address counter to zero	165
Cursor Home	0	0	0	0	0	0	0	0	1	x	Set adress counter to zero, return shifted display to original position. DD RAM contents remains unchanged.	3
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S	Set cursor move direction (I/D) and specify automatic display shift (S).	3
Display Control	0	0	0	0	0	0	1	D	C	B	Turn display (D), cursor on/off (C), and cursor blinking (B).	3
Cursor / Display shift	0	0	0	0	0	1	S/C	R/L	x	x	Shift display or move cursor (S/C) and specify direction (R/L).	3
Function Set	0	0	0	0	1	DL	N	F	x	x	Set interface data width (DL), number of display lines (N) and character font (F).	3
Set CGRAM Address	0	0	0	1	CGRAM Address						Set CGRAM address. CGRAM data is sent afterwards.	3
Set DDRAM Address	0	0	1	DDRAM Address							Set DDRAM address. DDRAM data is sent afterwards.	3
Busy Flag & Address	0	1	BF	Address Counter							Read busy flag (BF) and address counter	0
Write Data	1	0	Data								Write data into DDRAM or CGRAM	3
Read Data	1	1	Data								Read data from DDRAM or CGRAM	3

x : Don't care	I/D	1 0	Increment Decrement	R/L	1 0	Shift to the right Shift to the left
	S	1 0	Automatic display shift	DL	1 0	8 bit interface 4 bit interface
	D	1 0	Display ON Display OFF	N	1 0	2 lines 1 line
	C	1 0	Cursor ON Cursor OFF	F	1 0	5x10 dots 5x7 dots
	B	1 0	Cursor blinking	DDRAM : Display Data RAM CGRAM : Character Generator RAM		
	S/C	1 0	Display shift Cursor move			

LCD Display with 2 lines x 40 characters :



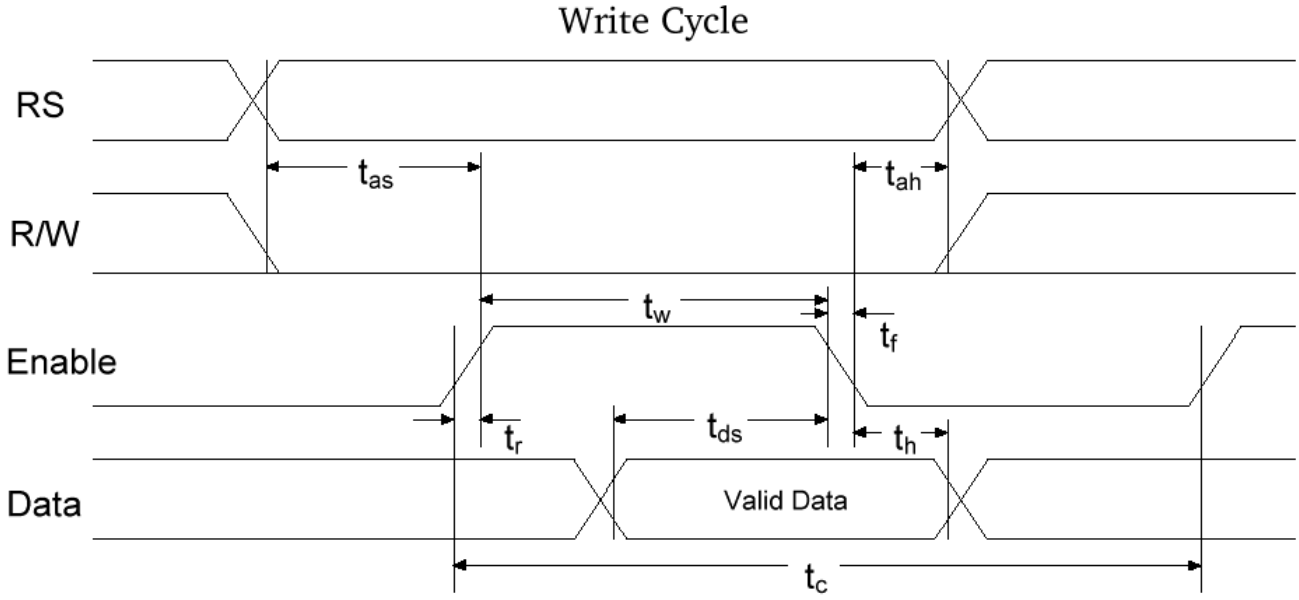
LCD Display with 2 lines x 16 characters :



Pin No	Name	Function	Description
1	Vss	Power	GND
2	Vdd	Power	+ 5 V
3	Vee	Contrast Adj.	(-2) 0 - 5 V
4	RS	Command	Register Select
5	R/W	Command	Read / Write
6	E	Command	Enable (Strobe)
7	D0	I/O	Data LSB
8	D1	I/O	Data
9	D2	I/O	Data
10	D3	I/O	Data
11	D4	I/O	Data
12	D5	I/O	Data
13	D6	I/O	Data
14	D7	I/O	Data MSB

Bus Timing Characteristics

(Ta = - 20 to + 75°C)



Write-Cycle	**V_{DD}**	2.7 - 4.5 V (2)	4.5 - 5.5 V (2)		2.7 - 4.5 V (2)	4.5 - 5.5 V (2)	
Parameter	**Symbol**	**Min(1)**		**Typ(1)**	**Max(1)**		**Unit**
Enable Cycle Time	t_c	1000	500	-	-	-	ns
Enable Pulse Width (High)	t_w	450	230	-	-	-	ns
Enable Rise/Fall Time	t_r, t_f	-	-	-	25	20	ns
Address Setup Time	t_{as}	60	40	-	-	-	ns
Address Hold Time	t_{ah}	20	10	-	-	-	ns
Data Setup Time	t_{ds}	195	80	-	-	-	ns
Data Hold Time	t_h	10	10	-	-	-	ns

(1) *The above specifications are indications only (based on Hitachi HD44780). Timing will vary from manufacturer to manufacturer.*

(2) *Power Supply :* *HD44780 S :* V_{DD} *= 4.5 - 5.5 V*
HD44780 U : V_{DD} *= 2.7 - 5.5 V*

This data sheet refers to specifications for the Hitachi HD44780 LCD Driver chip, which is used for most LCD modules.

Common types are :
1 line x 20 characters
2 lines x 16 characters
2 lines x 20 characters
2 lines x 40 characters
4 lines x 20 characters
4 lines x 40 characters

National Semiconductor

May 2003

LM78LXX Series
3-Terminal Positive Regulators

General Description

The LM78LXX series of three terminal positive regulators is available with several fixed output voltages making them useful in a wide range of applications. When used as a zener diode/resistor combination replacement, the LM78LXX usually results in an effective output impedance improvement of two orders of magnitude, and lower quiescent current. These regulators can provide local on card regulation, eliminating the distribution problems associated with single point regulation. The voltages available allow the LM78LXX to be used in logic systems, instrumentation, HiFi, and other solid state electronic equipment.

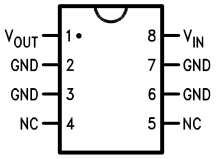
The LM78LXX is available in the plastic TO-92 (Z) package, the plastic SO-8 (M) package and a chip sized package (8-Bump micro SMD) using National's micro SMD package technology. With adequate heat sinking the regulator can deliver 100mA output current. Current limiting is included to limit the peak output current to a safe value. Safe area protection for the output transistors is provided to limit internal power dissipation. If internal power dissipation becomes too high for the heat sinking provided, the thermal shutdown circuit takes over preventing the IC from overheating.

Features

- LM78L05 in micro SMD package
- Output voltage tolerances of ±5% over the temperature range
- Output current of 100mA
- Internal thermal overload protection
- Output transistor safe area protection
- Internal short circuit current limit
- Available in plastic TO-92 and plastic SO-8 low profile packages
- No external components
- Output voltages of 5.0V, 6.2V, 8.2V, 9.0V, 12V, 15V
- See AN-1112 for micro SMD considerations

Connection Diagrams

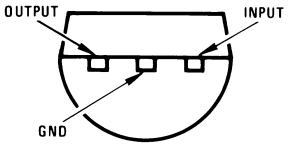
SO-8 Plastic (M)
(Narrow Body)



00774402

Top View

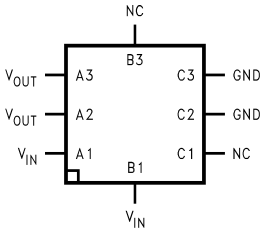
(TO-92)
Plastic Package (Z)



00774403

Bottom View

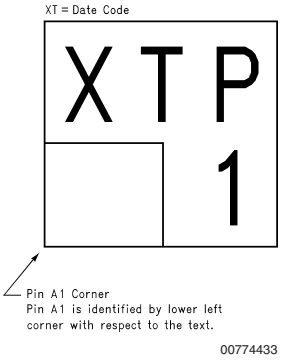
8-Bump micro SMD



00774424

Top View
(Bump Side Down)

micro SMD Marking Orientation



00774433

Top View

Absolute Maximum Ratings (Note 1)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/ Distributors for availability and specifications.

Power Dissipation (Note 5)	Internally Limited
Input Voltage	35V
Storage Temperature	–65˚C to +150˚C
ESD Susceptibility (Note 2)	1kV
Operating Junction Temperature	
SO-8, TO-92	0˚C to 125˚C
micro SMD	–40˚C to 85˚C
Soldering Information	
Infrared or Convection (20 sec.)	235˚C
Wave Soldering (10 sec.)	260˚C (lead time)

LM78LXX Electrical Characteristics

Limits in standard typeface are for T_J = 25˚C, **Bold typeface applies over 0˚C to 125˚C for SO-8 and TO-92 packages, and –40˚C to 85˚C for micro SMD package.** Limits are guaranteed by production testing or correlation techniques using standard Statistical Quality Control (SQC) methods. Unless otherwise specified: I_O = 40mA, C_I = 0.33µF, C_O = 0.1µF.

LM78L05

Unless otherwise specified, V_{IN} = 10V

Symbol	Parameter	Conditions	Min	Typ	Max	Units
V_O	Output Voltage		4.8	5	5.2	V
		7V ≤ V_{IN} ≤ 20V 1mA ≤ I_O ≤ 40mA (Note 3)	**4.75**		**5.25**	
		1mA ≤ I_O ≤ 70mA (Note 3)	**4.75**		**5.25**	
ΔV_O	Line Regulation	7V ≤ V_{IN} ≤ 20V		18	75	mV
		8V ≤ V_{IN} ≤ 20V		10	54	
ΔV_O	Load Regulation	1mA ≤ I_O ≤ 100mA		20	60	
		1mA ≤ I_O ≤ 40mA		5	30	
I_Q	Quiescent Current			3	5	mA
ΔI_Q	Quiescent Current Change	8V ≤ V_{IN} ≤ 20V			**1.0**	
		1mA ≤ I_O ≤ 40mA			**0.1**	
V_n	Output Noise Voltage	f = 10 Hz to 100 kHz (Note 4)		40		µV
$\frac{\Delta V_{IN}}{\Delta V_{OUT}}$	Ripple Rejection	f = 120 Hz 8V ≤ V_{IN} ≤ 16V	47	62		dB
I_{PK}	Peak Output Current			140		mA
$\frac{\Delta V_O}{\Delta T}$	Average Output Voltage Tempco	I_O = 5mA		–0.65		mV/˚C
V_{IN} (Min)	Minimum Value of Input Voltage Required to Maintain Line Regulation			6.7	7	V
θ_{JA}	Thermal Resistance (8-Bump micro SMD)			230.9		˚C/W

LM78L62AC

Unless otherwise specified, V_{IN} = 12V

Symbol	Parameter	Conditions	Min	Typ	Max	Units
V_O	Output Voltage		5.95	6.2	6.45	V
		8.5V ≤ V_{IN} ≤ 20V 1mA ≤ I_O ≤ 40mA (Note 3)	**5.9**		**6.5**	
		1mA ≤ I_O ≤ 70mA (Note 3)	**5.9**		**6.5**	

LM78LXX Electrical Characteristics

Limits in standard typeface are for T_J = 25˚C, **Bold typeface applies over 0˚C to 125˚C for SO-8 and TO-92 packages, and –40˚C to 85˚C for micro SMD package.** Limits are guaranteed by production testing or correlation techniques using standard Statistical Quality Control (SQC) methods. Unless otherwise specified: I_O = 40mA, C_I = 0.33µF, C_O = 0.1µF. (Continued)

LM78L62AC (Continued)

Unless otherwise specified, V_{IN} = 12V

Symbol	Parameter	Conditions	Min	Typ	Max	Units
ΔV_O	Line Regulation	8.5V ≤ V_{IN} ≤ 20V		65	175	mV
		9V ≤ V_{IN} ≤ 20V		55	125	
ΔV_O	Load Regulation	1mA ≤ I_O ≤ 100mA		13	80	
		1mA ≤ I_O ≤ 40mA		6	40	
I_Q	Quiescent Current			2	5.5	mA
ΔI_Q	Quiescent Current Change	8V ≤ V_{IN} ≤ 20V			**1.5**	
		1mA ≤ I_O ≤ 40mA			**0.1**	
V_n	Output Noise Voltage	f = 10 Hz to 100 kHz (Note 4)		50		µV
$\frac{\Delta V_{IN}}{\Delta V_{OUT}}$	Ripple Rejection	f = 120 Hz 10V ≤ V_{IN} ≤ 20V	40	46		dB
I_{PK}	Peak Output Current			140		mA
$\frac{\Delta V_O}{\Delta T}$	Average Output Voltage Tempco	I_O = 5mA		–0.75		mV/˚C
V_{IN} (Min)	Minimum Value of Input Voltage Required to Maintain Line Regulation			7.9		V

LM78L82AC

Unless otherwise specified, V_{IN} = 14V

Symbol	Parameter	Conditions	Min	Typ	Max	Units
V_O	Output Voltage		7.87	8.2	8.53	V
		11V ≤ V_{IN} ≤ 23V 1mA ≤ I_O ≤ 40mA (Note 3)	**7.8**		**8.6**	
		1mA ≤ I_O ≤ 70mA (Note 3)	**7.8**		**8.6**	
ΔV_O	Line Regulation	11V ≤ V_{IN} ≤ 23V		80	175	mV
		12V ≤ V_{IN} ≤ 23V		70	125	
ΔV_O	Load Regulation	1mA ≤ I_O ≤ 100mA		15	80	
		1mA ≤ I_O ≤ 40mA		8	40	
I_Q	Quiescent Current			2	5.5	mA
ΔI_Q	Quiescent Current Change	12V ≤ V_{IN} ≤ 23V			**1.5**	
		1mA ≤ I_O ≤ 40mA			**0.1**	
V_n	Output Noise Voltage	f = 10 Hz to 100 kHz (Note 4)		60		µV
$\frac{\Delta V_{IN}}{\Delta V_{OUT}}$	Ripple Rejection	f = 120 Hz 12V ≤ V_{IN} ≤ 22V	39	45		dB
I_{PK}	Peak Output Current			140		mA
$\frac{\Delta V_O}{\Delta T}$	Average Output Voltage Tempco	I_O = 5mA		–0.8		mV/˚C
V_{IN} (Min)	Minimum Value of Input Voltage Required to Maintain Line Regulation			9.9		V

LM78LXX Electrical Characteristics

Limits in standard typeface are for T_J = 25˚C, **Bold typeface applies over 0˚C to 125˚C for SO-8 and TO-92 packages, and –40˚C to 85˚C for micro SMD package.** Limits are guaranteed by production testing or correlation techniques using standard Statistical Quality Control (SQC) methods. Unless otherwise specified: I_O = 40mA, C_I = 0.33µF, C_O = 0.1µF. (Continued)

LM78L09AC

Unless otherwise specified, V_{IN} = 15V

Symbol	Parameter	Conditions	Min	Typ	Max	Units
V_O	Output Voltage		8.64	9.0	9.36	V
		11.5V ≤ V_{IN} ≤ 24V 1mA ≤ I_O ≤ 40mA (Note 3)	**8.55**		**9.45**	
		1mA ≤ I_O ≤ 70mA (Note 3)	**8.55**		**9.45**	
ΔV_O	Line Regulation	11.5V ≤ V_{IN} ≤ 24V		100	200	mV
		13V ≤ V_{IN} ≤ 24V		90	150	
ΔV_O	Load Regulation	1mA ≤ I_O ≤ 100mA		20	90	
		1mA ≤ I_O ≤ 40mA		10	45	
I_Q	Quiescent Current			2	5.5	mA
ΔI_Q	Quiescent Current Change	11.5V ≤ V_{IN} ≤ 24V			**1.5**	
		1mA ≤ I_O ≤ 40mA			**0.1**	
V_n	Output Noise Voltage			70		µV
$\frac{\Delta V_{IN}}{\Delta V_{OUT}}$	Ripple Rejection	f = 120 Hz 15V ≤ V_{IN} ≤ 25V	38	44		dB
I_{PK}	Peak Output Current			140		mA
$\frac{\Delta V_O}{\Delta T}$	Average Output Voltage Tempco	I_O = 5mA		–0.9		mV/˚C
V_{IN} (Min)	Minimum Value of Input Voltage Required to Maintain Line Regulation			10.7		V

LM78L12AC

Unless otherwise specified, V_{IN} = 19V

Symbol	Parameter	Conditions	Min	Typ	Max	Units
V_O	Output Voltage		11.5	12	12.5	V
		14.5V ≤ V_{IN} ≤ 27V 1mA ≤ I_O ≤ 40mA (Note 3)	**11.4**		**12.6**	
		1mA ≤ I_O ≤ 70mA (Note 3)	**11.4**		**12.6**	
ΔV_O	Line Regulation	14.5V ≤ V_{IN} ≤ 27V		30	180	mV
		16V ≤ V_{IN} ≤ 27V		20	110	
ΔV_O	Load Regulation	1mA ≤ I_O ≤ 100mA		30	100	
		1mA ≤ I_O ≤ 40mA		10	50	
I_Q	Quiescent Current			3	5	mA
ΔI_Q	Quiescent Current Change	16V ≤ V_{IN} ≤ 27V			**1**	
		1mA ≤ I_O ≤ 40mA			**0.1**	
V_n	Output Noise Voltage			80		µV
$\frac{\Delta V_{IN}}{\Delta V_{OUT}}$	Ripple Rejection	f = 120 Hz 15V ≤ V_{IN} ≤ 25	40	54		dB
I_{PK}	Peak Output Current			140		mA
$\frac{\Delta V_O}{\Delta T}$	Average Output Voltage Tempco	I_O = 5mA		–1.0		mV/˚C

LM78LXX Electrical Characteristics

Limits in standard typeface are for T_J = 25˚C, **Bold typeface applies over 0˚C to 125˚C for SO-8 and TO-92 packages, and –40˚C to 85˚C for micro SMD package.** Limits are guaranteed by production testing or correlation techniques using standard Statistical Quality Control (SQC) methods. Unless otherwise specified: I_O = 40mA, C_I = 0.33µF, C_O = 0.1µF. (Continued)

LM78L12AC (Continued)

Unless otherwise specified, V_{IN} = 19V

Symbol	Parameter	Conditions	Min	Typ	Max	Units
V_{IN} (Min)	Minimum Value of Input Voltage Required to Maintain Line Regulation			13.7	14.5	V

LM78L15AC

Unless otherwise specified, V_{IN} = 23V

Symbol	Parameter	Conditions	Min	Typ	Max	Units
V_O	Output Voltage		14.4	15.0	15.6	V
		17.5V ≤ V_{IN} ≤ 30V 1mA ≤ I_O ≤ 40mA (Note 3)	**14.25**		**15.75**	V
		1mA ≤ I_O ≤ 70mA (Note 3)	**14.25**		**15.75**	V
ΔV_O	Line Regulation	17.5V ≤ V_{IN} ≤ 30V		37	250	mV
		20V ≤ V_{IN} ≤ 30V		25	140	mV
ΔV_O	Load Regulation	1mA ≤ I_O ≤ 100mA		35	150	mV
		1mA ≤ I_O ≤ 40mA		12	75	mV
I_Q	Quiescent Current			3	5	mA
ΔI_Q	Quiescent Current Change	20V ≤ V_{IN} ≤ 30V			**1**	mA
		1mA ≤ I_O ≤ 40mA			**0.1**	mA
V_n	Output Noise Voltage			90		µV
$\frac{\Delta V_{IN}}{\Delta V_{OUT}}$	Ripple Rejection	f = 120 Hz 18.5V ≤ V_{IN} ≤ 28.5V	37	51		dB
I_{PK}	Peak Output Current			140		mA
$\frac{\Delta V_O}{\Delta T}$	Average Output Voltage Tempco	I_O = 5mA		–1.3		mV/˚C
V_{IN} (Min)	Minimum Value of Input Voltage Required to Maintain Line Regulation			16.7	17.5	V

Note 1: Absolute Maximum Ratings indicate limits beyond which damage to the device may occur. Electrical specifications do not apply when operating the device outside of its stated operating conditions.

Note 2: Human body model, 1.5 kΩ in series with 100pF.

Note 3: Power dissipation ≤ 0.75W.

Note 4: Recommended minimum load capacitance of 0.01µF to limit high frequency noise.

