

## [第1章]

信頼性の高い通信を行う

# 中距離マイコン間インターフェース CANとは

### ● CANとは

CANとは、「Controller Area Network」の略で、BOSCH社が開発した車載ネットワークのことです。元々は自動車内の機器間の通信で使用されるためのシリアル・バスですが、信頼性の高さや優れた故障検出機能により、オートメーション機器の制御などにも利用されています。

CANは仕様が公開されているため、ほかの自動車メーカなどもこの方式を採用しているところがあります。

最近のカー・エレクトロニクスではコンピュータ化が進み、いたるところに多数のマイコンが使われています。これらの機器を個別に配線していたのでは、膨大な量の配線が必要になり、重量もコストも増大します。この問題を解消する手段として開発されたのがCANです。ネットワーク上で短い情報(コマンドやステータスなど)をやり取りすることで各マイコンを制御できるため、車内配線の大幅な省力化が可能です。

CANの特徴として、比較的長距離の伝送が可能、通信レートが高い、差動通信路のためコモン・モード(同相)ノイズに強い、エラー検出機能が充実してリカバリが容易などが挙げられます。また、マルチ・マスタ、というより、すべてのデバイスがマスタでありスレーブであり、優先順位もメッセージの内容に依存するというのも大きな特徴です。

ただし、一度に送受信できるデータは最大8バイトと短く、大量のデータをやり取りするのには向いていません。

本書では、比較的安価で手軽に実現できるマイコン機器間の中距離通信の通信手段としてCANを取り上げ、マイコン機器を制御する具体的な方法を解説していきます。

### ● 転送レート

CANバスは比較的長い距離でも通信できるのが特徴の一つですが、当然ながら、伝送路が長いと通信速度も落ちます。規格上の最高レートは伝送路が40mのときで1Mbps、1000mのときでは50kbpsとなっています。

CANは通信レートの違いにより250kbps以上のものはハイ・スピードCAN、それ未満で、125kbps程度のはロー・スピードCANと分類されています。実際の自動車では、エンジンやブレーキなどの制御系にはハイ・スピードCAN、パワーウィンドやワイパなどの制御系にはロー・スピードCANやLIN(これもクルマ車載ネットワークの一つの標準規格)というように数種類のバスが

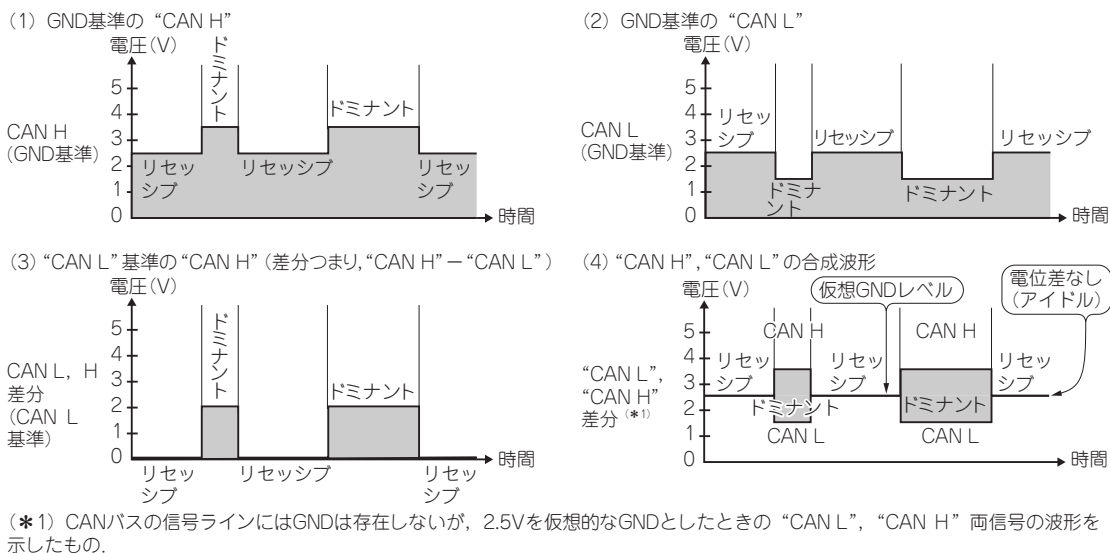


図1-1 CANバスの信号

これらのタイムチャートはCANバスの信号の電圧レベルを基準電圧を変えて表現したものです。

混在しています。

同一のCANバスに接続されているすべてのCANデバイスは、同じ転送レートで動作している必要があります。

## ● CANバスの信号——ドミナントとリセツシブ

CANバスの信号は、2本の信号ラインで伝達されます。コモン線(GND)は不要です。この二つの信号は、“CAN H”、“CAN L”と呼ばれ、両信号間の電位差(電圧差があるか、ないか)で信号を伝達します。I<sup>2</sup>CやTTLレベルの信号のようにGNDに対する電位レベルで信号を伝達するのとは異なります。

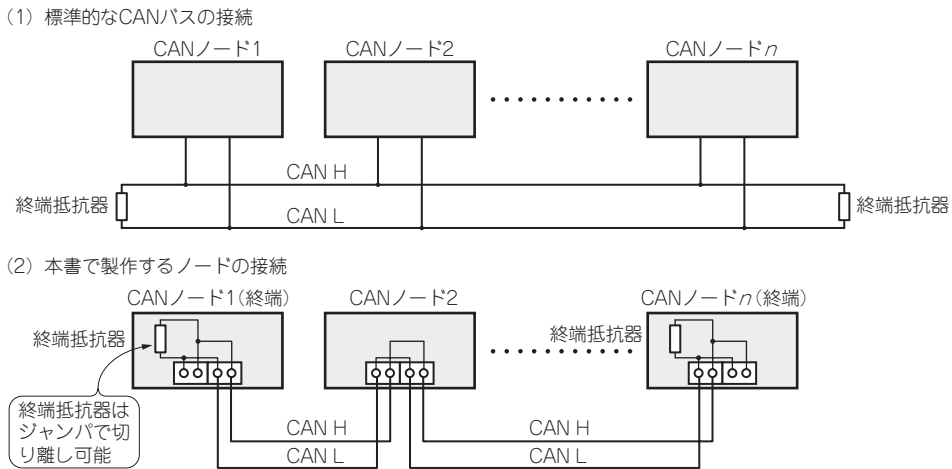
実際の自動車内の配線では、CANラインと一緒に電源もケーブルで分配されていて、電源を共用している関係でGNDがシャーシなどで接続されていますが、本来は電源を共用する必要もないため、2本の信号線だけで通信できます。

