Freescale Semiconductor
Application Note

Document Number: AN5045
Rev. 0, 11/2014

Using the MPC5777M MCAN Module to Exchange CAN FD Messages

by: Graham Rice

1 Introduction

A CAN network (Controller Area Network) is an asynchronous serial bus network that connects devices, sensors and actuators for control applications. This multimaster communication protocol was first developed in 1986 for automotive application data rate of up to 1 Mbps with high integrity. CAN is now standardised in ISO 11898, ISO 16845 and SAE J1939 for automotive, industrial and general embedded communications. Since 1993, Freescale have included CAN controllers to power management chips and many 8-bit, 16-bit and 32-bit embedded architectures, including Qorivva automotive microcontrollers and QorIQ network processors.

The MPC57xx family of microcontrollers embed the Multimaster CAN (MCAN) module. This application note demonstrates how to use the MPC5777 MCAN module and more specifically the latest CAN FD (Flexible Data) features that are included in the MCAN module.

Using the Freescale MPC57xx motherboard and MPC5777M expansion card with the example code provided, previous CAN experience is not

Contents

necessary. If you do require basic CAN training before reading this application note, please go to CAN link. Examples used in this document are designed to demonstrate the latest MCAN module capabilities.

1.1 Objectives

After reading this document the reader will be able to:

- Understand the improvements available with CAN FD
- Perform basic transmit and receive operations with the MCAN module
- Configure memory space to transmit and receive messages
- Filter the received messages by the message ID number
- Send and receive CAN FD (Flexible Data) frames at higher speed.

2 CAN FD overview

This section provides an overview of CAN FD. It also describes FD message format and structure.

2.1 CAN FD

'FD' stands for flexible data. It means there is a change in bit rate at the end of the arbitration phase, for the data phase.

The CAN 2.0 specification limits a CAN frame to 8 data bytes at a maximum bit rate of 1 Mbps.

The CAN FD standard allows extended frames to be sent, up to 64 data bytes in a single frame at a higher bit rate for the data phase of a frame, up to 8 Mbps. The CAN FD standard introduces the ability to switch from one bit rate to another mid-frame. Extended Data Length (EDL), as shown in Figure 2, sets a data length of up to 8 or up to 64 data bytes. Bit Rate Switching (BRS) indicates whether two bit rates (the data phase is transmitted at a different bit rate to the arbitration phase) are enabled.

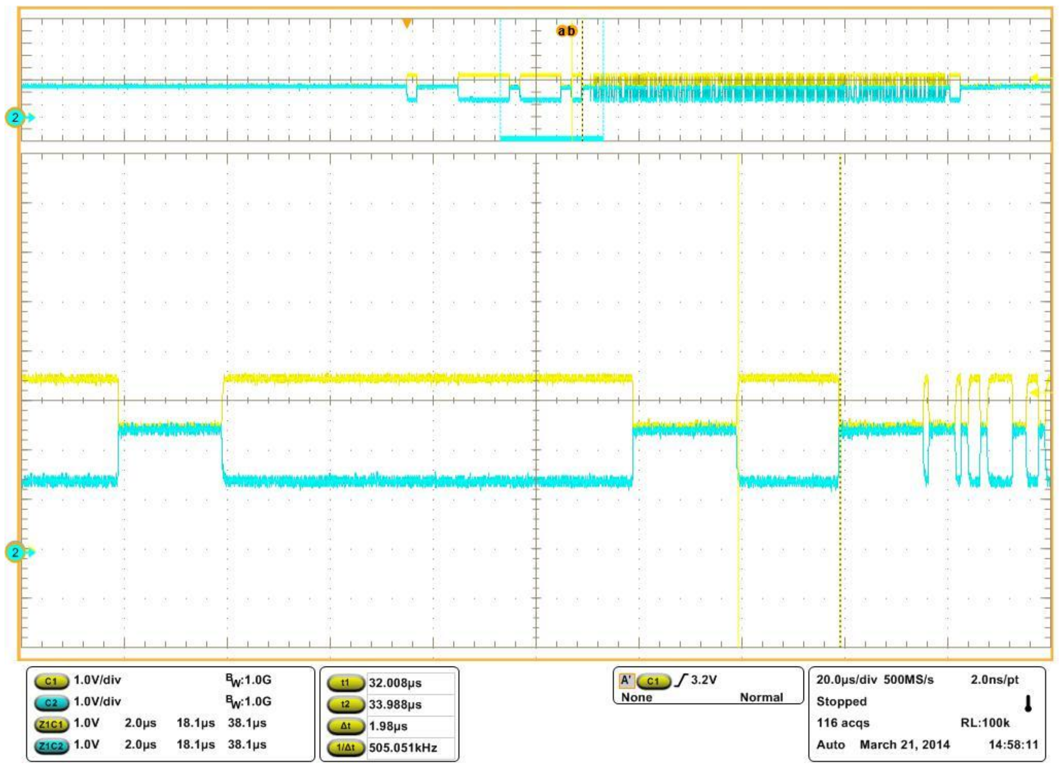


Figure 1 Bit rate change mid-message

2.2 FD message format and structure

The data sent is packaged into a message as shown in Figure 2, consisting of:

- an arbitration phase
- a data transmission phase
- an ACK phase.

The arbitration phase is a message header consisting of an ID number and other bits to indicate the purpose of the message (supplying or requesting data), the speed and format configuration (CAN or CAN FD).

This is followed by the data transmission phase, consisting of the Data Length Code (DLC), to indicate how many data bytes the message contains. The data the user wishes to send the CRC and finally a dominant bit.

The ACK (acknowledgment) is transmitted by other nodes on the bus, if at least one has successfully received the message.

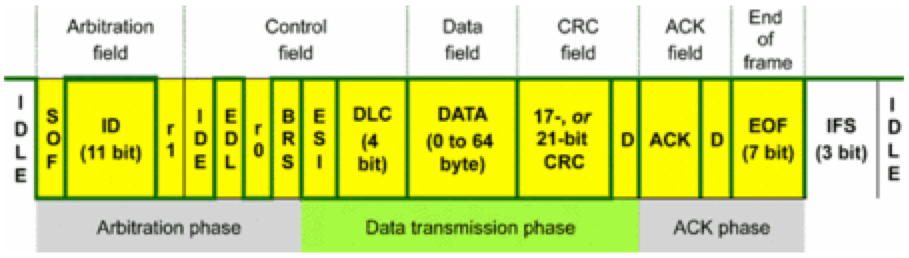


Figure 2 CAN FD frame

3 Configuring and using the MCAN module

Figure 3 shows the steps required to initialise and configure an MPC5777M MCAN module, to create a low level MCAN driver. Each step within the flowchart is detailed in the following sections of this application note.

