インターフェース 増刊

見本

トライアル シリーズ

インターネットに接続できる32ビット・マイコン

すぐにつながる！ Ethernetマイコン・モジュール

インターフェース編集部 編

詳しい解説付き！

Ethernetで
インターネットとつながる

32ビット・マイコン
ColdFire MCF52233搭載

MC68000の流れをくむ
エレガントなアーキテクチャ

3軸加速度センサで楽しめる

C言語インタプリタで
アプリが作成できる

CD-ROM 付き

ブレッド・ボードにも接続可能

CQ出版社

目 次

第 1 章　Ethernet で広がる世界　5

1-1　インターネットに接続するには …… 5

第 2 章　付属マイコン基板をインターネットに接続してみよう　9

2-1　付属マイコン基板の概要 …… 9
2-2　プログラムの開発環境……11
2-3　付属マイコン基板のハードウェア……12
2-4　マイコン基板をネットワークに接続する……16
2-5　マイコン基板をインターネットに接続する……20
2-6　SilentC のコマンド・モードによるファイル操作 ……21
column　アップデートとリカバリ処理 ……24

第 3 章　マイコン基板を使ったネットワーク通信　27

3-1　UDP を使ってみよう ……27
3-2　UDP を利用した電圧計 ……29
3-3　HTTP サーバの CGI 機能を利用してみよう……31
3-4　SMTP サーバでメールを送信してみよう……35
3-5　POP サーバにアクセスしてみよう ……39
3-6　NTP サーバにアクセスして日時を取得してみよう ……41
3-7　パーソナル・リマインダの作成……44

第 4 章　ブレッドボードを使って周辺回路を接続する　49

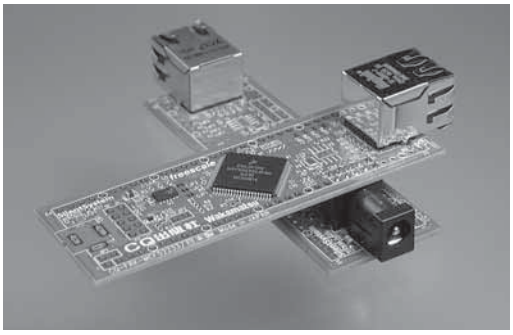
4-1　ブレッドボードを使ってみよう……49
4-2　Ethernet をシリアルに変換する……50
4-3　USB ジョイスティック……52
4-4　I²C で温度センサを接続する……55
4-5　サーミスタを接続する……56
4-6　BDM プローブを接続する……58
column　SilentC を機能拡張する……58

第 5 章　統合開発環境 CodeWarrior を使ったプログラミング　61

5-1　スイッチと LED を追加する……61
5-2　開発環境のインストール……62
5-3　Processor Expert の使い方……64
5-4　A-D コンバータによる加速度センサの利用……69
5-5　PWM の利用……72
5-6　フラッシュ・プログラミング……75

第 6 章　C インタプリタ SilentC の使い方　77

6-1　プロンプトからの使い方……77



第 7 章　付属マイコン基板搭載 MCF5223x シリーズ　83

7-1　ColdFire とは　……83

第 8 章　3 軸加速度センサ MMA73xxL ファミリ　87

8-1　3 軸加速度センサの概要　……87

索引　……91

筆者紹介　……95

本書掲載プログラムのダウンロードについて

本誌に掲載されているプログラムは下記 URL からダウンロードできます.

http://shop.cqpub.co.jp/hanbai/books/MIF/MIFZ201104.html

付属基板および CD-ROM は，月刊 Interface 2008 年 9 月号と同等品です.

第 2 章

付属マイコン基板をインターネットに接続してみよう

付属基板の特徴から基板の組み立て，基本的操作方法まで

ここでは付属マイコン基板に実装されている ColdFire マイコン MCF52233 の特徴から，基板の組み立て方法や動作確認方法を解説します．CPU 内蔵フラッシュ ROM に書き込み済みのシステム“SilentC”を操作して，MAC アドレスや IP アドレスの各種設定からファイル転送など基本的な操作方法について解説します．

2-1　付属マイコン基板の概要

本書に付属するマイコン基板の外観とブロック図を，**写真 2-1** と**図 2-1** に示します．この付属マイコン基板（以降，本基板あるいはマイコン基板）の最大の特徴は，Ethernet に接続できることです．Ethernet への接続が可能になれば LAN やインターネットを利用できるので，その応用範囲を飛躍的に広げられます．また，本基板は，実験や評価という枠を超えて，十分に実用的なアプリケーションを実現す

