Freescale Semiconductor
Application Note

AN3488
Rev. 0, 06/2007

MPC5510 New FlexCAN Module Features

by: David Paterson
MCD Applications, East Kilbride

1 Introduction

The MPC5510 family of 32-bit microcontrollers is Freescale Semiconductor's latest achievement in integrated automotive application controllers.

The on-chip FlexCAN modules have been taken from the MPC5500 family and modified to provide some powerful new features while obtaining backwards compatibility with older CAN modules.

There are up to six enhanced full CAN (FlexCAN) modules with configurable buffers on MPC5510 microcontrollers. Each FlexCAN module is a communication controller implementing the CAN protocol according to Bosch Specification version 2.0B and ISO Standard 11898.

Contents

1 Introduction 1
2 New Module Features 2
2.1 Rx FIFO 2
2.2 Local Priority Transmission 5
2.3 Hardware Cancellation (Transmission Abort Mechanism) 5
3 Conclusion 6
Appendix A
CANx Module Configuration Register (CANx_MCR) 7
Appendix B
Configuring Tx Buffers Code Example 7

The FlexCAN module has been adapted to include some powerful new features and allow backwards compatibility with older CAN modules. These new features are:

- Full featured Rx FIFO
  - Storage capacity for six frames and internal pointer managing
  - Powerful Rx FIFO ID filtering, capable of matching incoming IDs against eight extended, 16 standard, or 32 partial (8 bits) IDs, with individual masking capability
- Additional local priority programmable transmission on individual Tx message buffers.
- Hardware cancellation on Tx message buffers.

These new features have selectable backwards compatibility with previous FlexCAN versions.

2 New Module Features

This section describes and discusses each of the new module features and examples of how to use these features.

2.1 Rx FIFO

When the FIFO Enable (FEN) bit is set in the CANx_MCR (see Appendix A), the memory area from 0x80 to 0xFF (which is normally occupied by MBs 0 to 7) is used by the reception FIFO engine.

Figure 1 shows the start of the memory structure for the message buffers when FEN equals 0.

Figure 2 shows the Rx FIFO data structure when FEN equals 1.

- Region 0x0 – 0xC contains an MB structure that is the port which the CPU reads data through from the FIFO (the oldest frame received and not read yet).
- The region 0x10 – 0xDF is reserved for internal use of the FIFO engine.
- The region 0xE0 – 0xFF contains an eight-entry ID table that specifies filtering criteria for accepting frames into the FIFO. Figure 3 shows the three different formats that the elements of the ID table can assume, depending on the IDAM field of the CANx_MCR. All elements of the table must have the same format.

This allows the CPU to read the received frames sequentially, in the order they were received, by repeatedly accessing a message buffer structure at the beginning of the memory.

NOTE

This means that message buffers 0-7 can not be used as standard Tx or Rx buffers.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x0					CODE					SRR	IDE	RTR	LENGTH				TIME STAMP															
0x4	PRIO			ID (Extended/Standard)											ID (Extended)																	
0x8	Data Byte 0								Data Byte 1								Data Byte 2								Data Byte 3							
0xC	Data Byte 4								Data Byte 5								Data Byte 6								Data Byte 7							
0x10					CODE					SRR	IDE	RTR	LENGTH				TIME STAMP															
0x14	PRIO			ID (Extended/Standard)											ID (Extended)																	
0x18	Data Byte 0								Data Byte 1								Data Byte 2								Data Byte 3							
0x1C	Data Byte 4								Data Byte 5								Data Byte 6								Data Byte 7							
0x20					CODE					SRR	IDE	RTR	LENGTH				TIME STAMP															
0x24	PRIO			ID (Extended/Standard)											ID (Extended)																	
0x28	Data Byte 0								Data Byte 1								Data Byte 2								Data Byte 3							
0x2C	Data Byte 4								Data Byte 5								Data Byte 6								Data Byte 7							

Figure 1. Default Memory Buffer Structure

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x0										SRR	IDE	RTR	LENGTH				TIME STAMP															
0x4				ID (Extended/Standard)											ID (Extended)																	
0x8	Data Byte 0								Data Byte 1								Data Byte 2								Data Byte 3							
0xC	Data Byte 4								Data Byte 5								Data Byte 6								Data Byte 7							
0x10 to 0xDF	Reserved																															
0xE0	ID Table 0																															
0xE4	ID Table 1																															
0xE8	ID Table 2																															
0xEC	ID Table 3																															
0xF0	ID Table 4																															
0xF4	ID Table 5																															
0xF8	ID Table 6																															
0xFC	ID Table 7																															

