
PIC18C ECAN 'C' Routines

*Authors: Caio Gübel and Nilesh Rajbharti
Microchip Technology, Inc.*

INTRODUCTION

The Enhanced Controller Area Network (ECAN) module, offered by many of the PIC18F family of PICmicro® microcontrollers, is the latest enhancement to the existing legacy CAN module. Devices such as the PIC18C658/858 and PIC18F248/258/448/458 use the legacy CAN module.

ECAN offers many enhancements over the legacy CAN module in terms of more transmit/receive buffers, acceptance filters, and hardware FIFO operation. At the same time, the ECAN module is fully backward compatible with the legacy CAN module.

ECAN provides three modes of operation – Mode 0, Mode 1 and Mode 2. Mode 0 is fully backward compatible with the legacy CAN module. Applications developed for the legacy CAN module would continue to work without any change using ECAN. Mode 1 is the Enhanced Legacy mode with increased buffers and filters. Mode 2 has the same resources as Mode 1, but with a hardware managed receive FIFO. Given its features and flexibility, ECAN would prove useful to many CAN-based applications.

This application note implements 'C' routines to access all features of the ECAN module. This document does not describe ECAN and the related PIC18F family in detail. Readers are encouraged to read the PIC18F6585/6680/8585/8680 device data sheet (DS30491) for more information.

OVERVIEW OF ECAN MODULE

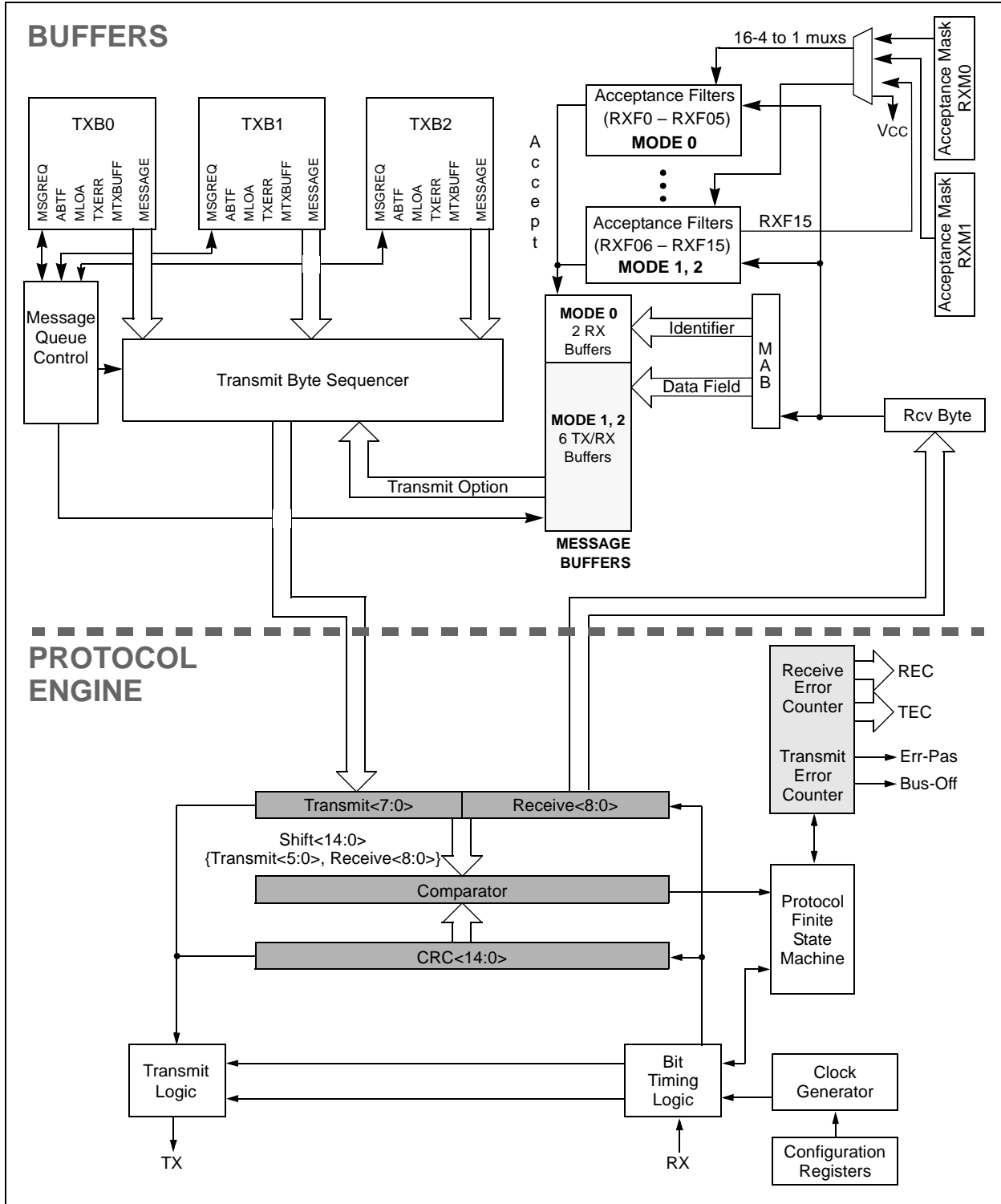
Following are the main features of the ECAN module:

- Fully backward compatible with the legacy CAN module
- Three functional modes:
 - Mode 0 – Fully backward compatible Legacy mode
 - Mode 1 – Enhanced Legacy mode
 - Mode 2 – Hardware FIFO mode
- Implementation of these CAN protocols:
 - CAN 1.2, CAN 2.0A and CAN 2.0B
- Standard and extended data frames
- Data length of 0-8 bytes
- Programmable bit rate up to 1 Mbps
- Support for automatic Remote Transmission Request frame handling
- Dedicated double-buffered receiver with two prioritized storage buffers
- Three dedicated transmit buffers with application specified prioritization and abort capability
- Six full (Standard/Extended Identifier) programmable receive/transmit buffers
- Sixteen full acceptance filters with dynamic association to receive buffers
- Three full acceptance filter masks with dynamic association to receive filters
- Programmable wake-up functionality with integrated low-pass filter
- Programmable Loopback mode and programmable state clocking supports self-test operation
- Signaling via interrupt capabilities for all CAN receiver and transmitter error codes
- Programmable clock source
- Programmable link-to-timer module for time-stamping and network synchronization
- Low-power Sleep/Disable mode
- DeviceNet™ data byte filter support
- Advanced error management support features

AN878

Figure 1 shows a block diagram of the ECAN module buffers and protocol engine.

FIGURE 1: ECAN BUFFERS AND PROTOCOL ENGINE BLOCK DIAGRAM



OVERVIEW OF ECAN ROUTINES

- Out-of-the-box support for Microchip C18 and Hi-Tech PICC 18™ C compilers
- Offers simple abstract interface to ECAN module for most applications
- Additional functions/macros are available for advanced applications
- Supports all three functional modes
- Provides access to all ECAN features in Polling mode
 - Easily modifiable to Interrupt Driven mode
- Operates in two main modes:
 - Run-Time Library mode and Fixed Library mode
- Various compile time options to customize routines to a specific application
 - Also available as the Microchip Application Maestro™ module to simplify customization

ECAN FUNCTIONS ORGANIZATION AND USAGE

These functions are developed for the Microchip C18 and Hi-Tech PICC 18 C compilers. ECAN routine source files automatically detect the compiler in use and re-define corresponding symbols. If required, users can easily port this file to any C compiler for PICmicro devices.

Source code for the ECAN routines is divided into the following three files:

- ECAN.c
- ECAN.h
- ECAN.def

Even though ECAN functions provide a high level interface to the ECAN module, users must be familiar with the features and capabilities of the ECAN module. This knowledge is absolutely required when setting up compile time options in the ECAN.def file.

To employ these ECAN routines in your application, perform the following steps:

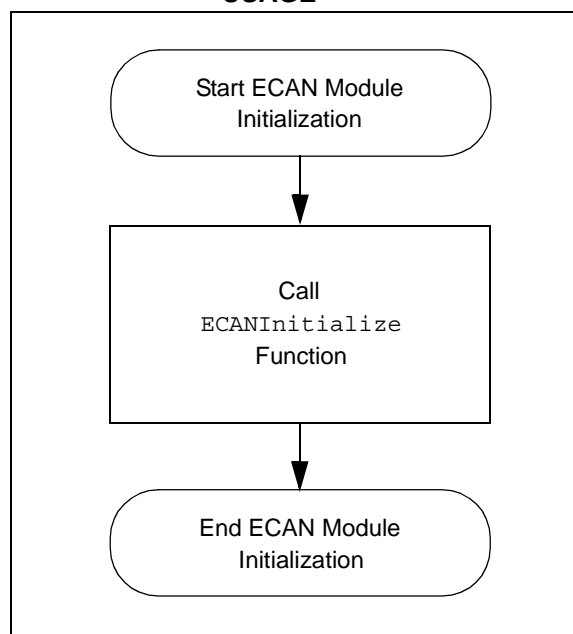
1. Copy ECAN.c, ECAN.h and ECAN.def into your application project directory.
2. Modify ECAN.def (compile time options) as per your application requirements. You may manually modify this file or use the Microchip Application Maestro software. See **Appendix A: “ECAN Routines Compile Time Options”** for more information.
3. Include ECAN.c in your application project.
4. Add #include “ECAN.h” line in each source file that will be calling ECAN routines. Make calls to ECAN routines as required. See **Appendix B: “ECAN Functions”** for the list of available routines.

