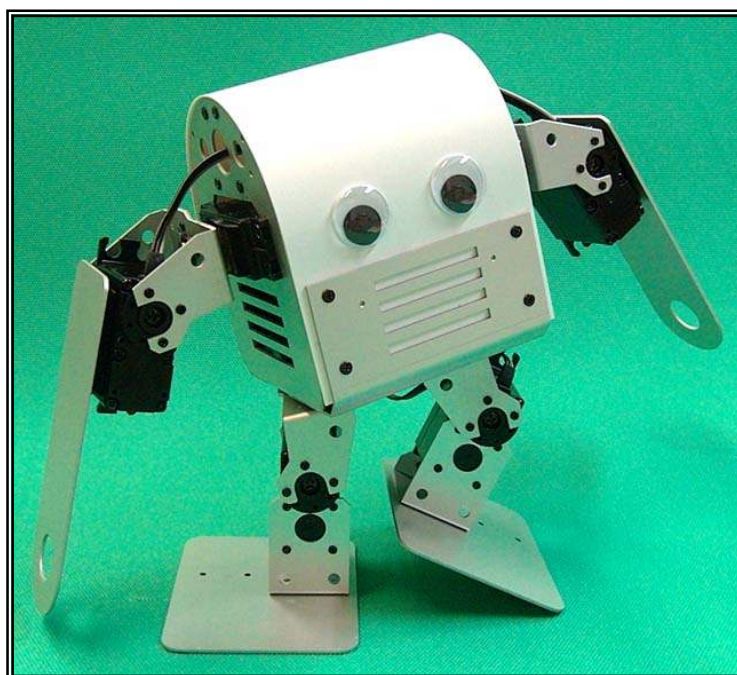


TK-3687mini オプション

“二足歩行ロボット事始め”

Version 091222



ほんの少し前まで人間のようには二本の足で歩くロボットはバーチャルな世界の中だけのことでした。しかし、ある有名企業が二本足のロボットを開発し歩く様子がコマーシャルなどで放映されてから、二足歩行ロボットはぐっと身近なものとなり、アマチュアが趣味で楽しむまでになりました。現在ではいくつかの企業がホビーとしての二足歩行ロボットキットを発売しています。このマニュアルでは 8 軸の二足歩行ロボットの、プログラムの設計や考え方を学習します。

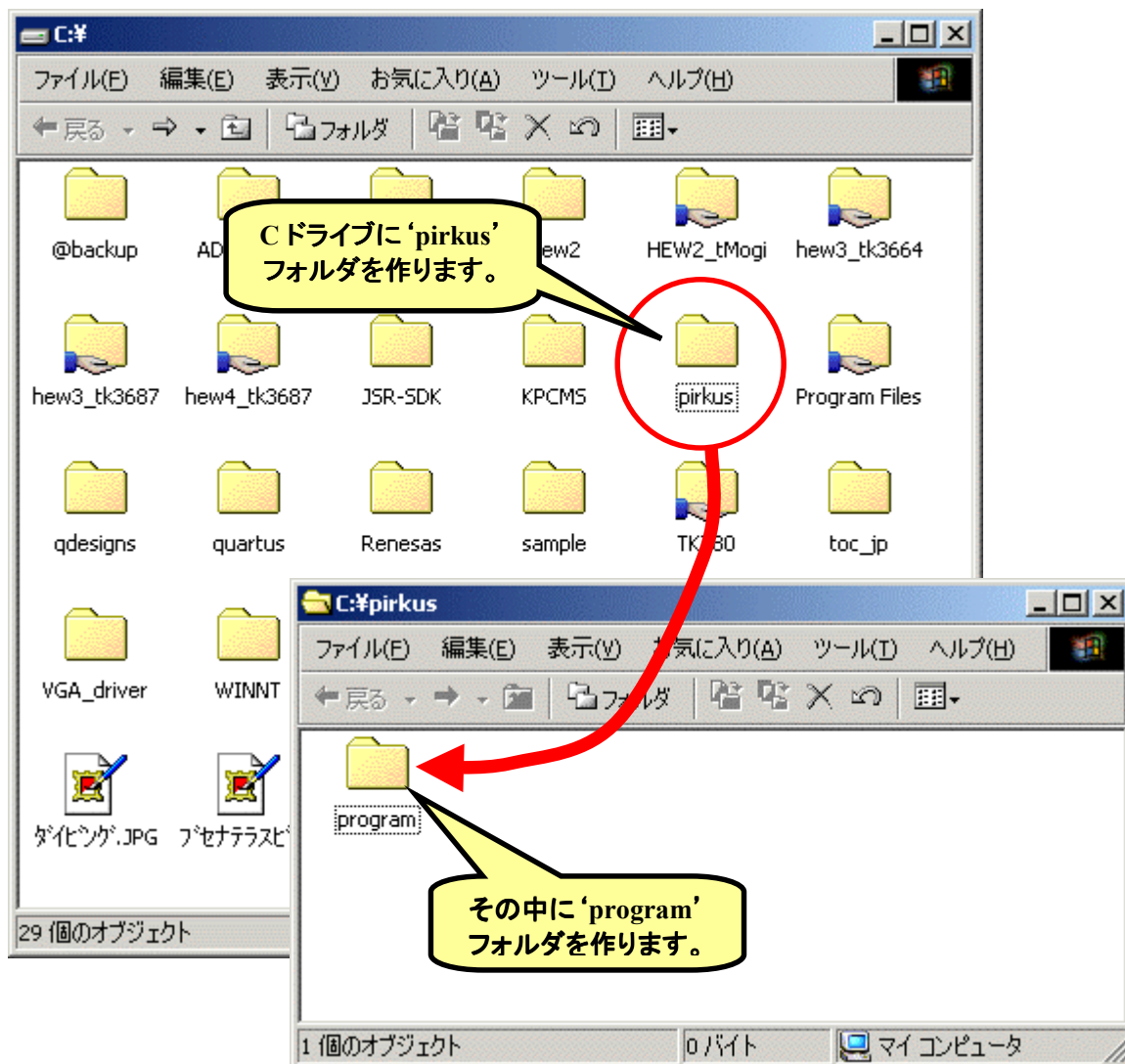
目次

1 “Pirkus・R Type-02”を組み立てよう	P. 2
2 RC サーボモータの制御	P. 6
3 ホームポジションを作る	P. 9
4 二足歩行にチャレンジ(スムージング無し)	P. 16
5 スムージング付き二足歩行	P. 24
6 パソコンで“Pirkus・R Type-02”を制御する(モーションエディタの使用)	P. 33
7 マイコンで“Pirkus・R Type-02”を制御する	P. 34
8 赤外線リモコンで“Pirkus・R Type-02”を制御する	P. 42
付録	P. 84

(株)東洋リンクス

学習を始める前に・・・

Pirkus 専用作業フォルダとして、C ドライブに 'pirkus' を作り、さらにその中に 'program' フォルダを作るか、製品付属の CD-R の 'pirkus' フォルダをパソコンの C ドライブにコピーしてください。このマニュアルのプロジェクトは全てこのフォルダに作成します。



このマニュアルの本文では、おもに二足歩行ロボットの制御について説明しています。TK-3687mini の回路図、ソフトウェアツール(ハイパーH8・HEW・FDT)のインストールや使い方については付録に掲載していますのでそちらをご覧ください。

次のページから“Pirkus・R Type-02”を組み立てますが、その前に開発環境を整えておきましょう。付録を参照しながら、ハイパーターミナルの設定、HEW のダウンロードとインストール、FDT のダウンロードとインストールを行なって下さい。それぞれの使い方は学習を進めていく中で覚えることにしましょう。

さあ、ここまでできたら準備完了です。それでは、“Pirkus・R Type-02”の組み立てから始めましょう。

- 写真はプロトタイプです。実際の製品とは異なる場合があります。
- 外観・定格・仕様・マニュアルの内容、プログラムの内容等は性能改善のため将来予告なく変更する場合があります。
- キットの組み立ての際に必要な工具等は付属しておりません。別途ご用意ください。
- TYPE-02 II にバージョンアップされました。基本的なメカニズムは変わっていないので、このマニュアルの内容はそのまま利用できます。詳しくは「_始めにお読みください」フォルダの「TYPE-02 II ご購入の方へ. pdf」をご覧ください。

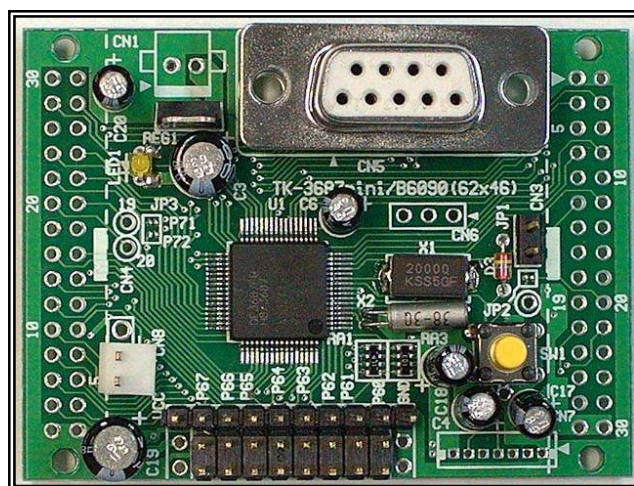
1 “Pirkus・R Type-02”を組み立てよう

今回使用するロボットキット，“Pirkus・R Type-02”（アイ・ビー株式会社）には、次のような特徴があります。



- 身長 …………… 約 17cm
- 重さ …………… 約 700g
- 関節数 …………… 8 関節
- アクチュエータ …… PRS-FF09P を 8 個

“Pirkus・R Type-02”にはコントロールボードは付属していないので、別途マイコンボードやロボット用コントローラを用意する必要があります。このマニュアルでは、マイコンボードとして TK-3687mini (Pirkus・R Type-02 版, 右写真参照)を使用します。



■ 組み立て

“Pirkus・R Type-02”の組み立ては、製品付属の CD-ROM に含まれている「組み立てマニュアル」の手順に従います。8 ページの「2. サーボモータ取り扱いの注意」まで、まずお読みください。

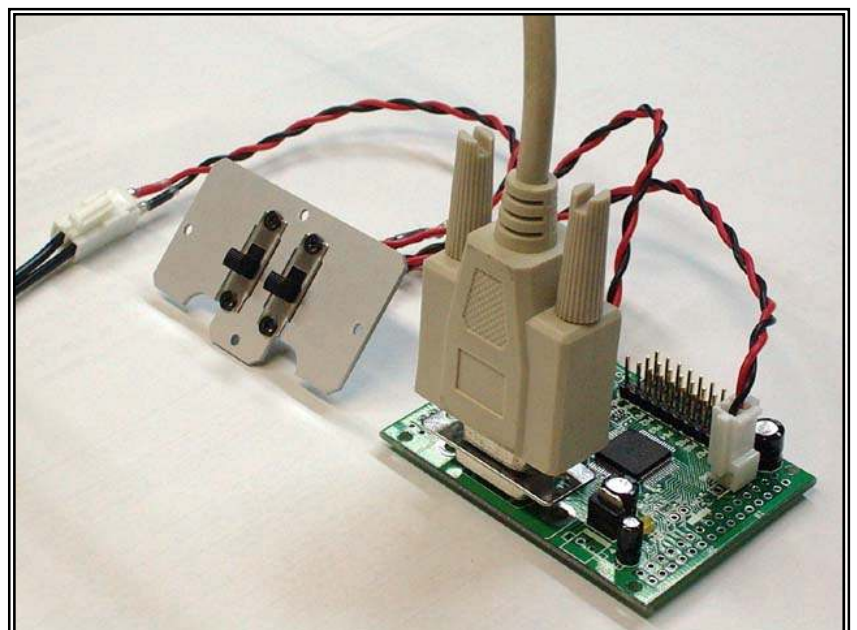
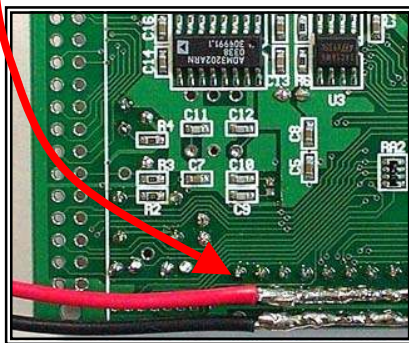
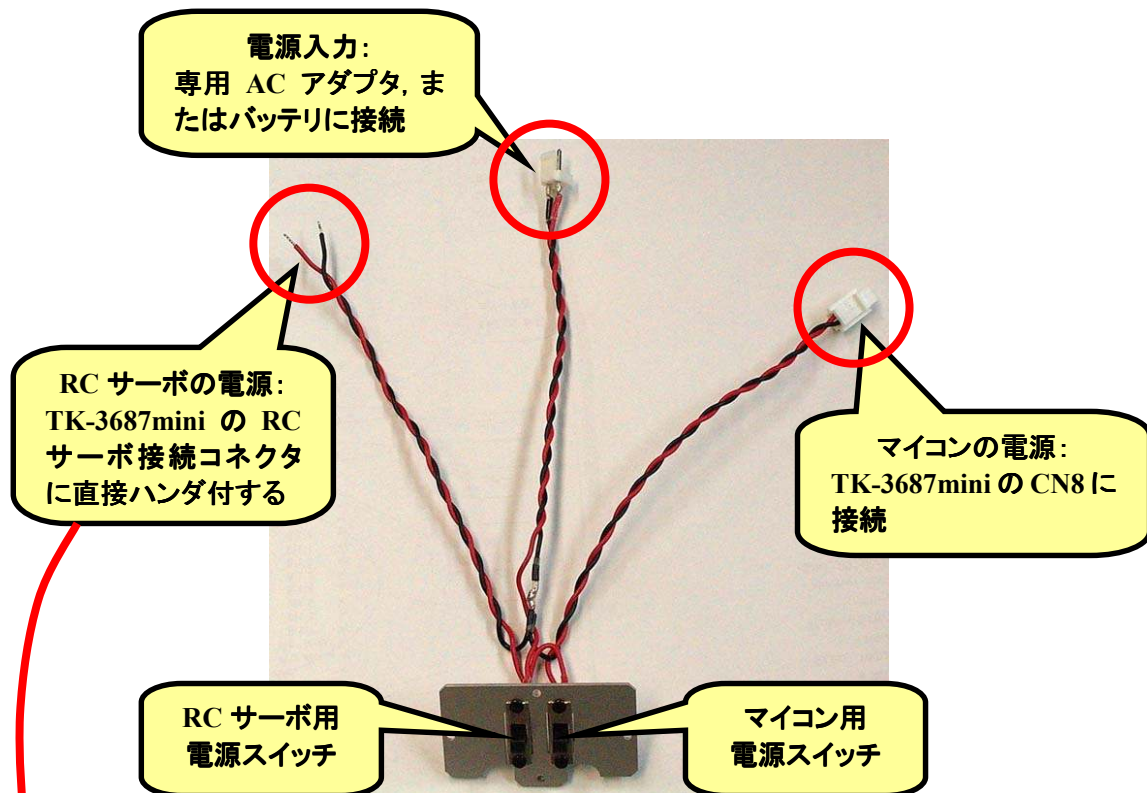
さて、「組み立てマニュアル」の 9 ページから足の組み立てに入りますが、その前に、ちょっと一手間かけて確認作業をしておきましょう。組み立てたあとの調整が楽になります。

ロボット制御で要になるのは位置指定です。もし、使用している RC サーボモータごとに位置の基準が異なっているとどうなるでしょうか。当然、位置指定が難しくなります。それで、同じ制御信号を入れたときに、だいたい同じ位置になるようにあらかじめ調整しておきます。

もっとも、RC サーボ自体は出荷時に調整されています。問題なのは RC サーボの軸に取り付けられているホーン（丸い円盤状の部品）の角度です。ほとんどは大丈夫なのですが、まれにずれているものがまぎれてしまうことがあります。確認し、ずれているときは調整します。



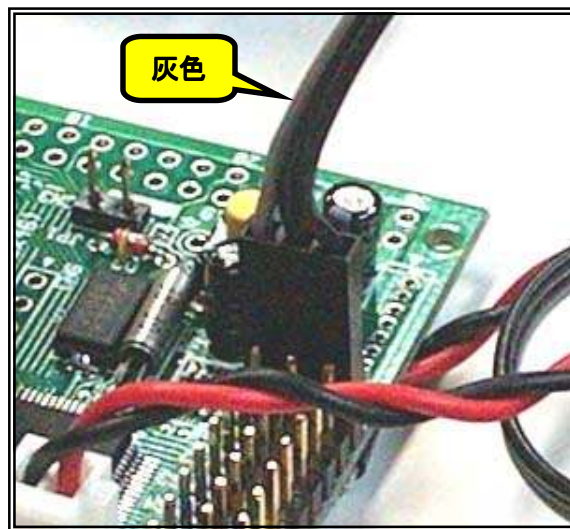
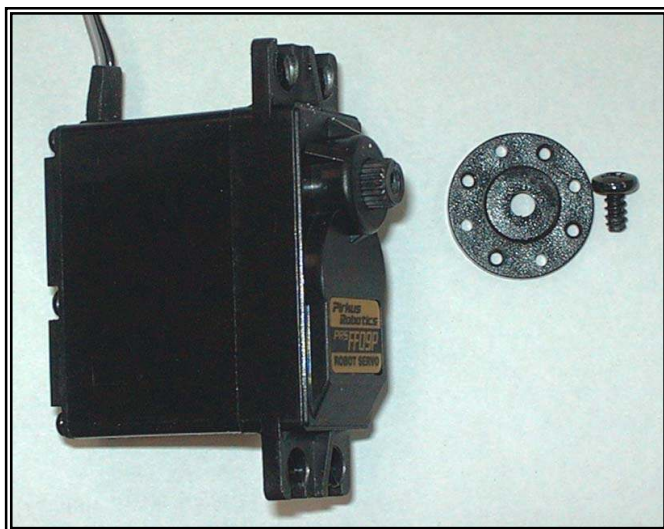
まずは制御信号を RC サーボに加えるため、TK-3687mini に電源ラインを仮配線します。“Pirkus・R Type-02”の中に下の写真のような電源スイッチと電源ラインの部品が入っています。この電源ラインを次のように TK-3687mini に接続し、パソコンとシリアルケーブルで接続します。



出荷時の TK-3687mini にはハイパーH8 が書き込まれています。パソコンのハイパーターミナルを起動し、マイコン用電源スイッチをオンするとハイパーH8 の画面が表示されます。‘Pirkus’フォルダの中から‘HomePos. mot’をダウンロード・実行してください。

このマニュアルの巻末付録の「ハイパーH8」に‘HomePos. mot’を実行するまでの詳しい手順が説明されています。

RC サーボのコネクタを TK-3687mini の P60 に接続します。このとき、**灰色のケーブルが基板の内側になる方向で接続**します。RC サーボ用電源スイッチをオンしましょう。RC サーボを中央位置にする信号が出力されます。この状態で、下の写真の状態に近い状態か確認してください(微調整はソフトで行なうので心配しないで下さい)。もし大きく違うなら、RC サーボからホーンをはずします(**注意:ネジを緩めるとき決して無理をしないこと!! 慎重に作業しましょう**)。次に最も近くなるようにホーンを取り付けます。最後にホーンをネジ止めします(**注意:決して無理をしないこと!! 締めすぎに注意しましょう**)。



全ての RC サーボモータについてこの作業を行ないます。終わったら、TK-3687mini の仮配線を外しておきましょう。

それでは、「組み立てマニュアル」の 9 ページ、「3. 組み立て ～足～」に戻り、順番に組み立てていきましょう。31 ページまで「組み立てマニュアル」どおり進んでください。



どうでしょうか。完成が見えてきましたか。さて、「組み立てマニュアル」の 32 ページ、「7. 組み立て ～腕と頭、目～」まできたら、ちょっとマニュアルとはちがうコースで進めていきます。

「組み立てマニュアル」では次にスイッチパネルにスイッチを取り付けて、スイッチパネルを胴体に取り付けることになっています。しかし、電源ラインの配線の関係上、スイッチパネルの取り付けは後回しにします。

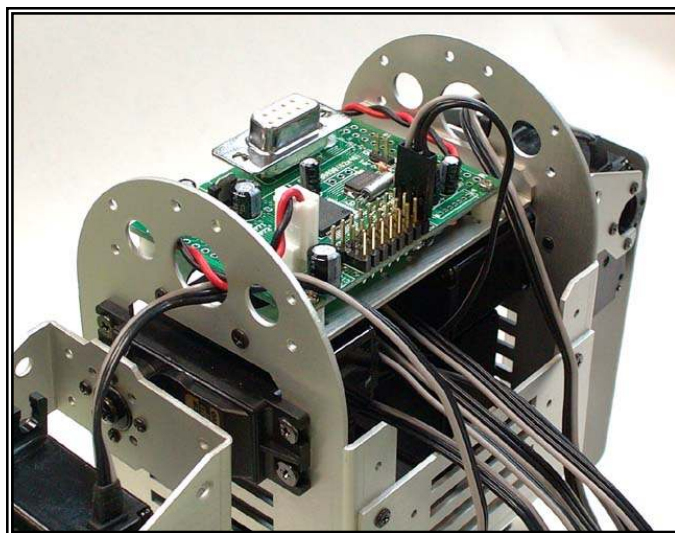
というわけで、次は 35 ページの腕の取り付けです。36 ページの基板ベースの取り付けまで済ませます。

ここでスイッチパネルを取り付けます。「組み立てマニュアル」の 34 ページを見て、スイッチパネルを取り付けてください。

次に、RC サーボの電源ラインを TK-3687mini の RC サーボ接続コネクタに直接ハンダ付けします。(このマニュアルの 3 ページの写真を参照)

ハンダ付けが終わったら、TK-3687mini を基板ベースに取り付けます。TK-3687mini に付属しているスペーサとネジで TK-3687mini を基板ベースに取り付けてください。

あとはマイコンの電源ラインを CN8 に、TK-3687mini と RC サーボモータを接続します。**RC サーボの灰色のラインが TK-3687mini の内側**になるように接続します。



TK-3687mini と RC サーボの接続は次のように対応させます。

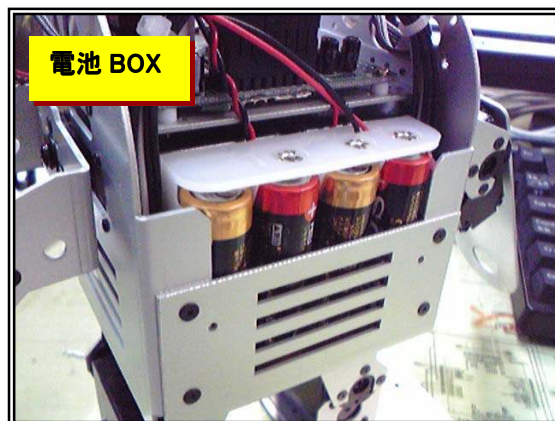
左				右			
膝	股	肩	腕	膝	股	肩	腕
P60	P61	P62	P63	P64	P65	P66	P67



ここまで来たら「組み立てマニュアル」の 37 ページからの手順に戻ります。【補強】を取り付けるところから始めて、最後まで組み立てます。ただし、デバッグ中は頭のプラスチック板を外していたほうが便利です。このマニュアルの写真は外した状態になっています。



これで組み立ては完成です。では、実際に動かす前に、ロボットの要、RC サーボモータの制御とプログラムについて調べてみましょう。

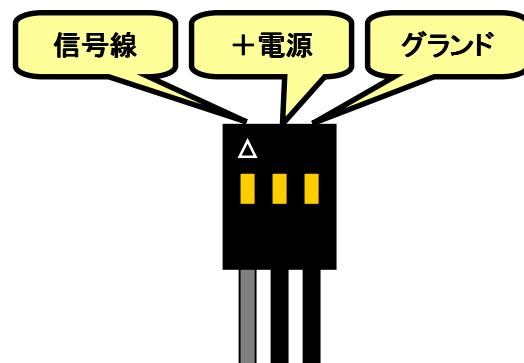


2 RC サーボモータの制御

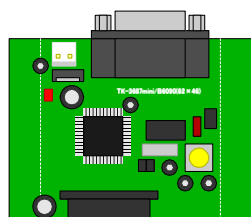
RC サーボモータは、もともとラジコン飛行機やラジコンカーなどのステアリングや補助翼といった位置制御に使われているホビー用のサーボモータです。二足歩行ロボットでは RC サーボモータを関節として利用します。もともと RC サーボはその名のおりラジコン用のため、二足歩行ロボットの関節として使うのは、メーカーが本来想定していた使い方ではありません。そのようなわけで最近ではロボット専用の RC サーボも開発・発売されるようになりました。

■ RC サーボモータの動かし方

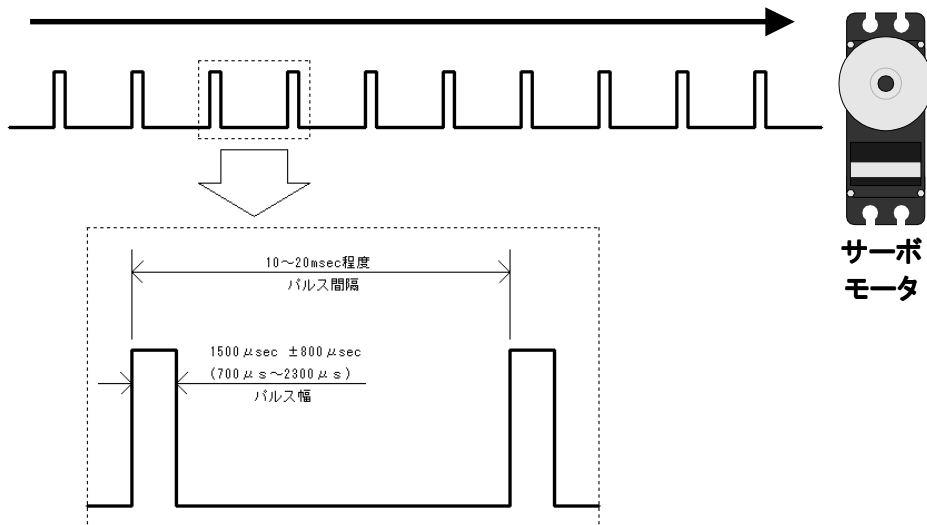
RC サーボはプラス電源とグラウンドのほかに 1 本のパルス信号を加えるだけで、稼動範囲 $\pm 60^\circ \sim \pm 90^\circ$ (メーカーや型番によって異なる) の位置制御を行なうことができます。一般的なピン配置は各メーカー共通で右のようになっています(ケーブルの色はメーカーによって異なる)。



RC サーボの位置制御は、信号線に 10~20ms 周期で $700 \mu s \sim 2300 \mu s$ (中心位置: $1500 \mu s$) の High パルスを加えることで、パルス幅に対応した位置にセットすることができます。パルスを加えるのを止めると(無信号にすると)RC サーボはパワーリダクション(脱力状態)になります。それで、特定の位置で固定する場合はパルスを加えつづける必要があります。



マイコン
(TK-3687mini)



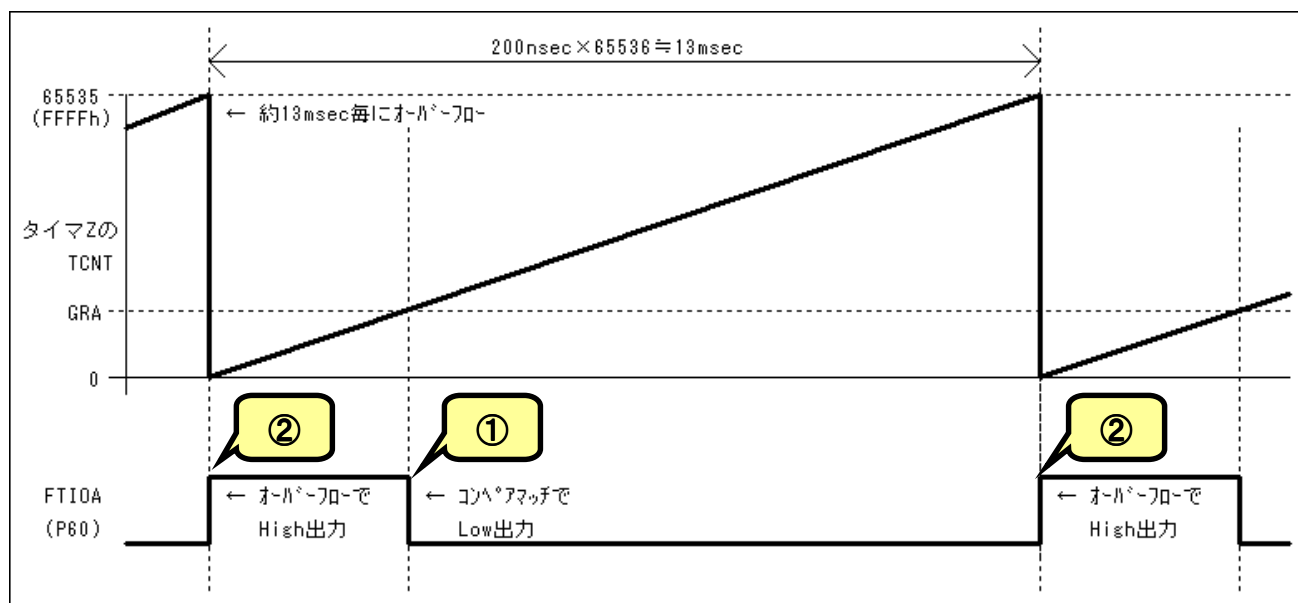
■ RC サーボモータ制御プログラム

RC サーボを制御するためにどのような信号が必要かわかりました。次は、TK-3687mini でその信号をどのように作るか考えてみましょう。

一つの方法はパルス周期もパルス幅もプログラムだけで管理する方法です。直感的でわかりやすいのですが、パルス周期やパルス幅を作る以外ができなくなります。パルス周期の方は結構アバウトなのでまだいいのですが、パルス幅の方は $8.9 \mu s$ 変化すると 1° 変化しますのでかなり正確に出力しなければなりません。 1° という小さな変化に思えるかもしれませんが、ロボットで 1° ちがうというのは相当大的な誤差になります。