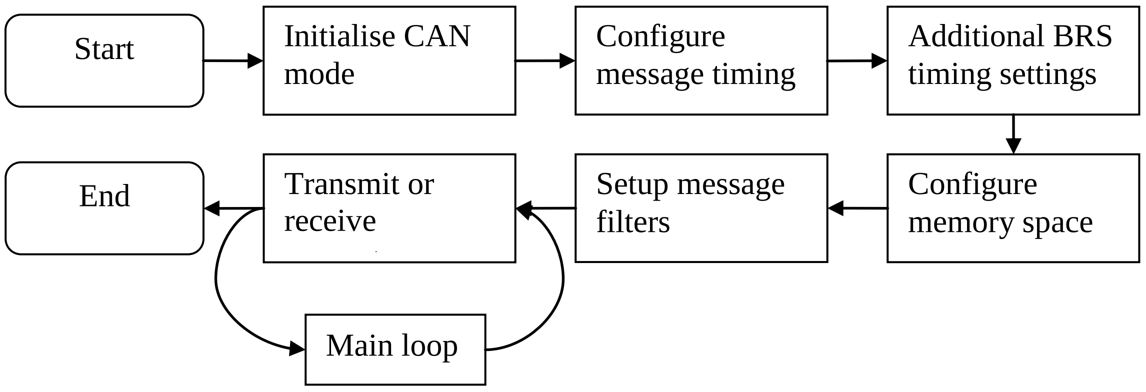


Figure 3 Steps to initialise and configure the MCAN module

4 Configuring the MCAN transmission scheme

4.1 Description

The backward compatible MCAN module on the MPC5777M can operate under the CAN 2.0 specification, and make use of the new CAN FD features. The module must be configured for CAN according to ISO11898-1, CAN FD or CAN FD with BRS. Timing parameters must be applied to the module, depending upon the scheme chosen. These timing constants define the time for a single bit and are used for the transmitter and receiver, so that they are configured identically.

The initialisation must also set the input and output pins to operate with the MCAN module(s).

All the MCAN modules share an auxiliary clock source which has a maximum frequency of 50 MHz on MPC5777M. Each module can be configured with its own operation scheme and timing parameters, which allows an MCU to operate on as many different buses as there are nodes at different bit rates.

4.2 Implementation

For the following examples, the auxiliary clock source will be configured as 40 MHz, taken directly from XOSC. This will allow the examples to run up to a maximum bit rate of 8 Mbps. The clock provided to the MCAN modules can be configured with the MPC5777M's Clock Generation Module (MC_CGM) configuration registers (AC8).

For transmitting and receiving a CAN FD message, the module must be configured for CAN FD with a single set of timing parameters provided. The scheme is selected by use of the CAN Control Register (CCCR) and the timing by the Bit Timing and Prescaler Register (BTP).

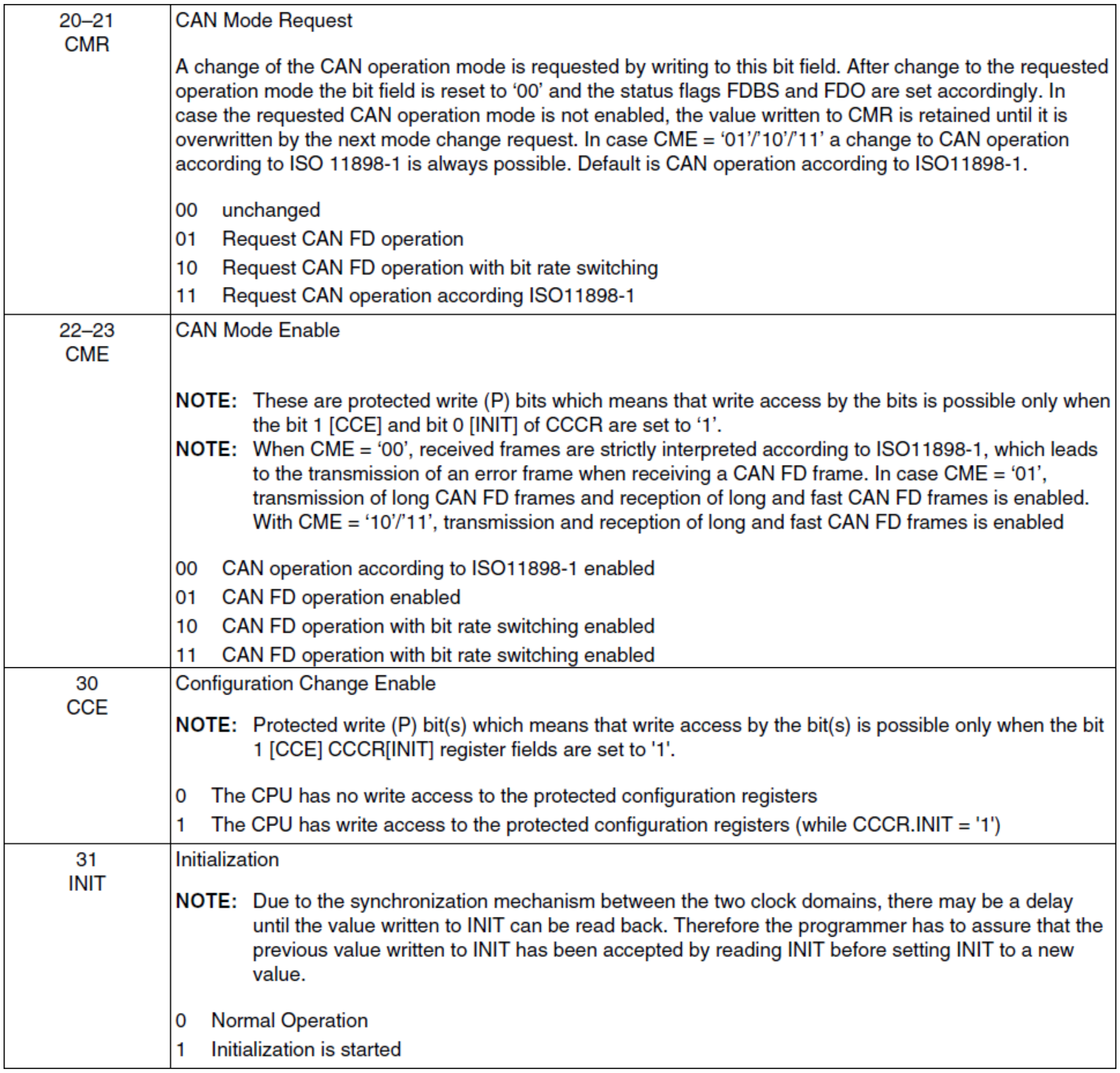
Note that the CCCR and BTP registers must be unlocked in order to change the configuration settings. The locking mechanism prevents changes being made while the module is operating. They can be unlocked by the CCE and INIT bits of the CCCR register.

