|  |
| --- |
|  |
| Smith Drive  Embedded Software |
|  |
|  |
| CAN LIBRARY FOR  TEXAS INSTRUMENTS TMS320F28335 Microcontroller |
|  |
|  |
| Version 0.0.1 |
|  |

|  |  |  |  |
| --- | --- | --- | --- |
| Function | Responsible | Approved | Date |
| Embedded Software Engineer | C Barlow |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |
| --- |
| **This document including without limitation the specifications for Project constitutes SEV Confidential Information and Background Intellectual Property. Do not use or disclose except to the extent permitted under the agreement between all members.** |
|  |
| **SMITH INTERNAL ONLY** |

Contents

[Contents 2](#_Toc349033845)

[Change Log 3](#_Toc349033846)

[1. Introduction 4](#_Toc349033847)

[2. Architecture Overview 4](#_Toc349033848)

[1.1 Execution principle 4](#_Toc349033849)

[1.2 eCAN Mailboxes 5](#_Toc349033850)

[1.3 Library Interface Structures 6](#_Toc349033851)

[1.4 State machine 7](#_Toc349033852)

[1.5 Library Functions 9](#_Toc349033853)

[1.5.1 Mailbox Configuration 9](#_Toc349033854)

[1.5.2 Updating the Mailbox State Machine 9](#_Toc349033855)

[1.5.3 Checking the State of a Mailbox 10](#_Toc349033856)

[1.5.4 Loading a Mailbox 10](#_Toc349033857)

[1.5.5 Transmitting from a Mailbox 11](#_Toc349033858)

[1.5.6 Reading from Mailbox 11](#_Toc349033859)

[3. References 12](#_Toc349033860)

Change Log

|  |  |  |  |
| --- | --- | --- | --- |
| Ver. | Date | Description | Initials |
| 0.0.1 | 18/02/2013 | Preliminary draft | CB |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

1. Introduction

The CAN Library operates in the library layer of the Smith Drive embedded software, and provides access to the CAN ports and mailboxes implemented in the eCAN area of the TI TMS320F28335 microcontroller.

Library functions are designed to be ‘one-shot’ functions, to allow periodical execution from the Smith Drive firmware’s Time-Triggered scheduler.

1. Architecture Overview

The CAN library itself features 3 layers that provide a logical separation between the embedded software and the TI TMS320F28335 eCAN mailboxes:

**Library functions** – Allow access to the library from the firmware tasks.

**State machine** – Controls the state of each message object in each eCAN port.

**Interface structures** – Allow safe access to the eCAN registers that control the eCAN ports and their mailboxes.

**Message exchange structures** - Provides a logical mapping between the raw CAN data and the variables contained within the message.

**Task Code**

**Message Exchange Structures**

**Interface Structures**

**State machine**

**eCAN mailboxes (onboard MCU)**

**Library functions**

Figure 1: CAN library software layers

## Execution principle

The library is designed in such a way that all code run from the task scheduler only needs to access the Library Functions. There are four ways in which the task code needs to interact with the CAN library:

* Mailbox configuration – Mailboxes are configured to be either transmit boxes (CAN\_TX) or receive boxes (CAN\_RX).
* State machine updating – a periodic task keeps the state of each mailbox up to date.
* Message reception handling – periodic tasks check the state of CAN\_RX mailboxes for received messages and read the data into the Message Exchange Structures.
* Message transmission handling – periodic tasks check the state of CAN\_TX mailboxes to ensure that there is no message pending transmission. Data is loaded from the message exchange structures into the mailbox, and a timer is used to initiate transmission at the appropriate time.

The library functions used to perform these operations are described in detail in 2.5.

## eCAN Mailboxes

The TMS320F28335 microcontroller contains an on-board CAN controller called eCAN, which has 2 CAN ports, CAN A and CAN B. Each CAN port has 32 mailboxes that can be configured to be transmit (Tx) boxes or receive (Rx) boxes. These boxes are used, along with a set of control and status registers, to transmit and receive messages over the CAN bus. [1]



Figure 2: eCAN Memory Map [1]

## Library Interface Structures

The Interface Structures take the eCAN Mailboxes and arrange them into logical arrays to enable indexed access to each mailbox and its associated registers. A mailbox is identified by CAN port (CANPORT\_A or CANPORT\_B) and mailbox number (0 – 31).