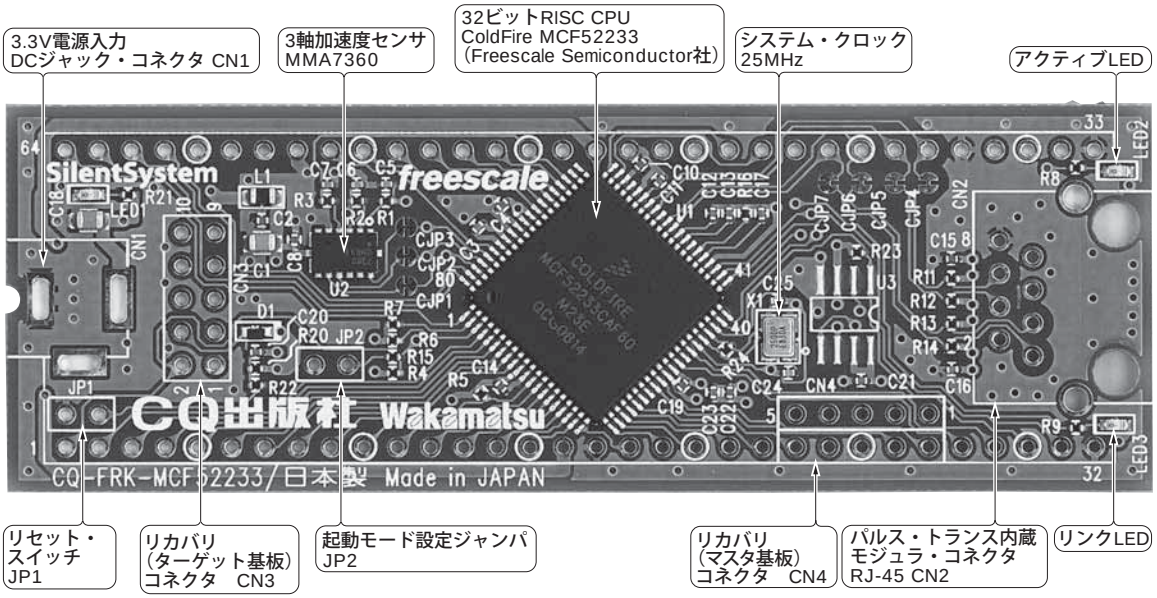


写真 2-1　マイコン基板の外形と各部の説明

る可能性も持っています．

本基板に使用しているのは，米国 Freescale Semiconductor 社[注1] の ColdFire（型名は MCF52233）というマイクロプロセッサです．ColdFire という名前を聞きなれない読者も多いと思いますが，米国 Motorola 社の代表的なマイクロプロセッサであった MC68000 の正当なる直系の子孫です．MC68000 は，米国 Intel 社の 8086 に比べるとエレガントなアーキテクチャであったため，それに魅せられたファンは現在でも数多く存在します．

本基板に使用している MCF52233 は，論理層はもちろん物理層まで含んだ Ethernet コントローラをチップ内に内蔵しているので，極めて簡単に Ethernet へ接続できます．また，内蔵メモリとして，書き

