E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

Diseño de una red CAN bus con Arduino

Grado en Ingeniería
en Tecnologías Industriales

Trabajo Fin de Grado

Autor: Alejandro García Osés

Tutor: Ignacio del Villar Fernández

Pamplona, 24 de Junio de 2015

upna Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

1. RESUMEN **4**

2. OBJETIVOS **4**

3. MOTIVACIÓN DEL TRABAJO **4**

4. ANTECEDENTES **5**

4.1 Origen de CAN 5
4.2 Capas de la pila OSI 5
4.2.1 Capa de enlace de datos *6*
4.2.2 Capa física *7*
4.3 Trama de CAN bus 8
4.3.1 Tipos de tramas de CAN bus *10*
4.4 Sincronización 11
4.4.1 Tipos de sincronización *12*

5. METODOLOGÍA **14**

5.1 Hardware utilizado 14
5.2. Software utilizado 18

6. DESARROLLO DEL TRABAJO **19**

6.1 Planteamiento Nº1 19
6.2 TTCAN (Time-triggered CAN) 31
6.2.1 Funcionamiento básico *31*
6.2.2 Ventajas de TTCAN *33*
6.2.3 Implementación de TTCAN *33*
6.2.4 Conclusiones *34*
6.3 Planteamiento Nº 2 34
6.4 Desarrollo final 47
6.4.1 Implementación sin LCD *47*
6.4.2 Implementación con LCD *53*
6.4.3 Resultados *58*
6.5 Adquisición de datos de un vehículo con CAN Bus 60
6.5.1 Modelo del vehículo y protocolo utilizado *60*
6.5.2 Desarrollo del programa para la adquisición de datos *64*

7. PRESUPUESTO **71**

7.1 Gastos materiales 71
7.2 Gastos mano de obra 71
7.3 Gasto total 72

8. CONCLUSIONES **72**

ANEXOS **73**

Librería CAN-BUS Shield 73
Archivo mcp_can.h *73*
Archivo mcp_can.cpp *75*
Archivo mcp_can_dfs.h *92*
Librería Display LCD de Adafruit 98
Archivo Adafruit_MCP23017.h *98*
Archivo Adafruit_MCP23017.cpp *99*
Archivo Adafruit_RGBLCDShield.h *103*
Archivo Adafruit_RGBLCDShield.cpp *105*

BIBLIOGRAFÍA **112**

LISTA DE PALABRAS CLAVE

- Arduino UNO
- CAN bus
- OBD-II
- PDI
- CAN-Bus shield
- Display LCD
- DB9
- TTCAN
- ISO 9141
- MCP2515
- MCP2551
- Comunicación ISP
- Comunicación I2C
- Automoción
- Comunicaciones industriales
- Adquisición de datos

1. Resumen

El presente trabajo tiene como objetivo la creación de una red de CAN bus mediante la utilización de Arduino y de un shield. Un shield es un módulo externo que, acoplado en la tarjeta del microcontrolador, permite realizar determinadas funciones que no podría por sí solo. En este caso, el shield es el encargado de la comunicación con el protocolo CAN, ya que hace de interfaz entre el puerto serial del Arduino y CAN bus. Está paso servirá para acometer la fase final del trabajo, la monitorización de datos de la red Can bus de un vehículo utilizando un cable OBD2. Dichos datos serán mostrados en un display LCD.

2. Objetivos

El trabajo presenta dos objetivos fundamentales:

- Intercambiar datos entre hasta tres nodos Arduino, con protocolo CAN bus: una vez evaluados el envío y la correcta recepción entre los distintos nodos, se utilizará un display LCD para poder visualizar los datos que se intercambian a través de la red.

- Monitorización de distintos parámetros de un vehículo tales como las revoluciones por minuto o la velocidad: para poder monitorizarlos, se usarán al igual que antes, una tarjeta Arduino que, junto con el display LCD, permitirá representar de forma visual los datos del vehículo.

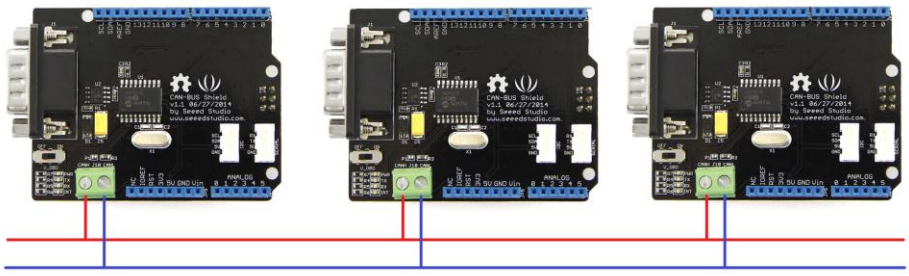

Figura 2. Esquema red CAN con 3 nodos

3. Motivación del trabajo

Se ha optado por la realización de este trabajo dado que el uso del protocolo CAN bus está muy extendido en la automoción y en la industria. De hecho CAN bus establece la base para determinados protocolos utilizados en redes industriales, como DeviceNet o

CANOpen. La sencillez de CAN junto con su robustez, hacen que esté muy arraigado a la tecnología y que por lo tanto, es difícil que desaparezca a lo largo de los años.

Además, el creciente mercado del automóvil augura un buen futuro a este protocolo, ya que es clave para mantener la seguridad en el vehículo y para la comunicación entre la cada vez más extensa gama de dispositivos y sensores que existen en un automóvil.

4. Antecedentes

4.1 Origen de CAN

El bus CAN (Controller Area Network), surge de la necesidad en los ochenta de encontrar una forma de interconectar los distintos dispositivos de un automóvil de una manera sencilla y reduciendo significativamente las conexiones. De esta forma, la empresa Robert Bosch GmbH logra desarrollar el bus CAN, que posteriormente se estandariza en la norma ISO 11898-1.

4.2 Capas de la pila OSI

Dentro de la pila OSI, CAN bus cubre la capa de Enlace de datos y la Física. En concreto la estandarización divide las diferentes capas. Por ejemplo, ISO 11898-1 cubre la capa de enlace de datos, mientras que la física se divide en dos dependiendo de la velocidad: ISO 11898-2 para alta velocidad e ISO 11898-3 para baja velocidad.

Figura 3.1. Pila de protocolo OSI [1]

4.2.1 Capa de enlace de datos

El protocolo de acceso al medio utilizado es el CSMA/CD + AMP (Carrier Sense Multiple Access/ Collision Detection + Arbitration on Message Priority). En dicho protocolo, los distintos nodos conectados realizan una escucha del medio, de forma que conocen cuándo un mensaje está siendo transmitido. De esta forma, los nodos evitarán mandar una trama de datos cuando la red está ocupada. Puede darse la circunstancia en la que dos nodos manden mensajes a la vez. Este problema se solventa mediante prioridad. En caso de colisión, el nodo con el identificador más bajo será el que logre enviar el mensaje. En la figura 3.2 se muestra cómo funciona el campo de arbitrio:

Cada nodo (A, B y C) lee el valor del bus y lo compara con el enviado, si el valor del bus está en estado dominante y el nodo tiene un valor recesivo, este deja de transmitir. El campo de arbitrio se encuentra al comienzo de la trama, luego es al comienzo donde se decide quién tiene la mayor prioridad.

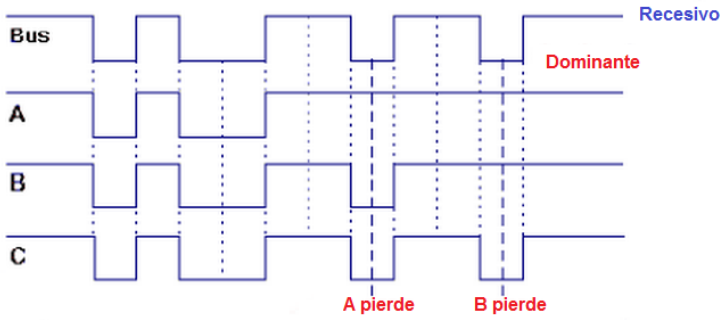

Figura 3.2. Señal de la red CAN bus y sus diferentes nodos [2]

Dependiendo del protocolo, CAN bus posee uno o varios búferes de transmisión y/o recepción. Dichos búferes permiten controlar el tráfico de mensajes mediante filtros y máscaras. La aceptación o no de un mensaje funciona mediante el mecanismo representado en la siguiente figura:

	Nodo 1	Nodo 2
Identificador	10110	10111
Máscara	00111	00111
AND	00110	00111
Filtro	00111	00111
¿Coinciden?	NO	SÍ

Se realiza un AND entre el identificador del mensaje y la máscara del búfer, si el resultado coincide con el filtro del búfer, el mensaje será aceptado.

4.2.2 Capa física

El medio de comunicación deberá ser capaz de transmitir y recibir varias señales a la vez, ya que el objetivo de CAN bus es la intercomunicación entre varios nodos. Además, debe ser capaz de representar los estados recesivo y dominante. La capa física se puede dividir en tres subcapas:

- Physical Signaling Layer (PSL)

Es la encargada de la sincronización de los bits y temporización de los bits

- Physical Medium Attachment (PMA)

Convierte los niveles lógicos de transmisión y recepción a los que vienen indicados por el protocolo CAN bus.

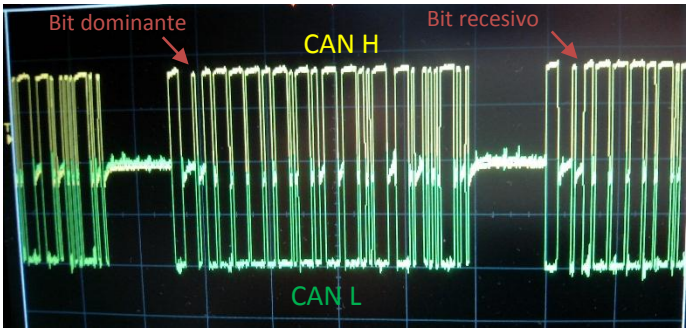

Figura 3.3. Representación de una trama de CAN Bus

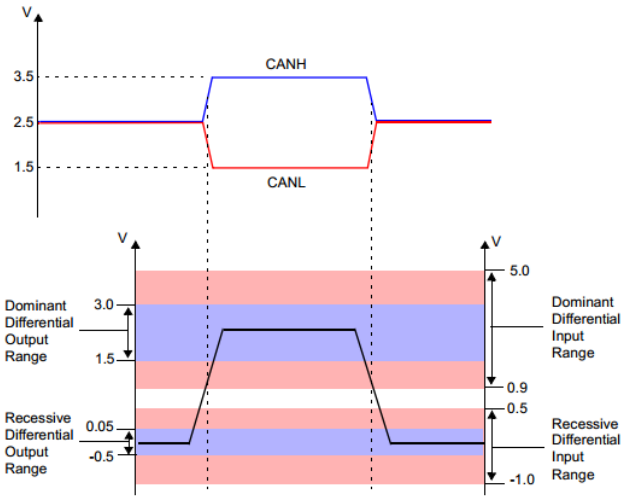

Figura 3.4. Señales CAN-H y CAN-L de un fragmento de trama (Arriba) y su diferencia (Abajo) [3]

La comunicación se realiza de modo diferencial mediante las señales CAN-H y CAN-L. La diferencia de ambas da como resultado una señal entre 0 y 2 voltios que indicará el nivel lógico de un bit. El modo diferencial permite el rechazo al ruido.

- Medium Dependent Interface (MDI)

La capa de interfaz dependiente del medio indica el conector y el medio de transmisión. El estándar ISO 11898-2 especifica lo siguiente:

- Dos cables con resistencias de terminación de 120Ω
- Cables trenzados o apantallados
- Evitar derivaciones
- Utilización de diferentes longitudes de línea mínimas en base a la velocidad

Además, habrá que tener en cuenta las tolerancias de los distintos elementos de la capa física, como el oscilador de cuarzo.

CAN in Automation (CiA) incluye otras características, como el uso de conectores DB9, el cual utilizaremos en la segunda parte del trabajo.

4.3 Trama de CAN bus

La trama CAN bus posee diferentes campos que informan de diferentes aspectos del mensaje. A continuación se exponen los campos de una trama CAN. Las tramas utilizan bit stuffing, es decir, cuando se suceden 5 bits iguales, se introduce un bit extra de valor contrario para evitar la desincronización. Como se verá más adelante, existen dos tipos de tramas, estándar y extendida, en base a número de bits del identificador. Durante el trabajo se utilizarán tramas estándar dado que al usar tan solo tres nodos no se necesita un gran campo de arbitrio.

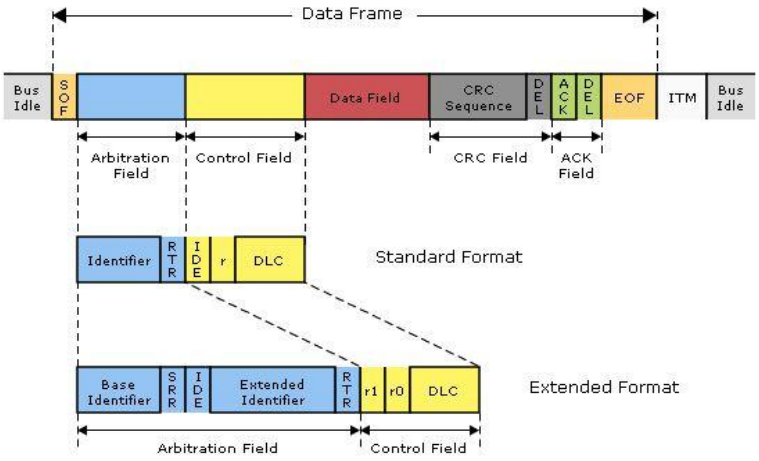

Figura 3.5. Trama CAN Bus [4]

SOF (Start of Frame bit)

Indica el comienzo del mensaje y permite la sincronización de todos los nodos conectados a la red. Este bit tiene estado dominante (0 lógico).

Campo de arbitrio

Está formado por 12 bits o 32 bits dependiendo del tipo de trama. Dentro del campo se encuentra el identificador, el cual indica la prioridad del nodo. El nodo con mayor prioridad es aquel que tiene el identificador más bajo. El bit RTR se utiliza para distinguir entre una trama remota o una trama de datos. Se expondrán más adelante las diferentes tramas de CAN Bus.

Campo de control

Formado por 6 bits. El bit IDE indica con un estado dominante que la trama enviada es estándar. El bit RB0, está reservado y se establece en estado dominante por el protocolo CAN.

El resto de bits, el Data Length Code (DLC) indica el número de bytes de datos que contiene el mensaje. Una trama extendida tiene un bit adicional RB1.

Campo de datos

Puede estar formado por hasta 8 bytes, dependiendo de lo que especifiquemos en el DLC. En este campo están contenidos los datos del mensaje.

Campo de verificación por redundancia cíclica

Este campo de 15 bits, detecta errores en la transmisión del mensaje. Se delimita con un bit final en estado recesivo.

Campo de reconocimiento

El último campo de la trama, está formado por 2 bits. El nodo transmisor manda una trama con el bit de ACK (Acknowledge) en estado recesivo, mientras que los receptores, si han recibido el mensaje correctamente, mandarán un mensaje en estado dominante. Contiene un bit delimitador.

Campo de fin de trama

Una serie de 7 bits recesivos indican el fin de la trama.

4.3.1 Tipos de tramas de CAN bus

Las tramas CAN bus se clasifican principalmente en base al tamaño de su identificador:

Trama estándar

La trama estándar tiene un identificador de 11 bits con el cual se establece la prioridad de los diferentes nodos de la red.

Trama extendida

La trama extendida difiere en la estándar en que el campo identificador tiene un total de 29 bits para el identificador además de dos bits extra: SSR y RB1, en estado recesivo y dominante respectivamente. El primer bit está situado al final de los 11 primeros bits del identificador mientras que el RB1 está situado antes que el bit RB0. Al añadir 20 bits extra a la trama, no se utiliza en redes que requieren de un desempeño muy alto.

Adicionalmente las tramas se pueden clasificar según su función:

Trama de datos

Como su propio nombre indica, dichas tramas se utilizan para enviar datos a través de la red. Los datos se incluirán en el campo de datos y pueden tener una extensión de 0 a 8 bytes.

Trama remota

Un nodo tiene la capacidad de solicitar un mensaje de otro nodo usando tramas remotas. El identificador de la trama debe ser el mismo que el del nodo del cual se quiere recibir el mensaje. Además el campo de datos será 0. Una vez que el nodo receptor reciba el mensaje, éste enviará sus datos.

Trama de error

Se genera al detectar un error en la red por parte de un nodo. Está formada por un campo indicador de error y un campo delimitador.

Trama de saturación (Overload frame)

A diferencia de la trama de error, la trama de saturación sólo se da entre tramas. Es generada al detectar un bit dominante en el espacio entre tramas o al no ser posible el envío de un mensaje por problemas internos.

4.4 Sincronización

CAN bus utiliza codificación NRZ (Non Return to Zero), luego no existe la posibilidad de incluir una señal de reloj dentro del flujo de datos, como por ejemplo en codificación Manchester. Por lo tanto los nodos sincronizan su reloj interno al del nodo transmisor. Esto se consigue tratando que el tiempo nominal de bit de cada uno de los nodos sea igual.

La unidad mínima de tiempo viene dada por la frecuencia del cristal y se denomina cuanto de tiempo (time quantum). Viene definido por la siguiente ecuación:

$$TQ = \frac{2 * BRP}{F_{OSC}}$$

Siendo BRP un preescalador programable y F_{OSC} la frecuencia de oscilación del reloj.

El tiempo nominal de bit viene dado por 4 segmentos:

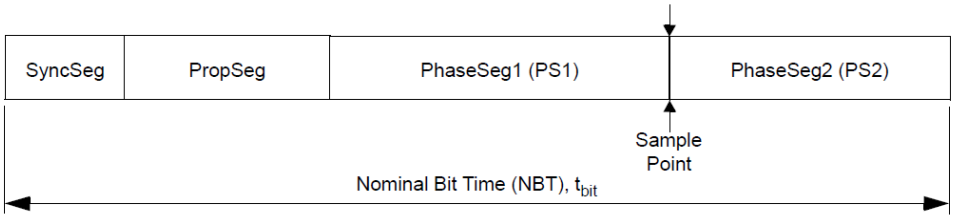

Figura 3.6. Tiempo nominal de bit y sus partes [5]

Syncseg

Es el primer segmento del tiempo nominal de bit. Es la zona donde se espera recibir el flanco de subida o bajada de la señal. Se usa para la sincronización de todos los nodos de la red y tiene una longitud fija de 1TQ (Time quantum)

Propseg

Este segundo segmento se utiliza para compensar los posibles retardos de señal ocasionados por el medio físico. Se le puede asignar un valor de 1 a 8 TQ.

PhaseSeg1 y PhaseSeg2

Los dos segmentos de fase compensan los posibles retardos de fase de la señal. Ambos segmentos son variables, con valores de 1 a 8 TQ y de 2 a 8 TQ respectivamente

Punto de muestreo (Sample point)

Es aquel punto del bit nominal donde se interpreta si el bit está en estado dominante o recesivo.

4.4.1 Tipos de sincronización

Hard synchronization

La sincronización denominada hard actúa cuando hay un cambio de estado de recesivo a dominante en el bus mientras éste se encuentra en estado ocioso (recesivo), es decir, cuando no está transmitiendo. Este cambio de estado viene dado por el SOF (Start of Frame bit), que indica el comienzo de la transmisión de un mensaje. Esta señal permite una sincronización de todos los nodos de la red fijando el valor de transición del bit dentro del segmento de sincronización (Syncseg).

Resynchronization

Como hemos visto, la Hard Synchronization solo actúa durante el comienzo de la transmisión, pero no es suficiente controlar el tiempo de bit, ya que pueden existir pequeños desfases entre el receptor y transmisor durante el envío de la trama. Para evitar esto, se utiliza la denominada Resincronización (Resynchronization), que corrige los pequeños desajustes de fase que puedan ocurrir.

Como hemos visto, el objetivo es que la transición de recesivo a dominante se dé dentro del Syncseg para que el punto de muestreo no interprete un estado de bit equivocado. No

obstante, puede haber casos en los que la transición se de en otro segmento del tiempo nominal de bit. Para corregir esto, se ajusta el punto de muestreo modificando la longitud de los segmentos PhaseSeg1 y PhaseSeg2. Dichos segmentos de pueden modificar (acortar o alargar) hasta una longitud dada por el Synchronization Jump Width (SJW).

En base a si el estado de transición se produce dentro de SyncSeg, antes del punto de muestreo o después, se modificarán los segmentos PhaseSeg1 y PhaseSeg2 de manera distinta:

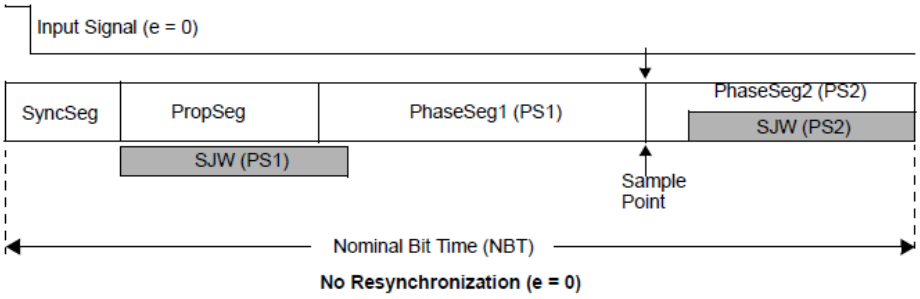

Figura 3.7. Error de fase nulo [5]

En este primer caso no existe retardo de fase, ya que el estado de transición se da dentro del segmento de sincronización luego no se modifican los segmentos de fase.

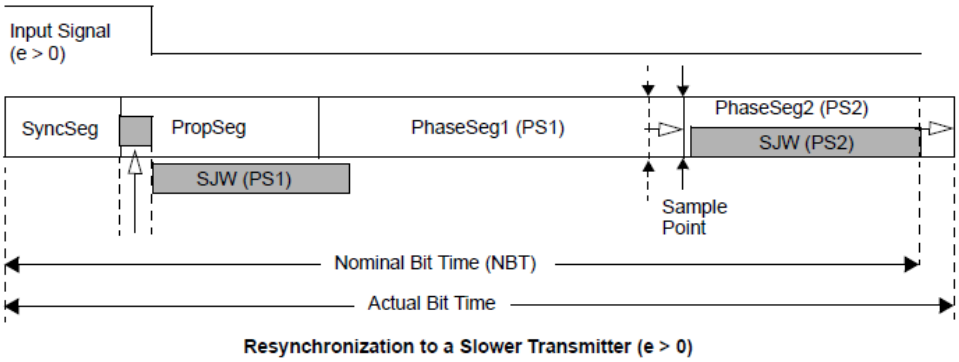

Figura 3.8. Error de fase positivo [5]

En este caso, el estado de transición se da antes del punto de muestro y fuera del segmento de sincronización. Para corregir dicha situación, se modifica el punto de muestreo, alargando PhaseSeg1 una cantidad igual al error de fase.

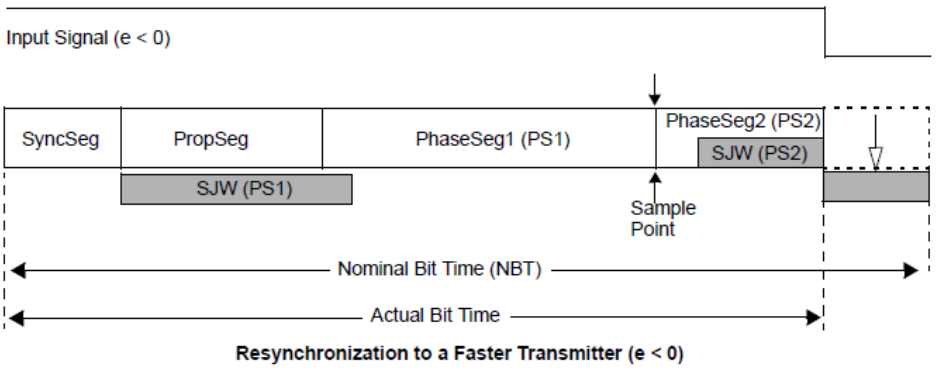

Figura 3.9. Error de fase negativo [5]

En éste último caso, el estado de transición se da después del punto de muestreo del bit anterior, por lo tanto se sustrae a PhaseSeg2 el error de fase.

5. Metodología

5.1 Hardware utilizado

Arduino UNO

El Arduino UNO es una placa de microcontrolador muy popular por su sencillez, coste y dimensiones. Utiliza el microcontrolador ATmega328P, fabricado por ATMEL. Sus especificaciones son las siguientes:

- Voltaje: 5V
- Voltaje de entrada (recomendado): 7-12V
- Voltaje de entrada (límites): 6-20V
- Pines de entradas y salidas digitales: 14
- Pines de entrada analógica: 6
- Corriente en pines de entrada-salida: 40mA
- Corriente en pin de 3.3V: 50mA
- Memoria Flash: 32KB
- SRAM: 2KB
- EEPROM: 1KB
- Velocidad reloj: 16MHz
- Dimensiones: 68.6x53.4mm

Figura 4.1. Arduino UNO [6]

Se ha optado por su utilización debido a que la universidad posee estás placas para su uso en prácticas. Se podría haber utilizado otro modelo de Arduino o incluso un PIC más sencillo, debido a que no se requieren muchas salidas o entradas digitales.

CAN-BUS Shield

El shield de CAN bus de Seeed Studio es una tarjeta especialmente creada para que una tarjeta Arduino o Seeeduino pueda trabajar con el protocolo CAN. Sus especificaciones son las siguientes:

- Voltaje: 5V
- Dimensiones: 68x53mm
- Peso: 50g

Tal y como se observa, las dimensiones encajan perfectamente con las del Arduino UNO. Con los pines de la parte inferior se puede acoplar a éste sin necesidad de soldaduras. Es posible su conexión con otro tipo de modelos de Arduino, como el Arduino Mega o el Arduino Leonardo, aunque para ello es necesario realizar un conexionado diferente el cual viene indicado en la página web del producto **[7]**.

El shield tiene varias partes diferenciadas, las más importantes son las siguientes:

- MCP2515
- MCP2551
- Comunicación ISP
- Cristal 16MHz
- Conector DB9
- Terminal CAN

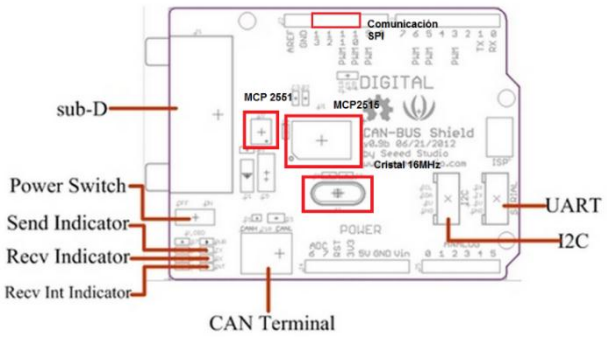

Figura 4.2. Partes del shield CAN-BUS [7]

Al igual que el Arduino UNO, se ha escogido este shield debido a que la universidad los poseía con anterioridad. Pese a esto, el shield de seeedstudio se muestra óptimo para redes CAN bus. Además otros modelos de shield que se encuentran en el mercado contienen los mismos componentes principales, luego no difieren mucho entre sí **[8]**.

MCP2515

El MCP2515 es un circuito integrado fabricado por Microchip el cual realiza las funciones de controlador. Implementa CAN V2.0B y CAN V2.0A con una velocidad de hasta 1Mbps. Tiene unas características las cuales hacen que se pueda utilizar también en variantes de

CAN, como el protocolo TTCAN (time-triggered CAN), el cual nos resultará muy útil en el posterior desarrollo del trabajo.

MCP2551

El circuito integrado MCP2551, también fabricado por Microchip, funciona como transceptor, es decir, hace de interfaz entre el controlador de CAN bus y el bus físico.

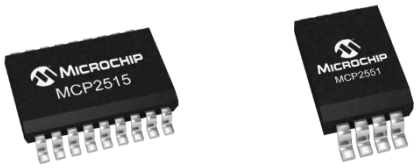

Figura 4.4. MCP2515 (izquierda); MCP2551 (derecha) [13]

Comunicación SPI

La comunicación entre el microcontrolador de la placa Arduino (ATMega328P) y el shield, se realiza mediante SPI (Serial Peripheral Interface), un tipo de comunicación síncrona entre dispositivos electrónicos, la cual necesita de 4 pines. En concreto Arduino UNO utiliza los pines 13, 12, 11 y 10, que corresponden con los terminales SCK, MISO, MOSI y SS respectivamente.

Cristal de cuarzo de 16MHz

El cristal permite el correcto funcionamiento de la tarjeta y establece la unidad mínima de tiempo para la sincronización de los bits durante la comunicación.

Conector DB9

El conector DB9 permite la comunicación con la un dispositivo externo. En el caso del presente trabajo, se utilizará para comunicar el shield de seeedstudio con un vehículo.

Terminal CAN

Consta de dos terminales, CAN-H y CAN-L, que conectados con el terminal correspondiente en otro nodo, permiten el intercambio de datos.

Display LCD

Para la monitorización de los datos, se ha utilizado el shield LCD de Adafruit, el cual solamente requiere de 2 pines para su comunicación con la placa de microcontrolador. Normalmente, las placas LCDs requieren de hasta 9 pines digitales, 6 para el control del

LCD y 3 para el control de la luminosidad. No obstante, el shield de Adafruit necesita solamente 2 conexiones para el control total del display, ya que la comunicación se realiza mediante conexión I2C.

El circuito integrado que viene junto con el shield, permite controlar hasta 16 pines de entrada y/o salida con solo una conexión I2C, de esta manera sobran 7 pines (16 – 9) para poder conectar botones con los que interactuar con el display.

Figura 4.5. Display LCD de Adafruit [15]

Se ha elegido este shield debido a los pocos pines que se necesitan, quedando libres las salidas y entradas del Arduino para monitorización de sensores o control de actuadores, en caso de que fuese necesario. No obstante, una desventaja existente es la limitada velocidad del protocolo I2C comparada con el protocolo SPI que utiliza el shield de CAN bus.

Conectores

Cables Dupont

Se han utilizado cables Dupont para la conexión entre los diferentes nodos por su sencillez a la hora de conectarlos.

Figura 4.6. Cables Dupont [16]

Cable OBD2-DB9

Para la comunicación entre el nodo CAN y el vehículo, se utilizará un cable con un conector DB9 hembra y un conector OBD2 macho. El conector OBD2 es el que todos los automóviles usan por defecto para la monitorización de sus datos.

Figura 4.7. Cable OBD2-DB9 y conexionado de pines [17]

Cable USB tipo A/B

Para la comunicación entre el ordenador y el Arduino Uno es necesario un cable USB de tipo A/B. Con este cable se programará el microcontrolador del Arduino y también se monitorizarán los datos enviados al ordenador desde la placa Arduino. En otras palabras la comunicación puede ser bidireccional: se puede programar desde el ordenador y el ordenador puede recibir datos de la tarjeta, aunque no se puedan realizar las dos tareas al mismo tiempo.

Figura 4.8. Cable USB tipo A/B [18]

5.2. Software utilizado

Arduino IDE

Es el software que se utiliza por defecto para la programación con Arduino. Este software utiliza un lenguaje C++ simplificado. Con una serie de funciones que contiene Arduino por defecto al descargar el IDE, se pueden explotar la mayoría de funcionalidades del

microcontrolador. Esto es debido a una serie de librerías escritas en C++ creadas expresamente para que la herramienta Arduino y su software sean sencillos de utilizar. Sin embargo, si se adquiere un dispositivo externo como puede ser el caso de un shield, es necesario conseguir una librería creada específicamente para dicho dispositivo.

