

# AN10674

## NXP LPC2000 CAN driver with FullCAN mode

Rev. 01 — 10 January 2008

Application note

### Document information

Info	Content
<b>Keywords</b>	Application Note, CAN Bus, Full CAN, LPC2000, ARM7, SJA1000
<b>Abstract</b>	This application note describes the Full CAN driver routines for the CAN controller of NXP LPC2000. Also provides the demo project developed under KEIL uVision3, using evaluation board MCB2300. This demo used UART0 for communication with PC, and print CAN transfer information with Tera term terminal.

**Revision history**

Rev	Date	Description
01	20080110	Initial version.

**Contact information**

For additional information, please visit: <http://www.nxp.com>

For sales office addresses, please send an email to: [salesaddresses@nxp.com](mailto:salesaddresses@nxp.com)

## 1. Introduction

---

The Controller Area Network (CAN) is a serial, asynchronous, multi-master communication protocol for connecting electronic control modules, sensors and actuators in automotive and industrial applications.

In the NXP LPC2000 microcontroller family, there are several microcontrollers with a CAN Controller. The newest members in the family, like the LPC2300/LPC2400/LP2900, have an improved CAN controller. This improved version is also available for the microcontrollers with revision /01.

The major improvement is the FULLCAN mode, which automatically stores received messages with selected CAN message identifiers into a message buffer.

This application note explains some of the main features of the CAN Controller including the FullCAN operation mode.

At the end of the document, a demo program is given for the LPC2300/LPC2400 microcontroller.

## 2. Main features

---

The main features for the CAN controller of the LPC2000:

- Data rates up to 1 Mbit/s on each bus

- 32-bit register and RAM access

- Compatible with CAN specifications 2.0B, ISO 11898-1

- Global Acceptance Filter recognizes 11-bit and 29-bit RX identifiers for all CAN buses

- Acceptance Filter can provide FullCAN-style automatic reception for selected Standard identifiers

### 2.1 Main features of the FullCAN operation mode

The NXP LPC2000 microcontrollers with CAN interface feature a FullCAN operation mode that directly stores received messages with selected CAN message identifiers into a message buffer.

The user provides a list of CAN message identifiers that should be received by the CAN interface. The CAN peripheral automatically scans every incoming CAN message and when an identifier match is detected, the message is copied into the associated receive buffer.

Regular FullCAN implementations typically have a limit of 16 or 32 such receive buffers, often called 'message objects'. In the LPC2000 microcontrollers, hundreds (max 146) of filters can be used. The exact number depends on multiple parameters, such as the number of CAN interfaces sharing the reception filter.

## 3. Acceptance Filter

---

The Acceptance Filter (AF) implements a fast hardware search algorithm supporting a large number of identifiers. The AF recognizes 11-bit and 29-bit identifiers. A Filter Table configures the acceptance Filter. This table is stored in a specific address in memory; its maximum size is 2048 bytes and it is shared between the CAN controllers.

The Filter Table allows configuring 6 optional sections. The sections are configured in the following order:

1. FullCAN ID's
2. Explicit 11-bit identifiers
3. Groups of 11-bit identifiers
4. Explicit 29-bit identifiers
5. Groups of 29-bit identifiers
6. FullCAN Message Object Data

When FullCAN mode is used, the FullCAN ID section defines the message ID's for automatic reception. The FullCAN Message Object Data is the area where the messages are automatically stored.

Each FullCAN Message ID entry has an interrupt enable bit, but only the first 64 FullCAN message ID's can generate an interrupt. The microcontroller generates this interrupt after storing all the message bytes in the corresponding FULLCAN Message ID Object. The receive ISR needs to read registers (FCANIC0 and FCANIC1) to identify the message ID that generated the interrupt.

In order to generate an interrupt for the explicit and group message ID sections, the RI (Receive Interrupt) flag in the CANxICR register needs to be set. The ISR needs to copy the received data bytes from the CAN controller registers to the RAM area.

The following table shows how many bytes are used to define an entry in each section. The FullCAN Message ID definition uses only 2 bytes, since the automatic reception only works for 11-bit identifiers. Please note for each FULLCAN message ID entry, 12 additional bytes need to be reserved to store the received bytes.

**Table 1. Bytes used to define an entry**

Section	Bytes
FullCAN ID	(2)
11-bit Explicit ID's	(2)
11-bit Group ID's	(4)
29-bit Explicit ID's	(4)
29-bit Group ID's	(8)
FullCAN Message Object Data	(12)

Table 2 shows five examples of different AF configurations. Example A shows that the maximum number of 11-bit identifiers defined is 1024, when no other identifiers are used. Example B shows the maximum number of FULLCAN message ID entries is 146 (14 x 146 = 2044).

**Table 2. Acceptance filter calculation examples**

# of identifiers in	Example				
	A	B	C	D	E
FullCAN ID's	0	146	0	100	50
11-bit Explicit ID's	1024	0	0	284	600
11-bit Group ID's	0	0	512	20	2
29-bit Explicit ID's	0	0	0	0	3
29-bit Group ID's	0	0	0	0	10
<b>Used Look_up Table RAM size (bytes)</b>	<b>2048</b>	<b>2044</b>	<b>2048</b>	<b>2048</b>	<b>2000</b>

### 3.1 Acceptance Filter message reception process

The basic message reception process is the following:

1. The CAN Controller is listening all messages in the CAN bus.
2. When a message is received, the CAN controller starts executing a HW fast search algorithm to try to match the received message with an entry in the Acceptance Filter. If the message is not in defined, the message is discarded. Note that at this point the core has not been interrupted; therefore it is executing the application code.
3. When there is a match, the CAN controller interrupts the microcontroller by setting bit RBx (Received message available) in register CANRxSR. The CAN ISR should copy the CAN message from the CAN Controller registers to RAM memory, and then release the CAN Controller receive registers by setting the flag RBB (Release Receive Buffer) in the Command Register (CAN1CMR).

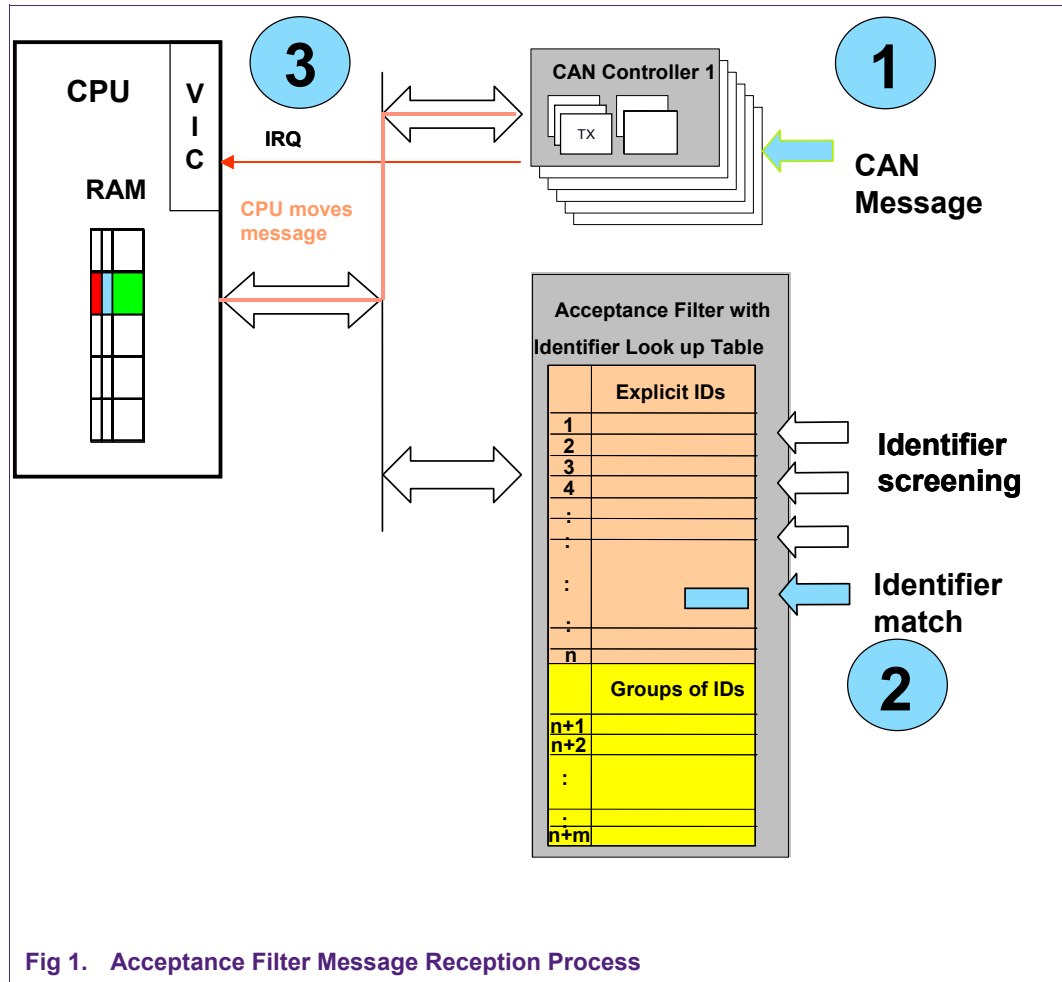


Fig 1. Acceptance Filter Message Reception Process

### 3.2 FullCAN message reception process

The FullCAN message reception process is the following:

1. The CAN Controller is listening all messages in the CAN bus.
2. When a message is received, the CAN controller starts executing a HW fast search algorithm to try to match the received message with an entry in the Acceptance Filter. If the message is not in defined, the message is discarded. If the message ID received is in the Explicit or Group of ID's, the reception process jumps to step 4. When a message ID is matched in the FULLCAN section, the reception process continues to step 3.
3. The Message Handler copies the received data into the corresponding FullCAN Message Object. Each AF entry has an index, and the Message Handler uses it to calculate the corresponding destination address:  $EndOfTable + (12 * index)$ .
4. When there is a match, the CAN generates an interrupt:
  - a. For a FullCAN message, the corresponding bit in the FCANIC0 or FCANIC1 is set. The ISR should read the received message from the FullCAN message Object section, using the same formula:  $EndOfTable + (12 * index)$  to obtain the starting address of the received message.

