Semiconductors

AN10438

Philips LPC2000 CAN driver

Rev. 01 — 02 March 2006 Application note

Document information

Info	**Content**
Keywords	CAN BUS, MCU, LPC2000, ARM7, SJA1000
Abstract	This application note describes the CAN controller hardware application programming interface driver routines for the CAN controller of Philips LPC2000. Also provides the demo project developed under KEIL uVision3 with ARM tools ADS1.2, using evaluation board MCB210. This demo used UART0 for communication with PC, and print CAN communication information with Hyper terminal.

PHILIPS

Revision history

Rev	Date	Description
01	20060302	Initial version.

Contact information

For additional information, please visit: http://www.semiconductors.philips.com

For sales office addresses, please send an email to: sales.addresses@www.semiconductors.philips.com

1. Introduction

1.1 Introduction

The LPC2292/LPC2294 each contain two/four CAN controllers. The CAN is a serial communication protocol, which efficiently supports distributed real-time control with a very high level of security. Its domain of application ranges from high-speed networks to lower cost multiplex wiring.

Each CAN Controller has a register structure similar to the Philips SJA1000 and the PeliCAN Library block, but the 8-bit registers of those devices have been combined in 32-bit words to allow simultaneous access in the ARM environment. The main operational difference is that the recognition of received Identifiers, known in CAN terminology as Acceptance Filtering, has been removed from the CAN controllers and centralized in a global Acceptance Filter.

1.2 Main features

The main features for CAN controller of LPC2000:

- Data rates up to 1 Mbit/s on each bus
- 32-bit register and RAM access
- Compatible with CAN specification 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

1.3 More useful features

You can also find the following useful features when you drive LPC2000 CAN controllers in your applications:

- Error counters with read/write access
- Programmable error warning limit
- Error interrupted for each CAN-bus error
- Arbitration lost interrupted with detailed bit position
- Single-shot transmission (no re-transmission)
- Listen only mode (no acknowledge, no active error flags)
- Reception of ‘own’ messages (self reception request).

For detailed usage of the CAN controller, please reference the LPC2000 parts data sheet and user manual.

1.4 CAN BUS timing setting

In this CAN driver, included are the pre-defined values for CAN Bit timing.

Bus Timing Register – CANBTR controls this function; it consists of the following bits:

Bit 0:9 – BRP:	Baud Rate Prescaler. The VPB clock is divided by (this value plus one) to produce the CAN clock
Bit 14:15 – SJW:	The Synchronization Jump Width is (this value plus one) CAN clocks
Bit 16:19 – TSEG1:	The delay from the nominal Sync point to the sample point is (this value plus one) CAN clocks
Bit 20:22 – TSEG2:	The delay from the sample point to the next nominal sync point is (this value plus one) CAN clocks. The nominal CAN bit time is (this value plus the value in TSEG1 plus 3) CAN clocks
Bit 23 – SAM:	1: the bus is sampled 3 times (recommended for low to medium speed buses)
	0: the bus is sampled once (recommended for high speed buses)

The definition of the CAN bit timing value is something flexible, but you can obey a simple principle as follows:

Define:

$\mathrm{Tscl} = \frac{\mathrm{BRP}+1}{\mathrm{Fvpb}}$ as CAN clock;

Tseg1 = (TSEG1+1)*Tscl; Tseg2 = (TSEG2+1)*Tscl; Tsjw = (SJW+1)*Tscl

Then the principle as follows:

- Tseg2 >= 2Tscl
- Tseg2 >= Tsjw
- Tseg1 >= Tseg2

For example, in this CAN driver, the CAN bit rate can be set as 100 k @ 24 MHZ (VPB clock)

Bus Timing Register – CANBTR can be set to: 0x001C000E

Meaning:

BRP = 14;

SJW = 0;

TSEG1 = 12;

TSEG2 = 1;

The CAN bit rate can be calculated by the following equation:

$$\text{CANbps} = \frac{\text{Fvpb}}{(\text{TSEG1} + \text{TSEG2} + 3)*(\text{BRP} + 1)}$$

• CANbps:	CAN bit rate
• Fvpb:	VPB frequency

As shown in the equation, when Fvpb = 24 MHZ, the CANbps = 100 kbits/s

While:

$$\text{MaxCANbps} = \frac{\text{Fvpb}}{(\text{TSEG1} + \text{TSEG2} + 3 - (\text{SJW} + 1))*(\text{BRP} + 1)} = 106\text{Kbit/s}$$

$$\text{MinCANbps} = \frac{\text{Fvpb}}{(\text{TSEG1} + \text{TSEG2} + 3 + (\text{SJW} + 1))*(\text{BRP} + 1)} = 94\text{Kbit/s}$$

2. Introduction for CAN controller driver

This application note describes a project under KEIL uVision3, which is a live demo based on the LPC2000 CAN driver routines supplied as “C” source code.

It is one project, but also can be regarded as two parts:

1. The CAN controller driver routines for LPC2000
2. A demo code based on KEIL MCB2100 evaluation board, using UART print CAN BUS transmit and receive information with PC Hyper Terminal.

The CAN controller driver includes functions, which cover basically the handling of the CAN controller. This Application Note will provide information about all functions made available with the driver, and how the driver interacts with the device for each of these function calls.

The demo tools, method, and operation steps will be described in a later chapter.

3. General CAN controller driver description

3.1 Software development support

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

- IDE – KEIL uVision3
- Emulator - ULINK
- ARM develop tool – ARM ADS, Version 1.2

3.2 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.

3.3 CAN controller driver files

The CAN controller driver consists of the following files:

Table 1: CAN controller driver files

CAN controller driver files description	
Config.h	This file contains typedef statements of the data types and macro definitions. This file should be #included in the application source file and must be added to the project. This file must **not** be modified.
LPC2294.h	This file contains #define statements of LPC2000 CAN controller, configuring the SFR address. This file should be #included in the application source file and must be added to the project. This file must **not** be modified.
LPC2000_CAN.h	This file contains function prototypes, typedef struct statements of receive and transmit data buffers, #define statements of all function error return codes and all pre-defined values for various baud rates. This file should be #included in the application source file and must be added to the project. This file must **not** be modified.
LPC2000_CAN_GAF.h	This file contains typedef struct statements of Global Acceptance Filter units. This file must be added to the project. This file must **not** be modified.
LPC2000_CAN_Driver.c	This file contains the LPC2000 CAN driver function description. It includes sent data definition and an example for the Global Acceptance Filter configuration. This file must be added to the project. This file may be modified for sending needed data and setting up needed Global Acceptance Filter.

3.4 Used data types

Due to different implementations of data types of available C compilers, the following data types from the CAN controller driver are defined in the header file “config.h”.

Table 2: CAN controller driver typedef statements

Mnemonic	**C type**	
UInt8	unsigned char	8 bit unsigned integer
pUInt8	unsigned char*	8 bit unsigned integer pointer
UInt32	unsigned long	32 bit unsigned integer
pUInt32	unsigned long*	32 bit unsigned integer pointer
CanStatusCode	unsigned long	Function status return code

3.5 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 3). Both structures are defined in the header file LPC2000_CAN.h.

Table 3: Transmit and receive Message Buffer defined in LPC2000_CAN.h

Transmit buffer: lpc2000CANdriver_TXObj_t	**Receive buffer:** lpc2000CANdriver_RXObj_t

typedef struct
{
   UInt32   TFI;
   UInt32   ID;
   UInt32   DataField[2];
 	
} lpc2000CANdriver_TXObj_t;
	typedef struct
	{
	   UInt32 RFS;
	   UInt32 ID;
	   UInt32 DataField[2];
	
	} lpc2000CANdriver_RXObj_t;

Table 4 shows how the 32-bit wide array Data Field and the CAN Message Buffer data is organized.

Table 4: Array DataField

	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

Table 5 and Table 6 are examples of how both Message Buffer structures are used in the main Application Program.

