

CAN Board C & Assembly Strategy

Introduction

This document will suggest basic strategies for creating 'C' and Assembly code for the CAN E-Block. Because CAN is more complex than many other communication protocols and because this E-Block can be used with a variety of upstream boards (e.g. PICmicro multi-programmer board), this document will not provide all of the information required. See the "further reading" section for more complete reference information.

CAN was originally developed to provide a robust communication system for automotive applications and has been widely adopted in this area, but it is also being used in other applications where fast, error-free system-to-system communication is required.

In a CAN system there can be any number of 'nodes', each of which can send and receive CAN 'messages'. Each node must continually monitor the CAN bus and react to any messages intended for that node. The node can also send its own messages, but must only do so when there is a period of inactivity on the bus. There is extensive error and collision detection, which means that CAN is a very robust protocol.

At the heart of the CAN E-Block is a Microchip MCP2515 stand-alone CAN controller, and a MCP2551 CAN transceiver. An upstream E-Block (e.g. PICmicro multi-programmer) is required to initialise the CAN controller and to initiate and react to CAN messages.

Communication between the CAN controller and the upstream device is performed by SPI, so a working knowledge of that protocol is also required. The strategy notes below assume that the user can perform chip-to-chip communications using SPI.

Implementing a strategy

The following strategy is specific for a PICmicro microcontroller, but should be adaptable to any upstream device.

SPI interface

The CAN controller has over 100 addressable registers, each of which is readable and writeable via SPI. All aspects of CAN communication are controlled by these registers, so implementing SPI communication will be at the heart of your strategy. There are a number of SPI commands that can be sent to the CAN controller, e.g. to reset the device and to read and write the specific CAN registers. Details can be found in the MCP2515 datasheet.

Initialisation of the CAN controller

Probably the first thing to do is send a "reset" command to the CAN controller - this will make sure the controller is in a known state.

Initialisation of the CAN controller is probably where most of the work is. Most of the registers concerned with initialisation can only be written to when the CAN controller is in "configuration" mode. Once initialisation has been complete, the controller needs to be placed into "normal" mode. There are a number of other modes of operation, but a typical CAN application will not use these.

Setting up the CAN bus timing is achieved by writing appropriate values to the CNF1, CNF2 and CNF3 registers. See the relevant section in the datasheet to calculate the values required (the CAN E-Block uses a fixed 20MHz crystal).

There are 3 switches and 2 LED's on the CAN E-Block that can be configured as general-purpose i/o lines, or be used as message-received indicators (the LED's) and request-to-send inputs (the switches). The TXRTSCTRL and BFPCTRL registers control their behaviour.

The CAN controller has an interrupt output which signals when an interrupt has occurred. The events which cause an interrupt can be selected by appropriately configuring the CANINTE register. For simple applications, an interrupt should probably only be triggered when an error occurs or when an appropriate message has been received.

Finally, the receive masks and filters need to be set-up. We could set the controller up to interrupt when any CAN message has been received, and then use our controller firmware to work out which message has been received and react appropriately. A much better idea is to set the filters and masks up so that only the messages our specific mode wants are received. Details on setting up these masks and filters can be found in the MCP2515 datasheet.

Due to the large number of registers that need to be initialised, a good way of performing this initialisation in assembly is to use the "write" SPI command to write to every register by using a table-read routine (see the example in the AN215 datasheet).

Reacting to an incoming message

If the interrupts, masks and filters have been set up appropriately, we will be informed of an acceptable incoming message by a change of state on the INT pin (it will be driven low when an interrupt occurs). This implies that either the INT pin must be continually polled or it must be connected to a interruptible input on the PICmicro.

Once an interrupt has occurred, the source of the interrupt needs to be found out by reading the CANINTF register. If it indicates an error, then your firmware must take appropriate action to recover from the error state. If a message has been received, one of the RXnIF flags will be set indicating which message buffer has been filled. Your firmware will now know which receive buffer contains the message. The details about the message can be read by reading the appropriate buffer's registers.

Your firmware must reset the appropriate flag in the CANINTF register once it has dealt with that specific interrupt using the "bit modify" SPI message. It must do this so that other messages can be received.

Sending a message

To send a CAN message, the message itself needs to be written to one of the 3 transmit buffers. A CAN message consists of an identifier (this should be a unique number representing this specific message) and up to 8 additional bytes of data. The identifier can be standard (11-bit) or extended (29-bit), but for simple applications a standard identifier is usually more than sufficient. At a minimum, the TXBnSIDH, TXBnSIDL and TXBnDLC registers must be loaded.

Once the message has been stored in a buffer, a "request-to-send" action needs to be sent to the CAN controller. This can be done in a number of ways - by sending an SPI "RTS" command, setting the TXREQ bit in the TXBnCTRL register or by pressing on of the CAN E-Block switches. The CAN controller will then wait for the bus to become available before actually sending the message. Once the message has been sent, the TXREQ bit will be cleared and the TXnIF flag will be set in the CANINTF register.

Further reading

We have produced a similar C/ASM guide for using our SPI E-Block - this will be a good start for those needing to understand how to perform SPI communication. If using a microcontroller with an in-built SPI module, see the relevant section in that microcontroller's datasheet for specific information regarding that module and how to program it.

Microchip have produced a number of useful document about the CAN protocol and the datasheets for the MCP2515 itself is an invaluable source of information, e.g.:

- AN713 - Controller Area Network (CAN) Basics
- AN215 - A Simple CAN Node (including some useful assembly source code)
- The MCP2515 datasheet itself