CONTROLLER AREA NETWORK
PROTOTYPING WITH ARDUINO

COPPERHILL TECHNOLOGIES

http://www.copperhilltech.com
From the Author

It seems like a million years since I had a soldering iron in my hand and that I have been engaging in my most favorite programming activity, namely programming of embedded systems. In the past, I did shy away from the expenses that came with embedded programming, but with the emergence of inexpensive prototyping systems such as the Arduino or Raspberry Pi this concern doesn’t exist anymore.

Add to this a virtually non-existing learning curve. With my Arduino Uno I ordered a book explaining Arduino Sketches, and I read it for about 30 minutes to scan through the most important information. Then it took maybe another 30 minutes to get my first application running.

I know, I am joining the enormous club of Arduino users who made and expressed the same experience, but that doesn’t change the fact that the Arduino is the perfect environment for prototyping of embedded computer systems.

Naturally, with my knowledge of all kinds of Controller Area Network topics, I was eager to convert that knowledge into the real thing, namely a working CAN application. That CAN application will be the basics of an USB-to-CAN Gateway with CAN network monitoring and diagnostic features as explained in the chapters to follow. From here on, with the knowledge gained through this project, I encourage you to let your mind flow and extend the application. The possibilities are plenty. Enjoy!

However, before we get there let me explain the approach of writing this book: I could have engaged into writing many pages about Arduino basics, what it is, where it comes from, how to use it, etc., for the mere purpose of adding more pages and, as a result, being able to charge more money for my book. However, there are myriads of books on Arduino, Arduino Sketches, and Arduino Shields available in the market, and I won’t waste your money or time. However, references to Arduino basics may appear but only in passing.

That being said, this book assumes some knowledge of the Arduino hardware and its programming.

It also assumes some basic knowledge of Controller Area Network (CAN). I will refer briefly to some aspects of CAN, but these are the mere basics of the actual protocol, just enough to understand the concept. In all truth, there is no need to understand all details of the protocol, since 100% of the protocol is implemented on a chip, the CAN controller. All we need to do in this book is to receive, transmit, and process data. The rest is up to your fantasy.
Nevertheless, the CAN protocol utilizes some ingenious features, and if you are interested in learning more, please refer to *A Comprehensible Guide to Controller Area Network* as mentioned in the literature appendix of this book.

Last, but not least, let me lose some words on my programming style that is definitely different than what you usually see.

I put great emphasis not only on readability of code; I also have debugging in mind when I write code. I am using a slightly modified version of the Hungarian Notation, meaning looking at a variable’s or function’s name provides you some information about its nature. For instance, the prefix *n* indicates an integer variable (e.g. `nVariable`). In addition, being familiar with a number of programming languages, I attempt to keep the best of all worlds. For instance, I add comments behind almost every bracket to indicate information such as *end if* or *end while*, etc., which helps identify program blocks. This may be helpful for *Visual Basic* programmers who are new to C/C++ programming.

Like under Visual Basic, my functions/routines start with either *Sub* (the return code is *void*) or *xFct* (where *x* indicates the type of the return code, for instance, *n* for integer).

**About the Author**

I am the author of the “Comprehensible Guide” series of technical literature covering topics like Controller Area Network (CAN), SAE J1939, Industrial Ethernet, and Servo Motor Sizing. I have worked in the CAN industry since 1997 and before that was a motion control engineer in the paper manufacturing industry. I have a master’s degree in electrical engineering from the University of Wuppertal in Germany.

During the past years, I have conducted numerous seminars on industrial fieldbus systems such as CAN, CANopen, SAE J1939, Industrial Ethernet, and more during various *Real Time Embedded And Computing Conferences* (RTECC), ISA (Instrumentation, Systems, and Automation Society) conferences and various other events all over the United States and Canada.

I had the opportunity of traveling the world extensively, but settled in New England in 1989. I presently live in an old farmhouse in Greenfield, Massachusetts with my red-haired, green-eyed Irish-American wife and our son Patrick.
For more information on my works and to contact me, see my website at http://copperhilltech.com.

**Contact the Author**

Despite all efforts in preparing this book, there is always the possibility that some aspects or facts will not find everybody’s approval, which prompts us, author and publisher, to ask for your feedback. If you would like to propose any amendments or corrections, please send us your comment. We look forward to any support in supplementing this book, and we welcome all discussions that contribute to making the topic of this book as thorough and objective as possible.

To submit amendments and corrections please log on to the author’s website at http://copperhilltech.com/contact-us/ and leave a note.

