マイコンカーラリーキット Ver.4 対応

データ解析
実習マニュアル
kit07版

第 1.26 版

2008.09.01

ジャパンマイコンカーラリー実行委員会

注 意 事 項 (rev.1.2)

著作権

・本マニュアルに関する著作権はジャパンマイコンカーラリー実行委員会に帰属します。
・本マニュアルは著作権法および、国際著作権条約により保護されています。

禁止事項

ユーザーは以下の内容を行うことはできません。
・第三者に対して、本マニュアルを販売、販売を目的とした宣伝、使用、営業、複製などを行うこと
・第三者に対して、本マニュアルの使用権を譲渡または再承諾すること
・本マニュアルの一部または全部を改変、除去すること
・本マニュアルを無許可で翻訳すること
・本マニュアルの内容を使用しての、人命や人体に危害を及ぼす恐れのある用途での使用

転載、複製

本マニュアルの転載、複製については、文章によるジャパンマイコンカーラリー実行委員会の事前の承諾が必要です。

責任の制限

本マニュアルに記載した情報は、正確を期すため、慎重に制作したものですが万一本マニュアルの記述誤りに起因する損害が生じた場合でも、ジャパンマイコンカーラリー実行委員会はその責任を負いません。

その他

本マニュアルに記載の情報は本マニュアル発行時点のものであり、ジャパンマイコンカーラリー実行委員会は、予告なしに、本マニュアルに記載した情報または仕様を変更することがあります。製作に当たりましては、こと前にマイコンカー公式ホームページ(http://www.mcr.gr.jp/)などを通じて公開される情報に常にご注意ください。

連絡先

ルネサステクノロジ　マイコンカーラリー事務局
〒162-0824　東京都新宿区揚場町 2-1　軽子坂MNビル
TEL　(03)-3266-8510
E-mail:official@mcr.gr.jp

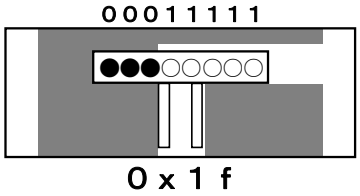
目　次

1. EEP-ROM(24C256)を使ったデータ解析

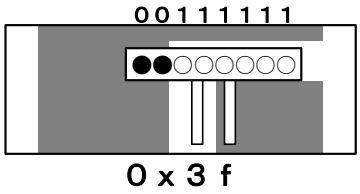
1.1 概要

マイコンカーを走らせると、脱輪することがあります。なぜ脱輪するのか… もちろん、回路の間違いやプログラムの文法的な間違いは、直さなければいけません。しかし、それらがきちんとできていても脱輪することがあります。これは、コースの検出状態や、スピード(エンコーダの値)など、想定とは違う状態になるからです。

例えば、kit07 では、右クランクと判断するセンサ状態は「0x1f」の状態です。



しかし、たまに右クランクをそのまま通過してしまい、脱輪することがありました。そのため、これから紹介する方法で脱輪したときのセンサの状態を 10ms ごとに記録、パソコンで解析してみました。すると、下図のように「0x1f」ではない状態で右クランクを検出していることが判明しました。



プログラムは、「0x3f は右クランクなので右に曲がりなさい」という内容が入っていません。そのため、そのまま進んでしまうのです。脱輪してしまいますが、マイコンカーはプログラムどおりに動いているだけです。脱輪しないためには、「0x3f」になったならどうしないといけないか、プログラムを追加しなければいけません。

最近のマイコンカーは速度が速くなり、センサの状態を目で見て確認することは難しくなってきました。「カン」に頼っても、分からないものは分かりません。データを記録することにより、「カン」に頼らない論理的な解析ができ、プログラムに反映させることができます。

ただし、プログラムに反映させるためには、**自分が想定しているマイコンカー(センサ)の状態とプログラムを理解していなければいけません**。

・自分が想定しているセンサの値に対して、プログラムはこうなっている
・だから脱輪してしまう
・そのためには、ここのプログラムを直さなければいけない

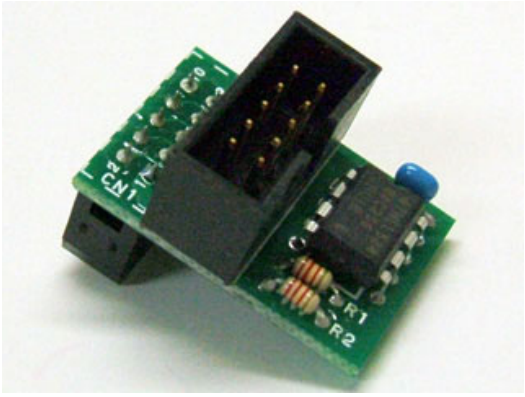
というように、データ解析を有効活用するためには、制御プログラムの理解が不可欠です。データ解析はあくまで、プログラムをデバッグするための補助ツールなのです。

本マニュアルでは、
・データの記録方法
・パソコンへの転送方法
・解析方法
を紹介していきます。最後に、実例を紹介します。

1.2 EEP-ROMとは？

本書では、データ解析を行うためにEEP-ROMというICを利用します。EEP-ROMは、電気的に内容を書き換えることができるROMでイーイーピーロムと読みます。EEP-ROMは、Electronically Erasable and Programmable Read Only Memoryの略称です。ROMなので、電源を切っても内容は消えません。

本書では、I2C(アイ・スクエア・シィ)バスインタフェース方式のEEP-ROMである「24C256」という型式のIC(ディップの8ピン)を使用します。



▲市販されているEEP-ROM搭載の基板

1.3 EEP-ROMを使う意義

H8/3048F-ONEには内蔵RAMが4KBあります。そのうち1.5KB程度はプログラムで使用しますが、残りの2.5KB程度は空いています。このメモリをデータ記録に使用すれば、わざわざEEP-ROMを買って基板を作る必要はありません。なぜ、そこまでしてEEP-ROMを使う必要があるのでしょうか。下の表に長所、短所をまとめてみました。

記憶メモリ	マイコン内蔵RAM	外付けEEP-ROM(24C256)
記憶容量	2.5KB程度	**EEP-ROM　1個 32KB** 24C256を4つ接続すれば 4倍(128KB)まで対応可能
長所	**H8/3048F-ONE 内蔵のメモリを使用するため、手軽に利用できる**	8ピンのディップICで基板作成が容易にできる 4 つまで増設可能(32KB×4)で保存容量を増やすことができる **電源が消えてもデータが消えない！**
短所	容量が少ない 電源を切ると消えてしまう	**1回データ書き込み後、最大 10ms 間はEEP-ROMへアクセスできない(1回に1～64バイトのデータを書き込み可能)**

EEP-ROM(24C256)を使う意義は、

・記憶容量が32KBもある

・電源が消えてもデータが消えない

というのが最大の理由です。短所は、EEP-ROMへデータ書き込み後、最大10ms間アクセスできません。そのため、最短でも10msごとの書き込みしかできません。ただし、マイコンカーでのデータ記録には十分です。

参考までに記録時間を計算してみます。10msごとに8個のデータを保存することとします。

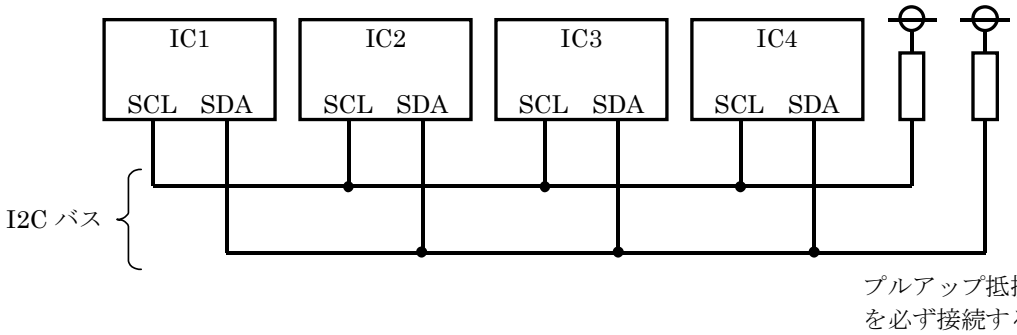
内蔵 RAM が保存できる容量 2.5KB　÷1 回の保存数 8 個×保存間隔 10ms＝3.2 秒
EEP-ROM が保存できる容量 32KB　÷1 回の保存数 8 個×保存間隔 10ms＝41.0 秒

内蔵 RAM の場合は、たったの 3 秒分しか記録できません。EEP-ROM は 41 秒分も記録できますので、地区大会レベルのコースなら 1 周分は記憶することができます。
このような理由から、EEP-ROM 基板を作り、データを保存します。

1.4 EEP-ROMへ読み書きする仕組み

(1) I2Cバスインタフェース方式

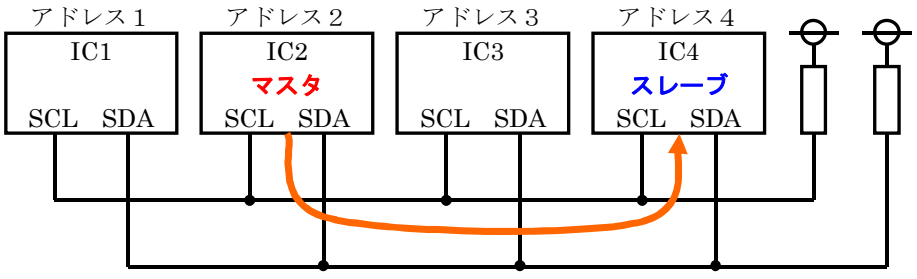
EEP-ROM は、24C256 を使用します。
24C256 は 2 本の線を CPU と接続します。2 本の線でシリアル通信を行って、データの書き込みや読み込みを行います。
使用するシリアル通信の方法は「I2C(アイ・スクエア・シィ)バスインタフェース方式」という通信方式です。この方式は、フィリップス社が考案した方式です。通信は、SDA(serial data)とSCL(serial clock)と呼ばれる2本の信号を使用して行います。IC 間をこの信号線で数珠つなぎに接続していきます(下図)。詳しい説明は書籍やインターネットに掲載されていますので省略しますが、ここでは概要だけ紹介します。



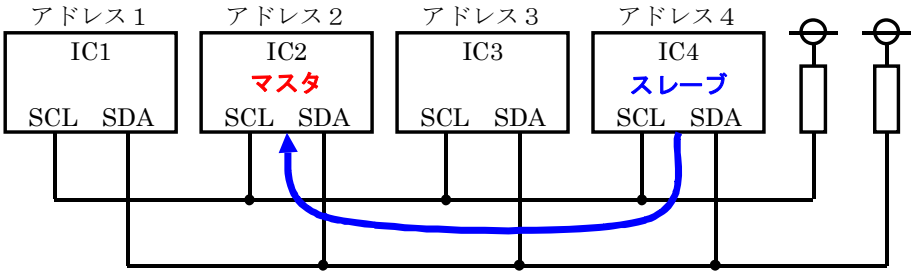
I2C とは「Inter Integrated Circuit」の略称です。「Integrated Circuit」は「集積回路(単に IC でも良いと思います)」です。「inter～」は「～の間」という意味になります。通信ということを考えると、「IC 間通信」というような意味合いになります。主に同一基板内などの近距離に配置された IC 間での高速通信(100Kbps/400Kbps/ 3.4Mbps)を行うための方式です。IICと表記される場合もあります。「I2C」読み方は、「アイ・スクエア・シィ」と読みます。そのまま「アイ・ツー・シィ」と読みがちですが、この読み方は間違いです。

(2) マスタ・スレーブ方式

I2C は、マスタ・スレーブ方式という方法で通信を行います。これは、マスタ(主)がスレーブ(従)へ命令を送り、スレーブ(従)からマスタ(主)へ命令された内容を返し、データのやり取りを行います。
バスは1本ですが、IC は複数有ります。まず命令を出したい IC がマスタとなり、対象となる IC のアドレスを電文に含めて電文を送ります。今回はアドレス 4 とします(下図)。

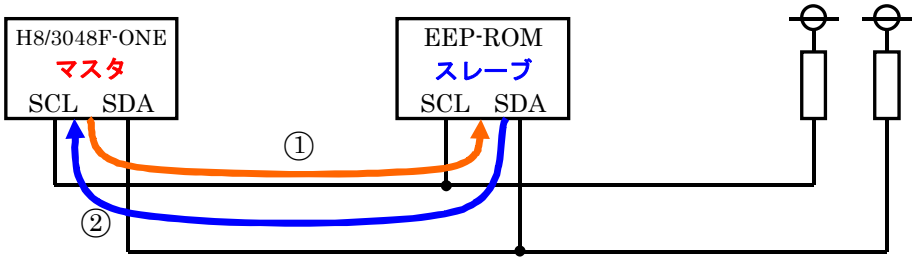


アドレス 4 の IC が通信できる状態なら、命令に対する電文を返信します(下図)。



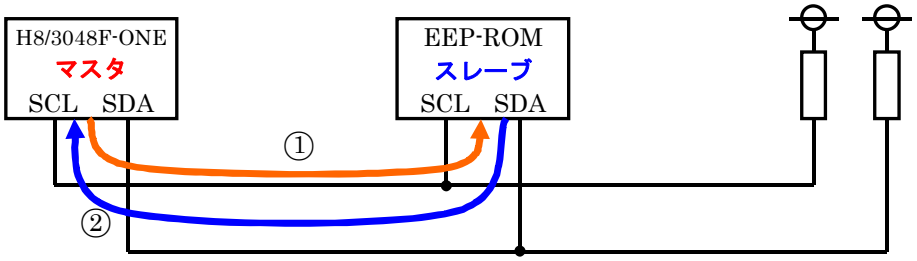
このように、マスタになった IC がバスを占有します。スレーブは、マスタの質問に答える形でメッセージを返します。勝手に、メッセージを送ってはいけません。また、マスタでもスレーブでもない IC1(上図のアドレス 1 の IC)や IC3(上図のアドレス 3 の IC)メッセージが来ても無視していなければいけません。もしマスタになりたいなら、現在の通信が終わるのを待ってから、やり取りしたい IC へ電文を送信します。

(3) 24C256 へデータを書き込むとき



1. H8(マスタ)が EEP-ROM(スレーブ)へ書き込む番地とデータを送ります。
2. EEP-ROM が命令を受け取ると、H8 へ命令を受け取った旨を返信し書き込み作業を行います。

(4) 24C256 へデータを読み込むとき

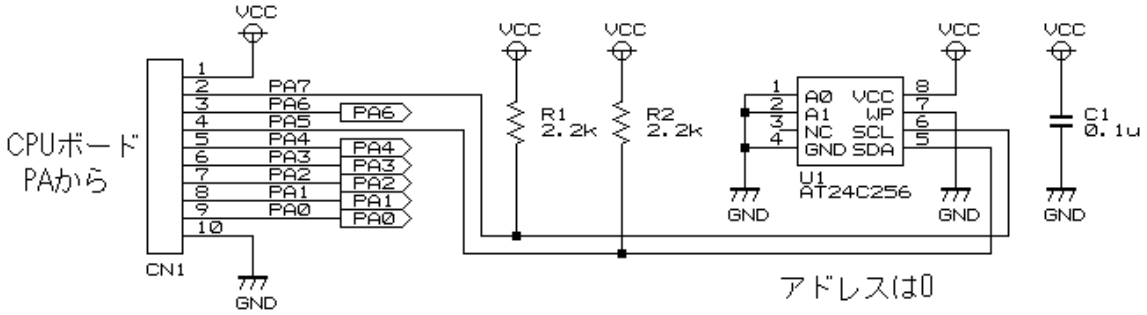


1. H8(マスタ)が EEP-ROM(スレーブ)へ読み込む番地を送ります。
2. EEP-ROM が命令を受け取ると、H8 へ指示された番地のデータを返信します。

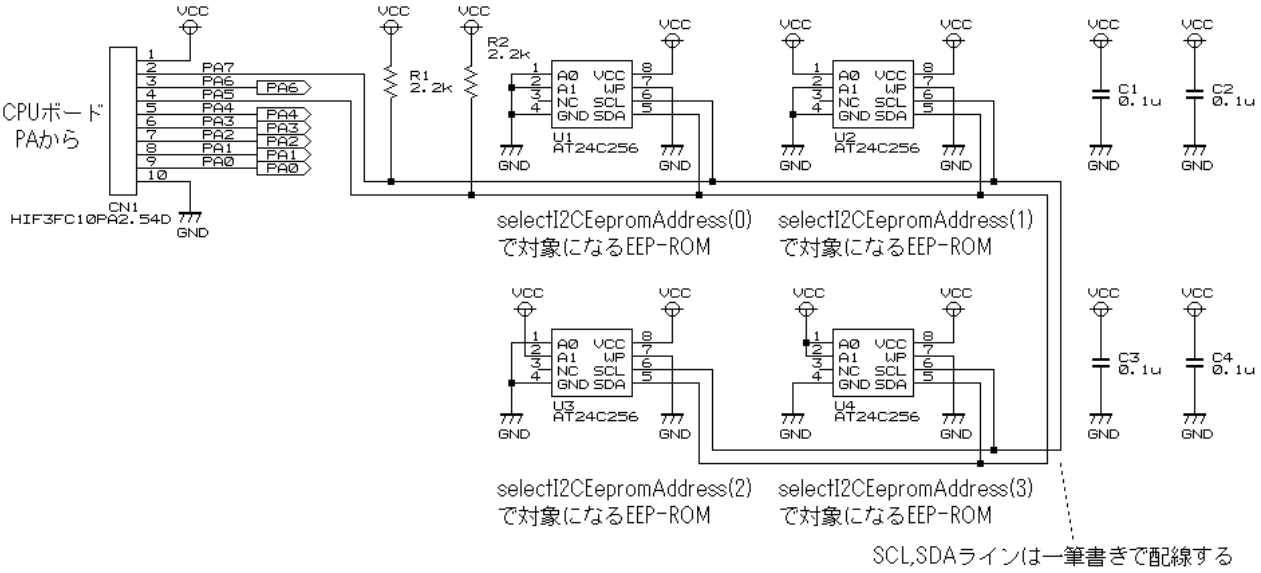
1.5 EEP-ROMの回路

EEP-ROM は、ポート A へ接続します。

番号	部品番号	部品名	型式・仕様	メ　ー　カ	数量	備考
1	U1	EEP-ROM	24C256	ATMEL など	1	CN1 の 10 ピンメスコネクタは、秋月電子で販売されている安価なコネクタでも対応可能です
2	R1,2	抵抗	2.2kΩ		2	
3	C1	積層セラミックコンデンサ	0.1μF 程度		1	
4	CN1	2×5 ピンメスコネクタ	HIF3FB-10DA-2.54DSA(71)	ヒロセ電機(株)	1	
5		ユニバーサル基板	7×9 ピッチ分		1	



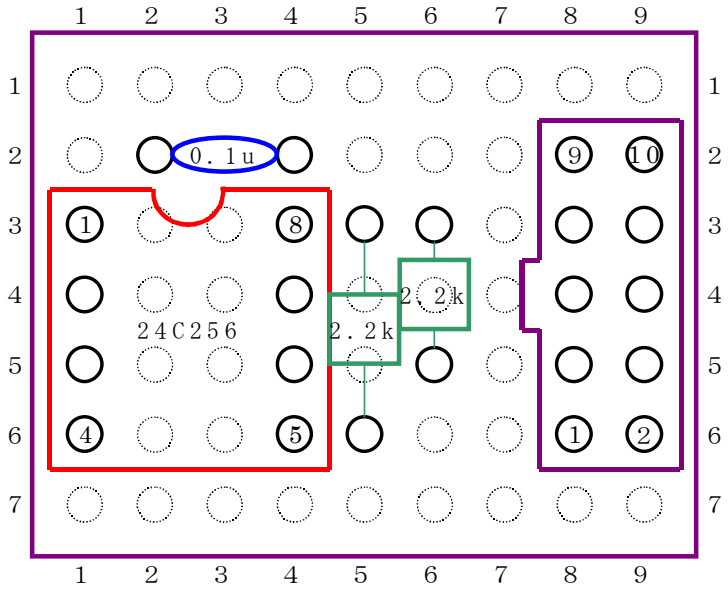
24C256　1個で 32KB のメモリ容量があります。足りない場合は下記のように A1,A0 の接続を変えて、増設することができます。下図に 4 個の 24C256 を接続した回路例を示します。



※どの EEP-ROM に保存するかは、selectI2CEepromAddress 関数で選択します。

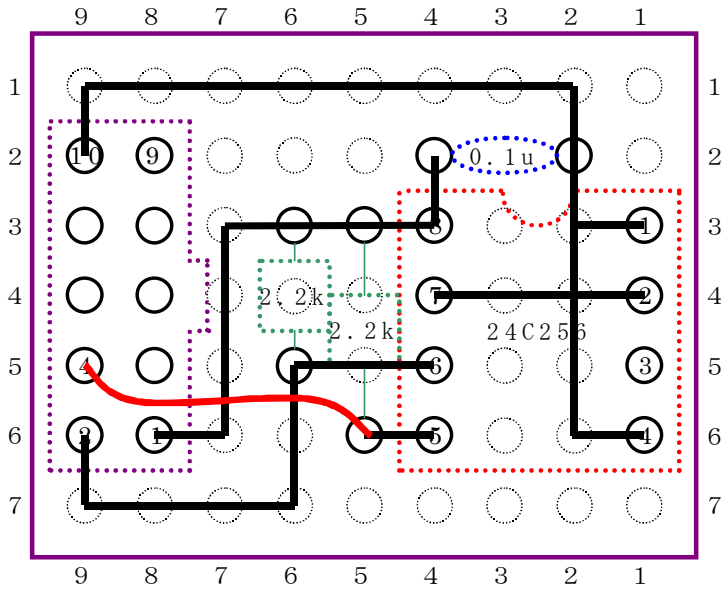
1.6 EEP-ROM基板の自作

EEP-ROM 基板を製作してみましょう。まず、部品配置を考えます。ユニバーサル基板（穴あき基板）を縦 7×横 9 の大きさにカットします。下図のように 5 つの部品を取り付けます。

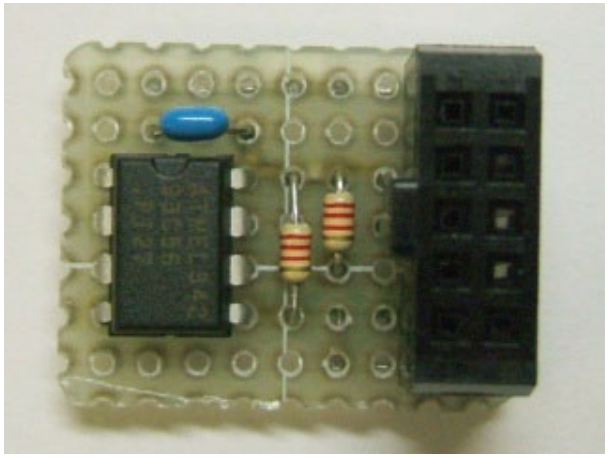


部 品 面 （ 表 ） か ら み た 基 板 の 図

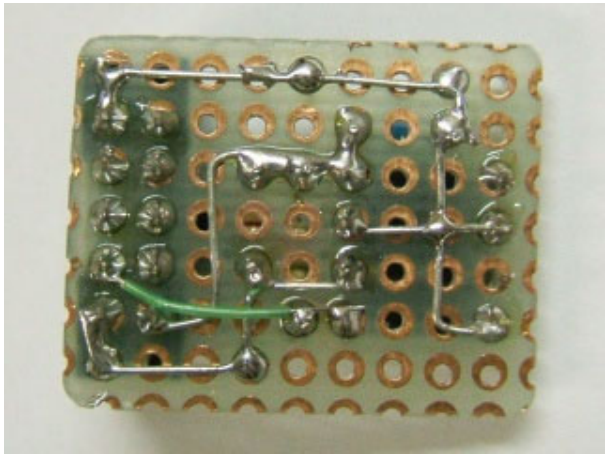
裏返（半田面）にして、下図のように配線します。



半 田 面 （ 裏 ） か ら み た 基 板 の 図

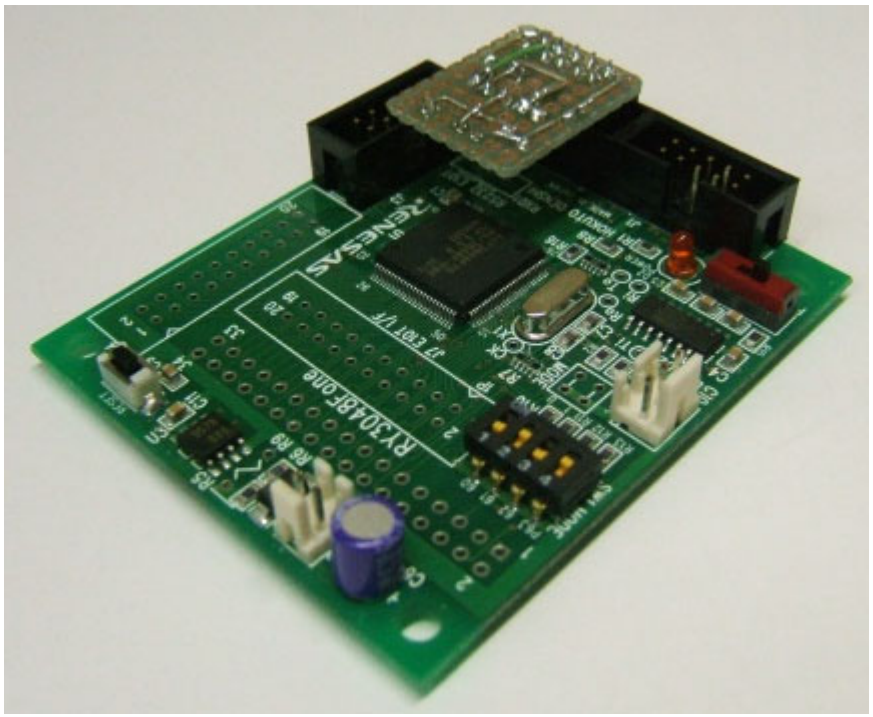


▲部品面から見た基板



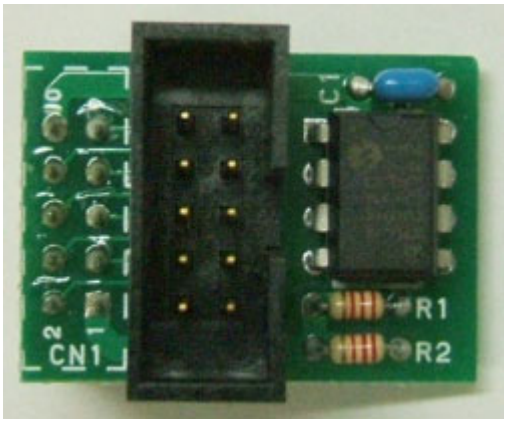
▲裏から見た基板

EEP-ROM 基板をポート A のコネクタに写真のように接続すれば、完成です。

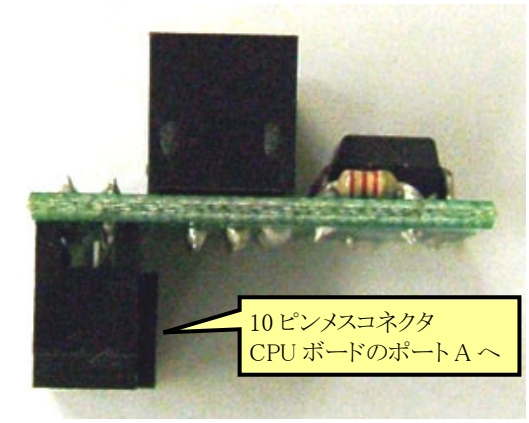


1.7 市販のEEP-ROM基板を使う

EEP-ROM 基板 Ver.2 がキットとして販売されています。回路は、前記の自作 EEP-ROM 基板と同様です。

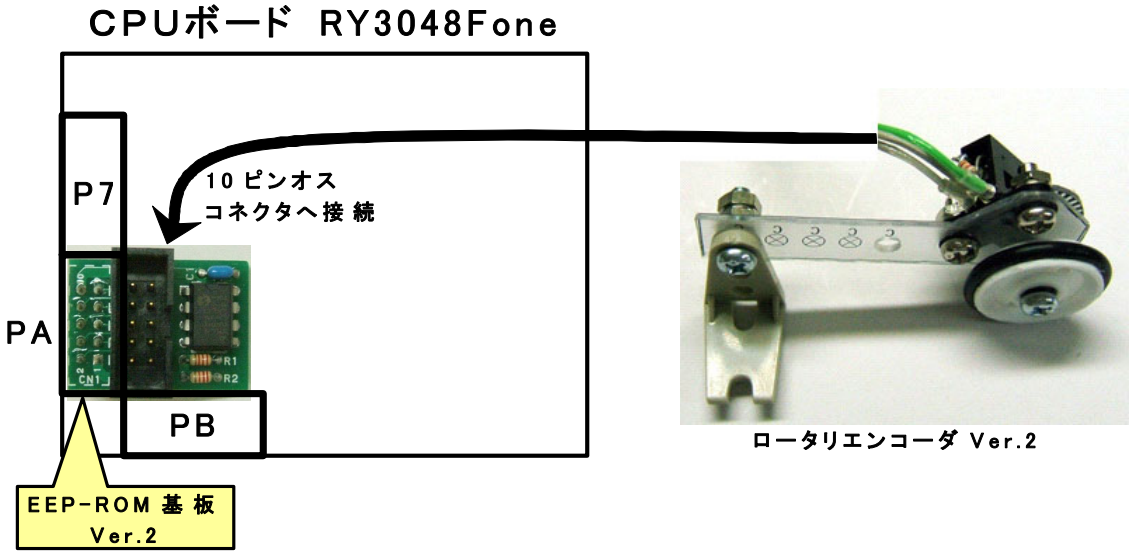


▲EEP-ROM 基板 Ver.2 の部品面



▲EEP-ROM 基板 Ver.2 を横から見たところ

EEP-ROM 基板 Ver.2 は、メスコネクタを RY3048Fone ボードのポート A に接続します。オスコネクタは、ポート A に接続する他の機器を接続することができます。例えば、ロータリエンコーダを繋ぐことができます。下図に接続例を示します。

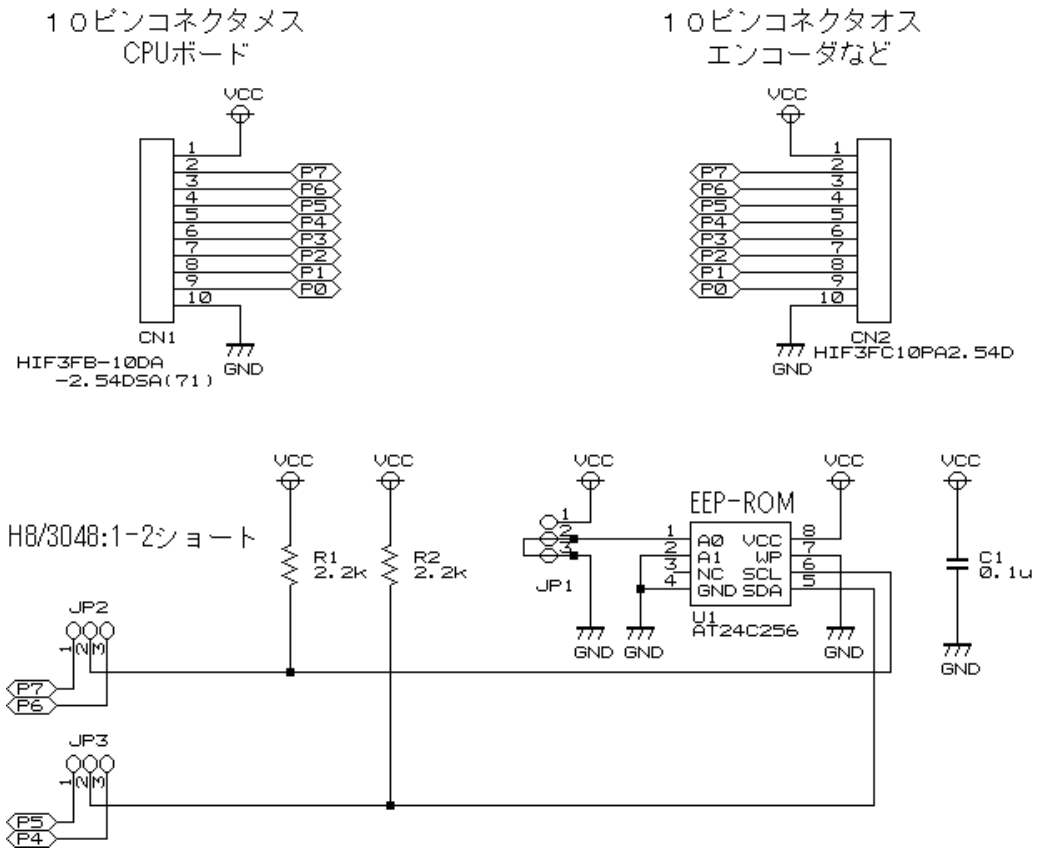


※EEP-ROM は PA7 と PA5 を使ってします。そのため、10 ピンオスコネクタに接続する機器が PA7 または PA5 を使っている場合は、信号がぶつかるため接続することができません。ロータリエンコーダは PA0 を使用するため接続することができます。

市販の EEP-ROM 基板 Ver.2 の製作方法については、マイコンカーラリーサイトの「EEP-ROM 基板 Ver.2 製作マニュアル」を参照してください。

1.8 EEP-ROM基板の回路図

EEP-ROM 基板 Ver.2 の回路図は下図のようになっています。



1.9 「i2c_eeprom.c」で使用できる関数

EEP-ROM にデータを読み書きする、専用の関数が用意されています。ファイル名は、「i2c_eeprom.c」です。
EEP-ROM を使用するときは、プロジェクトに「i2c_eeprom.c」を追加して使用します。
「i2c_eeprom.c」は、「C:¥WorkSpace¥common」フォルダにあります。
このファイルを追加して実行できる関数は下記のようです。

関数名	内容
initI2CEeprom **※2007.02 より使い方が変更になりました。**	void initI2CEeprom(unsigned char* ddrport, unsigned char* drport, unsigned char ddrdata, unsigned char scl, unsigned char sda); EEP-ROM へ読み書きする準備をします。最初に必ず実行します。 引 数:EEP-ROM の繋がっている DDR ポートの指定(&を付ける) EEP-ROM の繋がっている DR ポートの指定(&を付ける) DDR ポートの入出力設定値 EEP-ROM の SCL 端子の繋がっているビット番号 EEP-ROM の SDA 端子の繋がっているビット番号 戻り値:なし 例) initI2CEeprom(&PADDR, &PADR, 0x5f, 7, 5); EEP-ROM はポート A に接続、1 つ目の引数は PADDR を指定します。 EEP-ROM はポート A に接続、2 つ目の引数は PADR を指定します。 ポート A の入出力設定値は 0x5f です。 SCL 端子は、bit7 に接続します。 SDA 端子は、bit5 に接続します。
selectI2C EepromAddress	void selectI2CEepromAddress(unsigned char address); I2C バスに接続されているどのアドレスの EEP-ROM を使用するか選択します。 initI2CEeprom 関数実行時は、アドレス 0 が選択されています。 引 数:EEP-ROM のアドレス 戻り値:なし 例)selectI2CEepromAddress(1); 1 番を使用 EEP-ROM A1="0" A0="1"に接続されている EEP-ROM を選択します。
readI2CEeprom	char readI2CEeprom(unsigned int address); EEP-ROM からデータを読み込みます。 引 数:unsigned int アドレス 0～32767(0x7fff) 戻り値:char データ 例) i = readI2CEeprom(0x0005); EEP-ROM の 0x0005 番地のデータを変数 i に代入します。
writeI2CEeprom	void writeI2CEeprom(unsigned int address, char write); EEP-ROM へデータを書き込みます。**書き込み後、最大 10ms は書き込み作業中のため、アクセスできません。** 引 数:unsigned int アドレス 0～32767(0x7fff) ,char データ 戻り値:なし 例)writeI2CEeprom(0x2000, -100); EEP-ROM の 0x2000 番地に-100 を書き込みます

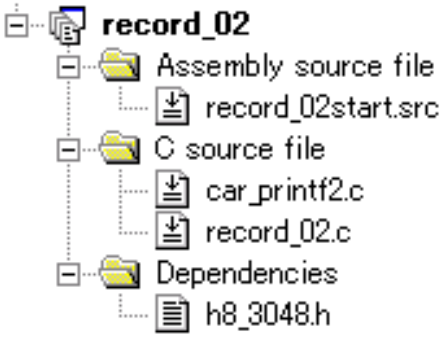
setPageWrite
I2CEeprom
	void setPageWriteI2CEeprom (unsigned int address, int count, char* data);
	EEP-ROM へ複数バイトのデータを書き込みます。**書き込み準備を行うだけですぐに終了します。実際の書き込みは I2CEepromProcess 関数で行います。書き込むデータ数は、2 の n 乗個とします。2 バイト、4 バイト、8 バイト…です。**
	
	引 数:unsigned int アドレス 0～32767 , int 個数 1～64 ,char* データがあるアドレス
	戻り値:なし
	例)char d[4];
	d[0]=5; d[1]=4; d[2]=1; d[3]=10;
	setPageWriteI2CEeprom(0x1000, 4, d); 書き込み準備のみ
	
	while(1) {
	I2CEepromProcess(); 実際の書き込みはこの関数
	}

I2CEeprom
Process
	void I2CEepromProcess(void);
	この関数は、setPageWriteI2CEeprom 関数で書き込みの準備をしたデータを、実際に EEP-ROM に書き込みます。
	書き込み作業は、「少しずつ行いすぐに終わる」を何度も繰り返しておこなっています。こうして、書き込み処理にかかりっきりにならないようにしています。この関数は、書き込みデータ数+5 回以上実行してください。例えば、8 バイト書き込むときは最低でも 13 回、この関数を実行します。
	通常はループ内に入れておきます。書き込み作業がないときは何もしませんので、入れておくだけで OK です。
	引 数:なし
	戻り値:なし
	例)main {
	init();
	while(1) {
	I2CEepromProcess(); 常に実行するようにする
	その他の処理
	}
	}

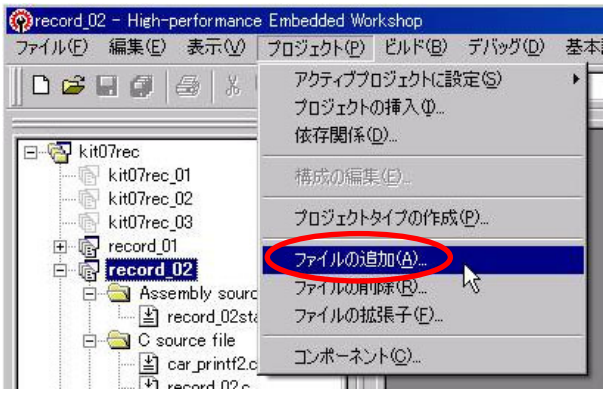
clearI2CEeprom	void clearI2CEeprom(void); EEP-ROM のデータをオールクリアします。**実行には数秒かかります**。実測で 4～5 秒です。 引 数:なし 戻り値:なし 例)clearI2CEeprom();
checkI2CEeprom	int checkI2CEeprom(void); 書き込み後、書き込み作業が終わったかどうかチェックします。 引 数:なし 戻り値:1→読み書き可能 0→書き込み作業中(アクセス不可) 例)while(!checkEeprom()); 書き込みが終了したかチェックします。

1.10 「i2c_eeprom.c」を登録する方法

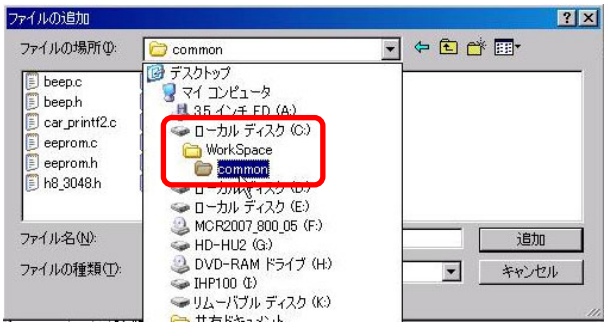
EEP-ROM を使うためには、プロジェクトに「i2c_eeprom.c」を登録しなければ行けません。例としてプロジェクト「record_02」に「i2c_eeprom.c」を追加する手順を説明します。



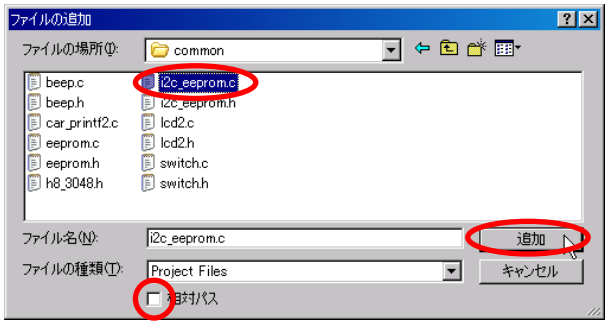
1. プロジェクト「record_02」には、まだ「i2c_eeprom.c」が登録されていないとします（実際は登録されています）。



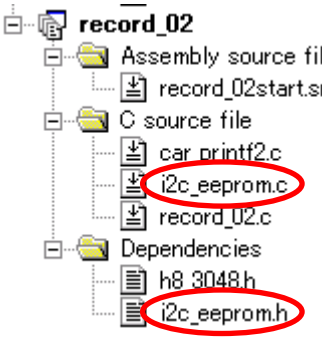
2. 「プロジェクト→ファイルの追加」を選択します。



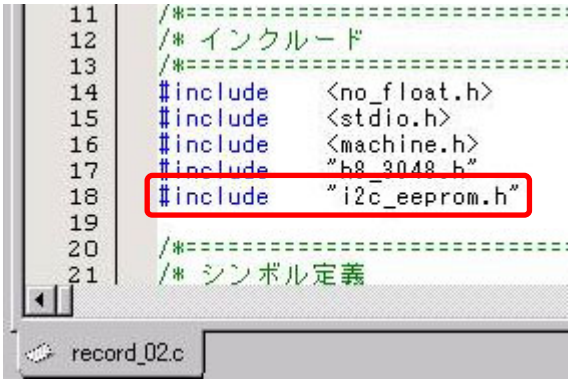
3. 「C:¥WorkSpace¥common」フォルダを選択します。



4. 「相対パス」のチェックを外します。「i2c_eeprom.c」を選択、追加をクリックします。



5. 「i2c_eeprom.c」がファイルリストに登録されました。Dependencies 欄には、自動的に「i2c_eeprom.h」が登録されます。



6. 「record_02.c」で「i2c_eeprom.c」内の関数を使用するために、インクルード欄に**□部分**を追加します。

上記手順で「i2c_eeprom.c」を追加します。
EEP-ROM を使用するＣソースファイルに「#include "i2c_eeprom.h"」の追加を忘れないようにしてください。