Si quisiéramos crear dicha librería por nuestra cuenta, se necesitaría un programa más complejo y orientado a programación de más alto nivel.

Arduino IDE es adecuado para usuarios que estén iniciados en el mundo de los controladores o que no necesitan realizar un proyecto excesivamente complejo. Esto supone una serie de desventajas para alguien que quiera trabajar a más nivel, ya que no permiten crear proyectos, no se pueden buscar variables o funciones de forma rápida, y no permite abrir archivos con extensión ".cpp" o ".h". Otro inconveniente es la inexistencia de un debugger oficial de Arduino, siendo necesarios dispositivos externos como el AVR JTAGICE mkII o ATMEL Dragon.

Eclipse

Para poder programar a alto nivel y crear y modificar las librerías de Arduino se ha optado por la utilización de Eclipse. Este programa ofrece una interacción más amigable con el usuario, indicando errores de forma precisa y dando la posibilidad de navegar entre archivos rápidamente.

Existe la posibilidad de programar en Eclipse en vez de con el IDE oficial, no obstante, no ha sido necesario, ya que no ha requerido crear una librería desde cero, sino que solamente se han modificado partes de ella.

Existen otras herramientas para el desarrollo de código en C++, como NetBeans, no obstante se ha utilizado Eclipse por la menor carga computacional que este presenta.

6. Desarrollo del trabajo

6.1 Planteamiento Nº1

De acuerdo con lo indicado en la sección de objetivos, se realizará la comunicación de varios nodos CAN bus conectados en red. Concretamente se conectarán tres nodos mediante comunicación CAN bus. Estos tres nodos podrían asemejarse a diferentes partes del coche, como un sensor de temperatura, un sensor de velocidad y el sistema de control del ABS. Hay que tener en cuenta que el tiempo es algo muy crítico ya que una rápida

valoración de los datos resulta en una mayor anticipación a los problemas y por lo tanto una mayor seguridad. Cada nodo tendrá una ID única que lo distinguirá de los demás y que le otorgará una mayor o menor prioridad en el caso de que dos nodos manden mensajes al mismo tiempo.

La primera alternativa pensada fue el envío constante de datos por parte de todos los nodos, de manera que habrá colisiones que serán resueltas por la prioridad que otorga el campo identificador. El problema que presenta esta solución es la aleatoriedad en la que los nodos envían sus mensajes. Si las colisiones entre nodos son habituales, solo el nodo con identificador más alto logrará mandar su mensaje, lo cual hace que los nodos con identificador bajo tengan pocas posibilidades de mandar un mensaje a la red:

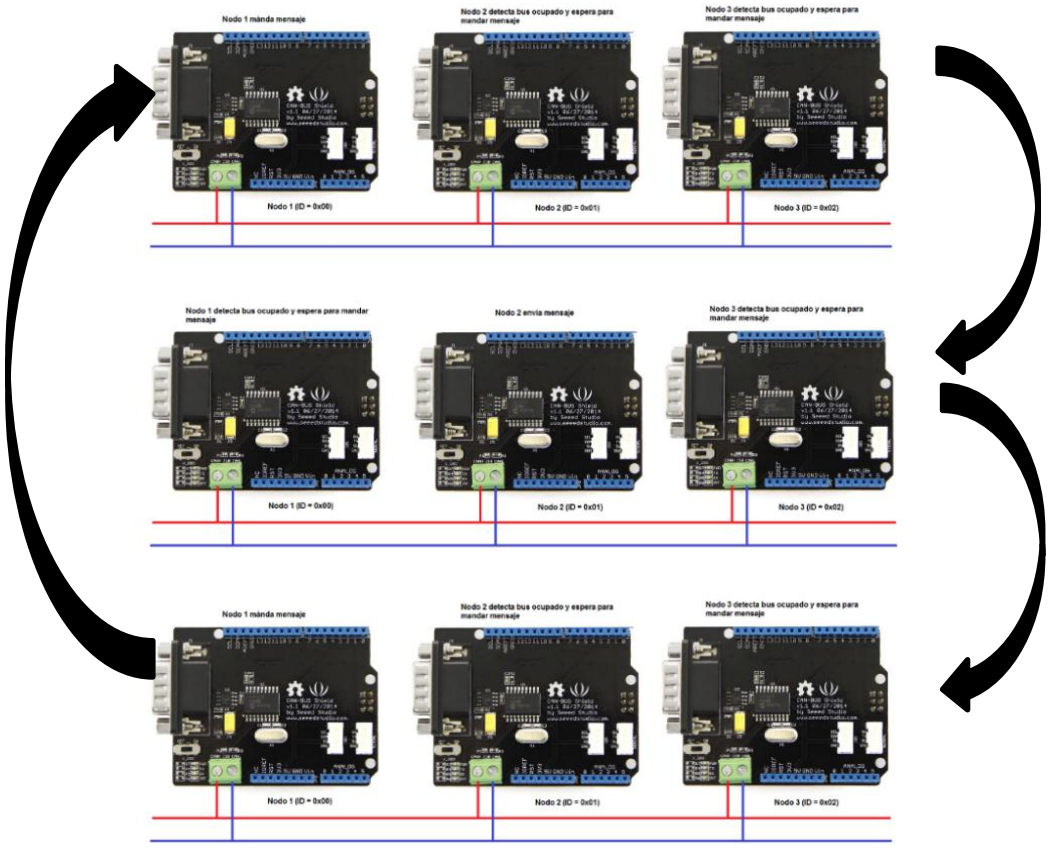

Figura 6.1. Esquema funcionamiento programa

Como se observa, dado el alto identificador del nodo 3, éste permanecerá en espera continuamente mientras otros nodos con mayor prioridad manden sus mensajes. Estadísticamente, habrá algún momento en el que el nodo 3 logre enviar su mensaje debido a que la orden de envío de un mensaje se haya dado antes que la de un nodo con mayor prioridad. En un sistema en el que los datos deben ser recibidos continuamente, no se puede permitir que un nodo se comunique en momentos aleatorios

Teniendo en cuenta lo anterior, se llega a la conclusión de que los nodos deben respetar una serie de turnos, es decir seguir una secuencia a la hora de mandar los mensajes para evitar colisiones y asegurar que cualquier nodo pueda comunicarse.

En primer lugar se realizará la comunicación entre dos nodos para confirmar que la comunicación es correcta. A los nodos se les asignaran dos identificadores 0x10 y 0x20. El diagrama de flujo del programa es el siguiente:

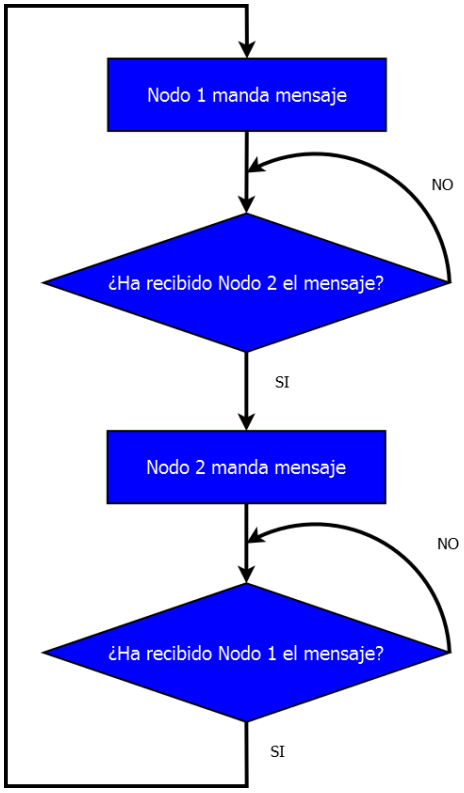

Figura 6.2. Diagrama de flujo 1

A la hora de programar se utilizará la librería que acompaña al Shield CAN-BUS **[19]**. Debido a que las funciones públicas que posee son limitadas, se crean dos adicionales:

```
/*********************************************************************************************************
 ** Function name:           DisGloInt
** Descriptions:            Enable global interrupts
*********************************************************************************************************/
INT8U MCP_CAN::DisGloInt(void)
{
    mcp2515_setRegister(0x2B,0x00 );

}
```

Figura 6.3. Código de la función DisGloInt()

```
/*********************************************************************************************************
 ** Function name:           setRcvInt
** Descriptions:            Enable Interrupts when message arrives if true
*********************************************************************************************************/

INT8U MCP_CAN::setRcvInt(bool res)
{
    if(res){
        mcp2515_setRegister(0x0C,0xFF );
        mcp2515_modifyRegister(0x2B,0x03,0x03 );
        }
    else {
            mcp2515_setRegister(0x0C,0x00 );
            mcp2515_modifyRegister(0x2B,0x03,0x00 );

        }

}
```

Figura 6.4. Código de la función setRcvInt()

La función DisGloInt() deshabilita las interrupciones globales del microcontrolador MCP2515, de esta forma nos aseguramos que el programa no salte a la rutina de atención a la interrupción cuando no lo deseemos.

La función setRcvInt() habilita la interrupción cuando se ha recibido un mensaje que haya sido aceptado por el filtro y la máscara.

Dentro del código de las funciones nos encontramos con otras denominadas mcp2515_setRegister y mcp2515_modifyRegister, funciones privadas las cuales se encuentran por defecto en la librería del shield CAN bus. La primera es utilizada para modificar un registro del microcontrolador: tiene dos argumentos, el primero indica la dirección del registro y el segundo el valor que queremos darle. La otra solamente varía partes del registro: el primer argumento indica la dirección del registro, el segundo el valor de la máscara y el tercero el valor que queremos dar al registro teniendo en cuenta la máscara. La máscara actúa de la siguiente forma:

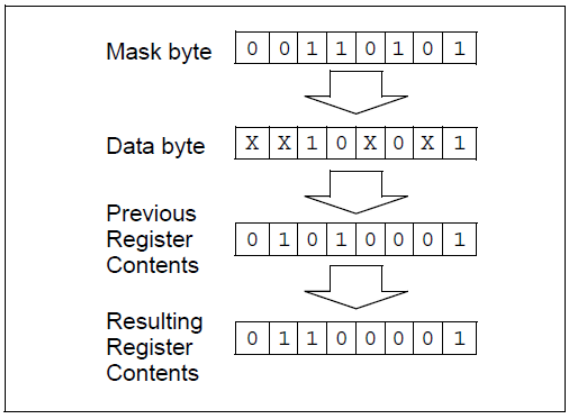

Figura 6.5. Funcionamiento máscaras y filtros [5]

Se han modificado los registros 0x2B y 0x0C:

El registro 0x2B habilita las interrupciones del microcontrolador:

REGISTER 7-1: CANINTE – INTERRUPT ENABLE (ADDRESS: 2Bh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
MERRE	WAKIE	ERRIE	TX2IE	TX1IE	TX0IE	RX1IE	RX0IE
bit 7							bit 0

Figura 6.6. Registro CANINTE [5]

El registro 0x0C modifica las funciones de los pines RX1BF y RX0BF:

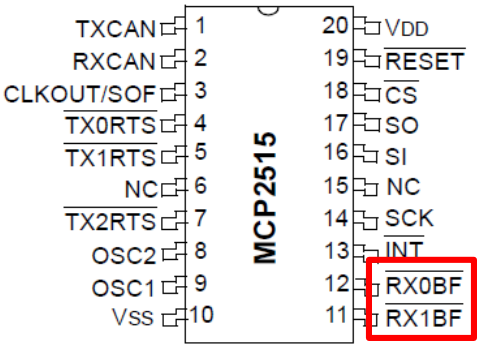

Figura 6.7. Pines del controlador MCP2515 [5]

REGISTER 4-3: BFPCTRL – RXnBF PIN CONTROL AND STATUS (ADDRESS: 0Ch)

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	B1BFS	B0BFS	B1BFE	B0BFE	B1BFM	B0BFM
bit 7							bit 0

Figura 6.8. Registro BFPCTRL [5]

Los pines RX1BF y RX0BF tienen distintas configuraciones, siendo una de ellas la posibilidad de trabajar como pines de interrupción ante la llegada de un mensaje al buffer. Nos interesa dicha funcionalidad, luego ponemos los dos últimos bits a 1.

El procedimiento para modificar la librería en Eclipse es el siguiente:

En primer lugar se requiere poder programar en C++ dentro de eclipse: Para ello es necesario un plugin que instalamos siguiendo los siguientes pasos: Nos dirigimos al menú (Help > Install New Software…) y en la ventana que aparece pulsamos “Add…”. En el campo “Name” escribimos “cdt” y en “Location” pegamos el siguiente enlace: http://download.eclipse.org/tools/cdt/releases/8.6. Pulsamos OK, marcamos las casillas que nos aparecen y finalmente pulsamos Next.

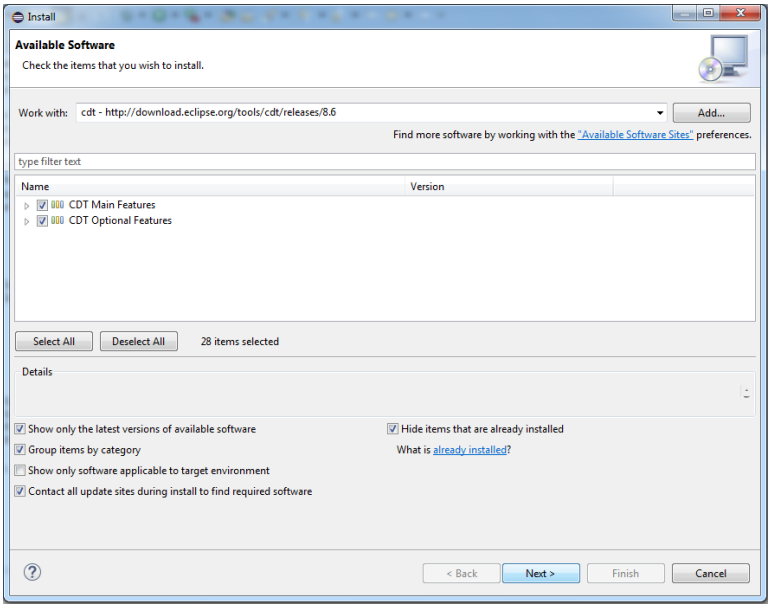

Figura 6.9. Menú de instalación de software en Eclipse

Ya podemos abrir los archivos con extensión *.cpp y *.h de la librería. En primer lugar abriremos el archivo mcp_can.h donde nos encontraremos dos tipos de funciones, privadas y públicas. Las funciones públicas son las únicas que se pueden utilizar desde el Arduino IDE luego las funciones que creemos serán públicas. Para crear una función nueva simplemente pegamos el nombre de la función junto con los argumentos y su tipo.

```
public:
    MCP_CAN(INT8U _CS);
    INT8U begin(INT8U speedset);                                    /* init can                  */
    INT8U init_Mask(INT8U num, INT8U ext, INT32U ulData);           /* init Masks                */
    INT8U init_Filt(INT8U num, INT8U ext, INT32U ulData);           /* init filters              */
    INT8U sendMsgBuf(INT32U id, INT8U ext, INT8U rtr, INT8U len, INT8U *buf);   /* send buf                  */
    INT8U sendMsgBuf(INT32U id, INT8U ext, INT8U len, INT8U *buf);   /* send buf                  */
    INT8U readMsgBuf(INT8U *len, INT8U *buf);                       /* read buf                  */
    INT8U readMsgBufID(INT32U *ID, INT8U *len, INT8U *buf);         /* read buf with object ID   */
    INT8U checkReceive(void);                                       /* if something received     */
    INT8U checkError(void);                                         /* if something error        */
    INT32U getCanId(void);                                          /* get can id when receive   */
    INT8U isRemoteRequest(void);                                    /* get RR flag when receive  */
    INT8U writeStId(INT8U id);
    INT8U setRcvInt(bool res);
    INT8U sendRmtMsg(INT8U id);
    INT8U loadMsg(INT32U id, INT8U ext, INT8U rtr, INT8U len, INT8U *buf);
    INT8U lowRcvFlag(void);
    INT8U DisGloInt(void);
    INT8U checkRmtMsg(void);
    INT8U checkTrFlag(void);
    INT8U lowTrFlag(void);
    INT8U setTrInt(bool res);
    INT8U OSMenable(void);
    INT8U setRcvInt0(bool res);
    INT8U setRcvInt1(bool res);
    INT8U MendFiltering(void);
    };

#endif
```

Figura 6.10. Funciones públicas de la librería

A continuación se modificará el archivo mcp_can.cpp, escribiendo la función siguiendo el patrón de las funciones ya existentes, indicando el nombre y la descripción.

```
INT8U MCP_CAN::isRemoteRequest(void)
{
    return m_nRtr;
}

/*********************************************************************************************************
** Function name:           writeStId
** Descriptions:            Establish Standard Identifier
*********************************************************************************************************/
INT8U MCP_CAN::writeStId(INT8U id)
{
        uint16_t canid;
        INT8U tbufdata[4];

        canid = (uint16_t)(id & 0x0FFFF);

            tbufdata[MCP_SIDH] = (INT8U) (canid >> 3 );
            tbufdata[MCP_SIDL] = (INT8U) ((canid & 0x07 ) << 5);
            tbufdata[MCP_EID0] = 0;
            tbufdata[MCP_EID8] = 0;

        mcp2515_setRegisterS(0x31, tbufdata, 4 );
}

/*********************************************************************************************************
 ** Function name:           setRcvInt
** Descriptions:            Enable Interrupts when message arrives if true
*********************************************************************************************************/

INT8U MCP_CAN::setRcvInt(bool res)
{
    if(res){
        mcp2515_setRegister(0x0C,0xFF );
        mcp2515_modifyRegister(0x2B,0x03,0x03 );
        }
    else {
            mcp2515_setRegister(0x0C,0x00 );
            mcp2515_modifyRegister(0x2B,0x03,0x00 );

        }

}

/*********************************************************************************************************
 ** Function name:           sendRmtMsg
** Descriptions:            send a Remote Frame to a certain node
*********************************************************************************************************/
```

Figura 6.11. Código de la función setRcvInt() dentro del archivo .cpp

Una vez hayamos modificado la librería, deberemos pegar los archivos en el directorio de librerías del Arduino IDE, que se encuentra en la siguiente ruta: C:\Program Files\Arduino\libraries\CAN_BUS_Shield-master. De esta forma ya podremos utilizar las funciones creadas. A la hora de usar una función, se escribe delante suya "CAN.", siendo CAN una instancia de la Clase MCP_CAN.

El programa realizado en Arduino IDE es el siguiente:

```
#include <SPI.h>                                    //Se incluye el header de la libreria SPI
                                                    //(Serial Peripheral Interface)
#include "mcp_can.h"                                //Se incluye el header de la libreria del
                                                    //Shield CAN-BUS

MCP_CAN CAN(10);                                    //Se crea una instancia de la clase MCP_CAN


unsigned char len = 0;                              //Variable que indica la longitud de los datos
unsigned char buf[8];                               //Vector donde se guardan los datos
unsigned char stmp[8] = {0, 1, 1, 1, 1, 1, 1, 1};   //Se crea un vector de caracteres de longitud 8
boolean enable = true;

void setup()
{
    Serial.begin(115200);                           //Se inicializa el puerto serial a 115200 baudios


START_INIT:

    if(CAN_OK == CAN.begin(CAN_500KBPS))            // Se establecen los baudios = 500k
    {
      Serial.println("CAN BUS Shield init ok!");  //Si el microcontrolador se inicia correctamente,
```

```
                                                //escribe por puerto Serial "CAN BUS Shield init ok!"
    }
    else                                        //Si el microcontrolador no se inicia correctamente:
    {
        Serial.println("CAN BUS Shield init fail");  //Escribe por el puerto Serial:
        Serial.println("Init CAN BUS Shield again"); //"CAN BUS Shield init fail" y en la linea
                                                     //siguiente "Init CAN BUS Shield again"
        delay(100);                                  //Retardo de 100 milisegundos
        goto START_INIT;                             //Vuelve a la rutina de inicialización
    }

    CAN.DisGloInt();                                //Se deshabilitan las variables globales
    CAN.setRcvInt(true);                            //Habilitada la interrupcion por recepción de
mensaje
    attachInterrupt(0, MCP2515_ISR, FALLING);       //Se establece la interrupción del MCP2515
}

void MCP2515_ISR()                                  //Interrupción del MCP2515
{     delay(10);                                    //Retardo de 10 milisegundos
     CAN.readMsgBuf(&len, buf);                     // Lee los datos y la longutud de estos y los guarda
                                                        // en el vector buf y la variable len
respectivamente
     //Impresión de los datos

     for(int i = 0; i<len; i++) {
        Serial.print(buf[i]);Serial.print("\t");
      }
      Serial.println();
      delay(10);                                    //Retardo de 10 milisegundos

     //ENVIO DE LOS DATOS

    // Parámetros mensaje:  id = 0x10, Trama estándar, longitud datos = 8, stmp: datos
    CAN.sendMsgBuf(0x10, 0, 8, stmp);          //Manda un mensaje indicando:
                                               //(identificador,tipo de trama,bytes de datos, datos)
    enable = false;                            //Como se ha entrado a interrupción, enable=false
}

void loop()                                         //Bucle infinito
{

  if(enable){                                      //Si el nodo 2 no ha contestado con un mensaje:
    CAN.sendMsgBuf(0x10, 0, 8, stmp);              //El nodo 1 manda un mensaje

    }
  delay(100);                                      //Retardo de 100 milisegundos

    }

/*********************************************************************************************************
  END FILE
*********************************************************************************************************/
```

El programa anterior corresponde al nodo número 1, dado que el identificador asignado es el 0x10 y el mensaje enviado es {0 1 1 1 1 1 1 1}. Se puede observar como el programa se ciñe al diagrama de flujo. Cuando un nodo recibe un mensaje, éste lo lee y manda el suyo propio a continuación. El Nodo1 mandará mensajes continuamente en el Bucle infinito hasta que el Nodo2 conteste con otro mensaje. El programa correspondiente al nodo 2 será igual salvo por unas salvedades, el identificador del mensaje será 0x20 y el mensaje que se envíe será {0 2 2 2 2 2 2 2}. Es importante indicar que las funciones se deberán introducir después de la rutina de inicialización, en caso contrario el programa no funcionará.

El resultado final es satisfactorio, los dos nodos reciben los mensajes del otro con éxito. Esto se puede observar mediante los monitores puerto serie que contiene Arduino. En

concreto monitorizaremos el puerto COM8, que es el correspondiente con el nodo 1 y el COM7 que es el correspondiente con el nodo 2:

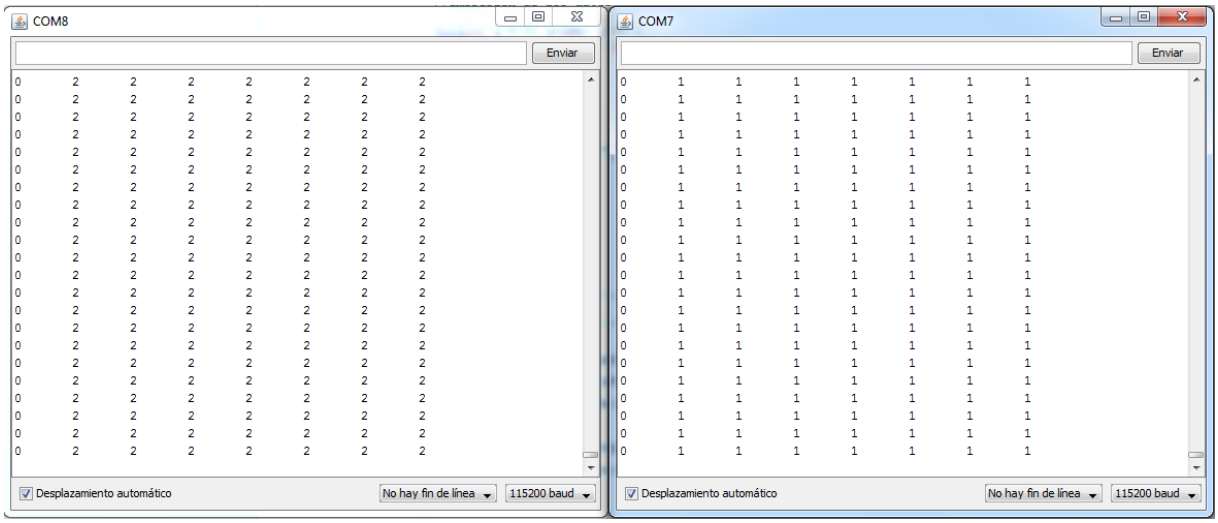

Figura 6.12. Monitor Serial del nodo 1 (izda.) y del nodo 2 (dcha.)

Una vez comprobamos que se puede realizar una comunicación entre los dos nodos, el siguiente paso es comunicar tres nodos:

Al igual que antes, la comunicación será secuencial, en primer lugar mandará mensaje el nodo 1, a continuación el nodo 3 y por último el nodo 2. El diagrama de flujo es el siguiente.

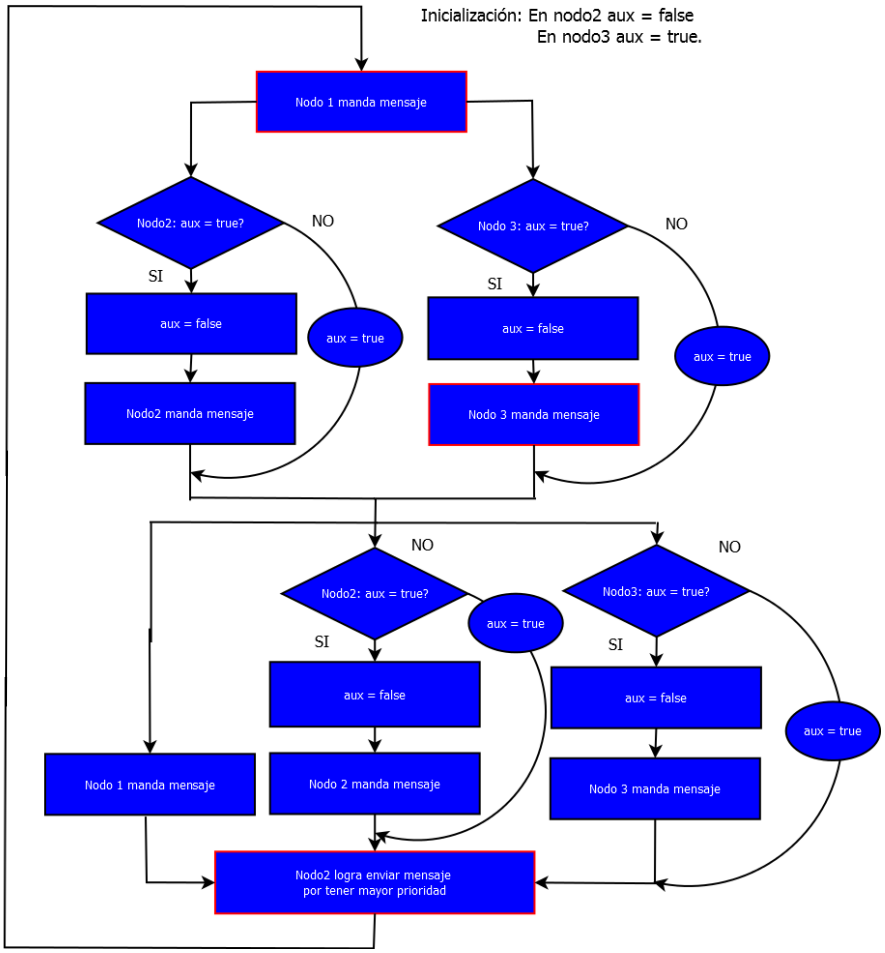

Figura 6.13. Diagrama de flujo 2

El programa en del nodo 1 se mantendrá igual que el creado anteriormente. Sin embargo, al añadir un nodo más a la red, es necesario modificar el programa del nodo 2 para que no mande un mensaje al mismo tiempo que el nodo 3:

```
include <SPI.h>                    //Se incluye el header de la libreria SPI (Serial Peripheral Interface)
#include "mcp_can.h"                       //Se incluye el header de la libreria del Shield CAN-BUS

MCP_CAN CAN(10);                                        //Se crea una instancia de la clase MCP_CAN


unsigned char len = 0;                                  //Variable que indica la longitud de los datos
unsigned char buf[8];                                   //Vector donde se guardan los datos
unsigned char stmp[8] = {0, 2, 2, 2, 2, 2, 2, 2};       //Se crea un vector de caracteres de longitud 8
boolean aux = false;                                    //Inicializamos la variable aux a FALSE
int cont = 0;

void setup()
{
    Serial.begin(115200);                               //Se inicializa el puerto serial a 115200 baudios


START_INIT:

    if(CAN_OK == CAN.begin(CAN_500KBPS))                //// init can bus : baudrate = 500k
    {
        Serial.println("CAN BUS Shield init ok!");      //Si el microcontrolador se inicia correctamente,
                                                        //escribe por el puerto Serial
                                                        //"CAN BUS Shield init ok!"
    }
    else                                            //Si el microcontrolador no se inicia correctamente:
    {
        Serial.println("CAN BUS Shield init fail");     //Escribe por el puerto Serial
```

```
        Serial.println("Init CAN BUS Shield again"); //"CAN BUS Shield init fail" y en la linea siguiente
                                                     //"Init CAN BUS Shield again"
        delay(100);                                  //Retardo de 100 milisegundos
        goto START_INIT;                             //Vuelve a la rutina de inicialización
    }
    CAN.DisGloInt();                                  //Se deshabilitan las variables globales
    CAN.setRcvInt(true);                             //Se habilitan las interrupciones por recepción
                                                     //de mensaje

    attachInterrupt(0, MCP2515_ISR, FALLING);        //Se establece la interrupción del MCP2515
}

void MCP2515_ISR()                                   //Interrupción del MCP2515
{
     CAN.readMsgBuf(&len, buf);                      //Lee los datos y la longutud de estos y los
                                                     //guarda en el vector buf y la
                                                     // variable len respectivamente
     if(cont !=3 ){                          //Si ha mandado el nodo 2 un mensaje no se actualiza aux
      if (aux){                                       //Si aux = true

       //Impresión de los datos

       for(int i = 0; i<len; i++) {
          Serial.print(buf[i]);Serial.print("\t");
        }
        Serial.println();
        delay(10);                                    //Retardo de 10 milisegundos

       //ENVIO DE LOS DATOS

      // Parámetros mensaje:  id = 0x10, Trama estándar, longitud datos = 8, stmp: datos
      CAN.sendMsgBuf(0x09, 0, 8, stmp);                //Manda un mensaje indicando:
                                                //(identificador,tipo de trama,bytes de datos,datos)
      aux = false;                               //Con aux = false en la siguiente iteración el nodo 2
                                                 //no mandará mensaje.
      }else aux = true;                          //Si no ha mandado mensaje, en la siguiente interrupción
                                                 //lo mandará(aux = true)
    } else cont = 0;

}

void loop()                                  //Bucle infinito
{

    }

/*********************************************************************************************************
  END FILE
*********************************************************************************************************/
```

Para el nodo 3, el programa difiere en el identificador, la inicialización de la variable 'aux' y el mensaje enviado. Para que los nodos sigan la secuencia acordada, el nodo 2 debe tener un identificador menor que el nodo 1, para que cuando ambos nodos manden sus mensajes simultáneamente, sea el nodo 2 el que realmente haga llegar su mensaje. La variable 'aux' se inicializará en true para que al comenzar la secuencia, tras mandar el nodo 1 su mensaje, sea el nodo 3 el que mande otro mensaje a continuación, y no el nodo 2. Finalmente el mensaje de 1 byte que enviará el nodo3 será: {0, 3, 3, 3, 3, 3, 3, 3}. Cuando el nodo 2 envíe su mensaje, el ciclo comenzará de nuevo.