そこで、TK-3687mini に実装されているマイコン、H8/3687 に内蔵されているタイマを使って、ハード的にパルス信号を出力することにします。タイマというのは一定の間隔でカウンタの値を+1していく機能です。H8/3687 には何種類かのタイマが内蔵されていますが、今回はタイマ Z を使います。

タイマ Z は 2 チャンネルの 16 ビットタイマです。1チャンネルにつき 4 種類の値(GRA, GRB, GRC, GED)と比較することができ(コンペアマッチ機能), その値と一致したら出力(FTIOA, FTIOB, FTIOC, FTIOD)を変化させます。下図をご覧ください。



斜めの線がタイマによって+1されるカウンタ(TCNT)の値, GRA とかかれた点線が比較する値をセットするレジスタ, FTIOAは出力される波形を表しています。FTIOAはP60と兼用ピンになっていますので, RC サーボへの出力波形は P60 に出てきます。

- ① タイマ Z によって TCNT がどんどん+1 されていくと, そのうち GRA と一致します(コンペアマッチ)。一致したら FTIOA を Low にします
- ② タイマ Z によって+1 されていく値には上限があつて, その上限になると 0 に戻ります。この状態をオーバーフローと言いますが, オーバーフローしたら FTIOA を High にします。

この①②を繰り返すことで, 一定の間隔でパルスを出力することができます。タイマ Z の TCNT は CPU クロックの 4 分周で+1 するよう設定します ($20\text{MHz} \div 4 = 5\text{MHz}$, よって 200ns 毎に+1 する)。16ビットカウンタということは, オーバーフローするまでに 65536 回カウントするので, $200\text{ns} \times 65536 = 13.1072\text{ms}$ がパルス周期になります。比較する値, GRA はカウント数で指定するので, 例えば中心位置にするために $1500\mu\text{s}$ のパルスを出したいときは, $1500\mu\text{s} \div 200\text{ns} = 7500$ を GRA にセットします。下表に角度, パルス幅, GRA セット値の関係を示します。

角度	パルス幅	GRA セット値
+90°	2300 μs	11500
+45°	1900 μs	9500
0°	1500 μs	7500
-45°	1100 μs	5500
-90°	700 μs	3500

ところで、①の FTIOA を Low にするのはタイマ Z が自動的にこなってくれるのですが、②の FTIOA を High にするのは自動的にこなしてくれません。そこで、割り込み機能を使います。TCNT がオーバーフローしたら割り込みがかかるように設定し、割り込みプログラムの中で FTIOA を High に設定します。

さて、図では GRA だけ考えましたが、GRB, GRC, GRD も全く同じです。さらに同じ機能がもう 1 チャンネルあるので、合計 8 個の出力を制御することができます。“Pirkus・R Type-02”は 8 個の RC サーボを使うので、TK-3687mini はちょうどいいですね。

なお、タイマ Z はほかにもいろいろな機能を持っています。タイマ Z の詳細について知りたい方はルネサステクノロジの「H8/3687 シリーズ ハードウェアマニュアル」, 「13. タイマ Z」をご覧ください。

3 ホームポジションを作る

早速「二足歩行にチャレンジ！」といきたいのですが、その前にしなければいけないことがあります。それが、「ホームポジションを作る」ということです。

これまで、RC サーボは信号線に 10~20ms 周期で 700~2300 μ s の High パルスを加えるとパルス幅に対応した位置にセットすることができる、中心位置にセットするときは 1500 μ s の High パルスを加える、と説明してきました。その説明自体は間違っていないです。しかし、RC サーボが複雑な機能を内蔵した機構部品である以上、どうしても個体差が生じ、同じメーカーの同じ型番の RC サーボに同じパルスを加えても位置が微妙に違う場合があります。組み立てる前に RC サーボのホーンの取り付けを調整しましたが、同じ信号を入力しているはずなのに多少のずれがあったと思います。そのため、RC サーボが壊れたので交換したら動きがおかしくなった、ということもあります。

また、“Pirkus・R Type-02”のフレーム自体の精度や RC サーボの取付精度の問題もあります。1 個の RC サーボなら誤差範囲で済んだものが、複数個の RC サーボを組み合わせて使うとなると問題が生じることがあります。

そこで、このずれを考慮した上で RC サーボに加えるパルス幅を指定しなければなりません。例えば、中心位置にするときのカウント値が 7500 の RC サーボを使い、中心位置からプラス 200 カウントの位置にする場合、タイマ Z の GRA に次のようにセットします。

```
TZO. GRA =7700; //P60, 左膝
```

さて、この RC サーボが壊れて別の RC サーボに交換したところ今までとは違う位置になりました。交換した RC サーボは中心位置にするときのカウント値が 7450 だったからです。わずか 50 カウントなのでパルス幅は 10 μ s 違うだけですが、角度にすると 1. 125° 異なりロボット全体としては大きな影響になります。それで、以前と同じ位置にするときは、

```
TZO. GRA =7650; //P60, 左膝
```

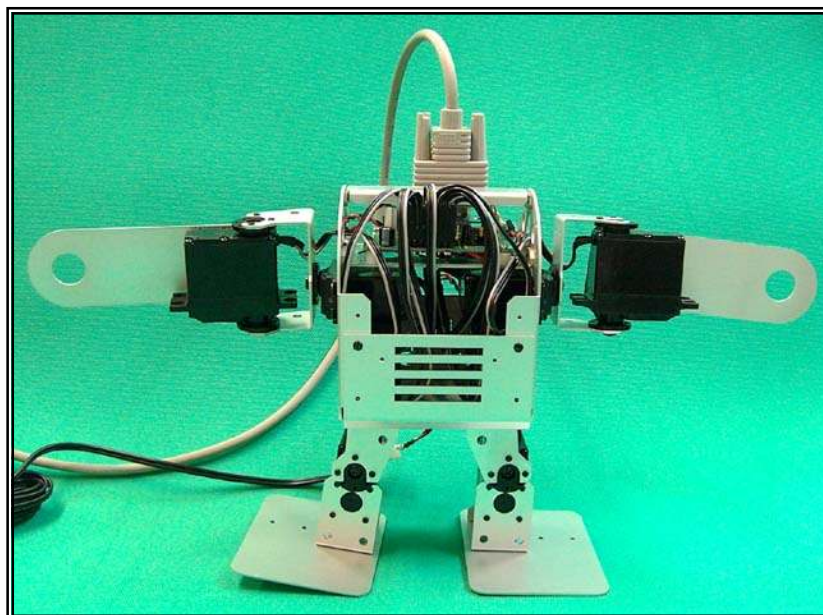
にしなければなりません。ところで、今のように一ヶ所だけならまだよいのですが、これが何ヶ所も変更しなければいけないとしたらどうでしょうか。また、常に中心位置のカウント値を意識しながらタイマ Z のカウント値を設定するのも大変です。

そこで、中心位置のずれをプログラムで修正することにします。具体的には中心位置を ‘HomePos[]’ という配列にあらかじめセットしておき (RC サーボを 8 個使うので)、位置の指定は中心位置からの相対値で指定するようにします。例えば、中心位置にするときのカウント値が 7450 の RC サーボを使い、中心位置からプラス 200 カウントの位置にする場合、タイマ Z の GRA に次のようにセットします。

```
HomePos[0] = 7450;  
.  
.  
.  
TZO. GRA =HomePos[0] + 200; //P60, 左膝
```

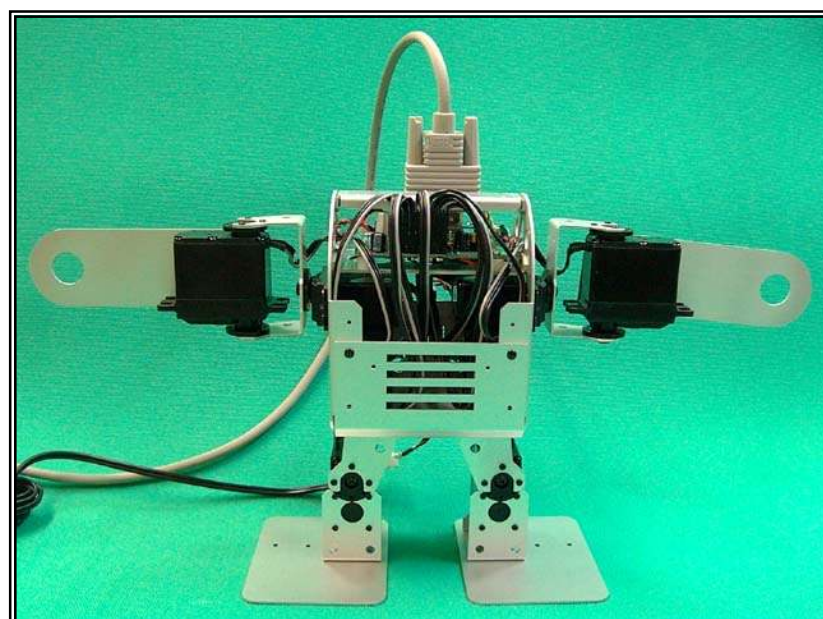
こうしておけば、RC サーボを交換したときは ‘HomePos[]’ の値を変更するだけでよいことになります。

ではここで、ホームポジションの調整をしない場合とする場合を比較してみましょう。次の写真はホームポジションの調整をせず、全ての RC サーボに $1500\mu\text{s}$ (カウント値 7500) のパルスを加えたときの筆者が作成した“Pirkus・R Type-02”の状態です。(デバッグ中なので顔ははずしています)



思ったより大きなずれですよね。ではホームポジションを調整したときの状態も見てください。筆者が作成した“Pirkus・R Type-02”の場合は次のようなカウント値をセットしました。

```
unsigned int HomePos[8] = {7400 //P60, 左膝  
, 7500 //P61, 左股  
, 7350 //P62, 左肩  
, 7500 //P63, 左腕  
, 7250 //P64, 右膝  
, 7650 //P65, 右股  
, 7300 //P66, 右肩  
, 7750 //P67, 右腕  
};
```



それでは、皆さんが作成した“Pirkus・R Type-02”のホームポジションも調整しましょう。まずは、

c:¥pirkus¥program¥HomePos¥HomePos. hws

をダブルクリックして HEW を起動しましょう。

次にハイパーターミナルを起動します。TK-3687mini にはハイパーH8 が書き込まれているはずなので(組み立てキットの場合は付録を見てハイパーH8 を書き込んでください)、パソコンのシリアルポートとTK-3687mini をつないでTK-3687mini の電源をオンするとハイパーH8 の画面が出てきます。‘L’コマンドを使って次のファイルをダウンロードします。

c:¥pirkus¥program¥HomePos¥HomePos¥Debug¥HomePos. mot

その後、‘G’コマンドでスタートするとRCサーボがカウント値に対応した位置になります。このときRCサーボが中心位置になるように‘HomePos[]’の値を少しずつ変更、ビルド、ダウンロード、実行してみてください。

ソースリストは次のとおりです。ファイル名は‘HomePos. c’です。

```

/*****/
/*                                          */
/* FILE      :HomePos. c                    */
/* DATE      :Fri, Feb 03, 2006            */
/* DESCRIPTION :Main Program              */
/* CPU TYPE   :H8/3687                     */
/*                                          */
/* This file is programed by TOYO-LINX Co.,Ltd. / yKikuchi */
/*                                          */
/*****/

/*****
サーボモータのホームポジションのカウント値をチェックする。
-----
サーボモータは 1500 μs で中心位置になるが機械的誤差が生じる。このプログラムは中心位置でパルスを出しつづけるので、実際の中心位置になるようカウント値(HomePos)を調整してプログラム作成に役立てる。
なお、1500 μs のときのカウント値は、
    1500 μs × 5 = 7500
になる。
*****/

/*****
履歴
-----
2006-02-03 : プラグラム開始
*****/

/*****
インクルードファイル
*****/
#include <machine.h> //H8 特有の命令を使う
#include "iodefine.h" //内蔵 I/O のラベル定義

/*****
定数の定義 (直接指定)
*****/
```

```

#define OK 0 //戻り値
#define NG -1 //戻り値

/*****
定数エリアの定義 (ROM)
*****/

/*****
グローバル変数の定義とイニシャライズ (RAM)
*****/
// サーボモータの中心 (カウント値)
// 理論上の中心位置は 1500  $\mu$ s, カウント値は 7500 (1 カウント=200ns)
unsigned int HomePos[8] = {7500 //P60, 左膝
, 7500 //P61, 左股
, 7500 //P62, 左肩
, 7500 //P63, 左腕
, 7500 //P64, 右膝
, 7500 //P65, 右股
, 7500 //P66, 右肩
, 7500 //P67, 右腕
};

```

全ての RC サーボが中心位置になるように、この値をカット&トライで調整する。

```

/*****
関数の定義
*****/
void init_tmz(void);
void intprog_tmz0(void);
void main(void);

/*****
メインプログラム
*****/
void main(void)
{
    int i, j;

    // イニシャライズ -----
    init_tmz();

    // メインループ -----
    while(1) {}
}

/*****
タイマZ イニシャライズ
*****/
void init_tmz(void)
{
    TZ.TSTR.BYTE = 0x00; //TCNT0, 1 停止

    TZ0.TCR.BYTE = 0xe2; //同期クリア,  $\phi/4$ 
    TZ1.TCR.BYTE = 0xe2; //同期クリア,  $\phi/4$ 

    TZ.TMDR.BYTE = 0x0f; //TCNT0, 1 は同期動作
}

```

```

TZ. TOCR. BYTE = 0xff; //初期出力=1

TZ. TOER. BYTE = 0x00; //出力端子イネーブル

TZ0. TIORA. BYTE = 0x99; //GRA, GRB はコンペアマッチで 0 出力
TZ0. TIORC. BYTE = 0x99; //GRC, GRD はコンペアマッチで 0 出力
TZ1. TIORA. BYTE = 0x99; //GRA, GRB はコンペアマッチで 0 出力
TZ1. TIORC. BYTE = 0x99; //GRC, GRD はコンペアマッチで 0 出力

TZ0. TSR. BYTE = 0x00; //割込みフラグクリア
TZ1. TSR. BYTE = 0x00; //割込みフラグクリア

TZ0. TIER. BYTE = 0x10; //オーバーフローインターラプトイネーブル
TZ1. TIER. BYTE = 0x00; //インターラプトディセーブル

TZ0. GRA = HomePos[0]; //カウント初期値
TZ0. GRB = HomePos[1]; //カウント初期値
TZ0. GRC = HomePos[2]; //カウント初期値
TZ0. GRD = HomePos[3]; //カウント初期値
TZ1. GRA = HomePos[4]; //カウント初期値
TZ1. GRB = HomePos[5]; //カウント初期値
TZ1. GRC = HomePos[6]; //カウント初期値
TZ1. GRD = HomePos[7]; //カウント初期値

TZ0. TCNT = 0x0000; //TCNT0=0
TZ1. TCNT = 0x0000; //TCNT1=0

TZ. TSTR. BYTE = 0x03; //TCNT0, 1 カウントスタート
}

/*****
 タイマ Z チャンネル 0 割込み
 *****/
#pragma regsave (intprog_tmz0)
void intprog_tmz0(void)
{
    //タイマ Z オーバフローインターラプトフラグ クリア
    TZ0. TSR. BIT. OVF =0;

    //出力を 1 にする
    TZ. TOCR. BYTE = 0xff;
}

```

このプログラムはタイマZのオーバーフロー割込みを使っています。それで、‘intprg. c’をデフォルトから次のように変更しています。

```

/*****
/*
/* FILE : intprg. c
/* DATE : Fri, Feb 03, 2006
/* DESCRIPTION : Interrupt Program
/* CPU TYPE : H8/3687
/*
/* This file is generated by Renesas Project Generator (Ver. 4. 0).
*/

```



```

/*
/*****

#include <machine.h>

extern void intprog_tmz0(void);

#pragma section IntPRG
// vector 1 Reserved

// vector 2 Reserved

// vector 3 Reserved

// vector 4 Reserved

// vector 5 Reserved

// vector 6 Reserved

// vector 7 NMI
__interrupt(vect=7) void INT_NMI(void) { /* sleep(); */}
// vector 8 TRAP #0
__interrupt(vect=8) void INT_TRAP0(void) { /* sleep(); */}
// vector 9 TRAP #1
__interrupt(vect=9) void INT_TRAP1(void) { /* sleep(); */}
// vector 10 TRAP #2
__interrupt(vect=10) void INT_TRAP2(void) { /* sleep(); */}
// vector 11 TRAP #3
__interrupt(vect=11) void INT_TRAP3(void) { /* sleep(); */}
// vector 12 Address break
__interrupt(vect=12) void INT_ABRK(void) { /* sleep(); */}
// vector 13 SLEEP
__interrupt(vect=13) void INT_SLEEP(void) { /* sleep(); */}
// vector 14 IRQ0
__interrupt(vect=14) void INT_IRQ0(void) { /* sleep(); */}
// vector 15 IRQ1
__interrupt(vect=15) void INT_IRQ1(void) { /* sleep(); */}
// vector 16 IRQ2
__interrupt(vect=16) void INT_IRQ2(void) { /* sleep(); */}
// vector 17 IRQ3
__interrupt(vect=17) void INT_IRQ3(void) { /* sleep(); */}
// vector 18 WKP
__interrupt(vect=18) void INT_WKP(void) { /* sleep(); */}
// vector 19 RTC
__interrupt(vect=19) void INT_RTC(void) { /* sleep(); */}
// vector 20 Reserved

// vector 21 Reserved

// vector 22 Timer V
__interrupt(vect=22) void INT_TimerV(void) { /* sleep(); */}
// vector 23 SC13

```

追加

```
__interrupt(vect=23) void INT_SCI3(void) { /* sleep(); */  
// vector 24 IIC2  
__interrupt(vect=24) void INT_IIC2(void) { /* sleep(); */  
// vector 25 ADI  
__interrupt(vect=25) void INT_ADI(void) { /* sleep(); */  
// vector 26 Timer Z0  
__interrupt(vect=26) void INT_TimerZ0(void) {intprog_tmz0();}  
// vector 27 Timer Z1  
__interrupt(vect=27) void INT_TimerZ1(void) { /* sleep(); */  
// vector 28 Reserved  
  
// vector 29 Timer B1  
__interrupt(vect=29) void INT_TimerB1(void) { /* sleep(); */  
// vector 30 Reserved  
  
// vector 31 Reserved  
  
// vector 32 SCI3_2  
__interrupt(vect=32) void INT_SCI3_2(void) { /* sleep(); */
```

変更

これで、RC サーボの基準設定が完了しました。ここで得られた値を次章以降も使用します。では、次の章で文字通り第一歩を踏み出しましょう。

4 二足歩行にチャレンジ(スージング無し)

ロボットの歩行には次の二種類があります。

静歩行:倒れないよう重心を常に足裏に置いたまま足を前に出す。

動歩行:重心は足裏にない。倒れようとする力で前に進む。(人間の歩行パターン)

すばやく歩けるのは動歩行ですが、さすがに人間の歩行パターンをまねるだけあって制御は非常に難しいです。また、“Pirkus・R Type-02”の足の関節数は片足2つ、両足で4つしかない、という制限もあります。それで、このマニュアルでは静歩行にチャレンジします。ゆっくりとしか歩けませんが、ある意味ロボットらしい動きです。

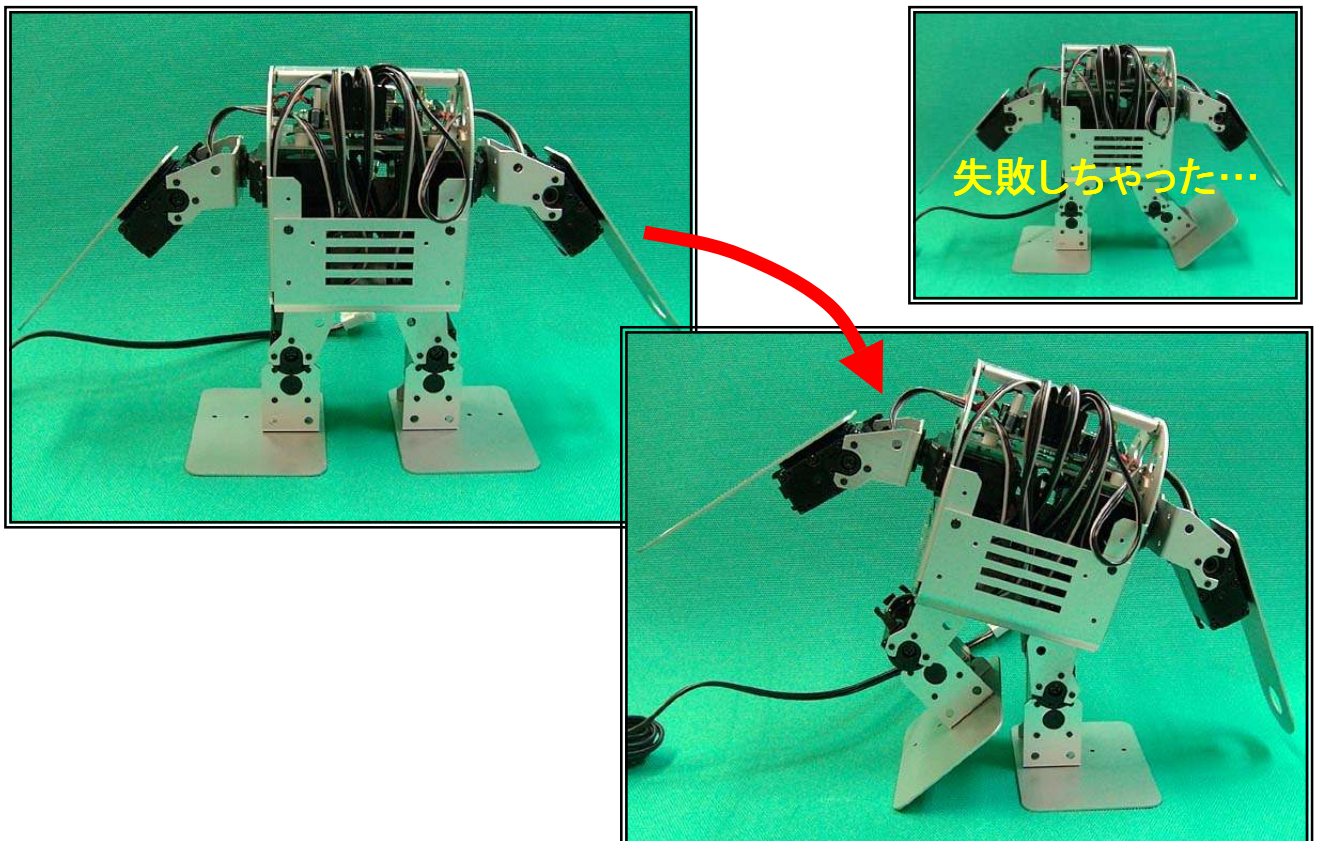
静歩行の動作を分解すると、

1. 重心を片足に移動する。
2. 浮いた足を前に出す。
3. 重心を元に戻す。

です。これを左右交互に行ない、繰り返していけば、二足歩行が完成します。

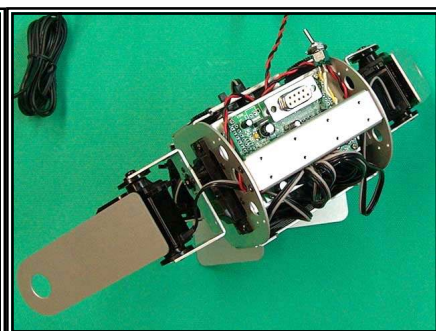
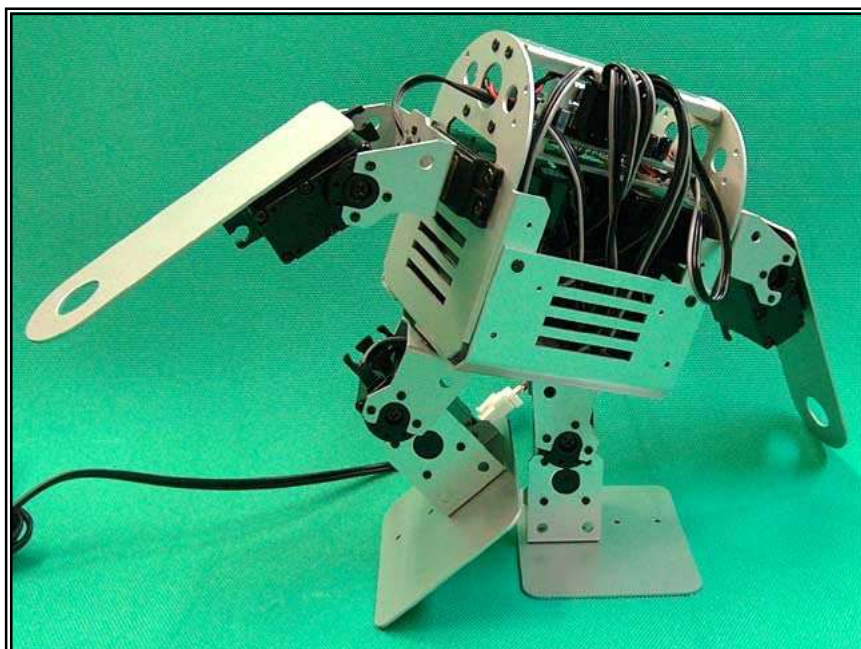
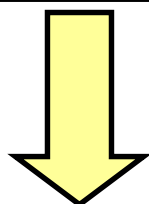
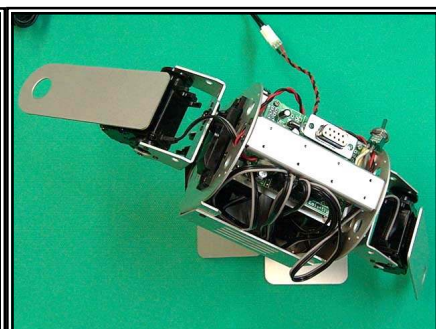
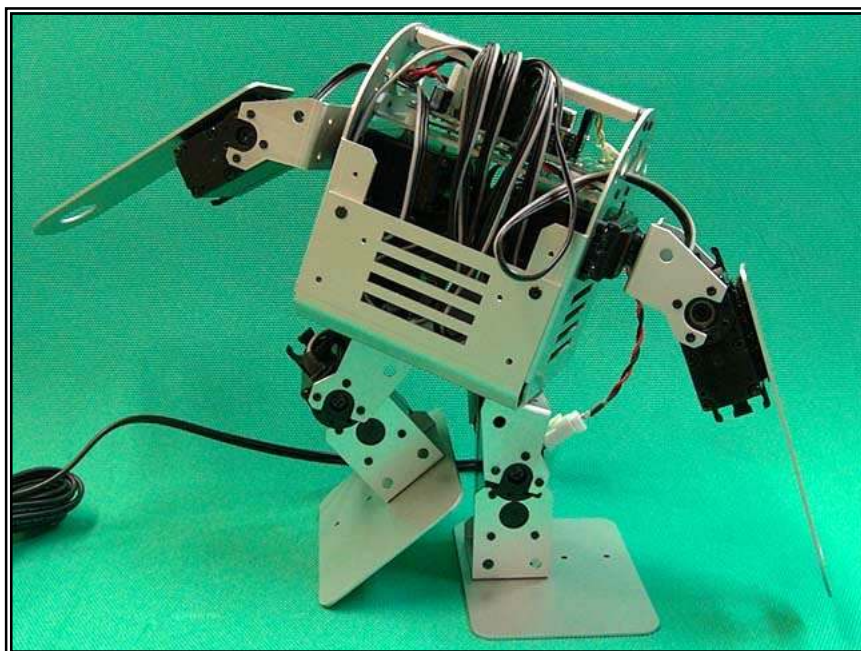
■ 重心を片足に移動する