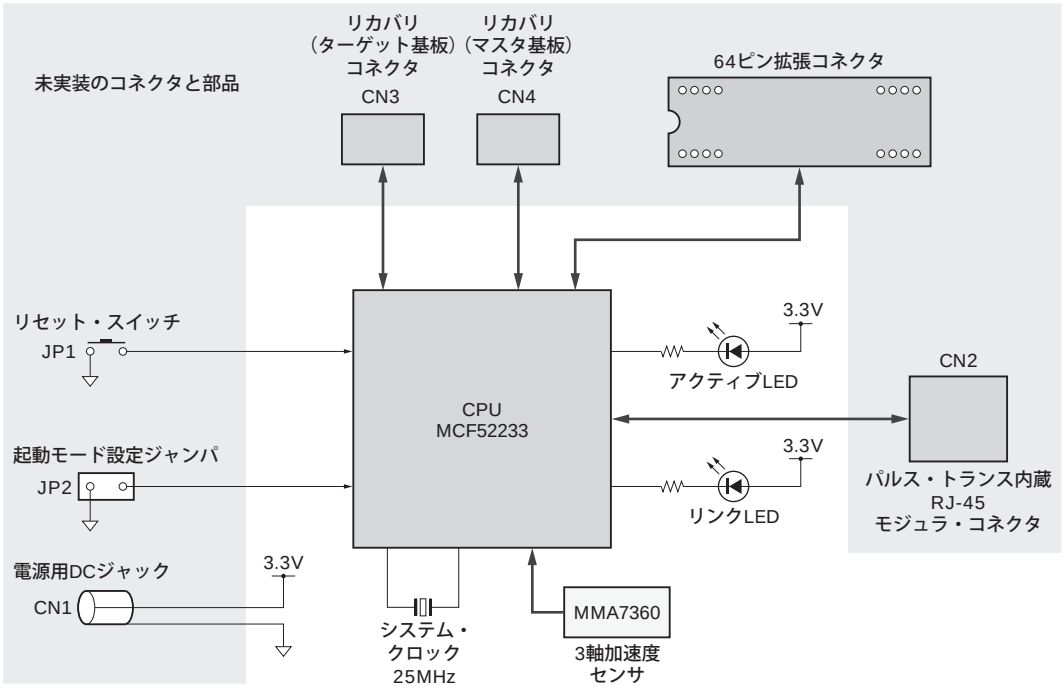


図 2-1　マイコン基板のブロック図

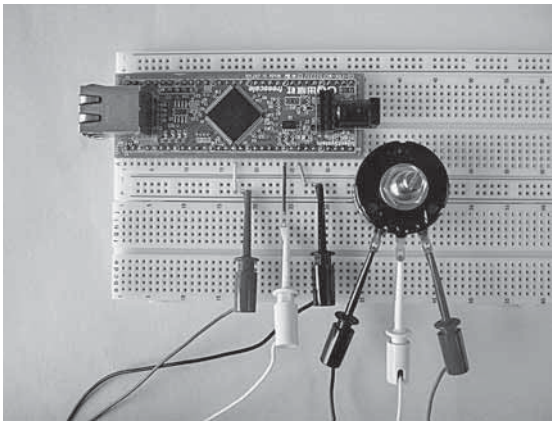


写真 2-2　マイコン基板はブレッドボードで実験ができる

注 1 ： Freescale Semiconductor 社は，Motorola 社の半導体部門が分離独立してできた会社．

換え可能な 256K バイトのフラッシュ ROM と 32K バイトの RAM を持ち，一般的なネットワーク対応アプリケーションを問題なく動作させられます．

さらに，本基板上には MMA7360 という 3 軸加速度センサも実装しています．したがって，加速度を検出することにより，この基板を動かす方向を判定したり，揺れを検出したりすることでいろいろな応用が可能になります．

本基板をよく見ると細長い形状をしていますが，この形状には意味があります．すなわち，いろいろなデバイスを接続して実験する際に便利なブレッドボードに対応できるようになっているのです．ブレッドボードに本基板を実装すると，すべてのピンに対してワイヤを差し込むだけで結線が可能です．これを利用すれば，さまざまなテーマを簡単に実験できるので教材としても使えます(**写真 2-2**)．

2-2　プログラムの開発環境

● 付属マイコン基板に書き込まれているソフトウェア

本基板には，CPU の内蔵フラッシュ ROM に 2 種類のソフトウェアが書き込まれています．MCF52233 の 256K バイトのフラッシュ ROM のうち，前半の 128K バイトには(有)サイレントシステムの C 言語インタプリタ・システム“SilentC”が，後半の 128K バイトにはデバッガ GDB と通信するのに必要な GDB スタブが書き込まれています(**図 2-2**)．リセット直後，マイコン基板上の JP2 がオープン状態であれば SilentC が起動し，ショート状態であれば GDB スタブが起動します(**写真 2-3**)．

Ethernet を活用するためには，どうしても TCP/IP スタックが必要です．SilentC には TCP/IP スタックが実装されているので，すぐにソケットを利用したネットワーク・プログラムを開発できます．また，会話型にプログラムを開発できる C 言語インタープリタも利用可能です．したがって，SilentC を利用すると，本基板だけで実用的なネットワーク・プログラムを簡単に作成できます．一昔前のマイコン

