Trabajo de Fin de Grado

Grado en Ingeniería en Tecnologías Industriales

Test, depuración y validación de un driver CANFD escrito en Python

MEMORIA

Autor: Pau Barnils i Vilar
Director: Manuel Moreno Eguílaz

Convocatoria: Septiembre 2022

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

Resumen

En la actualidad, uno de los protocolos de comunicación más utilizados en la industria de automoción es el denominado CANFD, una versión más rápida y más flexible que el archiconocido bus CAN. En consecuencia, este proyecto está enfocado en desarrollar una librería Python para el controlador de bus CANFD denominado MCP-2517FD, partiendo de una librería en C++ y del TFG del estudiante de ingeniería Albert Burgués.

En la memoria de este proyecto, primero se explicarán las bases para entender el funcionamiento del CANFD, algunos conceptos utilizados para desarrollar la librería y el hardware utilizado para progresar con el TFG. También se explica la situación del driver en el momento que se inició este TFG, repasando todas las funciones que se utilizarán y como mejorar la situación actual.

Finalmente, se explica paso a paso, cómo se ha ido desarrollando la librería en Python, todas la funcionalidades añadidas y las pruebas para demostrar que funciona correctamente. Para terminar, se incluyen las conclusiones del proyecto, describiendo el estado del driver una vez finalizado el TFG.

Sumario

ETSEIB

1. Glosario

- **CAN** *Controller Area Network*
- **CAN FD** *Controller Area Network with Flexible Data-rate*
- **CLK** *Clock signal*
- **CRC** *Ciclic Redundancy Check*
- **Cs/SS** *Chip Select signal*
- **C++** *Lenguaje de Programación*
- **FIFO** *First-In-First-Out stack*
- **GPIO** *General Purpose Input-Output*
- **MISO** *Master In, Slave Out*
- **MOSI** *Master Out, Slave In*
- **SFR** *Special Function Register*
- **SPI** *Serial Peripheral Interface*
- **RAM** *Random Access Memory*
- **SPI** *Serial Peripheral Interface*
- **.py** *Python file extension*
- **.txt** *Text file*

2. Prefacio

2.1. Origen del proyecto

El origen del proyecto proviene de la intención de migrar el driver del lenguaje C++ [13] a Python por las ventajas de utilizar este código de programación, como su sencillez.

El objetivo personal de hacer este trabajo, es el poder adentrarse en el mundo de la programación para observar cómo se utiliza Python en un entorno diferente al utilizado durante el grado.

2.2. Requisitos previos

Para entender el trabajo desarrollado y los conceptos que se tratan en este proyecto, se debe tener un mínimo nivel de programación con Python y tener conocimiento de los trabajos previos de Albert Burgués (TFG) [11] y Joaquín Cortés (TFM) [12], ya que es una continuación de ambos proyectos.

Se explicarán los conceptos relacionados directamente con el proyecto, para describir paso a paso toda la evolución del trabajo, intentando aclarar cualquier duda de éste.

3. Introducción

3.1. Objetivos del proyecto

El principal objetivo de este proyecto es depurar el driver para el controlador CAN-FD MCP-2517FD, un controlador de CAN-FD compatible con una Raspberry Pi 3b y escrito en lenguaje de programación Python por Albert Burgués [11].

Posteriormente a la depuración de este driver, también hay como objetivo introducir nuevas funcionalidades. Estas funcionalidades serán la implementación de lectura y escritura con detección de errores mediante códigos redundantes tipo CRC.

3.2. Alcance del proyecto

Una vez establecido el objetivo del proyecto, el alcance lo definiremos según el segundo objetivo de ampliar las funcionalidades del CAN-FD.

El proyecto comienza por los conceptos que se utilizarán en el trabajo, siguiendo con la depuración de errores del driver y finalmente, con la introducción de funciones CRC, que están implementados en el driver original escrito en lenguaje C++ [13].

4. Conceptos utilizados y descripción del hardware

Para entender el trabajo realizado, se introducirán unos conceptos tanto de las características de un controlador CANFD y su protocolo, como de las características del hardware utilizado, la configuración de éstos y otros conceptos utilizados para el proyecto.

4.1. Protocolos de comunicación

En informática, un protocolo de comunicación es el sistema de reglas que permite que dos o más dispositivos de un sistema de comunicación se comuniquen entre ellos para transmitir información por medio de un canal. Se basa en unas reglas y acuerdos que define la semántica, sintaxis y sincronización de la comunicación. Pueden ser implementados por software, por hardware o una combinación de ambos.

En este proyecto, se utiliza el protocolo CAN-FD (Controller Area Network Flexible Data-Rate), presente hoy en día en el mundo de la automoción.

4.1.1. SPI Communication

El “Serial Peripheral Interface” (SPI) es uno de los estándares de comunicación más utilizados entre un microcontrolador y periféricos, siendo periféricos sensores, ADCs (convertidor analógico/digital), DACs (convertidor digital/analógico) y otros. Dicho protocolo se utiliza principalmente para la transferencia de datos.

El SPI es un interfaz síncrono, que utiliza un flujo de bits tipo serie para comunicarse con los otros dispositivos y utiliza una señal de reloj común para la sincronización de nodos.

Funcionamiento del SPI

Una de las características del SPI es su estructura Maestro/Seguidor, siendo el “maestro” habitualmente un microcontrolador y los “seguidores”, distintos periféricos como, por ejemplo, sensores. Éstos están conectados mediante un bus serie, minimizando el número de conductores y pines utilizados. Se utilizan 4 señales digitales:

- SCLK (Clock): Es la señal que permite la sincronización. Con cada flanco positivo de la señal de reloj, se lee o se envía un bit. También llamado TAKT (en alemán).
- MOSI (Master Output Slave Input): Salida de datos del Master y entrada de datos al Seguidor. También llamada SIMO.

- MISO (Master Input Slave Output): Salida de datos del Seguidor y entrada al Master. También conocida por SOMI.
- SS/CSelect: Permite seleccionar un Seguidor, o para que el Master le diga al Seguidor que se active. También llamada SSTE.

La cadena de bits es enviada de manera síncrona con los pulsos del reloj. El dispositivo maestro es responsable de configurar la frecuencia del pulso, la polaridad y la fase. La cadena suele ser de 8 bits. Para habilitar un periférico, el maestro fuerza la señal SS (CS) a nivel bajo (LOW), y el seguidor se activa para empezar la transmisión con el pulso del reloj.

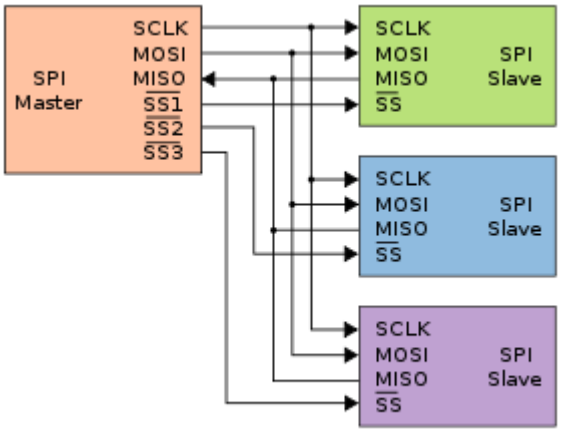


Fig. 1: Representación de un sistema de comunicación SPI con un maestro y tres seguidores. Fuente: [4].

4.1.2. Protocolo CAN FD

CANFD es un protocolo para intercambiar información entre unidades electrónicas de control (ECUs), sensores y actuadores en un vehículo automóvil.

La diferencia entre el protocolo clásico CAN y el nuevo CANFD es que se pueden enviar más datos a más velocidad. En efecto, el CAN clásico admite un máximo de 8 bytes por mensaje, mientras que el CANFD hasta un máximo de 64 bytes. La velocidad máxima del CAN clásico es de 1 Mbit/s, mientras que el CANFD, en condiciones de laboratorio, puede trabajar ocho veces más rápido, es decir, con tasas de transmisión de 8 Mbit/s.

Una de las características del CAN es que admite un control centralizado de sus dispositivos electrónicos conectados en el bus.Cada dispositivo conectado al bus CAN se llama nodo y éste se compone habitualmente de un microcontrolador, un controlador CAN y un transceptor CAN, tal y como se muestra en la Fig. 2. En este ejemplo el controlador es el modelo MCP-2517FD, el transceptor, el dispositivo comercial ATA6563, que se conectan a

un procesador central (Raspberry Pi 3B).

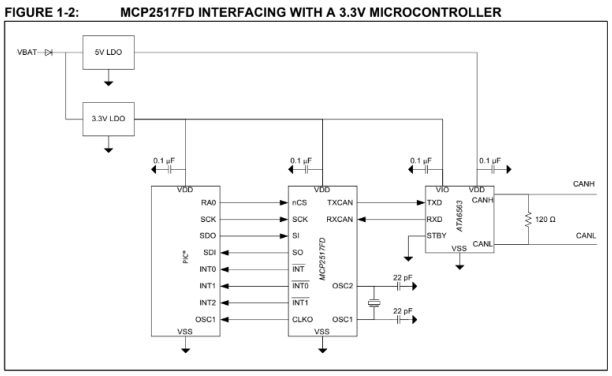


Fig. 2. Esquema de un posible nodo de bus CANFD. Fuente: [1].

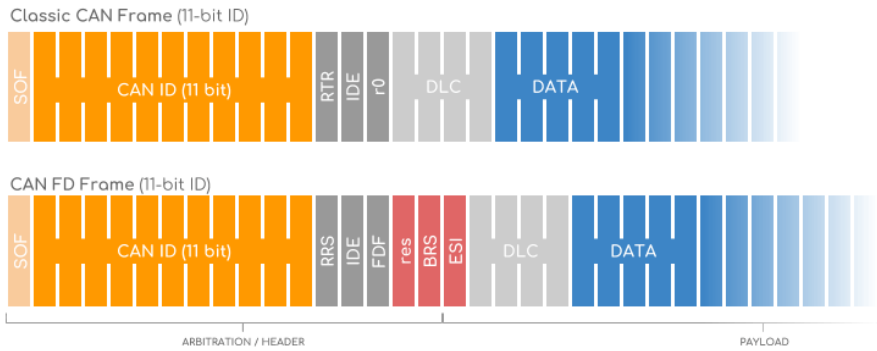
Dado que el bus CANFD es un medio compartido, se utiliza un sistema de arbitraje para priorizar el orden de los mensajes. También se dispone de un complejo sistema de detección de errores integrado, deteniendo la transmisión cuando es necesario.

La implementación del CAN FD también aporta un bajo coste, una alta velocidad, suficiente para muchos usos de industrias específicas, durabilidad y flexibilidad.

Estructura de un mensaje de datos en CANFD

Como se ha mencionado anteriormente, el CANFD permite utilizar diferentes tamaños de datos hasta un máximo de 64 bytes y mejor velocidad de transferencia que el bus CAN tradicional. Todo ello implica una estructura diferente en los bits enviados.

La información enviada en un mensaje de bus CAN y CANFD se empaqueta en lo que se denomina una trama (frame), que contiene una serie de campos para delimitar su inicio y final, como son:



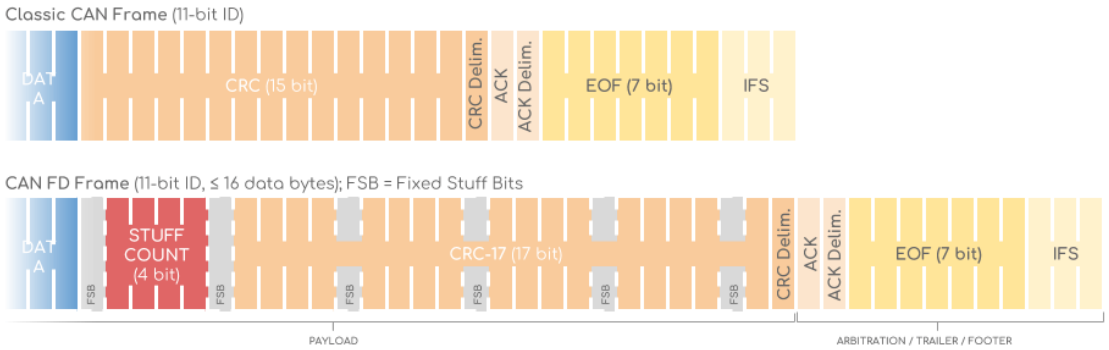


Fig. 3: Estructura de una trama de CAN y de CANFD. Fuente: [6].