Table 5: Example for preparing a transmit message

Example: Loading a Transmit Message into the CAN controllers Transmit Buffer

```
/* Transmit Object Structure and Function Call */

UInt8 Reture_Value =0;                                      // variable  to report CAN status
lpc2000CANdriver_TXObj_t   CAN_Send_Data[ ] =              // Tx Message Object
 { {0x00080000,0x00000000,0x04030201,0x08070605},
   {0x00080000,0x000001AC,0x14131211,0x18171615}
 };

main()
{
  UInt32 i=0;
  plpc2000CANdriver_TXObj_t    pCAN_Send_Data;

  ……
  pCAN_Send_Data = CAN_Send_Data;
  for( i=0;i<2;i++)
    {
      Reture_Value =
       lpc2000CANdriver_CertainTxBufTransmitMessage(LPC2000_CANCHANNEL_1,
                                                   pCAN_Send_Data,LPC2000_TXB_1);

      pCAN_Send_Data++;
    }
  ……
}
```

Table 6: Example for reading the Id, RFS and data bytes from received message

Example: Reading a CAN Message Data from the Receive Buffer

```
/* Receive Object Structure and Function Call */

UInt8 Reture_Value =0;                                   // variable  to report CAN status
lpc2000CANdriver_RXObj_t   CAN_Rcv_Data[2];             // Rx Message Object
UInt32 CAN_Rcv_Data_Counter=0;                          // transmit value between main and IRQ

void   __irq IRQ_CAN2Rx(void)
{
  plpc2000CANdriver_RXObj_t    pCAN2_Rcv_Data_IRQ;

  ……
  pCAN2_Rcv_Data_IRQ = CAN_Rcv_Data + CAN_Rcv_Data_Counter;
  Reture_Value = lpc2000CANdriver_ReceiveMessageCh2 (pCAN2_Rcv_Data_IRQ);
  CAN_Rcv_Data_Counter++;
  ……
}

main()
{
  ……
  RFS      = CAN_Rcv_Data[1] . RFS;
  ID       = CAN_Rcv_Data[1] . ID;
  Data_1  = (CAN_Rcv_Data[1] . DataField[0] & 0x000000FF);
  Data_2  = (CAN_Rcv_Data[1] . DataField[0] & 0x0000FF00) >> 8;
  Data_3  = (CAN_Rcv_Data[1] . DataField[0] & 0x00FF0000) >> 16;
  Data_4  = (CAN_Rcv_Data[1] . DataField[0] & 0XFF000000) >> 24;
  Data_5  = (CAN_Rcv_Data[1] . DataField[1] & 0x000000FF);
  Data_6  = (CAN_Rcv_Data[1] . DataField[1] & 0x0000FF00) >> 8;
  Data_7  = (CAN_Rcv_Data[1] . DataField[1] & 0x00FF0000) >> 16;
  Data_8  = (CAN_Rcv_Data[1] . DataField[1] & 0xFF000000) >> 24;
  ……
}
```

3.6 Global acceptance filter configuration

In this AN, the Global Acceptance Filter (GAF) configuration lists the initialization of four Look-up table sections.

For the CAN controller driver the file LPC2000_CAN_Driver.c contains the four 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.

Table 7: ID look-up table definitions of the LPC2000_CAN_Driver.c

Section Definition :	Section of the ID Look_up Table Memory
#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

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 8 to Table 11, 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 8: 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 9: 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 10: Example of explicit extended frame format identifier section

```
const lpc2000CANdriver_ACFilterx_t  gklpc2000CANdriver_ExtIndividualSection[ ] =
  {
    /* Channel                          29-bit Identifier  ( =< 0x1FFFFFFF) */
    {LPC2000_CANDRIVER_SCC_2,      0x00002288},
    {LPC2000_CANDRIVER_SCC_2,      0x00003377},
    {LPC2000_CANDRIVER_SCC_2,      0x00005566},
    {LPC2000_CANDRIVER_SCC_2,      0x00006677}
  };
```

Table 11: 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

  };
```

3.6.1 Guidelines for setting ID look-up table

All identifier sections of the ID Look-up Table have to be programmed in such a way that each active section is organized as a sorted list or table with an **increasing order of the SCC**. Entries with the same SCC have to provide an **increasing order of the CAN identifier**. Additionally, for entries in each section, the following rules apply:

Table 12: General rules of section entries

Explicit Standard Frame Format ID Section Explicit Extended Frame Format ID Section	Even or odd number of entries is allowed
Group of Standard Frame Format ID Section Group of Extended Frame Format ID Section	Lower and upper bound has to be defined for each group. Even or odd numbers of ground are allowed

The following function-call loads all pre-defined CAN identifier entries into the ID Look-up Table Memory and configures the section start address registers according to the pre-defined CAN identifiers in the enabled sections.

Table 13: GAF function call configuration

```
{
  UInt8 Reture_Value =0;                              // variable  to report CAN status