El resultado al observar el puerto serial y los LEDs indicadores de los nodos no ha sido el esperado. Durante el programa el nodo 1 no baja el flag y por lo tanto no puede entrar a la interrupción una segunda vez como se observa en la siguiente imagen:

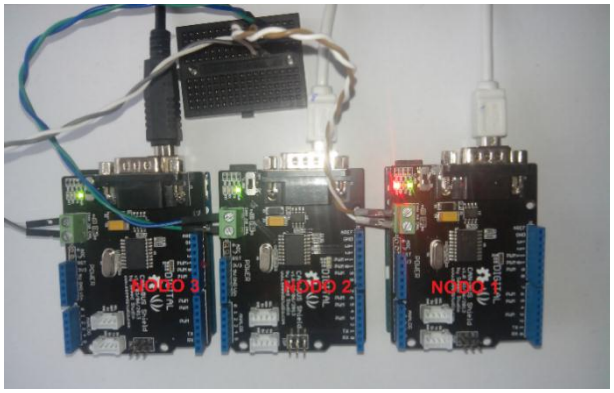

Figura 6.14. Error en el nodo 1

El LED INT 1 se mantiene encendido lo cual indica un error en la bajada del flag. Si observamos el monitor serial nos encontramos con que solamente el Nodo 3 consigue mandar su mensaje:

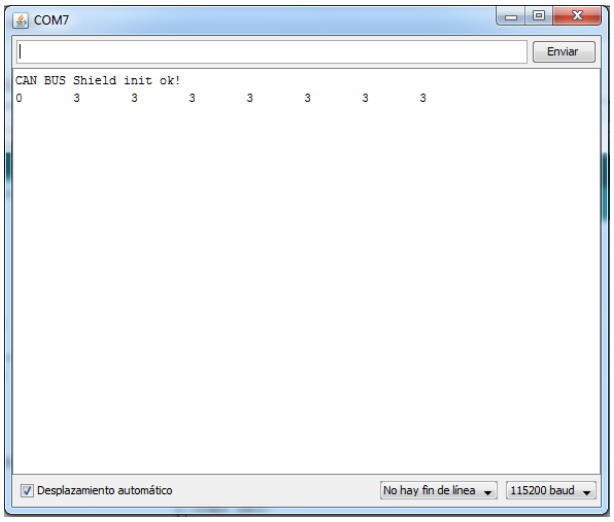

Figura 6.15. Monitor serial del nodo 1

El problema puede ser debido al uso de funciones dentro de la interrupción o a el uso de la impresión por el puerto serie. Ambas provocan que haya una gran carga de programa dentro de la interrupción, provocando un funcionamiento incorrecto de ésta. Hay errores documentados por el uso de la función Serial.print() dentro de la rutina de interrupción **[20] [21]**.

Visto que el programa no satisface completamente las expectativas previstas, se propone otro modo de funcionamiento entre los nodos de la red CAN. Para el siguiente planteamiento, es necesario conocer el protocolo TTCAN, una variante del protocolo CAN bus que indica cómo realizar una comunicación entre varios nodos basada en el TDMA (Time Division Multiple Access):

6.2 TTCAN (Time-triggered CAN)

El protocolo TTCAN se encuentra por encima de la capa de enlace de datos según la pila OSI. Es compatible con el ya mencionado CSMA/CD + AMP y con los dos tipos de capas físicas (de alta y baja velocidad) y está estandarizado en la ISO 11898-4. TTCAN fue ideado para el envío periódico de datos, por lo tanto parece adecuado para el trabajo.

6.2.1 Funcionamiento básico

El protocolo TTCAN, a diferencia del protocolo CAN, contiene un mensaje de referencia. Éste indica el comienzo de las comunicaciones por parte de los nodos que conforman la red y es reconocido mediante su identificador. El tiempo entre dos mensajes de referencia se denomina ciclo básico. El tiempo se divide en diferentes ranuras en las cuales uno o varios nodos mandan sus mensajes. Además existen distintos tipos de ranuras de tiempo:

- Ranuras exclusivas
- Ranuras libres
- Ranuras de arbitrio

Las ranuras exclusivas son aquellas en las que un único nodo manda su trama correspondiente. Las ranuras libres son aquellas en las que no se realiza ninguna actividad por parte de los nodos. Por último, las ranuras de arbitrio funcionan como el protocolo CAN bus habitual, en ellas se produce un mensaje simultáneo de dos o más nodos, y sólo logra mandar el mensaje aquel nodo con mayor prioridad. En el protocolo CAN bus estándar, un nodo retransmite su mensaje al perder una contienda. Esto no ocurre en TTCAN: Si suponemos un ejemplo en el que hay una colisión entre dos nodos con identificador 0x00 y 0x01, cuando el nodo 0x00 gane la contienda, el nodo con identificador 0x01 no retransmitirá su mensaje a continuación.

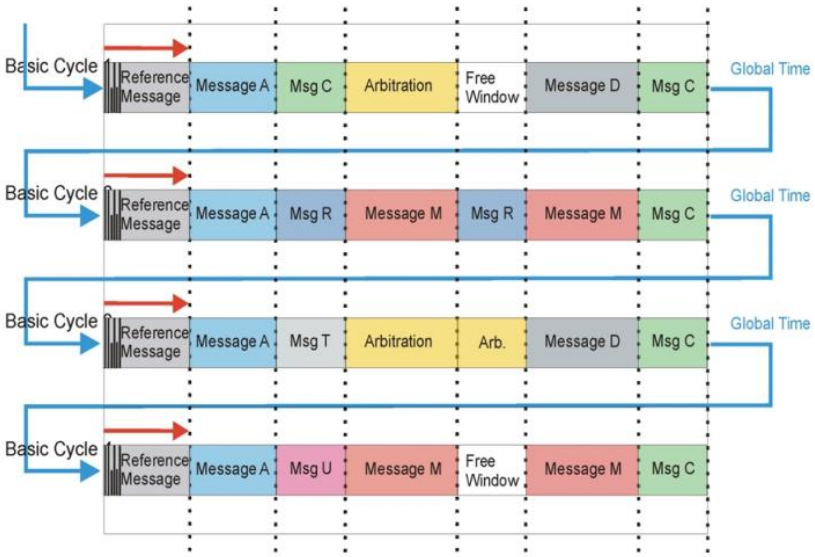

Figura 6.16. Matriz de tiempos del protocolo TTCAN [22]

Existen dos niveles de TTCAN en base a las características que implementan. El nivel 1 de TTCAN se puede implementar vía software siempre que se tenga un controlador de CAN adecuado. El nivel 2 requiere de un hardware especializado y orientado al protocolo TTCAN. La mayor diferencia entre los dos niveles recae en la complejidad en la que ambos llevan a cabo la sincronización. Se utilizan tres unidades de tiempo:

Tiempo local

El tiempo local es un contador perteneciente a cada uno de los nodos el cual aumenta en base a la frecuencia del reloj utilizada y a un parámetro denominado TUR (Time unit ratio). El tiempo local aumenta cada unidad de tiempo de red (NTU). La frecuencia de la NTU se define de la siguiente forma:

$$f(NTU) = \frac{f(Reloj)}{TUR}$$

La diferencia que reside entre los dos niveles de TTCAN respecto al tiempo local es la resolución de este, teniendo el nivel 2 una mayor resolución.

Tiempo de ciclo

El tiempo de ciclo se define como la diferencia entre dos mensajes de referencia. Para poder medir adecuadamente el tiempo de ciclo, debemos ser capaces de monitorizar la señal de start of frame (SOF) del mensaje de referencia. Esto solo puede realizarse mediante hardware.

Tiempo global

El tiempo global solo es utilizado por el nivel 2 de TTCAN, por lo tanto no se utilizará en el trabajo, ya que el controlador utilizado no ha sido diseñado para tal propósito. El tiempo global se calcula de la siguiente forma:

En primer lugar se realiza la diferencia entre el tiempo del mensaje de referencia visto por un nodo cualquiera y por el nodo maestro (aquel que manda el mensaje de referencia). A esta diferencia se la denomina Offset Local. Finalmente al tiempo local se le resta dicho Offset Local obteniendo el tiempo global.

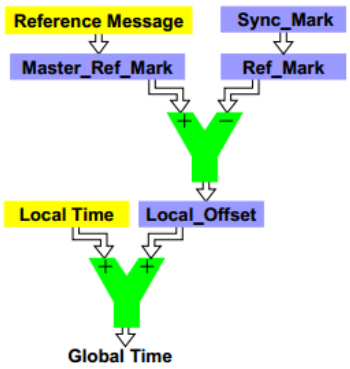

Figura 6.17. Definición del tiempo global [23]

6.2.2 Ventajas de TTCAN

El hecho de que la comunicación en TTCAN se divida en slots de tiempo permite que se realice un control en lazo cerrado en las redes basadas en este protocolo. Además el intercambio de datos en tiempo real mejora significativamente con el uso de TTCAN.

6.2.3 Implementación de TTCAN

Después de ser estandarizado en la ISO, muchos fabricantes se interesaron en el desarrollo de dispositivos que utilizaran dicho protocolo. Bosch fue el primer fabricante que logró implementarlo en una FPGA y debido a la reacción del mercado, decidió implementarlo en un chip que permitía utilizar tanto el nivel 1 como el nivel 2 de TTCAN.

Para implementar el nivel 1 de TTCAN se necesita al menos un controlador que permita utilizar el modo One-shot, el cual solo inhibe la retransmisión de mensaje por parte de los nodos.

6.2.4 Conclusiones

TTCAN parece ideal para el propósito del trabajo, ya que al dividir el tiempo en ranuras se evitan colisiones y se permite que todos los nodos manden mensajes secuencialmente. Además el controlador que utiliza el shield CAN-BUS, el MCP2515, permite el uso del modo One-shot, indispensable para llevar a cabo el nivel 1 de TTCAN. Además, el controlador también permite monitorizar el SOF de cada trama, herramienta muy útil para poder medir la duración de las tramas y así construir una matriz de tiempos correctamente. A la hora de crear la matriz de tiempos se utiliza software especializado. No obstante, en nuestro caso, la crearemos teniendo en cuenta la duración de una trama CAN bus estándar con ayuda de un osciloscopio.

TTCAN fue contemplado como una revolución en cuanto a la comunicación de dispositivos del automóvil. Sin embargo, hoy en día, pocas son las fuentes que lo mencionan, ya que los artículos escritos sobre este tema datan de hace una década aproximadamente **[24] [25]**.

6.3 Planteamiento Nº 2

Una vez visto que TTCAN se ajusta a nuestras necesidades, se intentará crear un programa que siga las instrucciones de dicho protocolo:

Como se ha mencionado antes, TTCAN se caracteriza por utilizar un mensaje de referencia para indicar el inicio de un nuevo ciclo. Así pues, si antes utilizábamos tres nodos, ahora se utilizarán cuatro. Tres de ellos mandarán datos concernientes a la monitorización y el nodo restante se encargará de enviar el mensaje de referencia.

Para poder crear la matriz de tiempos, se contará con un osciloscopio el cual mida la longitud estándar de una trama de CAN. Para ello se establecerá un tipo de trama determinada: las tramas enviadas tendrán datos con longitud de 1 byte y serán de tipo estándar.

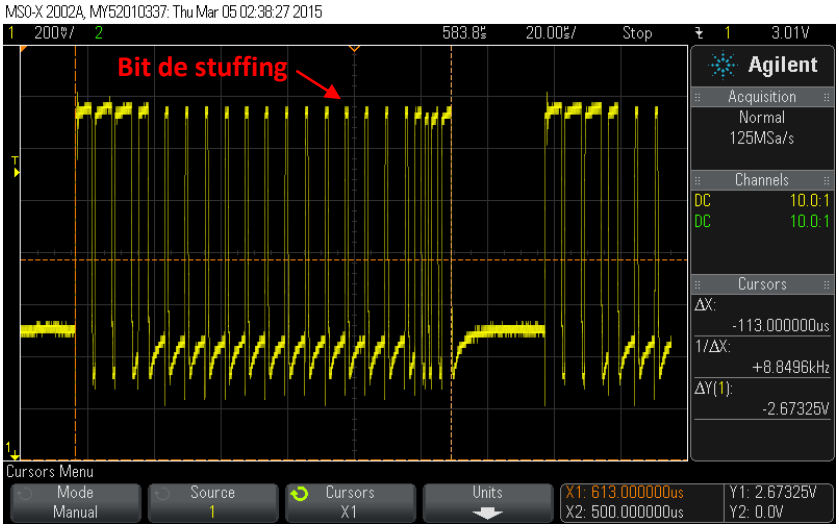

Figura 6.18. Longitud trama con ID y datos a 0 a 1000Kbps

Para determinar el tamaño de la trama, monitorizamos el peor caso posible, con tantos bits de stuffing como sea posible, lo cual conseguiremos con un identificador con valor 0xFF y datos con valor 0xFF. En la imagen superior se puede observar una sucesión de picos que corresponden con los bits de stuffing. Por último, se utilizará la velocidad máxima que nos ofrece el controlador de CAN bus, 1000Kbps.

Se determina que la longitud de la trama es de 113 µs. Se dejará un margen de seguridad de 17µs quedando un total de 130µs entre envíos de mensajes.

Por lo tanto, teniendo en cuenta dicha longitud y el número de nodos, se establece las siguientes ranuras de tiempo:

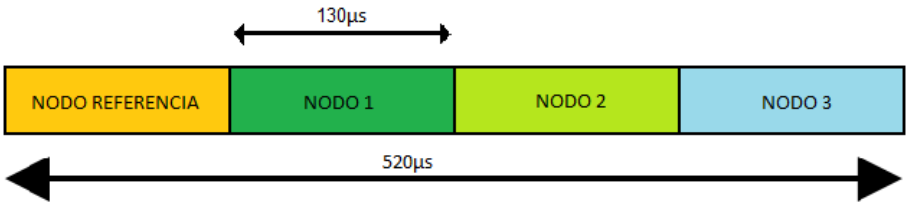

Figura 6.19. Ranuras de tiempo 1

A la hora de realizar el programa, se necesitara una herramienta para medir el tiempo, para ello se utilizará el Timer2 del Arduino UNO.

Tiene hasta 3 modos de funcionamiento:

- Modo normal (Normal mode)
- Modo comparación (Clear Timer on Compare Match mode)

- Modo PWM rápido (Fast PWM Mode)

El primer modo incrementa el valor contador del Timer 2 con cada ciclo de trabajo. Si está habilitada, se generará una interrupción al superar el contador su valor máximo. La frecuencia del reloj puede ser modificada via software.

El modo CTC (Modo comparación) incrementa el valor del contador del Timer 2 hasta un valor que se ha determinado mediante software. Cuando el contador alcanza dicho valor, este se reinicia a cero automáticamente y continúa incrementándose. Si se ha habilitado la interrupción correspondiente, está se activará cuando el valor del contador alcance al determinado por software.

Finalmente el modo PWM permite obtener una señal modulada por ancho de pulso pudiendo variar la frecuencia de esta modificando adecuadamente determinados registros del microcontrolador.

Para el presente trabajo utilizaremos el modo CTC, ya que resulta útil el hecho de que el contador se reinicie automáticamente cuando alcanza un valor establecido.

La configuración de este modo es la siguiente:

Se establece el modo del Timer 2 modificando los bits WGM en los registros TCCR2B y TCCR2A.

TCCR2A – Timer/Counter Control Register A

Bit	7	6	5	4	3	2	1	0	
(0xB0)	COM2A1	COM2A0	COM2B1	COM2B0	–	–	WGM21	WGM20	TCCR2A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 6.20. Registro TCCR2A [5]

TCCR2B – Timer/Counter Control Register B

Bit	7	6	5	4	3	2	1	0	
(0xB1)	FOC2A	FOC2B	–	–	WGM22	CS22	CS21	CS20	TCCR2B
Read/Write	W	W	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 6.21. Registro TCCR2B [5]

El bit WGM21 y WGM20 se establecen a ‘1’ y ‘0’ respectivamente en el registro TCCR2A y el bit WGM22 a ‘0’.

El pin OC2A cambia de valor lógico cada vez que que el contador alcanza el valor que hemos establecido y la frecuencia viene dada por la siguiente fórmula:

$$f_{OCnx} = \frac{f_{\text{clk\_I/O}}}{2 \cdot N \cdot (1 + OCRnx)}$$

Nos interesa conocer en cuanto tiempo se alcanza el valor establecido una sola vez. Para ello multiplicamos la frecuencia anterior por 2 y a continuación realizamos la inversa de la fórmula obtenida. El resultado obtenido es el siguiente:

$$t = \frac{N * (1 + OCR2A)}{f_{clk}}$$

Siendo N el valor del preescalador y f_{clk} el valor de la frecuencia del reloj. El parámetro N se puede cambiar mediante software modificando los valores de los bits CS22, CS21 y CS20 del registro TCCR2B. La siguiente tabla muestra la relación entre los valores de los bits y el preescalador:

CS22	CS21	CS20	N
0	0	0	Timer off
0	0	1	0
0	1	0	8
0	1	1	32
1	0	0	64
1	0	1	128
1	1	0	256
1	1	1	1024

El valor al que tiene que llegar el Timer se establece en el registro OCR2A y/o en el registro OCR2B:

OCR2A – Output Compare Register A

Bit	7	6	5	4	3	2	1	0
(0xB3)	OCR2A[7:0]							
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Figura 6.22. Ranuras de tiempo 1 [5]

OCR2B – Output Compare Register B

Bit	7	6	5	4	3	2	1	0
(0xB4)	OCR2B[7:0]							
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Figura 6.23. Ranuras de tiempo 1 [5]

Para habilitar la interrupción por comparación se modifica el registro TIMSK2:

TIMSK2 – Timer/Counter2 Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0
(0x70)	–	–	–	–	–	OCIE2B	OCIE2A	TOIE2
Read/Write	R	R	R	R	R	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Figura 6.24. Ranuras de tiempo 1 [5]

Los bits OCIE2B y OCIE2A habilitan las interrupciones por comparación del Timer 2 con los registros OCR2A y OCR2B respectivamente. El bit menos significativo del registro indica cuando hay un desbordamiento del Timer. En el desarrollo del programa utilizaremos la comparación con el valor del registro OCR2A, luego solo pondremos a 1 el bit OCIE2A.

Por último, se tendrá en cuenta el registro TCNT2 para reiniciar el valor del contador del Timer 2 en caso de que sea necesario.

Los nodos pertenecientes a la red deben ser capaces de diferenciar entre un mensaje normal y uno de referencia. Para ello existen dos posibilidades:

La primera opción es utilizar la funcion getCanId() que se encuentra por defecto en la librería del Shield CAN-BUS. Esta función retorna automáticamente el identificador del mensaje recibido y permite guardarlo en una variable. Sin embargo, dicha función permite obtener el identificador una vez leído el mensaje almacenado en el buffer, lo cual implica que los nodos detecten que el mensaje era uno de referencia demasiado tarde.

Para solventar dicho problema, existe una segunda opción en la que actúa el propio hardware del controlador y por lo tanto es mucho más rápida. Consiste en la utilización de los filtros e interrupciones de recepción de los nodos.

El controlador de CAN contiene dos búferes de recepción con sus propias máscaras y filtros. Existe la posibilidad de que se active una interrupción cuando haya llegado un mensaje correctamente a uno de los dos búferes, lo cual resultará muy útil en el trabajo.

Para que los nodos reconozcan un mensaje como mensaje de referencia, estableceremos los filtros y las máscaras de manera que el identificador de dicho mensaje solo pueda ser aceptado por uno de los búferes. Cuando los nodos lo hayan recibido correctamente, saltará una interrupción. Respecto a los mensajes estándar, estos serán almacenados en el búfer restante normalmente, ya que deshabilitaremos la interrupción por recepción de este. De esta manera hemos obtenido una manera de reconocer mediante interrupción cuando un mensaje de referencia ha sido enviado.

Primeramente se crea un programa para el nodo que envía los mensajes de referencia. Como se ha mencionado anteriormente, para poder configurar el Timer 2 es necesario modificar registros dentro del Arduino IDE.

Para que el estado lógico de un bit sea uno se utiliza la función asignación con OR binario. El registro realiza la operación OR con el valor que esta a la derecha de la igualdad y se le asigna el resultado. En el siguiente ejemplo se pondrían los 3 bits menos significativos a 1.

REGISTRO |= B00000111;

En el caso querer que el bit pase a ser un cero, se utiliza la función asignación con AND binario, la cual funciona de la misma forma que la anterior. En el siguiente ejemplo se pone el bit número 3 a cero.

REGISTRO &= B11110111;

A continuación se muestra el programa desarrollado para el nodo director, aquel que manda el mensaje de referencia:

```
#include <SPI.h>            //Se incluye el header de la libreria SPI (Serial Peripheral Interface)
#include "mcp_can.h"       //Se incluye el header de la libreria del Shield CAN-BUS

MCP_CAN CAN(10);                                    //Se crea una instancia de la clase MCP_CAN


int counter = 0;                                          // Contador para el Timer 2
unsigned char stmp[8] = {1, 1, 1, 1, 1, 1, 1, 1};       //Se crea un vector de caracteres de longitud
8
unsigned char len = 0;                                    //Variable que indica la longitud de los
datos
```

```
unsigned char buf[8];                                    //Vector donde se guardan los datos


void setup(){

Serial.begin(115200);                                  //Se inicializa el puerto serial a 115200 baudios

//INICIALIZAMOS EL MCP2515

START_INIT:

    if(CAN_OK == CAN.begin(CAN_500KBPS))         // inicializar can bus : baudrate = 1000k
    {
      Serial.println("CAN BUS Shield init ok!");//Si el microcontrolador se inicia correctamente,
                                                 //escribe por el puerto Serial:
                                                 //"CAN BUS Shield init ok!"
    }
    else                                         //Si el microcontrolador no se inicia correctamente:
    {
      Serial.println("CAN BUS Shield init fail"); //Escribe por el puerto Serial
      Serial.println("Init CAN BUS Shield again");//"CAN BUS Shield init fail" y:
                                                   // "Init CAN BUS Shield again"
        delay(100);                                //Retardo de 100 milisegundos
        goto START_INIT;                           //Vuelve a la rutina de inicialización
    }


//PROGRAMAMOS EL TIMER 2

TCCR2B |=  B00000111;                            //Configuramos el preescaler (1024)

TCCR2B &=  B11110111;                            //Configuramos los bits WGM para establecer
TCCR2A &=  B11111110; TCCR2A |=B00000010;        //el modo comparación

TIMSK2 |= B00000010 ;                            //Habilitamos interrupción del comparador A

OCR2A = B11111111;                               //Establecemos valor del contador A (256)

TCNT2 = B00000000;                               //Iniciamos el valor del TMR a 0

//Deshabilitamos interrupciones globales del MCP2515

CAN.DisGloInt();

}


ISR(TIMER2_COMPA_vect){                 //Interrupción por comparación del Timer 2
  counter++;                            //Se aumenta el valor del contador
  if(counter == 300){                   //Pasados aproximadamente 5s

    CAN.sendMsgBuf(0x00, 0, 0, stmp); //Se manda un mensaje de calibración con
                                        //id cero, trama estándar, longitud cero
                                        //y datos cualuiera.
    counter = 0;                        //Se reinicia el contador

  }

}


void loop(){                           //Bucle infinito

}
```

El programa es sencillo, simplemente mandamos un mensaje cada vez que el Timer 2 llegue 300 veces al valor establecido en el registro OCR2A. Según la formula del Timer 2 y los parámetros elegidos, saltará una interrupción por comparación cada 4,9344 segundos. Se comprueba que funciona correctamente chequeando el led TX del shield CAN-BUS, que

se enciende cada 5 segundos aproximadamente. A el nodo director se le asignará el identificador de mayor prioridad, el 0x00.

A continuación se mostrará el programa de un segundo nodo que sea capaz de reconocer los mensajes de referencia y enviar una trama de datos en el momento que se haya configurado:

```
#include <SPI.h>       //Se incluye el header de la libreria SPI (Serial Peripheral Interface)
#include "mcp_can.h"   //Se incluye el header de la libreria del Shield CAN-BUS

MCP_CAN CAN(10);       //Se crea una instancia de la clase MCP_CAN


//Interrupción

int counter = 0;         // Contador para el Timer 2
boolean synch = false;   //Variable que controla la lectura de mensajes
boolean enable = false; //Variable que controla el aumento del contador del Timer2
unsigned char stmp[8] = {1, 1, 1, 1, 1, 1, 1, 1}; //Se crea un vector de caracteres de longitud 8
unsigned char len = 0;                             //Variable que indica la longitud de los datos
unsigned char buf[8];                              //Vector donde se guardan los datos


void setup(){

Serial.begin(115200);                            //Se inicializa el puerto serial a 115200 baudios

//INICIALIZAMOS EL MCP2515

START_INIT:

    if(CAN_OK == CAN.begin(CAN_500KBPS))             // inicializar can bus : baudrate = 500k
    {
        Serial.println("CAN BUS Shield init ok!"); //Si el microcontrolador se inicia
                                                   //correctamente, escribe por el puerto Serial:
                                                   //"CAN BUS Shield init ok!"
    }
    else                                         //Si el microcontrolador no se inicia correctamente:
    {
        Serial.println("CAN BUS Shield init fail");     //Escribe por el puerto Serial
        Serial.println("Init CAN BUS Shield again");    //"CAN BUS Shield init fail" y
                                                        //en la linea siguiente:
                                                        // "Init CAN BUS Shield again"
        delay(100);                                     //Retardo de 100 milisegundos
        goto START_INIT;                                //Vuelve a la rutina de inicialización
    }

//CONFIGURAMOS LAS MÁSCARAS PARA QUE EN EL BUFFER 0 SOLO SE RECIBAN MENSAJES CON IDENTIFICADOR 0X00

      CAN.init_Mask(0, 0, 0xFF);     //Establecemos la máscara del buffer 0
                                     //(número de máscara, tipo de trama e identificador)
      CAN.init_Mask(1, 0, 0x00);     //Establecemos la máscara del buffer 1

      //Establecemos los dos filtros del buffer 0

      CAN.init_Filt(0, 0, 0x00);
      CAN.init_Filt(1, 0, 0x00);

      //Establecemos los dos filtros del buffer 1

      CAN.init_Filt(2, 0, 0x00);
      CAN.init_Filt(3, 0, 0x00);
      CAN.init_Filt(4, 0, 0x00);
      CAN.init_Filt(5, 0, 0x00);

//PROGRAMAMOS EL TIMER 2

TCCR2B |=  B00000111;                          //Configuramos el preescaler (1024)

TCCR2B &=  B11110111;                          //Configuramos los bits WGM para
                                               //establecer el modo comparación
TCCR2A &=  B11111110; TCCR2A |=B00000010;
```

```
TIMSK2 &= B11111101 ;                           //Deshabilitamos interrupción del comparador A

OCR2A = B11111111;                              //Establecemos valor del contador A (256)

TCNT2 = B00000000;                              //Iniciamos el valor del TMR a 0

//AÑADIMOS INTERRUPCION POR MENSAJE DE REFERENCIA

attachInterrupt(0, MCP2515_ISR, FALLING); // Se añade la interrupción del Shield

//CONFIGURAMOS INTERRUPCIONES

CAN.DisGloInt();        //Deshabilitamos interrupciones globales del MCP2515
CAN.setRcvInt0(true);   //Se habilita la interrupción por recepción en el buffer 0
CAN.setRcvInt1(false); //Se deshabilita la interrupción por recepción en el buffer 1


}


void MCP2515_ISR()      //Interrupción general del Shield
{
CAN.lowRcvFlag();       //Se baja el flag de recepción de mensaje
enable = true;          //Se habilita el aumentado del contador
synch = true;           //Se habilita la lectura de mensajes
TIMSK2 |= B00000010 ;   //Habilitamos interrupción del comparador A
TCNT2 = B00000000;      //Iniciamos el valor del TMR a 0

}


ISR(TIMER2_COMPA_vect){ //Interrupción por comparación del Timer 2


  if(enable){     //Si se ha recibido un mensaje de referencia
  counter++;      //Se aumenta el contador
  if(counter == 100){ //Cuando hallan pasado 1,6s aproximadamente

    CAN.sendMsgBuf(0x04, 0, 8, stmp);   //Se manda un mensaje con identificador 0x04, estandar,
                                        //longitud 8 y datos {1 1 1 1 1 1 1 1}
    TIMSK2 &= B11111101 ;               //Deshabilitamos interrupción del comparador A
    counter = 0;                        //Se reinicia el contador
    enable = false;                     //Se deshabilita el aumento del contador

      }
  }
}


void loop(){   //Bucle infinito

  if(synch){     //Si se ha recibido un mensaje de referencia


      if (CAN_MSGAVAIL == CAN.checkReceive()) { //Si se ha recibido un mensaje valido

        CAN.readMsgBuf(&len, buf);// Lee los datos del mensaje y su longitud

        //Se imprimen los datos
        for(int i = 0; i<len; i++) {
          Serial.print(buf[i]);Serial.print("\t");
        }
        Serial.println();
      }
  }
}
```

Para poder comprender el programa, se describirán distintas partes individualmente:

Mediante las funciones de la librería init_Mask() y init_Filt() se configuran lás máscaras y los filtros de manera que el búfer 0 solo acepte aquellos mensajes con identificador 0x00, es decir, los de referencia:

	Búfer 0
Identificador	00000000000
Máscara	11111111111
AND	00000000000
Filtros	00000000000
¿Coinciden?	SI

	Búfer 0
Identificador	00000000001
Máscara	11111111111
AND	00000000001
Filtros	00000000000
¿Coinciden?	NO

La máscara y los filtros del búfer 1 se configuran de manera que se acepten todos los mensajes:

	Búfer 1
Identificador	XXXXXXXXXXX
Máscara	00000000000
AND	00000000000
Filtros	00000000000
¿Coinciden?	SI

Se crean dos funciones nuevas para habilitar o deshabilitar las interrupciones por recepción en los distintos búferes. Al igual que la función setRcvInt() , se modifica el registro CANINTE, pero esta vez individualmente para cada búfer.

La función setRcvInt0() habilita o deshabilita la interrupción de recepción de mensaje en el búfer 0 dependiendo de si su argumento es true o false.

```
/*********************************************************************************************************
 ** Function name:           setRcvInt0
** Descriptions:            Enable Interrupts when message arrives if true
*********************************************************************************************************/

INT8U MCP_CAN::setRcvInt0(bool res)
{
    if(res == true){
        mcp2515_modifyRegister(0x2B,0x01,0x01 );
        }
    else {
        mcp2515_modifyRegister(0x2B,0x01,0x00 );

        }


}
```

Figura 6.25. Código de la función setRcvInt0()

setRcvInt1 tiene la misma funcionalidad solo que orientada al búfer 1.

```
/*********************************************************************************************************
 ** Function name:           setRcvInt1
** Descriptions:            Enable Interrupts when message arrives if true
*********************************************************************************************************/

INT8U MCP_CAN::setRcvInt1(bool res)
{
    if(res == true ){
        mcp2515_modifyRegister(0x2B,0x02,0x02 );
        }
    else {
        mcp2515_modifyRegister(0x2B,0x02,0x00 );

        }

}
```

Figura 6.26. Código de la función setRcvInt1()

El nodo 2 contiene en el bucle infinito código para monitorizar cuándo un mensaje ha llegado correctamente. Para ello se utiliza la función CAN.checkReceive() que viene por defecto en la librería del shield, la cual devuelve CAN_MSGAVAIL si se ha recibido un mensaje válido y CAN_NOMSG si no ha recibido ningún mensaje. De esta forma leeremos el búfer de recepción únicamente cuando se haya recibido un mensaje.

Si por el contrario se leyese el búfer en todo momento y luego mandara los datos por el puerto Serial, no obtendriamos ningún valor la mayoría de las veces ya que no están llegando mensajes al búfer continuamente.