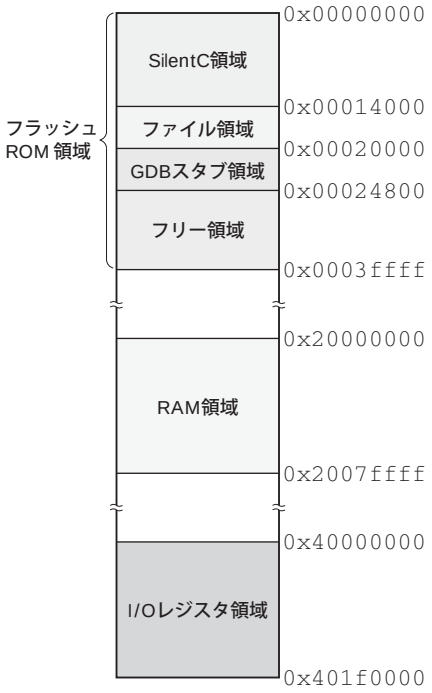


図 2-2　MCF52233 のメモリ・マップ

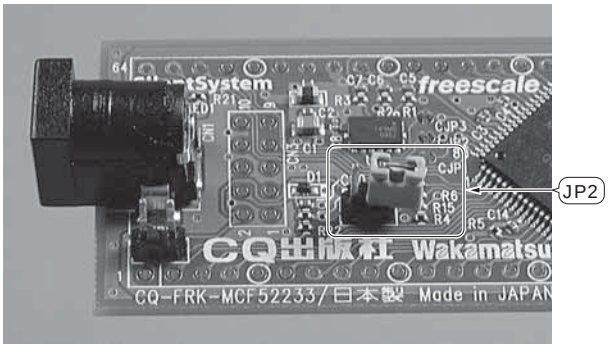


写真 2-3　ジャンパ JP2 の設定
JP2 を未実装またはオープン状態で起動すると SilentC が，ショート状態で起動すると GDB スタブが起動する．

第3章 マイコン基板を使ったネットワーク通信

UDP/HTTP/SMTP/POP/NTP を利用する

本書に付属するマイコン基板を実際にインターネットへと接続するためには，さまざまな通信プロトコルに従って通信を行う必要があります．ここでは実際に UDP や HTTP，SMTP などの通信プロトコルを使います．

3-1 UDP を使ってみよう

本書の付属マイコン基板(以降，マイコン基板もしくは本基板)をネットワークに接続できるようになったら，早速，実際にネットワーク通信を体験してみましょう．ネットワークでデータを送受信するためにはプロトコルを使用します．プロトコルには利用する目的によりさまざまなものがありますが，まず一番初めにチャレンジするのは UDP(User Datagram Protocol)です．

UDP は TCP(Transmission Control Protocol)と並ぶインターネットの代表的なプロトコルです．IP アドレスを管理する IPv4 プロトコルの上で利用されるので，UDP/IP や TCP/IP とも呼ばれています．UDP と TCP の違いは，確実に受信側がデータを受け取ったことを送信側に知らせるかどうかです．TCP はデータの受信に失敗すると，送信側に再送を要求しますが，UDP は送りっぱなしです．そのため UDP は TCP に比べると高速にデータを送信できます．したがって，文書ファイルなどのようにデータを欠落させず確実に相手側に送りたいときは TCP を利用しますが，音声や動画のリアルタイム送信などのように，多少データが欠落してもよいときは UDP を利用します．

ただし，UDP もデータを何度も送信したり確認のパケットを受信したりすることによって，実用的なデータ交換ができます．そこで，まず UDP を使ってデータをマイコン基板からパソコンに送信してみましょう．

CQ 出版社のダウンロード・ページからプログラム・ソース集をダウンロードして解凍します．その中の 3-1 というフォルダ内の Main を tftp を使ってパソコンからマイコン基板に転送してください(tftp は UDP 上でファイルを転送するためのプロトコルであり，tftp がデータの確認応答をしている)．ただし，既にマイコン基板上には Main が存在しているので，転送する前に Main を削除する必要があります．または，telnet でマイコン基板に接続して**リスト 3-1** のプログラムをそのまま入力してもかまいません．その後，

```
save Main
```

で入力したプログラムを Main という名前で保存します．このどちらの方法で Main を作成してもかまいませんが，list コマンドを使って**リスト 3-1** に示すプログラムが表示されるようにしてください．

このプログラムは，0 から 9999 までの数字を指定された IP へ連続的に送信します．次に，送信された UDP データをパソコン側で受け取って表示するソフトウェアを用意します．米国 Microsoft 社の Web サ

リスト 3-1　0 ～ 9999 までの数字を指定された IP に連続的に送信するプログラム

```
10 main(char *s){char *d,soc=CreateSocket(0);long ip=GetIP(s);int n;   //引数から IP アドレスを得る
20 d=MemoryAlloc(6);for(n=0;n<10000;n++){GetDigit(n,d);               //数字バッファを確保して 10000 までのループ開始
30 if(SendTo(soc,ip,30049,d,StrLen(d))<=0)break;}                     //数字を UDP で送信する
40 CloseSocket(soc);MemoryFree(d);}                                   //ソケットと数字バッファを開放
```