このように電位差で動作するものを差動(ディファレンシャル)式といいます。差動式は伝送路などで拾う同相のノイズが打ち消されるため、外来ノイズの影響を受けにくく、自動車などのノイズの発生源の多いところではとくに有用な方式です。差動式は、RS422などの長距離伝送にも利用されています。

CANバスの信号状態には図1-1のように2通りあり、“CAN L”、“CAN H”両ラインの、

- 電位差がある状態をドミナント (Dominant)
- 電位差がない状態をリセツシブ (Recessive)

といいます。ドミナントは信号がアクティブである、リセツシブは非アクティブ(アイドル)である、と言い換えることもできます。このように、ドミナントもしくはリセツシブの状態と長さの組み合わせにより、データを伝達します。



**図1-2 CANノードの接続**

一般的なCANノードの接続方法と、本書で製作するCANノードの接続形態を示す。本書で製作するものはCANの端子を二つ持ち、内部で並列に結線されているため、ノードを数珠つなぎにできる。

CANは、同期用のクロック信号がなく、同時に双方向の通信ができないことから、非同期の半二重通信方式といえます。

同一バス上で複数のノード(接続されているCANデバイス)が信号を出力する際、あるノードがリセッシブ状態にある場合でも、ほかのノードがドミナントを出力すると、バスの状態はドミナントに上書きされます。リセッシブはバスがアイドル状態と考えればわかりやすいでしょう。この仕組みはバスのアービトレーション(調停)やACKの応答に利用されます(後述)。

CANバスの両端には終端抵抗が必要です。図1-2はバスにつながるCANデバイス(ノード)間の配線例を示したものです。

## ● 通信波形の確認

図1-3はCANバス・トランシーバを二つ接続して、片側にオシレータ(発振器)をつなぎ、CANバスの“CAN H”と“CAN L”の2本の信号をオシロスコープで実際に測定したものです。オシレータの周波数はロー・スピードCANの代表値である125kHzに設定してあります。

トランシーバのGNDをオシロスコープのGNDレベルにして、“CAN H”、“CAN L”の両信号をオシロスコープのCH<sub>1</sub>とCH<sub>2</sub>に接続し、2チャンネル同時に表示させています。

約2.5Vを中心として約±1Vで信号が伝達されている様子が確認できます。なお、CANバス・トランシーバについては第2章以降で説明しています。

## ● CANノード

CANバスに接続される各デバイスは「ノード」と呼ばれます。同じくシリアル・バスの規格であるI<sup>2</sup>C(シングル・マスタ・モードの場合)やSPIのように一つのマスタに対して複数のスレーブが接続されているというのではなく、各ノードがマスタであり、スレーブでもあります。通信を始めるノー

## [第2章]

CAN専用コントローラを用いて各種マイコンから活用を図る

# CANコントローラMCP2515の機能

本章では、本書でのCAN通信の要となるCANコントローラMCP2515について詳しく説明します。このコントローラさえ使えるようになれば、多様なCPU用の制御プログラムにも応用ができるようになります。

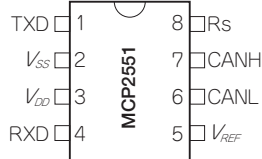
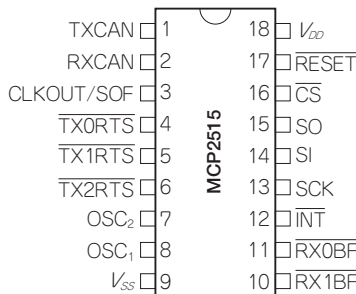
## 2-1 CANコントローラの概要

### ● MCP2515 CANコントローラの概要

マイクロチップ社のMCP2515はSPI(Serial Peripheral Interface)通信で操作することできるCANコントローラです。CAN V2.0Bに対応しています。CANトランシーバMCP2551と組み合わせて簡単にCANノードを作ることができます。図2-1にMCP2515のピン配列を示します。DIPパッケージも用意されているため、ブレッドボードなどでも使用できます。また、フラット・パッケージを含めた外観を写真2-1に示します。

#### おもな仕様

- ▶ CAN V2.0B準拠 最高ビットレート 1Mbps
- ▶ メッセージ長は11ビット, 29ビット両対応

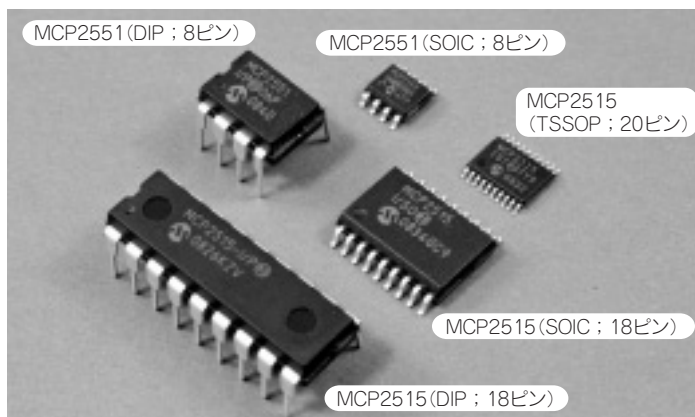


(a) CANコントローラ MCP2515(DIP/SOIC) ピンアウト(\*1)

(b) CANトランシーバ MCP2551(DIP/SOIC) ピンアウト(\*2)

### 図2-1<sup>(1),(2)</sup> CANデバイスのピン・アウト

本書で使用するCANコントローラ(MCP2515)とCANトランシーバ(MCP2551)のピン配列を示す。MCP2515のTSSOPパッケージは20ピンで一部ピン配置が異なるので注意。



### 写真2-1 CANコントローラIC

マイクロチップ社のCANコントローラ(MCP2515)、CANトランシーバ(MCP2551)の外観を示す。MCP2515のTSSOPのピン数、ピン配列は、DIPとSOICのものとは異なるので注意。