Data is loaded into, or read from a mailbox by means of an 8 – byte data array. CAN bytes are mapped to the data array index, ie CAN byte 0 = data[0], CAN byte 1 = data[1], and so on.

|  |  |
| --- | --- |
| **CAN\_Ports** | |
| CANPORT\_A | CANPORT\_B |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Control and Status Registers (ECAN\_REGS)** | **Message\_Objects** | | | | | | |
| 0 | 1 | 2 | …. | 29 | 30 | 31 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Mailbox State** | **Mailbox (MBOXn)** | **Local Acceptance Mask  (LAMn)** | **Message Object Timeout (MOTOn)** | **Message Object Timestamp (MOTSn)** |

|  |  |  |  |
| --- | --- | --- | --- |
| **CAN ID (MSGID)** | **Message Control (MSGCTRL)** | **Message raw data** | |
| Uint32 data[0] | Uint32 data[1] |

Figure 3: Library Interface Structures

## Message Exchange Structures

The Message Exchange Structures are a central location to store CAN variable values and their mapping to the CAN message raw data.

A CAN Rx Message is defined by the following elements:

* Variables bitfield, e.g. struct CAN\_DATA\_VARS\_RX\_7AB
  + Defines the size, byte and bit position of the variables in the CAN message
  + Byte variables defined in the order:

Uint16 B3:8;

Uint16 B2:8;

Uint16 B1:8;

Uint16 B0:8;

Uint16 B7:8;

Uint16 B6:8;

Uint16 B5:8;

Uint16 B4:8;

* + bit variables for each Byte defined in the order:

Uint16 b0:1;

Uint16 b1:1;

Uint16 b2:1;

Uint16 b3:1;

Uint16 b4:1;

Uint16 b5:1;

Uint16 b6:1;

Uint16 b7:1;

* + Use combinations of the above to build up the CAN message:

**struct** CAN\_DATA\_VARS\_TX\_7AB{

/\*byte variable examples (bytes 3 to 1) \*/

Uint16 DB3:8;

Uint16 DB2:8;

Uint16 DB1:8;

/\*bit variable examples (byte 0) \*/

Uint16 Db0\_0:1;

Uint16 Db0\_1:1;

Uint16 Db0\_2:1;

Uint16 Db0\_3:1;

Uint16 Db0\_4:1;

Uint16 Db0\_5:1;

Uint16 Db0\_6:1;

Uint16 Db0\_7:1;

/\* 16 bit variable examples \*/

Uint16 DB7\_6; /\* Low byte: DB7, High byte: DB6 \*/

Uint16 DB5\_4;

}

* Data union, e.g. union CAN\_DATA\_RX\_7AB
  + Maps the variables bitfield to a Uint32 rawData[2] array
* Message structure, e.g. canRxMessage\_t canMessage\_Rx\_7AB
  + Assigns CAN ID and DLC fields to the CAN message
  + Allows global access to the rawData[2] array
* Message array
  + Used to assign CAN message to a mailbox (array index represents the mailbox number that will handle this message)
  + ***It is important to handle the mailbox configuration such that Rx and Tx messages do not occupy the same mailbox.***

***e.g:***

**mailBox** = i + numTxCANMsgs;

configureMailbox(CANPORT\_A, **mailBox**, CAN\_RX, ID\_STD, CAN\_RxMessages[i]->canID, CAN\_RxMessages[i]->canDLC);

In addition to the above, CAN Tx message structures contain the following:

* Message offset
  + To be used as a decrementing counter to control message transmission timing.
  + Initial value used to control the first instance of the message
* Message period
  + The value used to reload the message offset field, thus controlling the periodic timing of the message.