Note 5: Typical thermal resistance values for the packages are:

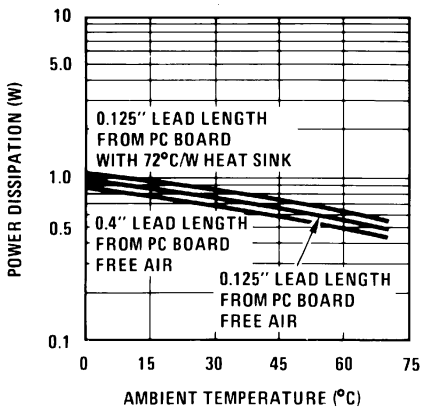
Z Package: θ_{JC} = 60 ˚C/W, = θ_{JA} = 230 ˚C/W

M Package: θ_{JA} = 180 ˚C/W

micro SMD Package: θ_{JA} = 230.9˚C/W

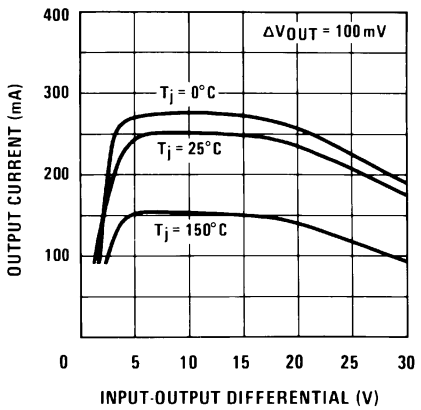
Typical Performance Characteristics

Maximum Average Power Dissipation (Z Package)



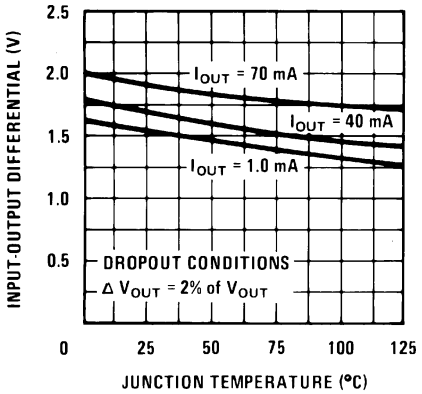
00774414

Peak Output Current



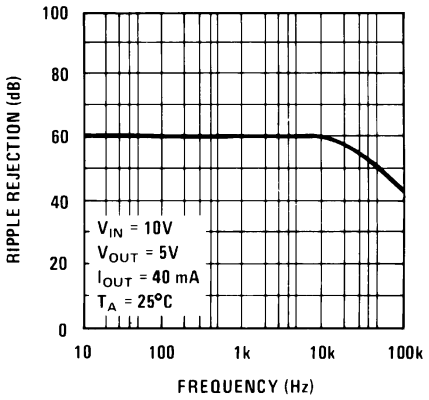
00774416

Dropout Voltage



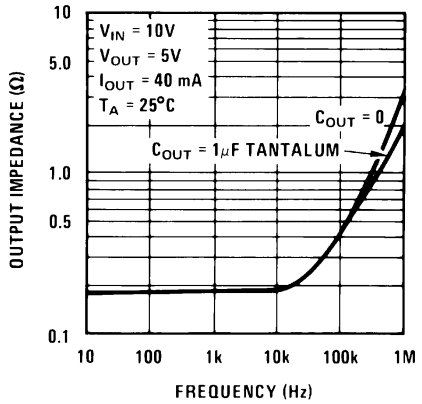
00774417

Ripple Rejection



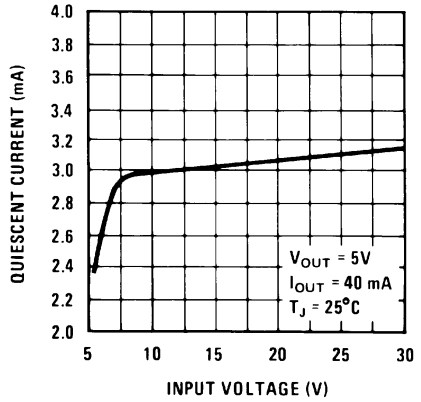
00774418

Output Impedance



00774419

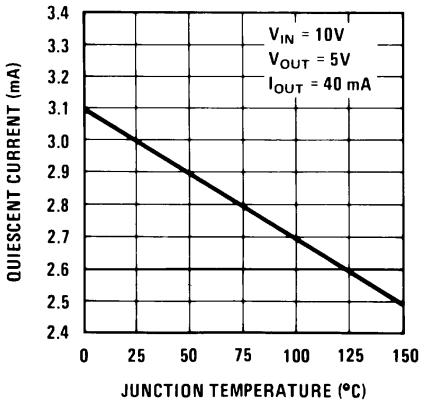
Quiescent Current



00774420

Typical Performance Characteristics (Continued)

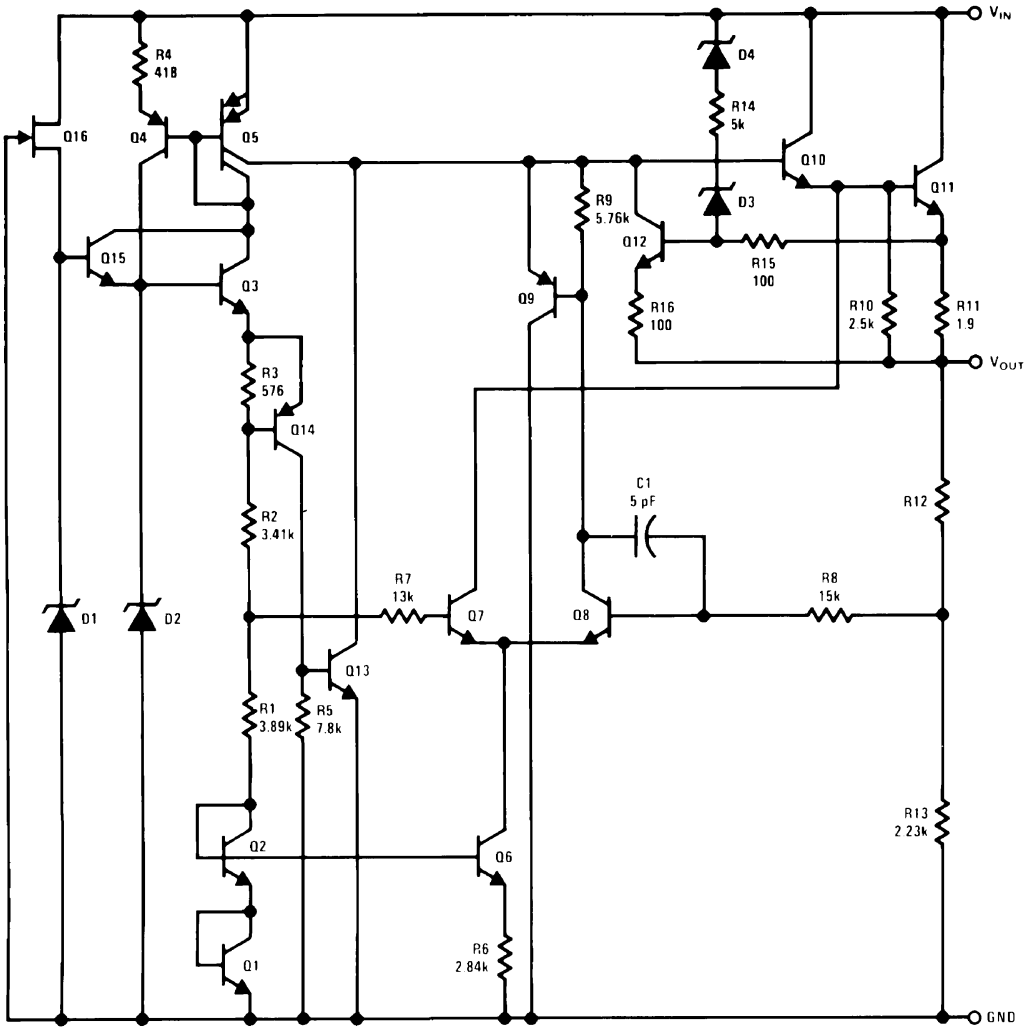
Quiescent Current



00774421

Equivalent Circuit

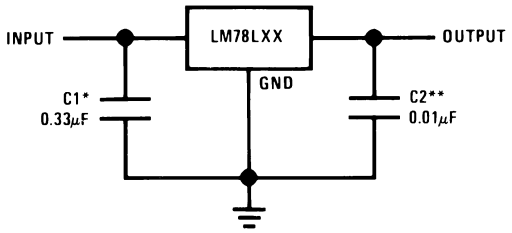
LM78LXX



00774407

Typical Applications

Fixed Output Regulator

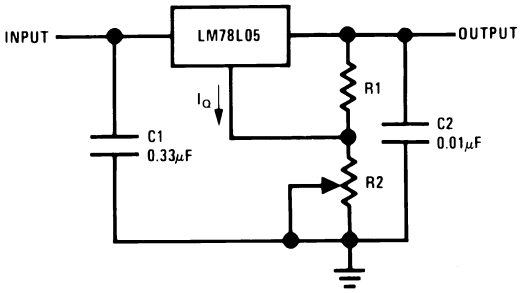


00774408

*Required if the regulator is located more than 3" from the power supply filter.

**See (Note 4) in the electrical characteristics table.

Adjustable Output Regulator

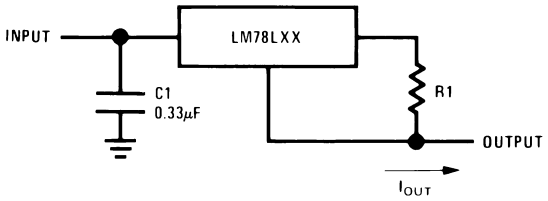


00774409

V_{OUT} = 5V + (5V/R1 + I_Q) R2

5V/R1 > 3 I_Q, load regulation (L_r) ≈ [(R1 + R2)/R1] (L_r of LM78L05)

Current Regulator

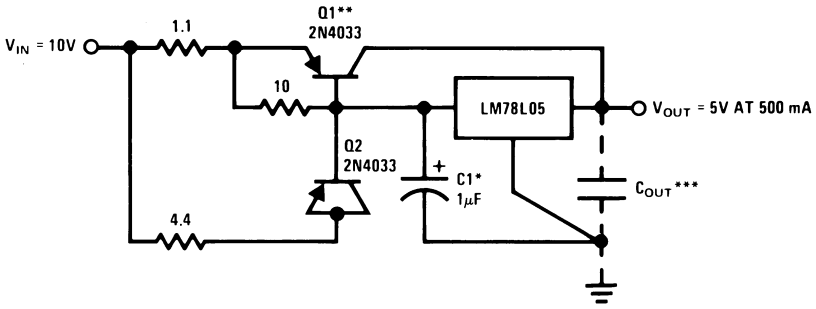


00774410

I_{OUT} = (V_{OUT}/R1) + I_Q

>I_Q = 1.5mA over line and load changes

5V, 500mA Regulator with Short Circuit Protection



00774411

*Solid tantalum.

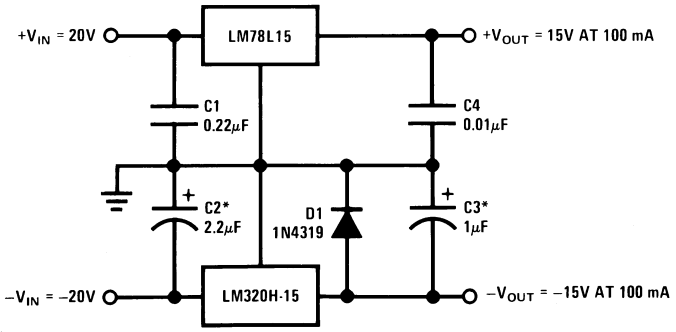
**Heat sink Q1.

***Optional: Improves ripple rejection and transient response.

Load Regulation: 0.6% 0 ≤ I_L ≤ 250mA pulsed with t_{ON} = 50ms.

Typical Applications (Continued)

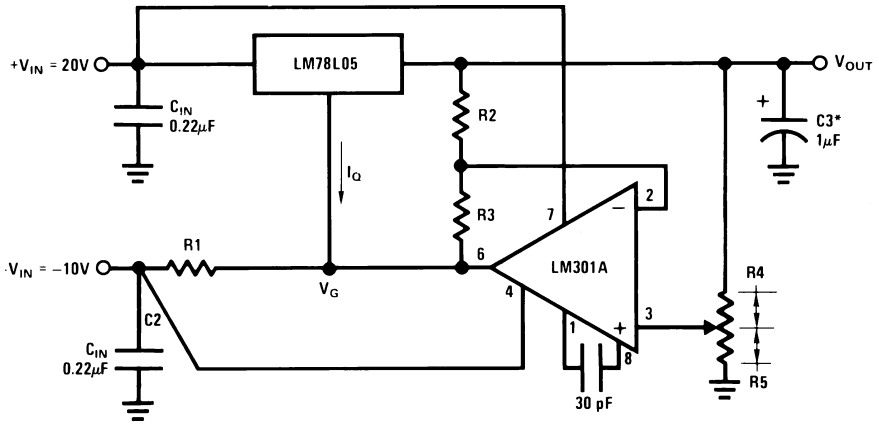
±15V, 100mA Dual Power Supply



00774412

*Solid tantalum.

Variable Output Regulator 0.5V-18V



00774413

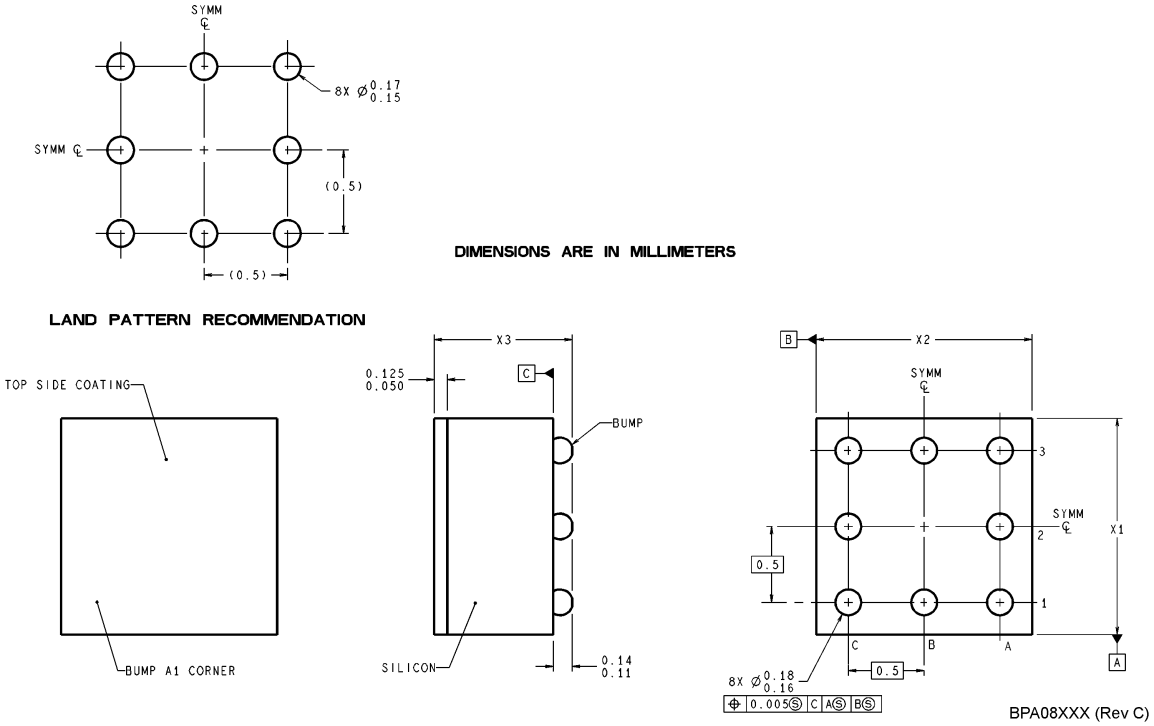
*Solid tantalum.

$V_{OUT} = V_G + 5V$, $R1 = (-V_{IN}/I_{Q\ LM78L05})$

$V_{OUT} = 5V\ (R2/R4)$ for $(R2 + R3) = (R4 + R5)$

A 0.5V output will correspond to $(R2/R4) = 0.1\ (R3/R4) = 0.9$

Physical Dimensions inches (millimeters) unless otherwise noted

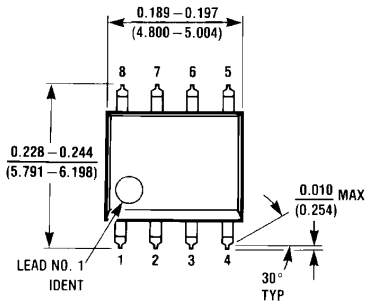


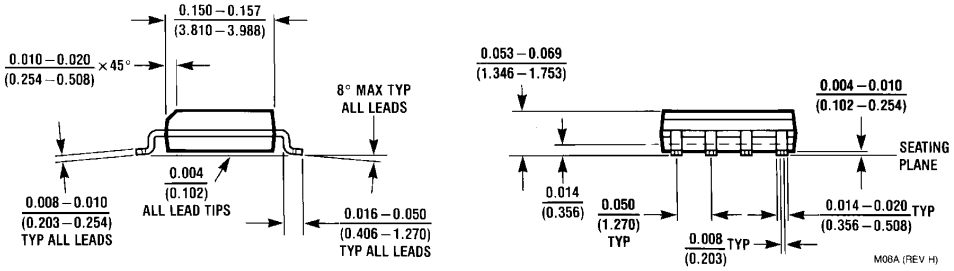
NOTES: UNLESS OTHERWISE SPECIFIED

1. EPOXY COATING
2. 63Sn/37Pb EUTECTIC BUMP
3. RECOMMEND NON-SOLDER MASK DEFINED LANDING PAD.
4. PIN A1 IS ESTABLISHED BY LOWER LEFT CORNER WITH RESPECT TO TEXT ORIENTATION. REMAINING PINS ARE NUMBERED COUNTERCLOCKWISE.
5. XXX IN DRAWING NUMBER REPRESENTS PACKAGE SIZE VARIATION WHERE X_1 IS PACKAGE WIDTH, X_2 IS PACKAGE LENGTH AND X_3 IS PACKAGE HEIGHT.
6. REFERENCE JEDEC REGISTRATION MO-211, VARIATION BC.

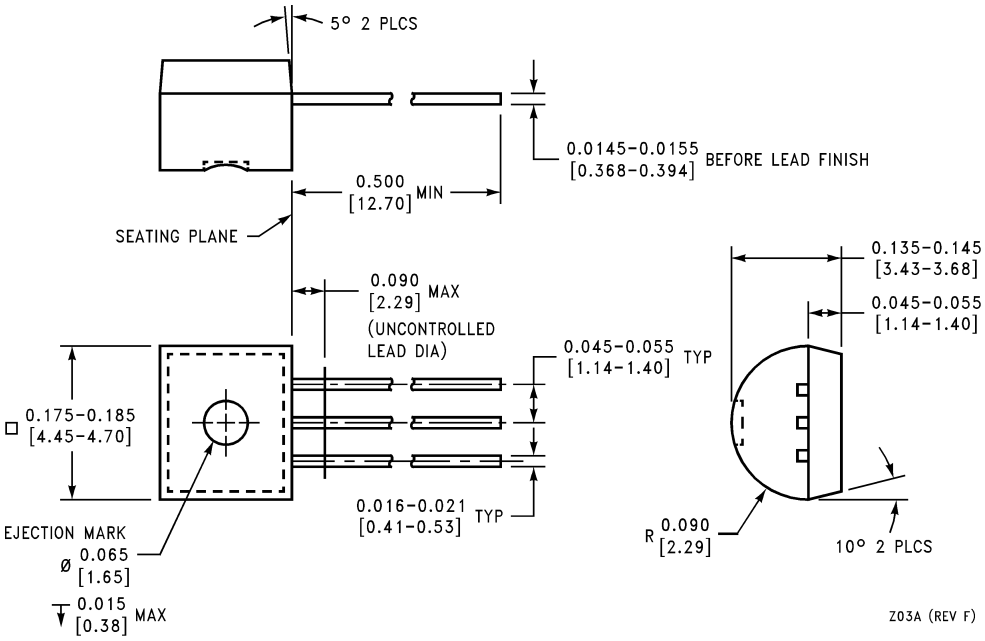
8-Bump micro SMD
Order Number LM78L05IBP or LM78L05IBPX
NS Package Number BPA08AAB
X_1 = 1.285 X_2 = 1.285 X_3 = 0.850

Physical Dimensions inches (millimeters) unless otherwise noted (Continued)





S.O. Package (M)
Order Number LM78L05ACM, LM78L05ACMX, LM78L12ACM, LM78L12ACMX or LM78L15ACM, LM78L15ACMX
NS Package Number M08A



Molded Offset TO-92 (Z)
Order Number LM78L05ACZ, LM78L09ACZ, LM78L12ACZ,
LM78L15ACZ, LM78L62ACZ or LM78L82ACZ
NS Package Number Z03A

Notes

LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT AND GENERAL COUNSEL OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

National Semiconductor
Americas Customer
Support Center
Email: new.feedback@nsc.com
Tel: 1-800-272-9959

www.national.com

National Semiconductor
Europe Customer Support Center
Fax: +49 (0) 180-530 85 86
Email: europe.support@nsc.com
Deutsch Tel: +49 (0) 69 9508 6208
English Tel: +44 (0) 870 24 0 2171
Français Tel: +33 (0) 1 41 91 8790

National Semiconductor
Asia Pacific Customer
Support Center
Email: ap.support@nsc.com

National Semiconductor
Japan Customer Support Center
Fax: 81-3-5639-7507
Email: jpn.feedback@nsc.com
Tel: 81-3-5639-7560

November 2000

National Semiconductor

LM35
Precision Centigrade Temperature Sensors

General Description

The LM35 series are precision integrated-circuit temperature sensors, whose output voltage is linearly proportional to the Celsius (Centigrade) temperature. The LM35 thus has an advantage over linear temperature sensors calibrated in ˚ Kelvin, as the user is not required to subtract a large constant voltage from its output to obtain convenient Centigrade scaling. The LM35 does not require any external calibration or trimming to provide typical accuracies of ±¼˚C at room temperature and ±¾˚C over a full −55 to +150˚C temperature range. Low cost is assured by trimming and calibration at the wafer level. The LM35's low output impedance, linear output, and precise inherent calibration make interfacing to readout or control circuitry especially easy. It can be used with single power supplies, or with plus and minus supplies. As it draws only 60 µA from its supply, it has very low self-heating, less than 0.1˚C in still air. The LM35 is rated to operate over a −55˚ to +150˚C temperature range, while the LM35C is rated for a −40˚ to +110˚C range (−10˚ with improved accuracy). The LM35 series is available packaged in hermetic TO-46 transistor packages, while the LM35C, LM35CA, and LM35D are also available in the plastic TO-92 transistor package. The LM35D is also available in an 8-lead surface mount small outline package and a plastic TO-220 package.