- ▶二つの受信バッファをもつ
- ▶三つの送信バッファをもつ
- ▶六つの29ビット・フィルタと二つの29ビット・マスクをもつ
- ▶三つのRTS信号入力ピンあり
- ▶二つのRXBUF信号出力ピンあり
- ▶ハイ・スピードSPI(10MHz)
- ▶クロック出力端子あり(ホスト・マイコンのクロックなどに利用可能)
- ▶送受信タイミングなど多様な割り込み発生機能あり
- ▶ロー・パワーCMOSテクノロジーで動作電圧は2.7V～5.5V動作時消費電力 5mA(標準)
- ▶動作保証温度 - 40℃～+85℃(産業用)
- ▶スリープ・モードあり データ受信などのイベントでウェイクアップ可能

このコントローラは、PIC18Fxx8などに内蔵されているECANモジュールのレガシ・モード相当の機能があるものと思われます。

レジスタのアクセスの際は、PICがもつSPI通信でインストラクションやデータを送受信するため、特定の手順が必要ですが、その反面、SPIさえ備えていれば、接続先を選ばないという利点があります。

#### ■ RTS<sub>n</sub>信号( $n=0\sim 2$ )

MCP2515には三つのRTS信号入力があります。この三つはそれぞれ三つの送信バッファに対応していて、ハードウェアにより送信を開始させるトリガ信号となっています。この機能を利用しない場合は、切り替えにより単純な汎用の入力ポートとしても使えます。

#### ■ RX<sub>n</sub>BUF信号( $n=0,1$ )

MCP2515には二つのRXBUF出力信号があります。この二つはそれぞれ二つの受信バッファに対応していて、メッセージが受信されたタイミングをハードウェアで検出できるようになっています。割り込み信号として利用できます。この機能を利用しない場合は、切り替えにより単純な汎用の出力

## [第3章]

実際にノード間で通信を行う

# CANノード(ハードウェア)の製作

本章では、MCP2515を使ったCANノードを製作します。通信の確認には、CAN対応のマイコン・セットが最低2台必要ですが、できれば3台以上あったほうがいろいろ実験ができます。制御ソフトウェアに関しては、第4章以降で説明します。

## 3-1 製作する機器

本書では、CANノードとして使用できるマイコン・ボードを2種類と、PIC以外のマイコンにも接続できるSPI-CANブリッジ、SPI-CANブリッジと組み合わせて使用するマイコン制御のリレー・ボードの4種類を製作します。この章以降のアプリケーションでは、これらのボードを組み合わせて使用します。“#219”などの番号は、筆者が製作したプリント基板の通し番号です。

次に各機器の概要を説明します。

### ■ PIC CANコントローラ(#219)

CAN対応のPICマイコン・ボードです。温度センサが接続できるため温度測定が可能です。また、リレーも1回路用意してあります。

通信確認のCANノードとして使用するほか、応用編のCANバス・モニタやシリアル-CANブリッジなどに使用します。

### ■ CAN制御LCDボード(#221)

CAN制御のLCD(液晶表示器)コントローラです。ほかのノードからのコマンドでLCDへ文字を表示させます。

### ■ SPI-CANブリッジ(#217, #220)

CANコントローラとCANトランシーバのみを搭載したもので、SPIでCANコントローラを操作します。PIC CANコントローラのCAN関係の回路を抜き出したものとはほぼ同じです。AVRなどほかのCPUにも接続可能で、簡単にCANノードが製作できます。

回路はほぼ同じで、ブレッドボードに実装できる小型のもの(#220)とケースなどにネジ留めできるちょっと大きめのもの(#217)の2種類を製作していますが、配線方法(ブレッドボードを使うか、ナイロン・コネクタで接続するか)を変更すれば、どちらでも同じようにして使用できます。

### ■ リレー・ボード(#222)

PICで4系統のリレーを制御するボードです。本書ではSPI-CANブリッジと接続し、CANコント

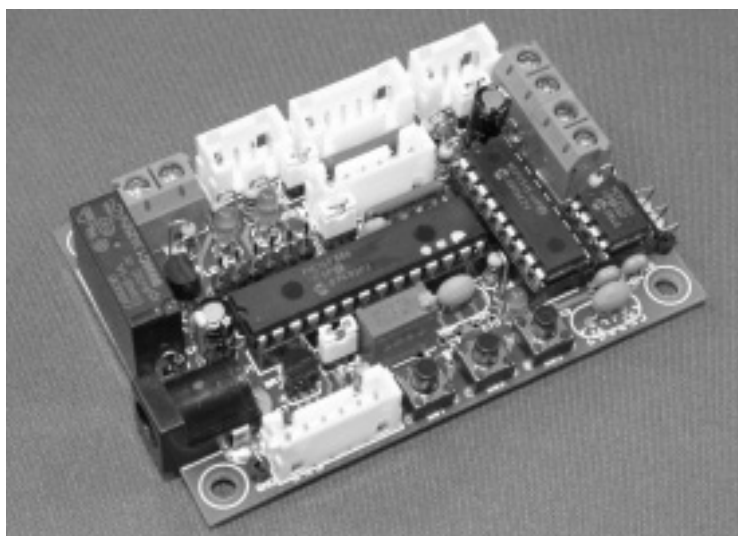


写真3-1 PIC CANコントローラ(#219)

ホスト・マイコンにPICに16F886を使用したCANノードの1号機。この写真は試作時のもので、一部のナイロン・コネクタなど使用しない部品も実装されている。

ローラのホストとしてプログラミングして、CAN制御のリレー・ボードとして利用します。

## 3-2 PIC CANコントローラ(#219)

### ● 概要

28ピンのPIC(16F886など)にCANコントローラのMCP2515とCANトランシーバのMCP2551を搭載したマイコン・ボードです。写真3-1に専用基板で製作したボード本体の外観を示します。

CAN端子は、2Pの端子台またはナイロン・コネクタが二つ用意してあり(どちらか一方だけ実装可能)、端子台を使う場合はバラ線のケーブルが直接接続できます。CAN端子が二つあるのは、図1-2(b)のように、ノードをチェーン接続させるためです。ターミネータも内蔵しており、ジャンパで接続、切り離しの切り替えが可能です。

文字を表示するようなものは何も付いていませんが、TTLレベルの非同期シリアル信号がコネクタから取り出せるため、外部にRS-232Cのレベル・コンバータやUSB変換アダプタを接続することで、PCとシリアル接続できます。

また、CAN制御のリモート機器の製作を意図して設計したので、リレー1個とアナログ信号の入力コネクタ、A-D変換のリファレンス電圧発生用のシャント・レギュレータが実装可能です。応用編では、アナログ入力に温度センサのLM60を接続して温度を測定します。

