Semiconductors

AN10324

Implementing FullCAN-like message handling on the LPC21xx and LPC22xx

Rev. 01 — 26 August 2004 Application note

Document information

Info	Content
Keywords	Application Note, CANbus, FullCAN, LPC2119, LPC2129, LPC2219, LPC2229
Abstract	This document describes how to implement a FullCAN-like operating mode on the Philips LPC21xx and LPC22xx microcontrollers with CAN interface.

Revision history

Rev	Date	Description
01	20040826	Initial version.

1. Introduction

The Philips LPC21xx and LPC22xx microcontrollers with CAN interface feature a FullCAN operation mode that directly stores received messages with selected CAN message identifiers into a message buffer. Unfortunately this operation mode does not function reliably on the early revisions.

This application note shows and explains how the FullCAN operation mode can be implemented in software. The accompanying software is packed in the file *LPC2_FullCAN.zip* available from http://groups.yahoo.com/group/lpc2000/files/

2. Main features of the FullCAN operation mode

The FullCAN operation mode provides a FullCAN-like behavior of the CAN interface as commonly used by many CAN controllers. 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’). On the LPC21xx and LPC22xx microcontrollers hundreds of such filters can be used. The exact number depends on multiple parameters, such as the number of CAN interfaces sharing the reception filter and if other operation and filter modes are used in parallel.

3. Functionality of the CAN driver software

The CAN driver software package *LPC2_FullCAN* implements the FullCAN operation mode in software, using a combination of hardware acceptance filters and the CAN receive interrupts to handle incoming messages. In addition it provides functions that simplify the initialization of the CAN interfaces and the reception filters.

The entire driver is implemented in the module *LPC_FullCAN_SW*. The definitions for the special function registers are located in *LPC21XX.h* (provided by Keil Software). The module was written for and tested with the GNU compiler.

3.1 Defines

Two compiler definitions are made in file *LPC_FullCAN_SW.h*, they define the maximum number of CAN interfaces used and the maximum number of reception filters used.

3.1.1 MAX_CANPORTS

```
#define MAX_CANPORTS 2
```

This setting specifies how many CAN interfaces are used. The minimum is 1, the maximum may not exceed the number of CAN interfaces available on the LPC21xx or LPC22xx derivative used.

3.1.2 MAX_FILTERS

```
#define MAX_FILTERS 20
```

This setting specifies how many CAN reception filters are used in total. With each filter a 12-byte reception buffer in RAM is used for the storage of the received message.

3.2 Data types and global variables

The FullCAN driver uses a common 12-byte structure for the CAN messages and allocates an array of such messages as a reception buffer.

3.2.1 Structure FULLCAN_MSG

```
typedef struct
{
  unsigned int Dat1; // Bits  0..10: CAN Message ID
                     // Bits 13..15: CAN interface number (1..4)
                     // Bits 16..19: DLC - Data Length Counter
                     // Bits 24..25: Semaphore bits
  unsigned int DatA; // CAN Message Data Bytes 0-3
  unsigned int DatB; // CAN Message Data Bytes 4-7
} FULLCAN_MSG;
```

This 12-byte message structure is compatible to the message structure specified in the LPC21xx and LPC22xx user manual.

3.2.2 Reception buffer

```
FULLCAN_MSG gFullCANList[MAX_FILTERS];
```

This array of CAN messages holds the incoming CAN messages. If a CAN receive interrupt service routine finds that an incoming message matches one of the specified matching message IDs, the message gets copied into the associated filed in this array.

3.3 Functions

The FullCAN driver provides several functions for the initialization of the CAN interface and to handle the transmission and reception of CAN messages.

3.3.1 Initialization of a CAN interface

```
/*****************************************************************************
DOES:    Initializes one CAN interface of the LPC2000
GLOBALS: Resets all FullCAN filters, sets and enables CAN receive interrupt
RETURNS: One if initialization successful, else zero
*****************************************************************************/
short FullCAN_Init (
  unsigned short can_port,    // CAN interface to init (1, 2, 3 or 4)
  unsigned short can_isrvect, // Interrupt vector number to use for Rx ISR (0-15)
  unsigned int can_btr        // CAN BTR value used to set CAN baud rate
  );
```

This function performs all steps of the initialization of a CAN interface. It ensures that the pins used by this CAN interface get configured, it clears all reception filters and sets up the interrupt service routine for CAN message reception.