- b. For the explicit or group Ids, the bit RBx (Received message available) in register CANRxSR is set. The CAN ISR should copy the CAN message from the CAN Controller registers to RAM memory, and then release the CAN Controller receive registers by setting the flag RRB (Release Receive Buffer) in the Command Register (CAN1CMR).

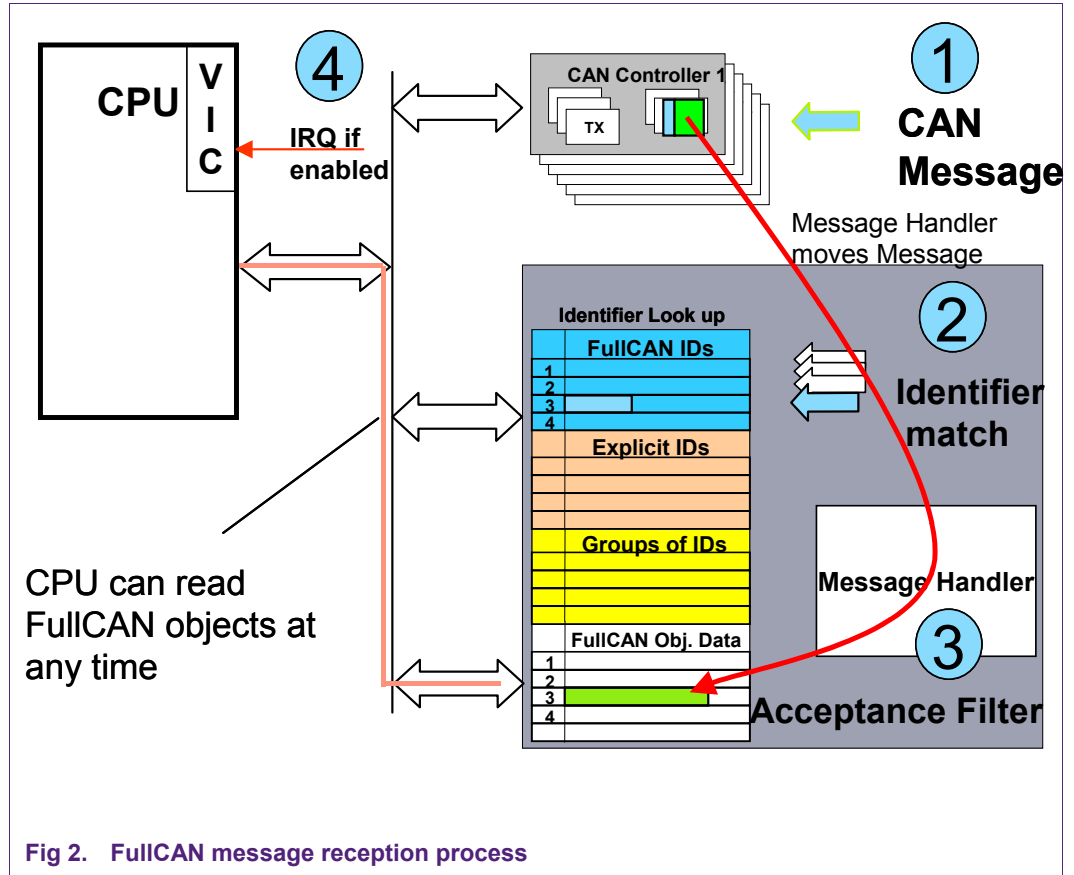


Fig 2. FullCAN message reception process

## 4. Programming the Acceptance Filter

### 4.1 CAN controller initialization

In the LPC2300/LPC2400 family each peripheral has an independent clock divider. When the Acceptance Filter is used, the clock divider used for the CAN Controller and for the Acceptance Filter should be the same.

The following bits are used to control the peripheral clock divider associated with the CAN Controller:

PCLKSEL0 27:26: PCLK\_CAN1 Peripheral clock selection for CAN1.

PCLKSEL0 29:28: PCLK\_CAN2 Peripheral clock selection for CAN2.

PCLKSEL0 31:30: PCLK\_ACF Peripheral clock selection for CAN Filtering.

The CAN Controller initialization routine should power up the CAN controller in the PCONP register. For the LPC2300, the bits 13 (CAN1) and 14 (CAN2) enable the CAN controllers.

#### 4.2 Acceptance Filter modes

The AF can be configured in four different modes by setting the Acceptance Filter Mode Register (AFMR).

The demo example program shows how to use the AF in Bypass, Enabled or FullCAN modes.

**Table 3. Acceptance filter modes**

Acceptance Filter modes	Description	AFMR Value
Acceptance Filter Disabled	All Rx messages on all CAN buses are ignored.	0x01 (Reset Value)
Acceptance Filter Bypass	All Rx messages are accepted on enabled CAN controllers	0x03
Acceptance Filter Enabled	Only the message ID's defined in the Filter Table produce an interrupt.	0x00
AF enabled and FullCAN mode enabled	The microcontroller is interrupted when a message ID defined in any of the explicit or group of ID sections is received. The message is stored automatically if the message received is defined in the FullCAN ID section.	0x04

#### 4.3 Acceptance Filter and FullCAN mode enabled

There are five pointers associated with the AF:

- Standard Frame Individual Start Address Register (SFF\_sa)
- Standard Frame Group Start Address Register (SFF\_GRP\_sa)
- Extended Frame Start Address Register (EFF\_sa)
- Extended Frame Group Start Address Register (EFF\_GRP\_sa)
- End of AF Tables Register (ENDofTable)

Each of the pointers is used to define the start of a section. They are relative (offset) pointers to the starting address of the Filter Table. When a section is not defined, the register should be set to the current free entry in the Filter Table. It is assumed that for the FULLCAN message ID section the starting address (offset) is always 0.

The general procedure to configure the Acceptance Filter with FullCAN mode enabled is the following:

1. Set the CAN Controller Mode Register ((CANxMOD) to Operating Mode (0x00).
2. Store in ascending order the FullCAN message ID's. Each entry is a 16-bit value, the FullCAN entries have 2 control bits: Message disable bit and the interrupt enable bit. The SCC field specifies the Source CAN Channel.



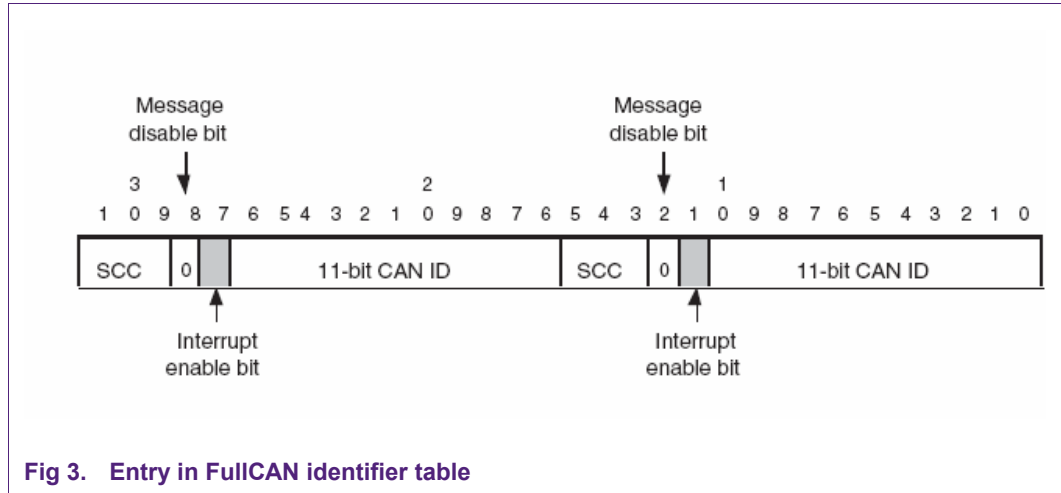


Fig 3. Entry in FullCAN identifier table

3. Set “Standard Frame Individual Start Address Register” (SFF\_sa) to the next free entry after the last FullCAN message ID entry. If SFF\_sa is set to zero, no FullCAN message ID’s are defined.
4. Configure in ascending order the explicit 11-bit identifiers. Each entry is 16 bits, including a Message disable bit.
5. Set “Standard Frame Group Start Address Register” (SFF\_GRP\_SA) to the next free Filter Table entry.
6. Configure in ascending order the 11-bit groups. The lower bound is defined first and then the upper bound. Each group is defined using 32 bits. There is Disable bit for each bound. If a group is disabled, the Upper and Lower bound disable bits should be set.

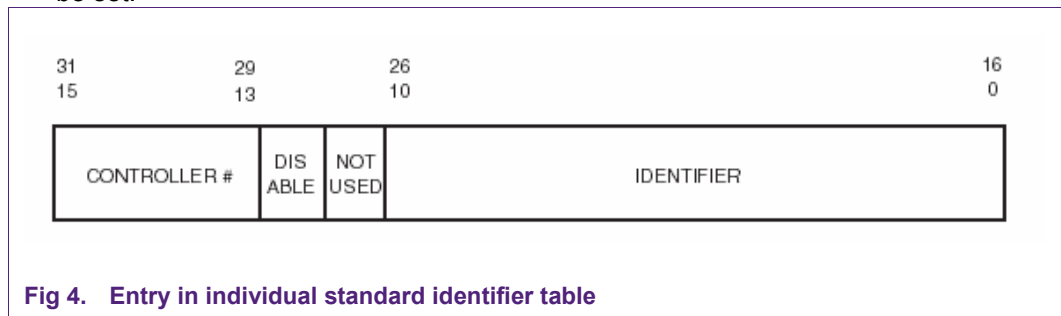


Fig 4. Entry in individual standard identifier table

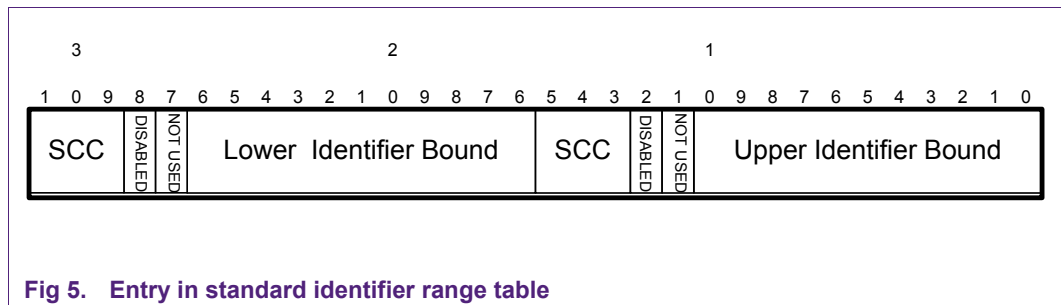


Fig 5. Entry in standard identifier range table

7. Set “Extended Frame Individual Start Address Register” (EFF\_sa) to the next free Filter Table entry.
8. Store in ascending order the 29-bit identifiers. Each entry is 32 bits.