### ● 使用する部品

コントローラにマイクロチップ社のPIC16F886を使用しています。16F876(A)やプログラム容量が半分の16F873(A)も使用できますが、コンフィギュレーション・ビットやアナログ関係の設定が

## [第4章]

MCP2515のレジスタを使いこなす

# CANコントローラの制御プログラム

本章では、実際にCANコントローラMCP2515を制御する関数やマクロを定義して、使用法などを説明します。

## 4-1 はじめに

### ● 二つの課題

初めてMCP2515を使うにあたっては、大きく二つの課題があると思います。一つはMCP2515を使うための「レジスタの操作方法」、もう一つはそのレジスタを操作するために用いられる「SPIコマンドの使い方」です。

CANは豊富なエラー検出機能があり、それらを完璧に使いこなすのは大変ですが、まずはエラーに関する処理はあまり気にせず、とりあえず通信できることを第一の目標として話を進めます。

筆者は、まず初めにSPIコマンドについて調べました。まずは実際にどのようにレジスタ・アドレスを指定して、どうやって値を読み書きするか、また、どのような種類のコマンドがあるか、ということです。これをまとめたのが第2章の図2-3(p.29)、図2-4(p.31)です。このようにまとめてしまうと、レジスタへのアクセスはそれほど難しくはないということがわかるでしょう。

実際のコードは、基本的には単にSCKクロックでタイミングをとりながらビット・データSI、SOを読み書きするだけなので非常に簡単です。今回はPIC内蔵のMSSPやSSPを使わないでソフトウェアだけで制御しているため、AVRなどほかのCPUのCコンパイラでも移植が容易です。

残る課題はレジスタ操作ですが、これが意外と単純ですので、できてしまえば、拍子抜けする(?)ぐらい簡単でした。ただ、送受信バッファの数が多いので、レジスタ番号などを混乱して間違えないようにすることが肝心です。

今回、レジスタ・アドレスやビット定義などを独自に記述しましたが、記述ミスを見つけるのに3日ほど費やしたものの、それらのミスを除けば、単純なCAN通信はあっけなく実現しました。

CANコントローラのエンジンは、ビット・スタッフィングやCRC計算などいろいろ複雑なことをやってくれているのですが、ハードウェアのおかげで制御ソフトウェアは非常に素直に作成できます。

使い方、動作の確認として、第5章ではいくつかの項目をプロジェクト単位で段階的に動かして、少しずつ変更を加えて機能を拡張して行きます。ソース・ファイルもプロジェクト単位でフォルダを作成してまとめてあります。

## ● たくさんのレジスタ

MPC2515はたくさんのレジスタもっていますが、落ち着いてじっくり眺めてみると、14バイトの送信バッファが3本、14バイトの受信バッファが2本、マスクやフィルタのレジスタが合わせて8本、コンフィギュレーションとステータス、コントロールのレジスタがいくつかという具合に、それほど種類が多いわけではありません。

送受信バッファは一部の内容が異なりますが同じ構成になっています(第2章図2-2参照)。また、マスクやフィルタ関係のレジスタも送受信バッファのID部分と同じような形になっています。レジスタの命名規則もバッファ名に連番が振られているようなものなので、パターンを覚えてしまえば扱いも簡単でしょう。

データシートでは、レジスタの説明は機能ごとに分散して記述されていますが、本書ではAppendix-A(pp.178～181, Appendix-B(pp.182～183)にまとめてあります。

## ● 関数、マクロ化の方針

PIC16シリーズはスタック・レベルが8しかないので、関数のネスト(入れ子)をあまり深くすることができません。油断するとすぐにスタック・オーバフローになってしまいます。Forest Electronics Development社のPIC用CコンパイラWIZ-Cでアプリケーション・デザイナー(APD)を使って開発しているときはとくにそうです。ただ、WIZ-CにはPICスタックを使わないでRAM上にスタックを作るという手段も用意されているので、最悪の場合そちらのオプションを使えば、8を超えるスタック・レベルでも対応できます。ただしオーバヘッドが増えるため、できることなら、そのオプションは使わないにこしたことはありません。PIC18やAVRではこのような心配は不要なのですが…。

対応策としては、小さな関数を減らして、コードをべた書き(同じ処理でも使うところでそのたびに同じコードを記述する)すれば、スタックの消費は抑えられますが、その分コードが増大します。

そこで今回は、極力重複コードを避け、スタックをあまり消費しないように機能のある程度まとめて関数化し、その関数をマクロから呼び出すという方法をとっています。

たとえば、レジスタ・リードとステータス・リード、受信ステータス・リードの三つのコマンドはインストラクション・コードが違うだけで、処理内容はほぼ同じです。そこで、この三つの機能を一つの関数で処理させて、インストラクション・コードとレジスタ・アドレスを引数で切り替えるような三つのマクロを用意して、一見別々の3種類の関数があるように見せかけます。

例を挙げると、この共用される関数の実体は次のようなものです(定義部分のみ)。

```
BYTE CANRegRead2B(BYTE inst, BYTE adrs);
```

この関数は以下の三つの関数型マクロにより、3種類の関数と同じように扱えます。

```
// (1)レジスタバイト・リード
```

```
#define CANReadReg(adrs) CANRegRead2B(SPI_INST_READ, adrs)
```

```
// (2)ステータス・リード
```

```
#define CANReadStat() CANRegRead2B(SPI_INST_RD_STAT, 0)
```

```
// (3)受信ステータス・リード
```

```
#define CANReadRXStat() CANRegRead2B(SPI_INST_RX_STAT, 0)
```

SPI\_INST\_xxxというのが、SPIコマンドのインストラクション・コードを定義したリテラル

(定数)です。(2)ステータス・リードと(3)受信ステータス・リードではレジスタ・アドレスは不要なので、マクロの引数にはありません。この場合は関数CANRegRead2B()の引数の'0'はダミーのアドレスです。

このようにマクロと組み合わせることで、SPIコマンドのインストラクション・コードを意識する必要も、使わない引数を指定する必要もなくなります。

送受信バッファやマスク、フィルタ関係のマクロは、引数でバッファ番号などを指定するのではなく、個別にマクロを定義してあります。したがって、変数で、バッファ番号を指定したいような場合は、直接マクロをコールできないため注意してください(コール元でif文やswitch-case文によるマクロの切り替えが必要)。

