MPX-2515 Programmer's Guide

Commell MPX-2515 CAN 2.0B USB card features USB 2.0 to CAN 2.0B bus interface in Mini-PCIe form factor. This MPX-2515 Programmer's Guide provides the Commell CAN Protocol (CCANP), which is the protocol used in between the USB host side and the firmware of the MPX-2515 card. Command packets and response packets are defined and carried through the CCANP. Microsoft Windows programmers use the CCP to request the MPX-2515 firmware to execute the commands defined in the CCANP.

Contents

1 Commell CAN Protocol 3
1.1 Introduction 3
1.2 Interrupt In Packet 4
1.3 Request Packet vs. Response Packet 5
1.4 Defined Constants 7
1.5 CCP_CAN_SYSTEM_SERVICE 8
1.5.1 CAN_GET_INFO 8
1.5.2 CAN_GET_TICKCOUNT 9
1.5.3 CAN_SET_LED_SWITCH 10
1.5.4 CAN_SET_TRANSFER_RATE 10
1.5.5 CAN_RESET_INSTRUCT 11
1.5.6 CAN_READ_INSTRUCT 11
1.5.7 CAN_WRITE_INSTRUCT 12
1.6 CCP_CAN_READ_FMB Service 14
1.6.1 Introduction to Message Reception 14
1.6.2 Firmware Messages Buffer (FMB) 14
1.6.3 CCP_CAN_RED_FMB 15
1.7 CCP_CAN_RX_MASKS_FILTERS Service 18
1.7.1 Introduction to Message Acceptance Filters and Masks 18
1.7.2 MCP_RX_MASK_0 22
1.7.3 MCP_RX_MASK_1 22
1.7.4 MCP_RX_FILTER_0 23
1.7.5 MCP_RX_FILTER_1 23
1.7.6 MCP_RX_FILTER_2 24
1.7.7 MCP_RX_FILTER_3 24
1.7.8 MCP_RX_FILTER_4 25
1.7.9 MCP_RX_FILTER_5 25
1.8 CCP_CAN_TX Service 26
1.8.1 Introduction to Message Transmission 26
1.8.2 MCP_LOAD_TXB0SIDH 27
1.8.3 MCP_LOAD_TXB0D0 28
1.8.4 MCP_LOAD_TXB1SIDH 29
1.8.5 MCP_LOAD_TXB1D0 30
1.8.6 MCP_LOAD_TXB2SIDH 31
1.8.7 MCP_LOAD_TXB2D0 32
1.8.8 MCP_ABORT_TXB0 32
1.8.9 MCP_ABORT_TXB1 33
1.8.10 MCP_ABORT_TXB2 33
1.9 CCP_CAN_BIT_MODIFY Service 34
1.9.1 Introduction to Bit Modify 34
1.9.2 CAN_BIT_MODIFY Command 35
2 Reference 38

1 Commell CAN Protocol

1.1 Introduction

This chapter defines the Commell CAN Protocol (CCANP), which is used to carry CAN bus request packets from the USB host side to the USB device side (MPX-2515). After the MPX-2515 firmware executed the request packet, the USB host side application then retrieves the corresponding response packet from the MPX-2515 firmware.

The MPX-2515 firmware implements four endpoints to facilitate the CCANP and therefore the host applications.

- Endpoint 1 - The endpoint 1 implements the Bulk In transfers. The USB host applications read endpoint 1 to return response packets.

- Endpoint 2 - The endpoint 2 implements the Bulk Out transfers. The USB host applications write request packets to this endpoint in order for the MPX-2515 firmware to carry on the requested command.

- Endpoint 3 - Endpoint 3 implements the Interrupt In transfer. The USB host applications can hence read the interrupt in packet at anytime in order to realize some important status of the MPX-2515 firmware.

- Endpoint 4 - In order to speed up the process on the received CAN messages, an endpoint 4 was implemented to allow the USB host applications to read the Firmware Message Buffer (FMB) at anytime. The MPX-2515 firmware updates this buffer very often. The USB host applications determine the received messages by checking the timestamp and other information incorporated in the header of the FMB.

The following figure shows the transaction types and operation sequences of CCANP.

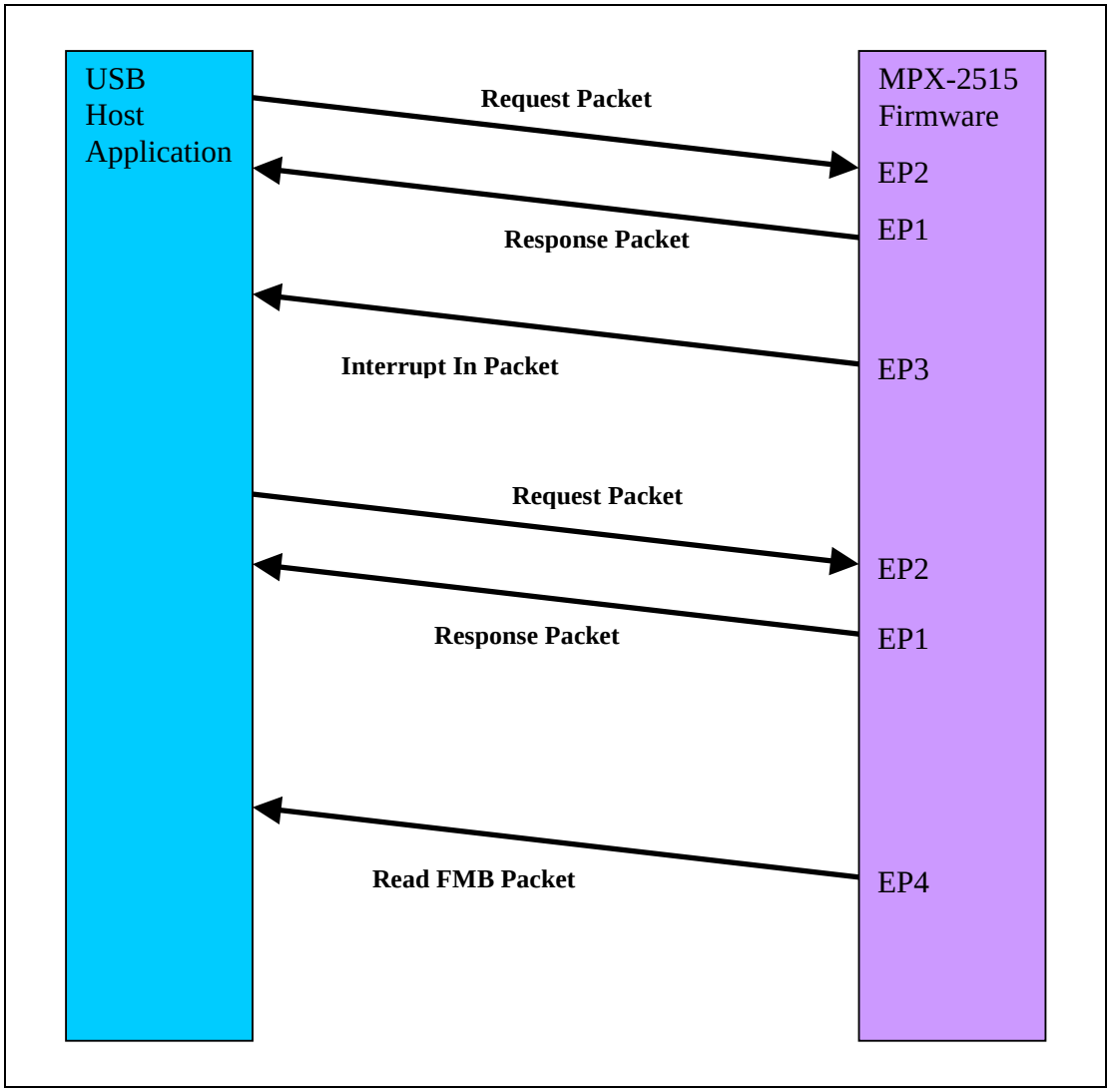

Figure 1 CCANP Transactions and Operation Sequences

A response packet is corresponding to a request packet. And hence USB host applications usually first issue a CCANP Request Packet to the MPX-2515 card via a BULK OUT (at endpoint 2) transfer and then issue a CCP Response Packet to the MPX-2515 card via a BULK IN (at endpoint 1) transfer. The application should always check the Status Code filed in the response packet in order to determine the final status of the corresponding request carried out in the connected CAN bus.

1.2 Interrupt In Packet

The MPX-2515 card provides status packet that the USB host can read out in order to realize the status of the MPX-2515 controller as well as the connecting CAN bus activities.

This status packet can be obtained via issuing an INTERRUPT IN transfer over the endpoint 3 of the MPX-2515 card. This status packet contains eight bytes per packet. This section defines each byte of the status packet.

Please be noted that interrupt in transfers can be issued at anytime. MPX-2515 applications are encouraged to retrieve the status packets often in order to check out the status that the application has concerned.

The following table defines the format of the Status Packet. MPX-2515 application programmers can also refer to the Microchip MCP2515 data sheet for more information.