You may also create an object or library file for ECAN.c and use the output file in your project, rather than using the actual source code files.

The ECAN module offers various modes of operation and configuration. Most of the applications will set these configurations only once upon start. Given that many of these configurations may not need to be changed at run-time, ECAN routines provide a set of compile-time options to configure the module at design time. When compiled with the appropriate compile time options set, ECAN routines will generate “customized” code for your application. This approach results in significantly less code than that generated entirely by run-time routines.

It is anticipated that there may be cases where applications may need to change modes and configurations at run-time depending on specific application conditions. ECAN routines provide special compile time options that allow applications to change all ECAN features at run-time. When this mode is enabled, the resulting code will be larger. The ECAN.def file contains all available compile time options and their values.

FIGURE 2: TYPICAL ECAN ROUTINES USAGE



AN878

SAMPLE APPLICATION PROGRAM USING ECAN ROUTINES

This application note includes the complete source code for ECAN routines. In addition, it also includes the sample application (ECANDemo.c) that demonstrates a simple “echo” application using ECAN routines. There are also two MPLAB® projects:

1. ECANDemo for Microchip C18 compiler.
2. ECANDemoHt for Hi-Tech PICC 18 compiler.

EXAMPLE 1: SAMPLE APPLICATION – TYPICAL

```
#include "ecan.h"

void main(void)
{
    unsigned long id;
    BYTE data[4];
    BYTE dataLen;
    ECAN_RX_MSG_FLAGS flags;

    // ECAN.def file must be set up correctly.
    ECANInitialize();

    while( !ECANSendMessage(0x123, data, 0, ECAN_TX_STD_FRAME) );

    do
    {
        // Wait for a message to get received.
        while( !ECANReceiveMessage(&id, data, &dataLen, &flags) );

        // Increment received id and echo it back.
        id++;
        while( !ECANSendMessage(id, data, dataLen, flags) );
    } while(1);
}
```

EXAMPLE 2: SAMPLE APPLICATION – ADVANCED

```

#include "ecan.h"

void RunTimeInitialization(void);

void main(void)
{
    unsigned long id;
    BYTE data[4];
    BYTE dataLen;
    ECAN_RX_MSG_FLAGS flags;

    // Initialize ECAN as per compile-time options.
    ECANInitialize();

    // Receive and send messages. Upon receiving certain message
    // change ECAN settings at run-time.
    ...

    RunTimeInitialization();

    // Continue with CAN communication
    ...
}

void RunTimeInitialization(void)
{
    // Must be in Config mode to change many of settings.
    ECANSetOperationMode(ECAN_OP_MODE_CONFIG);

    // Select Mode 1
    ECANSetFunctionalMode(ECAN_MODE_1);

    // Make B0 as receiver.
    ECANSetBnTxRxMode(B0, ECAN_BUFFER_RX);

    // RXB0 will receive Standard messages only.
    ECANSetRxBnRxMode(RXB0, ECAN_RECEIVE_ALL_VALID);

    // B0 will receive Extended
    ECANSetBnRxMode(B0, ECAN_RECEIVE_ALL_VALID);

    // Link RXF0 to RXB0 and RXF1 to B0
    ECANLinkRXF0F1ToBuffer(RXB0, B0);
    ECANLinkRXF0Thru3ToMask(ECAN_RXM0,
                            ECAN_RXM1,
                            ECAN_RXM0,
                            ECAN_RXM0);

    ECANSetRXF0Value(0, ECAN_MSG_STD);
    ECANSetRXF1Value(0, ECAN_MSG_XTD);

    ECANSetRXM0Value(0, ECAN_MSG_STD);
    ECANSetRXM1Value(0, ECAN_MSG_XTD);

    // Set 125kbps @ 25MHz
    ECANSetBaudRate(2, 4, 8, 8, 8);

    // Return to Normal mode to communicate.
    ECANSetOperationMode(ECAN_OP_MODE_NORMAL);
}

```

APPENDIX A: ECAN ROUTINES COMPILE TIME OPTIONS

TABLE A-1: ECAN.def COMPILE TIME OPTIONS

Option	Possible Values	Purpose	Note
ECAN_Bn_AUTORTR_MODE (0 <= n <= 5)	ECAN_AUTORTR_MODE_DISABLE ECAN_AUTORTR_MODE_ENABLE	Sets Buffer Bn in Automatic RTR Handling mode. Buffer Bn must be set up in Transmit mode using ECAN_Bn_TXRX_MODE_VAL. ECAN_AUTORTR_MODE_DISABLE disables auto-RTR. ECAN_AUTORTR_MODE_ENABLE enables auto-RTR.	Mode 1, Mode 2
ECAN_Bn_MODE_VAL (0 <= n <= 5)	ECAN_RECEIVE_ALL_VALID ECAN_RECEIVE_STANDARD ECAN_RECEIVE_EXTENDED ECAN_RECEIVE_ALL	Configures Bn Buffer Receive mode. This is used only if Bn buffer is configured as receive buffer using above option. Meaning of values is same as ECAN_RXB0_MODE_VAL.	Mode 1, Mode 2
ECAN_Bn_TXRX_MODE_VAL (0 <= n <= 5)	ECAN_BUFFER_TX ECAN_BUFFER_RX	Configures buffer Bn as transmit or receive. There are a total of five options – one for each programmable buffer. ECAN_BUFFER_TX selects transmit. ECAN_BUFFER_RX selects receive.	Mode 1, Mode 2
ECAN_BRP_VAL	1 through 8 inclusive	Sets baud rate prescale value.	
ECAN_BUS_SAMPLE_MODE_VAL	ECAN_BUS_SAMPLE_MODE_THRICE ECAN_BUS_SAMPLE_MODE_ONCE	Sets CAN bus Sample mode. ECAN_BUS_SAMPLE_MODE_THRICE selects three Sample modes. ECAN_BUS_SAMPLE_MODE_ONCE selects one Sample mode.	
ECAN_CAPTURE_MODE_VAL	ECAN_CAPTURE_MODE_DISABLE ECAN_CAPTURE_MODE_ENABLE	Sets CAN Capture mode. CCP1 must be configured separately. ECAN_CAPTURE_MODE_DISABLE disables CAN Capture mode. ECAN_CAPTURE_MODE_ENABLE enables CAN Capture mode.	
ECAN_FILTER_MODE_VAL	ECAN_FILTER_MODE_ENABLE ECAN_FILTER_MODE_DISABLE	Select low-pass filter for CAN bus activity wake-up. ECAN_FILTER_MODE_ENABLE selects wake-up filter. ECAN_FILTER_MODE_DISABLE deselects wake-up filter.	
ECAN_FUNC_MODE_VAL	ECAN_MODE_0 ECAN_MODE_1 ECAN_MODE_2	Sets initial functional mode for ECAN module. ECAN_MODE_0 selects Mode 0. ECAN_MODE_1 selects Mode 1. ECAN_MODE_2 selects Mode 2.	
ECAN_INIT_MODE	ECAN_INIT_NORMAL ECAN_INIT_CONFIGURATION ECAN_INIT_LOOPBACK ECAN_INIT_DISABLE ECAN_INIT_LISTEN_ONLY	Sets initial operational mode for ECAN module. ECAN_INIT_NORMAL selects "normal" mode. ECAN_INIT_CONFIGURATION selects "Configuration" mode. ECAN_INIT_LOOPBACK selects "Loopback" mode. ECAN_INIT_DISABLE selects "Disable" or "Sleep" mode. ECAN_INIT_LISTEN_ONLY selects "Listen Only" mode.	
ECAN_LIB_MODE_VAL	ECAN_LIB_MODE_FIXED ECAN_LIB_MODE_RUNTIME	Sets ECAN Routine Library mode. ECAN_LIB_MODE_FIXED disables run-time selection code. ECAN_LIB_MODE_RUNTIME enables run-time selection code.	
ECAN_PHSEG1_VAL	1-8 inclusive	Sets Phase Segment 1 value.	
ECAN_PHSEG2_MODE_VAL	ECAN_PHSEG2_MODE_PROGRAMMABLE ECAN_PHSEG2_MODE_AUTOMATIC	Sets Phase Segment 2 Programming mode. ECAN_PHSEG2_MODE_PROGRAMMABLE sets Freely Programmable mode. ECAN_PHSEG2_MODE_AUTOMATIC allows ECAN module to set it.	
ECAN_PHSEG2_VAL	1-8 inclusive	Sets Phase Segment 2 value.	
ECAN_PROPSEG_VAL	1-8 inclusive	Sets Propagation Segment value.	
ECAN_RXB0_DBL_BUFFER_MODE_VAL	ECAN_DBL_BUFFER_MODE_DISABLE ECAN_DBL_BUFFER_MODE_ENABLE	Configures RXB0 in Hardware Double-Buffer mode. This is used in Mode 0 only.	Mode 0
ECAN_RXB0_MODE_VAL	ECAN_RECEIVE_ALL_VALID ECAN_RECEIVE_STANDARD ECAN_RECEIVE_EXTENDED ECAN_RECEIVE_ALL	Sets RXB0 Buffer Receive mode. ECAN_RECEIVE_ALL_VALID causes RXB0 to receive all valid messages. ECAN_RECEIVE_STANDARD causes RXB0 to receive Standard messages. ECAN_RECEIVE_EXTENDED causes RXB0 to receive Extended messages. ECAN_RECEIVE_ALL causes RXB0 to receive all messages.	