RRS: La Sustitución de solicitud remota (RRS) siempre es dominante (0). En el CAN clásico se utiliza para identificar tramas (RTR).

FDF: En CAN clásico, r0 es reservado y dominante (0). En CANFD, se denomina FDF y es recesivo (1).

Después del bit FDF, el protocolo CANFD agrega "3 nuevos bits". Se debe tener en cuenta que los nodos que no son compatibles con CANFD producen una trama de error después del bit FDF.

res: Este nuevo bit reservado desempeña el mismo papel que r0, es decir, en el futuro puede establecerse en recesivo (1) para indicar un nuevo protocolo.

BRS: El interruptor de tasa de bits (BRS) puede ser dominante (0), lo que significa que la trama de datos CAN FD se envía a la tasa de arbitraje (es decir, hasta un máximo de 1 Mbit/s). Establecerlo en recesivo (1) significa que la parte restante de la trama de datos se envía a una tasa de bits más alta (hasta 5 Mbit/s).

ESI: El bit del indicador del estatus del modo de error. Si es dominante (0), está en estado del error activo, en (1), pasivo.

Longitud de datos DLC Bus CAN Velocidad de datos flexible

DLC (Data Length Code): Al igual que en CAN clásico, el DLC CAN FD es de un palabra de código de4 bits, que indica la cantidad de bytes de datos en la trama en cuestión. La tabla de la Fig. 4 muestra cómo los dos protocolos usan el DLC consistentemente hasta 8 bytes de datos. Para mantener un DLC de 4 bits, CANFD utiliza los 7 valores restantes del 9 al 15 para indicar el número de bytes de datos utilizados (12, 16, 20, 24, 32, 48, 64).

DLC (bin)	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
DLC (dec)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Classic CAN	0	1	2	3	4	5	6	7	8	8	8	8	8	8	8	8
CAN FD	0	1	2	3	4	5	6	7	8	12	16	20	24	32	48	64

Fig. 4: DLC y número de bytes en CAN y CANFD. Fuente: [6].

SBC: El Stuff Bit Count (SBC) precede al CRC y consta de 3 bits codificados y un bit de paridad. El siguiente bit fijo se puede considerar como un segundo bit de paridad. El SBC se agrega para mejorar la confiabilidad de la comunicación.

CRC: La comprobación de redundancia cíclica (CRC) es de 15 bits en el CAN clásico, mientras que en el CANFD es de 17 bits (para hasta 16 bytes de datos) o de 21 bits (para 20-64 bytes de datos). En CAN clásico, puede haber de 0 a 3 bits de relleno en el CRC, mientras que en CANFD siempre hay cuatro bits de relleno fijos para mejorar la confiabilidad de la comunicación.

ACK: La fase de datos (también conocida como carga útil) de una trama de datos CANFD se detiene en el bit ACK, que también marca el final de la tasa de bits potencialmente aumentada.

4.2. Cyclic Redundancy Check (CRC)

4.2.1. Funcionamiento del CRC

La verificación de redundancia cíclica es un método diseñado para detectar errores en la comunicación en este caso con el SPI. Funciona como un “Checksum”, es decir, una suma de comprobaciones que se obtiene de un origen, usualmente el mensaje, este se le calcula el CRC y se guarda en el mensaje de la transmisión. Después de la transmisión se recalcula con el mensaje recibido y se compara el CRC del mensaje con CRC obtenido del cálculo para comprobar que los datos de la transmisión sean los mismos sin ningún cambio de bit por otros factores. Si ha habido una diferencia entre la comparación del CRC obtenido salta la señal de error.

Este cálculo de detección de errores se ha utilizado desde el principio de las transmisiones en informático, propuesto por primera vez en 1961 por W. Wesley Peterson. También es utilizado en archivos de formato .zip o similares.

El CRC es atractivo por una serie de características que son:

- El resultado es más pequeño que el mensaje, en este caso 16 bits.
- El resultado es casi único.
- Un pequeño cambio en el origen, en este caso el mensaje, produce muchos cambios en el resultado.
- Fácil de implementar y sencillo de analizar matemáticamente.

Con estas características se utiliza el CRC en la detección de errores en la comunicación entre un procesador (Raspberry Pi) y el controlador de CANFD.. En este proyecto se utilizará un CRC de 16 bits, CRC-16. Esto afecta a qué tipo de errores detecta y el porcentaje de detección. Estos errores se pueden clasificar en:

Random error: Se pueden producir por algún o algunos bits cambiados durante la transmisión. El CRC-16 tiene un porcentaje de precisión de 0,0015 %.

Burts error: A veces se produce un error en unos cuantos bits consecutivos. Estos errores se llaman “Burst errors”, muy comunes en la comunicación de datos. El CRC-16 detecta estos errores más pequeños de 16 bits.

4.2.2. Calculo del CRC

El CRC se puede calcular con diversos algoritmos, dando diferentes resultados con el mismo mensaje dependiendo del algoritmo que se use.

En este proyecto se utilizara el método de cálculo de CRC16, y CRC 16/USB, que utilizan una tabla tipo “LookUp Table”, que es una memorización de un cálculo que tiene que repetirse por cada byte del mensaje [8][9].

Esta técnica de memorización es una técnica de optimización que se utiliza para acelerar programas informáticos, en este caso para obtener más rápido un resultado del cálculo del CRC. Esta optimización consiste del almacenamiento de resultados que llaman las funciones y retornando el resultado almacenado otras veces cuando las entradas se repiten.

La lógica en el cálculo del CRC con memorización es la siguiente:

```
Función CRC32
    Entrada:
      datos: Bytes      // Matriz de bytes
   Salida:
      crc32: UInt32     // Valor CRC-32 sin signo de 32 bits

// Inicializar CRC-32 al valor inicial
crc32 ← 0xFFFFFFFF

 para cada byte en datos do
   nLookupIndex ← (crc32 xor byte) y 0xFF
   crc32 ← (crc32 shr 8) xor CRCTable[nLookupIndex]   // CRCTable es
una matriz de 256 constantes de 32 bits

// Finalice el valor CRC-32 invirtiendo todos los bits
crc32 ← crc32 xor 0xFFFFFFFF return crc32
```

Fig. 5. Lógica del cálculo del CRC-16/USB. Fuente: [9].

4.3. Hardware

En la actualidad hay una gran cantidad de procesadores, controladores, sensores y dispositivos electrónicos de diferentes marcas que utilizan diferentes formatos programas y drivers para funcionar. Así que primero debemos estudiar los dispositivos que tenemos, para conocer sus limitaciones y funcionamiento para desarrollar el driver correctamente.

4.3.1. Rapberry Pi 3 B

Como procesador, tenemos el de una Raspberry Pi 3B, siendo ésta la tercera generación de una serie de computadoras personales, caracterizadas por el bajo consumo, el bajo precio y el tamaño de éstas. La Raspberry Pi 3B utiliza el Raspbian como sistemas operativo (OS), una variante de Linux disponible en la página web de Raspberry.

Este ordenador está compuesto de:

- **Componentes en la placa base**: Procesador de 4 núcleos, 1 GB de RAM, Wireless PCI (para el Wi-fi), Bluetooth.
- **Conectores**:
    - 4 conectores USB 2: Donde irán conectado el ratón, teclado y un lector de tarjetas.
    - 40 pines GPIO extendido: Son los pines donde se pueden conectar los periféricos con diferentes estándares de comunicación, como el SPI, I2C, etc. La Raspberry Pi 3B tiene contiene los siguientes periféricos de comunicaciones: 2 SPI, 1 I2C y una UART. Estos pines son los responsables de enviar y recibir información con los periféricos.

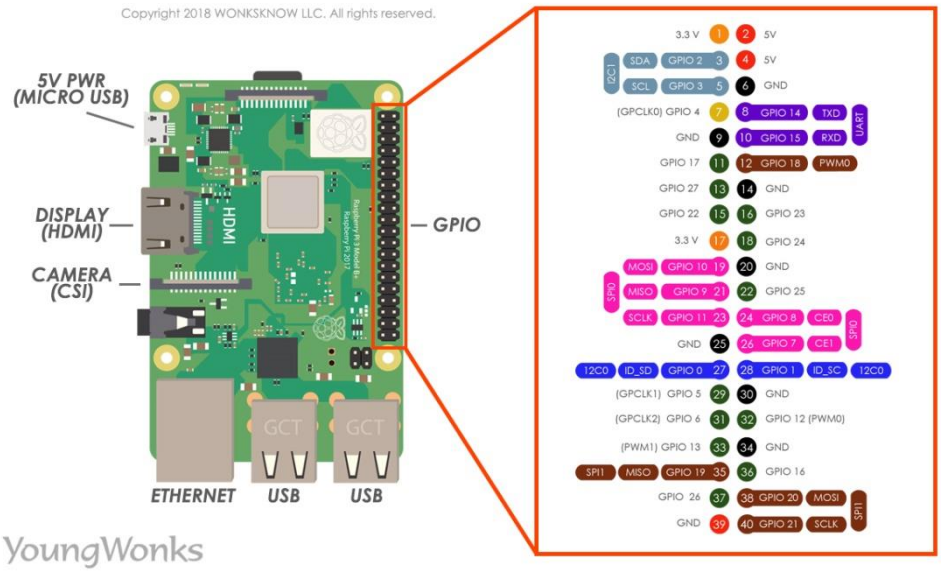


Fig. 6. Raspberry Pi i Pines Fuente: [14].

- HDMI: Donde conectaremos la pantalla.
- Micro-SD Card, Micro USB 5 V como entrada para la fuente de alimentación (Cargador de móvil de 60 W).

4.3.2. Controlador de CANFD MCP-2517FD

El MCP-2517FD es un controlador externo de CANFD de la compañía Microchip Technology Inc. [1]. Dicho controlador comercial es de pequeñas dimensiones, bajo consumo, bajo coste e incluye comunicaciones SPI.

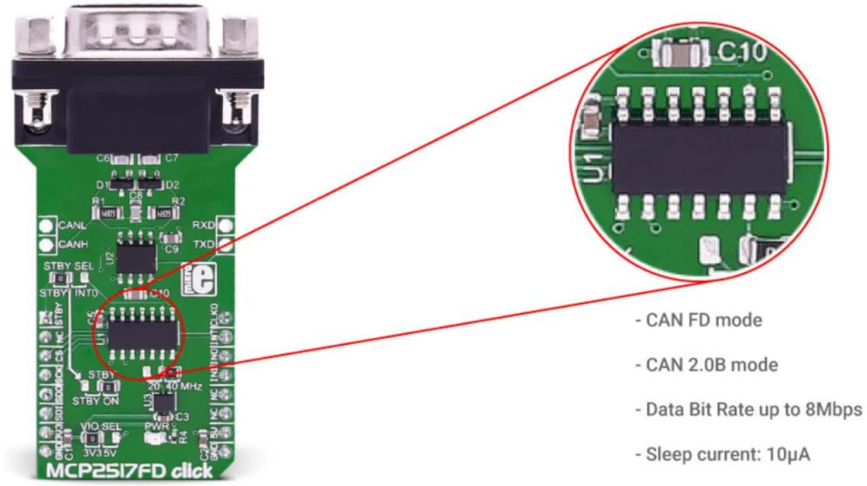


Fig. 7. Controlador MCP2517FD de CANFD sobre una placa tipo "click". Fuente: [1].

El MCP-2517FD tiene las siguientes especificaciones:

- Soporte para CAN 2.0B y **CAN FD**
- Velocidad de bits de arbitraje de hasta 1 Mbps
- Velocidad de bits de datos de hasta 8 Mbps
- Velocidad de reloj SPI de hasta 20 MHz
- 31 FIFO configurables como transmisión o recepción
- 32 filtros flexibles y objetos de máscara
- Una cola de transmisión
- Corriente activa máx: 12 mA a 5,5 V, 40 MHz de reloj CAN
- Corriente de suspensión: 10 uA, típico
- Diagnóstico de estado de bus y contadores de error
- Dimensiones 4.5 x 3 x 0.85 mm

Funcionamiento

El controlador MCP-2517FD está diseñado para interactuar directamente con un procesador, en este caso el de la Raspberry, con los pines del periférico de comunicaciones SPI.