OFFSET	BYTE	NAME
0	1	READ STATUS INSTRUCTION (return byte) **Bit 7 CANNTF.TX2IF** **Bit 6 TXB2CNTRL.TXREQ** **Bit 5 CANINTF.TX1IF** **Bit 4 TXB1CNTRL.TXREQ** **Bit 3 CANINTF.TX0IF** **Bit 2 TXB0CNTRL.TXREQ** **Bit 1 CANINTFL.RX1IF** **Bit 0 CANINTF.RX0IF**
1	1	RX STATUS INSTRUCTION (return byte): **Bit 7:6 Receive Message** 00: No RX message 01: Message in RXB0 10: Message in RXB1 11: Messages in both buffers **Bit 5 Don't Care** **Bit 4:3 Message Type Received** 00: Standard data frame 01: Standard remote frame 10: Extended data frame 11: Extended remote frame **Bit 2:1:0 Filter Match** 000: RXF0 001: RXF1 010: RXF2 011: RXF3 100: RXF4 101: RXF5 110: RXF0 (rollover to RXB1) 111: RXF1 (rollover to RXB1)
2	1	EFLG - ERROR FLAG **Bit 7 RX1OVR:** Receive Buffer 1 overflow flag bit. **Bit 6 RX0OVR:** Receive Buffer 0 overflow flag bit. **Bit 5 TXBO:** Bus-Off Error flag bit

		Bit 4 TXEP: Transmit Error-Passive flag bit. **Bit 3 RXEP:** Receive Error-Passive flag bit. **Bit 2 TXWAR:** Transmit Error Warning flag bit. **Bit 1 RXWAR:** Receive Error Warning flag bit. **Bit 0 EWARN:** Error Warning flag bit.
3	1	Firmware Message Buffer (FMB) status: **Bit 7:6:5:4:3 Buffer 2 Overwritten Number** This number increases by value of one if a CAN message has arrived and has overwritten at Buffer 2 storage prior to the previous stored has read back by the USB host. **Bit 2 Buffer 2 Indicator** 0 / 1: empty / occupied **Bit 1 Buffer 1 Indicator** 0 / 1: empty / occupied **Bit 0 Buffer 0 Indicator** 0 / 1: empty / occupied
4	1	Reserved.
5	1	Reserved.
6	1	Reserved.
7	1	Reserved.

Table 1 Status Packet Format

1.3 Request Packet vs. Response Packet

This section defines the format of Request Packet as well as the format of Response Packet. At this moment, both Request Packets and Response Packets are limited to 64 bytes per packet.

The Request Packet is used to send command to the CY8C24794 micro controller in the MPX-2515 CAN USB Card to carry out over the connecting CAN bus. A Request Packet is sent to endpoint 2 of the MPX-2515 card through a BULK OUT transfer.

While, the Response Packet is used to retrieve the result of an executed command from the CY8C24794 micro controller. A Response Packet is returned from endpoint 1 of the MPX-2515 card through a BULK IN transfer.

The following table defines the Request Packet used in CCA-P.

OFFSET	NAME	DESCRIPTION
0	HEADER_MSB	0X55
1	HEADER_LSB	0XAA
2	VERSION	0X01
3	SIZEOF_DATA	The number of bytes residing in the DATA field.

4	COMMAND	The command category code.
5	ERROR	0X00
6	DATA_0 or COMMAND_0	Byte 0 of the data field or command_0 byte.
7	DATA_1 or COMMAND_1	Byte 1 of the data filed or command_1 byte.
8	DATA_2 or COMMAND_2	Byte 2 of the data field or command_2 byte.
9	DATA_3 or COMMAND_3	Byte 3 of the data field or command_3 byte.
10 to N	DATA	The DATA field; up to 53 bytes.
N+1	CHECKSUM	2's complement checksum of all fields except this checksum field.

Table 2 CCANP Request Packet Format

The following table defines the format of CCAP Response Packet.

OFFSET	NAME	DESCRIPTION
0	HEADER_MSB	0X66
1	HEADER_LSB	0XBB
2	VERSION	0X01
3	SIZEOF_DATA	The number of bytes residing in the DATA field.
4	COMMAND	The corresponding command category code.
5	ERROR	The error code.
6 to N	DATA	The DATA field; up to 57 bytes.
N+1	CHECKSUM	2's complement checksum of all fields except this checksum field.

Table 3 CCANP Response Packet Format

1.4 Defined Constants

This section defines constants used in MPX-2515 programming. Please be noted that all these constants are defined in C programming language style.

```
/* COMMAND: Service code */
#define CCP_CAN_SYSTEM_SERVICE  0X20
#define CCP_CAN_RX_MASKS_FILTERS  0X22
#define CCP_CAN_TX  0X23
#define CCP_CAN_BIT_MODIFY  0X25
#define CCP_CAN_READ_FMB  0X26

/* DATA_0: sub-command code */
#define CAN_GET_INFO  0X01
#define CAN_GET_TICKCOUNT  0X02
```

```
#define CAN_SET_LED_SWITCH  0X03
#define CAN_SET_TRANSFER_RATE  0X04
#define CAN_RESET_INSTRUCT  0X85

#define MCP_RX_MASK_0  0X20
#define MCP_RX_MASK_1  0X24
#define MCP_RX_FILTER_0  0X00
#define MCP_RX_FILTER_1  0X04
#define MCP_RX_FILTER_2  0X08
#define MCP_RX_FILTER_3  0X10
#define MCP_RX_FILTER_4  0X14
#define MCP_RX_FILTER_5  0X18

#define MCP_LOAD_TXB0SIDH  0X40
#define MCP_LOAD_TXB0D0  0X41
#define MCP_LOAD_TXB1SIDH  0X42
#define MCP_LOAD_TXB1D0  0X43
#define MCP_LOAD_TXB2SIDH  0X44
#define MCP_LOAD_TXB2D0  0X45
#define MCP_ABORT_TXB0  0X38
#define MCP_ABORT_TXB1  0X48
#define MCP_ABORT_TXB2  0X58

/* MCP2515 Register Address */
#define RXM0SIDH  0X20
#define RXM1SIDH  0X24
#define RXF0SIDH  0X00
#define RXF1SIDH  0X04
#define RXF2SIDH  0X08
#define RXF3SIDH  0X10
#define RXF4SIDH  0X14
#define RXF5SIDH  0X18

/* Others */
#define EP1  1 /* Endpoint 1 */
#define EP2  2 /* Endpoint 2 */
#define EP3  3 /* Endpoint 3 */
#define EP4  4 /* Endpoint 4 */
```

1.5 CCP_CAN_SYSTEM_SERVICE

The CCP_CAN_SYSTEM_SERVICE uses 0X20 as its command category service code. This service however contains several sub-services, which are identified by the Sub-Command Codes that is starting at the first byte of the DATA field. This section defines this service.

1.5.1 CAN_GET_INFO

The CAN_GET_INFO command returns the USB vendor ID, product ID, and the firmware version of the targeted MPX-2515 card.

```
#define CCP_CAN_SYSTEM_SERVICE  0X20 /* command category code */
#define CAN_GET_INFO  0X01  /* sub-command code */
#define EP2  2 /* Endpoint 2 */
#define EP3  3 /* Endpoint 3 */
```

OFFSET	BULK OUT (EP2)	BULK IN (EP1)
SIZEOF_DATA	0X01	0X07
COMMAND	0X20	0X20
ERROR	0X00	Error code
DATA_0	0X01	Number of bytes return (0X06)
DATA_1	CHECKSUM	VID_MSB (0XCE)
DATA_2		VID_LSB (0XCE)
DATA_3		PID_MSB (0X25)
DATA_4		PID_LSB (0X15)
DATA_5		Firmware version major
DATA_6		Firmware version minor
DATA_7		CHECKSUM

Table 4 CAN_GET_INFO Format

1.5.2 CAN_GET_TICKCOUNT

The CAN_GET_TICKCOUNT command retrieves the MPX-2515 system tick count since the system get started. Please be noted that each tick count is 100 micro seconds inside MPX-2515 system.

```
/* Each tickcount is 100 micro-second */
#define CCP_CAN_SYSTEM_SERVICE  0X20 /* command category code */
#define CAN_GET_TICKCOUNT  0X02  /* sub-command code */
#define EP2  2 /* Endpoint 2 */
#define EP3  3 /* Endpoint 3 */
```

OFFSET	BULK OUT (EP2)	BULK IN (EP1)
SIZEOF_DATA	0X01	0X06
COMMAND	0X20	0X20
ERROR	0X00	Error code
DATA_0	0X02	Number of bytes return (0X05)
DATA_1	CHECKSUM	[Timestamp + 3]: LSB
DATA_2		[Timestamp + 2]
DATA_3		[Timestamp + 1]
DATA_4		[Timestamp + 0]: MSB
DATA_5		CHECKSUM

Table 5 CAN_GET_TICKCOUNT Format

1.5.3 CAN_SET_LED_SWITCH

The CAN_SET_LED_SWITCH command turns the main LED switch on or off. Every individual LED indicator will be in the off state if application sets the main LED switch to the OFF state via this command. Please be noted that the main LED switch is default to ON state each time the MPX-2515 card be powered on.

```
#define CCP_CAN_SYSTEM_SERVICE  0X20 /* COMMAND: service code */
#define CAN_SET_LED_SWITCH  0X03  /* DATA_0: sub-command code */
#define LED_SWITCH_OFF  0X00  /* @ DATA_1 */
#define LED_SWITCH_ON  0X01  /* @ DATA_1 */
#define EP2  2 /* Endpoint 2 */
#define EP3  3 /* Endpoint 3 */
```

OFFSET	BULK OUT (EP2)	BULK IN (EP1)
SIZEOF_DATA	0X02	0X01
COMMAND	0X20	0X20
ERROR	0X00	Error code
DATA_0	0X03	Number of byte written (0X01)
DATA_1	0X00 / 0X01	Checksum
DATA_2	Checksum	
DATA_3		
DATA_4		
DATA_5		

1.5.4 CAN_SET_TRANSFER_RATE

The CAN_SET_TRANSFER_RATE command sets the MCP2515 bit transfer rate on the connecting CAN bus.