**Code And Projects Download**

Any additional information created after the publishing date of this book plus project & source code (Arduino and Windows) are available as a free download through the author’s website at http://copperhilltech.com/controller-area-network-can-prototyping-with-arduino/
# Table of Content

1. Introduction to Controller Area Network ................................................................. 1
2. Prototyping Hardware and its Variants ................................................................. 3
   2.1 Arduino ............................................................................................................. 3
   2.2 Intel Galileo .................................................................................................... 4
   2.3 LeafLabs Maple Microcontroller Board ......................................................... 4
3. Arduino CAN Shields ............................................................................................... 5
   3.1 Microchip MCP2515 CAN Controller ............................................................ 5
   3.2 Arduino CAN-Bus Shield by SK Pang electronics ............................................. 7
   3.3 CAN-BUS Shield by Seeed Studio ................................................................. 9
4. Arduino CAN Sketches ............................................................................................ 10
   4.1 The MCP2515 Library .................................................................................... 11
      4.1.1 Function Calls ....................................................................................... 11
      4.1.2 Implementation ..................................................................................... 12
   4.2 CAN Programming ......................................................................................... 13
      4.2.1 Simple CAN Shield Test ........................................................................ 13
      4.2.2 Extended CAN Shield Test .................................................................... 15
      4.2.3 A Simple CAN Network Monitoring and Diagnostics Program ............ 17
   4.3 CAN Network Monitoring under Windows .................................................... 25
5 Conclusion ............................................................................................................... 31
Appendix – Recommended Literature ..................................................................... 33
1. Introduction to Controller Area Network

Controller Area Network (CAN) is a serial network technology that was originally designed for the automotive industry, especially for European cars, but has also become a popular bus in industrial automation as well as other applications. The CAN bus is primarily used in embedded systems, and as its name implies, is a network technology that provides fast communication among microcontrollers up to real-time requirements, eliminating the need for the much more expensive and complex technology of a Dual-Ported RAM.

CAN is a two-wire, half duplex, high-speed network system, that is far superior to conventional serial technologies such as RS232 in regards to functionality and reliability and yet CAN implementations are more cost effective.

While, for instance, TCP/IP is designed for the transport of large data amounts, CAN is designed for real-time requirements and with its 1 MBit/sec baud rate can easily beat a 100 MBit/sec TCP/IP connection when it comes to short reaction times, timely error detection, quick error recovery and error repair.

CAN networks can be used as an embedded communication system for microcontrollers as well as an open communication system for intelligent devices. Some users, for example in the field of medical engineering, opted for CAN because they have to meet particularly stringent safety requirements.

Similar requirements had to be considered by manufacturers of other equipment with very high safety or reliability requirements (e.g. robots, lifts and transportation systems).

The greatest advantage of Controller Area Network lies in the reduced amount of wiring combined with an ingenious prevention of message collision (meaning no data will be lost during message transmission).
2. Prototyping Hardware and its Variants

As I had mentioned earlier in this book, it is assumed that you have some basic knowledge of the Arduino, Arduino Sketches, and Arduino Shields. I will nevertheless take the opportunity of mentioning the prototyping hardware and its variants.

It is important to know that the Arduino, even though perfect for prototyping due to its low price and ease of programming, is not, in its bare form, an industrial-strength solution, not only in terms of environmental specs (e.g. temperature range, etc.) but also in terms of execution speed and memory resources.

Specifically, when it comes to CAN applications at 1 Mbit/sec and high data traffic, the Arduino may reach its limits quickly. There are, however, advanced and yet compatible alternatives to the Arduino as explained in the following chapters.

2.1 Arduino

In order to develop and test the sample programs (sketches) as shown in this book, I used the Arduino Uno. The hardware consists of an open-source hardware board, usually designed around an 8-bit Atmel AVR microcontroller with 2 KB RAM (working memory), 32 KB Flash Memory (sketches) and 1 KB EEPROM (non-volatile).

These technical specifications are more than sufficient for basic prototyping of CAN applications and the proof of concept. However, to re-iterate the point, with growing demands for execution speed and extended functionality, the Arduino may quickly reach its limits.

Note: All Arduino programs (sketches) as shown in this book were developed and tested with the Arduino Uno. There is no guarantee that these programs will work “as is” on any other compatible system.
If your current Arduino-based project is pushing against the performance limits of the ATmega, porting it over to Maple may be the fastest and easiest way to continue developing your project without starting from scratch.

