TB3266

Basic Configuration of the PIC18 CAN FD Module

Introduction

Controller Area Network Flexible Datarate (CAN FD) is a new version of the Controller Area Network (CAN) specification that allows for more data throughput while keeping the advantages of the CAN bus. The CAN FD module on PIC18F devices has a number of new features that make configuring the module quite different from previous CAN modules. This document will outline the basics of setting up the CAN FD module to both transmit and receive CAN FD frames.

Table of Contents

Introduction......1

1. Differences between CAN FD and CAN 2.0......3

2. CAN Modes and Mode Changes...... 4
    2.1. Configuration Mode......4
    2.2. Normal Modes......4

3. Baud Rate Setup......5
    3.1. Clock Setup and Requirements......5
    3.2. Nominal Bit Rate (NBR)...... 5
    3.3. Data Bit Rate (DBR)......6
    3.4. Bit Rate Setup With the Microchip Code Configurator (MCC)......7

4. FIFOs and Memory Usage......8
    4.1. Memory Setup and FIFOBA registers...... 8
    4.2. Transmit Event FIFO......8
    4.3. Transmit Queue (TXQ)......8
    4.4. Configurable FIFOs 1-3......8
    4.5. Memory Setup with MCC...... 9

5. Transmission......10
    5.1. Writing to the Transmit FIFO...... 10
    5.2. Requesting Transmit......11
    5.3. MCC Transmit APIs......11

6. Reception......13
    6.1. Mask and Filter Setup...... 13
    6.2. Reading from Receive FIFOs......13
    6.3. CAN Reception Interrupts...... 14
    6.4. MCC Reception Setup/APIs......14

7. Conclusion...... 16

The Microchip Website......17

Product Change Notification Service......17

Customer Support...... 17

Microchip Devices Code Protection Feature......17

Legal Notice...... 17

Trademarks...... 18

Quality Management System...... 18

Worldwide Sales and Service......19

1. Differences between CAN FD and CAN 2.0

CAN FD differs from CAN 2.0 in two primary ways. The first and most notable is that CAN FD has two baud rates instead of one. The first is the nominal baud rate, which is used for the ID, overhead, and the CRC, and has the same data rate limitations as CAN 2.0. The second is the data baud rate, which is used exclusively for the data bytes, and can be a much higher data rate. In addition, the data section of a CAN FD frame can be up to 64 bytes, up from the 8 byte maximum of CAN 2.0.

The other differences between the two protocols primarily are more minor and several stem from the two major changes, namely:

- The addition of BRS and FDF bits to indicate CAN FD frames/speeds and the removal of the RTR (remote transmit request) bit.
- The CRC has been increased from 15 bits to either 17 or 21 bits.

The CAN FD module in PIC18 devices has had several major changes since the CAN 2.0 module, both in interfacing with the new CAN FD hardware and handling the changes to CAN FD protocol.

2. CAN Modes and Mode Changes

The CAN FD module has several modes of operation:

- Configuration mode
- Normal (CAN FD) mode
- Normal (CAN 2.0) mode
- Disable mode
- Listen Only mode
- Restricted Operation mode
- Internal Loopback mode
- External Loopback mode

Most of these modes are primarily useful in development, error or debug situations. The primary modes of concern for basic configuration and transmission/reception are Configuration mode and the Normal modes.

2.1 Configuration Mode

Configuration mode is entered by setting the REQOP[2:0] bits of the C1CONT register to `0b100`. Configuration mode is active when the OPMOD[2:0] bits of the C1CONU register are equal to `0b100`.

The following registers and bit fields can only be programmed during Configuration mode:

- C1CON: WAKFIL, CLKSEL, PXEDIS, ISOCRECEN, TXQEN, STEF, SERRLOM, ESIGM, RTXAT
- C1NBTCFG, C1DBTCFG and C1TDC
- C1TXQCON: PLSIZE[2:0], FSIZE[4:0]
- C1FIFOCON: TXEN, RXTSEN, PLSIZE[2:0], FSIZE[4:0]
- C1TEFCON: TEFTSEN, FSIZE[4:0]
- CxFIFOBA

In addition, all FIFOs and the TXQ are Reset in Configuration mode.

