Can driver Documentation

Release 0.8.0

ANSSI

Apr 28, 2020

TABLE OF CONTENTS

This library is an implementation of the STM32F4 CANx device driver.

It provide an abstraction of the CAN device interactions through high level API in order to interact with CAN buses.

The supported CAN standard version is CAN 2.0 A and B. This driver support the various debugging modes of the CAN device, including loop mode, receive only mode, etc.

CHAPTER

ONE

ABOUT THE CAN DRIVER

1.1 Principles

1.1.1 About CAN protocol

CAN (Controller Area Network) is a widely used, robust, serial communication bus which allows a set of MCU to communicate with each others. CAN buses are mostly used in vehicular systems. CAN is a message based protocol broadcasted on a shared communication link. CAN protocol handle priority based on device identifier.

Most of the time, devices use function-oriented protocols on the top of the CAN protocol, such as OBD-II.

CHAPTER

TWO

ABOUT THE CAN DRIVER API

2.1 Initializing the flash driver

Initializing the CAN driver is done with the following API:

```
#include "libcan.h"

mbed_error_t can_declare(__inout can_context_t *ctx);
mbed_error_t can_initialize(__inout can_context_t *ctx);
```

This two functions use th can_context_t structure to hold the CAN context. This context must be keeped by the upper layer and passed to all CAN driver functions. This permits to keep the libcan reentrant and allows the usage of multiple contexts by the same application.

The CAN driver declaration must be executed before the end of the task initialization phase (see EwoK kernel API). This function declare the device to the kernel, requesting an access to it. The only required field needed in the context is the CAN identifier (*id* field) which specify which CAN device is to be used.

At device initialization time, other fields are required:

- **CAN mode, which may be:**
  - normal (standard CAN interaction)
  - silent (transmission without reception)
  - loopback (all messages sent are received, no message is sent on the CAN bus
- CAN access mode, which can be poll mode (no interrupt) or interrupt based
- Time trigger activation, which mark CAN messages header with local timestamping
- auto bus offload management (dis)enable, handling CAN bus offloading
- auto wakeup (dis)enable, which allow sleep mode and wakeup mode switching on CAN message reception
- auto message retransmission (dis)enable, which automatically resent messages that were not correctly transmitted the first time

2.2 Starting and stopping the CAN device

The CAN device is configured in a specific mode, named INIT mode. Before starting to receive or send CAN messages, the CAN device must be started explicitely. This is the goal of the following API:

```
mbed_error_t can_start(__inout can_context_t *ctx);
```

It is also possible to stop the CAN device. No more message is received after this event and while *can_start()* is not called again. This is done using:

```
mbed_error_t can_stop(__inout can_context_t *ctx);
```

2.3 Sending and receiving messages

Sending and receiving CAN messages is done using the following API:

```
mbed_error_t can_xmit(const __in  can_context_t *ctx,
                            __in  can_header_t  *header,
                            __in  can_data_t    *data,
                           __out can_mbox_t    *mbox);

mbed_error_t can_is_txmsg_pending(const __in  can_context_t *ctx,
                                        __in  can_mbox_t mbox,
                                        __out bool *status);

mbed_error_t can_receive(const __in  can_context_t *ctx,
                         const __in  can_fifo_t     fifo,
                               __out can_header_t  *header,
                               __out can_data_t    *data);
```