# ssCAN
# User's Manual

Version 1.0
Revised February 29th, 2012
Created by the CAN Experts!

Simma Software

# ssCAN Device Driver License

# TABLE OF CONTENTS

# Chapter 1

# Introduction

ssCAN is an extreme performance CAN bus device driver with low interrupt latency and low CPU overhead, written in ANSI C. ssCAN adheres to both the ISO 11898 specification and to the software development best practices described in the MISRA C guidelines.

ssCAN is a modularized design with an emphasis on software readability and performance. ssCAN is easy to understand and target specific allowing high-level CAN protocols to be completely platform independent. ssCAN has been shown to be up to 800% faster and 80% smaller than other commercially available CAN device drivers.

ssCAN implements the data link layer in accordance to ISO 11898-1, -2, -3, and -5.

| Filenames | File Description |
|---|---|
| can.c | Core source file for ssCAN. Do not modify. |
| can.h | Core header file for ssCAN. Do not modify. |

**Table 1-1: ssCAN files**

# Chapter 2

# Integration of ssCAN

This chapter describes how to integrate ssCAN into your application. After this is complete, you will be able to receive and transmit CAN data frames over a CAN bus. For implementation details, please see the chapters covering the CAN API.

## Integration Steps:

1. Depending on your target, CAN related interrupts may need to be added to your microcontrollers vector table.

2. Configure your CAN baud rate settings by selecting one of the baud rate configurations in can.h

3. Depending on your target, enable corresponding CAN_H and CAN_L GPIO pins for CAN operation.

4. Enable clock to CAN peripheral.

5. Before using any of the modules features, make sure the CAN driver has been initialized by calling can_init(). Typically it is called shortly after power-on reset and before the application is started.

# Chapter 3

# CAN Hardware Abstraction Layer

The hardware abstraction layer (HAL) is a software module that provides functions for receiving and transmitting controller area network (CAN) data frames. Because CAN peripherals typically differ from one microcontroller to another, this module is responsible for encompassing all platform depended aspects of CAN communications.

The HAL contains three functions that are responsible for initializing the CAN hardware and handling buffered reception and transmission of CAN frames.

| Function Prototype | Function Description |
|---|---|
| void can_init ( void ) | Initializes CAN hardware |
| uint8_t can_rx ( can_t *frame ) | Receives CAN frame (buffered I/O) |
| uint8_t can_tx ( can_t *frame ) | Transmits CAN frame (buffered I/O) |

**Table 3-1: HAL functions**

## 3.1 Data Type Definitions

### Data type:

can_t

### Description:

can_t is a data type used to store CAN frames. It contains the CAN frame identifier, the CAN frame data, and the size of data. NOTE: If the most significant bit of id (i.e. bit 31) is set, it indicates an extended CAN frame, else it indicates a standard CAN frame.

### Definition:

```
typedef struct {

  uint32_t id;
  uint8_t buf[8];
  uint8_t buf_len;

} can_t;
```

## 3.2 Function APIs

# can_init

### Function Prototype:

```
void can_init( void );
```

### Description:

can_init initializes the CAN peripheral for reception and transmission of CAN frames at a network speed configured in can.h. Any external hardware that needs to be initialized, for example enabling a CAN transceiver, can be done inside of can_init.

High-level CAN protocols such as SAE J1939 and ISO 15765 recommend a sample point as close to 0.875 as possible, but never exceeding it. See J1939/11 and ISO 15765 for additional bit timing and sample point information.

### Parameters:

void

### Return Value:

void

# can_rx

## Function Prototype:

uint8_t can_rx ( can_t *frame );

## Description:

can_rx checks to see if there is a CAN data frame available in the receive buffer. If one is available, it is copied into the can_t structure which is pointed to by frame. If the most significant bit of frame->id (i.e. bit 31) is set, it indicates an extended CAN frame, else it indicates a standard CAN frame.

## Parameters:

frame: Points to memory where the received CAN frame should be stored.

## Return Value:

1: No CAN frame was read from the receive buffer.
0: A CAN frame was successfully read from the receive buffer.

# can_tx

### Function Prototype:

```
uint8_t can_tx ( can_t *frame );
```

### Description:

If memory is available inside of the transmit buffer, can_tx copies the memory pointed to by frame to the transmit buffer. If transmission of CAN frames is not currently in progress, then it will be initiated. If the most significant bit of frame->id (i.e. bit 31) is set, it indicates an extended CAN frame, else it indicates a standard CAN frame.

### Parameters:

frame: Points to the CAN frame that should be copied to the transmit buffer.

### Return Value:

1: No CAN frame was written to the transmit buffer.
0: The CAN frame was successfully written to the transmit buffer.

# Chapter 4
# Configuration

This chapter describes all configurable items of the ssCAN module. All of these configurations are defined at the top of can.c or can.h.

## Receive Buffer Count

This configuration defines how many incoming CAN data frames can be held in the receive FIFO.

```
#define CAN_RX_BUF_SIZE          10
```

## Transmit Buffer Size

This configuration defines how many incoming CAN data frames can be held in the transmit FIFO.

```
#define CAN_TX_BUF_SIZE          6
```

# Chapter 5
# Examples

This chapter gives examples of how to receive and receive CAN data frames.

## 5.1 Transmit CAN Frame Example:

```
void
can_app_tx ( void )
{
	can_t frame;

	frame.id = 0x55;
	frame.buf_len = 1;
	frame.buf[0] = 0x7F;

	/* transmit a CAN frame */
	if( can_tx(&frame) == 0 )
	  printf("success\n");
	else
	  printf("failure\n");
}
```

## 5.2 Receive CAN Frame Example:

```
void
can_rx_example ( void )
{
        can_t  frame;

        /* receive a CAN frame */
        if( can_rx(&frame) == 0 )
          printf("success, CAN ID is %d\n", frame.id);
        else
          printf("failure\n");
}
```