まずは左足に重心を移動することを考えてみましょう。単純に考えると左足の膝を曲げればよさそうなのですが、それだけでは重心は移動してくれません。右足を伸ばして体を左側に持ち上げる必要があります。ここで、足裏の形が意味を持ってきます。外側に幅広になっているので、この部分を利用して足の長さを伸ばすことができます。下の写真のように膝の RC サーボを設定します。



■ 浮いた足を前に出す

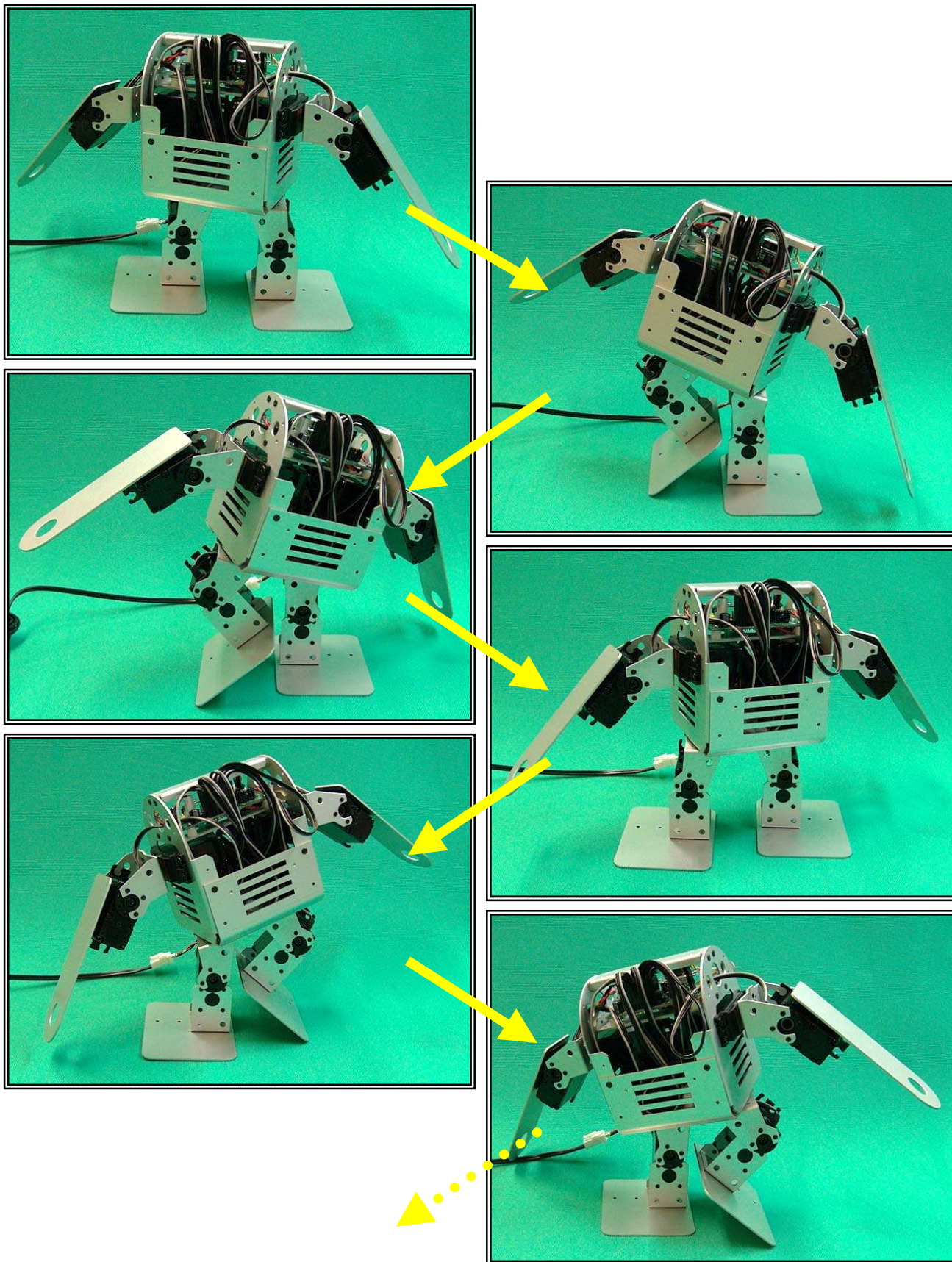
重心は左側にかかっていますから、この状態で両股の RC サーボを同じ方向に回すと浮いている足が前に出ます。



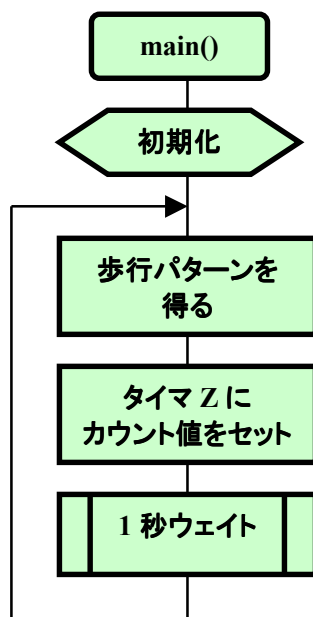
■ 重心を元に戻す

最後に両膝を元に戻します。そうすると右足が前に出た状態になります。

後は左右逆に同じ動きをすれば、今度は左足を前に出すことができます。これをワンセットにして繰り返していけば二足歩行の完成です。次の写真は今まで考えてきた、重心移動・足を前に出す・重心を元に戻す、というステップを利用する歩行パターンです。

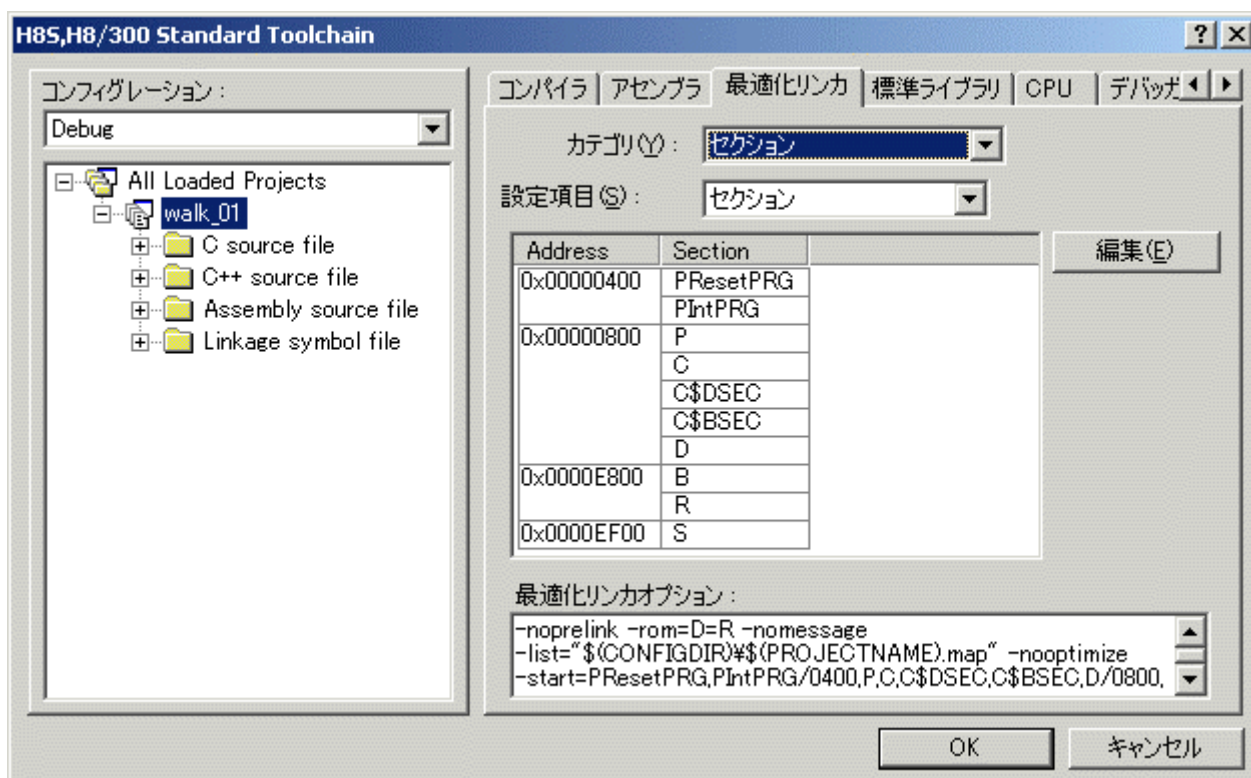


この歩行パターンの概略フローは次のとおりです。以外と簡単でしょう？



このフローをもとにしたソースリスト(ファイル名 'walk_01.c')は次のとおりです。なお、歩行パターンのカウント値は、中心位置のパルス幅(理論値=1500 μ s)からの差分($\pm \mu$ 秒)で指定しています。

このプログラムはフラッシュ ROM に書くので、HEW のツールチェーンを開き、セクションの設定を次のように行ないビルドします。



ノーエラーならば FDT で次の mot ファイルを TK-3687mini に書き込んで下さい。(FDT の使い方は巻末の付録を参照)

c:¥pirkus¥program¥walk_01¥walk_01¥Debug¥walk_01.mot

皆さんの“Pirkus・R Type-02”はちゃんと歩きましたか？

```

/*****/
/*                                     */
/* FILE      :walk_01.c                */
/* DATE      :Thu, Jan 26, 2006        */
/* DESCRIPTION :Main Program           */
/* CPU TYPE   :H8/3687                 */
/*                                     */
/* This file is programed by TOYO-LINX Co.,Ltd. / yKikuchi */
/*                                     */
/*****/

履歴

-----
2006-01-26 : プラグラム開始
/*****/

インクルードファイル
/*****/
#include <machine.h> //H8 特有の命令を使う
#include "iodefine.h" //内蔵 I/O のラベル定義

定数の定義 (直接指定)
/*****/
#define OK      0 //戻り値
#define NG     -1 //戻り値

#define STEP    6 //歩行パターンステップ数

定数エリアの定義 (ROM)
/*****/
// 歩行パターン
const int SequenceTable[STEP][8] = {
// 左膝 左股 左肩 左腕 右膝 右股 右肩 右腕
{0, 150, -570, 300, 0, 150, 570, -300}, //step1
{-125, 150, -570, 300, -400, 150, 570, -300}, //step2
{-125, -150, -570, 300, -400, -150, 570, -300}, //step3
{0, -150, -570, 300, 0, -150, 570, -300}, //step4
{400, -150, -570, 300, 125, -150, 570, -300}, //step5
{400, 150, -570, 300, 125, 150, 570, -300}, //step6
};

グローバル変数の定義とイニシャライズ (RAM)
/*****/
// サーボモータの中心 (カウント値)
unsigned int HomePos[8] = {7400 //P60, 左膝
, 7500 //P61, 左股
, 7350 //P62, 左肩
, 7500 //P63, 左腕
};

```



```

, 7250 //P64, 右膝
, 7650 //P65, 右股
, 7300 //P66, 右肩
, 7750 //P67, 右腕
};

// サーボモータの位置 (パルス幅, 中点からの±μ秒)
int ServoPos[8] = {0, 0, 0, 0, 0, 0, 0, 0};

/*****
関数の定義
*****/
void init_tmz(void);
void intprog_tmz0(void);
void main(void);
void servo_set(void);
void wait(void);

/*****
メインプログラム
*****/
void main(void)
{
    int i, j;

    // イニシャライズ -----
    init_tmz();

    // メインループ -----
    while(1) {
        for (i=0; i<STEP; i++) {
            for (j=0; j<8; j++) {
                ServoPos[j] = SequenceTable[i][j];
            }
            servo_set();
            wait();
        }
    }
}

/*****
サーボモータデータセット
*****/
void servo_set(void)
{
    TZ0.GRA = HomePos[0] + ServoPos[0] * 5; //P60, 左膝
    TZ0.GRB = HomePos[1] + ServoPos[1] * 5; //P61, 左股
    TZ0.GRC = HomePos[2] + ServoPos[2] * 5; //P62, 左肩
    TZ0.GRD = HomePos[3] + ServoPos[3] * 5; //P63, 左腕
    TZ1.GRA = HomePos[4] + ServoPos[4] * 5; //P64, 右膝
    TZ1.GRB = HomePos[5] + ServoPos[5] * 5; //P65, 右股
    TZ1.GRC = HomePos[6] + ServoPos[6] * 5; //P66, 右肩
    TZ1.GRD = HomePos[7] + ServoPos[7] * 5; //P67, 右腕
}

```

```

/*****
    タイマ Z イニシャライズ
*****/
void init_tmz(void)
{
    TZ. TSTR. BYTE = 0x00;    //TCNT0, 1 停止

    TZ0. TCR. BYTE = 0xe2;    //同期クリア, φ/4
    TZ1. TCR. BYTE = 0xe2;    //同期クリア, φ/4

    TZ. TMDR. BYTE = 0x0f;    //TCNT0, 1 は同期動作

    TZ. TOCR. BYTE = 0xff;    //初期出力=1

    TZ. TOER. BYTE = 0x00;    //出力端子イネーブル

    TZ0. TIORA. BYTE = 0x99;  //GRA, GRB はコンペアマッチで 0 出力
    TZ0. TIORC. BYTE = 0x99;  //GRC, GRD はコンペアマッチで 0 出力
    TZ1. TIORA. BYTE = 0x99;  //GRA, GRB はコンペアマッチで 0 出力
    TZ1. TIORC. BYTE = 0x99;  //GRC, GRD はコンペアマッチで 0 出力

    TZ0. TSR. BYTE = 0x00;    //割込みフラグクリア
    TZ1. TSR. BYTE = 0x00;    //割込みフラグクリア

    TZ0. TIER. BYTE = 0x10;    //オーバーフローインターラプトイネーブル
    TZ1. TIER. BYTE = 0x00;    //インターラプトディセーブル

    TZ0. GRA      = HomePos[0]; //カウント初期値
    TZ0. GRB      = HomePos[1]; //カウント初期値
    TZ0. GRC      = HomePos[2]; //カウント初期値
    TZ0. GRD      = HomePos[3]; //カウント初期値
    TZ1. GRA      = HomePos[4]; //カウント初期値
    TZ1. GRB      = HomePos[5]; //カウント初期値
    TZ1. GRC      = HomePos[6]; //カウント初期値
    TZ1. GRD      = HomePos[7]; //カウント初期値

    TZ0. TCNT     = 0x0000;    //TCNT0=0
    TZ1. TCNT     = 0x0000;    //TCNT1=0

    TZ. TSTR. BYTE = 0x03;    //TCNT0, 1 カウントスタート
}

/*****
    タイマ Z チャンネル 0 割込み
*****/
#pragma regsave (intprog_tmz0)
void intprog_tmz0(void)
{
    //タイマ Z オーバフローインターラプトフラグ クリア
    TZ0. TSR. BIT. OVF =0;

    //出力を 1 にする
    TZ. TOCR. BYTE = 0xff;
}

```

```
/******  
    ウェイト(1000ms)  
*****/  
void wait(void)  
{  
    unsigned long i;  
  
    for (i=0;i<3333333;i++) {}  
}
```



では次に、より滑らかな動きが出せるようにスムージング処理を加えてみましょう。きっと、その違いに驚くこと間違いなしです！

5 スムージング付き二足歩行

前の章で二足歩行ができるようになりました。でも、ちょっと不満が残りませんか？動きがぎこちなくて、思ったようにはまっすぐ動いてくれなかったと思います。なんとというか、

ガン!, ゴン!, ガン!, ギン!, ガン!, …

という感じですね。この章ではもっと滑らかに動かす方法を考えてみましょう。

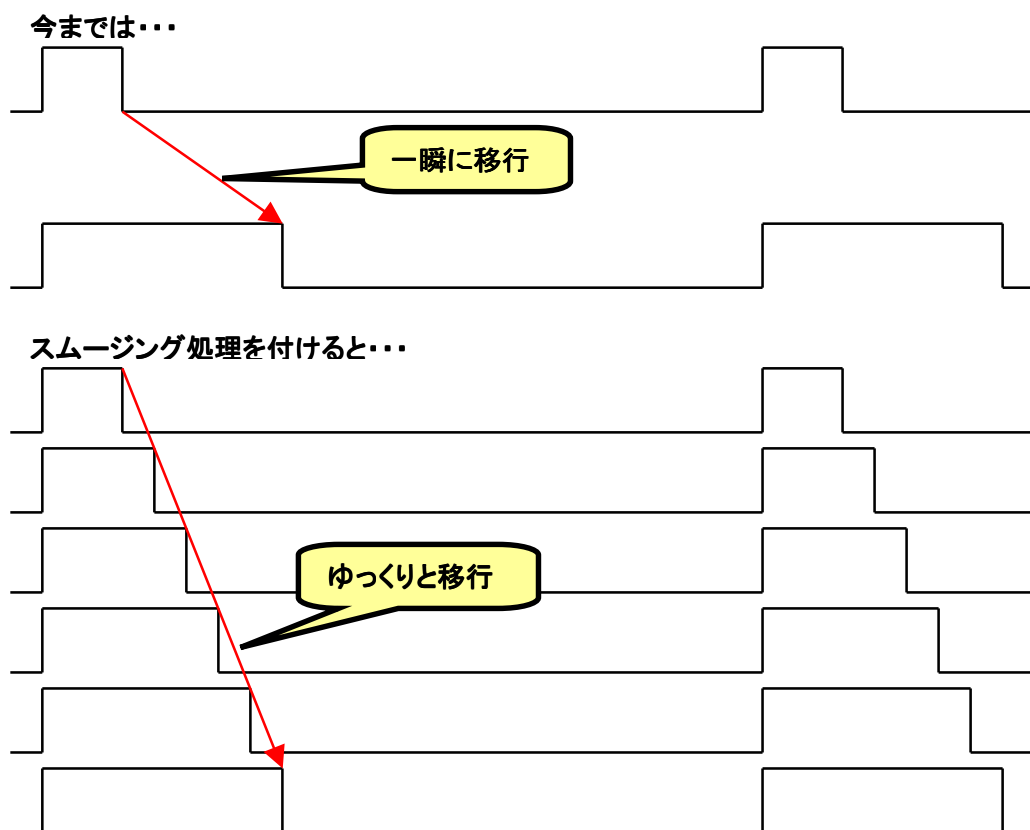
■ 滑らかに動かないのはなぜ？

動きを観察してまず気づくのは「RC サーボの動きが急すぎる」ということです。ある位置から次の位置までの間隔は1秒くらいあるのですが、例えば今まで $+500\mu\text{s}$ の位置だったのが、次のステップで $-500\mu\text{s}$ の位置いきなり動きます。すると、その勢いでロボット全体が動いてしまいます。

RCサーボの動作スピードは 60° 動くのに何秒かかるかによって表されます。メーカーや、同じメーカーでも型番によって様々ですが、 $0.1\text{秒}/60^\circ \sim 0.2\text{秒}/60^\circ$ が多いようです。ということは、最大稼働幅の 180° 動く場合でも $0.3 \sim 0.6$ 秒しかかかりません。ロボット全体のRCサーボがそれだけの勢いで動くわけなので、ロボット各部の加速度を考えると思ったように動いたり止まったりしてくれないわけです。

■ スムージング処理を追加する

そこで、目的の位置までゆっくり移行するように制御します(1~2秒くらいかけて)。これをスムージング処理と呼ぶことにしましょう。



■ カウント値の計算方法

スムージング処理のためには、いきなり目標のカウント値をタイマ Z にセットしてはだめです。では、どのようにカウント値を計算するか考えてみましょう。

まずはどれくらいの時間をかけて次の位置まで到達するか決めましょう。RC サーボに加えているパルス信号の周期は 13.1072ms でした。当然ながら、この時間より速くパルスの長さを変えることはできません。それで、13.1072ms 毎にパルスの長さを変化させることにします。あとは何段階で目標のカウント値にするかですが、ここは 128 段階にしましょう。今回は特に仕様が決められているわけではないので 1~2 秒になれば何でもよいのですが、2 の N 乗の数値の方がマイコンにとって計算が楽です。ちなみに 128 段階のとき目標の位置に到達するまでにかかる時間を計算すると、

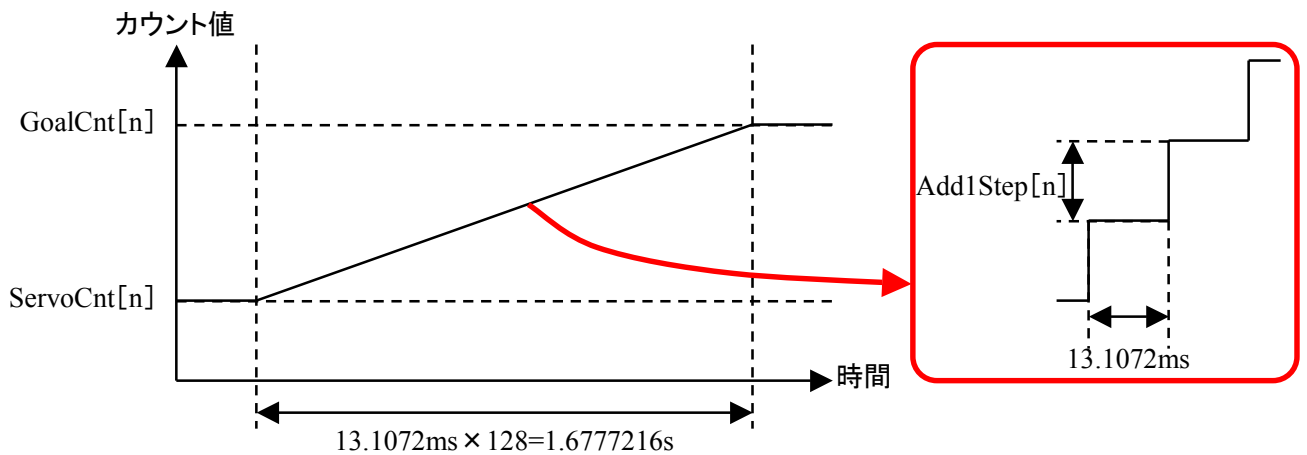
$$13.1072\text{ms} \times 128 \text{ 段階} = 1.6777216 \text{ 秒}$$

となります。ちょうどいいですね。

さて、目標のカウント値を $\text{GoalCnt}[n]$ 、現在のカウント値を $\text{ServoCnt}[n]$ 、1 段階に加算するカウント値を $\text{Add1Step}[n]$ とします。すると、

$$\text{Add1Step}[n] = \frac{\text{GoalCnt}[n] - \text{ServoCnt}[n]}{128}$$

というふうに計算できます。あとはタイマ Z のオーバーフロー割込みのたびに(つまり 13.1072ms 毎に)、 $\text{ServoCnt}[n]$ に $\text{Add1Step}[n]$ を加算し、 $\text{ServoCnt}[n]$ をタイマ Z にセットしていけば OK です。



ところで、タイマ Z にセットする値は 16 ビットです。ということは、 $\text{GoalCnt}[n]$ 、 $\text{ServoCnt}[n]$ 、 $\text{Add1Step}[n]$ は 16 ビットデータ、つまり int で定義してよいのでしょうか。

具体的な数値で考えてみましょう。例えば最も大きな差になるのは $-800 \mu\text{s}$ から $+800 \mu\text{s}$ にするときです。カウント値でいうと 3500 から 11500 になります。その差は 8000 なので、 $\text{Add1Step}[n]$ は 62.5 になりますが、小数点以下は切捨てになります。では、0.5 ぐらい誤差範囲と考えてよいのでしょうか。62 \times 128 = 7936 です。3500 + 7936 = 11436、つまり、目標値に対しカウント値で 64 誤差がでます。カウント値 64 は $12.8 \mu\text{s}$ になり、角度にすると 1.44° です。ロボットとしては無視できない誤差です。

小さい方も考えてみましょう。カウント値 1 の差のときです。128 で割ると 0.0078125 なので小数点以下切り捨てだと $\text{Add1Step}[n]$ は 0 になり、128 回加算しても目標値になりません。あるいは $1 \mu\text{s}$ 変化させる場合のカウント値の差は 5 で 128 で割ると 0.0390625 です。ということは $\text{Add1Step}[n]$ は 0 になり、やはり 128 回加算しても目標値になりません。

つまり、16 ビットでは下の桁が足りないのです。小数点以下も計算できるようにしなければなりません。そこで 32 ビット、つまり long で定義し、上位 16 ビットを整数部分、下位 16 ビットを小数点以下とします。

ServoCnt[n], GoalCnt[n], Add1Step[n]																															
整数部分																小数部分															
bit31.....bit16																bit15.....bit0															
↓																															
タイマ Z にセットする																															

もう一度、具体的な数値で考えてみましょう。最も大きな差になる $-800 \mu s$ から $+800 \mu s$ にするときです。カウント値でいうと 3500 から 11500 になります。マイコン内部では 16 進数で扱っていますから、カウント値は $0x0DAC$ から $0x2CEC$ になります。

さて、現在値が $0x0DAC$ なので $ServoCnt[n]=0x0DAC$ になるのですが、これまで考えてきたとおり下位 16 ビットは小数点以下です。というわけで、

$$ServoCnt[n]=0x0DAC0000$$

となります。これは、 $0x0DAC.0000$ を表しています。目標値も同じように、

$$GoalCnt[n]=0x2CEC0000$$

となり、これは $0x2CEC.0000$ を表しています。では、 $Add1Step[n]$ を計算してみましょう。

$$\begin{aligned} Add1Step[n] &= \frac{GoalCnt[n] - ServoCnt[n]}{128} \\ &= \frac{0x2CEC0000 - 0x0DAC0000}{128} \\ &= 0x003E8000 \end{aligned}$$

これは、 $0x003E.8000$ を表しています。それで、13.1072ms 毎に $ServoCnt[n]$ に $0x003E8000$ を加算します。そして、タイマ Z には $ServoCnt[n]$ の上位 16 ビットの値をセットします。

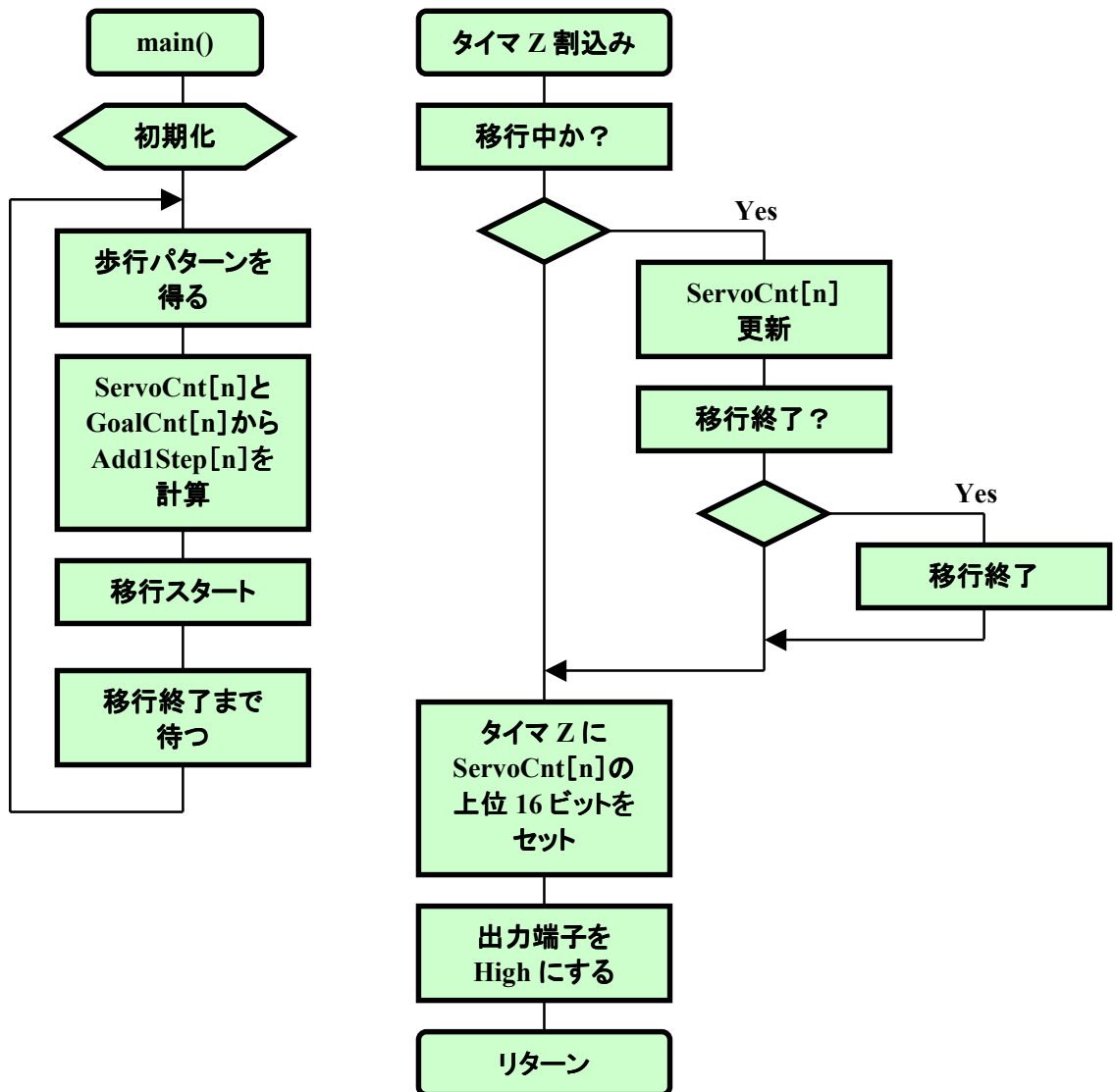
小さい方も計算してみましょう。1 μs の差、例えば $0 \mu s$ から $+1 \mu s$ にするときです。カウント値でいうと 7500 から 7505 になります。16 進数では $0x1D4C$ から $0x1D51$ になります。先ほどと同じように計算すると、

$$\begin{aligned} Add1Step[n] &= \frac{GoalCnt[n] - ServoCnt[n]}{128} \\ &= \frac{0x1D510000 - 0x1D4C0000}{128} \\ &= 0x0000A00 \end{aligned}$$