TABLE A-1: ECAN.def COMPILE TIME OPTIONS (CONTINUED)

Option	Possible Values	Purpose	Note
ECAN_RXB1_MODE_VAL	ECAN_RECEIVE_ALL_VALID ECAN_RECEIVE_STANDARD ECAN_RECEIVE_EXTENDED ECAN_RECEIVE_ALL	Configures RXB1 Buffer Receive mode. Meaning of values is same as ECAN_RXB0_MODE_VAL.	
ECAN_RXF0_MASK_VAL	ECAN_RXM0 ECAN_RXM1 ECAN_RXMF15	Links a filter to a specific mask register. If selected, mask is ECAN_RXF15, RXF15 must not be linked to any buffer. ECAN_RXM0 links to RXM0 mask. ECAN_RXM1 links to RXM1 mask. ECAN_RXMF15 links to RXF15 filter as mask.	Mode 1, Mode 2
ECAN_RXFn_BUFFER_VAL	RXB0 RXB1 B0 B1 B2 B3 B4 B5	Links a filter to a specific receive buffer. If selected, buffer is a programmable buffer, it must be configured as a receive buffer using ECAN_Bn_TXRX_MODE. RXBn links to one of the dedicated receive buffers. Bn links to one of five programmable receive buffers.	Mode 1, Mode 2
ECAN_RXFn_MODE_VAL (0 <= n <= 15)	ECAN_RXFn_ENABLE ECAN_RXFn_DISABLE	Enables/disables RXFn receive filters. In Mode 0, only RXF0-RXF5 are available and they are always enabled. ECAN_RXFn_ENABLE enables corresponding RXFn filter. ECAN_RXFn_DISABLE disables corresponding RXFn filter.	Mode 1, Mode 2
ECAN_RXFn_MSG_TYPE_VAL (0 <= n <= 15)	ECAN_MSG_STD ECAN_MSG_XTD	Sets Standard/Extended filter type. In Mode 0, only RXF0-RXF5 are available. ECAN_MSG_STD defines Standard filter type. ECAN_MSG_XTD defines Extended filter type.	
ECAN_RXFn_VAL	11-bit or 29-bit value as per filter type	Assigns 11 or 29-bit value to a filter. In Mode 0, only RXF0-RXF5 are available.	
ECAN_RXMn_MSG_TYPE (0 <= n <= 1)	ECAN_MSG_STD ECAN_MSG_XTD	Sets Standard/Extended mask type. ECAN_MSG_STD defines Standard mask type. ECAN_MSG_XTD defines Extended mask type.	
ECAN_SJW_VAL	1-4 inclusive	Sets synchronization jump width value.	
ECAN_TX2_MODE_VAL	ECAN_TX2_MODE_DISABLE ECAN_TX2_MODE_ENABLE	Enables/disables CANTX2 pin. ECAN_TX2_MODE_DISABLE disables CANTX2 pin. ECAN_TX2_MODE_ENABLE enables CANTX2 pin.	
ECAN_TX2_SOURCE_VAL	ECAN_TX2_SOURCE_COMP ECAN_TX2_SOURCE_CLOCK	Sets CANTX2 signal source. ECAN_TX2_SOURCE_COMP sets complement of CANTX1 as source. ECAN_TX2_SOURCE_CLOCK sets CAN block as source.	
ECAN_TXDRIVE_MODE_VAL	ECAN_TXDRIVE_MODE_TRISTATE ECAN_TXDRIVE_MODE_VDD	Defines how CANTX1 pin will be driven in Recessive mode. ECAN_TXDRIVE_MODE_TRISTATE sets CANTX pin in Tri-State mode. ECAN_TXDRIVE_MODE_VDD sets CANTX pin in VDD mode.	
ECAN_WAKEUP_MODE_VAL	ECAN_WAKEUP_MODE_ENABLE ECAN_WAKEUP_MODE_DISABLE	Sets CAN bus Activity mode. ECAN_WAKEUP_MODE_ENABLE enables Wake-up mode. ECAN_WAKEUP_MODE_DISABLE disables Wake-up mode.	

APPENDIX B: ECAN FUNCTIONS

ECANAbortAll

This macro aborts all pending messages from the ECAN module. See the PIC18F6585/6680/8585/8680 data sheet regarding message abortion.

Syntax

```
void ECANAbortAll(void)
```

Parameters

None

Return Values

None

Pre-condition

None

Side Effects

None

Remarks

Use `ECANIsAllAborted` to verify whether any messages are still pending or in progress.

Example: Usage of ECANAbortAll

```
...
```

```
ECANAbortAll();
```

```
...
```


ECANDisableCANTX2

This macro sets the CANTX2 pin as a digital I/O.

Syntax

```
void ECANDisableCANTX2(void)
```

Parameters

None

Return Values

None

Pre-condition

None

Side Effects

None

Remarks

None

Example: Usage of ECANDisableCANTX2

```
// Configure CANTX2 pin as digital I/O pin.  
ECANDisableCANTX2();  
...
```

AN878

ECANGetFilterHitInfo

This macro returns the number of the filter that caused the last message reception.

Syntax

```
BYTE ECANGetFilterHitInfo(void)
```

Parameters

None

Return Values

0 through 15: 0 means RXF0, 1 means RXF1 and so on.

Pre-condition

ECANReceiveMessage() was called and returned TRUE.

Side Effects

None

Remarks

This macro returns filter hit information that was stored by previous call to ECANReceiveMessage() function.

Example: Usage of ECANGetFilterHitInfo

```
// Use ECAN.def options to initialize ECAN module.
ECANInitialize();

...

while(1)
{
    // Send a message.
    if ( !ECANReceiveMessage(&myID,
                             &myData,
                             &dataLen,
                             &rxFlags) )
    {
        // There was no message waiting. Try again...
        ...
    }

    // A message is received.

    // Find out which filter caused this acceptance.
    filterNumber = ECANGetFilterHitInfo();
    ...
}
```

ECANGetFunctionalMode

This macro returns the current ECAN functional mode.

Syntax

```
BYTE ECANGetFunctionalMode(void)
```

Parameters

None

Return Values

ECAN functional mode. Returned value will be one of following:

Value	Meaning
ECAN_MODE_0	Mode 0
ECAN_MODE_1	Mode 1
ECAN_MODE_2	Mode 2

Pre-condition

None

Side Effects

None

Remarks

None

Example: Usage of ECANGetFunctionalMode

```
// Set configuration mode.
ECANSetOperationMode(ECAN_OP_MODE_CONFIG);

// Now change functional mode
ECANSetFunctionalMode(ECAN_MODE_2);

...

// Check current functional mode
currentMode = ECANGetFunctionalMode();
if ( currentMode == ECAN_MODE_0 )
    // in Mode 0

...

else if ( currentMode == ECAN_MODE_1 )
    // in Mode 1

...

else
    // in Mode 2
    ...
...

```

AN878

ECANGetOperationMode

This macro gets the current ECAN module operation mode.