MPX-2515 defaults to 125 Kbps transfer rate each time it boots up. Application issues this command to change to its desired CAN bus transfer rate. There are four transfer rates that the application can set via this command. Applications need to program all relevant MCP2515 registers if other transfer rate is desired.

```
#define CCP_CAN_SYSTEM_SERVICE  0X20 /* COMMAND: service code */
#define CAN_SET_TRANSFER_RATE  0X04  /* DATA_0: sub-command code */
#define CAN_1000K_BPS  0X00  /* @DATA_1 */
#define CAN_500K_BPS  0X01  /* @DATA_1 */
#define CAN_250K_BPS  0X03  /* @DATA_1 */
#define CAN_125K_BPS  0X07  /* @DATA_1 */
#define EP2  2 /* Endpoint 2 */
#define EP3  3 /* Endpoint 3 */
```

OFFSET	BULK OUT (EP2)	BULK IN (EP1)
SIZEOF_DATA	0X02	0X01
COMMAND	0X20	0X20

ERROR	0X00	Error code
DATA_0	0X04	Number of byte written (0X01)
DATA_1	Desired rate.	Checksum
DATA_2	Checksum	
DATA_3		
DATA_4		
DATA_5		

1.5.5 CAN_RESET_INSTRUCT

The CAN_RESET_INSTRUCT implements MCP2515 Reset Instruction. The Reset instruction can be used to re-initialize the internal registers of the MCP2515 and set Configuration mode. This command provides the same functionality, via the SPI interface, as the RESET# pin.

```
#define CCP_CAN_SYSTEM_SERVICE  0X20 /* COMMAND: command code */
#define MCP2515_RESET_INSTRUCT  0X85  /* DATA_0: sub-command code
*/
#define EP2  2 /* Endpoint 2 */
#define EP3  3 /* Endpoint 3 */
```

OFFSET	BULK OUT (EP2)	BULK IN (EP1)
SIZEOF_DATA	0X01	0X00
COMMAND	0X20	0X20
ERROR	0X00	Error code
DATA_0	0X85	CHECKSUM
DATA_1	CHECKSUM	
DATA_2		
DATA_3		
DATA_4		
DATA_5		

Table 6 CAN_RESET_INSTRUCT Format

1.5.6 CAN_READ_INSTRUCT

The CAN_READ_INSTRUCT implements MCP2515 Read Instruction. The Read instruction is started by lowering the CS# pin. The Read instruction is then sent to the MCP2515 followed by the 8-bit address (A7 through A0). Next, the data stored in the register at the selected address will be shifted out on the SO pin.

The internal address pointer is automatically incremented to the next address once each byte of data is shifted out. Therefore, it is possible to read the next consecutive register address by continuing to provide clock pulses. Any number of consecutive

register locations can be read sequentially using this method. The read operation is terminated by raising the CS# pin. The following figure shows the MCP2515 Read instruction.

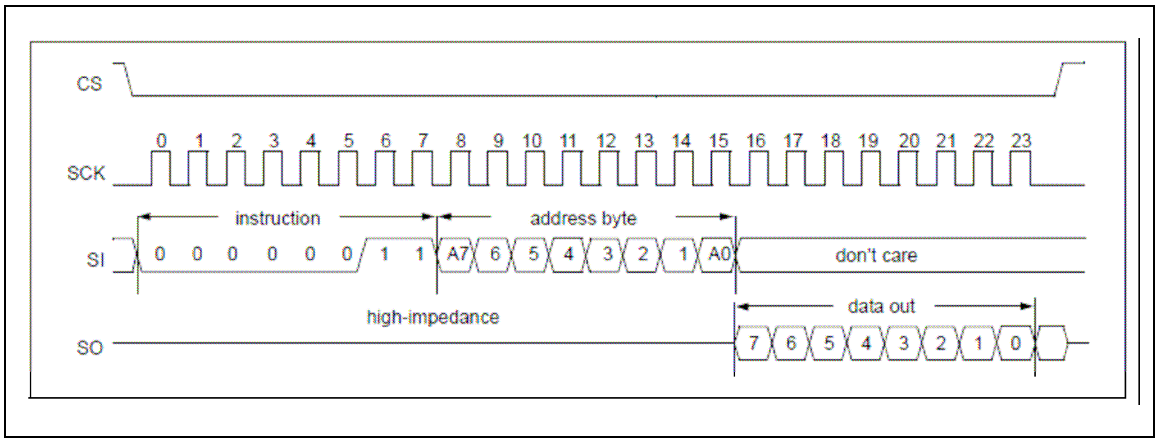

Figure 2 MCP2515 Read Instruction

```
#define CCP_CAN_SYSTEM_SERVICE  0X20 /* COMMAND: command code */
#define CAN_READ_INSTRUCT  0X86  /* DATA_0: sub-command code */
#define EP2  2 /* Endpoint 2 */
#define EP3  3 /* Endpoint 3 */
```

OFFSET	BULK OUT (EP2)	BULK IN (EP1)
SIZEOF_DATA	0X04	Number of bytes returned
COMMAND	0X20	Byte_0 (return byte 0)
ERROR	0X00	Byte_1 (return byte 1)
DATA_0	0X86	Byte_2 (return byte 2)
DATA_1	Number of bytes to read	...
DATA_2	0X03 (constant)	...
DATA_3	Starting register address	...
DATA_4	Checksum	...
...		
DATA_n		Checksum

1.5.7 CAN_WRITE_INSTRUCT

The CAN_WRITE_INSTRUCT implements MCP2515 Write Instruction. The Write instruction is started by lowering the CS# pin. The Write instruction is then sent to the MCP2515 followed by the address and at least one byte of data.

It is possible to write to sequential registers by continuing to clock in data bytes, as long as CS# is held low. Data will actually be written to the register on the rising edge of the SCK line for the D0 bit. If the CS# line is brought high before eight bits are loaded, the write will be aborted for that data byte and previous bytes in the command will have been written. The following figure shows the timing diagram of the byte write sequence.

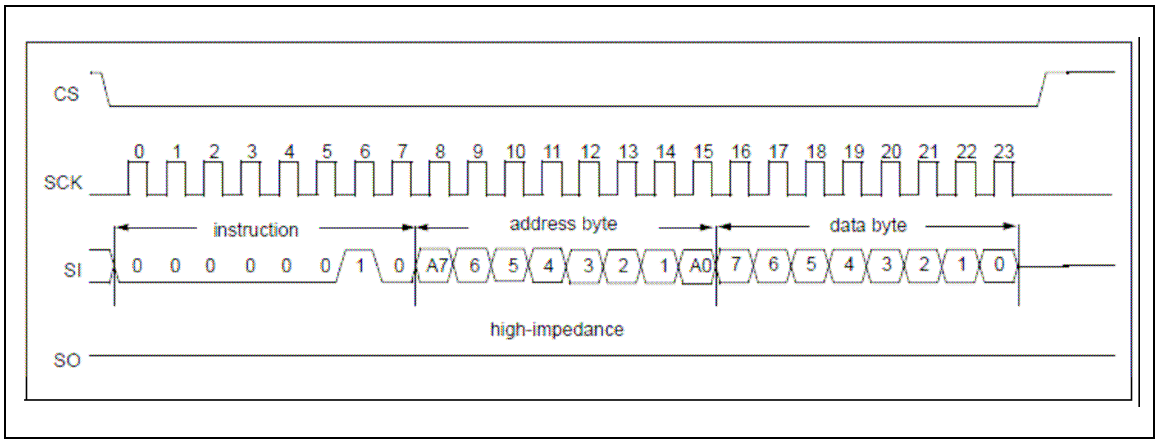

Figure 3 MCP2515 Write Instruction

```
#define CCP_CAN_SYSTEM_SERVICE  0X20 /* COMMAND: command code */
#define CAN_WRITE_INSTRUCT  0X88  /* DATA_0: sub-command code */
#define EP2  2 /* Endpoint 2 */
#define EP3  3 /* Endpoint 3 */
```

OFFSET	BULK OUT (EP2)	BULK IN (EP1)
SIZEOF_DATA	??	0X01
COMMAND	0X20	0X20
ERROR	0X00	Error code
DATA_0	0X88	Number of bytes written
DATA_1	Number of bytes to write	Checksum
DATA_2	0X02 (constant)	
DATA_3	Starting register address	
DATA_4	Byte_0	
DATA_5	Byte_1	
DATA_6	Byte_2	
...	...	
...	...	
DATA_N	Checksum	

1.6 CCP_CAN_READ_FMB Service

1.6.1 Introduction to Message Reception

The MCP2515 includes two full receive buffers with multiple acceptance filters for each. There is also a separate Message Assembly Buffer (MAB) that acts as a third receive buffer.

Of the three receive buffers, the MAB is always committed to receiving the next message from the bus. The MAB assembles all messages received. These messages will be transferred to the RXBn buffers only if the acceptance filter criteria is met.

The remaining two receive buffers, called RXB0 and RXB1, can receive a complete message from the protocol engine via the MAB. The MCU can access one buffer, while the other buffer is available for message reception, or for holding a previously received message.