Features

- Calibrated directly in ˚ Celsius (Centigrade)
- Linear + 10.0 mV/˚C scale factor
- 0.5˚C accuracy guaranteeable (at +25˚C)
- Rated for full −55˚ to +150˚C range
- Suitable for remote applications
- Low cost due to wafer-level trimming
- Operates from 4 to 30 volts
- Less than 60 µA current drain
- Low self-heating, 0.08˚C in still air
- Nonlinearity only ±¼˚C typical
- Low impedance output, 0.1 Ω for 1 mA load

Typical Applications

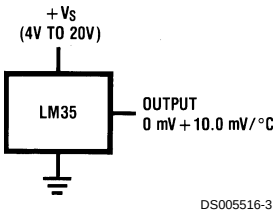
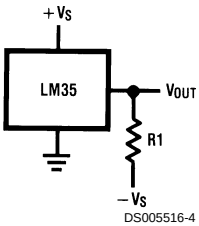


FIGURE 1. Basic Centigrade Temperature Sensor (+2˚C to +150˚C)

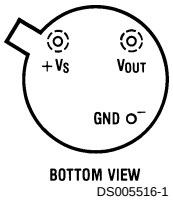


Choose $R_1 = -V_S/50\ \mu A$
V_{OUT}=+1,500 mV at +150˚C
= +250 mV at +25˚C
= −550 mV at −55˚C

FIGURE 2. Full-Range Centigrade Temperature Sensor

Connection Diagrams

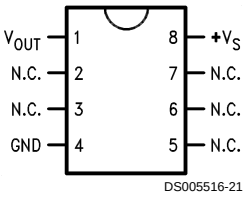
TO-46
Metal Can Package*



*Case is connected to negative pin (GND)

Order Number LM35H, LM35AH, LM35CH, LM35CAH or LM35DH
See NS Package Number H03H

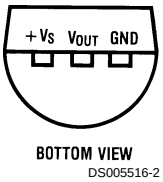
SO-8
Small Outline Molded Package



N.C. = No Connection

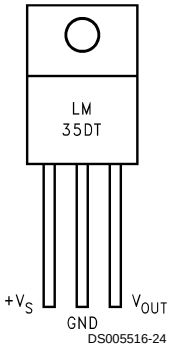
Top View
Order Number LM35DM
See NS Package Number M08A

TO-92
Plastic Package



Order Number LM35CZ, LM35CAZ or LM35DZ
See NS Package Number Z03A

TO-220
Plastic Package*



*Tab is connected to the negative pin (GND).
Note: The LM35DT pinout is different than the discontinued LM35DP.

Order Number LM35DT
See NS Package Number TA03F

Absolute Maximum Ratings (Note 10)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/ Distributors for availability and specifications.

Supply Voltage	+35V to −0.2V
Output Voltage	+6V to −1.0V
Output Current	10 mA
Storage Temp.;	
TO-46 Package,	−60˚C to +180˚C
TO-92 Package,	−60˚C to +150˚C
SO-8 Package,	−65˚C to +150˚C
TO-220 Package,	−65˚C to +150˚C
Lead Temp.:	
TO-46 Package, (Soldering, 10 seconds)	300˚C
TO-92 and TO-220 Package, (Soldering, 10 seconds)	260˚C
SO Package (Note 12)	
Vapor Phase (60 seconds)	215˚C
Infrared (15 seconds)	220˚C
ESD Susceptibility (Note 11)	2500V
Specified Operating Temperature Range: T_{MIN} to T_{MAX} (Note 2)	
LM35, LM35A	−55˚C to +150˚C
LM35C, LM35CA	−40˚C to +110˚C
LM35D	0˚C to +100˚C

Electrical Characteristics

(Notes 1, 6)

Parameter	Conditions	LM35A			LM35CA			Units (Max.)
		Typical	Tested Limit (Note 4)	Design Limit (Note 5)	Typical	Tested Limit (Note 4)	Design Limit (Note 5)	
Accuracy (Note 7)	T_A=+25˚C	±0.2	±0.5		±0.2	±0.5		˚C
	T_A=−10˚C	±0.3			±0.3		±1.0	˚C
	T_A=T_{MAX}	±0.4	±1.0		±0.4	±1.0		˚C
	T_A=T_{MIN}	±0.4	±1.0		±0.4		±1.5	˚C
Nonlinearity (Note 8)	T_{MIN}≤T_A≤T_{MAX}	**±0.18**		**±0.35**	**±0.15**		**±0.3**	˚C
Sensor Gain (Average Slope)	T_{MIN}≤T_A≤T_{MAX}	**+10.0**	**+9.9, +10.1**		**+10.0**		**+9.9, +10.1**	mV/˚C
Load Regulation (Note 3) 0≤I_L≤1 mA	T_A=+25˚C	±0.4	±1.0		±0.4	±1.0		mV/mA
	T_{MIN}≤T_A≤T_{MAX}	**±0.5**		**±3.0**	**±0.5**		**±3.0**	mV/mA
Line Regulation (Note 3)	T_A=+25˚C	±0.01	±0.05		±0.01	±0.05		mV/V
	4V≤V_S≤30V	**±0.02**		**±0.1**	**±0.02**		**±0.1**	mV/V
Quiescent Current (Note 9)	V_S=+5V, +25˚C	56	67		56	67		µA
	V_S=+5V	**105**		**131**	**91**		**114**	µA
	V_S=+30V, +25˚C	56.2	68		56.2	68		µA
	V_S=+30V	**105.5**		**133**	**91.5**		**116**	µA
Change of Quiescent Current (Note 3)	4V≤V_S≤30V, +25˚C	0.2	1.0		0.2	1.0		µA
	4V≤V_S≤30V	**0.5**		**2.0**	**0.5**		**2.0**	µA
Temperature Coefficient of Quiescent Current		**+0.39**		**+0.5**	**+0.39**		**+0.5**	µA/˚C
Minimum Temperature for Rated Accuracy	In circuit of *Figure 1*, I_L=0	+1.5		+2.0	+1.5		+2.0	˚C
Long Term Stability	T_J=T_{MAX}, for 1000 hours	±0.08			±0.08			˚C

Electrical Characteristics

(Notes 1, 6)

Parameter	Conditions	LM35			LM35C, LM35D			Units (Max.)
		Typical	Tested Limit (Note 4)	Design Limit (Note 5)	Typical	Tested Limit (Note 4)	Design Limit (Note 5)	
Accuracy, LM35, LM35C (Note 7)	T_A=+25˚C	±0.4	±1.0		±0.4	±1.0		˚C
	T_A=−10˚C	±0.5			±0.5		±1.5	˚C
	$T_A=T_{MAX}$	±0.8	±1.5		±0.8		±1.5	˚C
	$T_A=T_{MIN}$	±0.8		±1.5	±0.8		±2.0	˚C
Accuracy, LM35D (Note 7)	T_A=+25˚C				±0.6	±1.5		˚C
	$T_A=T_{MAX}$				±0.9		±2.0	˚C
	$T_A=T_{MIN}$				±0.9		±2.0	˚C
Nonlinearity (Note 8)	$T_{MIN}\leq T_A\leq T_{MAX}$	**±0.3**		**±0.5**	**±0.2**		**±0.5**	˚C
Sensor Gain (Average Slope)	$T_{MIN}\leq T_A\leq T_{MAX}$	**+10.0**	**+9.8, +10.2**		**+10.0**		**+9.8, +10.2**	mV/˚C
Load Regulation (Note 3) $0\leq I_L\leq 1$ mA	T_A=+25˚C	±0.4	±2.0		±0.4	±2.0		mV/mA
	$T_{MIN}\leq T_A\leq T_{MAX}$	**±0.5**		**±5.0**	**±0.5**		**±5.0**	mV/mA
Line Regulation (Note 3)	T_A=+25˚C	±0.01	±0.1		±0.01	±0.1		mV/V
	$4V\leq V_S\leq 30V$	**±0.02**		**±0.2**	**±0.02**		**±0.2**	mV/V
Quiescent Current (Note 9)	V_S=+5V, +25˚C	56	80		56	80		µA
	V_S=+5V	**105**		**158**	**91**		**138**	µA
	V_S=+30V, +25˚C	56.2	82		56.2	82		µA
	V_S=+30V	**105.5**		**161**	**91.5**		**141**	µA
Change of Quiescent Current (Note 3)	$4V\leq V_S\leq 30V$, +25˚C	0.2	2.0		0.2	2.0		µA
	$4V\leq V_S\leq 30V$	**0.5**		**3.0**	**0.5**		**3.0**	µA
Temperature Coefficient of Quiescent Current		**+0.39**		**+0.7**	**+0.39**		**+0.7**	µA/˚C
Minimum Temperature for Rated Accuracy	In circuit of *Figure 1*, I_L=0	+1.5		+2.0	+1.5		+2.0	˚C
Long Term Stability	$T_J=T_{MAX}$, for 1000 hours	±0.08			±0.08			˚C

Note 1: Unless otherwise noted, these specifications apply: −55˚C≤T_J≤+150˚C for the LM35 and LM35A; −40˚≤T_J≤+110˚C for the LM35C and LM35CA; and 0˚≤T_J≤+100˚C for the LM35D. V_S=+5Vdc and I_{LOAD}=50 µA, in the circuit of *Figure 2*. These specifications also apply from +2˚C to T_{MAX} in the circuit of *Figure 1*. Specifications in **boldface** apply over the full rated temperature range.

Note 2: Thermal resistance of the TO-46 package is 400˚C/W, junction to ambient, and 24˚C/W junction to case. Thermal resistance of the TO-92 package is 180˚C/W junction to ambient. Thermal resistance of the small outline molded package is 220˚C/W junction to ambient. Thermal resistance of the TO-220 package is 90˚C/W junction to ambient. For additional thermal resistance information see table in the Applications section.

Note 3: Regulation is measured at constant junction temperature, using pulse testing with a low duty cycle. Changes in output due to heating effects can be computed by multiplying the internal dissipation by the thermal resistance.

Note 4: Tested Limits are guaranteed and 100% tested in production.

Note 5: Design Limits are guaranteed (but not 100% production tested) over the indicated temperature and supply voltage ranges. These limits are not used to calculate outgoing quality levels.

Note 6: Specifications in **boldface** apply over the full rated temperature range.

Note 7: Accuracy is defined as the error between the output voltage and 10mv/˚C times the device's case temperature, at specified conditions of voltage, current, and temperature (expressed in ˚C).

Note 8: Nonlinearity is defined as the deviation of the output-voltage-versus-temperature curve from the best-fit straight line, over the device's rated temperature range.

Note 9: Quiescent current is defined in the circuit of *Figure 1*.

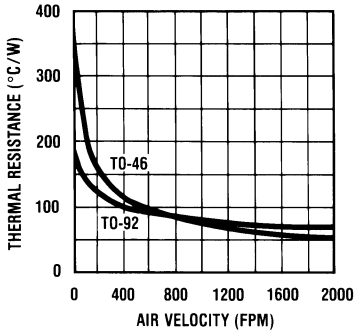
Note 10: Absolute Maximum Ratings indicate limits beyond which damage to the device may occur. DC and AC electrical specifications do not apply when operating the device beyond its rated operating conditions. See Note 1.

Note 11: Human body model, 100 pF discharged through a 1.5 kΩ resistor.

Note 12: See AN-450 "Surface Mounting Methods and Their Effect on Product Reliability" or the section titled "Surface Mount" found in a current National Semiconductor Linear Data Book for other methods of soldering surface mount devices.

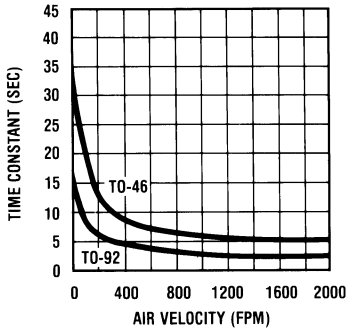
Typical Performance Characteristics

Thermal Resistance Junction to Air



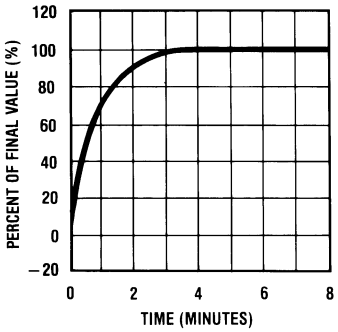
DS005516-25

Thermal Time Constant



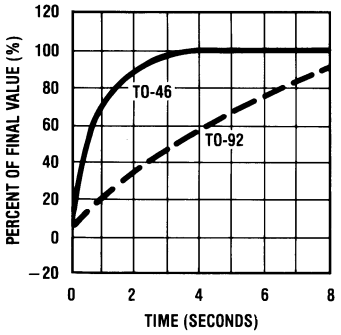
DS005516-26

Thermal Response in Still Air



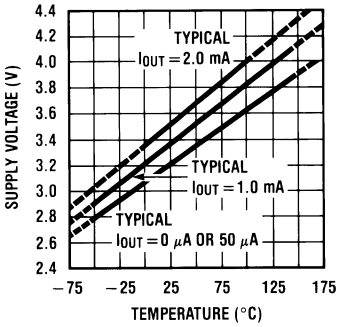
DS005516-27

Thermal Response in Stirred Oil Bath



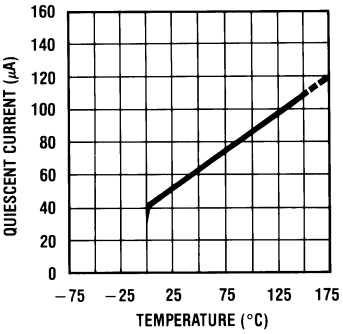
DS005516-28

Minimum Supply Voltage vs. Temperature



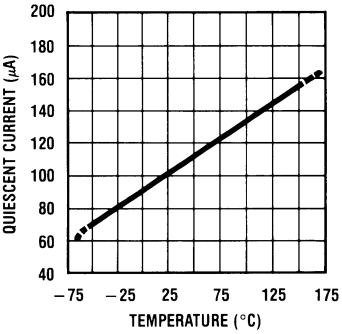
DS005516-29

Quiescent Current vs. Temperature (In Circuit of *Figure 1*.)



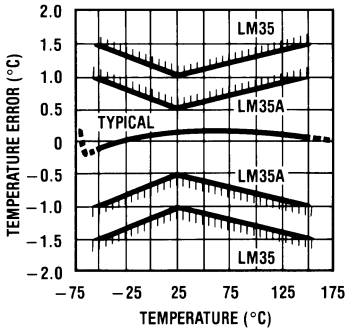
DS005516-30

Quiescent Current vs. Temperature (In Circuit of *Figure 2*.)



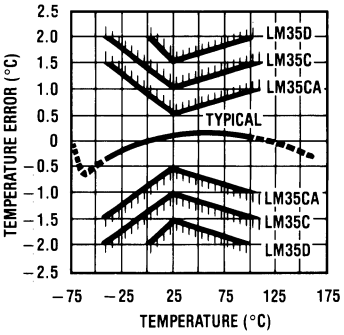
DS005516-31

Accuracy vs. Temperature (Guaranteed)



DS005516-32

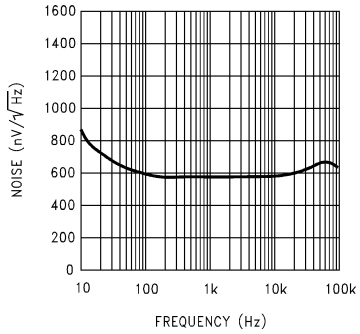
Accuracy vs. Temperature (Guaranteed)



DS005516-33

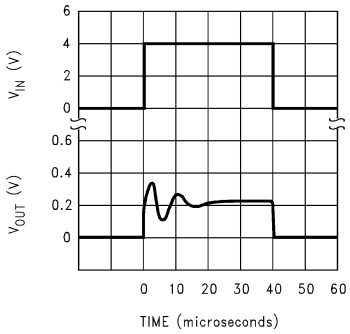
Typical Performance Characteristics (Continued)

Noise Voltage



DS005516-34

Start-Up Response



DS005516-35

Applications

The LM35 can be applied easily in the same way as other integrated-circuit temperature sensors. It can be glued or cemented to a surface and its temperature will be within about 0.01˚C of the surface temperature.

This presumes that the ambient air temperature is almost the same as the surface temperature; if the air temperature were much higher or lower than the surface temperature, the actual temperature of the LM35 die would be at an intermediate temperature between the surface temperature and the air temperature. This is expecially true for the TO-92 plastic package, where the copper leads are the principal thermal path to carry heat into the device, so its temperature might be closer to the air temperature than to the surface temperature.

To minimize this problem, be sure that the wiring to the LM35, as it leaves the device, is held at the same temperature as the surface of interest. The easiest way to do this is to cover up these wires with a bead of epoxy which will insure that the leads and wires are all at the same temperature as the surface, and that the LM35 die's temperature will not be affected by the air temperature.

The TO-46 metal package can also be soldered to a metal surface or pipe without damage. Of course, in that case the V− terminal of the circuit will be grounded to that metal. Alternatively, the LM35 can be mounted inside a sealed-end metal tube, and can then be dipped into a bath or screwed into a threaded hole in a tank. As with any IC, the LM35 and accompanying wiring and circuits must be kept insulated and dry, to avoid leakage and corrosion. This is especially true if the circuit may operate at cold temperatures where condensation can occur. Printed-circuit coatings and varnishes such as Humiseal and epoxy paints or dips are often used to insure that moisture cannot corrode the LM35 or its connections.

These devices are sometimes soldered to a small light-weight heat fin, to decrease the thermal time constant and speed up the response in slowly-moving air. On the other hand, a small thermal mass may be added to the sensor, to give the steadiest reading despite small deviations in the air temperature.

Temperature Rise of LM35 Due To Self-heating (Thermal Resistance, θ_{JA})

	TO-46, no heat sink	TO-46*, small heat fin	TO-92, no heat sink	TO-92**, small heat fin	SO-8 no heat sink	SO-8** small heat fin	TO-220 no heat sink
Still air	400˚C/W	100˚C/W	180˚C/W	140˚C/W	220˚C/W	110˚C/W	90˚C/W
Moving air	100˚C/W	40˚C/W	90˚C/W	70˚C/W	105˚C/W	90˚C/W	26˚C/W
Still oil	100˚C/W	40˚C/W	90˚C/W	70˚C/W			
Stirred oil	50˚C/W	30˚C/W	45˚C/W	40˚C/W			
(Clamped to metal, Infinite heat sink)	(24˚C/W)				(55˚C/W)		

*Wakefield type 201, or 1" disc of 0.020" sheet brass, soldered to case, or similar.
**TO-92 and SO-8 packages glued and leads soldered to 1" square of 1/16" printed circuit board with 2 oz. foil or similar.

Typical Applications

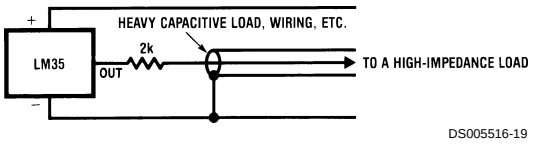


FIGURE 3. LM35 with Decoupling from Capacitive Load

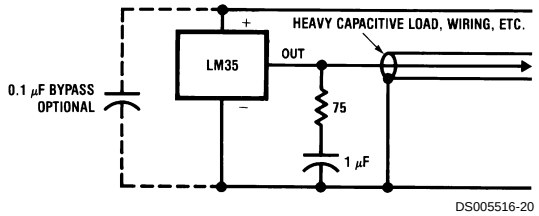


FIGURE 4. LM35 with R-C Damper

CAPACITIVE LOADS

Like most micropower circuits, the LM35 has a limited ability to drive heavy capacitive loads. The LM35 by itself is able to drive 50 pf without special precautions. If heavier loads are anticipated, it is easy to isolate or decouple the load with a resistor; see *Figure 3*. Or you can improve the tolerance of capacitance with a series R-C damper from output to ground; see *Figure 4*.

When the LM35 is applied with a 200Ω load resistor as shown in *Figure 5*, *Figure 6* or *Figure 8* it is relatively immune to wiring capacitance because the capacitance forms a bypass from ground to input, not on the output. However, as with any linear circuit connected to wires in a hostile environment, its performance can be affected adversely by intense electromagnetic sources such as relays, radio transmitters, motors with arcing brushes, SCR transients, etc, as its wiring can act as a receiving antenna and its internal junctions can act as rectifiers. For best results in such cases, a bypass capacitor from V_{IN} to ground and a series R-C damper such as 75Ω in series with 0.2 or 1 µF from output to ground are often useful. These are shown in *Figure 13*, *Figure 14*, and *Figure 16*.