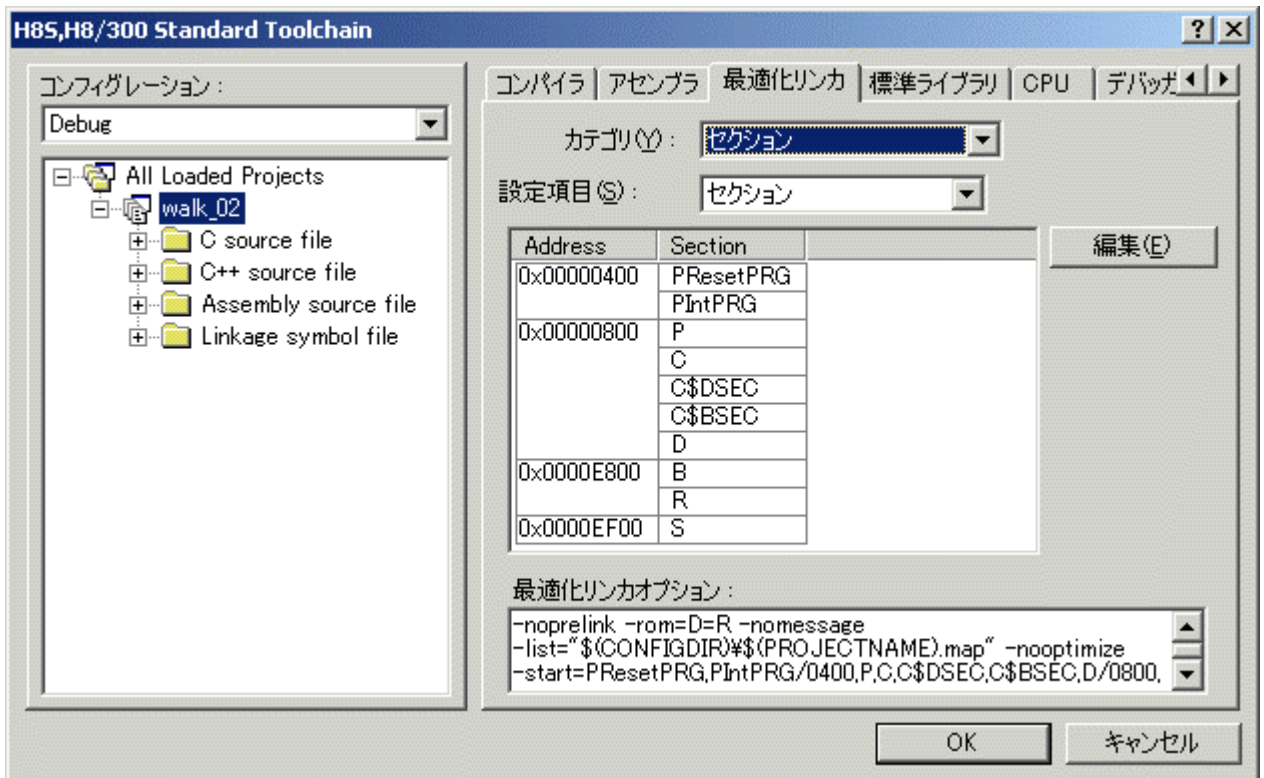
これは、 $0x0000.A00$ を表しています。それで、13.1072ms 毎に $ServoCnt[n]$ に $0x0000A00$ を加算します。そして、タイマ Z には $ServoCnt[n]$ の上位 16 ビットの値をセットします。

■ フローチャートとプログラム

これまで考えてきたことを踏まえて概略フローを考えてみましょう。次のとおりです。



このフローをもとに作ったソースリスト(ファイル名 'walk_02. c')は次のとおりです。フラッシュROM に書くので、HEW のツールチェーンを開き、セクションの設定を次のように行ないビルドします。



ノーエラーならば FDT で次の mot ファイルを TK-3687mini に書き込んで下さい。

c:¥pirkus¥program¥walk_02¥walk_02¥Debug¥walk_02. mot

```

/*****
/*                                     */
/* FILE      :walk_02. c               */
/* DATE      :Tue, Feb 14, 2006        */
/* DESCRIPTION :Main Program           */
/* CPU TYPE   :H8/3687                 */
/*                                     */
/* This file is programed by TOYO-LINX Co.,Ltd. / yKikuchi */
/*                                     */
/*****

履歴

-----
2006-02-14 : プラグラム開始
*****

*****
インクルードファイル
*****
#include <machine.h> //H8 特有の命令を使う
#include "iodefine.h" //内蔵 I/O のラベル定義

*****

```



```

定数の定義（直接指定）
*****/
#define      OK          0          //戻り値
#define      NG          -1         //戻り値

#define      STEP        6          //歩行パターンステップ数

#define      SERVO_STEP  128        //サーボモータの1シーケンスにおける移行段階

/*****
定数エリアの定義 (ROM)
*****/
// 歩行パターン
const int SequenceTable[STEP][8] = {
// 左膝   左股   左肩   左腕   右膝   右股   右肩   右腕
  {0,    150,  -570,  300,   0,    150,   570,  -300}, //step1
  {-125, 150,  -570,  300,  -400,  150,   570,  -300}, //step2
  {-125, -150, -570,  300,  -400,  -150,  570,  -300}, //step3
  {0,    -150, -570,  300,   0,    -150,  570,  -300}, //step4
  {400,  -150, -570,  300,  125,  -150,  570,  -300}, //step5
  {400,  150,  -570,  300,  125,  150,   570,  -300}, //step6
};

/*****
グローバル変数の定義とイニシャライズ (RAM)
*****/
// サーボモータの中心（カウント値）/理論値は 1500μs ÷ 200ns = 7500
unsigned int HomePos[8] = {7400 //P60, 左膝
                          , 7500 //P61, 左股
                          , 7350 //P62, 左肩
                          , 7500 //P63, 左腕
                          , 7250 //P64, 右膝
                          , 7650 //P65, 右股
                          , 7300 //P66, 右肩
                          , 7750 //P67, 右腕
                          };
// サーボモータの位置（パルス幅, 中心からの±μ秒）
int ServoPos[8] = {0, 0, 0, 0, 0, 0, 0, 0};
// サーボモータの現在カウント値
long ServoCnt[8]; //上位 16 ビットをタイマ Z にセットする
// サーボモータの目標カウント値
long GoalCnt[8];
// サーボモータカウント値移行時の1段階加算値
long Add1Step[8];
// サーボモータカウント値移行カウンタ
unsigned int MoveCnt;
// サーボモータカウント値移行時のシーケンス
unsigned char SequenceStage = 0; // 0:停止
                                     // 1:移行中
                                     // 2:終了

/*****
関数の定義
*****/
void init_servo(void);

```

```

void      init_tmz(void);
void      intprog_tmz0(void);
void      main(void);
void      servo_set(void);
void      wait(void);

/*****
    メインプログラム
*****/
void main(void)
{
    int i, j;

    // イニシャライズ -----
    init_tmz();
    init_servo();

    // メインループ -----
    while(1) {
        for (i=0; i<STEP; i++) {
            for (j=0; j<8; j++) {
                ServoPos[j] = SequenceTable[i][j];
            }
            servo_set();
            while(SequenceStage<2) {nop();}
        }
    }
}

/*****
    サーボモータ イニシャライズ
*****/
void init_servo(void)
{
    int i;

    for (i=0; i<8; i++) {
        ServoCnt[i] = HomePos[i] * 0x10000;
    }
}

/*****
    サーボモータデータセット
*****/
void servo_set(void)
{
    int i;

    for (i=0; i<8; i++) {
        GoalCnt[i] = (HomePos[i] + ServoPos[i] * 5) * 0x10000;
        Add1Step[i] = (GoalCnt[i] - ServoCnt[i]) / SERVO_STEP;
    }

    MoveCnt = SERVO_STEP;
    SequenceStage = 1;
}

```

```

}

/*****
   タイマ Z イニシャライズ
*****/
void init_tmz(void)
{
    TZ.TSTR.BYTE = 0x00;    //TCNT0, 1 停止

    TZ0.TCR.BYTE = 0xe2;    //同期クリア, φ/4
    TZ1.TCR.BYTE = 0xe2;    //同期クリア, φ/4

    TZ.TMDR.BYTE = 0x0f;    //TCNT0, 1 は同期動作

    TZ.TOCR.BYTE = 0xff;    //初期出力=1

    TZ.TOER.BYTE = 0x00;    //出力端子イネーブル

    TZ0.TIORA.BYTE = 0x99;  //GRA, GRB はコンペアマッチで 0 出力
    TZ0.TIORC.BYTE = 0x99;  //GRC, GRD はコンペアマッチで 0 出力
    TZ1.TIORA.BYTE = 0x99;  //GRA, GRB はコンペアマッチで 0 出力
    TZ1.TIORC.BYTE = 0x99;  //GRC, GRD はコンペアマッチで 0 出力

    TZ0.TSR.BYTE = 0x00;    //割込みフラグクリア
    TZ1.TSR.BYTE = 0x00;    //割込みフラグクリア

    TZ0.TIER.BYTE = 0x10;    //オーバーフローインターラプトイネーブル
    TZ1.TIER.BYTE = 0x00;    //インターラプトディセーブル

    TZ0.GRA = HomePos[0];    //カウント初期値
    TZ0.GRB = HomePos[1];    //カウント初期値
    TZ0.GRC = HomePos[2];    //カウント初期値
    TZ0.GRD = HomePos[3];    //カウント初期値
    TZ1.GRA = HomePos[4];    //カウント初期値
    TZ1.GRB = HomePos[5];    //カウント初期値
    TZ1.GRC = HomePos[6];    //カウント初期値
    TZ1.GRD = HomePos[7];    //カウント初期値

    TZ0.TCNT = 0x0000;    //TCNT0=0
    TZ1.TCNT = 0x0000;    //TCNT1=0

    TZ.TSTR.BYTE = 0x03;    //TCNT0, 1 カウントスタート
}

/*****
   タイマ Z チャンネル 0 割込み
*****/
#pragma regsave (intprog_tmz0)
void intprog_tmz0(void)
{
    int i;

    //タイマ Z オーバフローインターラプトフラグ クリア
    TZ0.TSR.BIT.OVF =0;

```

```

//TCNT0,1 停止
TZ.TSTR.BYTE = 0x00;

//カウント値の加算
if (SequenceStage==1) {
    for (i=0; i<8; i++){
        ServoCnt[i] = ServoCnt[i] + Add1Step[i];
    }
    MoveCnt--; if (MoveCnt==0) {SequenceStage = 2;}
}

//タイマZにカウント値をセット
TZ0.GRA = ServoCnt[0] / 0x10000;
TZ0.GRB = ServoCnt[1] / 0x10000;
TZ0.GRC = ServoCnt[2] / 0x10000;
TZ0.GRD = ServoCnt[3] / 0x10000;
TZ1.GRA = ServoCnt[4] / 0x10000;
TZ1.GRB = ServoCnt[5] / 0x10000;
TZ1.GRC = ServoCnt[6] / 0x10000;
TZ1.GRD = ServoCnt[7] / 0x10000;

//出力を1にする
TZ.TOCR.BYTE = 0xff;

//TCNT0,1 カウントスタート
TZ0.TCNT = 0x0000;
TZ1.TCNT = 0x0000;
TZ.TSTR.BYTE = 0x03;
}

/*****
ウェイト(1000ms)
*****/
void wait(void)
{
    unsigned long i;

    for (i=0;i<3333333;i++) {}
}

```



基本的な二足歩行の考え方は以上です。あとは、ステップ数を増やしたり、回転の角度を変えたり、歩くだけではなくいろいろな動きを加えたり、皆さんの工夫しだいで面白い動きができると思います。いろいろ試してみてください。

このプログラムは変数に整数型を使っているため、小数点の位置を固定して整数部と小数部に分けました。一方、C 言語には浮動小数点型の変数もあるので、これを使うことも考えられます。RC サーボモータの制御方法や、スムージング機能の考え方は同じなので、興味のある方はこちらも試してみてください。

6 パソコンで“Pirkus・R Type-02”を制御する

これまでは、プログラム上で指定した数値に従い“Pirkus・R Type-02”を動かしてきました。しかし、これだと動作を変更するたびにプログラムをビルドしなおし、mot ファイルをダウンロードしなければなりません。また、数値を直接指定するよりも、実際にポーズをつけながら動作を指定していったほうが、直感的でわかりやすいのも事実です。

そこで、“Pirkus・R Type-02”の販売元の「アイ・ビー株式会社」が提供している“モーションエディタ”を利用して、パソコンで“Pirkus・R Type-02”を動かしてみましょう。

■ “PCLINK”の実装

モーションエディタのコマンドを受信・解析し、“Pirkus・R Type-02”の RC サーボに指定されたパルスを出力するプログラムです。FDT で次の mot ファイルを TK-3687mini に書き込んで下さい。

```
c:¥pirkus¥program¥pclink¥pclink¥Debug¥pclink.mot
```

あとは電源をオンして待ちます。

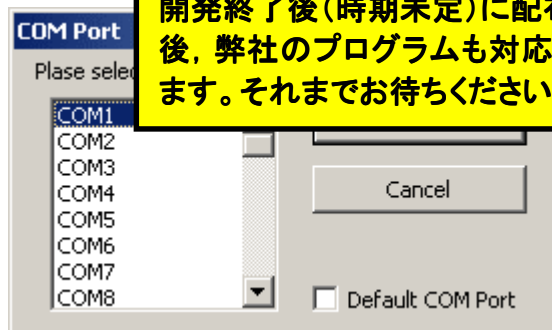
■ モーションエディタの起動

コマンドはシリアルポートから送信されます。パソコンと TK-3687mini のシリアルポートをつなぎましょう。

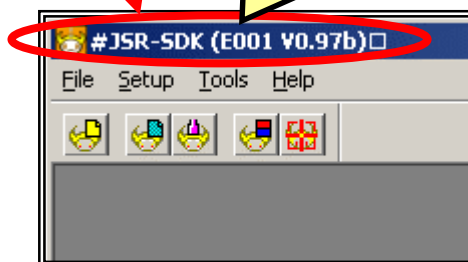
モーションエディタの起動
TK-3687mini を
モーションエディタのウィンドウ
は PCLINK のバ



マニュアル執筆時点で「アイ・ビー株式会社」は「モーションエディタ」の販売を行っておりません。新しいバージョンの「モーションエディタ」を開発中とのことです。コマンドも一新される予定です。開発終了後(時期未定)に配布再開される予定です。その後、弊社のプログラムも対応するためバージョンアップします。それまでお待ちください。

ダイアログが開きます。
下さい。モーションエ
です(表示される内容



バージョンが表示されたので接続完了



接続できないときは、シリアルポートの番号やケーブルの接続を確認して下さい。そのあと、モーションエディタを起動し直すか、“”ボタンをクリックしてください。モーションエディタのウィンドウのタイトルに PCLINK のバージョンが表示され、ボタンが“”に変化したら接続完了です。

これで、モーションエディタから“Pirkus・R Type-02”を制御できるようになりました。あとは、モーションエディタのマニュアルを見ながら動かしてみてください。

7 マイコンで“Pirkus・R Type-02”を制御する

前の章ではパソコンのプログラム，“モーションエディタ”で“Pirkus・R Type-02”を制御しました。複雑な動きも比較的簡単に実行できるのが特徴です。しかし，あらかじめ動作をプログラムするので，状況に応じて動かす，というのはあまり得意ではありません。

そこで，マイコンで操縦器を作って，ロボットをリモコン操作してみましょう。気分は“○人 28 号”といたったところでしょうか・・・(ちょっと古いですかね)。

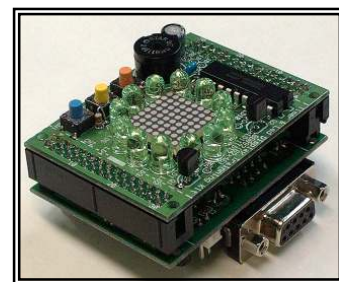
■ “Pirkus・R Type-02”に実装するプログラム

これは“PCLINK”をそのまま流用します。今までは“モーションエディタ”を実行しているパソコンからコマンドを“Pirkus・R Type-02”に送っていましたが，今回はパソコンのかわりにマイコン内蔵操縦器から“モーションエディタ”のコマンドを送るようにします。それで，前の章と同じように，FDT で次の mot ファイルを“Pirkus・R Type-02”の TK-3687mini に書き込んで下さい。

```
c:\¥pirkus¥program¥pclink¥pclink¥Debug¥pclink.mot
```

■ 操縦器のハードウェア

タイマ&LED ディスプレイキット(B6092, 東洋リンクス)を使います。タクトスイッチが 3 個実装されているので，その組み合わせで 7 種類の動作を指示します。マイコンは TK-3687mini を使います。(タイマ&LED ディスプレイキットと TK-3687mini は弊社より別途ご購入ください)



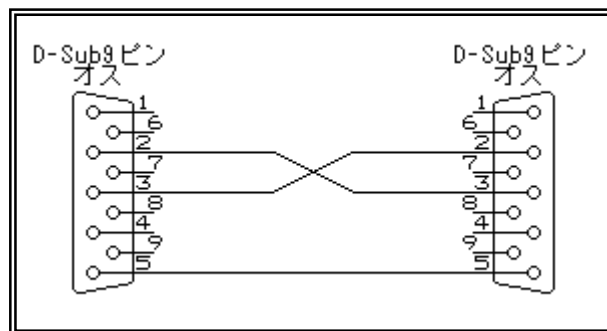
■ 操縦器のプログラムをダウンロードする

操縦器の TK-3687mini に FDT で次の mot ファイルを書き込んで下さい。

```
c:\¥pirkus¥program¥remocon2¥remocon2¥Debug¥remocon2.mot
```

■ 操縦器と“Pirkus・R Type-02”の接続

パソコンとつなぐときはストレートケーブル(D-Sub9 ピン, オス-メス)を使いました。しかし，今回は TK-3687mini 同士をつなぐため，**クロスケーブル(D-Sub9 ピン, オス-オス)**を使います。市販のものが使えますが，右の結線図を参考にして自作してみてもいいでしょう。



■ 操縦方法

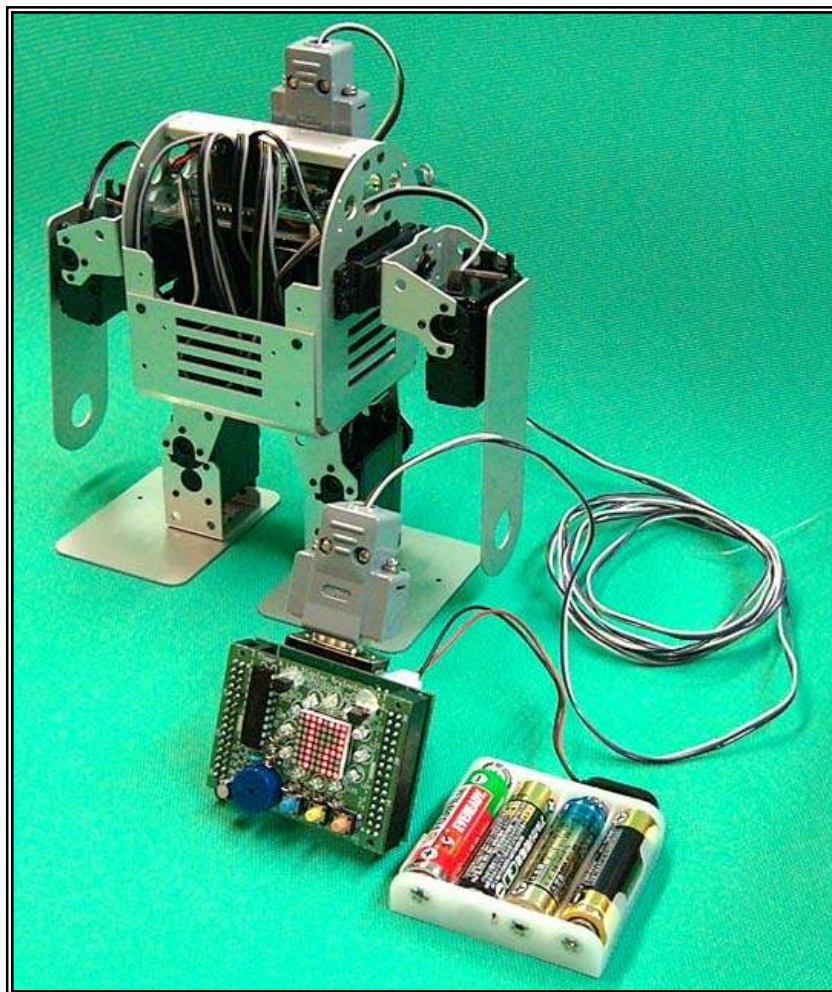
B6092 のタクトスイッチの組み合わせで動作を指示します。デフォルトでは次のような動作がプログラムされています。

SW3	SW2	SW1	動作
Off	Off	Off	何もしない
Off	Off	On	斜め右に前進
Off	On	Off	斜め左に前進
Off	On	On	前進
On	Off	Off	ホームポジション
On	Off	On	持ち上げ
On	On	Off	リアアット
On	On	On	すくい投げ



■ 電源オン

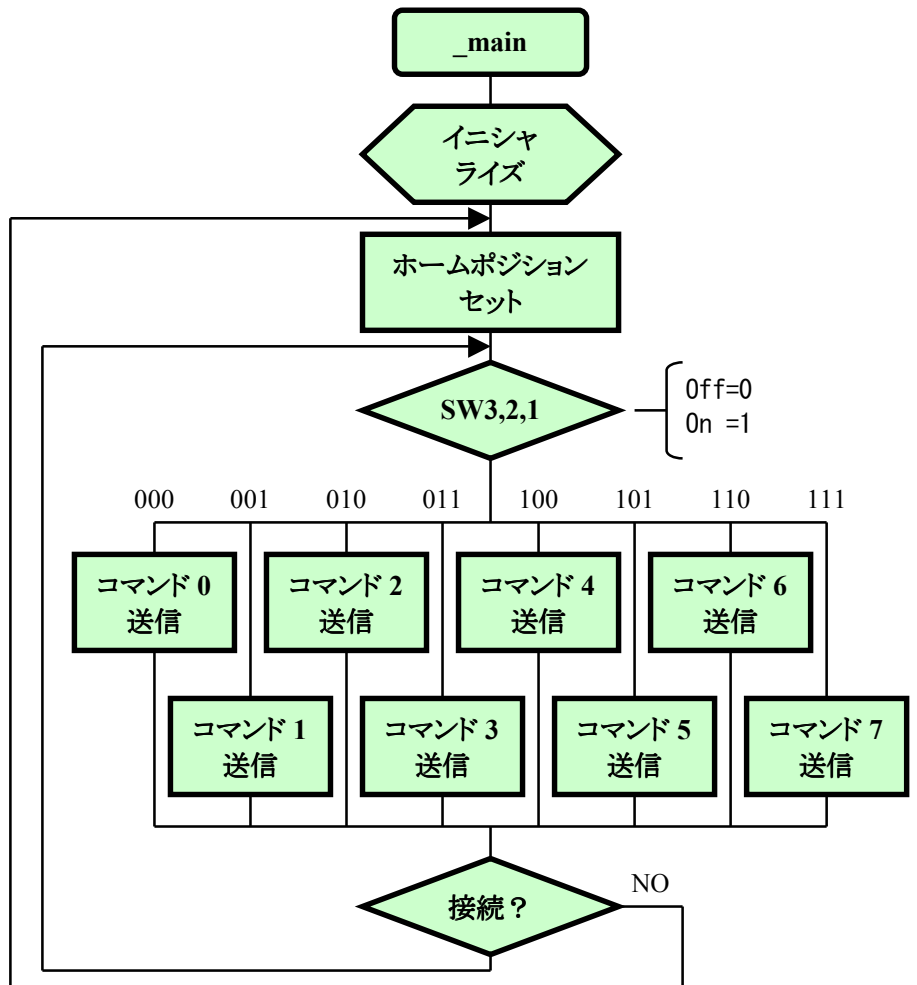
それでは動かしてみましよう。①“Pirkus・R Type-02”, ②操縦器, の順番で電源をオンしてください。操縦器のタクトスイッチをオンすると, 組み合わせに応じた動作をします。



ところで, 操縦器のプログラムはどのように考えればよいのでしょうか。次に, 操縦器のプログラムの中身を説明します。

■ フローチャート

プログラムの基本的な構造は非常に簡単です。概略フローチャートとそれをもとにしたメインプログラムのソースリストは次のとおりです。



```

/*****
メインプログラム
*****/
void main(void)
{
    // イニシャライズ -----
    init_io();
    init_tmv();
    init_tmb1();
    init_tmz0();
    init_sci3();
    init_rxbuf();

    while(1) {
        // ホームポジションにする
        while (send_cmd(4, SwData2) == NG) {
            set_Picture(8); //つながっていない
        }

        // メインループ -----
        while(1) {
            // スイッチの状態振り分ける
            switch(SwData2 & 0x38) {
                // SW1=Off, SW2=Off, SW3=Off. ....
                case 0x00:

```

```

        BGMO_n = 0;
        set_Picture(0);
        if (send_cmd(0, SwData2)==NG) {goto NON_CONNECT;}
        break;
//SW1=0n , SW2=0ff, SW3=0ff. ....
case 0x08:
    BGMO_n = 1;
    set_Picture(1);
    if (send_cmd(1, SwData2)==NG) {goto NON_CONNECT;}
    break;
//SW1=0ff, SW2=0n , SW3=0ff. ....
case 0x10:
    BGMO_n = 1;
    set_Picture(2);
    if (send_cmd(2, SwData2)==NG) {goto NON_CONNECT;}
    break;
//SW1=0n , SW2=0n , SW3=0ff. ....
case 0x18:
    BGMO_n = 1;
    set_Picture(3);
    if (send_cmd(3, SwData2)==NG) {goto NON_CONNECT;}
    break;
//SW1=0ff, SW2=0ff, SW3=0n .....
case 0x20:
    set_Picture(4);
    BGMO_n = 0;
    if (send_cmd(4, SwData2)==NG) {goto NON_CONNECT;}
    break;
//SW1=0n , SW2=0ff, SW3=0n .....
case 0x28:
    BGMO_n = 0;
    set_Picture(5);
    if (send_cmd(5, SwData2)==NG) {goto NON_CONNECT;}
    break;
//SW1=0ff, SW2=0n , SW3=0n .....
case 0x30:
    BGMO_n = 0;
    set_Picture(6);
    if (send_cmd(6, SwData2)==NG) {goto NON_CONNECT;}
    break;
//SW1=0n , SW2=0n , SW3=0n .....
case 0x38:
    BGMO_n = 0;
    set_Picture(7);
    if (send_cmd(7, SwData2)==NG) {goto NON_CONNECT;}
    break;
    }
}

// ケーブルが外れた
NON_CONNECT:
    BGMO_n = 0;
}
}