  ……
  Reture_Value = lpc2000CANdriver_SetACFMode( LPC2000_ACC_BYPASS );
  Reture_Value = lpc2000CANdriver_LoadAcceptanceFilter ();
  ……
}
```

3.7 Function return values

All functions that may cause an error condition will return an error code. The error codes are defined in the Header File LPC2000_CAN.h

Table 14: CAN controller driver function return values

Return_Value	Description
LPC2000_CANDRIVER_INITIAL	Start up status
LPC2000_CANDRIVER_OK	No error occurred
LPC2000_CANDRIVER_ERR	General error
LPC2000_CANDRIVER_ERR_WRONG_CAN_CHANNEL_NUMBER	Channel number is out of range
LPC2000_CANDRIVER_ERR_WRONG_CAN_TxBUFFER_NUMBER	Wrong CAN transmit buffer number
LPC2000_CANDRIVER_ERR_TRANSMIT_BUFFER1_NOT_FREE	CAN transmit buffer 1 is not free
LPC2000_CANDRIVER_ERR_TRANSMIT_BUFFER2_NOT_FREE	CAN transmit buffer 2 is not free
LPC2000_CANDRIVER_ERR_TRANSMIT_BUFFER3_NOT_FREE	CAN transmit buffer 3 is not free
LPC2000_CANDRIVER_ERR_TABLE_ERROR_IN_STD_INDIVIDUAL_SECTION	A look-up table error was detected in the Std Individual Section
LPC2000_CANDRIVER_ERR_TABLE_ERROR_IN_STD_GROUP_SECTION	A look-up table error was detected in the Std Group Section
LPC2000_CANDRIVER_ERR_TABLE_ERROR_IN_EXT_INDIVIDUAL_SECTION	A look-up table error was detected in the Ext Individual Section
LPC2000_CANDRIVER_ERR_TABLE_ERROR_IN_EXT_GROUP_SECTION	A look-up table error was detected in the Ext Group Section
LPC2000_CANDRIVER_ERR_NOT_ENOUGH_MEMORY_SPACE	The available memory space is not sufficient for the configured CAN identifiers

3.8 Pre-defined CAN bit timing values

According the principle for calculating CAN BUS Timing illustrated in Chapter 1.4, you can use the following Pre-defined value.

Table 15: Pre-defined values for the CAN Bit Timing

Defined CAN Bit Timing Values	Description
LPC2000_CANDRIVER_CANBITRATE100K16MHZ	100 kbit/s for Fvpb = 16 MHz clock frequency
LPC2000_CANDRIVER_CANBITRATE125K16MHZ	125 kbit/s for Fvpb = 16 MHz clock frequency
LPC2000_CANDRIVER_CANBITRATE250K16MHZ	250 kbit/s for Fvpb = 16 MHz clock frequency
LPC2000_CANDRIVER_CANBITRATE500K16MHZ	500 kbit/s for Fvpb = 16 MHz clock frequency
LPC2000_CANDRIVER_CANBITRATE1000K16MHZ	1000 kbit/s for Fvpb = 16 MHz clock frequency
LPC2000_CANDRIVER_CANBITRATE100K20MHZ	100 kbit/s for Fvpb = 20 MHz clock frequency
LPC2000_CANDRIVER_CANBITRATE125K20MHZ	125 kbit/s for Fvpb = 20 MHz clock frequency
LPC2000_CANDRIVER_CANBITRATE250K20MHZ	250 kbit/s for Fvpb = 20 MHz clock frequency
LPC2000_CANDRIVER_CANBITRATE500K20MHZ	500 kbit/s for Fvpb = 20 MHz clock frequency
LPC2000_CANDRIVER_CANBITRATE1000K20MHZ	1000 kbit/s for Fvpb = 20 MHz clock frequency
LPC2000_CANDRIVER_CANBITRATE100K24MHZ	100 kbit/s for Fvpb = 24 MHz clock frequency
LPC2000_CANDRIVER_CANBITRATE125K24MHZ	125 kbit/s for Fvpb = 24 MHz clock frequency
LPC2000_CANDRIVER_CANBITRATE250K24MHZ	250 kbit/s for Fvpb = 24 MHz clock frequency
LPC2000_CANDRIVER_CANBITRATE500K24MHZ	500 kbit/s for Fvpb = 24 MHz clock frequency
LPC2000_CANDRIVER_CANBITRATE1000K24MHZ	1000 kbit/s for Fvpb = 24 MHz clock frequency
LPC2000_CANDRIVER_CANBITRATE100K48MHZ	100 kbit/s for Fvpb = 48 MHz clock frequency
LPC2000_CANDRIVER_CANBITRATE125K48MHZ	125 kbit/s for Fvpb = 48 MHz clock frequency
LPC2000_CANDRIVER_CANBITRATE250K48MHZ	250 kbit/s for Fvpb = 48 MHz clock frequency
LPC2000_CANDRIVER_CANBITRATE500K48MHZ	500 kbit/s for Fvpb = 48 MHz clock frequency
LPC2000_CANDRIVER_CANBITRATE1000K48MHZ	1000 kbit/s for Fvpb = 48 MHz clock frequency
LPC2000_CANDRIVER_CANBITRATE100K60MHZ	100 kbit/s for Fvpb = 60 MHz clock frequency
LPC2000_CANDRIVER_CANBITRATE125K60MHZ	125 kbit/s for Fvpb = 60 MHz clock frequency
LPC2000_CANDRIVER_CANBITRATE250K60MHZ	250 kbit/s for Fvpb = 60 MHz clock frequency
LPC2000_CANDRIVER_CANBITRATE500K60MHZ	500 kbit/s for Fvpb = 60 MHz clock frequency
LPC2000_CANDRIVER_CANBITRATE1000K60MHZ	1000 kbit/s for Fvpb = 60 MHz clock frequency

3.9 Pre-defined function arguments

When calling functions, several parameters have to be passed. Instead of using bare numbers it is more concise to use pre-defined function arguments. The following arguments are already defined in the header file LPC2000_CAN.h and can be utilized by the user.

Table 16: Pre-defined function arguments in file LPC2000_CAN.h

Defined Function Arguments	Description
LPC2000_CANCHANNEL_1	Handle of CAN Channel 1
LPC2000_CANCHANNEL_2	Handle of CAN Channel 2
LPC2000_CANCHANNEL_3	Handle of CAN Channel 3
LPC2000_CANCHANNEL_4	Handle of CAN Channel 4
LPC2000_TXB_1	Handle of Transmit Buffer 1
LPC2000_TXB_2	Handle of Transmit Buffer 2
LPC2000_TXB_3	Handle of Transmit Buffer 3
LPC2000_ACC_ON	Global Acceptance Filter ON mode
LPC2000_ACC_OFF	Global Acceptance Filter OFF mode
LPC2000_ACC_BYPASS	Global Acceptance Filter BYPASS mode
CAN_OPERATING_MODE	CAN Controller Operating Mode
CAN_RESET_MODE	CAN Controller Reset Mode
CAN_LISTENONLY_MODE	CAN Controller Listen-Only Mode
CAN_SELFTEST_MODE	CAN Controller Selftest Mode

4. CAN driver functional description

The following sub-chapters describe the CAN controller driver functions in detail.

Table 17: CAN controller driver function summary

Name	Description
lpc2000CANdriver_CANInit	Initialize a CAN controller
lpc2000CANdriver_SetCANMode	Set or change the CAN controller mode
lpc2000CANdriver_SetACFMode	Set or change the Global Acceptance Filter mode
lpc2000CANdriver_CertainTxBufTransmitMessage	Load a certain Transmit Buffer and force a message transmission
lpc2000CANdriver_LoadAcceptanceFilter	Load the Global Acceptance Filter Look-up Table RAM with pre-defined CAN identifiers
lpc2000CANdriver_ReceiveMessageCh1	Copy a received message from CAN controller 1 into a certain location within the User RAM
lpc2000CANdriver_ReceiveMessageCh2	Copy a received message from CAN controller 2 into a certain location within the User RAM
lpc2000CANdriver_ReceiveMessageCh3	Copy a received message from CAN controller 3 into a certain location within the User RAM
lpc2000CANdriver_ReceiveMessageCh4	Copy a received message from CAN controller 4 into a certain location within the User RAM
Rcv_Data_Output_to_Screen	Print a received message to PC by UART0
Print_Chars_to_Screen	Print a sect of characters to PC by UART0
Print_4bits_to_Screen	Print 4bits of one 8 bits byte separately to PC by UART0

The last three functions are used for demo.

4.1 Interface description of CAN driver functions

4.1.1 lpc2000CANdriver_CANInit

Table 18: Interface description of lpc2000CANdriver_CANInit

Name	lpc2000CANdriver_CANInit
SYNOPSIS	CanStatusCode lpc2000CANdriver_CANInit (**UInt32** canChannel, **UInt32** canBitrate);
IN	canChannel Channel number of CAN controller: 1,2,3, or 4 CanBitrate Pre-defined CAN Bit Timing value
RETURN VALUE	Return values (see also Chapter 3.7)
POSSIBLE RETURN VALUE	LPC2000_CANDRIVER_OK LPC2000_CANDRIVER_ERR_WRONG_CAN_CHANNEL_NUMBER