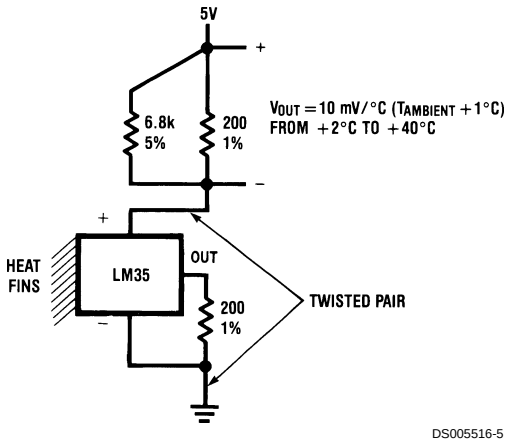


FIGURE 5. Two-Wire Remote Temperature Sensor (Grounded Sensor)

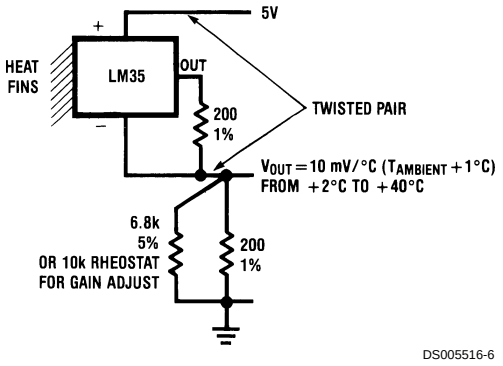


FIGURE 6. Two-Wire Remote Temperature Sensor (Output Referred to Ground)

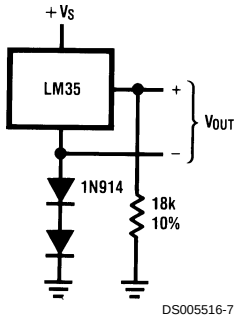


FIGURE 7. Temperature Sensor, Single Supply, −55˚ to +150˚C

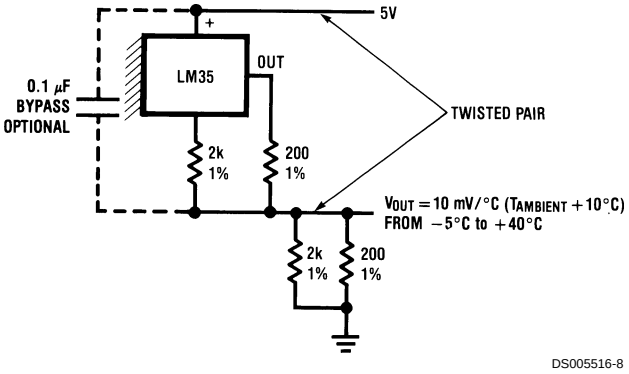


FIGURE 8. Two-Wire Remote Temperature Sensor (Output Referred to Ground)

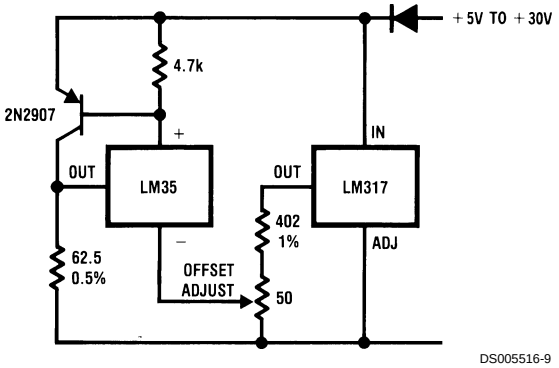


FIGURE 9. 4-To-20 mA Current Source (0˚C to +100˚C)

Typical Applications (Continued)

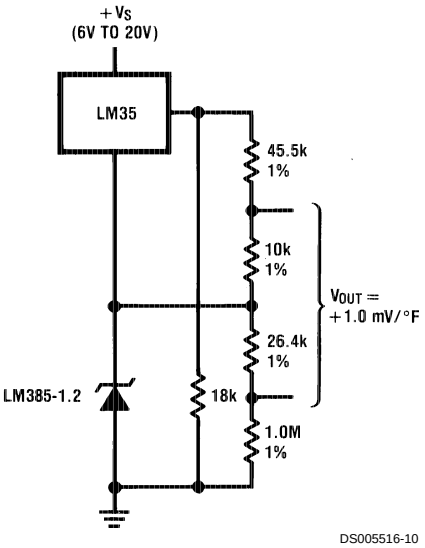


FIGURE 10. Fahrenheit Thermometer

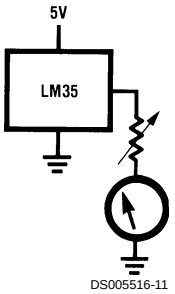


FIGURE 11. Centigrade Thermometer (Analog Meter)

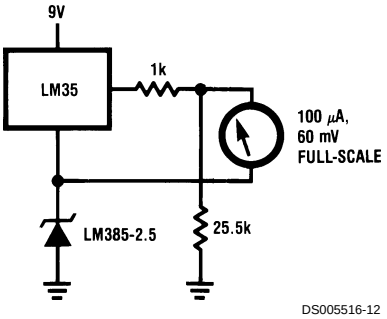


FIGURE 12. Fahrenheit ThermometerExpanded Scale Thermometer
(50˚ to 80˚ Fahrenheit, for Example Shown)

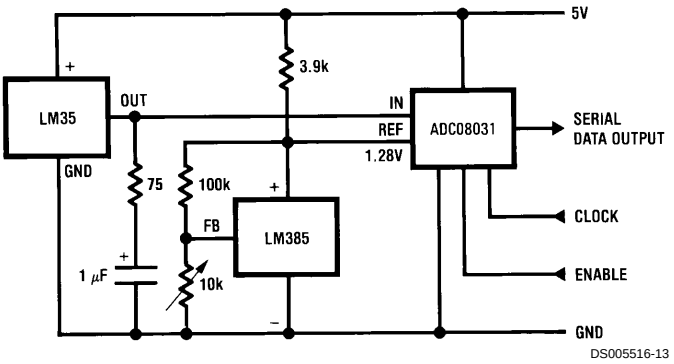


FIGURE 13. Temperature To Digital Converter (Serial Output) (+128˚C Full Scale)

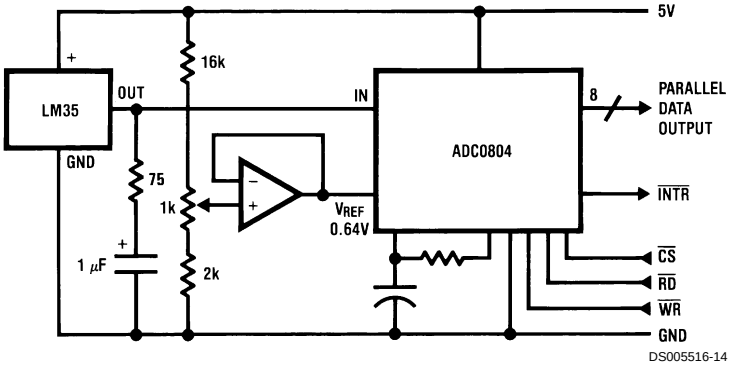
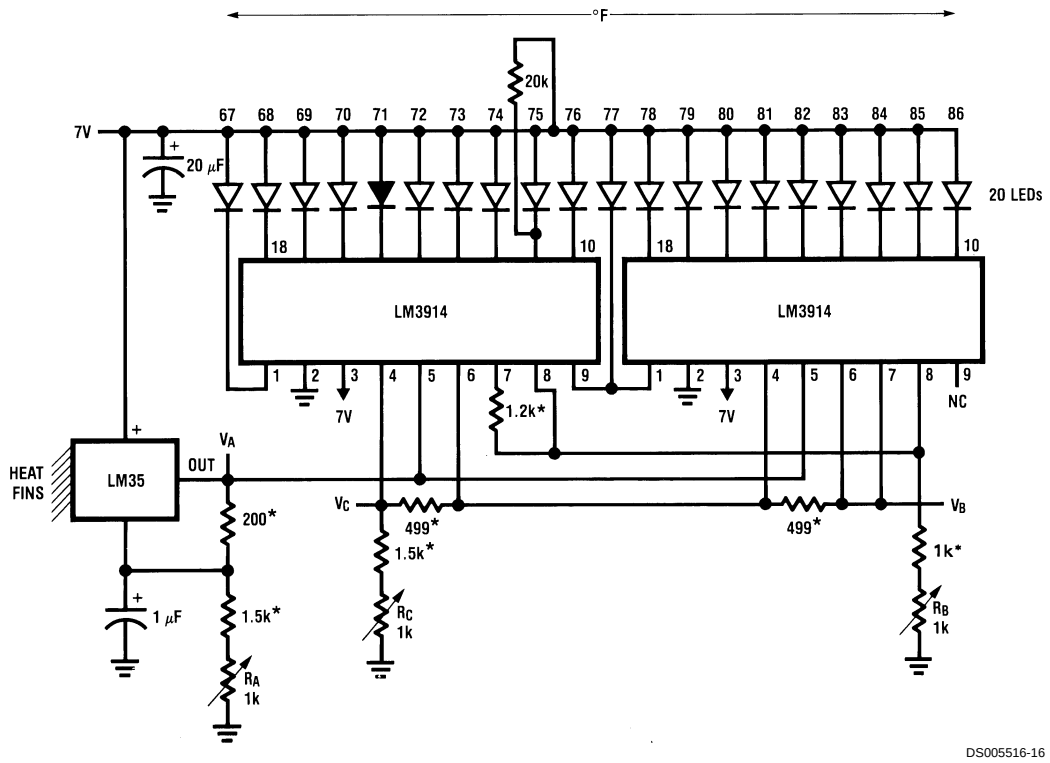


FIGURE 14. Temperature To Digital Converter (Parallel TRI-STATE™ Outputs for Standard Data Bus to μP Interface) (128˚C Full Scale)

Typical Applications (Continued)



*=1% or 2% film resistor
Trim R_B for V_B=3.075V
Trim R_C for V_C=1.955V
Trim R_A for V_A=0.075V + 100mV/°C x $T_{ambient}$
Example, V_A=2.275V at 22°C

FIGURE 15. Bar-Graph Temperature Display (Dot Mode)

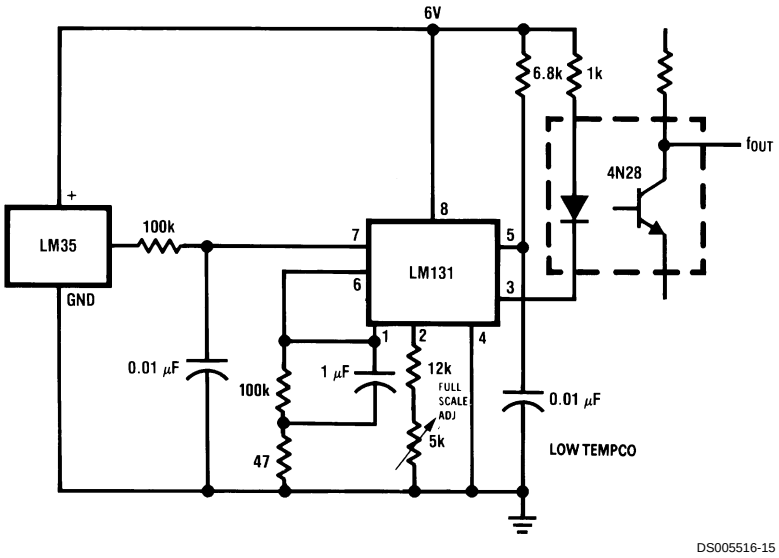
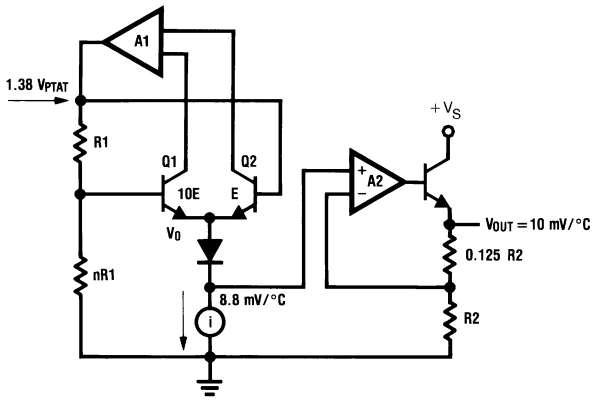


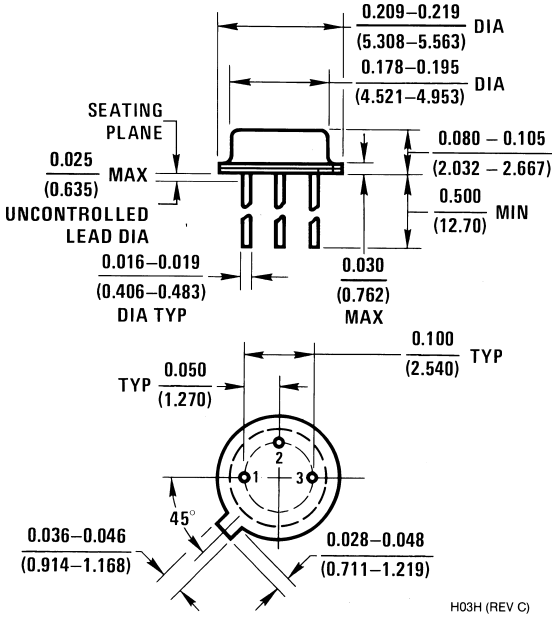
FIGURE 16. LM35 With Voltage-To-Frequency Converter And Isolated Output (2°C to +150°C; 20 Hz to 1500 Hz)

Block Diagram

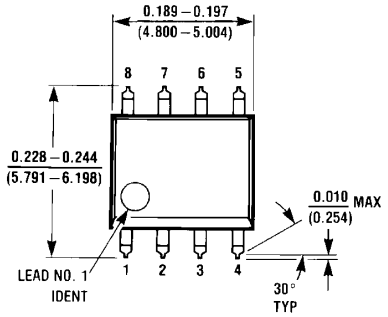


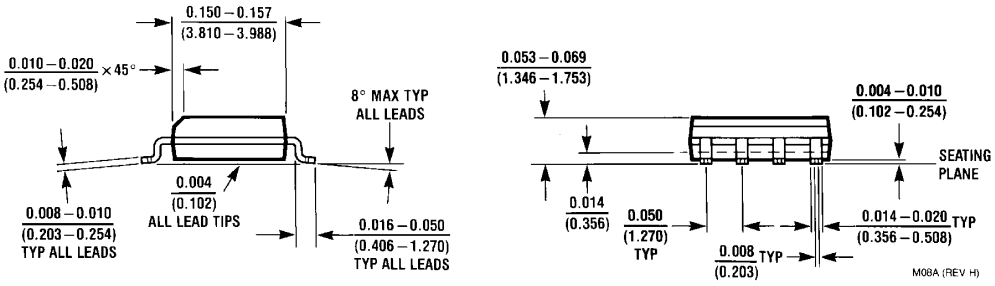
DS005516-23

Physical Dimensions inches (millimeters) unless otherwise noted



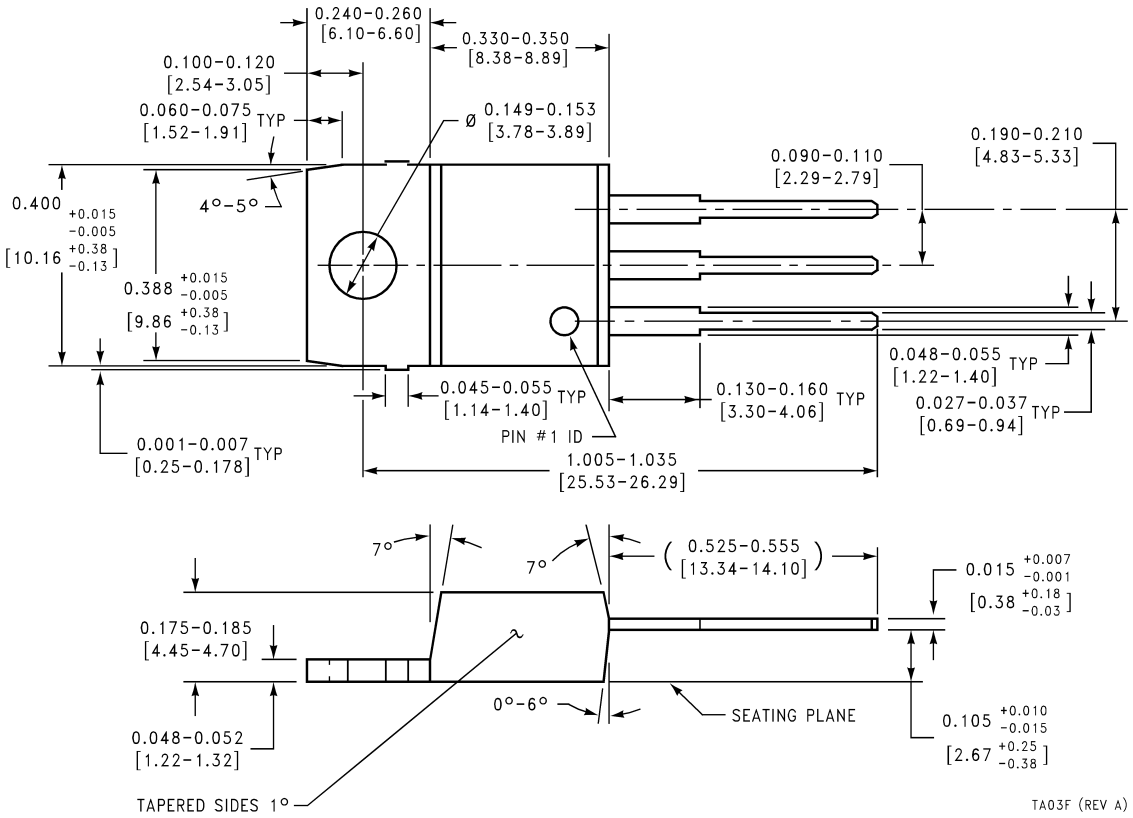
TO-46 Metal Can Package (H)
Order Number LM35H, LM35AH, LM35CH, LM35CAH, or LM35DH
NS Package Number H03H





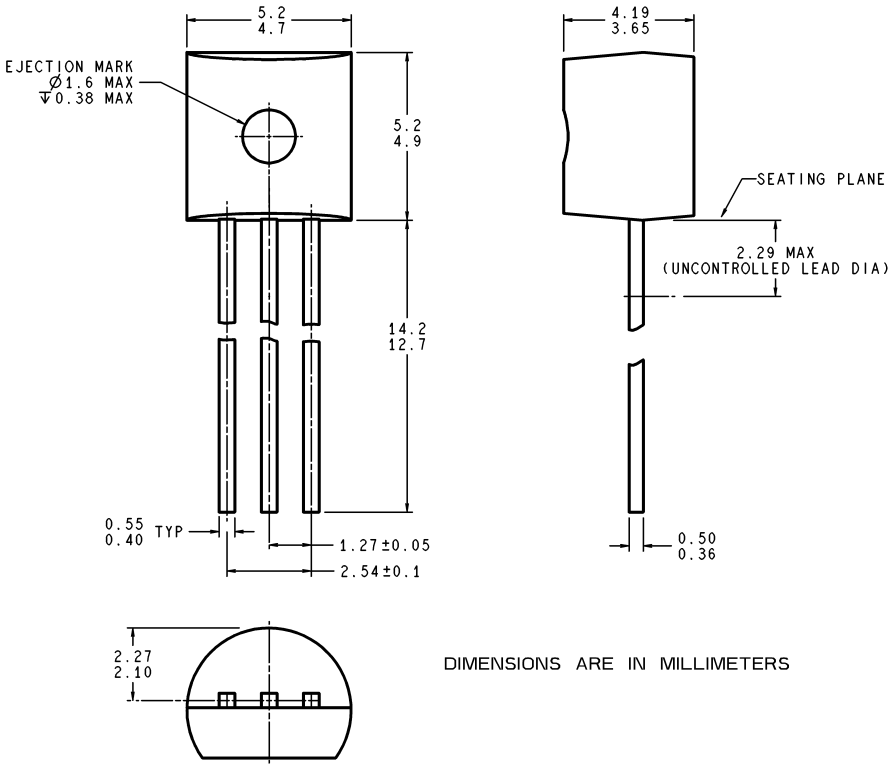
SO-8 Molded Small Outline Package (M)
Order Number LM35DM
NS Package Number M08A

Physical Dimensions inches (millimeters) unless otherwise noted (Continued)



Power Package TO-220 (T)
Order Number LM35DT
NS Package Number TA03F

Physical Dimensions inches (millimeters) unless otherwise noted (Continued)



TO-92 Plastic Package (Z)
Order Number LM35CZ, LM35CAZ or LM35DZ
NS Package Number Z03A

LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT AND GENERAL COUNSEL OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

National Semiconductor Corporation
Americas
Tel: 1-800-272-9959
Fax: 1-800-737-7018
Email: support@nsc.com
www.national.com

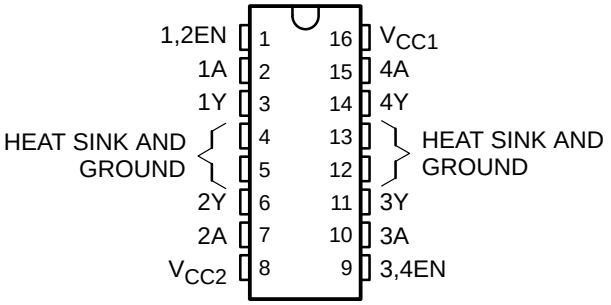
National Semiconductor Europe
Fax: +49 (0) 180-530 85 86
Email: europe.support@nsc.com
Deutsch Tel: +49 (0) 69 9508 6208
English Tel: +44 (0) 870 24 0 2171
Français Tel: +33 (0) 1 41 91 8790

National Semiconductor Asia Pacific Customer Response Group
Tel: 65-2544466
Fax: 65-2504466
Email: ap.support@nsc.com

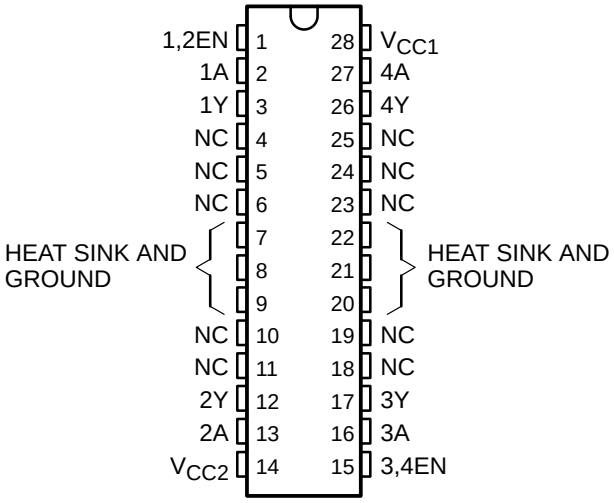
National Semiconductor Japan Ltd.
Tel: 81-3-5639-7560
Fax: 81-3-5639-7507

- **Featuring Unitrode L293 and L293D Products Now From Texas Instruments**
- **Wide Supply-Voltage Range: 4.5 V to 36 V**
- **Separate Input-Logic Supply**
- **Internal ESD Protection**
- **Thermal Shutdown**
- **High-Noise-Immunity Inputs**
- **Functional Replacements for SGS L293 and SGS L293D**
- **Output Current 1 A Per Channel (600 mA for L293D)**
- **Peak Output Current 2 A Per Channel (1.2 A for L293D)**
- **Output Clamp Diodes for Inductive Transient Suppression (L293D)**

N, NE PACKAGE
(TOP VIEW)



DWP PACKAGE
(TOP VIEW)



description

The L293 and L293D are quadruple high-current half-H drivers. The L293 is designed to provide bidirectional drive currents of up to 1 A at voltages from 4.5 V to 36 V. The L293D is designed to provide bidirectional drive currents of up to 600-mA at voltages from 4.5 V to 36 V. Both devices are designed to drive inductive loads such as relays, solenoids, dc and bipolar stepping motors, as well as other high-current/high-voltage loads in positive-supply applications.

