BOSCH
Invented for life

M_CAN

Modular CAN IP-module

Interrupt Handling

Application Note M_CAN_AN003

Document Revision 1.1
29.06.2018

Robert Bosch GmbH
Automotive Electronics

LEGAL NOTICE

Revision History

Version	Date	Remark
1.0	28.09.2015	First version for M_CAN 3.1.0 - 3.2.1
1.1	29.06.2018	Source code updated

Conventions

The following conventions are used within this document:

Register Names	**RXBC, SIDFC**
Names of files and directories	`directoryname/filename`
Source code	m_can_interrupt_init(. .)

References

This document refers to the following documents:

Ref	Author	Title
[1]	AE/PJ-SCI	M_CAN User's Manual
[2]	AE/PJ-SCI	M_CAN System Integration Guide

Terms and Abbreviations

This document uses the following terms and abbreviations:

Term	Meaning
BRP	Baud Rate Prescaler
CAN	Controller Area Network
CRC	Cyclic Redundancy Check
DLC	Data Length Code
ISR	Interrupt Service Routine

Table of Contents

1 Target

This application note describes the **interrupt handling** in the **M_CAN versions 3.1.0 up to 3.2.x**

Note: Software examples in this application note are only for illustration purposes. Use the examples on own risk.

2 Interrupt Lines

Number of Lines

The M_CAN has 2 interrupt lines: **`m_can_int0`** and **`m_can_int1`**.

Interrupt Type

The M_CAN provides **level-triggered** interrupt lines:

- high ('1') = interrupt active
- low ('0') = interrupt not active

When the M_CAN has to signal interrupts it sets the interrupt line 1 and/or 2 to logic '1'. The M_CAN holds it at that level until the Host services the interrupt.

3 Interrupt registers

General

The registers used for interrupt handling are:

- **IR**: Interrupt Register

  The **IR** register contains the interrupt flags. The M_CAN sets the flags based on specific conditions. For example **IR.RF0N** flag (Rx FIFO 0 New Message) is set when a new message arrives in Rx FIFO 0. The flags remain set until the Host clears them.

  The M_CAN always sets the interrupt flags in **IR** register when the respective condition occurs. But, only those interrupt flags that are enabled in **IE** register will be signalled via an interrupt line.

- **IE**: Interrupt Enable

  Each interrupt flag can be enabled/disabled via register **IE**. Disabling an interrupt via **IE** (Interrupt Enable) register has no effect on the setting of the interrupt flag in **IR** register. The **IE** register masks the **IR** register before the values of the interrupt lines are generated. Figure 1 on page 5 shows how the **IE** register is involved in the interrupt generation. By default all interrupts are disabled.

- **ILS**: Interrupt Line Select

  Each interrupt flag can be assigned to one of the two interrupt lines by configuring the register **ILS**. By default all interrupts are assigned to **`m_can_int0`**.

- **ILE**: Interrupt Line Enable

  Register **ILE** is used to enable/disable the interrupt lines. Each of the two interrupt lines to the CPU can be enabled / disabled separately.

Additional

- The interrupt flag **IR.TCF** (Transmission Cancellation Finished) is set based on the configuration in register **TXBCIE** (Tx Buffer Cancellation Finished Interrupt Enable). M_CAN can have a maximum of 32 Tx Buffers. Each Tx Buffer can be enabled/disabled to trigger an **IR.TCF** interrupt by configuring register **TXBCIE**. For example, if **TXBCIE** = 0x2 and the cancellation of a transmission request via Tx Buffer 1 has finished (can be known from **TXBCF** register), then **IR.TCF** interrupt flag will be set in **IR** register.

- The interrupt flag **IR.TC** (Transmission Completed) is set based on the configuration in register **TXBTIE** (Tx Buffer Transmission Interrupt Enable). M_CAN can have a maximum of 32 Tx Buffers. Each Tx Buffer can be enabled/disabled to trigger an **IR.TC** interrupt by configuring register **TXBTIE**. For example, if **TXBTIE** = 0x4 and the transmission of a message via Tx Buffer 2 has finished (can be known from **TXBTO** register), then **IR.TC** interrupt flag will be set in **IR** register.