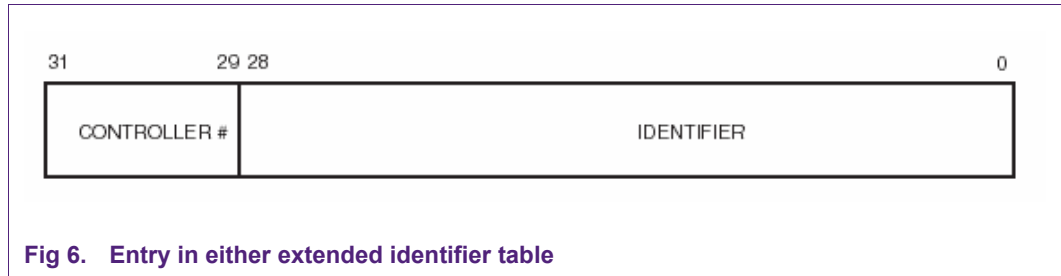


Fig 6. Entry in either extended identifier table

9. Set “Extended Frame Group Start Address Register” (EFF\_GRP\_sa) to the next free Filter Table entry.
10. Store in ascending order the 29-bit groups. The lower bound is defined first and then the upper bound. Each group entry is 64 bits.

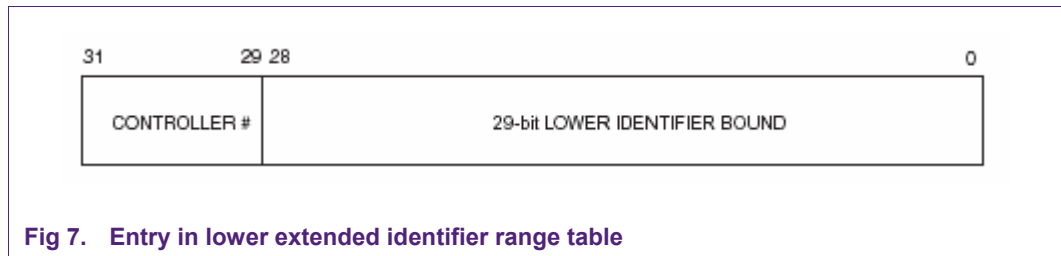


Fig 7. Entry in lower extended identifier range table

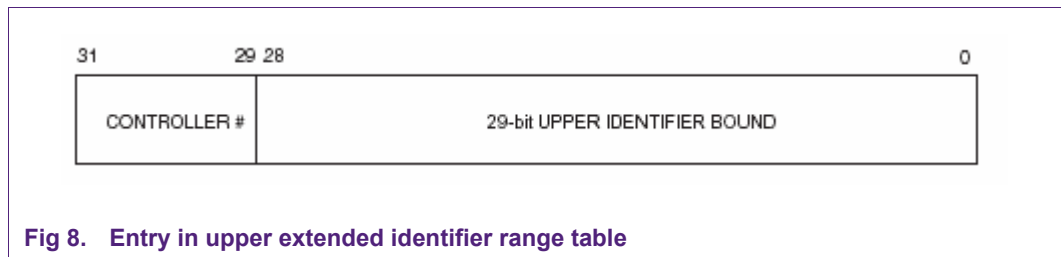


Fig 8. Entry in upper extended identifier range table

11. Set the “End of AF Tables Register” (ENDofTable) to the next free Filter Table entry.
12. Set the CAN Acceptance Filter Mode Register to “AF with FULLCAN mode enabled”, CAN\_AFMR = 0x00000004.

4.3.1 Example

The following figure shows the configuration of the filter table when the following filters are defined:

1. FullCAN ID's: 0x20,0x1BC,0x255,0x26F
2. Explicit 11-bit ID: 0x10,0x1AC,0x245,0x25F
3. 11-bit groups: 0x300-0x3FF, 0x400-0x47F
4. Explicit 29 bit ID: 0x18EF101E, 0x18EF1E10,0x18EFFF10,0x18FFFC2
5. 29-bit groups: 0x7700-0x77FF, 0x85F7-0x8802

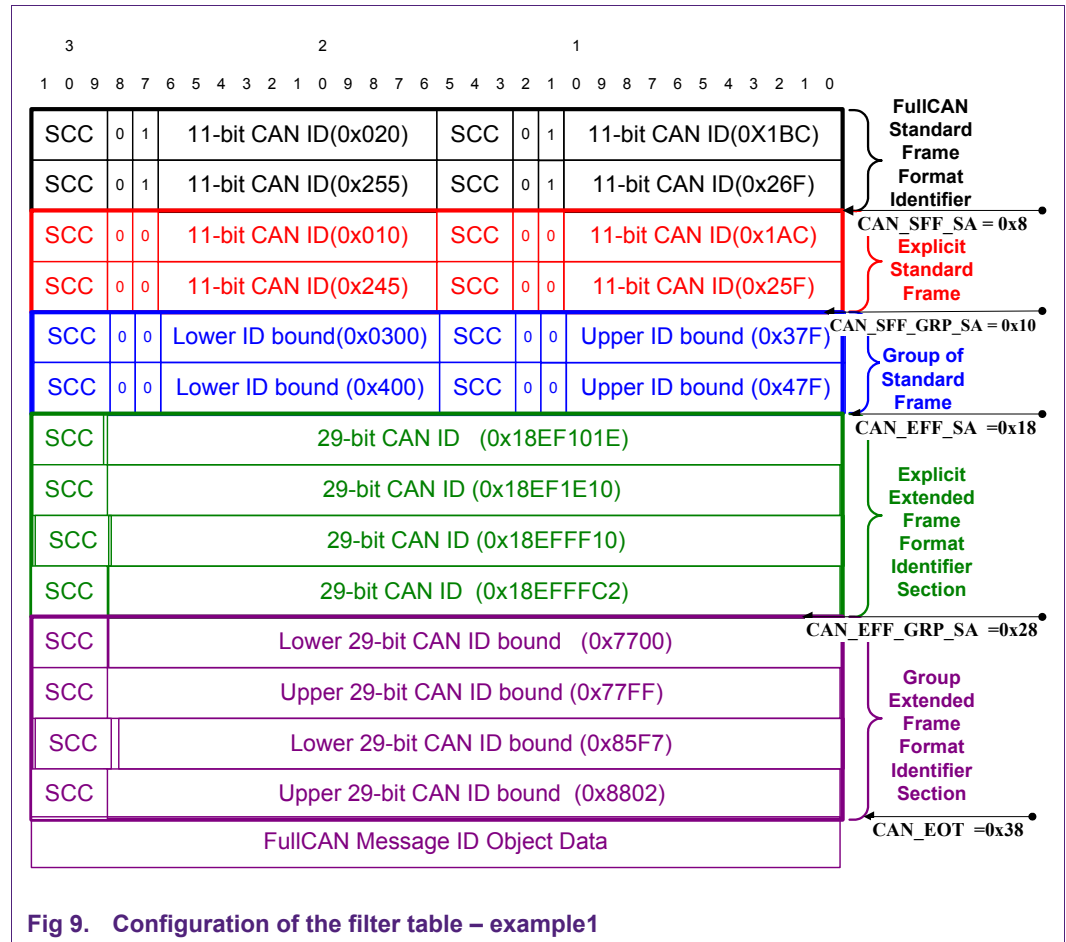
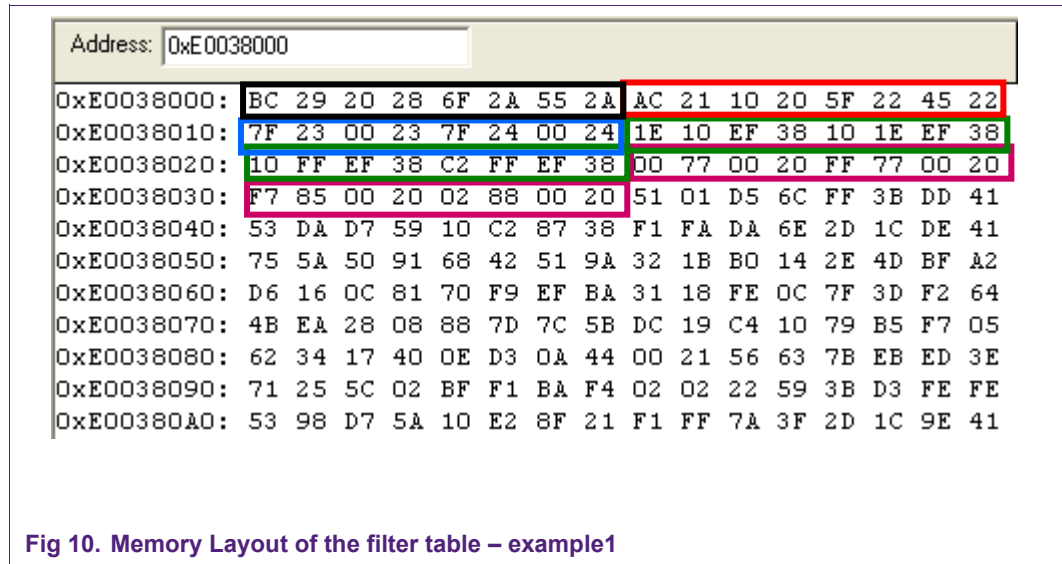


Figure 10 shows the Memory layout for the previous example:



### 4.3.2 Another example

Below is another example of the Acceptance Filter configuration. In this example, there are four CAN controllers (the LPC23xx/LPC24xx have only two CAN channels, but NXP also has four CAN channels microcontrollers, e.g. LPC2294) and each one defines the following entries:

1. CAN Controller 1

Standard 11-bit identifiers: 0x50, 0x51, 0x101, 0x110, 0x120, 0x130, 0x132, 0x134, 0x200, 0x203, 0x208, 0x300, 0x601, 0x602; Group of extended 29-bit identifiers: 0x8000-0x8020

2. CAN Controller 2

Standard 11-bit identifiers: 0x30, 0x32, 0x47, 0x59, 0x222, 0x237, 0x340, 0x352; Group of extended 29-bit identifiers: 0x4000-0x4020.