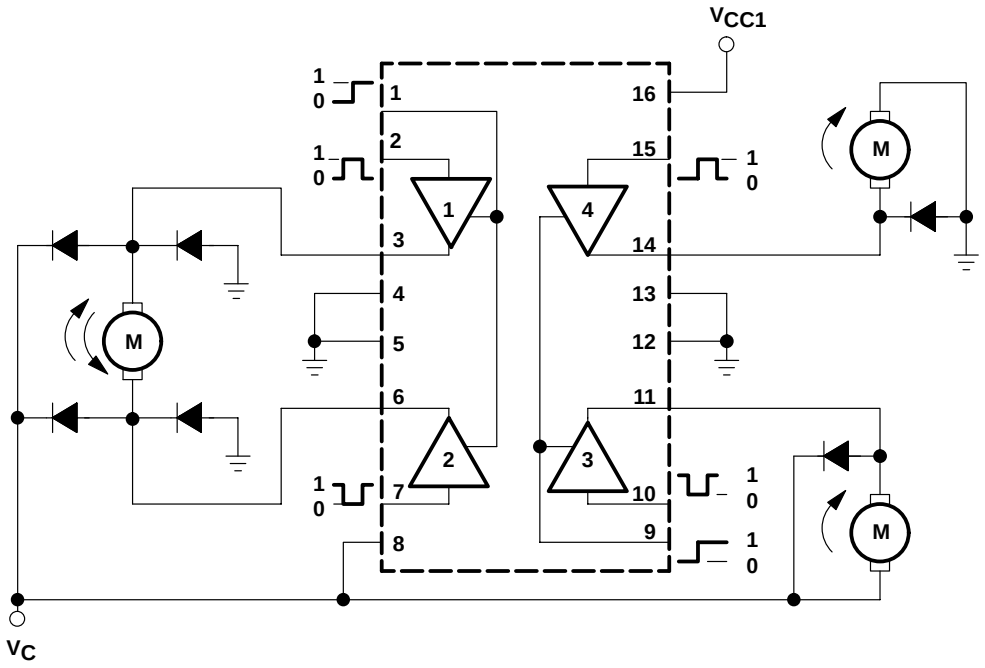
All inputs are TTL compatible. Each output is a complete totem-pole drive circuit, with a Darlington transistor sink and a pseudo-Darlington source. Drivers are enabled in pairs, with drivers 1 and 2 enabled by 1,2EN and drivers 3 and 4 enabled by 3,4EN. When an enable input is high, the associated drivers are enabled and their outputs are active and in phase with their inputs. When the enable input is low, those drivers are disabled and their outputs are off and in the high-impedance state. With the proper data inputs, each pair of drivers forms a full-H (or bridge) reversible drive suitable for solenoid or motor applications.

On the L293, external high-speed output clamp diodes should be used for inductive transient suppression.

A V_{CC1} terminal, separate from V_{CC2}, is provided for the logic inputs to minimize device power dissipation.

The L293and L293D are characterized for operation from 0°C to 70°C.

block diagram



NOTE: Output diodes are internal in L293D.

TEXAS INSTRUMENTS AVAILABLE OPTIONS

T_A	PACKAGE
	PLASTIC DIP (NE)
0°C to 70°C	L293NE L293DNE

Unitrode Products from Texas Instruments

AVAILABLE OPTIONS

T_A	PACKAGED DEVICES	
	SMALL OUTLINE (DWP)	PLASTIC DIP (N)
0°C to 70°C	L293DWP L293DDWP	L293N L293DN

The DWP package is available taped and reeled. Add the suffix TR to device type (e.g., L293DWPTR).

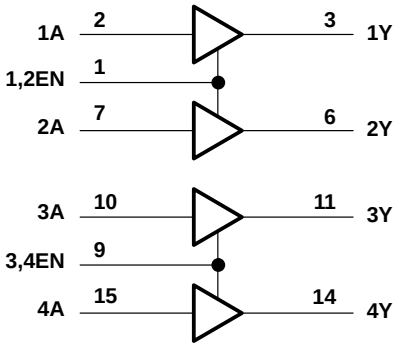
FUNCTION TABLE
(each driver)

INPUTS†		OUTPUT
A	EN	Y
H	H	H
L	H	L
X	L	Z

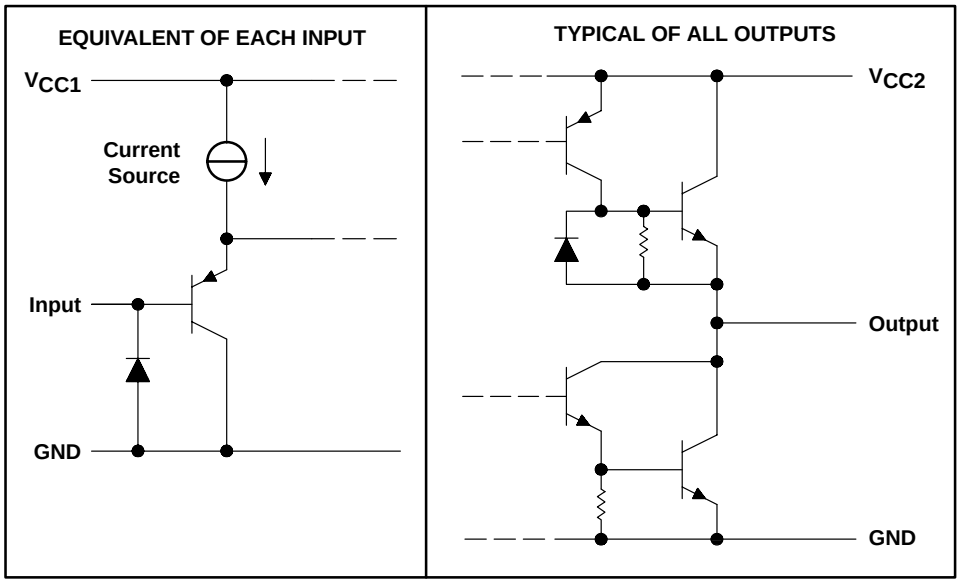
H = high level, L = low level, X = irrelevant, Z = high impedance (off)

† In the thermal shutdown mode, the output is in the high-impedance state, regardless of the input levels.

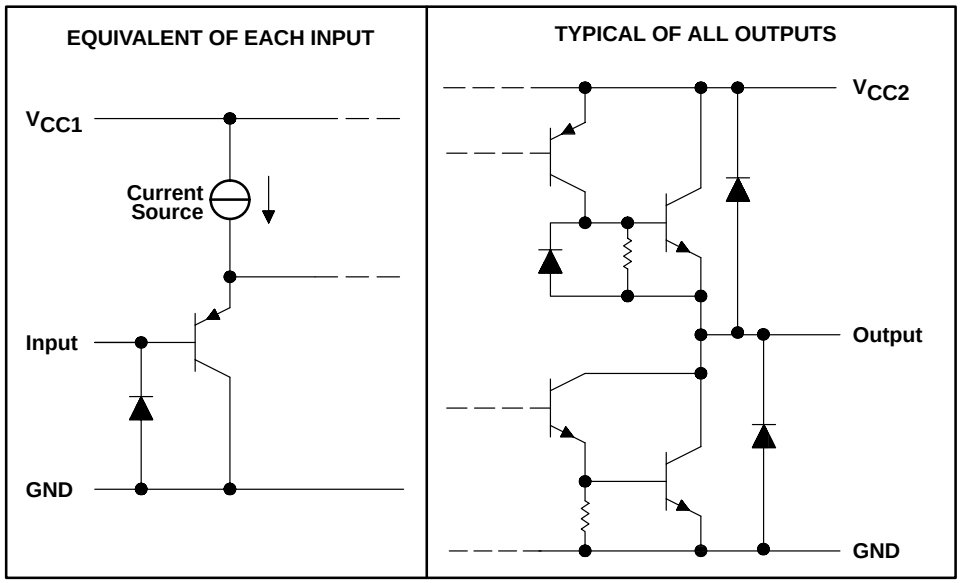
logic diagram



schematics of inputs and outputs (L293)



schematics of inputs and outputs (L293D)



absolute maximum ratings over operating free-air temperature range (unless otherwise noted)†

Supply voltage, V_{CC1} (see Note 1)	36 V
Output supply voltage, V_{CC2}	36 V
Input voltage, V_I	7 V
Output voltage range, V_O	–3 V to V_{CC2} + 3 V
Peak output current, I_O (nonrepetitive, t ≤ 5 ms): L293	±2 A
Peak output current, I_O (nonrepetitive, t ≤ 100 µs): L293D	±1.2 A
Continuous output current, I_O: L293	±1 A
Continuous output current, I_O: L293D	±600 mA
Continuous total dissipation at (or below) 25°C free-air temperature (see Notes 2 and 3)	2075 mW
Continuous total dissipation at 80°C case temperature (see Note 3)	5000 mW
Maximum junction temperature, T_J	150°C
Lead temperature 1,6 mm (1/16 inch) from case for 10 seconds	260°C
Storage temperature range, T_{stg}	–65°C to 150°C

† Stresses beyond those listed under “absolute maximum ratings” may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated under “recommended operating conditions” is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

NOTES:
1. All voltage values are with respect to the network ground terminal.
2. For operation above 25°C free-air temperature, derate linearly at the rate of 16.6 mW/°C.
3. For operation above 25°C case temperature, derate linearly at the rate of 71.4 mW/°C. Due to variations in individual device electrical characteristics and thermal resistance, the built-in thermal overload protection may be activated at power levels slightly above or below the rated dissipation.

recommended operating conditions

			MIN	MAX	UNIT
	Supply voltage	V_{CC1}	4.5	7	V
		V_{CC2}	V_{CC1}	36	
V_{IH}	High-level input voltage	$V_{CC1} \le 7$ V	2.3	V_{CC1}	V
		$V_{CC1} \ge 7$ V	2.3	7	V
V_{IL}	Low-level output voltage		–0.3†	1.5	V
T_A	Operating free-air temperature		0	70	°C

† The algebraic convention, in which the least positive (most negative) designated minimum, is used in this data sheet for logic voltage levels.

electrical characteristics, V_{CC1} = 5 V, V_{CC2} = 24 V, T_A = 25°C

PARAMETER			TEST CONDITIONS		MIN	TYP	MAX	UNIT
V_{OH}	High-level output voltage		L293: $I_{OH} = -1$ A L293D: $I_{OH} = -0.6$ A		V_{CC2}–1.8	V_{CC2}–1.4		V
V_{OL}	Low-level output voltage		L293: $I_{OL} = 1$ A L293D: $I_{OL} = 0.6$ A			1.2	1.8	V
V_{OKH}	High-level output clamp voltage		L293D: $I_{OK} = -0.6$ A			V_{CC2} + 1.3		V
V_{OKL}	Low-level output clamp voltage		L293D: $I_{OK} = 0.6$ A			1.3		V
I_{IH}	High-level input current	A	$V_I = 7$ V			0.2	100	µA
		EN				0.2	10	
I_{IL}	Low-level input current	A	$V_I = 0$			–3	–10	µA
		EN				–2	–100	
I_{CC1}	Logic supply current		$I_O = 0$	All outputs at high level		13	22	mA
				All outputs at low level		35	60	
				All outputs at high impedance		8	24	
I_{CC2}	Output supply current		$I_O = 0$	All outputs at high level		14	24	mA
				All outputs at low level		2	6	
				All outputs at high impedance		2	4	

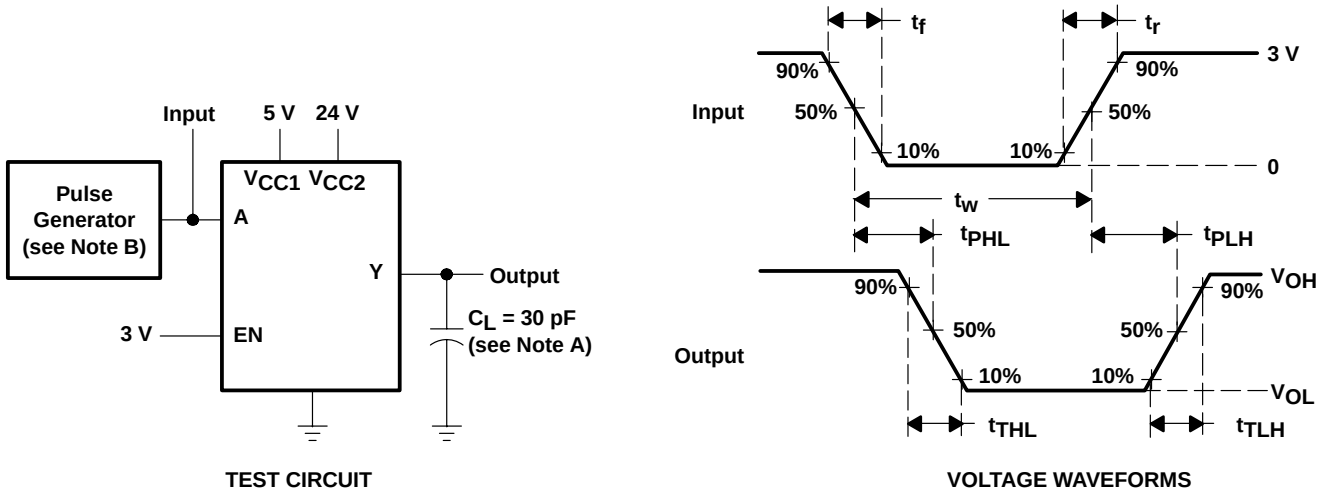
switching characteristics, V_{CC1} = 5 V, V_{CC2} = 24 V, T_A = 25°C

PARAMETER		TEST CONDITIONS	L293NE, L293DNE			UNIT
			MIN	TYP	MAX	
t_{PLH}	Propagation delay time, low-to-high-level output from A input	$C_L = 30$ pF, See Figure 1		800		ns
t_{PHL}	Propagation delay time, high-to-low-level output from A input			400		ns
t_{TLH}	Transition time, low-to-high-level output			300		ns
t_{THL}	Transition time, high-to-low-level output			300		ns

switching characteristics, V_{CC1} = 5 V, V_{CC2} = 24 V, T_A = 25°C

PARAMETER		TEST CONDITIONS	L293DWP, L293N L293DDWP, L293DN			UNIT
			MIN	TYP	MAX	
t_{PLH}	Propagation delay time, low-to-high-level output from A input	$C_L = 30$ pF, See Figure 1		750		ns
t_{PHL}	Propagation delay time, high-to-low-level output from A input			200		ns
t_{TLH}	Transition time, low-to-high-level output			100		ns
t_{THL}	Transition time, high-to-low-level output			350		ns

PARAMETER MEASUREMENT INFORMATION



NOTES: A. C_L includes probe and jig capacitance.
B. The pulse generator has the following characteristics: $t_r \leq 10$ ns, $t_f \leq 10$ ns, $t_W = 10$ µs, PRR = 5 kHz, $Z_O = 50\ \Omega$.

Figure 1. Test Circuit and Voltage Waveforms

APPLICATION INFORMATION

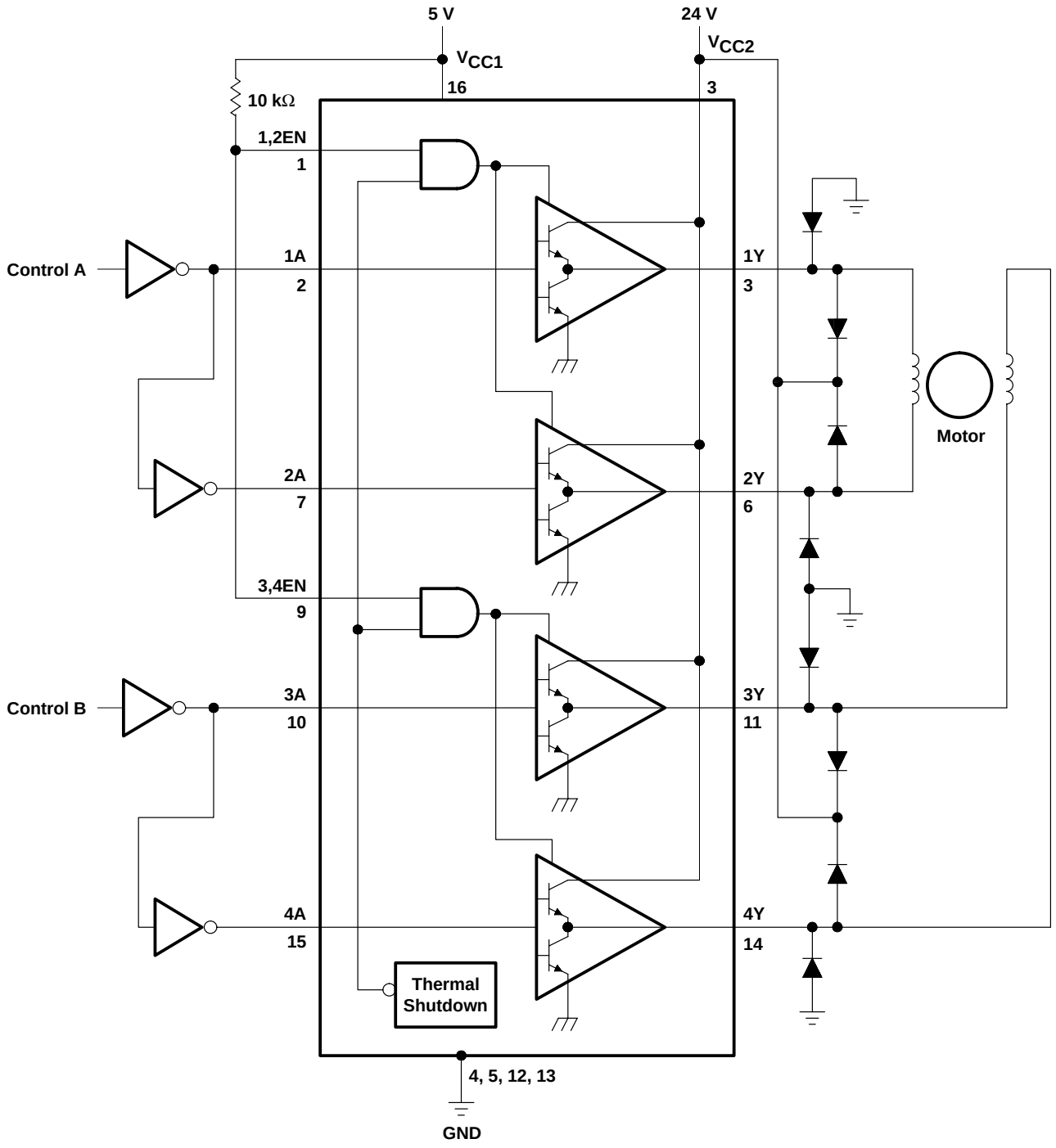


Figure 2. Two-Phase Motor Driver (L293)