4 Clear an Interrupt

Each interrupt flag can be cleared individually. To clear a flag, the Host (CPU) has to write a ‘1’ to the corresponding bit position. Writing a ‘0’ has no effect.

5 Interrupt Notification

Figure 1 shows how the M_CAN notifies the Host about a generated interrupt via an interrupt line. The process can be summarized as:

1. The M_CAN sets the interrupt flags in **IR** register always when the respective flag condition occurs. Two Interrupt flags **IR.TCF** and **IR.TC** are set based on the configuration of registers **TXBCIE** and **TXBTIE** and the status of registers **TXBCF** and **TCBTO** respectively.
2. **Only those** interrupts that are enabled via register **IE** are signalled via an interrupt line to the Host. Consider, that the M_CAN sets an interrupt flag in the **IR** register even if that interrupt is disabled via the **IE** register.
3. Each interrupt can be assigned to one of the two interrupt lines by configuring register **ILS.**
4. The Host is notified of an interrupt via an interrupt line, if the selected interrupt line is enabled based on the bits **ILE.EINT0** and **ILE.EINT1** in **ILE** register.

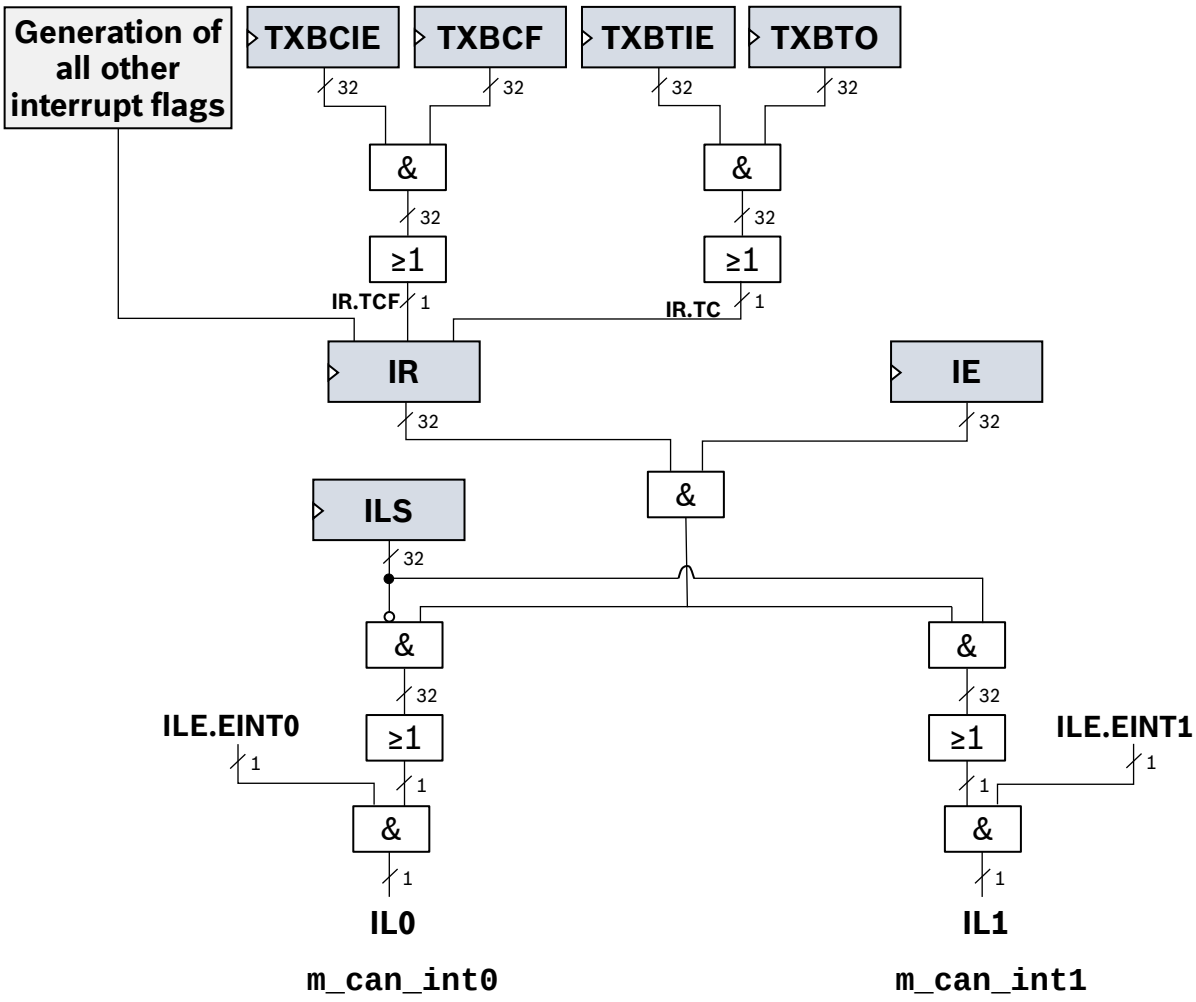

Figure 1: Interrupt notification in M_CAN

6 Interrupt Operation

Steps

1. The Host configures the interrupt registers. See chapter 3 and 5.
2. M_CAN will signal the interrupts via the 2 interrupt lines.
3. The Host executes an interrupt service routine (ISR) to handle the M_CAN's interrupts.
4. The ISR reads the **IR** register of the M_CAN, to find out which interrupt flags are set.
5. Optional: The Host can mask out all the interrupt flags that it is not interested in.
6. The Host ISR handles all interrupt flags and clears the flags. See chapter 4.
7. The ISR terminates.

Hints

Carefully design your interrupt handling strategy, i.e. how you handle individual interrupts and how you clear the corresponding flag. Following two examples related to Rx FIFO0 demonstrates this.

1. Example: Clear interrupt flag **IR.RF0N** and then read all messages from Rx FIFO0