## 4-2 MCP2515ドライバ関数、マクロのコード

### ● 関数、マクロについて

MCP2515のアクセス用に用意した関数、マクロはAppendix-C(pp.184~186)に一覧でまとめてあるので、そちらを参照してください。

基本的にはSPIでデータを入出力するだけなので、一部の関数を除いて、個別のコードの説明は省略します。また、SPIコマンドの転送データのフォーマットは第2章の図2-3、図2-4も参考にしてください。

### ● レジスタの読み出し関数

代表的なものとして、4-1項の例で述べたレジスタの読み出し関数のコードについて説明します。この関数には、SPIの送信と受信の両方のコードが含まれているので、ほかの関数の参考になります。

関数のコードをリスト4-1に示します。なおこの関数は前述のように、直接呼び出さずにCANReadReg(), CANReadStat(), CANReadRXStat()のいずれかのマクロから呼び出されます。引数のinstは、マクロにより固定値が設定されます。

CAN\_SPI\_CSはSPIの $\overline{CS}$ 信号を“H”または“L”レベルに設定するマクロで、“0”を設定すると $\overline{CS}$ 信号を“L”レベルにできます。同様に、CAN\_SPI\_SCKは、SCK信号を切り替えるマクロです。CAN\_SPI\_SIはSPIのデータ入力ポート(SI)、CAN\_SPI\_SOはSPIのデータ出力ポート(SO)で、それぞれの信号の状態をリード/ライトできます。

この関数で送信されるコマンドは、データを2回(2バイト)出力した後に1回(1バイト)入力します。

出力の際は、1バイトのデータを上位から1ビットずつ順に取り出してCAN\_SPI\_SOポートに設定し、CAN\_SPI\_SCKでクロック・パルスを出力してそれを8回繰り返します。入力の際は、CAN\_SPI\_SIポートの状態を読み出してCAN\_SPI\_SCKでクロック・パルスを出力し、それを8回繰り返して、8ビットのデータに合成します。

SPI通信の間は、CAN\_SPI\_CSで $\overline{CS}$ 信号を“L”レベルに設定しておく必要があります。

なお、SPI通信では本来、送信と受信が同時に行われますが、MCP2515のアクセスではどちらか一方の通信だけが有効なので、使わないほうの通信は無視してかまいません。

図4-1は、ビット・モデファイ・コマンドを実行した際の出力波形の例です。この図はWIZ-C付属

## [第5章]

### CANドライバの使用例

# 基本動作の確認

本章では、第4章で説明したCANドライバ関数/マクロの実際の使用法を示し、その動作を実機で確認します。

処理内容は極力単純にして、動作を確認しながら段階的に機能を追加していきます。

後の章で説明する応用編のプログラムは、これらの使用例を組み合わせたものがベースになっています。

なお、AVRのプログラミングでも、ドライバ/マクロの使用法は同じなので、参考にしてください。また、一部を除いて説明の中でソース・コードを抜き出して掲載しているので、コードの詳細はWebのサポート・ページからダウンロードできるソース・ファイルを参照してください。

## 5-1 基本的な送受信の確認…プロジェクト001

### ● 概要

二つのノードを使用して、ノード1からノード2へ単方向にメッセージを送信します。ノード1にはPIC CANコントローラ(#219)、ノード2にはCAN制御LCDボード(#221)を使用します。

ノード1から送信するメッセージはノード2上のLEDをON/OFFさせるコマンドです。ノード1は1秒周期でLEDのON/OFFを表す6種類のメッセージを順番にノード2へ送信します。

ノード2では、受信したメッセージの内容に応じてLEDをON/OFFさせ、LCD(液晶表示器)に受信メッセージのSID値とLED番号などを表示させます。

図5-1に実験概要のブロック図を示します。

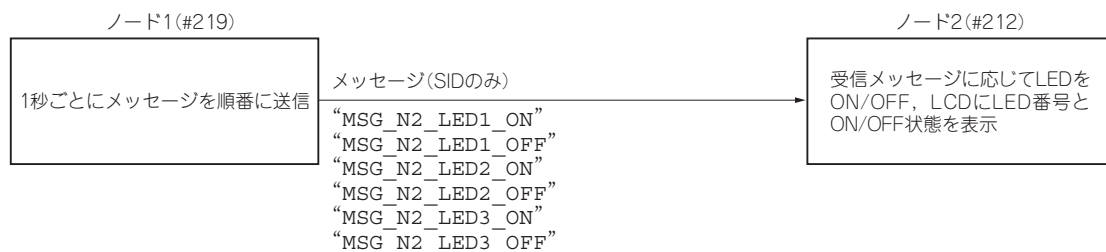


図5-1 プロジェクト001

最も簡単な一方通行の通信実験。ノード1からのメッセージに応じてノード2上のLEDをONまたはOFFさせる。

メッセージ・シンボルの“N1”や“N2”といった記号や番号は、あて先のノードを示しています。“N1”がノード1，“N2”がノード2というようになっています。この命名規則は以降のプロジェクトでも同様です。

## ● プログラム・ファイル、フォルダの構成

ノードごとにプロジェクト・フォルダを用意します。フォルダは次のような階層構成になっています。“[]”はフォルダを示します。

[Prj001]

[219-N1] …… ノード1(#219用)プロジェクト・フォルダ

[221-N2] …… ノード2(#221用)プロジェクト・フォルダ

CANMsg.h …… ノード1，ノード2共通のメッセージ定義ファイル

“219-N1”と“221-N2”は、それぞれ、WIZ-C(FED社)またはCCS-C(CCS社)のプロジェクト単位のフォルダです。“CANMsg.h”は両プロジェクトから参照される共用のヘッダ・ファイルで、共用されるメッセージのリテラルが定義されています。

プロジェクト002以降も基本的にはこのような構成になっています。

そのほか、全プログラムで共通で使われる定義やドライバ関数/マクロ、共通関数などのファイルは、“Common”というフォルダにまとめてあります。WIZ-CとCCS-Cで少し構成が異なりますが、次のようになっています。このフォルダは[Prjxxx]と同じ階層にあります。

### WIZ-C

[Common]

2515Reg.h …… MCP2515のレジスタ・アドレスなどの定義

2515Tag.h …… MPC2515のレジスタのビット定義

CAN2515.h …… CANコントローラのドライバ関数，マクロのヘッダ・ファイル

WSN219.h …… #219基板のI/O関係の定義

WSN221.h …… #221基板のI/O関係の定義

WSN222.h …… #222基板のI/O関係の定義(応用編で使用)