Syntax

```
ECAN_OP_MODE ECANGetOperationMode(void)
```

Parameters

None

Return Values

Value	Meaning
ECAN_OP_MODE_NORMAL	Specifies normal mode of operation
ECAN_OP_MODE_SLEEP	Specifies Sleep mode of operation
ECAN_OP_MODE_LOOP	Specifies Loopback mode of operation
ECAN_OP_MODE_LISTEN	Specifies Listen Only mode of operation
ECAN_OP_MODE_CONFIG	Specifies Configuration mode of operation

Pre-condition

None

Side Effects

None

Remarks

None

Example: Usage of ECANGetOperationMode

```
// Set configuration mode.
ECANSetOperationModeNowait(ECAN_OP_MODE_CONFIG);

// Wait for mode to get accepted
while( ECANGetOperationMode() != ECAN_OP_MODE_CONFIG )
{
    // Do something while mode is being accepted.
}
...
```

ECANGetRxErrorCount

This macro returns the ECAN receive error count as defined by the BOSCH CAN Specification. See the PIC18F6585/6680/8585/8680 data sheet for more information.

Syntax

```
BYTE ECANGetRxErrorCount(void)
```

Parameters

None

Return Values

Current value of receive error count.

Pre-condition

None

Side Effects

None

Remarks

None

Example: Usage of ECANGetRxErrorCount

```
BYTE rxErrorCount;  
  
rxErrorCount = ECANGetRxErrorCount();  
...
```

AN878

ECANGetTxErrorCount

This macro returns the ECAN transmit error count as defined by the BOSCH CAN Specification. See the PIC18F6585/6680/8585/8680 data sheet for more information.

Syntax

```
BYTE ECANGetTxErrorCount(void)
```

Parameters

None

Return Values

Current value of transmit error count.

Pre-condition

None

Side Effects

None

Remarks

None

Example: Usage of ECANGetTxErrorCount

```
BYTE txErrorCount;  
  
txErrorCount = ECANGetTxErrorCount();  
...
```

ECANInitialize

This function initializes the ECAN module with the compile time options as defined in the `ECAN.def` file.

Syntax

```
void ECANInitialize()
```

Parameters

None

Return Values

None

Pre-condition

None

Side Effects

All pending CAN message transmissions are aborted.

Remarks

This function puts the ECAN module into Configuration mode and sets all bit rate, transmit/receive buffer, acceptance filter and mask registers. Values and type of initialization are determined by compile time options defined in the `ECAN.def` file. Upon completion, the ECAN module is set to compile time defined operational mode. If the application is ready to communicate on the CAN bus, the compile time option of normal mode should be selected.

Example: Usage of ECANInitialize

```
// Use ECAN.def options to initialize ECAN module.
ECANInitialize();

// ECAN module will be in operation mode as per ECAN_INIT_MODE
```

AN878

ECANIsAllAborted

This macro returns the status of the previous `ECANAbortAll` request.

Syntax

```
BOOL ECANIsAllAborted(void)
```

Parameters

None

Return Values

TRUE: If there is no pending transmission.

FALSE: If abort is still in progress.

Pre-condition

None

Side Effects

None

Remarks

None

Example: Usage of ECANIsAllAborted

```
...  
  
ECANAbortAll();  
  
while( !ECANIsAllAborted() )  
{  
    // Do something while abort is in progress...  
}  
...
```


ECANIsBusOff

This macro returns the current ECAN bus on/off state.

Syntax

```
BOOL ECANIsBusOff(void)
```

Parameters

None

Return Values

TRUE: If the ECAN module is in the bus off state.

FALSE: If the ECAN module is not in bus off state.

Pre-condition

None

Side Effects

None

Remarks

None

Example: Usage of ECANIsBusOff

```
...  
  
if ( ECANIsBusOff() )  
    // ECAN is Bus off  
...  

```

AN878

ECANIsRxPassive

This macro returns the current ECAN receive error status as defined by the BOSCH CAN Specification. See the PIC18F6585/6680/8585/8680 data sheet for more information.

Syntax

```
BOOL ECANIsRxPassive(void)
```

Parameters

None

Return Values

TRUE: If the ECAN module is in receive error passive state.

FALSE: If the ECAN module is not in receive error passive state.

Pre-condition

None

Side Effects

None

Remarks

None

Example: Usage of ECANIsRxPassive

```
...  
  
if ( ECANIsRxPassive() )  
    // ECAN is in receive passive state  
...  

```

ECANIsTxPassive

This macro returns the current ECAN transmit error status as defined by the BOSCH CAN Specification. See the PIC18F6585/6680/8585/8680 data sheet for more information.

Syntax

```
BOOL ECANIsTxPassive(void)
```

Parameters

None

Return Values

TRUE: If the ECAN module is in transmit error passive state.

FALSE: If the ECAN module is not in transmit error passive state.

Pre-condition

None

Side Effects

None

Remarks

None

Example: Usage of ECANIsTxPassive

```
...  
  
if ( ECANIsTxPassive() )  
    // ECAN is in transmit passive state  
...  

```

AN878

ECANLinkRXFnFmToBuffer

This macro links filters to buffers. There are a total of eight macros. Each macro links two filters at a time. ECANLinkRXF0F1ToBuffer links RXF0 and RXF1 filters to buffers. These macros are available in Mode 1 and Mode 2 only.

Syntax

```
void ECANLinkRXF0F1ToBuffer(RXF0Buffer, RXF1Buffer)
void ECANLinkRXF2F3ToBuffer(RXF2Buffer, RXF3Buffer)
void ECANLinkRXF4F5ToBuffer(RXF4Buffer, RXF5Buffer)
void ECANLinkRXF6F7ToBuffer(RXF6Buffer, RXF7Buffer)
void ECANLinkRXF8F9ToBuffer(RXF8Buffer, RXF9Buffer)
void ECANLinkRXF10F11ToBuffer(RXF10Buffer, RXF11Buffer)
void ECANLinkRXF12F13ToBuffer(RXF12Buffer, RXF13Buffer)
void ECANLinkRXF14F15ToBuffer(RXF14Buffer, RXF15Buffer)
```

Parameters

RXFnBuffer

[in] Name of buffer that is to be linked to RXFn filter. The only permitted values are:

Value	Meaning
RXB0	Link to RXB0 buffer
RXB1	Link to RXB1 buffer
B0	Link to B0 buffer
B1	Link to B1 buffer
B2	Link to B2 buffer
B3	Link to B3 buffer
B4	Link to B4 buffer
B5	Link to B5 buffer

RXFmBuffer

[in] Name of buffer that is to be linked to RXFm filter. The only permitted values are:

Value	Meaning
RXB0	Link to RXB0 buffer
RXB1	Link to RXB1 buffer
B0	Link to B0 buffer
B1	Link to B1 buffer
B2	Link to B2 buffer
B3	Link to B3 buffer
B4	Link to B4 buffer
B5	Link to B5 buffer

Return Values

None

Pre-condition

```
(ECAN_LIB_MODE_VAL == ECAN_LIB_MODE_RUN_TIME)
```

or

```
(ECAN_FUNC_MODE_VAL != ECAN_MODE_0)
```

Side Effects

None

ECANLinkRXFnFmToBuffer (Continued)

Remarks

Buffer value must be a constant of permitted values only. A variable parameter would cause a compile time error. For example, `ECANLinkRXF0F1ToBuffer(myRXF0Buffer, myRXF1Buffer)` would not compile.

Example: Usage of ECANLinkRXFnFmToBuffer

```
// Link RXF0 to RXB0 and RXF1 to B3
ECANLinkRXF0F1ToBuffer(RXB0, B3);

// Link RXF6 and RXF7 to B2
ECANLinkRXF6F7ToBuffer(B2, B2);
...
```

AN878

ECANLinkRXFnThrumToMask

This macro links filters to masks. There are a total of four macros. Each macro links four filters at a time. ECANLinkRXF0Thru3ToMask links RXF0, RXF1, RXF2 and RXF3 filters to masks. These macros are available in Mode 1 and Mode 2 only.

Syntax

```
void ECANLinkRXF0Thru3ToMask(m0, m1, m2, m3)
void ECANLinkRXF4Thru7ToMask(m4, m5, m6, m7)
void ECANLinkRXF8Thru11ToMask(m8, m9, m10, m11)
void ECANLinkRXF12Thru15ToMask(m12, m13, m14, m15)
```

Parameters

m

[in] Name of mask that is to be linked to RXFn filter. The only permitted values are:

Value	Meaning
ECAN_RXM0	Link to RXM0 mask
ECAN_RXM1	Link to RXM1 mask
ECAN_RXMF15	Link to RXF15 mask

Return Values

None

Pre-condition

```
(ECAN_LIB_MODE_VAL == ECAN_LIB_MODE_RUN_TIME)
or
(ECAN_FUNC_MODE_VAL != ECAN_MODE_0)
```

Side Effects

None

Remarks

These macros perform compile time operations to reduce generated code. If possible, always supply a constant value of permitted type. A variable argument will result in larger code.