1.11 EEP-ROMの接続端子を変える場合の設定

本書で紹介している回路は、PA7 に EEP-ROM の SCL 端子、PA5 に EEP-ROM の SDA 端子を接続しています。これらの端子をすでに使って、違う端子を使用したい場合は、initI2CEeprom 関数の引数を変えます。引数は次のようになります。

```
initI2CEeprom( &1, &2, 3, 4, 5 );
1…&EEP-ROM の DDR レジスタ
2…&EEP-ROM の DR レジスタ
3…1 で指定した DDR ポートの入出力設定値
4…SCL 端子が接続されているビット
5…SDA 端子が接続されているビット
```

※ポート 7 には接続できません。

例えば、EEP-ROM をポート 6 に接続するとします。ポート 6 に繋がっている内容と入出力設定は下記のようにするとします。

bit	7	6	5	4	3	2	1	0
接続名	―	EEP-ROM の SCL 端子	EEP-ROM の SDA 端子	未接続	ディップスイッチ	ディップスイッチ	ディップスイッチ	ディップスイッチ
入力 or 出力	出力	**入力**	**入力**	出力	入力	入力	入力	入力

DDR レジスタの設定は、入力"0"、出力"1"にするだけです。

bit	7	6	5	4	3	2	1	0
0 or 1	1	0	0	1	0	0	0	0

2 進数で"1001 0000"ですので、16 進数で 0x90 となります。これらから、下記のように設定します。

```
initI2CEeprom( &P6DDR, &P6DR, 0x90, 6, 5 );
```

※init 関数内での P6DDR の設定

initI2CEeprom 関数で設定していますので、init 関数内でポート 6 の入出力設定は必要ありません。今までのプログラムどおり、設定しておきたい場合は、initI2CEeprom 関数の引数と同じ入出力設定をしてください。下記にプログラム例を示します。

```
void init( void ) {
    …
    P6DDR = 0x90;
    …
}
```

2. サンプルプログラム

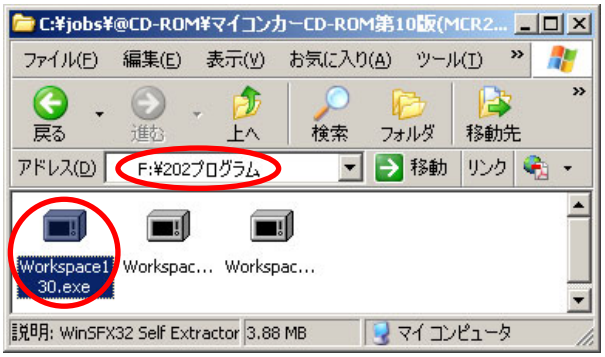
2.1 ルネサス統合開発環境

サンプルプログラムは、ルネサス統合開発環境(High-performance Embedded Workshop)を使用して開発するように作っています。ルネサス統合開発環境についてのインストール、開発方法は、「ルネサス統合開発環境操作マニュアル 導入編」を参照してください。

2.2 サンプルプログラムのインストール

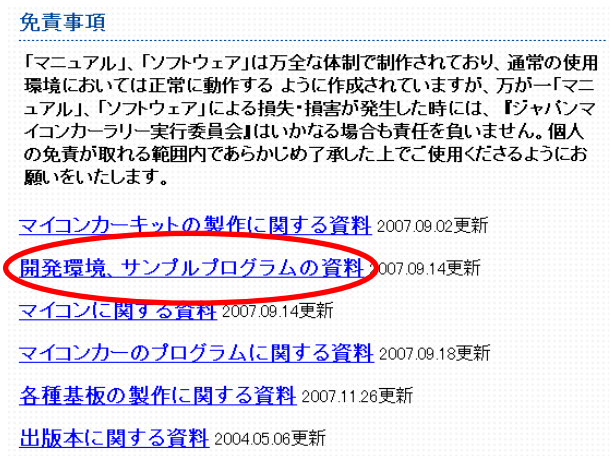
サンプルプログラムをインストールします。

2.2.1 CDからソフトを取得する



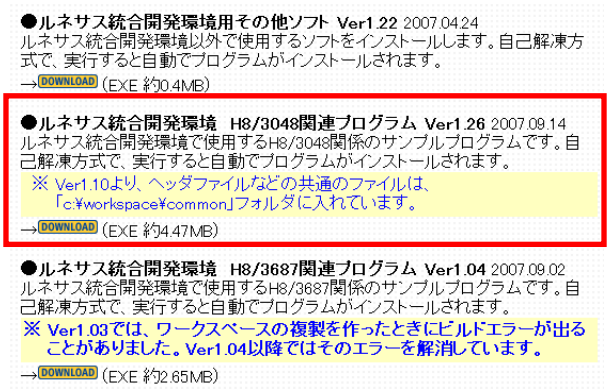
2007 年以降の講習会 CD がある場合、「CD ドライブ→202 プログラム」フォルダにある、「Workspace130.exe」を実行します。数字の 130 は、バージョンにより異なります。

2.2.2 ホームページからソフトを取得する



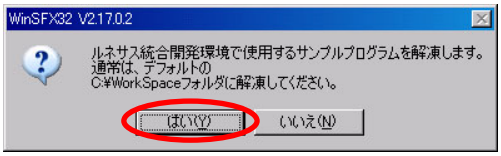
1.マイコンカーラリーサイト「http://www.mcr.gr.jp/」の技術情報→ダウンロード内のページへ行きます。

2.「開発環境、サンプルプログラムの資料」をクリックします。

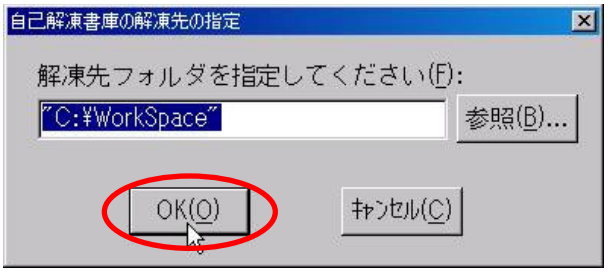


3.「ルネサス統合開発環境　H8/3048 関連プログラム」をダウンロードします。

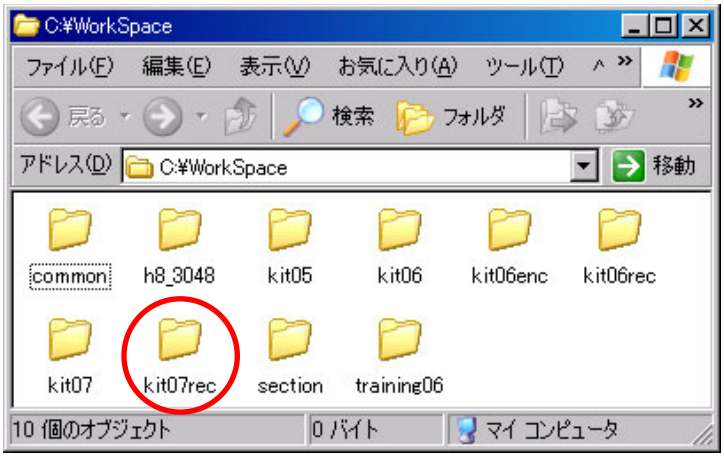
2.2.3 インストール



1.CD またはダウンロードした「Workspace130.exe」を実行します。はいをクリックします。



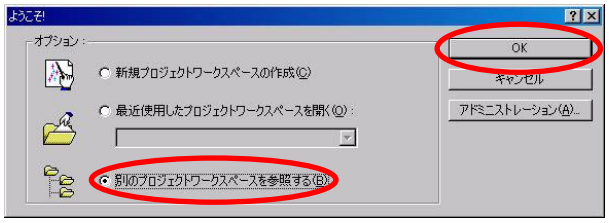
2.ファイルの解凍先を選択します。OK をクリックします。**このフォルダは変更できません。**



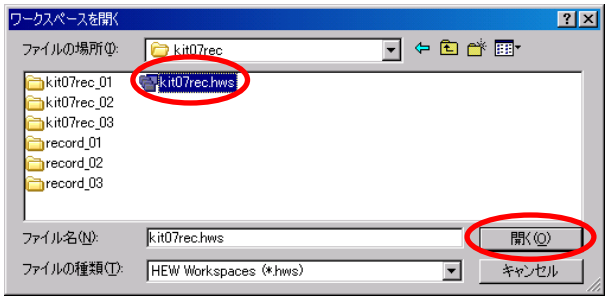
3. 解凍が終わったら、エクスプローラで「C ドライブ→Workspace」フォルダを開いてみてください。複数のフォルダがあります。今回使用するのは、「**kit07rec**」です。

2.3 ワーススペース「kit07rec」を開く

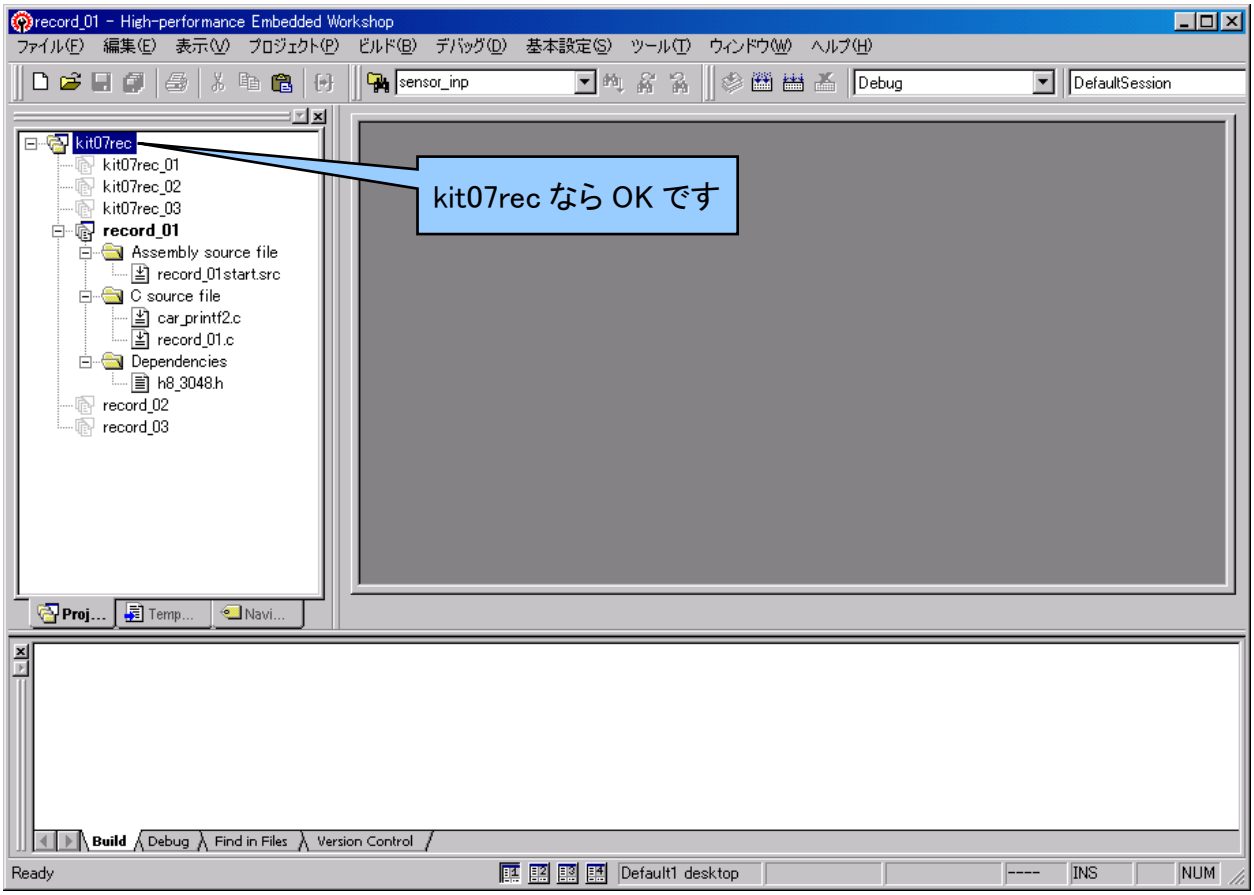
1. ルネサス統合開発環境を実行します。



2.「別のプロジェクトワークスペースを参照する」を選択、OK をクリックします。

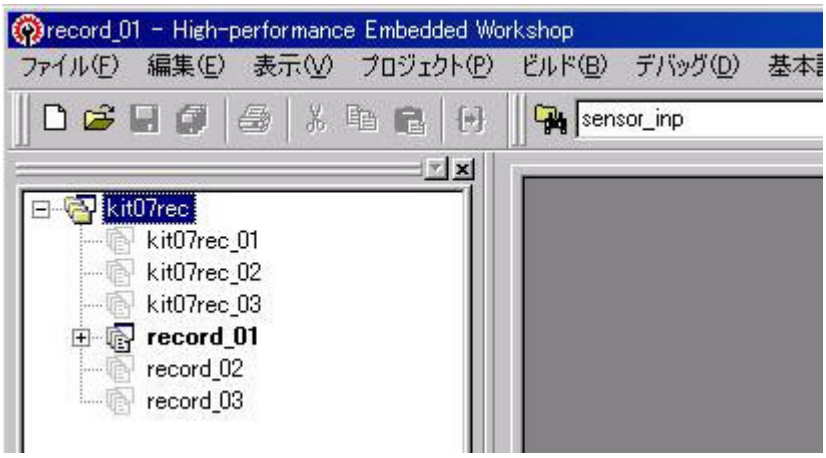


3. Cドライブ→Workspace→kit07rec の「kit07rec.hws」を選択、開く をクリックします。



4. kit07rec というワークスペースが開かれます。

2.4 プロジェクト



ワークスペース「kit07rec」には、6 つのプロジェクトが登録されています。

プロジェクト名	内容
record_01	H8/3048F-ONE の内蔵 RAM を使用し、データを記録、転送するプログラムです。内蔵 RAM に保存する、動作理解用のサンプルプログラムです。
record_02	24C256 という外付けの EEP-ROM を使用し、データを記録、転送するプログラムです。EEP-ROM に保存する、動作理解用のサンプルプログラムです。
record_03	record_02.c を改造し、転送データをあらかじめ 2 進数に変換して出力します。
kit07rec_01	走行データの記録をします。記録には、H8/3048F-ONE の内蔵 RAM を使用します。約 2.5KB 保存することができます。
kit07rec_02	走行データの記録をします。記録には、24C256 という外付けの EEP-ROM を使用します。32KB 保存することができます。
kit07rec_03	kit07rec_02.c プログラムを改造して、ロータリエンコーダを使えるようにしたプロジェクトです。記録データは、今までの内容に走行スピードを追加しています。

3. プロジェクト「record_01」 内蔵RAMにデータ保存

3.1 概要

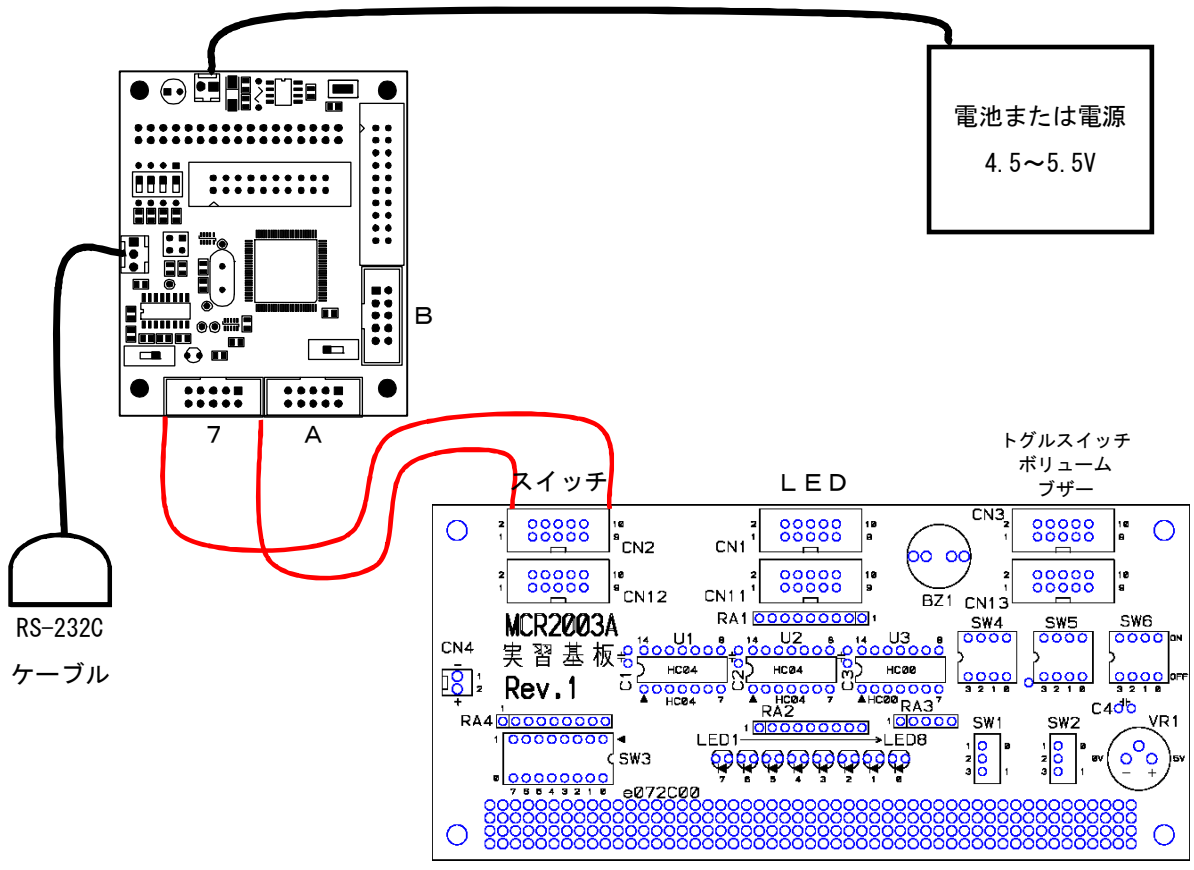
このプログラムは、
・ポート 7 に接続されているディップスイッチの値
・CPU ボードのディップスイッチの値
を、10ms ごとに内蔵 RAM に保存します。保存時間は 10 秒です。保存時間は自由に変えられますが上限はメモリがいっぱいになるまでです(今回は約 12.5 秒、1 回に保存するデータ数で変わります)。保存後、RS232C を通してパソコンへ保存した情報を出力します。

ここでは、
・データを保存する方法
・CPU ボードからパソコンへデータを出力する方法
・パソコンで受信したデータを、エクセルに取り込む方法
・エクセルに取り込んだデータの解析方法
について、説明します。

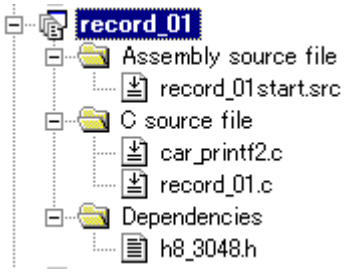
内蔵RAMについては、「**1.3 EEP-ROMを使う意義**」を参照してください。

3.2 接続

・CPU ボードのポート 7 と、実習基板のスイッチ部をフラットケーブルで接続します。
※ポート 7 のディップスイッチをセンサ基板に変えると、センサの反応を記録することができます。



3.3 プロジェクトの構成



・record_01start.src
・record_01.c
・car_printf2.c
の 3 ファイルあります。
h8_3048.h は record_01.c、car_printf2.c でインクルードされているファイルです。

3.4 プログラム

```
/*****************************************************************************/
/* 内蔵RAMデータ保存演習プログラム「record_01.c」                              */
/*                          2006.04 ジャパンマイコンカーラリー実行委員会       */
/*****************************************************************************/
/*
本プログラムはH8/3048F-ONEの内蔵RAM(2.5KB程度)にポート７のデータ、
CPUボード上のディップスイッチの値を10msごとに保存します。
その後、保存したデータをパソコンへ転送します。
*/

/*====================================*/
/* インクルード                       */
/*====================================*/
#include    <no_float.h>                /* stdioの簡略化 最初に置く*/
#include    <stdio.h>
#include    <machine.h>
#include    "h8_3048.h"

/*====================================*/
/* シンボル定義                       */
/*====================================*/
#define         SAVE_SIZE   1000        /* データ保存数             */

/*====================================*/
/* プロトタイプ宣言                   */
/*====================================*/
void init( void );
unsigned char dipsw_get( void );

/*====================================*/
/* グローバル変数の宣言               */
/*====================================*/
unsigned long   cnt1;                   /* main内で使用             */
int             pattern;                /* パターン番号             */

/* データ保存関連 saveDataのサイズは最大で2.5KB程度としてください */
int             iTimer10;               /* 取得間隔計算用           */
unsigned char   saveData[SAVE_SIZE][2]; /* 保存エリア               */
int             saveIndex;              /* 保存インデックス         */
int             saveSendIndex;          /* 送信インデックス         */
int             saveFlag;               /* 保存フラグ               */

/*****************************************************************************/
/* メインプログラム                                                          */
/*****************************************************************************/
void main( void )
{
    /* マイコン機能の初期化 */
    init();                             /* 初期化                   */
    init_sci1( 0x00, 79 );              /* SCI1初期化               */
    set_ccr( 0x00 );                    /* 全体割り込み許可         */

    while( 1 ) {
    switch( pattern ) {

    case 0:
        /* 1秒待ち */
        if( cnt1 > 1000 ) {
            printf( "¥n" );
            printf( "data recording...¥n" );
            pattern = 1;
            saveIndex = 0;
            saveFlag = 1;               /* データ保存開始           */
            cnt1 = 0;
        }
```

```
 66 :          break;
 67 :
 68 :      case 1:
 69 :          /* データ保存中　保存自体は割り込みの中で行う */
 70 :          if( saveFlag == 0 ) {
 71 :              pattern = 2;
 72 :          }
 73 :          break;
 74 :
 75 :      case 2:
 76 :          /* 転送 */
 77 :          printf( "¥n" );
 78 :          printf( "record_01 Data Out¥n" );
 79 :          printf( "P7 data,dip sw data¥n" );
 80 :
 81 :          saveSendIndex = 0;
 82 :          pattern = 3;
 83 :          break;
 84 :
 85 :      case 3:
 86 :          /* データ転送 */
 87 :          printf( "%02x,%d¥n",
 88 :              saveData[saveSendIndex][0],
 89 :              saveData[saveSendIndex][1] );
 90 :          saveSendIndex++;
 91 :          if( saveIndex <= saveSendIndex ) {
 92 :              pattern = 4;
 93 :              cnt1 = 0;
 94 :          }
 95 :          break;
 96 :
 97 :      case 4:
 98 :          /* 転送終了 */
 99 :          break;
100 :
101 :      default:
102 :          /* どれでもない場合は待機状態に戻す */
103 :          pattern = 0;
104 :          break;
105 :      }
106 :      }
107 :  }
108 :
109 :  /************************************************************************/
110 :  /* H8/3048F-ONE 内蔵周辺機能　初期化                                    */
111 :  /************************************************************************/
112 :  void init( void )
113 :  {
114 :      /* I/Oポートの入出力設定 */
115 :      P1DDR = 0xff;
116 :      P2DDR = 0xff;
117 :      P3DDR = 0xff;
118 :      P4DDR = 0xff;
119 :      P5DDR = 0xff;
120 :      P6DDR = 0xf0;                       /* CPU基板上のDIP SW        */
121 :      P8DDR = 0xff;
122 :      P9DDR = 0xf7;                       /* 通信ポート               */
123 :      PADDR = 0xff;
124 :      PBDDR = 0xff;
125 :      /* ポート７は、入力専用なので入出力設定はありません　*/
126 :
127 :      /* ITU0 1msごとの割り込み */
128 :      ITU0_TCR = 0x20;
129 :      ITU0_GRA = 24575;
130 :      ITU0_IER = 0x01;
131 :
132 :      /* ITUのカウントスタート */
133 :      ITU_STR = 0x01;
134 :  }
135 :
136 :  /************************************************************************/
137 :  /* ITU0 割り込み処理                                                    */
138 :  /************************************************************************/
139 :  #pragma interrupt( interrupt_timer0 )
140 :  void interrupt_timer0( void )
141 :  {
142 :      ITU0_TSR &= 0xfe;                   /* フラグクリア             */
143 :      cnt1++;
144 :
145 :      /* データ保存関連 */
146 :      iTimer10++;
147 :      if( iTimer10 >= 10 ) {
148 :          iTimer10 = 0;
149 :          if( saveFlag ) {
150 :              saveData[saveIndex][0] = P7DR;
151 :              saveData[saveIndex][1] = dipsw_get();
152 :              saveIndex++;
153 :              if( saveIndex >= SAVE_SIZE ) saveFlag = 0;
154 :          }
155 :      }
156 :  }
```

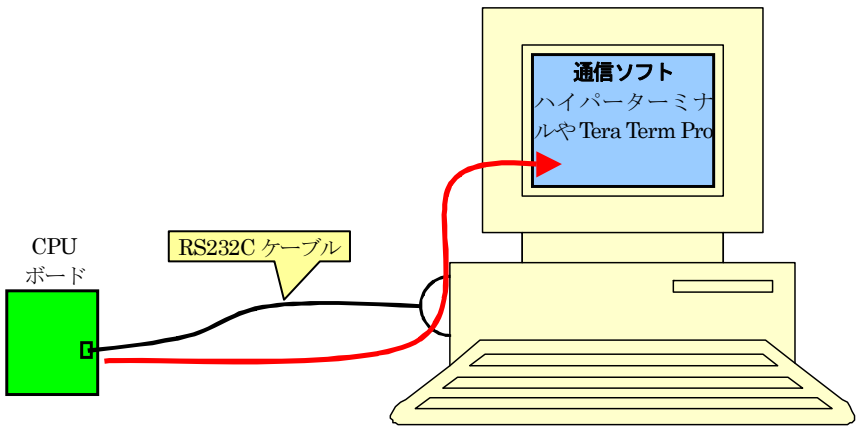
```
157 :
158 :  /************************************************************************/
159 :  /* ディップスイッチ値読み込み                                           */
160 :  /* 戻り値 スイッチ値 0～15                                               */
161 :  /************************************************************************/
162 :  unsigned char dipsw_get( void )
163 :  {
164 :      unsigned char sw;
165 :
166 :      sw  = ~P6DR;                        /* ディップスイッチ読み込み */
167 :      sw &= 0x0f;
168 :
169 :      return  sw;
170 :  }
171 :
172 :  /************************************************************************/
173 :  /* end of file                                                          */
174 :  /************************************************************************/
```

3.5 データをパソコンへ送る

※詳しくは、H8/3048F-ONE 実習マニュアルのプロジェクト「sio」部分を参照してください。ここでは簡単に説明します。

CPU ボードからパソコンへのデータ転送は、RS232C ケーブルを通して行います。

パソコンには、ハイパーターミナルや TeraTermPro などの通信ソフトを立ち上げておきます。通信ソフトは、CPU ボードから送られてきたデータを RS232C ケーブルを通して受信、通信ソフトの画面上に表示します。表示すると共に、受信データをファイルに保存することもできます。エクセルなどでそのファイルを取り込むことにより、さまざまな解析をすることができます。



CPU ボード→RS232C ケーブル→通信ソフトの画面やファイルへ保存

今回、

```
printf("Hello World!¥n");
```

とすると、H8 マイコンは RS232C に printf 文のメッセージを出力します。通信ソフトは、RS232C からデータを受信します。そして通信ソフトの画面上に

```
Hello World!
```

と表示します。

H8 の C 言語プログラムとパソコンの C 言語プログラムが printf 文を実行したとき、出力先が違うことを押さえておいてください。

printf 文を実行する機器	出力先	表示
パソコン	ディスプレイ端子	ディスプレイ
H8 マイコン	RS232C 端子	通信ソフトの画面

H8 マイコンの場合、printf 文の出力先を RS232C にするのが、「car_printf2.c」です。本プロジェクトには 3 つの C ソースファイルや src ソースファイルがあります。それぞれのファイル内容は、下記のようになっています。

ファイル名	内容
record_01start.src	ベクタアドレスの設定、スタートアップルーチンが入っているファイルです。
record_01.c	C言語のメインプログラムファイルです。
car_printf2.c	**・通信するための設定** **・printf 関数の出力先、scanf 関数の入力元を RS232C ポートにするための設定** **・セクション D,R,B を初期化する設定** **を行っています。** **プロジェクト「record_01」に限らず、これらを行うプロジェクトには、car_printf2.c ファイルを追加します。**

3.6 プログラムの解説

3.6.1 内蔵周辺機能の初期設定

```
109 :  /************************************************************************/
110 :  /* H8/3048F-ONE 内蔵周辺機能　初期化                                    */
111 :  /************************************************************************/
112 :  void init( void )
113 :  {
114 :      /* I/O ポートの入出力設定 */
115 :      P1DDR = 0xff;
116 :      P2DDR = 0xff;
117 :      P3DDR = 0xff;
118 :      P4DDR = 0xff;
119 :      P5DDR = 0xff;
120 :      P6DDR = 0xf0;                       /* CPU 基板上の DIP SW        */
121 :      P8DDR = 0xff;
122 :      P9DDR = 0xf7;                       /* 通信ポート                 */
123 :      PADDR = 0xff;
124 :      PBDDR = 0xff;
125 :      /* ポート 7 は、入力専用なので入出力設定はありません    */
126 :
127 :      /* ITU0 1ms ごとの割り込み */
128 :      ITU0_TCR = 0x20;
129 :      ITU0_GRA = 24575;
130 :      ITU0_IER = 0x01;
131 :
132 :      /* ITU のカウントスタート */
133 :      ITU_STR = 0x01;
134 :  }
```

ポート 6 の bit3～0 は、ディップスイッチ入力です。ポート 7 は、実習基板のディップスイッチ、またはセンサ基板入力です。それぞれの端子を入出力設定します。

他に、ITU0 を 1ms ごとの割り込み設定にします。

3.6.2 データ取得関係の定義、変数

```
19 :  /*======================================*/
20 :  /* シンボル定義                         */
21 :  /*======================================*/
22 :  #define         SAVE_SIZE   1000        /* データ保存数             */
23 :
24 :  /*======================================*/
25 :  /* プロトタイプ宣言                     */
26 :  /*======================================*/
27 :  void init( void );
28 :  unsigned char dipsw_get( void );
29 :
30 :  /*======================================*/
31 :  /* グローバル変数の宣言                 */
32 :  /*======================================*/
33 :  unsigned long   cnt1;                   /* main 内で使用            */
34 :  int             pattern;                /* パターン番号             */
35 :
36 :  /* データ保存関連 saveData のサイズは最大で 2.5kbytes 程度としてください */
37 :  int             iTimer10;               /* 取得間隔計算用           */
38 :  unsigned char   saveData[SAVE_SIZE][2]; /* 保存エリア               */
39 :  int             saveIndex;              /* 保存インデックス         */
40 :  int             saveSendIndex;          /* 送信インデックス         */
41 :  int             saveFlag;               /* 保存フラグ               */
```

太字部分が、今回追加した変数です。

変数名	意味	内容
iTimer10	保存間隔計算用	データを保存する間隔を調整するために、この変数で計算します。割り込みの間隔は 1ms、保存する間隔は 10ms です。そのため、この変数を割り込みごとに+1 します。10 になったら、10ms たったと判断して、データ保存処理を行います。
saveData	データ保存用	saveData は 2 次元配列で、SAVE_SIZE 個分の値を保存します。saveData は unsigned char 型なので、メモリは 1 データ 1 バイト使用します。「saveData[SAVE_SIZE][**2**]」なので、1 回の保存で 2 バイト使用します。H8/3048F-ONE の RAM 領域は 4KB ありますが、すべて使えるわけではありません。関数の戻り先アドレスの保存、レジスタの値保存など、マイコン側でも使用します。そのため、最大で約 2.5KB の確保が限界です。今回は1回で 2 バイト分のメモリを使用するので、保存回数は 1250 回が限界です。
saveIndex	保存インデックス	配列の何番目に保存するかを指定する変数です。
saveSendIndex	保存送信インデックス	配列の何番目のデータを送信するかを指定する変数です。
saveFlag	保存フラグ	1 なら割り込みプログラム内で 10ms ごとにデータを保存します。0 なら保存しません。

3.6.3 パソコンとの通信するための初期設定

```
48 :      /* マイコン機能の初期化 */
49 :      init();                          /* 初期化                  */
50 :      init_sci1( 0x00, 79 );           /* SCI1初期化              */
51 :      set_ccr( 0x00 );                 /* 全体割り込み許可        */
```

パソコンと通信するために、H8/3048F-ONE の内蔵周辺機能である SCI1 の初期化を行います。car_printf2.c 内に、簡単に SCI1 関係のレジスタを初期化することができる init_sci1 関数がありますので、この関数を使って初期化します。この設定により、9600bps で通信することができます。詳細については、「H8/3048F-ONE 実習マニュアル」のプロジェクト「sio」の解説を参照してください。

3.6.4 パターン 0:1 秒待ち

```
56 :      case 0:
57 :          /* 1秒待ち */
58 :          if( cnt1 > 1000 ) {
59 :              printf( "¥n" );
60 :              printf( "data recording...¥n" );
61 :              pattern = 1;
62 :              saveIndex = 0;
63 :              saveFlag = 1;               /* データ保存開始          */
64 :              cnt1 = 0;
65 :          }
66 :          break;
```

CPU ボードに電源を入れてから、1 秒待ちます。1 秒たつと
59～60 行…データを記録する旨、メッセージを出力します。
61 行…次に実行するときはパターン 1 になるよう、変数の設定をします。
62 行…保存する配列の番号用の変数を初期化します。
63 行…データの保存を開始します。
64 行…時間計測用の cnt1 変数をクリアします。

次から、パターン 1 に移ります。

3.6.5 パターン 1:データ保存

```
68 :      case 1:
69 :          /* データ保存中　保存自体は割り込みの中で行う */
70 :          if( saveFlag == 0 ) {
71 :              pattern = 2;
72 :          }
73 :          break;
```

1 なら保存中
0 なら保存終了

データの保存を行います。保存は 10ms ごとに割り込み内で行われています。そのため、メインプログラムでは保存処理はしていません。保存する配列が一杯になると、割り込み内で saveFlag を 0 にします。メインプログラムでは、saveFlag 変数が 0 になったかを常にチェック、なったなら保存が終了したと判断して、パターン 2 へ移ります。

3.6.6 パターン 2:タイトル転送、準備

```
75 :        case 2:
76 :            /* タイトル転送、準備*/
77 :            printf( "¥n" );
78 :            printf( "record_01 Data Out¥n" );
79 :            printf( "P7 data,dip sw data¥n" );
80 :
81 :            saveSendIndex = 0;
82 :            pattern = 3;
83 :            break;
```

データを送信する前にタイトルなどを送信します。次に、saveSendIndex 変数を 0 にして、送信するデータの番号を初期化します。パターン 3 へ移ります。

3.6.7 パターン 3:データの転送

```
85 :        case 3:
86 :            /* データ転送 */
87 :            printf( "%02x,%d¥n",
88 :                saveData[saveSendIndex][0],        ポート７のデータ読み込み
89 :                saveData[saveSendIndex][1] );      ディップスイッチのデータ読み込み
90 :            saveSendIndex++;
91 :            if( saveIndex <= saveSendIndex ) {
92 :                pattern = 4;
93 :                cnt1 = 0;
94 :            }
95 :            break;
```

データを送信します。

87～89 行…配列からデータを読み込みながら、printf 文でデータを送信します。

printf 文のカッコ内の意味は、

「%02x」…　2 桁に満たない場合は 0 で埋めて 16 進数表記

「,」……そのまま表示されます。

「%d」……　10 進数表記

「¥n」……　改行

となります。

90 行…saveSendIndex 変数を＋1して、saveData 配列のデータを次に出力するデータにしておきます。

91 行…送信しようとしている配列が保存数より大きくなったならパターン 4 に移り、送信を終えます。

下記に、出力例を示します。

data recording...	ff,12
	ff,12
record_01 Data Out	ff,8
P7 data,dip sw data	ff,0
aa,15	ff,0
aa,15	ff,4
ab,15	ff,4
ab,15	ff,4
af,15	ff,6
af,15	ff,6
bf,15	ff,6
bf,15	ff,6
ff,15	ff,7
ff,15	ff,7
ff,14	ff,7
ff,14	ff,7
ff,14	ff,7

3.6.8 パターン 4:転送終了

```
97 :        case 4:
98 :            /* 転送終了 */
99 :            break;
```

終わりです。何もしません。

3.6.9 割り込み処理

```
136 :  /************************************************************************/
137 :  /* ITU0 割り込み処理                                                    */
138 :  /************************************************************************/
139 :  #pragma interrupt( interrupt_timer0 )
140 :  void interrupt_timer0( void )
141 :  {
142 :      ITU0_TSR &= 0xfe;                   /* フラグクリア             */
143 :      cnt1++;
144 :
145 :      /* データ保存関連 */
146 :      iTimer10++;
147 :      if( iTimer10 >= 10 ) {
148 :          iTimer10 = 0;
149 :          if( saveFlag ) {
150 :              saveData[saveIndex][0] = P7DR;
151 :              saveData[saveIndex][1] = dipsw_get();
152 :              saveIndex++;
153 :              if( saveIndex >= SAVE_SIZE ) saveFlag = 0;
154 :          }
155 :      }
156 :  }
```

保存するデータを変えたい場合は、ここを変更する

割り込みプログラムは、1ms ごとに実行されます。データの保存は 10ms ごとです。そのため、146 行で割り込み 1 回ごとに iTimer10 を+1 して、147 行で 10 になったかチェック、10 になったなら 148 行以降を実行します。これでカッコの中は 10ms ごとに実行されます。

148 行で、次の 10ms 後に備えて、iTimer 変数をクリアします。

149 行で、saveFlag 変数をチェック、0 以外ならデータの保存を行うと解釈し、150～151 行で saveData 配列にデータを保存します。保存するデータは、ポート7の入力値とディップスイッチ値となります。

152 行で次回の保存に備えて、配列の添字([]の中の数字)である、saveIndex 変数を＋1しておきます。

153 行で配列の上限に達していないかチェック、上限になったなら saveFlag を 0 にして、保存を止めます。

saveData は下図のようなイメージです。SAVE_SIZE は 1000 とします。

	B=0	B=1
saveData[0][B] →	10ms 後の P7DR	10ms 後のスイッチ値
saveData[1][B] →	20ms 後の P7DR	20ms 後のスイッチ値
saveData[2][B] →	30ms 後の P7DR	30ms 後のスイッチ値
saveData[3][B] →	40ms 後の P7DR	40ms 後のスイッチ値
saveData[4][B] →	50ms 後の P7DR	50ms 後のスイッチ値
…		
saveData[998][B] →	9990ms 後の P7DR	9990ms 後のスイッチ値
saveData[999][B] →	10000ms 後の P7DR	10000ms 後のスイッチ値

1000 行

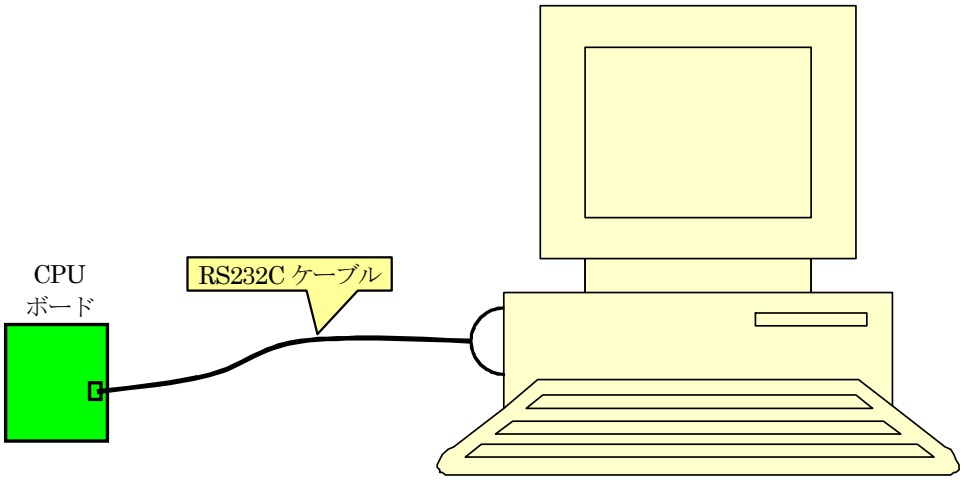
2 列

2 列 1000 行、1マスが 1 バイトなので、saveData 配列は合計 2000 バイトのサイズとなります。

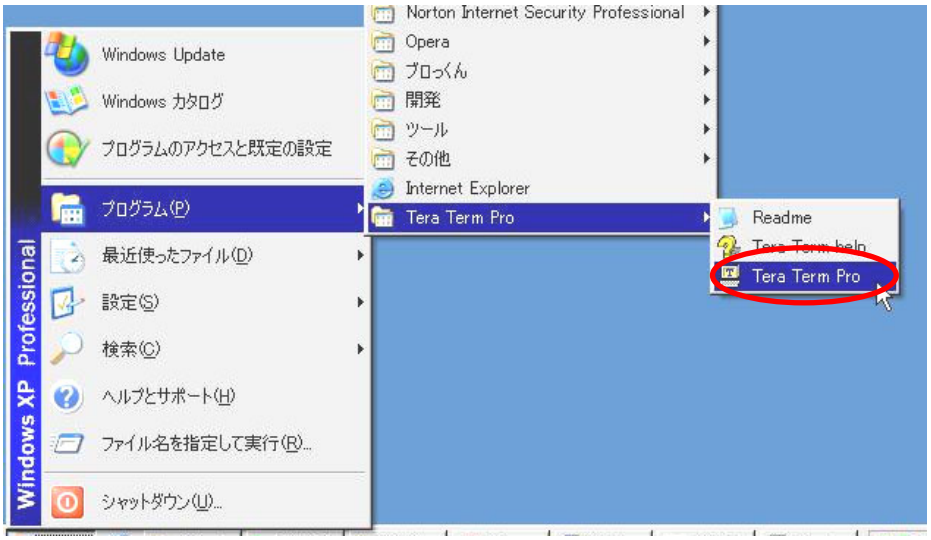
H8 マイコンの RAM 容量 4KB の内、2000 バイトをデータ保存エリアとして使用します。