Para enviar un mensaje, primero se accede a la RAM desde el procesador para escribir el mensaje en ella, utilizando los registros. Cada registro está configurado y tiene sus propósitos. Entonces, se añade a la cola para transmitir el mensaje al Bus CANFD(TXQ) si el registro está pre-configurado para ello.

Para acceder a la memoria, el controlador funciona con instrucciones. Estas instrucciones tienen diferentes estructuras, dependiendo de si se escribe o lee. También depende de si se quiere acceder a la RAM o a la memoria de registros especiales SFR.

A continuación se muestra un ejemplo de cómo están estructuradas las instrucciones,siendo primero la activación del SPI, (señal chip_select CS a 0), después la función que deseamos ejecutar y la dirección de la memoria. Finalmente, se incluye en esta instrucción la información y desactivación del SPI (señal chip_select CS a 1).

SFR read Instruction

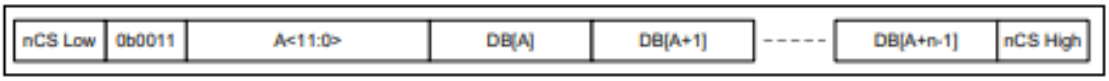


Fig. 8. Instrucción SFR read. Fuente: [1].

Memoria

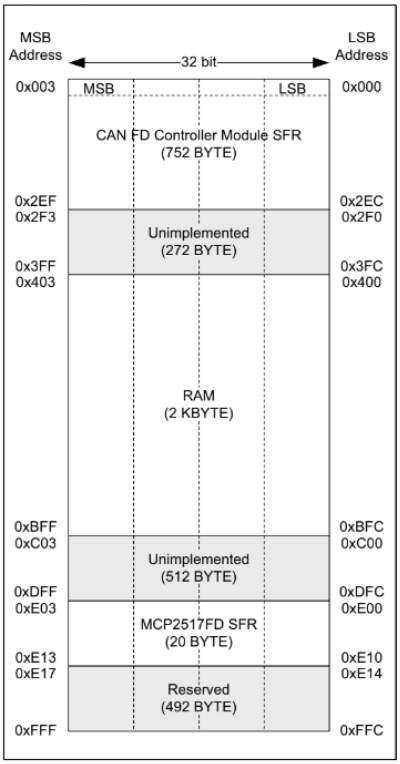


Fig. 9. División de la memoria de controlador MCP-2517FD. Fuente: [1].

La memoria se divide en secciones, cada una tiene un propósito diferente por el cual queremos acceder. Estas secciones son:

- CANFD Controller Module SFR **[0x000-0x2EF]**: Memoria donde se guardan las funciones del controlador.
- Message Memory **RAM**: Memoria general donde se guardan los mensajes.
- MCP-Special Function Registers (SFR): Memoria donde se guardan bits de funciones especiales del controlador, por ejemplo, la detección de

error del CRC.

La memoria funciona a base de direcciones, siendo ésta de una anchura (longitud de palabra) de 32 bits, 4 Bytes siendo el 0x000 el Less Significant Byte (LSB), y el 0x003 el Most Significant Byte (MSB). Utilizado para saber el formato de escritura de la RAM, ya que usa el BIG endian, formato que parece más natural a la hora de escribir. Por ejemplo:

Los datos de información “13” y "trece", codificados en hexadecimal son 0x3133 y 0x74726563650d0a.En formato Big Endian se expresan como:

```
0x31 0x33
y
0x74 0x72 0x65 0x63 0x65 0x0d 0x0a
```

Y en formato Little Indian, es decir, el otro formato posible, quedarían:

```
0x33 0x31
y
0x0a 0x0d 0x65 0x63 0x65 0x72 0x74
```

4.3.3. Configuración y montaje

El primer paso para empezar el trabajo es conectar los pines de la Raspberry adecuadamente con el controlador, para que todo funcione correctamente.

Primero se conectan los periféricos del ordenador, el ratón, el teclado y el lector de tarjetas en los puertos USB, además del HDMI en la pantalla y el cargador para tener una fuente de alimentación en el micro-USB clásico.

Seguidamente, montamos los pines del controlador y la Raspberry pi 3B. En la Raspberry Pi cada pin corresponde a una funcionalidad específica. Así, conectamos los pines correspondientes del SPI al controlador MCP-2517FD, como se indica en las *Figs. 10 y 11*.

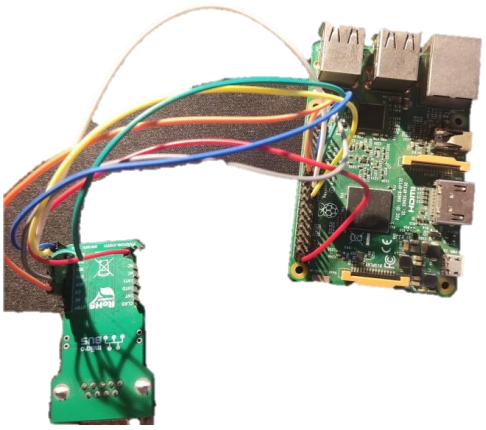


Fig. 10. Raspberry Pi montada. Fuente: [Propia]

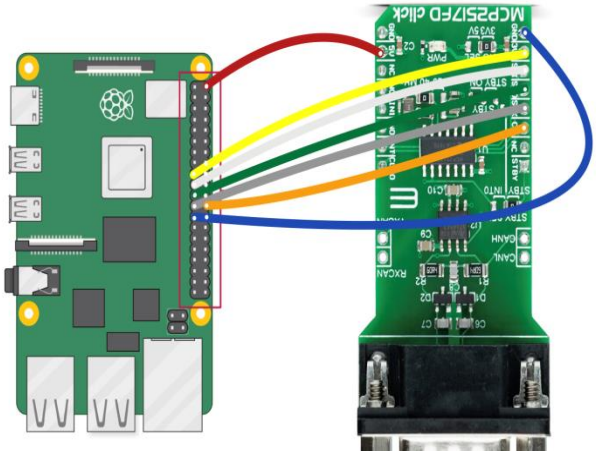


Fig. 1. Cables Raspberry pi. Fuente: [Propia]

Como se puede observar, una vez esta todo conectado se configura la Raspberry Pi y ya se puede empezar a trabajar con el proyecto.

Para configurar la Raspberry Pi, se debe poner el teclado al idioma correspondiente y se deben activar los pines de la Raspberry a través de comandos de la terminal. Una vez activados los GPIOs (los pines), se empieza a trabajar y analizar el estado actual del driver.

5. Situación actual del driver

Para empezar el trabajo primero veremos de manera extendida la situación actual del driver en Python, partiendo de la base que dejó el estudiante Albert Burgués en su TFG [11].

Partimos de la librería principal CANFD_SPI(), en el fichero “canfdlib2.py”, y del fichero de para hacer los test para ver si el driver funciona, denominado “rw.py”.

Funciones para leer y escribir en la memoria

El fichero “canfdlib2.py” empieza con las siguientes funciones, que se tienen que verificar y comprobar su correcta funcionalidad.

Estas funciones son:

- **readByte(**Dirección**)**: Función por leer el primer byte de la dirección deseada.
- **readWord(**Dirección**)**: Función por leer la palabra (4 bytes), de la dirección deseada.
- **readByteArray(**Dirección**,** nº de Bytes**)**: Función por leer el número de Bytes deseados en una dirección.
- **readWordArray(**Dirección**,** nº de Words): Función por leer el número palabras (4 Bytes) en una dirección.
- **writeByte(**Dirección, Mensaje**):** Función por escribir un Byte en una dirección
- **writeWord(**Dirección, Mensaje**)**: Para leer la palabra (4 bytes), de la dirección deseada.
- **writeByteArray(**Dirección, Mensaje**)**: Función por escribir una lista con Bytes como elementos.
- **writeWordArray(**Dirección, Mensaje): Función por escribir una lista de palabras.

Palabra: Término utilizado para describir 4 Bytes que van en una dirección.

Estructura de las funciones básicas

Para entender cómo funcionan estas funciones internamente, debemos saber cómo interpretará el controlador los bits que le enviamos para guardar en su memoria, así como la estructura del mensaje para realizar la función deseada.

Para leer y escribir, en **la hora característica** del controlador MCP-2517FD [1], se explica detalladamente cómo se debe estructurar el mensaje para leer y escribir en la memoria de éste. Para escribir y leer se debe tener una estructura como la siguiente:

FIGURE 4-5: MESSAGE MEMORY READ INSTRUCTION

nCS Low	0b0011	A<11:0>	DW[A]				nCS High
			DB[A]	DB[A+1]	DB[A+2]	DB[A+3]	

FIGURE 4-6: MESSAGE MEMORY WRITE INSTRUCTION

nCS Low	0b0010	A<11:0>	DW[A]				nCS High
			DB[A]	DB[A+1]	DB[A+2]	DB[A+3]	

Fig. 22. Instrucciones Write/Read en la memoria RAM. Fuente: [1].

La figura 12 se ilustran las instrucciones para leer y escribir mientras se accede a la RAM, resumido en el siguiente diagrama de bloques:

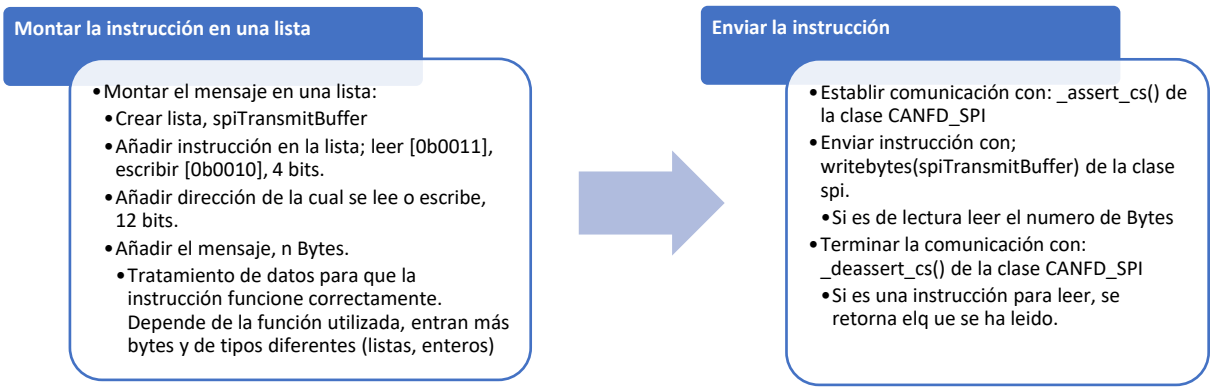


Fig. 13. Diagrama lógico de las funciones. Fuente: [Propia]

Test para comprobar el correcto funcionamiento del driver

El fichero “rw.py” se utiliza para hacer las pruebas y comprobar que las funciones de la clase funcionen correctamente.

En el primer estado éste tiene 7 pruebas, cada una con una funcionalidad distinta.

Los test 0 y 1 son los que prueban que las funciones del controlador, CANFD_SPI.readbytes() y CANFD_SPI.writebytes(), utilizadas para escribir funciones a un nivel más alto, escriban y lean en la memoria deseada correctamente. El primer estado de éstas es un test de un solo valor probado en la memoria SFR, en la dirección 0xE00 y la dirección 0x000 de la memoria del controlador, para leer y escribir, respectivamente.

Los test 2 y 3 son las pruebas para ver si las funciones escritas en el fichero canfdlib2.py funcionan correctamente. Inicialmente se prueban estas funciones con un solo mensaje de 1 byte con las funciones readByte(), writeByte(), y 4 bytes en las otras funciones explicadas

anteriormente. Estos test prueban enviar un solo mensaje en la dirección 0x000 de la memoria del controlador.

Los test 4, 5 y 6 son test de comprobación para ver si el controlador funciona correctamente. Los test 4 y 5 prueban si se puede leer y escribir correctamente en los registros de la memoria (SFR) y en la RAM, respectivamente. El último prueba si se pueden hacer los cambios de estados del controlador, el Device Mode, cada uno para utilizar el controlador de una manera distinta, ya que éste tiene 7 modos diferentes definidos en el archivo "constants.py" [1].