WSN242.h …… #242基板のI/O関係の定義(応用編で使用)

CAN2515.c …… CANコントローラのドライバ関数本体

CommFunc.c …… 共通関数(応用編で使用)

LcdNW.c …… LCDドライバ(Waitなし；応用編で使用)

### CCS-C

[Common]

2515Reg.h …… MCP2515のレジスタ・アドレスなどの定義

# AVRでMCP2515コントローラを使う

本章では、MCP2515とAVRをSPIで接続してAVRからMCP2515を制御します。AVRにはATmega168(88/48)を使用していますが、簡単な実験ならプログラム容量の少ないATtiny2313でも使用可能ですので、そちらのプログラミングについても説明します。

## 6-1 実験回路

### ● AVRの選定

AVRの種類は基本的には何でもよいのですが、プログラムROMの容量が4kW(ワード)以上、できれば、8kW以上あるほうがいろいろ実験できてよいと思います。筆者は手持ちの関係で、mega168を使用しましたが、容量が半分のmega88でも使えます。なお、本章で掲載している実験用のプログラムは結果的に2kWで収まったので、ATtiny2313での作成例も後ほど説明します。

### ● CANコントローラの回路

CANコントローラ回路には第3章で解説したSPI-CANブリッジ(#220)を使用していますが、DIPタイプのICを使えば、ブレッドボードなどでも製作可能なので、その場合は回路図を参考に製作してください。

AVR(ATmega168/88/48)のクロックは20MHzです。16MHzなどに変更してもかまいませんが、タイマの周期が変わるので、メッセージの送信間隔が変わります。それで不都合がある場合は、OCR1Aレジスタへ設定するコンペア値を変更してください。後に述べるATtiny版では、内蔵のRCオシレータを8MHzで使っています。

写真6-1(p.98)はブレッドボード上でATmega168(DIP)とCAN-SPIブリッジ(#220)を配線したものです。AVR側の周辺回路は発振子と電源、SPI関係の配線だけですが、ISP用のピン・ヘッドも取り付けて、AVR ISPなどでプログラムが書き込めるようにしています。タイマ関係のパラメータの変更が必要ですが、内蔵の8MHzのRCオシレータを使えば、さらに簡単な回路で済みます。

その他、動作確認用のLEDを1kΩ程度の抵抗器を通して、出力ポートへ接続しておきます。

ATmega版の回路図は図6-1に示しますが、全実験で同じ回路のものを使うので、プロジェクトごとのI/Oポートのアサインもすべて同じです。任意のポートが使えるため、配線がしやすいような並びになっています。

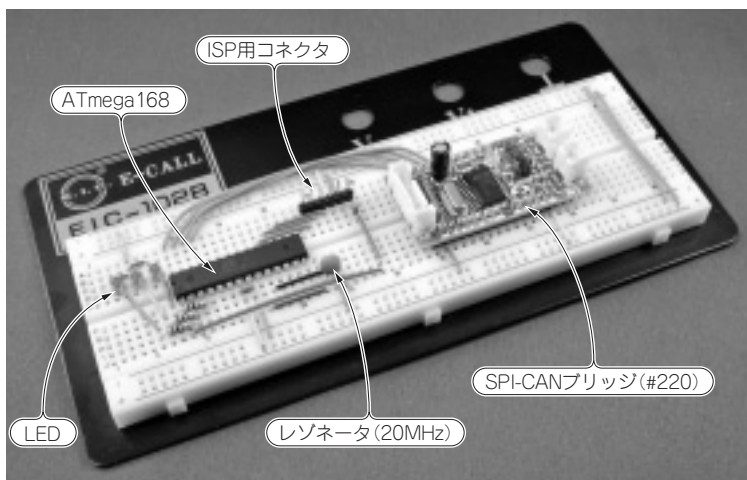


写真6-1 mega168版 CAN ノード  
ブレッドボードとSPI-CANブリッジ基板で製作したmega168版のCAN ノード。

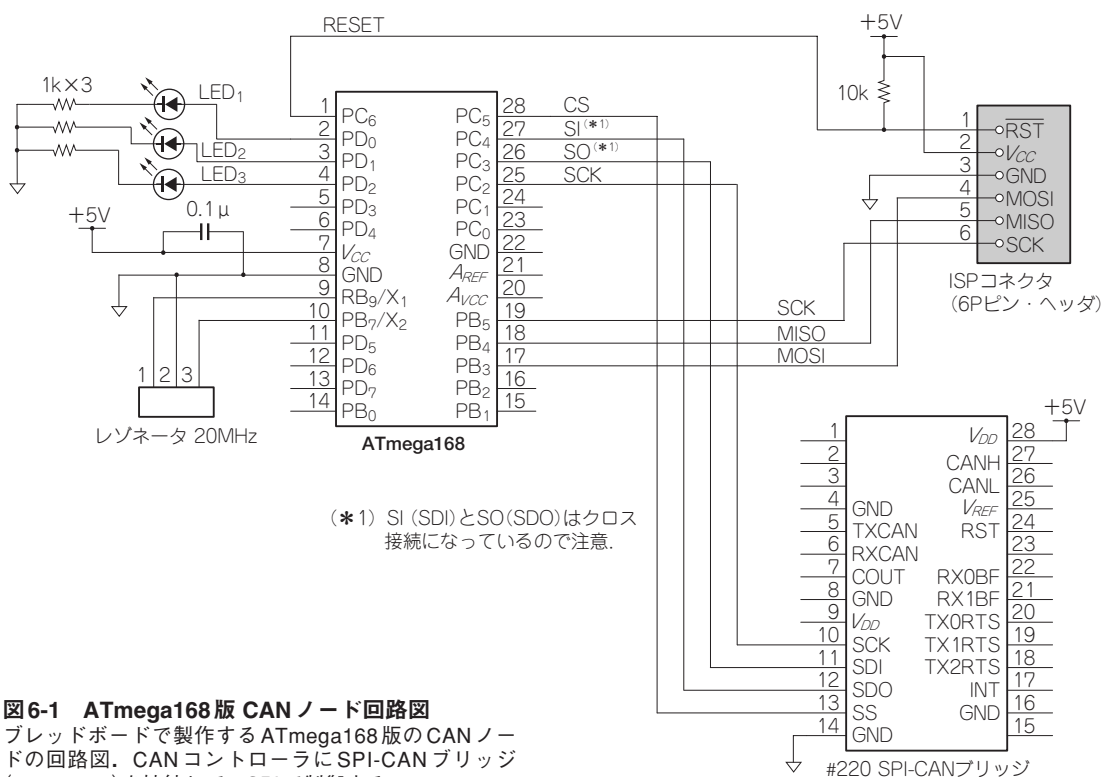


図6-1 ATmega168版 CAN ノード回路図  
ブレッドボードで製作するATmega168版のCAN ノードの回路図。CAN コントローラにSPI-CANブリッジ (#220/#217)を接続して、SPIで制御する。