リスト 3-2　UDP 通信を実験するプログラム（VisualC# でコンパイル）

```
using System;
using System.Collections.Generic;
using System.Text;
namespace ConsoleApplication1
{
 class Program
 {
  static void Main(string[] args)
  {
   System.Text.Encoding enc = System.Text.Encoding.UTF8; //UTF8 を指定する
   System.Net.Sockets.UdpClient udp =                    //ローカル・ポート番号をバインドする
    new System.Net.Sockets.UdpClient(30049);
   for(;;)                                               //データを受信するループ
   {
    System.Net.IPEndPoint EndPoint = null;               // UDP エンドポイントを用意する
    byte[] rcvBytes = udp.Receive(ref EndPoint);         // UDP データを受信する
    string rcvMsg = enc.GetString(rcvBytes);             // 受信したデータから文字列に変換
    Console.Write("送信元アドレス:{0}/ポート番号:{1} /受信したデータ:{2}   \r",
     EndPoint.Address, EndPoint.Port, rcvMsg);           // 到着したデータの表示
    if(Console.KeyAvailable)                             // キーをセンス
     break;                                              // ループを抜ける
   }
   udp.Close();                                          //UDP 接続を終了
  }
 }
}
```

イトから無料でダウンロードできる Visual C# 2005 Express Edition をパソコンにインストールして，**リスト 3-2** のプログラムを入力してコンパイルします．もし，この作業が面倒であれば，3-1 フォルダ内に既にコンパイルされた UDP_Disp.exe があるので，それを利用してください．なお，このプログラムを実行するには .NET Framework 2.0 が必要となるので，あらかじめパソコンにインストールしておく必要があります．

UDP 通信の実験を始める前に，パソコンの IP アドレスを調べておいてください．コマンド・プロンプトから ipconfig を実行すると次のような画面が表示されるので，パソコン側の IP アドレスを調べられます．

```
C:¥>ipconfig
Windows IP Configuration
Ethernet adapter ローカル エリア接続:
        Connection-specific DNS Suffix  . :
        IP Address. . . . . . . . . . . . : 192.168.1.2
        Subnet Mask . . . . . . . . . . . : 255.255.255.0
        Default Gateway . . . . . . . . . : 192.168.1.1
```

この操作により，パソコンの IP アドレスが 192.168.1.2 だということが分かります．これで準備完了です．そこで，パソコン側で UDP_Disp.exe を実行します．初回の起動時に Windows のファイアウォールからの警告ダイアログが出るので，「ブロックを解除する」をクリックしてください．

次に，telnet でマイコン基板に接続し，OK プロンプトを確認して以下のように入力します．

```
main("192.168.1.2")
OK
```

main に与える IP アドレスは，先ほど調べたパソコンの IP アドレスに合わせて変更してください．パソコンのコマンド・プロンプト画面上に 0 から 9999 までの数字が表示されます．

10,000 個のパケットを送信する時間を測定すると，1 個当たりの送信時間を算出できます．筆者の環境では 27 秒ほどかかったので，1 パケット当たり 2.7ms という計算になります．

Main(**リスト 3-1**)の動作は，まず与えられた引数から IP アドレスを得ます．次に，数字を格納するためのバッファを 6 バイト確保して，for で 0 から 9999 までループします．ループの中ではループ変数を数字文字列に変換して，SendTo を用いて対象とする IP アドレスの 30,049 番ポートへ数字を送信します．ループが終了したらソケットを開放し，数字を格納するバッファを開放して終了しています．

UDP 通信は，相手の IP アドレスとポート番号さえ指定すれば，いつでもデータを送信できます．しかし，相手がデータを受け取ったかどうかは，送信側では全く分かりません．もし，確実に相手がデータを受け取ったかどうかをチェックしたいなら，相手からもデータを受け取ったという確認の信号を UDP で送信する必要があります．

3-2　UDP を利用した電圧計

MCF52233 には A-D 変換器が内蔵されています．そこで，UDP を利用して，この A-D 変換器で得られた値をパソコンの画面上でアナログ・メータのように表示させてみましょう(**図 3-1**)．UDP でデータを転送するのは前述の例と変わりありませんが，ここではマイコン基板上に実装されている加速度センサから出力される電圧を A-D 変換して，そのデータを UDP で送信します．基板を傾けると，それに応じてパソコンのメータの針が動くのはなかなか楽しいものです．

パソコン側のソフトウェアは，Windows アプリケーションです．皆さんの環境でコンパイルするためには，リソースを含むプロジェクト・ファイルのすべてが必要になりますが，3-2 フォルダ内に，Visual C# 2005 Express Edition 向けのプロジェクト・ファイルが入っています．実行部分のソース・ファイルを**リスト 3-3** に示します．