2.2 Normal Modes

There are two normal modes:

- CAN FD normal mode, entered by setting the REQOP[2:0] bits of C1CONT to `0b000`
- CAN 2.0 normal mode, entered by setting the REQOP[2:0] bits of C1CONT to `0b110`

Configuration mode must be exited and normal mode must be entered before the CAN FD will perform any transmissions or receptions. The CAN FD normal mode can send either CAN FD or CAN 2.0 frames, depending on the setting of the FDF bit of the CAN packet. Comparatively, the CAN 2.0 normal mode can only send CAN 2.0 frames and should only be used if the device is to be used on an exclusively CAN 2.0 bus.

3. Baud Rate Setup

The first element of setting up the CAN FD module is to set up the baud rate(s) for communication. CAN FD has two separate baud rates and as such has two separate sets of registers to configure them:

- Nominal Bit Rate (NBR), which uses the C1NBTCFGT/U/H/L registers
- Data Bit Rate (DBR), which uses the C1DBTCFGT/U/H/L registers

3.1 Clock Setup and Requirements

The CAN clock can be chosen from one of two sources: the system clock or the external clock. In many cases, these two clocks will be the same, with one notable exception. The external clock setting bypasses the 4x PLL, allowing for the CAN to be clocked directly by an external clock, while the device is clocked by the output of the 4x PLL circuit.

The CAN FD module is recommended to be clocked only by specific input frequencies, specifically 10 MHz, 20 MHz or 40 MHz. This clocking restriction is an inherent limitation of the CAN FD hardware and can lead to unexpected and/or undesirable behavior, if not followed. However, using other clock frequencies is possible for debugging and development, although inadvisable for actual applications. Applications should use an external crystal/clock at one of the aforementioned recommended frequencies.

3.2 Nominal Bit Rate (NBR)

The Nominal Bit Rate (NBR) is the bit rate of the arbitration section before the data and the CRC section after the data. It is limited by the propagation delay of the physical CAN bus, much like with CAN 2.0, and as such has lower maximum limits on its data rate.

3.2.1 Nominal Bit Rate Basics

Setting up the Nominal Bit Rate is done in several steps, the first of which is choosing a Nominal Time Quanta (NTQ). Each CAN bit is broken down into a specific number of these Time Quanta (TQ), and further subdivided into smaller segments that are similarly divided into Time Quanta. Determining the Nominal Time Quanta is selected by the BRP bits of the C1NBTCFGT register using the following equation:

Equation 3-1. Nominal Time Quanta

$$NTQ = NBRP \times T_{SYSCLK} = \frac{NBRP}{F_{SYSCLK}}$$

Where NTQ is the Nominal Time Quanta, NBRP is the value of the BRP bits of the C1NBTCFGT register, and F_{SYSCLK} is the selected CAN clock.

Once the Nominal Time Quanta is determined, it can be used alongside the intended bit rate to determine the number of Time Quanta to use per bit time with the following equation:

Equation 3-2. Time Quanta Per Bit

$$Number\ of Time\ Quanta\ \ Per\ Bit = \frac{1}{NTQ\ \times Bit\ Rate}$$

From there, each bit time is broken down into four major segments:

- Synchronization Segment
- Propagation Segment
- Phase Segment 1
- Phase Segment 2

The Synchronization Segment is the section of the bit time in which the edge is expected to occur. It is the only segment that is not configurable and is always hard coded as 1 Time Quanta. The Propagation Segment is compensation for the propagation delay on the physical CAN bus. This segment must be held longer than the maximum propagation delay of the signal on the bus. Phase Segments 1 and 2 serve the purpose of determining when the bit is sampled: the sample point occurs on the transition from Phase Segment 1 to Phase Segment 2. The two Phase Segments can also be lengthened or shortened to adjust for phase shifts in the edges of the bits.

Configuring the individual segments is performed by bits in the C1NBTCFGU and C1NBTCFGH registers. The Propagation Segment and Phase Segment 1 are combined to TSEG1, which is controlled through the TSEG1 bits of C1NBTCFGU, and Phase Segment 2 is controlled by the TSEG2 bits of the C1NBTFGH register. As previously mentioned, the Synchronization Segment is fixed at 1 Time Quanta, so combining these configurations with the previous equations gives us the following equation:

Equation 3-3. Calculating TSEG1, TSEG2 and BRP

$$\frac{FSYSCLK}{(BRP + 1) \times Bit\ Rate} = TSEG1 + TSEG2 + 1$$

Where any configured values of TSEG1, TSEG2 and BRP that make the equation true, are valid options. For example, with an input clock of 40 MHz and a desired bit rate of 500 kHz, it can be reduced to:

Equation 3-4. TSEG1, TSEG2 and BRP Example

$$80 = (BRP + 1)(TSEG1 + TSEG2 + 1)$$

From here, there are a few considerations in determining BRP, TSEG1 and TSEG2.