DESCRIPTION	Calling this function initializes the selected CAN controller with the specified Bit Timing and sets the CAN controller into Operating Mode.
NOTES	None
EXAMPLE	Return_Value = lpc2000CANdriver_CANInit(LPC2000_CANCHANNEL_1, LPC2000_CANDRIVER_CANBITRATE100K24MHZ);

4.1.2 lpc2000CANdriver_SetCANMode

Table 19: Interface description of lpc2000CANdriver_SetCANMode

Name	**lpc2000CANdriver_SetCANMode**
SYNOPSIS	CanStatusCode lpc2000CANdriver_SetCANMode (**UInt32** canChannel, **UInt8** CANMode);
IN	canChannel Channel number of CAN controller: 1,2,3, or 4 CANMode CAN_OPERATING_MODE: Operation Mode CAN_RESET_MODE: Reset Mode CAN_LISTENONLY_MODE: Listen Only Mode CAN_SELFTEST_MODE: Self Test Mode
RETURN VALUE	Return values (see also Chapter 3.7)
POSSIBLE RETUEN VALUE	LPC2000_CANDRIVER_OK LPC2000_CANDRIVER_ERR
DESCRIPTION	Calling this function changes the CAN controller mode.
NOTES	None

EXAMPLE	Return_Value = lpc2000CANdriver_SetCANMode (
	LPC2000_CANCHANNEL_1, CAN_OPERATING_MODE);

4.1.3 lpc2000CANdriver_SetACFMode

Table 20: Interface description of lpc2000CANdriver_SetACFMode

Name	lpc2000CANdriver_SetACFMode
SYNOPSIS	CanStatusCode lpc2000CANdriver_SetACFMode(**UInt8** ACFMode);
IN	ACFMode LPC2000_ACC_ON: Global Acceptance Filter On LPC2000_ACC_OFF: Global Acceptance Filter On LPC2000_ACC_BYPASS: Global Acceptance Filter On
RETURN VALUE	Return values (see also Chapter 3.7)
POSSIBLE RETUEN VALUE	LPC2000_CANDRIVER_OK LPC2000_CANDRIVER_ERR
DESCRIPTION	Calling this function changes the CAN Global Acceptance Filter mode.
NOTES	None
EXAMPLE	Return_Value = lpc2000CANdriver_SetACFMode (LPC2000_ACC_BYPASS);

4.1.4 lpc2000CANdriver _CertainTxBufTransmitMessage

Table 21: Interface description of lpc2000CANdriver_ CertainTxBufTransmitMessage

Name	lpc2000CANdriver_CertainTxBufTransmitMessage
SYNOPSIS	CanStatusCode lpc2000CANdriver_CertainTxBufTransmitMessage (**UInt32** canChannel, **plpc2000CANdriver_TXObj_t** pTransmitBuf, **UInt32** txbuffer);
IN	canChannel Channel number of CAN controller: 1,2,3, or 4 plpc2000CANdriver_TXObj_t Pointer to a Transmit Buffer in User RAM txbuffer Transmit Buffer number: 1,2,or 3
RETURN VALUE	Return values (see also Chapter 3.7)
POSSIBLE RETUEN VALUE	LPC2000_CANDRIVER_OK LPC2000_CANDRIVER_ERR LPC2000_CANDRIVER_ERR_WRONG_CAN_TxBUFFER_NUMBER LPC2000_CANDRIVER_ERR_TRANSMIT_BUFFER1_NOT_FREE LPC2000_CANDRIVER_ERR_TRANSMIT_BUFFER2_NOT_FREE LPC2000_CANDRIVER_ERR_TRANSMIT_BUFFER3_NOT_FREE
DESCRIPTION	Calling this function forces the LPC2000 to copy transmit data from User RAM into a certain transmit buffer. A “buffer not free” return value is given if a certain transmit buffer is occupied.
NOTES	None
EXAMPLE	Return_Value = lpc2000CANdriver_CertainTxBufTransmitMessage (LPC2000_CANCHANNEL_1, pCAN_Send_Data, LPC2000_TXB_1);

4.1.5 lpc2000CANdriver_ReceiveMessageChx

Table 22: Interface description of lpc2000CANdriver_ ReceiveMessageChx

Name	**lpc2000CANdriver_ReceiveMessageCh1** **lpc2000CANdriver_ReceiveMessageCh2** **lpc2000CANdriver_ReceiveMessageCh3** **lpc2000CANdriver_ReceiveMessageCh4**
SYNOPSIS	CanStatusCode lpc2000CANdriver_ ReceiveMessageCh1 (**plpc2000CANdriver_RXObj_t** pReceiveBuf);
IN	pReceiveBuf Pointer to Receive Buffer in User RAM
RETURN VALUE	1
DESCRIPTION	Calling this function copies the current Receive Buffer contents into the specified Receive Buffer in User RAM. After copying the message data the CAN controller receive buffer is released.
NOTES	None

EXAMPLE	Return_Value = lpc2000CANdriver_ReceiveMessageCh2
	(pCAN2_Rcv_Data_IRQ);

4.1.6 lpc2000CANdriver_LoadAcceptanceFilter

Table 23: Interface description of lpc2000CANdriver_ LoadAcceptanceFilter

Name	lpc2000CANdriver_LoadAcceptanceFilter
SYNOPSIS	CanStatusCode lpc2000CANdriver_ LoadAcceptanceFilter(void);
IN	None
RETURN VALUE	Return values (see also Chapter 3.7)
POSSIBLE RETUEN VALUE	LPC2000_CANDRIVER_ERR_TABLE_ERROR_IN_STD_INDIVIDUAL_SECTION LPC2000_CANDRIVER_ERR_TABLE_ERROR_IN_STD_GROUP_SECTION LPC2000_CANDRIVER_ERR_TABLE_ERROR_IN_EXT_INDIVIDUAL_SECTION LPC2000_CANDRIVER_ERR_TABLE_ERROR_IN_EXT_GROUP_SECTION LPC2000_CANDRIVER_ERR_NOT_ENOUGH_MEMORY_SPACE
DESCRIPTION	Calling this function loads the Global Acceptance Filter Look_up Table RAM with pre-defined values stored in LPC2000_CAN_Driver.c. Depending on the number of pre-defined identifiers the register start addresses for the Global Acceptance Filter RAM are calculated and initialized automatically.
NOTES	None
EXAMPLE	Return_Value = lpc2000CANdriver_LoadAcceptanceFilter ();

4.1.7 Rcv_Data_Output_to_Screen

Table 24: Interface description of Rcv_Data_Output_to_Screen

Name	**Rcv_Data_Output_to_Screen**
SYNOPSIS	CanStatusCode Rcv_Data_Output_to_Screen (**plpc2000CANdriver_RXObj_t** pReceiveBuf);
IN	pReceiveBuf Pointer to Receive Buffer in User RAM
RETURN VALUE	0
DESCRIPTION	Calling this function display one CAN frame of Received Data information stored in User RAM to PC Screen by Uart0
NOTES	None
EXAMPLE	Rcv_Data_Output_to_Screen(pCAN_Rcv_Data);

4.1.8 Print_Chars_to_Screen

Table 25: Interface description of Print_Chars_to_Screen

Name	**Print_Chars_to_Screen**
SYNOPSIS	CanStatusCode Print_Chars_to_Screen (**UInt8 const *** pscreen_print);
IN	pscreen_print Pointer to the head of one sect of Chars that be prepared to printed
RETURN VALUE	0
DESCRIPTION	Calling this function display one sect of chars to PC Screen by Uart0
NOTES	None
EXAMPLE	Print_Chars_to_Screen(Demo_Sent_Choice);

4.1.9 Print_4bits_to_Screen

Table 26: Interface description of Print_4bits_to_Screen

Name	Print_4bits_to_Screen
SYNOPSIS	CanStatusCode Print_4bits_to_Screen(**UInt32** Value_Hex_4bits);
IN	Value_Hex_4bits 4bits HEX value that be prepared to printed
RETURN VALUE	0
DESCRIPTION	Calling this function display 4bits HEX value to PC Screen by Uart0
NOTES	None
EXAMPLE	Print_4bits_to_Screen((Temp_Data_Output[0] & 0x0000000F));

5. Demo description

5.1 Demo tools

IDE: KEIL uVision3

Demo board: One KEIL MCB2100 Evaluation Board

Info display: Hyper Terminal

Please visit Keil's website for detailed MCB2100 board information.

5.2 Demo function description

CAN communication uses one MCB2100 board for demo. The electrical diagram for the CAN part is shown in Fig 1.

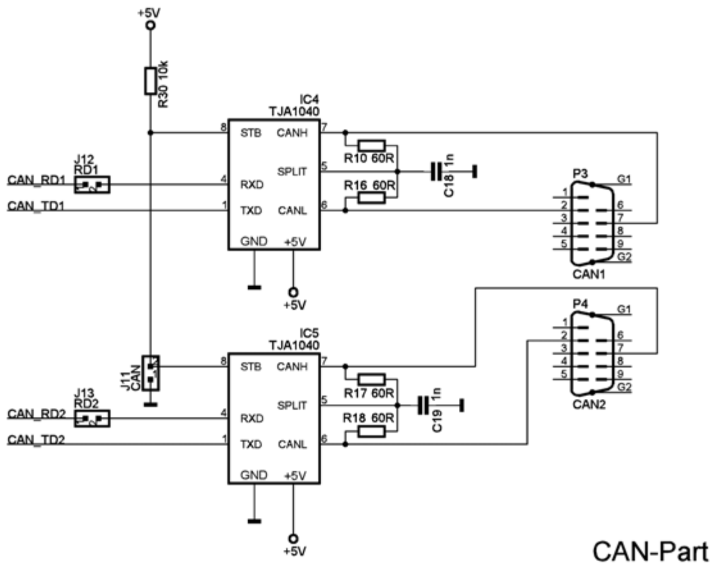


Fig 1. CAN demo electrical diagram

The LPC2129 CAN1 transmits data to the CAN BUS, by UART0 print CAN1 transmit message and Global Acceptance Filter Look-up Table to PC screen.

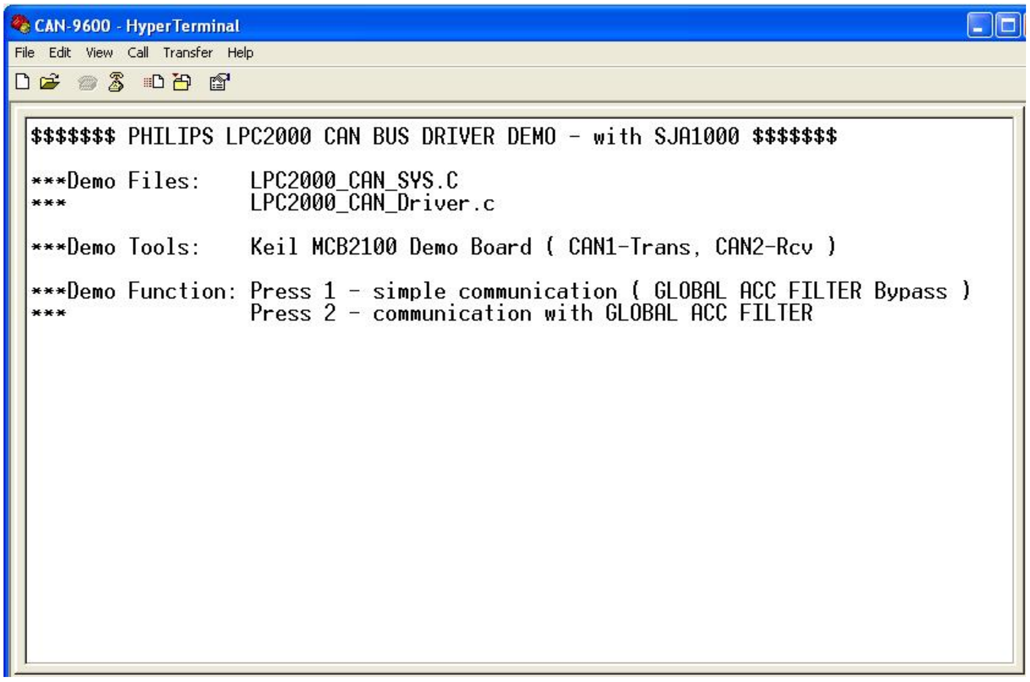
LPC2129 CAN2 receives data from the CAN BUS, by UART0 print CAN2 received message.

This demo demonstrates two kinds of CAN communication methods:

1. Simple CAN BUS communication without GAF (Global Acceptance Filter) enabled
2. CAN BUS communication with GAF enabled.

5.2.1 Simple CAN BUS communication demo

When you open the demo project and start debug in uVision3, on Hyper Terminal, it will display the following demo start information:

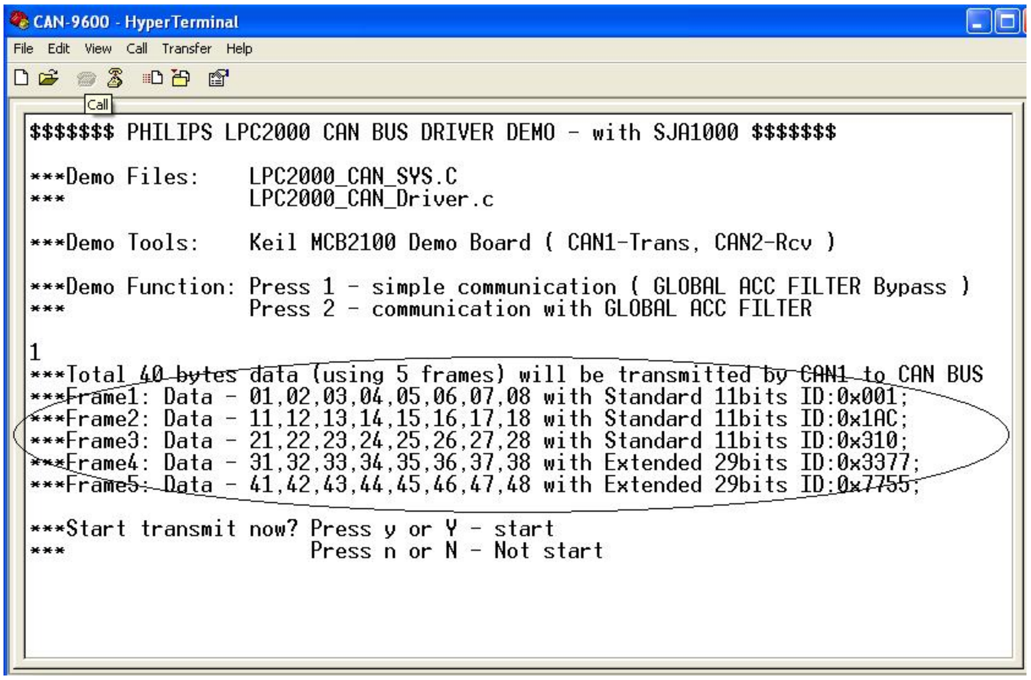