## 6-2 CAN通信の実験

### ● 実験内容

AVRを使った実験回路でも第5章のPIC版とほぼ同じ実験ができますが、ここではLCDをつけていないので、その部分は省略します。なお、この程度の処理なら、ATtiny2313でも使用可能です。ノード2にLCDを付けたい場合は、後述のtiny2313版の二つめのノードのプログラム(ノード22)やPIC版のプログラムを参考にして、修正してください。

ここでは「5-2 プロジェクト002」(pp.90～91)の双方向通信相当の実験を行います。図5-2のブロック図を参照してください(ただしノード2のLCDはなし)。

## 6-3 WinAVRによるプログラミング

コンパイルには、無償で入手可能なWinAVRのgcc(GNU Cコンパイラ)を使用していますが、AVR StudioやWinAVRの使用方などは省略します。使い方などは参考文献などを参照してください。

### ● MCP2515の制御

MCP2515のSPIコマンドによる制御部分のプログラムは、基本的にPICで使用したものと同じです。違うところは、SPIで使用するI/Oポートの初期化や、1秒周期を得るためのタイマのプログラム部分だけです(詳細は後述)。

ドライバやメイン処理の共通部分は、関数レベルでソース・コードを兼用できるように考慮してあります。そのため、多少冗長な部分もあります。関数やマクロの定義、使用方法については「第4章 CANコントローラの制御プログラム」、その他PICでの使用例も参照してください。

### ● ソース・ファイルのフォルダ構成

AVRのプロジェクトでは、アプリケーション側のソース・ファイルから、ドライバ関数のソース・ファイルをインクルードして、ドライバ部分を含めた1本のソース・ファイルとしてコンパイルしています。これは、第5章でCCS-C(CCS社)でリンカなしでプログラムを作成していたのと同じ方法です。

ドライバ関数のソース・ファイルはCommonフォルダに置かれてほかのプロジェクトと共用されますが、これをアプリケーションのコードの一部として扱うことで、プロジェクトごとに固有のI/Oポートを割り当てられるようにしています。WIZ-C(FED社)ではヘッダ・ファイルのパスの扱いが異なるため、違う方法を採用しています。

フォルダの階層構造は次のようになっています。

[Prj002]

```
[m168-N1] ..... ノード1(mega168)プロジェクト・フォルダ
2515SpiPort.h ... MCP2515(SPI)用ポートのアサインの定義
IOPort.h ..... LEDなどのポートのアサインの定義
```

## [第8章]

### CAN機器の応用装置を作る

# シリアル-CANブリッジの製作

CAN機器の応用例として、PCからの操作でCANバスへメッセージを送信したり、受信したメッセージをPC側で表示させることができるブリッジを製作します。

## 8-1 シリアル-CANブリッジの機能

### ● 機能の概要

本器はPCとUSBやRS-232Cインターフェースで接続して、PC側の操作でCANバスへデータを送出したり、CANバスからデータを受信してそれをPC上で表示させたりするためのものです。ほかのCANノードのデバッグの際など、PCからの操作でノードを動作させ、動作を確認するのに使います。また、シリアル通信でコマンドを送受信して制御するようなWindowsアプリケーションを作成すれば、そのアプリケーションからほかのCANノードを操作することもできます。

### ● PCとのインターフェース

PCとシリアル-CANブリッジ(以下CANブリッジ)との通信はシリアル通信で行います。今回は、非同期シリアルとUSBを相互に変換できるUSB変換ボードを使って、PCとの接続にはUSBを使用します。

USB変換ボードを使うことにより、PC側のアプリケーション、CANブリッジ側のファームウェアはRS-232Cなどの非同期シリアル通信のソフトウェアがそのまま利用できます。

### ● 操作の仕様

まず、PCからどういうことを操作するかを決めます。今回の機器は、CANバスへのメッセージの送出、CANバスからのメッセージの受信が目的ですので、それらを満たすように設計します。

メッセージの送出にあたっては、あらかじめ、メッセージID、データを送信バッファへ設定しておく必要があります。送信データを少なくすることもあり、メッセージ(SID)とデータ・フィールドのデータは別々に設定するようにしてあります。設定したメッセージ(データを含む)は送信開始要求コマンドで送信を始めます。

それ以外に、CANブリッジのCANコントローラのオペレーションとして、CANビットレート設定やマスク値、フィルタ値の設定や、各種ステータス、エラー・コードなどを読み出すコマンドも用

意しています。

## ● CANブリッジのハードウェア

CANブリッジには、CANコントローラMCP2515を制御するマイコンにシリアル・インターフェースを備えたものがが必要です。前章のCANバス・モニタとまったく同じものが使えます。回路図は第7章の図7-1を参照してください。

USBの代わりにRS-232Cのレベル・コンバータ回路を接続して、RS-232CでPCと接続することもできます。

## 8-2 制御ソフトウェア

### ● ソフトウェアの概要

CANブリッジのソフトウェアは、実験で使用したノードにシリアル通信のコマンドでCANコントローラを操作できるようにしたものです。また、前章のCANバス・モニタの内容とも似ています。実際、CANバス・モニタの派生バージョンとして、CANバス・モニタのプログラムを改造して作成しました。

### ● 処理の概要

シリアル通信でコマンドを受信した場合、その内容を解析してコマンドに応じた処理を実行します。通常のコマンドは、CANコントローラを制御したり、そこからステータスを読み出すものです。