3.2.2 BRP Considerations

For BRP, it is generally recommended to choose the smallest value that will still allow the equation to work, given TSEG1 and TSEG2 limitations. This is because doing so gives more fine control over the sample point (smaller Time Quanta size).

3.2.3 TSEG1 and TSEG2 Considerations

The two considerations for TSEG1 and TSEG2 are:

1. Propagation Time
2. Sample Point

Propagation time is part of TSEG1, and as such TSEG1 needs to be at least greater than the maximum propagation delay of the bus, which is given by the following equation.

Equation 3-5. Maximum Propagation Delay

$$T_{PROP} = 2 \times (t_{TXD-RXD} + T_{BUS})$$

Where $t_{TXD\text{-}RXD}$ is the propagation delay of the transceiver and T_{BUS} is the delay of the CAN bus.

The sample point is the percentage through the bit at which the bit is sampled, located at the point between TSEG1 and TSEG2.

3.2.4 Synchronization Jump Width (SJW)

The final piece of bit rate setup is the Synchronization Jump Width (SJW). This is used in resynchronization when the starting edge of an incoming bit does not fall within the Synchronization Segment. In this case, PHSEG1 or PHSEG2 can be adjusted to get the bit time back within sync with the actual signal. The SJW is the maximum amount that either PHSEG1 or PHSEG2 are adjusted to allow this resynchronization. This should be set to the maximum possible value (while not overshooting PHSEG1 or PHSEG2) as this increases oscillator tolerance of the CAN node.

3.3 Data Bit Rate (DBR)

The Data Bit Rate (DBR) is the bit rate of the data bytes sent during the CAN FD frame. Since it is the segment of the frame in which it is guaranteed only one node on the bus will be transmitting, it has less stringent requirements, allowing for higher data rates.

Setting up the Data Bit Rate is performed in the same way as the Nominal Bit Rate, using the C1DBTCFGT, C1DBTCFGU, C1DBTCFGH, and C1DBTCFGL registers, instead of the C1NBTCFG registers. All equations and bits are similarly setup. The primary difference is that TSEG1, TSEG2 and SJW for the Data Bit Rate have much lower maximum numbers, as the number of Time Quanta per bit time is naturally going to be lower given the higher bit rate used in the data segment of the CAN frame.

The considerations for setting up BRP, TSEG1, TSEG2 and SJW for the Data Bit Rate are similar to the Nominal Bit Rate considerations with the three primary differences listed below.

1. It is highly recommended to set the Data Bit Rate Prescaler (DBRP) to be the same as the Nominal Bit Rate Prescaler (NBRP), as this prevents errors during the bit rate switch.
2. As previously mentioned, TSEG1, TSEG2, and SJW have lower maximum values, and with the same BRP, there will generally be a lower number of Time Quanta in these segments for the data rate, as it is a faster bit rate.
3. The TSEG1 value no longer needs to take propagation delay into consideration, as the propagation is only a concern during arbitration, and the data rate occurs when only one node is transmitting.

3.4 Bit Rate Setup With the Microchip Code Configurator (MCC)

MCC has an easy setup tool that allows the user to enter the clock frequency, desired Nominal and Data Bit Rates, number of Time Quanta, and sample point percentage, and it will write all relevant registers. An example of MCC code generated that creates a 500 kbit Nominal Bit Rate and a 2 Mbit Data Bit Rate from a 40 MHz source clock is shown below.

Example 3-1. Example MCC Bit Rate Setup Code

```
static void CAN1_BitRateConfiguration(void)
{
    // SJW 14;
    C1NBTCFGL = 0x0E;

    // TSEG2 14;
    C1NBTCFGH = 0x0E;

    // TSEG1 63;
    C1NBTCFGU = 0x3F;

    // BRP 0;
    C1NBTCFGT = 0x00;

    // SJW 4;
    C1DBTCFGL = 0x04;

    // TSEG2 4;
    C1DBTCFGH = 0x04;

    // TSEG1 13;
    C1DBTCFGU = 0x0D;

    // BRP 0;
    C1DBTCFGT = 0x00;

    // TDCO 14;
    C1TDCH = 0x0E;

    // TDCMOD Auto;
    C1TDCU = 0x02;
}
```

4. FIFOs and Memory Usage

The CAN FD module contains implementations of data FIFOs to store received and transmitted messages. All of these FIFOs are contained within a segment of memory intended for the CAN FIFOs, which is located at the end of general purpose RAM.