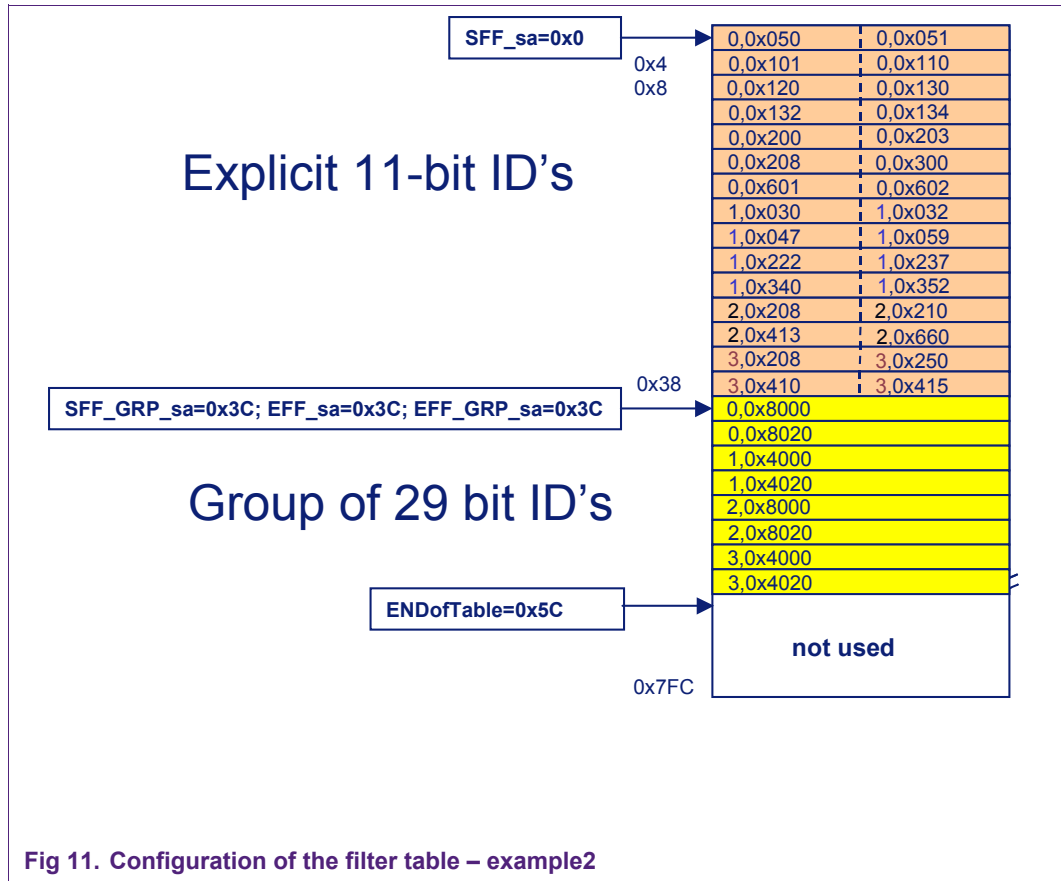
3. CAN Controller 3

Standard 11-bit identifiers: 0x208, 0x210, 0x413, 0x660; Group of extended 29-bit identifiers: 0x8000-0x8020

4. CAN Controller 4

Standard 11-bit identifiers: 0x208, 0x250, 0x410, 0x415; Group of extended 29-bit identifiers: 0x8000-0x8020

In the latest CAN Controller (LPC2300/LPC2400/LPC2900, revisions '/01'), the SCC value equals CAN\_controller – 1, e.g. SCC=0 matches CAN1.



#### 4.4 Filter Table programming guidelines

1. The entries should be stored in increasing order for the Source CAN Channel (SCC) first, and then for the CAN identifier (including disabled identifiers)
2. If there is an odd-number of identifiers, an extra ID should be added with the disable bit set. This is because the next section should start in a word boundary
3. To disable a group entry in the AF, both entries need to be disabled.

### 5. Reading FullCAN Message Objects

A message object is reserved for each of the FullCAN message ID entries. Each of the entries is assigned an index. The CAN controller uses this index to automatically store the FullCAN message in memory. When the application code wants to read the FullCAN Message Object, it uses the formula: EndOfTable + (12 \* index) to calculate the Message ID Object starting address. Each of the message objects use 3 32-bits words:

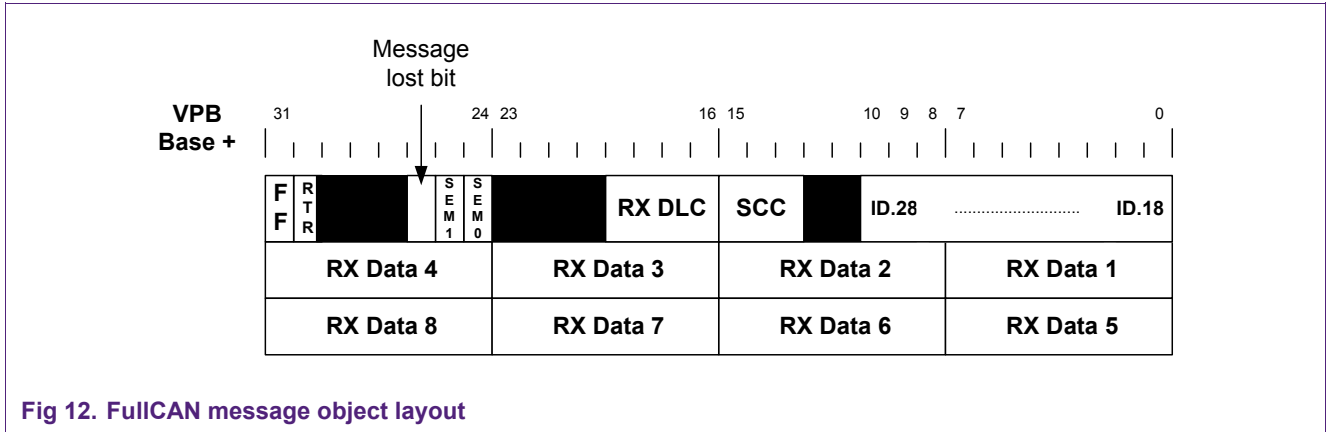


Fig 12. FullCAN message object layout

The Message Lost bit (MsgLst) indicates whether more than one FullCAN message has been received since last time this message object was read. The Source CAN Channel (SCC) identifies the CAN channel that received the message.

The CAN controller and the user application can access the FullCAN Message Objects. In order to avoid corrupting the data in the message object or reading incorrect data bytes, there is a semaphore mechanism implemented in each of the FullCAN Message Objects.

Table 4. FullCAN semaphore operation

SEM1	SEM0	Activity
0	1	Acceptance Filter is updating the content
1	1	Acceptance Filter has finished updating the content
0	0	CPU is in process of reading from the Acceptance Filter

The user application code should implement the following procedure to access the Message Object:

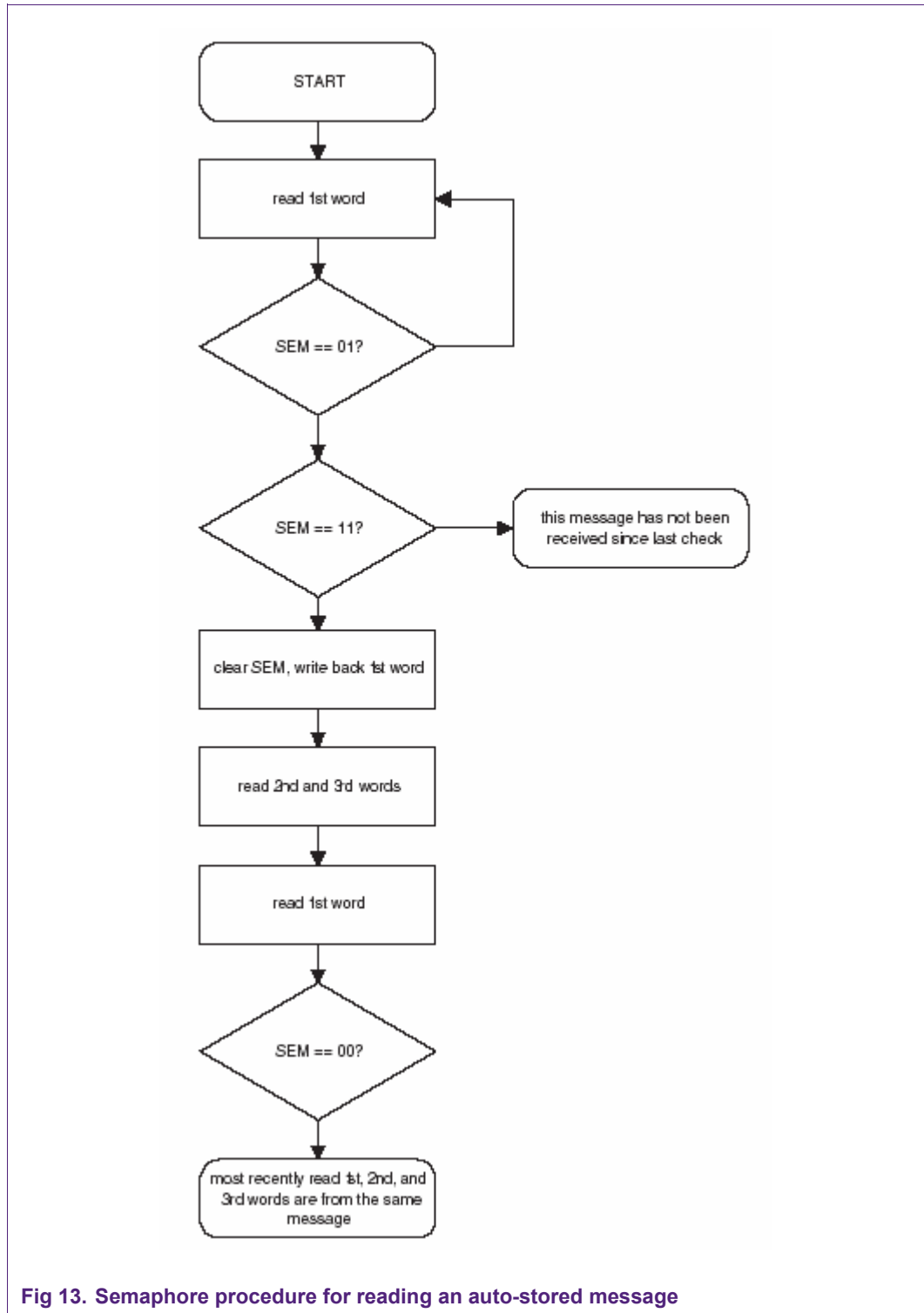
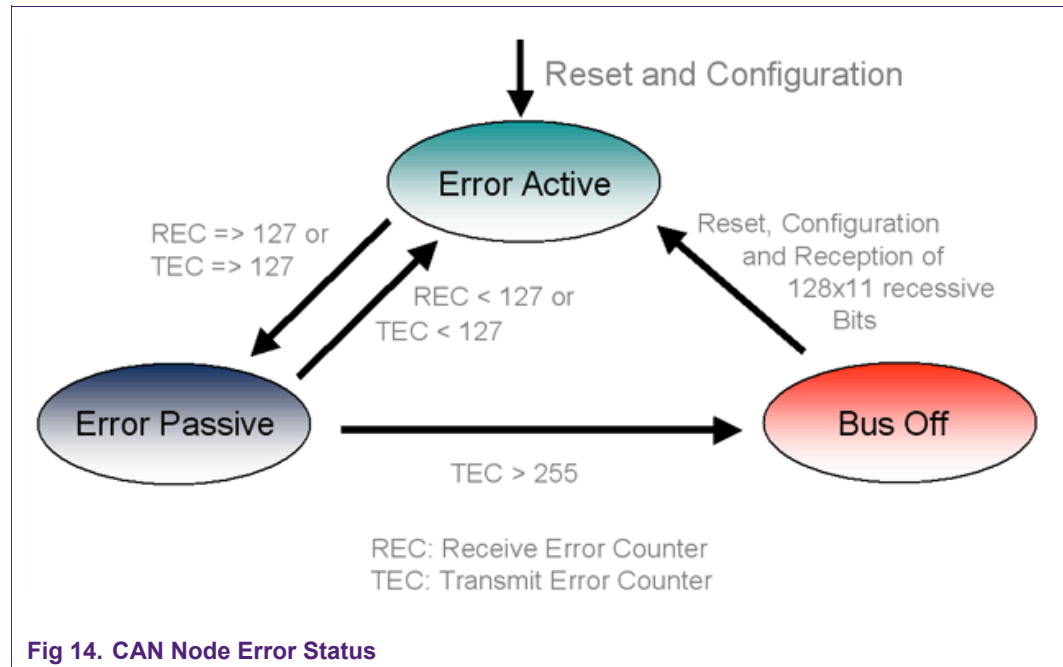