Example: Usage of ECANLinkRXFnThrumToMask

```
// Link RXF0 to RXM0, RXF1 to RXM0, RXF2 to RXM1 and RXF3 to RXM1
ECANLinkRXF0Thru3ToMask(ECAN_RXM0,
                        ECAN_RXM0,
                        ECAN_RXM1,
                        ECAN_RXM1);

...
```

ECANLoadRTRBuffer

This function loads the given message to the specified buffer that is configured for automatic RTR handling. This function is available in Mode 1 and Mode 2 only.

Syntax

```

BOOL ECANLoadRTRBuffer (BYTE buffer,
                        unsigned long id,
                        BYTE *data,
                        BYTE dataLen,
                        BYTE type)

```

Parameters*buffer*

[in] Programmable buffer number that is to be loaded. The possible values are 0 through 5, inclusive. Use 0 for B0, 1 for B1 and so on.

id

[in] 32-bit Identifier value which may correspond to right justified 11-bit Standard Identifier or 29-bit Extended Identifier. The exact number of bits to use depends on *type* parameter.

data

[in] Pointer to zero or more data bytes to send.

dataLen

[in] Number of bytes to send.

type

[in] Specifies an enumerated value of the message type. The possible values of all variables are listed in the following table:

Value	Meaning
ECAN_MSG_STD	Standard Message type
ECAN_MAS_XTD	Extended Message type

Return Values

TRUE: If the given message was loaded into the given buffer.

FALSE: If the given buffer was not set up for automatic RTR handling or is in the middle of automatic transmission.

Pre-condition

buffer must be configured for automatic RTR handling.

Side Effects

None

Remarks

This function makes sure that the given buffer is not in the middle of auto-transmitting the RTR response. Since the automatic RTR response may get transmitted at any time, even while the function is loading new data, this function first starts by clearing the DLC byte. As a result, if the automatic RTR response is to start in the middle of the update, it will result in a 0 byte long response. The remote node that made the RTR request must detect the 0 byte long message as incomplete and issue another RTR.

AN878

ECANLoadRTRBuffer (*Continued*)

Example: Usage of ECANLoadRTRBuffer

```
// Use ECAN.def options to initialize ECAN module.
// ECAN is set up in Mode 1 and B5 is configured as AutoRTR
ECANInitialize();
// B5 is not set up correctly to be AutoRTR buffer.

...

while(1)
{
    // Perform regular logic
    ...

    // At some point in time, we need to update RTR buffer
    if ( !ECANLoadRTRBuffer(5, // Load B5 buffer
                            newData,
                            newDataLen,
                            ECAN_MSG_STD) )
    {
        // B5 buffer must be in middle of transmission
        // Either try again or do it in next execution.
    }

    ...
}
```


ECANReceiveMessage

This function copies one of the full receive buffer messages into the given buffer and marks the full receive buffer as empty.

Syntax

```

BOOL ECANReceiveMessage(unsigned long *id,
                        BYTE *data,
                        BYTE dataLen,
                        ECAN_RX_MSG_FLAGS *msgFlags)

```

Parameters*id*

[out] 32-bit Identifier value which may correspond to right justified 11-bit Standard Identifier or 29-bit Extended Identifier. The exact number of bits to use depends on *msgFlags* parameter.

data

[out] Pointer to zero or more data bytes to send.

dataLen

[out] Number of bytes to send.

msgFlags

[out] Specifies an enumerated value of the type `ECAN_RX_MSG_FLAGS`. This represents the logical OR of one or more flags. The possible values of all variables are listed in the table below:

Value	Meaning
<code>ECAN_RX_OVERFLOW</code>	Specifies receive buffer overflow
<code>ECAN_RX_INVALID_MSG</code>	Specifies invalid message
<code>ECAN_RX_XTD_FRAME</code>	Specifies Extended Identifier message
<code>ECAN_RX_STD_FRAME</code>	Specifies Standard Identifier message
<code>ECAN_RX_DBL_BUFFERED</code>	Specifies that this message was double-buffered

If a flag bit is set, the corresponding meaning is `TRUE`; if cleared, the corresponding meaning is `FALSE`.

Return Values

`TRUE`: If new message was copied to the given buffer.

`FALSE`: If no new message was found.

Pre-condition

The *id*, *data*, *dataLen* and *msgflags* pointer must point to the desired and valid memory locations.

Side Effects

None

Remarks

This function will return `FALSE` if there are no new messages to read. Caller may check the return value to determine new message availability. Additional information, such as which filter caused this reception, may be determined by calling the `ECANGetFilterHitInfo()` macro.

AN878

ECANReceiveMessage (*Continued*)

Example: Usage of ECANReceiveMessage

```
// Use ECAN.def options to initialize ECAN module.
ECANInitialize();

...

while(1)
{
    // Send a message.
    if ( !ECANReceiveMessage(&myID,
                             &myData,
                             &dataLen,
                             &rxFlags) )
    {
        // There was no message waiting. Try again...
        ...
    }

    // A message is received.
    if ( rxFlags & ECAN_RX_XTD_FRAME )
        // This is Extended message.
    else
        // This is Standard message.

    // Find out what filter causes this acceptance.
    filterNumber = ECANGetFilterHitInfo();

    ...
}
```

ECANSendMessage

This function copies the given message to one of the empty transmit buffers and marks it as ready to be transmitted.

Syntax

```
void ECANSendMessage(unsigned long id,
                    BYTE *data,
                    BYTE dataLen,
                    ECAN_TX_MSG_FLAGS msgFlags)
```

Parameters

id

[in] 32-bit Identifier value which may correspond to right justified 11-bit Standard Identifier or 29-bit Extended Identifier. The exact number of bits to use depends on *msgFlags* parameter.

data

[in] Pointer to zero or more data bytes to send.

dataLen

[in] Number of bytes to send.

msgFlags

[in] Specifies an enumerated value of the type `ECAN_TX_MSG_FLAGS`. This represents the logical OR of a priority value, an Identifier type value and a message type value. The possible values of all variables are listed in the tables below:

Priority Value	Meaning
<code>ECAN_TX_PRIORITY_0</code>	Specifies Transmit Priority 0
<code>ECAN_TX_PRIORITY_1</code>	Specifies Transmit Priority 1
<code>ECAN_TX_PRIORITY_2</code>	Specifies Transmit Priority 2
<code>ECAN_TX_PRIORITY_3</code>	Specifies Transmit Priority 3

Note: See the PIC18F6585/6680/8585/8680 data sheet for further details on transmit priority.

Identifier Type Value	Meaning
<code>ECAN_TX_STD_FRAME</code>	Specifies Standard Identifier message
<code>ECAN_TX_XTD_FRAME</code>	Specifies Extended Identifier message

Message Value	Meaning
<code>ECAN_TX_NO_RTR_FRAME</code>	Specifies regular message – not RTR
<code>ECAN_TX_RTR_FRAME</code>	Specifies RTR message

Return Values

TRUE: If the given message was successfully placed in one of the empty transmit buffers.

FALSE: If all transmit buffers were full.

Pre-condition

None

Side Effects

None

Remarks

None

AN878

ECANSendMessage (*Continued*)

Example: Usage of ECANSendMessage

```
// Use ECAN.def options to initialize ECAN module.
ECANInitialize();

...

while(1)
{
    // Send a message.
    if ( !ECANSendMessage(myID,
                          myData,
                          dataLen,
                          CAN_TX_PRIORITY_0) )
    {
        // All buffers are full. Try again...
        ...
    }

    // Message was successfully loaded in an empty buffer.
    ...
}
```

ECANSetBaudRate

This macro sets baud rate values.

Syntax

```
void ECANSetBaudRate(sjw, brp, phseg1, phseg2, propseg)
```

Parameters

sjw

[in] SJW value - must be between 1 through 4.

brp

[in] BRP value - must be between 1 through 64.

phseg1

[in] PHSEG1 value - must be between 1 through 8.

phseg2

[in] PHSEG2 value - must be between 1 through 8.

propseg

[in] PROPSEG value - must be between 1 through 8.

Return Values

None

Pre-condition

ECAN must be in Configuration mode.

Side Effects

None

Remarks

These macros perform compile time operations to reduce generated code. If possible, always supply a constant value of permitted type. A variable argument will result in larger code.