4.1 Memory Setup and FIFOBA registers

The C1FIFOBA set of registers determines where in memory this FIFO RAM begins. In most cases, it is recommended on 8-bit CAN FD devices to set FIFOBA to the beginning of the CAN FIFO designated area memory, as this gives the most space for CAN FIFOs, but in some cases this segment of memory may be desired to be used for other purposes, in which case, C1FIFOBA can be used to relocate the beginning of the CAN FIFOs further into the memory section.

On PIC18-Q84 devices, the CAN RAM section is limited to 2 KB of RAM, so keep this in mind when determining the number and size of FIFOs to be used in applications.

The FIFO section starts at the location defined by C1FIFOBA, and memory is allocated in the following order:

1. TEF (Transmit Event FIFO)
2. TXQ (Transmit Queue)
3. FIFO 1
4. FIFO 2
5. FIFO 3

Any FIFOs that are not enabled or not used are skipped and do not take up any of the CAN RAM section.

4.2 Transmit Event FIFO

The Transmit Event FIFO (TEF) is a FIFO that records transmitted messages, with a sequence number and optional timestamp. The TEF does not store the payload of the transmitted messages. The most essential Transmit Event FIFO configurations are:

- The TEF is enabled by setting the STEF bit of the C1CONU register.
- Configuring the number of message objects stored in the TEF by changing the FSIZE bits of the C1TEFCONT register.
- The TEFTSEN bit of the C1TEFCONL register enables or disables timestamping of message objects in the TEF.

4.3 Transmit Queue (TXQ)

The Transmit Queue (TXQ) is a transmit-only FIFO. Setting the TXQEN bit of C1CONU enables the TXQ, the FSIZE bits of C1TXQCONT, configure the number of message objects that can be stored in the TXQ. The PLSIZE bits of C1TXQCONT configure the payload size of the message objects (all objects in the same FIFO share a payload size).

4.4 Configurable FIFOs 1-3

The other three FIFOs (1-3) can be configured either as transmit or receive FIFOs. Each can be configured as a transmit FIFO by setting the TXEN bit in the C1FIFOCONxL register, where x is the FIFO number to be controlled. Clearing the TXEN bit will configure the FIFO as a receive FIFO. Like the Transmit Queue, the FSIZE bits of the C1FIFOCONxT register configure the number of message objects that can be stored in the FIFO, and the PLSIZE bits of the C1FIFOCONxT register configure the payload size of the message objects. When the FIFO is in Receive mode, the RXTSEN bit of the C1FIFOCONxL register can be set to enable timestamping of received messages.

4.5 Memory Setup with MCC

MCC Easy Setup will setup FIFO settings, allowing the enabling of TXQ, as well as all three reprogrammable FIFOs, and allows configuration of the following:

- FIFO RX or TX (except for TXQ)
- FIFO Depth (number of message objects)
- Payload (number of data bytes in each message object)
- Custom names (for TX FIFOs)
- TX Priority (for TX FIFOs)

The FIFO setup also summarizes the amount of data being used by the FIFOs, showing whether there is enough space left for the FIFOs in the CAN memory. Below is an example of code generated by MCC to setup FIFO 1 as an RX FIFO, FIFO 2 as a TX FIFO as well as the TXQ, all with a depth of 6 and a payload size of 32 bytes.

Example 4-1. FIFO Setup MCC Code

```
static void CAN1_RX_FIFO_Configuration(void)
{
    // TXEN disabled; RTREN disabled; RXTSEN disabled; TXATIE disabled; RXOVIE
disabled; TFERFFIE disabled; TFHRFHIE disabled; TFNRFNIE disabled;
    C1FIFOCON1L = 0x00;

    // FRESET enabled; TXREQ disabled; UINC disabled;
    C1FIFOCON1H = 0x04;

    // TXAT Unlimited number of retransmission attempts; TXPRI 1;
    C1FIFOCON1U = 0x60;

    // PLSIZE 32; FSIZE 6;
    C1FIFOCON1T = 0xA5;

}
```

```
static void CAN1_TX_FIFO_Configuration(void)
{
    // TXATIE disabled; TXQEIE disabled; TXQNIE disabled;
    C1TXQCONL = 0x00;

    // FRESET enabled; UINC disabled;
    C1TXQCONH = 0x04;

    // TXAT 3; TXPRI 1;
    C1TXQCONU = 0x60;

    // PLSIZE 32; FSIZE 6;
    C1TXQCONT = 0xA5;

    // TXEN enabled; RTREN disabled; RXTSEN disabled; TXATIE disabled; RXOVIE
disabled; TFERFFIE disabled; TFHRFHIE disabled; TFNRFNIE disabled;
    C1FIFOCON2L = 0x80;

    // FRESET enabled; TXREQ disabled; UINC disabled;
    C1FIFOCON2H = 0x04;

    // TXAT Unlimited number of retransmission attempts; TXPRI 1;
    C1FIFOCON2U = 0x60;

    // PLSIZE 32; FSIZE 6;
    C1FIFOCON2T = 0xA5;

}
```