Fig 13. Semaphore procedure for reading an auto-stored message

## 6. CAN error management

The CAN Controllers count and handle the transmit and receive errors as specified in CAN Spec 2.0B. The Transmit and Receive Error Counters are incremented for each

detected error and are decremented when operation is error-free. Figures 16 and 17 show how the incrementing/decrementing is done depending on each type of error.



If the Transmit Error counter contains 255 and another error occurs, the CAN Controller is forced into a state called Bus-Off. In this state, the following register bits and register values are set:

- BS (Bus Off Set) in CANxSR
- BEI (Bus Error Interrupt) and EI (Error Warning Interrupt) in CANxICR
- RM (Reset Mode) in CANxMOD
- Transmit Error Counter is set to 127
- Receive Error Counter is cleared

The application code is responsible for clearing the RM bit to exit from the reset mode. The CAN controller won't be able to communicate until 128 occurrences of the Bus Free condition (11 consecutive recessive bits) have occurred.

## 6.1 Error Interrupts

Three interrupt sources have been implemented to signal error conditions. In the CAN protocol, the user application doesn't have to worry about manually retransmitting the message, the CAN controller does this automatically. The following three interrupts could be used as statistical information or to find a faulty node in the CAN network.

Each interrupt can be enabled separately in the Interrupt Enable Register (CANxIER)

**Bus Error Interrupt: (bit BEIE):** This interrupt is generated upon any error condition on the CAN bus.

**Error Warning Interrupt: (bit EIE):** The Error Warning Interrupt is generated if the error warning limit is passed. Furthermore it is generated if the CAN controller enters the bus-



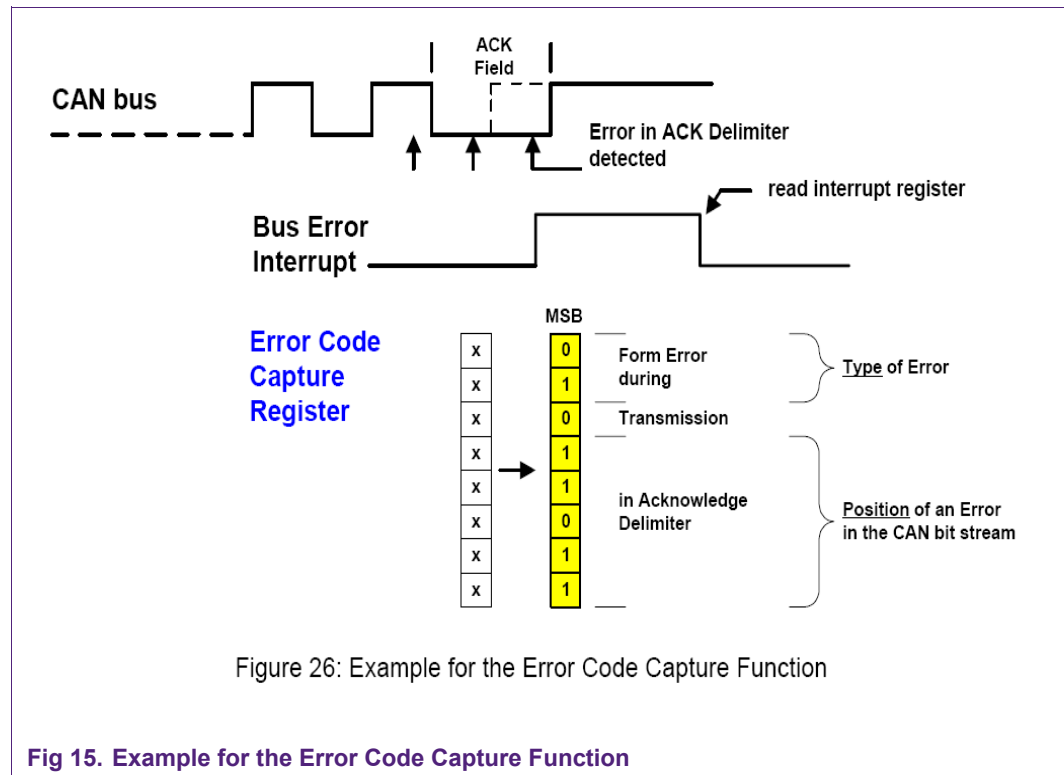
off state and upon re-entry into error active state. The error-warning limit of the Can Controller is programmable in reset mode. The default value upon reset is 96.

Error Passive Interrupt: (EPI) If the error status changes from error active to error passive or vice versa an error passive interrupt is set.

### 6.2 Error Code Capture

The LPC2000 CAN Controller performs the full error confinement specified in the CAN2.0B specifications. Like in every CAN controller, the whole process of handling errors is executed fully automatically. However, to provide the user with additional details about a certain error condition the CAN Controller contains the Error Code Capture function.

Whenever a CAN bus error occurs, the corresponding bus error interrupt is set. At the same time, the current bit position is captured into the Interrupt and Capture Register. The captured data is kept until the host controller has read it. Afterwards, the capture mechanism is activated again. The register content distinguishes four different types of errors: form, stuff, bit and other errors. The register additionally indicates whether the error occurred during reception or transmission of a message. Five bits in this register indicate the erroneous bit position in the CAN frame.



The next two figures show all possible errors during transmission and reception of CAN messages. The left part contains the position and the type of an error, captured by the Error Code Capture Register. The right part of each table is a translation into an upper

level error description and can be derived directly from the register contents. With the help of these tables, further information concerning error counter change and the erroneous state at the transmit and receive pins of the device can be derived. While using this table, e.g., in the error analysis software it is possible to analyze every single error situation in detail. The information about type and position of CAN errors can be used for error statistics and system maintenance or for corrective actions during system optimization.

Error Code Capture		RX Error Count	Description
Position of an Error in the CAN bit stream	Type of Error		
Identifier SRR, IDE and RTR bit Reserved Bits Data Length Code Data Field CRC Sequence	Stuff	+ 1	more than 5 consecutive bits with same level received
CRC Delimiter	Form Stuff	+ 1 + 1	Rx = dominant more than 5 consecutive bits with same level received
Acknowledge Slot	Bit	+ 1	Tx = dominant but Rx = recessive
Acknowledge Delimiter <sup>1</sup>	Form	+ 1	Rx = dominant or CRC error detected <sup>1</sup>
End of Frame	Form Other	+ 1 ± 0	Rx = dominant in first 6 bits Rx = dominant in last bit
Intermission	Other	± 0	Rx = dominant
Active Error Flag	Bit	+ 8	Tx = dominant but Rx = recessive
Tolerate Dominant Bits	Other	+ 8	Rx = dominant in first bit upon error flag Rx = dominant for more than 7 bits upon error or overload flag
Error Delimiter	Form Other	+ 1 ± 0	Rx = dominant within first 7 bits Rx = dominant in last bit of delimiter
Overload Flag	Bit	+ 8	Tx = dominant but Rx = recessive