Example: Usage of ECANSetBaudRate

```
// Switch to Configuration Mode
ECANSetOperationMode(ECAN_OP_MODE_CONFIG);

// Set 125kbps @ 20MHz
ECANSetBaudRate(1, 5, 7, 6, 2);

...
```

AN878

ECANSetBnAutoRTRMode

This macro enables/disables the automatic RTR handling capability of the specified programmable buffer. There are a total of 6 macros, one for each programmable buffer. For example, for Buffer B0, use `ECANSetB0AutoRTRMode`, for B1, use `ECANSetB1AutoRTRMode` and so on.

Syntax

```
void ECANSetBnAutoRTRMode (BYTE mode)
```

Parameters

mode

[in] The only permitted values are:

Value	Meaning
<code>ECAN_AUTORTR_MODE_DISABLE</code>	Do not configure buffer for automatic RTR handling
<code>ECAN_AUTORTR_MODE_ENABLE</code>	Configure buffer for automatic RTR handling

Return Values

None

Pre-condition

```
(ECAN_LIB_MODE_VAL == ECAN_LIB_MODE_RUN_TIME)
```

or

```
(ECAN_FUNC_MODE_VAL != ECAN_MODE_0)
```

Side Effects

None

Remarks

The selected buffer must also be set up as a transmit buffer using the `ECAN_Bn_TXRX_MODE_VAL` compile time option or by using the `ECANSetBnTxRxMode` macro at run-time.

Example: Usage of `ECANSetBnAutoRTRMode`

```
// Set B4 and B5 for automatic RTR handling
ECANSetB4AutoRTRMode (ECAN_AUTORTR_MODE_ENABLE);
ECANSetB5AutoRTRMode (ECAN_AUTORTR_MODE_ENABLE);
...
```

ECANSetBnRxMode

This macro sets the Receive mode for the programmable receive buffer. This macro is available in Mode 1 and Mode 2 only.

Syntax

```
void ECANSetBnRxMode(buffer, mode)
```

Parameters

buffer

[in] Name of programmable receive buffer that needs to be set up. The only permitted values are:

Value	Meaning
B0	Set up B0 buffer
B1	Set up B1 buffer
B2	Set up B2 buffer
B3	Set up B3 buffer
B4	Set up B4 buffer
B5	Set up B5 buffer

mode

[in] Mode to set up. The only permitted values are:

Value	Meaning
ECAN_RECEIVE_ALL_VALID	Receive all valid messages
ECAN_RECEIVE_ALL	Receive all including invalid messages

Return Values

None

Pre-condition

```
(ECAN_LIB_MODE_VAL == ECAN_LIB_MODE_RUN_TIME)
```

or

```
(ECAN_FUNC_MODE_VAL != ECAN_MODE_0)
```

Side Effects

None

Remarks

Programmable buffer value must be a constant of permitted values only. A variable parameter would cause a compile time error. For example, `ECANSetBnRxMode(myBuffer, ECAN_RECEIVE_ALL)` would not compile.

Example: Usage of ECANSetBnRxMode

```
// Set B0 to receive all valid messages.
ECANSetBnRxMode(B0, ECAN_RECEIVE_ALL);

// Set B1 to receive only Standard messages
ECANSetBnRxMode(B1, ECAN_RECEIVE_STANDARD);
...
```

AN878

ECANSetBnTxRxMode

This macro sets the Transmit or Receive mode for a specified buffer. This macro is available in Mode 1 and Mode 2 only.

Syntax

```
void ECANSetBnTxRxMode(buffer, mode)
```

Parameters

buffer

[in] Name of the programmable buffer that needs to be set up. The only permitted values are:

Value	Meaning
B0	Set up B0 buffer
B1	Set up B1 buffer
B2	Set up B2 buffer
B3	Set up B3 buffer
B4	Set up B4 buffer
B5	Set up B5 buffer

mode

[in] Mode to set up. The only permitted values are:

Value	Meaning
ECAN_BUFFER_RX	Buffer will be configured as receiver
ECAN_BUFFER_TX	Buffer will be configured as transmitter

Return Values

None

Pre-condition

```
(ECAN_LIB_MODE_VAL == ECAN_LIB_MODE_RUN_TIME)  
or  
(ECAN_FUNC_MODE_VAL != ECAN_MODE_0)
```

Side Effects

None

Remarks

Parameter *buffer* must be a constant of permitted values only. A variable parameter would cause a compile time error. For example, `ECANSetBnTxRxMode(myBuffer, ECAN_BUFFER_TX)` would not compile.

Example: Usage of ECANSetBnTxRxMode

```
// Set B0-B3 as receiver.  
ECANSetBnTxRxMode(B0, ECAN_BUFFER_RX);  
ECANSetBnTxRxMode(B1, ECAN_BUFFER_RX);  
ECANSetBnTxRxMode(B2, ECAN_BUFFER_RX);  
ECANSetBnTxRxMode(B3, ECAN_BUFFER_RX);  
...
```


ECANSetBusSampleMode

This macro sets the CAN bus Sampling mode.

Syntax

```
void ECANSetBusSampleMode (BYTE mode)
```

Parameters

mode

[in] The only permitted values are:

Value	Meaning
ECAN_BUS_SAMPLE_MODE_THRICE	Specifies that the CAN bus be sampled three times before sampling point
ECAN_BUS_SAMPLE_MODE_ONCE	Specifies that the CAN bus be sampled once at sampling point

Return Values

None

Pre-condition

ECAN module must be in Configuration mode.

Side Effects

None

Remarks

None

Example 1: Usage of ECANSetBusSampleMode

```
// Set configuration mode.
ECANSetOperationMode (ECAN_OP_MODE_CONFIG);

// Change Bus Sampling mode
ECANSetBusSampleMode (ECAN_BUS_SAMPLE_MODE_THRICE);

...
```

AN878

ECANSetCANTX2Mode

This macro sets the CANTX2 pin source mode.

Syntax

```
void ECANSetCANTX2Mode (BYTE mode)
```

Parameters

mode

[in] The only permitted values are:

Value	Meaning
ECAN_TX2_SOURCE_COMP	Specifies complement of CANTX1 as source.
ECAN_TX2_SOURCE_CLOCK	Specifies CAN clock as source.

Return Values

None

Pre-condition

None

Side Effects

None

Remarks

This macro automatically enables the CANTX2 pin as CANTX pin. Use ECANDisableCANTX2() to configure CANTX2 pin as a digital I/O.

Example: Usage of ECANSetCANTX2Mode

```
// Set CAN clock as CANTX2 source.  
ECANSetCANTX2Mode (ECAN_TX2_SOURCE_CLOCK);  
...
```

ECANSetCaptureMode

This macro enables the CAN Time-Stamp mode.

Syntax

```
void ECANSetCaptureMode (BYTE mode)
```

Parameters

mode

[in] The only permitted values are:

Value	Meaning
ECAN_CAPTURE_MODE_ENABLE	Enables Time-Stamp mode. CCP1 must be configured separately.
ECAN_CAPTURE_MODE_DISABLE	Disables Time-Stamp mode

Return Values

None

Pre-condition

None

Side Effects

None

Remarks

None

Example: Usage of ECANSetCaptureMode

```
// Enable timestamp mode
ECANSetCaptureMode (ECAN_CAPTURE_MODE_ENABLE);

// Must set up CCP1 correctly...
...
```

AN878

ECANSetFilterMode

This macro sets the CAN bus Wake-up Filter mode.

Syntax

```
void ECANSetFilterMode(BYTE mode)
```

Parameters

mode

[in] The only permitted values are:

Value	Meaning
ECAN_FILTER_MODE_DISABLE	Specifies that the low-pass filter be disabled
ECAN_FILTER_MODE_ENABLE	Specifies that the low-pass filter be enabled

Return Values

None

Pre-condition

ECAN module must be in Configuration mode.

Side Effects

None

Remarks

None

Example: Usage of ECANSetFilterMode

```
// Set configuration mode.
ECANSetOperationMode(ECAN_OP_MODE_CONFIG);

// Enable wakeup mode
ECANSetWakeupMode(ECAN_WAKEUP_MODE_ENABLE);

// Enable low-pass filter
ECANSetFilterMode(ECAN_FILTER_MODE_ENABLE);
...
```

ECANSetFunctionalMode

This macro changes the ECAN module functional mode.

Syntax

```
void ECANSetFunctionalMode (BYTE mode)
```

Parameters

mode

[in] The only permitted values are:

Value	Meaning
ECAN_MODE_0	Specifies Mode 0
ECAN_MODE_1	Specifies Mode 1
ECAN_MODE_2	Specifies Mode 2

Return Values

None

Pre-condition

The ECAN module must be in Configuration mode and `ECAN_LIB_MODE_VAL = ECAN_LIB_MODE_RUN_TIME`.