```
$$$$$$$ PHILIPS LPC2000 CAN BUS DRIVER DEMO - with SJA1000 $$$$$$$

***Demo Files:     LPC2000_CAN_SYS.C
***                LPC2000_CAN_Driver.c

***Demo Tools:     Keil MCB2100 Demo Board ( CAN1-Trans, CAN2-Rcv )

***Demo Function: Press 1 - simple communication ( GLOBAL ACC FILTER Bypass )
***                Press 2 - communication with GLOBAL ACC FILTER
```

Fig 2. Welcome page for the demo

Choosing the first function will demo simple CAN BUS communication, with GAF (Global Acceptance Filter) set to Bypass mode.

The demo will display the CAN message that will be prepared to be transmitted.



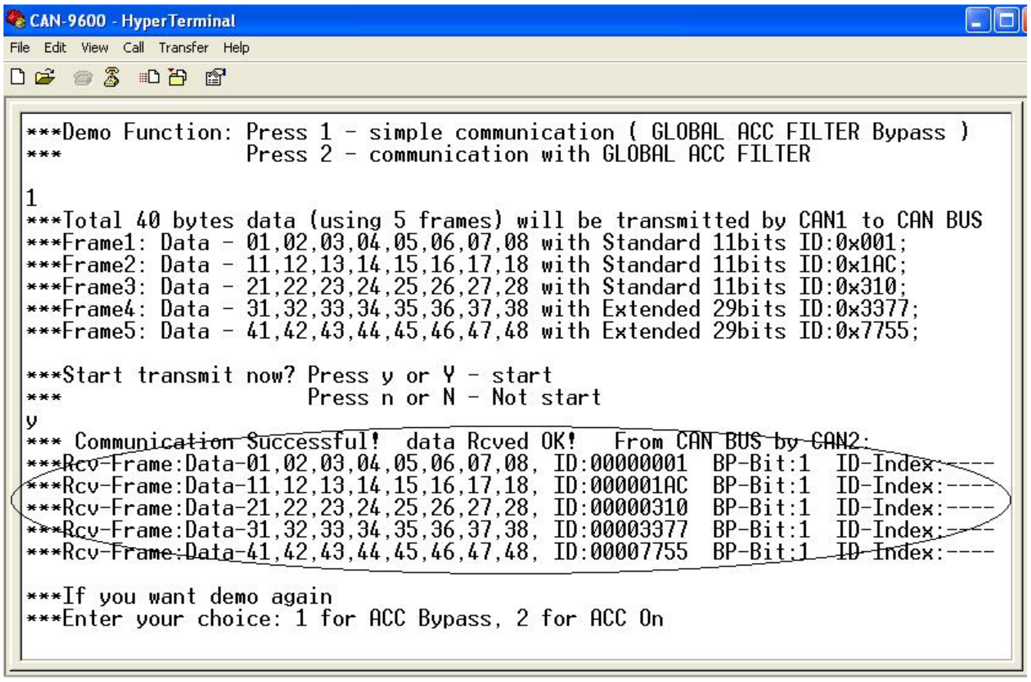
```
$$$$$$$ PHILIPS LPC2000 CAN BUS DRIVER DEMO - with SJA1000 $$$$$$$

***Demo Files:     LPC2000_CAN_SYS.C
***                LPC2000_CAN_Driver.c

***Demo Tools:     Keil MCB2100 Demo Board ( CAN1-Trans, CAN2-Rcv )

***Demo Function: Press 1 - simple communication ( GLOBAL ACC FILTER Bypass )
***                Press 2 - communication with GLOBAL ACC FILTER

1
***Total 40 bytes data (using 5 frames) will be transmitted by CAN1 to CAN BUS
***Frame1: Data - 01,02,03,04,05,06,07,08 with Standard 11bits ID:0x001;
***Frame2: Data - 11,12,13,14,15,16,17,18 with Standard 11bits ID:0x1AC;
***Frame3: Data - 21,22,23,24,25,26,27,28 with Standard 11bits ID:0x310;
***Frame4: Data - 31,32,33,34,35,36,37,38 with Extended 29bits ID:0x3377;
***Frame5: Data - 41,42,43,44,45,46,47,48 with Extended 29bits ID:0x7755;

***Start transmit now? Press y or Y - start
***                    Press n or N - Not start
```

Fig 3. CAN message prepared to be sent demo page

When you confirm transmission, demo code will start transmit.

In this mode, every data transmitted will be received if CAN BUS works normally.

If received successfully, then demo code will display the received data.



```
***Demo Function: Press 1 - simple communication ( GLOBAL ACC FILTER Bypass )
***               Press 2 - communication with GLOBAL ACC FILTER

1
***Total 40 bytes data (using 5 frames) will be transmitted by CAN1 to CAN BUS
***Frame1: Data - 01,02,03,04,05,06,07,08 with Standard 11bits ID:0x001;
***Frame2: Data - 11,12,13,14,15,16,17,18 with Standard 11bits ID:0x1AC;
***Frame3: Data - 21,22,23,24,25,26,27,28 with Standard 11bits ID:0x310;
***Frame4: Data - 31,32,33,34,35,36,37,38 with Extended 29bits ID:0x3377;
***Frame5: Data - 41,42,43,44,45,46,47,48 with Extended 29bits ID:0x7755;

***Start transmit now? Press y or Y - start
***                    Press n or N - Not start
y
*** Communication Successful!  data Rcved OK!   From CAN BUS by CAN2:
***Rcv-Frame:Data-01,02,03,04,05,06,07,08, ID:00000001  BP-Bit:1  ID-Index:----
***Rcv-Frame:Data-11,12,13,14,15,16,17,18, ID:000001AC  BP-Bit:1  ID-Index:----
***Rcv-Frame:Data-21,22,23,24,25,26,27,28, ID:00000310  BP-Bit:1  ID-Index:----
***Rcv-Frame:Data-31,32,33,34,35,36,37,38, ID:00003377  BP-Bit:1  ID-Index:----
***Rcv-Frame:Data-41,42,43,44,45,46,47,48, ID:00007755  BP-Bit:1  ID-Index:----