By swapping the popular "avr-gcc" compiler with CodeSourcery’s "arm-none-eabi-gcc," LeafLabs manages to provide a nearly identical programming experience to Arduino despite targeting a completely different architecture. Also, while some Arduino shields are incompatible due to certain capabilities being allocated to different pins, several of them are currently supported and there are more to come. There is also a guide available on the product page for porting Arduino libraries and source code over to Maple. *(Source: LeafLabs open electronics)*

For more information see: http://leaflabs.com/docs/hardware/maple.html

### 3. Arduino CAN Shields

Since Controller Area Network (CAN) is predominantly targeted at industrial solutions (versus the vastly more popular USB for non-industrial use such as home and lab), there aren't too many choices available in the market.

Through some research (i.e. browsing) I found two very similar solutions, and they both work with the same CAN library (as explained in a later chapter). Both solutions use the Microchip MCP2515 CAN controller. Also, both solutions are distributed through worldwide online resources.

#### 3.1 Microchip MCP2515 CAN Controller

Microchip Technology's MCP2515 is a stand-alone Controller Area Network (CAN) controller that implements the CAN specification, version 2.0B. It is capable of transmitting and receiving both standard and extended data and remote frames. The MCP2515 has two acceptance masks and six acceptance filters that are used to filter out unwanted messages, thereby reducing the host MCUs overhead. The MCP2515 interfaces with microcontrollers (MCUs) via an industry standard Serial Peripheral Interface (SPI).

The features include two receive buffers with prioritized message storage, six 29-bit filters, two 29-bit masks, and three transmit buffers with prioritization and abort features. *(Source: Microchip Datasheet)*

*Note: CAN specification 2.0B refers to the capability of using standard CAN frames with 11-bit message identifier plus the extended format with a 29-bit message ID.*
To download the full MCP2515 datasheet log on to:

Both CAN shields as described in the following chapters utilize the Microchip MCP2551 CAN transceiver, which converts the internal TTL signals to a differential voltage as demanded by the CAN standard.

To download the full MCP2551 datasheet log on to:
3.3 CAN-BUS Shield by Seeed Studio

In terms of CAN capabilities, the shield by Seeek Studio provides the same functionality as the one by SK Pang electronics, however, it comes with a much lower price tag, because it does not have any additional components besides the CAN interface.

Over all, the device makes a solid impression, especially since the CAN connection is according to standard and in addition provides CAN connectivity through easily accessible terminals.

Features

- Implements CAN V2.0B at up to 1 Mb/s
- SPI Interface up to 10 MHz
- Standard (11 bit) and extended (29 bit) data and remote frames
- Two receive buffers with prioritized message storage
- Industrial standard 9 pin sub-D connector
- Two LED indicators

Notes

- No cables included

All technical information regarding the use of the CAN controller can be found on the company’s wiki website at:
http://www.seeedstudio.com/wiki/CAN-BUS_Shield

Ordering Information

To order the Seeed Studio CAN shield, you can use the following resources (or browse for “Arduino CAN-BUS Shield” for further options):

In order to test and verify the proper transmission and reception of CAN messages, I used the ADFweb CAN-to-USB gateway with its Windows interface.

**Note:** *In order to test a CAN application, you need at least two CAN nodes to establish a network communication. The second node can be another Arduino with CAN shield or (if the budget allows) another CAN device with CAN data monitoring capabilities.*

### 4.1 The MCP2515 Library

As with any serial networking controller, the essential functions are:

1. Initialization
2. Read Data
3. Write Data
4. Check Status

In case of the MCP2515 library, these functions are represented by:

1. **Initialization:** CAN0.begin
2. **Read Data:** CAN0.readMsgBuf
   incl. CAN0.checkReceive, CAN0.getCanId
3. **Write Data:** CAN0.sendMsgBuf
4. **Check Status:** CAN0.checkError

#### 4.1.1 Function Calls

<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
<th>Parameter</th>
<th>Return Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAN0.begin</td>
<td>Initializes the CAN controller and sets the speed (baud rate)</td>
<td>CAN_5KPS ... CAN_1000KPS (See mcp_can_dfs.h)</td>
<td>CAN_OK = Initialization okay</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CAN_FAILINIT = Initialization failed</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
<th>Parameter</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CAN0.checkReceive</td>
<td>Check if message was received</td>
<td>None</td>
<td></td>
</tr>
</tbody>
</table>
In the Arduino project file add the following on top:

```
#include "mcp_can.h"
#include <SPI.h>
MCP_CAN CAN0(10);
```

**Let me repeat here:** The Seeed Studio CAN bus shield has been undergoing some hardware changes to become compatible with systems such as the Arduino Mega 2560. The version 1.0 will work with the Arduino Uno, while all higher versions also work with the Mega 2560. This will also affect the code of the Arduino projects, specifically the line “MCP_CAN CAN0(10);” in the main module selecting the CS pin. That line must change to “MCP_CAN CAN0(9);” for all CAN bus shield versions above 1.0.