プログラムの動作を説明します．まず，マイコン基板から UDP データを受け取って udpval という変数に格納するというスレッドを起動しておきます．そして，タイマ割り込みを利用して，200ms 間隔で udpval の値に従って，アナログ・メータ風の画面をウィンドウに描画します．このアナログ・メータをそのまま利用する場合は，3-2 フォルダ内にある Analog.exe を実行してください．このとき，Visual C# 2005 Express Edition をインストールする必要はありませんが，プログラムを実行させるには.NET Framework 2.0 が必要になるので，あらかじめインストールしておいてください．

前述したように，初回の起動時に Windows のファイアウォールからの警告ダイアログが出るので，「ブロックを解除する」をクリックします．このアナログ・メータのプログラムは，常に UDP パケットを待っているので，終了させる際には UDP パケットを受信している最中に終了させてください．そうしないと，タスク・マネージャで Analog.exe を探してプロセスを終了させる必要があります．

次に，telnet でマイコン基板に接続して OK プロンプトを確認して，以下のように入力します．

```
ad::udpsend("192.168.1.2")
```

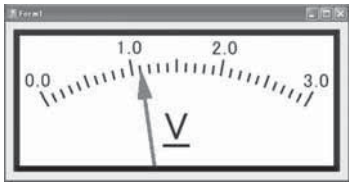


図 3-1
UDP で送信されたデータを表示する電圧計

第6章 Cインタプリタ SilentCの使い方

BASIC のように対話的にコマンドを入力できる

付属マイコン基板には C 言語インタプリタ「SilentC」が搭載されています．これは C 言語の文法でありながら，コンパイルすることなくプログラムを実行できるインタプリタ言語です．本章では，この SilentC の使い方を説明します．

6-1　プロンプトからの使い方

● Main プログラムの実行/修正

付属マイコン基板上には，出荷時から SilentC が管理するファイル領域があります．これに書き込まれている Main ファイルには，簡単なサンプルが含まれています．“OK”というプロンプトが出ているときに run と入力すると，画面におなじみの“Hello World!!”が表示されます．

```
run↵
Hello World!!
OK
```

この Main ファイルの中身をのぞいてみましょう．ファイル内容の表示には type コマンドを使います．SilentC で利用できるコマンドに関しては第 2 章の表 2-3 を参照してください．

```
type Main↵
main(){PrStr("Hello World!!¥r¥n");}
OK
```

表示されたのは最もシンプルな C 言語のプログラムです．main()関数の中で呼ばれている PrStr という関数は，文字列を表示するためのライブラリ関数です．SilentC に用意されているライブラリ関数に関しては**表 6-1** を参照してください．

それでは，この Main を編集してみましょう．まず，load コマンドで編集したいファイルを編集モードにします．

```
load Main↵
Main loaded↵
OK
```

list コマンドは，編集モードにしたファイルの内容を行番号を付加して表示します．

```
list↵
10 main(){PrStr("Hello World!!¥r¥n");}
OK
```

先頭の 10 は BASIC の行番号と同じ働きをします．先頭に数字を付けてテキストを入力すると，その

表 6-1　SilentC に用意されているライブラリ関数

void PrStr(char *str)
コンソールへの文字列出力
void PrChar(char c)
コンソールへの 1 文字出力
void PrNum(long num)
コンソールへの 10 進数値出力
void PrHex(long num)
コンソールへの 16 進数値出力(32 ビット)
void PrHexWord(int num)
コンソールへの 16 進数値出力(16 ビット)
void PrHexByte(int num)
コンソールへの 16 進数値出力(8 ビット)
void PrAdrs(long adrs)
コンソールへの IP アドレス出力(32 ビット)
void GetDigit(long num, char *buf)
数値を文字列に変換する．結果は buf に格納される
long Atoi(char *buf)
文字列を数値に変換する
char Gets(char *buf, char len)
コンソールから 1 行入力する．結果は buf に格納される．len は最大サイズ
char Getc(char wait)
wait が 1 ならコンソールから 1 文字入力するのを待つ．wait が 0 ならキーセンスしてすぐに戻る(#stop 0 が必要)

(a) コンソール・ライブラリ

void *MemoryAlloc(int size)
size の大きさのヒープ領域を確保してそのポインタを返す
void MemoryFree(void *memory)
MemoryAlloc で確保した領域を開放する
void BufCopy(void *dest, void *src, int size)
src から dst に向けて size バイト分コピーする
void MemClear(void *dest, int size)
dest から size バイトをクリアする