3.7 データの取り込み方

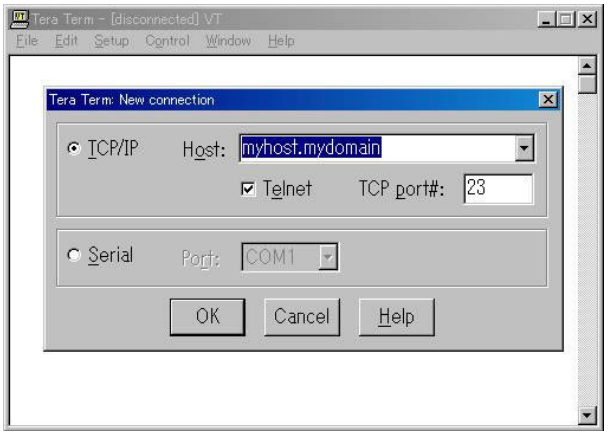
パソコンと通信ソフトを使ってデータを取り込む方法を説明します。



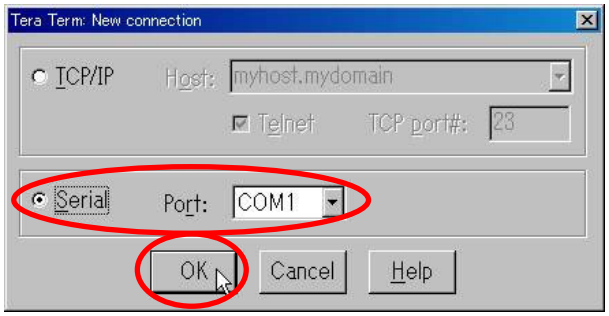
1. プロジェクト「record_01」をビルドして、「record_01.mot」ファイルを CPU ボードに書き込んでください。CPU ボードとパソコン間の RS232C ケーブルは繋いだままにしておきます。



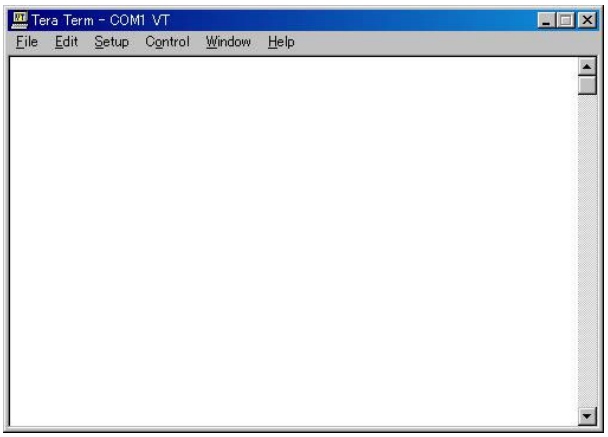
2. 「スタート→すべてのプログラム(またはプログラム)→Tera Term Pro→Tera Term Pro」で Tera Term Pro を立ち上げます。Tera Term Pro をまだインストールしていない場合は、H8/3048F-ONE 実習マニュアルのプロジェクト「sio」にある Tera Term Pro のインストール欄を参照してインストールしてください。



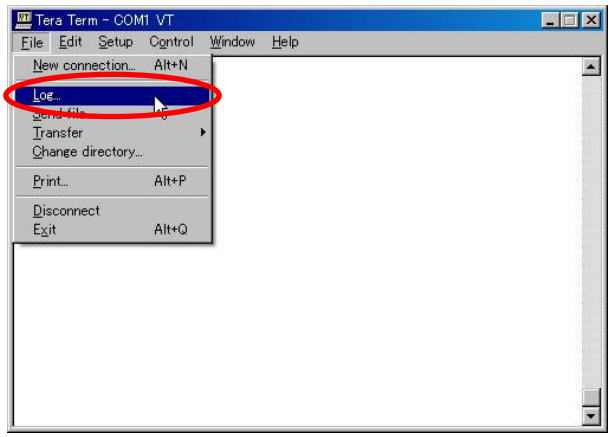
3. 接続先を確認する画面が表示されます。



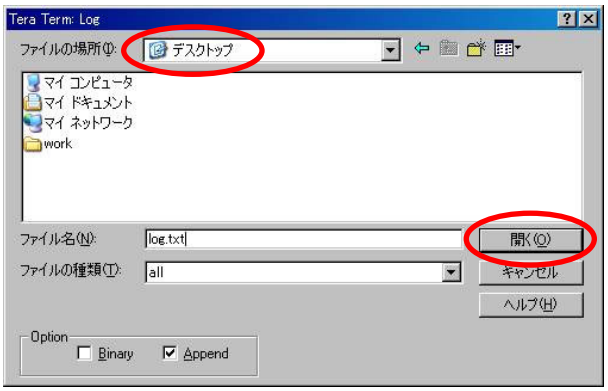
4. 「Serial」を選んで、各自のパソコンに合わせてポート番号を選びます。選択後、OK をクリックして次へ進みます。



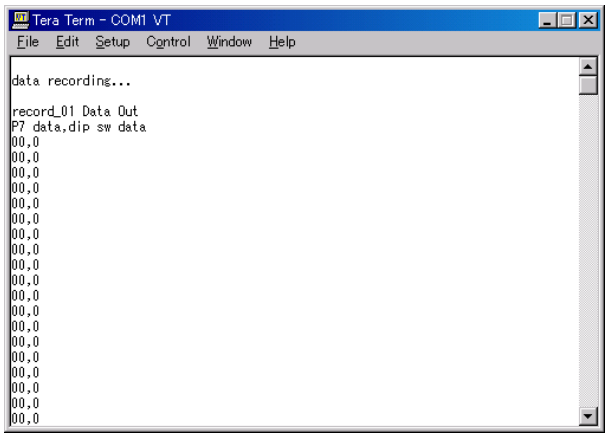
5. 立ち上がりました。



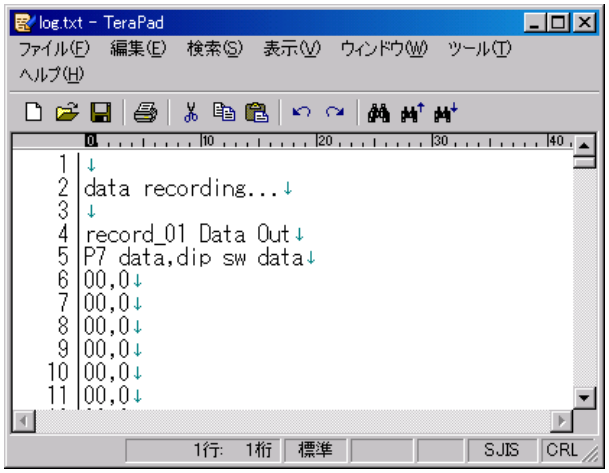
6. 受信データをファイルに保存する設定をします。「File→Log」を選択します。



7. 保存ファイル名を入力します。ここでは「log.txt」と入力します。保存するフォルダも分かりやすい位置に変更しておきましょう。今回は、「デスクトップ」にしています。ファイル名を設定できたら、開く をクリックします。



8. CPU ボードの電源を入れると、「data recording...」と表示され、10ms ごとにポート 7 とディップスイッチの値が保存されます。配列は 1000 個分確保しているので 10 秒保存します。10 秒たつと、保存したデータが TeraTermPro に送られてきます。転送が終了したら TeraTermPro は終了します。



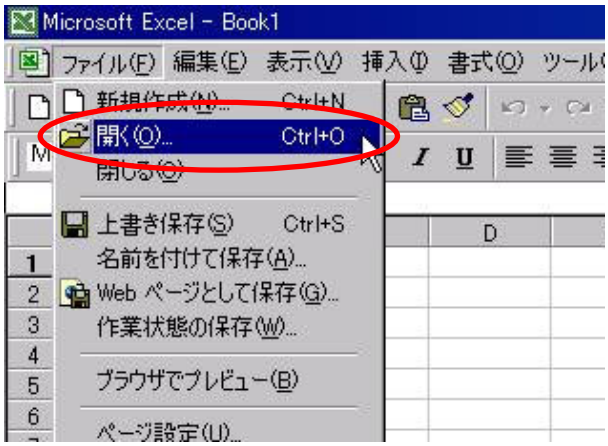
9. log.txt をエディタで開いてみました。保存されています。次は、このデータをエクセルに取り込んでみましょう。

※注意

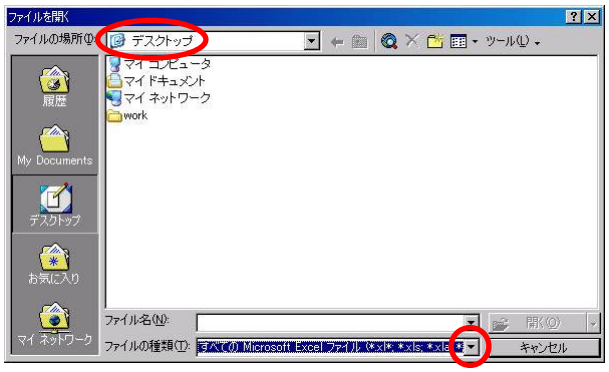
TeraTermPro は、受信前に「Flie→Log」で保存するファイル名を決めます。その後、受信したデータをファイルに保存していきます。

受信したデータは画面に表示されますが、表示されるだけで残りません。データを受信してから、「Flie→Log」を実行しても受信データは保存されませんので気をつけてください。

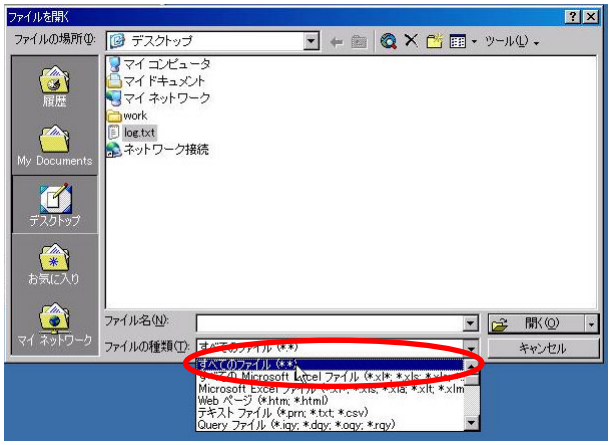
3.8 エクセルへの取り込み方



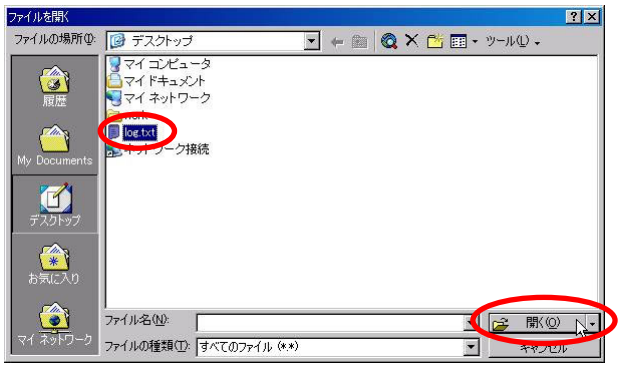
1. エクセルを立ち上げます。「ファイル→開く」を選択します。



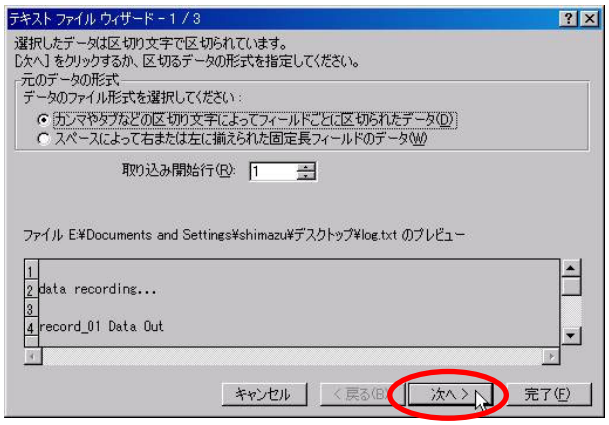
2. 「ファイルの場所」は、先ほど保存したフォルダを選びます。「ファイルの種類」の▼をクリックします。



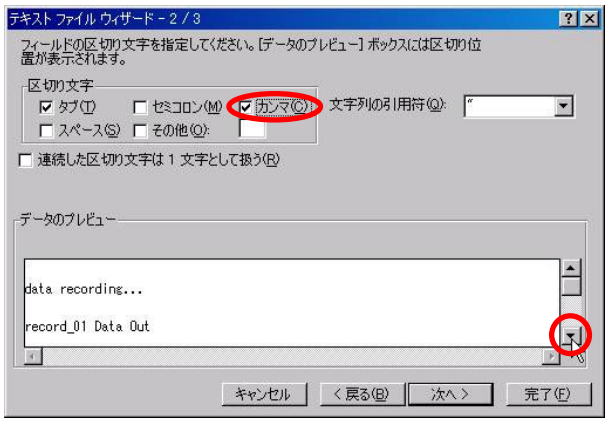
3. 「すべてのファイル(*.*)」を選択します。



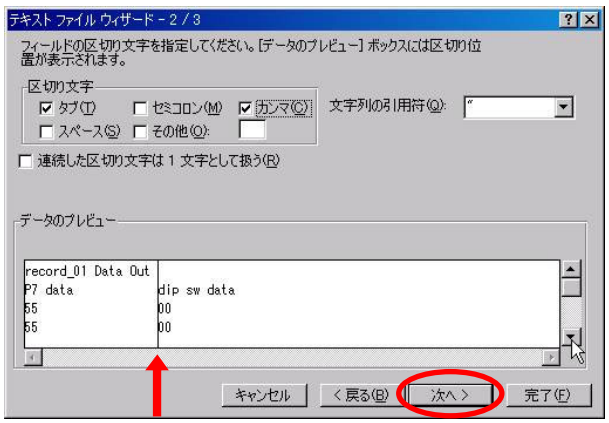
4. 「log.txt」が表示されました。「log.txt」を選択、開くをクリックします。



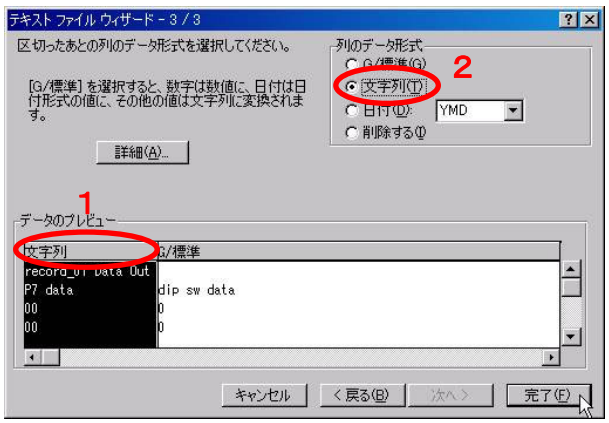
5. 開こうとしているデータは、テキストデータです。エクセルデータ(セル)に変換する作業を行います。次へをクリックします。



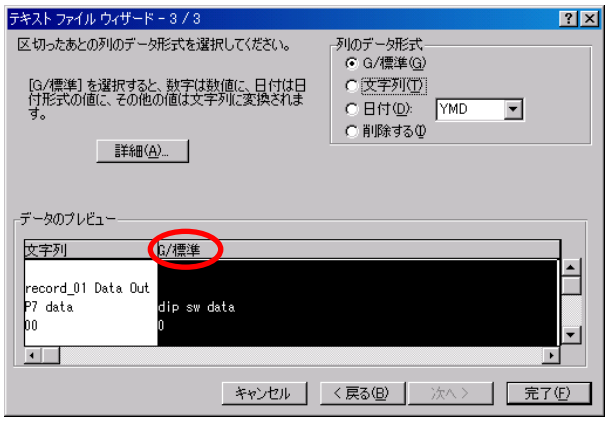
6. データはカンマで区切られています。区切り文字の「カンマ」のチェックを付けます。▼でデータのプレビュー欄を下げます。



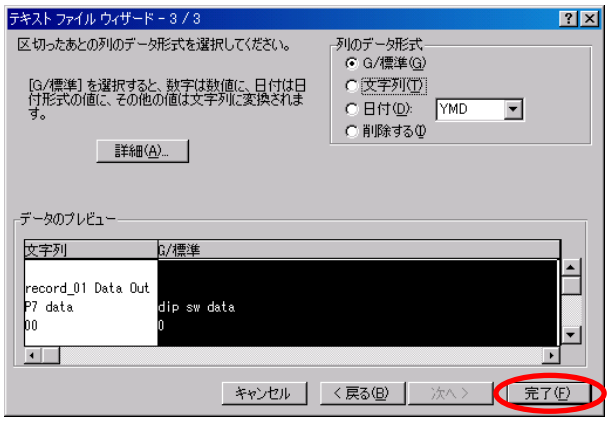
7. 矢印(↑)部分に縦線が入りました。これが列の区切りです。縦線が入ってなければ「区切り文字」のチェック指定がおかしいので、確かめます。次へをクリックします。



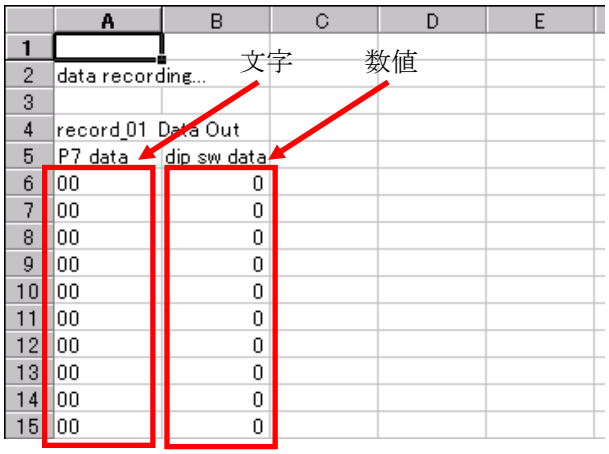
8. 次に、列のデータ形式を指定します。まず、1列目(1の○部分)をクリックします。次に2部分の「文字列」を選択します。ポート7データの列は、16進数です。そのため、文字列にします。



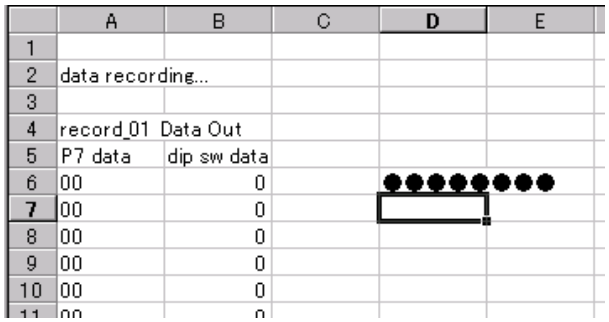
9. 次に、2 列目の○部分をクリックします。2 列目はディップスイッチ値を 10 進数で記録した内容なので、「標準」にします。最初から標準になっているので特に変更する必要はありません。



10. 完了をクリックして完了します。

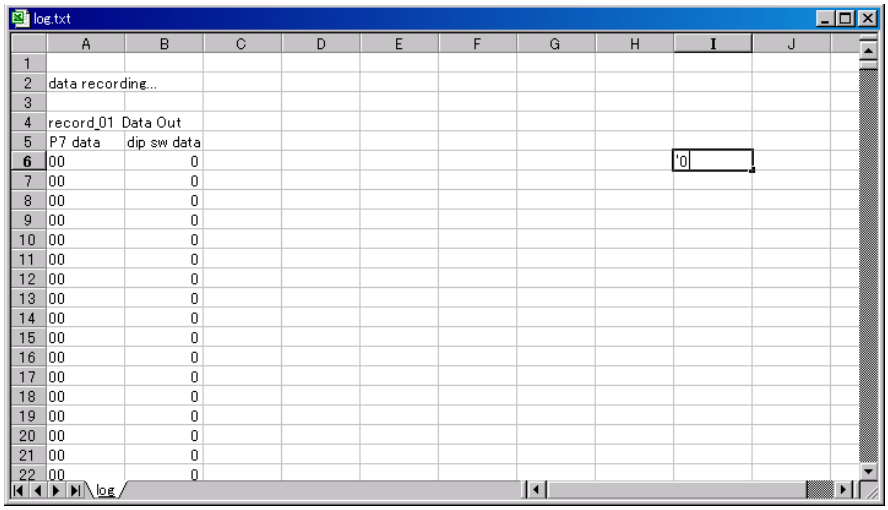


11. セルに変換されました。A 列は文字列です。例えば、A6 は「00」という文字です。B 列は数値です。例えば、B6 は「0」という数値です。関数で変換するときは、全く別な扱いになりますので気をつけてください。

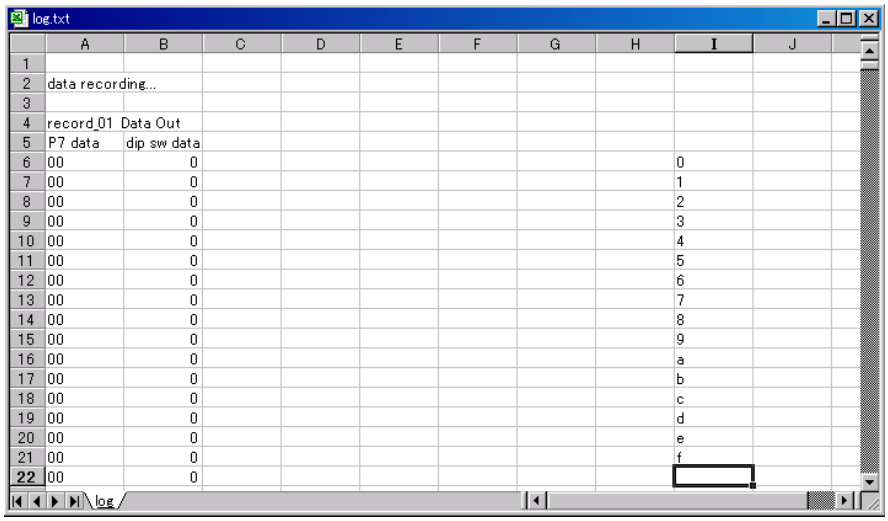


12. ポート 7 のデータは例えばセンサの値とします。センサ値が「00～ff」の 16 進数で保存されています。16 進数といっても、保存されている扱いは文字列です。これを 2 進数に変換してみましょう。0 を●、1 を○に変換してみます。打ち込んでみました。これをすべて自分で打ち込むのは非常に大変です。

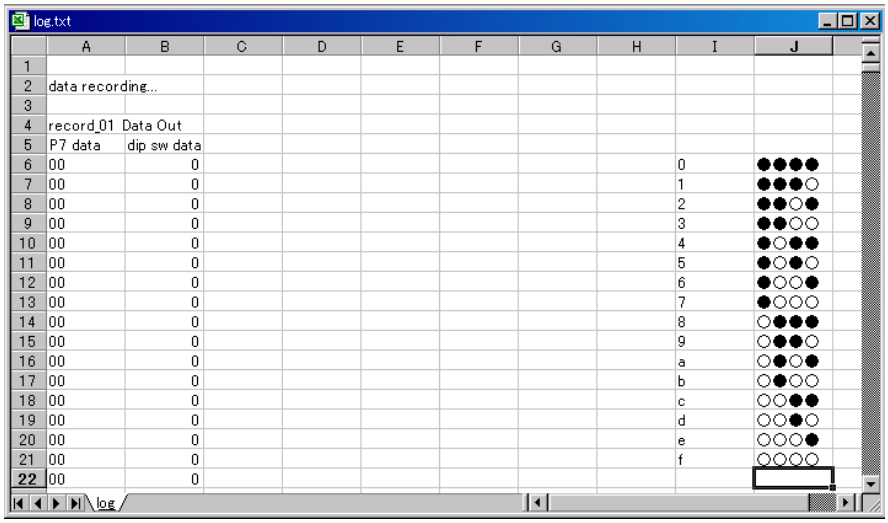
エクセルには、関数という便利な機能があります。関数を使えば、元のデータを様々な形に変換することができます。今回のように 16 進数を 2 進数に変換するなど、お手の物です。この関数を使ってポート7の 16 進数データを 2 進数に変換してみます。エクセルの関数については、書籍やホームページで多数紹介されていますので、詳しく知りたい場合はそちらを参照してください。



I 列に 0～f までの 16 進数を入力します。I6 に「’0」と半角で入力します。**この「’」がポイントです**。ただの「0」と入力すると数値になります。「’0」と先頭にアポストロフィを付けると、文字の「0」となります。数字ではありません。
「’0」と入力して、エンタを押すと、「0」とだけ表示されます。しかし、左詰めで表示されます。これが文字の「0」ですよという意味です。ちなみに B6 は右詰で「0」が表示されています。これは数字の「0」ですよ、という意味です。

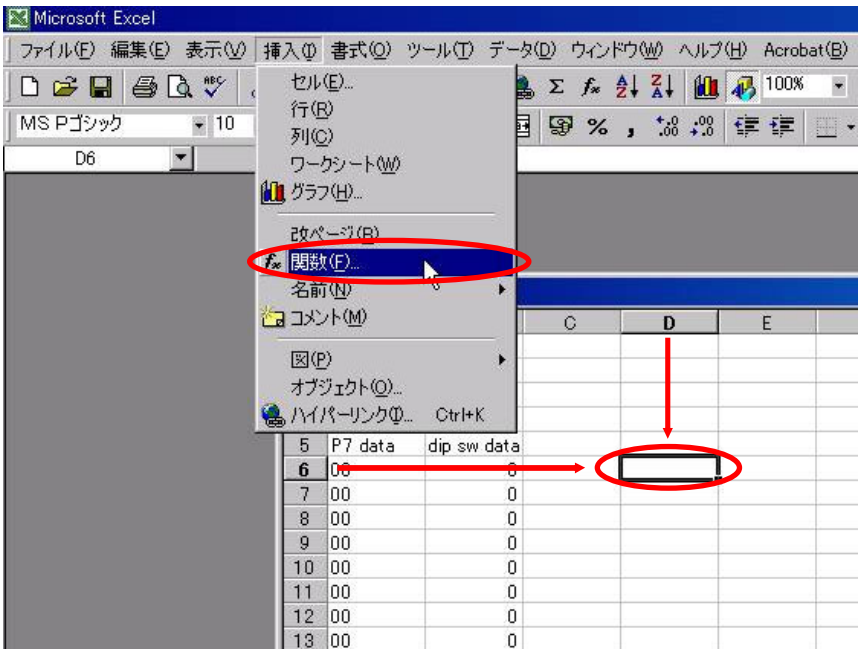


その下に「’1」と入力、エンタを押します。その下に「’2」…というように入力していきます。16 進数なので、「’9」の次は「’a」です。最後は「’f」です。
※アルファベットは小文字で入力します。

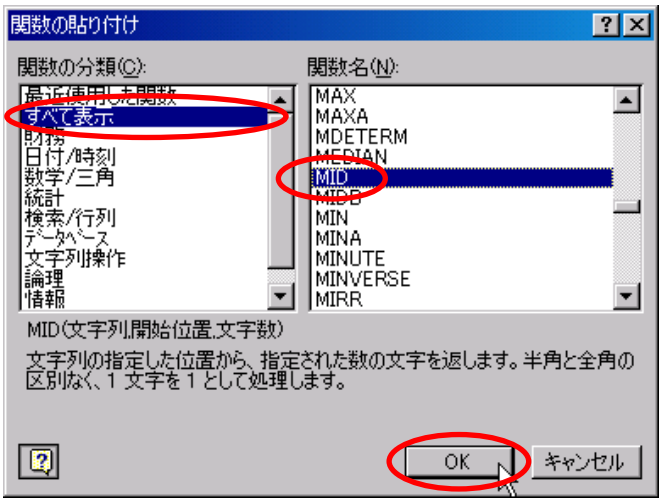


今打ち込んだ16進数の右側のセルに2進数を入力します。0＝●、1＝○として入力します。
0なら「●●●●」と入力します。最後のfは「○○○○」となります。

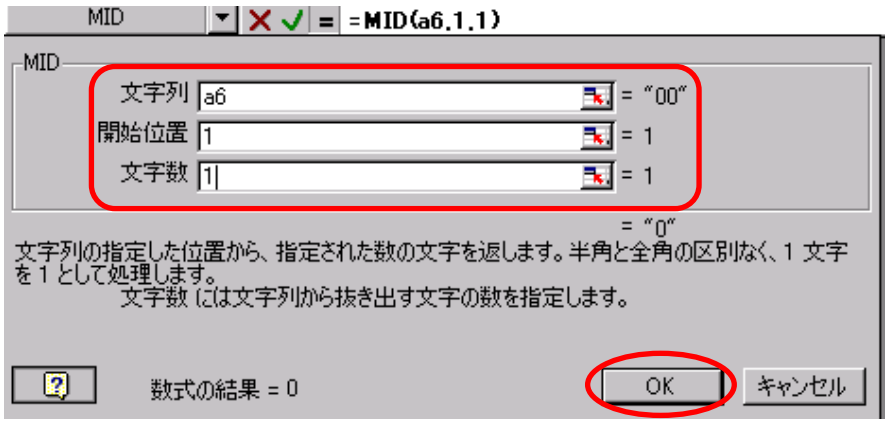
このI6からJ21までが、16進数→2進数変換表になります。



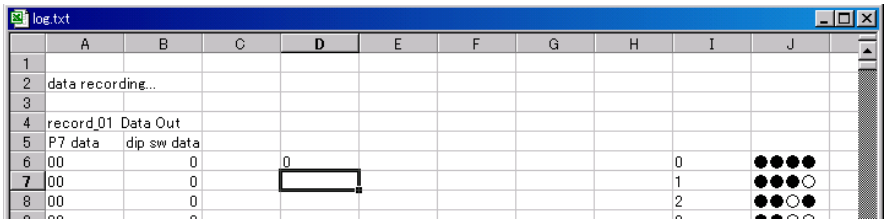
D6をクリックします。「挿入→関数」を選択します。



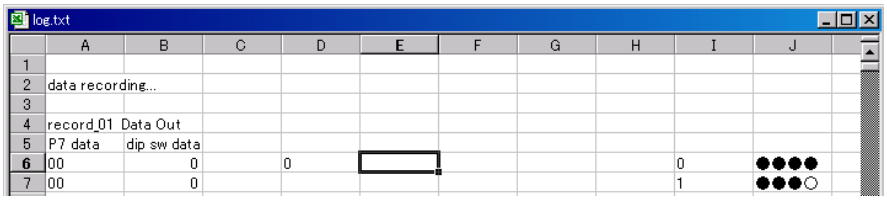
関数の分類を「**すべて表示**」、関数名は「**MID**」を選択、OK をクリックします。



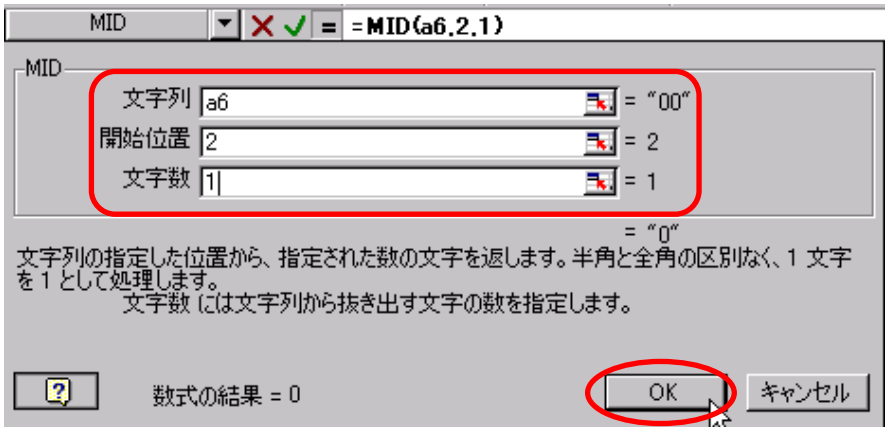
「文字列:**a6**」
「開始位置:**1**」
「文字数:**1**」
を入力、OK をクリックします。
これは、
セル A6 にある文字の
左 1 文字目から
1 文字
取り出しなさい
という意味です。



D6 に「0」と表示されました。
セル A6 の文字の
左 1 文字目から
1 文字
取り出されています。

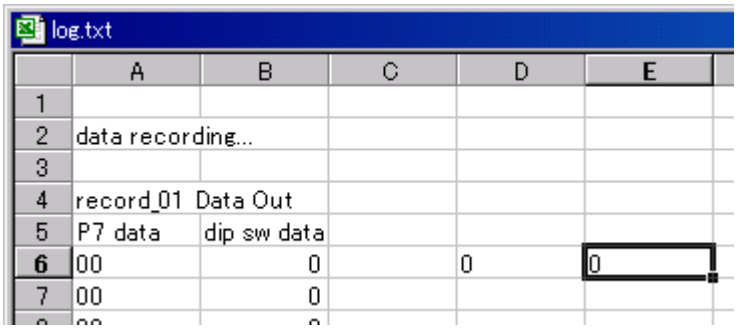


次に E6 を選択します。
「挿入→関数」を選択します。
関数の分類を「**すべて表示**」、関数名は「**MID**」を選択、OK をクリックします。

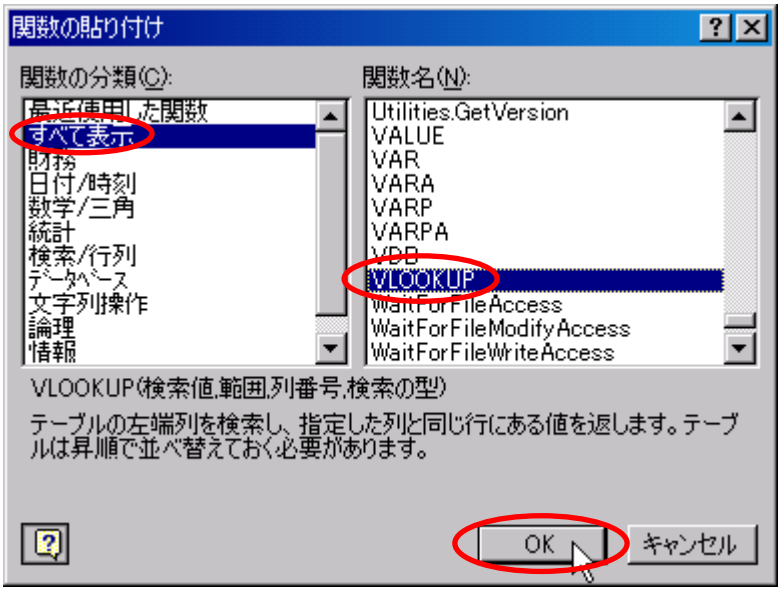


「文字列:**a6**」
「開始位置:**2**」
「文字数:**1**」
を入力、OK をクリックします。

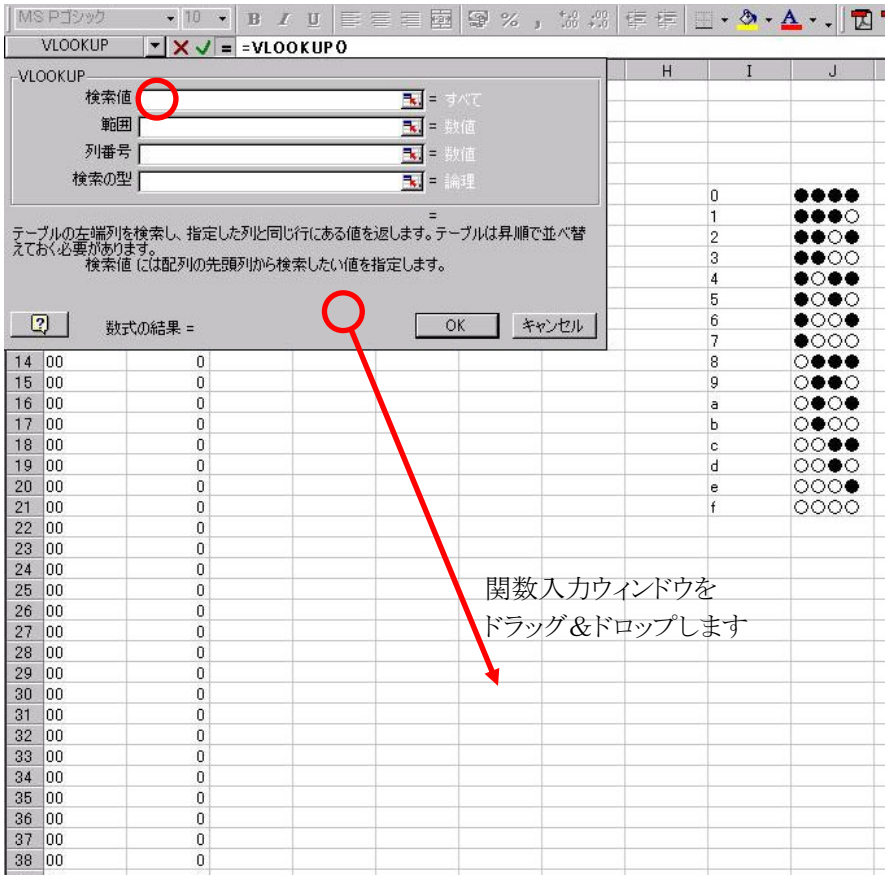
これは、
セル A6 の文字の
左 2 文字目から
1 文字
取り出しなさい
という意味です。



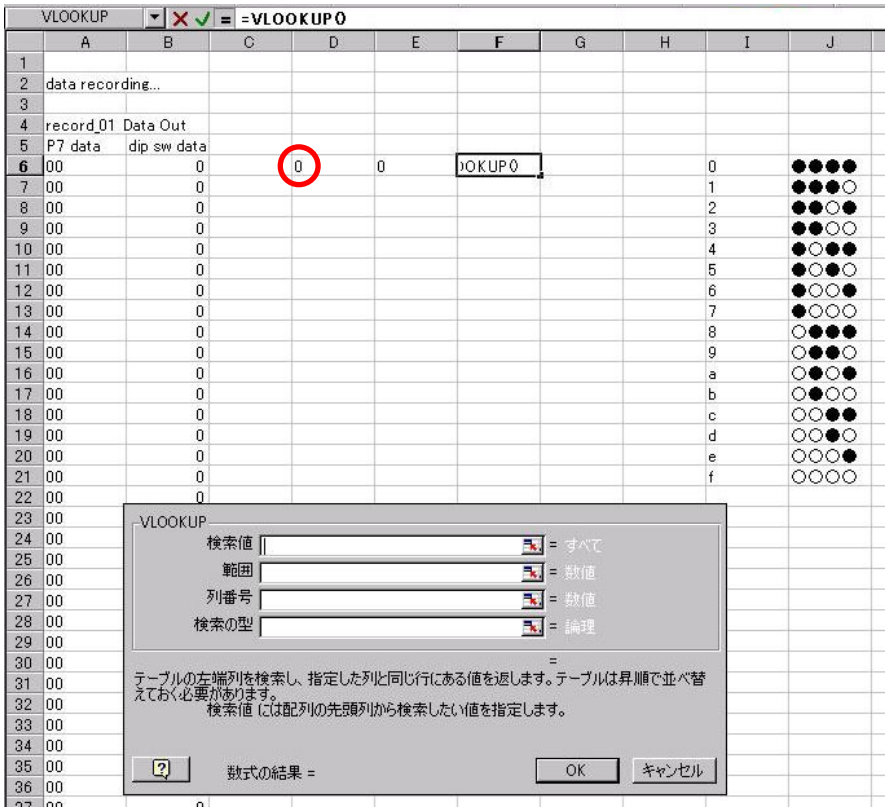
E6 に「0」と表示されました。
セル A6 の文字の
左 2 文字目から
1 文字
取り出されています。



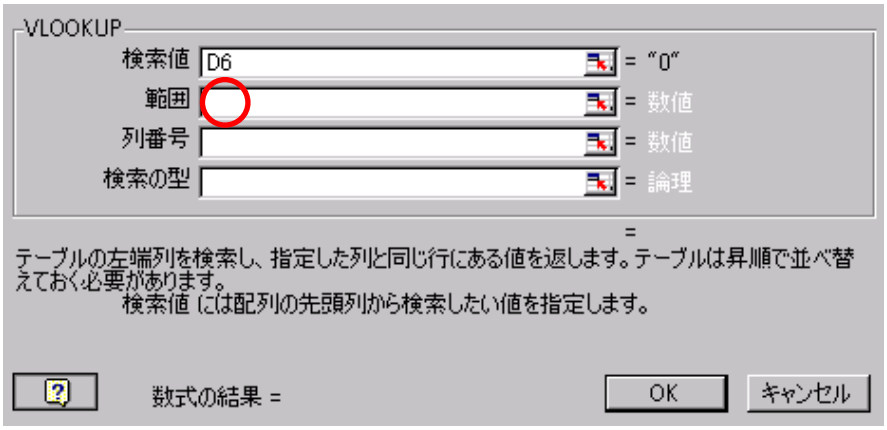
次にセルF6を選択します。
「挿入→関数」を選択します。
関数の分類を「**すべて表示**」、関数名は「**VLOOKUP**」を選択、OK をクリックします。