El test 7 actualmente no funciona y se prueba el loopbackMode con un fichero separado, donde simula un envío de un mensaje hacia el bus CAN, y después lo recibe y comprueba que el mensaje enviado sea el mismo que el recibido. Este test está actualmente en el "loopbackMode3.py" y funciona de la siguiente manera:

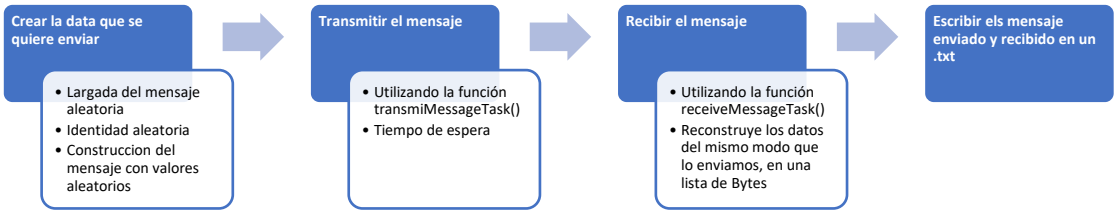


Fig. 14. Diagrama lógico del test 7. Fuente: [Propia]

Estos 3 ficheros con las siguientes funciones son los que se modificarán para verificar errores y progresar en la mejora de este driver en lenguaje python.

6. Depuración de errores

6.1. Detección de errores

En este apartado se definirá como hacer unos test extensivos para detectar posibles errores en las funciones actualmente disponibles en el driver.

Para hacer el test más exhaustivo se utilizará el test 1, 2 y 3 del fichero "*rw.py"*. También se hará un test de la simulación del CANFD, con el modo "external loopbackmode", donde se verificará la utilidad de los filtros y las máscaras y se podrá simular si todo funciona correctamente. Este test pasará a ser el test 7 del "rw.py", se arreglará y mirará que funcione correctamente para comprimir todo el trabajo en menos ficheros y poder correr todos los test desde un mismo lugar.

Para hacer los test exhaustivos, nos basaremos en 2 hipótesis. Los datos "data" que enviaremos serán totalmente aleatorios, dentro los límites de los bits. Por ejemplo, si ese trata de 1 byte, los datos que se comprobarán estarán entre 0x00 y 0xFF.

Estos datos se enviarán a la RAM de la placa, con una capacidad de 2 Kbytes, y que comprende un rango de direcciones entre 0x400 y 0xBFF, donde se almacenarán. Una vez almacenados los datos, se hará una comprobación con los que hemos enviado y con la que leemos con esas mismas funciones.

Los resultados de los test se guardan en los ficheros ".txt", que están dentro de la carpeta de trabajo.

6.1.1. Test 1

El objetivo de este test es comprobar qué porcentaje o qué errores pueden surgir dentro de las funciones del SPI, *writebytes o readbytes*. Se trata de probar esas funciones con diferentes direcciones para probar que funciona, no sólo en la memoria SFR.

Así que mejoraremos estos 3 puntos:

- La dirección del mensaje será aleatoria (random) dentro de la memoria RAM.
- El mensaje tendrá un valor aleatorio, dentro de los 4 Bytes para poder escribir en la RAM, característica descrita en el manual del controlador [1].

- Iteración de diferentes valores del mensaje.

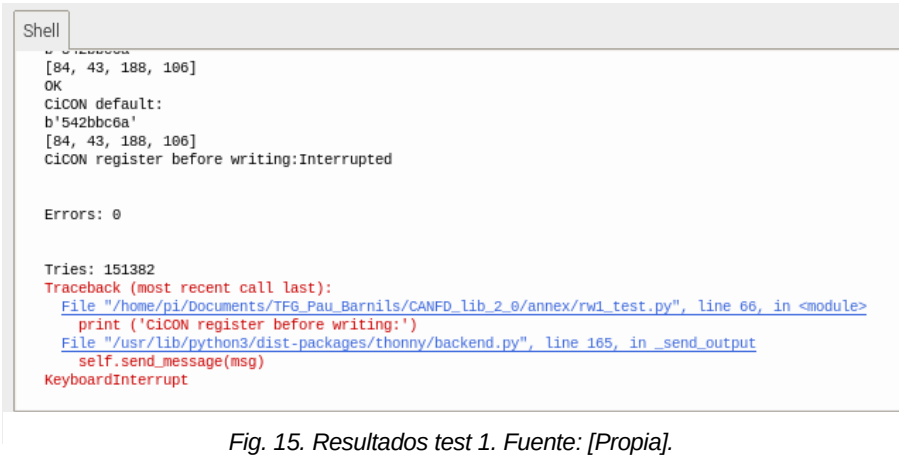


Fig. 15. Resultados test 1. Fuente: [Propia].

Finalmente, ya que el objetivo del test es probar y observar si se producen errores, se ejecuta el test para observar si se producen errores relacionado con el código. La *Fig. 15* representa el resultado después de ejecutar el test durante unas horas.

6.1.2. Test 2

El objetivo de este test es mirar qué porcentaje o qué errores pueden surgir dentro de las funciones *readWord, readByte, readByteArray y readWordArray* y solucionarlos si los hay.

Se mejora el test con:

- La dirección del mensaje será aleatoria (random) dentro de la memoria de la SFR.
- Iteración de diferentes direcciones.

Para probar se utilizará este test comprobando la memoria SFR sea siempre el mismo valor. Este test se ha ampliado utilizando una lista obtenida de la memoria SFR que está entre la dirección 0x000 y 0x2EC, que siempre es la misma cuando se reinicia el controlador, y al ejecutar el programa se compara el valor leído con el valor de la lista requerida.

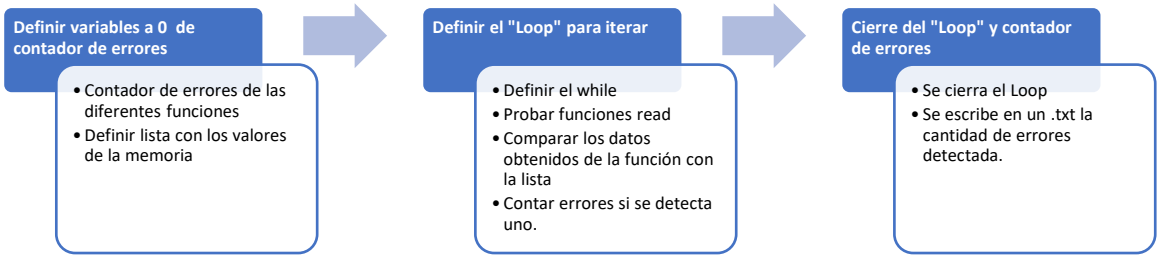


Fig. 16. Diagrama de bloques test exhaustivo. Fuente: [Propia].

6.1.3. Test 3

El objetivo de este test es mirar qué porcentaje o qué errores pueden surgir dentro de las funciones *writeWord, writeByte, writeByteArray y writeWordArray* y solucionarlos si los hay.

Se mejora el test con los siguientes puntos:

- La dirección del mensaje será aleatoria (random) dentro de la memoria RAM.
- Diferentes mensajes y diferentes largada del mensaje si la función lo permite.
- Los resultado y errores obtenidos se imprimen en el fichero "Errors.txt", para poder observar y analizar mejor las posibles mejoras, se ha añadido el **registro de fallos**.

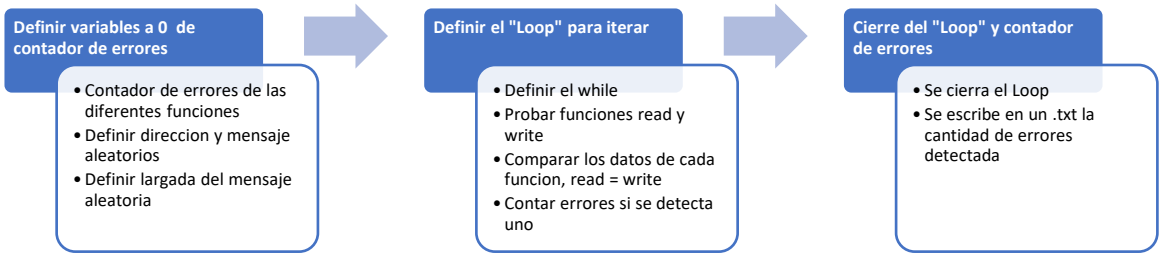


Fig. 17. Diagrama test exhaustivo. Fuente: [Propia].

```
Interrupted!
 Errors Write_Word: 0
 Errors Write_Byte: 87871
 Errors Write_ByteArray: 0
 Errors Write_WordArray: 0
```

Fig.18. Resultados primera prueba del test 3. Fuente: [Propia]

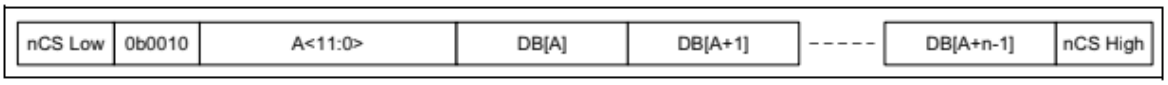
Como podemos observar en la *Fig. 18*, cuando ampliamos la dirección de donde escribimos a la RAM, la función writeByte() no funciona correctamente.

Escribir en la RAM y escribir en la memoria SFR

En este primer test, se vio las diferencias de estructura que deben tener las funciones cuando se escribe a la RAM o a la memoria SFR.

Así que para arreglar, se puede observar en la guía del controlador [1], que si se escribe en la RAM, los datos transferidos tienen que ser un múltiplo de 4 bytes. También se aplica en la función writeByteArray(), que posteriormente se cambió el test para que los mensajes tengan diferentes longitudes, no solo listas de 4 Bytes y más adelante también se aplica a las funciones con detección de errores cíclicos.

SFR write instruction



RAM write instruction

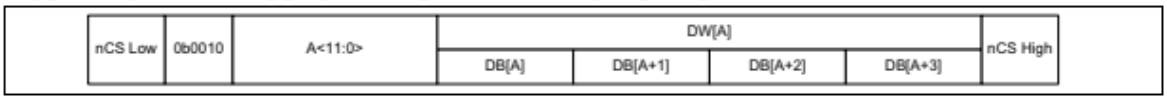


Fig. 19. Instrucciones de las funciones read, write. Fuente: [1].

Se ha añadido un código que añade los 0 necesarios para que se pueda escribir en la RAM. Tanto en la función writeByte() y writeByteArray() se tiene que añadir este código de tratamiento de los datos para que sean múltiples de **4 bytes** y pueda escribir en la RAM.

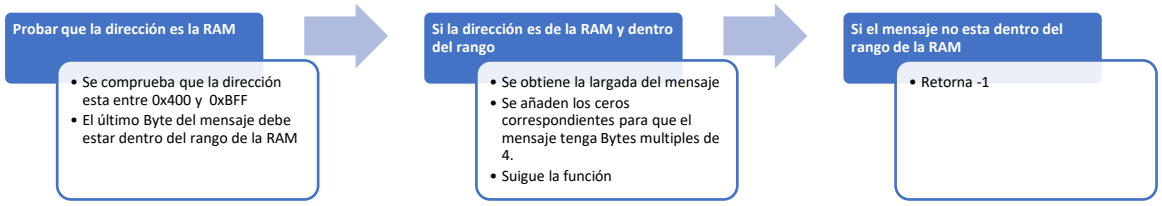


Fig. 20. Diagrama lógico de comprobación de la RAM. Fuente [Propia]

Al añadir esta función, se separaron los tramos de memoria donde se escribe está dentro de los límites de la RAM. Esta función ha permitido detectar errores de escritura donde el mensaje es demasiado largo y no termina dentro de la memoria RAM, entonces no se escribe correctamente.

Este error de ha denominado **ErrorData**, una nueva variable para ver que el mensaje no se ha escrito correctamente para volver a escribir el mensaje si es necesario en otra dirección. Esta variable está incluida en las todas las funciones que se han desarrollado en este trabajo.

6.1.4. Test LoopBack Mode

El objetivo de este test es incorporar el archivo “loopbackMode3.py” dentro del archivo de los test “rw.py” y probar cuántos errores tenemos con diferentes filtros y máscaras.

El segundo objetivo de este test es ver si la función de filtros y máscaras funciona correctamente. Así que las mejoras de éste serán:

- Comprobación automática de que el mensaje enviado es el mismo que el recibido.
- Comprobación del funcionamiento correcto de los filtros.
- Los resultados se imprimen en el fichero “Data.txt”.