(b) メモリ操作ライブラリ

void Sleep(int time)
time × 10ms の間実行を中断する
void SystemSleep(void)
自分のスレッドを中断してほかのスレッドに制御を移す
char CreateTimer(char timer, int interval, char *func)
タイマを作成する．timer はタイマ・ハンドルで 0 なら新規，それ以外は既存のタイマ・ハンドル．interval は 10ms 単位のカウント時間，func は通常は 0 を指定．文字列でファイル名::関数名を指定するとタイマが 0 になったらその関数を呼び出す．タイマ・ハンドルを返す
int GetTimerCount(char timer)
タイマの現在のカウント値を返す．timer は調べるタイマ・ハンドル

(d) タイマ・ライブラリ

char CreateSocket(char prot)
prot が 0 なら UDP ソケットを 1 ならタイムアウトなし TCP/IP ソケットを作成する．2 以上の場合にはその数値× 10 秒をタイムアウトとする TCP/IP ソケット作成．ソケット・ハンドルを返す．その後のネットアクセスにはすべてこのハンドルを指定
void CloseSocket(char socket)
ソケット・ハンドルを与えてソケットを開放する
long GetSenderIP(char socket)
到着したパケットの送信元の IP アドレスを得る
int GetSenderPort(char socket)
到着したパケットの送信元のポート番号を得る
int SendTo(char socket, long ip, int port, char *buf, int len)
UDP データを送信する．socket：ソケット・ハンドル(以下同)，ip：相手先の IP アドレス(以下同)，port：相手先のポート番号(以下同)，buf：送信データへのポインタ，len：送信するバイト数．正常に送信されれば送信されたバイト数を返す．エラーの場合は負の値
int RecvFrom(char socket, int timeout)
UDP データグラムを受信する．timeout=10ms 単位のタイムアウト値(以下同)．成功なら受信したバイト数，エラーなら－1，タイムアウトなら－2 を返す
char *GetReceiveBuffer(char socket, char release)
受信したデータへのポインタを得る．release ＝ 0 ならバッファを保留，0 以外ではバッファを開放する．受信データはヒープ領域に置かれる．データを処理し終わったら必ず MemoryFree を呼び出して領域を開放すること．
char Bind(char socket, int port, char maxsoc)
ソケットにポート番号を関連付ける．port：ソケットに結び付けるローカル・ポート番号，maxsoc：常に 1 を指定．
char Accept(char socket, int timeout)
TCP 接続を待つ(サーバ・モード)．TCP 接続要求があればセッションを確立して新しいソケットのハンドルを返す．無効ハンドルなら－1，タイムアウトは－2，セッションを確立できなかった場合は－3 を返す
char Connect(char socket, long ip, int port)
TCP 接続を要求する(クライアント・モード)．成功すると 1 を，エラーが発生した場合には負の値を返す
int Write(char socket, char *buf, int len)
TCP/IP でデータを送信する．buf：送信データへのポインタ，len：送信するバイト数．正常に送信されれば送信されたバイト数を返す．エラーの場合は負の値．送信後は WaitWriteComplete で相手に到着したかどうかを確認する必要がある
int Read(char socket,int timeout)
TCP/IP データを受信する．成功なら受信したバイト数，エラーなら－1，タイムアウトなら－2 を返す
int GetNetLine(char socket, char *buf, char size, char func)
TCP/IP で 1 行入力する．buf：読み込んだ行を格納するポインタ，size：バッファの最大サイズ，func ＝ 0 なら初期化，1 なら終了時の作業メモリ開放，2 以上ならタイムアウト値を指定．このライブラリはスタティック変数を使用しているのでリエントラントではない．正常に受信されれば受信したバイト数を返．エラーの場合は負の値
char CheckWriteComplete(char socket)
TCP/IP で送信したデータが相手に受け取られたかチェックする．完了すれば 1 を，未完なら 0 を，エラーなら負の値を返す
char WaitWriteComplete(char socket)
TCP/IP で送信したデータが相手に受け取られるまで待つ．エラーが発生したら負の値を返す
int GetDefMtu(char socket, int size)
TCP/IP の際の MTU 値を size に設定する．size に 0 を与えると現在の値を返す
long GetHostByName(char *name)
name で示される名前のサーバの IP アドレスを返す．エラーの際には 0 を返す
long GetIP(char *ipstr)
文字列から 32 ビットの IP アドレス(long)に変換する