5. Transmission

Transmitting messages using the CAN FD module consists of two steps: writing to the transmit FIFO and requesting a transmit from that FIFO. Transmits can be performed either by the TXQ or any of the reconfigurable FIFOs, if they are configured in Transmit mode.

5.1 Writing to the Transmit FIFO

Before writing to a transmit FIFO, it is important to check that the FIFO is not full. This can be determined by checking the TFNRFNIF bit of either the C1FIFOSTAxL or C1TXQSTAL register. If the flag is set, the FIFO is not full. Each transmit FIFO has a set of User Address registers (C1FIFOxUA or C1TXQUA) that determine the RAM location of the head of the FIFO. Writing to the transmit FIFO is a matter of writing the transmit message object to this address. The transmitted message object is stored according to the following table (with an example payload of n bytes, giving a total transmit object size of m bytes).

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	SID[7:0]							
1	EID[4:0]					SID[10:8]		
2	EID[12:5]							
3	-	-	SID11	EID[17:13]				
4	FDF	BRS	RTR	IDE	DLC[3:0]			
5	SEQ[6:0]							ESI
6	SEQ[14:7]							
7	SEQ[22:15]							
8	Transmit Data Byte 0							
9	Transmit Data byte 1							
10	Transmit Data byte 2							
11	Transmit Data byte 3							
12	Transmit Data byte 4							
13	Transmit Data byte 5							
14	Transmit Data byte 6							
15	Transmit Data byte 7							
…………								
m-3	Transmit Data byte n-3							
m-2	Transmit Data byte n-2							
m-1	Transmit Data byte n-1							
m	Transmit Data byte n							

Once a message has been loaded into the transmit FIFO, setting the UINC bit of the C1FIFOCONxH or C1TXQCONH register will increment the head of the respective FIFO.

5.2 Requesting Transmit

After loading messages into the transmit FIFO, transmitting messages from the FIFO is accomplished by writing the TXREQ bit of either the C1FIFOCONxH or C1TXQCONH registers. Setting this bit will transmit all messages in the respective FIFO and clear the TXREQ bit, once all messages are transmitted. Multiple FIFOs and the TXQ can be requested to transmit at the same time, with the highest priority FIFO/TXQ transmitting first. Highest priority is determined by the TXPRI[4:0] bits of the C1FIFOCONxU or C1TXQCONU registers. Messages in FIFOs configured as transmit will be sent First-in First-Out, while messages in the TXQ will be sent based on their message ID (lower message IDs are sent first).

It is possible to continue loading messages into the FIFO or TXQ while transmitting messages, but when doing so, the TXREQ and UINC bits should be set at the same time to ensure that any appended messages are transmitted

In addition to the TXREQ bits located in the individual FIFO Configuration registers, the C1TXREQL register contains mirror bits of all TXREQ bits that can be written to or read from as if they were the bits in those individual registers. This allows for easy monitoring and control of message transmissions from one location.

5.3 MCC Transmit APIs

MCC, in addition to the setup of the transmit FIFOs, creates CAN FD transmit APIs, as well as data structures for easier transmission of message objects. Th most essential of these are the CAN_MSG_OBJ struct and CAN_MSG_FIELD union.

These two data types contain all of the data contained within a CAN FD message object, with the CAN_MSG_FIELD containing the ID type, frame type, DLC, format type and bit rate switch, and the CAN_MSG_OBJ containing the CAN ID, the CAN_MSG_FIELD, and a pointer to the data bytes.

Each section of the CAN_MSG_FIELD has an enum associated with it for easier use:

- The BRS mode can be set to CAN_NON_BRS_MODE or CAN_BRS_MODE, depending on if the message changes bit rate.
- The ID type can be set to CAN_FRAME_STD or CAN_FRAME EXT, depending on if the message has a standard or extended ID.
- The Frame mode can be set to CAN_FRAME_DATA or CAN_FRAME_RTR (keeping in mind this should only be used with CAN 2.0 messages, as CAN FD does not support Remote Frames).
- The Format mode can be set to CAN_2_0_FORMAT or CAN_FD_FORMAT, depending on if the message is a CAN 2.0 message or CAN FD message (keeping in mind that CAN 2.0 messages will not support BRS and CAN FD messages will not support RTR).