Diagrama de bloques para el filtro y la máscara es el siguiente:

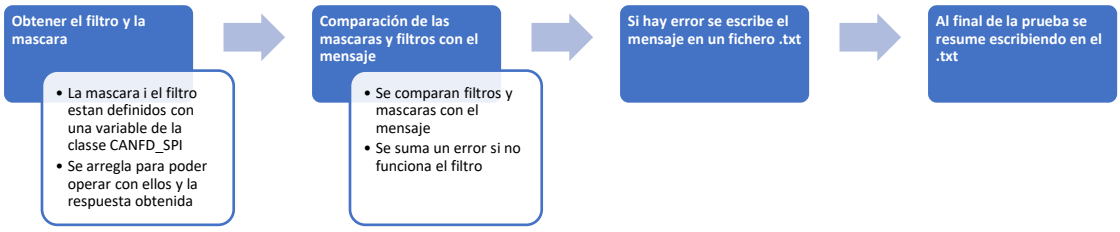


Fig. 21. Diagrama lógico filtros y máscaras. Fuente [Propia]

La lógica utilizada para comparar el filtro y la máscara es la siguiente:

Lógica para comparar y comprobar si los filtros y las máscaras funcionan:

```
Obtener filtros y máscaras  # dentro de la clase CANFD_SPI
SID = Filtro
MSID = Máscara
Definir masknot = 0b1111111
x = (SID ^ iden) ^ masknot  #Xor del filtro y el mensaje, 1 si no
                            coinciden, 0 si coinciden
                            #Xor con el masknot, 0 si bits no
coinciden, entonces la máscara en este bit debe ser 0.
Si los bits del mensaje y el filtro coinciden es 1, entonces da
igual la máscara.



Lista de MSID y de la máscara, de 0 y 1 con strings
Se iguala la longitud de las listas x y de la máscara, con un for

if longitud de la lista != 11:  #Filtros de 11 bits
      num_af = 11 - longitud de la lista  #Numero de 0 para añadir
      for cada bit en el rango de 0 para añadir:
            insertar '0' al principio de la lista

Comparación de las listas
X y la lista de la máscara se comparan bit a bit.

--> Si los bits de X es 0, el bit de la máscara debe ser 0, es
decir no significativo.
--> Si el bit de X es 1, el bit de la máscara da igual

Detecta el error
--> Si el bit de la máscara es 1 --> El bit de X debe ser 1 --> El
bit de la identidad es el mismo que el del filtro.
```

```
Filter_Test = 1      #Detector de error, 1 no se detecta error.
for por indice en rango 11:
      if l[i] == '1' bit mascara = ‘1’ and elemento list(X)== ’0’
-->           Filter_Test = 0 #Detecta error

Sumar al contador de errores de filtros
```

Fig. 22. Lógica en test de filtros. Fuente [Propia]

Para comprobar que el test de filtros funciona, se ha comprobado manualmente con 2 filtros diferentes. Para comprobar manualmente que funciona, se ha cambiado el filtro en el fichero “canfdlib2.py”, en la línea 111 dónde se aplican los filtros y las máscaras.

El filtro se puede aplicar en el mensaje. En este proyecto se aplica en el identificador del mensaje. Como se ha explicado anteriormente, el identificador es parte del mensaje de una trama de CANFD.

Finalmente, se inicia el test “loopbackMode”, y se guardan los resultados en un fichero, “Data.txt”. Se dejan pasar unos segundos para tener diferentes pruebas y se compara, uno a uno, si el filtro y la máscara funcionan según la dirección de los mensajes que se ha obtenido. Si el filtro funciona correctamente sólo deja pasar esos mensajes que tienen una serie de bits en el identificador y la máscara configura cuáles de esos bits de los filtros son importantes.

En el siguiente esquema se muestra un ejemplo para entender cómo funciona:

Identidad	0x	0	1	0	1	0	0
Filtro	0x	0	1	1	0	1	0
Mascara	0x	0	0	0/**1**	0	0	1
		Máscara 0, bit NO significativo	Máscara 0, bit NO significativo		Máscara 0, bit NO significativo	Máscara 0, bit NO significativo	Máscara 1, bit significativo. Bit Filtro == Bit dirección

Fig. 23. Tabla explicación de filtros y máscara. Fuente [Propia]

En el ejemplo anterior, en la *Fig. 23*, si el 3er bit de la máscara es 0, el mensaje es válido, ya

que el identificador es compatible con el filtro, el último bit coincide. Ese filtro con esta máscara sólo permite escribir esos mensajes que tienen identificadores pares, cuando el último bit es 0.

Si el 3er Bit de la máscara es 1, el mensaje no es escrito, ya que el 3er bit del filtro es diferente que el de el identificador del mensaje.

6.1.4.1. Pruebas del Filtro y la Máscara

Filtro 1 y máscara 1

En la primera prueba se utilizará el filtro 0x0000000000 y la máscara 0x0000000001. Esta sólo dejará enviar y recibir los mensajes con un identificador par.

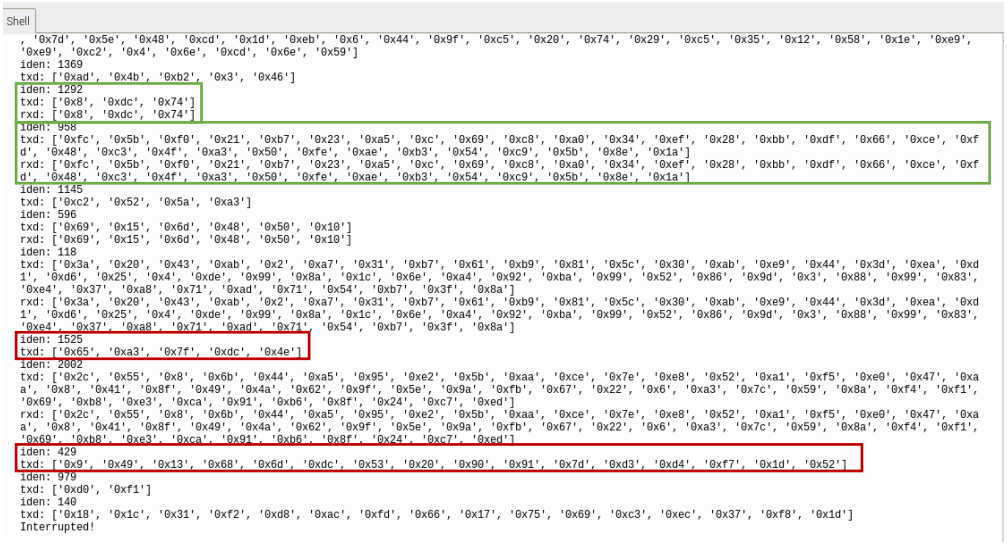


Fig. 24. Registro test 7 filtro 1. Fuente [Propia]

En la *figura 24*, se puede observar los resultados de este test, tenemos la identidad, el txd que es el mensaje que se quiere enviar, y el rxd, el mensaje que se recibe.

La identidad es el atributo que filtra, es decir, solo recibiremos esos mensajes con una identidad par, esas identidades que tienen el último bit 0. Se han marcado 2 de las identidades pares en verde, y 2 de las identidades impares en rojo en la *figura 19*. Dos ejemplos de cuando el filtro funciona, enviamos y recibimos, y cuando no, solo enviamos el mensaje “txd” pero no lo recibimos “rxd”.

Filtro 2 y máscara 2

En la primera prueba se utilizará el filtro 0x0000000101 y la máscara 0x0000000101. Esta combinación dejará pasar los mensajes con identificador que tienen los últimos bits 1X1, cla X puede ser 0 y 1.

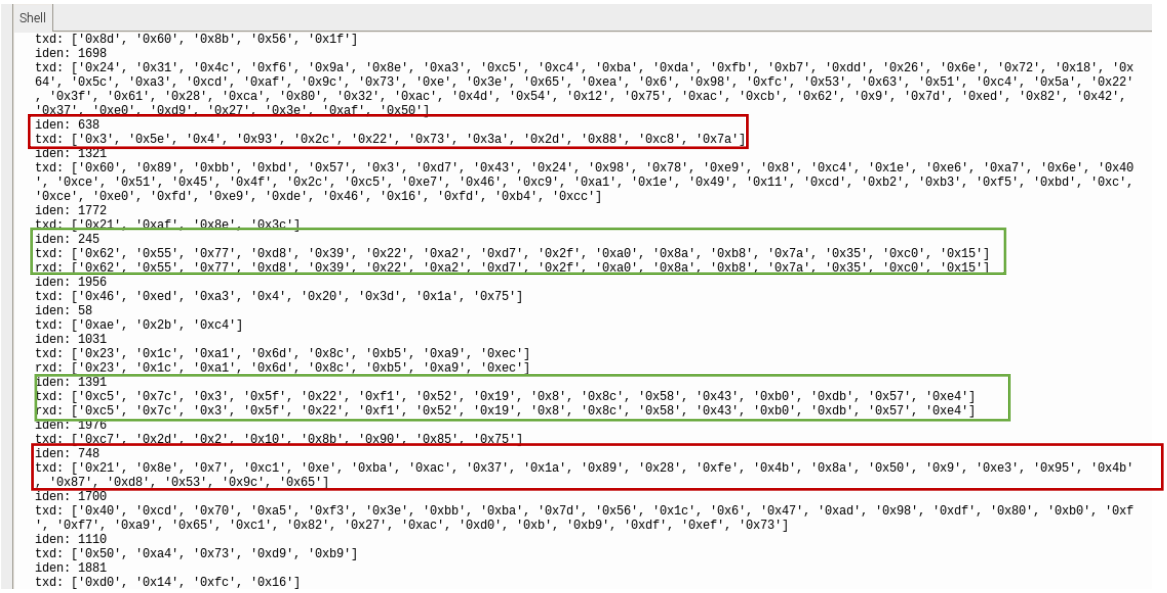


Fig. 25. Registro test 7 filtro 2. Fuente [Propia]

En la *Fig. 25*, se puede observar los resultados de este test, tenemos la identidad, el txd que es el mensaje que se quiere enviar, y el rxd, el mensaje que se recibe.

La identidad es el atributo que filtra, es decir, solo recibiremos esos mensajes con una identidad par, esas identidades que tienen los últimos bits 101 o 111. Se han marcado 2 de las identidades pares que pasan el filtro en verde, y 2 de las identidades que no pasan el filtro en rojo en la *Fig. 25*.

Identidades que pasan el filtro: **245** → 0b11110101 y **1391** → 0b10101101111

Identidades que no pasan el filtro: **638** → 0b1001111110 y **748** → 0b1011101100

Dos ejemplos de cuando el filtro funciona, enviamos y recibimos, y cuando no, solo enviamos el mensaje “txd” pero no lo recibimos “rxd”.

Finalmente se ejecutó el test 7 durante 1 hora para poder observar si ocurría algún error no previsto, el resultado de este está representado en la *figura 26*.

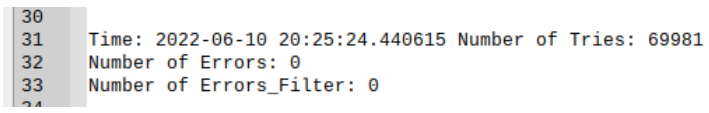


Fig. 26 Resultados test 7. Fuente: [Propia]

7. Funciones con verificación de redundancia cíclica [CRC]

En este proyecto utilizaremos el algoritmo CRC-16 con una tabla. Esta tabla, denominada “Lookup table”, es una lista que reemplaza tiempo de cálculo con una operación de indexación. Esta operación optimiza el tiempo de cálculo del algoritmo [9].

7.1. Instrucciones CRC

7.1.1. Algoritmo utilizado e introducción del CRC en el controlador MCP-2517FD

El MCP2517FD utiliza el algoritmo CRC-16/USB para calcular el CRC. En este caso utilizaremos una función para calcular el CRC con una “lookup table”, para optimizar el código y utilizando el driver con C++ como base.

Con la placa base MCP2517FD al escribir un mensaje con la instrucción CRC, se envía y registra en la RAM, en paralelo, se calcula el código CRC. Con el resultado de esta operación se compara si hay diferencias para saber si hay algún error detectado. En caso que se detecte un error, el CRC.CRCERRIF es activado, y una interrupción es generada.