***If you want demo again
***Enter your choice: 1 for ACC Bypass, 2 for ACC On
```

Fig 4. CAN message received successfully demo page

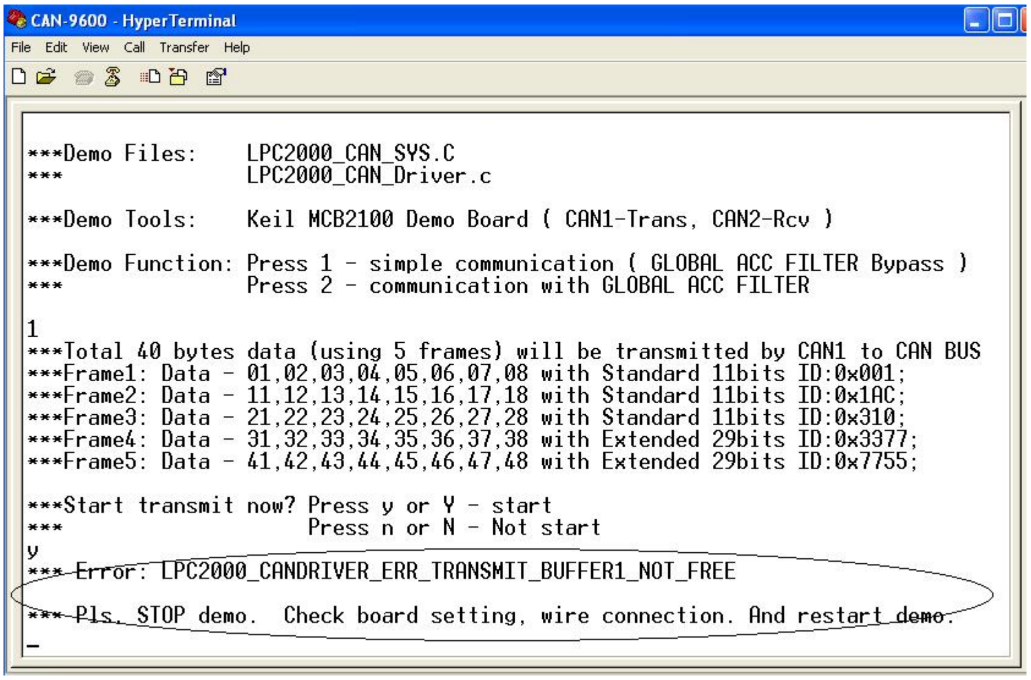
You will notice that the BP-Bit (ACC Bypass indicator) is set, and in this case, the ID-Index is meaningless.

If a CAN BUS error occurs, the demo code will display an error message.

There are two kinds of errors that will be demonstrated in this demo:

One kind of error(s) are CAN BUS status error(s). These errors are reported by function return value. You can simulate this error by disconnecting the wire between two CAN ports.

Error reported as shown in Fig 5:



```
***Demo Files:    LPC2000_CAN_SYS.C
***               LPC2000_CAN_Driver.c

***Demo Tools:    Keil MCB2100 Demo Board ( CAN1-Trans, CAN2-Rcv )

***Demo Function: Press 1 - simple communication ( GLOBAL ACC FILTER Bypass )
***               Press 2 - communication with GLOBAL ACC FILTER

1
***Total 40 bytes data (using 5 frames) will be transmitted by CAN1 to CAN BUS
***Frame1: Data - 01,02,03,04,05,06,07,08 with Standard 11bits ID:0x001;
***Frame2: Data - 11,12,13,14,15,16,17,18 with Standard 11bits ID:0x1AC;
***Frame3: Data - 21,22,23,24,25,26,27,28 with Standard 11bits ID:0x310;
***Frame4: Data - 31,32,33,34,35,36,37,38 with Extended 29bits ID:0x3377;
***Frame5: Data - 41,42,43,44,45,46,47,48 with Extended 29bits ID:0x7755;

***Start transmit now? Press y or Y - start
***                    Press n or N - Not start
y
*** Error: LPC2000_CANDRIVER_ERR_TRANSMIT_BUFFER1_NOT_FREE

*** Pls. STOP demo.  Check board setting, wire connection. And restart demo.
_
```

Fig 5. CAN BUS Error-1 demo page

Another kind of error(s) are CAN BUS communicating error(s). These errors are reported by CAN2 interrupt.

You can simulate this error by edit line258-259 in LPC2000_CAN_SYS.c.

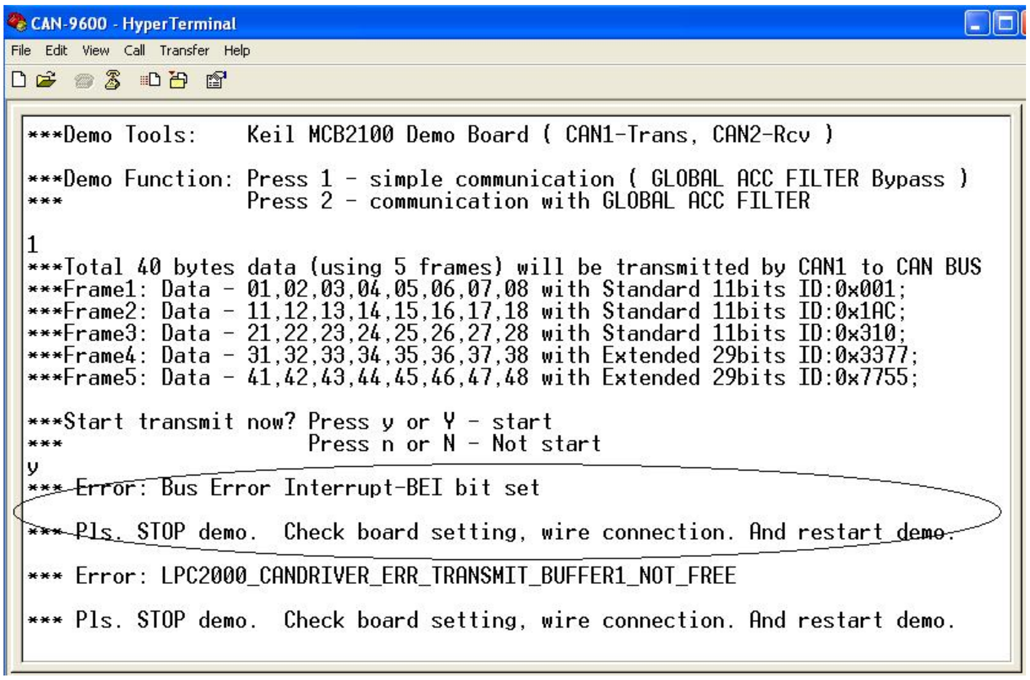
Forbidding Line258, enable Line259.

Setting:

PINSEL1 |= 0x00004000;   // P0.23-RD2, only enable receive for CAN2

Thus no ACK info will be sent by CAN2. That will result in an error.

Error reported as shown in Fig 6:



```
***Demo Tools:     Keil MCB2100 Demo Board ( CAN1-Trans, CAN2-Rcv )

***Demo Function:  Press 1 - simple communication ( GLOBAL ACC FILTER Bypass )
***                Press 2 - communication with GLOBAL ACC FILTER

1
***Total 40 bytes data (using 5 frames) will be transmitted by CAN1 to CAN BUS
***Frame1: Data - 01,02,03,04,05,06,07,08 with Standard 11bits ID:0x001;
***Frame2: Data - 11,12,13,14,15,16,17,18 with Standard 11bits ID:0x1AC;
***Frame3: Data - 21,22,23,24,25,26,27,28 with Standard 11bits ID:0x310;
***Frame4: Data - 31,32,33,34,35,36,37,38 with Extended 29bits ID:0x3377;
***Frame5: Data - 41,42,43,44,45,46,47,48 with Extended 29bits ID:0x7755;

***Start transmit now? Press y or Y - start
***                    Press n or N - Not start
y
*** Error: Bus Error Interrupt-BEI bit set