En conclusión, el funcionamiento general de la red es el siguiente:

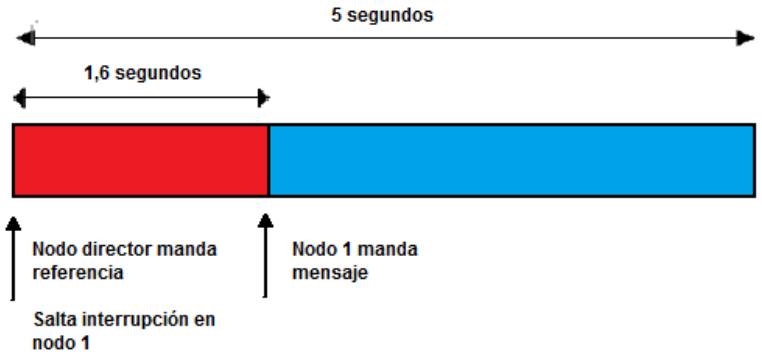

Figura 6.27. Ranuras de tiempo 2

Dado que el nodo director no se le ha asignado la capacidad de leer los mensajes que envía el nodo 1, no se puede utilizar el puerto serial para comprobar el correcto funcionamiento del programa. Sin embargo, se pueden observar los leds de cada uno de los nodos para monitorizar el programa.

El led de transmisión del nodo director se enciende cada 5 segundos tal y como dicta el programa. Pasados 1,6 segundos, el led de recepción del nodo director y el led de

transmisión e interrupción del nodo 1 se encienden simultánemente, concluyendo que el programa funciona correctamente.

Visto que la sincronización de los nodos has sido satisfactoria, se añade un nodo más a la red para acercarnos más al objetivo del trabajo. El resultado a obtener es el siguiente:

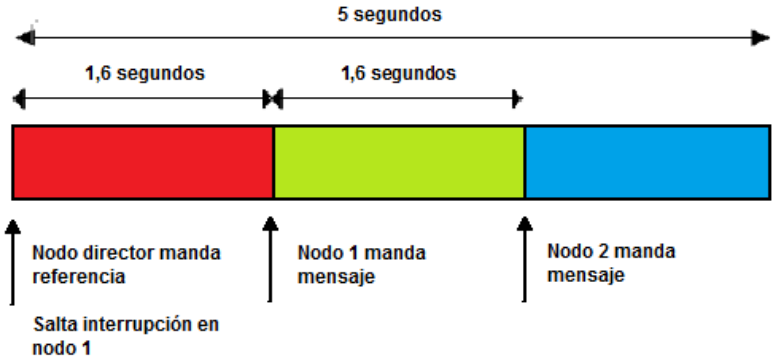

Figura 6.28. Ranuras de tiempo 3

Para ello se requiere crear un programa nuevo para el Nodo 2. Este se realizará variando algunos parámetros del programa creado para el Nodo 1:

Al Nodo 2 se le asignará un identificador diferente, el 0x02. Además mandará su mensaje 3.2 segundos tras recibir el mensaje de referencia, por lo tanto la variable counter pasará a ser 200 en vez de 100.

Una vez realizado lo anterior se cargan los diferentes programas en los nodos y se comprueba el resultado:

Al observar los LEDs en el nodo director el programa parece funcionar correctamente. En primer lugar se encienden los LEDS TX y RX indicando el envío del mensaje de referencia y a continuación pasados 1,6 segundos y 3,3 segundos, se enciende el LED de recepción RX indicando que se han recibido los mensajes por parte de los otros dos nodos.

Al observar el Monitor Serial se llega a la conclusión de que los mensajes se transmiten, pero no siempre contienen datos. Por ejemplo si monitorizamos el Nodo 1:

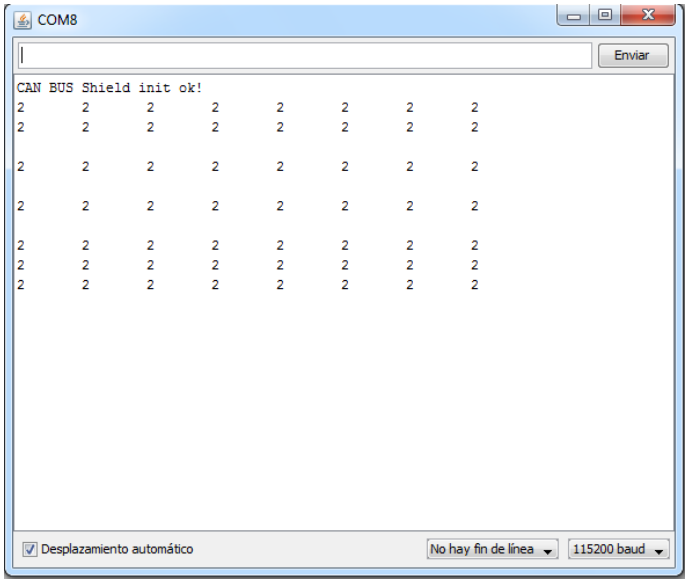

Figura 6.29. Monitor Serial del Nodo 1

Las líneas vacías indican que el mensaje no se ha recibido de forma correcta. En un sistema donde todos los datos son críticos no se podría utilizar el programa mostrado anteriormente.

Se ha observado adicionalmente otro problema relacionado con la rutina de atención a la interrupción del nodo 1. Tras estar en funcionamiento aproximadamente un minuto, el nodo 1 se queda permanentemente en la interrupción como se muestra en la siguiente imagen:

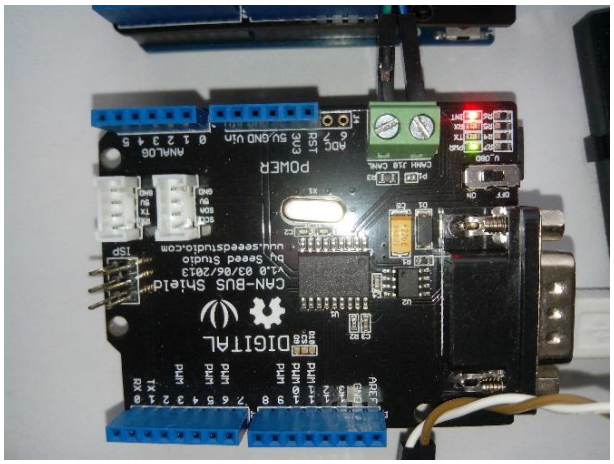

Figura 6.30. Error en nodo 1

En planteamientos anteriores de programa ocurrió lo mismo, sin embargo en el programa actual no se ha utilizado ninguna función Serial.print() dentro de las interrupciones, por lo tanto hay que buscar otra posible causa para el mal funcionamiento de la red.

El problema puede ser debido al uso de funciones provenientes de librerias en las rutinas de atención a la interrupción como la función CAN.sendMsgBuf() dentro de la interrupción del MCP2515. Además se han utilizado funciones del shield dentro de una una interrupción del microcontrolador ATMega 328P lo cual ha podido dar lugar a algún tipo de colisión entre el controlador ATMega y el MCP2515.

6.4 Desarrollo final

6.4.1 Implementación sin LCD

Visto que el anterior programa mostraba ciertos errores, se opta por cambiar algunas líneas del planteamiento anterior:

- El nodo director será el único que podrá leer los mensajes de la red.
- Se evitará usar funciones dentro de las interrupciones
- Los Nodos 1 2 y 3 solamente mandarán mensajes dentro de su ranura de tiempo.
- Se optimizará el tiempo lo máximo posible.
- El nodo director mostrará los datos recibidos en un display LCD.

En primer lugar se crea el programa para el nodo director:

```
#include <SPI.h>       //Se incluye el header de la libreria SPI (Serial Peripheral Interface)
#include "mcp_can.h"   //Se incluye el header de la libreria del Shield CAN-BUS

MCP_CAN CAN(10);                                    //Se crea una instancia de la clase MCP_CAN

//Interrupción

int counter = 0;                                     // Contador para el Timer 2
unsigned char stmp[8] = {0, 1, 2, 3, 4, 5, 6, 7};    //Se crea un vector de caracteres de longitud 8
unsigned char len = 0;                               //Variable que indica la longitud de los datos
unsigned char buf[8];                                //Vector donde se guardan los datos
unsigned char Flag_Recv = 0;                         //Variable que indica la recepción de un mensaje
boolean ensend = false;                              //Variable que habilita el envío de mensajes


void setup(){

Serial.begin(115200);                                 //Se inicializa el puerto serial a 115200
baudios

//INICIALIZAMOS EL MCP2515

START_INIT:

    if(CAN_OK == CAN.begin(CAN_500KBPS))          // inicializar can bus : baudrate = 1000k
    {
        Serial.println("CAN BUS Shield init ok!");//Si el microcontrolador se inicia correctamente,
                                                  //escribe por el puerto Serial:
                                                  //"CAN BUS Shield init ok!"
    }
    else                                          //Si el microcontrolador no se inicia
correctamente:
    {
        Serial.println("CAN BUS Shield init fail");     //Escribe por el puerto Serial
        Serial.println("Init CAN BUS Shield again");    //"CAN BUS Shield init fail" y en
```

```
                                                    //la linea siguiente:
                                                    //"Init CAN BUS Shield again"
        delay(100);                                 //Retardo de 100 milisegundos
        goto START_INIT;                            //Vuelve a la rutina de inicialización
    }


//PROGRAMAMOS EL TIMER 2

TCCR2B |=  B00000001;            //Configuramos el preescaler (1)

TCCR2B &=  B11110111;           //Configuramos los bits WGM
TCCR2A &=  B11111110;           //para establecer el modo comparación
TCCR2A |=B00000010;

TIMSK2 |= B00000010 ;           //Habilitamos interrupción del comparador A
OCR2A = B00011111;              //Establecemos valor del contador A (256)
TCNT2 = B00000000;              //Iniciamos el valor del TMR a 0



attachInterrupt(0, MCP2515_ISR, FALLING); // Añadimos la interrupción general del MCP2515


}

void MCP2515_ISR()          //Interrupción del MCP2515
{
     Flag_Recv = 1;         //Si llega un mensaje ponemos el flag a 1
}



ISR(TIMER2_COMPA_vect){     //Interrupción del Timer 2 por comparación
  counter++;                //Aumenta el contador
  if(counter == 261){       //Pasados 522 us
    ensend = true;          //Se habiilita el envío de mensajes y
    counter = 0;            //se reinicia el contador
  }
}



void loop(){

  if(ensend == true){        // Si ensend es igual a true
  CAN.sendMsgBuf(0x00, 0, 8, stmp); //Manda mensaje de referencia
  ensend = false;            //Inhabilitamos más mensajes
  }

    if(Flag_Recv) {              // Comprueba si llegan datos

    Flag_Recv = 0;               // Baja el flag


    while (CAN_MSGAVAIL == CAN.checkReceive()) { // Si el mensaje es válido

       CAN.readMsgBuf(&len, buf); //Lo lee

      //Impresión del identificador del mensaje

      Serial.println("\r\n------------------------------------------------------------------");
        Serial.print("Get Data From id: ");
        Serial.println(CAN.getCanId());

      // Impresión de los datos del mensaje

      for(int i = 0; i<len; i++) {
          Serial.print(buf[i]);Serial.print("\t");
      }
      Serial.println();
    }
  }

}
```

Como se puede comprobar, en este programa no se ha utilizado ninguna función dentro de las interrupciones. Ahora las interrupciones solo activan determinadas variables, como *ensend* o *Flag_Recv*, que harán que el programa ejecute cierta parte de código en el bucle infinito.

Para optimizar el tiempo se ha establecido que el envío de mensajes de referencia sea cada 522 µs de acuerdo a la fórmula del Timer 2. Este tiempo no ha sido elegido al azar, ya que establecimos que una trama de datos estándar a 1000 kbps tarda en torno a 130 µs en enviarse y que si la multiplicamos por cuatro (tramas) da un total de 520 µs, el cual aumentamos con un margen de 2µs. Luego nuestro tiempo de ciclo es de 522µs. En este tiempo tenemos acceso a los 3 datos provenientes de los nodos, lo cual es una velocidad más que aceptable para un sistema en tiempo real.

Dentro del bucle infinito se encuentran tanto la función de envío como de recepción. En este programa, además de imprimir los datos que son almacenados en el búfer, también se imprimirán los identificadores de los nodos, consiguiendo saber en todo momento que nodo ha mandado el mensaje. Para ello se ha utilizado una de las funciones que se incluía por defecto en la librería del shield CAN, la función CAN.getCanId().

El programa de el Nodo 1 tendrá la siguiente forma:

```
#include <SPI.h>       //Se incluye el header de la libreria SPI (Serial Peripheral Interface)
#include "mcp_can.h"   //Se incluye el header de la libreria del Shield CAN-BUS

MCP_CAN CAN(10);                                    //Se crea una instancia de la clase MCP_CAN


//Interrupción

int counter = 0;                                     // Contador para el Timer 2
boolean enable = false;                              //Variable que habilita el aumento del contador
unsigned char stmp[8] = {1, 1, 1, 1, 1, 1, 1, 1};    //Se crea un vector de caracteres de longitud 8
unsigned char len = 0;                               //Variable que indica la longitud de los datos
unsigned char buf[8];                                //Vector donde se guardan los datos
boolean ensend = false;                              //Variable que habilita el envío de mensajes


void setup(){

Serial.begin(115200);                  //Se inicializa el puerto serial a 115200 baudios

//INICIALIZAMOS EL MCP2515

START_INIT:

    if(CAN_OK == CAN.begin(CAN_1000KBPS))        // Inicializar can bus : baudrate = 1000k
    {
        Serial.println("CAN BUS Shield init ok!");//Si el microcontrolador se inicia correctamente,
                                                 //escribe por el puerto Serial:
                                                 //"CAN BUS Shield init ok!"
    }
    else                                         //Si el microcontrolador no se inicia
correctamente:
    {
        Serial.println("CAN BUS Shield init fail");     //Escribe por el puerto Serial
        Serial.println("Init CAN BUS Shield again");    //"CAN BUS Shield init fail" y en
                                                        //la linea siguiente:
                                                        //"Init CAN BUS Shield again"
```

```
        delay(100);                                     //Retardo de 100 milisegundos
        goto START_INIT;                                //Vuelve a la rutina de inicialización
    }

//CONFIGURAMOS LAS MÁSCARAS PARA QUE EN EL BUFFER 0 SOLO SE RECIBAN MENSAJES CON IDENTIFICADOR 0X00

      CAN.init_Mask(0, 0, 0xFF);    //Establecemos la máscara del buffer 0
      CAN.init_Mask(1, 0, 0x00);    //Establecemos la máscara del buffer 1

      //Establecemos los dos filtros del buffer 0

      CAN.init_Filt(0, 0, 0x00);
      CAN.init_Filt(1, 0, 0x00);

      //Establecemos los dos filtros del buffer 1

      CAN.init_Filt(2, 0, 0x00);
      CAN.init_Filt(3, 0, 0x00);
      CAN.init_Filt(4, 0, 0x00);
      CAN.init_Filt(5, 0, 0x00);

//PROGRAMAMOS EL TIMER 2

TCCR2B |=  B00000001;                   //Configuramos el preescaler (1)

TCCR2B &=  B11110111;                   //Configuramos los bits WGM
TCCR2A &=  B11111110;                   //para establecer el modo comparación
TCCR2A |=B00000010;

TIMSK2 &= B11111101 ;                   //Deshabilitamos interrupción del comparador A
OCR2A = B00011111;                      //Establecemos valor del contador A (31)
TCNT2 = B00000000;                      //Iniciamos el valor del TMR a 0

//AÑADIMOS INTERRUPCION POR MENSAJE DE REFERENCIA

attachInterrupt(0, MCP2515_ISR, FALLING);// Añadimos la interrupción general del MCP2515

//HABILITAMOS INTERRUPCION CUANDO MENSAJE DE REFERENCIA LLEGE A BUFFER 0

CAN.DisGloInt();
CAN.setRcvInt0(true);
CAN.setRcvInt1(false);



//ACTIVAMOS MODO ONE-SHOT

CAN.OSMenable();

}


void MCP2515_ISR() //Interrupción del MCP2515
{
CAN.lowRcvFlag();  //Bajada del flag de recepción
enable = true;     //Habilitamos el aumento del contador
TIMSK2 |= B00000010 ; //Habilitamos interrupción del comparador A
TCNT2 = B00000000;   //Iniciamos el valor del TMR a 0

}


ISR(TIMER2_COMPA_vect){      //Interrupción del Timer 2 por comparación


  if(enable){               //Si enable == true
  counter++;                //Aumenta el contador
  if(counter == 1){         //Pasados 2us

    TIMSK2 &= B11111101;    //Deshabilitamos interrupción del comparador A
    counter = 0;             //Se reinicia el contador
    enable = false;          //Se deshabilita el aumento del contador
    ensend = true;           //Se habiilita el envío de mensajes
    }
  }
}



void loop(){
```

```
  if(ensend){                          // Si ensend es true
  CAN.sendMsgBuf(0x01, 0, 8, stmp); //Manda mensaje
  ensend = false;                      //Inhabilitamos más mensajes
  }
}
```

Al igual que en planteamientos anteriores, se filtran los mensajes de manera que sólo los mensajes de referencia (identificador 0x00) puedan llegar al búfer 0 y desencadenar una interrupción.

En este caso se ha creado una nueva función, la función OSMenable(). Dicha función habilita el modo One-shot, el cual es requerido para un nivel 1 de TTCAN. Normalmente un nodo retransmite su mensaje cuando ha habido una colisión, pero el modo One-shot evita dicho comportamiento, consiguiendo de esta forma que los nodos respeten las ranuras de tiempo del resto de nodos. Esto no se realizó en el nodo director dado que este tiene que ser capaz de sincronizar en cualquier momento la red, por lo tanto debe ser capaz de retransmitir.

```
/*********************************************************************************************************
** Function name:           OSMenable
** Descriptions:            Enable One-Shot Mode
*********************************************************************************************************/
INT8U MCP_CAN::OSMenable(void)
{
mcp2515_modifyRegister(0x0F, 0x08, 0x08);

}
```

Figura 6.31. Código de la función OSMenable()

En la función se modifica el registro CANCTRL, dentro del cual el bit 3 OSM pemite habilitar o deshabilitar el modo One-shot dependiendo de si el bit se encuentra a 1 o a 0.

CANCTRL – CAN CONTROL REGISTER
(ADDRESS: XFh)

R/W-1	R/W-0	R/W-0	R/W-0	R/W-0	R/W-1	R/W-1	R/W-1
REQOP2	REQOP1	REQOP0	ABAT	OSM	CLKEN	CLKPRE1	CLKPRE0
bit 7							bit 0

Figura 6.32. Registro CANCTRL [5]

El nodo 1 manda un mensaje 2µs después de haber recibido el mensaje de referencia. Su identificador será el 0x01 y el los datos del mensaje serán {1, 1, 1, 1, 1, 1, 1, 1} .

El sistema de interrupciones no contiene ninguna función salvo excepto lowRcvFlag(). Esta función deshabilita el flag de interrupción por recepción de mensajes. Si no se bajara dicho flag, no se podría volver a entrar a la interrupción. Por defecto, la función readMsgBuf() es la encargada de bajar el flag de recepción,pero no es utilizada dado que

el nodo director es el único que puede leer los mensajes de la red. La función lowRcvFlag() no existe por defecto en la librería del shield CAN y tuvo que ser creada:

```
/*********************************************************************************************************
 ** Function name:           lowRcvFlag
 ** Descriptions:            LOW RECEIVE FLAGS
*********************************************************************************************************/
INT8U MCP_CAN::lowRcvFlag(void)
{
    mcp2515_modifyRegister(0x2C, 3, 0);

}
```

Figura 6.33. Código de la función lowRcvFlag()

Como se observa se tuvo que modificar el registro CANINTF con dirección 0x2C:

CANINTF – INTERRUPT FLAG (ADDRESS: 2Ch)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
MERRF	WAKIF	ERRIF	TX2IF	TX1IF	TX0IF	RX1IF	RX0IF
bit 7							bit 0

Figura 6.34. Registro CANINTF [5]

Se ponen los bits RX1IF y RX0IF (Flags de interrupción) a 0, para que se pueda volver a entrar a la interrupción general del MCP2515 cuando se haya recibido un mensaje.

En el resto de nodos se mantendrá la estructura del programa pero se modificarán ciertas variables:

En el nodo 2, se transmitirá el mensaje 130 µs después de la referencia, tendra como identificador 0x02 y mandará el mensaje {2,2,2,2,2,2,2,2}.

Nodo 2

```
  if(counter == 65){          //Pasados 2us

    TIMSK2 &= B11111101;     //Deshabilitamos interrupción del comparador A
    counter = 0;              //Se reinicia el contador
    enable = false;           //Se deshabilita el aumento del contador
    ensend = true;            //Se habiilita el envío de mensajes
    }

CAN.sendMsgBuf(0x01, 0, 8, stmp);
unsigned char stmp[8] = {2, 2, 2, 2, 2, 2, 2, 1};
```

En el nodo 3, se transmitirá el mensaje 260 µs después de la referencia, tendrá como identificador 0x03 y mandará el mensaje {3,3,3,3,3,3,3,3}.

Nodo 3

```
  if(counter == 65){          //Pasados 2us

    TIMSK2 &= B11111101;     //Deshabilitamos interrupción del comparador A
    counter = 0;              //Se reinicia el contador
    enable = false;           //Se deshabilita el aumento del contador
    ensend = true;            //Se habiilita el envío de mensajes
```

```
    }

CAN.sendMsgBuf(0x01, 0, 8, stmp);

unsigned char stmp[8] = {2, 2, 2, 2, 2, 2, 2, 1};
```

La matriz de tiempos que se pretende obtener es la siguiente:

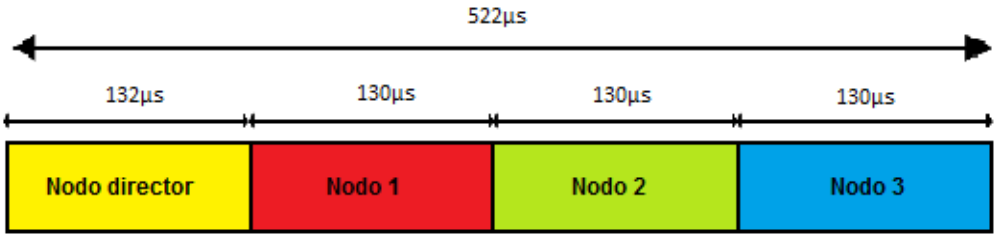

Figura 6.35. Ranuras de tiempo 4

Se comprueba el correcto funcionamiento de la red observando el puerto serial correspondiente del nodo director mediante el monitor serial de arduino.

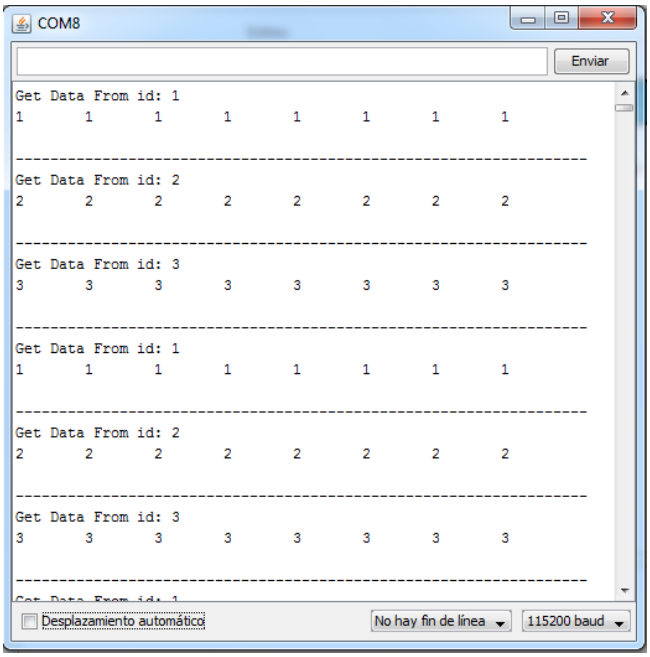

Figura 6.36. Monitor Serial del nodo director

Como se puede observar, el nodo director reccibe correctamente los mensajes de todos los nodos junto con su identificador.

6.4.2 Implementación con LCD

Una vez comprobado el correcto funcionamiento de la red, se procede a monitorizar los datos con el Display LCD:

Para poder controlar el display, se dispone de una librería diseñada exclusivamente para él **[26]**. Para poder utilizar las funciones que ésta contiene, se deben copiar los ficheros de la librería en la siguiente ruta: C:\Program Files\Arduino\libraries.

A la hora de representar los datos, se simula la red de un automóvil. Los diferentes nodos representarían diferentes sensores que envían sus datos periódicamente. Se han elegido tres parámetros: RPM (revoluciones por minuto) temperatura y velocidad, enviados por los nodos 1, 2 y 3 respectivamente.

El programa que representa los datos por el Display es el siguiente:

```
#include <SPI.h>       //Se incluye el header de la libreria SPI (Serial Peripheral Interface)
#include "mcp_can.h"   //Se incluye el header de la libreria del Shield CAN-BUS
#include <Wire.h>      //Se incluye el header de la libreria Wire
#include <Adafruit_MCP23017.h> //Se incluye el header del IC del LCD
#include <Adafruit_RGBLCDShield.h> //Se incluye el header de la libreria del LCD

MCP_CAN CAN(10);                                    //Se crea una instancia de la clase MCP_CAN
Adafruit_RGBLCDShield lcd = Adafruit_RGBLCDShield();//Se crea una instancia de la clase
                                                    //Adafruit_RGBLCDShield

// Se define el color de la retroiluminación

#define WHITE 0x7

//Interrupción

int counter = 0;                                    // Contador para el Timer 2
unsigned char stmp[8] = {0, 1, 2, 3, 4, 5, 6, 7};   //Se crea un vector de caracteres de longitud 8
unsigned char len = 0;                              //Variable que indica la longitud de los datos
unsigned char buf[8];                               //Vector donde se guardan los datos
unsigned char Flag_Recv = 0;                        //Variable que indica la recepción de un mensaje
boolean ensend = false;                             //Variable que habilita el envío de mensajes


void setup(){

Serial.begin(115200);                             //Se inicializa el puerto serial a 115200 baudios


//INICIALIZAMOS EL MCP2515

START_INIT:

    if(CAN_OK == CAN.begin(CAN_1000KBPS))          // inicializar can bus : baudrate = 1000k
    {
        Serial.println("CAN BUS Shield init ok!");//Si el microcontrolador se inicia correctamente,
                                                  //escribe por el puerto Serial:
                                                  //"CAN BUS Shield init ok!"
    }
    else                                          //Si el microcontrolador no se inicia
correctamente:
    {
        Serial.println("CAN BUS Shield init fail");     //Escribe por el puerto Serial
        Serial.println("Init CAN BUS Shield again");    //"CAN BUS Shield init fail" y en
                                                        //la linea siguiente:
                                                        //"Init CAN BUS Shield again"
        delay(100);                                     //Retardo de 100 milisegundos
        goto START_INIT;                                //Vuelve a la rutina de inicialización
    }

//PROGRAMAMOS EL TIMER 2

TCCR2B |=  B00000001;            //Configuramos el preescaler (1)

TCCR2B &=  B11110111;           //Configuramos los bits WGM
TCCR2A &=  B11111110;           //para establecer el modo comparación
```

I

II

```
TCCR2A |=B00000010;

TIMSK2 |= B00000010 ;           //Habilitamos interrupción del comparador A
OCR2A = B00011111;              //Establecemos valor del contador A (256)
TCNT2 = B00000000;              //Iniciamos el valor del TMR a 0
attachInterrupt(0, MCP2515_ISR, FALLING); // Añadimos la interrupción general del MCP2515
```

III

```
lcd.begin(16, 2);   // Se indica el numero de columnas y filas del display
lcd.setBacklight(WHITE); //Se establece el color de la luz de retroiluminación

//Escritura de los nombres de los datos de la red

lcd.setCursor(0, 0); //Función para fijar el cursor
lcd.print("RPM ");   //Fucnión para la impresión de los datos
lcd.setCursor(8, 0);
lcd.print("Tem");
lcd.setCursor(0, 1);
lcd.print("Vel");
```

IV

```
}

void MCP2515_ISR()          //Interrupción del MCP2515
{
     Flag_Recv = 1;         //Si llega un mensaje ponemos el flag a 1
}
```

V

```
ISR(TIMER2_COMPA_vect){     //Interrupción del Timer 2 por comparación
  counter++;                //Aumenta el contador
  if(counter == 261){       //Pasados 522 us
    ensend = true;          //Se habiilita el envío de mensajes y
    counter = 0;            //Se reinicia el contador
  }
}
```

VI

```
void loop(){

  if(ensend == true){               // Si ensend es igual a true
  CAN.sendMsgBuf(0x00, 0, 8, stmp); //Manda mensaje de referencia
  ensend = false;                   //Inhabilitamos más mensajes
  }

    if(Flag_Recv) {               // Comprueba si llegan datos

    Flag_Recv = 0;                // Baja el flag

    while (CAN_MSGAVAIL == CAN.checkReceive()) { // Si el mensaje es válido

      CAN.readMsgBuf(&len, buf); //Lee el mensaje

      switch(CAN.getCanId()){ //Si el identificador del mensaje es:

      // 1 se escribirán los datos correspondientes a las RPM
      case 1:
      lcd.setCursor(4,0);
      //Imprime los datos
      for(int i = 0; i<4; i++) {
          lcd.print(buf[i]);
      }
      break;


      // 2 se escribirán los datos correspondientes a la Temperatura
      case 2:
      lcd.setCursor(12,0);
      //Imprime los datos
      for(int i = 0; i<3; i++) {
          lcd.print(buf[i]);
      }
      break;


       // 3 se escribirán los datos correspondientes a la velocidad
       case 3:
       lcd.setCursor(4,1);
```

```
        //Imprime los datos
        for(int i = 0; i<3; i++) {
            lcd.print(buf[i]);
        }
        break;

        //En otro caso no hace nada
        default:

        break;

        }
      }
    }
}
```

En cuanto a la configuración del timer y el sistema de interrupciónes, el programa del nodo director funciona de la misma forma que cuando no controlaba el LCD. Sin embargo en la parte **III** del programa vemos las nuevas funciones que se introducen con el uso de el display. Aunque la librería permite controlar los botones del shield, no se ha visto necesario utilizarlos y tan solo se utilizarán funciones concernientes al display LCD:

La función lcd.begin() inicializa el display indicando el número de columnas y filas de éste en función del tamaño del display que se esté utilizando.

```
lcd.begin(16, 2);
```

La función lcd.setBacklight() indica el color de el led que retroilumina la pantalla. En otros modelos existe la posibilidad de utilizar una gama de colores, sin embargo en el que se ha adquirido solo se puede utilizar el color blanco:

```
lcd.setBacklight(WHITE);
```

Las dos últimas funciones indican el lugar donde se quiere escribir (columna, fila) y la cadena de caracteres a mostrar por pantalla:

```
lcd.setCursor(0, 0);
lcd.print("");
```

El bucle infinito (Parte **VI** del programa) también presenta diferencias respecto al programa creado anteriormente:

Una vez haya llegado un mensaje válido, se leerá y en base a su identificador se colocarán los datos en distintas partes del display. Por ejemplo, si el mensaje recibido tiene identificador 1, sabemos que los datos corresponden con las RPM, por lo tanto irán colocadas junto a la cadena de caracteres “RPM” en el LCD.