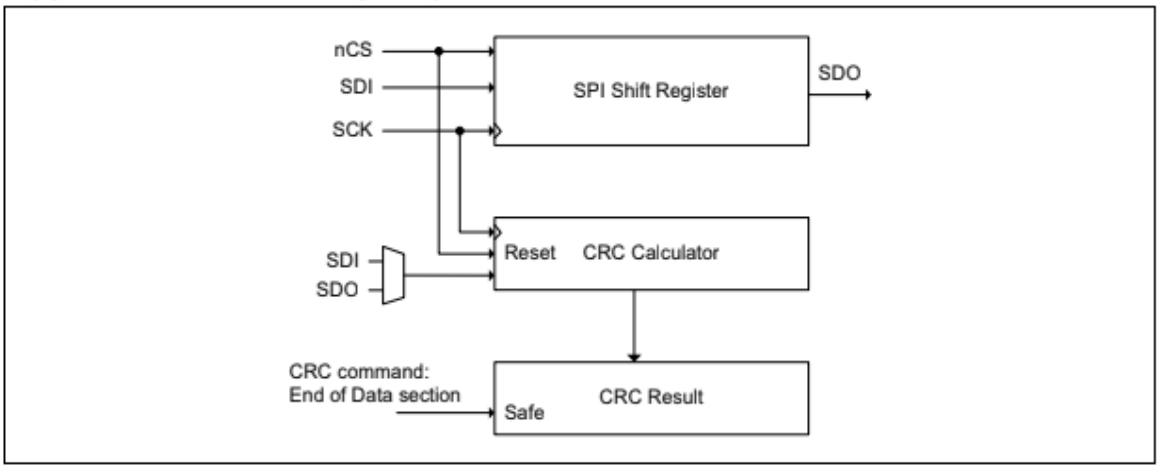


Fig. 273: Esquema de cómo funciona el CRC. Fuente: [1].

Este bit es el último de la dirección 0xE0A, cómo se puede observar en la siguiente tabla. Este bit se debe leer para saber si la instrucción ha funcionado correctamente.

MCP-2517FD Register Summary

Address	Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
E03	OSC	31:24	—	—	—	—	—	—	—	—
E02		23:16	—	—	—	—	—	—	—	—
E01		15:8	—	—	—	SCLKRDY	—	OSCRDY	—	PLLRDY
E00(1)		7:0	—	CLKODIV<1:0>		SCLKDIV	—	OSCDIS	—	PLLEN
	IOCON	31:24	—	INTOD	SOF	TXCANOD	—	—	PM1	PM0
		23:16	—	—	—	—	—	—	GPIO1	GPIO0
		15:8	—	—	—	—	—	—	LAT1	LAT0
E04		7:0	—	XSTBYEN	—	—	—	—	TRIS1	TRIS0
	CRC	31:24	—	—	—	—	—	—	FERRIE	CRCERRIE
		23:16	—	—	—	—	—	—	FERRIF	CRCERRIF
		15:8	CRC<15:8>							
E08		7:0	CRC<7:0>							
	ECCCON	31:24	—	—	—	—	—	—	—	—
		23:16	—	—	—	—	—	—	—	—
		15:8	—	PARITY<6:0>						
E0C		7:0	—	—	—	—	—	DEDIE	SECIE	ECCEN
	ECCSTAT	31:24	—	—	—	—	ERRADDR<11:8>			
		23:16	ERRADDR<7:0>							
		15:8	—	—	—	—	—	—	—	—
E10		7:0	—	—	—	—	—	DEDIF	SECIF	—

Note 1: The lower order byte of the 32-bit register resides at the low-order address.

Fig.28. Registro SFR. Fuente: [1].

En la figura siguiente se puede observar la instrucción del *write/read_CRC*, con los bytes respectivos de la dirección, de los datos que queremos transmitir y el CRC.

FIGURE 4-11: MESSAGE MEMORY READ WITH CRC INSTRUCTION

nCS Low	0b1011	A<11:0>	N<7:0>	DW[A]				CRC<15:8>	CRC<7:0>	nCS High
				DB[A]	DB[A+1]	DB[A+2]	DB[A+3]			

FIGURE 4-12: MESSAGE MEMORY WRITE WITH CRC INSTRUCTION

nCS Low	0b1010	A<11:0>	N<7:0>	DW[A]				CRC<15:8>	CRC<7:0>	nCS High
				DB[A]	DB[A+1]	DB[A+2]	DB[A+3]			

Fig. 29. Instrucciones read/write en la RAM con CRC Fuente: [1].

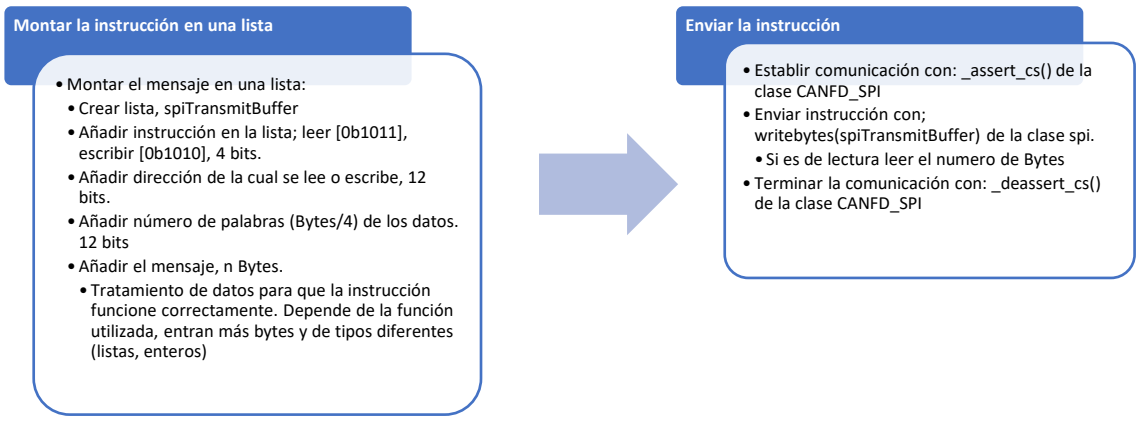


Fig. 30. Diagrama de bloques de la instrucción. Fuente: [Propia]

En esta estructura de los datos que enviamos tanto para leer como para escribir, podemos observar que tenemos que añadir el número de bytes que tiene el mensaje en la lista de bytes que se envía al controlador. Se describe en el siguiente diagrama de bloques:

7.1.2. Funciones con CRC

Para las funciones CRC utilizaremos la misma estructura de las funciones anteriores, añadiendo el número de bytes que se transmiten y el CRC. Después se comprueba si el CRC está bien, tanto al leer los datos como al escribir estos en la memoria, tanto de la RAM cómo de la SFR.

Seguidamente se explicarán con diagramas cómo funcionan estas funciones, separando las funciones de escritura con las de lectura, ya que funcionan de manera distinta.

7.1.2.1. Funciones para escribir con CRC

En consecuencia de añadir el CRC, muchos de los mensajes no son múltiples de 4. Entonces, se añade el tratamiento del mensaje de la función *writeByte() y writeByteArray()* en estas funciones CRC para escribir. Una vez se ha escrito el mensaje, se comprueba que se ha escrito en la memoria RAM o la SFR, utilizando un bit, CRC.CRCERRIF, [0xE0A].

El siguiente diagrama de bloques describe el funcionamiento del tratamiento de los datos para que sean múltiples de 4 y la comprobación de si se ha escrito correctamente en la memoria.

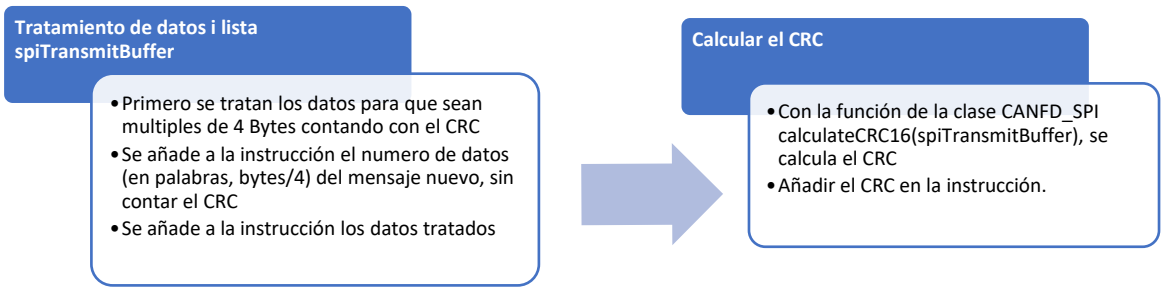


Fig. 31. Diagrama de bloques del código del CRC de la instrucción. Fuente: [Propia]

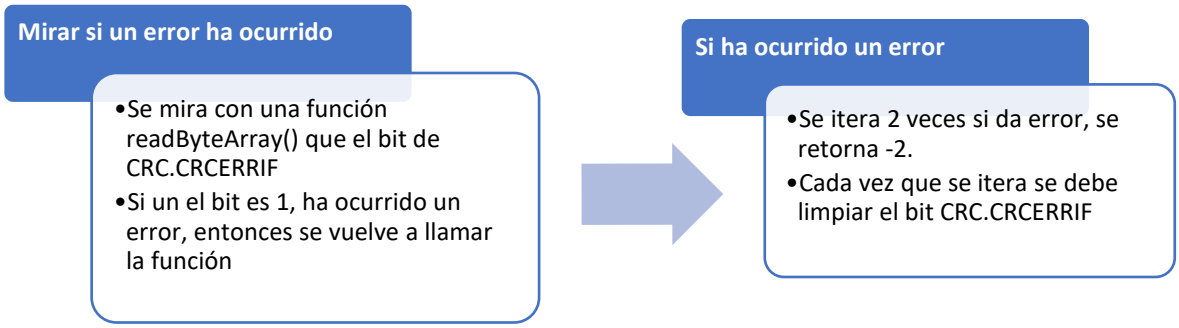


Fig. 32. Diagrama de bloques del código de comprobación del CRC. Fuente: [Propia]

Cuando se escribe el CRC en la instrucción, se debe observar que se escribe invertido. Esta característica se puede observar en la instrucción, *Fig. 29*.

Un ejemplo de esta aplicación. Si el CRC obtenido es 0xb820, se debe escribir en el Buffer, primero el 0x20 y después el 0xb8.

7.1.2.2. Funciones para leer con CRC

En las funciones de leer la memoria también ha habido cambios, gracias a añadir el CRC y al tener que comprobar si el mensaje recibido es correcto.

Para leer también se debe saber si los datos que se quieren leer tiene diferentes números de Bytes que los datos originales que se ha escrito en la memoria, si ésta no era múltiple de 4 Bytes, así que se ha escrito un tratamiento para corregir la longitud y saber cuáles son los números CRC y el mensaje, y no tener errores en este punto.

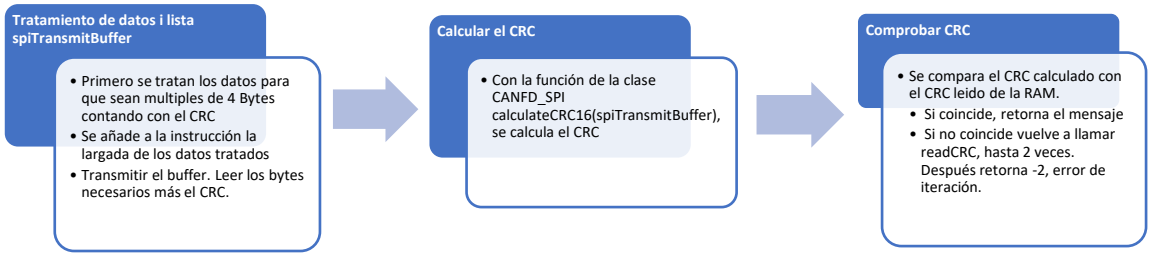


Fig. 33. Diagrama de bloques de la instrucción y cálculo del CRC. Fuente: [Propia]

7.1.2.3. Funciones con CRC y WriteSafe

El controlador MCP-2517FD tiene otra instrucción para escribir en la memoria con el CRC. Esta se llama *Safe*. Las funciones *Safe* aseguran que el mensaje se escriba en la memoria correctamente.