```


■ PCLINK のコマンド

PCLINK がサポートしているコマンドの中で、リモコンで必要なのはたった一つです。それは、

‘ma’コマンド（現在の位置から指定された時間で X の状態まで移行する）

です。ma コマンドは次のような構造をしています。（数値は例です）

文字	16 進数	
m	6D	コマンドシグネチャ
a	61	
,	2C	区切り
0	30	移行時間(0050=1 秒) 例:0.5 秒
0	30	
2	32	
5	35	
,	2C	区切り
0	30	キーフレームまでの移行時間(0050=1 秒) 例:1 秒
0	30	
5	35	
0	30	
,	2C	区切り
0	30	リザーブ
0	30	
0	30	
0	30	
,	2C	区切り
0	30	サーボ 0(左膝)の相対角度 例:0 度
0	30	
0	30	
0	30	
,	2C	区切り
-	2D	サーボ 1(左股)の相対角度 例:-35 度
0	30	
3	33	
5	35	
0	30	区切り
,	2C	
-	2D	
0	30	
1	31	サーボ 7(右腕)の相対角度 例:-12.3 度
2	32	
3	33	
3	33	

		↓
0	30	サーボ 2(左肩)の相対角度 例:20 度
2	32	
0	30	
0	30	
,	2C	区切り
0	30	サーボ 3(左腕)の相対角度 例:30 度
3	33	
0	30	
0	30	
,	2C	区切り
0	30	サーボ 4(右膝)の相対角度 例:35 度
3	33	
5	35	
0	30	
,	2C	区切り
0	30	サーボ 5(右股)の相対角度 例:40 度
4	34	
0	30	
0	30	
,	2C	区切り
0	30	サーボ 6(右肩)の相対角度 例:45 度
4	34	
5	35	
0	30	
,	2C	区切り
-	2D	サーボ 7(右腕)の相対角度 例:-12.3 度
0	30	
1	31	
2	32	
3	33	区切り
,	2C	
-	2D	
0	30	

移行時間

現在の位置から、ma コマンドで指定されたサーボの相対角度まで移行するまでの時間。BCD4 桁で指定する。0050=1 秒。

キーフレームまでの移行時間

キーフレームに移行するまでの時間。ma コマンドでは意味を持たない。BCD4 桁で指定する。0050=1 秒。

サーボの相対角度

各 RC サーボの角度。BCD4 桁で指定する。原点(RC サーボの中心)を 0 度とし、そこからのプラス・マイナスの相対的な角度で指定する。最小単位は 0.1 度。ただし、マイナスのときは BCD4 桁の前に‘-’を付け 5 桁で指定する(例:-90 度のときは-0900)。

PCLINK は ma コマンドを受信し、指定された位置まで RC サーボを移動すると、アンサーバックを返します。操縦器の方はアンサーバックを受信してから次の ma コマンドを送信します。アンサーバックは次のとおりです。

文字	16進数	
LF	0A	
CR	0D	
\$	24	
O	4F	
K	4B	
LF	0A	
CR	0D	

■ ma コマンドで二足歩行を行なう

ma コマンドを使えば RC サーボを任意の位置に設定することができます。そして、スイッチを押したときに一連の ma コマンドを順番に送信するようにプログラムすれば二足歩行できます。二足歩行の基本的な考え方は 4 章と 5 章で説明しました。重心移動と足の動かし方は全く一緒です。では、二足歩行のときにどんなコマンドを実際に送信しているかソースリストを見てみましょう。このリストの中の黄色い行が二足歩行のコマンドです。

```

/*****
定数エリアの定義 (ROM)
*****/
// コマンドテーブル -----
const char Command[8][512] = {
    //SW1=0ff, SW2=0ff, SW3=0ff : 何もしない
    {""},
    //SW1=0n , SW2=0ff, SW3=0ff : 斜め右に前進
    {"ma, 0040, 0050, 0000, 0000, -0084, 0000, 0700, 0000, -0169, 0000, -0700¥nma, 0040, 0050, 0000, 0450, -0084, 0
000, 0700, 0141, -0169, 0000, -0700¥nma, 0040, 0050, 0000, 0450, 0084, 0000, 0700, 0141, 0169, 0000, -0700¥nma, 0040
, 0050, 0000, 0000, 0084, 0000, 0700, 0000, 0169, 0000, -0700¥nma, 0040, 0050, 0000, -0141, 0084, 0000, 0700, -0450, 0
169, 0000, -0700¥nma, 0040, 0050, 0000, -0141, -0084, 0000, 0700, -0450, -0169, 0000, -0700¥n"},
    //SW1=0ff, SW2=0n , SW3=0ff : 斜め左に前進
    {"ma, 0040, 0050, 0000, 0000, 0169, 0000, 0700, 0000, 0084, 0000, -0700¥nma, 0040, 0050, 0000, -0141, 0169, 000
0, 0700, -0450, 0084, 0000, -0700¥nma, 0040, 0050, 0000, -0141, -0169, 0000, 0700, -0450, -0084, 0000, -0700¥nma, 00
50, 0050, 0000, 0000, -0169, 0000, 0700, 0000, -0084, 0000, -0700¥nma, 0040, 0050, 0000, 0450, -0169, 0000, 0700, 014
1, -0084, 0000, -0700¥nma, 0040, 0050, 0000, 0450, 0169, 0000, 0700, 0141, 0084, 0000, -0700¥n"},
    //SW1=0n , SW2=0n , SW3=0ff : 前進
    {"ma, 0040, 0050, 0000, 0000, 0169, 0000, 0700, 0000, 0169, 0000, -0700¥nma, 0040, 0050, 0000, -0141, 0169, 00
00, 0700, -0450, 0169, 0000, -0700¥nma, 0050, 0050, 0000, -0141, -0169, 0000, 0700, -0450, -0169, 0000, -0700¥nma,
0050, 0050, 0000, 0000, -0169, 0000, 0700, 0000, -0169, 0000, -0700¥nma, 0040, 0050, 0000, 0450, -0169, 0000, 0700,
0141, -0169, 0000, -0700¥nma, 0040, 0050, 0000, 0450, 0169, 0000, 0700, 0141, 0169, 0000, -0700¥n"},
    //SW1=0ff, SW2=0ff, SW3=0n : ホームポジション
    {"ma, 0025, 0050, 0000, 0000, 0000, -0700, 0700, 0000, 0000, 0700, -0700¥n"},
    //SW1=0n , SW2=0ff, SW3=0n : 持ち上げ
    {"ma, 0050, 0050, 0000, 0000, 0000, 0000, 0700, 0000, 0000, 0000, -0700¥nma, 0035, 0050, 0000, 0000, 0000, 0700
, 0700, 0000, 0000, -0700, -0700¥n"},
    //SW1=0ff, SW2=0n , SW3=0n : ラリアット
    {"ma, 0050, 0050, 0000, 0000, 0000, 0000, -0350, 0000, 0000, 0000, 0350¥nma, 0035, 0050, 0000, 0000, 0000, 0000
, 0700, 0000, 0000, 0000, -0700¥n"},
    //SW1=0n , SW2=0n , SW3=0n : すくい投げ

```

```

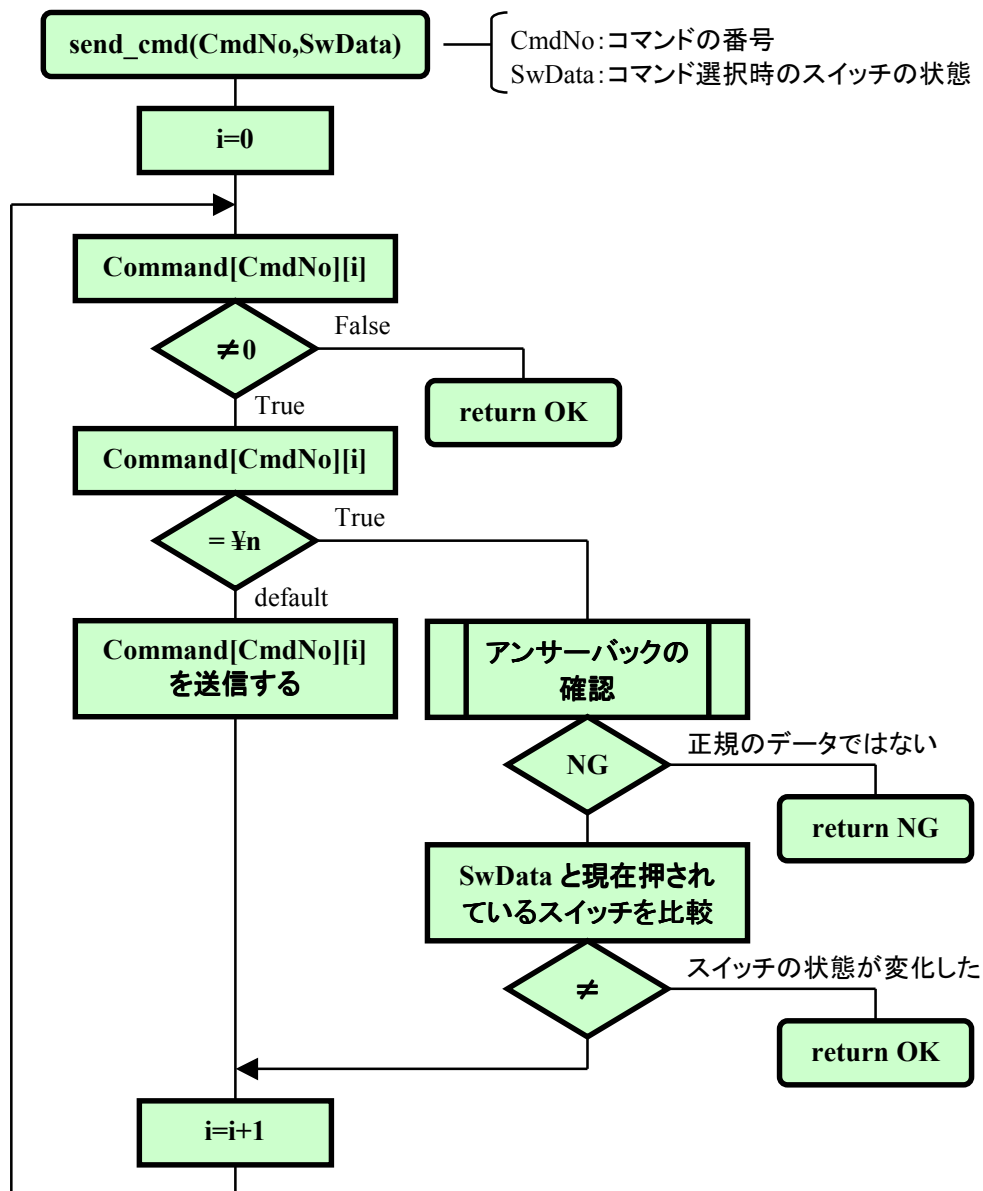
{"ma, 0050, 0050, 0000, 0000, 0000, -0700, 0700, 0000, 0000, 0700, -0700¥nma, 0035, 0050, 0000, 0000, 0000, 000
0, 0700, 0000, 0000, 0000, -0700¥n"}
};

```

さて、4章と5章で説明しましたが、二足歩行は6ステップで行なっていました。それで、ma コマンドを 6 回送信して左右の足のそれぞれを一步前に出します。それぞれの ma コマンドの区切りは '¥n' (改行, =0x0D) とします。また、ソースリストから分かるように、コマンドはコマンドテーブルの配列に文字列として定義しています。そのため、文字列の最後には自動的にナル(NULL, =0x00) が付け加えられます。それで、二足歩行のコマンドテーブルはメモリ上に次のように割り付けられます。

m	a	,	0	0	4	0	,	0	7	0	0	¥n	m	a	,	0	0
6D	61	2C	30	30	34	30	2C	30	37	30	30	0D	6D	61	2C	30	30
¥n	m	a	,	9	0	0	0	0	0	,	-	0	7	0	0	¥n	NULL
0D	6D	61	2C	39	2C	30	30	30	30	2C	2D	30	37	30	30	0D	00

この図を見るとプログラムをどのように作ればよいか見えてきますね。コマンドテーブルの内容を 1 バイトずつ送信, ¥n(0x0D) だったら“Pirkus・R Type-02”からアンサーバックが送られてくるのを待つ, NULL(0x00) だったらコマンド送信終了, という感じです。フローチャートにしてみましょう。



フローチャートをもとにコーディングすると、次のようになります。

```
/******  
 コマンドの送信  
******/  
int send_cmd(unsigned char CmdNo, unsigned char SwData)  
{  
    unsigned int i;  
    unsigned int len = 0;  
    unsigned char d;  
  
    while (get_rxbuf(&d)==OK) {} //受信バッファのクリア  
  
    for (i=0; Command[CmdNo][i]!=0; i++) {  
        switch (Command[CmdNo][i]) {  
            case '¥n':  
                if (ok_wait(cal_ansback_wait(&Command[CmdNo][i], len))==NG) {  
                    return NG;  
                }  
                if (SwData!=SwData2) return OK;  
                len = 0;  
                break;  
            default:  
                txone(Command[CmdNo][i]);  
                len++;  
                break;  
        }  
    }  
  
    return OK;  
}
```



あとは、コマンドの内容を変えればいろいろな動きをさせることができます。面白い動きを考えてみてください。

8 赤外リモコンで“Pirkus・R Type-02”を制御する

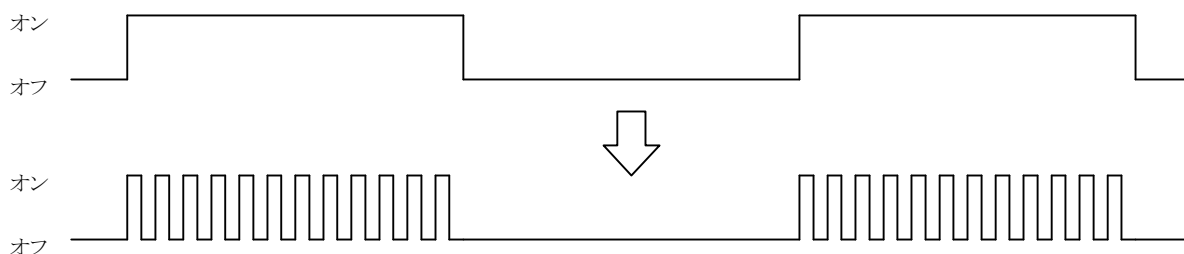
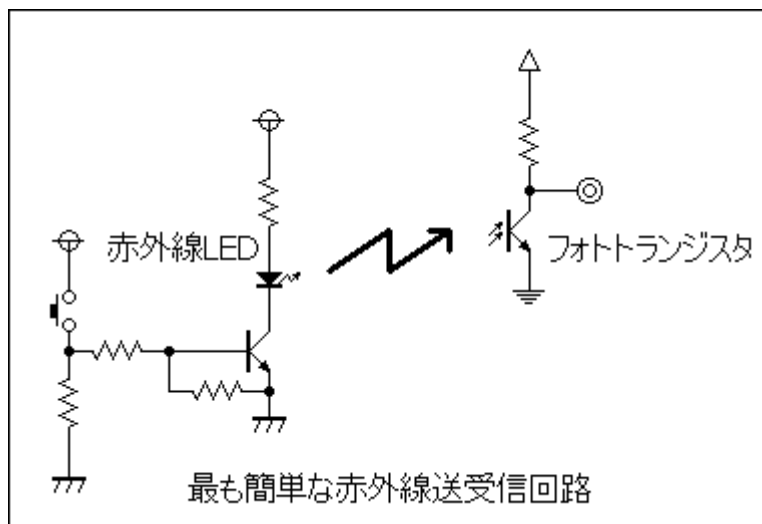
前の章の冒頭で「そこで、マイコンで操縦器を作って、ロボットをリモコン操作してみましょう。気分は“○人 28号”といったところでしょうか…(ちょっと古いですかね。)」と書きましたが、“鉄○28号”というにはケーブルが邪魔です。そこで、赤外リモコンで操作することを考えてみましょう。

■ 赤外線送受信回路

赤外線送受信回路の基本的な形は、赤外線 LED とフォトランジスタ(もしくはフォトダイオード)を使った右のような回路になります。(スイッチをオンすると赤外線 LED が発光し、フォトランジスタがオンする)

ただ、実際にやってみるとわかりますが、この回路はまったくと言っていいほど使い物になりません。おそらく数センチ離しただけで届かなくなり、送信側のスイッチを押していないにもかかわらず受信側は頻りにオンします。

原因は、わたしたちの周囲に赤外線を出すものが氾濫していることです(太陽光線、白熱電球、蛍光灯など)。そのため、それらがノイズとなって通信を邪魔します。そこで、伝えたいデータをキャリアで変調する方法が使われています(下図参照、この図は正論理で描いています)。なぜ、キャリアで変調するとノイズに強くなるのでしょうか。



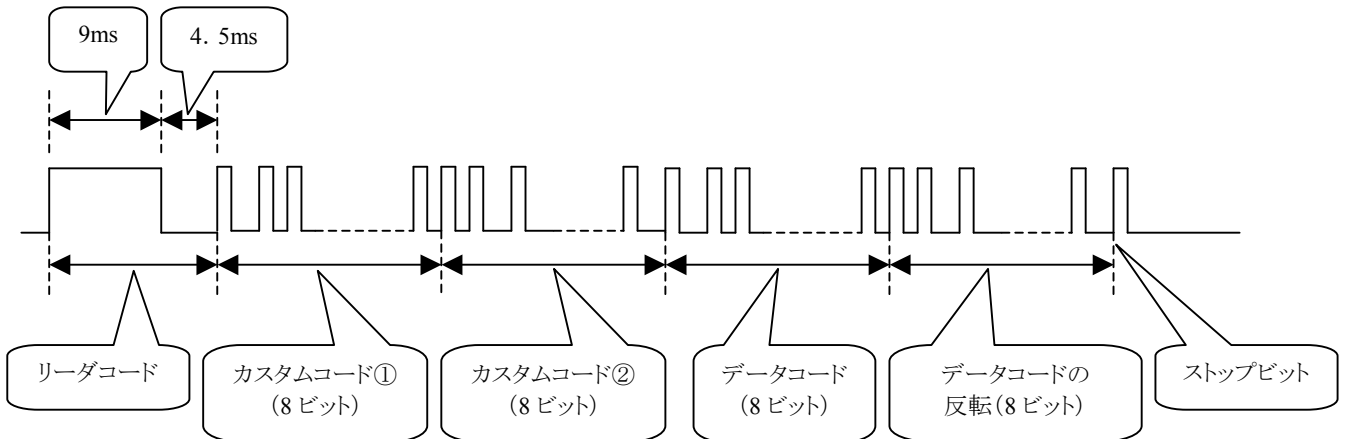
ノイズに強くするためには、強力な赤外線を発光してデータの強さがノイズレベル以上になるようにする必要があります。そのために赤外発光 LED に大きな電流を流したいところです。ところが、LED の最大定格を見てみると(TLN115 の場合)直流順電流は 100mA までです。これだと弱すぎて遠くまで信号を伝えることができません。もちろん、最大定格を超える電流を流しつづけることはできません。しかし、よく見るとデータシートにはもう一つ、パルス順電流というのが載せられていて、TLN115 の場合 1A です。これは、ごく短い時間であれば 1A まで流すことができることを示しています。そこで、変調することによって(つまりパルス波形にして=短い時間だけオンするようにして)大きな電流を流し、より強い赤外線を発光するようにします。

さらに、変調することにより、受信側にキャリア周波数だけ通すバンドパスフィルタ回路を追加することでノイズを除去することができます。多くの赤外リモコンはキャリア周波数として 38KHz を採用しています。赤外信号受光 IC (IS1U60, TSOP1738, 等)は 38KHz だけを通すバンドパスフィルタが内蔵されており、キャリアを除去したあとの復調された信号が負論理で出力されます。

しかし、実際にはこれでも誤動作することがあります。赤外信号受信 IC のバンドパスフィルタでも除去しきれずに出力されてしまったり、他の赤外線が重なることで信号が欠けてしまったりするためです。それで、通常はマイコンと組み合わせてプログラムでさらにノイズを除去し、正しいデータを取り出すようにします。次に、赤外リモコンの信号フォーマットを見てみましょう。

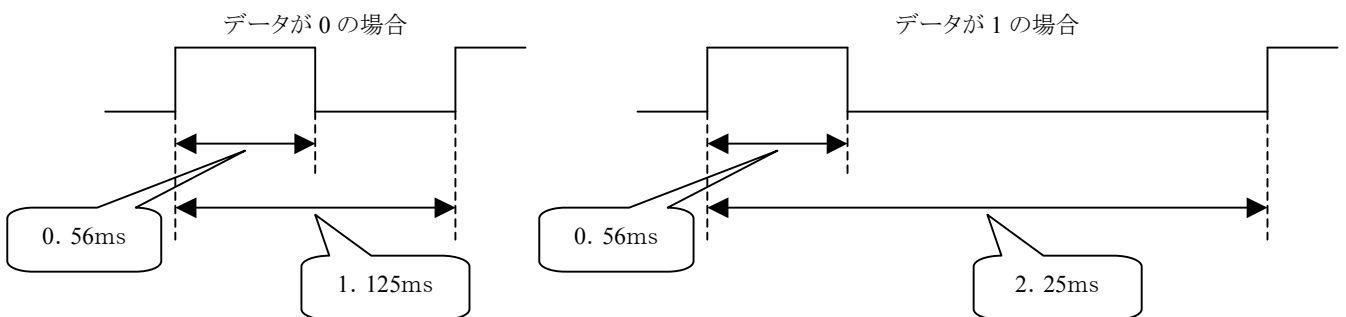
■ NEC フォーマット

赤外リモコンでは、赤外線を利用してデータを低速で送信し、受信側は赤外線を検知してデータを受け取ります。リモコン信号はシリアル信号なので、受信プログラムでは、どこからデータが始まったのか、本当に正しく受信できたのか、判断する必要があります。そのためのデータフォーマットが定められています。世の中でよく使われている標準的なフォーマットが何種類か存在しますが、その一つ、NEC フォーマットを見てみましょう。NEC フォーマットは 1 バイト (=8 ビット) のデータを送受信するフォーマットで、次のような構成になっています(この図は正論理で描いています、また、実際は送信時に変調されます)。



リーダーコードはこれからデータが始まることを示すコードです。9ms の期間オンの状態が続き、その後 4.5ms の期間オフ状態になります。他のコード(カスタムコードやデータコード)と比較して波形が大きく異なるため、容易にリーダーコードであることがわかります。

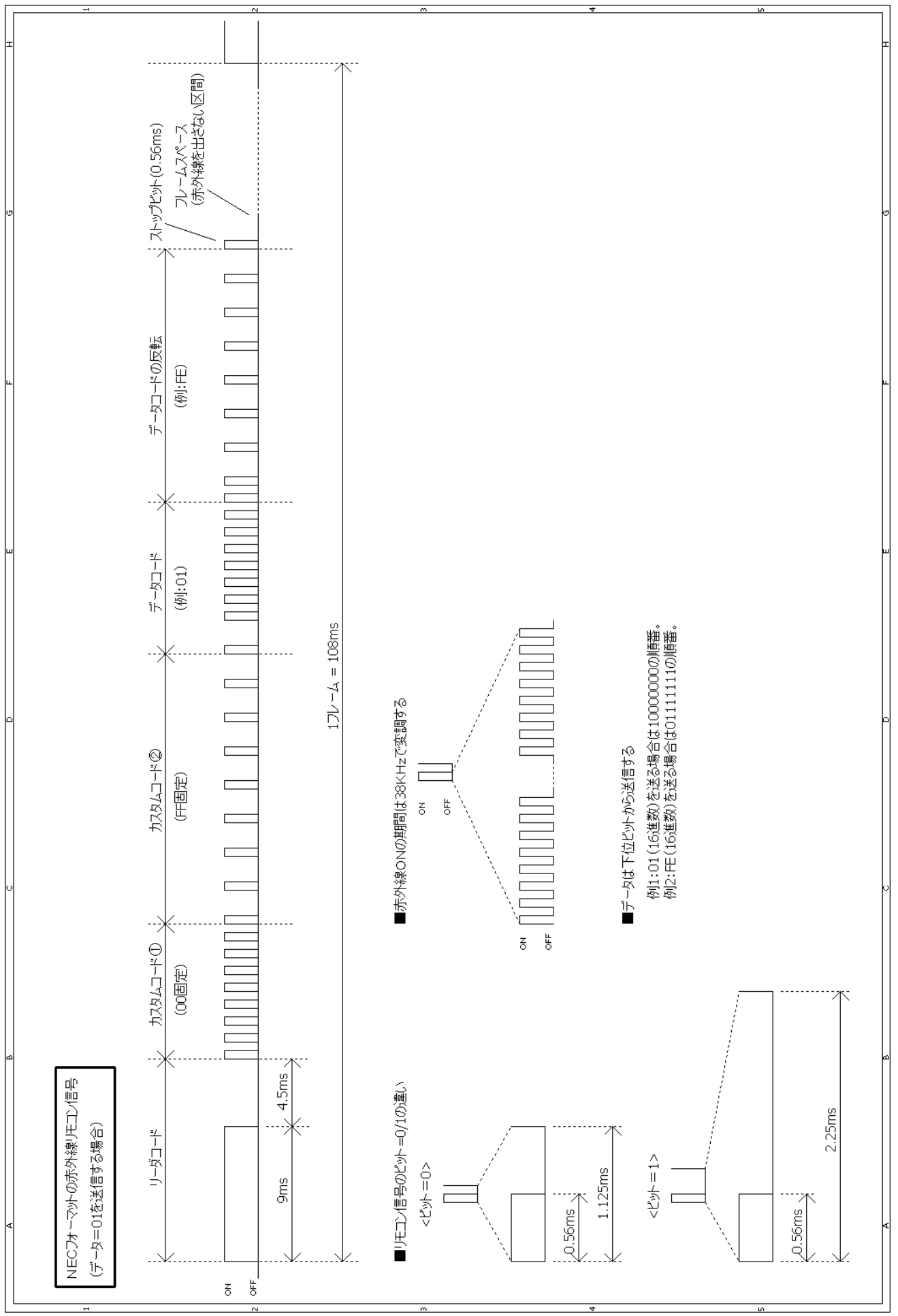
続く、カスタムコードやデータコードが 0/1 のデータを含む部分です。各部分は下位ビットから送信されます。そして、0/1 は赤外線の有無ではなく、下図のように信号の長さで区別します(この図は正論理で描いています)。



カスタムコード①、②は、メーカーによって誤動作しないよう区別するコードで、メーカーごとに NEC から割り当てられます。NEC に登録しなくてもメーカーに関係なく自由に使えるコードもあり、今回は“00”、“FF”を使っています(注意: このカスタムコードを使っている他の赤外リモコン機器があれば動作してしまう可能性があります)。

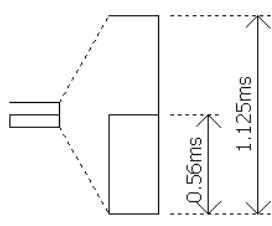
次に、データコードを送信します。データコードは 8 ビットなので 0~255 (00h~FFh) まで 256 種類のデータを送信することができます。続いて、データコードの全てのビットを反転したデータを送信します。受信プログラムでは、受信したデータコードと続けて受信した反転データコードを比較して正しい値か確認し、間違ったデータを受信したときはそのデータを捨て、正しいデータだけを採用します。

次のページにデータ“01”を NEC フォーマットで送信する場合のタイミングチャートを示します。カスタムコードは“00”、“FF”にしました。

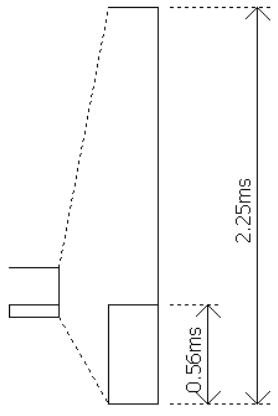


NECフォーマットの赤外線リモコン信号
(データ=01を送信する場合)