Fig 16. Possible errors during reception

Error Code Capture			
Position of an Error in the CAN bit stream	Type of Error	TX Error Count	Description
Start Of Frame	Bit	+ 8	Tx = dominant but Rx = recessive can't write dominant bit
Identifier	Bit Stuff	+ 8 ± 0	Tx = dominant but Rx = recessive can't write dominant bit - - Tx = recessive but Rx = dominant
SRR Bit	Bit Stuff	+ 8 ± 0	Tx = dominant but Rx = recessive can't write dominant bit - - Tx = recessive but Rx = dominant
IDE and RTR Bit	Bit Stuff	+ 8 + 8	Tx = dominant but Rx = recessive can't write dominant bit - - Tx = recessive but Rx = dominant
Reserved Bits, Data Length Code, Data Field, CRC Sequence,	Bit	+ 8	Tx = dominant but Rx = recessive can't write dominant bit
CRC Delimiter	Form	+ 8	Rx = dominant bit has to be recessive
Acknowledge Slot	Other Other	+ 8 ± 0	Rx = recessive (error active) no acknowledge Rx = recessive (error passive) no acknowledge, node is probably alone on the bus
Acknowledge Delimiter	Form	+ 8	Rx = dominant critical bus timing or bus length
End of Frame	Form Other	+ 8 + 8	Rx = dominant within first 6 bits - - + 8 Rx = dominant in last bit frame has already been received by some nodes, re-transmission may result in data duplication in receivers
Intermission	Other	± 0	Rx = dominant overload flag from 'old' CAN controllers
Active Error Flag Overload Flag	Bit	+ 8	Tx = dominant but Rx = recessive can't write dominant bit
Tolerate Dominant Bits	Form	+ 8	Rx = dominant for more than 7 bit times after active error flag or overload flag - -
Error Delimiter	Form Other	+ 8 ± 0	Rx = dominant within first 7 bits - - ± 0 Rx = dominant in last bit of delimiter overload flag from 'old' CAN controller
Passive Error Flag	Other	+ 8	Rx = dominant (error passive) no acknowledge received, node is not alone on the bus

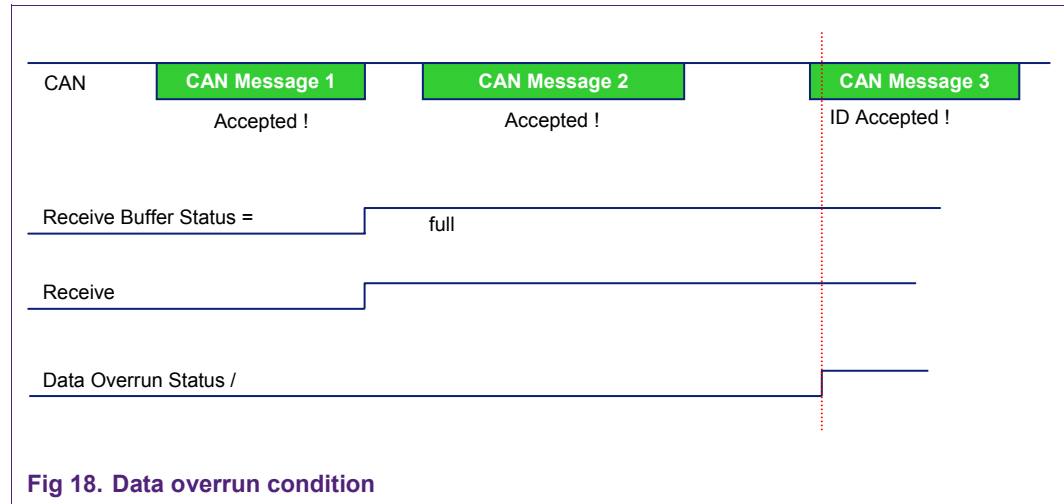
Fig 17. Possible errors during transmission

### 6.3 Receive overrun condition

Each CAN controller has a double receive buffer. Only one receive buffer is accessible by the user application. When a message is completely received, an interrupt is

generated. While the user application is accessing the buffer, the CAN controller is still able to receive another message. The overrun condition occurs when a third message is received, but the application code has not released the receive buffers.

Figure 18 illustrates this condition:



To avoid the overrun condition, the user should configure the AF in such way that most of non-important messages are filtered out. Also the user could configure the CAN controller interrupt as a FIQ (Fast interrupt Request), so the interrupt will have the highest priority in the system.

## 7. General CAN controller driver description

### 7.1 Software development support

The CAN controller driver was tested with the following software development environments:

- IDE – KEIL uVision3 (MDK3.03 or MDK3.11)
- Emulator – ULINK (used for MDK3.03) or ULINK2 (used for MDK3.11)
- Keil Realview 3.5x compiler

The CAN driver example was based in the AN10438: Application Note “NXP LPC2000 CAN driver”. The microcontroller used in this example was the LPC2378.

### 7.2 CAN controller driver functionality overview

The CAN controller driver routines described in this Application Note are designed to provide the application programmer with a higher-level interface for communication with each CAN controller module, thus relieving the programmer from having to understand the detailed operation of the CAN controller module. The application programmer can design the application interface to the CAN controller driver routines with the knowledge that the driver routines will take all necessary actions for transmitting and receiving CAN messages on the CAN bus.

The CAN controller driver routines perform the following functions:

- Initialization of the CAN controller
- Configuration of the CAN controller for various baud rates
- Prepare transmission of data pool and CAN message
- Receiving and storing CAN messages in the appropriate receive data pool
- Providing pre-defined values for a set of bit-rates for various clock frequencies
- Mode switching of the CAN controller
- Easy read/write access to CAN controller registers
- Printing information about CAN BUS transmitted/received data by the UART.

The Global Acceptance Filter (GAF) configuration lists the initialization of five Look-up table sections.

For the CAN controller driver the file LPC2000\_CAN\_Driver.c contains the five-identifier sections that can be configured.

Note: Commenting the according definition statements out can disable those sections. This should be done if some sections are not used.

Furthermore, the LPC2000\_CAN\_Driver.c contains lists of example CAN identifiers with their associated Source CAN Channel (SCC) separated for each section. The following tables, Table 6 to Table 10, are extracted from the LPC2000\_CAN\_Driver.c file and show all pre-defined CAN identifiers and their SCC for all sections. The user can change the list in each section to suit each application's needs.

**Table 5. ID look-up table definitions of the LPC2000\_CAN\_Driver.c**

Section Definition	Section of the ID Lookup Table Memory
#define LPC2000_CANDRIVER_STD_FullCAN	FullCAN Frame Format ID Section
#define LPC2000_CANDRIVER_STD_INDIVIDUAL	Explicit Standard Frame Format ID Section
#define LPC2000_CANDRIVER_STD_GROUP	Group of Standard Frame Format ID Section
#define LPC2000_CANDRIVER_EXT_INDIVIDUAL	Explicit Extended Frame Format ID Section
#define LPC2000_CANDRIVER_EXT_GROUP	Group of Extended Frame Format ID Section

**Table 6. Example of FullCAN frame format identifier section**

```
const lpc2000CANdriver_ACFilter_t gklpc2000CANdriver_StdFullCAN_Section[] =
{
    /* Channel(1-4) ,          11-bit Identifier ( <7FF Double than ram size ) */
    {LPC2000_CANDRIVER_SCC_2, 0x0020},
    {LPC2000_CANDRIVER_SCC_2, 0x01BC},
    {LPC2000_CANDRIVER_SCC_2, 0x0255},
}
```

```
{LPC2000_CANDRIVER_SCC_2, 0x026F}
};
```

**Table 7. Example of explicit standard frame format identifier section**

```
const lpc2000CANdriver_ACFilter_t gklpc2000CANdriver_StdIndividualSection[ ] =
{
/* Channel(1-4) , 11-bit Identifier */
{LPC2000_CANDRIVER_SCC_2, 0x0010},
{LPC2000_CANDRIVER_SCC_2, 0x01AC},
{LPC2000_CANDRIVER_SCC_2, 0x0245},
{LPC2000_CANDRIVER_SCC_2, 0x025F}
};
```

**Table 8. Example group of standard frame format identifier section**

```
const lpc2000CANdriver_ACFilter_t gklpc2000CANdriver_StdGroupSection[ ] =
{
/* Channel 11-bit Identifier */
{LPC2000_CANDRIVER_SCC_2, 0x0300}, // lower bound, Group 1
{LPC2000_CANDRIVER_SCC_2, 0x037F}, // upper bound, Group 1
{LPC2000_CANDRIVER_SCC_2, 0x0400}, // lower bound, Group 2
{LPC2000_CANDRIVER_SCC_2, 0x047F} // upper bound, Group 1
};
```

**Table 9. Example of explicit extended frame format identifier section**

```
const lpc2000CANdriver_ACFilterx_t gklpc2000CANdriver_ExtIndividualSection[] =
{
/* Channel          29-bit Identifier ( =< 0x1FFFFFFF) */
{LPC2000_CANDRIVER_SCC_2, 0x18EF101E},
{LPC2000_CANDRIVER_SCC_2, 0x18EF1E10},
{LPC2000_CANDRIVER_SCC_2, 0x18EFFF10},
{LPC2000_CANDRIVER_SCC_2, 0x18EFFF2}
};
```

**Table 10. Example group of extended frame format identifier section**

```
const lpc2000CANdriver_ACFilterx_t gklpc2000CANdriver_ExtGroupSection[ ] =
{
/* Channel 29-bit Identifier ( =< 0x1FFFFFFF) */
{LPC2000_CANDRIVER_SCC_2, 0x00007700}, // lower bound, Group 1
{LPC2000_CANDRIVER_SCC_2, 0x000077FF}, // upper bound, Group 1
{LPC2000_CANDRIVER_SCC_2, 0x000085F7}, // lower bound, Group 2
{LPC2000_CANDRIVER_SCC_2, 0x00008802} // upper bound, Group 2
};
```

### 7.3 CAN message buffers data structure

For transmission and reception of a CAN message, Message Buffers for transmitting and receiving messages are used. Both buffers are defined as a structure and hold the following data fields (see Table 11). Both structures are defined in the header file LPC2000\_CAN.h.

**Table 11. Transmit and receive Message Buffer defined in LPC2000\_CAN.h**

Transmit buffer: <code>lpc2000CANdriver_TXObj_t</code>	Receive buffer: <code>lpc2000CANdriver_RXObj_t</code>
<pre>typedef struct {     UInt32  TFI;     UInt32  ID;     UInt32  DataField[2]; } lpc2000CANdriver_TXObj_t;</pre>	<pre>typedef struct {     unsigned char FULLCALmsg;     UInt32  RFS;     UInt32  ID;     UInt32  DataField[2]; } lpc2000CANdriver_RXObj_t;</pre>

Table 12 shows how the 32-bit wide array Data Field and the CAN Message Buffer data is organized.