*** Pls. STOP demo.  Check board setting, wire connection. And restart demo.

*** Error: LPC2000_CANDRIVER_ERR_TRANSMIT_BUFFER1_NOT_FREE

*** Pls. STOP demo.  Check board setting, wire connection. And restart demo.
```

Fig 6. **CAN BUS Error-2 demo page**

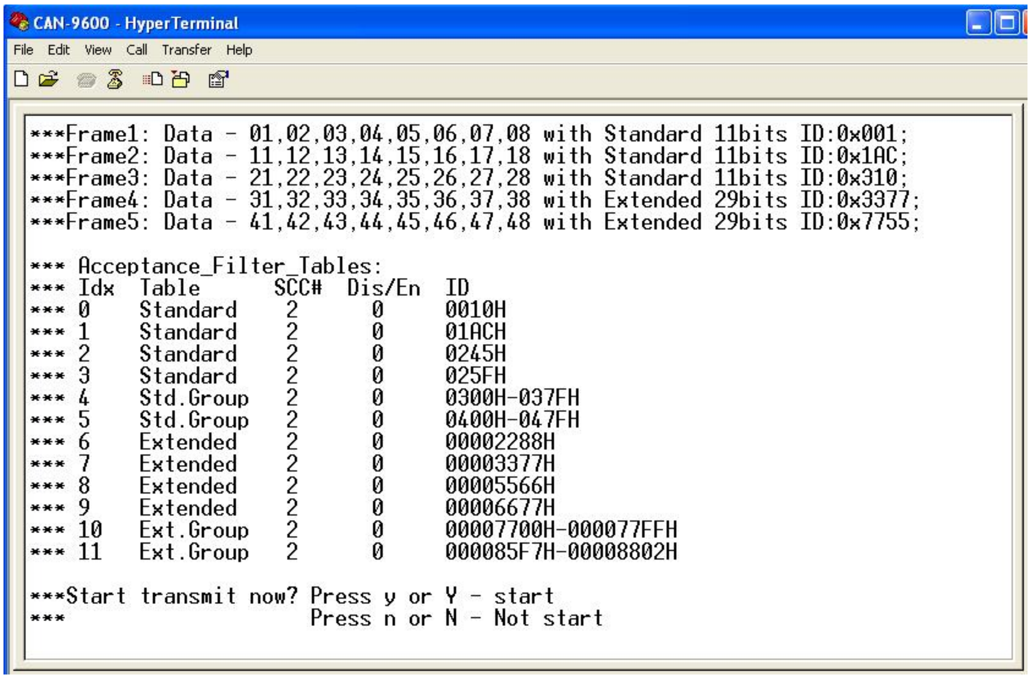
Note: in ACC ON mode, this error will not be reported, because the ID of the first CAN message does not match the defined ID in the Global Acceptance Filter Look-up table.

5.2.2 CAN BUS communicate with GAF enable

In this demo function, you need to choose the second function in Welcome Page.

It will demo CAN BUS communication with enabled GAF; GAF is set to ACC ON mode.

The CAN message send page not only displays sent messages, but also displays the Global Acceptance Filter Look-up table pre-defined in LPC2000_CAN_Driver.c.



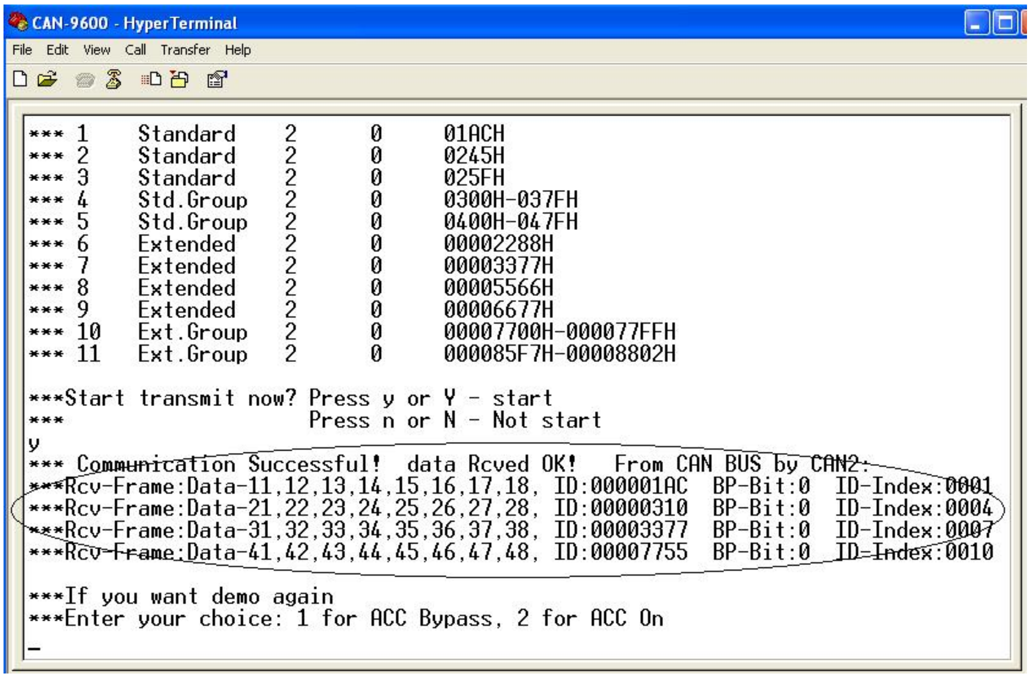
```
***Frame1: Data - 01,02,03,04,05,06,07,08 with Standard 11bits ID:0x001;
***Frame2: Data - 11,12,13,14,15,16,17,18 with Standard 11bits ID:0x1AC;
***Frame3: Data - 21,22,23,24,25,26,27,28 with Standard 11bits ID:0x310;
***Frame4: Data - 31,32,33,34,35,36,37,38 with Extended 29bits ID:0x3377;
***Frame5: Data - 41,42,43,44,45,46,47,48 with Extended 29bits ID:0x7755;

*** Acceptance_Filter_Tables:
*** Idx  Table      SCC#  Dis/En  ID
*** 0    Standard    2      0     0010H
*** 1    Standard    2      0     01ACH
*** 2    Standard    2      0     0245H
*** 3    Standard    2      0     025FH
*** 4    Std.Group   2      0     0300H-037FH
*** 5    Std.Group   2      0     0400H-047FH
*** 6    Extended    2      0     00002288H
*** 7    Extended    2      0     00003377H
*** 8    Extended    2      0     00005566H
*** 9    Extended    2      0     00006677H
*** 10   Ext.Group   2      0     00007700H-000077FFH
*** 11   Ext.Group   2      0     000085F7H-00008802H

***Start transmit now? Press y or Y - start
***                    Press n or N - Not start
```

Fig 7. CAN message prepared to be sent with ACC ON demo page

When you confirm transmission, Demo code will start transmit.

In this mode, only the data fitted with ID and SCC# in GAF Look-up table will be received if the CAN BUS works normally.

If received successfully, the demo code will display the received data as shown in Fig 8:



```
*** 1     Standard    2     0     01ACH
*** 2     Standard    2     0     0245H
*** 3     Standard    2     0     025FH
*** 4     Std.Group   2     0     0300H-037FH
*** 5     Std.Group   2     0     0400H-047FH
*** 6     Extended    2     0     00002288H
*** 7     Extended    2     0     00003377H
*** 8     Extended    2     0     00005566H
*** 9     Extended    2     0     00006677H
*** 10    Ext.Group   2     0     00007700H-000077FFH
*** 11    Ext.Group   2     0     000085F7H-00008802H

***Start transmit now? Press y or Y - start
***                    Press n or N - Not start
y
*** Communication Successful!  data Rcved OK!   From CAN BUS by CAN2:
***Rcv-Frame:Data-11,12,13,14,15,16,17,18, ID:000001AC  BP-Bit:0  ID-Index:0001
***Rcv-Frame:Data-21,22,23,24,25,26,27,28, ID:00000310  BP-Bit:0  ID-Index:0004
***Rcv-Frame:Data-31,32,33,34,35,36,37,38, ID:00003377  BP-Bit:0  ID-Index:0007
***Rcv-Frame:Data-41,42,43,44,45,46,47,48, ID:00007755  BP-Bit:0  ID-Index:0010