Using these data structures is a matter of declaring a CAN_MSG_OBJ object, then assigning the proper value to each of the fields. In the following example, a standard ID, CAN FD message with Bit Rate Switching Enabled and 32 data bytes is setup.

Example 5-1. Setting up a CAN_MSG_OBJ

```
CAN_MSG_OBJ Periodic_Transmit;
    uint8_t
Transmit_Data[32]={0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,0x0A,0x0B,
                               0x0C,0x0D,0x0E,0x0F,
0x10,0x11,0x12,0x13,0x14,0x15,0x16,0x17,
                               0x18,0x19,0x1A,0x1B,0x1C,0x1D,0x1E,0x1F};
    Periodic_Transmit.msgId=0x001;
    Periodic_Transmit.field.formatType=CAN_FD_FORMAT;
    Periodic_Transmit.field.idType = CAN_FRAME_STD;
    Periodic_Transmit.field.dlc = DLC_32;
    Periodic_Transmit.field.brs = CAN_BRS_MODE;
    Periodic_Transmit.field.frameType = CAN_FRAME_DATA;
    Periodic_Transmit.data = Transmit_Data;
```

⚠ CAUTION Failing to initialize any part of the CAN_MSG_OBJ can cause undesirable results, as the fields may have unassociated leftover data in RAM.

Once a CAN_MSG_OBJ is created and initialized, there are a few functions setup by MCC to aid in transmitting message objects. The `CAN1_TransmitFIFOStatusGet(const CAN1_TX_FIFO_CHANNELS fifoChannel)` returns whether a transmit FIFO is full or not, and the `CAN1Transmit(const CAN1_TX_FIFO_CHANNELS fifoChannel, CAN_MSG_OBJ *txCanMsg)` function takes arguments of a CAN transmit FIFO and a reference to a CAN_MSG_OBJ. Continuing the previous example, the following code checks whether the TXQ is full, and if not, transmits the previously initialized message object.

```
    if(CAN_TX_FIFO_AVAILABLE == (CAN1_TransmitFIFOStatusGet(TXQ) & CAN_TX_FIFO_AVAILABLE))
                {
                    CAN1_Transmit(TXQ, &Periodic_Transmit);
                }
```

The `CAN1Transmit` function will automatically handle writing to and incrementing the FIFO and setting the transmit request.

6. Reception

Reception of messages on the CAN FD module are similar to transmission in terms of dealing with the FIFOs, but has an additional consideration: masks and filters. The CAN protocol is message-based and any application is going to only want to respond to specific CAN IDs. The masks and filters allow the CAN FD module to ignore any messages not on that specific list.

6.1 Mask and Filter Setup

The filters are configured using the C1FLTCONx and C1FLTOBJx registers. Each filter has a set of C1FLTOBJ registers, which contain the 11/12 standard identifier bits, the 18 extended identifier bits and the EXIDE bit, which determines if they will accept only extended ID or standard ID messages. Setting or clearing the EXIDE bit, then setting the SID/EID bits to the ID of the message that the filter should accept, configures that particular filter to accept that message ID.

The C1FLTCONx registers are then used to associate filters with FIFOs. C1FLTCON0L configures filter 0, C1FLTCON0H configures filter 1, C1FLTCON0U configures filter 2, and so on up to C1FLTCON2T configuring filter 11. The FnBP bits determine which FIFO each filter is associated with, then FLTENn enables the filter.

Note: FLTENn must not be set until after the associated C1FLTOBJn and FnBP registers are fully configured, as those registers or bits become locked once the filter is enabled, and any writes to those registers or bits will not be reflected in the actual registers.

While Filters allow for individual message IDs to be accepted, masks allow for a single FIFO to treat certain bits of the ID as “don’t care,” allowing for several messages to be accepted by only using one or two filters. Each filter has an associated mask configured by C1MASKn, which has the same bits as the filter registers (standard and extended ID, as well as the MIDE, that determines if the filter cares about whether a message is standard or extended ID, or whether it will accept both).