**Table 12. Array Data Field**

	MSB		LSB	
DataField[0]	Data Byte 4	Data Byte 3	Data Byte 2	Data Byte 1
DataField[1]	Data Byte 8	Data Byte 7	Data Byte 6	Data Byte 5

## 8. Demo description

### 8.1 Demo hardware

Using KEIL MCB2300 Evaluation Board.

You can find detailed information on the MCB2300 board at:

<http://www.nxp.com/redirect/keil.com>

### 8.2 Demo setup

#### 8.2.1 Introduction

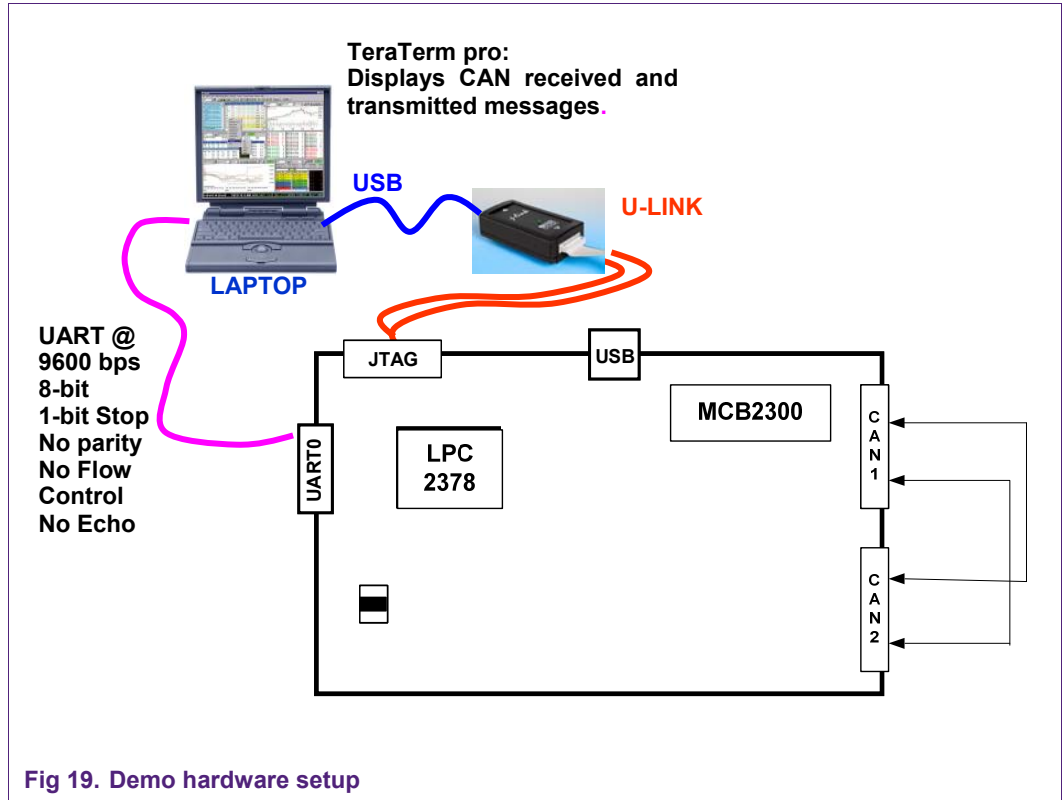
LPC2378 microcontroller has 2 CAN channels.

In the demo, the two CAN channels were interconnected, in such way that:

CAN1 transmits data to the CAN BUS, the UART0 displays the messages transmitted and the Global Acceptance Filter Look-up Table to PC screen.

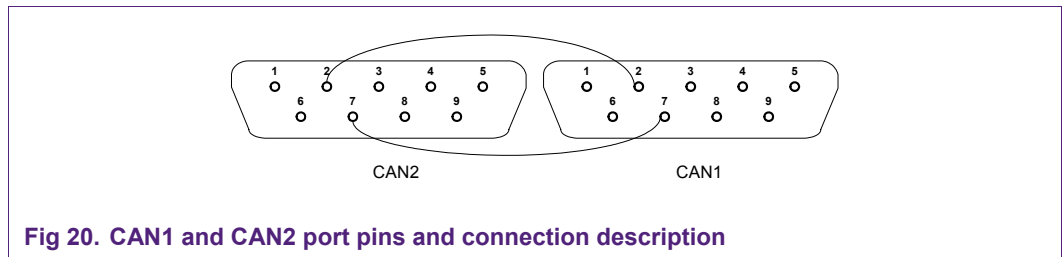
CAN2 receives data from the CAN BUS, the UART0 displays the messages received.

8.2.2 Hardware setup



8.2.3 Demo operation step

1. Connect the two Pin-2 between CAN1 port and CAN2 port; Connect the two Pin-7 between CAN1 port and CAN2 port; Or you can find one UART cable with both female header and pin-to-pin connected



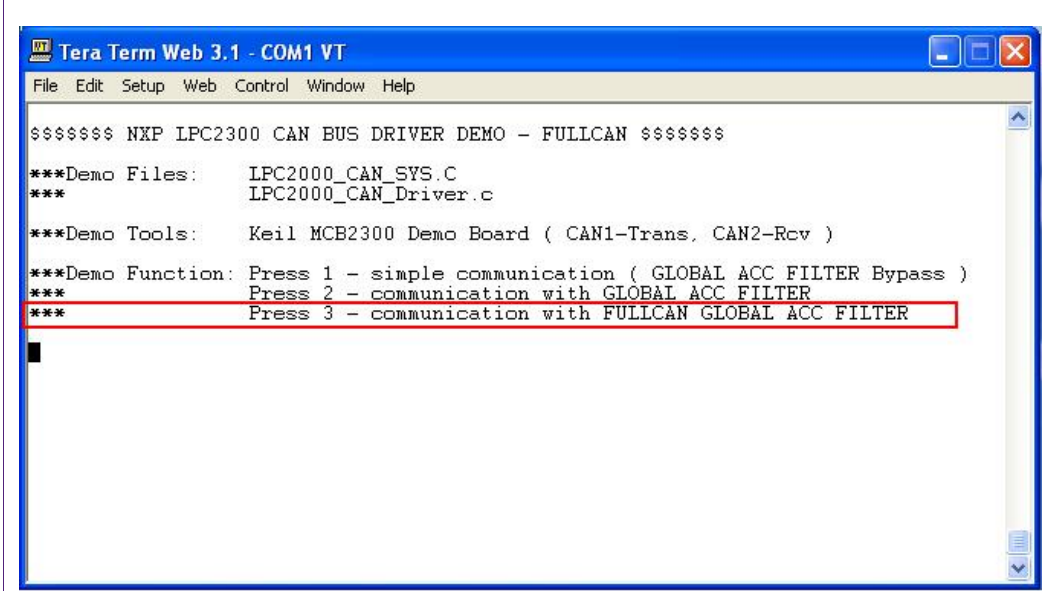
2. Make sure MCB2300 RST (J9) jumper is off and ISP (J10) jumper is OFF
3. Connect UART0 of MCB2300 to PC UART port
4. Connect MCB2300 JTAG port to PC USB port with U-LINK
5. Open and download the demo project[LPC2000\_CAN.Uv2]
6. Open Tera Term Pro in your PC



7. Power On. **Press Reset key** on MCB2300, it will enable ULINK in uVision3
8. Enter debug mode in uVision3
9. Run the demo

### 8.3 Demo displayed information

When you open the demo project and start debug in uVision3, on Tera Term Terminal, it will displays following demo start information:



```

Tera Term Web 3.1 - COM1 VT
File Edit Setup Web Control Window Help
$$$$$$$ NXP LPC2300 CAN BUS DRIVER DEMO - FULLCAN $$$$$$$
***Demo Files:   LPC2000_CAN_SYS.C
***              LPC2000_CAN_Driver.c
***Demo Tools:   Keil MCB2300 Demo Board ( CAN1-Trans, CAN2-Rcv )
***Demo Function: Press 1 - simple communication ( GLOBAL ACC FILTER Bypass )
***              Press 2 - communication with GLOBAL ACC FILTER
***              Press 3 - communication with FULLCAN GLOBAL ACC FILTER
█

```

Fig 21. Welcome page for the Demo

When you press 3, CAN message send page will come, not only displays sent messages, but also displays Global Acceptance Filter Look-up table pre-defined in LPC2000\_CAN\_Driver.c

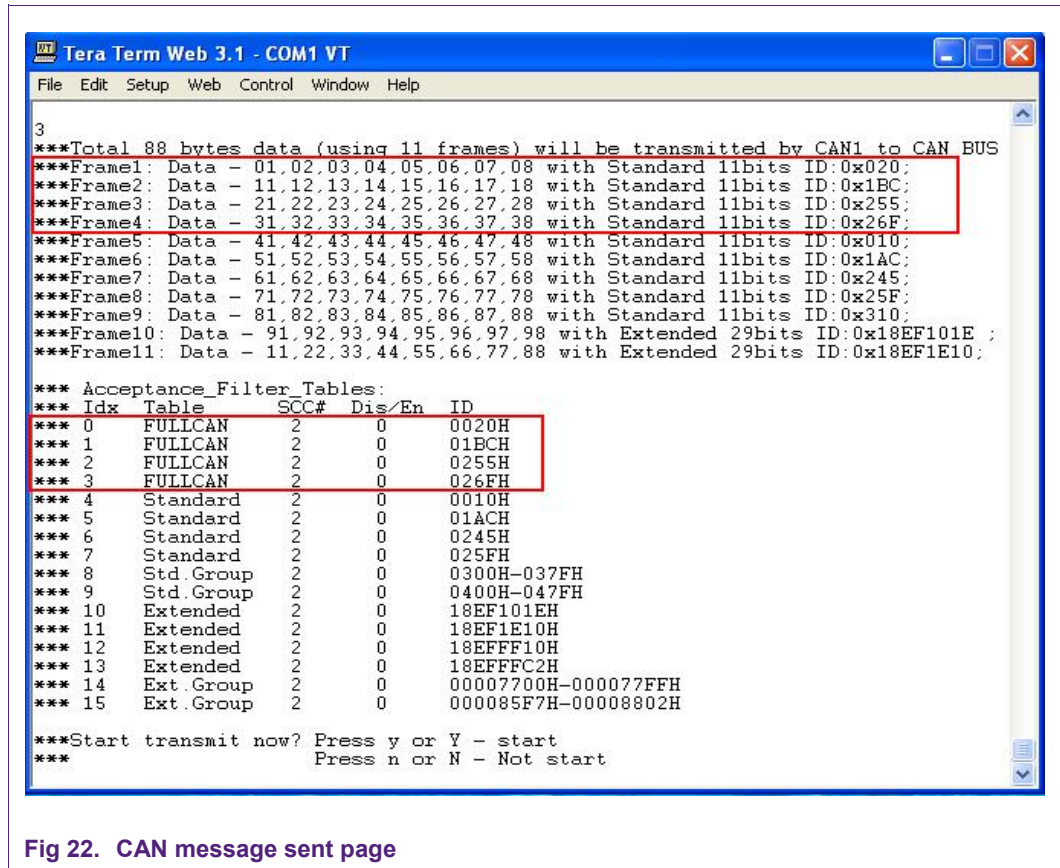


Fig 22. CAN message sent page

When you conform transmission, Demo code will start transmit.