You are now ready to go. The following chapters will describe how to implement the function calls.

### 4.2 CAN Programming

The most exciting part about this project is when it comes to the point where two CAN nodes communicate with each other. I started off with writing a simple program that sent messages that were received by my USB-to-CAN gateway and its Windows monitoring software. From there on, I extended the program to also receive CAN messages and display them on the Arduino serial monitor.

In a later chapter, I will also show a Windows programming example that establishes a communication with the Arduino.

#### 4.2.1 Simple CAN Shield Test

The following represents a very simple CAN test program that periodically (i.e. every 1 second) sends out a CAN message with a 29-bit identifier at a baud rate of 250 kbit/sec.

```cpp
// Simple CAN Shield Test
#include "mcp_can.h"
#include <SPI.h>
MCP_CAN CAN0(10); // Set CS to pin 10

unsigned char stmp[8] = {0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37};

// SYSTEM: Setup routine runs on power-up or reset
void setup() {
    // Set the serial interface baud rate
```
In the `loop()` routine, we first declare some variables for the message reception and code conversion. We still send out the same message as before by calling the `CAN0.sendMsgBuf()` function.

Next, we check for the reception of a CAN message, and if that is the case, we read the message into the assigned buffer and retrieve the message ID. The following code is all about converting the received data into a human-readable format (ASCII) and display it on the Arduino serial monitor.

Last, but not least, we halt the system for one second (1000 milliseconds). Naturally, under real-life conditions, this delay is not reasonable, since there can occur literally thousands of messages per one second. However, this code is meant merely as a demo sample that proves that the actual CAN communication can be accomplished with very little code.

If you load this program onto two separate Arduinos with CAN shield, you have not only accomplished a full CAN network, you can also see the CAN messages as they are exchanged between the two nodes.

**Note:** *It may sound obvious, but please make sure, in case you use more than one CAN node, that all nodes are initialized with the same baud rate. Using different baud rates is the most common cause when data communication fails.*

### 4.2.3 A Simple CAN Network Monitoring and Diagnostics Program

The Arduino board in combination with the CAN shield provides the hardware for a full-fledged CAN network monitoring tool, and this next Arduino program is a first step in that direction.

However, before we get into more detail, let me issue some warnings regarding possible restrictions of the system:

- The MPC2515 has only two receive buffers, which limits the system’s capabilities to respond in a timely fashion while receiving and processing the data traffic. For high-speed, high-busload applications, it is recommended to use the message filter functions to reduce the processing load on the CPU.

- Besides the limited processing speed of the 8-bit CPU, the Arduino comes with only 32 kByte program memory, which is sufficient for a great number of small applications. However, when it comes to more demanding tasks such as a CAN monitoring tool, the memory resources may be exhausted
quicker than expected. For instance, the following application already uses roughly 20 percent of the total memory space, and it provides only a very rudimentary version of a monitoring tool.

In all consequence, if you are serious about creating a more-or-less professional application, you might want to consider alternative hardware solutions as discussed in a previous chapter.

In order to create a CAN monitoring system, we need to:

1. Receive CAN messages and display them
2. Be able to enter CAN messages and transmit them

With the previous two programming samples in mind, we have already accomplished step #1, but the next step (entering CAN messages) needs a bit more work.

The idea is to enter the CAN message into Arduino’s serial monitor and transmit the result by clicking the Send button. In order to accomplish that, we need to follow a data entry format as shown in the following.

**Command:**  **Send CAN Message (11 bit)**

**Description:** Node receives a message and transmits it into the CAN bus

**Format:**  

```
#SM id n dd dd....
```

- id = Message ID (2 bytes, hex)
- n = Number of bytes (1 byte, 0 to 8)
- d = data bytes (hex, up to eight bytes)

**Example:**
```
#SM 01FF 8 30 31 32 33 34 35 36 37
```

In this previous example, we design a CAN message with an ID of 01FF and a data length of 8 bytes. These 8 bytes are represented by the number 30 (hex) to 37, which is the equivalent of ASCII-0 to ASCII-7.
The `loop()` routine, however, looks extremely simple, but that only means that the major part of the functionality has been distributed to a number of new functions.