El método de funcionamiento de las funciones Safe es el mismo que las otras, explicada anteriormente en la Fig. 16, pero se diferencian en 3 características que se resumen en:

- Se asegura que el mensaje este escrito correctamente: No tendrán las limitaciones de iteraciones.
- La instrucción es ligeramente diferente, no se escribe la largada del mensaje.
- Son mensajes de una palabra, **4 bytes**.

Message memory Write Safe

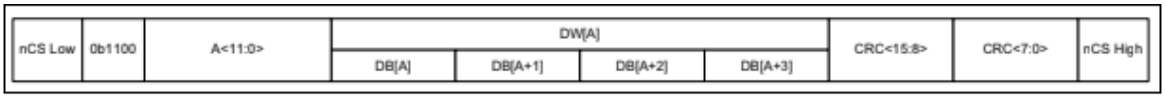


Fig. 34. Instruicción write_Safe en la RAM. Fuente [1]

Como podemos observar en la *Fig. 34*, esta instrucción tiene las características mencionadas anteriormente.

Entonces la lógica de la función quedaría de la siguiente manera:

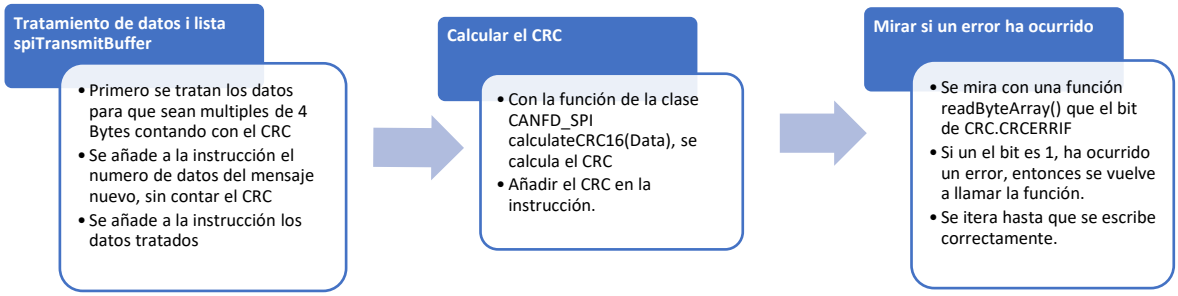


Fig. 35. Diagrama de bloques de la instrucción. Fuente: [Propia]

7.1.2.4. Función de cálculo del CRC: calculateCRC16()

Para utilizar el CRC, se ha descrito el método de cálculo en la teoría y que algoritmo se utilizará para esta función, *pag.* 16. Tenemos el CRC-16 con un “LookUpTable”, el método de cálculo para optimizar este. Cómo se ha explicado la lógica de estas funciones en la teoría, pág. 16., en este caso procederemos con el código en Python.

Esta función pertenece a la clase CANFD_SPI y está escrita en el fichero “canfd.py”.

Inicialmente, se ha utilizado la tabla que está en el fichero de “constantes.py”, aunque en el futuro surjan errores y se testeen otras “LookUpTables” para descartar y probar posibles errores.

Esta “LookUpTables” es la que se obtiene con el algoritmo **CRC-16/USB (0x8005)**, con el valor inicial y final 0xFFFF, con un RefIn y RefOut es igual a True. El polinomio utilizado para calcular el CRC es el 0x8005.

El RefIn y RefOut es una descripción de los parámetros utilizados para calcular el CRC, el RefIn significa que al calcular el CRC cada byte es “reflectado”. El RefOut significa que al final del último Xor, el valor del CRC también es reflectado.

Cada tipo de algoritmo CRC que utiliza las denominadas “LookUpTable” utilizan estos parámetros.

Estos parámetros están en la siguiente tabla:

Algorithm	Result	Poly	Init	RefIn	RefOut	XorOut
CRC-16/USB	0xXXXX	0x8005	0xFFFF	true	true	0xFFFF

Fig. 36. Tabla con los parámetros del CRC-16/USB. Fuente: [Propia]

Y la “LookUpTable” utilizada para este algoritmo es la *CRC16/USB Table* [1], del annexo.

7.1.3. Funciones readHalfByte and writeHalfWord

Para el desarrollo del CRC, en el driver original programado por C++, se utilizan estas funciones para leer el Bit CRCERRIF, la señal de error de las instrucciones con CRC, y utilizando como referencia este driver, las cuales retornan media palabra, 2 Bytes.

Se ha utilizado la misma estructura básica para las funciones sin CRC, ya que tienen la misma instrucción, como se muestra en la *Fig. 7*.

Estas 2 funciones se han añadido en la librería sin hacer pruebas exhaustivas, además que sólo están programadas para leer y escribir en la Memoria SFR, excluyendo cualquier aplicación de la memoria RAM. Esto se debe a qué solo se utilizan en las funciones CRC para leer el Bit de error [RCERRIF] y más adelante se cambiaron estas dos funciones por el writeByteArray() y readByteArray().

7.2. Pruebas de las funciones con CRC

Para comprobar las funciones CRC funcionen correctamente, se ha aplicado las pruebas exhaustivas diseñadas para las funciones *readByte(), readWord(), readByteArray() y readWordArray()* del apartado 6, a estas funciones con alguna modificación.

7.2.1. Test 8

También se añadirán las funciones *writeByteArraySafe() y writeWordSafe(),* teniendo cada uno su contador de errores. La lógica de este test seguirá el mismo que el Test 3. Se crearán mensajes de longitud aleatoria y se escribirá en la RAM para la comprobación de estas funciones. Seguidamente se lee el mensaje en la misma dirección y se comprueba si el CRC es correcto y no ha ocurrido ningún error.

La lógica en diagrama de bloques es:

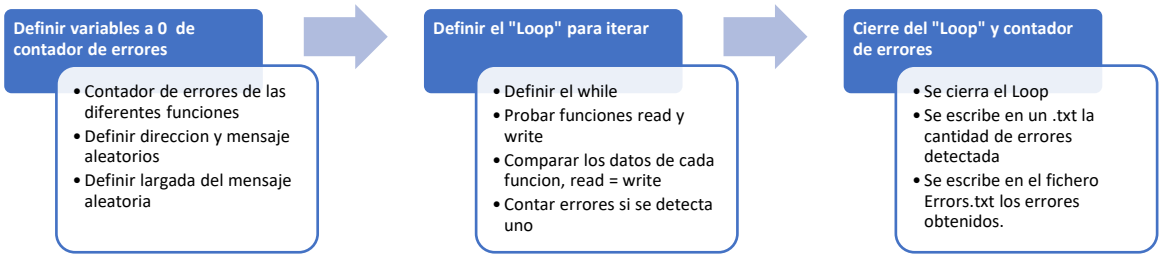


Fig. 37. Diagrama de bloques del test 8. Fuente: [Propia]

Este sería la primera prueba para observar si las funciones funcionan correctamente, ya que se puede observar si, en la globalidad, las funciones trabajan correctamente.

En una primera prueba inicial, éste funcionaba correctamente a excepción de las funciones con la instrucción Safe.

Al ser un testeo que escribes y lees con las 2 funciones que estás probando simultáneamente, se debe separar las funciones de alguna manera para encontrar algún error.

El test tiene un **registro de fallos**, cuando alguna función de este falla se escribe en el fichero “ErrorsCRC.txt”. Cuando se interrumpe la ejecución del test, se escriben el número de errores que ha obtenido y el número de intentos que se ha hecho de cada uno de las funciones. Este registro es importante para observar en cuál de los datos ha fallado y el número de este, se ha utilizado por la detección de errores, la depuración y la programación de las funciones CRC y CRC_Safe.

7.2.1.1. Test 8.1

Con este objetivo de intentar encontrar un fallo en el funcionamiento de las funciones, se ha planteado un test para hacer que el CRC sea diferente y ver que si salta el bit de error (CRCERRIF).

El planteamiento de este test es cambiar el CRC para que este sea erróneo y ver si escribe el mensaje en la memoria del controlador. Para esta función añadiremos en las **funciones** una línea que usualmente estará comentada. Ésta suma un 1 al CRC, dando como consecuencia el error que necesitamos, para atacar al funcionamiento de las funciones con CRC. Esta modificación se ha hecho en cada una de las funciones con CRC, para probar que todas funcionen.

Al hacer este test se observó que las funciones con CRC escribían en la memoria los mensajes, aunque el CRC no fuera correcto. De la función de lectura no se obtuvo ningún error. De este test se pudo observar y entender más cómo funciona el CRC y la instrucción de lectura y escritura.

Proceso de desarrollo

En este apartado se verá un poco más detallado el proceso de desarrollar las funciones CRC, explicando los errores que se obtuvieron, el estudio detrás del desarrollo y las comprobaciones hasta llegar al driver actual. Este proceso de detección de errores ha requerido una gran cantidad de tiempo, por el estudio y las pruebas hechas por entender donde se está el error.

1er Error : Las funciones CRC se escriben aunque el CRC no es correcto.

Gracias a este test se observaron fallos que no detectaban, uno de estos fue la escritura del mensaje sin tener un CRC correcto. Este error se debió a 4 características de las instrucciones del CRC que son:

1. El cálculo del CRC, debe ser de la instrucción y de los datos que se desean escribir. El código es el siguiente: self.calculateCRC16(**spiTransmitBuffer**)

2. En la instrucción, el número escrito N (8 bits), *figura 18*, después de escribir la dirección es el número de palabras (4 Bytes) NO el número de Bytes de los datos.

$$\text{Numero de Bytes} = \text{Numero de palabras} * 4$$

3. Se comprobó que la función **calculateCRC16(**List**),** funciona correctamente y se

obtuvo un mayor conocimiento de los algoritmos CRC. Fuente [XX stack overflow] Este estudio se hizo por la duda de si el CRC calculado por la función, era el mismo CRC que el controlador calculaba.

Este error también se pudo analizar, al poner un impreso (print) de cuantas veces intentaba escribir la función es específico. Con esta información se pudo observar que en el estado inicial este intentaba escribir 3 veces, el límite establecido por el driver en C++.

2no Error: Algunas de las funciones Safe no escriben.

En el primer estado de estas funciones se pudo observar gracias a el test 8 y su registro en el "ErrorsCRC.txt", que algunas de estas funciones no escribían correctamente.

Este error, se observó fácilmente en el registro que sólo ocurría en las funciones que intentaban escribir un mensaje de más de 4 Bytes. Es decir, la **función Safe solo permite escribir una palabra**.

7.2.1.2. Test 8.2

Finalmente, se comprueba la eficacia de las funciones CRC al leer y escribir el mensaje. Esta comprobación se ejecutará produciendo mucho ruido electrónico para qué produzca cambios de bits en el controlador o durante la transmisión de datos mientras los test 3 y 8 se están ejecutando ya que este test tiene el objetivo de comparar los errores que se obtienen de las funciones CRC y sin las funciones CRC, cuando hay mucho ruido electrónico.

Este ruido electrónico se intentó producir encendiendo un taladro al lado del controlador MCP-2517FD. Y se hizo una prueba de 5 minutos para ver la cantidad de errores que se obtenían ejecutando el test. El resultado de esta primera prueba no tuvo resultados, significativos y satisfactorios, ya que los dos obtuvieron la misma cantidad de errores, considerando el orden de magnitud de estos resultados.

En consecuencia, se tuvo que intentar hacer más ruido electrónico, el suficiente ruido para poder cambiar los bits durante la transmisión de los mensajes y el tiempo de almacenamiento del controlador. Este aumento del ruido electrónico se ha conseguido pegando un imán en el taladro y encendiendo este para producir una variación del campo electromagnético mayor al anterior.

Con esta solución rudimentaria, se ha podido probar la eficacia de las funciones CRC. Ya que se han obtenido algunos errores más que sólo encendiendo el taladro.

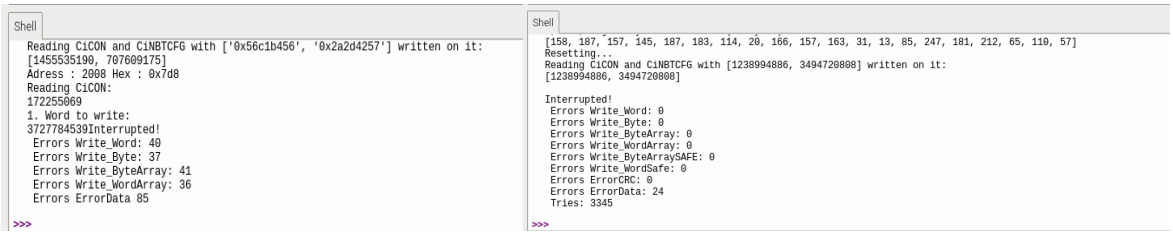


Fig. 38. Resultados test sin CRC y test con CRC . Fuente: [Propia]

Cómo se puede observar en la *Fig. 38*, al ejecutar los test 3 y 8 durante 5 minutos, se obtienen más errores en el test de las funciones sin CRC, relacionados con los datos leídos y enviados, equiparado con el test 8 con funciones con CRC.