まず、検索値の空欄をクリックします。
関数入力ウィンドウをドラッグ&ドロップで、下に移動します。

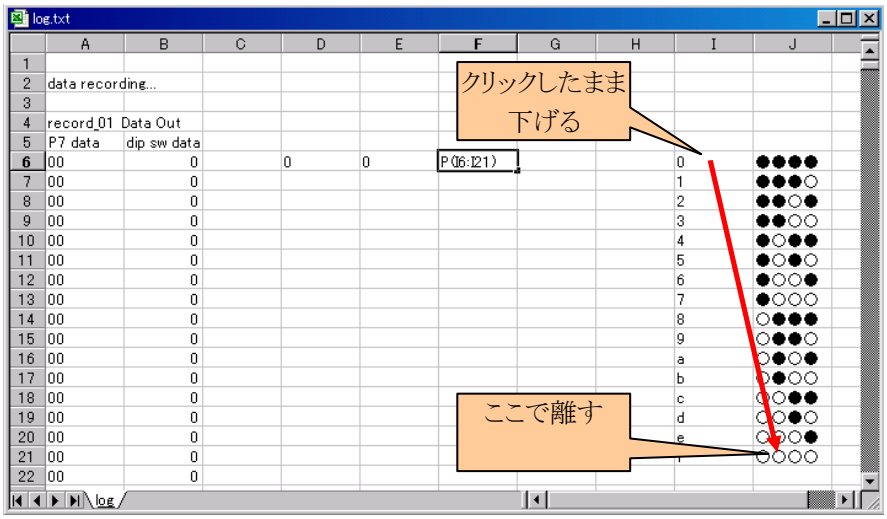


変換元である D6 をクリックします。

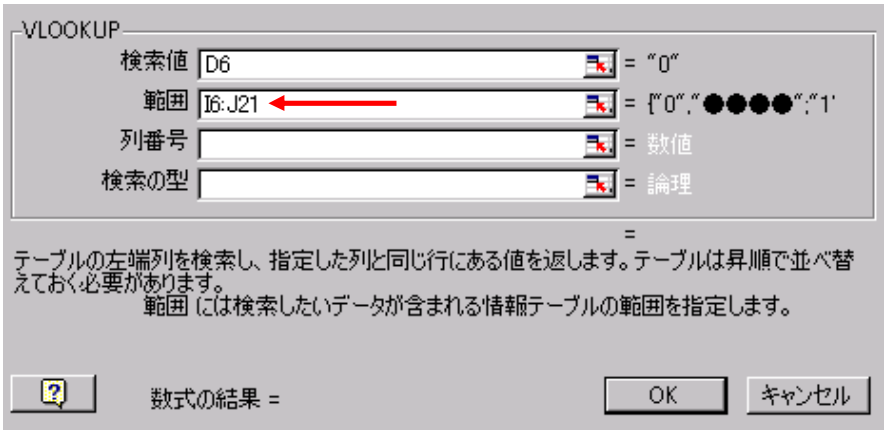


検索値の欄には
D6
と入力されます。

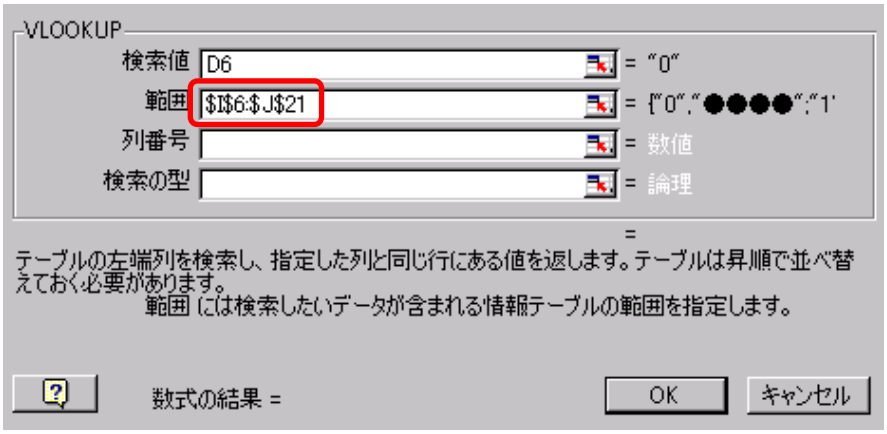
次に「範囲」の欄をクリックします。○部分です。



変換表全体を選択します。I6の「0」をクリックします。クリックしたまま、J21の「○○○○」までマウスを持っていき、離します。
（変換表の左上から右下）



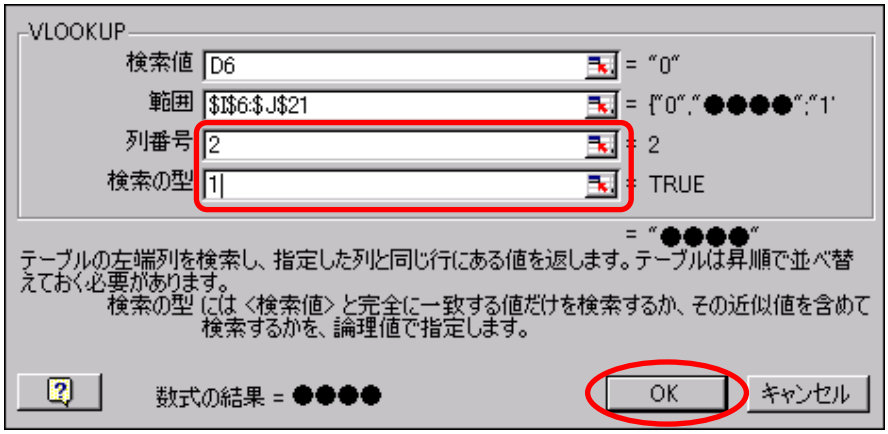
範囲の欄には
I6:J21
と入力されます。



範囲欄を下記のように「**$**」を 4 箇所追加します。

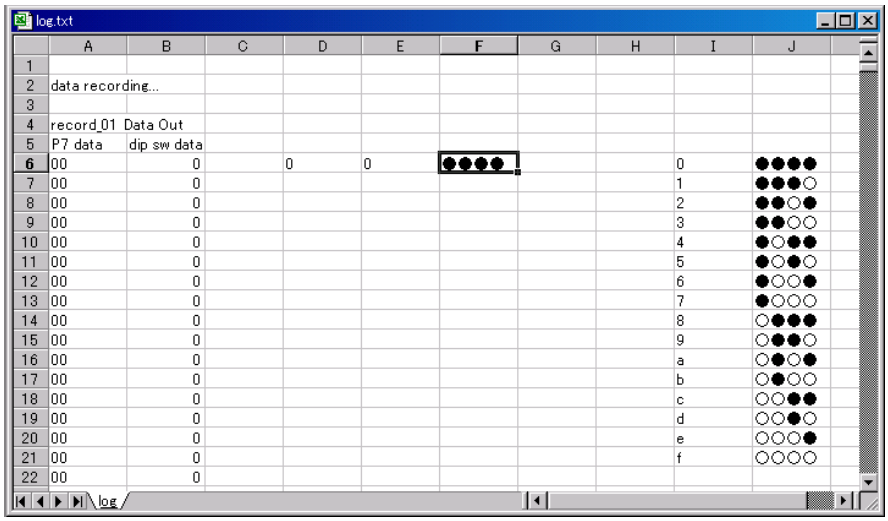
$I$6：$J$21

※「＄」は絶対参照という意味です。検索サイトで
「エクセル　絶対参照」
で調べるといろいろ出てきます。

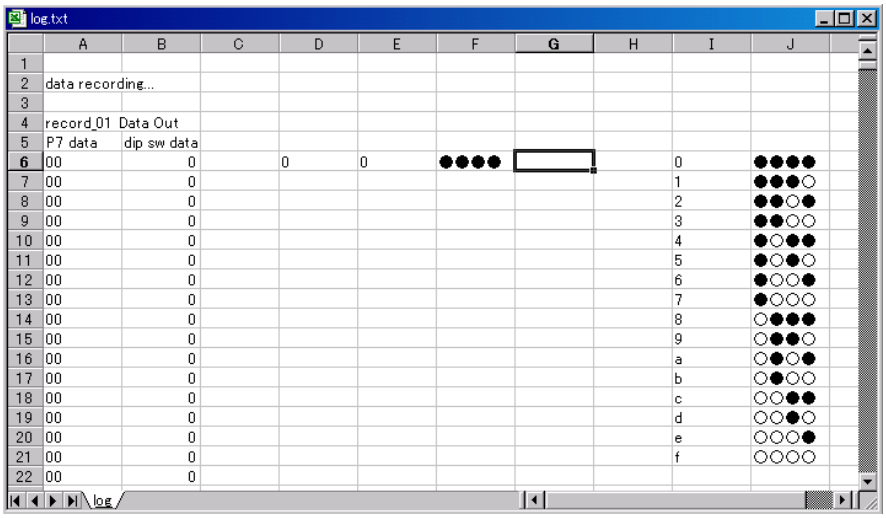


「列番号:**2**」
「検索の型:**1**」
と入力します。

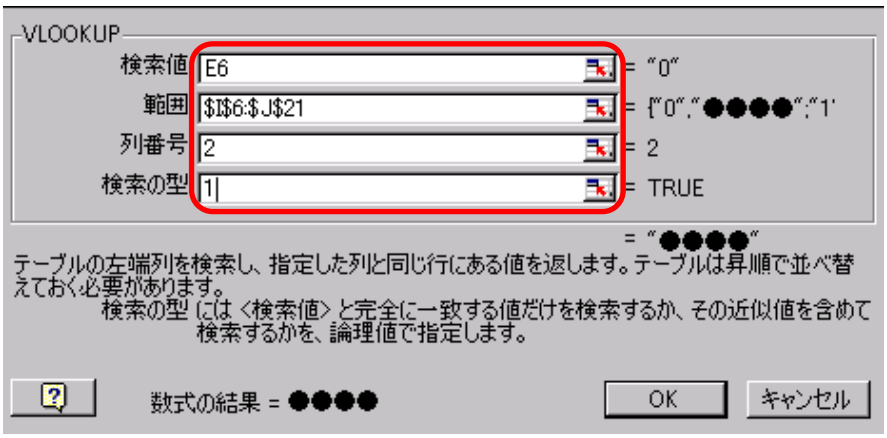
OK をクリックします。



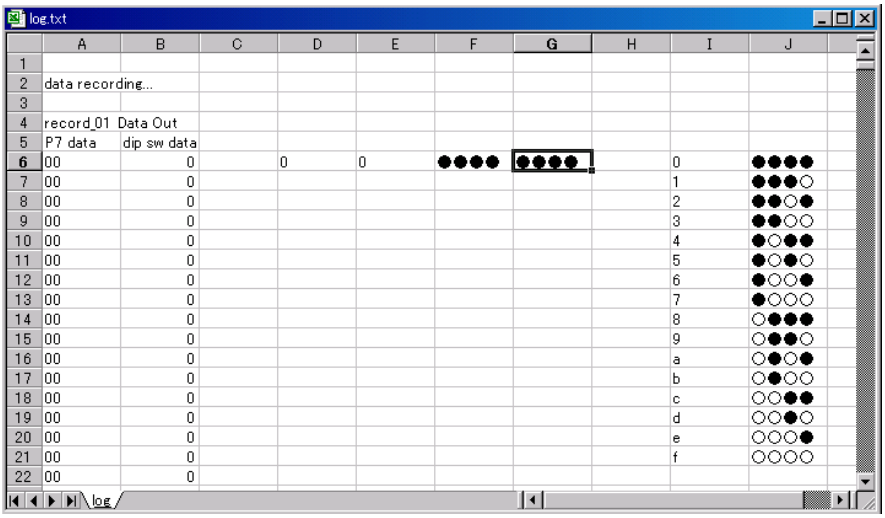
セル F6 には、セル D6 に入力されている 16 進数を 2 進数に変換した値が表示されます。
今回、D6 は「0」なので「●●●●」です。



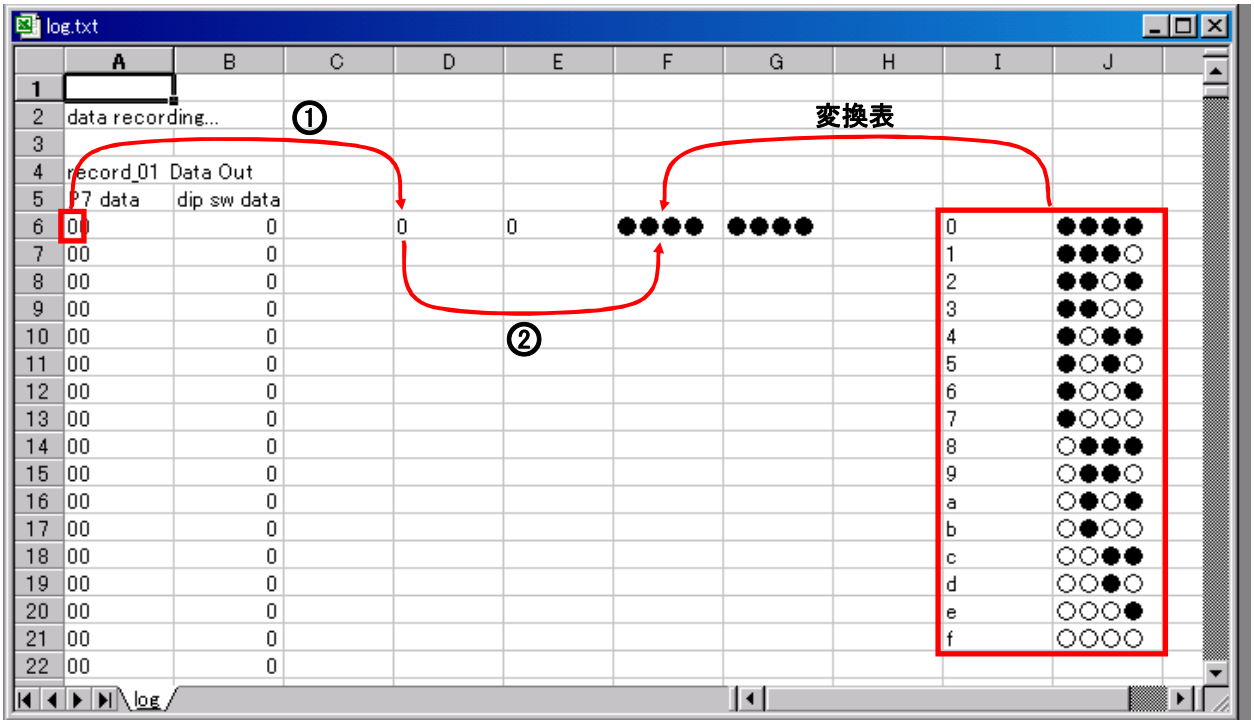
次にセル G6 を選択します。
「挿入→関数」を選択します。
関数の分類を「**すべて表示**」、関数名は「**VLOOKUP**」を選択、OK をクリックします。



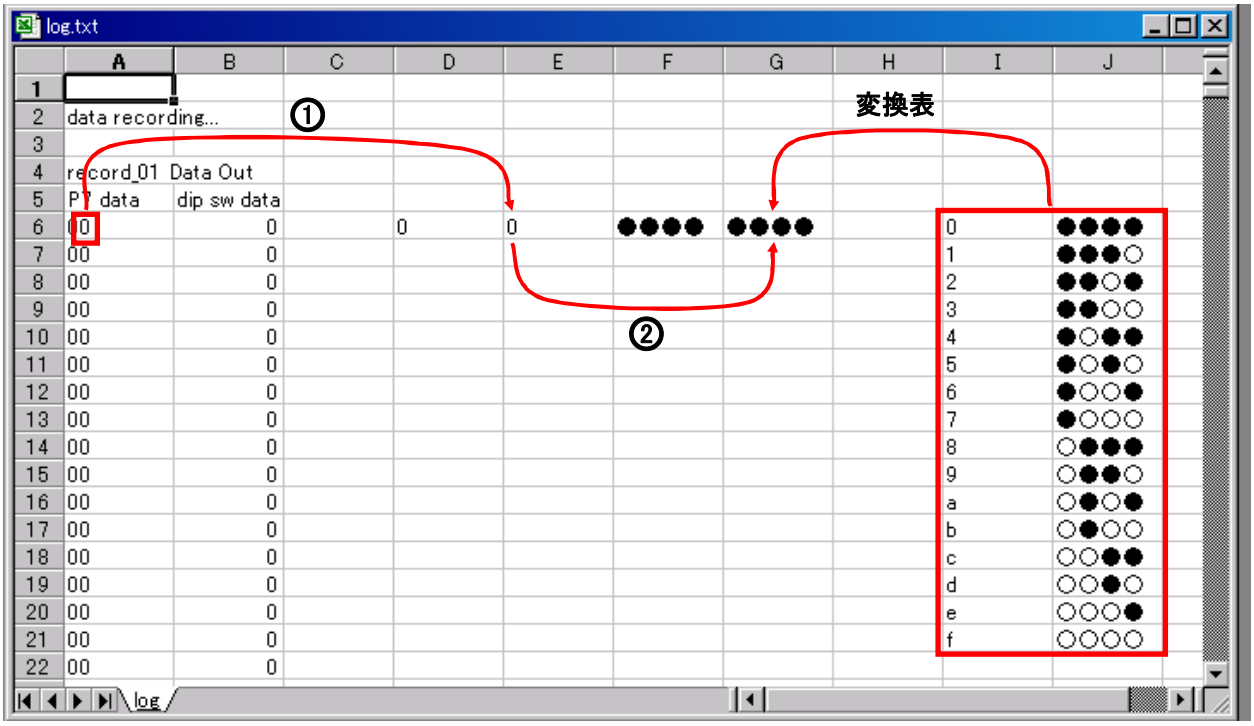
「検索値:**E6**(変換元)」
「範囲:**$I$6:$J$21**」
(変換表の左上から右下)
「列番号:**2**」
「検索の型:**1**」
を入力します。OK をクリックします。



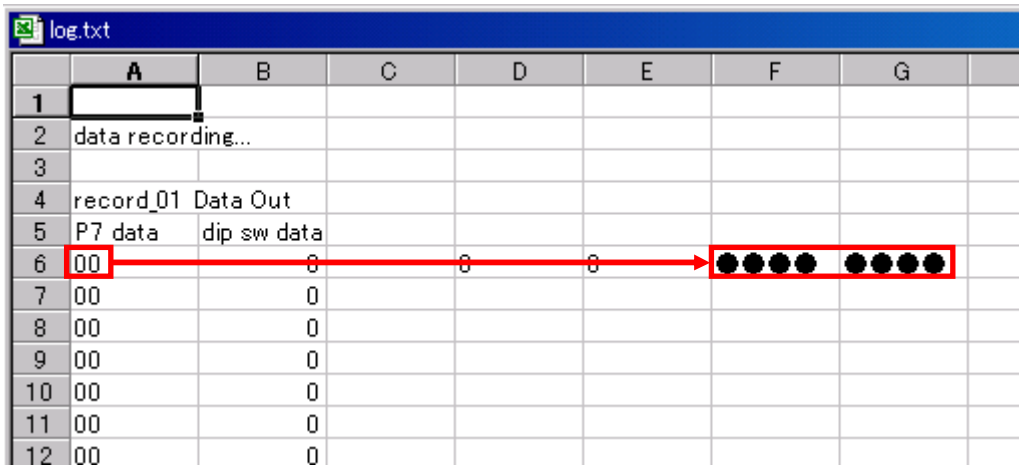
セルG6には、セルE6の値を16進数に変換した値が表示されます。
今回、E6は「0」なので「●●●●」です。



まとめると、
①…D6 は、A6 の16進数2桁のデータから、10の位のみを取り出します。
②…F6 は、D6 と変換表を見比べます。変換表内のI6に「0」があります。すぐ右の「●●●●」を取り出して表示します。



①…E6 は、A6 の16進数2桁のデータから、1の位のみを取り出します。
②…G6 は、E6 と変換表を見比べます。変換表内のI6に「0」があります。すぐ右の「●●●●」を取り出して表示します。

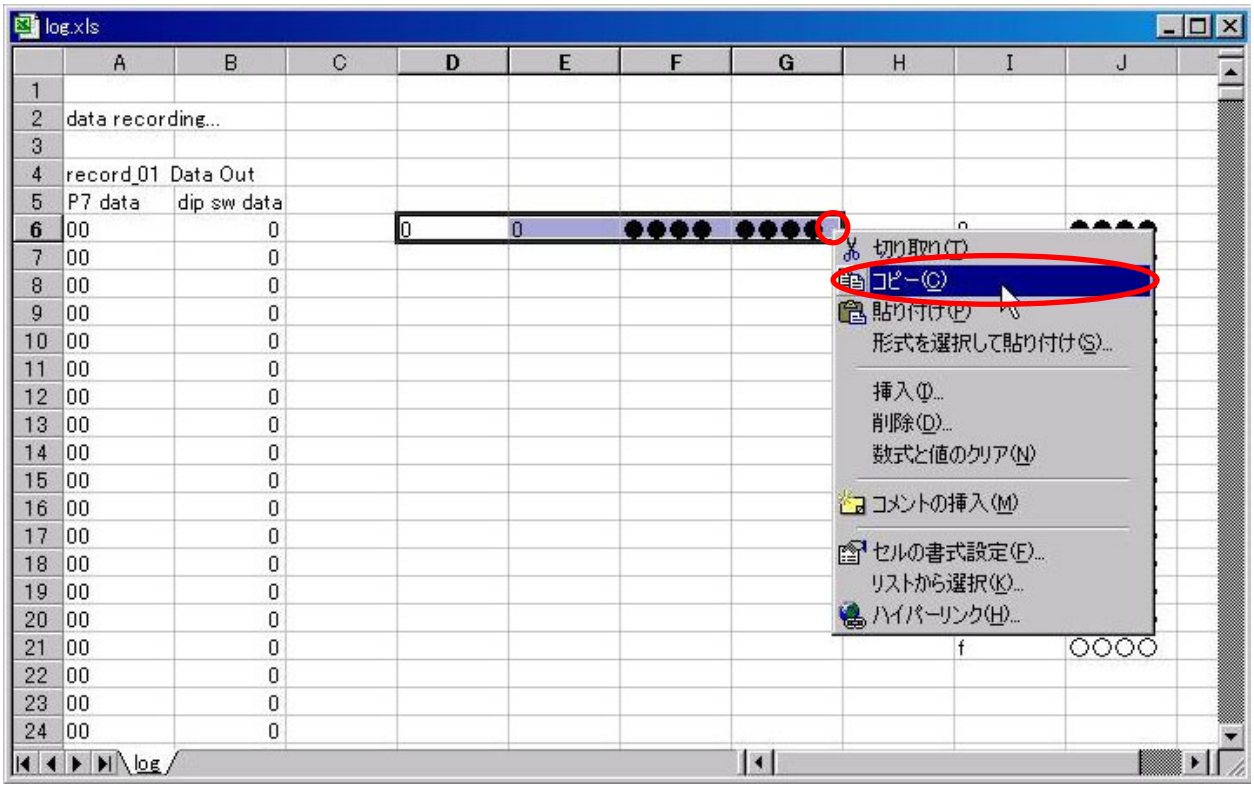


結果、あたかも、**セルA6にあるの16進数2桁「00」が、セルF6とG6に2進数「0000　0000」で変換されているようになります**。

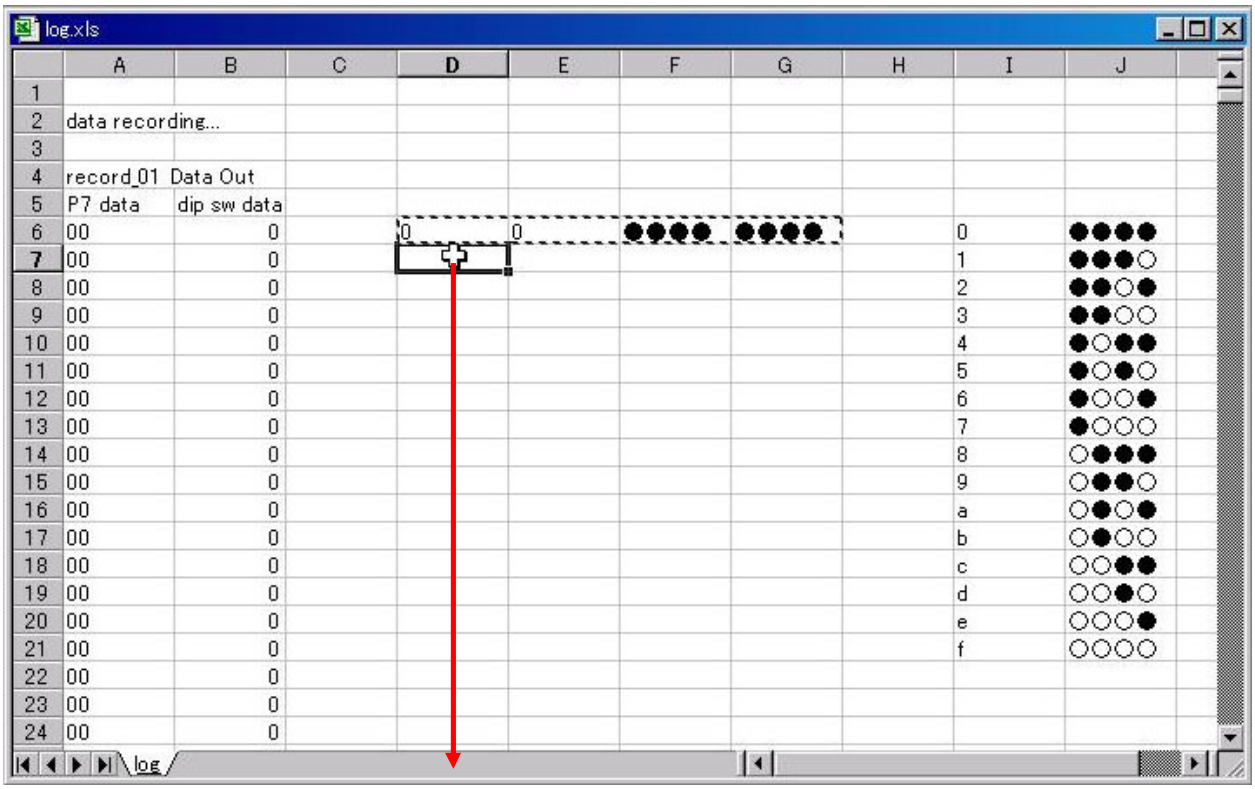
本当は、セル A6 から直接 2 進数に変換すれば良いのですが、このような関数がありません。そのため、段階的に変換して、最後に表示したい形式にします。

変換表も「００」～「ｆｆ」まで256通り記入すれば良いのですが、入力が大変です。そのため、16進数を1桁ずつに分けて「0」～「f」までの 16 通りを入力、変換しています。

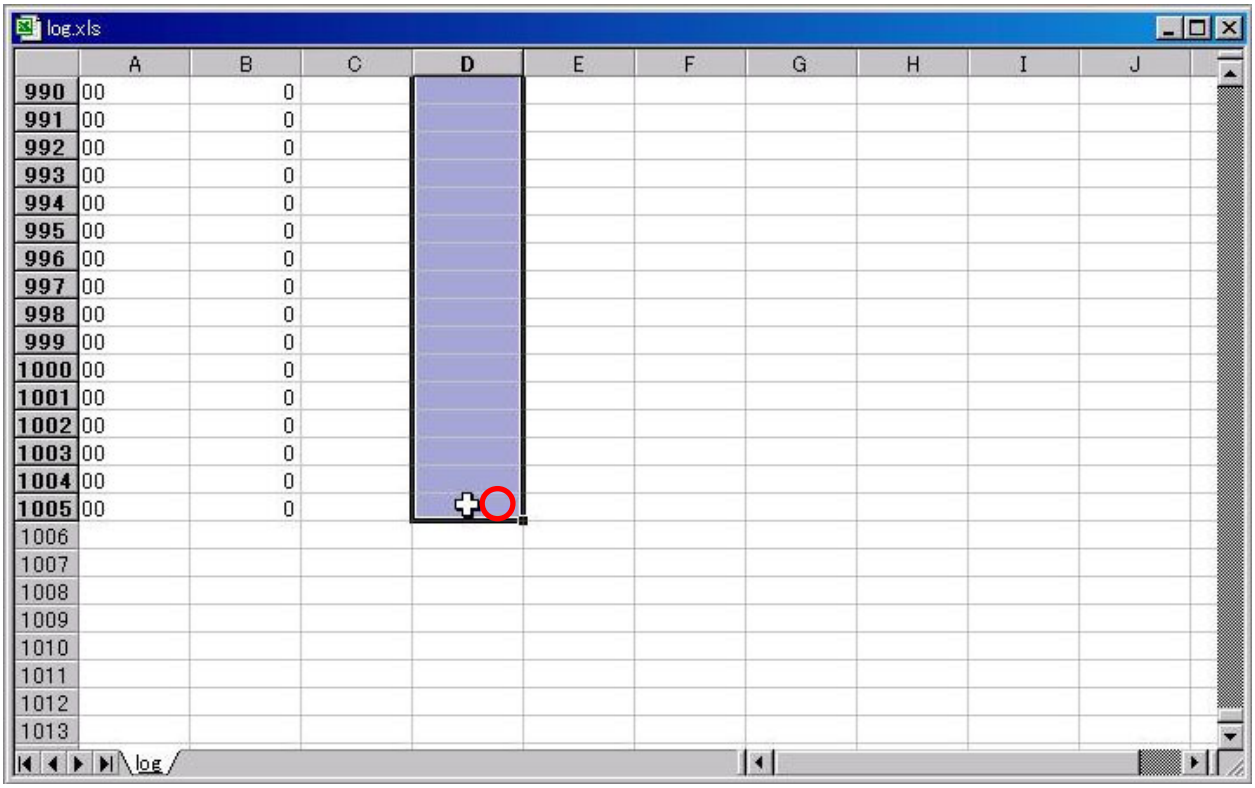
今回、6 行目のデータを変換しました。6 行目より下には 999 個分のデータが続きます。同様に 999 回繰り返します･･･　そんな時間はありません！コピー作業を使えば、1回の作業で実現できます。



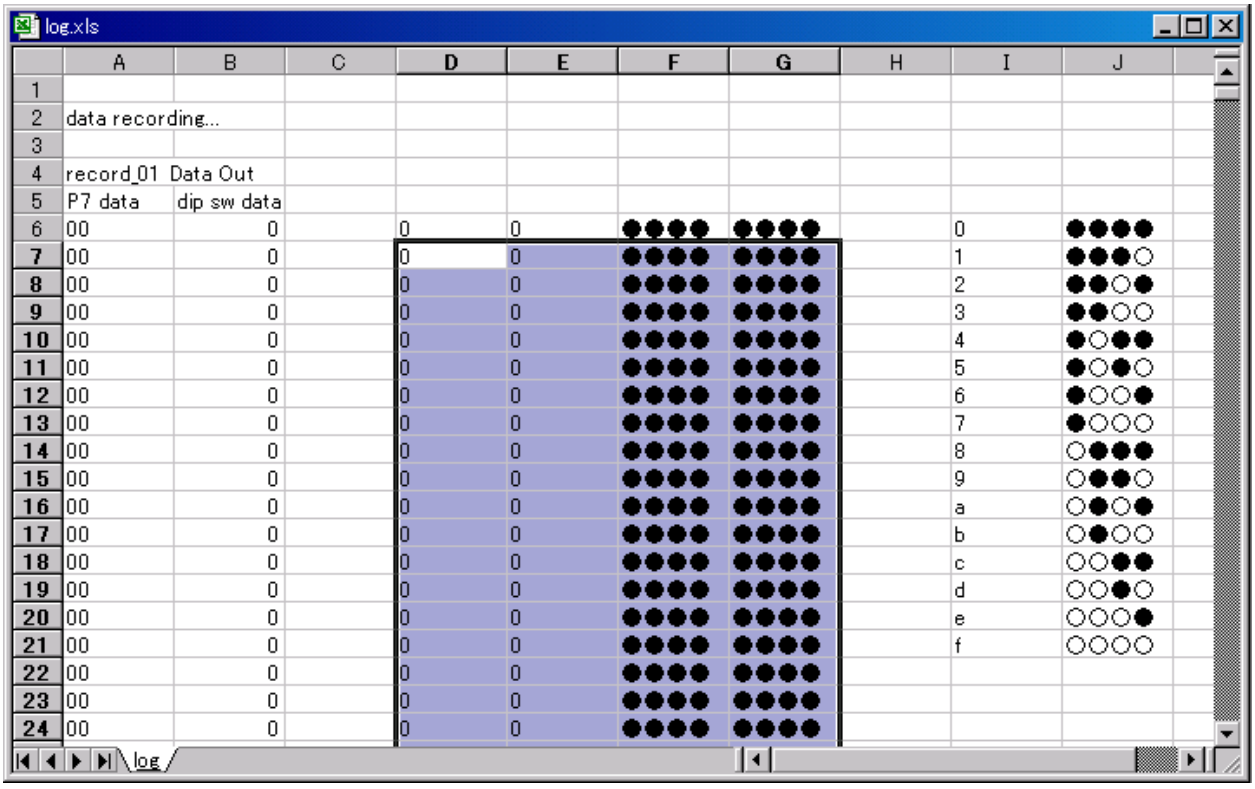
D6～G6のセルを選択して、○当たりで右クリック、「コピー」を選択します。



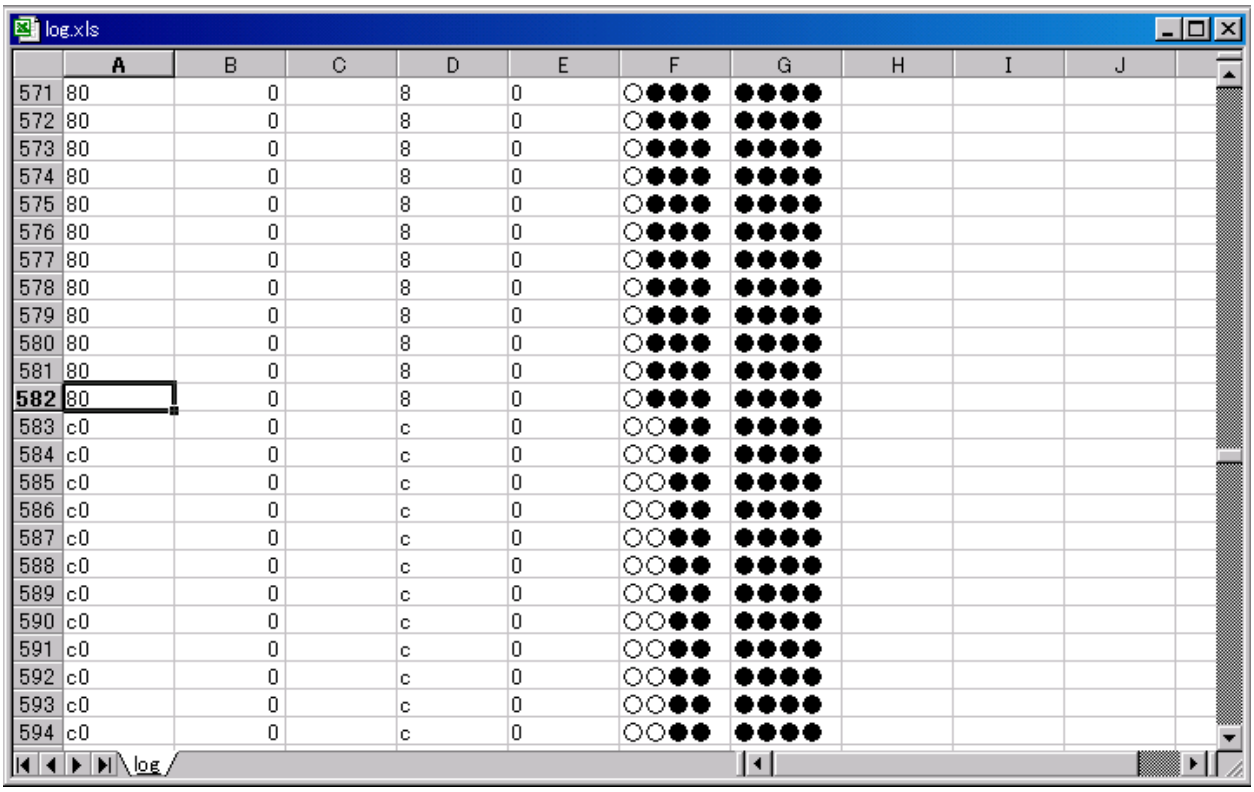
セル D7 をクリック、そのまま下に降ろしていきます。画面をはみ出してもクリックしたままにしておきます。自動的に下へスクロールします。



最後の行までマウスを左クリックしたまま下に持ってきます。下へ行きすぎたら、上にカーソルを上げると、戻ります(マウスの左ボタンは押したままです)。D1005 まで来たら、左クリックを止めます。〇部分で右クリック、「貼り付け」を選択します。



全行に貼り付けられました。



例えば、582 行は 16 進数「80」、2 進数「○●●● ●●●●」、583 行は、16 進数「C0」、2 進数「○○●● ●●●●」と変換されています。きちんと変換されていることが分かります。

↓キーを押し続けます。●と○が、あたかも実際に反応して動いているように表示されます。パラパラアニメのようです。マイコンカーのセンサの値をこの方法で変換、解析してみるとセンサの反応状態が分かります。もしかしたら、自分が予期していないセンサ状態があるかもしれません。いろいろ解析してみると良いでしょう。

4. プロジェクト「record_02」 外付けEEP-ROMにデータ保存

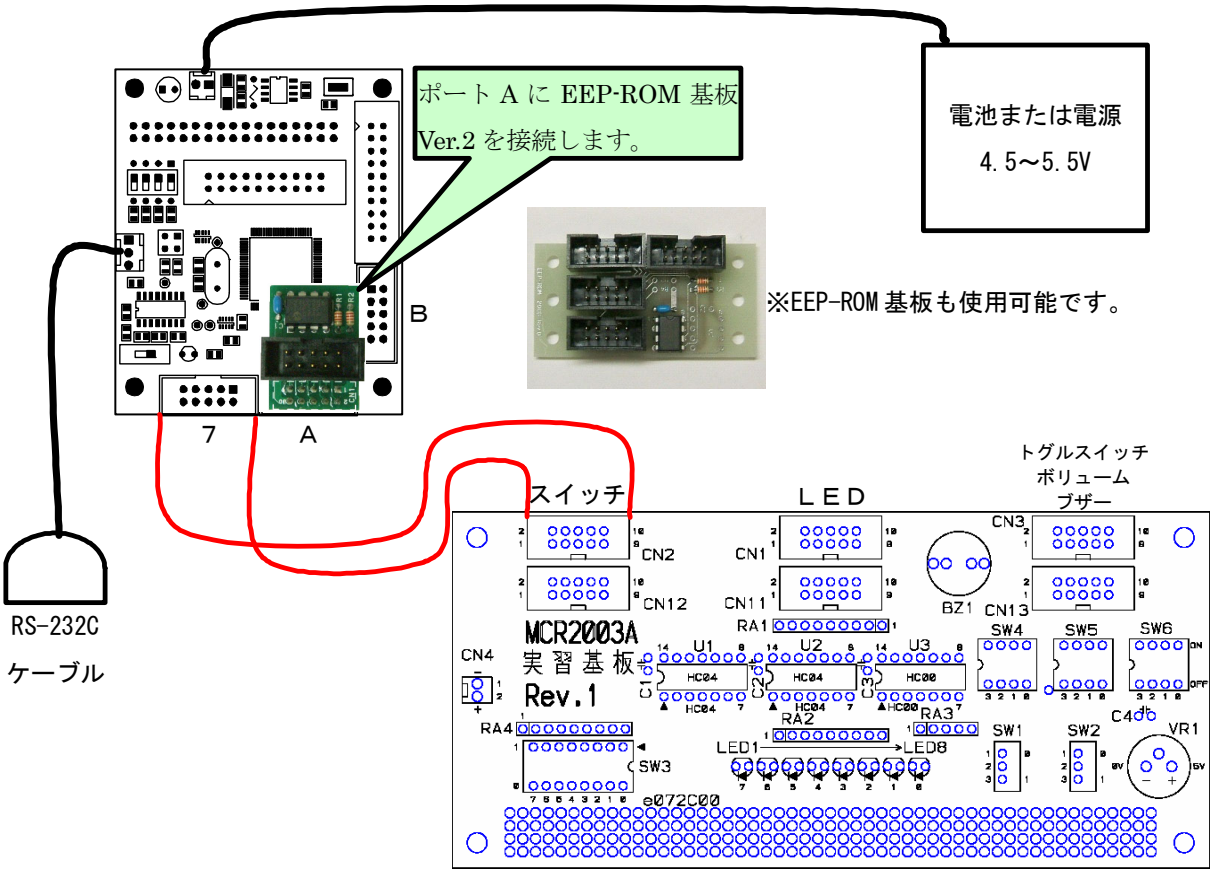
4.1 概要

このプログラムは、
・ポート 7 に接続されているディップスイッチの値
・CPU ボードのディップスイッチの値
を、10ms ごとに**外付け EEP-ROM** に保存します。保存時間は 10 秒です。保存時間は自由に変えられますが上限はメモリがいっぱいになるまでです(今回は約 164 秒、1 回に保存するデータ数で変わります)。保存後、RS232C を通してパソコンへ保存した情報を出力します。

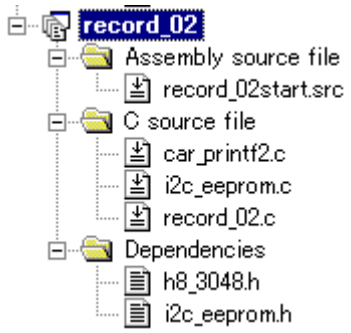
ここでは、
・前章と比べて、EEP-ROM を使った場合の変更点
について、説明します。

4.2 接続

・CPU ボードのポート 7 と、実習基板のスイッチ部をフラットケーブルで接続します。
・CPU ボードのポート A と、EEP-ROM 基板 Ver.2 を接続します。
※ポート 7 のディップスイッチをセンサ基板に変えると、センサの反応を記録することができます。



4.3 プロジェクトの構成



・record_02start.src
・record_02.c
・car_printf2.c
・i2c_eeprom.c
の 4 ファイルあります。
h8_3048.h は record_02.c、car_printf2.c、i2c_eeprom.c でインクルードされているファイルです。
i2c_eeprom.h は record_02.c、i2c_eeprom.c でインクルードされているファイルです。