   - This method ensures that all messages will be read by the ISR from Rx FIFO0.
   - This is because the interrupt flag **IR.RF0N** will be set again if a new message arrives during reading Rx FIFO0. Consequently, the ISR will be called again and the Host will read the new message from the Rx FIFO0 at that time.

2. Example: Read all messages from Rx FIFO0 and clear interrupt flag **IR.RF0N**
   - This method **does not** ensure that all messages will be read by the ISR from Rx FIFO0.
   - Imagine the case that after reading the messages from Rx FIFO0, but before clearing the interrupt flag **IR.RF0N,** a new message arrives. Since in this case the interrupt flag **IR.RF0N** is cleared before reading the message, the ISR won't be called again. This means this message is forgotten in Rx FIFO0.
   - The forgotten message will also be read out, next time when a new message is stored in the Rx FIFO0 and the ISR is called again.

7 Software Examples

Table 1 lists C functions that demonstrate interrupt handling operations. The functions are provided with this application note.

Table 1: C functions that demonstrate interrupt handling

Name:	`m_can_interrupt_init(..)`
File:	`m_can/m_can.c`
Description:	This function configures the interrupt registers for handling the interrupts. The M_CAN should be in configuration change enable mode when this function is called.
Name:	`m_can_process_IRQ(..)`
File:	`m_can/m_can_irq_handling.c`
Description:	This function serves as the interrupt service routine of the M_CAN.
Name:	`m_can_an001_high_priority_message_handling(..)`
File:	`app_notes/app_note_001_rx_handling.c`
Description:	This example demonstrates high priority message handling. Interrupt registers of the Rx node are configured to handle specific interrupts like **IR.HPM** and **IR.RF0W.**

This application note contains all C-source files that are necessary to compile the examples. The file `_info.txt` contains a short description of each provided source file.

8 List of Tables

9 List of Figures