Figure 4 shows that the INIT bit of the CCCR register operates in a different clock domain to the CPU and so care must be taken when attempting to change this bit. A suitable methodology is detailed in step 1 and step 2 of Table 1.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0															
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	TXP	FDBS	FDO	CMR		CME		TEST	DAR	MON	CSR	CSA	ASM	CCE	INIT
W	▨		▨	▨									▨			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

M_CAN_CCCR field descriptions



20–21 CMR	CAN Mode Request A change of the CAN operation mode is requested by writing to this bit field. After change to the requested operation mode the bit field is reset to ‘00’ and the status flags FDBS and FDO are set accordingly. In case the requested CAN operation mode is not enabled, the value written to CMR is retained until it is overwritten by the next mode change request. In case CME = ‘01’/’10’/’11’ a change to CAN operation according to ISO 11898-1 is always possible. Default is CAN operation according to ISO11898-1. 00 unchanged 01 Request CAN FD operation 10 Request CAN FD operation with bit rate switching 11 Request CAN operation according ISO11898-1
22–23 CME	CAN Mode Enable **NOTE:** These are protected write (P) bits which means that write access by the bits is possible only when the bit 1 [CCE] and bit 0 [INIT] of CCCR are set to ‘1’. **NOTE:** When CME = ‘00’, received frames are strictly interpreted according to ISO11898-1, which leads to the transmission of an error frame when receiving a CAN FD frame. In case CME = ‘01’, transmission of long CAN FD frames and reception of long and fast CAN FD frames is enabled. With CME = ‘10’/’11’, transmission and reception of long and fast CAN FD frames is enabled 00 CAN operation according to ISO11898-1 enabled 01 CAN FD operation enabled 10 CAN FD operation with bit rate switching enabled 11 CAN FD operation with bit rate switching enabled
30 CCE	Configuration Change Enable **NOTE:** Protected write (P) bit(s) which means that write access by the bit(s) is possible only when the bit 1 [CCE] CCCR[INIT] register fields are set to '1'. 0 The CPU has no write access to the protected configuration registers 1 The CPU has write access to the protected configuration registers (while CCCR.INIT = '1')
31 INIT	Initialization **NOTE:** Due to the synchronization mechanism between the two clock domains, there may be a delay until the value written to INIT can be read back. Therefore the programmer has to assure that the previous value written to INIT has been accepted by reading INIT before setting INIT to a new value. 0 Normal Operation 1 Initialization is started

Figure 4 MCAN_CCCR register and bit definitions

The bit time is defined by a multiple number of time quanta (tq), with 1 tq equal to the MCAN clock period. Each bit consists of a SYNC time, a Time Segment 1(TSEG1) and Time Segment 2(TSEG2). A 40 MHz MCAN clock will give a tq of 25 ns, with the programmed multipliers defining the length of these time segments. The SYNC value is always 1 and the remaining two segments can be user programmed. The receiver will sample and record the value of the transmitted bit between TSEG1 and TSEG2.

Address: 0h base + 1Ch offset = 1Ch

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0						BRP										0		TSEG1						TSEG2				SJW			
W	▨	▨	▨	▨	▨	▨											▨	▨														
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	1	1	0	0	1	1

M_CAN_BTP field descriptions

Field	Description
0–5 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
6–15 BRP	Baud Rate Prescaler (0x000-0x3FF) The value by which the oscillator frequency is divided for generating the bit time quanta. The bit time is built up from a multiple of this quanta. Valid values for the Baud Rate Prescaler are 0 to 1023. The actual interpretation by the hardware of this value is such that one more than the value prog rammed here is used. **NOTE:** Protected write (P) bit(s) which means that write access by the bit(s) is possible only when the bit 1 [CCE] CCCR[INIT] register fields are set to '1'.
16–17 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
18–23 TSEG1	(0x01-0x3F) Valid values are 1 to 63. The actual interpretation by the hardware of this value is such that one more than the prog rammed value is used. **NOTE:** Protected write (P) bit(s) which means that write access by the bit(s) is possible only when the bit 1 [CCE] CCCR[INIT] register fields are set to '1'.
24–27 TSEG2	(0x0-0xF) Valid values are 0 to 15. The actual interpretation by the hardware of this value is such that one more than the prog rammed value is used. **NOTE:** Protected write (P) bit(s) which means that write access by the bit(s) is possible only when the bit 1 [CCE] CCCR[INIT] register fields are set to '1'.
28–31 SJW	(0x0-0xF) Valid values are 0 to 15. The actual interpretation by the hardware of this value is such that one more than the value prog rammed here is used **NOTE:** Protected write (P) bit(s) which means that write access by the bit(s) is possible only when the bit 1 [CCE] CCCR[INIT] register fields are set to '1'.

Figure 5 MCAN_BTP register and bit definitions

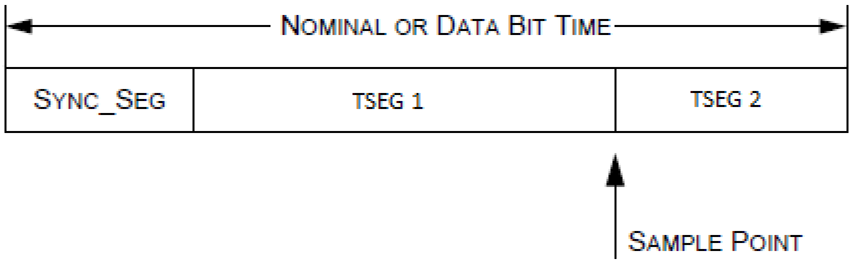


Figure 6 Timing breakdown for a single bit

Timing can be optimised for data speed on the bus, or to maintain data integrity on a reactive bus. This can be achieved by varying the ratio between TSEG1 and TSEG2 thus the sample point for a bit.