***If you want demo again
***Enter your choice: 1 for ACC Bypass, 2 for ACC On
_
```

Fig 8. CAN message received ok with ACC ON demo page

You will notice the BP-Bit (ACC Bypass indicator) is clear, and in this case, the ID-Index value is the zero-based number of the Look-up table RAM entry in which the Global Acceptance Filter matches the received Identifier.

5.3 Demo operation step

Operation step:

1. Setting MCB2100 jumpers, J1, J10, and J8 - OFF, Others - ON.
2. Connect both 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 connect pin-to-pin.

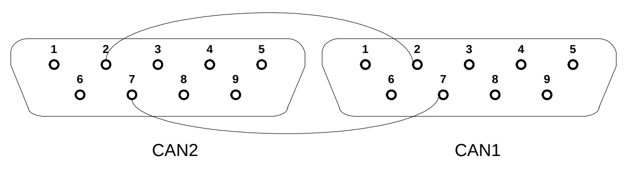


Fig 9. CAN1 and CAN2 port pins and connection description

3. Connect UART0 of MCB2100 to PC UART port you used, with UART cable.
4. Connect MCB2100 JTAG port to PC USB port with ULINK.
5. Connect DC power (above 6 V, the center hole provides positive voltage) to MCB2100.
6. Copy the folder <LPC2000_CAN_Driver_Demo> to your PC. Make sure that ADS1.2 is installed in your C:\Program Files\ARM\ADSV1_2
7. Open uVision3 in your PC (**Note:** don't double click to open demo project, otherwise it will open uVision2, if you have installed uVision2 on your PC).
8. Open demo project: LPC2000_CAN.Uv2.
9. Open one connection of Hyper Terminal on your PC. The UART port connection needs to be set as: Baud rate – 9600; Data bits – 8; Parity – None; Stop bit – 1; Flow control – None; For easy observation, in ASCII SETUP, check the Echo typed characters locally function for ASCII Sending.
10. Power On. **Press Reset key** on MCB2100, it will enable ULINK in uVision3.
11. Enter debug mode in uVision3.
12. Run the demo.

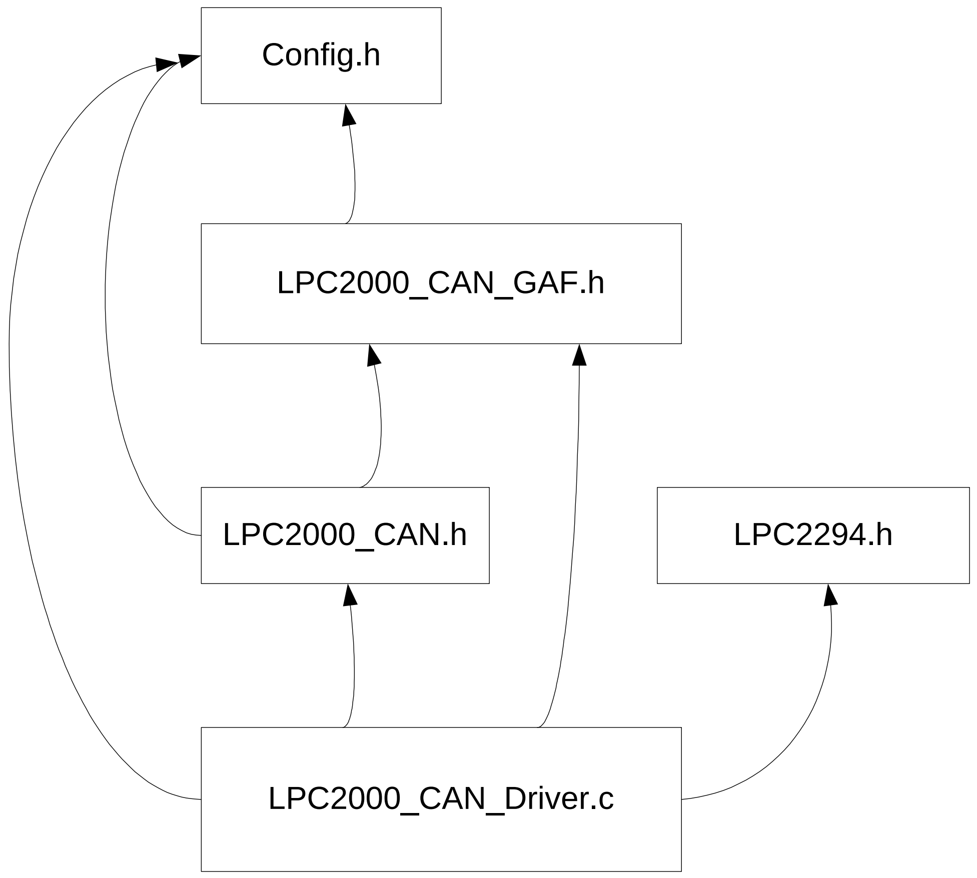
6. References

[1] CAN Controller HwAPI Driver for the SJA2020

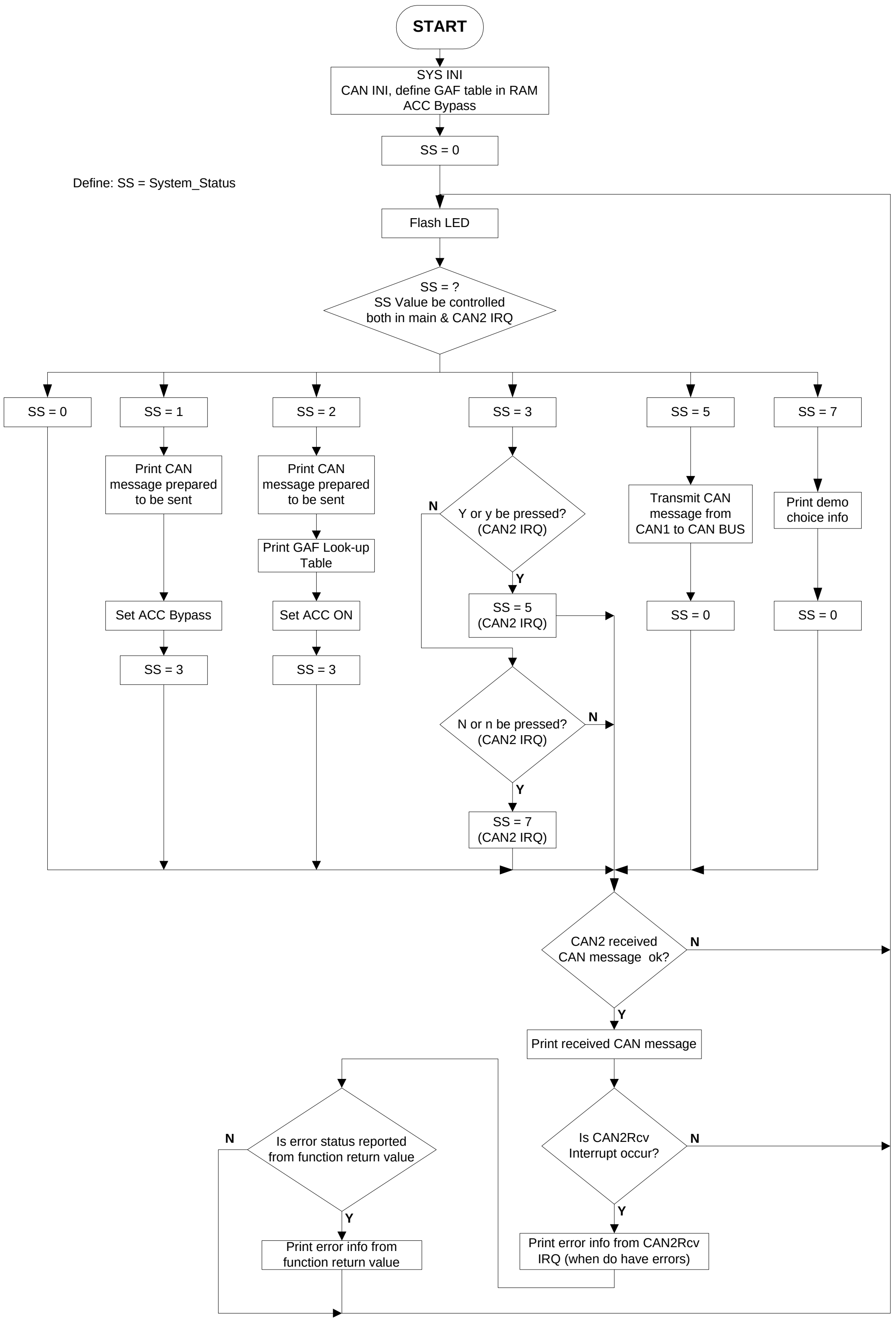
[2] LPC2294 Data Sheet – Version 03

[3] LPC2294 User Manual – Version 02

Appendix A Including dependence graph



Appendix B Software flowchart for demo


START
SYS INI
CAN INI, define GAF table in RAM
ACC Bypass
SS = 0
Define: SS = System_Status
Flash LED
SS = ?
SS Value be controlled both in main & CAN2 IRQ
SS = 0
SS = 1
SS = 2
SS = 3
SS = 5
SS = 7
Print CAN message prepared to be sent
Print CAN message prepared to be sent
Print GAF Look-up Table
Y or y be pressed? (CAN2 IRQ)
N
Y
Transmit CAN message from CAN1 to CAN BUS
Print demo choice info
Set ACC Bypass
Set ACC ON
SS = 5 (CAN2 IRQ)
SS = 0
SS = 0
SS = 3
SS = 3
N or n be pressed? (CAN2 IRQ)
N
Y
SS = 7 (CAN2 IRQ)
CAN2 received CAN message ok?
N
Y
Print received CAN message
Is error status reported from function return value
N
Y
Is CAN2Rcv Interrupt occur?
N
Y
Print error info from function return value
Print error info from CAN2Rcv IRQ (when do have errors)

7. 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 license 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.

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

8. Trademarks

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

9. Contents

1. Introduction 3
1.1 Introduction 3
1.2 Main features 3
1.3 More useful features 3
1.4 CAN BUS timing setting.....................................3
2. Introduction for CAN controller driver..............5
3. General CAN controller driver description........6
3.1 Software development support 6
3.2 CAN controller driver functionality overview........6
3.3 CAN controller driver files 7
3.4 Used data types..8
3.5 CAN Message Buffers data structure..................8
3.6 Global Acceptance Filter configuration 11
3.6.1 Guidelines for setting ID Look-up Table............13
3.7 Function return values 14
3.8 Pre-defined CAN Bit Timing values...................15
3.9 Pre-defined function arguments........................16
4. CAN driver functional description 17
4.1 Interface description of CAN driver functions....18
4.1.1 lpc2000CANdriver_CANInit...............................18
4.1.2 lpc2000CANdriver_SetCANMode 19
4.1.3 lpc2000CANdriver_SetACFMode 20
4.1.4 lpc2000CANdriver
_CertainTxBufTransmitMessage 21
4.1.5 lpc2000CANdriver_ReceiveMessageChx 22
4.1.6 lpc2000CANdriver_LoadAcceptanceFilter 23
4.1.7 Rcv_Data_Output_to_Screen 24
4.1.8 Print_Chars_to_Screen.....................................24
4.1.9 Print_4bits_to_Screen.......................................25
5. Demo description 26
5.1 Demo Tools..26
5.2 Demo function description................................26
5.2.1 Simple CAN BUS communication demo 26
5.2.2 CAN BUS communicate with GAF enable 32
5.3 Demo operation step..34
6. References..34
Appendix A Including dependence graph 35
Appendix B Software flowchart for demo..................36
7. Disclaimers 37
8. Trademarks...37
9. Contents...38

Date of release: 02 March 2006
Document number: AN10438_1

Published in The Netherlands