4.4 プログラム

```
 1 : /*****************************************************************************/
 2 : /* 外付けEEP-ROMデータ保存演習プログラム「record_02.c」                       */
 3 : /*                              2006.04 ジャパンマイコンカーラリー実行委員会 */
 4 : /*****************************************************************************/
 5 : /*
 6 : 本プログラムは外付けEEP-ROM(24C256 32KB)にポート７のデータ、
 7 : CPUボード上のディップスイッチの値を10msごとに保存します。
 8 : その後、保存したデータをパソコンへ転送します。
 9 : */
10 :
11 : /*====================================*/
12 : /* インクルード                       */
13 : /*====================================*/
14 : #include    <no_float.h>              /* stdioの簡略化 最初に置く*/
15 : #include    <stdio.h>
16 : #include    <machine.h>
17 : #include    "h8_3048.h"
18 : #include    "i2c_eeprom.h"            /* EEP-ROM追加(データ記録)  */
19 :
20 : /*====================================*/
21 : /* シンボル定義                       */
22 : /*====================================*/
23 :
24 : /*====================================*/
25 : /* プロトタイプ宣言                   */
26 : /*====================================*/
27 : void init( void );
28 : unsigned char dipsw_get( void );
29 :
30 : /*====================================*/
31 : /* グローバル変数の宣言               */
32 : /*====================================*/
33 : unsigned long   cnt1;                 /* main内で使用             */
34 : int             pattern;              /* パターン番号             */
35 :
36 : /* データ保存関連 */
37 : int             iTimer10;             /* 取得間隔計算用           */
38 : int             saveIndex;            /* 保存インデックス         */
39 : int             saveSendIndex;        /* 送信インデックス         */
40 : int             saveFlag;             /* 保存フラグ               */
41 : char            saveData[8];          /* 一時保存エリア           */
42 :
43 : /*****************************************************************************/
44 : /* メインプログラム                                                          */
45 : /*****************************************************************************/
46 : void main( void )
47 : {
48 :     /* マイコン機能の初期化 */
49 :     init();                           /* 初期化                   */
50 :     initI2CEeprom( &PADDR, &PADR, 0x5f, 7, 5);  /* EEP-ROM初期設定  */
51 :     init_sci1( 0x00, 79 );            /* SCI1初期化               */
52 :     set_ccr( 0x00 );                  /* 全体割り込み許可         */
53 :
54 :     while( 1 ) {
55 :
56 :     I2CEepromProcess();               /* I2C EEP-ROM保存処理      */
57 :
58 :     switch( pattern ) {
59 :     case 0:
60 :         /* EEP-ROMクリア */
61 :         clearI2CEeprom();             /* 数秒かかる               */
```

```
 62 :          printf( "¥n" );
 63 :          printf( "data recording...¥n" );
 64 :          pattern = 1;
 65 :          saveIndex = 0;
 66 :          saveFlag = 1;                   /* データ保存開始           */
 67 :          cnt1 = 0;
 68 :          break;
 69 :
 70 :      case 1:
 71 :          /* データ保存中  保存自体は割り込みの中で行う */
 72 :          if( cnt1 >= 10000 ) {
 73 :              saveFlag = 0;               /* 保存終了                 */
 74 :              pattern = 2;                /* データ転送処理へ         */
 75 :          }
 76 :          break;
 77 :
 78 :      case 2:
 79 :          /* タイトル転送、準備 */
 80 :          while( !checkI2CEeprom() );     /* 最後のデータ書き込むまで待つ*/
 81 :
 82 :          printf( "¥n" );
 83 :          printf( "record_02 Data Out¥n" );
 84 :          printf( "P7 data,dip sw data¥n" );
 85 :
 86 :          saveSendIndex = 0;
 87 :          pattern = 3;
 88 :          break;
 89 :
 90 :      case 3:
 91 :          /* データ転送 */
 92 :          printf( "%02x,%02x¥n",
 93 :              (unsigned char)readI2CEeprom( saveSendIndex+0 ),
 94 :              (unsigned char)readI2CEeprom( saveSendIndex+1 ) );
 95 :          saveSendIndex += 2;
 96 :          if( saveIndex <= saveSendIndex ) {
 97 :              pattern = 4;
 98 :              cnt1 = 0;
 99 :          }
100 :          break;
101 :
102 :      case 4:
103 :          /* 転送終了 */
104 :          break;
105 :
106 :      default:
107 :          /* どれでもない場合は待機状態に戻す */
108 :          pattern = 0;
109 :          break;
110 :      }
111 :      }
112 :  }
113 :
114 :  /************************************************************************/
115 :  /* H8/3048F-ONE 内蔵周辺機能  初期化                                    */
116 :  /************************************************************************/
117 :  void init( void )
118 :  {
119 :      /* I/Oポートの入出力設定 */
120 :      P1DDR = 0xff;
121 :      P2DDR = 0xff;
122 :      P3DDR = 0xff;
123 :      P4DDR = 0xff;
124 :      P5DDR = 0xff;
125 :      P6DDR = 0xf0;                       /* CPU基板上のDIP SW        */
126 :      P8DDR = 0xff;
127 :      P9DDR = 0xf7;                       /* 通信ポート               */
128 :      PADDR = 0x5f;                       /* EEP-ROM基板              */
129 :      PBDDR = 0xff;
130 :      /* ポート７は、入力専用なので入出力設定はありません   */
131 :
132 :      /* ITU0 1msごとの割り込み */
133 :      ITU0_TCR = 0x20;
134 :      ITU0_GRA = 24575;
135 :      ITU0_IER = 0x01;
136 :
137 :      /* ITUのカウントスタート */
138 :      ITU_STR = 0x01;
139 :  }
140 :
141 :  /************************************************************************/
142 :  /* ITU0 割り込み処理                                                    */
143 :  /************************************************************************/
144 :  #pragma interrupt( interrupt_timer0 )
145 :  void interrupt_timer0( void )
146 :  {
147 :      ITU0_TSR &= 0xfe;                   /* フラグクリア             */
148 :      cnt1++;
149 :
150 :      /* データ保存関連 */
151 :      iTimer10++;
152 :      if( iTimer10 >= 10 ) {
```

```
153 :          iTimer10 = 0;
154 :          if( saveFlag ) {
155 :              saveData[0] = P7DR;
156 :              saveData[1] = dipsw_get();
157 :              setPageWriteI2CEeprom( saveIndex, 2, saveData );
158 :              saveIndex += 2;
159 :              if( saveIndex >= 0x8000 ) saveFlag = 0;
160 :          }
161 :      }
162 :  }
163 :
164 :  /************************************************************************/
165 :  /* ディップスイッチ値読み込み                                           */
166 :  /* 戻り値 スイッチ値 0～15                                              */
167 :  /************************************************************************/
168 :  unsigned char dipsw_get( void )
169 :  {
170 :      unsigned char sw;
171 :
172 :      sw  = ~P6DR;                        /* ディップスイッチ読み込み */
173 :      sw &= 0x0f;
174 :
175 :      return  sw;
176 :  }
177 :
178 :  /************************************************************************/
179 :  /* end of file                                                          */
180 :  /************************************************************************/
```

4.5 プログラムの解説

4.5.1 内蔵周辺機能の初期化

```
114 :  /*********************************************************************/
115 :  /* H8/3048F-ONE 内蔵周辺機能　初期化                                 */
116 :  /*********************************************************************/
117 :  void init( void )
118 :  {
119 :      /* I/Oポートの入出力設定 */
120 :      P1DDR = 0xff;
121 :      P2DDR = 0xff;
122 :      P3DDR = 0xff;
123 :      P4DDR = 0xff;
124 :      P5DDR = 0xff;
125 :      P6DDR = 0xf0;                    /* CPU基板上のDIP SW        */
126 :      P8DDR = 0xff;
127 :      P9DDR = 0xf7;                    /* 通信ポート               */
128 :      PADDR = 0x5f;                    /* EEP-ROM基板              */
129 :      PBDDR = 0xff;
130 :      /* ポート７は、入力専用なので入出力設定はありません    */
131 :
132 :      /* ITU0 1msごとの割り込み */
133 :      ITU0_TCR = 0x20;
134 :      ITU0_GRA = 24575;
135 :      ITU0_IER = 0x01;
136 :
137 :      /* ITUのカウントスタート */
138 :      ITU_STR = 0x01;
139 :  }
```

ポートAは、EEP-ROMを接続します。EEP-ROMの接続端子は入力設定にします。詳しくは、13ページ「1.11 EEP-ROMの接続端子を変える場合の設定」を参照してください。今回は、0x5fとなります。

ITU0 は 1ms ごとの割り込み設定を行います。

4.5.2 データ保存関連の変数

```
 36 :  /* データ保存関連 */
 37 :  int             iTimer10;          /* 取得間隔計算用           */
 38 :  int             saveIndex;         /* 保存インデックス         */
 39 :  int             saveSendIndex;     /* 送信インデックス         */
 40 :  int             saveFlag;          /* 保存フラグ               */
 41 :  char            saveData[8];       /* 一時保存エリア           */
```

37～40 行の変数は、プロジェクト「record_01」と同じです。

41 行の saveData 配列は、EEP-ROM に保存するデータを一時的に格納する変数です。

saveData[0]= ポート 7 のデータ　　saveData[1]= ディップスイッチ値　saveData[2]= 未使用
saveData[3]= 未使用　　saveData[4]= 未使用　　saveData[5]= 未使用
saveData[6]= 未使用　　saveData[7]= 未使用
のそれぞれのデータを格納します。

4.5.3 初期設定

```
 46 :  void main( void )
 47 :  {
 48 :      /* マイコン機能の初期化 */
 49 :      init();                              /* 初期化                    */
 50 :      initI2CEeprom( &PADDR, &PADR, 0x5f, 7, 5);  /* EEP-ROM初期設定  */
 51 :      init_sci1( 0x00, 79 );               /* SCI1初期化                */
 52 :      set_ccr( 0x00 );                     /* 全体割り込み許可          */
```

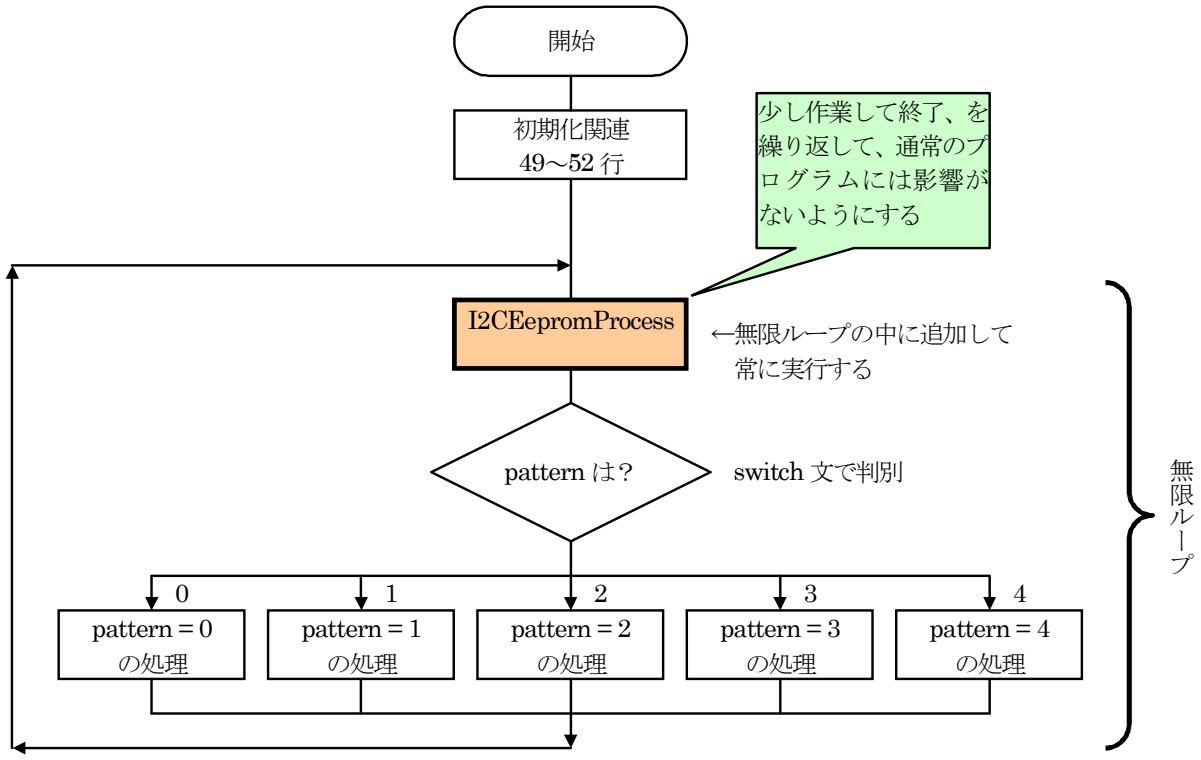
EEP-ROM 関連の初期設定を行います。この関数は、**EEP-ROM の内容をクリアするのではなく、EEP-ROM を使用するための準備をする関数です**。

EEP-ROM はポート A に接続、ポート A の入出力設定は 0x5f、EEP-ROM の SCL 端子は bit7、EEP-ROM の SDA 端子は bit5 という設定です。

4.5.4 メイン部の全体

```
 54 :      while( 1 ) {
 55 :
 56 :      I2CEepromProcess();                  /* I2C EEP-ROM保存処理       */
 57 :
 58 :      switch( pattern ) {
 59 :      case 0:
               EEP-ROMの内容クリア、保存開始、パターン１へ
 68 :          break;
 69 :
 70 :      case 1:
               10秒たったなら保存終了、パターン２へ
               割り込みプログラムで10msごとにデータを取得、EEP-ROMへ保存する準備を行う
 76 :          break;
 77 :
 78 :      case 2:
               タイトルの送信
 88 :          break;
 89 :
 90 :      case 3:
               データ送信
100 :          break;
101 :
102 :      case 4:
103 :          /* 転送終了 */
104 :          break;
105 :
106 :      default:
107 :          /* どれでもない場合は待機状態に戻す */
108 :          pattern = 0;
109 :          break;
110 :      }
111 :      }
112 :  }
```

56 行で、I2CEepromProcess 関数をループの中に置いて、常に実行されるようにしています。イメージとしては下記のようになります。



EEP-ROM への書き込み作業は、かなり時間がかかります。そのため、それだけにかかりっきりになると他の処理ができなくなり問題となることがあります。マイコンカーの場合は、センサのチェック漏れやモータ制御の遅れがおこり、脱輪してしまうこともあるかもしれません。

そこで、setPageWriteI2CEeprom 関数と I2CEepromProcess 関数を使用します。この関数は

・setPageWriteI2CEeprom 関数で書き込む準備をします（すぐに終わります）。10ms ごとに割り込み内で行います。後述します。
・I2CEepromProcess 関数で少しずつ書き込んで、1回の処理を短時間で終えるようにしています。

I2CEepromProcess 関数は、ループ内に入れて、常に実行するようにします。

4.5.5 パターン 0:データ保存の前準備

```
 59 :      case 0:
 60 :          /* EEP-ROMクリア */
 61 :          clearI2CEeprom();               /* 数秒かかる              */
 62 :          printf( "¥n" );
 63 :          printf( "data recording...¥n" );
 64 :          pattern = 1;
 65 :          saveIndex = 0;
 66 :          saveFlag = 1;                   /* データ保存開始          */
 67 :          cnt1 = 0;
 68 :          break;
```

61 行…EEP-ROM のクリアを行います。数秒かかります。
62～63 行…データを記録する旨、メッセージを出力します。
64 行…次に実行するときはパターン 1 になるよう、変数の設定をします。
65 行…保存する配列の番号用の変数を初期化します。
66 行…データの保存を開始します。10ms ごとに割り込み内でポート 7 の値と、ディップスイッチの値を保存します。
67 行…時間計測用の cnt1 変数をクリアします。

4.5.6 パターン 1:データ保存中の処理

```
 70 :      case 1:
 71 :          /* データ保存中　保存自体は割り込みの中で行う */
 72 :          if( cnt1 >= 10000 ) {
 73 :              saveFlag = 0;               /* 保存終了                */
 74 :              pattern = 2;                /* データ転送処理へ        */
 75 :          }
 76 :          break;
```

データの保存自体は、割り込み内で行っています。

72 行…10000ms たったかチェックして、たったなら保存を終了します。
74 行…パターンを 2 に移します。

EEP-ROM は 32KB(32768 バイト)あります。10ms ごとに 2 バイト分のデータを保存するので、
32768×0.01 秒／1回の保存数 2 バイト＝163.84 秒＝2 分 43 秒 84
の時間だけ、保存することができます。保存時間を変えたい場合は、72 行の 10000 を変えてください。ただし、上限は 163.84 秒(163840ms)です。

4.5.7 パターン 2:タイトル転送、準備

```
 78 :      case 2:
 79 :          /* タイトル転送、準備 */
 80 :          while( !checkI2CEeprom() );     /* 最後のデータ書き込むまで待つ*/
 81 :
 82 :          printf( "¥n" );
 83 :          printf( "record_02 Data Out¥n" );
 84 :          printf( "P7 data,dip sw data¥n" );
 85 :
 86 :          saveSendIndex = 0;
 87 :          pattern = 3;
 88 :          break;
```

80 行…最後の書き込みが終わるまで待ちます。
82～84 行…タイトルを送信します。
86 行…送信位置をクリアしています。
87 行…パターンを 3 に移します。

4.5.8 パターン 3:データ転送

```
 90 :      case 3:
 91 :          /* データ転送 */
 92 :          printf( "%02x,%02x¥n",
 93 :              (unsigned char)readI2CEeprom( saveSendIndex+0 ),
 94 :              (unsigned char)readI2CEeprom( saveSendIndex+1 ) );
 95 :          saveSendIndex += 2;
 96 :          if( saveIndex <= saveSendIndex ) {
 97 :              pattern = 4;
 98 :              cnt1 = 0;
 99 :          }
100 :          break;
```

データを転送します。
送信フォーマットは、92 行の「%02x,%02x¥n」です。まず、
「%02x」… 2 桁に満たない場合は 0 で埋めて 16 進数表記
「,」…… そのまま表示されます。
「%02x」… 2 桁に満たない場合は 0 で埋めて 16 進数表記
「¥n」…… 改行
となります。
EEP-ROM からデータを読み込む関数は、

```
readI2CEeprom( EEP-ROMから読み込みたいアドレス );
```

です。saveSendIndex が読み込むアドレスになります。

```
printf(%02x,%02x¥n",
    (unsigned char)readI2CEeprom( saveSendIndex+0 ),    ポート7のデータ読み込み
    (unsigned char)readI2CEeprom( saveSendIndex+1 ) );  ディップスイッチのデータ読み込み
```

データを読み込みながら、printf 文でデータを送信します。因みに、readI2CEeprom 関数の戻り値は、char 型（-128～127）です。出力値は 16 進数なので、unsigned char 型に型変換しています。

```
        saveSendIndex += 2;
```

でアドレスを+2 して、次に読み込むアドレスにします。
96 行で、保存数より送信しようとしているアドレスの方が大きくなったら、送信終了と見なしてパターン 4 へ移ります。

4.5.9 パターン 4:転送終了

```
102 :       case 4:
103 :           /* 転送終了 */
104 :           break;
```

終わりです。何もしません。

4.5.10 割り込みプログラム

```
141 :  /************************************************************************/
142 :  /* ITU0 割り込み処理                                                    */
143 :  /************************************************************************/
144 :  #pragma interrupt( interrupt_timer0 )
145 :  void interrupt_timer0( void )
146 :  {
147 :      ITU0_TSR &= 0xfe;                   /* フラグクリア             */
148 :      cnt1++;
149 :
150 :      /* データ保存関連 */
151 :      iTimer10++;
152 :      if( iTimer10 >= 10 ) {
153 :          iTimer10 = 0;
154 :          if( saveFlag ) {
155 :              saveData[0] = P7DR;
156 :              saveData[1] = dipsw_get();
157 :              setPageWriteI2CEeprom( saveIndex, 2, saveData );
158 :              saveIndex += 2;
159 :              if( saveIndex >= 0x8000 ) saveFlag = 0;
160 :          }
161 :      }
162 :  }
```

割り込みプログラムは、1ms ごとに実行されます。データの保存は、10ms ごとなので 151 行の iTimer 変数で 10 回数えます。10 なら 10ms たったと判断して、EEP-ROM にデータを書き込みます。手順は、
・保存したいデータを saveData 配列にセットします（155～156 行）。
・setPageWriteI2CEeprom 関数を実行します（157 行）。

setPageWriteI2CEeprom 関数は、EEP-ROM に書き込む**準備をするだけ**です。この関数の引数は、

```
setPageWriteI2CEeprom( saveIndex, 2, saveData );
                         ↑       ↑     ↑
                         1       2     3
```

1…EEP-ROM の何番地に保存するか指定します。
2…保存するデータ数を指定します。
3…保存データの格納されている配列を指定します。

実際の書き込みは、I2CEepromProcess 関数で行っています。
158 行で、保存するアドレスを+2 して、次に保存するアドレスをセットしています。
159 行で、アドレスの上限である 0x8000 を超えていないかチェック、超えていたらデータ保存を終了します。

4.5.11 int型の値を保存する場合

EEP-ROM に保存できるデータは、char 型(8bit 幅、-128～127、または 0～255)です。そのため、int 型(16bit 幅)を保存する場合は、下記のように上位 8bit と下位 8bit に分けて保存します。

```
i = int 型のデータ
saveData[0] = i >> 8;              /* 上位 8bit 保存 */
saveData[1] = i & 0xff;            /* 下位 8bit 保存 */
```

呼び出すときは、下記のようにします。

```
i = (int)((unsigned char)readI2CEeprom( アドレス )*0x100 +
                         (unsigned char)readI2CEeprom( アドレス+1 ) );
```

4.5.12 long型の値を保存する場合

EEP-ROM に long 型(32bit 幅)を保存する場合は、下記のように 4 分割して保存します。保存する変数は、lEncoderTotal 変数を例にします。

```
l = lEncoderTotal;                  /* 走行距離          */
saveData[8]  = l >> 24;
saveData[9]  = (l & 0x00ff0000) >> 16;
saveData[10] = (l & 0x0000ff00) >>  8;
saveData[11] =  l & 0x000000ff;
```

呼び出すときは、下記のようにします。

```
l  = (long)(unsigned char)readI2CEeprom( saveSendIndex+ 8 )*0x1000000;
l += (long)(unsigned char)readI2CEeprom( saveSendIndex+ 9 )*0x10000;
l += (long)(unsigned char)readI2CEeprom( saveSendIndex+10 )*0x100;
l += (long)(unsigned char)readI2CEeprom( saveSendIndex+11 );
```

ちなみに printf 文で出力するとき、この変数は long 型なので変換指定文字は「%ld」(エルとディ)を使用します。

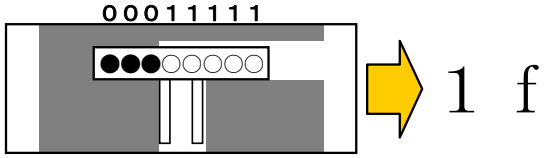
```
printf( "%ld¥n", l );
```

5. プロジェクト「record_03」 外付けEEP-ROMにデータ保存(2 進数変換)

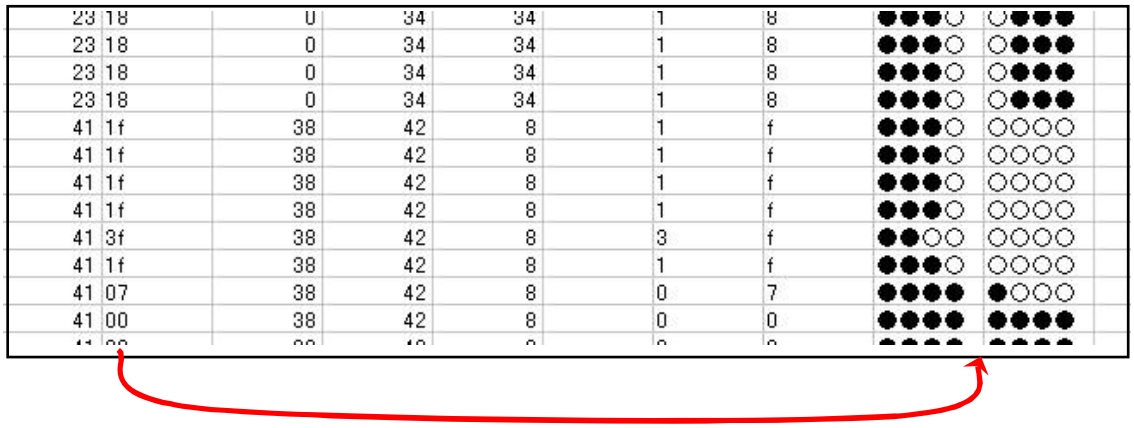
5.1 概要

このプログラムは、プロジェクト「record_02」を一部改造した内容です。

今までのプログラムは、保存したデータを出力するとき、16 進数や 10 進数でした。例えば、ポート 7 に接続されているセンサ基板の情報を記録し、16 進数でパソコンへ出力したとき、下記のようになります。



printf 文が出力する数値の形式は、8 進数、10 進数、16 進数しかありません。そのため、センサ情報は"0"か"1"かの 2 進数情報ですが、いちばん分かりやすい 16 進数で出力していました。といっても、元々 2 進数なので、やはり 2 進数で出力した方が分かりやすいです。プロジェクト「record_01」では、エクセルを使った 16 進数→2 進数変換方法を紹介しました。



23	18	0	34	34	1	8	●●●○ ○●●●
23	18	0	34	34	1	8	●●●○ ○●●●
23	18	0	34	34	1	8	●●●○ ○●●●
23	18	0	34	34	1	8	●●●○ ○●●●
41	1f	38	42	8	1	f	●●●○ ○○○○
41	1f	38	42	8	1	f	●●●○ ○○○○
41	1f	38	42	8	1	f	●●●○ ○○○○
41	1f	38	42	8	1	f	●●●○ ○○○○
41	3f	38	42	8	3	f	●●○○ ○○○○
41	1f	38	42	8	1	f	●●●○ ○○○○
41	07	38	42	8	0	7	●●●● ●○○○
41	00	38	42	8	0	0	●●●● ●●●●

エクセルの関数を使って変換

この方法なら、センサの状態が分かりますが、転送するたびにエクセルの関数を使って変換するのはちょっと大変です。

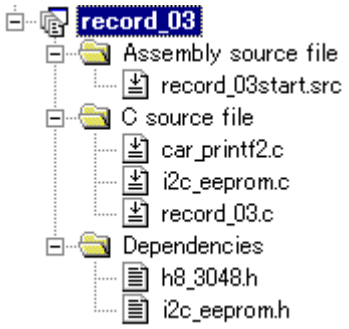
そこで、マイコン側のプログラムで、16 進数から 2 進数に変換する関数を作り、2 進数で直接出力します。そうすれば、いちいち変換する必要が無く、解析が楽になります。

本章では、変換プログラムと出力方法を紹介します。

5.2 接続

「**4.2 接続**」と同じです。

5.3 プロジェクトの構成



・record_03start.src
・record_03.c
・car_printf2.c
・**i2c_eeprom.c**
の 4 ファイルあります。
h8_3048.h は record_03.c、car_printf2.c、i2c_eeprom.c でインクルードされているファイルです。
i2c_eeprom.h は record_03.c、i2c_eeprom.c でインクルードされているファイルです。

5.4 プログラム

```
 1 :  /************************************************************************/
 2 :  /* 外付けEEP-ROMデータ保存演習プログラム「record_03.c」                  */
 3 :  /*                              2006.04 ジャパンマイコンカーラリー実行委員会    */
 4 :  /************************************************************************/
 5 :  /*
 6 :  本プログラムは外付けEEP-ROM(24C256 32KB)にポート７のデータ、
 7 :  CPUボード上のディップスイッチの値を10msごとに保存します。
 8 :  その後、保存したデータをパソコンへ転送します。
 9 :  */
10 :
11 :  /*====================================*/
12 :  /* インクルード                       */
13 :  /*====================================*/
14 :  #include    <no_float.h>              /* stdioの簡略化 最初に置く*/
15 :  #include    <stdio.h>
16 :  #include    <machine.h>
17 :  #include    "h8_3048.h"
18 :  #include    "i2c_eeprom.h"            /* EEP-ROM追加(データ記録)  */
19 :
20 :  /*====================================*/
21 :  /* シンボル定義                       */
22 :  /*====================================*/
23 :
24 :  /*====================================*/
25 :  /* プロトタイプ宣言                   */
26 :  /*====================================*/
27 :  void init( void );
28 :  unsigned char dipsw_get( void );
29 :  void convertHexToBin( unsigned char hex, char *s );
30 :
31 :  /*====================================*/
32 :  /* グローバル変数の宣言               */
33 :  /*====================================*/
34 :  unsigned long   cnt1;                 /* main内で使用             */
35 :  int             pattern;              /* パターン番号             */
36 :
37 :  /* データ保存関連 */
38 :  int             iTimer10;             /* 取得間隔計算用           */
39 :  int             saveIndex;            /* 保存インデックス         */
40 :  int             saveSendIndex;        /* 送信インデックス         */
41 :  int             saveFlag;             /* 保存フラグ               */
42 :  char            saveData[8];          /* 一時保存エリア           */
43 :
44 :  /************************************************************************/
45 :  /* メインプログラム                                                     */
46 :  /************************************************************************/
47 :  void main( void )
48 :  {
49 :      char s[8];
50 :
51 :      /* マイコン機能の初期化 */
52 :      init();                           /* 初期化                   */
53 :      initI2CEeprom( &PADDR, &PADR, 0x5f, 7, 5);  /* EEP-ROM初期設定  */
54 :      init_sci1( 0x00, 79 );            /* SCI1初期化               */
55 :      set_ccr( 0x00 );                  /* 全体割り込み許可         */
56 :
57 :      while( 1 ) {
58 :
59 :      I2CEepromProcess();               /* I2C EEP-ROM保存処理      */
60 :
61 :      switch( pattern ) {
```

convertHexToBin 関数を新たに作りました。プロトタイプ宣言を忘れずに行います。

2 進数変換後のデータを保存する配列です。

中略

```
 93 :      case 3:
 94 :          /* データ転送 */
 95 :          convertHexToBin( readI2CEeprom( saveSendIndex+0 ), s );
 96 :          printf( "=¥"%8s¥",%02x¥n", s,
 97 :                      (unsigned char)readI2CEeprom( saveSendIndex+1 ) );
 98 :          saveSendIndex += 2;
 99 :          if( saveIndex <= saveSendIndex ) {
100 :              pattern = 4;
101 :              cnt1 = 0;
102 :          }
103 :          break;
```

2 進数に変換して printf 文で出力します。

中略

```
181 : /*************************************************************************/
182 : /* １６進数→２進数変換                                                  */
183 : /* 引数　 １６進数データ、変換後のデータ格納アドレス                     */
184 : /* 戻り値 なし                                                           */
185 : /*************************************************************************/
186 : void convertHexToBin( unsigned char hex, char *s )
187 : {
188 :     int     i;
189 :
190 :     for( i=0; i<8; i++ ) {
191 :         if( hex & 0x80 ) {
192 :             *s++ = '1';                 /* "1"のときの変換データ     */
193 :         } else {
194 :             *s++ = '0';                 /* "0"のときの変換データ     */
195 :         }
196 :         hex <<= 1;
197 :     }
198 : }
199 :
200 : /*************************************************************************/
201 : /* end of file                                                           */
202 : /*************************************************************************/
```

unsigned char 型の変数(0～255 または 0x00～0xff)を 2 進数に変換する関数です。

5.5 プログラムの解説

5.5.1 変数の追加

```
 44 : /*************************************************************************/
 45 : /* メインプログラム                                                      */
 46 : /*************************************************************************/
 47 : void main( void )
 48 : {
 49 :     char s[8];
```

char 型の s という配列を追加します。8 バイト分です。convertHexToBin 関数で使用します。配列 s には、変換後の 2 進数データが文字列で格納されます。詳しくは、convertHexToBin 関数の説明を参照してください。

5.5.2 2 進数変換を行うconvertHexToBin関数

```
181 :  /*************************************************************************/
182 :  /* １６進数→２進数変換                                                  */
183 :  /* 引数    １６進数データ、変換後のデータ格納アドレス                    */
184 :  /* 戻り値 なし                                                           */
185 :  /*************************************************************************/
186 :  void convertHexToBin( unsigned char hex, char *s )
187 :  {
188 :      int     i;
189 :
190 :      for( i=0; i<8; i++ ) {
191 :          if( hex & 0x80 ) {
192 :              *s++ = '1';                 /* "1"のときの変換データ     */
193 :          } else {
194 :              *s++ = '0';                 /* "0"のときの変換データ     */
195 :          }
196 :          hex <<= 1;
197 :      }
198 :  }
```

convertHexToBin という関数を作りました。使い方は下記のようになります。

```
            convertHexToBin( 0x55, s );
                             ↑   ↑
                             1   2
```

1…2 進数に変換する元データです。
2…2進数に変換したデータを保存する配列を指定します。ここで指定する配列は、char型で8バイト以上の大きさが必要です。

上記を実行した場合、配列 s には下記のように変換されたデータが、**'1'または'0'の文字として保存されます**。

s[0]	s[1]	s[2]	s[3]	s[4]	s[5]	s[6]	s[7]
'0'	'1'	'0'	'1'	'0'	'1'	'0'	'1'

printf 文で配列 s に格納されている文字列 8 文字を出力すれば、変換された 2 進数データが出力されることになります。

5.5.3 printf出力

```
 95 :          convertHexToBin( readI2CEeprom( saveSendIndex+0 ), s );
 96 :          printf( "=¥"%8s¥",%02x¥n", s,
 97 :                      (unsigned char)readI2CEeprom( saveSendIndex+1 ) );
```

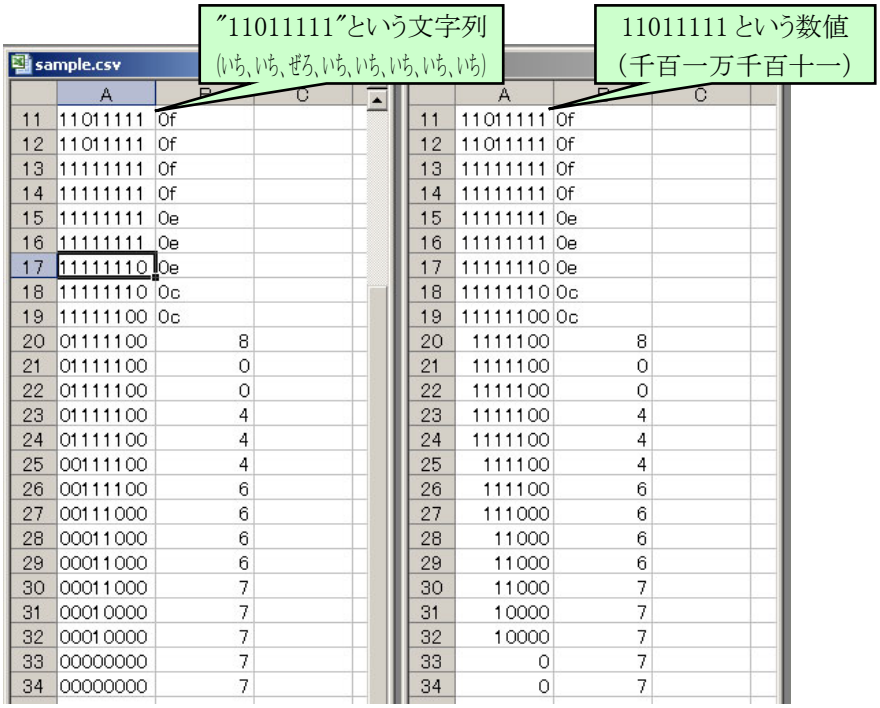
95 行…EEP-ROM の(saveSendIndex+0)番地に保存されたデータを 2 進数に変換します。
96行…2進データは、文字列8文字なので、printf文では、「%8s」で配列sに格納している文字列を出力します。
¥"の 2 文字は、"の 1 文字として出力されます。

5.6 データ例

下記に、出力例を示します。

data recording...	="11111110",0c
	="11111100",0c
record_03 Data Out	="01111100",08
P7 data,dip sw data	="01111100",00
="01010101",0f	="01111100",00
="01010101",0f	="01111100",04
="01010111",0f	="01111100",04
="01010111",0f	="00111100",04
="01011111",0f	="00111100",06
="01011111",0f	="00111000",06
="11011111",0f	="00011000",06
="11011111",0f	="00011000",06
="11111111",0f	="00011000",07
="11111111",0f	="00010000",07
="11111111",0e	="00010000",07
="11111111",0e	="00000000",07
="11111110",0e	="00000000",07

2 進数は、『="00000000"』の形式で出力しています。保存するファイル名の拡張子は、「**csv**」として保存します。例えば、「sample.csv」とします。このファイルをエクセルの入っているパソコンでダブルクリックすると、自動でエクセルが立ち上がり、下図(左)のようになります。A 列は文字として扱います。もし、2 進数を『00000000』の形式で出力した場合はどうなるのでしょう。同じく拡張子 csv で保存、ダブルクリックするとエクセルが立ち上がりますが、下図(右)のように A 列は数値として扱い、表示形式が崩れてしまいます。ちなみに、B 列は 16 進数ですがエクセルの 20 行目は数値の 8 として扱うため、表示形式がくずれています。16 進数も同様に『="xx"』の形式で出力すると、表示形式が崩れません。



6. プロジェクト「kit07rec_01」 走行データを内蔵RAMにデータ保存

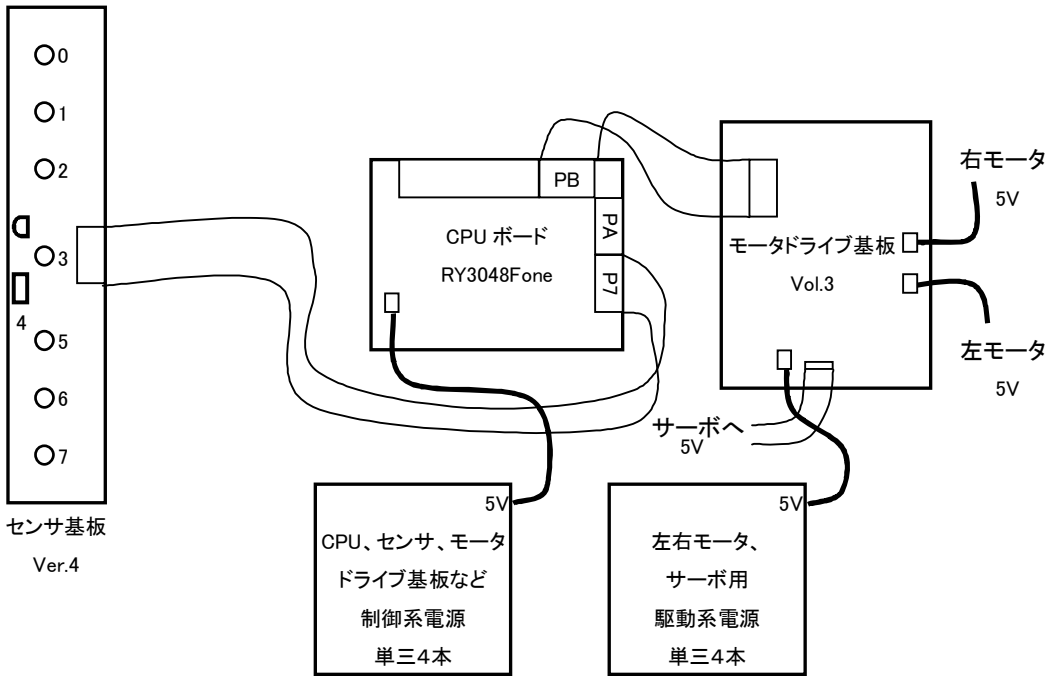
6.1 概要

マイコンカーの走行中のデータを、H8 マイコンの内蔵 RAM に保存します。保存する内容は、
・パターン番号
・センサの値
の2つです。これらのデータが走行開始から 10ms ごとに、1000 回分保存されます。10 秒間です。10 秒たっても走行はそのまま行いますが、データ保存は行いません。

保存した走行データをパソコンに送り、マイコンカーがどう走ったかパソコン上で解析します。この情報を基に、プログラムのデバッグに役立てます。

6.2 マイコンカーの構成

マイコンカーキット Ver.4 の構成です。LM350 追加セットで電池 8 本を直列に繋いでいる構成でも OK です。
・CPU ボードのポート 7 と、マイコンカーキット Ver.4 のセンサ基板 Ver.4 を接続します。
・CPU ボードのポート B と、マイコンカーキット Ver.4 のモータドライブ基板 Vol.3 を接続します。



6.3 プロジェクトの構成

・kit07rec_01start.src
・kit07rec_01.c
・car_printf2.c
の 3 ファイルあります。
h8_3048.h は kit07rec_01.c、car_printf2.c でインクルードされているファイルです。

6.4 プログラム

```
 1 :  /*************************************************************************/
 2 :  /* 走行データ記録マイコンカートレース基本プログラム「kit07rec_01.c」    */
 3 :  /*                              2007.05 ジャパンマイコンカーラリー実行委員会  */
 4 :  /*************************************************************************/
 5 :  /*
 6 :  本プログラムはkit07.cをベースに走行データを記録するプログラムです。
 7 :
 8 :  kit07rec_01.cは、H8/3048F-ONEの内蔵RAM(2.5KB程度)にマイコンカーの走行
 9 :  データを保存します。その後、保存したデータをパソコンへ転送します。
10 :  パソコンでデータを解析すれば、なぜ脱輪したかなど問題を「見える化」
11 :  することが出来、プログラムのデバッグに役立てることが出来ます。
12 :  */
13 :
14 :  /*====================================*/
15 :  /* インクルード                       */
16 :  /*====================================*/
17 :  #include    <no_float.h>              /* stdioの簡略化 最初に置く*/
18 :  #include    <stdio.h>
19 :  #include    <machine.h>
20 :  #include    "h8_3048.h"
21 :
22 :  /*====================================*/
23 :  /* シンボル定義                       */
24 :  /*====================================*/
25 :
26 :  /* 定数設定 */
27 :  #define         TIMER_CYCLE     3071    /* タイマのサイクル 1ms      */
28 :                                          /* φ/8で使用する場合、      */
29 :                                          /* φ/8 = 325.5[ns]          */
30 :                                          /* ∴TIMER_CYCLE =           */
31 :                                          /*       1[ms] / 325.5[ns]   */
32 :                                          /*                 = 3072    */
33 :  #define         PWM_CYCLE       49151   /* PWMのサイクル 16ms        */
34 :                                          /* ∴PWM_CYCLE =             */
35 :                                          /*       16[ms] / 325.5[ns]  */
36 :                                          /*                 = 49152   */
37 :  #define         SERVO_CENTER    5000    /* サーボのセンタ値          */
38 :  #define         HANDLE_STEP     26      /* 1° 分の値                 */
39 :
40 :  /* マスク値設定 ×：マスクあり(無効)  ○：マスク無し(有効) */
41 :  #define         MASK2_2         0x66    /* ××○○××○×          */
42 :  #define         MASK2_0         0x60    /* ××○○×××××          */
43 :  #define         MASK0_2         0x06    /* ××××××○×          */
44 :  #define         MASK3_3         0xe7    /* ○○○××○○○          */
45 :  #define         MASK0_3         0x07    /* ××××××○○          */
46 :  #define         MASK3_0         0xe0    /* ○○○×××××          */
47 :  #define         MASK4_0         0xf0    /* ○○○○×××××          */
48 :  #define         MASK0_4         0x0f    /* ××××○○○○          */
49 :  #define         MASK4_4         0xff    /* ○○○○○○○○          */
50 :
51 :  #define         SAVE_SIZE       1000    /* データ保存数              */
52 :
53 :  /*====================================*/
54 :  /* プロトタイプ宣言                   */
55 :  /*====================================*/
56 :  void init( void );
57 :  void timer( unsigned long timer_set );
58 :  int check_crossline( void );
59 :  int check_rightline( void );
60 :  int check_leftline( void );
61 :  unsigned char sensor_inp( unsigned char mask );
62 :  unsigned char dipsw_get( void );
63 :  unsigned char pushsw_get( void );
```

printf 文を使用するので、stdio.h をインクルードします。

データを保存する数です。
保存時間は、
保存間隔 10ms×1000＝10 秒
となります。

```
 64 :  unsigned char startbar_get( void );
 65 :  void led_out( unsigned char led );
 66 :  void speed( int accele_l, int accele_r );
 67 :  void handle( int angle );
 68 :  char unsigned bit_change( char unsigned in );
 69 :  void convertHexToBin( unsigned char hex, char *s );
 70 :
 71 :  /*====================================*/
 72 :  /* グローバル変数の宣言               */
 73 :  /*====================================*/
 74 :  unsigned long   cnt0;                  /* timer関数用              */
 75 :  unsigned long   cnt1;                  /* main内で使用             */
 76 :  int             pattern;               /* パターン番号             */
 77 :
 78 :  /* データ保存関連 saveDataのサイズは最大で2.5KB程度としてください */
 79 :  int             iTimer10;              /* 取得間隔計算用           */
 80 :  unsigned char   saveData[SAVE_SIZE][2]; /* 保存エリア               */
 81 :  int             saveIndex;             /* 保存インデックス         */
 82 :  int             saveSendIndex;         /* 送信インデックス         */
 83 :  int             saveFlag;              /* 保存フラグ               */
 84 :
 85 :  /*************************************************************************/
 86 :  /* メインプログラム                                                      */
 87 :  /*************************************************************************/
 88 :  void main( void )
 89 :  {
 90 :      int     i;
 91 :      char    s[8];
 92 :
 93 :      /* マイコン機能の初期化 */
 94 :      init();                            /* 初期化                   */
 95 :      init_sci1( 0x00, 79 );             /* SCI1初期化               */
 96 :      set_ccr( 0x00 );                   /* 全体割り込み許可         */
 97 :
 98 :      /* マイコンカーの状態初期化 */
 99 :      handle( 0 );
100 :      speed( 0, 0 );
101 :
102 :      while( 1 ) {
103 :
104 :      P4DR = ~pattern;                   /* デバッグ用にパターン出力 */
105 :
106 :      switch( pattern ) {
```