また、CANバスからメッセージを受信した場合は、その内容をシリアル通信でPCへ転送します。

### ● シリアル通信コマンド

PCから操作する制御コマンドとして次のようなものを用意しています。コマンドはアルファベット1文字で表し、大文字、小文字の区別はありません。

- ▶マスク値設定('K')
- ▶フィルタ値設定('F')
- ▶フィルタ・オプション設定('O')
- ▶メッセージ設定('M')
- ▶データ設定('D')
- ▶送信要求('T')
- ▶ステータス・リード('S')
- ▶エコーバック切り替え('E')

「メッセージ設定(M)」はアービトレーション・フィールドのID値(SID値)を設定するものです。MCP2515の三つある送信バッファの中で、指定のバッファへ設定します。

コマンド文字列は“m0211”のように設定します。‘0’は送信バッファ0，“211”は11ビットのSID値を示すヘキサ文字列です。

「データ設定(D)」はデータ・フィールドの値を設定するものです。0バイトから最大8バイトの

# 組み合わせ応用例

複数のノードを一つのCANバスにつないで互いに通信するような実験セットを製作します。温度の測定や表示、リレーの操作などを行います。

## 9-1 機器の構成

### ● 機器構成

使用するノードの構成を図9-1(p.144)に示します。各ノードは比較的単機能で、単純な動作しか行いません。わざわざノードに分けて処理させるまでもないのですが、それぞれのノードが離れた場所に別々に設置されているという想定で実験を行います。

写真9-1は評価用に各ノードを1枚のトレイの上に固定したものです。本来は、各ノードは離れた場所にあるべきものですが、デバッグ、評価用ということで、1枚のトレイ上にまとめてみました。

トレイは100円ショップで購入した、底の浅いプラスチック製のものです。ノード基板は10mm長のスペーサとM3のビスで固定してあります。

写真9-1の各ノード基板は試作品のため、部品をフル実装してありますが、使っていない部品(コネクタなど)もありますので注意してください。

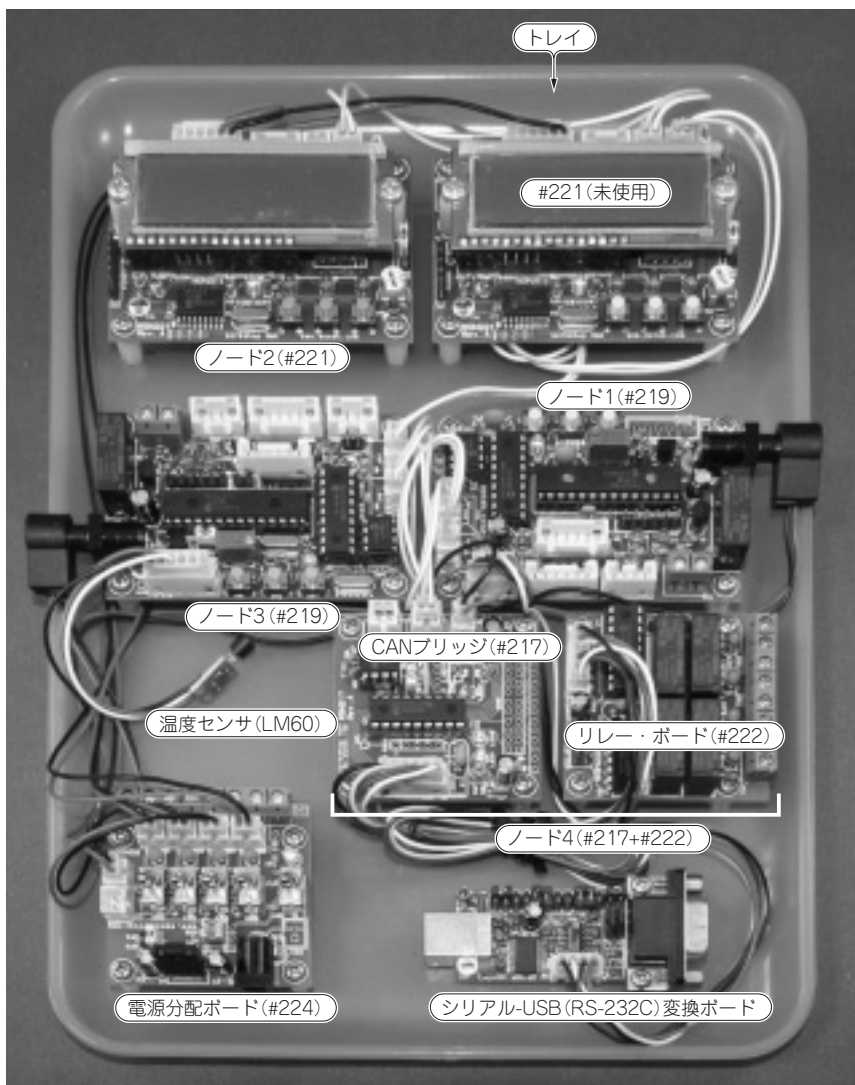
複数の基板を使う場合はいつも電源をどうするかが悩ましいのですが、今回は、電源を分配するボードも製作しました(写真左下)。このボードに関しては、コラム9-1(p.146)に簡単な説明がありますので、そちらも参照してください。

CANバス・モニタを接続する場合は、どこか適当なノードからCAN信号を取り出して接続します。

### ● 実験、評価の目的

一つのCANバス上で、あるノードは自分勝手にほかのノードと通信を行い、それにユーザが割り込んでほかの処理をさせる、というようにいろいろなメッセージが混在しているような環境を作って動作を確認するのが本章での目的です。

具体的には、温度を測定して、その値をほかのノードの表示器に表示させ、通常はこの二つのノード間は周期的に通信を行っています。そこへユーザが別のノードから温度を要求すると、温度を測定しているノードは表示用ノードの通信とは別のノードに温度値を送り返す、というようなことを想定しています。また、温度表示には直接関係なくリレーを操作するなど、関係のあるメッセージ、関係



### 写真9-1 評価セット

すべてのノードをトレイ上に設置した評価セットの写真。右上の #221(CAN制御LCDボード)は使用していない。また、各ボードは試作品のため、使用していない部品が実装されているものもある。

のないメッセージがCANバスに入り乱れるような環境を作りたいと考えました。

### ● ノード間のメッセージ

既製品のデバイスに接続するわけではないため、メッセージの内容は任意ですが、今回は、バス・モニタでログを採ったときにメッセージが区別しやすいように、図9-1のようなフォーマットでメッセージを定義しました。メッセージ長は標準ID(SID)で11ビットですが、上位3ビットは宛先のノー