Stage 5 in Table 1 shows how to configure the BTP timing parameters for a bit time of 2 µs, and a sample point after 80% of the bit time, i.e. after 1.6 µs. The calculation below demonstrates how this is achieved by setting TSEG1 to 31 tq, TSEG2 to 8 tq and dividing the clock speed by 2.

$$BTP: tq = \frac{2}{40\ MHz} = 50\ ns, bit\ time = (1 + 31 + 8) * 50\ ns = 2\ \mu s\ \rightarrow 0.5\ Mbit/s$$

$$BTP\ sample\ point = \frac{SYNC + TSEG1}{SYNC + TSEG1 + TSEG2} = \frac{1 + 31}{1 + 31 + 8} = 80\%$$

Table 1 Steps to configure the MCAN scheme

Step	Operation	Description	Pseudo Code
1	Initialise MCAN_CCCR	Set INIT bit and check that it has been set	INIT = 1; If INIT ≠ 1, wait until it is
2	Unlock protected registers	Set CCCR.CCE bit	CCE = 1;
3	Request CAN mode change	Set CCCR.CMR to CAN FD	CMR = 1;
4	Change CAN mode	Set CCCR.CME to CAN FD	CME = 1;
5	Set BTP for 0.5 Mbps	Clk/2, T1 = 31, T2 = 8, SJW = 8	M_CAN_1.BTP.R = 0x00011E77;
6	Lock protected registers	Clear CCCR.CCE bit	CCE = 0;
7	Return MCAN module to normal operation	Clear INIT bit and check it has been cleared	INIT = 0; If INIT ≠ 0, wait until it is
8	Configure T_X pin	Enable output buffer with high drive strength and select pad mode	T_X pin MSCR = 0x32000001;
9	Configure R_X pin	Enable input buffer and select pad mode	R_X pin MSCR = 0x00080002;

4.3 Code

The following code provides an initialisation function for MCAN 1. It unlocks the protected registers, then puts the modules into CAN FD mode and provides timing parameters. The protected registers are locked again and finally the module pins are configured for use by the MCAN 1 module.

```
void MCAN1_init() {                /* Init MCAN1 */
M_CAN_1.CCCR.B.INIT = 0x1;         //set to initialisation mode
while(M_CAN_1.CCCR.B.INIT == 0);
M_CAN_1.CCCR.B.CCE = 0x1;          // enable CCE bit to change configuration
M_CAN_1.CCCR.B.CMR = 0x2;          //request CAN FD
M_CAN_1.CCCR.B.CME = 0x2;          //enable CAN FD
M_CAN_1.BTP.R = 0x00011E77;        //tq for 0.5Mbps SYNC=1, TSEG1=30+1, TSEG2=7+1, SJW=7+1
M_CAN_1.CCCR.B.CCE = 0x0; // disable CCE to prevent configuration changes
M_CAN_1.CCCR.B.INIT = 0x0;//Return to normal operation
while(M_CAN_1.CCCR.B.INIT == 0x1);
SIUL2.MSCR[10].R = 0x32000001;     //MCAN1TX Pin, PA[10]
SIUL2.MSCR[758].R = 0x00080002;    //MCAN1RX Pin, PA[11]
}
```

5 Assigning message memory space

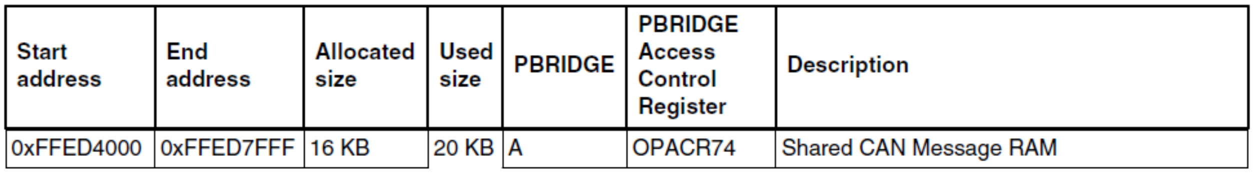
5.1 Description

Memory must be assigned during initialisation to define where to store received messages and where to store messages prior to transmission.

The quantity of data bytes per message must be configured to determine how much memory space is required per message. This can be configured for messages that will not contain as many as 64 bytes maximum allowed by CAN FD, resulting in more efficient memory usage and allowing more messages to be stored in the allocated memory space.

5.2 Implementation

Each MCAN module can configure its transmit buffer and receive FIFO memory space. They are configured with a start address offset and the number of memory elements to store. The starting address is predefined as the start address shown in Figure 7, and it is the user's responsibility to ensure that the number of elements per memory space does not cause them to overlap.



Start address	End address	Allocated size	Used size	PBRIDGE	PBRIDGE Access Control Register	Description
0xFFED4000	0xFFED7FFF	16 KB	20 KB	A	OPACR74	Shared CAN Message RAM

Figure 7 Shared memory and address allocation for MCAN modules

Table 2 shows the configuration of all message memory space for an MCAN module. Steps 2-4 setup the receive FIFOs and transmit buffer, and the data length allowance for each of those memory spaces in steps 5 and 6.

Table 2 Steps to configure the MCAN memory space

Step	Operation	Description	Pseudo Code
I	Unlock protected registers	Refer to section 4.2 'Implementation'	
1	Configure FIFO 0	Offset address = 0x400 Save 3 CAN messages Overwrite old instead of discarding new	Write M_CAN_1.RXF0C.R = 0x80030400;
2	Configure FIFO 1	Offset address = 0x800 Save 5 CAN messages Discard new messages when full	Write M_CAN_1.RXF1C.R = 0x00050800;
4	Configure T_X Buffer	Store 2 messages Offset address = 0xD00	Write M_CAN_1.T_XBC.R = 0x00020D00;
5	Set data limit for receive messages	48 data bytes/message FIFO 1 32 data bytes/message FIFO 0	Write M_CAN_1.RXESC.R = 0x00000065;

6	Set data limit for transmit messages	64 data bytes/message T_X buffer	Write M_CAN_1.TXESC.R = 0x00000007;
F	Lock protected registers	Refer to section 4.2 'Implementation'	

NOTE

The above registers are protected, and must be unlocked using the CCCR register, as demonstrated in section 4.2 'Implementation'.

6 Configuring CAN FD with bit rate switching

6.1 Description

To use Bit Rate Switching (BRS) the user is only required to configure an extra set of timing parameters and configure the module to use a different CAN scheme.

The original Bit Timing and Prescaler Register (BTP) timing parameters are used only for the arbitration phase of the message. The additional set is used for the data phase. The data phase bit timings are defined in exactly the same way as the arbitration message timing is configured using the Fast Bit Timing Prescaler register (FBTP).

Using the parameters demonstrated in Table 3, the calculation below shows how to achieve a bit time of 125 ns with a sample point after 80% of this bit time. Note that the clock divider is unused and default to ÷1.

$$FBTP: tq = \frac{1}{40\ MHz} = 25\ ns, bit\ time = (1+3+1)*25\ ns = 125\ ns\ \rightarrow 8\ Mbps$$

$$FBTP\ sample\ point = \frac{SYNC + TSEG1}{SYNC + TSEG1 + TSEG2} = \frac{1+3}{1+3+1} = 80\%$$

6.2 Transceiver delay compensation (TDC)

The CAN transceiver always compares what is seen by the receiver and what it is trying to transmit in order to identify when a more dominant node has control of the bus. This can cause issues during the faster data phase, when, due to delays on the CAN bus, the transceiver sees a difference between what is transmitting and what is being received. This will cause the transmitter to believe it does not have control of the bus and it will abort the transmission.