APPLICATION INFORMATION

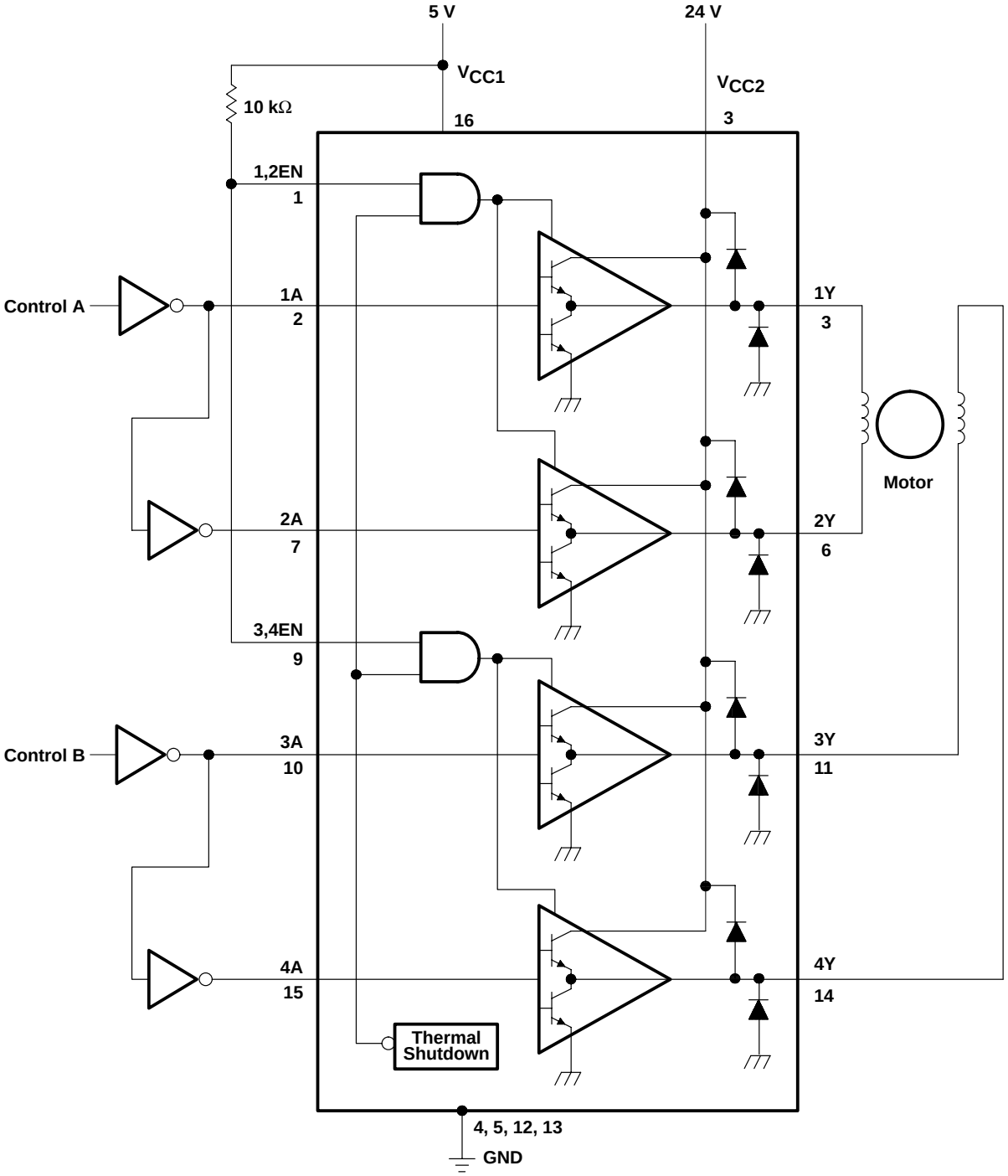


Figure 3. Two-Phase Motor Driver (L293D)

APPLICATION INFORMATION

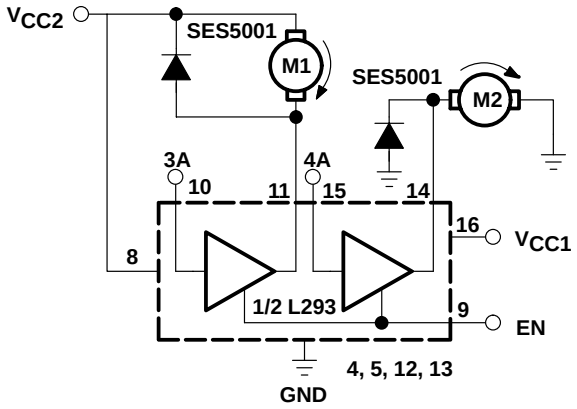


Figure 4. DC Motor Controls (connections to ground and to supply voltage)

EN	3A	M1	4A	M2
H	H	Fast motor stop	H	Run
H	L	Run	L	Fast motor stop
L	X	Free-running motor stop	X	Free-running motor stop

L = low, H = high, X = don't care

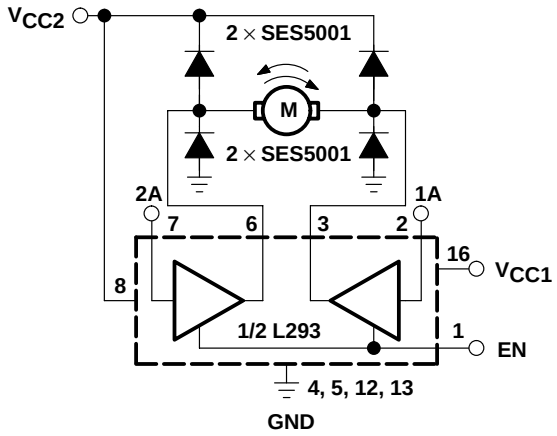
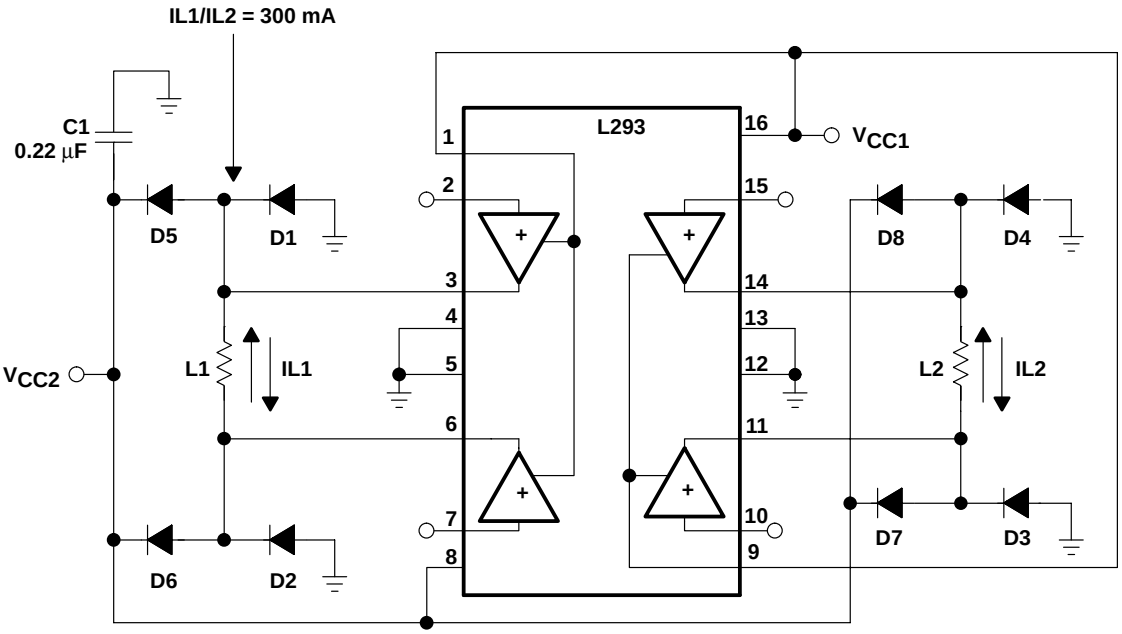


Figure 5. Bidirectional DC Motor Control

EN	1A	2A	FUNCTION
H	L	H	Turn right
H	H	L	Turn left
H	L	L	Fast motor stop
H	H	H	Fast motor stop
L	X	X	Fast motor stop

L = low, H = high, X = don't care

APPLICATION INFORMATION



D1–D8 = SES5001

Figure 6. Bipolar Stepping-Motor Control

mounting instructions

The Rthj-amp of the L293 can be reduced by soldering the GND pins to a suitable copper area of the printed circuit board or to an external heatsink.

Figure 9 shows the maximum package power P_{TOT} and the θ_{JA} as a function of the side ℓ of two equal square copper areas having a thickness of 35 µm (see Figure 7). In addition, an external heat sink can be used (see Figure 8).

During soldering, the pin temperature must not exceed 260°C, and the soldering time must not be longer than 12 seconds.

The external heatsink or printed circuit copper area must be connected to electrical ground.

APPLICATION INFORMATION

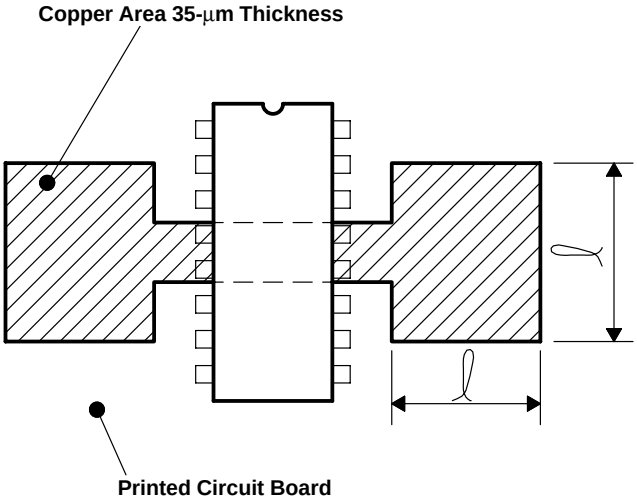


Figure 7. Example of Printed Circuit Board Copper Area (used as heat sink)

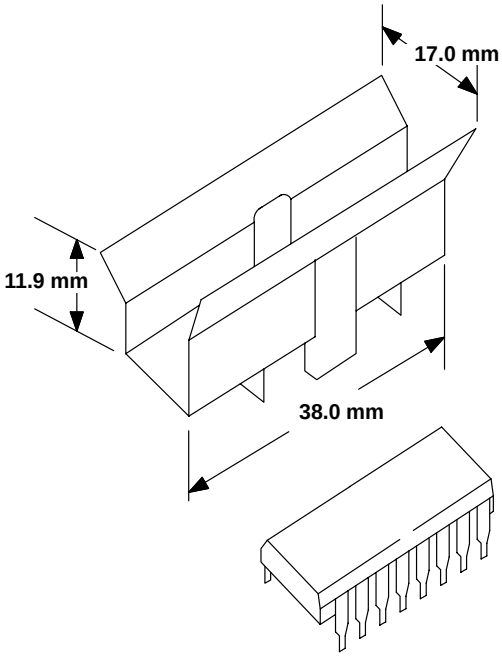


Figure 8. External Heat Sink Mounting Example (θ_{JA} = 25°C/W)

APPLICATION INFORMATION

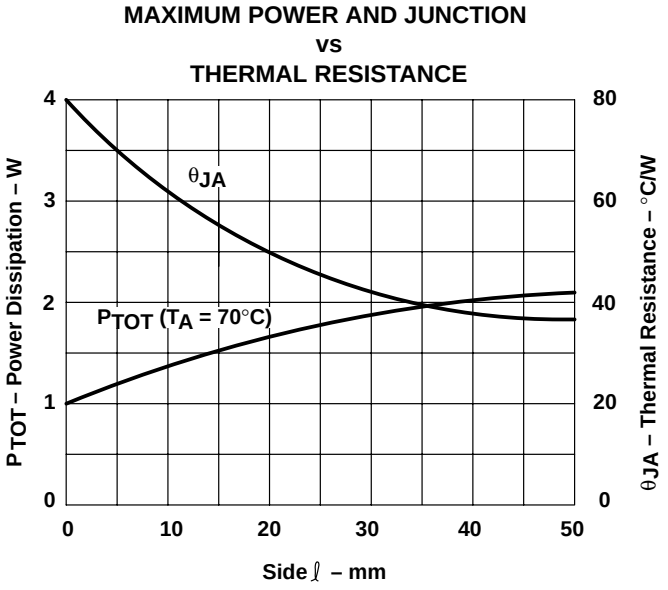


Figure 9

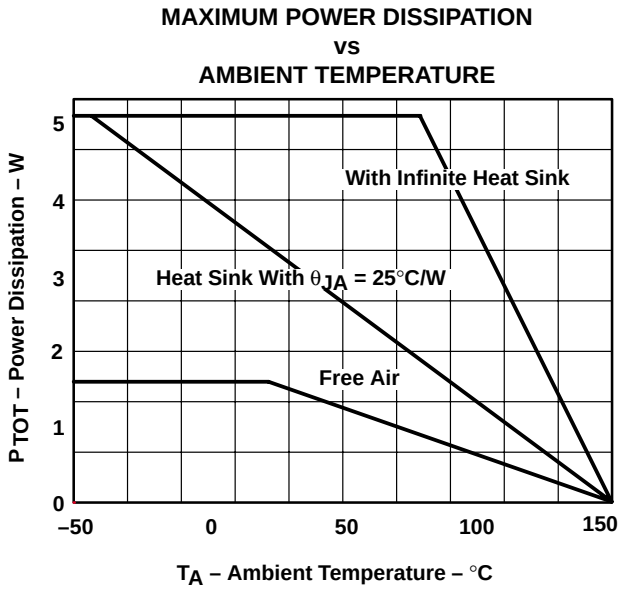


Figure 10

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265

Apéndice I

Protocolo SPI para tarjetas MMC y SD

Este apéndice contiene el capítulo 5 de la referencia [37], titulada *Secure Digital Card Product Manual*, Rev 1.0. En él se explica con mayor detalle el protocolo SPI para tarjetas *Secure Digital*.

5. SPI Protocol Definition

5.1. SPI Bus Protocol

While the Industrial Grade SD Card channel is based on command and data bit-streams, which are initiated by a start bit and terminated by a stop bit, the SPI channel is byte oriented. Every command or data block is built of eight bit bytes and is byte aligned (multiples of eight clocks) to the CS signal.

Similar to the SD Bus protocol, the SPI messages are built from command, response and data-block tokens. All communication between host and cards is controlled by the host (master). The host starts every bus transaction by asserting the CS signal low.

The response behavior in SPI Bus mode differs from the SD Bus mode in the following three ways:

- The selected card always responds to the command.
- An eight or 16-bit response structure is used.
- When the card encounters a data retrieval problem, it will respond with an error response (which replaces the expected data block) rather than time-out as in the SD Bus mode.

In addition to the command response, every data block sent to the card during write operations will be responded with a special data response token. A data block may be as big as one card write block (WRITE_BL_LEN) and as small as a single byte.[1]

5.1.1. Mode Selection

The Industrial Grade SD Card wakes up in the SD Bus mode. It will enter SPI mode if the CS signal is asserted (negative) during the reception of the reset command (CMD0). If the card recognizes that the SD Bus mode is required it will not respond to the command and remain in the SD Bus mode. If SPI mode is required, the card will switch to SPI mode and respond with the SPI mode R1 response.

The only way to return to the SD Bus mode is by power cycling the card. In SPI mode, the SD Card protocol state machine is not observed. All the SD Card commands supported in SPI mode are always available.

The default command structure/protocol for SPI mode is that CRC checking is disabled. Since the card powers up in SD Bus mode, CMD0 must be followed by a valid CRC byte (even though the command is sent using the SPI structure). Once in SPI mode, CRCs are disabled by default.

CMD0 is a static command and always generates the same 7-bit CRC of 4Ah. Adding the “1,” end bit (bit 0) to the CRC creates a CRC byte of 95h. The following hexadecimal sequence can be used to send CMD0 in all situations for SPI mode, since the CRC byte (although required) is ignored once in SPI mode. The entire CMD0 sequence appears as 40 00 00 00 00 95 (hexadecimal).

1) The default block length is as specified in the CSD (512 bytes). A set block length of less than 512 bytes will cause a write error. The only valid write set block length is 512 bytes. CMD16 is not mandatory if the default is accepted.

5.1.2. Bus Transfer Protection

Every Industrial Grade SD Card token transferred on the bus is protected by CRC bits. In SPI mode, the SD Card offers a non protected mode which enables systems built with reliable data links to exclude the hardware or firmware required for implementing the CRC generation and verification functions.

In the non-protected mode the CRC bits of the command, response and data tokens are still required in the tokens however, they are defined as “don’t care” for the transmitters and ignored by the receivers.

The SPI interface is initialized in the non-protected mode. The host can turn this option on and off using CRC_ON_OFF command (CMD59).

The CRC7/CRC16 polynomials are identical to that used in SD Bus mode. Refer to this section in the SD Bus mode chapter.

5.1.3. Data Read

SPI mode supports single block and multiple block read operations (SD Card CMD17 or CMD18). Upon reception of a valid read command the card will respond with a response token followed by a data token in the length defined in a previous SET_BLOCK_LENGTH (CMD16) command (see Figure 5-1).

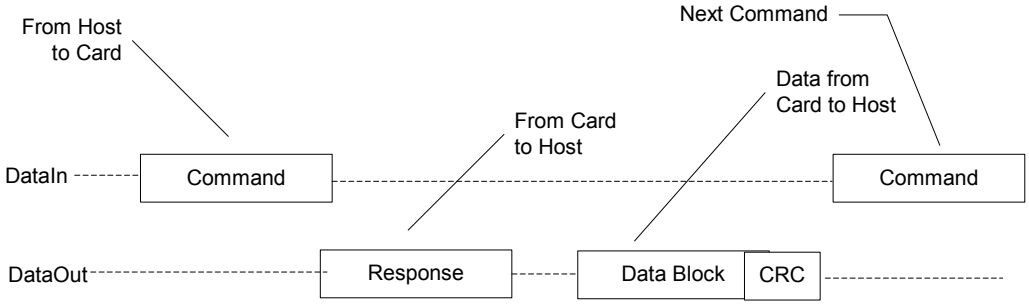


Figure 5-1. Single Block Read Operation

A valid data block is suffixed with a 16-bit CRC generated by the standard CCITT polynomial:

$$x^{16}+x^{12}+x^{5}+1.$$

The maximum block length is 512 bytes as defined by READ_BL_LEN (CSD parameter). Block lengths can be any number between 1 and READ_BL_LEN.

The start address can be any byte address in the valid address range of the card. Every block, however, must be contained in a single physical card sector.

In case of data retrieval error, the card will not transmit any data. Instead, a special data error token will be sent to the host. Figure 5-2 shows a data read operation, which terminated with an error token rather than a data block.

Figure 5-2. Read Operation—Data Error

In the case of a Multiple Block Read operation, every transferred block has a 16-bit CRC suffix. The Stop Transmission command (CMD12) will actually stop the data transfer operation (the same as in SD Bus mode).

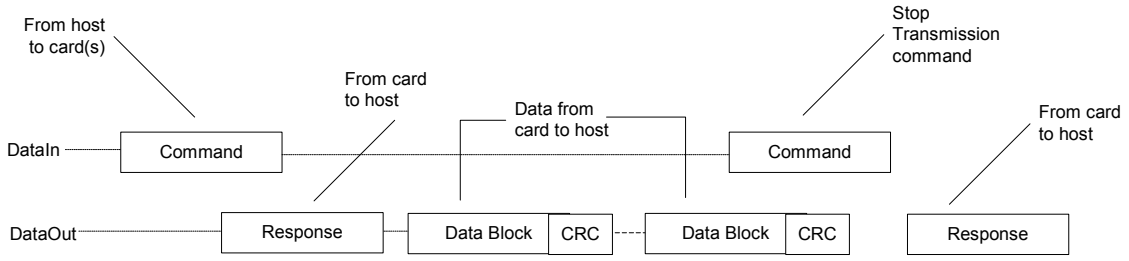


Figure 5-3. Multiple Block Read Operation

5.1.4. Data Write

In SPI mode, the Industrial Grade SD Card supports single block or multiple block write operations. Upon reception of a valid write command (SD Card CMD24 or CMD25), the card will respond with a response token and will wait for a data block to be sent from the host. CRC suffix and start address restrictions are identical to the read operation (see Figure 5-4). The only valid block length, however, is 512 bytes. Setting a smaller block length will cause a write error on the next write command.

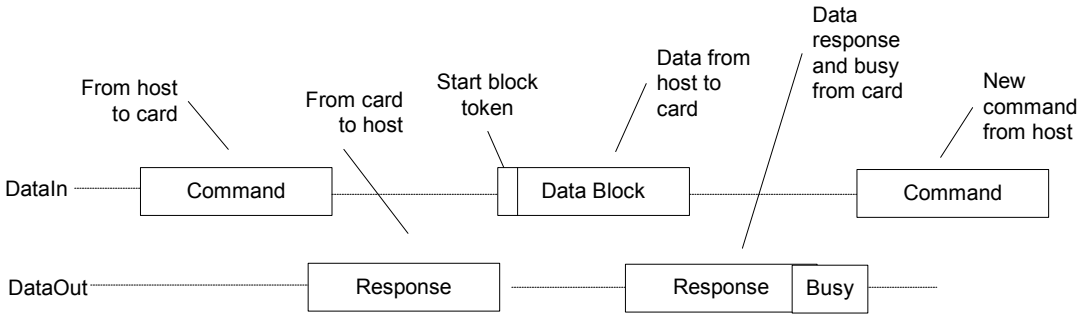


Figure 5-4. Single Block Write Operation

Every data block has a prefix or ‘start block’ token (one byte). After a data block is received the card will respond with a data-response token, and if the data block is received with no errors, it will be programmed. As long as the card is busy programming, a continuous stream of busy tokens will be sent to the host (effectively holding the dataOut line low).