El resultado es el siguiente:

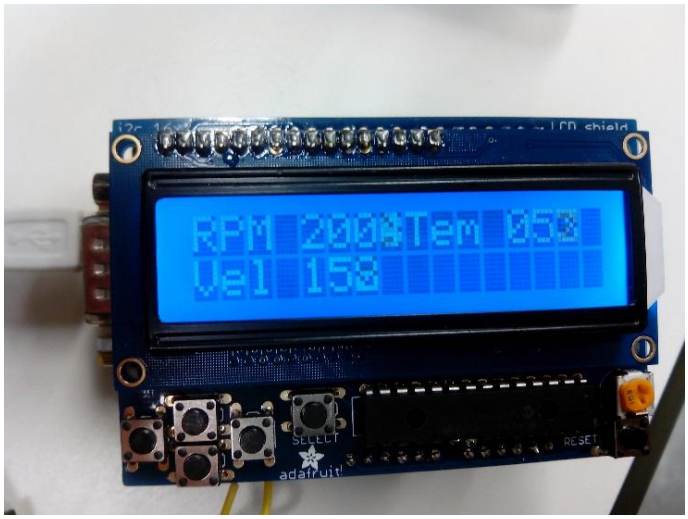

Figura 6.37. Display LCD mostrando datos

En cuanto al programa del resto de nodos, no cambiará mucho respecto a la última versión creada ya que solamente variarán los datos a enviar. Por ejemplo el nodo 1 mandará 4 digitos que representen las revoluciones por minuto:

```
unsigned char stmp[4] = {2, 0, 0, 0};
```

Además, para poder comprobar que todos los datos se reciben y visualizan correctamente, se modifica el programa de tal forma que el último digito varíe en cada mensaje. Este digito variará de 0 a 9 para poder comprobar en la pantalla LCD que se visualizan todos los números de la sucesión:

```
void loop(){

  if(ensend){

  CAN.sendMsgBuf(0x01, 0, 4, stmp);
  ensend = false;

  aux++;
  }

  if(aux == 9){
    aux=0;}
  stmp[3] =  aux ;
}
```

En este caso se varía stmp[3] ya que es el cuarto dígito.

El nodo 2 representará la temperatura por lo tanto en el fragmento anterior de código variarán tres líneas:

```
unsigned char stmp[3] = {0, 5, 0};
CAN.sendMsgBuf(0x02, 0, 3, stmp);
stmp[2] =  aux ;
```

Por último, en el nodo 3 que representa la velocidad las tres líneas de código anteriores tendrán la siguiente forma:

```
unsigned char stmp[3] = {1, 5, 0};
CAN.sendMsgBuf(0x02, 0, 3, stmp);
stmp[2] =  aux ;
```

En el nodo 2 y el nodo 3 se han utilizado tres dígitos para enviar los datos de temperatura y de la velocidad debido a que no se espera que la velocidad ni la temperatura alcancen un valor que supere las cuatro cifras.

6.4.3 Resultados

El envío recepción y comunicación de mensajes es satisfactorio, los nodos mandan sus tramas de datos de acuerdo a las ranuras de tiempo establecidas, lo cual implica que no hay colisiones de datos. Los datos de los nodos se envían cada 520 microsegundos, lo cúal es lo suficientemente rápido para un sistema en tiempo real. Se podría haber optimizado algo más el tiempo utilizado si las ranuras de tiempo se hubieran hecho mas estrechas. No obstante, esta opción no garantiza la completa evasión de las colisiones. Además, incluso optimizando la matriz de tiempos, no se puede conseguir una velocidad significativamente mayor, dado que estamos utilizando el controlador del MCP2515 a su máxima velocidad.

Esta red es ampliable a más dispositivos, incluso hasta 64 según indica el protocolo CAN bus. Una red de N dispositivos (incluyendo el nodo director) configurada de la misma forma mandaría datos periódicamente de acuerdo a la siguiente fórmula:

$$T = 130 * N$$

Luego una red en una red con 64 dispositivos un dato concreto se enviará cada 8320 microsegundos. Si dicha velocidad no es suficiente para determinados datos, se puede optar por modificar la matriz de tiempos consiguiendo que un dato se retransmita más de una vez a lo largo de un tiempo de ciclo.

Ha existido un problema a la hora de visualizar los datos en el LCD, ya que los datos, al refrescarse cada 520 microsegundos, no son visualizables. Para conseguir que el tiempo de refresco sea más lento, se puede modificar el tiempo de ciclo, es decir, el tiempo en el que se manda el mensaje de referencia:

Para ello se modifica simplemente el programa del nodo principal. Estableceremos el tiempo de ciclo a 1 segundo.

En primer lugar se configura el Timer 2:

```
TCCR2B |=B00000001;            //Configuramos el preescaler (1024)

TCCR2B &=B11110111;           //Configuramos los bits WGM
TCCR2A &=B11111110;           //para establecer el modo comparación
TCCR2A |=B00000010;

TIMSK2 |= B00000010;           //Habilitamos interrupción del comparador A
OCR2A = B01111100;              //Establecemos valor del contador A (124)
TCNT2 = B00000000;              //Iniciamos el valor del TMR a 0
```

Y finalmente el contador dentro de la interrupción del Timer 2:

```
ISR(TIMER2_COMPA_vect){      //Interrupción del Timer 2 por comparación
  counter++;                 //Aumenta el contador
  if(counter == 125){        //Pasados 522 us
    ensend = true;           //Se habilita el envío de mensajes y
    counter = 0;             //se reinicia el contador
  }
}
```

El resultado final mostrado en la pantalla es el siguiente:

Figura 6.38. Display LCD mostrando datos nítidos

De esta forma los datos se actualizan cada segundo, pudiéndose observar el cambio de dígitos en los distintos parámetros. La matriz de tiempos tiene el siguiente aspecto:

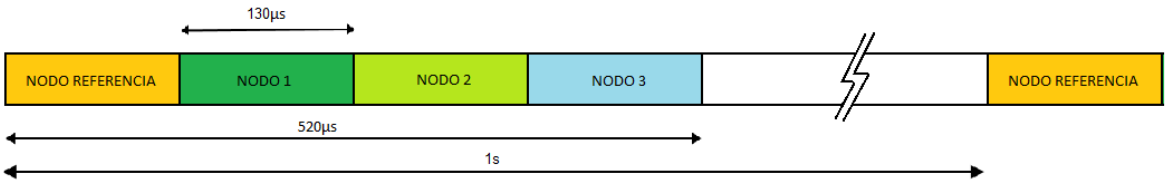

Figura 6.39. Ranuras de tiempo 5

El inconveniente de esta modificación es la menor velocidad en la que se reciben los datos, pero el nuevo funcionamiento permite la monitorización por parte de una persona.

El aspecto final de la red es el siguiente:

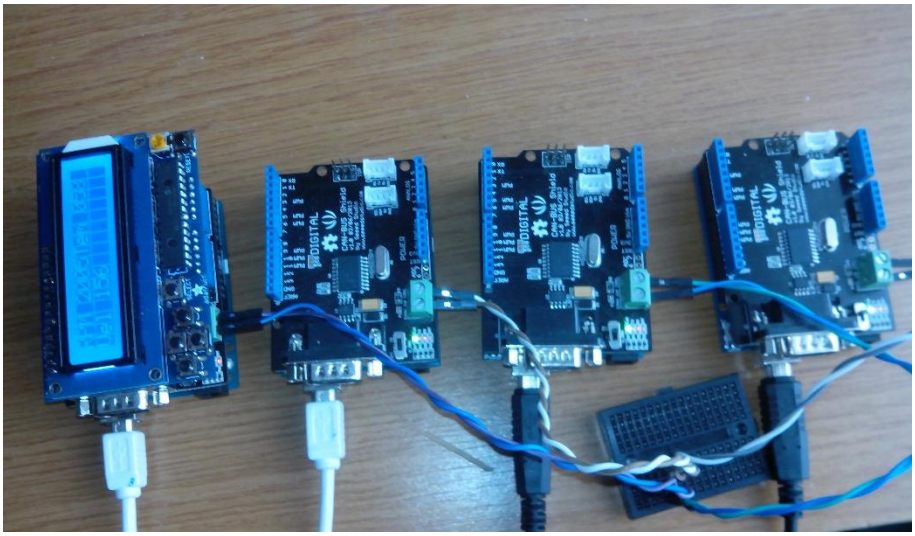

Figura 6.40. Red CAN bus con LCD para monitorizar los datos

6.5 Adquisición de datos de un vehículo con CAN Bus

6.5.1 Modelo del vehículo y protocolo utilizado

Protocolos utilizados

El protocolo OBD (On Board Diagnostics) se implementó para controlar las emisiones los vehículos. La llegada de la electrónica al control del motor permitió reducir los gases contaminantes de éstos.

Llegó por primera vez a Estados Unidos en 1996 mientras que en Europa se estableció a partir de la creación del estándar EURO3 para el control de las emisiones.

La primera versión del protocolo OBD se denominó OBD I y fue obligatoria para todos los vehículos a partir de 1991. A mediados de los noventa surgió otra versión llamada OBD-II u OBD2. Este incluye diversas mejoras frente a su antecesor, como la estandarización del conector y los pines, la implantación de una toma de la batería desde el conector y la creación de una lista de parámetros y su codificación.

En Europa existe una variación del protocolo OBD2 llamado EOBD el cuál es algo más sotisficado. El conector y las señales de comunicación son idénticas a la del OBD2, luego no se harán distinciones.

Todos los vehículos no muy antiguos contienen un terminal OBD-II para la monitorización de sus diferentes datos. El sistema de diagnóstico OBD-II soporta hasta cinco protocolos:

- SAE J1850 PWM (Pulse Width Modulation) - 41.6 kbps.
- SAE J1850 VPW (Variable Pulse Width) - 10.4 kbps.

- ISO 9141-2 – Comunicación serie asíncrona a 10.4 kbps. Similar al protocolo RS-232
- ISO 14230 KWP2000 (Keyword Protocol 2000).
- ISO 15765 CAN (250 kbps o 500 kbps).

El protocolo que utiliza el vehículo viene indicado por los contactos metálicos del interfaz OBD2. A continuación se muestran en que pines se encuentran los contactos metálicos en base al protocolo utilizado:

Figura 6.41. Red CAN bus con LCD para monitorizar los datos **[28]**

- J1850 PWM -- El conector debe tener contactos metálicos en los pines 2, 4, 5, 10, and 16.
- J1850 VPW -- El conector debe tener contactos metálicos en los pines 2, 4, 5, and 16, but not 10.
- ISO 9141-2 -- El conector debe tener contactos metálicos en los pines 4, 5, 7, 15 and 16.
- ISO 9141-2/KWP2000 -- El conector debe tener contactos metálicos en los pines 4, 5, 7, 15, and 16.
- CAN -- El conector debe tener contactos metálicos en los pines 4, 5, 6, 14 and 16.

Por existir tanta variedad, en Estados Unidos se estableció el uso obligatorio de CAN bus en los vehículos a partir de 2008. Este protocolo está ampliamente extendido y se encuentra en casi la totalidad de los coches actuales. Es importante destacar que el

conector puede tener contactos metálicos en los pines correspondientes a dos o más protocolos, aunque utilice solo uno de ellos.

Primer vehículo utilizado para la comunicación

El vehículo utilizado es un Fabia Combi de la marca Skoda.

Figura 6.42. Vehículo Skoda Fabia Combi **[29]**

Para determinar que protocolo utiliza el vehículo, se observan qué pines del terminal OBD2 están metalizados.

El terminal se encuentra debajo del salpicadero, en la parte izquierda y tras un compartimento.

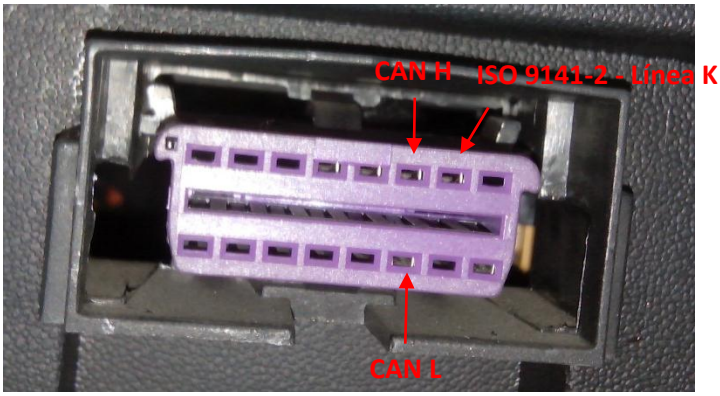

Figura 6.43. Terminal OBDII

La imagen anterior determina que el vehiculo utiliza o bien CAN-Bus o bien el protocolo ISO 9141. La única forma de comprobarlo es utilizado el shield CAN-Bus de Seeeduino junto con el cable OBD2.

Modos del vehículo y PIDs

El protocolo OBD2 puede soportar hasta 10 modos de diagnóstico. Si el vehículo es reciente, hay más posibilidades de que la ECU soporte un mayor rango de modos.

- Modo 1

Devuelve los valores de los diferentes sensores del vehículo. Cada sensor se identifica con un número llamado PID (Parameter Identifier). Por ejemplo, la velocidad tiene el PID número 0x0C.

- Modo 2

Devuelve una trama de datos instantánea de una falla en un sensor.

- Modo 3

Muestra los códigos de diagnóstico del vehículo. Son códigos estándar para todos los vehículos.

- Modo 4

Se utiliza para eliminar los códigos de falla almacenados y desactivar el piloto de error en el motor.

- Modo 5

Devuelve los resultados tras un auto-diagnóstico sobre los sensores de oxígeno

- Modo 6

Devuelve los resultados tras un auto-diagnóstico sobre parámetros que no están sometidos a un control constante

- Modo 7

Este modo muestra los códigos de error sin confirmar.

- Modo 8

Devuelve los resultados tras un auto-diagnóstico sobre parámetros en otros sistemas

- Modo 9

Proporciona información sobre el vehículo como el número identificador y los valores de calibración.

- Modo 10

Muestra los códigos de falla permanentes.

6.5.2 Desarrollo del programa para la adquisición de datos

Para la adquisición de datos utilizaremos el programa ejemplo llamado receive_check que venia incluido junto con la librería del shield y lo modificamos para que en lugar de mandar los datos recibidos a través del puerto serial, los muestre en el display LCD:

```
// demo: CAN-BUS Shield, receive data with check mode
// send data coming to fast, such as less than 10ms, you can use this way
// loovee, 2014-6-13


#include <SPI.h>
#include "mcp_can.h"
#include <Wire.h>
#include <Adafruit_MCP23017.h> //Se incluye el header del IC del LCD
#include <Adafruit_RGBLCDShield.h> //Se incluye el header de la libreria del LCD


unsigned char Flag_Recv = 0;
unsigned char len = 0;
unsigned char buf[8];
char str[20];


MCP_CAN CAN(10);         // Set CS to pin 10
Adafruit_RGBLCDShield lcd = Adafruit_RGBLCDShield();//Se crea una instancia

// Se define el color de la retroiluminación

#define WHITE 0x7


void setup()
{
    Serial.begin(115200);
    // Inicializamos el numero de columnas y filas del LCD
    lcd.begin(16, 2);
    lcd.setBacklight(WHITE);

START_INIT:
    if(CAN_OK == CAN.begin(CAN_500KBPS))//can bus : baudrate = 500k
    {
        Serial.println("CAN BUS Shield init ok!");
    }
    else
    {
        Serial.println("CAN BUS Shield init fail");
        Serial.println("Init CAN BUS Shield again");
        delay(100);
        goto START_INIT;
    }
}

void loop()
{
    lcd.setCursor(0, 0); //Colocamos el cursor en la fila cero columna cero
    if(CAN_MSGAVAIL == CAN.checkReceive())          // check if data coming
    {
        CAN.readMsgBuf(&len, buf);    // lee datos,  len: longitude datos, buf: data buf

        for(int i = 0; i<len; i++)    // imprime los datos
        {
            lcd.print(buf[i]);lcd.print("\t");
        }
        lcd.println();
    }
}

/*********************************************************************
  END FILE
*********************************************************************/
```

Este sencillo programa mostrará los datos provenientes del vehículo. Para ello se carga el programa en el Arduino UNO junto con el shield y el display y se conecta el terminal DB9. Se conecta el conector OBD2 correctamente al terminal del vehículo y se pone el coche en ignición. Esta operación se puede realizar sin ningún dispositivo externo que alimente al Arduino, dado que este se alimenta directamente de la batería.

Una vez realizados los pasos anteriores se observa la pantalla LCD para ver el resultado. El resultado difiere del esperado, ya que la pantalla se mantiene encendida pero no muestra ningún dato en ella.

Tras ahondar más en la comunicación CAN Bus de un vehículo, se comprueba que el vehículo no envía datos a través del terminal OBD2 de forma normal. Se debe hacer una petición a través del dispositivo que conectemos a la ECU (engine control unit) del vehículo **[30]**. Dicha petición se debe realizar mediante el envío de un mensaje con un identificador y unos datos determinados. El identificador utilizado es el 0x7DF.

Dependiendo del mensaje que enviemos, podremos obtener diferentes datos del coche. La estructura de los datos a enviar según la SAE (Society of Automotive Engineers) es la siguiente:

0	1	2	3	4	5	6	7
0x02	0x01	0xXX	0x00	0x00	0x00	0x00	0x00

El tercer byte del mensaje corresponde al PID (Performance Information Data) que se desea monitorizar (Velocidad, revoluciones por minuto, etc).

Lo anterior podrá variar según el vehículo utilizado, ya que no todos los fabricantes se ciñen al estándar.

Una vez recopilada la información, se procede a modificar el programa anteriormente creado. Tan sólo varía ligeramente:

- Se crea el vector stmp con una longitud de 8 caracteres:

```
unsigned char stmp[8];
```

- Se añaden los datos a mandar:

```
stmp[0] = 0x02;
stmp[1] = 0x01;
stmp[2] = 0x0D; //Velocidad del coche
stmp[3] = 0x00;
stmp[4] = 0x00;
stmp[5] = 0x00;
stmp[6] = 0x00;
stmp[7] = 0x00;
```

- Por último se manda un mensaje continuamente colocando la siguiente función al comienzo de la rutina loop():

```
CAN.sendMsgBuf(0x7DF, 0, 8, stmp);
```

Se carga el programa y una vez más se conecta el Arduino al vehículo mediante el cable OBD2/DB9. Al encender el coche, no se obtiene ningún dato. Es más, los leds del shield permanecen en rojo, indicando que los mensajes se han quedado en el buffer, es decir, que no están siendo recibidos. Para intentar solventarlo se modifica la velocidad de CAN-Bus a una diferente de 500Kbps. Sin embargo, dichas modifaciones tampoco tienen éxito.

Ante dichos problemas, se intenta realizar las pruebas en otro vehículo más actual.

Segundo vehículo utilizado para la comunicación

Se utiliza el Volkswagen Touran ya que es un vehículo que salió al mercado en 2013. Al ser un vehículo más reciente, es más probable que utilice comunicación CAN Bus.

Figura 6.44. Vehículo Volkswagen Touran **[31]**

Al igual que el Skoda Fabia, tiene el terminal OBD2 situado tras el compartimento del conductor. Para llegar al terminal, es preciso abrir el compartimento y tirar de él con cuidado. El terminal tiene los mismos pines metalizados que el Skoda.

Con el mismo programa de antes y conectando el terminal al VW Touran, el resultado es completamente diferente. Esta vez si se muestran datos en la pantalla del LCD. Se llega a la conclusión de que el Skoda Fabia no utilizaba protocolo CAN Bus, sino el ISO 9142 a través de su línea K.

Figura 6.45. LCD mostrando datos del vehículo

El display muestra una serie de datos correspondientes a la velocidad. Adicionalmente se monitorizan otros parámetros variando el byte 3 de datos, como las revoluciones por minuto y la temperatura.

El próximo paso es la interpretación de los datos:

3651300000202

El byte 1 de datos indica que estamos recibiendo datos actuales.

El byte 2 de datos indica el PID que estamos monitorizando.

Los bytes 3, 4, 5 y 6 contienen los datos del parámetro que estamos observando.

Cabe destaca que los datos mostrados en el LCD se muestran en decimal y no en hexadecimal. Esto puede ser resuelto fácilmente añadiendo el argumento HEX a la función lcd.print(): `lcd.print("",HEX)`

En base al parámetro que estemos observando, los bytes 4, 5, 6 y 7 se traducirán de una forma u otra al valor real del parámetro. En base a la complejidad del dato se utilizarán mas bytes de datos o menos.

Teniendo lo anterior en cuenta, se procede a crear un programa que muestre los datos reales de los parámetros del vehículo. Además se aprovecharán los pulsadores que posee el shield LCD para elegir el parámetro que se desee. En el programa siguiente se monitorizarán la temperatura ambiente, la velocidad y las revoluciones por minuto, ya que son valores faciles de corroborar con los mostrados en el salpicadero, puediendo comprobar que la traducción de datos está siendo correcta.

```
#include <SPI.h>       //Se incluye el header de la libreria SPI (Serial Peripheral Interface)
#include "mcp_can.h"   //Se incluye el header de la libreria del Shield CAN-BUS
#include <Wire.h>      //Se incluye el header de la libreria Wire
#include <Adafruit_MCP23017.h> //Se incluye el header del IC del LCD
#include <Adafruit_RGBLCDShield.h> //Se incluye el header de la libreria del LCD

MCP_CAN CAN(10);                                    //Se crea una instancia de la clase MCP_CAN
Adafruit_RGBLCDShield lcd = Adafruit_RGBLCDShield(); //Se crea una instancia
                                                     //de la clase Adafruit_RGBLCDShield

// Se define el color de la retroiluminación

#define WHITE 0x7
```

```
//Interrupción

unsigned char len = 0;                    //Variable que indica la longitud de los datos
unsigned char buf[8];                     //Vector donde se guardan los datos recibidos
unsigned char stmp[8];                    //Vector donde se guardan los datos a enviar
unsigned char Flag_Recv = 0;              //Variable que indica la recepción de un mensaje
boolean select_button = true;             //Variable que habilita la lectura de botones
int parametro = 0;                        //Variable para cada parámetro


void setup()
{
    Serial.begin(115200);                //Inicializa el puerto serial a 115200 baudios

//Inicialización del shield

START_INIT:

    if(CAN_OK == CAN.begin(CAN_500KBPS))      // can bus : baudrate = 500k
    {
        Serial.println("CAN BUS Shield init ok!");
    }
    else
    {
        Serial.println("CAN BUS Shield init fail");
        Serial.println("Init CAN BUS Shield again");
        delay(100);
        goto START_INIT;
    }


attachInterrupt(0, MCP2515_ISR, FALLING); // Añadimos la interrupción general del MCP2515

lcd.begin(16, 2);   // Se indica el numero de columnas y filas del display
lcd.setBacklight(WHITE); //Se establece el color de la luz de retroiluminación
lcd.clear();             //Se borra la pantalla LCD



lcd.setCursor(0, 0); //Función para fijar el cursor
lcd.print("ELIJA PARAMETRO"); //Se muestra en pantalla un mensaje

//Se establecen los datos que se enviarán

stmp[0] = 0x02;
stmp[1] = 0x01;
stmp[2] = 0x0D; //Velocidad del coche
stmp[3] = 0x00;
stmp[4] = 0x00;
stmp[5] = 0x00;
stmp[6] = 0x00;
stmp[7] = 0x00;

}


void MCP2515_ISR()          //Interrupción del MCP2515
{
     Flag_Recv = 1;         //Si llega un mensaje ponemos el flag a 1
}

void loop()
{


  while( select_button ){ //Se leen los botones del shield
                          //hasta que select_button = false

      uint8_t buttons = lcd.readButtons(); //Se guarda la lectura en una variable

      if (buttons) {      //Si se pulsa un boton cualquiera
      lcd.clear();        //Se borra la pantalla LCD
      lcd.setCursor(0,0); //SE coloca el cursor al comienzo del LCD
```

```
      if (buttons & BUTTON_LEFT) { //Si se pulsa el boton izquierdo
        delay(100);                //Retardo para evitar rebotes
        if (parametro != 0) {parametro--;}//Se disminuye el parametro
      }
      if (buttons & BUTTON_RIGHT) { //Si se pulsa el boton derecho
        delay(100);                 //Retardo para evitar rebotes
        if (parametro != 2){parametro++;} //Se aumenta el parametro
      }
      if (buttons & BUTTON_SELECT) { //Si se pulsa el boton select
        delay(100);                  //Retardo para evitar rebotes
        select_button = false;       //Se selecciona el parametro
      }

      //En base al parametro se escriben diferentes mensajes y se modifica el byte 2

      switch ( parametro ){

        case 0:         lcd.print("VELOCIDAD ");
                        stmp[2] = 0x0D; //Velocidad del coche
        break;

        case 1:         lcd.print("RPM ");
                        stmp[2] = 0x0C; //Velocidad del coche
        break;


        case 2:         lcd.print("TEMPERATURA ");
                        stmp[2] = 0x46; //Velocidad del coche
        break;


    }

  }

}



  CAN.sendMsgBuf(0x7DF, 0, 8, stmp); //Se envia un mensaje con identicador 7DF
  delay(10);                         //Retardo


      if(Flag_Recv) {              // Comprueba si llegan datos

      Flag_Recv = 0;               // Baja el flag

        while (CAN_MSGAVAIL == CAN.checkReceive()) { // Si el mensaje es válido
            lcd.clear();               //Borra lo escrito anteriormente
            CAN.readMsgBuf(&len, buf); //Lee el mensaje
            lcd.setCursor(0, 0);       //Fija el cursor en el comienzo

            //En base al parámetro escribe el byte tres traducido a la variable real
            // y escribe el nombre de este

            switch (parametro){


                    case 0:         lcd.print("VELOCIDAD ");
                      lcd.setCursor(0, 1); //Función para fijar el cursor
                      lcd.print(buf[3]);
                    break;

                    case 1:         lcd.print("RPM ");
                      lcd.setCursor(0, 1); //Función para fijar el cursor
                      lcd.print(((buf[3]*256)+buf[4])/4);
                    break;

                    case 2:         lcd.print("TEMPERATURA ");
                      lcd.setCursor(0, 1); //Función para fijar el cursor
                      lcd.print(buf[3]-40);
                    break;


              }
          }
```

```
        }
}

/*****************************************************************************************************
  END FILE
*****************************************************************************************************/
```

Se comprueba efectivamente que los parámetros mostrados en el display corresponden con los mostrados en el salpicadero.

Figura 6.46. Comparación temperatura en LCD y en salpicadero

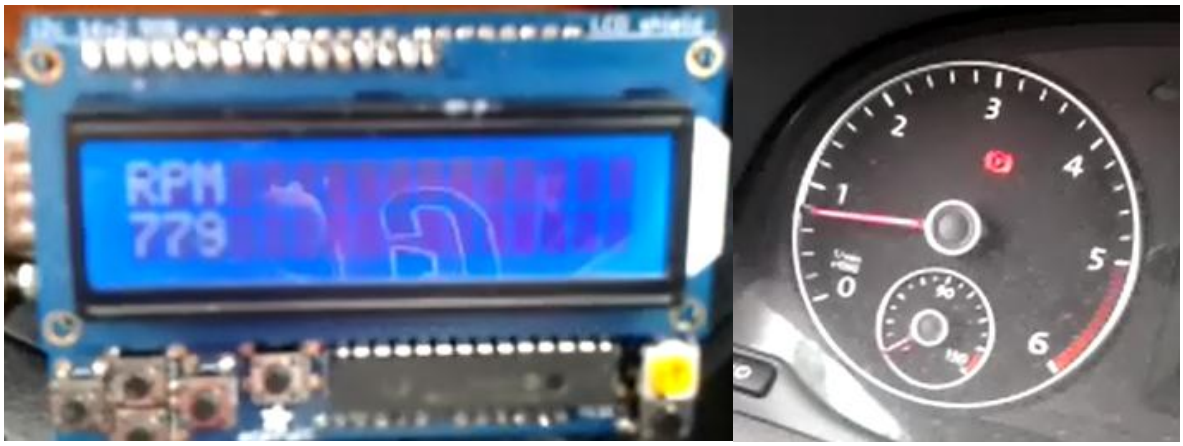

Figura 6.47. Comparación revoluciones por minuto en LCD y en salpicadero

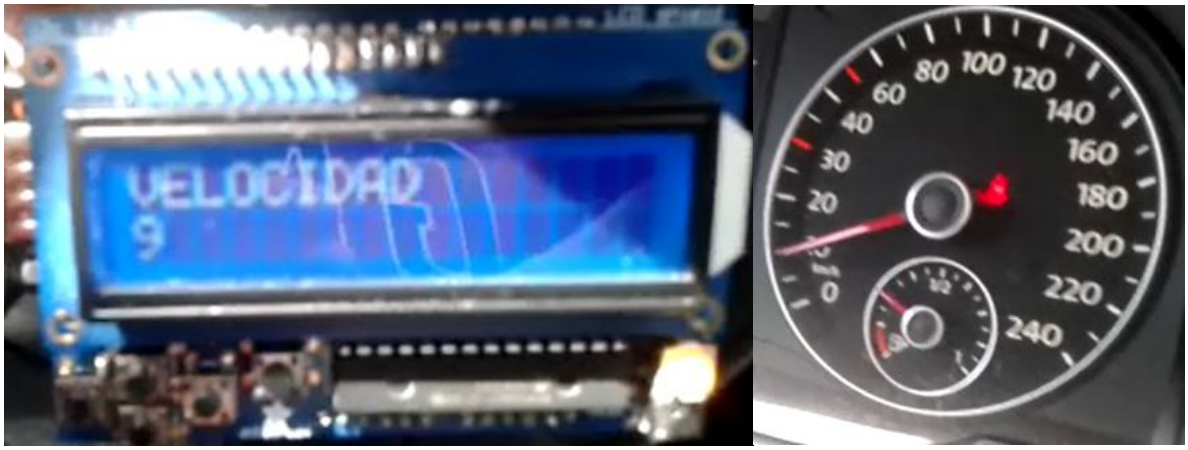

Figura 6.48. Comparación velocidad en LCD y en salpicadero

Para poder mostrar el cambio de los parámetros en tiempo real se adjunta un video en el que se observa el proceso de adquisición de datos:

https://drive.google.com/file/d/0Bz-s78Jo-5_0UzM4SDZxekUxbkE/view?usp=sharing

Queda comprobada la compatibilidad del shield con el vehículo y la posibilidad de adquisición de datos a través de este.

7. Presupuesto

7.1 Gastos materiales

	Cantidad	Precio Unitario	Precio total
CAN Bus Shield de Seeduino	4	21,05 €	84,20 €
Arduino UNO	4	26,50 €	106,00 €
Display LCD de Adafruit	1	13,23 €	13,23 €
Cable USB tipo A/B (1,8 metros)	4	4,99 €	19,96 €
Cable OBD2-DB9	1	7,03 €	7,03 €
Cables Dupont macho-macho (pack de 10)	1	2,10 €	2,10 €
Miniprotoboard	1	2,38 €	2,38 €
Importe Total			234,90 €

7.2 Gastos mano de obra

Puesto	Trabajo realizado	Trabajadores	Horas	Salario	Importe
Ingeniero Industrial, esp. electrónica industrial	Realización TFG	1	360	14,00 €	5.040,00 €

7.3 Gasto total

Importe total gastos directos	5274,9 €
Importe total gastos indirectos (10% gastos directos)	527,49 €
Importe total antes de impuestos	5802,39 €
Importe total después de impuestos (21% IVA)	7020,89 €

El coste total del trabajo asciende a la cantidad de SIETE MIL VEINTE euros y OCHENTA Y NUEVE céntimos de euro.

8. Conclusiones

Se han resuelto satisfactoriamente las dos partes del trabajo. En primer lugar se ha conseguido realizar una red CAN Bus con un presupuesto reducido debido al hardware empleado. El Arduino Uno debido a su software libre y a su bajo consumo ha superado las expectativas, si bien es cierto que se podría haber reducido la carga de trabajo utilizando un microcontrolador en el cual se pueda utilizar un debugger. El Shield de Seeeduino ha dado un buen resultado a la hora de transmitir datos mediante protocolo Can Bus, además su terminal DB9 integrado ha facilitado en gran medida la comunicación con el vehículo.

