CANFD
Software Design

Version 0.6

Business Unit: Radar

Project Name: mmWave SDK

TABLE OF CONTENTS

TABLE OF FIGURES

Revision Control

Author Name	Description	Version	Date
Jyothi Pandit	Initial version	0.5	12/20/2016
Jyothi Pandit	Added section software workarounds	0.6	04/08/2018

1. Purpose

This document describes the design and implementation considerations for CANFD driver software and its features. It is intended for internal (development team mainly) audience.

2. Functional Overview

The CANFD driver exposes the following hardware functionality of the MCAN IP present in AR16xx SOC. The DCAN IP supports CAN protocol version 2.0 part A, B.

- Enable and configure the CAN IP.
- Create the receive and transmit message objects.
- Setup standard and extended filters.
- Transmit and read the classic CAN data over the CAN interface.
- Transmit and read the CAN FD data over the CAN interface.
- Register for error and status interrupts
- Handle parity error, ECC and ECC diagnostics.

3. Assumptions

1. The driver registers for the error and status interrupts and calls the callback API registered by the application. It does not take any action to correct the errors. It is up to the application to take appropriate action. Helper APIs are provided to facilitate the same.
2. Following features are yet to be implemented:
    a. DMA accesses.
    b. Power down/wakeup

4. Definitions, Abbreviations, Acronyms

Term	Definition
CAN	**Controller Area Network**
CANFD	**Controller Area Network with Flexible Data-rate**
MCAN	**Modular Controller Area Network**
DMA	**Direct Memory Access**
ECC	**Error checking and Correction**
LEC	**Last error code**

5. References

1. Functional Requirement: https://jira.itg.ti.com/browse/MMWSDK-176

2. MCAN IP spec:

3. https://sps08.itg.ti.com/sites/autoradar/Shared%20Documents/AR1xxx%20PG1.0%20Architecture/AR1642_AutoRadar_ASD_V1.1.pdf

4. AR1xxx Architecture Spec: https://sps08.itg.ti.com/sites/autoradar/Shared%20Documents/AR1xxx%20PG1.0%20Architecture/AR1642_AutoRadar_ASD_V1.1.pdf

5. MCAN register map

   http://www-open.india.ti.com/~wdccm/altius/verif/docs/magillem_db/html/MSS_HTML/MSS_MCAN_CFG.html

   http://www-open.india.ti.com/~wdccm/altius/verif/docs/magillem_db/html/MSS_HTML/MSS_MCAN_MSGMEM.html

6. Design

6.1 System Architecture

The CANFD Driver provides a mechanism to transfer data between 2 CAN peripherals.

6.2 Platforms supported

The driver is supported on R4F architecture and AR16XX SoCs.

The MCAN performs CAN protocol communication according to ISO 11898-1 (identical to Bosch CAN protocol specification 2.0 A, B). The bit rate can be programmed to values up to 10 Mbps. Additional transceiver hardware is required for the connection to the physical layer (CAN bus).

6.3 External Interfaces (API)

Please refer to the doxygen documentation under the canfd/docs directory for information on the internal and external APIs/structures.

Following sub-sections provide information on some of these APIs and their implementation.

6.3.1 Initialization

This function must be called once per system and before any other CANFD driver APIs. The function performs the following:

1. Malloc the driver state context.
2. Store the hardware SoC related attributes.
3. Reset the MCAN module.
4. Register interrupt line 0 interrupts such as Tx, Rx, error and status interrupts if enabled.
5. Register interrupt line 1 for ECC interrupts if enabled.
6. Configure the MCAN module with the specified configuration.
7. Configure the message RAM.

Return a handle to the driver instance.

6.3.2 Set bit timings

This function performs the following:

1. Reset the MCAN module.
2. Configure the bit timing parameters.
3. Set the mode to Normal so the module can be operational.

This function can be called multiple times to reconfigure the timing parameters.

6.3.3 Deinit

This function performs the following:

1. Close and cleanup the resources used by the CANFD driver.
2. The CANFD handle is no longer valid and should not be used after the API has been invoked.

6.3.4 Create/Delete message objects

Message objects are used to transmit or receive data over the CAN peripheral. The create API does the following:

1. Malloc the message object context.
2. Configure the transmit message object, allocate transmit buffer and register for transmit complete interrupts.
3. Configure the receive message object, allocate receive buffer and register for receive interrupt.
4. Track the message object handles for bookkeeping.

Return a handle to the message object. Any further operations involving message objects must use the handle.

The delete API does the following:

1. Cleanup the message object context.
2. Update the message object handles for bookkeeping.
3. The message object handle is no longer valid and should not be used after the API has been invoked.

6.3.5 Transmit

Max of 64 bytes of data can be transmitted using the message object. The tx buffer in message RAM is used to transfer the data. The application will be notified of a successful transmission via a callback if the transmit complete interrupt was enabled initialization.

6.3.6 Receive

If the receive interrupts was enabled during initialization, the driver will notify the application when the data has arrived via the callback. The application needs to call the CANFD_getData() function to read the received data. Max of 64 bytes of data can be received.

6.3.7 Error and Status change Handling

The application can monitor the parity error, Bus off error and Protocol error by enabling the error interrupts. The driver will call the registered callback function to indicate which error and status fields caused the interrupt. It is up to the application to take appropriate action. More details are available on doxygen documentation.

6.3.8 GET/SET options

Helper APIs to get and set various statistics, error counters, enabling loopback, ECC, ECC diagnostics, have been provided. More details are available on doxygen documentation.

6.4 Error Handling

Every external API does parameter checking and returns a valid error code on error detection.

6.5 Driver Resources

CANFD driver uses OSAL malloc API to allocate memory of sizes:

- sizeof(CANFD_DriverMCB)
- sizeof(CANFD_MessageObject) for every message object created.

6.6 Application programming sequence

Please refer to the doxygen documentation and CANFD driver's unit test code for more details.

6.7 Software workarounds

CANFD driver notifies the application via callback APIs to notify when a transmission has successfully occurred or a pending transmission has been successfully canceled. During the interrupt processing, the driver reads the buffer transmission occurred register or the buffer cancellation finished register to check the status of the pending request. These registers are read only registers and do not clear on read. Hence a software workaround is added to keep track of pending transmit requests. The callback function is now called only if the software tracked requests were pending.

7. Standards, Conventions and Procedures

7.1 Documentation Standards

The driver's software is documented using doxygen. Test code is not documented with doxygen because of future restructuring for MCPI test framework compliance.

7.2 Coding Standards

MCPI coding standards are used:

https://mcu-twiki.design.ti.com/bin/view/MCU/OneMCUSW/CodingStandard

7.3 Software development tools

Refer to the mmWave SDK release notes for details on the software needed for compilation and debugging of this driver.

7.4 Safety Standards

MISRA-C compliance is under study and will be implemented in future as part of the mmWave SDK requirement.

8. Appendix