Once the programming operation is completed, the host must check the results of the programming using the SEND_STATUS command (CMD13). Some errors (e.g., address out of range, write protect violation, etc.) are detected during programming only. The only validation check performed on the data block and communicated to the host via the data-response token is CRC and general Write Error indication.

In Multiple Block write operation the stop transmission will be done by sending 'Stop Tran' token instead of 'Start Block' token at the beginning of the next block. In case of Write Error indication (on the data response) the host shall use SEND_NUM_WR_BLOCKS (ACMD22) in order to get the number of well written write blocks. The data token's description is given in Section 5.2.4.

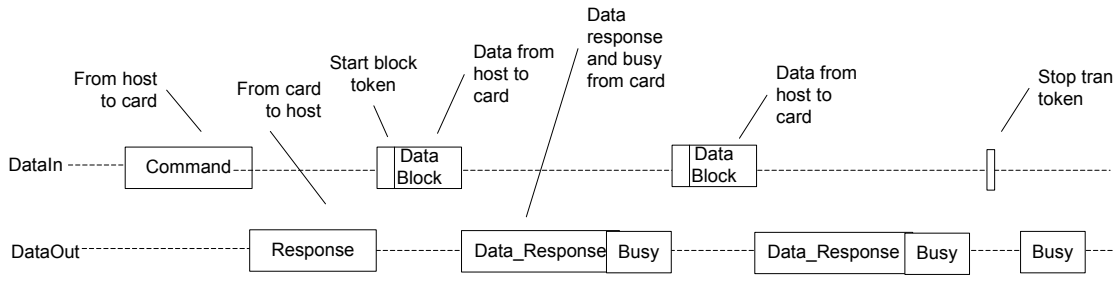


Figure 5-5. Multiple Block Write Operation

Resetting the CS signal while the card is busy does not terminate the programming process. The card releases the dataOut line (tristate) and continue to program. If the card is reselected before the programming is done, the dataOut line will be forced back to low and all commands will be rejected.

Resetting a card (using CMD0) will terminate any pending or active programming operation. This may destroy the data formats on the card. It is the host's responsibility to prevent it.

5.1.5. Erase and Write Protect Management

The erase and write protect management procedures in the SPI mode are identical to the SD Bus mode. While the card is erasing or changing the write protection bits of the predefined sector list it will be in a busy state and will hold the dataOut line low. Figure 5-6 illustrates a "no data" bus transaction with and without busy signaling.

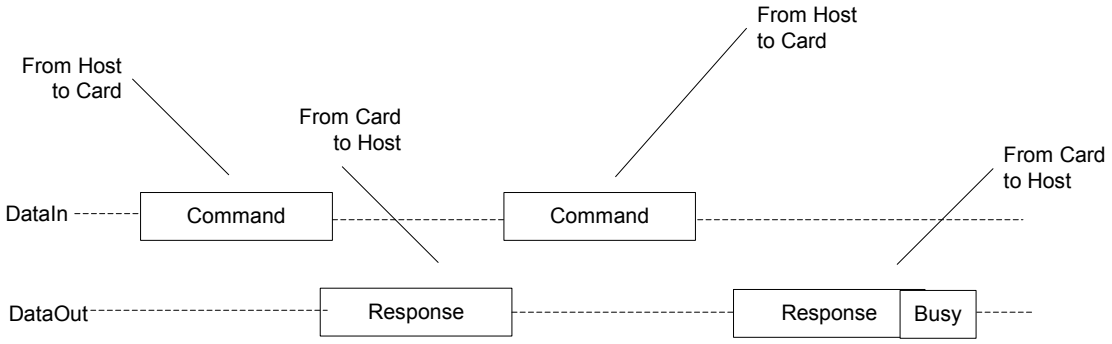


Figure 5-6. "No Data" Operations

5.1.6. Read CID/CSD Registers

Unlike the SD Bus protocol (where the register contents are sent as a command response), reading the contents of the CSD and CID registers in SPI mode is a simple read-block transaction. The card will respond with a standard response token followed by a data block of 16 bytes suffixed with a 16-bit CRC.

The data time out for the CSD command cannot be set to the card TAAC since this value is stored in the CSD. Therefore, the standard response time-out value (N_{CR}) is used for read latency of the CSD register.

5.1.7. Reset Sequence

The Industrial Grade SD Card requires a defined reset sequence. After power on reset or CMD0 (software reset), the card enters an idle state. At this state, the only legal host commands are CMD1 (SEND_OP_COND), ACMD41 (SD_SEND_OP_COND), CMD59 (CRC_ON_OFF) and CMD58 (READ_OCR).

The host must poll the card (by repeatedly sending CMD1) until the 'in-idle-state' bit in the card response indicates (by being set to 0) that the card completed its initialization processes and is ready for the next command.

In SPI mode, however, CMD1 has no operands and does not return the contents of the OCR register. Instead, the host can use CMD58 (SPI Mode Only) to read the OCR register. It is the responsibility of the host to refrain from accessing cards that do not support its voltage range.

The use of CMD58 is not restricted to the initialization phase only, but can be issued at any time. The host must poll the card (by repeatedly sending CMD1) until the 'in-idle-state' bit in the card response indicates (by being set to 0) that the card has completed its initialization process and is ready for the next command.

5.1.8. Clock Control

The SPI bus clock signal can be used by the SPI host to set the cards to energy-saving mode or to control the data flow (to avoid under-run or over-run conditions) on the bus. The host is allowed to change the clock frequency or shut it down.

There are a few restrictions the SPI host must follow:

- The bus frequency can be changed at any time (under the restrictions of maximum data transfer frequency, defined by the SD cards).
- It is an obvious requirement that the clock must be running for the Industrial Grade SD Card to output data or response tokens. After the last SPI bus transaction, the host is required to provide 8 (eight) clock cycles for the card to complete the operation before shutting down the clock. Throughout this 8-clock period, the state of the CS signal is irrelevant. It can be asserted or de-asserted. Following is a list of the various SPI bus transactions:
  - A command/response sequence. Eight clocks after the card response end bit. The CS signal can be asserted or de-asserted during these 8 clocks.
  - A read data transaction. Eight clocks after the end bit of the last data block.
  - A write data transaction. Eight clocks after the CRC status token.
- The host is allowed to shut down the clock of a "busy" card. The Industrial Grade SD Card will complete the programming operation regardless of the host clock. However, the host must provide a clock edge for the card to turn off its busy signal. Without a clock edge, the SD Card (unless previously disconnected by de-asserting the CS signal) will force the dataOut line down, permanently.

5.1.9. Error Conditions

The following sections provide valuable information on error conditions.

5.1.9.1. CRC and Illegal Commands

Unlike the SD Card protocol, in SPI mode the card will always respond to a command. The response indicates acceptance or rejection of the command. A command may be rejected in any one of the following cases:

- It is sent while the card is in read operation (except CMD12 which is legal).
- It is sent while the card is in Busy.
- Card is locked and it is other than Class 0 or 7 commands.
- It is not supported (illegal opcode).
- CRC check failed.
- It contains an illegal operand.
- It was out of sequence during an erase sequence.

Note that in case the host sends command while the card sends data in read operation then the response with an illegal command indication may disturb the data transfer.

5.1.9.2. Read, Write and Erase Time-out Conditions

The times after which a time-out condition for read operations occur are (card independent) **either 100 times longer** than the typical access times for these operations given below **or 100ms**. The times after which a time-out condition for Write/Erase operations occur are (card independent) **either 100 times longer** than the typical program times for these operations given below **or 250ms**. A card shall complete the command within this time period, or give up and return an error message. If the host does not get any response with the given time out it should assume the card is not going to respond anymore and try to recover (for example; reset the card, power cycle, reject). The typical access and program times are defined in the following sections.

For more information, refer to Table 4-17 in Section 4.0, Table 5-5 in Section 5.0 and the applications note in Appendix A, "Host Design Considerations: NAND MMC and SD-based Products."

Read

The read access time is defined as the sum of the two times given by the CSD parameters TAAC and NSAC. These card parameters define the typical delay between the end bit of the read command and the start bit of the data block.

Write

The R2W_FACTOR field in the CSD is used to calculate the typical block program time obtained by multiplying the read access time by this factor. It applies to all write/erase commands (e.g., SET (CLEAR)_WRITE_PROTECT, PROGRAM_CSD (CID) and the block write commands).

Erase

The duration of an erase command will be (order of magnitude) the number of write blocks (WRITE_BL) to be erased multiplied by the block write delay.

5.1.10. Memory Array Partitioning

Same as for SD Card mode.

5.1.11. Card Lock/Unlock

The Card Lock/Unlock feature is currently in the SanDisk Industrial Grade SD Card.

5.1.12. Application Specific Commands

The Application Specific commands are identical to SD mode with the exception of the APP_CMD status bit which is not available in SPI.

5.1.13. Copyright Protection Commands

All the special Copyright Protection ACMDs and security functionality are the same as for SD mode.

5.2. SPI Command Set

The following sections provide valuable information on the SPI Command Set.

5.2.1. Command Format

All the SD Card commands are 6 bytes long and transmitted MSB first.

Byte 1			Bytes 2—5	Byte 6	
7	6	5 0	31 0	7	0
0	1	Command	Command Argument	CRC	1

Commands and arguments are listed in Table 5-2.

7-bit CRC Calculation: $G(x) = x^{7} + x^{3} + 1$

$M(x) = (\text{start bit}) * x^{39} + (\text{host bit}) * x^{38} + ... + (\text{last bit before CRC}) * x^{0}$

$CRC[6...0] = \text{Remainder}[(M(x) * x^{7})/G(x)]$

5.2.2. Command Classes

As in SD mode, the SPI commands are divided into several classes (See Table 5-1). Each class supports a set of card functions. An Industrial Grade SD Card will support the same set of optional command classes in both communication modes (there is only one command class table in the CSD register). The available command classes, and the supported commands for a specific class, however, are different in the SD Card and the SPI communication mode.

Note that except the classes that are not supported in SPI mode (class 1, 3 and 9), the mandatory required classes for the SD mode are the same for the SPI mode.

Table 5-1. Command Classes in SPI Mode

Card CMD Class (CCC)	Class Description	Supported Commands																						
		0	1	9	10	12	13	16	17	18	24	25	27	28	29	30	32	33	38	42	55	56	58	59
class 0	Basic	+	+	+	+	+	+																+	+
class 1	Not supported in SPI																							
class 2	Block read							+	+	+														
class 3	Not supported in SPI																							
class 4	Block write										+	+	+											
class 5	Erase																+	+	+					
class 6	Write-protection (Optional)													+	+	+								
class 7	Lock Card (Optional)*																			+				
class 8	Application specific																				+	+		
class 9	Not supported in SPI																							
class 10-11	Reserved																							

* The Lock Card command is supported in the Industrial Grade SD Card.

5.2.2.1. Detailed Command Description

The following table provides a detailed description of the SPI bus commands. The responses are defined in Section 5.2.3. Table 5-2 lists all Industrial Grade SD Card commands. A “yes” in the SPI mode column indicates that the command is supported in SPI mode. With these restrictions, the command class description in the CSD is still valid. If a command does not require an argument, the value of this field should be set to zero. The reserved commands are reserved in SD Card mode as well.

The binary code of a command is defined by the mnemonic symbol. As an example, the content of the **Command** field for CMD0 is (binary) ‘000000’ and for CMD39 is (binary) ‘100111.’

Table 5-2. Description of SPI Bus Commands

CMD INDEX	SPI Mode	Argument	Resp	Abbreviation	Command Description
CMD0	Yes	None	R1	GO_IDLE_STATE	Resets the SD Card
CMD1	Yes	None	R1	SEND_OP_COND	Activates the card's initialization process.
CMD2	No				
CMD3	No				
CMD4	No				
CMD5	Reserved				
CMD6	Reserved				
CMD7	No				
CMD8	Reserved				
CMD9	Yes	None	R1	SEND_CSD	Asks the selected card to send its card-specific data (CSD).
CMD10	Yes	None	R1	SEND_CID	Asks the selected card to send its card identification (CID).
CMD11	No				
CMD12	Yes	None	R1b	STOP _TRANSMISSION	Forces the card to stop transmission during a multiple block read operation.
CMD13	Yes	None	R2	SEND_STATUS	Asks the selected card to send its status register.
CMD14	No				
CMD15	No				
CMD16	Yes	[31:0] block length	R1	SET_BLOCKLEN	Selects a block length (in bytes) for all following block commands (read & write).[1]
CMD17	Yes	[31:0] data address	R1	READ_SINGLE _BLOCK	Reads a block of the size selected by the SET_BLOCKLEN command.[2]
CMD18	Yes	[31:0] data address	R1	READ_MULTIPLE _BLOCK	Continuously transfers data blocks from card to host until interrupted by a STOP_ TRANSMISSION command.
CMD19	Reserved				
CMD20	No				
CMD21 ... CMD23	Reserved				
CMD24	Yes	[31:0] data address	R1[3]	WRITE_BLOCK	Writes a block of the size selected by the SET_BLOCKLEN command.[4]
CMD25	Yes	[31:0] data address	R1	WRITE_MULTIPLE_BLOCK	Continuously writes blocks of data until a stop transmission token is sent (instead of 'start block').

1) The only valid block length for write is 512 bytes. The valid block length for read is 1 to 512 bytes. A set block length of less than 512 bytes will cause a write error. The card has a default block length of 512 bytes. CMD16 is not mandatory if the default is accepted.
2) The start address and block length must be set so that the data transferred will not cross a physical block boundary.
3) Data followed by data response plus busy.
4) The start address must be aligned on a sector boundary. The block length is always 512 bytes.

CMD INDEX	SPI Mode	Argument	Resp	Abbreviation	Command Description
CMD26	No				
CMD27	Yes	None	R1	PROGRAM_CSD	Programming of the programmable bits of the CSD.
CMD28[1]	Yes	[31:0] data address	R1b	SET_WRITE_PROT	If the card has write protection features, this command sets the write protection bit of the addressed group. The properties of write protection are coded in the card specific data (WP_GRP_SIZE).
CMD29[4]	Yes	[31:0] data address	R1b	CLR_WRITE_PROT	If the card has write protection features, this command clears the write protection bit of the addressed group.
CMD30	Yes	[31:0] write protect data address	R1	SEND_WRITE_ PROT	If the card has write protection features, this command asks the card to send the status of the write protection bits.[2]
CMD31	Reserved				
CMD32	Yes	[31:0] data address	R1	ERASE_WR_BLK_START _ADDR	Sets the address of the first write block to be erased.
CMD33	Yes	[31:0] data address	R1	ERASE_WR_BLK_END _ADDR	Sets the address of the last write block in a continuous range to be erased.
CMD34 .... CMD37	Reserved				
CMD38	Yes	[31:0] don't care*	R1b	ERASE	Erases all previously selected write blocks.
CMD39	No				
CMD40	No				
CMD41 ... CMD54	Reserved				
CMD55	Yes	[31:0] stuff bits	R1	APP_CMD	Notifies the card that the next command is an application specific command rather than a standard command.
CMD56	Yes	[31:0] stuff bits [0]: RD/WR.[3]	R1	GEN_CMD	Used either to transfer a Data Block to the card or to get a Data Block from the card for general purpose/application specific commands. The size of the Data Block is defined with SET_BLOCK_LEN command.
CMD57	Reserved				
CMD58	Yes	None	R3	READ_OCR	Reads the OCR register of a card.
CMD59	Yes	[31:1] don't care* [0:0] CRC option	R1	CRC_ON_OFF	Turns the CRC option on or off. A '1' in the CRC option bit will turn the option on, a '0' will turn it off.
CMD60-63	No				

* The bit places must be filled but the values are irrelevant.

1) These features are not currently supported in the SanDisk SD Card.
2) 32 write protection bits (representing 32 write protect groups starting at the specified address) followed by 16 CRC bits are transferred in a payload format via the data line.
3) RD/WR_: "1"=the host will get a block of data from the card. "0"=the host sends a block of data to the card.

Table 5-3 describes all the application specific commands supported or reserved by the Industrial Grade SD Card. All the following commands should be preceded with APP_CMD (CMD55).

Table 5-3. Application Specific Commands Used or Reserved by the SD Card–SPI Mode

CMD INDEX	SPI Mode	Argument	Resp	Abbreviation	Command Description
ACMD6	No				
ACMD13	Yes	[31:0] stuff bits	R2	SD_STATUS	Send the SD Card status. The status fields are given in Table 4-21
ACMD17	Reserved				
ACMD18	Yes	--	--	--	Reserved for SD security applications1
ACMD19 to ACMD21	Reserved				
ACMD22	Yes	[31:0] stuff bits	R1	SEND_NUM_WR_ BLOCKS	Send the numbers of the well-written (without errors) blocks. Responds with 32bit+CRC data block.
ACMD23	Yes	[31:23] stuff bits [22:0]Number of blocks	R1	SET_WR_BLK_ ERASE_COUNT	Set the number of write blocks to be pre-erased before writing (to be used for faster Multiple Block WR command). “1”=default (one wr block)(2).
ACMD24	Reserved				
ACMD25	Yes	--	--	--	Reserved for SD security applications1
ACMD26	Yes	--	--	--	Reserved for SD security applications1
ACMD38	Yes	--	--	--	Reserved for SD security applications1
ACMD39 to ACMD40	Reserved				
ACMD41	Yes	None	R1	SEND_OP_ COND	Activates the card’s initialization process.
ACMD42	Yes	[31:1] stuff bits [0]set_cd	R1	SET_CLR_CARD_ DETECT	Connect[1]/Disconnect[0] the 50KOhm pull-up resistor on CD/DAT3 (pin 1) of the card. The pull-up may be used for card detection.
ACMD43 ... ACMD49	Yes	--	--	--	Reserved for SD security applications.[1]
ACMD51	Yes	[31:0] staff bits	R1	SEND_SCR	Reads the SD Configuration Register (SCR).

NOTES: (1) Refer to “SD Card Security Specification” for detailed explanation about the SD Security Features
(2) Command STOP_TRAN (CMD12) shall be used to stop the transmission in Write Multiple Block whether the pre-erase (ACMD23) feature is used or not.

5.2.3. Responses

There are several types of response tokens. As in the SD Card mode, all are transmitted MSB first.

5.2.3.1. Format R1

This response token is sent by the card after every command with the exception of SEND_STATUS commands. It is 1 byte long, the MSB is always set to zero and the other bits are error indications. A '1' signals error.

- In idle state—The card is in idle state and running initializing process.
- Erase reset—An erase sequence was cleared before executing because an out of erase sequence command was received.
- Illegal command—An illegal command code was detected.
- Communication CRC error—The CRC check of the last command failed.
- Erase sequence error—An error in the sequence of erase commands occurred.
- Address error—A misaligned address, which did not match the block length was used in the command.
- Parameter error—The command's argument (e.g., address, block length) was out of the allowed range for this card.

The structure of the R1 format is shown in Figure 5-7.

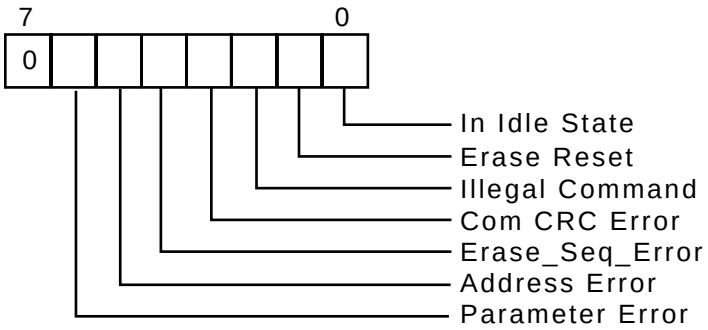


Figure 5-7. R1 Response Format

5.2.3.2. Format R1b

This response token is identical to R1 format with the optional addition of the busy signal. The busy signal token can be any number of bytes. A zero value indicates card is busy. A non-zero value indicates card is ready for the next command.

5.2.3.3. Format R2

This 2-bytes long response token is sent by the card as a response to the SEND_STATUS command. The format of the R2 status is shown in Figure 5-8.