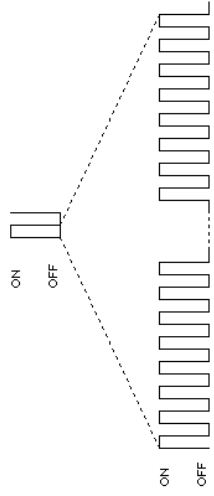
■リモコン信号のビット=0/1の違い
<ビット=0>



<ビット=1>



■赤外線ONの期間は38KHzで変動する



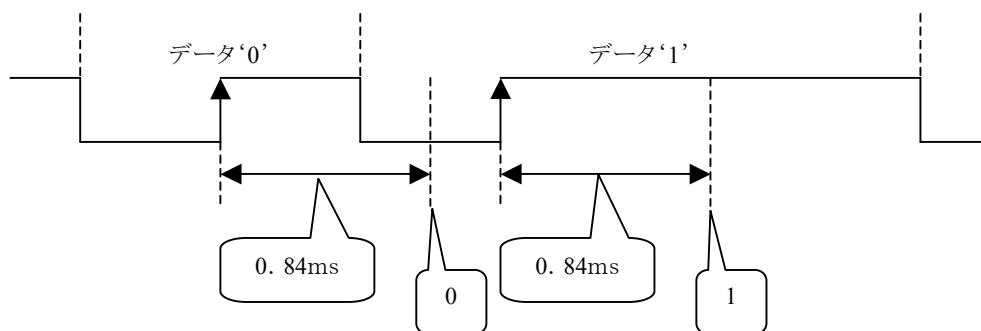
■データは下位ビットから送信する

例1: 01 (16進数)を送信する場合は100000000の順番。
例2: FE (16進数)を送信する場合は011111111の順番。

■ 受信プログラムの考え方

受信プログラムは、まず赤外線信号が入力されたかチェックし、入力されたならばその信号がリーダーコードかチェックします。リーダーコードかどうかは赤外線信号のオン時間とオフ時間がある範囲に収まるかで判断します。許容範囲をどれくらいにするかは使用する環境にもよりますが、今回は±10% (オン時間=8.1ms~9.9ms, オフ時間=4.05ms~4.95ms) にしました。

リーダーコードを受信したら、続いて受信プログラムは、赤外線信号のオン時間とオフ時間をチェックして0/1を判断していきます。下図のように、赤外信号受光ICの負論理出力の立ち上がりから0.84ms後のポートの状態で判断しています(この図は赤外信号受光ICの出力、つまり負論理で描いています)。



受信したカスタムコードが自分のコードであればデータを採用します。次に、受信したデータコードと続けて受信した反転データコードを比較し正しいデータだけ採用します。

ところで、信頼性を向上させるために、通常データ通信では正しく伝わったか互いに確認しながらデータを送受信します。しかし、赤外リモコンの場合、送信器から一方的にデータを送るだけなので(垂れ流しと言います)、正しく受け取ったか確認したり、もう一度送るよう要求したりすることができません。そこで、スイッチが押されている間は繰り返し同じデータを送信し続け、多少データを取りこぼしたとしても動作に影響しないように受信側もプログラムします。



NEC フォーマットについての詳しい説明は次の Web サイトをご覧ください。(2008 年 11 月 19 日現在)

http://www.necel.com/ja/faq/mi_com/_com_remo.html

■ 赤外リモコン送信機

最初に、部品表と比較して部品が全てそろっているか確認しましょう。部品によっては相当品使用の場合もあります。(部品が足りないときは巻末記載の連絡先までお問い合わせください。)

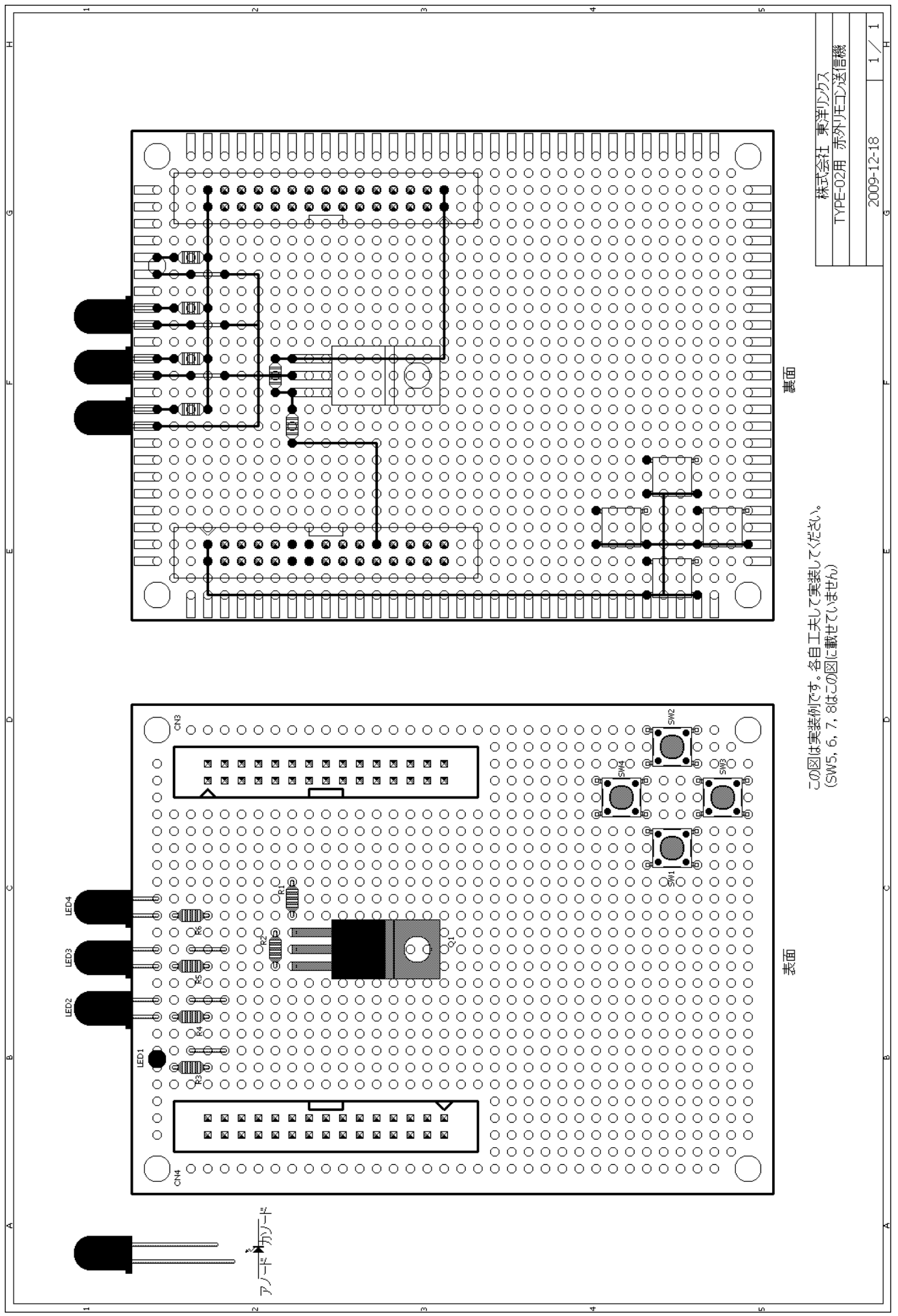
赤外リモコン送信機 組み立てキット

	部品番号	型名, 規格	メーカー	数量	付属数量	備考
1	■赤外リモコン送信機基板					
2	Q1	2SD1828		1	1	*1, ダーリントントランジスタ
3	LED1	HLMP-6300#A04	HP	1	1	*1
4	LED2~4	TLN115	東芝	3	3	*1, 赤外LED
5	SW1~8	SKHHAK/AM/DC	ALPS	8	8	*1, 赤2, 青2, 黄2, 白2
6	R1~3	1K Ω		3	3	
7	R4~6	10 Ω		3	3	
8	CN3,4	HIF3FC-30PA-2.54DSA	HRS	2	2	*1(CN1, 2は欠番です)
9	ラッピングケーブル	50cm		1	1	*2, メッキ線として使用
10	メッキ線			0	0	ハンダ面結線用 *2
11	ユニバーサル基板	B6093(95×72mm)	東洋リンクス	1	1	
12						
13	■CPUボード(組立キットの場合)					
14	CPUボード	TK-3687mini	東洋リンクス	1	1	フラットパッケージ実装済み
15	REG1	TA48M05F(S)	東芝	1	1	
16	X1	20MHz		1	1	メインクロック
17	X2	32.768KHz		1	1	サブクロック
18	D3	1SS133-T72	ROHM	1	1	*1
19	LED1	HLMP-6300#A04	HP	1	1	*1
20	C3,19	47~100 μ F/16V		2	2	
21	C4,6,17,18,20	10 μ F/16V		5	5	
22	SW1	SKHHAK/AM/DC	ALPS	1	1	*1
23	CN1	B2P-SHF-1AA	JST	1	1	電源用
24	CN3,4	HIF3FB-30DA-2.54DSA	HRS	2	2	基板間コネクタ, ハンダ面に実装
25	CN5	D-Sub9pin		1	1	ストレート
26	JP1	2pin		1	1	ピンとソケットのセット
27						
28	■その他					
29	電池ボックス			1	1	単3×4本用, ケーブル付
30	ゴム足			4	4	
31						

(*1)相当品を使用することがあります。

(*2)ラッピングケーブルの被覆をはがし2本をよじって使用します。また、抵抗やコンデンサの足も流用できます。

回路図と実装図は次のとおりです。



裏面

表面

この図は実装例です。各自工夫して実装してください。
(SW5, 6, 7, 8はこの図に載せていません)

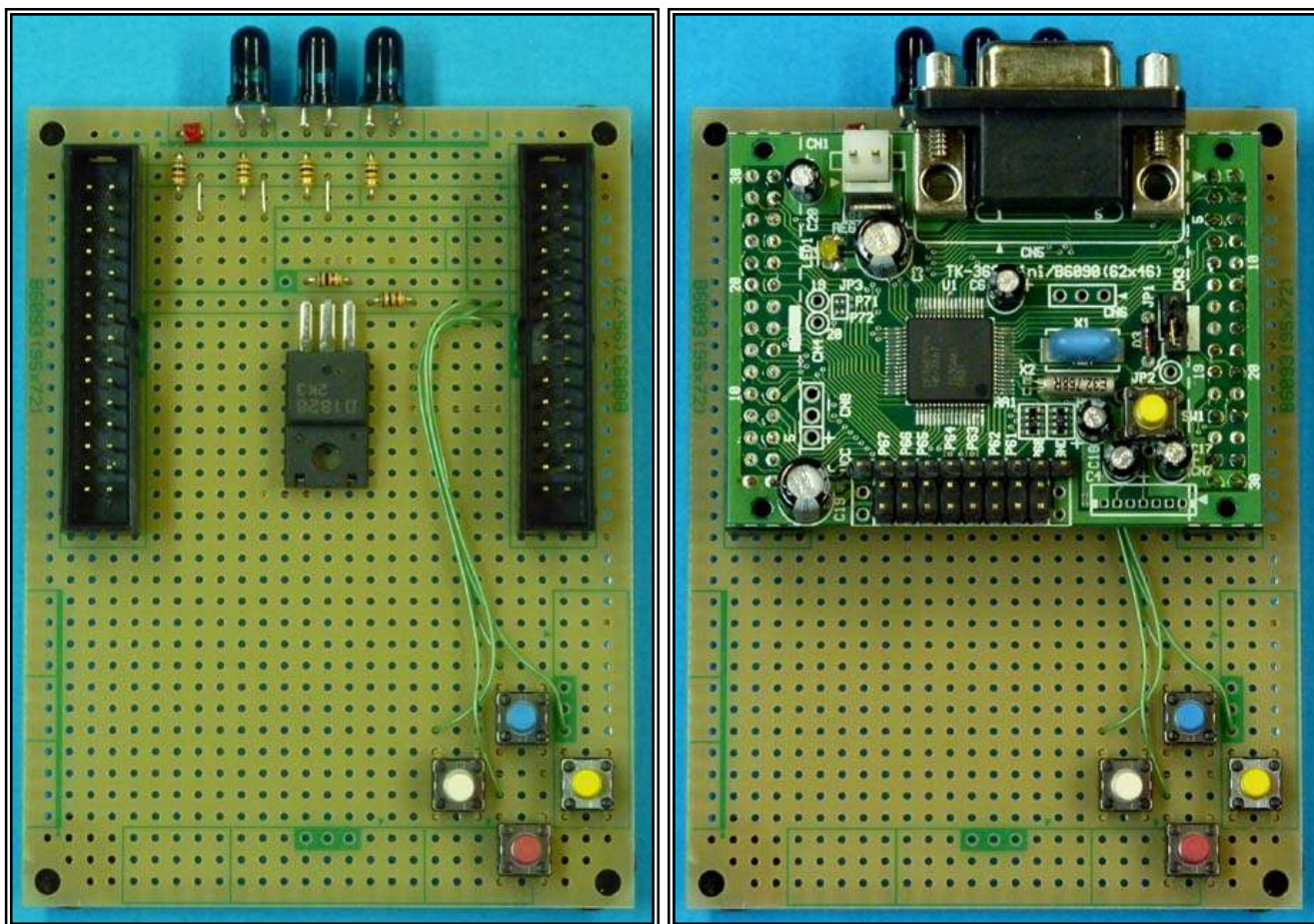
株式会社 東洋リリス
TYPE-02用 赤外線エコー送信機
2009-12-18
1 / 1

実装図の表面(部品面)を見てユニバーサル基板に部品を載せます。次に、実装図の裏面(半田面)を見てメッキ線での配線を済ませてください。最後にスイッチ信号をラッピングケーブルで配線します。(SW5, 6, 7, 8 は図や写真では実装していません。各自工夫して実装してください。)

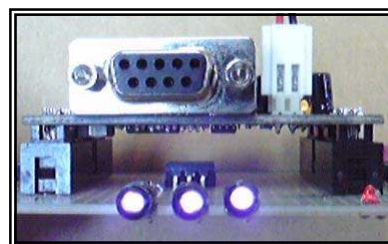
配線が終了したら、回路図どおり配線されているかもう一度確認して下さい。確認方法は、テスタで部品面の端子間の抵抗を測り、導通があるか、すなわち 0Ω か否かで判断します。また、半田付けがきちんと行なわれているか見ておきましょう。動かない原因の大部分は配線ミスと半田付け不良です。

次に TK-3687mini を組み立てましょう。CD の「¥TK-3687mini¥マニュアル¥」フォルダの「TK-3687mini 組み立て手順書.pdf」をご覧ください。

あとは、TK-3687mini を取り付けて組み立て終了です。TK-3687mini に FDT で“ir_remocon_send_mot”をダウンロードします。FDT については CD の「¥始めにお読みください¥」フォルダの「モニタプログラム書き込み手順書.pdf」を参考にしてください。



ここで、動作確認をしておきましょう。どれかスイッチを押してみてください。押している間、動作確認用の LED が点滅すれば、まずは OK です。4 つのスイッチのいずれを押したときにも LED が点滅することを確認してください。また、本当に赤外線が発光されているか確認したい場合は、カメラ付き携帯電話か PHS、もしくはデジカメで、スイッチを押しているときの赤外 LED の様子をファインダで見てください。大抵の場合、LED が光っている様子を見ることができます(カメラに使われているイメージセンサが可視光線だけでなく赤外線にも反応するものが多いため、但し、赤外線フィルタが入っていると見えません)。



リモコン送信機はいずれかのスイッチが押されたときに、カスタムコード“00”，“FF”に続いて、押されたスイッチに応じてつぎのようなコードを NEC フォーマットで送信します。

	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
	SW8	SW7	SW6	SW5	SW4	SW3	SW2	SW1
					青	赤	黄	白
1	オン	オン	オン	オン	オン	オン	オン	オン
0	オフ	オフ	オフ	オフ	オフ	オフ	オフ	オフ

プログラムは次のとおりです。

```

/*****/
/*                                          */
/* FILE      :ir_remocon.c                */
/* DATE      :Wed, Dec 16, 2009          */
/* DESCRIPTION:Main Program              */
/* CPU TYPE   :H8/3687                   */
/*                                          */
/* This file is programed by TOYO-LINX Co.,Ltd. / yKikuchi */
/*                                          */
/*****/

/*****
   スwitchの並び方とポート番号の対応
*****/
-----
          +-----+
          | SW8 |
          | (P5 ) |
          +-----+
          +-----+
+-----+   +-----+
| SW5 |   | SW6 |
| (P5 ) | | (P5 ) |
+-----+   +-----+
          +-----+
          | SW7 |
          | (P5 ) |
          +-----+
          +-----+
          | SW4 |
          | (P53) |
          +-----+
          +-----+   +-----+
+-----+   | SW1 |   | SW2 |
| (P5 ) |   | (P50) | | (P51) |
+-----+   +-----+   +-----+
          +-----+
          | SW3 |
          | (P52) |
          +-----+
*****/

/*****
   インクルードファイル
*****/
#include <machine.h> //H8特有の命令を使う
#include "iodefine.h" //内蔵I/Oのラベル定義
#include "binary.h" //Cで2進数を使うための定義

/*****
   定数の定義（直接指定）
*****/
#define OK      0 //戻り値
#define NG     -1 //戻り値

```

```

/*****
定数エリアの定義 (ROM)
*****/
//          0          1          2          3
//          0123456789012345678901234567890
const char StartChkCnstData[32] = {"TOYO-LINX Co.,Ltd. IR-Send  "};

/*****
グローバル変数の定義とイニシャライズ (RAM)
*****/
// スイッチ入力に関係した変数 -----
unsigned char SwData1 = 0; //ファーストリード
unsigned char SwData2 = 0; //ダブルリードにより決定したデータ
unsigned char SwData3 = 0; //前回のダブルリードで決定したデータ
unsigned char SwData4 = 0; //0→1に変化したデータ
unsigned char SwStatus = 0; //スイッチ入カステータス
// 0:ファーストリード
// 1:ダブルリード

// スタート識別に関係した変数 -----
char StartChkData[32]; //スタート識別データ

/*****
関数の定義
*****/
void button(unsigned char);
void id_set(void);
void init_io(void);
void init_tmv(void);
void intprog_tmv(void);
void main(void);
void switch_in(void);

void IR_sig_out(char); //アセンブラ関数の定義

/*****
メインプログラム
*****/
void main(void)
{
// イニシャライズ -----
init_io();
init_tmv();

// メインループ -----
while(1) {
if (SwData2!=0) {
button(SwData2);
SwData4 = 0;
}
else{
IO.PDR6.BYTE = _11111110B;
}
}
}

```

```

}

/*****
    ボタン/複数入力, 固定ID
*****/
void button(unsigned char sw)
{
    IR_sig_out(sw);
}

/*****
    I/Oポート イニシャライズ
*****/
void init_io(void)
{
    IO. PMR5. BYTE    = 0x00;    //ポート5, 汎用入出力ポート
    IO. PUCR5. BYTE   = 0xff;    //ポート5, 内蔵プルアップオン
    IO. PCR5          = 0x00;    //ポート5, P50-P57入力

    IO. PCR6          = 0xff;    //ポート6, P60-P67出力
    IO. PDR6. BYTE    = 0xfe;    //ポート6, 初期出力設定
}

/*****
    タイマV イニシャライズ
*****/
void init_tmv(void)
{
    TV. TCSR.V. BYTE  = 0x00;    //TOMV端子は使わない
    TV. TCORA         = 156;     //周期=2ms (156/156. 25KHz=1ms)
    TV. TCRV1. BYTE   = 0x01;    //TRGVトリガ入力禁止,
    TV. TCRV0. BYTE   = 0x4b;    //コンペアマッチA 割込みイネーブル
                                //コンペアマッチA でTCNTVクリア
                                //内部クロック  $\phi/128$  (20MHz/128=156. 25KHz)
}

/*****
    タイマV 割込み(1ms)
*****/
#pragma regsav (intprog_tmv)
void intprog_tmv(void)
{
    //コンペアマッチフラグA クリア
    TV. TCSR.V. BIT. CMFA = 0;

    //スイッチ入力
    switch_in();
}

/*****
    スイッチ入力
*****/
void switch_in(void)
{
    unsigned char a;

```



```

switch(SwStatus) {
  case 0:
    SwData1 = ~IO.PDR5.BYTE;
    if (SwData1!=0) {SwStatus = 1;}
    else          {SwData2 = SwData3 =0;}
    break;
  case 1:
    a = ~IO.PDR5.BYTE;
    if (SwData1==a) {
      SwData2 = SwData1;
      SwData4 = SwData4 | (SwData2 & (~SwData3));
      SwData3 = SwData2;
    }
    SwStatus = 0;
    break;
}
}

```

このプログラムは割込みを使っているので、「intprg. c」を追加・修正します。

```

/*****
/*
/* FILE      :intprg.c
/* DATE      :Wed, Dec 16, 2009
/* DESCRIPTION :Interrupt Program
/* CPU TYPE   :H8/3687
/*
/* This file is generated by Renesas Project Generator (Ver.4.16).
/*
*****/

#include <machine.h>

extern void intprog_tmv(void);

#pragma section IntPRG
// vector 1 Reserved

// vector 2 Reserved

// vector 3 Reserved

// vector 4 Reserved

// vector 5 Reserved

// vector 6 Reserved

// vector 7 NMI
__interrupt(vect=7) void INT_NMI(void) {/* sleep(); */}

```

```

// vector 8 TRAP #0
__interrupt(vect=8) void INT_TRAP0(void) { /* sleep(); */}
// vector 9 TRAP #1
__interrupt(vect=9) void INT_TRAP1(void) { /* sleep(); */}
// vector 10 TRAP #2
__interrupt(vect=10) void INT_TRAP2(void) { /* sleep(); */}
// vector 11 TRAP #3
__interrupt(vect=11) void INT_TRAP3(void) { /* sleep(); */}
// vector 12 Address break
__interrupt(vect=12) void INT_ABRK(void) { /* sleep(); */}
// vector 13 SLEEP
__interrupt(vect=13) void INT_SLEEP(void) { /* sleep(); */}
// vector 14 IRQ0
__interrupt(vect=14) void INT_IRQ0(void) { /* sleep(); */}
// vector 15 IRQ1
__interrupt(vect=15) void INT_IRQ1(void) { /* sleep(); */}
// vector 16 IRQ2
__interrupt(vect=16) void INT_IRQ2(void) { /* sleep(); */}
// vector 17 IRQ3
__interrupt(vect=17) void INT_IRQ3(void) { /* sleep(); */}
// vector 18 WKP
__interrupt(vect=18) void INT_WKP(void) { /* sleep(); */}
// vector 19 RTC
__interrupt(vect=19) void INT_RTC(void) { /* sleep(); */}
// vector 20 Reserved

// vector 21 Reserved

// vector 22 Timer V
__interrupt(vect=22) void INT_TimerV(void) {intprog_tmv();}
// vector 23 SCI3
__interrupt(vect=23) void INT_SCI3(void) { /* sleep(); */}
// vector 24 IIC2
__interrupt(vect=24) void INT_IIC2(void) { /* sleep(); */}
// vector 25 ADI
__interrupt(vect=25) void INT_ADI(void) { /* sleep(); */}
// vector 26 Timer Z0
__interrupt(vect=26) void INT_TimerZ0(void) { /* sleep(); */}
// vector 27 Timer Z1
__interrupt(vect=27) void INT_TimerZ1(void) { /* sleep(); */}
// vector 28 Reserved

// vector 29 Timer B1
__interrupt(vect=29) void INT_TimerB1(void) { /* sleep(); */}
// vector 30 Reserved

// vector 31 Reserved

// vector 32 SCI3_2
__interrupt(vect=32) void INT_SCI3_2(void) { /* sleep(); */}

```

このプログラムは赤外線リモコン信号出力サブルーチンをアセンブラで作りました。下記にソースリストを掲載しますが、アセンブラのプログラムはフローチャートにすると、処理の内容をより理解しやすくなります(プログラムの学習にも最適です)。がんばってフローチャートに直してみてください。ちなみに実際のプログラムのときは、フローチャ

ートを書いてからソースリストにコーディングします。

```
-----
;
;
; FILE      :asmprg.src
; DATE      :Wed, Sep 09, 2009
; DESCRIPTION :Sub Program
; CPU TYPE  :H8/3687
;
; This file is programed by TOYO-LINX Co.,Ltd. / yKikuchi
;
-----

.export      _IR_sig_out      ;Cからコールされる,"IR_sig_out(N);"
.section P, CODE, ALIGN=2

;*****
; 定数定義
;*****
PDR6      .equ    h'FFD9    ;ポートデータレジスタ 6
PCR6      .equ    h'FFE9    ;ポートコントロールレジスタ 6

;*****
; 赤外線リモコン信号出力 (NECフォーマット準拠)
;*****
;
; R0L: リモコン信号データ
;*****
_IR_sig_out:
    push.l    er2      ;ER0, ER1はCからコールした段階で自動的にPUSHされる
    push.l    er3
    push.l    er4
    push.l    er5
    push.l    er6

    mov.b     r0l, @IRBuf_2    ;データコード
    not.b     r0l              ;データコードの反転
    mov.b     r0l, @IRBuf_3
    mov.b     #h'00, r0l      ;カスタムコード-0
    mov.b     r0l, @IRBuf_0
    mov.b     #h'ff, r0l      ;カスタムコード-1
    mov.b     r0l, @IRBuf_1

    mov.l     #IRBuf_0, er1
    mov.w     #4, r2
    bsr      IR_signal_out:16

    pop.l     er6
    pop.l     er5
    pop.l     er4
    pop.l     er3
    pop.l     er2      ;ER0, ER1はCにリターンするとき自動的にPOPされる
    rts

;*****
```

```

; 赤外線リモコン信号出力
;*****
IR_signal_out:
    push.l  er1
    push.l  er2

    orc     #'80, ccr           ;割込み禁止

    bsr     IR_leader:16       ;リーダー

IR_signal_out_01:
    mov.b   @er1, r3l
    mov.b   #8, r3h
IR_signal_out_02:
    rotr.b  r3l
    bcs     IR_signal_out_03
    bsr     IR_bit0:16
    bra     IR_signal_out_04
IR_signal_out_03:
    bsr     IR_bit1:16
IR_signal_out_04:
    dec.b   r3h
    bne     IR_signal_out_02
    inc.l   #1, er1
    dec.w   #1, r2
    bne     IR_signal_out_01

    bsr     IR_trailer:16      ;トレーラ

    andc    #'7f, ccr         ;割込み許可

    pop.l   er2
    pop.l   er1
    rts

;*****
; IR / 38KHzのパルス (1周期)
;*****
IR_pulse_38khz:
    mov.b   @PDR6, r0l
    or.b    #'01, r0l
    mov.b   r0l, @PDR6

    mov.b   #42, r0l
IR_pulse_38khz_01:
    dec.b   r0l
    bne     IR_pulse_38khz_01

    mov.b   @PDR6, r0l
    and.b   #'fe, r0l
    mov.b   r0l, @PDR6

    mov.b   #39, r0l
IR_pulse_38khz_02:
    dec.b   r0l

```

```

    bne    IR_pulse_38khz_02

    rts

;*****
;   IR / 38KHzのパルス (1周期) と同じ時間Low
;*****
IR_non_pulse_38khz:
    mov.b  @PDR6, r0l
    and.b  #'fe, r0l
    mov.b  r0l, @PDR6

    mov.b  #42, r0l
IR_non_pulse_38khz_01:
    dec.b  r0l
    bne    IR_non_pulse_38khz_01

    mov.b  @PDR6, r0l
    and.b  #'fe, r0l
    mov.b  r0l, @PDR6

    mov.b  #39, r0l
IR_non_pulse_38khz_02:
    dec.b  r0l
    bne    IR_non_pulse_38khz_02

    rts

;*****
;   IR / 赤外線信号 bit=0
;*****
IR_bit0:
    mov.b  #21, r0h
IR_bit0_01:
    bsr    IR_pulse_38khz
    dec.b  r0h
    bne    IR_bit0_01

    mov.b  #21, r0h
IR_bit0_02:
    bsr    IR_non_pulse_38khz
    dec.b  r0h
    bne    IR_bit0_02

    rts

;*****
;   IR / 赤外線信号 bit=1
;*****
IR_bit1:
    mov.b  #21, r0h
IR_bit1_01:
    bsr    IR_pulse_38khz
    dec.b  r0h
    bne    IR_bit1_01

```

```

    mov. b    #64, r0h
IR_bit1_02:
    bsr      IR_non_pulse_38khz
    dec. b   r0h
    bne      IR_bit1_02

    rts

;*****
;   IR / リーダ部
;*****
IR_leader:
    mov. b   #171, r0h
IR_leader_01:
    bsr      IR_pulse_38khz
    dec. b   r0h
    bne      IR_leader_01

    mov. b   #171, r0h
IR_leader_03:
    bsr      IR_pulse_38khz
    dec. b   r0h
    bne      IR_leader_03

    mov. b   #171, r0h
IR_leader_02:
    bsr      IR_non_pulse_38khz
    dec. b   r0h
    bne      IR_leader_02

    rts

;*****
;   IR / トレーラ部
;*****
IR_trailer:
    mov. b   #21, r0h
IR_trailer_01:
    bsr      IR_pulse_38khz
    dec. b   r0h
    bne      IR_trailer_01

    mov. l   #133333, er0      ;40ms
IR_trailer_02:
    dec. l   #1, er0
    bne      IR_trailer_02

    rts

;=====
;   ワークエリア
;=====
.section B, data, ALIGN=2