6.2 Reading from Receive FIFOs

As previously mentioned, handling data in the receive FIFOs is very similar to doing so in the transmit FIFOs. In most use cases, the fact that a receive FIFO is not empty will be evident from interrupt triggering (as shown in CAN Reception Interrupts). But, if that is not already the case, checking the TFNRFNIF bit to ensure that the FIFO is not empty is required before attempting to read data from the FIFO. After doing so, the C1FIFOxUA registers will give the location in RAM that contains the CAN message object in the following format:

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	SID[7:0]							
1	EID[4:0]					SID[10:8]		
2	EID[12:5]							
3	-	-	SID11	EID[17:13]				
4	FILHIT[4:0]					-	-	ESI
5	FDF	BRS	RTR	IDE	DLC[3:0]			
6	-	-	-	-	-	-	-	-
7	-	-	-	-	-	-	-	-
8	Receive Data Byte 0							
9	Receive Data Byte 1							
10	Receive Data Byte 2							
11	Receive Data Byte 3							

12	Receive Data Byte 4
13	Receive Data Byte 5
14	Receive Data Byte 6
15	Receive Data Byte 7
....	
m-3	Receive Data Byte n-3
m-2	Receive Data Byte n-2
m-1	Receive Data Byte n-1
m	Receive Data Byte n

Once a message object is read from the Receive FIFO, the C1FIFOxUA bit needs to be set, which will increment the C1FIFOxUA registers to the next message location, allowing for the next message object from the FIFO to be the next received message.

6.3 CAN Reception Interrupts

A common use case for CAN reception is interrupt driven, and the CAN FD module has a dedicated interrupt for such a use case. The top level CANRX Interrupt Flag (CANRXIF) is a read-only interrupt flag that can be triggered by three different reception events:

1. Receive FIFO is not empty
2. Receive FIFO is half full
3. Receive FIFO is full

In addition, each receive FIFO can have these interrupts individually enabled and can trigger the top level interrupt.

In short, to setup the receiver interrupt, the following procedure must be followed:

1. Choose and enable whichever FIFO interrupt triggers is desired (not empty, half full, full) for any receive FIFOs that should generate interrupts. These are controlled by the C1FIFOCONxL TFERFFIE (full), TFHRFHIE (half full), and TFNRFNIE (not empty) bits of the FIFO number in question.
2. Enable the Receive Object Interrupt Enable (RXIE) bit in the C1INTU register.
3. Enable the CAN Receive Interrupt Enable bit in the PIE registers, the location may vary by device.

From here, the enabled receive FIFO conditions will generate an interrupt, and user code must check which FIFO/ conditions triggered the interrupt. This can be aided by the C1RXIFL register, which has individual bits for each FIFO indicating that the FIFO in question has an interrupt pending, and the individual TFERFFIF, TFHRFHIF, and TFNRFNIF bits of the C1FIFOSTAxL register for the FIFO in question.

6.4 MCC Reception Setup/APIs

MCC Easy Setup allows for configuration of masks and filters, as well as through filter object settings. Each filter can be individually enabled and configured, by selecting which receive FIFO the filter is associated with and which message IDs the filter should accept. The filter and mask configuration code will then be automatically generated and added to the project's code.

Reception uses the same data structures previously mentioned in MCC Transmit APIs, most importantly the CAN_MSG_OBJ data type. In this case, it is a matter of declaring a message object, then either through polling or interrupt-driven operation, to use the `CAN1_Receive(CAN_MSG_OBJ *rxCanMsg)` function. This function will place all of the data from the received message into the declared message object and handle the FIFO incrementing. Then, all of the required data can be processed from the declared CAN_MSG_OBJ. For example, the following code receives a message and outputs the 1st data byte onto a series of LEDs connected to port B and D.

Example 6-1. Receiving a CAN FD Message with MCC

```
CAN_MSG_OBJ LED_Message;
          uint8_t LED_Holder;
          //polling or interrupt code here
          CAN1_Receive(&LED_Message)
          LED_Holder=LED_Message.data[0]
          LATDbits.LATD0 = LED_Holder>>7;
          LATDbits.LATD1 = (LED_Holder-128)>>6;
          LATDbits.LATD2 = (LED_Holder-192)>>5;
          LATDbits.LATD3 = (LED_Holder-224)>>4;
          LATBbits.LATB0 = (LED_Holder-240)>>3;
          LATBbits.LATB1 = (LED_Holder-248)>>2;
          LATBbits.LATB2 = (LED_Holder-252)>>1;
          LATBbits.LATB3 = LED_Holder-254;
```

7. Conclusion

The CAN FD module represents a new kind of CAN interface for PIC18F microcontrollers, both in capability and in how the module is used. This document outlined the major areas that need to be configured before transmitting or receiving data using the CAN FD module. It highlighted bit rate configuration, FIFO and memory usage, as well as transmission and reception message object setup and message exchange. In addition, it outlined how the MPLAB Code Configurator (MCC) can be used to make each of these configuration steps easier to create code for.