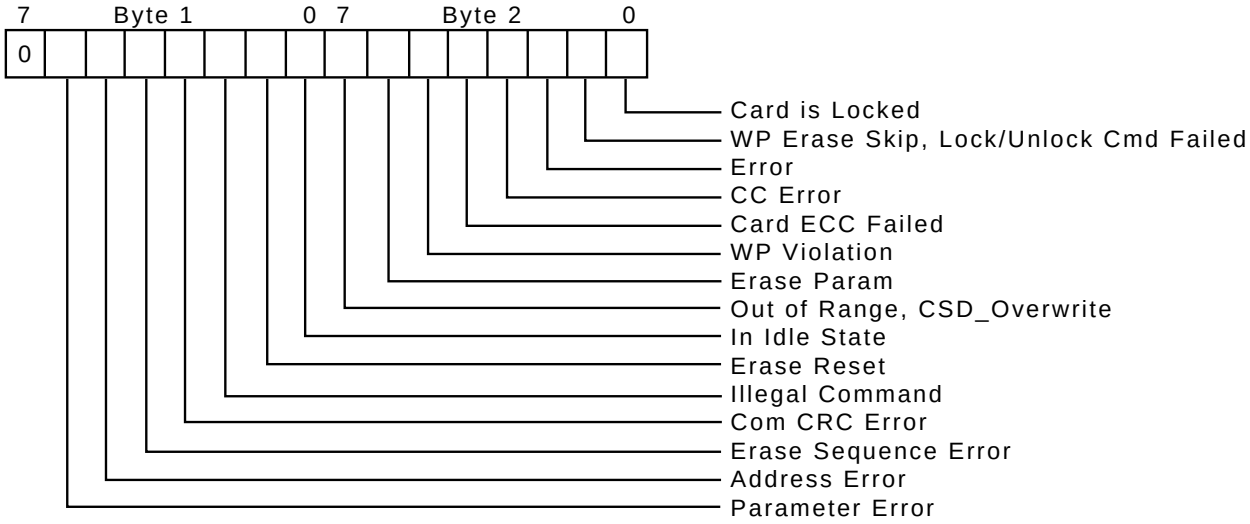


Figure 5-8. R2 Response Format

The first byte is identical to response R1. The content of the second byte is described below:

- **Erase param**—An invalid selection, sectors for erase.
- **Write protect violation**—The command tried to write a write-protected block.
- **Card ECC failed**—Card internal ECC was applied but failed to correct the data.
- **CC error**—Internal card controller error.
- **Error**—A general or an unknown error occurred during the operation.
- **Write protect erase skip**—Only partial address space was erased due to existing WP blocks.
- **Card is locked**—Supported by the SanDisk Industrial Grade SD Card.

5.2.3.4. Format R3

This response token is sent by the card when a READ_OCR command is received. The response length is 5 bytes. The structure of the first (MSB) byte is identical to response type R1. The other four bytes contain the OCR register.

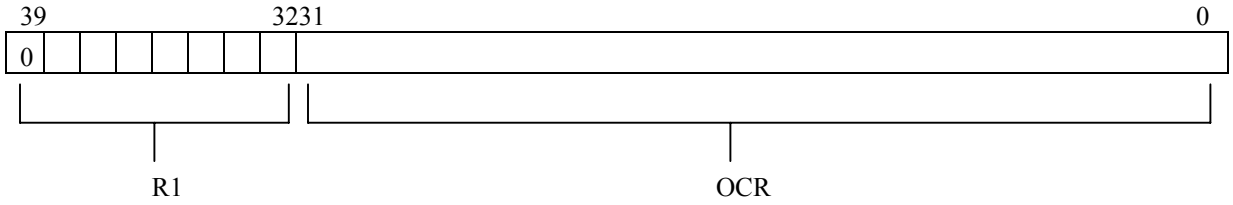


Figure 5-9. R3 Response Format

5.2.3.5. Data Response

Every data block written to the card is acknowledged by a data response token. It is one byte long and has the following format:

7	6				0
x	x	x	0	Status	1

The meaning of the status bits is defined as follows:

- ‘010’—Data accepted.
- ‘101’—Data rejected due to a CRC error.
- ’110’—Data Rejected due to a Write Error

In case of any error (CRC or Write Error) during Write Multiple Block operation, the host shall stop the data transmission using CMD12. In case of Write Error (response ’110’) the host may send CMD13 (SEND_STATUS) in order to get the cause of the write problem. ACMD22 can be used to find the number of well written write blocks.

5.2.4. Data Tokens

Read and write commands have data transfers associated with them. Data is being transmitted or received via data tokens. All data bytes are transmitted MSB.

Data tokens are 4 to 515 bytes long and have the following format:

For Single Block Read, Single Block Write and Multiple Block Read:

- First byte: Start Block.

7							0
1	1	1	1	1	1	1	0

- Bytes 2-513 (depends on the data block length): User data.
- Last two bytes: 16-bit CRC.

For Multiple Block Write operation:

- First byte of each block.

If data is to be transferred then—Start Block

7							0
1	1	1	1	1	1	0	0

If Stop transmission is requested—Stop Tran

7	0

1	1	1	1	1	1	0	1

Note that this format is used only for Multiple Block Write. In case of Multiple Block Read the stop transmission is done using STOP_TRAN Command (CMD12).

5.2.5. Data Error Token

If a read operation fails and the card cannot provide the required data it will send a data error token, instead. This token is one byte long and has the format shown in Figure 5-10.

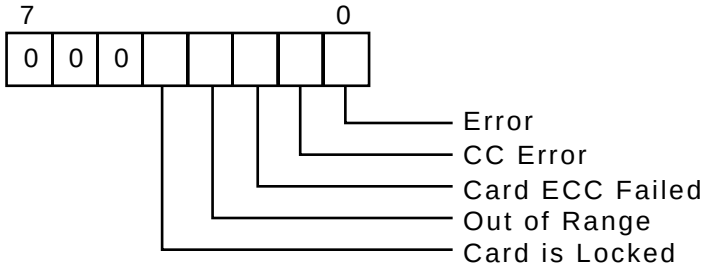


Figure 5-10. Data Error Token

The four least significant bits (LSB) are the same error bits as in response format R2.

5.2.6. Clearing Status Bits

As described in the previous paragraphs, in SPI mode, status bits are reported to the host in three different formats: response R1, response R2 and data error token (the same bits may exist in multiple response types—e.g., Card ECC failed). As in the SD mode, error bits are cleared when read by the host, regardless of the response format.

5.3. Card Registers

In SPI Mode, only the OCR, CSD and CID registers are accessible. Their format is identical to their format in the SD Card mode. However, a few fields are irrelevant in SPI mode.

5.4. SPI Bus Timing Diagrams

All timing diagrams use the schematics and abbreviations listed in Table 5-5.

Table 5-4. SPI Bus Timing Abbreviations

H	Signal is high (logical ‘1’)
L	Signal is low (logical ‘0’)
X	Don’t care
Z	High impedance state (-> = 1)
*	Repeater
Busy	Busy Token
Command	Command token
Response	Response token
Data block	Data token

All timing values are defined in Table 5-5. The host must keep the clock running for at least N_{CR} clock cycles after the card response is received. This restriction applied to command and data response tokens.

5.4.1. Command/Response

Host Command to Card Response–Card is Ready

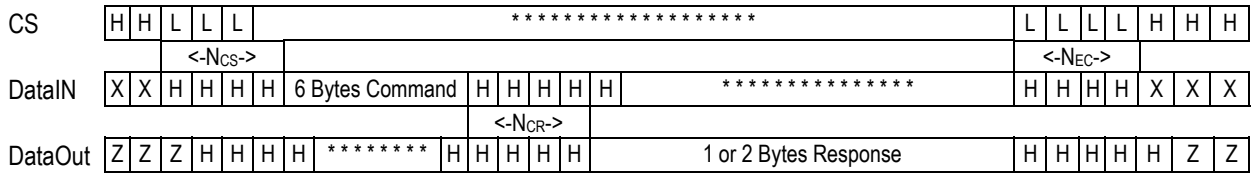


Figure 5-11. Host Command to Card Response–Card is Ready

Host Command to Card Response–Card is Busy

The following timing diagram describes the command response transaction for commands when the card responses which the R1b response type (e.g., SET_WRITE_PROT and ERASE). When the card is signaling busy, the host may deselect it (by raising the CS) at any time. The card will release the DataOut line one clock after the CS going high. To check if the card is still busy it needs to be reselected by asserting (set to low) the CS signal. The card will resume busy signal (pulling DataOut low) one clock after the falling edge of CS.

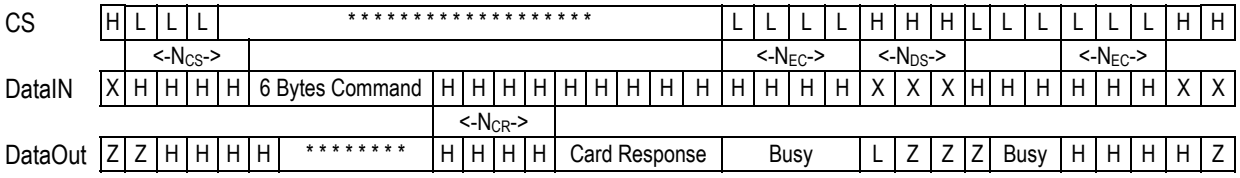


Figure 5-12. Host Command to Card Response–Card is Busy

Card Response to Host Command

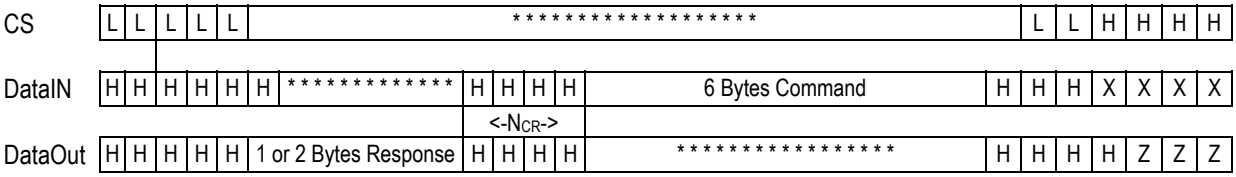


Figure 5-13. Card Response to Host Command

5.4.2. Data Read

The following timing diagram describes all single block read operations with the exception of SEND_CSD command.

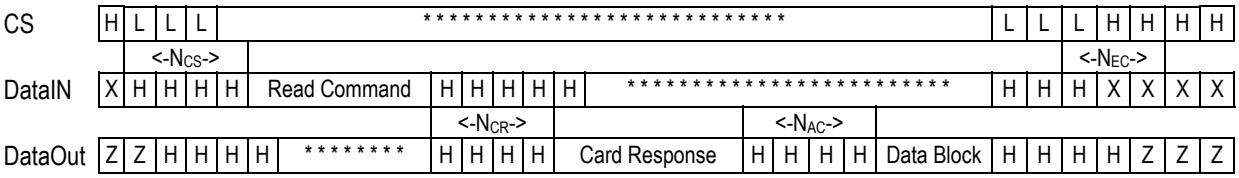


Figure 5-14. Single Block Read Timing

The following table describes Stop transmission operation in case of Multiple Block Read.

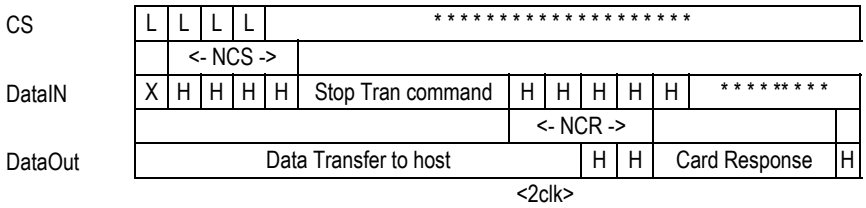


Figure 5-15. Multiple Block Read Timing

Reading the CSD Register

The following timing diagram describes the SEND_CSD command bus transaction. The timeout values for the response and the data block are N_{CR} (Since the N_{AC} is still unknown).

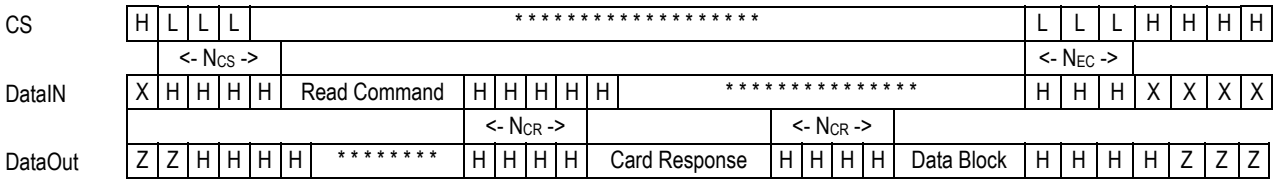


Figure 5-16. Reading the CSD Register

5.4.3. Data Write

The host may deselect a card (by raising the CS) at any time during the card busy period (refer to the given timing diagram). The card will release the DataOut line one clock after the CS going high. To check if the card is still busy it needs to be re-selected by asserting (set to low) the CS signal.

The card will resume busy signal (pulling DataOut low) one clock after the falling edge of CS.

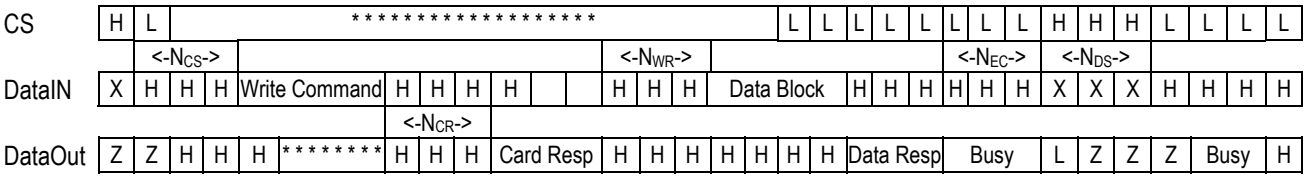


Figure 5-17. Device Write Timing

The following figure describes stop transmission operation in Multiple Block Write transfer.

CS	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	H	H	H	L	L	L	L
											<N_{WR} ->			<1byte->			<N_{BR} ->			<N_{EC} ->			<- N_{DS}->						
Data In	Data Block			H	H	H	H	H	H	H	H	H	stop tran token			H	H	H				X	X	X	H	H	H	H		
Data Out	H	H	H	H	Data Resp		Busy			H	H	H	H	H	H	H	H	H	Busy(1)			L	Z	Z	Z	Busy(1)		H		

(1) The Busy may appear within N_{BR} clocks after Stop Tran Token. If there is no Busy, the host may continue to the next command.

Figure 5-18. Stop Transmission Timing—Multiple Block Write

5.4.4. Timing Values

Table 5-5 shows the timing values and definitions. For more information, refer to Table 4-17 in Section 4.0, Section 5.1.9.2, and the applications note in Appendix A, "Host Design Considerations: NAND MMC and SD-based Products."

Table 5-5. Timing Constants Definitions

	Min	Max	Unit
N_{CS}	0	-	8 Clock Cycles
N_{CR}	0	8	8 Clock Cycles
N_{RC}	1	-	8 Clock Cycles
N_{AC}	1	See Note	8 Clock Cycles
N_{WR}	1	-	8 Clock Cycles
N_{EC}	0	-	8 Clock Cycles
N_{DS}	0	-	8 Clock Cycles
N_{BR}	0	1	8 Clock Cycles

NOTE: min [{{(TAAC ∗ f) + (NSAC ∗ 100)} ∗ 1/8}, {(100ms * f) * 1/8}] where units = (8 clocks) and "f" is the clock frequency.

5.5. SPI Electrical Interface

The SPI Mode electrical interface is identical to that of the SD Card mode.

5.6. SPI Bus Operating Conditions

Identical to SD Card mode.

5.7. Bus Timing

Identical to SD Card mode. The timing of the CS signal is the same as any other card input.

Bibliografía

[1] Profibus International. *Página oficial de Profibus*. World Wide Web, `http://www.profibus.com`, 2004. Último acceso: 10/06/2004.

[2] BITBUS European Users Group. *Página oficial de la Asociación de Usuarios de Bitbus*. World Wide Web, `http://www.bitbus.org`, 2004. Último acceso: 10/06/2004.

[3] INTERBUS Club. *Página oficial de Organización de Usuarios y Fabricantes que utilizan Interbus*. World Wide Web, `http://www.ibsclub.com`, 2004. Último acceso: 10/06/2004.

[4] Robert Bosch GmbH. *CAN Specification. Version 2.0, Parts A and B*, 1991. `http://www.can.bosch.com/content/literature.html`.

[5] CAN in Automation. *Página oficial de CiA*. World Wide Web, `http://www.can-cia.de`, 2004. Último acceso: 11/07/2004.

[6] Society of Automotive Engineers. *Página oficial de J1939*. World Wide Web, `http://www.sae.org`, 2004. Último acceso: 11/07/2004.

[7] K. Pazul. DS00713A. Controller Area Network (CAN) Basics. Technical report, Microchip Technology Inc., 1999.

[8] A. Sebastiá F. Mora J. Montesinos V. Herrero, F. Ballester. *El Bus CAN. Descripción y funcionamiento. Mundo Electrónico*, (318):44–50, Marzo 2001.

[9] Open DeviceNet Vendor Association. *Página oficial de DeviceNet*. World Wide Web, `http://www.odva.org`, 2004. Último acceso: 11/07/2004.

[10] X. Bogariz. *Módulo de Control de la Calefacción del Habitáculo de un Vehículo*. Proyecto fin de carrera, E.T.S. de Ingeniería, Universitat Rovira i Virgilia, Setiembre 2002.

[11] Hipólito Díaz Morales. *Análisis del sistema CAN bus de datos I, II y III. El periódico del taller. Formación técnica*, (22, 23 y 24), Abril, Julio y Octubre 2002. `http://www.serca.es`.

[12] F. Ballester A. Sebastià J. Cerdà, V. Herrero. Aplicación del bus can al control de ascensores. (314), Noviembre 2000.

[13] L. Gómez Chova J. Vila Francés, J. Calpe Maravilla. *Sistema de hilo musical en formato MP3 basado en bus CAN. Mundo Electrónico*, (330):40–46, Abril 2002.

[14] G. LUFFT Messund Regeltechnik. *Aplicaciones* CAN*bus: sistema de alarma. Monitoreo de control.* World Wide Web, `http://www.lufft.de`, 2004. Último acceso: 15/03/2004.

[15] MMCA Technical Committee. Mmc system summary v3.31. Technical report, Multimedia Card Association, 2003. Último acceso: 17/08/2004.

[16] SanDisk. MultiMediaCard Product Manual, Rev. 5.1. Technical report, SanDisk Corporation Headquarters, 2002. Último acceso: 17/08/2004.

[17] Microsoft Corporation. Microsoft extensible firmware initiative fat32 file system specification. Technical report, Microsoft Corporation, 2000.

[18] Microchip Technology Inc. *DS41159C. PIC18FXX8 Data Sheet. 28/40 High Performance, Enhanced FLASH Microcontrollers with CAN*, 2003. `http://www.microchip.com`.

[19] Microchip. *Página oficial de Microchip.* World Wide Web, `http://www.microchip.com`, 2003. Último acceso: 11/07/2004.

[20] Labcenter Electronics. *Página oficial de Labcenter Electronics.* World Wide Web, `http://www.labcenter.co.uk`, 2004. Último acceso: 15/09/2004.

[21] Microchip. *DS51288B. MPLAB C18 C Compiler User´s Guide.* Microchip Technology Inc., 2003.

[22] N. Rajbharti. DS00738B. PIC18C CAN Routines in C. Technical report, Microchip Technology Inc., 2001.

[23] Microsoft. *Página oficial de Microsoft sobre Visual Basic.* World Wide Web, `http://msdn.microsoft.com/vbasic`, 2004. Último acceso: 12/06/2004.

[24] MAXIM Dallas Semiconductor. *Página oficial de Dallas Semiconductor, MAXIM.* World Wide Web, `http://www.maxim-ic.com`, 2004. Último acceso: 15/09/2004.

[25] P. Richards. DS00228A. A CAN physical layer discussion. Technical report, App. Notes Microchip Inc., 2002.

[26] ISO. *ISO11898 Road vehicles – Controller area network (CAN) –.* International Standards Organization, 2003.

[27] CAN in Automation (CiA). *CANOpen, Cabling and Connector Pin Assignment*, 2001. `http://www.can-cia.de`.

[28] P. Richards. DS00754A. Undestanding Microchip´s CAN Module Bit Timing. Technical report, Microchip Technology Inc., 2001.

[29] Robert Bosch GmbH. The configuration of the can bit timing. Technical report, Robert Bosch GmbH. Último acceso: 17/08/2004.

[30] CANKingdom. *Página oficial de CANKingdom*. World Wide Web, `http://www.cankingdom.org`, 2004. Último acceso: 11/07/2004.

[31] CompactFlash Association. Compactflash specification revision 2.1. Technical report, CompactFlash Association, 2004. Último acceso: 17/08/2004.

[32] SSFDC Forum Technical Committee. Smartmediatm physical specifications web-online version 1.00. Technical report, Toshiba Corporation, 1999. Último acceso: 17/08/2004.

[33] SSFDC Forum Technical Committee. Smartmediatm electrical specifications web-online version 1.00. Technical report, Toshiba Corporation, 1999. Último acceso: 17/08/2004.

[34] SSFDC Forum Technical Committee. Smartmediatm physical format specifications web-online version 1.00. Technical report, Toshiba Corporation, 1999. Último acceso: 17/08/2004.

[35] SSFDC Forum Technical Committee. Smartmediatm logical format specifications web-online version 1.00. Technical report, Toshiba Corporation, 1999. Último acceso: 17/08/2004.

[36] SDA Technical Committee. Sd memory card specification. Technical report, Secure Digital Association, 2003. Último acceso: 17/08/2004.

[37] SanDisk. Secure Digital Card Product Manual, Rev 1.0. Technical report, SanDisk Corporation Headquarters, 2003. Último acceso: 17/08/2004.

[38] J. Dobiash. *FAT16 Structure Information*. World Wide Web, `http://home.teleport.com/~brainy/fat16.htm`, 1999. Último acceso: 17/08/2004.

[39] L. Wolcott. *FAT16 File System Driver for CompactFlash*. World Wide Web, `http://www.larrywolcott.com/FAT16/FAT16Driver.html`, 2004. Último acceso: 17/08/2004.

[40] Microchip. *DS51297B. MPLAB C18 C Compiler Libraries*. Microchip Technology Inc., 2003.