```



```

IRBuf_0 . res. b      1 ;カスタムコード-0
IRBuf_1 . res. b      1 ;カスタムコード-1
IRBuf_2 . res. b      1 ;データコード
IRBuf_3 . res. b      1 ;データコードの反転

. end

```

■ 赤外リモコン受信機

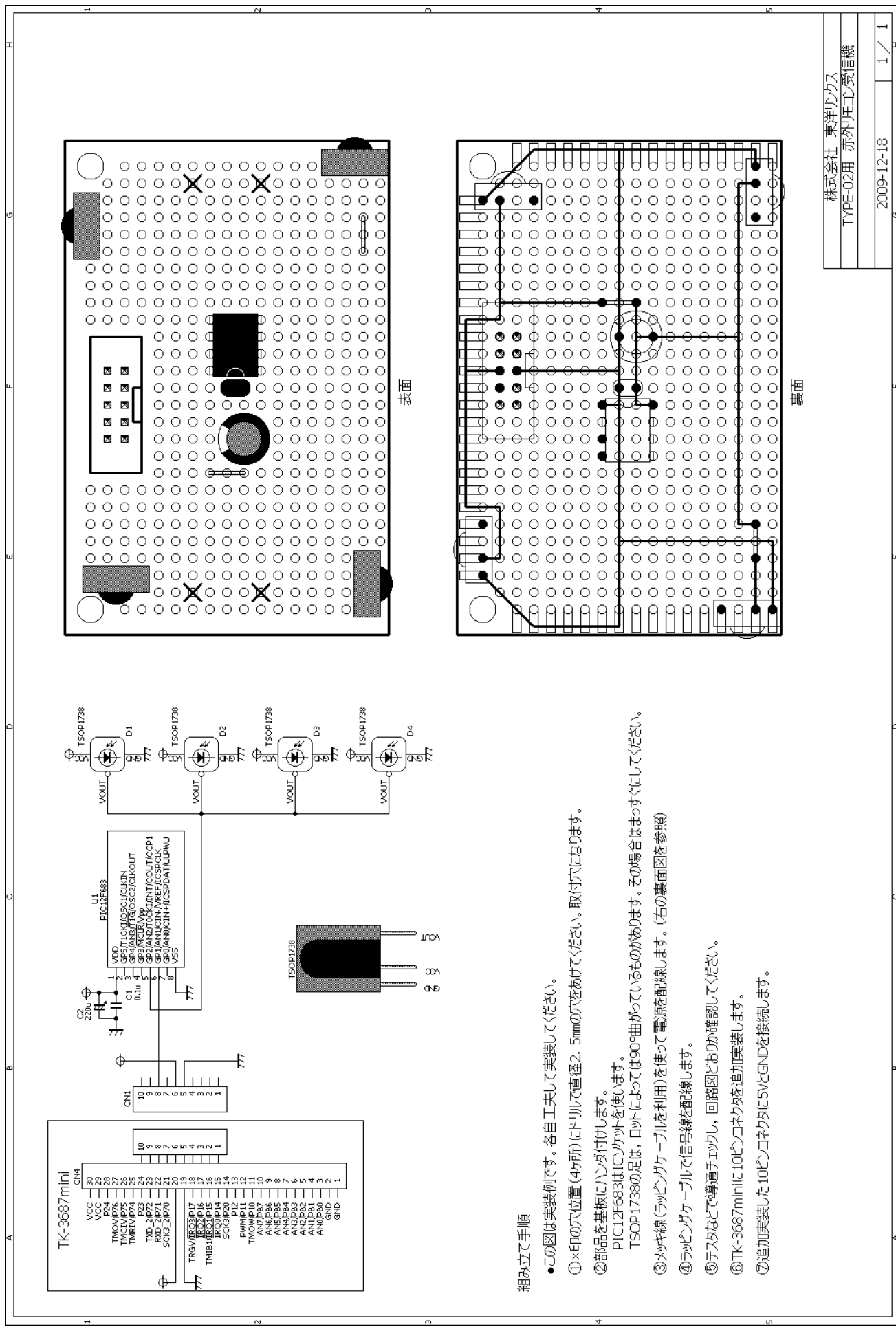
最初に、部品表と比較して部品が全てそろっているか確認しましょう。部品によっては相当品使用の場合もあります。(部品が足りないときは巻末記載の連絡先までお問い合わせください。)

赤外リモコン受信機 組み立てキット

	部品番号	型名, 規格	メーカー	数量	付属数量	備考
1	■赤外リモコン受信機基板					
2	U1&ICソケット	PIC12F683	マイクロチップテクノロジ	1	1	プログラム書き込み済み
3	D1,2,3,4	IS1U60 TSOP1738 PL-IRM0208-A538	SHARP Vishay Telefunken PARA LIGHT	4	4	*1
4	C1	0.1 μ F		1	1	
5	C2	220~470 μ F		1	1	
6	ラッピングケーブル	1m		1	1	*2, メッキ線として使用
7	メッキ線			0	0	ハンダ面結線用 *2
8	ユニバーサル基板	47.5 × 72mm	東洋リンクス	1	1	B6093を1/2にカット
9						
10	■機構部品					
11	ネジ	2-10mm		4	4	
12	スペーサ	H=5mm		4	4	
13						
14	■その他					
15	接続ケーブル	10芯 × 8cm~9cm		1	1	両端コネクタ付き
16		HIF3FC-10PA-2.54DSA	HRS	1	1	*1, TK-3687miniに実装
17	電源配線用ケーブル	10cm		1	1	TK-3687mini電源配線用
18						

(*1)相当品を使用することがあります。
(*2)ラッピングケーブルの被覆をはがし2本をよじて使用します。また、抵抗やコンデンサの足も流用できます。

回路図と実装図は次のとおりです。



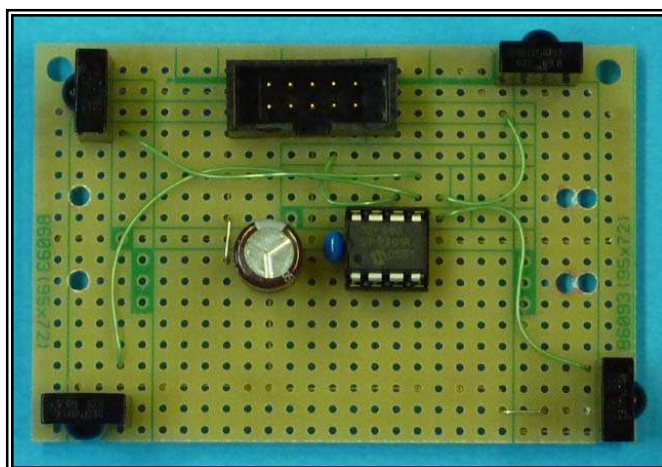
株式会社 東洋リカス	
TYPE-02用 赤外線コン受信機	
2009-12-18	1 / 1

この実装図は例です。取り付け穴も含め、各自工夫して実装していただいてもかまいません。特に、この例は顔のパーツ(プラ版)をつけずにTYPE-02の【補強】フレームに受信機を取り付けることを前提にしています。顔のパーツをつける場合は取り付け方法を工夫してください。

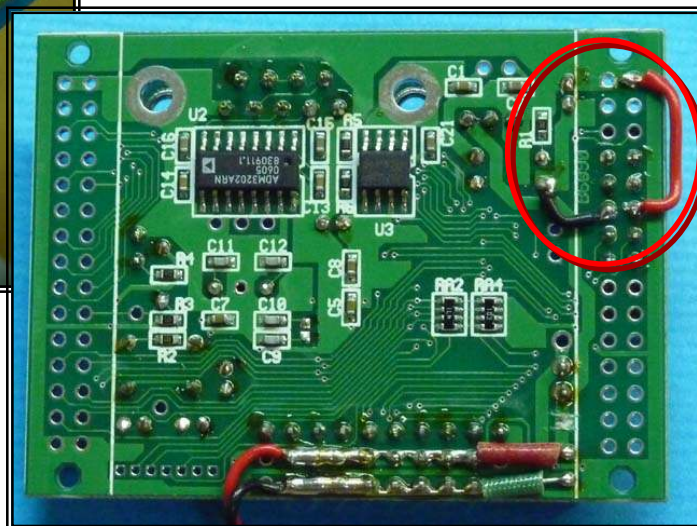
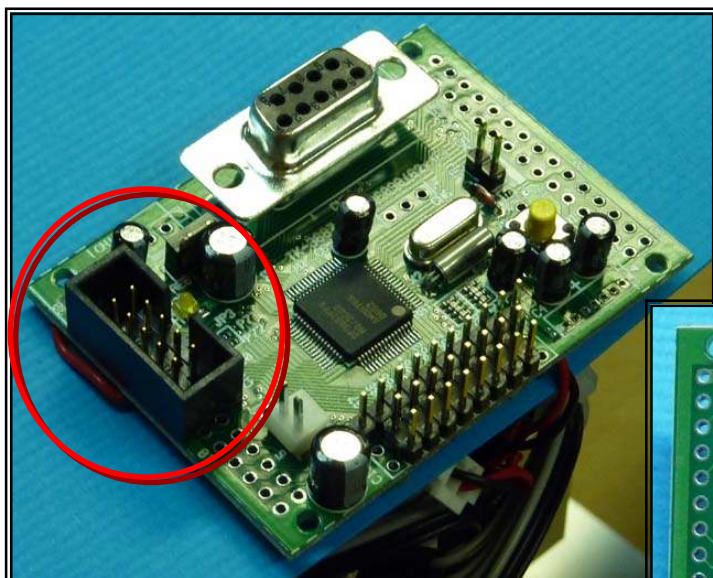
ユニバーサル基板に取り付け穴をあけます。ドリルで2.5Φの穴を4ヶ所開けてください。開ける場所は実装図の×印のところです。

実装図の表面(部品面)を見てユニバーサル基板に部品を載せます。次に、実装図の裏面(半田面)を見てメッキ線での配線を済ませてください。最後に信号線をラッピングケーブルで配線します。

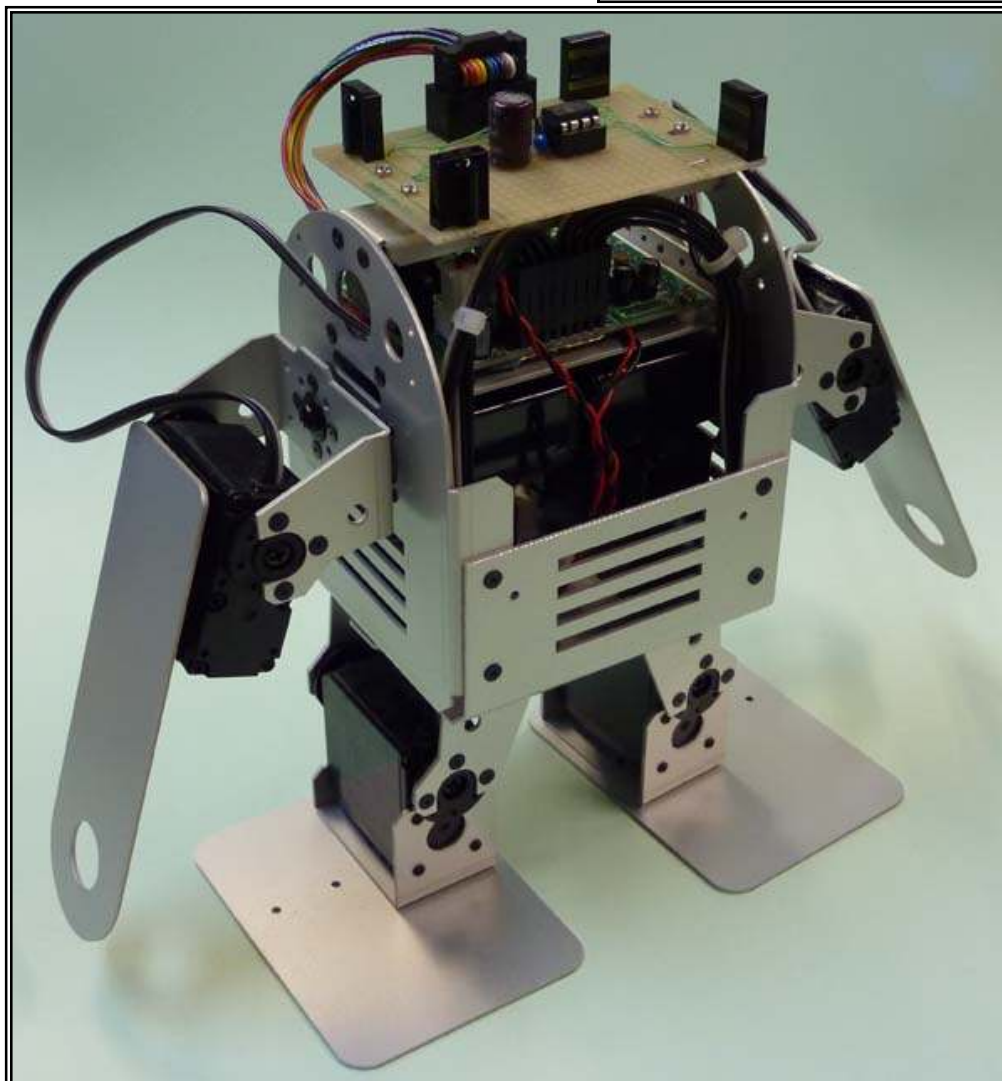
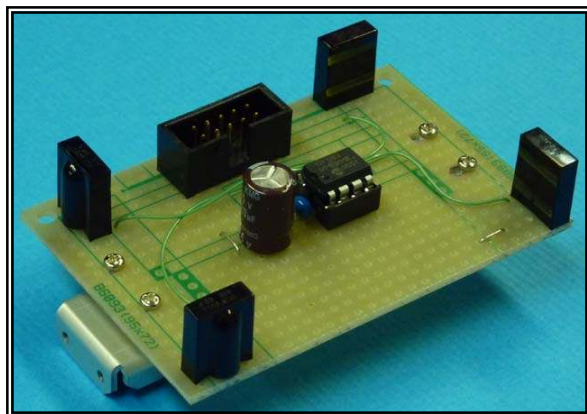
配線が終了したら、回路図どおり配線されているかももう一度確認して下さい。確認方法は、テスタで部品面の端子間の抵抗を測り、導通があるか、すなわち0Ωか否かで判断します。また、半田付けがきちんと行なわれているか見ておきましょう。動かない原因の大部分は配線ミスと半田付け不良です。



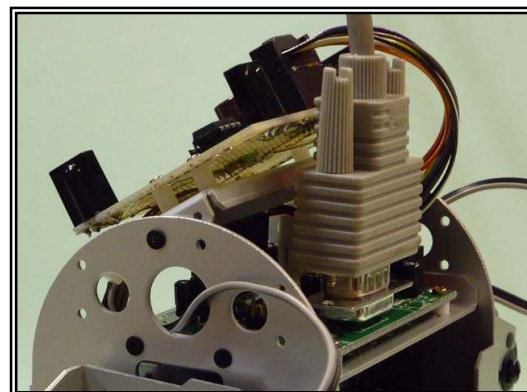
次にTK-3687miniに10ピンコネクタを実装します。TK-3687miniのCN4の15番ピンに、10ピンコネクタの1番ピンをあわせて実装します(CN4の24番ピンが10ピンコネクタの10番ピンにあいます)。最後に、実装した10ピンコネクタの6番ピンを5Vに、5番ピンをGNDに接続します。キット付属の赤黒のケーブルを使ってください。



では、全体を組み立てましょう。10ピンコネクタ取り付けのために【補強】フレームと TK-3687mini を取り外したと思います。まず TK-3687mini を取り付けてください。付属のコネクタ付 10 芯ケーブルも挿しておきます。次に、リモコン受信機を【補強】フレームに取り付けます。【補強】フレームに 8ヶ所タップが切ってあるはずですが、そのうち両端の 4ヶ所を使います。最後に【補強】フレームを TYPE-02 に取り付け、10 芯ケーブルを受信機のコネクタに挿せば組み立て終了です。



では、プログラムをダウンロードしましょう。TK-3687mini に FDT で“[ir_remocon_robo. mot](#)”をダウンロードします。FDT については CD の「¥ 始めにお読みください ¥」フォルダの「[モニタプログラム書き込み手順書. pdf](#)」を参考にしてください。なお、シリアルケーブルが赤外リモコン受信機にぶつかるので、ダウンロードする際に【補強】フレームを写真のようにずらしてシリアルケーブルを挿してください。



PIC12F683 に書き込まれているプログラムは、受信した NEC フォーマットの赤外リモコン信号からデータ部を取り出し、そのデータを調歩同期式のシリアル信号で送信するものです。このシリアル信号は、TK-3687mini のシリアルコミュニケーションインターフェース、SCI3_2 の RXD_2 につながっています。受信データをもとに歩行パターンテーブルを選択し、TYPE-02 を動かします。ソースリストは次のとおりです。(歩行プログラムの考え方は「5. スムージング付き二足歩行」と同じです。)

```

/*****/
/*                                     */
/* FILE      :ir_remocon_robo.c      */
/* DATE      :Thu, Dec 10, 2009      */
/* DESCRIPTION:Main Program          */
/* CPU TYPE   :H8/3687                */
/*                                     */
/* This file is programed by TOYO-LINX Co.,Ltd. / yKikuchi */
/*                                     */
/*****/

-----
履歴

-----
2009-12-10 : プラグラム開始
-----

/*****
インクルードファイル
-----
#include <machine.h> //H8特有の命令を使う
#include "iodefine.h" //内蔵I/Oのラベル定義
#include "binary.h" //Cで2進数を使うための定義

/*****
定数の定義 (直接指定)
-----
#define OK 0 //戻り値
#define NG -1 //戻り値

//ロボット -----
#define STEP 6 //歩行パターンステップ数
#define STEP_MAX 8 //最大パターンステップ数
#define TABLE_MAX 8 //最大パターンテーブル数

#define SERVO_STEP 128 //サーボモータの1シーケンスにおける移行段階

//通信 -----
#define RXBUF_SIZE 256 //RxBufのサイズ

//ソフトウェアタイマ -----
#define T0Const 1000 //T0(1000ms) 通信が途絶えてから停止するまでの時間 (リモコンモード)
#define T1Const 0 //T1(ms)
#define T2Const 0 //T2(ms)
#define T3Const 0 //T3(ms)
#define T4Const 0 //T4(ms)
#define T5Const 0 //T5(ms)
#define T6Const 0 //T6(ms)

```

```

#define      T7Const      0      //T7(ms)

/*****
定数エリアの定義 (ROM)
*****/
// 歩行パターン
const int SequenceTable[TABLE_MAX][STEP_MAX+1][8] = {
// テーブル(0)
{
//      左膝      左股      左肩      左腕      右膝      右股      右肩      右腕      //ステップ数
  {6, 0, 0, 0, 0, 0, 0, 0, }, //step1
  {0, 150, -570, 300, 0, 150, 570, -300}, //step2
  {-125, 150, -570, 300, -400, 150, 570, -300}, //step3
  {-125, -150, -570, 300, -320, -150, 570, -300}, //step4
  {0, -150, -570, 300, 0, -150, 570, -300}, //step5
  {400, -150, -570, 300, 125, -150, 570, -300}, //step6
  {320, 150, -570, 300, 125, 150, 570, -300}, //step7
  {0, 0, 0, 0, 0, 0, 0, 0}, //step8
  {0, 0, 0, 0, 0, 0, 0, 0} //step8
},
// テーブル(1)
{
//      左膝      左股      左肩      左腕      右膝      右股      右肩      右腕      //ステップ数
  {6, 0, 0, 0, 0, 0, 0, 0, }, //step1
  {0, 150, -570, 300, 0, 150, 570, -300}, //step2
  {400, 150, -570, 300, 125, 150, 570, -300}, //step3
  {320, -150, -570, 300, 125, -150, 570, -300}, //step4
  {0, -150, -570, 300, 0, -150, 570, -300}, //step5
  {-125, -150, -570, 300, -400, -150, 570, -300}, //step6
  {-125, 150, -570, 300, -320, 150, 570, -300}, //step7
  {0, 0, 0, 0, 0, 0, 0, 0}, //step8
  {0, 0, 0, 0, 0, 0, 0, 0} //step8
},
// テーブル(2)
{
//      左膝      左股      左肩      左腕      右膝      右股      右肩      右腕      //ステップ数
  {6, 0, 0, 0, 0, 0, 0, 0, }, //step1
  {0, 150, -570, 300, 0, 200, 570, -300}, //step2
  {-125, 150, -570, 300, -400, 200, 570, -300}, //step3
  {-125, -150, -570, 300, -320, -200, 570, -300}, //step4
  {0, -150, -570, 300, 0, -200, 570, -300}, //step5
  {400, -150, -570, 300, 125, -200, 570, -300}, //step6
  {320, 150, -570, 300, 125, 200, 570, -300}, //step7
  {0, 0, 0, 0, 0, 0, 0, 0}, //step8
  {0, 0, 0, 0, 0, 0, 0, 0} //step8
},
// テーブル(3)
{
//      左膝      左股      左肩      左腕      右膝      右股      右肩      右腕      //ステップ数
  {6, 0, 0, 0, 0, 0, 0, 0, }, //step1
  {0, 200, -570, 300, 0, 150, 570, -300}, //step2
  {-125, 200, -570, 300, -400, 150, 570, -300}, //step3
  {-125, -200, -570, 300, -320, -150, 570, -300}, //step4
  {0, -200, -570, 300, 0, -150, 570, -300}, //step5
  {400, -200, -570, 300, 125, -150, 570, -300}, //step5
}
}

```



```

        {320, 200, -570, 300, 125, 150, 570, -300}, //step6
        {0, 0, 0, 0, 0, 0, 0, 0}, //step7
        {0, 0, 0, 0, 0, 0, 0, 0} //step8
    },
// テーブル(4)
{
// 左膝 左股 左肩 左腕 右膝 右股 右肩 右腕
{6,0,0,0,0,0,0,0}, //ステップ数
{0, 0, 0, 0, 0, 0, 0, 0}, //step1
{0, 0, 0, 0, 0, 0, 0, 0}, //step2
{0, 0, 0, 0, 0, 0, 0, 0}, //step3
{0, 0, 0, 0, 0, 0, 0, 0}, //step4
{0, 0, 0, 0, 0, 0, 0, 0}, //step5
{0, 0, 0, 0, 0, 0, 0, 0}, //step6
{0, 0, 0, 0, 0, 0, 0, 0}, //step7
{0, 0, 0, 0, 0, 0, 0, 0} //step8
},
// テーブル(5)
{
// 左膝 左股 左肩 左腕 右膝 右股 右肩 右腕
{6,0,0,0,0,0,0,0}, //ステップ数
{0, 0, 0, 0, 0, 0, 0, 0}, //step1
{0, 0, 0, 0, 0, 0, 0, 0}, //step2
{0, 0, 0, 0, 0, 0, 0, 0}, //step3
{0, 0, 0, 0, 0, 0, 0, 0}, //step4
{0, 0, 0, 0, 0, 0, 0, 0}, //step5
{0, 0, 0, 0, 0, 0, 0, 0}, //step6
{0, 0, 0, 0, 0, 0, 0, 0}, //step7
{0, 0, 0, 0, 0, 0, 0, 0} //step8
},
// テーブル(6)
{
// 左膝 左股 左肩 左腕 右膝 右股 右肩 右腕
{6,0,0,0,0,0,0,0}, //ステップ数
{0, 0, 0, 0, 0, 0, 0, 0}, //step1
{0, 0, 0, 0, 0, 0, 0, 0}, //step2
{0, 0, 0, 0, 0, 0, 0, 0}, //step3
{0, 0, 0, 0, 0, 0, 0, 0}, //step4
{0, 0, 0, 0, 0, 0, 0, 0}, //step5
{0, 0, 0, 0, 0, 0, 0, 0}, //step6
{0, 0, 0, 0, 0, 0, 0, 0}, //step7
{0, 0, 0, 0, 0, 0, 0, 0} //step8
},
// テーブル(7)
{
// 左膝 左股 左肩 左腕 右膝 右股 右肩 右腕
{6,0,0,0,0,0,0,0}, //ステップ数
{0, 0, 0, 0, 0, 0, 0, 0}, //step1
{0, 0, 0, 0, 0, 0, 0, 0}, //step2
{0, 0, 0, 0, 0, 0, 0, 0}, //step3
{0, 0, 0, 0, 0, 0, 0, 0}, //step4
{0, 0, 0, 0, 0, 0, 0, 0}, //step5
{0, 0, 0, 0, 0, 0, 0, 0}, //step6
{0, 0, 0, 0, 0, 0, 0, 0}, //step7
{0, 0, 0, 0, 0, 0, 0, 0} //step8
}

```

```

}
};

/*****
グローバル変数の定義とイニシャライズ (RAM)
*****/
// サーボモータの中心 (カウント値) /理論値は1500 $\mu$ s  $\div$  200ns = 7500
/*
// 筆者の例 PirkusR TYPE-02
unsigned int HomePos[8] = {7400 //P60, 左膝
                          ,7500 //P61, 左股
                          ,7350 //P62, 左肩
                          ,7500 //P63, 左腕
                          ,7250 //P64, 右膝
                          ,7650 //P65, 右股
                          ,7800 //P66, 右肩
                          ,7750 //P67, 右腕
                          };

*/
// 筆者の例 PirkusR TYPE-02 II
unsigned int HomePos[8] = {7500 //P60, 左膝
                          ,6030 //P61, 左股
                          ,7600 //P62, 左肩
                          ,7500 //P63, 左腕
                          ,7500 //P64, 右膝
                          ,9050 //P65, 右股
                          ,7500 //P66, 右肩
                          ,7500 //P67, 右腕
                          };

// サーボモータの位置 (パルス幅, 中心からの $\pm$   $\mu$  秒)
int ServoPos[8] = {0, 0, 0, 0, 0, 0, 0, 0};
// サーボモータの現在カウント値
long ServoCnt[8]; //上位16ビットをタイムZにセットする
// サーボモータの目標カウント値
long GoalCnt[8];
// サーボモータカウント値移行時の1段階加算値
long Add1Step[8];
// サーボモータカウント値移行カウンタ
unsigned int MoveCnt;
// サーボモータカウント値移行時のシーケンス
unsigned char SequenceStage = 0; // 0:停止
// 1:移行中
// 2:終了

// ロボット制御
unsigned char StepCnt; //歩行ステップカウンタ
unsigned char StepCnst; //歩行ステップ数
int PatternTable = -1;
unsigned char RoboFlag = 0;

// 通信に関係した変数 -----
unsigned char RxBuf[RXBUF_SIZE]; //受信バッファ
unsigned char *RxBufWrPnt; //受信バッファライトポインタ
unsigned char *RxBufRdPnt; //受信バッファリードポインタ

```

```

unsigned char *RxBufMin; //受信バッファの最初
unsigned char *RxBufMax; //受信バッファの最後

// ソフトウェアタイマに関係した変数 -----
struct SoftTimer{ //ソフトウェアタイマの構造体タグ
    unsigned char Status; //タイマステータス
                        // 0:停止中(タイマ未使用)
                        // 1:スタート指令
                        // 2:カウント中
                        // 3:カウント終了
    unsigned long Count; //タイマカウンタ
};
struct SoftTimer TimT0; //T0タイマ
struct SoftTimer TimT1; //T1タイマ
struct SoftTimer TimT2; //T2タイマ
struct SoftTimer TimT3; //T3タイマ
struct SoftTimer TimT4; //T4タイマ
struct SoftTimer TimT5; //T5タイマ
struct SoftTimer TimT6; //T6タイマ
struct SoftTimer TimT7; //T7タイマ

unsigned long T0 = T0Const; //T0タイマカウンタ初期値
unsigned long T1 = T1Const; //T1タイマカウンタ初期値
unsigned long T2 = T2Const; //T2タイマカウンタ初期値
unsigned long T3 = T3Const; //T3タイマカウンタ初期値
unsigned long T4 = T4Const; //T4タイマカウンタ初期値
unsigned long T5 = T5Const; //T5タイマカウンタ初期値
unsigned long T6 = T6Const; //T6タイマカウンタ初期値
unsigned long T7 = T7Const; //T7タイマカウンタ初期値

/*****
関数の定義
*****/
void init_servo(void);
void init_tmz(void);
void intprog_tmz0(void);
void main(void);
void robocon(void);
void servo_set(void);
void wait(void);

void init_tmb1(void);

int put_rxbuf(unsigned char);
void rxerr_sci3_2(void);
void rxdata_sci3_2(void);
unsigned char rxone(void);
void txone(unsigned char);
int get_rxbuf(unsigned char *);
void init_rxbuf(void);
void init_sci3_2(void);
void intprog_sci3_2(void);

void init_soft_timer(void);
void dec_soft_timer(struct SoftTimer *,unsigned long);