The entire content of the MAB is moved into the receive buffer once a message is accepted. This means, that regardless of the type of identifier (standard or extended) and the number of data bytes received, the entire receive buffer is overwritten with the MAB contents. Therefore, the contents of all registers in the buffer must be assumed to have been modified when any message is received.

RXB0, the higher priority buffer, has one mask and two message acceptance filters associated with it. The receive message is applied to the mask and filters for RXB0 first.

RXB1 is the lower priority buffer, with one mask and four acceptance filters associated with it.

In addition to the message being applied to the RXB0 mask and filters first, the lower number of acceptance filters makes the match on RXB0 more restrictive and implies a higher priority for that buffer.

When a message is received, bits <3:0> of the RXBnCTRL register will indicate the acceptance filter number that enabled reception and whether the received message is a remote transfer request.

1.6.2 Firmware Messages Buffer (FMB)

In order to reduce the message overwritten possibility, the MPX-2515 firmware has reserved a 64-byte buffer in the memory to store acceptance messages that have received in the MCP2515 message buffer. This Firmware Messages Buffer (FMB) can hold up to three CAN messages associated with some selected MCP2515 registers values at the time the message is received. Applications is then able to analyze the situation of controller and the bus by checking these registers values.

The following figure shows the relationship between FMB of CY8C24794 and RXBn of MCP2515.

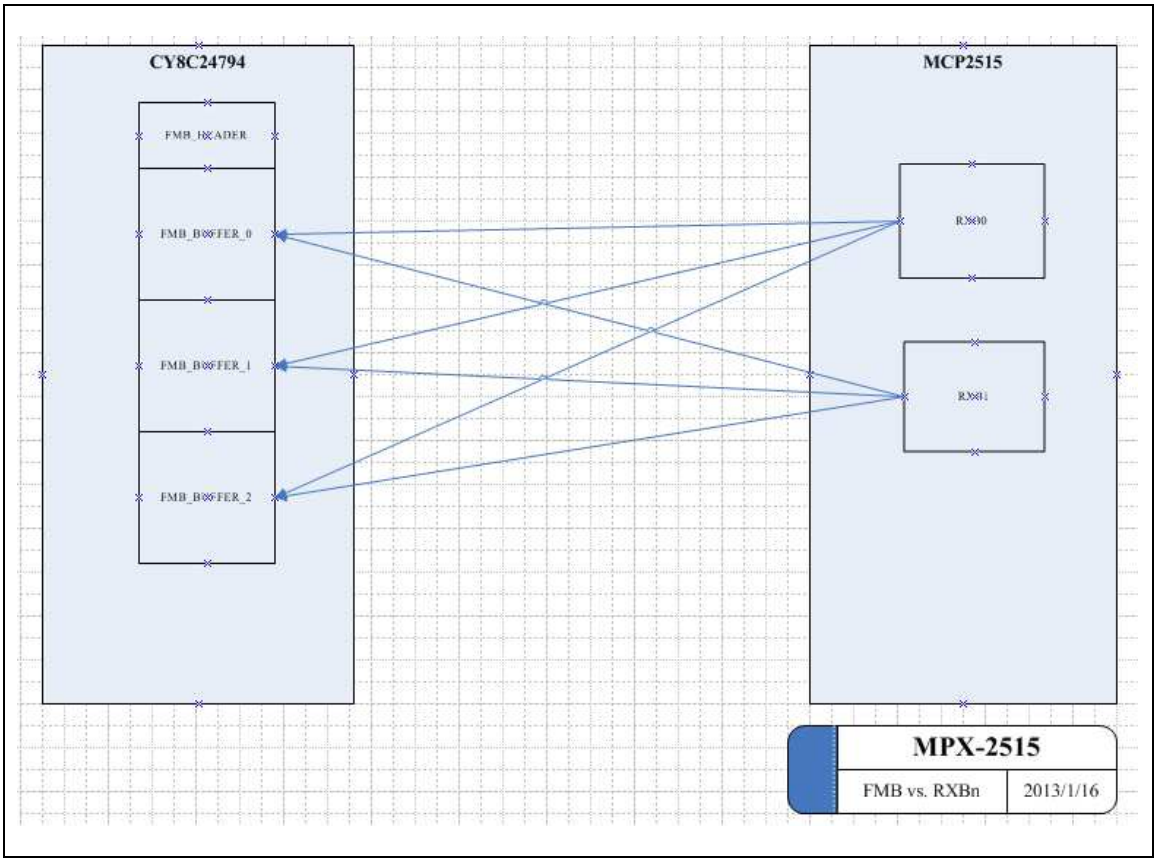

Figure 4 FMB vs. RXBn

1.6.3 CCP_CAN_RED_FMB

The CCP_CAN_READ_FMB service return 64-byte Firmware Message Buffer (FMB). Application issue a BULK IN transfer over the endpoint 4 to proceed this service. Neither CCP Request Packet format nor CCP Response Packet format applies to this service.

The whole Firmware Message Buffer (FMB) can be retrieved via BULK IN transfers over the endpoint 4 from the USB host. The first byte of each message contains ether 0X60 or 0X70, which denotes that this message is copied from RXB0 buffer or RXB1 buffer. The second byte is the associated RXBnCTRL register value. Applications can check this byte for more message received information. The third byte is EFLG at the receiving time. Then, comes a four bytes timestamp in 100 us time unit. Standard ID or Extended ID is followed. RXBnDLC byte has RTR bit indicating whether a Remote Transmit Request. RXBnDLC also

contains Data Length Code (DLC) bits that tell valid bytes in this message. All message bytes are then followed.

Please be noted that CCP provides a way that application can check the status of this FMB. Application issues an INTERRUPT IN transaction over endpoint 3 to retrieve an eight bytes Status Packet, which contains FMB status byte at the offset 3 (zero-based packet). Please refer to the INTERRUPT IN section for detail definitions.

Below is the suggestion of control flow.

1. Issue INTERRUPT IN transaction over endpoint 3.
2. Check FMB Status byte (at offset 3)
3. Issue BULK IN on endpoint 4 to retrieve 64-byte FMB
4. Process message(s)
5. Go back to Step 1.

The following table defines the format of Firmware Messages Buffer. Please refer to the MCP2515 data sheet for detail definition of MCP2515 registers values in the FMB.

OFFSET	NAME	DESCRIPTION
0	RXB_HEADER_MSB	The value of 0X5A
1	RXB_HEADER_LSB	The value of 0X6B
2	RXB_NEXT	Next available buffer (0, 1, 2)
Buffer 0 Start		
3	RXB0 / RXB1	0X60 / 0X70
4	RXBnCTRL	See below notes.
5	EFLG	See below notes.
6	Timestamp + 3	Timestamp LSB
7	Timestamp + 2	
8	Timestamp + 1	
9	Timestamp + 0	Timestamp MSB
10	RXBnSIDH	
11	RXBnSIDL	
12	RXBnEID8	
13	RXBnEID0	
14	RXBnDLC	
15	RXBnD0	
16	RXBnD1	
17	RXBnD2	
18	RXBnD3	

19	RXBnD4	
20	RXBnD5	
21	RXBnD6	
22	RXBnD7	
Buffer 1 Start		
23	RXB0 / RXB1	0X60 / 0X70
24	RXBnCTRL	
25	EFLG	
26	Timestamp + 3	LSB
27	Timestamp + 2	
28	Timestamp + 1	
29	Timestamp + 0	MSB
30	RXBnSIDH	
31	RXBnSIDL	
32	RXBnEID8	
33	RXBnEID0	
34	RXBnDLC	
35	RXBnD0	
36	RXBnD1	
37	RXBnD2	
38	RXBnD3	
39	RXBnD4	
40	RXBnD5	
41	RXBnD6	
42	RXBnD7	
Buffer 2 Start		
43	RXB0 / RXB1	0X60 / 0X70
44	RXBnCTRL	
45	EFLG	
46	Timestamp + 3	LSB
47	Timestamp + 2	
48	Timestamp + 1	
49	Timestamp + 0	MSB
50	RXBnSIDH	
51	RXBnSIDL	
52	RXBnEID8	
53	RXBnEID0	
54	RXBnDLC	
55	RXBnD0	
56	RXBnD1	
57	RXBnD2	
58	RXBnD3	
59	RXBnD4	
60	RXBnD5	
61	RXBnD6	

62	RXBnD7	
2's Complement Checksum		
63	Checksum	

Table 7 Firmware Message Buffer (FMB) format

Notes:

- RXB0 / RXB1: 0X60 if the message is from RXB0. 0X70 if the message is from RXB1.
- n value: 0 if the message is from RXB0; 1 if the message if from RXB1.
- RXBnCTRL: Contains the RXBnCTRL register value of MCP2515.
- EFLG: The value of EFLG register of MCP2515.
- Timestamp: Four bytes unsigned integer value. in 100 us unit.
- RXBnSIDH, RXBnSIDL, RXBnEID8, RXBnEID0, RXBnDLC, RXBnD0, RXBnD1, RXBnD2, RXBnD3, RXBnD4, RXBnD5, RXBnD6, and RXBnD7 are values from MCP2515 corresponding registers.

1.7 CCP_CAN_RX_MASKS_FILTERS Service

1.7.1 Introduction to Message Acceptance Filters and Masks

The CCP_CAN_RX_MASKS_FILTERS uses 0X22 as its command category code. This service provides ways to configure the masks and filters registers of the MCP2515 CAN bus controller.

The MCP2515 includes two full receive buffers (RxB0 and RxB1) with multiple acceptance filters for each. There is also a separate Message Assembly Buffer (MAB) that acts as a third receive buffer.