To prevent this from occurring, delay compensation can be configured for the MCAN node, allowing a delay between the transmitter circuitry applying a bit to the bus and the receiver reading it back from the bus. This delay is specific to the CAN Physical Layer (PHY) and bus, so when the correct delay has been determined, it can be stored to always ensure correct operation.

If the correct delay for the bus is unknown, the message may not be transmitted because the transceiver believes it has been dominated, and it does not indicate an error, instead the Transmit Buffer Request Pending register (TXBRP) will never clear to zero.

To find the correct delay for the bus, a trial and error method is the only way to realistically find the delay value. This process is only suitable for development and it would be unwise to use it in a real application; the result being part messages on the bus and time to tune the Transmit Delay Compensation Offset (TDCO) to allow the module to transmit and receive successfully. Instead the appropriate TDCO value should be found using this process during development, and that value programmed into the application device code. The process can be summarised as shown in Figure 8.

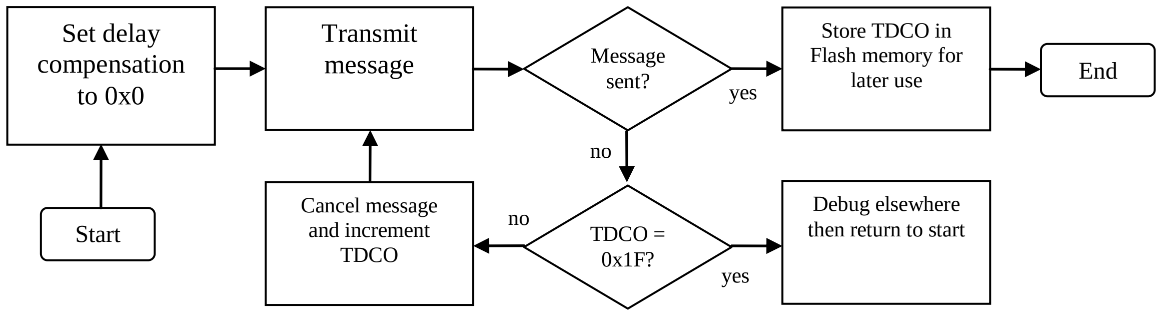

Set delay compensation to 0x0
Transmit message
Message sent?
yes
Store TDCO in Flash memory for later use
End
no
Start
Cancel message and increment TDCO
no
TDCO = 0x1F?
yes
Debug elsewhere then return to start

Figure 8 Finding the correct TDCO

The length of the delay can be found simply by increasing the number of time quanta from 0x0 to 0x1F until the messages can be sent and the correct delay has been found. If the TDCO value reaches 0x1F and no message has been sent successfully, it is more likely that the transmission fault lies elsewhere.

The delay value may not be a single value, rather a range of values that are suitable for successful transmission. It is a good practice to find the minimum and maximum delay values and then select a delay that is median of the range.

Table 3 shows the additional configuration steps for use of BRS, changing the mode of operation of the module and programming the fast bit timing parameters with a transceiver delay compensation value.

Table 3 Program FBTP register for BRS and apply a TDCO value

Step	Operation	Description	Pseudo Code
I	Unlock protected registers	Refer to section 4.2 'Implementation'	
1	Request CAN mode change	Set CCCR.CMR to CAN FD with BRS	CMR = 2;
2	Change CAN mode	Set CCCR.CME to CAN FD with BRS	CME = 2;

3	Set FBTP for 0.5 Mbps	Clk/1, T1 = 3, T2 = 1, SJW = 2
		TDC = ON
		TDCO = 5
			M_CAN_1.FBTP.R = 0x05800201;

F	Lock protected registers	Refer to section 4.2 'Implementation'	

7 Filtering messages by ID number

7.1 Introduction

Received messages can be filtered by ID number into either of the receive FIFOs.

The destination of ID numbers that do not match with any filter is decided by the Global Filter Configuration register (GFC). This register will also determine the destination of remote frames which request data from a node without sending any data.

7.2 Implementation

Up to 128 filter configurations can be implemented on each MCAN module in the device. To implement a classic filter for 11-bit ID numbers, a filter mask, ID comparison, and the destination FIFO for the message must be selected.