Inside `loop()` are only two function calls:

1. `SubCheckCANMessage()` checks for a received CAN message and displays it on the Arduino serial monitor.

2. `SubSerialMonitorCommand()` receives a string from Arduino's serial monitor, achieves some rudimentary syntax check, and sends out the CAN message.

The remaining function calls are:

- `nFctReadSerialMonitorString()` reads the data format string as entered by the user and returns the string length.
- `nFctCStringInt()` converts a string into integer and returns the integer data.
- `lFctCStringLong()` converts a string into long and returns the long data.

**Note:** Unlike C#, the C and C++ programming languages provide only limited support for data conversion, and sometimes writing your own conversion functions fits your application needs better than the provided library functions.

The following shows screen shots taken through a session with this programming example:

```
/dev/cu.usbmodemfd121

CAN Init OK.
Message ID: 0x80
Data: 0xef 0xee 0xed 0xec 0xeb 0xea 0xe9 0xe8
Message ID: 0x100
Data: 0x30 0x31 0x30 0x33 0x34 0x35 0x36 0x37
#SM 01FF 01 03 02 04 06 07
#SM 00EF 08 0E 0E 0E 0E 0E 0E 0E
```
First, we received two CAN messages (IDs 0x80 and 0x100), then we sent two CAN messages (IDs 01FF and 00EF).

For this operation, I used my standard test configuration (i.e. USB-to-CAN gateway with Windows monitoring tool as the second CAN node), but from here on, it is possible to use two Arduinos with CAN shield running the same application.

In order to extend the functionality of this programming example, the following commands would be helpful to provide a full-fledged monitoring and diagnostics tool:

- **CAN Start/Stop** – Starts or stops displaying messages on the Arduino serial monitor.
- **CAN Baud Rate** – Modify the CAN baud rate.
- **Request CAN Settings** – Reports the current settings such as baud rate and message ID mode.
- **Send CAN Message** in 29-bit format.
- **Add CAN Message Filter**
- **Delete CAN Message Filter**
- **Delete All CAN Message Filters**

And yes, there are multiple possibilities of extending this program toward a really professional version. However, what the Arduino cannot provide is a professionally looking graphical user interface, and this is where the existing USB connection to a PC opens the door to more possibilities.

**4.3 CAN Network Monitoring under Windows**

While programming the Arduino can be exciting (especially since everything works so smoothly), the real fun comes when you can extend the Arduino’s reach to a PC running Windows.

*Note: My apologies to all Mac and LINUX users for bringing a Windows programming example, but there is no better programming than using C# under Microsoft’s Visual Studio. I have enjoyed programming under OS-X and LINUX, but when it comes to producing quick and effective programming examples, I prefer to stay with Visual Studio. However, the experienced programmer should be able to replicate the functionality of the serial monitor.*
Reference: The handling of the USB port is based on an article by Ryan Alford (with added content by Arjun Walmiki, Gregory Krzywoszyja and Mahesh Chand) at: http://www.c-sharpcorner.com/uploadfile/eclipsed4utoo/communicating-with-serial-port-in-C-Sharp/

At program start, the user first needs to select the applicable USB COM port, which initializes the port (SelectedIndexChanged event).

Beyond that, the program functions as a simple USB terminal: Messages are typed in the top text box and sent by clicking on the Send command button. The larger text box displays the received data.

The following shows screen shots taken through a session with our Arduino CAN Network Monitoring and Diagnostics program:

In this case, we sent two CAN Messages with the same ID (0100) but different data. Next, we received to messages through my standard test configuration (i.e. USB-to-CAN gateway with Windows monitoring tool as the second CAN node).
Appendix – Recommended Literature

There is more than plenty and valuable literature available on the Arduino, but, being an experienced programmer, the one and only work I read was:

Programming Arduino
Getting Started with Sketches
By Simon Monk
ISBN 978-0-07-178422-1

Also recommended for providing more background information:

A Comprehensible Guide to Controller Area Network
By Wilfried Voss
ISBN 978-0976511601

A Comprehensible Guide to J1939
By Wilfried Voss
ISBN 978-0976511632

All works are available through Amazon.com and all their international online stores, Barnes & Noble, Abebooks.com and all their international online stores, and any other good book store.