The Microchip Website

Microchip provides online support via our website at www.microchip.com/. This website is used to make files and information easily available to customers. Some of the content available includes:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip design partner program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Product Change Notification Service

Microchip's product change notification service helps keep customers current on Microchip products. Subscribers will receive email notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, go to www.microchip.com/pcn and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Embedded Solutions Engineer (ESE)
- Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website at: www.microchip.com/support

Microchip Devices Code Protection Feature

Legal Notice

Trademarks

ISBN: 978-1-5224-6227-9

Quality Management System

For information regarding Microchip's Quality Management Systems, please visit www.microchip.com/quality.

Worldwide Sales and Service

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
www.microchip.com/support
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Austin, TX
Tel: 512-257-3370

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Novi, MI
Tel: 248-848-4000

Houston, TX
Tel: 281-894-5983

Indianapolis
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453
Tel: 317-536-2380

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608
Tel: 951-273-7800

Raleigh, NC
Tel: 919-844-7510

New York, NY
Tel: 631-435-6000

San Jose, CA
Tel: 408-735-9110
Tel: 408-436-4270

Canada - Toronto
Tel: 905-695-1980
Fax: 905-695-2078

ASIA/PACIFIC

Australia - Sydney
Tel: 61-2-9868-6733

China - Beijing
Tel: 86-10-8569-7000

China - Chengdu
Tel: 86-28-8665-5511

China - Chongqing
Tel: 86-23-8980-9588

China - Dongguan
Tel: 86-769-8702-9880

China - Guangzhou
Tel: 86-20-8755-8029

China - Hangzhou
Tel: 86-571-8792-8115

China - Hong Kong SAR
Tel: 852-2943-5100

China - Nanjing
Tel: 86-25-8473-2460

China - Qingdao
Tel: 86-532-8502-7355

China - Shanghai
Tel: 86-21-3326-8000

China - Shenyang
Tel: 86-24-2334-2829

China - Shenzhen
Tel: 86-755-8864-2200

China - Suzhou
Tel: 86-186-6233-1526

China - Wuhan
Tel: 86-27-5980-5300

China - Xian
Tel: 86-29-8833-7252

China - Xiamen
Tel: 86-592-2388138

China - Zhuhai
Tel: 86-756-3210040

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-3090-4444

India - New Delhi
Tel: 91-11-4160-8631

India - Pune
Tel: 91-20-4121-0141

Japan - Osaka
Tel: 81-6-6152-7160

Japan - Tokyo
Tel: 81-3-6880- 3770

Korea - Daegu
Tel: 82-53-744-4301

Korea - Seoul
Tel: 82-2-554-7200

Malaysia - Kuala Lumpur
Tel: 60-3-7651-7906

Malaysia - Penang
Tel: 60-4-227-8870

Philippines - Manila
Tel: 63-2-634-9065

Singapore
Tel: 65-6334-8870

Taiwan - Hsin Chu
Tel: 886-3-577-8366

Taiwan - Kaohsiung
Tel: 886-7-213-7830

Taiwan - Taipei
Tel: 886-2-2508-8600

Thailand - Bangkok
Tel: 66-2-694-1351

Vietnam - Ho Chi Minh
Tel: 84-28-5448-2100

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4485-5910
Fax: 45-4485-2829

Finland - Espoo
Tel: 358-9-4520-820

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Garching
Tel: 49-8931-9700

Germany - Haan
Tel: 49-2129-3766400

Germany - Heilbronn
Tel: 49-7131-72400

Germany - Karlsruhe
Tel: 49-721-625370

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Germany - Rosenheim
Tel: 49-8031-354-560

Israel - Ra'anana
Tel: 972-9-744-7705

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Italy - Padova
Tel: 39-049-7625286

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Norway - Trondheim
Tel: 47-72884388

Poland - Warsaw
Tel: 48-22-3325737

Romania - Bucharest
Tel: 40-21-407-87-50

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

Sweden - Gothenberg
Tel: 46-31-704-60-40

Sweden - Stockholm
Tel: 46-8-5090-4654

UK - Wokingham
Tel: 44-118-921-5800
Fax: 44-118-921-5820