En cuanto al protocolo utilizado en la red, se ha optado por el protocolo TTCAN ya que es adecuado para aplicaciones en las que se requiera un seguimiento de datos en un tiempo determinista. De acuerdo al programa realizado, los distintos nodos mandan un mensaje cada 522 µs, el cual será mostrado en el nodo director, que posee el display LCD.

Respecto a la segunda parte del trabajo, se ha llegado a la conclusión de que la tecnología CAN Bus en el futuro ocupará todo el mercado ya que todos los vehículos de menos de 6 años lo implementan. Al utilizarse un interfaz estandarizado (OBD2), ha sido sencilla la comunicación con el CAN Bus Shield mediante un cable OBD2/DB9. El programa realizado ordena el envío de un mensaje de consulta a la ECU del vehículo y a continuación escucha el mensaje que le llega desde esta. De esta manera se pueden monitorizar todos los sensores de los que disponga el vehículo. Con unos componentes sencillos: un Arduino UNO, un shield y un LCD, se consigue una interfaz para observar datos como la velocidad o las revoluciones por minuto del vehículo.

ANEXOS

Librería CAN-BUS Shield

Archivo mcp_can.h

```
#ifndef _MCP2515_H_
#define _MCP2515_H_

#include "mcp_can_dfs.h"

#define MAX_CHAR_IN_MESSAGE 8

class MCP_CAN
{
    private:

    INT8U   m_nExtFlg;                                  /* identifier xxxID             */
                                                        /* either extended (the 29 LSB) */
                                                        /* or standard (the 11 LSB)     */
    INT32U  m_nID;                                      /* can id                       */
    INT8U   m_nDlc;                                     /* data length:                 */
    INT8U   m_nDta[MAX_CHAR_IN_MESSAGE];                /* data                         */
    INT8U   m_nRtr;                                     /* rtr                          */
    INT8U   m_nfilhit;
    INT8U   SPICS;

/*
*  mcp2515 driver function
*/
   // private:
private:

    void mcp2515_reset(void);                           /* reset mcp2515                */

    INT8U mcp2515_readRegister(const INT8U address);    /* read mcp2515's register      */

    void mcp2515_readRegisterS(const INT8U address,
                               INT8U values[],
                                   const INT8U n);
    void mcp2515_setRegister(const INT8U address,       /* set mcp2515's register       */
                                 const INT8U value);

    void mcp2515_setRegisterS(const INT8U address,      /* set mcp2515's registers      */
                                  const INT8U values[],
                                  const INT8U n);

    void mcp2515_initCANBuffers(void);

    void mcp2515_modifyRegister(const INT8U address,    /* set bit of one register      */
                                    const INT8U mask,
                                    const INT8U data);

    INT8U mcp2515_readStatus(void);                     /* read mcp2515's Status        */
    INT8U mcp2515_readRXStatus(void);
    INT8U mcp2515_setCANCTRL_Mode(const INT8U newmode); /* set mode                     */
    INT8U mcp2515_configRate(const INT8U canSpeed);     /* set boadrate                 */
    INT8U mcp2515_init(const INT8U canSpeed);           /* mcp2515init                  */

    void mcp2515_write_id( const INT8U mcp_addr,        /* write can id                 */
                                  const INT8U ext,
```

```
56.                                 const INT32U id );
57.
58.         void mcp2515_read_id( const INT8U mcp_addr,                         /* read can id                 */
59.                                     INT8U* ext,
60.                                     INT32U* id );
61.
62.         void mcp2515_write_canMsg( const INT8U buffer_sidh_addr );          /* write can msg               */
63.         void mcp2515_read_canMsg( const INT8U buffer_sidh_addr);            /* read can msg                */
64.         void mcp2515_start_transmit(const INT8U mcp_addr);                  /* start transmit              */
65.         INT8U mcp2515_getNextFreeTXBuf(INT8U *txbuf_n);                     /* get Next free txbuf         */
66.
67.     /*
68.     *  can operator function
69.     */
70.
71.         INT8U setMsg(INT32U id, INT8U ext, INT8U len, INT8U rtr, INT8U *pData); /* set message                 */

72.         INT8U setMsg(INT32U id, INT8U ext, INT8U len, INT8U *pData); /* set message                 */
73.         INT8U clearMsg();                                           /* clear all message to zero    */
74.         INT8U readMsg();                                            /* read message                 */
75.         INT8U sendMsg();                                            /* send message                 */
76.
77.     public:
78.         MCP_CAN(INT8U _CS);
79.         INT8U begin(INT8U speedset);                                /* init can                     */
80.         INT8U init_Mask(INT8U num, INT8U ext, INT32U ulData);       /* init Masks                   */
81.         INT8U init_Filt(INT8U num, INT8U ext, INT32U ulData);       /* init filters                 */
82.         INT8U sendMsgBuf(INT32U id, INT8U ext, INT8U rtr, INT8U len, INT8U *buf);   /* send buf
          */
83.         INT8U sendMsgBuf(INT32U id, INT8U ext, INT8U len, INT8U *buf);   /* send buf                  */
84.         INT8U readMsgBuf(INT8U *len, INT8U *buf);                   /* read buf                     */
85.         INT8U readMsgBufID(INT32U *ID, INT8U *len, INT8U *buf);     /* read buf with object ID      */
86.         INT8U checkReceive(void);                                   /* if something received        */
87.         INT8U checkError(void);                                     /* if something error           */
88.         INT32U getCanId(void);                                      /* get can id when receive      */
89.         INT8U isRemoteRequest(void);                                /* get RR flag when receive     */
90.         INT8U writeStId(INT8U id);
91.         INT8U setRcvInt(bool res);
92.         INT8U lowRcvFlag(void);
93.         INT8U DisGloInt(void);
94.         INT8U OSMenable(void);
95.         INT8U setRcvInt0(bool res);
96.         INT8U setRcvInt1(bool res);
97.         };
98.
99.     #endif
100.    /*********************************************************************************************************
101.      END FILE
102.    *********************************************************************************************************/
```

Archivo mcp_can.cpp

```
/*
  mcp_can.cpp
  2012 Copyright (c) Seeed Technology Inc.  All right reserved.

  Author:Loovee
  Contributor: Cory J. Fowler
  2014-1-16
  This library is free software; you can redistribute it and/or
  modify it under the terms of the GNU Lesser General Public
  License as published by the Free Software Foundation; either
  version 2.1 of the License, or (at your option) any later version.

  This library is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
  Lesser General Public License for more details.

  You should have received a copy of the GNU Lesser General Public
  License along with this library; if not, write to the Free Software
  Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-
  1301  USA
*/
#include "mcp_can.h"

#define spi_readwrite SPI.transfer
#define spi_read() spi_readwrite(0x00)

/*********************************************************************************************************
** Function name:           mcp2515_reset
** Descriptions:            reset the device
*********************************************************************************************************/
void MCP_CAN::mcp2515_reset(void)
{
    MCP2515_SELECT();
    spi_readwrite(MCP_RESET);
    MCP2515_UNSELECT();
    delay(10);
}

/*********************************************************************************************************
** Function name:           mcp2515_readRegister
** Descriptions:            read register
*********************************************************************************************************/
INT8U MCP_CAN::mcp2515_readRegister(const INT8U address)
{
    INT8U ret;

    MCP2515_SELECT();
    spi_readwrite(MCP_READ);
    spi_readwrite(address);
    ret = spi_read();
    MCP2515_UNSELECT();

    return ret;
}

/*********************************************************************************************************
** Function name:           mcp2515_readRegisterS
** Descriptions:            read registerS
*********************************************************************************************************/
void MCP_CAN::mcp2515_readRegisterS(const INT8U address, INT8U values[], const INT8U n)
{
    INT8U i;
```

```
64.         MCP2515_SELECT();
65.         spi_readwrite(MCP_READ);
66.         spi_readwrite(address);
67.         // mcp2515 has auto-increment of address-pointer
68.         for (i=0; i<n && i<CAN_MAX_CHAR_IN_MESSAGE; i++) {
69.             values[i] = spi_read();
70.         }
71.         MCP2515_UNSELECT();
72.     }
73.
74.     /*********************************************************************************************************
75.     ** Function name:           mcp2515_setRegister
76.     ** Descriptions:            set register
77.     *********************************************************************************************************/
78.     void MCP_CAN::mcp2515_setRegister(const INT8U address, const INT8U value)
79.     {
80.         MCP2515_SELECT();
81.         spi_readwrite(MCP_WRITE);
82.         spi_readwrite(address);
83.         spi_readwrite(value);
84.         MCP2515_UNSELECT();
85.     }
86.
87.     /*********************************************************************************************************
88.     ** Function name:           mcp2515_setRegisterS
89.     ** Descriptions:            set registerS
90.     *********************************************************************************************************/
91.     void MCP_CAN::mcp2515_setRegisterS(const INT8U address, const INT8U values[], const INT8U n)
92.     {
93.         INT8U i;
94.         MCP2515_SELECT();
95.         spi_readwrite(MCP_WRITE);
96.         spi_readwrite(address);
97.
98.         for (i=0; i<n; i++)
99.         {
100.            spi_readwrite(values[i]);
101.        }
102.        MCP2515_UNSELECT();
103.    }
104.
105.    /*********************************************************************************************************
106.    ** Function name:           mcp2515_modifyRegister
107.    ** Descriptions:            set bit of one register
108.    *********************************************************************************************************/
109.    void MCP_CAN::mcp2515_modifyRegister(const INT8U address, const INT8U mask, const INT8U data)
110.    {
111.        MCP2515_SELECT();
112.        spi_readwrite(MCP_BITMOD);
113.        spi_readwrite(address);
114.        spi_readwrite(mask);
115.        spi_readwrite(data);
116.        MCP2515_UNSELECT();
117.    }
118.
119.    /*********************************************************************************************************
120.    ** Function name:           mcp2515_readStatus
121.    ** Descriptions:            read mcp2515's Status
122.    *********************************************************************************************************/
123.    INT8U MCP_CAN::mcp2515_readStatus(void)
124.    {
125.        INT8U i;
126.        MCP2515_SELECT();
127.        spi_readwrite(MCP_READ_STATUS);
128.        i = spi_read();
129.        MCP2515_UNSELECT();
```

```
130.
131.        return i;
132.    }
133.
134.    /*********************************************************************************************************
135.    ** Function name:           mcp2515_readRXStatus
136.    ** Descriptions:            read mcp2515's RX Status
137.    *********************************************************************************************************/
138.    INT8U MCP_CAN::mcp2515_readRXStatus(void)
139.    {
140.        INT8U i;
141.        MCP2515_SELECT();
142.        spi_readwrite(MCP_RX_STATUS);
143.        i = spi_read();
144.        MCP2515_UNSELECT();
145.
146.        return i;
147.    }
148.
149.    /*********************************************************************************************************
150.    ** Function name:           mcp2515_setCANCTRL_Mode
151.    ** Descriptions:            set control mode
152.    *********************************************************************************************************/
153.    INT8U MCP_CAN::mcp2515_setCANCTRL_Mode(const INT8U newmode)
154.    {
155.        INT8U i;
156.
157.        mcp2515_modifyRegister(MCP_CANCTRL, MODE_MASK, newmode);
158.
159.        i = mcp2515_readRegister(MCP_CANCTRL);
160.        i &= MODE_MASK;
161.
162.        if ( i == newmode )
163.        {
164.            return MCP2515_OK;
165.        }
166.
167.        return MCP2515_FAIL;
168.
169.    }
170.
171.    /*********************************************************************************************************
172.    ** Function name:           mcp2515_configRate
173.    ** Descriptions:            set boadrate
174.    *********************************************************************************************************/
175.    INT8U MCP_CAN::mcp2515_configRate(const INT8U canSpeed)
176.    {
177.        INT8U set, cfg1, cfg2, cfg3;
178.        set = 1;
179.        switch (canSpeed)
180.        {
181.            case (CAN_5KBPS):
182.            cfg1 = MCP_16MHz_5kBPS_CFG1;
183.            cfg2 = MCP_16MHz_5kBPS_CFG2;
184.            cfg3 = MCP_16MHz_5kBPS_CFG3;
185.            break;
186.
187.            case (CAN_10KBPS):
188.            cfg1 = MCP_16MHz_10kBPS_CFG1;
189.            cfg2 = MCP_16MHz_10kBPS_CFG2;
190.            cfg3 = MCP_16MHz_10kBPS_CFG3;
191.            break;
192.
193.            case (CAN_20KBPS):
194.            cfg1 = MCP_16MHz_20kBPS_CFG1;
195.            cfg2 = MCP_16MHz_20kBPS_CFG2;
```

```
196.            cfg3 = MCP_16MHz_20kBPS_CFG3;
197.            break;
198.
199.            case (CAN_31K25BPS):
200.            cfg1 = MCP_16MHz_31k25BPS_CFG1;
201.            cfg2 = MCP_16MHz_31k25BPS_CFG2;
202.            cfg3 = MCP_16MHz_31k25BPS_CFG3;
203.            break;
204.
205.            case (CAN_40KBPS):
206.            cfg1 = MCP_16MHz_40kBPS_CFG1;
207.            cfg2 = MCP_16MHz_40kBPS_CFG2;
208.            cfg3 = MCP_16MHz_40kBPS_CFG3;
209.            break;
210.
211.            case (CAN_50KBPS):
212.            cfg1 = MCP_16MHz_50kBPS_CFG1;
213.            cfg2 = MCP_16MHz_50kBPS_CFG2;
214.            cfg3 = MCP_16MHz_50kBPS_CFG3;
215.            break;
216.
217.            case (CAN_80KBPS):
218.            cfg1 = MCP_16MHz_80kBPS_CFG1;
219.            cfg2 = MCP_16MHz_80kBPS_CFG2;
220.            cfg3 = MCP_16MHz_80kBPS_CFG3;
221.            break;
222.
223.            case (CAN_100KBPS):                                    /* 100KBPS                    */
224.            cfg1 = MCP_16MHz_100kBPS_CFG1;
225.            cfg2 = MCP_16MHz_100kBPS_CFG2;
226.            cfg3 = MCP_16MHz_100kBPS_CFG3;
227.            break;
228.
229.            case (CAN_125KBPS):
230.            cfg1 = MCP_16MHz_125kBPS_CFG1;
231.            cfg2 = MCP_16MHz_125kBPS_CFG2;
232.            cfg3 = MCP_16MHz_125kBPS_CFG3;
233.            break;
234.
235.            case (CAN_200KBPS):
236.            cfg1 = MCP_16MHz_200kBPS_CFG1;
237.            cfg2 = MCP_16MHz_200kBPS_CFG2;
238.            cfg3 = MCP_16MHz_200kBPS_CFG3;
239.            break;
240.
241.            case (CAN_250KBPS):
242.            cfg1 = MCP_16MHz_250kBPS_CFG1;
243.            cfg2 = MCP_16MHz_250kBPS_CFG2;
244.            cfg3 = MCP_16MHz_250kBPS_CFG3;
245.            break;
246.
247.            case (CAN_500KBPS):
248.            cfg1 = MCP_16MHz_500kBPS_CFG1;
249.            cfg2 = MCP_16MHz_500kBPS_CFG2;
250.            cfg3 = MCP_16MHz_500kBPS_CFG3;
251.            break;
252.
253.            case (CAN_1000KBPS):
254.            cfg1 = MCP_16MHz_1000kBPS_CFG1;
255.            cfg2 = MCP_16MHz_1000kBPS_CFG2;
256.            cfg3 = MCP_16MHz_1000kBPS_CFG3;
257.            break;
258.
259.            default:
260.            set = 0;
261.            break;
```

```
262.        }
263.
264.        if (set) {
265.            mcp2515_setRegister(MCP_CNF1, cfg1);
266.            mcp2515_setRegister(MCP_CNF2, cfg2);
267.            mcp2515_setRegister(MCP_CNF3, cfg3);
268.            return MCP2515_OK;
269.        }
270.        else {
271.            return MCP2515_FAIL;
272.        }
273.    }
274.
275.    /*********************************************************************************************************
276.    ** Function name:           mcp2515_initCANBuffers
277.    ** Descriptions:            init canbuffers
278.    *********************************************************************************************************/
279.    void MCP_CAN::mcp2515_initCANBuffers(void)
280.    {
281.        INT8U i, a1, a2, a3;
282.
283.        INT8U std = 0;
284.        INT8U ext = 1;
285.        INT32U ulMask = 0x00, ulFilt = 0x00;
286.
287.
288.        //mcp2515_write_id(MCP_RXM0SIDH, ext, ulMask);          /*Set both masks to 0           */
289.        //mcp2515_write_id(MCP_RXM1SIDH, ext, ulMask);          /*Mask register ignores ext bit */
290.
291.                                                              /* Set all filters to 0         */
292.        //mcp2515_write_id(MCP_RXF0SIDH, ext, ulFilt);          /* RXB0: extended               */
293.        //mcp2515_write_id(MCP_RXF1SIDH, std, ulFilt);          /* RXB1: standard               */
294.        //mcp2515_write_id(MCP_RXF2SIDH, ext, ulFilt);          /* RXB2: extended               */
295.        //mcp2515_write_id(MCP_RXF3SIDH, std, ulFilt);          /* RXB3: standard               */
296.        //mcp2515_write_id(MCP_RXF4SIDH, ext, ulFilt);
297.        //mcp2515_write_id(MCP_RXF5SIDH, std, ulFilt);
298.
299.                                                                        /* Clear, deactivate the three  */
300.                                                                        /* transmit buffers             */
301.                                                                        /* TXBnCTRL -> TXBnD7           */
302.        a1 = MCP_TXB0CTRL;
303.        a2 = MCP_TXB1CTRL;
304.        a3 = MCP_TXB2CTRL;
305.        for (i = 0; i < 14; i++) {                                      /* in-buffer loop               */
306.            mcp2515_setRegister(a1, 0);
307.            mcp2515_setRegister(a2, 0);
308.            mcp2515_setRegister(a3, 0);
309.            a1++;
310.            a2++;
311.            a3++;
312.        }
313.        mcp2515_setRegister(MCP_RXB0CTRL, 0);
314.        mcp2515_setRegister(MCP_RXB1CTRL, 0);
315.    }
316.
317.    /*********************************************************************************************************
318.    ** Function name:           mcp2515_init
319.    ** Descriptions:            init the device
320.    *********************************************************************************************************/
321.    INT8U MCP_CAN::mcp2515_init(const INT8U canSpeed)                   /* mcp2515init                  */
322.    {
323.
324.      INT8U res;
325.
326.        mcp2515_reset();
327.
```

```
328.        res = mcp2515_setCANCTRL_Mode(MODE_CONFIG);
329.        if(res > 0)
330.        {
331.    #if DEBUG_MODE
332.          Serial.print("Enter setting mode fall\r\n");
333.    #else
334.          delay(10);
335.    #endif
336.          return res;
337.        }
338.    #if DEBUG_MODE
339.        Serial.print("Enter setting mode success \r\n");
340.    #else
341.        delay(10);
342.    #endif
343.
344.                                                        /* set boadrate                */
345.        if(mcp2515_configRate(canSpeed))
346.        {
347.    #if DEBUG_MODE
348.          Serial.print("set rate fall!!\r\n");
349.    #else
350.          delay(10);
351.    #endif
352.          return res;
353.        }
354.    #if DEBUG_MODE
355.        Serial.print("set rate success!!\r\n");
356.    #else
357.        delay(10);
358.    #endif
359.
360.        if ( res == MCP2515_OK ) {
361.
362.                                                        /* init canbuffers             */
363.            mcp2515_initCANBuffers();
364.
365.                                                        /* interrupt mode              */
366.            mcp2515_setRegister(MCP_CANINTE, MCP_RX0IF | MCP_RX1IF);
367.
368.    #if (DEBUG_RXANY==1)
369.                                                        /* enable both receive-buffers  */
370.                                                        /* to receive any message       */
371.                                                        /* and enable rollover          */
372.            mcp2515_modifyRegister(MCP_RXB0CTRL,
373.            MCP_RXB_RX_MASK | MCP_RXB_BUKT_MASK,
374.            MCP_RXB_RX_ANY | MCP_RXB_BUKT_MASK);
375.            mcp2515_modifyRegister(MCP_RXB1CTRL, MCP_RXB_RX_MASK,
376.            MCP_RXB_RX_ANY);
377.    #else
378.                                                        /* enable both receive-buffers  */
379.                                                        /* to receive messages          */
380.                                                        /* with std. and ext. identifie */
381.                                                        /* rs                           */
382.                                                        /* and enable rollover          */
383.            mcp2515_modifyRegister(MCP_RXB0CTRL,
384.            MCP_RXB_RX_MASK | MCP_RXB_BUKT_MASK,
385.            MCP_RXB_RX_STDEXT | MCP_RXB_BUKT_MASK );
386.            mcp2515_modifyRegister(MCP_RXB1CTRL, MCP_RXB_RX_MASK,
387.            MCP_RXB_RX_STDEXT);
388.    #endif
389.                                                        /* enter normal mode           */
390.            res = mcp2515_setCANCTRL_Mode(MODE_NORMAL);
391.            if(res)
392.            {
393.    #if DEBUG_MODE
```

```
394.            Serial.print("Enter Normal Mode Fall!!\r\n");
395.    #else
396.              delay(10);
397.    #endif
398.            return res;
399.          }
400.
401.
402.    #if DEBUG_MODE
403.            Serial.print("Enter Normal Mode Success!!\r\n");
404.    #else
405.              delay(10);
406.    #endif
407.
408.        }
409.        return res;
410.
411.    }
412.
413.    /*********************************************************************************************************
414.    ** Function name:           mcp2515_write_id
415.    ** Descriptions:            write can id
416.    *********************************************************************************************************/
417.    void MCP_CAN::mcp2515_write_id( const INT8U mcp_addr, const INT8U ext, const INT32U id )
418.    {
419.        uint16_t canid;
420.        INT8U tbufdata[4];
421.
422.        canid = (uint16_t)(id & 0x0FFFF);
423.
424.        if ( ext == 1)
425.        {
426.            tbufdata[MCP_EID0] = (INT8U) (canid & 0xFF);
427.            tbufdata[MCP_EID8] = (INT8U) (canid >> 8);
428.            canid = (uint16_t)(id >> 16);
429.            tbufdata[MCP_SIDL] = (INT8U) (canid & 0x03);
430.            tbufdata[MCP_SIDL] += (INT8U) ((canid & 0x1C) << 3);
431.            tbufdata[MCP_SIDL] |= MCP_TXB_EXIDE_M;
432.            tbufdata[MCP_SIDH] = (INT8U) (canid >> 5 );
433.        }
434.        else
435.        {
436.            tbufdata[MCP_SIDH] = (INT8U) (canid >> 3 );
437.            tbufdata[MCP_SIDL] = (INT8U) ((canid & 0x07 ) << 5);
438.            tbufdata[MCP_EID0] = 0;
439.            tbufdata[MCP_EID8] = 0;
440.        }
441.        mcp2515_setRegisterS( mcp_addr, tbufdata, 4 );
442.    }
443.
444.    /*********************************************************************************************************
445.    ** Function name:           mcp2515_read_id
446.    ** Descriptions:            read can id
447.    *********************************************************************************************************/
448.    void MCP_CAN::mcp2515_read_id( const INT8U mcp_addr, INT8U* ext, INT32U* id )
449.    {
450.        INT8U tbufdata[4];
451.
452.        *ext = 0;
453.        *id = 0;
454.
455.        mcp2515_readRegisterS( mcp_addr, tbufdata, 4 );
456.
457.        *id = (tbufdata[MCP_SIDH]<<3) + (tbufdata[MCP_SIDL]>>5);
458.
459.        if ( (tbufdata[MCP_SIDL] & MCP_TXB_EXIDE_M) ==  MCP_TXB_EXIDE_M )
```

```
460.        {
461.                                                                    /* extended id                  */
462.            *id = (*id<<2) + (tbufdata[MCP_SIDL] & 0x03);
463.            *id = (*id<<8) + tbufdata[MCP_EID8];
464.            *id = (*id<<8) + tbufdata[MCP_EID0];
465.            *ext = 1;
466.        }
467.    }
468.
469.    /*********************************************************************************************************
470.    ** Function name:           mcp2515_write_canMsg
471.    ** Descriptions:            write msg
472.    *********************************************************************************************************/
473.    void MCP_CAN::mcp2515_write_canMsg( const INT8U buffer_sidh_addr)
474.    {
475.        INT8U mcp_addr;
476.        mcp_addr = buffer_sidh_addr;
477.        mcp2515_setRegisterS(mcp_addr+5, m_nDta, m_nDlc );                  /* write data bytes             */
478.        if ( m_nRtr == 1)                                                   /* if RTR set bit in byte       */
479.        {
480.            m_nDlc |= MCP_RTR_MASK;
481.        }
482.        mcp2515_setRegister((mcp_addr+4), m_nDlc );                        /* write the RTR and DLC        */
483.        mcp2515_write_id(mcp_addr, m_nExtFlg, m_nID );                     /* write CAN id                 */
484.
485.    }
486.
487.    /*********************************************************************************************************
488.    ** Function name:           mcp2515_read_canMsg
489.    ** Descriptions:            read message
490.    *********************************************************************************************************/
491.    void MCP_CAN::mcp2515_read_canMsg( const INT8U buffer_sidh_addr)        /* read can msg                 */
492.    {
493.        INT8U mcp_addr, ctrl;
494.
495.        mcp_addr = buffer_sidh_addr;
496.
497.        mcp2515_read_id( mcp_addr, &m_nExtFlg,&m_nID );
498.
499.        ctrl = mcp2515_readRegister( mcp_addr-1 );
500.        m_nDlc = mcp2515_readRegister( mcp_addr+4 );
501.
502.        if ((ctrl & 0x08)) {
503.            m_nRtr = 1;
504.        }
505.        else {
506.            m_nRtr = 0;
507.        }
508.
509.        m_nDlc &= MCP_DLC_MASK;
510.        mcp2515_readRegisterS( mcp_addr+5, &(m_nDta[0]), m_nDlc );
511.    }
512.
513.    /*********************************************************************************************************
514.    ** Function name:           sendMsg
515.    ** Descriptions:            send message
516.    *********************************************************************************************************/
517.    void MCP_CAN::mcp2515_start_transmit(const INT8U mcp_addr)              /* start transmit               */
518.    {
519.        mcp2515_modifyRegister( mcp_addr-1 , MCP_TXB_TXREQ_M, MCP_TXB_TXREQ_M );
520.    }
521.
522.    /*********************************************************************************************************
523.    ** Function name:           sendMsg
524.    ** Descriptions:            send message
525.    *********************************************************************************************************/
```

```
526.  INT8U MCP_CAN::mcp2515_getNextFreeTXBuf(INT8U *txbuf_n)                 /* get Next free txbuf          */
527.  {
528.      INT8U res, i, ctrlval;
529.      INT8U ctrlregs[MCP_N_TXBUFFERS] = { MCP_TXB0CTRL, MCP_TXB1CTRL, MCP_TXB2CTRL };
530.
531.      res = MCP_ALLTXBUSY;
532.      *txbuf_n = 0x00;
533.
534.                                                                          /* check all 3 TX-Buffers       */
535.      for (i=0; i<MCP_N_TXBUFFERS; i++) {
536.          ctrlval = mcp2515_readRegister( ctrlregs[i] );
537.          if ( (ctrlval & MCP_TXB_TXREQ_M) == 0 ) {
538.              *txbuf_n = ctrlregs[i]+1;                                   /* return SIDH-address of Buffe */
539.                                                                          /* r                            */
540.              res = MCP2515_OK;
541.              return res;                                                 /* ! function exit              */
542.          }
543.      }
544.      return res;
545.  }
546.
547.  /*********************************************************************************************************
548.  ** Function name:           set CS
549.  ** Descriptions:            init CS pin and set UNSELECTED
550.  *********************************************************************************************************/
551.  MCP_CAN::MCP_CAN(INT8U _CS)
552.  {
553.      SPICS = _CS;
554.      pinMode(SPICS, OUTPUT);
555.      MCP2515_UNSELECT();
556.  }
557.
558.  /*********************************************************************************************************
559.  ** Function name:           init
560.  ** Descriptions:            init can and set speed
561.  *********************************************************************************************************/
562.  INT8U MCP_CAN::begin(INT8U speedset)
563.  {
564.      INT8U res;
565.
566.      SPI.begin();
567.      res = mcp2515_init(speedset);
568.      if (res == MCP2515_OK) return CAN_OK;
569.      else return CAN_FAILINIT;
570.  }
571.
572.  /*********************************************************************************************************
573.  ** Function name:           init_Mask
574.  ** Descriptions:            init canid Masks
575.  *********************************************************************************************************/
576.  INT8U MCP_CAN::init_Mask(INT8U num, INT8U ext, INT32U ulData)
577.  {
578.      INT8U res = MCP2515_OK;
579.  #if DEBUG_MODE
580.      Serial.print("Begin to set Mask!!\r\n");
581.  #else
582.      delay(10);
583.  #endif
584.      res = mcp2515_setCANCTRL_Mode(MODE_CONFIG);
585.      if(res > 0){
586.  #if DEBUG_MODE
587.      Serial.print("Enter setting mode fall\r\n");
588.  #else
589.      delay(10);
590.  #endif
591.      return res;
```

```
592.        }
593.
594.        if (num == 0){
595.            mcp2515_write_id(MCP_RXM0SIDH, ext, ulData);
596.
597.        }
598.        else if(num == 1){
599.            mcp2515_write_id(MCP_RXM1SIDH, ext, ulData);
600.        }
601.        else res =  MCP2515_FAIL;
602.
603.        res = mcp2515_setCANCTRL_Mode(MODE_NORMAL);
604.        if(res > 0){
605.    #if DEBUG_MODE
606.        Serial.print("Enter normal mode fall\r\n");
607.    #else
608.        delay(10);
609.    #endif
610.        return res;
611.      }
612.    #if DEBUG_MODE
613.        Serial.print("set Mask success!!\r\n");
614.    #else
615.        delay(10);
616.    #endif
617.        return res;
618.    }
619.
620.    /*********************************************************************************************************
621.    ** Function name:           init_Filt
622.    ** Descriptions:            init canid filters
623.    *********************************************************************************************************/
624.    INT8U MCP_CAN::init_Filt(INT8U num, INT8U ext, INT32U ulData)
625.    {
626.        INT8U res = MCP2515_OK;
627.    #if DEBUG_MODE
628.        Serial.print("Begin to set Filter!!\r\n");
629.    #else
630.        delay(10);
631.    #endif
632.        res = mcp2515_setCANCTRL_Mode(MODE_CONFIG);
633.        if(res > 0)
634.        {
635.    #if DEBUG_MODE
636.          Serial.print("Enter setting mode fall\r\n");
637.    #else
638.          delay(10);
639.    #endif
640.          return res;
641.        }
642.
643.        switch( num )
644.        {
645.            case 0:
646.            mcp2515_write_id(MCP_RXF0SIDH, ext, ulData);
647.            break;
648.
649.            case 1:
650.            mcp2515_write_id(MCP_RXF1SIDH, ext, ulData);
651.            break;
652.
653.            case 2:
654.            mcp2515_write_id(MCP_RXF2SIDH, ext, ulData);
655.            break;
656.
657.            case 3:
```

```
658.            mcp2515_write_id(MCP_RXF3SIDH, ext, ulData);
659.            break;
660.
661.            case 4:
662.            mcp2515_write_id(MCP_RXF4SIDH, ext, ulData);
663.            break;
664.
665.            case 5:
666.            mcp2515_write_id(MCP_RXF5SIDH, ext, ulData);
667.            break;
668.
669.            default:
670.            res = MCP2515_FAIL;
671.        }
672.
673.        res = mcp2515_setCANCTRL_Mode(MODE_NORMAL);
674.        if(res > 0)
675.        {
676.    #if DEBUG_MODE
677.          Serial.print("Enter normal mode fall\r\nSet filter fail!!\r\n");
678.    #else
679.          delay(10);
680.    #endif
681.          return res;
682.        }
683.    #if DEBUG_MODE
684.        Serial.print("set Filter success!!\r\n");
685.    #else
686.        delay(10);
687.    #endif
688.
689.        return res;
690.    }
691.
692.    /*********************************************************************************************************
693.    ** Function name:           setMsg
694.    ** Descriptions:            set can message, such as dlc, id, dta[] and so on
695.    *********************************************************************************************************/
696.    INT8U MCP_CAN::setMsg(INT32U id, INT8U ext, INT8U len, INT8U rtr, INT8U *pData)
697.    {
698.        int i = 0;
699.        m_nExtFlg = ext;
700.        m_nID     = id;
701.        m_nDlc    = len;
702.        m_nRtr    = rtr;
703.        for(i = 0; i<MAX_CHAR_IN_MESSAGE; i++)
704.        {
705.            m_nDta[i] = *(pData+i);
706.        }
707.        return MCP2515_OK;
708.    }
709.
710.
711.    /*********************************************************************************************************
712.    ** Function name:           setMsg
713.    ** Descriptions:            set can message, such as dlc, id, dta[] and so on
714.    *********************************************************************************************************/
715.    INT8U MCP_CAN::setMsg(INT32U id, INT8U ext, INT8U len, INT8U *pData)
716.    {
717.        int i = 0;
718.        m_nExtFlg = ext;
719.        m_nID     = id;
720.        m_nDlc    = len;
721.        for(i = 0; i<MAX_CHAR_IN_MESSAGE; i++)
722.        {
723.            m_nDta[i] = *(pData+i);
```

```
724.        }
725.        return MCP2515_OK;
726.    }
727.
728.    /*********************************************************************************************************
729.    ** Function name:           clearMsg
730.    ** Descriptions:            set all message to zero
731.    *********************************************************************************************************/
732.    INT8U MCP_CAN::clearMsg()
733.    {
734.        m_nID       = 0;
735.        m_nDlc      = 0;
736.        m_nExtFlg   = 0;
737.        m_nRtr      = 0;
738.        m_nfilhit   = 0;
739.        for(int i = 0; i<m_nDlc; i++ )
740.          m_nDta[i] = 0x00;
741.
742.        return MCP2515_OK;
743.    }
744.
745.    /*********************************************************************************************************
746.    ** Function name:           sendMsg
747.    ** Descriptions:            send message
748.    *********************************************************************************************************/
749.    INT8U MCP_CAN::sendMsg()
750.    {
751.        INT8U res, res1, txbuf_n;
752.        uint16_t uiTimeOut = 0;
753.
754.        do {
755.            res = mcp2515_getNextFreeTXBuf(&txbuf_n);                       /* info = addr.                */
756.            uiTimeOut++;
757.        } while (res == MCP_ALLTXBUSY && (uiTimeOut < TIMEOUTVALUE));
758.
759.        if(uiTimeOut == TIMEOUTVALUE)
760.        {
761.            return CAN_GETTXBFTIMEOUT;                                      /* get tx buff time out        */
762.        }
763.        uiTimeOut = 0;
764.        mcp2515_write_canMsg( txbuf_n);
765.        mcp2515_start_transmit( txbuf_n );
766.        do
767.        {
768.            uiTimeOut++;
769.            res1= mcp2515_readRegister(txbuf_n);                            /* read send buff ctrl reg  */
770.            res1 = res1 & 0x08;
771.        }while(res1 && (uiTimeOut < TIMEOUTVALUE));
772.        if(uiTimeOut == TIMEOUTVALUE)                                       /* send msg timeout            */
773.        {
774.            return CAN_SENDMSGTIMEOUT;
775.        }
776.        return CAN_OK;
777.
778.    }
779.
780.    /*********************************************************************************************************
781.    ** Function name:           sendMsgBuf
782.    ** Descriptions:            send buf
783.    *********************************************************************************************************/
784.    INT8U MCP_CAN::sendMsgBuf(INT32U id, INT8U ext, INT8U rtr, INT8U len, INT8U *buf)
785.    {
786.        setMsg(id, ext, len, rtr, buf);
787.        sendMsg();
788.    }
789.
```

```
790.  /*********************************************************************************************************
791.  ** Function name:           sendMsgBuf
792.  ** Descriptions:            send buf
793.  *********************************************************************************************************/
794.  INT8U MCP_CAN::sendMsgBuf(INT32U id, INT8U ext, INT8U len, INT8U *buf)
795.  {
796.      setMsg(id, ext, len, buf);
797.      sendMsg();
798.  }
799.
800.
801.  /*********************************************************************************************************
802.  ** Function name:           readMsg
803.  ** Descriptions:            read message
804.  *********************************************************************************************************/
805.  INT8U MCP_CAN::readMsg()
806.  {
807.      INT8U stat, res;
808.
809.      stat = mcp2515_readStatus();
810.
811.      if ( stat & MCP_STAT_RX0IF )                                    /* Msg in Buffer 0              */
812.      {
813.          mcp2515_read_canMsg( MCP_RXBUF_0);
814.          mcp2515_modifyRegister(MCP_CANINTF, MCP_RX0IF, 0);
815.          res = CAN_OK;
816.      }
817.      else if ( stat & MCP_STAT_RX1IF )                               /* Msg in Buffer 1              */
818.      {
819.          mcp2515_read_canMsg( MCP_RXBUF_1);
820.          mcp2515_modifyRegister(MCP_CANINTF, MCP_RX1IF, 0);
821.          res = CAN_OK;
822.      }
823.      else
824.      {
825.          res = CAN_NOMSG;
826.      }
827.      return res;
828.  }
829.
830.  /*********************************************************************************************************
831.  ** Function name:           readMsgBuf
832.  ** Descriptions:            read message buf
833.  *********************************************************************************************************/
834.  INT8U MCP_CAN::readMsgBuf(INT8U *len, INT8U buf[])
835.  {
836.      INT8U  rc;
837.
838.      rc = readMsg();
839.
840.      if (rc == CAN_OK) {
841.         *len = m_nDlc;
842.         for(int i = 0; i<m_nDlc; i++) {
843.           buf[i] = m_nDta[i];
844.         }
845.      } else {
846.           *len = 0;
847.      }
848.      return rc;
849.  }
850.
851.  /*********************************************************************************************************
852.  ** Function name:           readMsgBufID
853.  ** Descriptions:            read message buf and can bus source ID
854.  *********************************************************************************************************/
855.  INT8U MCP_CAN::readMsgBufID(INT32U *ID, INT8U *len, INT8U buf[])
```

```
856.  {
857.      INT8U rc;
858.      rc = readMsg();
859.
860.      if (rc == CAN_OK) {
861.         *len = m_nDlc;
862.         *ID  = m_nID;
863.         for(int i = 0; i<m_nDlc && i < MAX_CHAR_IN_MESSAGE; i++) {
864.            buf[i] = m_nDta[i];
865.         }
866.      } else {
867.         *len = 0;
868.      }
869.      return rc;
870.  }
871.
872.  /*********************************************************************************************************
873.  ** Function name:           checkReceive
874.  ** Descriptions:            check if got something
875.  *********************************************************************************************************/
876.  INT8U MCP_CAN::checkReceive(void)
877.  {
878.      INT8U res;
879.      res = mcp2515_readStatus();                                         /* RXnIF in Bit 1 and 0         */
880.      if ( res & MCP_STAT_RXIF_MASK )
881.      {
882.          return CAN_MSGAVAIL;
883.      }
884.      else
885.      {
886.          return CAN_NOMSG;
887.      }
888.  }
889.
890.  /*********************************************************************************************************
891.  ** Function name:           checkError
892.  ** Descriptions:            if something error
893.  *********************************************************************************************************/
894.  INT8U MCP_CAN::checkError(void)
895.  {
896.      INT8U eflg = mcp2515_readRegister(MCP_EFLG);
897.
898.      if ( eflg & MCP_EFLG_ERRORMASK )
899.      {
900.          return CAN_CTRLERROR;
901.      }
902.      else
903.      {
904.          return CAN_OK;
905.      }
906.  }
907.
908.  /*********************************************************************************************************
909.  ** Function name:           getCanId
910.  ** Descriptions:            when receive something ,u can get the can id!!
911.  *********************************************************************************************************/
912.  INT32U MCP_CAN::getCanId(void)
913.  {
914.      return m_nID;
915.  }
916.
917.  /*********************************************************************************************************
918.  ** Function name:           isRemoteRequest
919.  ** Descriptions:            when receive something ,u can check if it was a request
920.  *********************************************************************************************************/
921.  INT8U MCP_CAN::isRemoteRequest(void)
```

```
922.  {
923.      return m_nRtr;
924.  }
925.
926.  /*********************************************************************************************************
927.  ** Function name:           writeStId
928.  ** Descriptions:            Establish Standard Identifier
929.  *********************************************************************************************************/
930.  INT8U MCP_CAN::writeStId(INT8U id)
931.  {
932.          uint16_t canid;
933.          INT8U tbufdata[4];
934.
935.          canid = (uint16_t)(id & 0x0FFFF);
936.
937.              tbufdata[MCP_SIDH] = (INT8U) (canid >> 3 );
938.              tbufdata[MCP_SIDL] = (INT8U) ((canid & 0x07 ) << 5);
939.              tbufdata[MCP_EID0] = 0;
940.              tbufdata[MCP_EID8] = 0;
941.
942.          mcp2515_setRegisterS(0x31, tbufdata, 4 );
943.  }
944.
945.  /*********************************************************************************************************
946.   ** Function name:          setRcvInt
947.  ** Descriptions:            Enable Interrupts when message arrives if true
948.  *********************************************************************************************************/
949.
950.  INT8U MCP_CAN::setRcvInt(bool res)
951.  {
952.      if(res){
953.          mcp2515_setRegister(0x0C,0x00 );
954.          mcp2515_modifyRegister(0x2B,0x03,0x03 );
955.          }
956.      else {
957.              mcp2515_setRegister(0x0C,0x00 );
958.              mcp2515_modifyRegister(0x2B,0x03,0x00 );
959.
960.          }
961.
962.  }
963.
964.  /*********************************************************************************************************
965.   ** Function name:          setRcvInt
966.  ** Descriptions:            Enable Interrupts when message arrives if true
967.  *********************************************************************************************************/
968.  INT8U MCP_CAN::sendRmtMsg(INT8U id)
969.  {
970.      sendMsgBuf(id, 0, 1, 8, 0x00);
971.
972.  }
973.
974.  /*********************************************************************************************************
975.   ** Function name:          setRcvInt
976.  ** Descriptions:            Enable Interrupts when message arrives if true
977.  *********************************************************************************************************/
978.  INT8U MCP_CAN::loadMsg(INT32U id, INT8U ext, INT8U rtr, INT8U len, INT8U *buf)
979.  {
980.
981.          setMsg(id, ext, len, rtr, buf);//Set message parameters for the message
982.
983.
984.          //This part of the function writes the message into the buffer WITHOUT SENDING IT
985.
986.          INT8U res, res1, txbuf_n;
987.              uint16_t uiTimeOut = 0;
```

```
988.
989.                do {
990.                    res = mcp2515_getNextFreeTXBuf(&txbuf_n);                       /* info = addr.
        */
991.                    uiTimeOut++;
992.                } while (res == MCP_ALLTXBUSY && (uiTimeOut < TIMEOUTVALUE));
993.
994.                if(uiTimeOut == TIMEOUTVALUE)
995.                {
996.                    return CAN_GETTXBFTIMEOUT;                                      /* get tx buff time out
        */
997.                }
998.                uiTimeOut = 0;
999.                mcp2515_write_canMsg( txbuf_n);
1000.               //mcp2515_start_transmit( txbuf_n );
1001.               do
1002.               {
1003.                   uiTimeOut++;
1004.                   res1= mcp2515_readRegister(txbuf_n);                            /* read send buff ctrl reg  */

1005.                   res1 = res1 & 0x08;
1006.               }while(res1 && (uiTimeOut < TIMEOUTVALUE));
1007.               if(uiTimeOut == TIMEOUTVALUE)                                       /* send msg timeout
        */
1008.               {
1009.                   return CAN_SENDMSGTIMEOUT;
1010.               }
1011.               return CAN_OK;
1012.
1013.   }
1014.
1015.   /*********************************************************************************************************
1016.    ** Function name:          lowRcvFlag
1017.    ** Descriptions:           LOW RECEIVE FLAGS
1018.   *********************************************************************************************************/
1019.   INT8U MCP_CAN::lowRcvFlag(void)
1020.   {
1021.       mcp2515_modifyRegister(0x2C, 3, 0);
1022.       //mcp2515_modifyRegister(0x2C, 1, 0);
1023.
1024.   }
1025.
1026.   /*********************************************************************************************************
1027.    ** Function name:          DisGloInt
1028.   ** Descriptions:           Disable Global Interrupts
1029.   *********************************************************************************************************/
1030.   INT8U MCP_CAN::DisGloInt(void)
1031.   {
1032.       mcp2515_setRegister(0x2B,0x00 );
1033.
1034.   }
1035.
1036.
1037.
1038.   /*********************************************************************************************************
1039.   ** Function name:          OSMenable
1040.   ** Descriptions:           Enable One-Shot Mode
1041.   *********************************************************************************************************/
1042.   INT8U MCP_CAN::OSMenable(void)
1043.   {
1044.   mcp2515_modifyRegister(0x0F, 0x08, 0x08);
1045.
1046.
1047.   }
1048.
1049.   /*********************************************************************************************************
```

```
1050.   ** Function name:          setRcvInt1
1051.  ** Descriptions:           Enable Interrupts when message arrives on buffer 1 if true
1052.  *********************************************************************************************************/
1053.
1054.  INT8U MCP_CAN::setRcvInt1(bool res)
1055.  {
1056.      if(res == true ){
1057.          mcp2515_modifyRegister(0x2B,0x02,0x02 );
1058.          }
1059.      else {
1060.          mcp2515_modifyRegister(0x2B,0x02,0x00 );
1061.
1062.          }
1063.
1064.  }
1065.
1066.  /*********************************************************************************************************
1067.   ** Function name:          setRcvInt0
1068.  ** Descriptions:           Enable Interrupts when message arrives on buffer 0 if true
1069.  *********************************************************************************************************/
1070.
1071.  INT8U MCP_CAN::setRcvInt0(bool res)
1072.  {
1073.      if(res == true){
1074.          mcp2515_modifyRegister(0x2B,0x01,0x01 );
1075.          }
1076.      else {
1077.          mcp2515_modifyRegister(0x2B,0x01,0x00 );
1078.
1079.          }
1080.
1081.
1082.
1083.  }
1084.
1085.
1086.
1087.  /*********************************************************************************************************
1088.    END FILE
1089.  *********************************************************************************************************/
```