Figure 2. Rx FIFO Memory Structure

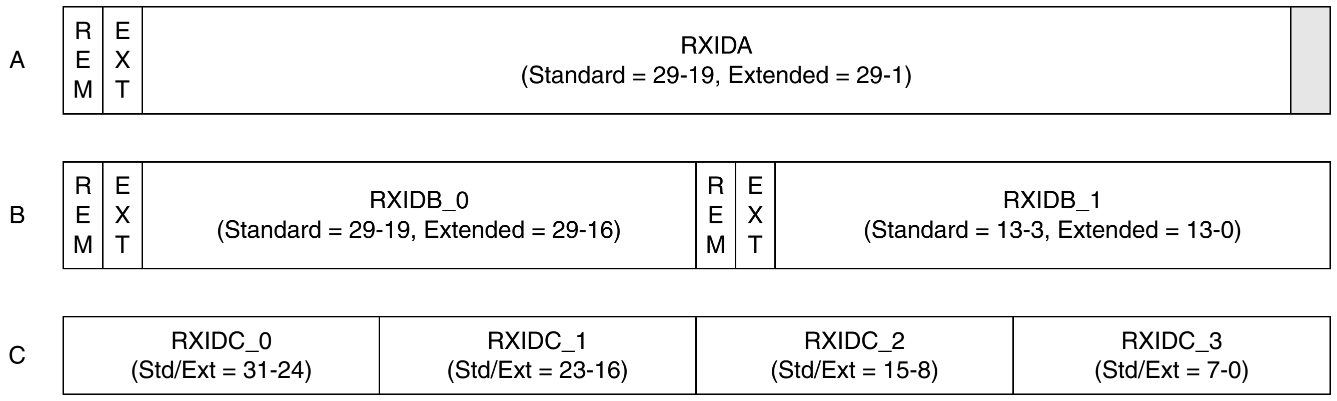

Figure 3. ID Table 0-7

Perform the following steps to use the Rx FIFO feature:

1. Set the FEN bit in the CANx_MCR.
2. Set the IDAM field in the CANx_MCR (see Figure 4).
3. Set the ID table entries with the appropriate IDs, depending on the format chosen in step 2.
4. Enable the CAN module for transmission and/or reception (after other initialization and configuration).
5. Allow polling of the FIFO interrupt flags (or set up interrupt routine if enabled) for FIFO activity:
   — BUF5I - Buffer MB5 Interrupt or Frames available in FIFO.
   — BUF6I - Buffer MB6 Interrupt or FIFO Warning. Four frames have accumulated in the FIFO.
   — BUF7I - Buffer MB7 Interrupt or FIFO Overflow. The FIFO is full and subsequent frames are not accepted until the CPU creates space in the FIFO by reading one or more frames. While the FIFO is full, the frames are only received if they are matched with another MB (MB8-MB63).
6. Read the frame (top FIFO entry).
7. Clear the FIFO interrupt flag. The act of clearing the interrupt triggers the FIFO engine to replace the MB with the next frame in the queue and then issues another interrupt to the CPU.

The FIFO can be disabled by clearing the FEN bit in the CANx_MCR when finished.

IDAM	Format	Explanation
00	A	One full ID (standard or extended) per filter element.
01	B	Two full standard IDs or two partial 14-bit extended IDs per filter element.
10	C	Four partial 8-bit IDs (standard or extended) per filter element.
11	D	All frames rejected.

Figure 4. IDAM Coding

The header file available from Freescale supports this new feature. If this is not used, the ID table can be declared as follows (CAN_A example):

```
#define  ID_Table0_CANA  (*(uint32_t* )(0xfffc00e0))
#define  ID_Table1_CANA  (*(uint32_t* )(0xfffc00e4))
#define  ID_Table7_CANA  (*(uint32_t* )(0xfffc00fC))
```

Without the Rx FIFO, receiving frames could involve message buffers having a high priority to read the data and then re-open the message buffer for the next frame. If the priority was not high enough or the CPU could not service the interrupt fast enough, it could result in Rx frames being overwritten. The Rx FIFO overcomes this problem by giving the CPU more time to service the message buffers, allowing up to six messages to be stored, giving more flexibility.

2.2 Local Priority Transmission

The term local priority refers to the priority of transmit messages of the host node. This allows increased control over the priority mechanism for transmitting messages. Figure 5 shows the placement of PRIO in the ID part of the message buffer.

An additional 3-bit field (PRIO) in the long-word ID part of the message buffer structure has been added for local priority determination. They are prefixed to the regular ID to define the transmission priority.

These bits are not transmitted and are intended only for Tx buffers.

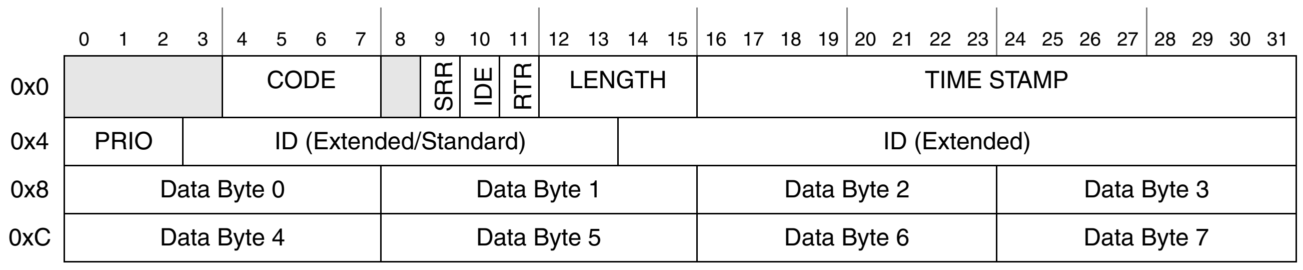

Figure 5. Message Buffer Structure

Perform the following to use the local priority feature:

1. Set the LPRIO_EN bit in the CANx_MCR.
2. Write the additional PRIO bits in the ID long-word of Tx message buffers when configuring the Tx buffers.