## State machine

The state machine enables the software to keep track of what each mailbox is doing. A mailbox state can be any one of the following:

Table 1: Mailbox state descriptions

|  |  |
| --- | --- |
| State | Description |
| DISABLED | Mailbox has not been configured for any operation |
| TX\_FREE | Mailbox is ready for next message for transmission. |
| TX\_PENDING | Message transmitted from the mailbox - waiting for acknowledgement |
| TX\_SENT | Message was sent successfully, waiting for flags to clear |
| TX\_ERR | Error occurred during message transmission |
| RX\_FREE | Mailbox is empty – waiting to receive CAN message |
| RX\_PENDING | CAN message has arrived in mailbox – waiting to be read |
| RX\_READ | CAN message has been read – waiting for flags to clear |

1

7

2

3

5

4

6

8

9

10

Figure 4: Mailbox state diagram

Table 2: State transition table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Transition Description | In scheduler | CAN library | eCAN |
| 1 | Configure mailbox  for Tx | **configureMailbox (port, mb, CAN\_TX,…, …, …)** |  |  |
|  | CANME = 1, CANMD = 0 |
| 2 | Load data into mailbox | **loadTxMailbox(port, mb, …, …)** |  |  |
| Send CAN message | **commitSendMailbox(port, mb)** |  |
|  | CANTRS = 1 |
| 3 | Successful message transmission |  |  | CANTRS = 0, CANTA = 1 |
| Clear ack flag | CANTA = 1 |  |
| 4 | Ack flag cleared successfully |  |  | CANTA = 0 |
| 5 | Error with transmission |  |  | CANTRS = 0, CANTA = 0 |
| 6 | To be defined\* |  |  |  |
| 7 | Configure mailbox  for Rx | **configureMailbox (port, mb, CAN\_RX,…, …, …)** |  |  |
|  | CANME = 1, CANMD = 1 |
| 8 | Message in mailbox |  |  | CANRMP = 1 |
| Clear message pending flag |  | CANRMP = 1 (clears flag) |  |
| 9 | Message pending flag cleared successfully |  |  | CANRMP = 0 |
| 10 | Read data from mailbox\*\* | **readRxMailbox(port, mb, …, …)** |  |  |

port = (CANPORT\_A or CANPORT\_B) mb = mailbox number

\*In the current implementation of the code, the TX\_ERR state automatically resets the state to TX\_FREE, with no additional error handling. It acts as a placeholder for future error handling logic if implementation is necessary.

\*\*If the eCAN detects an error during reading it will reset the CANRMP flag immediately after the read. If this occurs, the state machine will treat this as a normal message arrival and the message reception task(s) will have to re-read the message [1].

## Library Functions

This section describes the library functions that make the State Machine and Interface Structures accessible to the task code.

### Mailbox Configuration

**int16 configureMailbox(char port, char mbNum,   
mailboxDirection\_t direction, char IDE, Uint32 canID, Uint16 dataLength);**

This is used to configure and enable a mailbox during initialisation. Can be executed once per mailbox (there is currently no functionality to disable a mailbox once it has been enabled.

|  |  |  |
| --- | --- | --- |
| Arguments | Description | Values |
| **char port** | ID of the CAN port to configure | CANPORT\_A |
| CANPORT\_B |
| **char mbNum** | ID number of the mailbox to configure | 0 to 31 |
| **mailboxDirection\_t direction** | Direction to configure the mailbox to | CAN\_TX |
| CAN\_RX |
| **char IDE** | CAN ID type for the mailbox | IDE\_STD |
| IDE\_EXT |
| **Uint32 canID** | CAN ID to be associated with the mailbox. For a CAN\_TX box, this is the ID that the message will be transmitted under, for a CAN\_RX box, this is the CAN ID that will be accepted into the mailbox. | IDE\_STD:  0x000 to 0x7FF  IDE\_EXT:  0x00000000 to 0x1FFFFFFF |
| **Uint16 dataLength** | The number of bytes to be transmitted from or received by the mailbox. | 1 - 8 |

|  |  |
| --- | --- |
| Return Value | Description |
| **0 (zero)** | All cases |
|

### Updating the Mailbox State Machine

**void updateMailboxes(char port);**

This function iterates through all mailboxes associated with a CAN port and updates the state machine according to the eCAN control and status registers. All mailboxes are checked whether they have been configured or not to allow for predictable execution time.