Archivo mcp_can_dfs.h

```
#ifndef _MCP2515DFS_H_
#define _MCP2515DFS_H_

#include <Arduino.h>
#include <SPI.h>
#include <inttypes.h>

#ifndef INT32U
#define INT32U unsigned long
#endif

#ifndef INT8U
#define INT8U byte
#endif

// if print debug information
#define DEBUG_MODE 0

/*
 *   Begin mt
 */
#define TIMEOUTVALUE    50
#define MCP_SIDH        0
#define MCP_SIDL        1
#define MCP_EID8        2
#define MCP_EID0        3

#define MCP_TXB_EXIDE_M     0x08                                    /* In TXBnSIDL                  */
#define MCP_DLC_MASK        0x0F                                    /* 4 LSBits                     */
#define MCP_RTR_MASK        0x40                                    /* (1<<6) Bit 6                 */

#define MCP_RXB_RX_ANY      0x60
#define MCP_RXB_RX_EXT      0x40
#define MCP_RXB_RX_STD      0x20
#define MCP_RXB_RX_STDEXT   0x00
#define MCP_RXB_RX_MASK     0x60
#define MCP_RXB_BUKT_MASK   (1<<2)

/*
** Bits in the TXBnCTRL registers.
*/
#define MCP_TXB_TXBUFE_M    0x80
#define MCP_TXB_ABTF_M      0x40
#define MCP_TXB_MLOA_M      0x20
#define MCP_TXB_TXERR_M     0x10
#define MCP_TXB_TXREQ_M     0x08
#define MCP_TXB_TXIE_M      0x04
#define MCP_TXB_TXP10_M     0x03

#define MCP_TXB_RTR_M       0x40                                    /* In TXBnDLC                   */
#define MCP_RXB_IDE_M       0x08                                    /* In RXBnSIDL                  */
#define MCP_RXB_RTR_M       0x40                                    /* In RXBnDLC                   */

#define MCP_STAT_RXIF_MASK   (0x03)
#define MCP_STAT_RX0IF (1<<0)
#define MCP_STAT_RX1IF (1<<1)

#define MCP_EFLG_RX1OVR (1<<7)
#define MCP_EFLG_RX0OVR (1<<6)
#define MCP_EFLG_TXBO   (1<<5)
#define MCP_EFLG_TXEP   (1<<4)
#define MCP_EFLG_RXEP   (1<<3)
#define MCP_EFLG_TXWAR  (1<<2)
```

```
64.  #define MCP_EFLG_RXWAR  (1<<1)
65.  #define MCP_EFLG_EWARN  (1<<0)
66.  #define MCP_EFLG_ERRORMASK  (0xF8)                                  /* 5 MS-Bits                    */
67.
68.
69.  /*
70.   *   Define MCP2515 register addresses
71.   */
72.
73.  #define MCP_RXF0SIDH    0x00
74.  #define MCP_RXF0SIDL    0x01
75.  #define MCP_RXF0EID8    0x02
76.  #define MCP_RXF0EID0    0x03
77.  #define MCP_RXF1SIDH    0x04
78.  #define MCP_RXF1SIDL    0x05
79.  #define MCP_RXF1EID8    0x06
80.  #define MCP_RXF1EID0    0x07
81.  #define MCP_RXF2SIDH    0x08
82.  #define MCP_RXF2SIDL    0x09
83.  #define MCP_RXF2EID8    0x0A
84.  #define MCP_RXF2EID0    0x0B
85.  #define MCP_CANSTAT     0x0E
86.  #define MCP_CANCTRL     0x0F
87.  #define MCP_RXF3SIDH    0x10
88.  #define MCP_RXF3SIDL    0x11
89.  #define MCP_RXF3EID8    0x12
90.  #define MCP_RXF3EID0    0x13
91.  #define MCP_RXF4SIDH    0x14
92.  #define MCP_RXF4SIDL    0x15
93.  #define MCP_RXF4EID8    0x16
94.  #define MCP_RXF4EID0    0x17
95.  #define MCP_RXF5SIDH    0x18
96.  #define MCP_RXF5SIDL    0x19
97.  #define MCP_RXF5EID8    0x1A
98.  #define MCP_RXF5EID0    0x1B
99.  #define MCP_TEC         0x1C
100. #define MCP_REC         0x1D
101. #define MCP_RXM0SIDH    0x20
102. #define MCP_RXM0SIDL    0x21
103. #define MCP_RXM0EID8    0x22
104. #define MCP_RXM0EID0    0x23
105. #define MCP_RXM1SIDH    0x24
106. #define MCP_RXM1SIDL    0x25
107. #define MCP_RXM1EID8    0x26
108. #define MCP_RXM1EID0    0x27
109. #define MCP_CNF3        0x28
110. #define MCP_CNF2        0x29
111. #define MCP_CNF1        0x2A
112. #define MCP_CANINTE     0x2B
113. #define MCP_CANINTF     0x2C
114. #define MCP_EFLG        0x2D
115. #define MCP_TXB0CTRL    0x30
116. #define MCP_TXB1CTRL    0x40
117. #define MCP_TXB2CTRL    0x50
118. #define MCP_RXB0CTRL    0x60
119. #define MCP_RXB0SIDH    0x61
120. #define MCP_RXB1CTRL    0x70
121. #define MCP_RXB1SIDH    0x71
122.
123.
124. #define MCP_TX_INT          0x1C                                    // Enable all transmit interrup ts
125. #define MCP_TX01_INT        0x0C                                    // Enable TXB0 and TXB1 interru pts
126. #define MCP_RX_INT          0x03                                    // Enable receive interrupts
127. #define MCP_NO_INT          0x00                                    // Disable all interrupts
128.
129. #define MCP_TX01_MASK       0x14
```

```
130. #define MCP_TX_MASK         0x54
131.
132. /*
133.  *   Define SPI Instruction Set
134.  */
135.
136. #define MCP_WRITE           0x02
137.
138. #define MCP_READ            0x03
139.
140. #define MCP_BITMOD          0x05
141.
142. #define MCP_LOAD_TX0        0x40
143. #define MCP_LOAD_TX1        0x42
144. #define MCP_LOAD_TX2        0x44
145.
146. #define MCP_RTS_TX0         0x81
147. #define MCP_RTS_TX1         0x82
148. #define MCP_RTS_TX2         0x84
149. #define MCP_RTS_ALL         0x87
150.
151. #define MCP_READ_RX0        0x90
152. #define MCP_READ_RX1        0x94
153.
154. #define MCP_READ_STATUS     0xA0
155.
156. #define MCP_RX_STATUS       0xB0
157.
158. #define MCP_RESET           0xC0
159.
160.
161. /*
162.  *   CANCTRL Register Values
163.  */
164.
165. #define MODE_NORMAL     0x00
166. #define MODE_SLEEP      0x20
167. #define MODE_LOOPBACK   0x40
168. #define MODE_LISTENONLY 0x60
169. #define MODE_CONFIG     0x80
170. #define MODE_POWERUP    0xE0
171. #define MODE_MASK       0xE0
172. #define ABORT_TX        0x10
173. #define MODE_ONESHOT    0x08
174. #define CLKOUT_ENABLE   0x04
175. #define CLKOUT_DISABLE  0x00
176. #define CLKOUT_PS1      0x00
177. #define CLKOUT_PS2      0x01
178. #define CLKOUT_PS4      0x02
179. #define CLKOUT_PS8      0x03
180.
181.
182. /*
183.  *   CNF1 Register Values
184.  */
185.
186. #define SJW1            0x00
187. #define SJW2            0x40
188. #define SJW3            0x80
189. #define SJW4            0xC0
190.
191.
192. /*
193.  *   CNF2 Register Values
194.  */
195.
```

```
196. #define BTLMODE         0x80
197. #define SAMPLE_1X       0x00
198. #define SAMPLE_3X       0x40
199.
200.
201. /*
202.  *   CNF3 Register Values
203.  */
204.
205. #define SOF_ENABLE      0x80
206. #define SOF_DISABLE     0x00
207. #define WAKFIL_ENABLE   0x40
208. #define WAKFIL_DISABLE  0x00
209.
210.
211. /*
212.  *   CANINTF Register Bits
213.  */
214.
215. #define MCP_RX0IF       0x01
216. #define MCP_RX1IF       0x02
217. #define MCP_TX0IF       0x04
218. #define MCP_TX1IF       0x08
219. #define MCP_TX2IF       0x10
220. #define MCP_ERRIF       0x20
221. #define MCP_WAKIF       0x40
222. #define MCP_MERRF       0x80
223.
224. /*
225.  *  speed 16M
226.  */
227. #define MCP_16MHz_1000kBPS_CFG1 (0x00)
228. #define MCP_16MHz_1000kBPS_CFG2 (0xD0)
229. #define MCP_16MHz_1000kBPS_CFG3 (0x82)
230.
231. #define MCP_16MHz_500kBPS_CFG1 (0x00)
232. #define MCP_16MHz_500kBPS_CFG2 (0xF0)
233. #define MCP_16MHz_500kBPS_CFG3 (0x86)
234.
235. #define MCP_16MHz_250kBPS_CFG1 (0x41)
236. #define MCP_16MHz_250kBPS_CFG2 (0xF1)
237. #define MCP_16MHz_250kBPS_CFG3 (0x85)
238.
239. #define MCP_16MHz_200kBPS_CFG1 (0x01)
240. #define MCP_16MHz_200kBPS_CFG2 (0xFA)
241. #define MCP_16MHz_200kBPS_CFG3 (0x87)
242.
243. #define MCP_16MHz_125kBPS_CFG1 (0x03)
244. #define MCP_16MHz_125kBPS_CFG2 (0xF0)
245. #define MCP_16MHz_125kBPS_CFG3 (0x86)
246.
247. #define MCP_16MHz_100kBPS_CFG1 (0x03)
248. #define MCP_16MHz_100kBPS_CFG2 (0xFA)
249. #define MCP_16MHz_100kBPS_CFG3 (0x87)
250.
251. /*
252. #define MCP_16MHz_100kBPS_CFG1 (0x03)
253. #define MCP_16MHz_100kBPS_CFG2 (0xBA)
254. #define MCP_16MHz_100kBPS_CFG3 (0x07)
255. */
256.
257. #define MCP_16MHz_80kBPS_CFG1 (0x03)
258. #define MCP_16MHz_80kBPS_CFG2 (0xFF)
259. #define MCP_16MHz_80kBPS_CFG3 (0x87)
260.
261. #define MCP_16MHz_50kBPS_CFG1 (0x07)
```

```
#define MCP_16MHz_50kBPS_CFG2 (0xFA)
#define MCP_16MHz_50kBPS_CFG3 (0x87)

#define MCP_16MHz_40kBPS_CFG1 (0x07)
#define MCP_16MHz_40kBPS_CFG2 (0xFF)
#define MCP_16MHz_40kBPS_CFG3 (0x87)

#define MCP_16MHz_31k25BPS_CFG1 (0x0F)
#define MCP_16MHz_31k25BPS_CFG2 (0xF1)
#define MCP_16MHz_31k25BPS_CFG3 (0x85)

#define MCP_16MHz_20kBPS_CFG1 (0x0F)
#define MCP_16MHz_20kBPS_CFG2 (0xFF)
#define MCP_16MHz_20kBPS_CFG3 (0x87)

#define MCP_16MHz_10kBPS_CFG1 (0x1F)
#define MCP_16MHz_10kBPS_CFG2 (0xFF)
#define MCP_16MHz_10kBPS_CFG3 (0x87)

#define MCP_16MHz_5kBPS_CFG1 (0x3F)
#define MCP_16MHz_5kBPS_CFG2 (0xFF)
#define MCP_16MHz_5kBPS_CFG3 (0x87)


#define MCPDEBUG        (0)
#define MCPDEBUG_TXBUF  (0)
#define MCP_N_TXBUFFERS (3)

#define MCP_RXBUF_0 (MCP_RXB0SIDH)
#define MCP_RXBUF_1 (MCP_RXB1SIDH)

//#define SPICS 10
#define MCP2515_SELECT()   digitalWrite(SPICS, LOW)
#define MCP2515_UNSELECT() digitalWrite(SPICS, HIGH)

#define MCP2515_OK         (0)
#define MCP2515_FAIL       (1)
#define MCP_ALLTXBUSY      (2)

#define CANDEBUG   1

#define CANUSELOOP 0

#define CANSENDTIMEOUT (200)                                        /* milliseconds                */

/*
 *   initial value of gCANAutoProcess
 */
#define CANAUTOPROCESS (1)
#define CANAUTOON  (1)
#define CANAUTOOFF (0)

#define CAN_STDID (0)
#define CAN_EXTID (1)

#define CANDEFAULTIDENT    (0x55CC)
#define CANDEFAULTIDENTEXT (CAN_EXTID)

#define CAN_5KBPS    1
#define CAN_10KBPS   2
#define CAN_20KBPS   3
#define CAN_31K25BPS 4
#define CAN_40KBPS   5
#define CAN_50KBPS   6
#define CAN_80KBPS   7
```

```
328.  #define CAN_100KBPS  8
329.  #define CAN_125KBPS  9
330.  #define CAN_200KBPS  10
331.  #define CAN_250KBPS  11
332.  #define CAN_500KBPS  12
333.  #define CAN_1000KBPS 13
334.
335.  #define CAN_OK                (0)
336.  #define CAN_FAILINIT          (1)
337.  #define CAN_FAILTX            (2)
338.  #define CAN_MSGAVAIL          (3)
339.  #define CAN_NOMSG             (4)
340.  #define CAN_CTRLERROR         (5)
341.  #define CAN_GETTXBFTIMEOUT    (6)
342.  #define CAN_SENDMSGTIMEOUT    (7)
343.  #define CAN_FAIL              (0xff)
344.
345.  #define CAN_MAX_CHAR_IN_MESSAGE (8)
346.
347.  #endif
348.  /*********************************************************************************************************
349.    END FILE
350.  *********************************************************************************************************/
```

Librería Display LCD de Adafruit

Archivo Adafruit_MCP23017.h

```
/*************************************************
  This is a library for the MCP23017 i2c port expander

  These displays use I2C to communicate, 2 pins are required to
  interface
  Adafruit invests time and resources providing this open source code,
  please support Adafruit and open-source hardware by purchasing
  products from Adafruit!

  Written by Limor Fried/Ladyada for Adafruit Industries.
  BSD license, all text above must be included in any redistribution
 *************************************************/

#ifndef _Adafruit_MCP23017_H_
#define _Adafruit_MCP23017_H_

// Don't forget the Wire library
class Adafruit_MCP23017 {
public:
  void begin(uint8_t addr);
  void begin(void);

  void pinMode(uint8_t p, uint8_t d);
  void digitalWrite(uint8_t p, uint8_t d);
  void pullUp(uint8_t p, uint8_t d);
  uint8_t digitalRead(uint8_t p);
  void writeGPIOAB(uint16_t);
  uint16_t readGPIOAB();

 private:
  uint8_t i2caddr;
};

#define MCP23017_ADDRESS 0x20

// registers
#define MCP23017_IODIRA 0x00
#define MCP23017_IPOLA 0x02
#define MCP23017_GPINTENA 0x04
#define MCP23017_DEFVALA 0x06
#define MCP23017_INTCONA 0x08
#define MCP23017_IOCONA 0x0A
#define MCP23017_GPPUA 0x0C
#define MCP23017_INTFA 0x0E
#define MCP23017_INTCAPA 0x10
#define MCP23017_GPIOA 0x12
#define MCP23017_OLATA 0x14

#define MCP23017_IODIRB 0x01
#define MCP23017_IPOLB 0x03
#define MCP23017_GPINTENB 0x05
#define MCP23017_DEFVALB 0x07
#define MCP23017_INTCONB 0x09
#define MCP23017_IOCONB 0x0B
#define MCP23017_GPPUB 0x0D
#define MCP23017_INTFB 0x0F
#define MCP23017_INTCAPB 0x11
#define MCP23017_GPIOB 0x13
#define MCP23017_OLATB 0x15
#endif
```