データ保存関係の変数です。record_01 と同じです。

printf 文で通信を使用しますので、init_sci1 で通信を初期化します。

デバッグ用としてポート 4 にパターンを出力します。LED をポート 4 に繋ぎます。

中略

```
148 :      case 1:
149 :          /* スタートバーが開いたかチェック */
150 :          if( !startbar_get() ) {
151 :              /* スタート！！ */
152 :              led_out( 0x0 );
153 :              pattern = 11;
154 :              saveIndex = 0;
155 :              saveFlag = 1;              /* データ保存開始           */
156 :              cnt1 = 0;
157 :              break;
158 :          }
159 :          if( cnt1 < 50 ) {              /* LED点滅処理              */
160 :              led_out( 0x1 );
161 :          } else if( cnt1 < 100 ) {
162 :              led_out( 0x2 );
163 :          } else {
164 :              cnt1 = 0;
165 :          }
166 :          break;
167 :
168 :      case 11:
169 :          /* 通常トレース */
170 :          if( pushsw_get() ) {
171 :              pattern = 71;              /* データ転送処理へ         */
172 :              break;
173 :          }
174 :
175 :          if( check_crossline() ) {      /* クロスラインチェック     */
176 :              pattern = 21;
177 :              break;
178 :          }
179 :          if( check_rightline() ) {      /* 右ハーフラインチェック   */
180 :              pattern = 51;
181 :              break;
182 :          }
183 :          if( check_leftline() ) {       /* 左ハーフラインチェック   */
184 :              pattern = 61;
185 :              break;
186 :          }
187 :          switch( sensor_inp(MASK3_3) ) {
188 :              case 0x00:
189 :                  /* センター→まっすぐ */
190 :                  handle( 0 );
191 :                  speed( 100 ,100 );
192 :                  break;
```

saveIndex は保存する配列の位置を指定します。最初は 0 です。saveFlag=1 で保存を開始します。

パターン 11 のときに、スイッチを押すと、パターン 71 へ移ります。パターン 71 は、データを転送する部分です。

中略

```
507 :       case 71:
508 :           /* 停止 */
509 :           handle( 0 );
510 :           speed( 0, 0 );
511 :           saveFlag = 0;
512 :           saveSendIndex = 0;
513 :           pattern = 72;
514 :           cnt1 = 0;
515 :           break;
516 :
517 :       case 72:
518 :           /* 1s待ち */
519 :           if( cnt1 > 1000 ) {
520 :               pattern = 73;
521 :               cnt1 = 0;
522 :           }
523 :           break;
524 :
525 :       case 73:
526 :           /* スイッチが離されたかチェック */
527 :           if( !pushsw_get() ) {
528 :               pattern = 74;
529 :               cnt1 = 0;
530 :           }
531 :           break;
532 :
533 :       case 74:
534 :           /* スイッチが押されたかチェック */
535 :           led_out( (cnt1/500) % 2 + 1 );
536 :           if( pushsw_get() ) {
537 :               pattern = 75;
538 :               cnt1 = 0;
539 :           }
540 :           break;
541 :
542 :       case 75:
543 :           /* タイトル転送、準備 */
544 :           printf( "¥n" );
545 :           printf( "kit07rec_01 Data Out¥n" );
546 :           printf( "Pattern, Sensor¥n" );
547 :           pattern = 76;
548 :           break;
549 :
550 :       case 76:
551 :           /* データ転送 */
552 :           led_out( (saveSendIndex/32) % 2 + 1 ); /* LED点滅処理      */
553 :
554 :           convertHexToBin( saveData[saveSendIndex][1], s );
555 :           printf( "%d,=¥"%8s¥"¥n",
556 :               saveData[saveSendIndex][0],     /* パターン          */
557 :               s                               );  /* センサ値(2進数)  */
558 :
559 :           /* 終わりのチェック */
560 :           saveSendIndex++;
561 :           if( saveIndex <= saveSendIndex ) {
562 :               pattern = 77;
563 :               cnt1 = 0;
564 :           }
565 :           break;
566 :
567 :       case 77:
568 :           /* 転送終了 */
569 :           led_out( 0x3 );
570 :           break;
571 :
572 :       default:
573 :           /* どれでもない場合は待機状態に戻す */
574 :           pattern = 0;
575 :           break;
576 :       }
577 :       }
578 :   }
579 :
580 :   /*****************************************************************/
581 :   /* H8/3048F-ONE 内蔵周辺機能  初期化                             */
582 :   /*****************************************************************/
583 :   void init( void )
584 :   {
585 :       /* I/Oポートの入出力設定 */
586 :       P1DDR = 0xff;
587 :       P2DDR = 0xff;
588 :       P3DDR = 0xff;
589 :       P4DDR = 0xff;
590 :       P5DDR = 0xff;
591 :       P6DDR = 0xf0;                   /* CPU基板上のDIP SW      */
592 :       P8DDR = 0xff;
593 :       P9DDR = 0xf7;                   /* 通信ポート             */
594 :       PADDR = 0xff;
```

パターン 71 以降は、保存したデータを転送する部分です。
まず、マイコンカーを停止させます。
saveFlag=0 でデータ保存を終了します。saveSendIndex は転送する配列の位置を指定する変数です。0から順番に転送するので、最初にクリアします。パターン 72 へ移ります。

単純に1秒待ちます。1 秒後、パターン 73 へ移ります。

スイッチが離されていれば、パターン 74 へ移ります。

スイッチが押されれば、パターン 75 へ移ります。
押されるまでの間、LED を点滅させ、転送待機状態だということを外部に知らせます。

タイトルを転送します。すぐに、パターン 76 へ移ります。

センサ値を 2 進数に変換します。

データがある限り、転送し続けます。
保存数を超えてデータを転送しようとするとパターン 77 へ移り、転送を終了します。

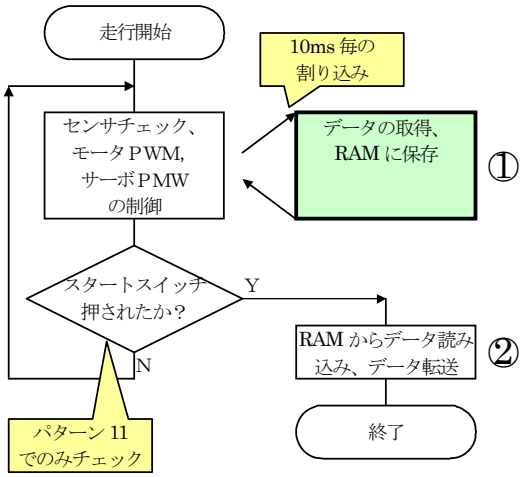
何もしません。電源が切れるのを待ちます。

```
595 :      PBDR  = 0xc0;
596 :      PBDDR = 0xfe;                          /* モータドライブ基板Vol.3  */
597 :      /* ※センサ基板のP7は、入力専用なので入出力設定はありません     */
598 :
599 :      /* ITU0 1msごとの割り込み */
600 :      ITU0_TCR = 0x23;
601 :      ITU0_GRA = TIMER_CYCLE;
602 :      ITU0_IER = 0x01;
603 :
604 :      /* ITU3,4 リセット同期PWMモード 左右モータ、サーボ用 */
605 :      ITU3_TCR = 0x23;
606 :      ITU_FCR  = 0x3e;
607 :      ITU3_GRA = PWM_CYCLE;                  /* 周期の設定               */
608 :      ITU3_GRB = ITU3_BRB = 0;               /* 左モータのPWM設定        */
609 :      ITU4_GRA = ITU4_BRA = 0;               /* 右モータのPWM設定        */
610 :      ITU4_GRB = ITU4_BRB = SERVO_CENTER;    /* サーボのPWM設定          */
611 :      ITU_TOER = 0x38;
612 :
613 :      /* ITUのカウントスタート */
614 :      ITU_STR = 0x09;
615 :  }
616 :
617 :  /*****************************************************************/
618 :  /* ITU0 割り込み処理                                             */
619 :  /*****************************************************************/
620 :  #pragma interrupt( interrupt_timer0 )
621 :  void interrupt_timer0( void )
622 :  {
623 :      ITU0_TSR &= 0xfe;                      /* フラグクリア             */
624 :      cnt0++;
625 :      cnt1++;
626 :
627 :      /* データ保存関連 */
628 :      iTimer10++;
629 :      if( iTimer10 >= 10 ) {
630 :          iTimer10 = 0;
631 :          if( saveFlag ) {
632 :              saveData[saveIndex][0] = pattern;          /* パターン */
633 :              saveData[saveIndex][1] = sensor_inp(0xff); /* センサ   */
634 :              saveIndex++;
635 :              if( saveIndex >= SAVE_SIZE ) saveFlag = 0;
636 :          }
637 :      }
638 :  }
```

10ms ごとに配列に保存します。1回にパターンとセンサ値の2バイトを保存します。
配列が一杯になると、saveFlag を 0 にして保存を強制的に終了します。

以下、略

6.5 プログラムの概要



マイコンカーは、
・センサ(コースセンサ、エンコーダなど)を見ながら
・駆動モータの制御
・サーボ(ステアリングモータ)の制御
を行っています。**これらの制御に影響が無いようにデータ保存を行わなければいけません。**

データの取得、保存は割り込み内で行います。できるだけ時間をかけないようにします(①部分)。

走行終了後、マイコンカーを取り上げてスタートスイッチを押します。パソコンと RS232C ケーブルを接続してデータを転送し、取得したデータがパソコンに取り込まれます(②部分)。

ポイントは、
・走行中、スイッチを押したときに止めることができるのはパターン 11 のみです。パターン 11 以外のときにマイコンカーを取り上げた場合、手でセンサの反応を上手く切り替え、パターン 11 にしてからスイッチを押します。
・電源を切ると保存したデータが消えてしまうので、電源を入れたままで転送を行います。
・保存するデータは、パターン番号とセンサ値です。自由に変更できます。

6.6 プログラムの解説

6.6.1 データ保存エリア

saveData 配列が、データを保存するエリアです。2 次元配列になっています。SAVE_SIZE は 1000 とします。

	A=0	A=1
saveData[0][A] →	10ms 後のパターン	10ms 後のセンサ値
saveData[1][A] →	20ms 後のパターン	20ms 後のセンサ値
saveData[2][A] →	30ms 後のパターン	30ms 後のセンサ値
saveData[3][A] →	40ms 後のパターン	40ms 後のセンサ値
saveData[4][A] →	50ms 後のパターン	50ms 後のセンサ値
…		
saveData[998][A] →	9990ms 後のパターン	9990ms 後のセンサ値
saveData[999][A] →	10000ms 後のパターン	10000ms 後のセンサ値

1000 行

2 列

2 列 1000 行、1マスが 1 バイトなので、saveData 配列は合計 2000 バイトのサイズとなります。
H8 の RAM 4KB の内、2000 バイトをデータ保存エリアとして使用します。
保存方法やデータ転送方法などは、「record_01」と同じです。

6.6.2 送信内容

```
550 :       case 76:
551 :           /* データ転送 */
552 :           led_out( (saveSendIndex/32) % 2 + 1 ); /* LED 点滅処理       */
553 :
554 :           convertHexToBin( saveData[saveSendIndex][1], s );
555 :           printf( "%d,=¥"%8s¥"¥n",
556 :               saveData[saveSendIndex][0],      /* パターン           */
557 :               s                              );   /* センサ値(2 進数)  */
558 :
559 :           /* 終わりのチェック */
560 :           saveSendIndex++;
561 :           if( saveIndex <= saveSendIndex ) {
562 :               pattern = 77;
563 :               cnt1 = 0;
564 :           }
565 :           break;
```

送信内容は、554～557 行の内容です。

```
printf( "%d,=¥"%8s¥"¥n",
                saveData[saveSendIndex][0],      /* パターン           */
                s                              );   /* センサ値(2 進数)  */
```

パターンは 10 進数、センサ値は 2 進数で出力します。10 進数は「%d」、2 進数は printf 文では直接出力できないので、convertHexToBin 関数でいったん 16 進数を 2 進数に変換します。その文字列を出力するので、「%8s」とします。

6.7 プログラムの調整

「kit07rec_01.c」の下記の内容を、自分のマイコンカーに合わせて調整します。

```
 37 :  #define          SERVO_CENTER     5000     /* サーボのセンタ値          */
中略
309 :                handle( -38 );                左クランクを曲がるときのハンドル角度
中略
318 :                handle( 38 );                 右クランクを曲がるときのハンドル角度
```

走らせた後は電源を切らずに持ち上げ、パターン 11（通常走行）の状態であることを確認して、スイッチを 1 回押します。マイコンカーはデータ停止して転送モードになり、LED がゆっくり点滅します。これは、下記のようにスイッチが押されたらデータ転送処理へ移るプログラム部分が、パターン 11 にしか無いためです。

```
168 :      case 11:
169 :          /* 通常トレース */
170 :          if( pushsw_get() ) {
171 :              pattern = 71;                 /* データ転送処理へ          */
172 :              break;
173 :          }
```

パターン 11 以外を実行中の場合、スイッチを押してもデータ転送モードになりません。もし、パターン 12 や 13、その他でもスイッチを押したときに停止してデータ転送処理へ移るようにしたい場合は、それぞれのパターンの先頭部分に、上記4行を追加します。例として、パターン 12 に追加した例を下記に示します。

```
249 :      case 12:
250 :          /* 右へ大曲げの終わりのチェック */
***:          if( pushsw_get() ) {
***:              pattern = 71;                 /* データ転送処理へ          */   } 追加
***:              break;
***:          }
251 :          if( check_crossline() ) {         /* 大曲げ中もクロスラインチェック */
252 :              pattern = 21;
253 :              break;
254 :          }
```

他にも、調整する部分は調整してください。

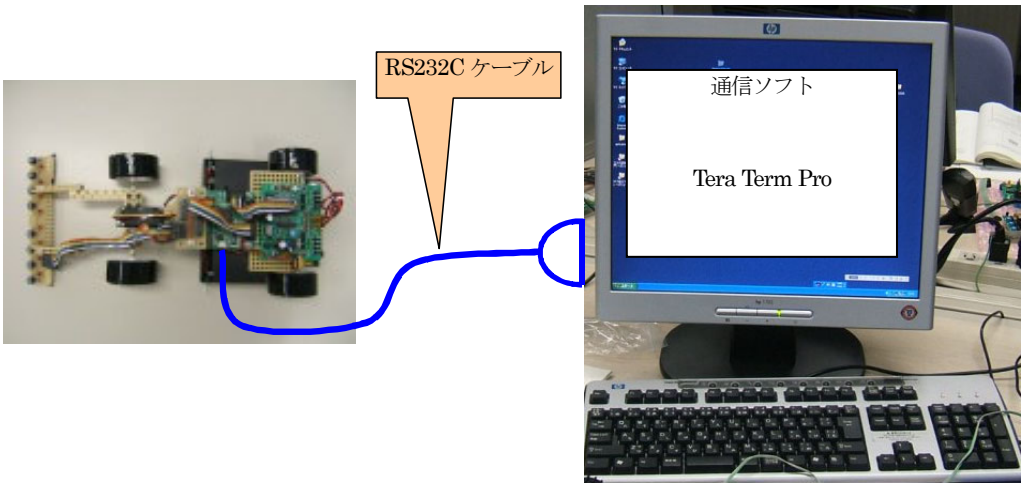
調整ができたら、プロジェクト「kit07rec_01」をビルドして、「kit07rec_01.mot」ファイルを CPU ボードに書き込みます。

6.8 走行からデータ転送までの流れ

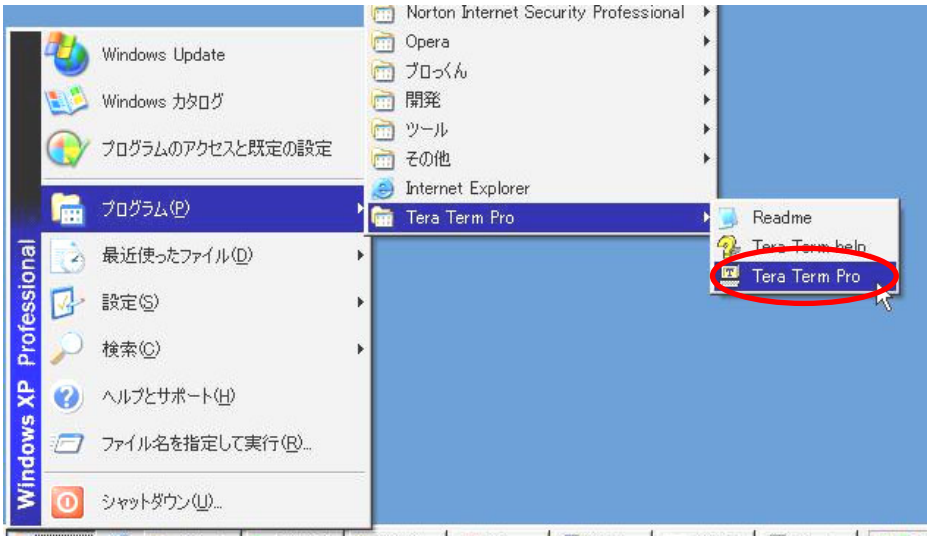
走行データを取りたい部分のコースを普通に走らせます。走行データが保存できるのは、スタートしてから 10 秒です。

走らせた後、**電源を切らずに持ち上げ、モータドライブ基板のスイッチを1回押します。マイコンカーは停止して転送モードになり、LED がゆっくり点滅します。**

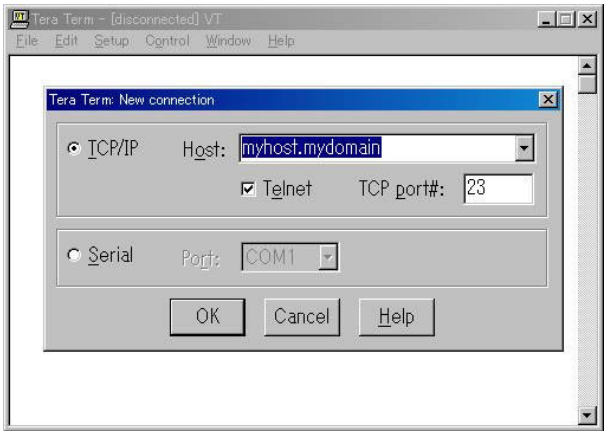
もし、スイッチを押しても転送モードにならない場合、マイコンカーが動作しているパターンに、スイッチを押したことを検出する部分が無いためです。「**6.7 プログラムの調整**」を参照してプログラムを追加するか、パターン 11 の状態でスイッチを押してください。



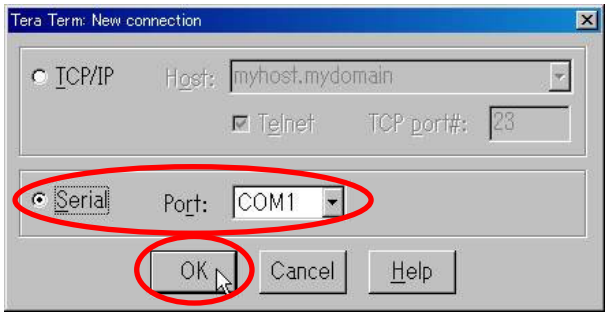
1. マイコンカーが止まり LED がゆっくり点滅している状態で、RS232C ケーブルをマイコンカーに接続します。



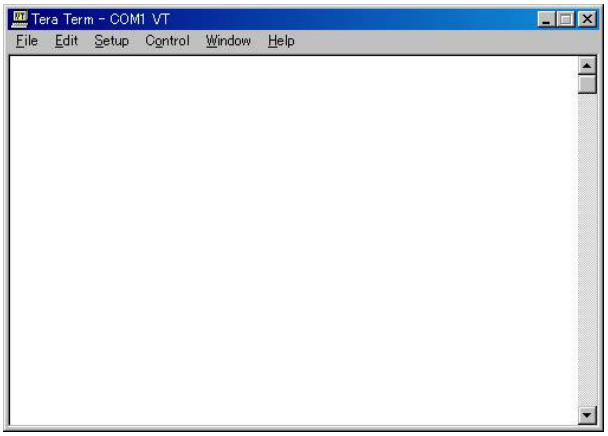
2.「スタート→すべてのプログラム(またはプログラム)→Tera Term Pro→Tera Term Pro」で Tera Term Pro を立ち上げます。



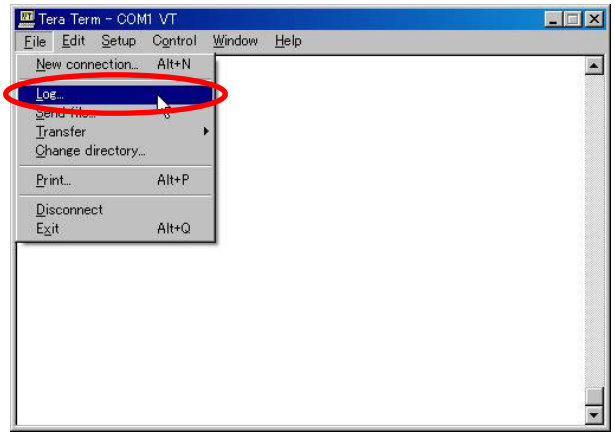
3. 最初に接続先を確認する画面が出てきます。



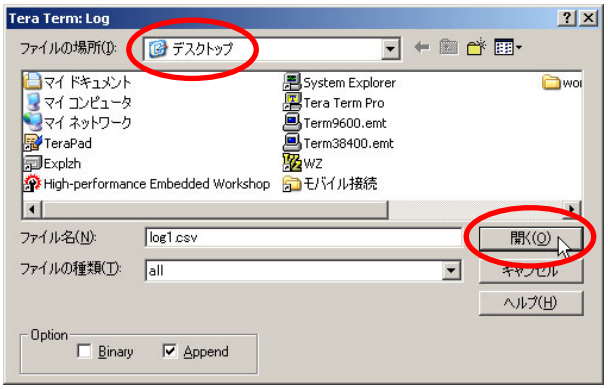
4.「Serial」を選んで、各自のパソコンに合わせてポート番号を選びます。選択後、OK をクリック、次へ進みます。



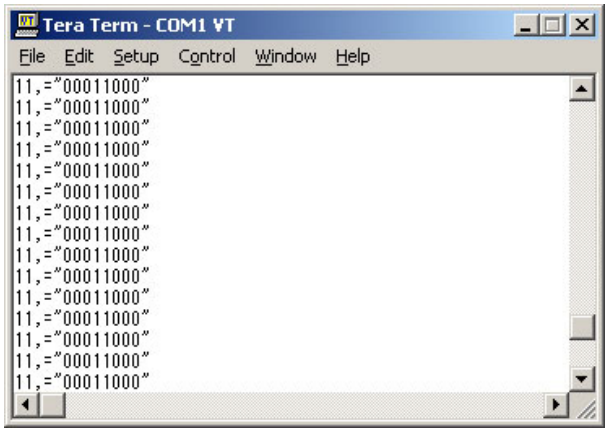
5. 立ち上がりました。



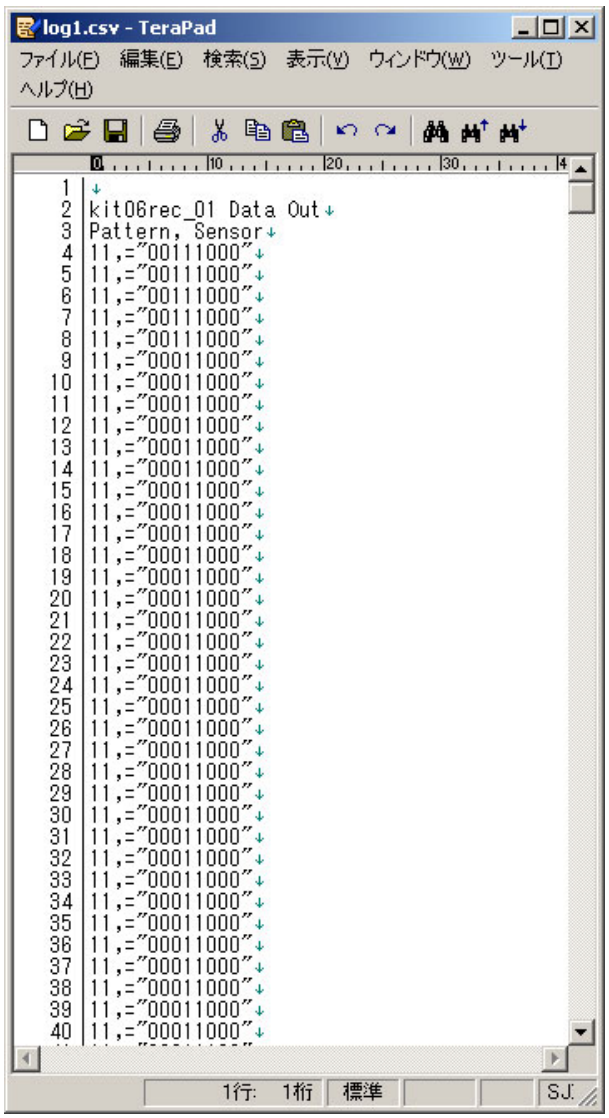
6. データをファイルに保存します。「File→Log」を選択します。



7.「ファイルの場所」に保存するフォルダを選択します。「ファイル名」には、保存するファイルの名前を入力します。ここでは「log1.csv」と入力します。開く をクリックします。



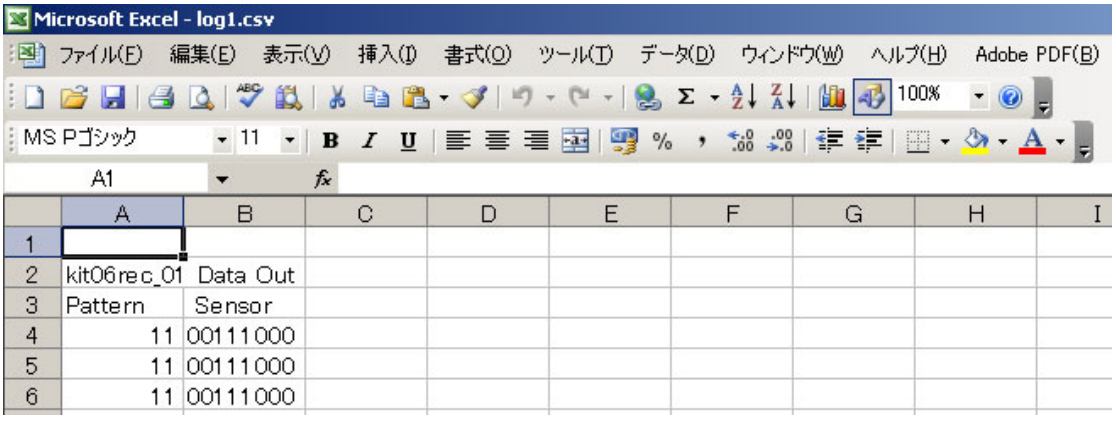
8. モータドライブ基板のスイッチを押すと、データが転送されます。転送が終わったら、TeraTermPo を終了して、マイコンカーの電源を切ってください。



9. log1.csv をエディタで開いてみました。このデータをエクセルに取り込んでみましょう。

6.9 エクセルへの取り込み方

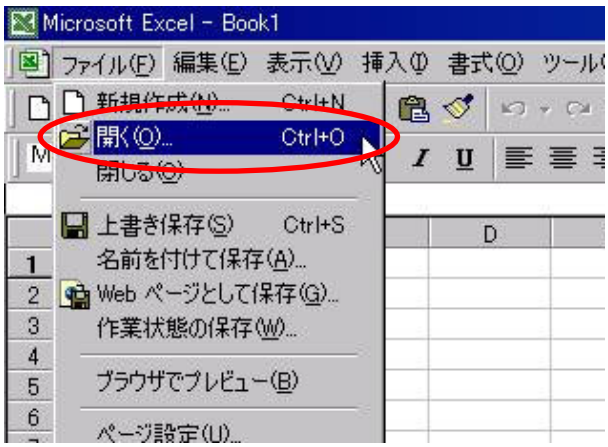
■ファイルをダブルクリックして開く



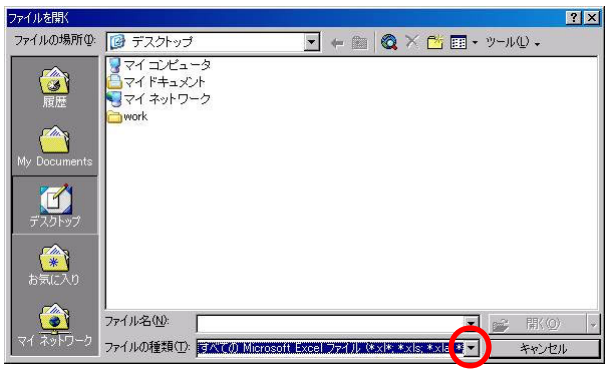
エクセルが入っていれば、「log1.csv」ファイルをダブルクリックするだけで、自動で立ち上がります。

■エクセルから手動で開く場合の操作方法

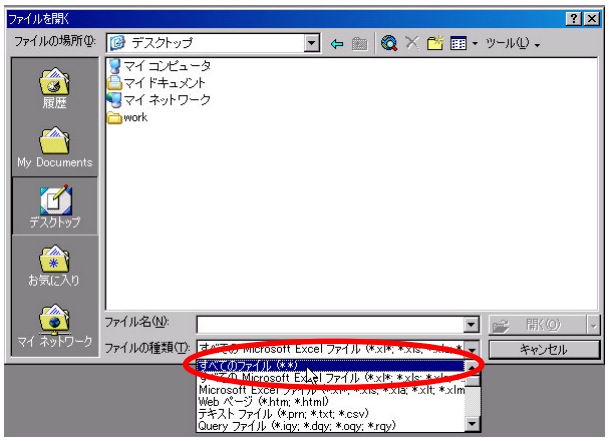
※ファイルの拡張子は、「txt」とします。



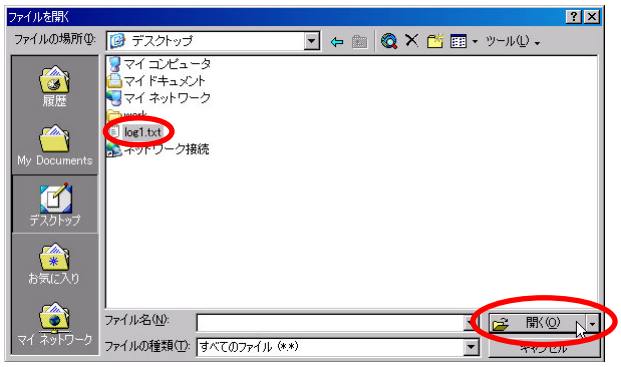
1. エクセルを立ち上げます。「ファイル→開く」を選択します。



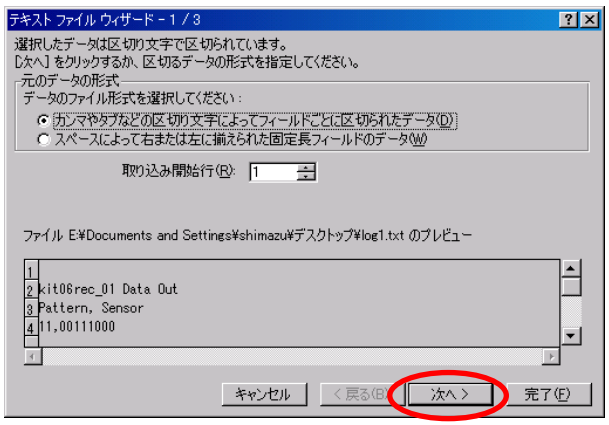
2. 「ファイルの種類」の▼をクリックします。



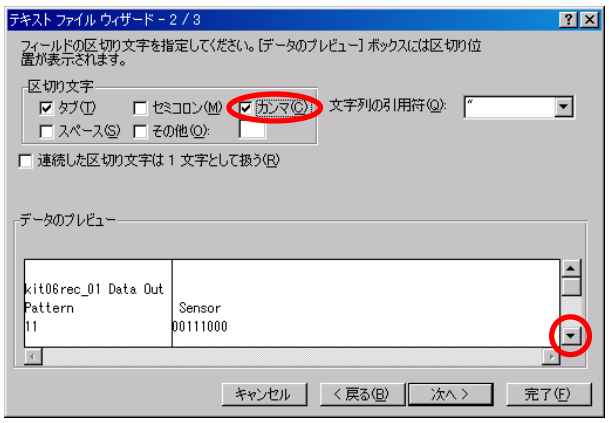
3. 「すべてのファイル(*.*)」を選択します。



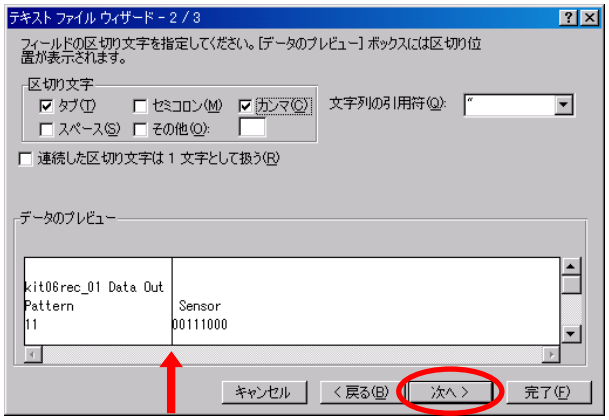
4. 「log1.txt」が表示されました。「log1.txt」を選択、開くをクリックします。



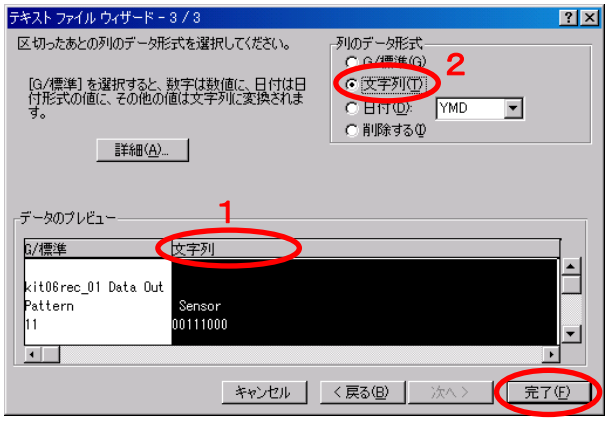
5. 開こうとしているデータは、テキストデータです。エクセルデータ(セル)に変換する作業を行います。次へをクリックします。



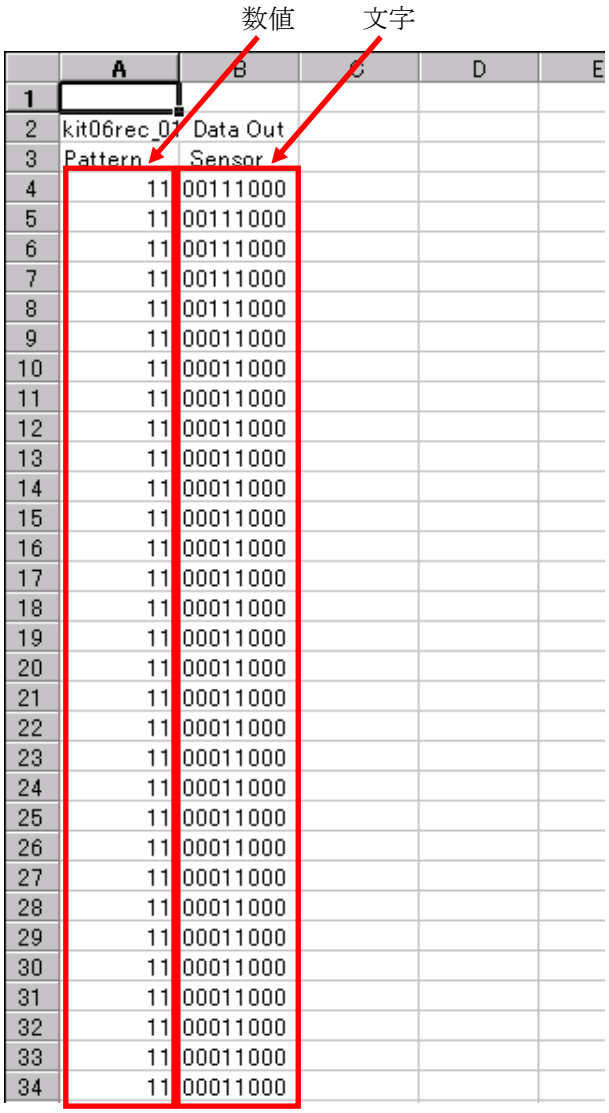
6. データはカンマで区切られています。区切り文字の「カンマ」のチェックを付けます。▼でデータのプレビュー欄を下げます。



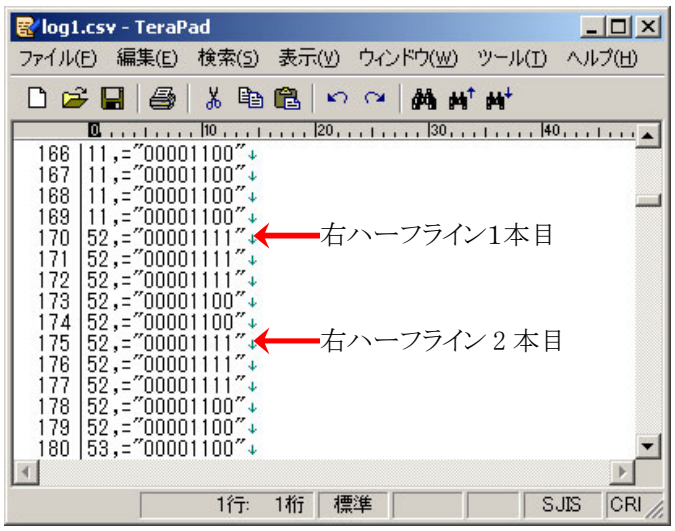
7. 矢印(↑)部分に縦線が入りました。これが列の区切りです。縦線が入ってなければ「区切り文字」のチェック指定がおかしいので確かめます。次へをクリックします。



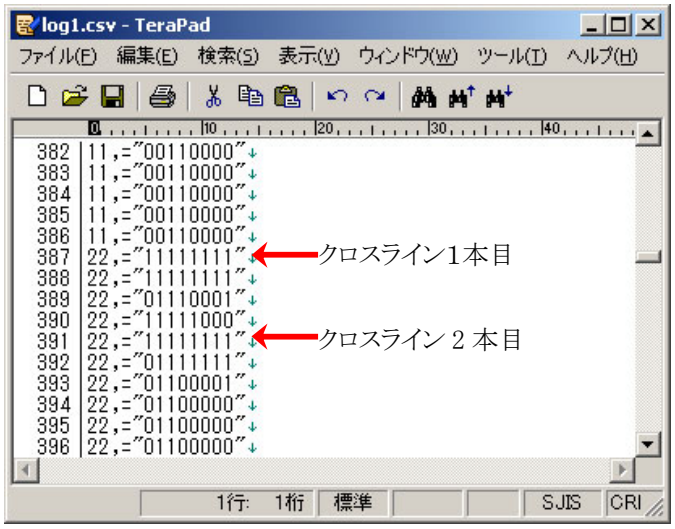
8. 次に、列のデータ形式を指定します。1列目は、パターンです。パターンは数値なので、標準のままでOKです。2 列目は、センサ値です。16 進数なので文字列扱いにします。1部分をクリックします。次に2部分の「文字列」を選択します。完了で完了です。



9. セルに変換されました。A 列はパターンを示す数値です。例えば、A4 は「11(じゅういち)」という 10 進数の数字です。B 列はセンサの状態を2進数で示す文字列です。