(c) ソケット・ライブラリ

char SciSense()
シリアル・ポート・センス．何か入力キューに入れば 1 を返す
char SciGetc()
シリアル・ポートから 1 文字入力
void SciPuts(char *str)
シリアル・ポートに文字列を出力する
void SciPutc(char ch)
シリアル・ポートに 1 文字出力する
int SciGets(char *buf, int max)
シリアル・ポートから 1 行入力する．結果は buf に格納される．最大長は max

(e) シリアル・ライブラリ (COM1，57600pbs)

int StrLen(char *str)
文字列の長さを得る
char *StrCpy(char *buf, char *str)
str から buf に文字列をコピーする．buf を返す
char *StrCat(char *head, char *tail)
文字列 head の末尾に文字列 tail を連結する．head を返す
int StrCmp(char *str1, char *str2)
文字列 str1 と文字列 str2 を比較する．0 なら同一文字列
int StrChr(char *str, char scan)
文字列 str の中に文字 scan があるかどうか調べる．なければ 0 を返す
char *StrStr(char *str, char *scan)
文字列 str の中に文字列 scan があるかどうか調べる．なければ 0 を返す

(f) 文字列操作ライブラリ

void InitAd(char portmask)
A-D 変換の初期設定を行う．portmask で指定したビット位置の A-D 変換チャネルを使用する
int GetAd(char portbit)
指定されたチャネルの A-D 変換データを入力する

(h) アナログ・ライブラリ

int UserDriver(char vec, int arg)
ユーザの作成した ColdFire のネイティブ・コードを実行する．vec はドライバを選択する番号，arg はドライバに渡すパラメータで a0 に渡される．SilentC の起動時に UserDriver.bin というファイルが存在していれば 0x13000 番地からファイルの内容をバイナリでプログラムする．領域の先頭には 4 バイトのアドレスを列挙していく．vec で 0 を指定すると 0x13000 番地に格納されているアドレスを参照して呼び出す．1 を指定すると 0x13004 番地の 4 バイトをアドレスとして参照する (先頭にジャンプ・テーブルを置くということになる)．ドライバからの戻り値は d0 に返す

(i) ユーザ・ライブラリ

int OpenFile(char *name)
ファイルを読み込みモードでオープンする．成功するとファイル・ハンドルを返す．エラーは負
int ReadFile(int handle, char *buf, int size)
handle で指定されるファイルから size バイト読み込む．結果は buf に格納
int SeekFile(int handle, long pos, char org)
handle で指定されたファイルの読み込み位置を pos に設定． org は 0 なら先頭から，1 なら現在位置から，2 なら終端からの位置の指定になる
int FileGets(int handle, char *buf, char size)
ファイルから 1 行読み込む．結果は buf に格納される．size は最大長の指定
void CloseFile(int handle)
ファイルを閉じて内部の作業用のメモリを開放する．
long GetFileSize(char *name)
ファイルのサイズを得る．存在しないファイルなら 0 を返す
char *FindFile(char init)
ファイルのリストを得る．init に 1 を指定して呼び出すと最初のファイル名へのポインタを返す．以後は init=0 で呼び出すことで次のファイル名を返す．最後は 0 を返す．返されたファイル名へのポインタはヒープ領域に確保されているので利用後に開放すること
char CreateFile(char *name)
ファイルを書き込みモードでオープンする．成功すると 0 を返す．同時に書き込めるファイルは 1 個だけなのでハンドルは 0 に固定される
int WriteFile(char *buf, int size)
ファイルに buf から size バイトのデータを書き込む．書いた長さを返す
int FilePuts(char *str)
ファイルに文字列を書き込む．書いた長さを返す
void CloseFile(0)
書き込みモードでオープンされたファイルを閉じる．ハンドルとして 0 を指定する
char DeleteFile(char *name)
ファイルを削除する．見つからないときには 1 を返す
char MoveFile(char *oldname, char *newname)
ファイルをリネームする． 成功すると 0 を，エラーなら正の値を返す
long GetFinalPos(void)
ファイル・システムで使用している最終位置を返す
void DefragFilesys(int key)
ファイル・システムを最適化する．プログラム中で使用する場合には自分自身が最適化で位置が移動しない場合にのみ使用可能．key には 16787 を指定しないと動作しない

(g) ファイル・ライブラリ

見本

インターネットに接続できる32ビット・マイコン

すぐにつながる！ Ethernetマイコン・モジュール

雑誌　01620-4
Ⓛ-2011.5/25

4910016200416
02800

インターフェース 増刊 すぐにつながる！ Ethernet マイコン・モジュール

CQ出版社
〒170-8461 東京都豊島区巣鴨1-14-2
TEL.(03)5395-2141(販売部)

定価2,940円 本体2,800円