See Appendix B for a code example on configuring the Tx buffers and setting the local priority bits.

With this extended ID concept, the arbitration process is based on the full 32-bit word. However, the actual transmitted ID continues to have 11 bits for standard frames and 29 bits for extended frames.

2.3 Hardware Cancellation (Transmission Abort Mechanism)

The abort mechanism provides a safe way to request the abortion of a pending transmission. A feedback mechanism is provided to inform the CPU if the transmission was aborted or if the frame could not be aborted and was transmitted instead. This feature allows transmission to be aborted for the message buffer to be updated.

Set the AEN bit in the CANx_MCR to enable the transmission abort mechanism.

The CPU must then write a specific abort code (1001) to the code field of the control and status word of the Tx message that needs to be aborted.

If the abort code is written to an MB currently being transmitted or to an MB already loaded into the SMB (serial message buffer) for transmission, the write operation is blocked and the MB is not deactivated. However, the abort request is captured and kept pending until one of the following conditions are satisfied:

- The module loses the bus arbitration
- There is an error during the transmission
- The module is put into freeze mode

The abort procedure can be summarized as follows:

- CPU writes 1001 (binary) into the code field of the C/S word
- CPU reads the CODE field and compares it to the value that was written
- If the CODE field that was read is different from the value that was written, the CPU must read the corresponding IFLAG to check if the frame was transmitted or it is being currently transmitted. If the corresponding IFLAG is set, the frame was transmitted. If the corresponding IFLAG is reset, the CPU must wait for it to be set, and then the CPU must read the CODE field to check if the MB was aborted (CODE=1001) or transmitted (CODE=1000).

As an application example, this could avoid sending out-of-date data.

For example, if the data to be transmitted has been updated before the corresponding message is pending transmission, it can be aborted. Then, the data can be updated and the transmission sequence can restart. This is controlled by the application software.

3 Conclusion

These three current features added to the FlexCAN modules allow increased flexibility and backwards compatibility with all CAN modules.

- The Rx FIFO allows a powerful filtering mechanism for Rx messages, with CPU interrupting.
- The local priority mechanism allows extended IDs for Tx messages.
- The hardware cancellation allows transmission to be aborted for the message buffer to be updated.

These additional features are exclusive to Freescale Semiconductor's FlexCAN module on the MPC5510 family of microcontrollers.

Appendix A
CANx Module Configuration Register (CANx_MCR)

The CANx Module Configuration Register (CANx_MCR) is mentioned numerous times throughout this application note. This is where the main controls are for enabling these new features, which allows backward compatability.

- FEN -FIFO Enable
  - — IDAM -ID Acceptance Mode.
- AEN -Abort Enable
- LPRIO_EN -Local Priority Enable

Offset: Base + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MDIS	FRZ	FEN	HALT	NOT_RDY	0	SOFT_RST	FRZ_ACK	1	0	WRN_EN	LPM_ACK	0	0	SRX_DIS	BCC
W					▨	▨		▨	▨	▨		▨	▨	▨		
Reset	1	0	0	1	1	0	0	0	1	0	0	1	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	LPRIO_EN	AEN	0	0	IDAM		0	0	MAXMB					
W	▨	▨			▨	▨			▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

Figure 6. MPC5510 CANx_MCR

Appendix B
Configuring Tx Buffers Code Example

This code example shows a basic function to configure the message buffers and includes local priority setting.

```
void CAN_Tx_start (UINT8 data, UINT8 data_inc, UINT8 length, UINT8 buf) {
UINT16 i;
UINT16 timer;
                 CAN_A.BUF[buf].DATA.W[0] = 0x00000000;
                 CAN_A.BUF[buf].DATA.W[1] = 0x00000000;// Clear Data fields
                 CAN_A.BUF[buf].CS.B.CODE = 0x8;// Hold the transmit buffer inactive
CAN_A.BUF[buf].ID.B.PRIO = can.ARX.NumMsgs>>1; // Set Local Priority
                 CAN_A.BUF[buf].ID.B.STD_ID = can.ATX.ID;  // Write standard MB IDs
                          for (i=0; i < length; i++){
                          CAN_A.BUF[buf].DATA.B[i] = data; // Write data
                          data += data_inc;
                          }
                 CAN_A.BUF[buf].CS.B.LENGTH = length;   // Write length
                 CAN_A.BUF[buf].CS.B.CODE = 0xC;// Write Code (C = Transmit)
                 timer = CAN_A.TIMER.R;              // Unlock Message buffers
}
```

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN3488
Rev. 0
06/2007