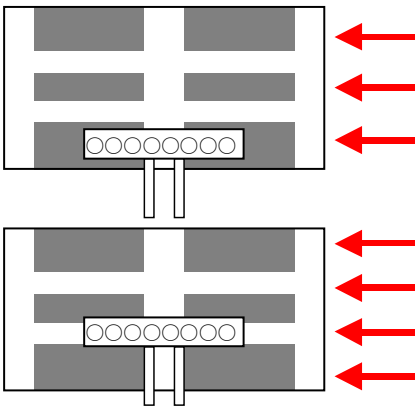
10．右ハーフラインを検出したところです。convertHexToBin 関数で 2 進数に変換されているので、log1.txt を直接見てセンサの状態が分かります。
16 進数でも想像できますが、やはり 2 進表記の方が分かりやすいです。



11. クロスラインを検出したところです。

6.10 取得タイミングについて

取得タイミングとマイコンカーのスピードによってはデータの取れ方が異なります。クロスラインを通過するときを例として見てみます。矢印が、ちょうど 10ms ごとにデータを取得している瞬間です。



左図のように、たまたま 10ms ごとの取得が、クロスラインを検出しない状態でした。ただし、パターンは 21,22,23 と変わってきますので、パターン番号からクロスラインを越えたと判断できます。

下左図は、クロスラインの白線を検出していますが、白線と白線の間の黒を検出していません。こちらもパターン番号から、クロスラインを越えたと判断できます。

7. プロジェクト「kit07rec_02」 外付けEEP-ROM にデータ保存

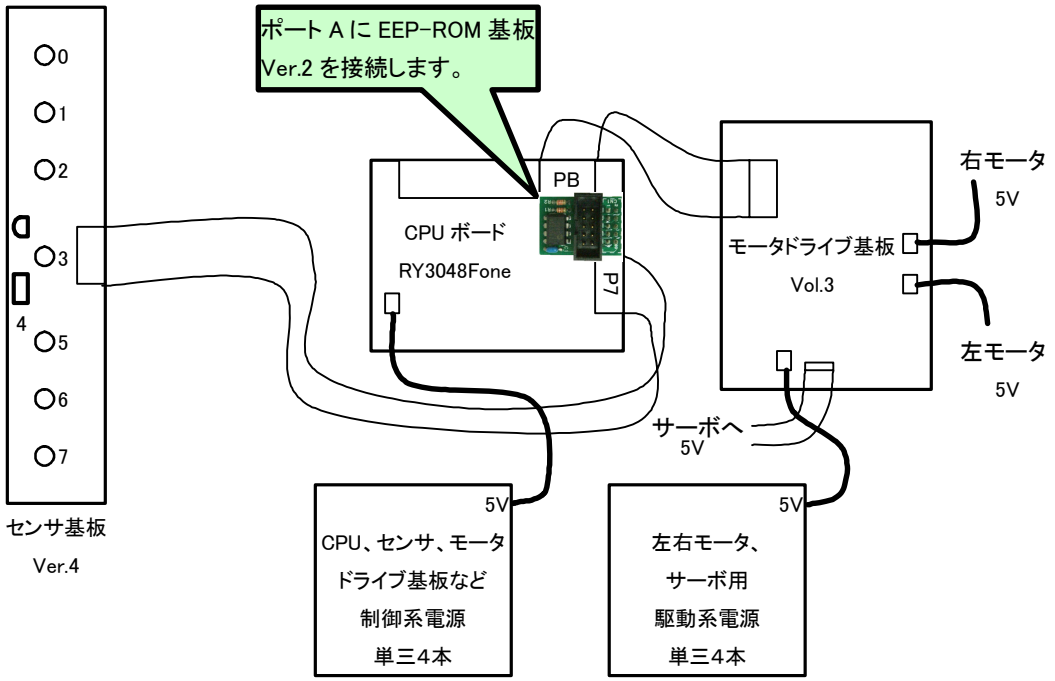
7.1 概要

マイコンカーの走行データを、外付け EEP-ROM に保存します。保存する内容は、
・パターン番号
・センサの値
・ハンドル角度
・左モータPWM値
・右モータPWM値
の 5 つです。これらのデータが走行開始から 10ms ごとに、4096 回分保存されます。40.96 秒間です。40.96 秒後、走行はそのまま行いますがデータ保存は自動で終了します。

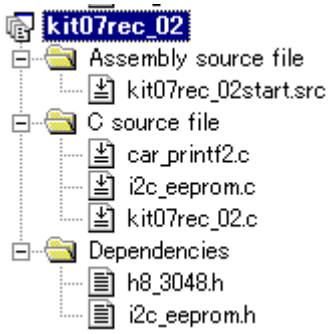
保存した走行データをパソコンに送り、マイコンカーがどう走ったかパソコン上で解析します。この情報を基に、プログラムのデバッグに役立てます。

7.2 マイコンカーの構成

マイコンカーキット Ver.4 に、EEP-ROM 基板を追加した構成です。LM350 追加セットで電池 8 本を直列に繋いでいる構成でも OK です。
・CPU ボードのポート 7 と、マイコンカーキット Ver.4 のセンサ基板 Ver.4 を接続します。
・CPU ボードのポート B と、マイコンカーキット Ver.4 のモータドライブ基板 Vol.3 を接続します。
・CPU ボードのポート A と、EEP-ROM 基板 Ver.2 を接続します。



7.3 プロジェクトの構成



・kit07rec_02start.src
・kit07rec_02.c
・car_printf2.c
・i2c_eeprom.c
の 4 ファイルあります。
h8_3048.h は kit07rec_02.c、car_printf2.c、i2c_eeprom.c でインクルードされているファイルです。
i2c_eeprom.h は kit07rec_02.c、i2c_eeprom.c でインクルードされているファイルです。

7.4 プログラム

```
 1 : /*********************************************************************************/
 2 : /* 走行データ記録マイコンカートレース基本プログラム「kit07rec_02.c」          */
 3 : /*                              2007.05 ジャパンマイコンカーラリー実行委員会     */
 4 : /*********************************************************************************/
 5 : /*
 6 : 本プログラムはkit07.cをベースに走行データを記録するプログラムです。
 7 :
 8 : kit07rec_02.cは、外付けEEP-ROM(24C256 32KB)にマイコンカーの走行
 9 : データを保存します。その後、保存したデータをパソコンへ転送します。
10 : パソコンでデータを解析すれば、なぜ脱輪したかなど問題を「見える化」
11 : することが出来、プログラムのデバッグに役立てることが出来ます。
12 : */
13 :
14 : /*====================================*/
15 : /* インクルード                       */
16 : /*====================================*/
17 : #include    <no_float.h>              /* stdioの簡略化 最初に置く */
18 : #include    <stdio.h>
19 : #include    <machine.h>
20 : #include    "h8_3048.h"
21 : #include    "i2c_eeprom.h"            /* EEP-ROM追加(データ記録)  */
22 : /*====================================*/
23 : /* シンボル定義                       */
24 : /*====================================*/
25 :
26 : /* 定数設定 */
27 : #define         TIMER_CYCLE     3071    /* タイマのサイクル 1ms     */
28 :                                         /* φ/8で使用する場合、     */
29 :                                         /* φ/8 = 325.5[ns]          */
30 :                                         /* ∴TIMER_CYCLE =           */
31 :                                         /*      1[ms] / 325.5[ns]   */
32 :                                         /*            = 3072        */
33 : #define         PWM_CYCLE       49151   /* PWMのサイクル 16ms       */
34 :                                         /* ∴PWM_CYCLE =             */
35 :                                         /*      16[ms] / 325.5[ns]  */
36 :                                         /*            = 49152       */
37 : #define         SERVO_CENTER    5000    /* サーボのセンタ値         */
38 : #define         HANDLE_STEP     26      /* 1° 分の値                */
39 :
40 : /* マスク値設定 ×：マスクあり(無効)  ○：マスク無し(有効) */
41 : #define         MASK2_2         0x66    /* ××○○××○×          */
42 : #define         MASK2_0         0x60    /* ××○○××××          */
43 : #define         MASK0_2         0x06    /* ××××××○×          */
44 : #define         MASK3_3         0xe7    /* ○○○××○○○          */
45 : #define         MASK0_3         0x07    /* ××××××○○          */
46 : #define         MASK3_0         0xe0    /* ○○○××××××          */
47 : #define         MASK4_0         0xf0    /* ○○○○××××          */
48 : #define         MASK0_4         0x0f    /* ××××○○○○          */
49 : #define         MASK4_4         0xff    /* ○○○○○○○○          */
50 :
51 : /*====================================*/
52 : /* プロトタイプ宣言                   */
53 : /*====================================*/
54 : void init( void );
55 : void timer( unsigned long timer_set );
56 : int check_crossline( void );
57 : int check_rightline( void );
58 : int check_leftline( void );
59 : unsigned char sensor_inp( unsigned char mask );
60 : unsigned char dipsw_get( void );
61 : unsigned char pushsw_get( void );
62 : unsigned char startbar_get( void );
63 : void led_out( unsigned char led );
64 : void speed( int accele_l, int accele_r );
```

printf 文を使用するので、stdio.h をインクルードします。
i2c_eeprom.c 内の関数を使うために、i2c_eeprom.h をインクルードします。

```
 65 :  void handle( int angle );
 66 :  char unsigned bit_change( char unsigned in );
 67 :  void convertHexToBin( unsigned char hex, char *s );
 68 :
 69 :  /*====================================*/
 70 :  /* グローバル変数の宣言               */
 71 :  /*====================================*/
 72 :  unsigned long   cnt0;                  /* timer関数用              */
 73 :  unsigned long   cnt1;                  /* main内で使用             */
 74 :  int             pattern;               /* パターン番号             */
 75 :
 76 :  /* データ保存関連 */
 77 :  int             iTimer10;              /* 取得間隔計算用           */
 78 :  int             saveIndex;             /* 保存インデックス         */
 79 :  int             saveSendIndex;         /* 送信インデックス         */
 80 :  int             saveFlag;              /* 保存フラグ               */
 81 :  char            saveData[8];           /* 一時保存エリア           */
 82 :  /*
 83 :  保存内容
 84 :  0:pattern   1:Sensor    2:handle    3:motor_l
 85 :  4:motor_r   5:          6:          7:
 86 :  */
 87 :
 88 :  /*************************************************************************/
 89 :  /* メインプログラム                                                      */
 90 :  /*************************************************************************/
 91 :  void main( void )
 92 :  {
 93 :      int     i;
 94 :      char    s[8];
 95 :
 96 :      /* マイコン機能の初期化 */
 97 :      init();                            /* 初期化                   */
 98 :      initI2CEeprom( &PADDR, &PADR, 0x5f, 7, 5);  /* EEP-ROM初期設定  */
 99 :      init_sci1( 0x00, 79 );             /* SCI1初期化               */
100 :      set_ccr( 0x00 );                   /* 全体割り込み許可         */
101 :
102 :      /* マイコンカーの状態初期化 */
103 :      handle( 0 );
104 :      speed( 0, 0 );
105 :
106 :      /* スタート時、スイッチが押されていればデータ転送モード */
107 :      if( pushsw_get() ) {
108 :          pattern = 71;
109 :          cnt1 = 0;
110 :      }
111 :
112 :      while( 1 ) {
113 :
114 :      P4DR = ~pattern;                   /* デバッグ用にパターン出力 */
115 :      I2CEepromProcess();                /* I2C EEP-ROM保存処理  */
116 :
117 :      switch( pattern ) {
```

中略

```
143 :      case 0:
144 :          /* スイッチ入力待ち */
145 :          if( pushsw_get() ) {
146 :              clearI2CEeprom();          /* 数秒かかる               */
147 :              pattern = 1;
148 :              cnt1 = 0;
149 :              break;
150 :          }
151 :          if( cnt1 < 100 ) {             /* LED点滅処理              */
152 :              led_out( 0x1 );
153 :          } else if( cnt1 < 200 ) {
154 :              led_out( 0x2 );
155 :          } else {
156 :              cnt1 = 0;
157 :          }
158 :          break;
159 :
160 :      case 1:
161 :          /* スタートバーが開いたかチェック */
162 :          if( !startbar_get() ) {
163 :              /* スタート！！ */
164 :              led_out( 0x0 );
165 :              pattern = 11;
166 :              saveIndex = 0;
167 :              saveFlag = 1;              /* データ保存開始           */
168 :              cnt1 = 0;
169 :              break;
170 :          }
171 :          if( cnt1 < 50 ) {              /* LED点滅処理              */
172 :              led_out( 0x1 );
173 :          } else if( cnt1 < 100 ) {
174 :              led_out( 0x2 );
175 :          } else {
176 :              cnt1 = 0;
177 :          }
```

データ保存関係の変数です。record_02 と同じです。

EEP-ROM を使用するために initI2CEeprom を実行します。printf 文で通信を使用しますので、init_sci1 で通信を初期化します。

電源を入れたとき、スイッチが押されていればデータ転送モードへ移ります。

デバッグ用としてポート 4 にパターンを出力します。LED をポート 4 に繋ぎます。

ループの中に I2CEepromProcess 関数を入れ、常に実行されるようにします。

EEP-ROM の内容をすべて0にします。

saveIndex は保存するアドレスを指定します。最初は 0 です。saveFlag=1 で保存を開始します。

```
178 :          break;
179 :
180 :      case 11:
181 :          /* 通常トレース */
182 :          if( pushsw_get() ) {
183 :              pattern = 71;               /* データ転送処理へ          */
184 :              break;
185 :          }
186 :
187 :          if( check_crossline() ) {       /* クロスラインチェック      */
188 :              pattern = 21;
189 :              break;
190 :          }
191 :          if( check_rightline() ) {       /* 右ハーフラインチェック    */
192 :              pattern = 51;
193 :              break;
194 :          }
195 :          if( check_leftline() ) {        /* 左ハーフラインチェック    */
196 :              pattern = 61;
197 :              break;
198 :          }
199 :          switch( sensor_inp(MASK3_3) ) {
200 :              case 0x00:
201 :                  /* センター→まっすぐ */
202 :                  handle( 0 );
203 :                  speed( 100 ,100 );
204 :                  break;
```

パターン 11 のときにスイッチを押すと、パターン 71 へ移ります。
パターン 71 は、データを転送する部分です。

中略

```
519 :      case 71:
520 :          /* 停止 */
521 :          handle( 0 );
522 :          speed( 0, 0 );
523 :          saveFlag = 0;
524 :          saveSendIndex = 0;
525 :          pattern = 72;
526 :          cnt1 = 0;
527 :          break;
528 :
529 :      case 72:
530 :          /* 1s待ち */
531 :          if( cnt1 > 1000 ) {
532 :              pattern = 73;
533 :              cnt1 = 0;
534 :          }
535 :          break;
536 :
537 :      case 73:
538 :          /* スイッチが離されたかチェック */
539 :          if( !pushsw_get() ) {
540 :              pattern = 74;
541 :              cnt1 = 0;
542 :          }
543 :          break;
544 :
545 :      case 74:
546 :          /* スイッチが押されたかチェック */
547 :          led_out( (cnt1/500) % 2 + 1 );
548 :          if( pushsw_get() ) {
549 :              pattern = 75;
550 :              cnt1 = 0;
551 :          }
552 :          break;
553 :
554 :      case 75:
555 :          /* タイトル転送、準備 */
556 :          printf( "¥n" );
557 :          printf( "kit07rec_02 Data Out¥n" );
558 :          printf( "Pattern, Sensor, ハンドル, 左モータ, 右モータ¥n" );
559 :          pattern = 76;
560 :          break;
561 :
562 :      case 76:
563 :          /* データ転送 */
564 :          led_out( (saveSendIndex/32) % 2 + 1 ); /* LED点滅処理       */
565 :
566 :          /* 終わりのチェック */
567 :          if( (readI2CEeprom( saveSendIndex )==0) ||
568 :                                  (saveSendIndex >= 0x8000) ) {
569 :              pattern = 77;
570 :              cnt1 = 0;
571 :              break;
572 :          }
573 :
574 :          /* データの転送 */
575 :          convertHexToBin( readI2CEeprom(saveSendIndex+1), s );
576 :          printf( "%d,=¥"%8s¥",%d,%d,%d¥n",
577 :              (char)readI2CEeprom( saveSendIndex+0 ), /* パターン      */
578 :              s,                                     /* センサ        */
579 :              (char)readI2CEeprom( saveSendIndex+2 ), /* ハンドル      */
```

パターン 71 以降は、保存したデータを転送する部分です。
まず、マイコンカーを停止させます。
saveFlag=0 でデータ保存を終了します。saveSendIndex は転送するアドレスを指定する変数です。0から順番に転送するので、最初にクリアします。
パターン 72 へ移ります。

単純に1秒待ちます。1 秒後、パターン 73 へ移ります。

スイッチが離されていれば、パターン 74 へ移ります。

スイッチが押されれば、パターン 75 へ移ります。
このとき、LED を点滅させ、転送待機状態だと言うことを外部に知らせます。

タイトルを転送します。すぐに、パターン 76 へ移ります。

データがある限り、転送し続けます。転送するデータは、5 種類です。
保存数を超えてデータを転送しようとすると次のパターンへ移り、転送を終了します。

センサ値を 2 進数に変換します。

```
580 :              (char)readI2CEeprom( saveSendIndex+3 ), /* 左モータ      */
581 :              (char)readI2CEeprom( saveSendIndex+4 )  /* 右モータ      */
582 :          );
583 :
584 :          saveSendIndex += 8;              /* 次の準備                  */
585 :          break;
586 :
587 :      case 77:
588 :          /* 転送終了 */
589 :          led_out( 0x3 );
590 :          break;
591 :
592 :      default:
593 :          /* どれでもない場合は待機状態に戻す */
594 :          pattern = 0;
595 :          break;
596 :      }
597 :      }
598 :  }
599 :
600 :  /************************************************************************/
601 :  /* H8/3048F-ONE 内蔵周辺機能  初期化                                    */
602 :  /************************************************************************/
603 :  void init( void )
604 :  {
605 :      /* I/Oポートの入出力設定 */
606 :      P1DDR = 0xff;
607 :      P2DDR = 0xff;
608 :      P3DDR = 0xff;
609 :      P4DDR = 0xff;
610 :      P5DDR = 0xff;
611 :      P6DDR = 0xf0;                    /* CPU基板上のDIP SW        */
612 :      P8DDR = 0xff;
613 :      P9DDR = 0xf7;                    /* 通信ポート               */
614 :      PADDR = 0x5f;                    /* EEP-ROM                  */
615 :      PBDR  = 0xc0;
616 :      PBDDR = 0xfe;                    /* モータドライブ基板Vol.3  */
617 :      /* ※センサ基板のP7は、入力専用なので入出力設定はありません    */
618 :
619 :      /* ITU0 1msごとの割り込み */
620 :      ITU0_TCR = 0x23;
621 :      ITU0_GRA = TIMER_CYCLE;
622 :      ITU0_IER = 0x01;
623 :
624 :      /* ITU3,4 リセット同期PWMモード 左右モータ、サーボ用 */
625 :      ITU3_TCR = 0x23;
626 :      ITU_FCR  = 0x3e;
627 :      ITU3_GRA = PWM_CYCLE;            /* 周期の設定               */
628 :      ITU3_GRB = ITU3_BRB = 0;         /* 左モータのPWM設定        */
629 :      ITU4_GRA = ITU4_BRA = 0;         /* 右モータのPWM設定        */
630 :      ITU4_GRB = ITU4_BRB = SERVO_CENTER; /* サーボのPWM設定        */
631 :      ITU_TOER = 0x38;
632 :
633 :      /* ITUのカウントスタート */
634 :      ITU_STR = 0x09;
635 :  }
636 :
637 :  /************************************************************************/
638 :  /* ITU0 割り込み処理                                                    */
639 :  /************************************************************************/
640 :  #pragma interrupt( interrupt_timer0 )
641 :  void interrupt_timer0( void )
642 :  {
643 :      ITU0_TSR &= 0xfe;                /* フラグクリア             */
644 :      cnt0++;
645 :      cnt1++;
646 :
647 :      /* データ保存関連 */
648 :      iTimer10++;
649 :      if( iTimer10 >= 10 ) {
650 :          iTimer10 = 0;
651 :          if( saveFlag ) {
652 :              saveData[0] = pattern;           /* パターン         */
653 :              saveData[1] = sensor_inp(0xff);  /* センサ           */
654 :              /* 2はハンドル関数内で保存 */
655 :              /* 3はモータ関数内で左モータPWM値保存 */
656 :              /* 4はモータ関数内で右モータPWM値保存 */
657 :              saveData[5] = 0;                 /* 予備             */
658 :              saveData[6] = 0;                 /* 予備             */
659 :              saveData[7] = 0;                 /* 予備             */
660 :              setPageWriteI2CEeprom( saveIndex, 8, saveData );
661 :              saveIndex += 8;
662 :              if( saveIndex >= 0x8000 ) saveFlag = 0;
663 :          }
664 :      }
665 :  }
```

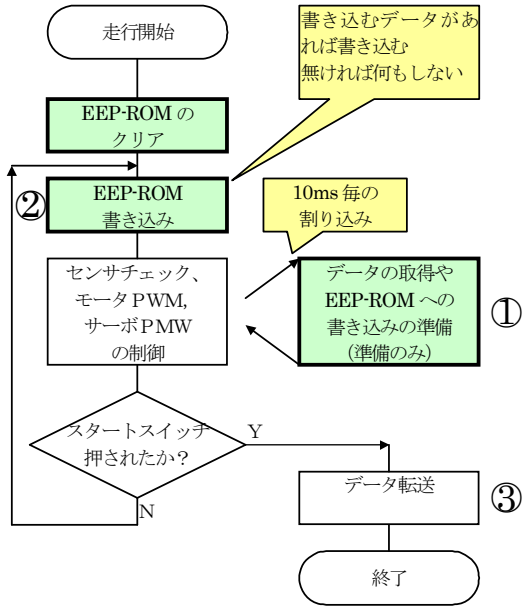
何もしません。電源が切られるのを待ちます。

ポート A には、
bit7:EEP-ROM SCL 端子
bit5:EEP-ROM SDA 端子
が接続されています。2 端子は入力にします。
入出力設定は
01011111→0x5f
となります。

10ms ごとに 8 バイト分のデータを EEP-ROM に書き込みます。ただし、割り込み内では、書き込みの準備をしているだけです。
EEP-ROM が一杯になると、saveFlag を 0 にして保存を強制的に終了します。

以下、略

7.5 プログラムの概要



マイコンカーは、
・センサ(コースセンサ、エンコーダ)を見ながら
・駆動モータの制御
・サーボ(ステアリングモータ)の制御
を行っています。**これらの走行に影響が無いようにデータ記録を行わなければいけません。**

データの取得や EEP-ROM への書き込みの準備は、10ms ごとに割り込みの中で行います。できるだけはやく命令を終わらせて走行に影響のないようにします(①部分)。

実際の書き込みも、走行に影響のないように少しずつ行います(②部分)。

このように、走行が最優先、その合間をぬってデータの書き込みを行います。

走行終了後、マイコンカーを取り上げてスタートスイッチを押します。パソコンと RS232C ケーブルを接続してデータを転送し、取得したデータがパソコンに取り込まれます(③部分)。

ポイントは、
・EEP-ROM への書き込みは遅いので、準備のみを行います(①部分)。
・無限ループ内で、あまり実行時間をかけないようにしながら少しずつ書き込みます(②部分)。
・EEP-ROM なので電源を切っても、保存データは消えません。
・保存するデータは、パターン番号、センサの値、ハンドル角度、左モータPWM値、右モータPWM値の5つです。

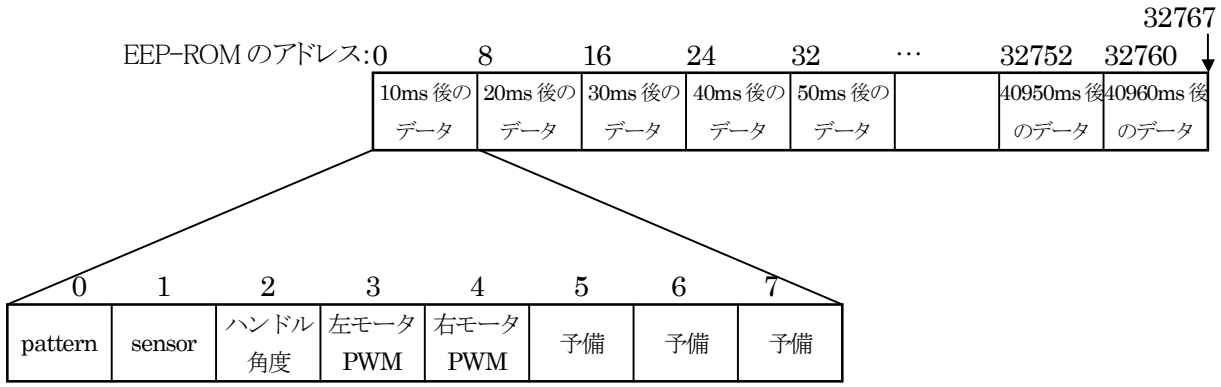
7.6 プログラムの解説

7.6.1 データの保存

EEP-ROM は 32KB あります。**データは 5 バイトですが、保存数は 2 の n 乗個ごとなので 8 個保存します。**保存間隔は 10ms です。保存できる時間は、

保存できる時間＝EEP-ROM の容量 32768×保存間隔 10ms÷1 回の保存数 8 バイト＝40.960 秒

となります。



7.6.2 送信内容

```
562 :       case 76:
563 :           /* データ転送 */
564 :           led_out( (saveSendIndex/32) % 2 + 1 ); /* LED 点滅処理       */
565 :
566 :           /* 終わりのチェック */
567 :           if( (readI2CEeprom( saveSendIndex )==0) ||
568 :                                   (saveSendIndex >= 0x8000) ) {
569 :               pattern = 77;
570 :               cnt1 = 0;
571 :               break;
572 :           }
573 :
574 :           /* データの転送 */
575 :           convertHexToBin( readI2CEeprom(saveSendIndex+1), s );
576 :           printf( "%d,=¥"%8s¥",%d,%d,%d¥n",
577 :               (char)readI2CEeprom( saveSendIndex+0 ), /* パターン      */
578 :               s,                                      /* センサ        */
579 :               (char)readI2CEeprom( saveSendIndex+2 ), /* ハンドル      */
580 :               (char)readI2CEeprom( saveSendIndex+3 ), /* 左モータ      */
581 :               (char)readI2CEeprom( saveSendIndex+4 )  /* 右モータ      */
582 :           );
583 :
584 :           saveSendIndex += 8;             /* 次の準備                  */
585 :           break;
```

送信内容は、576～582 行の内容です。

```
printf( "%d,=¥"%8s¥",%d,%d,%d¥n",
        (char)readI2CEeprom( saveSendIndex+0 ), /* パターン     */
         s,                                     /* センサ       */
        (char)readI2CEeprom( saveSendIndex+2 ), /* ハンドル     */
        (char)readI2CEeprom( saveSendIndex+3 ), /* 左モータ     */
        (char)readI2CEeprom( saveSendIndex+4 )  /* 右モータ     */
```

パターン、ハンドル、左モータ、右モータは 10 進数、センサは 8 文字の文字列で出力します。センサの値は、printf 文を実行する前に、convertHexToBin 関数で 2 進数に変換しています。変換した文字列が格納されている s 配列の内容を出力するだけです。

7.7 プログラムの調整

まず初めに、「kit07rec_02.c」の下記内容を自分のマイコンカーに合わせて調整します。他にも、調整する部分は調整してください。

```
37 ：  #define          SERVO_CENTER     5000     /* サーボのセンタ値          */
中略
321 ：                handle( -38 );                左クランクを曲がるときのハンドル角度
中略
330 ：                handle( 38 );                 右クランクを曲がるときのハンドル角度
```

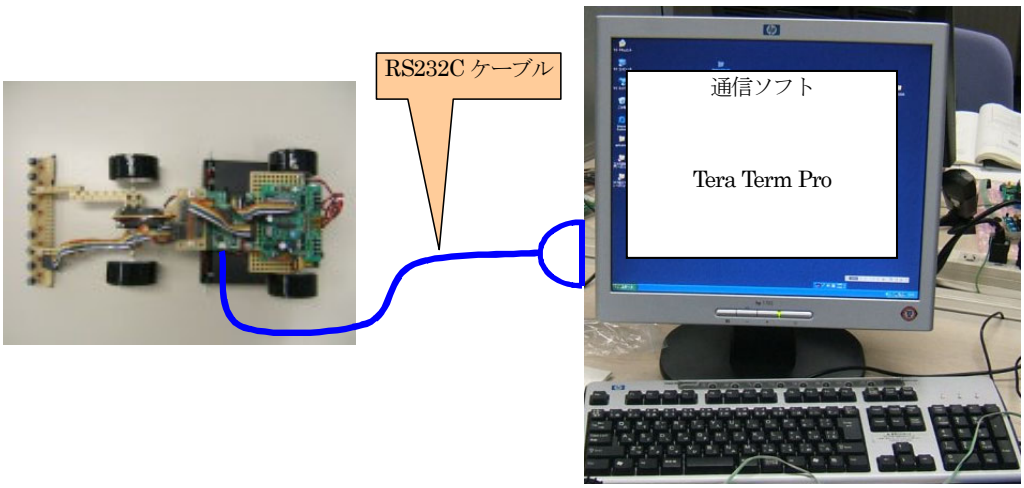
調整ができたら、プロジェクト「kit07rec_02」をビルドして、「kit07rec_02.mot」ファイルを CPU ボードに書き込みます。

7.8 走行からデータ転送までの流れ

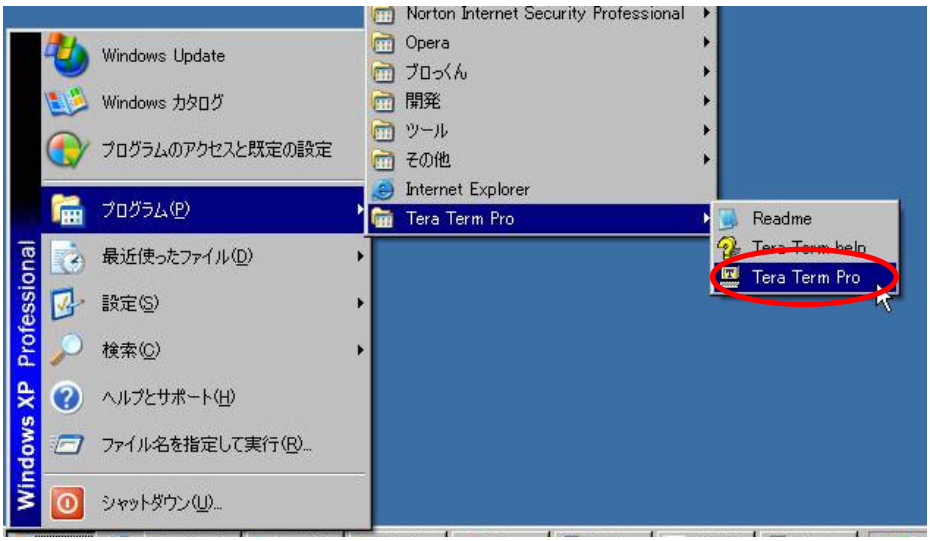
走行データを取りたい部分のコースを普通に走らせます。走行データが保存できるのは、スタートしてから約 40 秒です。

走らせた後、**電源を切ります。EEP-ROM なので、電源を切ってもデータは消えません。**

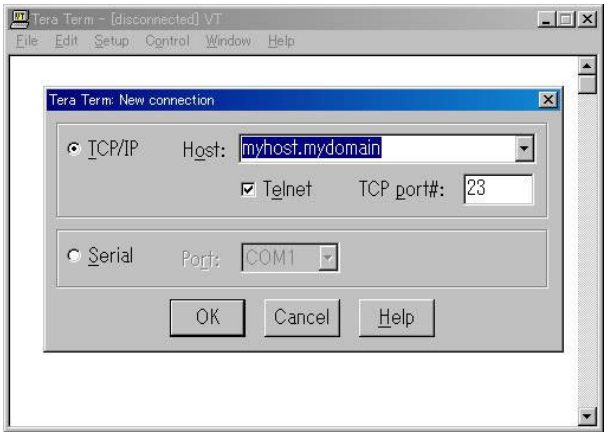
RS232C ケーブルをマイコンカーに接続します。



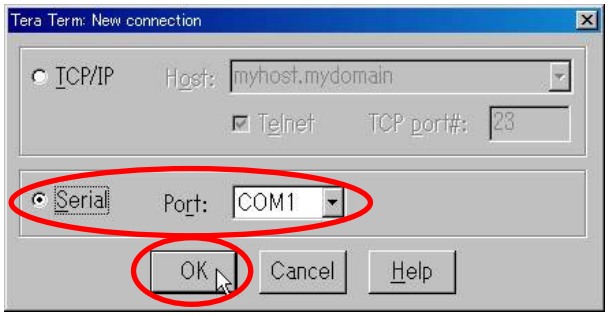
1. マイコンカーの電源は、まだ入れません。Tera Term Pro を立ち上げます。



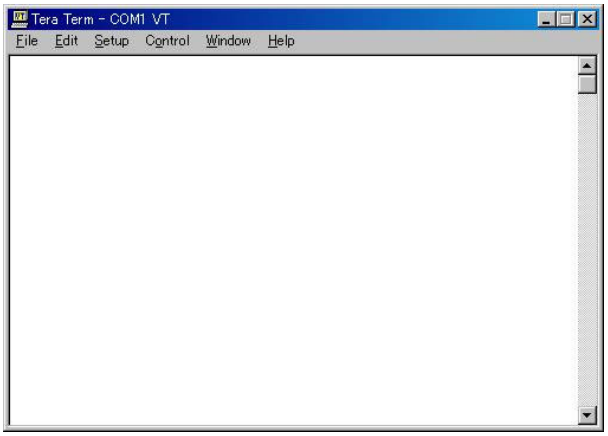
2.「スタート→すべてのプログラム(またはプログラム)→Tera Term Pro→Tera Term Pro」で Tera Term Pro を立ち上げます。



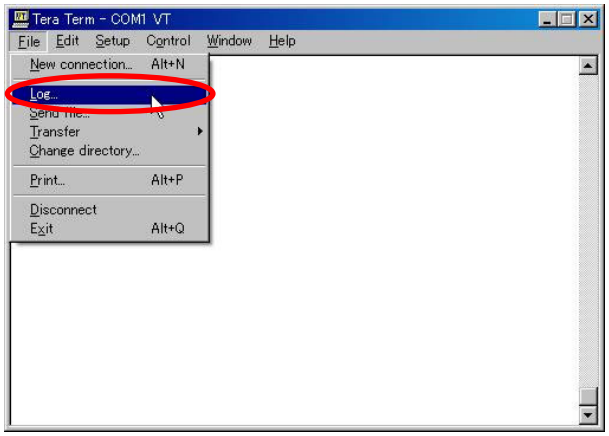
3. 最初に接続先を確認する画面が表示されます。



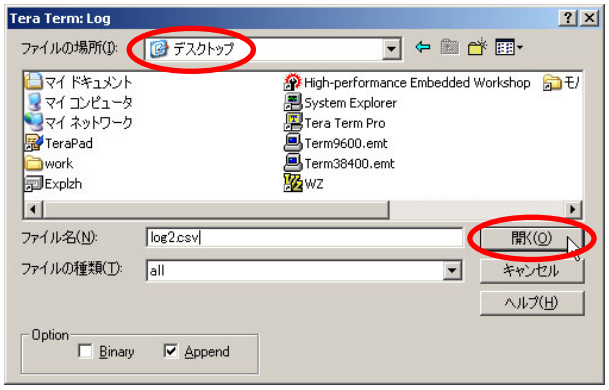
4.「Serial」を選んで、各自のパソコンに合わせてポート番号を選びます。選択後、OK をクリック、次へ進みます。

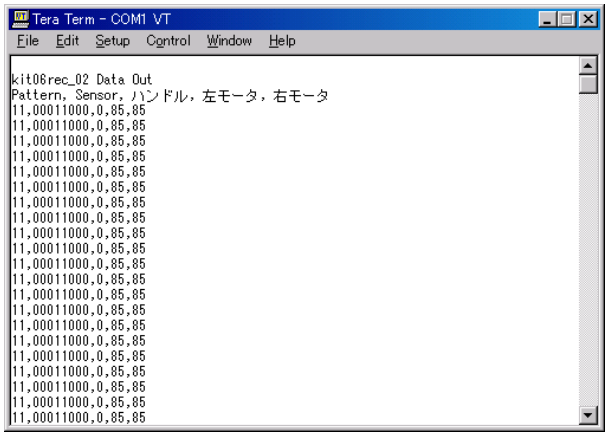


5. 立ち上がりました。



6. データをファイルに保存します。「File→Log」を選択します。





7. 「ファイルの場所」に保存するフォルダを選択します。「ファイル名」には、保存するファイルの名前を入力します。ここでは「log2.csv」と入力します。開く をクリックします。これで、パソコン側の準備は完了です。いよいよ、マイコンカーの操作をします。

8. **マイコンカーのモータドライブ基板のスイッチを押しながらマイコンカーの電源を入れ、すぐにスイッチを離します。LED の点滅が遅くなっているはずです。これが、データ転送モードの状態です。**
再度、スイッチを押すとデータが転送されます。転送中は、LED の点滅が高速になります。LED が 2 個とも点灯したら転送終了です。マイコンカーの電源を切って、TeraTermPro を終了してください。

7.9 エクセルへの取り込み方

	A	B	C	D	E	F
1						
2	kit06rec_02 Data Out					
3	Pattern	Sensor	ハンドル	左モータ	右モータ	
4	11	00011000	0	85	85	
5	11	00011000	0	85	85	
6	11	00011000	0	85	85	
7	11	00011000	0	85	85	
8	11	00011000	0	85	85	
9	11	00011000	0	85	85	
10	11	00011000	0	85	85	
11	11	00011000	0	85	85	
12	11	00011000	0	85	85	

1. csv ファイルをダブルクリックすると、自動的にエクセルで立ち上がります。解析してみます。

	パターン	センサ値	ハンドル	左モータ	右モータ	
178	11	00011000	0	85	85	
179	11	00011000	0	85	85	
180	11	00011000	0	85	85	
181	11	00011000	0	85	85	
182	22	11111111	0	0	0	←クロスライン1本目
183	22	11111111	0	0	0	
184	22	00011000	0	0	0	
185	22	11111000	0	0	0	
186	22	11111111	0	0	0	←クロスライン 2 本目
187	22	11111111	0	0	0	
188	22	00011000	0	0	0	
189	22	00011000	0	0	0	
190	22	00011000	0	0	0	

233	23	00011000	0	34	34	
234	23	00011000	0	34	34	
235	23	00011000	0	34	34	
236	31	11111000	-38	8	42	←左クランク発見!!
237	31	11111000	-38	8	42	
238	31	11111000	-38	8	42	
239	31	11111000	-38	8	42	
240	31	11111000	-38	8	42	
241	31	11111000	-38	8	42	
242	31	11111000	-38	8	42	
243	31	11000000	-38	8	42	
244	31	00000000	-38	8	42	
245	31	00000000	-38	8	42	
246	31	00000000	-38	8	42	

339	32	10000011	-38	8	42	
340	32	10000001	-38	8	42	
341	32	10000001	-38	8	42	
342	32	10000001	-38	8	42	
343	32	11000001	-38	8	42	
344	32	11000001	-38	8	42	
345	32	11000000	-38	8	42	
346	32	11000000	-38	8	42	
347	32	11000000	-38	8	42	
348	32	11100000	-38	8	42	
349	32	11100000	-38	8	42	
350	11	01100000	-10	58	68	←中心線へ復帰
351	11	01100000	-10	58	68	
352	11	01110000	-10	58	68	

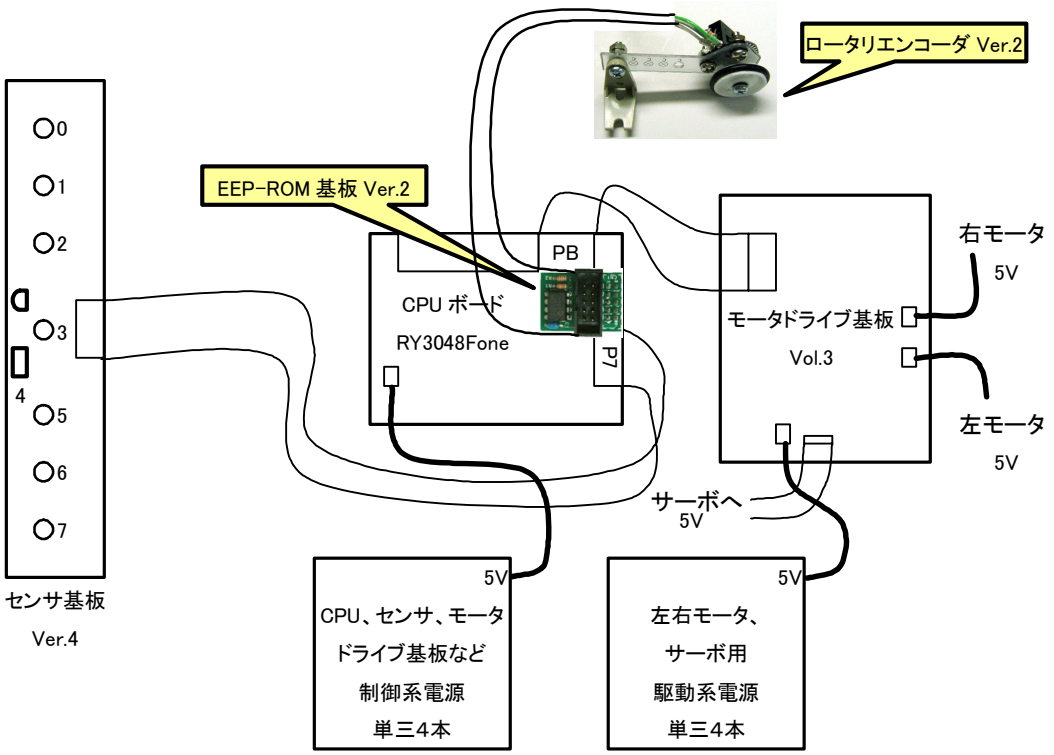
8. プロジェクト「kit07rec_03」 エンコーダプログラムの追加

8.1 概要

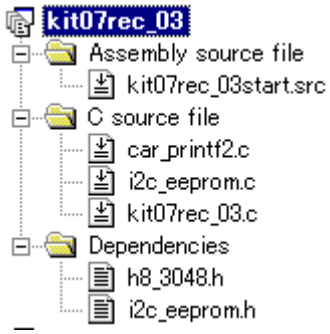
本プログラムは、
・**kit07rec_02.c**…外付け EEP-ROM(24C256 32KB)にマイコンカーの走行
・**kit07enc_03.c**…エンコーダによる速度制御
を合わせたプログラムです。マイコンカーの走行データを、外付け EEP-ROM に保存します。エンコーダ値も保存するので、速度が分かります。
※kit07enc_03.c は、ワークスペース「kit07enc」のプロジェクト「kit07enc_03」のファイルです。詳しくは、ロータリエンコーダ実習マニュアルを参照してください。