The MAB is always committed to receiving the next message from the bus. The MAB assembles all messages received. These messages will be transferred to the RxBn buffers only if the acceptance filter criteria is met.

The message acceptance filters and masks are used to determine if a message in the message assembly buffer should be loaded into either of the receive buffers. Once a valid message has been received into the MAB, the identifier fields of the message are compared to the filter values. If there is a match, that message will be loaded into the appropriate receive buffer.

FILTER MATCHING

The filter masks are used to determine which bits in the identifier are examined with the filters. A truth table is shown below that indicates how each bit in the identifier is compared to the masks and filters to determine if the message should be loaded into a receive buffer. The mask essentially determines which bits to apply

the acceptance filters to. If any mask bit is set to a zero, that bit will automatically be accepted, regardless of the filter bit.

Mask Bit n	Filter Bit n	Message Identifier bit	Accept or Reject bit n
0	X	X	Accept
1	0	0	Accept
1	0	1	Reject
1	1	0	Reject
1	1	1	Accept

Note: x = don't care

Table 8 Filter / Mask Truth Table

The following figure shows the receiver buffer block diagram. Please refer to the MCP2515 data sheet for detail information.

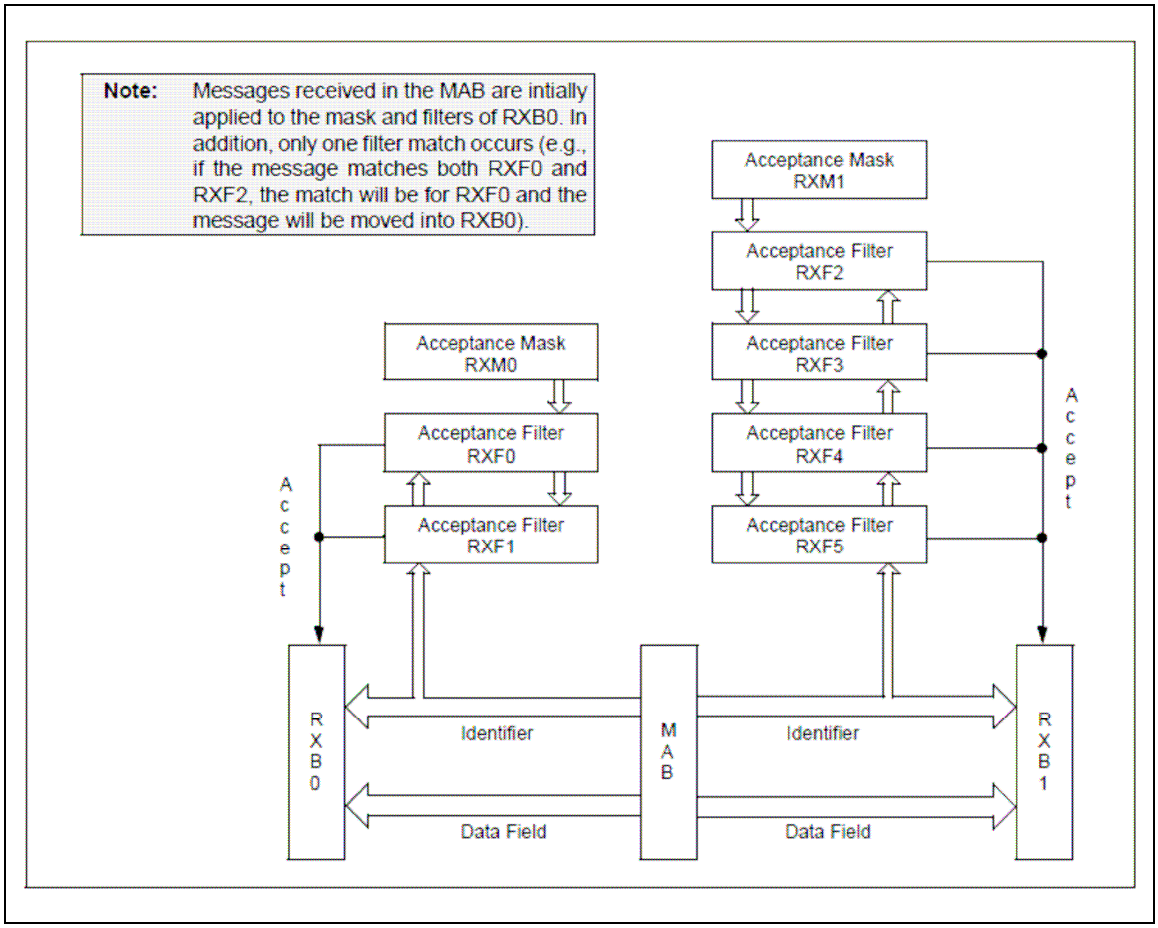

Figure 5 MCP2515 Receive Buffer Block Diagram

As shown in the above receive buffers block diagram, acceptance filters RXF0 and RXF1 (and filter mask RXM0 are associated with RXB0. Filters RXF2, RXF3, RXF4, RXF5 and mask RXM1 are associated with RXB1.

The following figure shows how the masks and filters apply to CAN frames.

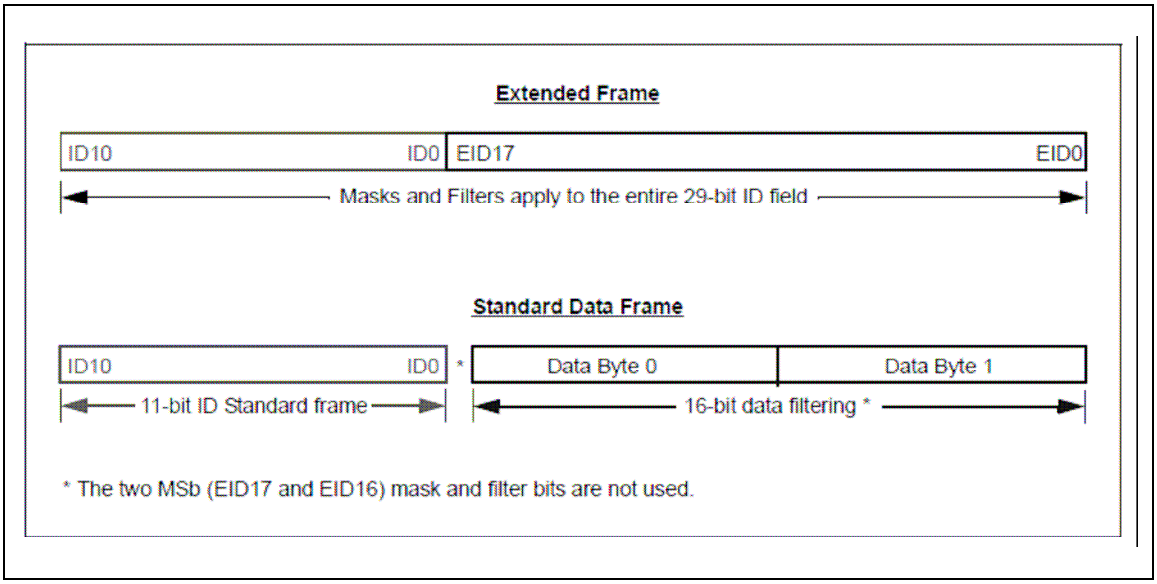

Figure 6 Masks and Filters Apply to CAN Frames

FILHIT BITS

Filter matches on received messages can be determined by the FILHIT bits in the associated RXBnCTRL register. RXB0CTRL.FILHIT0 for buffer 0 and RXB1CTR.FILHIT<2:0> for buffer 1.

The three FILHIT<2:0> bits for receive buffer 1 (RXB1) are coded as follows:

- 101 = Acceptance filter 5 (RXF5)
- 100 = Acceptance filter 4 (RXF4)
- 011 = Acceptance filter 3 (RXF3)
- 010 = Acceptance filter 2 (RXF2)
- 001 = Acceptance filter 1 (RXF1)
- 000 = Acceptance filter 0 (RXF0)

Note: 000 and 001 can only occur if the BUKT bit in RXB0CTRL is set, allowing RXB0 message to roll over into RXB1.

RXB0CTRL contains two copies of the BUKT bit and the FILHIT<0> bit.

The coding of the BUKT bit enables these three bits to be used similarly to the RXB1CTRL.FILHIT bits and to distinguish a hit on filter RXF0 and RXF1 in either RXB0 or after a roll over into RXB1.

- 111 = Acceptance Filter 1 (RXB1)
- 110 = Acceptance Filter 0 (RXB1)
- 001 = Acceptance Filter 1 (RXB0)
- 000 = Acceptance Filter 0 (RXB0)

Please refer to the MCP2515 data sheet for detail information.