Side Effects

None

Remarks

None

Example: Usage of ECANSetFunctionalMode

```
// Set configuration mode.
ECANSetOperationMode (ECAN_OP_MODE_CONFIG);

// Now change functional mode
ECANSetFunctionalMode (ECAN_MODE_2);

...
```

AN878

ECANSetOperationMode

This function changes the ECAN module operation mode.

Syntax

```
void ECANSetOperationMode(ECAN_OP_MODE mode)
```

Parameters

mode

[in] Specifies an enumerated value of the type `ECAN_OP_MODE`. The only permitted values are:

Value	Meaning
<code>ECAN_OP_MODE_NORMAL</code>	Specifies normal mode of operation
<code>ECAN_OP_MODE_SLEEP</code>	Specifies Sleep mode of operation
<code>ECAN_OP_MODE_LOOP</code>	Specifies Loopback mode of operation
<code>ECAN_OP_MODE_LISTEN</code>	Specifies Listen Only mode of operation
<code>ECAN_OP_MODE_CONFIG</code>	Specifies Configuration mode of operation

Return Values

None

Pre-condition

None

Side Effects

None

Remarks

This is a blocking function. It waits for a given mode to be accepted by the ECAN module and then returns the control. If a non-blocking call is required, see the `ECANSetOperationModeNoWait` macro.

Example: Usage of `ECANSetOperationMode`

```
// Set configuration mode.  
ECANSetOperationMode(ECAN_OP_MODE_CONFIG);  
// Module is in Configuration mode.  
...
```

ECANSetOperationModeNoWait

This macro changes the ECAN module operation mode.

Syntax

```
void ECANSetOperationMode(ECAN_OP_MODE mode)
```

Parameters

mode

[in] Specifies an enumerated value of the type `ECAN_OP_MODE`. The only permitted values are:

Value	Meaning
<code>ECAN_OP_MODE_NORMAL</code>	Specifies normal mode of operation
<code>ECAN_OP_MODE_SLEEP</code>	Specifies Sleep mode of operation
<code>ECAN_OP_MODE_LOOP</code>	Specifies Loopback mode of operation
<code>ECAN_OP_MODE_LISTEN</code>	Specifies Listen Only mode of operation
<code>ECAN_OP_MODE_CONFIG</code>	Specifies Configuration mode of operation

Return Values

None

Pre-condition

None

Side Effects

None

Remarks

This is a non-blocking macro. It requests the given mode of operation and immediately returns the control. Caller must ensure the desired mode of operation is set before performing any mode specific operation. If a blocking call is required, see the `ECANSetOperationMode` function.

Example: Usage of ECANSetOperationModeNoWait

```
// Set configuration mode.
ECANSetOperationModeNoWait(ECAN_OP_MODE_CONFIG);

// Wait for mode to get accepted
while( ECANGetOperationMode() != ECAN_OP_MODE_CONFIG )
{
    // Do something while mode is being accepted.
}
...
```

AN878

ECANSetPHSEG2Mode

This macro sets Phase Segment 2 Programmability mode.

Syntax

```
void ECANSetPHSEG2Mode (BYTE mode)
```

Parameters

mode

[in] The only permitted values are:

Value	Meaning
ECAN_PHSEG2_MODE_AUTOMATIC	Phase Segment 2 will be automatically programmed by ECAN module
ECAN_PHSEG2_MODE_PROGRAMMABLE	Phase Segment 2 will be manually programmable

Return Values

None

Pre-condition

The ECAN module must be in Configuration mode.

Side Effects

None

Remarks

None

Example: Usage of ECANSetPHSEG2Mode

```
// Set configuration mode.
ECANSetOperationMode (ECAN_OP_MODE_CONFIG);

// Set Phase Segment2 as manually programmable
ECANSetPHSEG2Mode (ECAN_PHSEG2_MODE_PROGRAMMABLE);

...
```


ECANSetRXB0Db1Buffer

This macro enables the hardware double-buffering option for the RXB0 buffer. This macro is available in Mode 1 only.

Syntax

```
void ECANSetRXB0Db1Buffer(mode)
```

Parameters

mode

[in] The only permitted values are:

Value	Meaning
ECAN_DBL_BUFFER_MODE_DISABLE	Disable double-buffering
ECAN_DBL_BUFFER_MODE_ENABLE	Enable double-buffering

Return Values

None

Pre-condition

```
(ECAN_LIB_MODE_VAL == ECAN_LIB_MODE_RUN_TIME)
```

or

```
(ECAN_FUNC_MODE_VAL == ECAN_MODE_0)
```

Side Effects

None

Remarks

None

Example: Usage of ECANSetRXB0Db1Buffer

```
// Enable hardware double buffering for RXB0
ECANSetRXB0Db1Buffer(ECAN_DBL_BUFFER_MODE_ENABLE);
...
```

AN878

ECANSetRxBnRxMode

This macro sets the Receive mode for the dedicated receive buffer.

Syntax

```
void ECANSetRxBnRxMode(buffer, mode)
```

Parameters

buffer

[in] Name of dedicated receive buffer that needs to be set up. The only permitted values are:

Value	Meaning
RXB0	Set up RXB0 buffer
RXB1	Set up RXB1 buffer

mode

[in] Mode to set up. The only permitted values are:

Value	Meaning
ECAN_RECEIVE_ALL_VALID	Receive all valid messages
ECAN_RECEIVE_STANDARD	Receive only Standard messages
ECAN_RECEIVE_EXTENDED	Receive only Extended messages
ECAN_RECEIVE_ALL	Receive all including invalid messages

Return Values

None

Pre-condition

None

Side Effects

None

Remarks

Dedicated buffer value must be a constant of permitted values only. A variable parameter would cause a compile time error. For example, `ECANSetRxBnRxMode(myBuffer, ECAN_RECEIVE_ALL)` would not compile.

Example: Usage of ECANSetRxBnRxMode

```
// Set RXB0 to receive all valid messages.
ECANSetRxBnRxMode(RXB0, ECAN_RECEIVE_ALL);

// Set RXB1 to receive only Standard messages
ECANSetRxBnRxMode(RXB1, ECAN_RECEIVE_STANDARD);
...
```

ECANSetRXFnValue

This macro sets the value for a filter register. There are total of 15 macros, one for each register. For example, for filter RXF0, use ECANSetRXF0Value, for RXF1, use ECANSetRXF1Value and so on.

Syntax

```
void ECANSetRXFnValue(value, type)
```

Parameters

value

[in] Value to be set. Range of value is dependent on *type*.

type

[in] Type of mask. The only permitted values are:

Value	Meaning
ECAN_MSG_STD	Standard type. 11-bit of <i>value</i> will be used.
ECAN_MSG_XTD	Extended type. 29-bit of <i>value</i> will be used.

Return Values

None

Pre-condition

The ECAN module must be in Configuration mode. To use ECANSetRXF6Value through ECANSetRXF15Value, (ECAN_LIB_MODE_VAL == ECAN_LIB_MODE_RUN_TIME)

or

(ECAN_FUNC_MODE_VAL != ECAN_MODE_0) must be TRUE.

Side Effects

None

Remarks

In Mode 0, only ECANSetRXF0Value through ECANSetRXF5Value are available. In Mode 1 and Mode 2, ECANRXF0Value through ECANRXF15Value are available.

Example: Usage of ECANSetRXFnValue

```
// Set RXF0 as Standard and RXF5 as Extended filter.
ECANSetRXF0Value(0x123, ECAN_MSG_STD);
ECANSetRXF5Value(0xabcd, ECAN_MSG_XTD);
...
```

AN878

ECANSetRXMnValue

This macro sets the value for a mask register. There are a total of two macros, one for each mask. For example, for mask RXM0, use ECANSetRXM0Value, for RXM1, use ECANSetRXM1Value and so on.

Syntax

```
void ECANSetRXMnValue(value, type)
```

Parameters

value

[in] Value to be set. Range of value is dependent on *type*.

type

[in] Type of mask. The only permitted values are:

Value	Meaning
ECAN_MSG_STD	Standard type. 11-bit of <i>value</i> will be used.
ECAN_MSG_XTD	Extended type. 29-bit of <i>value</i> will be used.

Return Values

None

Pre-condition

The ECAN module must be in Configuration mode.

Side Effects

None

Remarks

None

Example: Usage of ECANSetRXMnValue

```
// Set RXM0 as Standard and RXM1 as Extended mask values.  
ECANSetRXM0Value(0x123, ECAN_MSG_STD);  
ECANSetRXM1Value(0xabcd, ECAN_MSG_XTD);  
...
```

ECANSetTxDriveMode

This macro sets the CANTX pin Recessive State Drive mode.