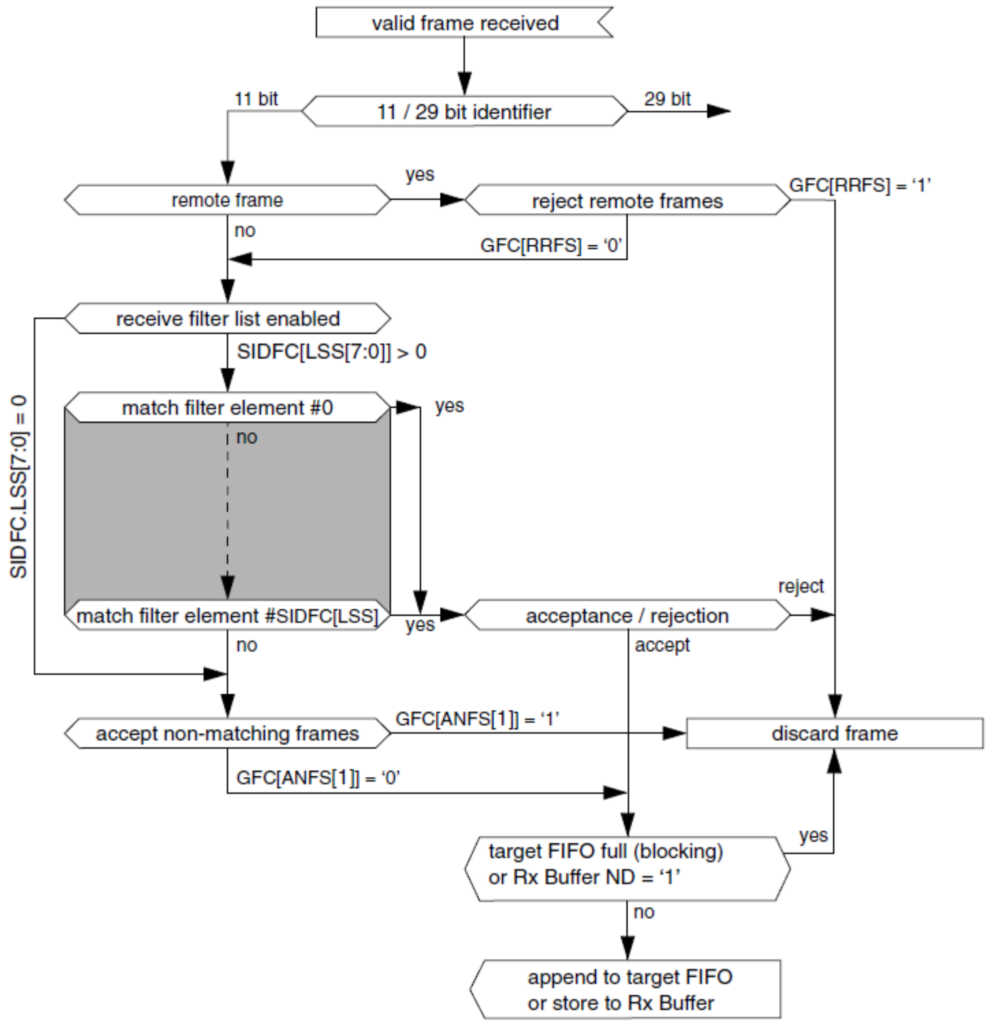


Figure 9 11-bit ID filter comparison path

Figure 9 demonstrates how messages are sorted. Focusing only on messages with 11-bit IDs and no remote frames, a received message is compared to each filter element in turn. The first element that matches with the message will decide the memory address where the message will be stored. No more comparisons will be made after the first positive result. If no positive result is obtained, the message can either be discarded or stored in a particular FIFO, depending on the MCAN_GFC register.

Figure 10 shows the five most significant bits of the filter mask will be considered (bits 21-25). As the masks are identical, matching of the ID number (bits 5-15) will determine if the received message is placed into FIFO 0 (top filter) or FIFO 1 (bottom filter), selected with bits 2-4. Bits 0-1 set the configuration to classic filter.

0	1	2		4	5										15	16				20	21				25						31
filter		FIFO			ID											Reserved					Filter Mask										
8				F				4				0				0				7				C				0			
1	0	0	0	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0
0x2		0x1			0x740																0x7C0										
0	1	2		4	5										15	16				20	21				25						31
filter		FIFO			ID											Reserved					Filter Mask										
9				7				C				0				0				7				C				0			
1	0	0	1	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0
0x2		0x2			0x7C0																0x7C0										

Figure 10 Example ID Filter configuration

Other filter settings are detailed in the section titled 'Standard message ID Filter element' of the MPC5777M reference manual.

Figure 11 shows how these two filters will select the destination of the received message. The masked ID number is compared to the filter ID. An exact match produces a positive result, selecting the message destination.

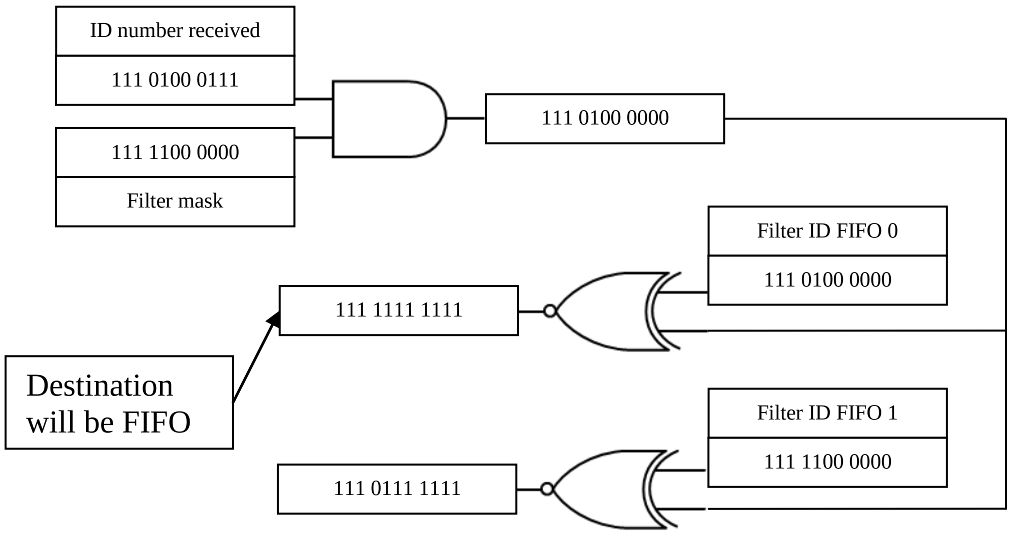


Figure 11 Application of a classic filter configuration to a received ID number

Location of the filter elements in RAM is programmed by the Standard ID Filter Configuration register (SIDFC). The base address (FLSSA) and number of elements (LSS) the module uses for comparisons must be provided.

Address: 0h base + 84h offset = 84h

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0								LSS								FLSSA														0	
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

M_CAN_SIDFC field descriptions

Field	Description
0–7 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
8–15 LSS	List Size Standard 0 No standard Message ID filter 1-128 Number of standard Message ID filter elements >128 Values greater than 128 are interpreted as 128 **NOTE:** Protected write (P) bit(s) which means that write access by the bit(s) is possible only when the bit 1 [CCE] CCCR[INIT] register fields are set to '1'.
16–29 FLSSA	Filter List Standard Start Address Start address of standard Message ID filter list (32-bit word address, see Message RAM). **NOTE:** Protected write (P) bit(s) which means that write access by the bit(s) is possible only when the bit 1 [CCE] CCCR[INIT] register fields are set to '1'.
30–31 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.

Figure 12 Programming the filter element memory locations

The FIFO start addresses are defined by the Receiver FIFO Configuration registers (RXFnC), see section 5 ‘Assigning message memory space’. For 29-bit ID lengths, the Extended ID Filter Configuration register (XIDFC) must be programmed in a similar way to the SIDFC register. Each 29-bit filter element configuration takes two words; for the filter and for the mask configuration, that is described in detail in ‘Extended message ID filter element’ of the MPC5777M reference manual.

Table 4 shows how to configure the module to use two filters, and then place those filters into memory. The code listing demonstrates placing some example filters into memory for two MCAN modules.