1.7.2 MCP_RX_MASK_0

The MCP_RX_MASK_0 request is defined as the sub-command code 0X20 of CCP_CMND_CAN_RX_MASKS_FILTERS service.

```
#define CCP_CAN_RX_MASKS_FILTERS  0X22
#define MCP_RX_MASK_0  0X20
#define EP2  2 /* Endpoint 2 */
#define EP3  3 /* Endpoint 3 */
```

OFFSET	BULK OUT (EP2)	BULK IN (EP1)
SIZEOF_DATA	0X06	0X01
COMMAND	0X22	0X22
ERROR	0X00	Error code
DATA_0	0X20	Number of bytes written
DATA_1	0X04: Number of bytes to write.	checksum
DATA_2	RXM0SIDH	
DATA_3	RXM0SIDL	
DATA_4	RXM0EID8	
DATA_5	RXM0EID0	
DATA_6	checksum	
DATA_7		

1.7.3 MCP_RX_MASK_1

The MCP_RX_MASK_1 request is defined as the sub-command code 0X24 of CCP_CMND_CAN_RX_MASKS_FILTERS service.

```
#define CCP_CAN_RX_MASKS_FILTERS  0X22
#define MCP_RX_MASK_1  0X24
#define EP2  2 /* Endpoint 2 */
#define EP3  3 /* Endpoint 3 */
```

OFFSET	BULK OUT (EP2)	BULK IN (EP1)
SIZEOF_DATA	0X06	0X01
COMMAND	0X22	0X22
ERROR	0X00	Error code
DATA_0	0X24	Number of bytes written
DATA_1	0X04: Number of bytes to write.	checksum

DATA_2	RXM1SIDH	
DATA_3	RXM1SIDL	
DATA_4	RXM1EID8	
DATA_5	RXM1EID0	
DATA_6	checksum	
DATA_7		

1.7.4 MCP_RX_FILTER_0

The MCP_RX_FILTER_0 request is defined as the sub-command code 0X00 of CCP_CMND_CAN_RX_MASKS_FILTERS service.

```
#define CCP_CAN_RX_MASKS_FILTERS  0X22
#define MCP_RX_FILTER_0  0X00
#define EP2  2 /* Endpoint 2 */
#define EP3  3 /* Endpoint 3 */
```

OFFSET	BULK OUT (EP2)	BULK IN (EP1)
SIZEOF_DATA	0X06	0X01
COMMAND	0X22	0X22
ERROR	0X00	Error code
DATA_0	0X00	Number of bytes written
DATA_1	0X04: Number of bytes to write.	checksum
DATA_2	RXF0SIDH	
DATA_3	RXF0SIDL	
DATA_4	RXF0EID8	
DATA_5	RXF0EID0	
DATA_6	checksum	
DATA_7		

1.7.5 MCP_RX_FILTER_1

The MCP_RX_FILTER_1 request is defined as the sub-command code 0X04 of CCP_CMND_CAN_RX_MASKS_FILTERS service.

```
#define CCP_CAN_RX_MASKS_FILTERS  0X22
#define MCP_RX_FILTER_1  0X04
#define EP2  2 /* Endpoint 2 */
#define EP3  3 /* Endpoint 3 */
```

OFFSET	BULK OUT (EP2)	BULK IN (EP1)
SIZEOF_DATA	0X06	0X01
COMMAND	0X22	0X22

ERROR	0X00	Error code
DATA_0	0X04	Number of bytes written
DATA_1	0X04: Number of bytes to write.	checksum
DATA_2	RXF1SIDH	
DATA_3	RXF1SIDL	
DATA_4	RXF1EID8	
DATA_5	RXF1EID0	
DATA_6	checksum	
DATA_7		

1.7.6 MCP_RX_FILTER_2

The MCP_RX_FILTER_2 request is defined as the sub-command code 0X08 of CCP_CMND_CAN_RX_MASKS_FILTERS service.

```
#define CCP_CAN_RX_MASKS_FILTERS  0X22
#define MCP_RX_FILTER_2  0X08
#define EP2  2 /* Endpoint 2 */
#define EP3  3 /* Endpoint 3 */
```

OFFSET	BULK OUT (EP2)	BULK IN (EP1)
SIZEOF_DATA	0X06	0X01
COMMAND	0X22	0X22
ERROR	0X00	Error code
DATA_0	0X08	Number of bytes written
DATA_1	0X04: Number of bytes to write.	checksum
DATA_2	RXF2SIDH	
DATA_3	RXF2SIDL	
DATA_4	RXF2EID8	
DATA_5	RXF2EID0	
DATA_6	checksum	
DATA_7		

1.7.7 MCP_RX_FILTER_3

The MCP_RX_FILTER_3 request is defined as the sub-command code 0X10 of CCP_CMND_CAN_RX_MASKS_FILTERS service.

```
#define CCP_CAN_RX_MASKS_FILTERS  0X22
#define MCP_RX_FILTER_3  0X10
#define EP2  2 /* Endpoint 2 */
#define EP3  3 /* Endpoint 3 */
```

OFFSET	BULK OUT (EP2)	BULK IN (EP1)
SIZEOF_DATA	0X06	0X01
COMMAND	0X22	0X22
ERROR	0X00	Error code
DATA_0	0X10	Number of bytes written
DATA_1	0X04: Number of bytes to write.	checksum
DATA_2	RXF3SIDH	
DATA_3	RXF3SIDL	
DATA_4	RXF3EID8	
DATA_5	RXF3EID0	
DATA_6	checksum	
DATA_7		

1.7.8 MCP_RX_FILTER_4

The MCP_RX_FILTER_4 request is defined as the sub-command code 0X14 of CCP_CMND_CAN_RX_MASKS_FILTERS service.

```
#define CCP_CAN_RX_MASKS_FILTERS  0X22
#define MCP_RX_FILTER_4  0X14
#define EP2  2 /* Endpoint 2 */
#define EP3  3 /* Endpoint 3 */
```

OFFSET	BULK OUT (EP2)	BULK IN (EP1)
SIZEOF_DATA	0X06	0X01
COMMAND	0X22	0X22
ERROR	0X00	Error code
DATA_0	0X14	Number of bytes written
DATA_1	0X04: Number of bytes to write.	checksum
DATA_2	RXF4SIDH	
DATA_3	RXF4SIDL	
DATA_4	RXF4EID8	
DATA_5	RXF4EID0	
DATA_6	checksum	
DATA_7		

1.7.9 MCP_RX_FILTER_5

The MCP_RX_FILTER_5 request is defined as the sub-command code 0X18 of CCP_CMND_CAN_RX_MASKS_FILTERS service.

```
#define CCP_CAN_RX_MASKS_FILTERS  0X22
#define MCP_RX_FILTER_5  0X18
#define EP2  2 /* Endpoint 2 */
#define EP3  3 /* Endpoint 3 */
```

OFFSET	BULK OUT (EP2)	BULK IN (EP1)
SIZEOF_DATA	0X06	0X01
COMMAND	0X22	0X22
ERROR	0X00	Error code
DATA_0	0X18	Number of bytes written
DATA_1	0X04: Number of bytes to write.	checksum
DATA_2	RXF5SIDH	
DATA_3	RXF5SIDL	
DATA_4	RXF5EID8	
DATA_5	RXF5EID0	
DATA_6	checksum	
DATA_7		

1.8 CCP_CAN_TX Service

1.8.1 Introduction to Message Transmission

TRANSMIT BUFFERS

The MCP2515 implements three transmit buffers. Each of these buffers occupies 14 bytes of SRAM and are mapped into the device memory map.

The first byte, TXBnCTRL, is a control register associated with the message buffer. The information is this register determines the conditions under which the message will be transmitted and indicates the status of the message transmission.

Five bytes are used to hold the standard and extended identifiers, as well as other message arbitration information.

The last eight bytes are for the eight possible data bytes of the message to be transmitted.

At a minimum, the TXBnSIDH, TXBnSIDL and TXBnDLC registers must be loaded. If data bytes are present in the message, the TXBnDm registers must also be loaded. If the message is to use extended identifiers, the TXBnEIDm registers must aso be loaded and the TXBnSIDL.EXIDE bit set.

TRANSMIT PRIORITY

Transmit priority is a prioritization within the MCP2515 of the pending transmittable messages. This is independent from, and not necessarily related to, any prioritization implicit in the message arbitration scheme built into the CAN protocol.

There are four levels of transmit priority. If TXBnCTRL.TXP<1:0> for a particular message buffer is set to 11, that buffer has the highest possible priority. If TXBnCTRL.TXP<1:0> for a particular buffer is 00, that buffer has the lowest possible priority.

ABORTING TRANSMISSION

The MCU can request to abort a message in a specific message buffer by clearing the associated TXBnCTRL.TXREQ bit.

In addition, all pending messages can be requested to be aborted by setting the CANCTRL.ABAT bit. This bit must be reset (typically after the TXREQ bits have been verified to be cleared) to continue transmitting messages.

Messages that were transmitting when the abort was requested will continue to transmit. If the message does not successfully complete transmission (i.e., lost arbitration or was interrupted by an error frame), it will then be aborted.