If received successfully, then demo code will display the received data as following:

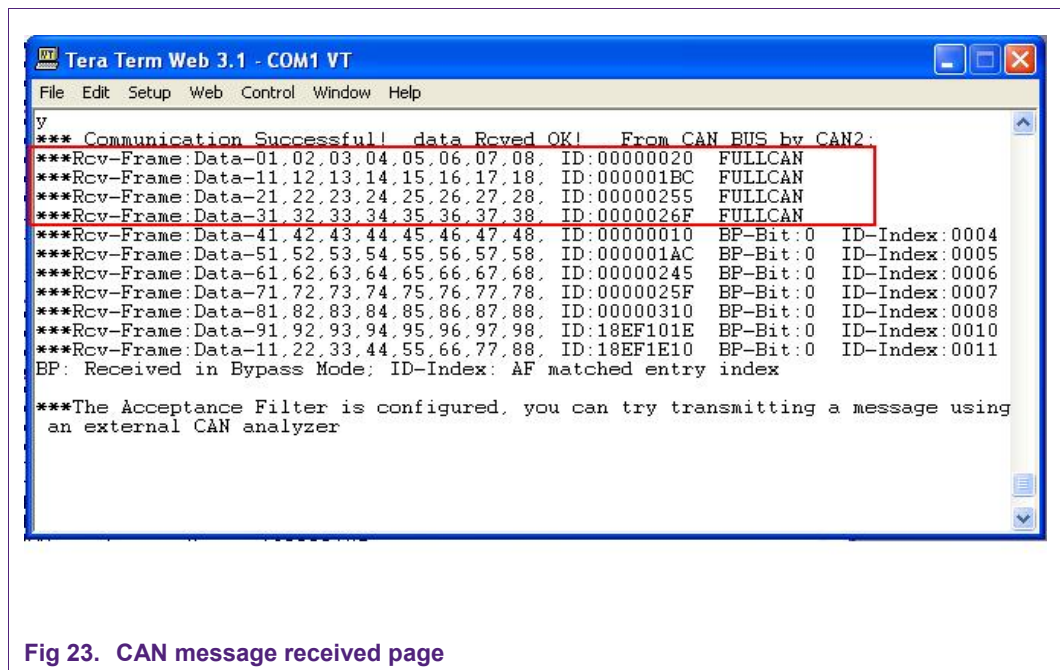


Fig 23. CAN message received page

## 9. Reference

---

- [1] AN10438: Application Note “NXP LPC2000 CAN driver”
- [2] LPC2378 Data Sheet – Version 3
- [3] UM10211: LPC2364/65/66/67/68/77/78/87/88 User Manual – Version 1

10. Appendix A Software flowchart for demo

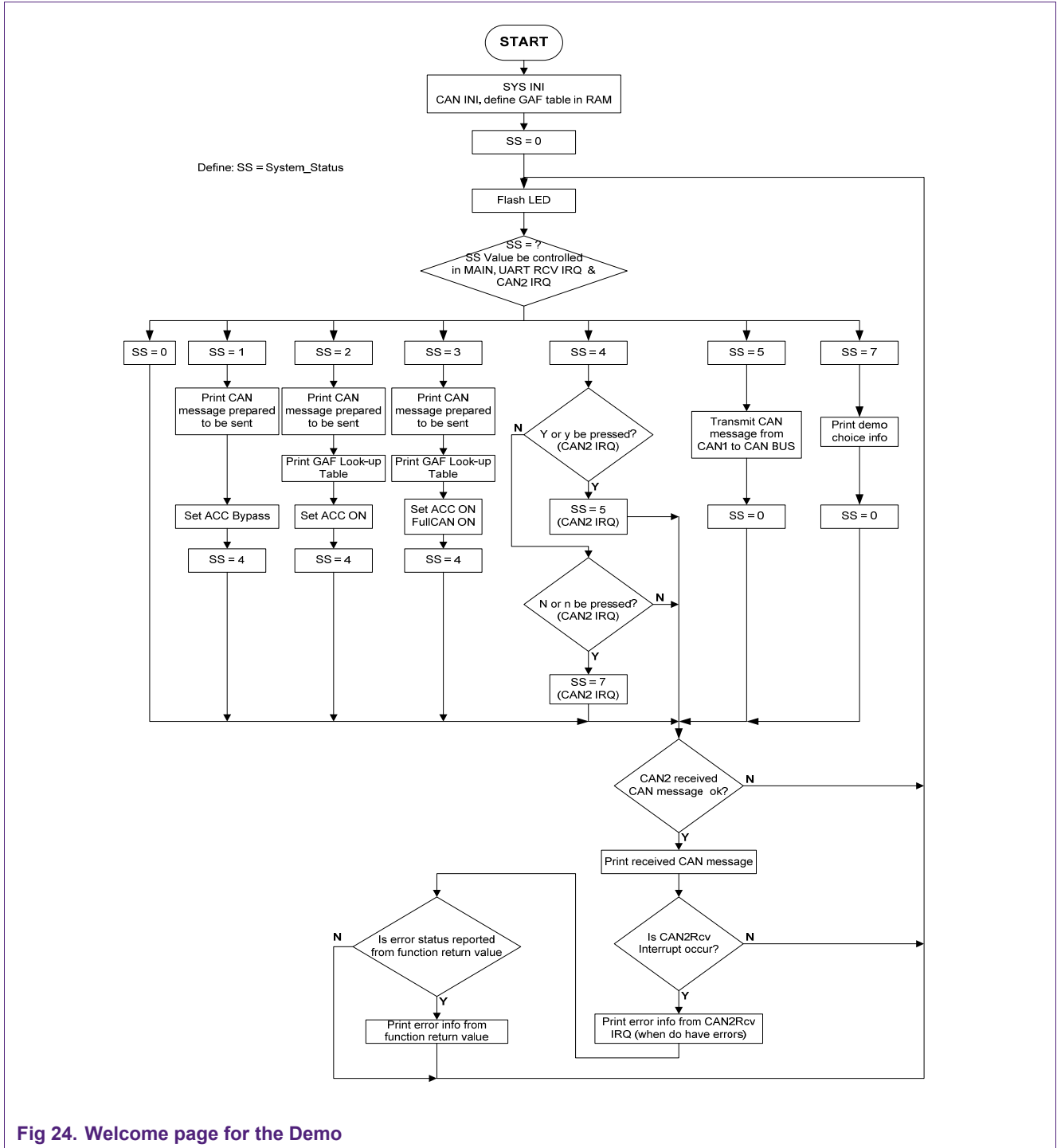


Fig 24. Welcome page for the Demo

## 11. Legal information

### 11.1 Definitions

**Draft** — The document is a draft version only. The content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

### 11.2 Disclaimers

**General** — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information.

**Right to make changes** — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

**Suitability for use** — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in medical, military, aircraft, space or life support equipment, nor in applications where failure or malfunction of a NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors accepts no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is for the customer's own risk.

**Applications** — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

### 11.3 Trademarks

Notice: All referenced brands, product names, service names and trademarks are property of their respective owners.

## 12. Contents

---

<b>1.</b>	<b>Introduction .....</b>	<b>3</b>
<b>2.</b>	<b>Main features .....</b>	<b>3</b>
2.1	Main features of the FullCAN operation mode ...	3
<b>3.</b>	<b>Acceptance Filter .....</b>	<b>3</b>
3.1	Acceptance Filter message reception process...	5
3.2	FullCAN message reception process .....	6
<b>4.</b>	<b>Programming the Acceptance Filter .....</b>	<b>7</b>
4.1	CAN controller initialization .....	7
4.2	Acceptance Filter modes.....	8
4.3	Acceptance Filter and FullCAN mode enabled...	8
4.3.1	Example .....	11
4.3.2	Another example .....	12
4.4	Filter Table programming guidelines .....	13
<b>5.</b>	<b>Reading FullCAN Message Objects .....</b>	<b>13</b>
<b>6.</b>	<b>CAN error management .....</b>	<b>15</b>
6.1	Error Interrupts .....	16
6.2	Error Code Capture .....	17
6.3	Receive overrun condition.....	19
<b>7.</b>	<b>General CAN controller driver description.....</b>	<b>20</b>
7.1	Software development support.....	20
7.2	CAN controller driver functionality overview .....	20
7.3	CAN message buffers data structure .....	23
<b>8.</b>	<b>Demo description .....</b>	<b>23</b>
8.1	Demo hardware.....	23
8.2	Demo setup.....	23
8.2.1	Introduction .....	23
8.2.2	Hardware setup.....	24
8.2.3	Demo operation step.....	24
8.3	Demo displayed information.....	25
<b>9.</b>	<b>Reference .....</b>	<b>27</b>
<b>10.</b>	<b>Appendix A Software flowchart for demo ....</b>	<b>28</b>
<b>11.</b>	<b>Legal information .....</b>	<b>29</b>
11.1	Definitions .....	29
11.2	Disclaimers.....	29
11.3	Trademarks.....	29
<b>12.</b>	<b>Contents.....</b>	<b>30</b>

---

Please be aware that important notices concerning this document and the product(s) described herein, have been included in the section 'Legal information'.

---

© NXP B.V. 2008. All rights reserved.

For more information, please visit: <http://www.nxp.com>  
 For sales office addresses, email to: [salesaddresses@nxp.com](mailto:salesaddresses@nxp.com)

Date of release: 10 January 2008  
 Document identifier: AN10674\_1