Table 4 Program the filters and their location in memory

Step	Operation	Description	Pseudo code
I	Unlock protected registers	Refer to section 4.2 ‘Implementation’	
1	Configure MCAN2 filter location and quantity	Offset address = 0xF00 No. of filters = 2	Write M_CAN_2.SIDFC.R = 0x00020F00;
I	Lock protected registers	Refer to section 4.2 ‘Implementation’	

2	Program MCAN2 filter 1	Classic filter
		Destination = FIFO 0
		ID = 0x740
		Mask = 0x7C0
			Write (0x0F00) = 0x8F0007C0;

3	Program MCAN2 filter 2	Classic filter
		Destination = FIFO 1
		ID = 0x7C0
		Mask = 0x7C0
			Write (0x0F04) = 0x978007C0;

7.3 Code

The following function programs example filters for the receiver module, with the resulting filters detailed in Table 5.

```
#define FULL_SRAM_BASE_ADDR_MSG_RAM 0xFFED4000       //MCAN shared SRAM base address
#define MCAN1_SIDFC_FLSSA 0x0000                     //MCAN 1 STANDARD MESSAGE ID FILTERS
#define MCAN2_SIDFC_FLSSA 0x0F00                     //MCAN 2 STANDARD MESSAGE ID FILTERS
void MCAN_ID_init() {
 //MCAN1_ID values
 *(vuint32_t*)(FULL_SRAM_BASE_ADDR_MSG_RAM + MCAN1_SIDFC_FLSSA) =  0x8F0007C0;
/*#1 Rx 11-bit Filter FIFO0*/
 *(vuint32_t*)(FULL_SRAM_BASE_ADDR_MSG_RAM + MCAN1_SIDFC_FLSSA + 0x04) =  0x978007C0;
/*#2 Rx 11-bit Filter FIFO1*/
 //MCAN2_ID values
 *(vuint32_t*)(FULL_SRAM_BASE_ADDR_MSG_RAM + MCAN2_SIDFC_FLSSA) = 0x8F4007C0;
/*#3 Rx 11-bit Filter FIFO 0*/
 *(vuint32_t*)(FULL_SRAM_BASE_ADDR_MSG_RAM + MCAN2_SIDFC_FLSSA + 0x04) =  0x97C007C0;
/*#4 Rx 11-bit Filter FIFO1*/
}
```

Table 5 Filters programmed by the code above

Filter element	Filter type	Destination	ID comparison	Mask
MCAN1 #1	Classic	MCAN1 FIFO 0	0x700	0x7C0
MCAN1 #2	Classic	MCAN1 FIFO 1	0x780	0x7C0
MCAN2 #3	Classic	MCAN2 FIFO 0	0x740	0x7C0
MCAN2 #4	Classic	MCAN2 FIFO 1	0x7C0	0x7C0

8 Transmitting, receiving and cancelling messages

8.1 Introduction

For a module to transmit a message, the message is formed within the defined memory space and the transmission is initiated. The message is sent by the node to a receiver, where it is formed in the receiver memory space.

8.2 Implementation

This example will configure MCAN modules 1 and 2. Section 4 'Configuring the MCAN transmission scheme' describes how to configure the memory space for the modules. As they share the same SRAM space, each module must be configured to use different address ranges. An example of this is shown in Table 6.

The message to be transmitted is formed in the transmit buffer following the format in Figure 13. The size and location of the MCAN transmit buffer is defined by a start address and number of buffer elements in the Transmit Buffer Configuration register (TXBC) and the maximum number of data bytes per message in the Transmit buffer Element Size Configuration register (TXESC) (see section 5 'Assigning message memory space').

T2 to T17 is the maximum allocated data space, DLC defines how many of these bytes will be taken, for example, if DLC is 0x7, but 64 bytes of data space has been allocated by the TXESC register, the module will take the first seven bytes, from DB0 to DB6.

Table 6 DLC codes

DLC code		No. of Bytes sent	DLC code		No. of Bytes sent
0x0	0000	0	0x8	1000	8

0x1	0001	1	0x9	1001	12
0x2	0010	2	0xA	1010	16
0x3	0011	3	0xB	1011	20
0x4	0100	4	0xC	1100	24
0x5	0101	5	0xD	1101	32
0x6	0110	6	0xE	1110	48
0x7	0111	7	0xF	1111	64

Only CAN FD messages are able to send messages with more than 8 data bytes, so to use DLC codes of 0x9 to 0xF the module must be configured for CAN FD in the CCCR register (see section 4 'Configuring the MCAN transmission scheme').

The receiver buffer follows the same format, but with additional bits to give further information about the message that has been received. The Accept Non-Matching Frames bit (ANMF) and Filter Index bits (FIDX), shown in Figure 14, combine to indicate which filter element the message ID matched with which could be used to give a particular message type. A time stamp can be attached to the message (refer to section 'Timestamp Counter Configuration Register' in MPC5777M Reference Manual).

	0 7	8 15	16 23	24 31
T0	ESI | XTD | RTR | ID[28:0]			
T1	MM[7:0]	EFC | res | DLC[3:0]	res	
T2	DB3[7:0]	DB2[7:0]	DB1[7:0]	DB0[7:0]
T3	DB7[7:0]	DB6[7:0]	DB5[7:0]	DB4[7:0]
...	...	...	...	...
Tn	DBm[7:0]	DBm-1[7:0]	DBm-2[7:0]	DBm-3[7:0]

T0 Bit 1 XTD: Extended Identifier	0 11-bit standard identifier 1 29-bit extended identifier
T0 Bit 2 RTR: Remote Transmission Request	0 Transmit data frame 1 Transmit remote frame **NOTE:** When RTR = 1, the M_CAN transmits a remote frame according to ISO11898-1, even if CCCR.CME enables the transmission in CAN FD format.
T0 Bits 3:31 ID[28:0]: Identifier	Standard or extended identifier depending on bit XTD. A standard identifier has to be written to ID[28:18].
T1 Bits 0:7 MM[7:0]: Message Marker	Written by CPU during Tx Buffer configuration. Copied into Tx Event FIFO element for identification of Tx message status.
T1 Bit 8 EFC	Event FIFO Control 0 Don't store Tx events 1 Store Tx events
T1 Bits 12:15 DLC[3:0]	Data Length Code 0-8 Transmit frame with 0-8 data bytes 9-15 Transmit frame with 8 data bytes 9-15 CAN FD: transmit frame has 2/16/20/24/32/48/64 data bytes
T2 Bits 0:7	DB3[7:0]: Data Byte 3
T2 Bits 8:15	DB2[7:0]: Data Byte 2
T2 Bits 16:23	DB1[7:0]: Data Byte 1
T2 Bits 24:31	DB0[7:0]: Data Byte 0
T3 Bits 0:7	DB7[7:0]: Data Byte 7
T3 Bits 8:15	DB6[7:0]: Data Byte 6
T3 Bits 16:23	DB5[7:0]: Data Byte 5
T3 Bits 24:31	DB4[7:0]: Data Byte 4
....	....
Tn Bits 0:7	DBm[7:0]: Data Byte m
Tn Bits 8:15	DBm-1[7:0]: Data Byte m-1
Tn Bits 16:23	DBm-2[7:0]: Data Byte m-2
Tn Bits 24:31	DBm-3[7:0]: Data Byte m-3