1.8.2 MCP_LOAD_TXB0SIDH

The MCP_LOAD_TXB0SIDH command loads the bytes in the Request Packet to the associated MCP2515 registers in order to complete the corresponding message transmission on the CAN bus.

```
#define CCP_CAN_TX_SERVICE  0X23 /* command category code */
#define MCP_LOAD_TXB0SIDH  0X40  /* sub-command code */
#define EP2  2 /* Endpoint 2 */
#define EP3  3 /* Endpoint 3 */
```

OFFSET	BULK OUT (EP2)	BULK IN (EP1)
SIZEOF_DATA	0X14	0X05
COMMAND	0X23	0X23
ERROR	0X00	Error code
DATA_0	0X40	Number of bytes return (0X04)
DATA_1	Overwrite flag	EFLG before transmitting
DATA_2	Reserved	TXB0CTRL before transmitting
DATA_3	0X30	EFLG after transmitted
DATA_4	Value to load to TXB0CTRL	TXB0CTRL after transmitted

DATA_5	0X31	checksum
DATA_6	0X0D	
DATA_7	TXB0SIDH	
DATA_8	TXB0SIDL	
DATA_9	TXB0EID8	
DATA_10	TXB0EID0	
DATA_11	TXB0DLC	
DATA_12	TXB0D0	
DATA_13	TXB0D1	
DATA_14	TXB0D2	
DATA_15	TXB0D3	
DATA_16	TXB0D4	
DATA_17	TXB0D5	
DATA_18	TXB0D6	
DATA_19	TXB0D7	
DATA_20	checksum	

1.8.3 MCP_LOAD_TXB0D0

The MCP_LOAD_TXB0D0 command loads the bytes in the Request Packet to the associated MCP2515 registers in order to complete the corresponding message transmission on the CAN bus.

```
#define CCP_CAN_TX_SERVICE  0X23 /* command category code */
#define MCP_LOAD_TXB0D0  0X41  /* sub-command code */
#define EP2  2 /* Endpoint 2 */
#define EP3  3 /* Endpoint 3 */
```

OFFSET	BULK OUT (EP2)	BULK IN (EP1)
SIZEOF_DATA	0X14	0X05
COMMAND	0X23	0X23
ERROR	0X00	Error code
DATA_0	0X41	Number of bytes return (0X04)
DATA_1	Overwrite flag	EFLG before transmitting
DATA_2	Reserved	TXB0CTRL before transmitting
DATA_3	0X30	EFLG after transmitted
DATA_4	Value to load to TXB0CTRL	TXB0CTRL after transmitted
DATA_5	0X36	checksum
DATA_6	0X09	
DATA_7	Reserved	
DATA_8	Reserved	
DATA_9	Reserved	

DATA_10	Reserved	
DATA_11	TXB0DLC	
DATA_12	TXB0D0	
DATA_13	TXB0D1	
DATA_14	TXB0D2	
DATA_15	TXB0D3	
DATA_16	TXB0D4	
DATA_17	TXB0D5	
DATA_18	TXB0D6	
DATA_19	TXB0D7	
DATA_20	checksum	

1.8.4 MCP_LOAD_TXB1SIDH

The MCP_LOAD_TXB1SIDH command loads the bytes in the Request Packet to the associated MCP2515 registers in order to complete the corresponding message transmission on the CAN bus.

```
#define CCP_CAN_TX_SERVICE  0X23 /* command category code */
#define MCP_LOAD_TXB1SIDH  0X42  /* sub-command code */
#define EP2  2 /* Endpoint 2 */
#define EP3  3 /* Endpoint 3 */
```

OFFSET	BULK OUT (EP2)	BULK IN (EP1)
SIZEOF_DATA	0X14	0X05
COMMAND	0X23	0X23
ERROR	0X00	Error code
DATA_0	0X42	Number of bytes return (0X04)
DATA_1	Overwrite flag	EFLG before transmitting
DATA_2	Reserved	TXB1CTRL before transmitting
DATA_3	0X40	EFLG after transmitted
DATA_4	Value to load to TXB1CTRL	TXB1CTRL after transmitted
DATA_5	0X41	checksum
DATA_6	0X0D	
DATA_7	TXB1SIDH	
DATA_8	TXB1SIDL	
DATA_9	TXB1EID8	
DATA_10	TXB1EID0	
DATA_11	TXB1DLC	
DATA_12	TXB1D0	
DATA_13	TXB1D1	
DATA_14	TXB1D2	

DATA_15	TXB1D3	
DATA_16	TXB1D4	
DATA_17	TXB1D5	
DATA_18	TXB1D6	
DATA_19	TXB1D7	
DATA_20	checksum	

1.8.5 MCP_LOAD_TXB1D0

The MCP_LOAD_TXB0SIDH command loads the bytes in the Request Packet to the associated MCP2515 registers in order to complete the corresponding message transmission on the CAN bus.

```
#define CCP_CAN_TX_SERVICE  0X23 /* command category code */
#define MCP_LOAD_TXB1D0  0X43  /* sub-command code */
#define EP2  2 /* Endpoint 2 */
#define EP3  3 /* Endpoint 3 */
```

OFFSET	BULK OUT (EP2)	BULK IN (EP1)
SIZEOF_DATA	0X14	0X05
COMMAND	0X23	0X23
ERROR	0X00	Error code
DATA_0	0X43	Number of bytes return (0X04)
DATA_1	Overwrite flag	EFLG before transmitting
DATA_2	Reserved	TXB1CTRL before transmitting
DATA_3	0X40	EFLG after transmitted
DATA_4	Value to load to TXB0CTRL	TXB1CTRL after transmitted
DATA_5	0X46	checksum
DATA_6	0X09	
DATA_7	Reserved	
DATA_8	Reserved	
DATA_9	Reserved	
DATA_10	Reserved	
DATA_11	TXB1DLC	
DATA_12	TXB1D0	
DATA_13	TXB1D1	
DATA_14	TXB1D2	
DATA_15	TXB1D3	
DATA_16	TXB1D4	
DATA_17	TXB1D5	
DATA_18	TXB1D6	
DATA_19	TXB1D7	

DATA_20	checksum	

1.8.6 MCP_LOAD_TXB2SIDH

The MCP_LOAD_TXB2SIDH command loads the bytes in the Request Packet to the associated MCP2515 registers in order to complete the corresponding message transmission on the CAN bus.

```
#define CCP_CAN_TX_SERVICE  0X23 /* command category code */
#define MCP_LOAD_TXB2SIDH  0X44  /* sub-command code */
#define EP2  2 /* Endpoint 2 */
#define EP3  3 /* Endpoint 3 */
```

OFFSET	BULK OUT (EP2)	BULK IN (EP1)
SIZEOF_DATA	0X14	0X05
COMMAND	0X23	0X23
ERROR	0X00	Error code
DATA_0	0X44	Number of bytes return (0X04)
DATA_1	Overwrite flag	EFLG before transmitting
DATA_2	Reserved	TXB2CTRL before transmitting
DATA_3	0X50	EFLG after transmitted
DATA_4	Value to load to TXB0CTRL	TXB2CTRL after transmitted
DATA_5	0X51	checksum
DATA_6	0X0D	
DATA_7	TXB2SIDH	
DATA_8	TXB2SIDL	
DATA_9	TXB2EID8	
DATA_10	TXB2EID0	
DATA_11	TXB2DLC	
DATA_12	TXB2D0	
DATA_13	TXB2D1	
DATA_14	TXB2D2	
DATA_15	TXB2D3	
DATA_16	TXB2D4	
DATA_17	TXB2D5	
DATA_18	TXB2D6	
DATA_19	TXB2D7	
DATA_20	checksum	

1.8.7 MCP_LOAD_TXB2D0

The MCP_LOAD_TXB2D0 command loads the bytes in the Request Packet to the associated MCP2515 registers in order to complete the corresponding message transmission on the CAN bus.

```
#define CCP_CAN_TX_SERVICE  0X23 /* command category code */
#define MCP_LOAD_TXB2d0  0X45  /* sub-command code */
#define EP2  2 /* Endpoint 2 */
#define EP3  3 /* Endpoint 3 */
```

OFFSET	BULK OUT (EP2)	BULK IN (EP1)
SIZEOF_DATA	0X14	0X05
COMMAND	0X23	0X23
ERROR	0X00	Error code
DATA_0	0X45	Number of bytes return (0X04)
DATA_1	Overwrite flag	EFLG before transmitting
DATA_2	Reserved	TXB2CTRL before transmitting
DATA_3	0X50	EFLG after transmitted
DATA_4	Value to load to TXB0CTRL	TXB2CTRL after transmitted
DATA_5	0X56	checksum
DATA_6	0X09	
DATA_7	Reserved	
DATA_8	Reserved	
DATA_9	Reserved	
DATA_10	Reserved	
DATA_11	TXB2DLC	
DATA_12	TXB2D0	
DATA_13	TXB2D1	
DATA_14	TXB2D2	
DATA_15	TXB2D3	
DATA_16	TXB2D4	
DATA_17	TXB2D5	
DATA_18	TXB2D6	
DATA_19	TXB2D7	
DATA_20	checksum	

1.8.8 MCP_ABORT_TXB0

The MCP_ABORT_TXB0 command set the TXREQ.ABTF to abort the corresponding message. Please refer to the ABORT TRANSMISSION paragraph and the MCP2515 data sheet for transmission abort limitations.

```
#define CCP_CAN_TX_SERVICE  0X23 /* command category code */
#define MCP_ABORT_TXB0  0X38  /* sub-command code */
#define EP2  2 /* Endpoint 2 */
#define EP3  3 /* Endpoint 3 */
```

OFFSET	BULK OUT (EP2)	BULK IN (EP1)
SIZEOF_DATA	0X04	0X01
COMMAND	0X23	0X23
ERROR	0X00	Error code
DATA_0	0X38	Abort status code
DATA_1	Reserved	checksum
DATA_2	0X30	
DATA_3	0X08	
DATA_4	checksum	
DATA_5		

1.8.9 MCP_ABORT_TXB1

The MCP_ABORT_TXB1 command set the TXREQ.ABTF to abort the corresponding message. Please refer to the ABORT TRANSMISSION paragraph and the MCP2515 data sheet for transmission abort limitations.

```
#define CCP_CAN_TX_SERVICE  0X23 /* command category code */
#define MCP_ABORT_TXB1  0X48  /* sub-command code */
#define EP2  2 /* Endpoint 2 */
#define EP3  3 /* Endpoint 3 */
```