Archivo Adafruit_MCP23017.cpp

```
1.  /***************************************************
2.    This is a library for the MCP23017 i2c port expander
3.
4.    These displays use I2C to communicate, 2 pins are required to
5.    interface
6.    Adafruit invests time and resources providing this open source code,
7.    please support Adafruit and open-source hardware by purchasing
8.    products from Adafruit!
9.
10.   Written by Limor Fried/Ladyada for Adafruit Industries.
11.   BSD license, all text above must be included in any redistribution
12.  ****************************************************/
13.
14. #include <Wire.h>
15. #include <avr/pgmspace.h>
16. #include "Adafruit_MCP23017.h"
17. #ifdef __AVR__
18.  #define WIRE Wire
19. #else // Arduino Due
20.  #define WIRE Wire1
21. #endif
22.
23. #if ARDUINO >= 100
24.  #include "Arduino.h"
25. #else
26.  #include "WProgram.h"
27. #endif
28.
29. // minihelper
30. static inline void wiresend(uint8_t x) {
31. #if ARDUINO >= 100
32.   WIRE.write((uint8_t)x);
33. #else
34.   WIRE.send(x);
35. #endif
36. }
37.
38. static inline uint8_t wirerecv(void) {
39. #if ARDUINO >= 100
40.   return WIRE.read();
41. #else
42.   return WIRE.receive();
43. #endif
44. }
45.
46. ////////////////////////////////////////////////////////////////////////////////
47.
48. void Adafruit_MCP23017::begin(uint8_t addr) {
49.   if (addr > 7) {
50.     addr = 7;
51.   }
52.   i2caddr = addr;
53.
54.   WIRE.begin();
55.
56.
57.   // set defaults!
58.   WIRE.beginTransmission(MCP23017_ADDRESS | i2caddr);
59.   wiresend(MCP23017_IODIRA);
60.   wiresend(0xFF);  // all inputs on port A
61.   WIRE.endTransmission();
62.
63.   WIRE.beginTransmission(MCP23017_ADDRESS | i2caddr);
```

```
64.    wiresend(MCP23017_IODIRB);
65.    wiresend(0xFF);  // all inputs on port B
66.    WIRE.endTransmission();
67.  }
68.
69.
70.  void Adafruit_MCP23017::begin(void) {
71.    begin(0);
72.  }
73.
74.  void Adafruit_MCP23017::pinMode(uint8_t p, uint8_t d) {
75.    uint8_t iodir;
76.    uint8_t iodiraddr;
77.
78.    // only 16 bits!
79.    if (p > 15)
80.      return;
81.
82.    if (p < 8)
83.      iodiraddr = MCP23017_IODIRA;
84.    else {
85.      iodiraddr = MCP23017_IODIRB;
86.      p -= 8;
87.    }
88.
89.    // read the current IODIR
90.    WIRE.beginTransmission(MCP23017_ADDRESS | i2caddr);
91.    wiresend(iodiraddr);
92.    WIRE.endTransmission();
93.
94.    WIRE.requestFrom(MCP23017_ADDRESS | i2caddr, 1);
95.    iodir = wirerecv();
96.
97.    // set the pin and direction
98.    if (d == INPUT) {
99.      iodir |= 1 << p;
100.   } else {
101.     iodir &= ~(1 << p);
102.   }
103.
104.   // write the new IODIR
105.   WIRE.beginTransmission(MCP23017_ADDRESS | i2caddr);
106.   wiresend(iodiraddr);
107.   wiresend(iodir);
108.   WIRE.endTransmission();
109. }
110.
111. uint16_t Adafruit_MCP23017::readGPIOAB() {
112.   uint16_t ba = 0;
113.   uint8_t a;
114.
115.   // read the current GPIO output latches
116.   WIRE.beginTransmission(MCP23017_ADDRESS | i2caddr);
117.   wiresend(MCP23017_GPIOA);
118.   WIRE.endTransmission();
119.
120.   WIRE.requestFrom(MCP23017_ADDRESS | i2caddr, 2);
121.   a = wirerecv();
122.   ba = wirerecv();
123.   ba <<= 8;
124.   ba |= a;
125.
126.   return ba;
127. }
128.
```

```
129. void Adafruit_MCP23017::writeGPIOAB(uint16_t ba) {
130.   WIRE.beginTransmission(MCP23017_ADDRESS | i2caddr);
131.   wiresend(MCP23017_GPIOA);
132.   wiresend(ba & 0xFF);
133.   wiresend(ba >> 8);
134.   WIRE.endTransmission();
135. }
136.
137. void Adafruit_MCP23017::digitalWrite(uint8_t p, uint8_t d) {
138.   uint8_t gpio;
139.   uint8_t gpioaddr, olataddr;
140.
141.   // only 16 bits!
142.   if (p > 15)
143.     return;
144.
145.   if (p < 8) {
146.     olataddr = MCP23017_OLATA;
147.     gpioaddr = MCP23017_GPIOA;
148.   } else {
149.     olataddr = MCP23017_OLATB;
150.     gpioaddr = MCP23017_GPIOB;
151.     p -= 8;
152.   }
153.
154.   // read the current GPIO output latches
155.   WIRE.beginTransmission(MCP23017_ADDRESS | i2caddr);
156.   wiresend(olataddr);
157.   WIRE.endTransmission();
158.
159.   WIRE.requestFrom(MCP23017_ADDRESS | i2caddr, 1);
160.    gpio = wirerecv();
161.
162.   // set the pin and direction
163.   if (d == HIGH) {
164.     gpio |= 1 << p;
165.   } else {
166.     gpio &= ~(1 << p);
167.   }
168.
169.   // write the new GPIO
170.   WIRE.beginTransmission(MCP23017_ADDRESS | i2caddr);
171.   wiresend(gpioaddr);
172.   wiresend(gpio);
173.   WIRE.endTransmission();
174. }
175.
176. void Adafruit_MCP23017::pullUp(uint8_t p, uint8_t d) {
177.   uint8_t gppu;
178.   uint8_t gppuaddr;
179.
180.   // only 16 bits!
181.   if (p > 15)
182.     return;
183.
184.   if (p < 8)
185.     gppuaddr = MCP23017_GPPUA;
186.   else {
187.     gppuaddr = MCP23017_GPPUB;
188.     p -= 8;
189.   }
190.
191.
192.   // read the current pullup resistor set
193.   WIRE.beginTransmission(MCP23017_ADDRESS | i2caddr);
```

```
194.   wiresend(gppuaddr);
195.   WIRE.endTransmission();
196.
197.   WIRE.requestFrom(MCP23017_ADDRESS | i2caddr, 1);
198.   gppu = wirerecv();
199.
200.   // set the pin and direction
201.   if (d == HIGH) {
202.     gppu |= 1 << p;
203.   } else {
204.     gppu &= ~(1 << p);
205.   }
206.
207.   // write the new GPIO
208.   WIRE.beginTransmission(MCP23017_ADDRESS | i2caddr);
209.   wiresend(gppuaddr);
210.   wiresend(gppu);
211.   WIRE.endTransmission();
212. }
213.
214. uint8_t Adafruit_MCP23017::digitalRead(uint8_t p) {
215.   uint8_t gpioaddr;
216.
217.   // only 16 bits!
218.   if (p > 15)
219.     return 0;
220.
221.   if (p < 8)
222.     gpioaddr = MCP23017_GPIOA;
223.   else {
224.     gpioaddr = MCP23017_GPIOB;
225.     p -= 8;
226.   }
227.
228.   // read the current GPIO
229.   WIRE.beginTransmission(MCP23017_ADDRESS | i2caddr);
230.   wiresend(gpioaddr);
231.   WIRE.endTransmission();
232.
233.   WIRE.requestFrom(MCP23017_ADDRESS | i2caddr, 1);
234.   return (wirerecv() >> p) & 0x1;
235. }
```

Archivo Adafruit_RGBLCDShield.h

```
/*************************************************
  This is a library for the Adafruit RGB 16x2 LCD Shield
  Pick one up at the Adafruit shop!
  ---------> http://http://www.adafruit.com/products/714

  The shield uses I2C to communicate, 2 pins are required to
  interface
  Adafruit invests time and resources providing this open source code,
  please support Adafruit and open-source hardware by purchasing
  products from Adafruit!

  Written by Limor Fried/Ladyada for Adafruit Industries.
  BSD license, all text above must be included in any redistribution
 *************************************************/

#ifndef Adafruit_RGBLCDShield_h
#define Adafruit_RGBLCDShield_h

#include <inttypes.h>
#include "Print.h"
#include <Adafruit_MCP23017.h>

// commands
#define LCD_CLEARDISPLAY 0x01
#define LCD_RETURNHOME 0x02
#define LCD_ENTRYMODESET 0x04
#define LCD_DISPLAYCONTROL 0x08
#define LCD_CURSORSHIFT 0x10
#define LCD_FUNCTIONSET 0x20
#define LCD_SETCGRAMADDR 0x40
#define LCD_SETDDRAMADDR 0x80

// flags for display entry mode
#define LCD_ENTRYRIGHT 0x00
#define LCD_ENTRYLEFT 0x02
#define LCD_ENTRYSHIFTINCREMENT 0x01
#define LCD_ENTRYSHIFTDECREMENT 0x00

// flags for display on/off control
#define LCD_DISPLAYON 0x04
#define LCD_DISPLAYOFF 0x00
#define LCD_CURSORON 0x02
#define LCD_CURSOROFF 0x00
#define LCD_BLINKON 0x01
#define LCD_BLINKOFF 0x00

// flags for display/cursor shift
#define LCD_DISPLAYMOVE 0x08
#define LCD_CURSORMOVE 0x00
#define LCD_MOVERIGHT 0x04
#define LCD_MOVELEFT 0x00

// flags for function set
#define LCD_8BITMODE 0x10
#define LCD_4BITMODE 0x00
#define LCD_2LINE 0x08
#define LCD_1LINE 0x00
#define LCD_5x10DOTS 0x04
#define LCD_5x8DOTS 0x00

#define BUTTON_UP 0x08
#define BUTTON_DOWN 0x04
#define BUTTON_LEFT 0x10
```

```
64. #define BUTTON_RIGHT 0x02
65. #define BUTTON_SELECT 0x01
66. 
67. class Adafruit_RGBLCDShield : public Print {
68. public:
69.   Adafruit_RGBLCDShield();
70. 
71.   void init(uint8_t fourbitmode, uint8_t rs, uint8_t rw, uint8_t enable,
72.         uint8_t d0, uint8_t d1, uint8_t d2, uint8_t d3,
73.         uint8_t d4, uint8_t d5, uint8_t d6, uint8_t d7);
74. 
75.   void begin(uint8_t cols, uint8_t rows, uint8_t charsize = LCD_5x8DOTS);
76.   void clear();
77.   void home();
78.   void noDisplay();
79.   void display();
80.   void noBlink();
81.   void blink();
82.   void noCursor();
83.   void cursor();
84.   void scrollDisplayLeft();
85.   void scrollDisplayRight();
86.   void leftToRight();
87.   void rightToLeft();
88.   void autoscroll();
89.   void noAutoscroll();
90. 
91.   // only if using backpack
92.   void setBacklight(uint8_t status);
93. 
94.   void createChar(uint8_t, uint8_t[]);
95.   void setCursor(uint8_t, uint8_t);
96. #if ARDUINO >= 100
97.   virtual size_t write(uint8_t);
98. #else
99.   virtual void write(uint8_t);
100.      #endif
101.        void command(uint8_t);
102.        uint8_t readButtons();
103. 
104.      private:
105.        void send(uint8_t, uint8_t);
106.        void write4bits(uint8_t);
107.        void write8bits(uint8_t);
108.        void pulseEnable();
109.        void _digitalWrite(uint8_t, uint8_t);
110.        void _pinMode(uint8_t, uint8_t);
111. 
112.        uint8_t _rs_pin; // LOW: command.  HIGH: character.
113.        uint8_t _rw_pin; // LOW: write to LCD.  HIGH: read from LCD.
114.        uint8_t _enable_pin; // activated by a HIGH pulse.
115.        uint8_t _data_pins[8];
116.        uint8_t _button_pins[5];
117.        uint8_t _displayfunction;
118.        uint8_t _displaycontrol;
119.        uint8_t _displaymode;
120. 
121.        uint8_t _initialized;
122. 
123.        uint8_t _numlines,_currline;
124. 
125.        uint8_t _i2cAddr;
126.        Adafruit_MCP23017 _i2c;
127.      };
128.       #endif
```

Archivo Adafruit_RGBLCDShield.cpp

```
1.  /*************************************************
2.    This is a library for the Adafruit RGB 16x2 LCD Shield
3.    Pick one up at the Adafruit shop!
4.    ---------> http://http://www.adafruit.com/products/714
5.
6.    The shield uses I2C to communicate, 2 pins are required to
7.    interface
8.    Adafruit invests time and resources providing this open source code,
9.    please support Adafruit and open-source hardware by purchasing
10.   products from Adafruit!
11.
12.   Written by Limor Fried/Ladyada for Adafruit Industries.
13.   BSD license, all text above must be included in any redistribution
14.  *************************************************/
15.
16.
17. #include "Adafruit_RGBLCDShield.h"
18.
19. #include <stdio.h>
20. #include <string.h>
21. #include <inttypes.h>
22. #include <Wire.h>
23. #ifdef __AVR__
24.  #define WIRE Wire
25. #else // Arduino Due
26.  #define WIRE Wire1
27. #endif
28.
29. #if ARDUINO >= 100
30.  #include "Arduino.h"
31. #else
32.  #include "WProgram.h"
33. #endif
34.
35. // When the display powers up, it is configured as follows:
36. //
37. // 1. Display clear
38. // 2. Function set:
39. //    DL = 1; 8-bit interface data
40. //    N = 0; 1-line display
41. //    F = 0; 5x8 dot character font
42. // 3. Display on/off control:
43. //    D = 0; Display off
44. //    C = 0; Cursor off
45. //    B = 0; Blinking off
46. // 4. Entry mode set:
47. //    I/D = 1; Increment by 1
48. //    S = 0; No shift
49. //
50. // Note, however, that resetting the Arduino doesn't reset the LCD, so we
51. // can't assume that its in that state when a sketch starts (and the
52. // RGBLCDShield constructor is called).
53.
54. Adafruit_RGBLCDShield::Adafruit_RGBLCDShield() {
55.   _i2cAddr = 0;
56.
57.   _displayfunction = LCD_4BITMODE | LCD_1LINE | LCD_5x8DOTS;
58.
59.   // the I/O expander pinout
60.   _rs_pin = 15;
61.   _rw_pin = 14;
62.   _enable_pin = 13;
63.   _data_pins[0] = 12;  // really d4
```

```
64.   _data_pins[1] = 11;  // really d5
65.   _data_pins[2] = 10;  // really d6
66.   _data_pins[3] = 9;  // really d7
67.
68.   _button_pins[0] = 0;
69.   _button_pins[1] = 1;
70.   _button_pins[2] = 2;
71.   _button_pins[3] = 3;
72.   _button_pins[4] = 4;
73.   // we can't begin() yet :(
74. }
75.
76.
77.
78.
79. void Adafruit_RGBLCDShield::init(uint8_t fourbitmode, uint8_t rs, uint8_t rw, uint8_
    t enable,
80.              uint8_t d0, uint8_t d1, uint8_t d2, uint8_t d3,
81.              uint8_t d4, uint8_t d5, uint8_t d6, uint8_t d7)
82. {
83.   _rs_pin = rs;
84.   _rw_pin = rw;
85.   _enable_pin = enable;
86.
87.   _data_pins[0] = d0;
88.   _data_pins[1] = d1;
89.   _data_pins[2] = d2;
90.   _data_pins[3] = d3;
91.   _data_pins[4] = d4;
92.   _data_pins[5] = d5;
93.   _data_pins[6] = d6;
94.   _data_pins[7] = d7;
95.
96.   _i2cAddr = 255;
97.
98.   _pinMode(_rs_pin, OUTPUT);
99.   // we can save 1 pin by not using RW. Indicate by passing 255 instead of pin#
100.         if (_rw_pin != 255) {
101.           _pinMode(_rw_pin, OUTPUT);
102.         }
103.         _pinMode(_enable_pin, OUTPUT);
104.
105.
106.         if (fourbitmode)
107.           _displayfunction = LCD_4BITMODE | LCD_1LINE | LCD_5x8DOTS;
108.         else
109.           _displayfunction = LCD_8BITMODE | LCD_1LINE | LCD_5x8DOTS;
110.
111.         begin(16, 1);
112.       }
113.
114.       void Adafruit_RGBLCDShield::begin(uint8_t cols, uint8_t lines, uint8_t dotsiz
    e) {
115.         // check if i2c
116.         if (_i2cAddr != 255) {
117.           //_i2c.begin(_i2cAddr);
118.           WIRE.begin();
119.           _i2c.begin();
120.
121.           _i2c.pinMode(8, OUTPUT);
122.           _i2c.pinMode(6, OUTPUT);
123.           _i2c.pinMode(7, OUTPUT);
124.           setBacklight(0x7);
125.
126.           if (_rw_pin)
```

```
127.             _i2c.pinMode(_rw_pin, OUTPUT);
128.
129.           _i2c.pinMode(_rs_pin, OUTPUT);
130.           _i2c.pinMode(_enable_pin, OUTPUT);
131.           for (uint8_t i=0; i<4; i++)
132.             _i2c.pinMode(_data_pins[i], OUTPUT);
133.
134.           for (uint8_t i=0; i<5; i++) {
135.             _i2c.pinMode(_button_pins[i], INPUT);
136.             _i2c.pullUp(_button_pins[i], 1);
137.           }
138.         }
139.
140.         if (lines > 1) {
141.           _displayfunction |= LCD_2LINE;
142.         }
143.         _numlines = lines;
144.         _currline = 0;
145.
146.         // for some 1 line displays you can select a 10 pixel high font
147.         if ((dotsize != 0) && (lines == 1)) {
148.           _displayfunction |= LCD_5x10DOTS;
149.         }
150.
151.         // SEE PAGE 45/46 FOR INITIALIZATION SPECIFICATION!
152.         // according to datasheet, we need at least 40ms after power rises above 2.
    7V
153.         // before sending commands. Arduino can turn on way befer 4.5V so we'll wai
    t 50
154.         delayMicroseconds(50000);
155.         // Now we pull both RS and R/W low to begin commands
156.         _digitalWrite(_rs_pin, LOW);
157.         _digitalWrite(_enable_pin, LOW);
158.         if (_rw_pin != 255) {
159.           _digitalWrite(_rw_pin, LOW);
160.         }
161.
162.         //put the LCD into 4 bit or 8 bit mode
163.         if (! (_displayfunction & LCD_8BITMODE)) {
164.           // this is according to the hitachi HD44780 datasheet
165.           // figure 24, pg 46
166.
167.           // we start in 8bit mode, try to set 4 bit mode
168.           write4bits(0x03);
169.           delayMicroseconds(4500); // wait min 4.1ms
170.
171.           // second try
172.           write4bits(0x03);
173.           delayMicroseconds(4500); // wait min 4.1ms
174.
175.           // third go!
176.           write4bits(0x03);
177.           delayMicroseconds(150);
178.
179.           // finally, set to 8-bit interface
180.           write4bits(0x02);
181.         } else {
182.           // this is according to the hitachi HD44780 datasheet
183.           // page 45 figure 23
184.
185.           // Send function set command sequence
186.           command(LCD_FUNCTIONSET | _displayfunction);
187.           delayMicroseconds(4500);  // wait more than 4.1ms
188.
189.           // second try
```

```
190.           command(LCD_FUNCTIONSET | _displayfunction);
191.           delayMicroseconds(150);
192.
193.           // third go
194.           command(LCD_FUNCTIONSET | _displayfunction);
195.         }
196.
197.         // finally, set # lines, font size, etc.
198.         command(LCD_FUNCTIONSET | _displayfunction);
199.
200.         // turn the display on with no cursor or blinking default
201.         _displaycontrol = LCD_DISPLAYON | LCD_CURSOROFF | LCD_BLINKOFF;
202.         display();
203.
204.         // clear it off
205.         clear();
206.
207.         // Initialize to default text direction (for romance languages)
208.         _displaymode = LCD_ENTRYLEFT | LCD_ENTRYSHIFTDECREMENT;
209.         // set the entry mode
210.         command(LCD_ENTRYMODESET | _displaymode);
211.
212.       }
213.
214.       /********** high level commands, for the user! */
215.       void Adafruit_RGBLCDShield::clear()
216.       {
217.         command(LCD_CLEARDISPLAY);  // clear display, set cursor position to zero
218.         delayMicroseconds(2000);  // this command takes a long time!
219.       }
220.
221.       void Adafruit_RGBLCDShield::home()
222.       {
223.         command(LCD_RETURNHOME);  // set cursor position to zero
224.         delayMicroseconds(2000);  // this command takes a long time!
225.       }
226.
227.       void Adafruit_RGBLCDShield::setCursor(uint8_t col, uint8_t row)
228.       {
229.         int row_offsets[] = { 0x00, 0x40, 0x14, 0x54 };
230.         if ( row > _numlines ) {
231.           row = _numlines-1;    // we count rows starting w/0
232.         }
233.
234.         command(LCD_SETDDRAMADDR | (col + row_offsets[row]));
235.       }
236.
237.       // Turn the display on/off (quickly)
238.       void Adafruit_RGBLCDShield::noDisplay() {
239.         _displaycontrol &= ~LCD_DISPLAYON;
240.         command(LCD_DISPLAYCONTROL | _displaycontrol);
241.       }
242.       void Adafruit_RGBLCDShield::display() {
243.         _displaycontrol |= LCD_DISPLAYON;
244.         command(LCD_DISPLAYCONTROL | _displaycontrol);
245.       }
246.
247.       // Turns the underline cursor on/off
248.       void Adafruit_RGBLCDShield::noCursor() {
249.         _displaycontrol &= ~LCD_CURSORON;
250.         command(LCD_DISPLAYCONTROL | _displaycontrol);
251.       }
252.       void Adafruit_RGBLCDShield::cursor() {
253.         _displaycontrol |= LCD_CURSORON;
254.         command(LCD_DISPLAYCONTROL | _displaycontrol);
```

```
255.      }
256.
257.      // Turn on and off the blinking cursor
258.      void Adafruit_RGBLCDShield::noBlink() {
259.        _displaycontrol &= ~LCD_BLINKON;
260.        command(LCD_DISPLAYCONTROL | _displaycontrol);
261.      }
262.      void Adafruit_RGBLCDShield::blink() {
263.        _displaycontrol |= LCD_BLINKON;
264.        command(LCD_DISPLAYCONTROL | _displaycontrol);
265.      }
266.
267.      // These commands scroll the display without changing the RAM
268.      void Adafruit_RGBLCDShield::scrollDisplayLeft(void) {
269.        command(LCD_CURSORSHIFT | LCD_DISPLAYMOVE | LCD_MOVELEFT);
270.      }
271.      void Adafruit_RGBLCDShield::scrollDisplayRight(void) {
272.        command(LCD_CURSORSHIFT | LCD_DISPLAYMOVE | LCD_MOVERIGHT);
273.      }
274.
275.      // This is for text that flows Left to Right
276.      void Adafruit_RGBLCDShield::leftToRight(void) {
277.        _displaymode |= LCD_ENTRYLEFT;
278.        command(LCD_ENTRYMODESET | _displaymode);
279.      }
280.
281.      // This is for text that flows Right to Left
282.      void Adafruit_RGBLCDShield::rightToLeft(void) {
283.        _displaymode &= ~LCD_ENTRYLEFT;
284.        command(LCD_ENTRYMODESET | _displaymode);
285.      }
286.
287.      // This will 'right justify' text from the cursor
288.      void Adafruit_RGBLCDShield::autoscroll(void) {
289.        _displaymode |= LCD_ENTRYSHIFTINCREMENT;
290.        command(LCD_ENTRYMODESET | _displaymode);
291.      }
292.
293.      // This will 'left justify' text from the cursor
294.      void Adafruit_RGBLCDShield::noAutoscroll(void) {
295.        _displaymode &= ~LCD_ENTRYSHIFTINCREMENT;
296.        command(LCD_ENTRYMODESET | _displaymode);
297.      }
298.
299.      // Allows us to fill the first 8 CGRAM locations
300.      // with custom characters
301.      void Adafruit_RGBLCDShield::createChar(uint8_t location, uint8_t charmap[]) {

302.        location &= 0x7; // we only have 8 locations 0-7
303.        command(LCD_SETCGRAMADDR | (location << 3));
304.        for (int i=0; i<8; i++) {
305.          write(charmap[i]);
306.        }
307.        command(LCD_SETDDRAMADDR);  // unfortunately resets the location to 0,0
308.      }
309.
310.      /*********** mid level commands, for sending data/cmds */
311.
312.      inline void Adafruit_RGBLCDShield::command(uint8_t value) {
313.        send(value, LOW);
314.      }
315.
316.      #if ARDUINO >= 100
317.      inline size_t Adafruit_RGBLCDShield::write(uint8_t value) {
318.        send(value, HIGH);
```

```
319.          return 1;
320.        }
321.        #else
322.        inline void Adafruit_RGBLCDShield::write(uint8_t value) {
323.          send(value, HIGH);
324.        }
325.        #endif
326.
327.        /************ low level data pushing commands **********/
328.
329.        // little wrapper for i/o writes
330.        void  Adafruit_RGBLCDShield::_digitalWrite(uint8_t p, uint8_t d) {
331.          if (_i2cAddr != 255) {
332.            // an i2c command
333.            _i2c.digitalWrite(p, d);
334.          } else {
335.            // straightup IO
336.            digitalWrite(p, d);
337.          }
338.        }
339.
340.        // Allows to set the backlight, if the LCD backpack is used
341.        void Adafruit_RGBLCDShield::setBacklight(uint8_t status) {
342.          // check if i2c or SPI
343.          _i2c.digitalWrite(8, ~(status >> 2) & 0x1);
344.          _i2c.digitalWrite(7, ~(status >> 1) & 0x1);
345.          _i2c.digitalWrite(6, ~status & 0x1);
346.        }
347.
348.        // little wrapper for i/o directions
349.        void  Adafruit_RGBLCDShield::_pinMode(uint8_t p, uint8_t d) {
350.          if (_i2cAddr != 255) {
351.            // an i2c command
352.            _i2c.pinMode(p, d);
353.          } else {
354.            // straightup IO
355.            pinMode(p, d);
356.          }
357.        }
358.
359.        // write either command or data, with automatic 4/8-bit selection
360.        void Adafruit_RGBLCDShield::send(uint8_t value, uint8_t mode) {
361.          _digitalWrite(_rs_pin, mode);
362.
363.          // if there is a RW pin indicated, set it low to Write
364.          if (_rw_pin != 255) {
365.            _digitalWrite(_rw_pin, LOW);
366.          }
367.
368.          if (_displayfunction & LCD_8BITMODE) {
369.            write8bits(value);
370.          } else {
371.            write4bits(value>>4);
372.            write4bits(value);
373.          }
374.        }
375.
376.        void Adafruit_RGBLCDShield::pulseEnable(void) {
377.          _digitalWrite(_enable_pin, LOW);
378.          delayMicroseconds(1);
379.          _digitalWrite(_enable_pin, HIGH);
380.          delayMicroseconds(1);    // enable pulse must be >450ns
381.          _digitalWrite(_enable_pin, LOW);
382.          delayMicroseconds(100);   // commands need > 37us to settle
383.        }
```

```
384.
385.      void Adafruit_RGBLCDShield::write4bits(uint8_t value) {
386.        if (_i2cAddr != 255) {
387.          uint16_t out = 0;
388.
389.          out = _i2c.readGPIOAB();
390.
391.          // speed up for i2c since its sluggish
392.          for (int i = 0; i < 4; i++) {
393.            out &= ~(1 << _data_pins[i]);
394.            out |= ((value >> i) & 0x1) << _data_pins[i];
395.          }
396.
397.          // make sure enable is low
398.          out &= ~(1 << _enable_pin);
399.
400.          _i2c.writeGPIOAB(out);
401.
402.          // pulse enable
403.          delayMicroseconds(1);
404.          out |= (1 << _enable_pin);
405.          _i2c.writeGPIOAB(out);
406.          delayMicroseconds(1);
407.          out &= ~(1 << _enable_pin);
408.          _i2c.writeGPIOAB(out);
409.          delayMicroseconds(100);
410.
411.        } else {
412.          for (int i = 0; i < 4; i++) {
413.           _pinMode(_data_pins[i], OUTPUT);
414.           _digitalWrite(_data_pins[i], (value >> i) & 0x01);
415.          }
416.          pulseEnable();
417.        }
418.      }
419.
420.      void Adafruit_RGBLCDShield::write8bits(uint8_t value) {
421.        for (int i = 0; i < 8; i++) {
422.          _pinMode(_data_pins[i], OUTPUT);
423.          _digitalWrite(_data_pins[i], (value >> i) & 0x01);
424.        }
425.
426.        pulseEnable();
427.      }
428.
429.      uint8_t Adafruit_RGBLCDShield::readButtons(void) {
430.        uint8_t reply = 0x1F;
431.
432.        for (uint8_t i=0; i<5; i++) {
433.          reply &= ~((_i2c.digitalRead(_button_pins[i])) << i);
434.        }
435.        return reply;
436.      }
```

Bibliografía

[1]: http://es.wikipedia.org/wiki/Modelo_OSI#/media/File:Pila-osi-es.svg

[2]: http://www.iuma.ulpgc.es/~avega/int_equipos/trab9899/comuindus/Image77.gif

[3]: http://electronics.stackexchange.com/questions/73303/for-the-can-bus-is-it-ok-to-swap-canh-and-canl-lines

[4]: http://1.bp.blogspot.com/_2kyRWM6UW8E/TKm06ytPHsI/AAAAAAAAATY/sKFfbvS5-9c/s1600/can121.JPG

[5]: MCP2515 Datasheet

[6]: http://5hertz.com/images/30066.jpg

[7]: http://www.seeedstudio.com/wiki/CAN-BUS_Shield

[8] https://www.sparkfun.com/products/10039

[13]: http://www.microchip.com/wwwproducts/devices.aspx?dDocName=en010406

[14]: http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en010405

[15]: https://www.adafruit.com/products/714

[16]: http://img.dxcdn.com/productimages/sku_326450_1.jpg

[17]: http://www.seeedstudio.com/wiki/CAN-BUS_Shield ; https://cdn.sparkfun.com/assets/learn_tutorials/8/3/cableG.jpg

[18]: http://store.arduino.cc/product/M000006

[19] https://github.com/Seeed-Studio/CAN_BUS_Shield

[20] http://forum.arduino.cc/index.php?topic=131801.0

[21] http://forum.jeelabs.net/node/1188.html

[22] http://www.can-cia.org/index.php?id=166

[23]: http://www.can-cia.org/fileadmin/cia/pdfs/technology/hartwich1.pdf

[24] http://www.sciencedirect.com/science/article/pii/S014193310100148X

[25] http://papers.sae.org/2001-01-0073/

[27] https://github.com/adafruit/Adafruit-RGB-LCD-Shield-Library

[28] http://www.e-auto.com.mx/imagenes/manuales/electronica/otros/terminales-conector-obdii.jpg

[29] http://onlycarspictures.com/uploads/skoda/skoda-fabia-combi/skoda-fabia-combi-08.jpg

[30] http://en.wikipedia.org/wiki/OBD-II_PIDs

[31]http://upload.wikimedia.org/wikipedia/commons/a/ac/VW_Touran_1.9_TDI_BlueMotion_Technology_Highline_Deep_Black.JPG

[32]Redes de comunicación industriales. Nuria Oliva

[33]http://ww1.microchip.com/downloads/en/DeviceDoc/21801e.pdf

[34] http://www.atmel.com/images/doc8161.pdf

[35]http://www.ti.com/lit/an/sloa101a/sloa101a.pdf

[36] http://www.seeedstudio.com/wiki/CAN-BUS_Shield

[37] http://arduino.cc/en/pmwiki.php?n=Hacking/LibraryTutorial

[38] http://www.bosch-semiconductors.de/media/pdf_1/canliteratur/cia99paper.pdf

[39] http://www.volkspage.net/technik/ssp/ssp/SSP_238.pdf

[40] http://www.outilsobdfacile.com/location-plug-connector-obd/Skoda-octavia-3

[41] http://www.can-cia.org/index.php?id=166

[42] http://hackaday.com/2013/10/29/can-hacking-protocols/

[43] http://en.wikipedia.org/wiki/CAN_bus

[44] http://www.kvaser.com/can-protocol-tutorial/

[45] http://www.adafruit.com/products/715

[46] http://ww1.microchip.com/downloads/en/DeviceDoc/21801e.pdf

[47] http://www.canbushack.com/blog/index.php

[48] http://www.outilsobdfacile.com/obd-mode-pid.php

[49] https://es.scribd.com/doc/106167326/12/ISO-9141-2

[50] https://www.youtube.com/watch?v=OH1CGU-TGbA