Figure 13 Transmit buffer element

	0 … 7	8 … 15	16 … 23	24 … 31
R0	ESI | XTD | RTR | ID[28:0]			
R1	ANMF | FIDX[6:0]	res | EDL | BRS | DLC[3:0]	RXTS[15:0]	
R2	DB3[7:0]	DB2[7:0]	DB1[7:0]	DB0[7:0]
R3	DB7[7:0]	DB6[7:0]	DB5[7:0]	DB4[7:0]
	...	...	...	...
Rn	DBm[7:0]	DBm-1[7:0]	DBm-2[7:0]	DBm-3[7:0]

R0 Bit 0 ESI: Error State Indicator	0 Transmitting node is error active 1 Transmitting node is error passive
R0 Bit 1 XTD: Extended Identifier	Signals to the Host whether the received frame has a standard or extended identifier 0 11-bit standard identifier 1 29-bit extended identifier
R0 Bit 2 RTR: Remote Transmission Request	Signals to the Host whether the received frame is a data frame or a remote frame. 0 Received frame is a data frame 1 Received frame is a remote frame **NOTE:** There are no remote frames in CAN FD format. In case a CAN FD frame was received (EDL = '1'), bit RTR reflects the state of the reserved bit r1.
R0 Bits 3:31 ID[28:0]: Identifier	Standard or extended identifier depending on bit XTD. A standard identifier is stored into ID[28:18].
R1 Bit 0 ANMF: Accepted Non-matching Frame	Acceptance of non-matching frames may be enabled via GFC[ANFS] and GFC[ANFE] 0 Received frame matching filter index FIDX 1 Received frame did not match any Rx filter element
R1 Bits 1:7 FIDX[6:0]: Filter Index	0-127=Index of matching Rx acceptance filter element (invalid if ANMF = '1'). Range is 0 to SIDFC[LSS] - 1 resp. XIDFC[LSE] - 1.
R1 Bit 10 EDL Extended Data Length	0 Standard frame format 1 CAN FD frame format (new DLC-coding and CRC)
R1 Bit 11 BRS Bit Rate Switch	0 Frame received without bit rate switching 1 Frame received with bit rate switching
R1 Bits 12:15 DLC[3:0]: Data Length Code	0-8 CAN + CAN FD: Received frame has 0-8 data bytes 9-15 CAN: Received frame has 8 data bytes 9-15 CAN FD: received frame has 12/16/20/24/32/48/64 data bytes
R1 Bits 16:31 RXTS[15:0]: Rx Timestamp	Timestamp Counter value captured on start of frame reception. Resolution depending on configuration of the Timestamp Counter Prescaler TSCC[TCP].
R2 Bits 0:7	DB3[7:0]: Data Byte 3
R2 Bits 8:15	DB2[7:0]: Data Byte 2
R2 Bits 16:23	DB1[7:0]: Data Byte 1
R2 Bits 24:31	DB0[7:0]: Data Byte 0
R3 Bits 0:7	DB7[7:0]: Data Byte 7
R3 Bits 8:15	DB6[7:0]: Data Byte 6
R3 Bits 16:23	DB5[7:0]: Data Byte 5
R3 Bits 24:31	DB4[7:0]: Data Byte 4
...	...
Rn Bits 0:7	DBm[7:0]: Data Byte m
Rn Bits 8:15	DBm-1[7:0]: Data Byte m-1
Rn Bits 16:23	DBm-2[7:0]: Data Byte m-2
Rn Bits 24:31	DBm-3[7:0]: Data Byte m-3

Figure 14 Receiver buffer element

A transmission is initiated by writing the number of prepared messages to the Transmit Buffer Add Request register (TXBAR). This will be ignored if the Transmit Buffer Request Pending register (TXBRP) is greater than 0, meaning a message(s) is already queued.

If a message transmission has failed in CAN FD BRS mode because the delay compensation is incorrect, the TXBRP register will be greater than 0 and no new messages are able to be sent, therefore the message must be cancelled before a new message can be queued. Messages can be cancelled by writing the number of messages to be cancelled into the TXBCR register. For example; if two messages were queued, but did not send, 0x2 should be written to the TXBCR register to cancel them both.

Table 7 and the following code listing show how to apply the message to be sent to the transmit buffer and prompt the module to send the message.

Table 7 Constructing and sending a CAN message

Step	Operation	Description	Pseudo Code
1	Transmit from MCAN 1	11 bit ID number = 5	T0 addr = 0x00140000;
		DLC = 4	T1 addr = 0x00040000;
		Insert data into buffer	T2 addr = 0xAA55AA55;
2	Initiate transmission	Apply number of messages for transmission to TXBAR	M_CAN_1.TXBAR.R = 0x1;

8.3 Code

The following code is an example of how to construct a 4 bytes message, with DLC and ID in the transmit buffer space. Finally it adds the message to the transmit queue by writing to the TXBAR register.

```
#define FULL_SRAM_BASE_ADDR_MSG_RAM 0xFFED4000       //MCAN shared SRAM base address
void MCAN1_tx() {
if (M_CAN_1.TXBRP.R == 0) {
int id = 5;
int dlc = 4;
char data[4] = {0xAA, 0x55, 0xAA, 0x55};
*(vuint32_t*)(FULL_SRAM_BASE_ADDR_MSG_RAM + 0x0D00 ) =  (id << 18);
/* Tx Buffer T0 */
*(vuint32_t*)(FULL_SRAM_BASE_ADDR_MSG_RAM + 0x0D04 ) = (dlc << 16);
/* Tx Buffer T1 */
*(vuint32_t*)(FULL_SRAM_BASE_ADDR_MSG_RAM + 0x0D08) = data[0];
/*T2 DB0*/
*(vuint32_t*)(FULL_SRAM_BASE_ADDR_MSG_RAM + 0x0D08) = data[1]<<8;
/*T2 DB1*/
*(vuint32_t*)(FULL_SRAM_BASE_ADDR_MSG_RAM + 0x0D08) = data[2]<<16;
/*T2 DB2*/
*(vuint32_t*)(FULL_SRAM_BASE_ADDR_MSG_RAM + 0x0D08) = data[3]<<24;
/*T2 DB3*/
M_CAN_1.TXBAR.R = 0x1;    /*start transmission*/
}
}
```

9 Conclusion

This application note has described all the steps required to configure an MCAN module to send and receive CAN, CAN FD and CAN FD BRS messages, forming the basis of a full MCAN module driver, including:

- Configure a module to transmit CAN 2.0 messages and observe with an oscilloscope or CAN bus analyser
- Configure two nodes to send CAN 2.0 messages between them, filter based on ID numbers into different memory locations
- Send CAN FD frames of up to 64 bytes and verify that the full message is accepted and stored correctly
- Introduce BRS to the CAN bus and tune the TDCO value for reliable communication. Verify that the message rate is greater at a higher bit rate.

Document Number AN5045
Revision 0, 11/2014