Syntax

```
void ECANSetTxDriveMode(BYTE mode)
```

Parameters

mode

[in] The only permitted values are:

Value	Meaning
ECAN_TXDRIVE_MODE_TRISTATE	Specifies that CANTX pin be driven to tri-state in recessive state
ECAN_TXDRIVE_MODE_VDD	Specifies that CANTX pin be driven to VDD in recessive state

Return Values

None

Pre-condition

None

Side Effects

None

Remarks

None

Example: Usage of ECANSetTxDriveMode

```
// Set CANTX to Vdd in recessive state.
ECANSetTxDriveMode(ECAN_TXDRIVE_MODE_VDD);
...
```

AN878

ECANSetWakeupMode

This macro sets the CAN bus Activity Wake-up mode.

Syntax

```
void ECANSetWakeupMode (BYTE mode)
```

Parameters

mode

[in] The only permitted values are:

Value	Meaning
ECAN_WAKEUP_MODE_ENABLE	Specifies that CAN bus Activity Wake-up mode be enabled
ECAN_WAKEUP_MODE_DISABLE	Specifies that CAN bus Activity Wake-up mode be disabled

Return Values

None

Pre-condition

ECAN module must be in Configuration mode.

Side Effects

None

Remarks

None

Example: Usage of ECANSetWakeupMode

```
// Set configuration mode.
ECANSetOperationMode (ECAN_OP_MODE_CONFIG);

// Enable wakeup mode
ECANSetWakeupMode (ECAN_WAKEUP_MODE_ENABLE);

...
```

APPENDIX C: SOURCE CODE

The complete source code, including any demo applications and necessary support files, is available for download as a single archive file from the Microchip corporate web site, at:

www.microchip.com

AN878

NOTES:

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Information contained in this publication regarding device applications and the like is intended through suggestion only and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. No representation or warranty is given and no liability is assumed by Microchip Technology Incorporated with respect to the accuracy or use of such information, or infringement of patents or other intellectual property rights arising from such use or otherwise. Use of Microchip's products as critical components in life support systems is not authorized except with express written approval by Microchip. No licenses are conveyed, implicitly or otherwise, under any intellectual property rights.

Trademarks

The Microchip name and logo, the Microchip logo, dsPIC, KEELOQ, MPLAB, PIC, PICmicro, PICSTART, PRO MATE and PowerSmart are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.


FilterLab, microID, MXDEV, MXLAB, PICMASTER, SEEVAL and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

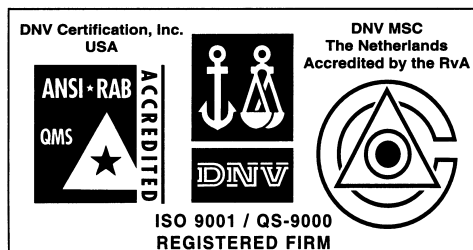
Accuron, Application Maestro, dsPICDEM, dsPICDEM.net, ECAN, ECONOMONITOR, FanSense, FlexROM, fuzzyLAB, In-Circuit Serial Programming, ICSP, ICEPIC, microPort, Migratable Memory, MPASM, MPLIB, MPLINK, MPSIM, PICC, PICkit, PICDEM, PICDEM.net, PowerCal, PowerInfo, PowerMate, PowerTool, rLAB, rPIC, Select Mode, SmartSensor, SmartShunt, SmartTel and Total Endurance are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

Serialized Quick Turn Programming (SQTP) is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.

© 2003, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

 Printed on recycled paper.



Microchip received QS-9000 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona in July 1999 and Mountain View, California in March 2002. The Company's quality system processes and procedures are QS-9000 compliant for its PICmicro® 8-bit MCUs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, non-volatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001 certified.



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: 480-792-7627
Web Address: <http://www.microchip.com>

Atlanta

3780 Mansell Road, Suite 130
Alpharetta, GA 30022
Tel: 770-640-0034
Fax: 770-640-0307

Boston

2 Lan Drive, Suite 120
Westford, MA 01886
Tel: 978-692-3848
Fax: 978-692-3821

Chicago

333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 630-285-0071
Fax: 630-285-0075

Dallas

4570 Westgrove Drive, Suite 160
Addison, TX 75001
Tel: 972-818-7423
Fax: 972-818-2924

Detroit

Tri-Atria Office Building
32255 Northwestern Highway, Suite 190
Farmington Hills, MI 48334
Tel: 248-538-2250
Fax: 248-538-2260

Kokomo

2767 S. Albright Road
Kokomo, IN 46902
Tel: 765-864-8360
Fax: 765-864-8387

Los Angeles

18201 Von Karman, Suite 1090
Irvine, CA 92612
Tel: 949-263-1888
Fax: 949-263-1338

Phoenix

2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7966
Fax: 480-792-4338

San Jose

2107 North First Street, Suite 590
San Jose, CA 95131
Tel: 408-436-7950
Fax: 408-436-7955

Toronto

6285 Northam Drive, Suite 108
Mississauga, Ontario L4V 1X5, Canada
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Australia

Suite 22, 41 Rawson Street
Epping 2121, NSW
Australia
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing

Unit 915
Bei Hai Wan Tai Bldg.
No. 6 Chaoyangmen Beidajie
Beijing, 100027, No. China
Tel: 86-10-85282100
Fax: 86-10-85282104

China - Chengdu

Rm. 2401-2402, 24th Floor,
Ming Xing Financial Tower
No. 88 TIDU Street
Chengdu 610016, China
Tel: 86-28-86766200
Fax: 86-28-86766599

China - Fuzhou

Unit 28F, World Trade Plaza
No. 71 Wusi Road
Fuzhou 350001, China
Tel: 86-591-7503506
Fax: 86-591-7503521

China - Hong Kong SAR

Unit 901-6, Tower 2, Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2401-1200
Fax: 852-2401-3431

China - Shanghai

Room 701, Bldg. B
Far East International Plaza
No. 317 Xian Xia Road
Shanghai, 200051
Tel: 86-21-6275-5700
Fax: 86-21-6275-5060

China - Shenzhen

Rm. 1812, 18/F, Building A, United Plaza
No. 5022 Binhe Road, Futian District
Shenzhen 518033, China
Tel: 86-755-82901380
Fax: 86-755-8295-1393

China - Shunde

Room 401, Hongjian Building
No. 2 Fengxiangnan Road, Ronggui Town
Shunde City, Guangdong 528303, China
Tel: 86-765-8395507 Fax: 86-765-8395571

China - Qingdao

Rm. B505A, Fullhope Plaza,
No. 12 Hong Kong Central Rd.
Qingdao 266071, China
Tel: 86-532-5027355 Fax: 86-532-5027205

India

Divyasree Chambers
1 Floor, Wing A (A3/A4)
No. 11, O'Shaughnessy Road
Bangalore, 560 025, India
Tel: 91-80-2290061 Fax: 91-80-2290062

Japan

Benex S-1 6F
3-18-20, Shinyokohama
Kohoku-Ku, Yokohama-shi
Kanagawa, 222-0033, Japan
Tel: 81-45-471-6166 Fax: 81-45-471-6122

Korea

168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku
Seoul, Korea 135-882
Tel: 82-2-554-7200 Fax: 82-2-558-5932 or
82-2-558-5934

Singapore

200 Middle Road
#07-02 Prime Centre
Singapore, 188980
Tel: 65-6334-8870 Fax: 65-6334-8850

Taiwan

Kaohsiung Branch
30F - 1 No. 8
Min Chuan 2nd Road
Kaohsiung 806, Taiwan
Tel: 886-7-536-4818
Fax: 886-7-536-4803

Taiwan

Taiwan Branch
11F-3, No. 207
Tung Hua North Road
Taipei, 105, Taiwan
Tel: 886-2-2717-7175 Fax: 886-2-2545-0139

EUROPE

Austria

Durisolstrasse 2
A-4600 Wels
Austria
Tel: 43-7242-2244-399
Fax: 43-7242-2244-393

Denmark

Regus Business Centre
Lautrup høj 1-3
Ballerup DK-2750 Denmark
Tel: 45-4420-9895 Fax: 45-4420-9910

France

Parc d'Activite du Moulin de Massy
43 Rue du Saule Trapu
Batiment A - 1er Etage
91300 Massy, France
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany

Steinheilstrasse 10
D-85737 Ismaning, Germany
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Italy

Via Quasimodo, 12
20025 Legnano (MI)
Milan, Italy
Tel: 39-0331-742611
Fax: 39-0331-466781

Netherlands

P. A. De Biesbosch 14
NL-5152 SC Drunen, Netherlands
Tel: 31-416-690399
Fax: 31-416-690340

United Kingdom

505 Eskdale Road
Winnersh Triangle
Wokingham
Berkshire, England RG41 5TU
Tel: 44-118-921-5869
Fax: 44-118-921-5820

07/28/03