3.3.2 Adding a reception filter

```
/*****************************************************************************
DOES:    Setup a FullCAN filter
GLOBALS: Adds the specified CAN identifier to the list of CAN messages
         received by this device
RETURNS: One if operation successful, else zero
*****************************************************************************/
short FullCAN_SetFilter
  (
  unsigned short can_port, // CAN interface to init (1, 2, 3 or 4)
  unsigned int CANID       // 11-bit CAN message identifier
  );
```

This function adds a CAN message identifier reception filter to the driver. Multiple calls may be made for each additional filter that needs to be set. Calls do not need to be in any order, the function automatically ensures that the filters get internally sorted as required by the CAN peripheral of the LPC21xx and LPC22xx.

3.3.3 Transmitting a message

```
/*****************************************************************************
DOES:    Adds a message to the three-buffer transmit queue of a selected
         CAN interface. The message is placed in the next available buffer.
GLOBALS: None
RETURNS: One if operation successful, else zero
*****************************************************************************/
short FullCAN_PushMessage (
  unsigned short can_port,  // CAN interface to use (1, 2, 3 or 4)
  FULLCAN_MSG *pTransmitBuf // Source pointer to a CAN message
  );
```

By calling this function a CAN message is entered into the 3-buffer transmit queue. When initializing the transmit message contents, the semaphore bits can be ignored and should be left a zero.

3.3.4 Receiving a message

```
/*****************************************************************************
DOES:    Poll the Full CAN message storage area for the next available
         CAN message received.
GLOBALS: If a CAN message is found, the matching semaphore bits are cleared
RETURNS: One if operation successful, else zero
*****************************************************************************/
short FullCAN_PullMessage (
  unsigned short can_port, // CAN interface to use (1, 2, 3 or 4)
  FULLCAN_MSG *pReceiveBuf // Destination pointer to a CAN message
  );
```

By calling this function, the driver checks if a CAN message was received into the FullCAN buffer. This function always checks the buffers from the beginning. As the buffers and filters are sorted by CAN message identifier (ascending) this method ensures that the messages are automatically checked ordered by priority.

4. Functionality of the example program

The provided example program implements a simple example using two CAN interfaces. For this example to work, the two CAN interfaces must be physically connected to the same CAN bus. Each CAN interface is initialized to use a CAN bit rate of 125 kbps. The receive interrupt for CAN port 1 uses interrupt vector zero and the one for CAN port 2 uses interrupt vector 1.

The following reception filters are set:

- CAN port 1, message 102h
- CAN port 1, message 202h
- CAN port 2, message 101h
- CAN port 2, message 201h

In the main endless loop, the function *FullCAN_PullMessage* is continuously called to check if any of the defined messages was received. If a message was received, the content is altered and the message gets re-transmitted using a different CAN identifier.

As a result the four messages defined above are bouncing back and forth between the two CAN interfaces.

5. Compatibility, performance and alternative

The main intention of this application note is to provide an implementation compatible to the internal FullCAN operation mode of the Philips LPC21xx and LPC22xx microcontrollers that does not operate reliably in the early revisions of these chips.

Due to its compatibility with the LPC21xx and LPC22xx hardware this *LPC_FullCAN_SW* driver can later easily be ported to a version using the internal FullCAN operation mode.

The price for this compatibility is performance weakness. In this FullCAN compatible implementation a call to *FullCAN_PullMessage* always starts over scanning all buffers if any of them contains a received message. With each increase of the number of FullCAN buffers this scan-loop is increased to.

If compatibility to the FullCAN mode is not required, a different input buffer method should be implemented to improve performance: a FIFO receive queue. Each incoming CAN message is copied into the queue and a pull function to empty the queue would not need to do any scanning at all. If the queue is not empty it would just take the oldest message from the queue and return it.

Such a receive queue method is implemented in the example *LPC_CANAll.zip* also available from http://groups.yahoo.com/group/lpc2000/files/.

6. Disclaimers

Life support — These products are not designed for use in life support appliances, devices, or systems where malfunction of these products can reasonably be expected to result in personal injury. Philips Semiconductors customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify Philips Semiconductors for any damages resulting from such application.

Right to make changes — Philips Semiconductors reserves the right to make changes in the products - including circuits, standard cells, and/or software - described or contained herein in order to improve design and/or performance. When the product is in full production (status 'Production'), relevant changes will be communicated via a Customer Product/Process Change Notification (CPCN). Philips Semiconductors assumes no responsibility or liability for the use of any of these products, conveys no licence or title under any patent, copyright, or mask work right to these products, and makes no representations or warranties that these products are free from patent, copyright, or mask work right infringement, unless otherwise specified.

7. Contents

Date of release: 26 August 2004
Document number: 9397 750 13999

Published in The U.S.A.