NOTE: As one check and update is made per mailbox per execution, it is advised to call this function at least twice as frequently as the tasks controlling message transmission and reception, to allow for the state machine to progress from the PENDING to FREE states in time.

|  |  |  |
| --- | --- | --- |
| Arguments | Description | Values |
| **char port** | ID of the CAN port to update | CANPORT\_A |
| CANPORT\_B |

|  |  |
| --- | --- |
| Return Value | Description |
| N/A | Void function |

### Checking the State of a Mailbox

**messageObjectStates\_t checkMailboxState(char port, char mbNum);**

Allows task functions to check the state of a given mailbox in order to keep track of the current state.

Due to the nature of the Time-Triggered scheduler, we know that the state of the port is not going to change until the next updateMailboxes() call.

|  |  |  |
| --- | --- | --- |
| Arguments | Description | Values |
| **char port** | ID of the CAN port to check | CANPORT\_A |
| CANPORT\_B |
| **char mbNum** | ID number of the mailbox to check | 0 to 31 |

|  |  |
| --- | --- |
| Return Value | Description |
| DISABLED | Mailbox state (see 2.3). |
| TX\_FREE |
| TX\_PENDING |
| TX\_SENT |
| TX\_ERR |
| RX\_FREE |
| RX\_PENDING |
| RX\_READ |

### Loading a Mailbox

**int16 loadTxMailbox(char port, char mbNum, char data[]);**

This function is used to load data into a mailbox ready for transmission. At this point the message isn’t transmitted, so this function can be used to keep the mailbox contents up-to-date, allowing another task to control the transmission timing.

|  |  |  |
| --- | --- | --- |
| Arguments | Description | Values |
| **char port** | ID of the CAN port to use | CANPORT\_A |
| CANPORT\_B |
| **char mbNum** | ID number of the mailbox to load | 0 to 31 |
| **Uint32 data[]** | The data array of 2 32-bit unsigned integers to loaded into the mailbox  data[0] = CAN Bytes 0 – 3,  data[1] = CAN Bytes 4 - 7 | Pointer to array |

|  |  |
| --- | --- |
| Return Value | Description |
| **0 to 8** | Number of bytes successfully loaded into the mailbox |
|
| **-1** | The specified mailbox was not configured as CAN\_TX |

### Transmitting from a Mailbox

**int16 commitSendMailbox(char port, char mbNum);**

This function commits a mailbox for transmission. (Actual transmission is handled by the eCAN module itself).

|  |  |  |
| --- | --- | --- |
| Arguments | Description | Values |
| **char port** | ID of the CAN port to use | CANPORT\_A |
| CANPORT\_B |
| **char mbNum** | ID number of the mailbox to transmit from | 0 to 31 |

|  |  |
| --- | --- |
| Return Value | Description |
| **0 (zero)** | All cases |
|

### Reading from Mailbox

**int16 readRxMailbox(char port, char mbNum, Uint32 data[]);**

This function reads pending data from an Rx mailbox into a data array. The data should be extracted from the data array using unions, eg:

**struct** CAN\_DATA\_BYTES\_RX0{

Uint16 DB3:8;

Uint16 DB2:8;

Uint16 DB1:8;

Uint16 DB0:8;

Uint16 DB7:8;

Uint16 DB6:8;

Uint16 DB5:8;

Uint16 DB4:8;

};

**union** CAN\_DATA\_RX0{

Uint32 data[2];

**struct** CAN\_DATA\_BYTES\_RX0 dataBytes;

};

|  |  |  |
| --- | --- | --- |
| Arguments | Description | Values |
| **char port** | ID of the CAN port to use | CANPORT\_A |
| CANPORT\_B |
| **char mbNum** | ID number of the mailbox to read from | 0 to 31 |
| **Uint32 data[]** | The data array of 2 32-bits to read into data[0] = CAN Bytes 0 – 3,  data[1] = CAN Bytes 4 - 7 | Pointer to array |

|  |  |
| --- | --- |
| Return Value | Description |
| **0 to 8** | Number of bytes read from the mailbox |
|
| **-1** | The specified mailbox was not configured as CAN\_RX |

1. References

|  |  |
| --- | --- |
| [1] | Texas Instruments, TMS320F2833x, 2823x Enhanced Controller Area Network (eCAN) Reference Guide, Dallas, Texas, 2009. |