OFFSET	BULK OUT (EP2)	BULK IN (EP1)
SIZEOF_DATA	0X04	0X01
COMMAND	0X23	0X23
ERROR	0X00	Error code
DATA_0	0X48	Abort status code
DATA_1	Reserved	checksum
DATA_2	0X40	
DATA_3	0X08	
DATA_4	checksum	
DATA_5		

1.8.10 MCP_ABORT_TXB2

The MCP_ABORT_TXB2 command set the TXREQ.ABTF to abort the corresponding message. Please refer to the ABORT TRANSMISSION paragraph and the MCP2515 data sheet for transmission abort limitations.

```
#define CCP_CAN_TX_SERVICE  0X23 /* command category code */
```

```
#define MCP_ABORT_TXB2  0X58  /* sub-command code */
#define EP2  2 /* Endpoint 2 */
#define EP3  3 /* Endpoint 3 */
```

OFFSET	BULK OUT (EP2)	BULK IN (EP1)
SIZEOF_DATA	0X04	0X01
COMMAND	0X23	0X23
ERROR	0X00	Error code
DATA_0	0X58	Abort status code
DATA_1	Reserved	checksum
DATA_2	0X50	
DATA_3	0X08	
DATA_4	checksum	
DATA_5		

1.9 CCP_CAN_BIT_MODIFY Service

1.9.1 Introduction to Bit Modify

The Bit Modify instruction provides a means for setting or clearing individual bits in specific status and control registers. This command is not for all registers. Please refer to the MCP2515 data sheet to determine which registers allow the use of this command.

Note:

Executing the Bit Modify command on registers that are not bit-modifiable will force the mask to FFh. This will allow byte writes to the registers, not bit modify.

The following figure shows the Bit Modify operations.

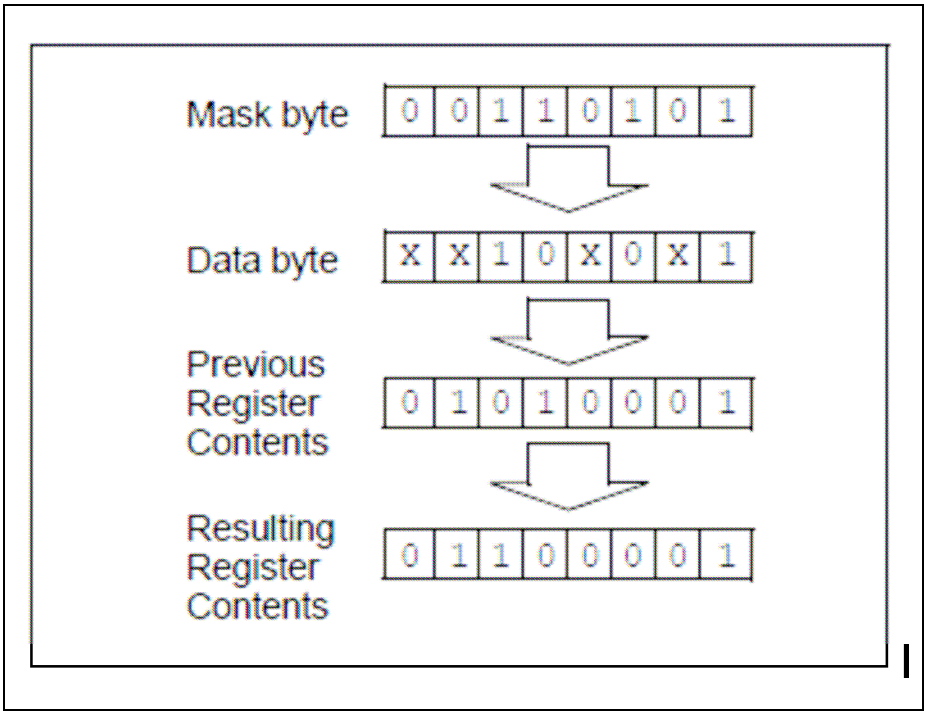

Figure 7 Bit Modify Instruction

The part is selected by lowering the CS# pin and the Bit Modify command byte is then sent to the MCP2515. The command is followed by the address of the register, the mask byte and finally the data byte.

The mask byte determines which bits in the register will be allowed to change. A '1' in the mask byte will allow a bit in the register to change, while a '0' will not.

The data byte determines what value the modified bits in the register will be changed to. A '1' in the data byte will set the bit and a '0' will clear a bit, provided that the mask for that bit is set to a '1'.

1.9.2 CAN_BIT_MODIFY Command

The CAN_BIT_MODIFY command is provided to implement the MCP2515 Bit Modify Instruction.

The following figure defines the MCP2515 Bit Modify Instruction. Please refer to the MCP2515 data sheet for detail information.

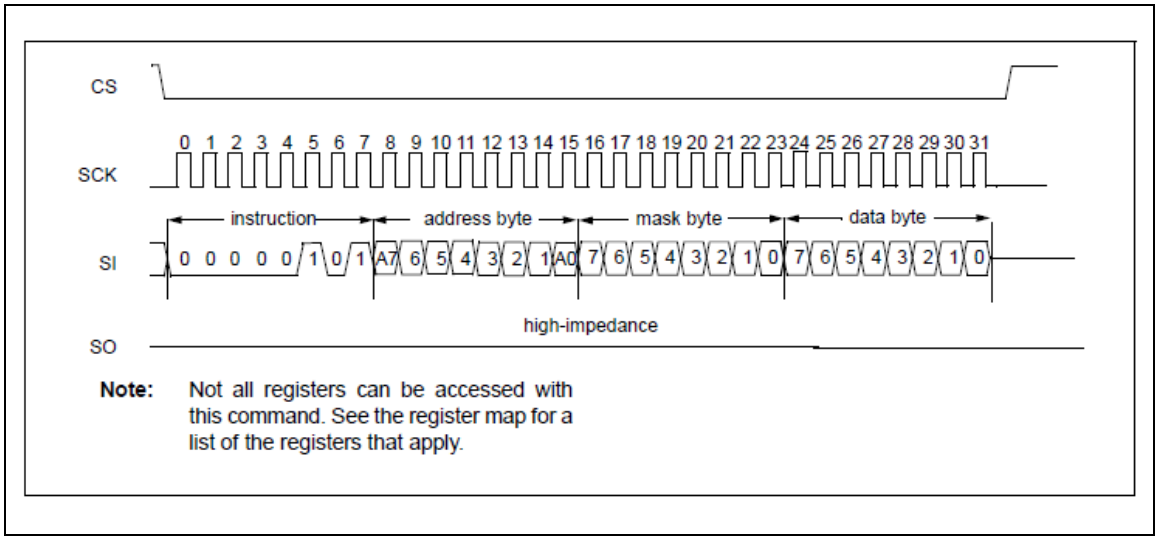

Figure 8 MCP2515 Bit Modify Instruction

The following registers can be modified by the Bit Modify Instruction. Please refer to the MCP2515 data sheet for detail information.

- **BFPCTRL (0X0C):** RXnBF Pin Control and Status Register
- **TXRTSCTRL (0X0D):** TXnRTS Pin Control and Status Register
- **CANCTRL (0X0F):** CAN Control Register
- **CNF3 (0X28):** Configuration 3 Register
- **CNF2(0X29):** Configuration 2 Register
- **CNF1(0X2A):** Configuration 1 Register
- **CANINTE (0X2B):** Interrupt Enable Register
- **CANINTF (0X2C):** Interrupt Flag Register
- **EFLG (0X2D):** Error Flag Register
- **TXB0CTRL (0X30):** Transmit Buffer 0 Control Register
- **TXB1CTRL (0X40):** Transmit Buffer 1 Control Register
- **TXB2CTRL (0X50):** Transmit Buffer 2 Control Register
- **RXB0CTRL (0X60):** Receive Buffer 0 Control Register
- **RXB1CTRL (0X70):** Receive Buffer 1 Control Register

```
/* command category code */
#define CCP_CAN_BIT_MODIFY_SERVICE  0X26

#define EP2  2 /* Endpoint 2 */
```

OFFSET	BULK OUT (EP2)	BULK IN (EP1)
SIZEOF_DATA	0X03	N/A
COMMAND	0X26	
ERROR	0X00	
DATA_0	Bit Modifiable register	

DATA_1	Bit Modify mask	
DATA_2	Bit Modify value	
DATA_3	Checksum	
DATA_4		
DATA_5		
DATA_6		
DATA_7		

NOTE. There is no BULK IN transfer on this command at endpoint 3. Applications shouldn't issue a BULK IN transfer by using the 0X26 command code.

2 Reference

[1] CY8C24x94 PSoC Programmable System-on-Chip Technical Reference Manual (TRM). Cypress Document No. 001-14463 Rev. *E.

[2] MCP2515 Stand-Alone CAN Controller With SPI Interface DS21801F, Microchip.

[3] MCP2551 High-Speed CAN Transceiver DS21667F, 2010 Microchip.

[4] Microsoft Visual Studio 2010 Help, Microsoft.

[5] CiA 301 CANopen application layer and communication profile version 4.2.0, 21 February 2011, CAN in Automation.

[6] CiA 103 Intrinsically safe capable physical layer version 1.0.0, February 2010, CAN in Automation.

[7] CiA 303 CANopen Recommendation Part 1 Cabling and connector pin assignment Version 1.8.0, 27 April 2012, CAN in Automation.

[8] A CAN Physical Layer Discussion, AN228, DS00228A, 2002 Microchip.