8.2 マイコンカーの構成

マイコンカーキット Ver.4 に、EEP-ROM 基板を追加した構成です。LM350 追加セットで電池 8 本を直列に繋いでいる構成でも OK です。
・CPU ボードのポート 7 と、マイコンカーキット Ver.4 のセンサ基板 Ver.4 を接続します。
・CPU ボードのポート B と、マイコンカーキット Ver.4 のモータドライブ基板 Vol.3 を接続します。
・CPU ボードのポート A と、EEP-ROM 基板 Ver.2 を接続します。
・EEP-ROM 基板 Ver.2 と、ロータリエンコーダを接続します。



8.3 プロジェクトの構成



・kit07rec_03start.src
・kit07rec_03.c
・car_printf2.c
・i2c_eeprom.c
の 4 ファイルあります。
h8_3048.h は kit07rec_03.c、car_printf2.c、i2c_eeprom.c でインクルードされているファイルです。
i2c_eeprom.h は kit07rec_03.c、i2c_eeprom.c でインクルードされているファイルです。

8.4 プログラムの解説

8.4.1 入出力設定

ポート A の bit0 にロータリエンコーダを追加します。そのため、ポート A の bit0 を入力端子にします。ポート A の接続は下記のようになります。

bit	7	6	5	4	3	2	1	0
入力 or 出力	**SCL 入力**	出力	**SDA 入力**	出力	出力	出力	出力	**エンコーダ入力**

入力は"0"、出力は"1"とします。

bit	7	6	5	4	3	2	1	0
0 or 1	**0**	1	**0**	1	1	1	1	**0**

下記がプログラムの変更部分です。

```
633 :  void init( void )
634 :  {
635 :      /* I/O ポートの入出力設定 */
636 :      P1DDR = 0xff;
637 :      P2DDR = 0xff;
638 :      P3DDR = 0xff;
639 :      P4DDR = 0xff;
640 :      P5DDR = 0xff;
641 :      P6DDR = 0xf0;                    /* CPU 基板上の DIP SW        */
642 :      P8DDR = 0xff;
643 :      P9DDR = 0xf7;                    /* 通信ポート                 */
644 :      PADDR = 0x5e;                    /* EEP-ROM,エンコーダ         */
645 :      PBDR  = 0xc0;
646 :      PBDDR = 0xfe;                    /* モータドライブ基板 Vol.3   */
```

2 進数で"0101 1110"なので、16 進数で 0x5e となります。「kit07rec_03.c」は下記のようになります。

```
 99 :  void main( void )
100 :  {
101 :      int     i;
102 :      char    s[8];
103 :
104 :      /* マイコン機能の初期化 */
105 :      init();                              /* 初期化                    */
106 :      initI2CEeprom( &PADDR, &PADR, 0x5e, 7, 5);  /* EEP-ROM 初期設定  */
107 :      init_scil( 0x00, 79 );               /* SCI1 初期化               */
108 :      set_ccr( 0x00 );                     /* 全体割り込み許可          */
```

EEP-ROM の初期化関数の入出力設定も 0x5e とします。

8.4.2 割り込みプログラム

```
673 :  #pragma interrupt( interrupt_timer0 )
674 :  void interrupt_timer0( void )
675 :  {
676 :      unsigned int i;
677 :
678 :      ITU0_TSR &= 0xfe;                    /* フラグクリア              */
679 :      cnt0++;
680 :      cnt1++;
681 :
682 :      iTimer10++;
683 :      if( iTimer10 >= 10 ) {
684 :          iTimer10 = 0;
685 :
686 :          /* エンコーダ関連 */
687 :          i = ITU2_CNT;
688 :          iEncoder       = i - uEncoderBuff;
689 :          lEncoderTotal += iEncoder;
690 :          if( iEncoder > iEncoderMax )
691 :                      iEncoderMax = iEncoder;
692 :          uEncoderBuff = i;
693 :
694 :          /* データ保存関連 */
695 :          if( saveFlag ) {
696 :              saveData[0] = pattern;              /* パターン          */
697 :              saveData[1] = sensor_inp(0xff);     /* センサ            */
698 :              /* 2 はハンドル関数内で保存 */
699 :              /* 3 はモータ関数内で左モータ PWM 値保存 */
700 :              /* 4 はモータ関数内で右モータ PWM 値保存 */
701 :              saveData[5] = iEncoder;             /* エンコーダ        */
702 :              saveData[6] = 0;                    /* 予備              */
703 :              saveData[7] = 0;                    /* 予備              */
704 :              setPageWriteI2CEeprom( saveIndex, 8, saveData );
705 :              saveIndex += 8;
706 :              if( saveIndex >= 0x8000 ) saveFlag = 0;
707 :          }
708 :      }
709 :  }
```

エンコーダ処理を追加しています。エンコーダ処理も 10ms ごとの処理なので、iTimer 変数を兼用して、10ms ごとの処理部分に入れています。
701 行で、エンコーダ値を保存しています。

8.4.3 送信内容

```
591 :       case 76:
592 :           /* データ転送 */
593 :           led_out( (saveSendIndex/32) % 2 + 1 ); /* LED 点滅処理      */
594 :
595 :           /* 終わりのチェック */
596 :           if( (readI2CEeprom( saveSendIndex )==0) ||
597 :                                   (saveSendIndex >= 0x8000) ) {
598 :               pattern = 77;
599 :               cnt1 = 0;
600 :               break;
601 :           }
602 :
603 :           /* データの転送 */
604 :           convertHexToBin( readI2CEeprom(saveSendIndex+1), s );
605 :           printf( "%d,=¥"%8s¥",%d,%d,%d,%d¥n",
606 :               (char)readI2CEeprom( saveSendIndex+0 ), /* パターン     */
607 :               s,                                      /* センサ       */
608 :               (char)readI2CEeprom( saveSendIndex+2 ), /* ハンドル     */
609 :               (char)readI2CEeprom( saveSendIndex+3 ), /* 左モータ     */
610 :               (char)readI2CEeprom( saveSendIndex+4 ), /* 右モータ     */
611 :               (char)readI2CEeprom( saveSendIndex+5 )  /* エンコーダ   */
612 :           );
613 :
614 :           saveSendIndex += 8;             /* 次の準備                 */
615 :           break;
```

送信内容は、605～612 行の内容です。

```
printf( "%d,=¥"%8s¥",%d,%d,%d,%d¥n",
        (char)readI2CEeprom( saveSendIndex+0 ), /* パターン     */
        s                                       /* センサ       */
        (char)readI2CEeprom( saveSendIndex+2 ), /* ハンドル     */
        (char)readI2CEeprom( saveSendIndex+3 ), /* 左モータ     */
        (char)readI2CEeprom( saveSendIndex+4 ), /* 右モータ     */
        (char)readI2CEeprom( saveSendIndex+5 )  /* エンコーダ   */
```

パターン、ハンドル、左モータ、右モータ、エンコーダは 10 進数、センサは 8 文字の文字列で出力します。センサの値は、printf 文を実行する前に、convertHexToBin 関数で 2 進数に変換しています。変換した文字列が格納されている s 配列の内容を出力するだけです。

8.5 ロータリエンコーダに関わる計算

プログラムを変更するに当たって、ロータリエンコーダに関わる値も変更する必要があります。プログラムの変更前に、下記内容について計算しておきましょう。

ロータリエンコーダのタイヤの半径	mm **(A)**
1回転のパルス数(ロータリエンコーダ Ver.2 は 72) ※標準は立ち上がり、立ち下がりでカウントする設定です。	パルス **(B)**
円周＝2π×(A)	mm **(C)**
1000mm 進んだときのパルス数は、 1000:x＝(C):(B) ∴x＝1000×(B)÷(C)	パルス **(D)**
100mm 進んだときのパルス数は、 (E)＝(D)×0.1 ※四捨五入した整数	パルス **(E)**
1m/s で進んだとき、10ms 間のパルス数は、 (F)＝(D)×0.01 ※四捨五入した整数	パルス **(F)**
2m/s で進んだとき、10ms 間のパルス数は、 (G)＝(D)×0.02 ※四捨五入した整数	パルス **(G)**

8.6 走行からデータ転送までの流れ

まず初めに、「kit07rec_03.c」の下記内容を自分のマイコンカーに合わせて調整します。他にも、調整する部分は調整してください。

```
37 :  #define          SERVO_CENTER     5000     /* サーボのセンタ値          */
```

各自マイコンカーのサーボセンタ値に変更します。

```
284 :          if( iEncoder >= 11 ) {
```

パターン 12 の右大曲げ処理時の速度を設定します。秒速 1m/s にする場合は、**(F)**の値を設定します。

```
308 :          if( iEncoder >= 11 ) {
```

パターン 13 の左大曲げ処理時の速度を設定します。秒速 1m/s にする場合は、**(F)**の値を設定します。

```
330 :          if( lEncoderTotal-lEncoderLine >= 109 ) {    /* 約 10cm たったか？ */
```

クロスラインを見つけたとき、センサを見ずに進む距離を設定します。10cm にする場合は、**(E)**の値を設定します。

```
341 :              handle( -38 );
```

左クランクを曲がるときの角度を設定します。左に曲げることができる最大切れ角にします。

```
350 :               handle( 38 );
```

右クランクを曲がるときの角度を設定します。左に曲げることができる最大切れ角にします。

```
356 :           if( iEncoder >= 11 ) {       /* クロスライン後のスピード制御 */
```

パターン 23 のクロスライン検出後、徐行して進むときの速度を設定します。秒速 1m/s にする場合は、**(F)**の値を設定します。

```
429 :           if( lEncoderTotal-lEncoderLine >= 109 ) {    /* 約 10cm たったか？ */
```

右ハーフラインを見つけたとき、センサを見ずに進む距離を設定します。10cm にする場合は、**(E)**の値を設定します。

```
444 :           if( iEncoder >= 11 ) {       /* ハーフラインライン後のスピード制御 */
```

パターン 53 の右ハーフライン検出後、徐行して進むときの速度を設定します。秒速 2m/s にする場合は、**(G)**の値を設定します。

```
494 :           if( lEncoderTotal-lEncoderLine >= 109 ) {    /* 約 10cm たったか？ */
```

左ハーフラインを見つけたとき、センサを見ずに進む距離を設定します。10cm にする場合は、**(E)**の値を設定します。

```
509 :           if( iEncoder >= 11 ) {       /* ハーフラインライン後のスピード制御 */
```

パターン 63 の左ハーフライン検出後、徐行して進むときの速度を設定します。秒速 2m/s にする場合は、**(G)**の値を設定します。

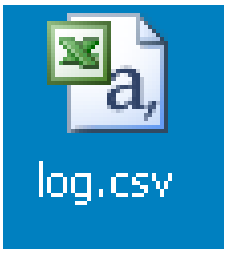
```
655 :       ITU2_TCR = 0x14;                    /* PA0 端子のパルスでカウント*/
```

エンコーダのパルスカウントを立ち上がりのみにする場合、0x14 を **0x04** にします。

調整ができたら、プロジェクト「kit07rec_03」をビルドして、「kit07rec_03.mot」ファイルを CPU ボードに書き込みます。

8.7 走行データのグラフ化

今回、スピードデータが取れたので、線グラフ化してみます。



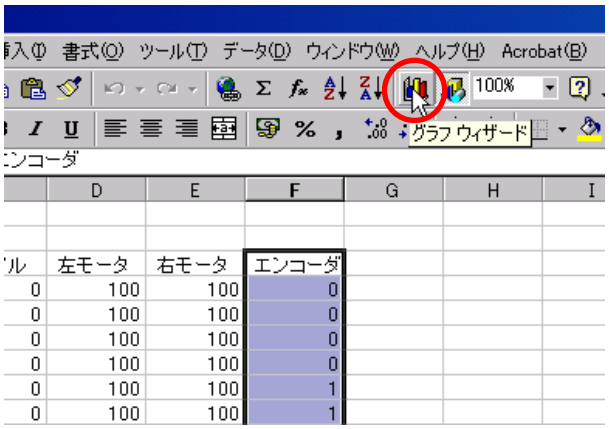
1. エクセルでデータを取り込みます。csv ファイルをダブルクリックするとエクセルで立ち上がります。

	A	B	C	D	E	F
1						
2	kit06rec_03	Data Out				
3	Pattern	Sensor	ハンドル	左モータ	右モータ	エンコーダ
4	11	00011000	0	100	100	0
5	11	00011000	0	100	100	0
6	11	00011000	0	100	100	0
7	11	00011000	0	100	100	0
8	11	00011000	0	100	100	1
9	11	00011000	0	100	100	1
10	11	00011000	0	100	100	1
11	11	00011000	0	100	100	1
12	11	00011000	0	100	100	2
13	11	00011000	0	100	100	1
14	11	00011000	0	100	100	3
15	11	00011000	0	100	100	2
16	11	00011000	0	100	100	2
17	11	00011000	0	100	100	3
18	11	00011000	0	100	100	3

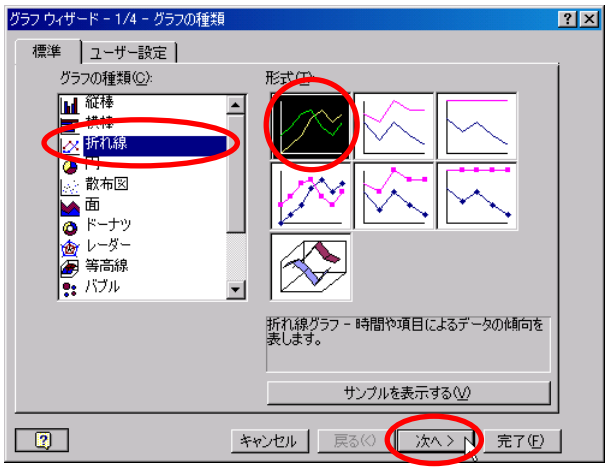
2. エクセルに取り込みました。

	A	B	C	D	E	F
1						
2	kit06rec_03	Data Out				
3	Pattern	Sensor	ハンドル	左モータ	右モータ	エンコーダ
4	11	00011000	0	100	100	0
5	11	00011000	0	100	100	0
6	11	00011000	0	100	100	0
7	11	00011000	0	100	100	0
8	11	00011000	0	100	100	1
9	11	00011000	0	100	100	1
10	11	00011000	0	100	100	1
11	11	00011000	0	100	100	1
12	11	00011000	0	100	100	2
13	11	00011000	0	100	100	1
14	11	00011000	0	100	100	3
15	11	00011000	0	100	100	2
16	11	00011000	0	100	100	2
17	11	00011000	0	100	100	3

3. エンコーダ値のセルを選択します。

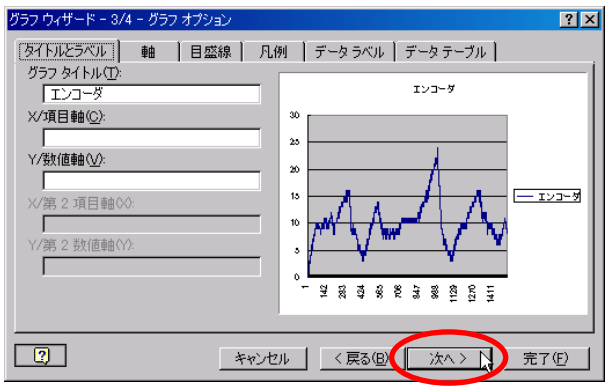


4. グラフ ウィザードを選択します。

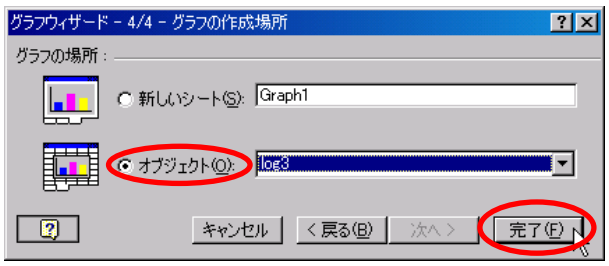


5. 折れ線グラフを選択します。次へをクリックします。

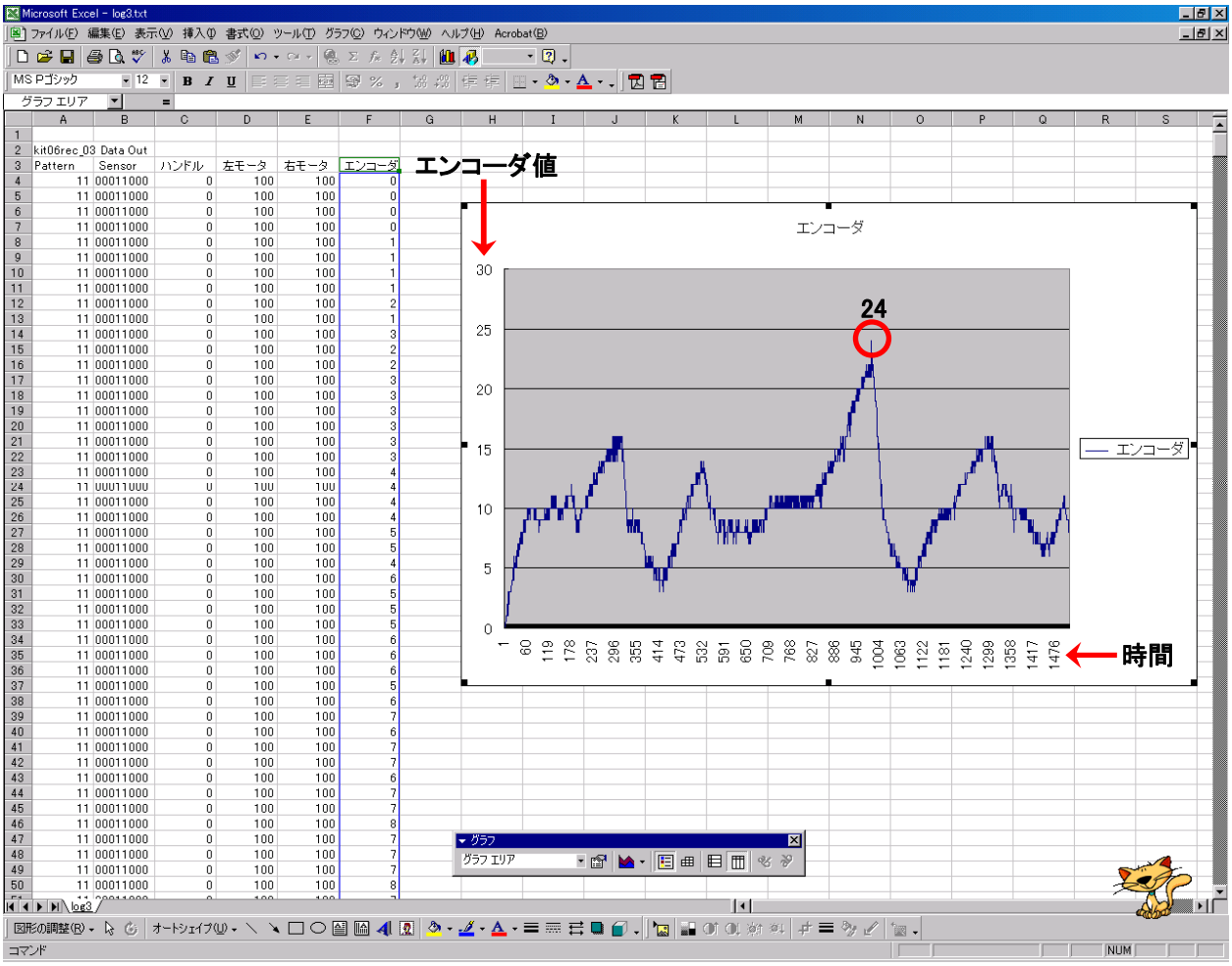
6. 次へをクリックします。



7. 各項目は、各自設定してください。特に設定しなくとも問題ありません。次へで進みます。



8. 何処にグラフを追加するか選択します。とりあえず、「オブジェクト」を選択して、完了で完了です。



エンコーダ値のグラフが追加されました。

x軸が時間です。10ms ごとにデータを取っているので、1当たり 10ms です。画面では、一番右が 1476 と表示されています。これは、スタートしてから 14760ms 後という意味です。

y 軸がスピードです。エンコーダ値が直接表示されています。

今回のエンコーダ値とスピードの関係は、「10.92 パルスで、1m/s」です。最速は 24 なので、

10.92:1＝24:最速のスピード

最速のスピード＝2.2m/s

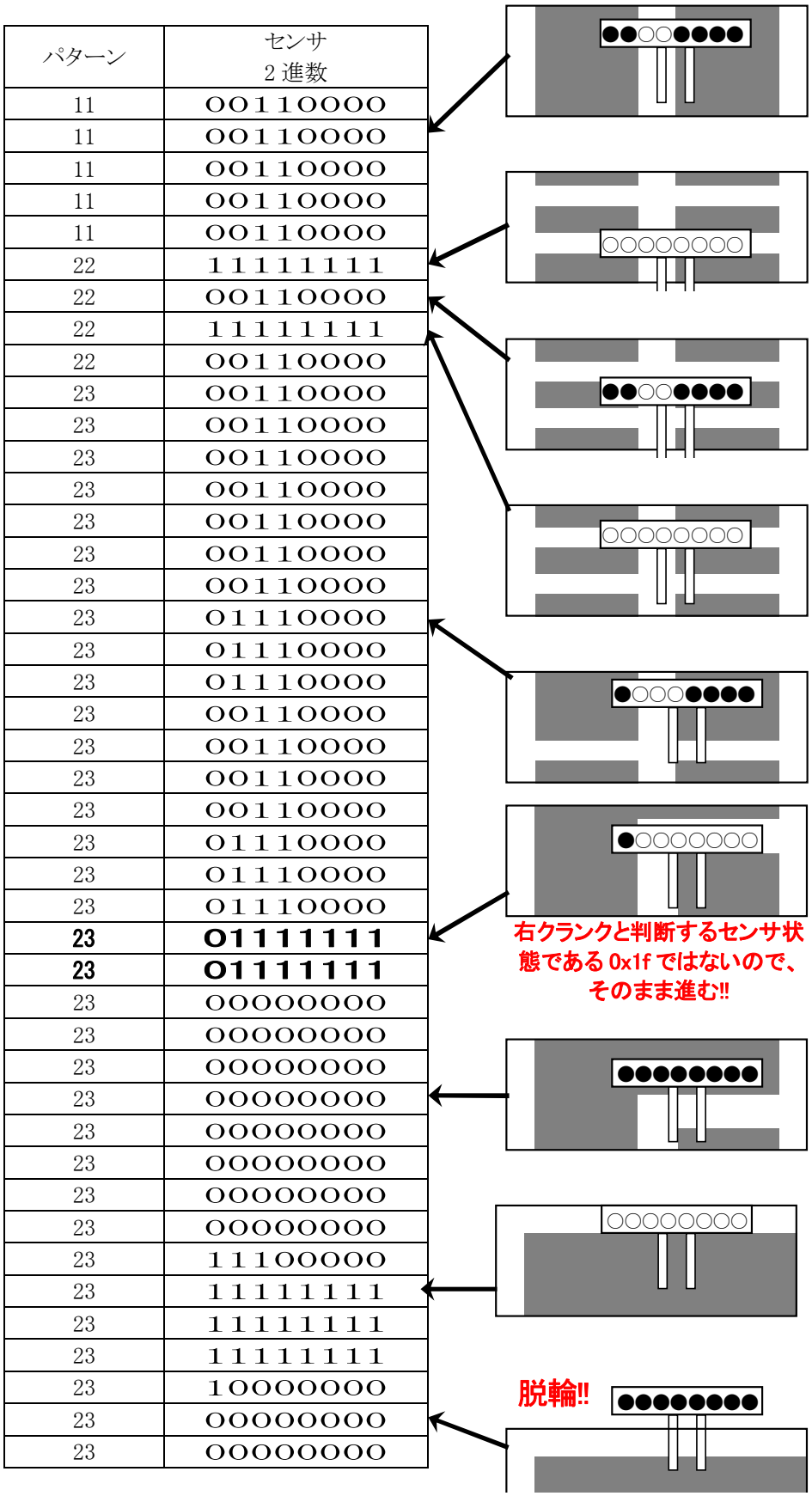
となります。その後、一気にスピードが落ちていますが、この部分は下り坂の後のクロスラインです。

例えば、カーブで脱輪したとします。このときのエンコーダ値を解析することにより、「そのスピード以上でカーブに進入したならブレーキをかけなさい」とプログラムすれば、脱輪を防ぐことができます。

9. データをエクセルで解析する

これは、実際にあったデータです。なぜか、直角部分をまっすぐ行ってしまい、脱輪してしまう現象が多発していました。そこで、データ取得して、解析してみました。

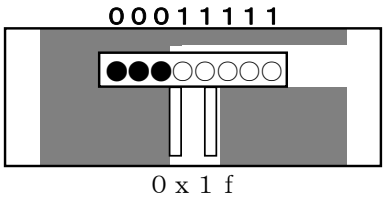
パターン	センサ 2 進数
11	OO11OOOO
11	OO11OOOO
11	OO11OOOO
11	OO11OOOO
11	OO11OOOO
22	11111111
22	OO11OOOO
22	11111111
22	OO11OOOO
23	OO11OOOO
23	OO11OOOO
23	OO11OOOO
23	OO11OOOO
23	OO11OOOO
23	OO11OOOO
23	OO11OOOO
23	O111OOOO
23	O111OOOO
23	O111OOOO
23	OO11OOOO
23	OO11OOOO
23	OO11OOOO
23	OO11OOOO
23	O111OOOO
23	O111OOOO
23	O111OOOO
23	**O1111111**
23	**O1111111**
23	OOOOOOOO
23	OOOOOOOO
23	OOOOOOOO
23	OOOOOOOO
23	OOOOOOOO
23	OOOOOOOO
23	OOOOOOOO
23	OOOOOOOO
23	OOOOOOOO
23	11100000
23	11111111
23	11111111
23	11111111
23	10000000
23	OOOOOOOO
23	OOOOOOOO



プログラムを見てみます。

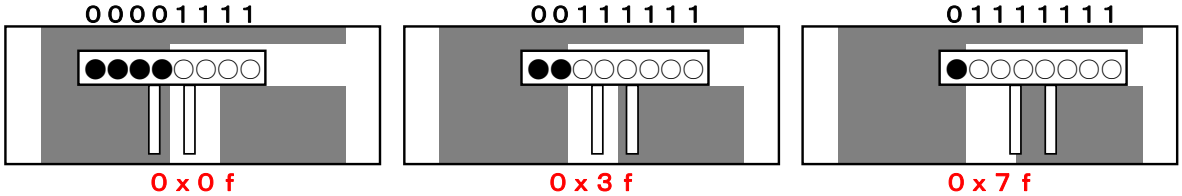
```
    case 23:
        /* クロスライン後のトレース、クランク検出 */
        if( sensor_inp(MASK4_4)==0xf8 ) {
            /* 左クランクと判断→左クランククリア処理へ */
            led_out( 0x1 );
            handle( -38 );
            speed( 10 ,50 );
            pattern = 31;
            cnt1 = 0;
            break;
        }
        if( sensor_inp(MASK4_4)==0x1f ) {
            /* 右クランクと判断→右クランククリア処理へ */
            led_out( 0x2 );
            handle( 38 );
            speed( 50 ,10 );
            pattern = 41;
            cnt1 = 0;
            break;
        }
        if( iEncoder >= 11 ) {       /* クロスライン後のスピード制御 */
            speed2( 0 ,0 );
        } else {
            speed2( 70 ,70 );
        }
        switch( sensor_inp(MASK3_3) ) {
            case 0x00:
                /* センター→まっすぐ */
                handle( 0 );
                break;
            case 0x04:
            case 0x06:
            case 0x07:
            case 0x03:
                /* 左寄り→右曲げ */
                handle( 8 );
                break;
            case 0x20:
            case 0x60:
            case 0xe0:
            case 0xc0:
                /* 右寄り→左曲げ */
                handle( -8 );
                break;
        }
        break;
```

センサ 8 つの状態が 0x1f でなければ右クランクとは見なしません(下図)。



０ｘ１ｆ

データ解析を何度も行うことにより、下図のような状態があることが分かりました。



０ｘ０ｆ　　０ｘ３ｆ　　０ｘ７ｆ

そこで、右クランクと判断するセンサの状態を 0x1f の他、0x0f、0x3f、0x7fも追加します。

```
void main( void )
{
    int     i;
    unsigned char b;                                    ローカル変数の追加

＝＝＝＝ 中略 ＝＝＝＝

    case 23:
        /* クロスライン後のトレース、クランク検出 */
        b = sensor_inp(MASK4_4);                        センサ値をいったんｂに保存
        if( b==0xf8 ) {
            /* 左クランクと判断→左クランククリア処理へ */
            led_out( 0x1 );
            handle( -38 );
            speed( 10 ,50 );
            pattern = 31;
            cnt1 = 0;
            break;
        }
        if( b==0x1f || b==0x0f || b==0x3f || b==0x7f ) {     右クランクと判断する状態を追加
            /* 右クランクと判断→右クランククリア処理へ */
            led_out( 0x2 );
            handle( 38 );
            speed( 50 ,10 );
            pattern = 41;
            cnt1 = 0;
            break;
        }
        if( iEncoder >= 11 ) {      /* クロスライン後のスピード制御 */
            speed2( 0 ,0 );
        } else {
            speed2( 70 ,70 );
        }
        switch( sensor_inp(MASK3_3) ) {
            case 0x00:
                /* センタ→まっすぐ */
                handle( 0 );
                break;
            case 0x04:
            case 0x06:
            case 0x07:
            case 0x03:
                /* 左寄り→右曲げ */
                handle( 8 );
                break;
            case 0x20:
            case 0x60:
            case 0xe0:
            case 0xc0:
                /* 右寄り→左曲げ */
                handle( -8 );
                break;
        }
        break;
```

この追加を行うことで、右クランクをクリアすることができました。

今回は、たまたま右クランクでセンサをチェックする状態が不足していましたが、左クランクもあり得ます。左クランクであり得るセンサの状態を自分で考えて、上記プログラムに追加してみてください。

10. 大容量EEP-ROM(24C1024)を使う

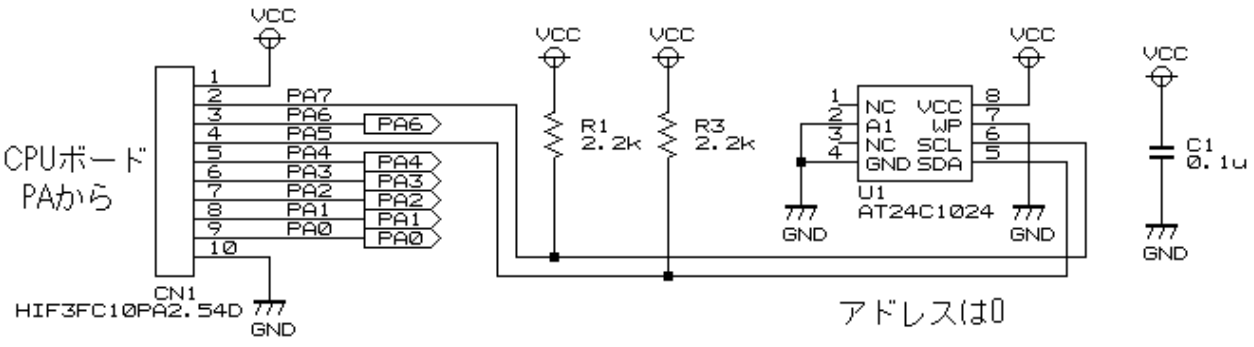
10.1 概要

ここまで、24C256 を使って説明してきました。メモリ容量の大きい 24C1024 という IC もあります。下記に違いをまとめます。

型式	24C256	24C1024
容量	32KB アドレスは 0～32767(0x7fff)	**128KB** アドレスは 0～131071 (0x1ffff)
同時 接続数	4 個まで 32KB×4＝128KB まで増設可能	2 個まで 128KB×2＝256KB まで増設可能
プロジェクトに追加 するファイル	i2c_eeprom.c	i2c_eeprom_1024.c
IC の ピンの違い		24C256 に対して、1 ピンが未接続 他は変更無し
特徴	**値段が手頃**	**メモリ容量が 24C256 に比べ 4 倍ある**

特徴のように、24C1024 は 24C256 に比べメモリ容量が 4 倍あるので、たくさんの情報を記録したいときに有効です。ただし、24C256 より値段が高いです。

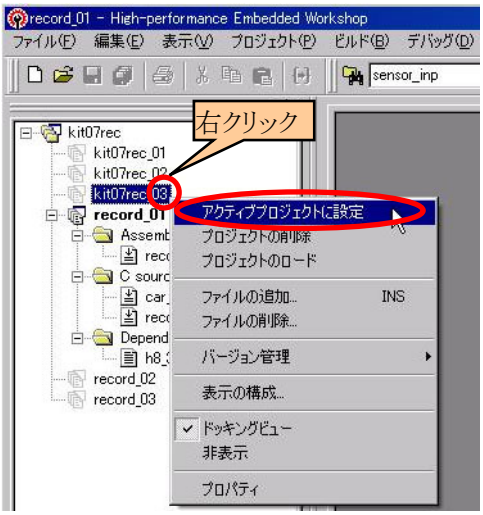
10.2 回路図



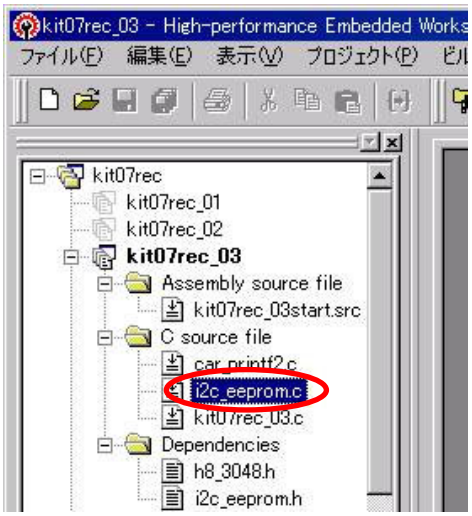
1 ピンが未接続になるだけで、他のピンの接続は変わりません。ソケットにしておけば、24C256 と 24C1024 は交換可能です。

10.3 プロジェクトへの登録方法

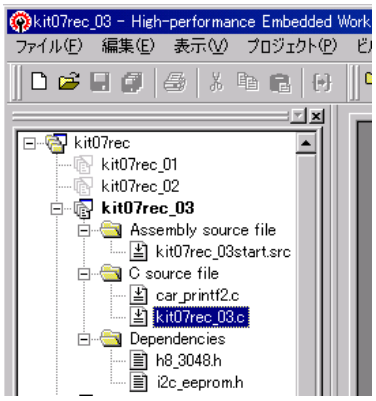
プロジェクト「kit07rec_03」を例に説明します。



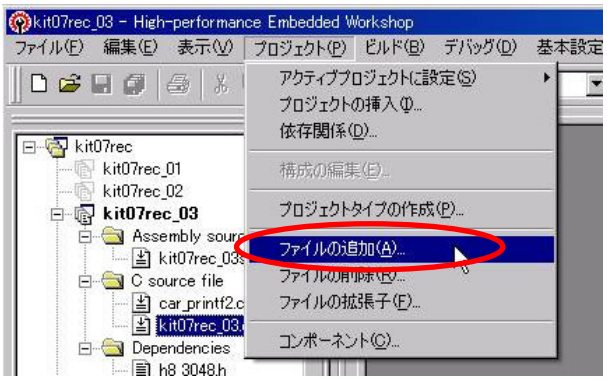
1. ワークスペース「kit07rec」を開きます。プロジェクト「kit07rec_03」をアクティブプロジェクトにします。



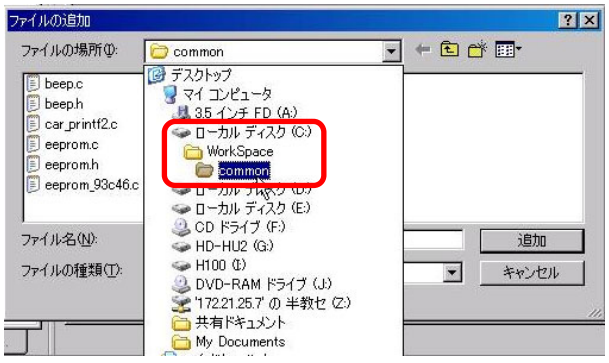
2. プロジェクト「kit07rec_03」の「**i2c_eeprom.c**」をクリックして、DEL キーを押します。



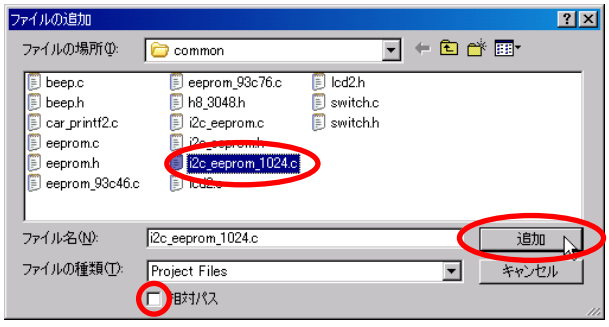
3. 「**i2c_eeprom.c**」が削除されました。



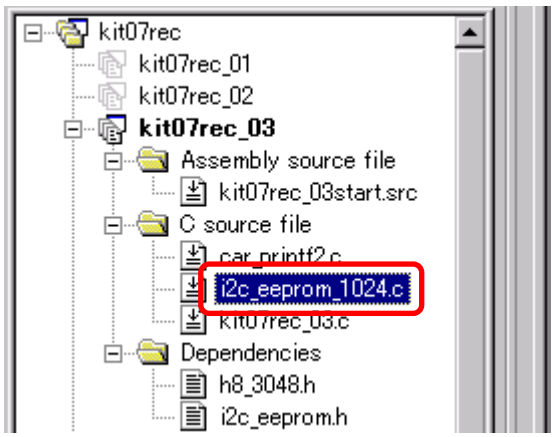
4. 「プロジェクト→ファイルの追加」を選択します。



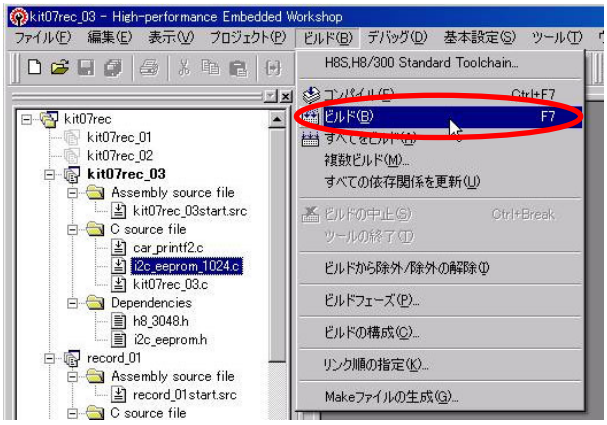
5. 「Cドライブ→Workspace→common」フォルダを選びます。



6. 「**i2c_eeprom_1024.c**」を選択します。「相対パス」のチェックは外します。追加をクリックします。



7.「i2c_eeprom_1024.c」が追加されました。



8.「ビルド→ビルド」でビルドすれば、24C1024 対応のプログラムの完成です。

※24C256 を接続しているときは「**i2c_eeprom.c**」、24C1024 を接続しているときは「**i2c_eeprom_1024.c**」を使用してください。互換はありません。

10.4 関数の変更点

initI2CEeprom	変更ありません。

selectI2C EepromAddress	どの EEP-ROM を使用するか選択します。
	24C1024 の 2 ピンに接続されている電圧により変わります。
	●2 ピンが 0V の 24C1024 を対象にしたい場合
	selectI2CEepromAddress(0);
	とします。最初はこの状態です。
	●2 ピンが 5V の 24C1024 を対象にしたい場合
	selectI2CEepromAddress(1);
	とします。

readI2CEeprom	アドレスが 0～0x1ffff になります。他は変わりません。 例)EEP-ROM の 0x1f000 番地のデータを読み込む場合 `i = readI2CEeprom( 0x1f000 );`
writeI2CEeprom	アドレスが 0～0x1ffff になります。他は変わりません。 例)EEP-ROM の 0x1f000 番地に、5 を書き込む場合 `writeI2CEeprom( 0x1f000, 5 );`
pageWrite I2CEeprom	アドレスが 0～0x1ffff になります。他は変わりません。
setPageWrite I2CEeprom	アドレスが 0～0x1ffff になります。他は変わりません。
I2CEeprom Process	変更ありません。
clearI2CEeprom	変更ありません。

11. 参考文献

・(株)ルネサス テクノロジ
H8/3048 シリーズ、H8/3048F-ZTAT™ (H8/3048F、H8/3048F-ONE)ハードウェアマニュアル 第 7 版
・(株)ルネサス テクノロジ 半導体トレーニングセンター C言語入門コーステキスト 第 1 版
・(株)オーム社 H8 マイコン完全マニュアル 藤澤幸穂著 第1版
・電波新聞社 マイコン入門講座 大須賀威彦著 第 1 版
・電波新聞社 C言語でH8マイコンを使いこなす 鹿取祐二著 第 1 版
・ソフトバンク(株) 新C言語入門シニア編 林晴比古著 初版
・共立出版(株) プログラマのための ANSI C 全書 L.Ammeraal 著
吉田敬一・竹内淑子・吉田恵美子訳 初版
・I2C バス仕様書バージョン 2.1
フィリップス社のホームページよりダウンロード可能
http://jp.semiconductors.philips.com/markets/mms/protocols/i2c/
http://www.semiconductors.philips.com/acrobat/literature/9398/39340011_jp.pdf
・AT24C256 のデータシート
ATMEL 社のホームページよりダウンロード可能
http://www.atmel.com/
http://www.atmel.com/dyn/resources/prod_documents/doc0670.pdf

マイコンカーラリーについての詳しい情報は、マイコンカーラリー公式ホームページをご覧ください。

http://www.mcr.gr.jp/

H8 マイコンについての詳しい情報は、(株)ルネサス テクノロジのホームページをご覧ください。

http://japan.renesas.com/

の「マイコン」→「H8 ファミリ」→「H8/3048B グループ」でご覧頂けます

※リンクは、2008 年 6 月現在の情報です。