El número de intentos del test 3 es de 5319 con los mismos 5 minutos, ya que tiene 2 funciones menos para comprobar.

Con este resultado, concluimos que para que las funciones CRC no escriban o lean el mensaje correctamente, se necesita una cantidad de ruido electrónico elevado y se cumple el propósito de las funciones con CRC, evitar errores a causas externas.

7. Presupuesto económico

El presupuesto económico de este proyecto es una aproximación del precio de este proyecto. Se ha dividido el presupuesto en 2 esquemas, las horas invertidas en este y el presupuesto material y total.

En este caso, las horas invertidas en este proyecto se pueden dividir en 5 etapas:

- **Investigación:** 40 horas
- **Montaje** de la Raspberry Pi 3B y configuración Linux**:** 10 horas
- **Programando el driver**
  - Programando: 30 horas
  - Testeando: 20 horas
  - Corrigiendo errores (Debugging): 150 horas
- **Elaboración de la memoria:** 50 horas

Con un total de 270 horas invertidas en este proyecto, al calcular el presupuesto, se cobrarán estas a sueldo de estudiante en prácticas. En el caso de la ETSEIB, se pagan 10 euros la hora brutos. Que a la empresa le cuesta casi el doble, 18.

En respecto al material, se contará el material utilizado por realizar el proyecto. Se contará el Windows y el Word para hacer el informe

Elemento	Cantidad	Precio [€]
Raspberry Pi 3B	1	60
MCP-2517FD Click	1	25
Monitor	1	200
Teclado	1	50
Ratón	1	20
Cable HDMI	1	6
Taladro	1	43
Software: Word (Mensual) y Windows	1	28
Horas	300	18
Total		**5289**

Fig. 39. Presupuesto. Fuente [Propia]

8. Impacto ambiental

El trabajo se ha realizado con 2 ordenadores, una Raspberry Pi 3B y un Windows, por lo tanto, se realizará el impacto ambiental contando el material e impacto energético de estos, cuando se ha utilizado para desarrollar este proyecto.

El gasto energético que tienen estos son:

- Raspberry Pi 3B: En estrés máximo 6,4 W [15]
- PC con Windows: Media de unos 75 W [Fuente: Propia]

Con esta potencia consumida, y utilizando los datos del presupuesto para saber cuántas horas se ha utilizado cada dispositivo, nos sale un gasto de 1344 Wh de la Raspberry y 6750 Wh, con un total de 8094 Wh.

Con este gasto total de energía, y utilizando el índice del año 2020 de 250 gCO_2/KWh del "Mix Elèctric" elaborado por "l'Oficina del canvi climàtic" [16], se obtiene que se ha producido un total de 2023,5 gramos de CO_2. Uno de los factores más importantes al hacer el impacto ambiental.

A contraparte, los dispositivos al ser producidos también tienen un impacto ambiental. Esta huella de carbono es de aproximadamente 270 Kg por el PC windows, sin contar el transporte de este y sus diferentes componentes. [18]

La huella de carbono de esta la Raspberry es mucho menor, ya que es más pequeña y está producida en UK. [17]

Una vez realizado el proyecto, se puede concluir que se han alcanzado los objetivos establecidos al principio del proyecto de manera satisfactoria. Se ha hecho unos test exhaustives para el driver CANFD del estudiante Albert Burgués [11], y se han ampliado las funcionalidades de este

9. Conclusiones

Una vez realizado el proyecto, se puede concluir que se han alcanzado los objetivos establecidos al principio del proyecto de manera satisfactoria. Se han hecho unos test exhaustivos para el driver CANFD del estudiante Albert Burgués [11], y se han ampliado las funcionalidades de este con las funciones CRC con los test necesarios para asegurar que funcionan correctamente.

También se ha consolidado los conocimientos de Python y se ha aprendido a trabajar con Linux y un periférico, el MCP-2517FD. Se ha trabajado como programador durante unos meses y se ha aprendido un método de trabajar relacionado con la lógica y utilizando el manual del controlador [1], que ha sido la base de este proyecto. También se ha comprendido un poco el lenguaje de C++ y como funciona este lenguaje de programación.

Los resultados del proyecto, han cumplido con el deseado, ya que se han utilizado unos test exhaustivos para todas las funciones y estas han pasado los test correctamente y cuando estos debían fallar, las funciones se comportaban de la manera prevista, con los errores esperados.

Los test 1, 2 y 3 han funcionado correctamente, probando con diferentes parámetros estas funciones.

El test 7, ha probado que el filtro y la máscara funcionan, comprobando automáticamente el mensaje.

El test 8 es la conclusión de que las funciones añadidas en la clase CANFD_SPI, están correctamente programadas, buscando el fallo en las funciones para corregirlos, test 8.1, y comparando el test 8 con el test 3, para demostrar así una mayor eficacia y seguridad cuando se desea escribir datos en el controlador, test 8.2.

Para finalizar, en proyectos futuros el driver puede seguir mejorando. Estas mejoras pueden ser, hacer un driver más óptimo limpiando el código y programarlo con una mayor optimización que en consecuencia tendría una mejor eficiencia y más velocidad de cómputo. En otro bando, se pueden implementar más funcionalidades.

10. Agradecimientos

Como último quiero transmitir mi agradecimiento a todos aquellos que me han ayudado a lo largo de esta etapa y han colaborado en esta investigación.

En primer lugar, a mi tutor Manuel Moreno Eguílaz, por su ayuda en la planificación, información y organización en este Trabajo de Fin de Grado y el interés en el TFG, y el estado de este durante todos los meses que se ha estado produciendo.

En segundo lugar, a mi familia que han estado a lo largo de toda mi carrera apoyándome en todo momento y animándome a seguir adelante.

Finalmente, expreso mi agradecimiento a la Universidad por todo el período de aprendizaje y crecimiento personal que ha conllevado hacer la carrera de ingeniería Industrial.

11. Bibliografía

Referencias bibliográficas

[1] *MCP2517fd Datasheet*

[2] *https://microcontrollerslab.com/introduction-to-spi-communication-protocol/*

[3] *https://www.analog.com/en/analog-dialogue/articles/introduction-to-spi-interface.html*

[4] *https://es.wikipedia.org/wiki/Serial_Peripheral_Interface#Ventajas*

[5] *https://oravatec.com/blog/que-es-el-bus-can-fd/*

[6] *https://www.csselectronics.com/pages/can-fd-flexible-data-rate-intro*

[7] *https://hmong.es/wiki/Cyclic_Redundancy_Check*

[8] *crccalc.com*

[9] *https://en.wikipedia.org/wiki/Cyclic_redundancy_check*

[10] *http://www.arumeinformatica.es/blog/los-formatos-big-endian-y-little-endian/*

[11] *https://upcommons.upc.edu/handle/2117/332952?show=full*

[12] *https://upcommons.upc.edu/bitstream/handle/2117/177412/tfm-can-fd-sniffer-joaquin-cortes.pdf?sequence=1&isAllowed=y*

[13] *https://www.microchip.com/en-us/product/MCP2517FD*

[14] *https://www.google.com/search?q=raspberry+pi+3b+pins&tbm=isch&ved=2ahUKEwjB2Zb9yZn6AhXQgc4BHR9RAcoQ2-cCegQIABAA&oq=raspberry+pi+3b+pins&gs_lcp=CgNpbWcQAzIECCMQJzIICAAQHhAIEBMyCAgAEB4QCBATMggIABAeEAgQEzIICAAQHhAIEBNQ0QIY0Qlg0BJoAHAAeACAAZMBiAGDApIBAzAuMpgBAKABAaoBC2d3cy13aXotaW1nwAEB&sclient=img&ei=kI8kY8HROtCDur4Pn6KF0Aw&bih=757&biw=1440&rlz=1C1GCEU_esES1023#imgrc=h2QbOcLztS7qgM*

[15] *https://bugeados.com/raspberry/cual-es-el-consumo-de-una-raspberry-pi-3-4/*

[16] *https://canviclimatic.gencat.cat/ca/actua/factors_demissio_associats_a_lenergia/*

[17] *https://forums.raspberrypi.com/viewtopic.php?t=77461*

[18] *https://indexdesarrollo.com/eficiencia-de-los-ordenadores-cuanto-contamina-pc/*

12. Annex

- *CRC16/USB Table* [1]*:*

```
0x0000 0xc0c1 0xc181 0x0140 0xc301 0x03c0 0x0280 0xc241
0xc601 0x06c0 0x0780 0xc741 0x0500 0xc5c1 0xc481 0x0440
0xcc01 0x0cc0 0x0d80 0xcd41 0x0f00 0xcfc1 0xce81 0x0e40
0x0a00 0xcac1 0xcb81 0x0b40 0xc901 0x09c0 0x0880 0xc841
0xd801 0x18c0 0x1980 0xd941 0x1b00 0xdbc1 0xda81 0x1a40
0x1e00 0xdec1 0xdf81 0x1f40 0xdd01 0x1dc0 0x1c80 0xdc41
0x1400 0xd4c1 0xd581 0x1540 0xd701 0x17c0 0x1680 0xd641
0xd201 0x12c0 0x1380 0xd341 0x1100 0xd1c1 0xd081 0x1040
0xf001 0x30c0 0x3180 0xf141 0x3300 0xf3c1 0xf281 0x3240
0x3600 0xf6c1 0xf781 0x3740 0xf501 0x35c0 0x3480 0xf441
0x3c00 0xfcc1 0xfd81 0x3d40 0xff01 0x3fc0 0x3e80 0xfe41
0xfa01 0x3ac0 0x3b80 0xfb41 0x3900 0xf9c1 0xf881 0x3840
0x2800 0xe8c1 0xe981 0x2940 0xeb01 0x2bc0 0x2a80 0xea41
0xee01 0x2ec0 0x2f80 0xef41 0x2d00 0xedc1 0xec81 0x2c40
0xe401 0x24c0 0x2580 0xe541 0x2700 0xe7c1 0xe681 0x2640
0x2200 0xe2c1 0xe381 0x2340 0xe101 0x21c0 0x2080 0xe041
0xa001 0x60c0 0x6180 0xa141 0x6300 0xa3c1 0xa281 0x6240
0x6600 0xa6c1 0xa781 0x6740 0xa501 0x65c0 0x6480 0xa441
0x6c00 0xacc1 0xad81 0x6d40 0xaf01 0x6fc0 0x6e80 0xae41
0xaa01 0x6ac0 0x6b80 0xab41 0x6900 0xa9c1 0xa881 0x6840
0x7800 0xb8c1 0xb981 0x7940 0xbb01 0x7bc0 0x7a80 0xba41
0xbe01 0x7ec0 0x7f80 0xbf41 0x7d00 0xbdc1 0xbc81 0x7c40
0xb401 0x74c0 0x7580 0xb541 0x7700 0xb7c1 0xb681 0x7640
0x7200 0xb2c1 0xb381 0x7340 0xb101 0x71c0 0x7080 0xb041
0x5000 0x90c1 0x9181 0x5140 0x9301 0x53c0 0x5280 0x9241
0x9601 0x56c0 0x5780 0x9741 0x5500 0x95c1 0x9481 0x5440
0x9c01 0x5cc0 0x5d80 0x9d41 0x5f00 0x9fc1 0x9e81 0x5e40
0x5a00 0x9ac1 0x9b81 0x5b40 0x9901 0x59c0 0x5880 0x9841
0x8801 0x48c0 0x4980 0x8941 0x4b00 0x8bc1 0x8a81 0x4a40
0x4e00 0x8ec1 0x8f81 0x4f40 0x8d01 0x4dc0 0x4c80 0x8c41
0x4400 0x84c1 0x8581 0x4540 0x8701 0x47c0 0x4680 0x8641
0x8201 0x42c0 0x4380 0x8341 0x4100 0x81c1 0x8081 0x4040
```