```

```

/*****
メインプログラム
*****/
void main(void)
{
    // イニシャライズ -----
    IO.PCR5 = 0xff;

    init_servo();
    init_tmz();
    init_soft_timer();
    init_tmb1();
    init_rxbuf();
    init_sci3_2();

    // メインループ -----
    while(1) {
        robocon();
    }
}

/*****
ロボット制御
*****/
void robocon(void)
{
    int pat, l, j;
    unsigned char dt;    //リモコン受信データ

    if (get_rxbuf(&dt)==OK) {
        TimT0.Status = 0;    //タイマストップ
        switch (dt) {
            case _00001000B:
                pat = 0;
                break;
            case _00000100B:
                pat = 1;
                break;
            case _00000010B:
                pat = 2;
                break;
            case _00000001B:
                pat = 3;
                break;
            default:
                pat = -1;
                break;
        }
        if ((pat!=-1)&&(pat!=PatternTable)) {
            PatternTable = pat;
            StepCnt = 0;
            StepCnst = SequenceTable[PatternTable][0][0];
            RoboFlag = 1;
        }
    }
}

```

```

    TimT0.Status = 1;    //タイマスタート
}

if ((RoboFlag==1)&&(SequenceStage!=1)) {
    for (j=0; j<8; j++){
        ServoPos[j] = SequenceTable[PatternTable][StepCnt+1][j];
    }
    servo_set();
    StepCnt = StepCnt + 1;
    if (StepCnt>=StepCnst)    {StepCnt = 0;}
}

if (TimT0.Status==3) {    //通信が途絶えた
    TimT0.Status = 0;    //タイマストップ
    PatternTable = -1;
    RoboFlag = 0;    //停止
}

IO.PDR5.BYTE = StepCnt;
}

/*****
サーボモータ イニシャライズ
*****/
void init_servo(void)
{
    int l;

    for (l=0; l<8; l++){
        ServoCnt[l] = HomePos[l] * 0x10000;
    }
}

/*****
サーボモータデータセット
*****/
void servo_set(void)
{
    int l;

    for (l=0; l<8; l++){
        GoalCnt[l] = (HomePos[l] + ServoPos[l] * 5) * 0x10000;
        Add1Step[l] = (GoalCnt[l] - ServoCnt[l]) / SERVO_STEP;
    }

    MoveCnt = SERVO_STEP;
    SequenceStage = 1;
}

/*****
タイマZ イニシャライズ
*****/
void init_tmz(void)
{
    TZ.TSTR.BYTE = 0x00;    //TCNT0,1 停止
}

```

```

TZ0. TCR. BYTE = 0xe2; //同期クリア, φ/4
TZ1. TCR. BYTE = 0xe2; //同期クリア, φ/4

TZ. TMDR. BYTE = 0x0f; //TCNT0, 1は同期動作

TZ. TOCR. BYTE = 0xff; //初期出力=1

TZ. TOER. BYTE = 0x00; //出力端子イネーブル

TZ0. TIOA. BYTE = 0x99; //GRA, GRBはコンペアマッチで0出力
TZ0. TIOB. BYTE = 0x99; //GRC, GRDはコンペアマッチで0出力
TZ1. TIOA. BYTE = 0x99; //GRA, GRBはコンペアマッチで0出力
TZ1. TIOB. BYTE = 0x99; //GRC, GRDはコンペアマッチで0出力

TZ0. TSR. BYTE = 0x00; //割込みフラグクリア
TZ1. TSR. BYTE = 0x00; //割込みフラグクリア

TZ0. TIER. BYTE = 0x10; //オーバーフローインターラプトイネーブル
TZ1. TIER. BYTE = 0x00; //インターラプトディセーブル

TZ0. GRA = HomePos[0]; //カウント初期値
TZ0. GRB = HomePos[1]; //カウント初期値
TZ0. GRC = HomePos[2]; //カウント初期値
TZ0. GRD = HomePos[3]; //カウント初期値
TZ1. GRA = HomePos[4]; //カウント初期値
TZ1. GRB = HomePos[5]; //カウント初期値
TZ1. GRC = HomePos[6]; //カウント初期値
TZ1. GRD = HomePos[7]; //カウント初期値

TZ0. TCNT = 0x0000; //TCNT0=0
TZ1. TCNT = 0x0000; //TCNT1=0

TZ. TSTR. BYTE = 0x03; //TCNT0, 1 カウントスタート
}

/*****
タイマZ チャンネル0 割込み
*****/
#pragma regsave (intprog_tmz0)
void intprog_tmz0(void)
{
    int l;

    //タイマZ オーバフローインターラプトフラグ クリア
    TZ0. TSR. BIT. OVF =0;

    //TCNT0, 1 停止
    TZ. TSTR. BYTE = 0x00;

    //カウント値の加算
    if (SequenceStage==1) {
        for (l=0; l<8; l++){
            ServoCnt[l] = ServoCnt[l] + Add1Step[l];
        }
    }
}

```



```

        MoveCnt--; if (MoveCnt==0) {SequenceStage = 2;}
    }

    //タイマZにカウント値をセット
    TZ0.GRA = ServoCnt[0] / 0x10000;
    TZ0.GRB = ServoCnt[1] / 0x10000;
    TZ0.GRC = ServoCnt[2] / 0x10000;
    TZ0.GRD = ServoCnt[3] / 0x10000;
    TZ1.GRA = ServoCnt[4] / 0x10000;
    TZ1.GRB = ServoCnt[5] / 0x10000;
    TZ1.GRC = ServoCnt[6] / 0x10000;
    TZ1.GRD = ServoCnt[7] / 0x10000;

    //出力を1にする
    TZ.TOCR.BYTE = 0xff;

    //TCNT0,1 カウントスタート
    TZ0.TCNT = 0x0000;
    TZ1.TCNT = 0x0000;
    TZ.TSTR.BYTE = 0x03;
}

/*****
    ウェイト(1000ms)
*****/
void wait(void)
{
    unsigned long l;

    for (l=0;l<333333;l++) {}
}

/*****
    タイマB1 イニシャライズ
*****/
void init_tmb1(void)
{
    TB1.TMB1.BYTE      = 0xf9;    //オートリロード, 内部クロックφ/2048
    TB1.TLB1           = 0-97;    //周期=10ms(100Hz)
    IRR2.BIT.IRRTB1    = 0;      //タイマB1割込み要求フラグ クリア
    IENR2.BIT.IENTB1   = 1;      //タイマB1割込み要求イネーブル
}

/*****
    タイマB1 割込み(10ms)
*****/
#pragma regsave (intprog_tmb1)
void intprog_tmb1(void)
{
    //タイマB1割込み要求フラグ クリア
    IRR2.BIT.IRRTB1 = 0;

    //ソフトウェアタイマ T0
    if (TimT0.Status==1 || TimT0.Status==2) {
        dec_soft_timer (&TimT0, T0/10);
    }
}

```

```

}
//ソフトウェアタイマ T1
if (TimT1.Status==1 || TimT1.Status==2) {
    dec_soft_timer (&TimT1, T1/10);
}
//ソフトウェアタイマ T2
if (TimT2.Status==1 || TimT2.Status==2) {
    dec_soft_timer (&TimT2, T2/10);
}
//ソフトウェアタイマ T3
if (TimT3.Status==1 || TimT3.Status==2) {
    dec_soft_timer (&TimT3, T3/10);
}
//ソフトウェアタイマ T4
if (TimT4.Status==1 || TimT4.Status==2) {
    dec_soft_timer (&TimT4, T4/10);
}
//ソフトウェアタイマ T5
if (TimT5.Status==1 || TimT5.Status==2) {
    dec_soft_timer (&TimT5, T5/10);
}
//ソフトウェアタイマ T6
if (TimT6.Status==1 || TimT6.Status==2) {
    dec_soft_timer (&TimT6, T6/10);
}
//ソフトウェアタイマ T7
if (TimT7.Status==1 || TimT7.Status==2) {
    dec_soft_timer (&TimT7, T7/10);
}
}

/*****
ソフトウェアタイマのデクリメント
-----
引数    *pst    ソフトウェアタイマ構造体のポインタ
        initial   タイマカウンタの初期値
*****/
void dec_soft_timer(struct SoftTimer *pst,unsigned long initial)
{
    if (pst->Status==1) {          //タイマスタート指令
        pst->Status = 2;          //カウント中セット
        pst->Count = initial;    //タイマカウンタ初期化
    }
    pst->Count--;                //カウンタ-1
    if (pst->Count==0)           //カウンタが0になった
        pst->Status = 3;        //カウント終了セット
}

/*****
ソフトウェアタイマのイニシャライズ
*****/
void init_soft_timer(void)
{
    TimT0.Status = 0; TimT1.Status = 0; TimT2.Status = 0; TimT3.Status = 0;
    TimT4.Status = 0; TimT5.Status = 0; TimT6.Status = 0; TimT7.Status = 0;
}

```

```

}

/*****
  RxBuf の初期化
*****/
void init_rxbuf(void)
{
  RxBufRdPnt = RxBuf;          //受信バッファリードポインタセット
  RxBufWrPnt = RxBufRdPnt;     //受信バッファライトポインタセット
  RxBufMin   = RxBuf;          //受信バッファの最初をセット
  RxBufMax   = RxBuf+RXBUF_SIZE-1; //受信バッファの最後をセット
}

/*****
  RxBuf にデータを格納する
*****/

```

引数	data	RxBufに格納するデータ
戻り値	OK	格納できた
	NG	エラー, バッファからあふれた

```

*****/
int put_rxbuf(unsigned char data)
{
  int ret_code;    //OK or NG

  if ((RxBufRdPnt==RxBufMin && RxBufWrPnt==RxBufMax) || RxBufWrPnt==RxBufRdPnt-1)
    ret_code = NG;    //バッファサイズを越えた
  else{
    *RxBufWrPnt = data;
    RxBufWrPnt++;
    if (RxBufWrPnt>RxBufMax)
      RxBufWrPnt=RxBufMin; //ライトポインタを先頭に戻す
    ret_code = OK;
  }
  return ret_code;
}

/*****
  RxBuf からデータを取り出す
*****/

```

引数	*pd	取り出したデータをセットするポインタ
戻り値	OK	取り出せた
	NG	エラー, バッファに何も入っていない

```

*****/
int get_rxbuf(unsigned char *pd)
{
  int ret_code;

  if (RxBufWrPnt==RxBufRdPnt)
    ret_code = NG;    //バッファに何も入っていない
  else{
    *pd = *RxBufRdPnt;
    RxBufRdPnt++;
    if (RxBufRdPnt>RxBufMax)

```

```

        RxBufRdPnt=RxBufMin: //リードポインタを先頭に戻す
        ret_code = OK;
    }
    return ret_code;
}

/*****
SCI3_2 イニシャライズ
*****/
void init_sci3_2(void)
{
    #define      MHz      20          // Clock=20MHz
    #define      BAUD     38400       // BaudRate
    #define      BTR (MHz*1000000)/BAUD/32-1
    #define      WAIT_1B (MHz*1000000)/6/BAUD

    unsigned long l;

//  IO.PMR1.BIT.TXD2 = 1;          //TxD_2端子イネーブル
    SCI3_2.SCR3.BYTE = 0x00;       //動作停止
    SCI3_2.SMR.BYTE = 0x00;        //調歩同期, 8bit, NonParity, StopBit=1
    SCI3_2.BRR = BTR;              //ビットレート
    for (l=0; l<WAIT_1B; l++) {};   //1bit期間 wait
    SCI3_2.SCR3.BYTE = 0x50;       //受信イネーブル, 受信割り込みイネーブル
}

/*****
SCI3_2 割り込み (割り込み要因によって振り分ける)
*****/
#pragma regsave (intprog_sci3_2)
void intprog_sci3_2(void)
{
    if (SCI3_2.SSR.BIT.OER==1
        || SCI3_2.SSR.BIT.FER==1
        || SCI3_2.SSR.BIT.PER==1)    rxerr_sci3_2();    //受信エラー
    else if (SCI3_2.SSR.BIT.RDRF==1) rxdata_sci3_2();   //受信
// else if (SCI3_2.SSR.BIT.TEND==1) txend_sci3_2();    //送信終了
// else if (SCI3_2.SSR.BIT.TDRE==1) txdata_sci3_2();   //送信
}

/*****
SCI3_2 受信エラー割り込み
*****/
void rxerr_sci3_2(void)
{
    unsigned char dmy;

    SCI3_2.SSR.BYTE = SCI3_2.SSR.BYTE & 0x87; //エラーフラグクリア
    dmy = SCI3_2.RDR;                          //ダミーリード
}

/*****
SCI3_2 受信割り込み
*****/
void rxdata_sci3_2(void)

```

```

{
    put_rxbuf(SCI3_2.RDR);    //データをRxBufにストア
}

/*****
SCI3_2 1文字送信 (ポーリング)
-----
引数txdata    送信データ
*****/
void txone(unsigned char txdata)
{
    while(SCI3_2.SSR.BIT.TDRE == 0) {}    //送信可能まで待つ
    SCI3_2.TDR = txdata;
}

/*****
SCI3_2 1文字受信 (ポーリング)
-----
戻り値  受信データ
*****/
unsigned char rxone(void)
{
    while(SCI3_2.SSR.BIT.RDRF == 0) {}    //受信するまで待つ
    return SCI3_2.RDR;
}

```

このプログラムは割込みを使っているので、「intprg. c」を追加・修正します。

```

/*****
/*
/* FILE      :intprg.c
/* DATE      :Thu, Dec 10, 2009
/* DESCRIPTION:Interrupt Program
/* CPU TYPE  :H8/3687
/*
/* This file is generated by Renesas Project Generator (Ver.4.16).
/*
*****/

#include <machine.h>

extern void intprog_tmb1(void);
extern void intprog_tmz0(void);
extern void intprog_sci3_2(void);

#pragma section IntPRG
// vector 1 Reserved

// vector 2 Reserved

// vector 3 Reserved

```

```

// vector 4 Reserved

// vector 5 Reserved

// vector 6 Reserved

// vector 7 NMI
__interrupt(vect=7) void INT_NMI(void) { /* sleep(); */}
// vector 8 TRAP #0
__interrupt(vect=8) void INT_TRAP0(void) { /* sleep(); */}
// vector 9 TRAP #1
__interrupt(vect=9) void INT_TRAP1(void) { /* sleep(); */}
// vector 10 TRAP #2
__interrupt(vect=10) void INT_TRAP2(void) { /* sleep(); */}
// vector 11 TRAP #3
__interrupt(vect=11) void INT_TRAP3(void) { /* sleep(); */}
// vector 12 Address break
__interrupt(vect=12) void INT_ABRK(void) { /* sleep(); */}
// vector 13 SLEEP
__interrupt(vect=13) void INT_SLEEP(void) { /* sleep(); */}
// vector 14 IRQ0
__interrupt(vect=14) void INT_IRQ0(void) { /* sleep(); */}
// vector 15 IRQ1
__interrupt(vect=15) void INT_IRQ1(void) { /* sleep(); */}
// vector 16 IRQ2
__interrupt(vect=16) void INT_IRQ2(void) { /* sleep(); */}
// vector 17 IRQ3
__interrupt(vect=17) void INT_IRQ3(void) { /* sleep(); */}
// vector 18 WKP
__interrupt(vect=18) void INT_WKP(void) { /* sleep(); */}
// vector 19 RTC
__interrupt(vect=19) void INT_RTC(void) { /* sleep(); */}
// vector 20 Reserved

// vector 21 Reserved

// vector 22 Timer V
__interrupt(vect=22) void INT_TimerV(void) { /* sleep(); */}
// vector 23 SCI3
__interrupt(vect=23) void INT_SCI3(void) { /* sleep(); */}
// vector 24 IIC2
__interrupt(vect=24) void INT_IIC2(void) { /* sleep(); */}
// vector 25 ADI
__interrupt(vect=25) void INT_ADI(void) { /* sleep(); */}
// vector 26 Timer Z0
__interrupt(vect=26) void INT_TimerZ0(void) {intprog_tmz0();}
// vector 27 Timer Z1
__interrupt(vect=27) void INT_TimerZ1(void) { /* sleep(); */}
// vector 28 Reserved

// vector 29 Timer B1
__interrupt(vect=29) void INT_TimerB1(void) {intprog_tmb1();}
// vector 30 Reserved

```



```
// vector 31 Reserved

// vector 32 SCI3_2
__interrupt(vect=32) void INT_SCI3_2(void) {intprog_sci3_2();}
```

参考までに PIC12F683 のソースリストも紹介します。興味のある方は解析してみてください。

```
-----
;
;
; FILE      :ir_rcv.asm
; DATE      :Wed, Nov 26, 2008
; DESCRIPTION :Main Program
; CPU TYPE  :PIC12F683
;
; This file is programed by TOYO-LINX Co.,Ltd. / yKikuchi
;
;-----

list      p=12f683
#include <p12f683.inc>

__CONFIG_FCMEN_ON & _IESO_ON & _BOD_ON & _CPD_OFF & _CP_OFF & _MCLRE_OFF & _PWRTE_ON & _WDT_OFF
& _INTRC_OSC_NOCLKOUT

RADIX    DEC

;*****
; Constant Diffinition
;*****
IR       equ     GP2      ;IR Signal Bit
TIMING   equ     GPO      ;Timig Check Signal

;*****
; Data Memory
;*****
UDATA    20h
ircmd    RES     1       ;IR Command
ircode   RES     1       ;IR Signal Code
ircnt    RES     1       ;IR Receive Counter

txdata   RES     1       ;Transimt Data
txcnt    RES     1       ;Transmit Counter
tx_tmp   RES     1       ;Test Program Data

wait_tmp RES     1       ;Wait Counter
wait_tmp2 RES     1      ;Wait Counter No.2

;*****
; Program Memory
;*****
ORG 0000h
GOTO    start
```

```

ORG 0004h

;*****
; Initialize
;*****
ORG 0010h
start
BSF    STATUS, RPO    ;Bank=1
MOVLW  01111000b    ;Internal OSC 8MHz
MOVWF  OSCCON
BCF    STATUS, RPO    ;Bank=0
MOVLW  000010b      ;GPIO Initial Out
MOVWF  GPIO
MOVLW  07h          ;Comparator Off(Use I/O Pin)
MOVWF  CMCONO
BSF    STATUS, RPO    ;Bank=1
CLRF   ANSEL        ;Analog Input Off(Use I/O Pin)
MOVLW  111100b      ;GP2, 3, 4, 5=Input / GP0, 1=Output
MOVWF  TRISIO
MOVLW  01111111b    ;GPIO Pull Up On
MOVWF  OPTION_REG
BCF    STATUS, RPO    ;Bank=0

;*****
; Main Program
;*****
main
CLRWDI

    BTFSC  GPIO, IR    ;Leader Code
    GOTO   main
    CALL   check_leader
    SUBLW  01h
    BTFSS  STATUS, Z
    GOTO   main

    CALL   getcode     ;Custom Code(1)
    MOVLW  000h
    SUBWF  ircode, W
    BTFSS  STATUS, Z
    GOTO   main

    CALL   getcode     ;Custom Code(2)
    MOVLW  0ffh
    SUBWF  ircode, W
    BTFSS  STATUS, Z
    GOTO   main

    CALL   getcode     ;Data
    MOVF   ircode, W
    MOVWF  ircmd

    CALL   getcode     ;Data(Compliment)
    COMF   ircode, W
    SUBWF  ircmd, W

```



```

BTFSS    GPIO, IR
GOTO     $-1

BSF      GPIO, TIMING
CALL     wait15bit
BCF      GPIO, TIMING

BTFSC    GPIO, IR
GOTO     getcode_03
BCF      STATUS, C
RRF      ircode, F

getcode_02
DECFSZ   ircnt, F ; 12
GOTO     getcode_01
RETURN

getcode_03
BSF      STATUS, C
RRF      ircode, F

BTFSC    GPIO, IR
GOTO     $-1
GOTO     getcode_02

;*****
; Wait 1.5bit(IR Signal)
;*****
wait15bit
MOVLW    209          ;IR Signal 1.5bit(0.84ms)
MOVWF    wait_tmp

wait15bit_01
CLRWDT
GOTO     $+1
GOTO     $+1
DECFSZ   wait_tmp, F
GOTO     wait15bit_01
RETURN

;*****
; Transmit 1-Charactor
;*****
TXD      equ      GP1      ;TXD Bit

txone
MOVLW    8
MOVWF    txcnt

BCF      GPIO, TXD ;Start Bit
CALL     wait1bit

txone_01
RRF      txdata, F ;Data Bit
BTFSC    STATUS, C
GOTO     txone_02
BCF      GPIO, TXD
GOTO     txone_03

```

```

txone_02
    BSF    GPIO, TXD
    NOP
txone_03
    CALL   wait1bit
    DECFSZ txcnt, F
    GOTO   txone_01

    BSF    GPIO, TXD ;Stop Bit (2bit)
    CALL   wait1bit
    CALL   wait1bit

    RETURN

;*****
; Wait 1-bit
;*****
wait1bit
    MOVLW  10          ;38400 baud
    MOVWF  wait_tmp
wait1bit_01
    CLRWDT
    DECFSZ wait_tmp, F
    GOTO   wait1bit_01
    RETURN

;*****
; Wait Timer
;*****
wait1ms
    MOVLW  249        ;1ms
    MOVWF  wait_tmp
wait1ms_01
    CLRWDT
    GOTO   $+1
    GOTO   $+1
    DECFSZ wait_tmp, F
    GOTO   wait1ms_01
    RETURN

wait100ms
    MOVLW  100        ;100ms
wait100ms_00
    MOVWF  wait_tmp2
wait100ms_01
    CALL   wait1ms
    DECFSZ wait_tmp2, F
    GOTO   wait100ms_01
    RETURN

wait200ms
    MOVLW  200        ;200ms
    GOTO   wait100ms_00

;*****

```

```

; Test Program 1
;*****
test01
    BSF    GPIO, 1
    BCF    GPIO, 1
    GOTO   test01

;*****
; Test Program 2
;*****
test02
    MOVLW  020h
    MOVWF  tx_tmp
test02_01
    MOVF   tx_tmp, W
    MOVWF  txdata
    call   txone
    INCF   tx_tmp, F
    BTFSS  tx_tmp, 6
    GOTO   test02_01
    MOVLW  0dh
    MOVWF  txdata
    CALL   txone
test02_02
    MOVF   tx_tmp, W
    MOVWF  txdata
    call   txone
    INCF   tx_tmp, F
    BTFSS  tx_tmp, 7
    GOTO   test02_02
    MOVLW  0dh
    MOVWF  txdata
    CALL   txone
    CALL   wait200ms
    CALL   wait200ms
    CALL   wait100ms
    GOTO   test02

;*****
    END

```


付録

ハイパーH8

■ ハイパーH8 って何？

ハイパーH8 は Windows シリーズに標準で搭載されているターミナルソフト、‘ハイパーターミナル’を使用した簡易モニタです。お手持ちのパソコンと TK-3687 のシリアルポートを RS-232C ケーブルで接続することで、簡単なモニタ環境を作ることができます。

ところでモニタとは何でしょうか。モニタ (monitor) には監視する、という意味があります。マイコンでいうモニタというプログラムは、マイコンの中身を監視するプログラムです。レジスタの値はどうなっているでしょうか。ROM や RAM にどんなデータが入っているでしょうか。I/O にどんなデータが入出力されているでしょうか。モニタが搭載されていれば、このようなマイコンの中身の情報を見ることができます。また、パソコンで作ったプログラムをマイコンに送り込む (ロード) こともできます。さらに、プログラムの動作そのものも制御することができ、ロードしたプログラムを実行したり、途中で止めることもできます。

“Pirkus・R Type-02” 付属の TK-3687mini は、出荷時にあらかじめハイパーH8 が書き込まれているので、電源オンですぐにマイコンの中身を見ることができます。なお、本マニュアルの 4 章以降はフラッシュメモリに書き込むため、ハイパーH8 は消去されます。再びハイパーH8 を使いたいときは、付録の「FDT によるプログラムの書き込み手順」を見てハイパーH8 を書き込んでください。

```
38400bps - ハイパーターミナル
ファイル(F) 編集(E) 表示(V) 通信(C) 転送(T) ヘルプ(H)

Hyper Monitor Program.
for H8/3687F -ver,040809-
Copyright (C)2003-2004 by TOYO-LINX,Co.,LTD.

< [?] = Command Help >

H8>L   Waiting for HEX File ...
*****
File Name   [led.mot]
Load Address [00E800-00EA00]
Finish!

H8>DEA00

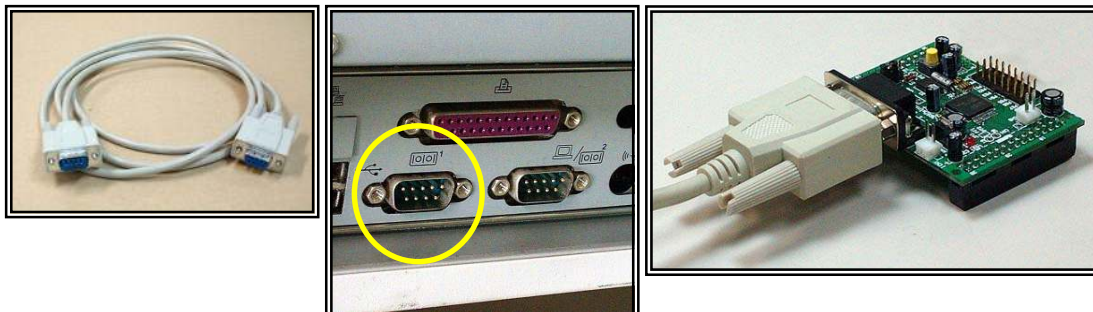
      +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F  -- ASCII CODE --
EA00 F8 01 38 E9 7F D9 72 00 7F D9 70 00 40 F6 00 00  ,,8,.,r,.,p,@,,,
EA10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ,,,,,,,,,,,,,,
EA20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ,,,,,,,,,,,,,,
EA30 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ,,,,,,,,,,,,,,
EA40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ,,,,,,,,,,,,,,
EA50 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ,,,,,,,,,,,,,,
EA60 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ,,,,,,,,,,,,,,
EA70 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ,,,,,,,,,,,,,,

H8>_

接続 00:05:25  ANSI  38400 8-N-1  SCROLL  CAPS  NUM  キャプチャ  ローを印刷します。
```

■ TK-3687mini とパソコンをつなぐ

まずバッテリーをはずした TK-3687mini とパソコンをつなぎます。D-Sub・9pin (オス) – 9pin (メス) ストレートケーブル (写真左) でメス側をパソコンの COM ポート (写真中) へ、オス側を TK-3687mini の CN5 (写真右) へ接続します。しっかりとさし込み、ケーブルにネジがついている場合はネジをしめて固定しましょう。写真のように COM ポートがいくつか空いている場合は1番につなげてください。なお、COM ポートが用意されていないパソコンでは USB-RS232C 変換ケーブルでつなげて下さい。



■ ハイパーターミナルの設定

それでは、通信ソフト‘ハイパーターミナル’を起動して、TK-3687mini と通信するためのセッティングを行きましょう。

まずハイパーターミナルを起動します。ハイパーターミナルは、



から起動できます。Windows のバージョンによっては、



から起動する場合があります。もし、スタートメニューにない場合は、



で‘hyperterm. exe’を検索してください。ハイパーターミナルを起動したら、出てくるダイアログウィンドウにしたがって設定していきましょう。

① 接続の設定 (1)

名前とアイコンを設定します。右の画面では、名前は接続速度がわかるように「38400bps」としました。名前を入力してアイコンを選択したら をクリックします。



② 接続の設定(2)

接続方法(N) : のプルダウンメニューから、ケーブルを接続した COM ポート(右の画面では COM1)を選択して **OK** をクリックします。



③ COM1 のプロパティ

各項目を次のように設定します。

ビット/秒(B) : 38400

データビット(D) : 8

パリティ(P) : なし

ストップビット(S) : 1

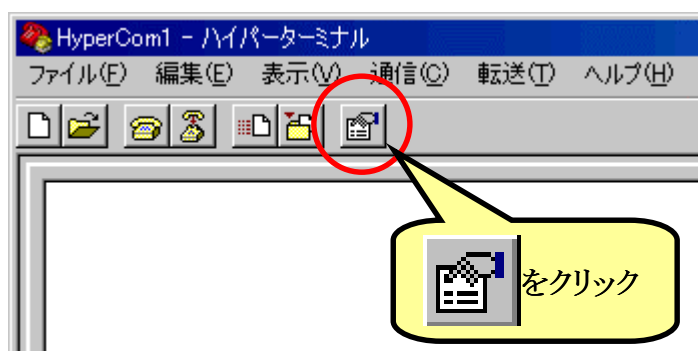
フロー制御(F) : Xon/Xoff

設定し終わったら **OK** をクリックします。



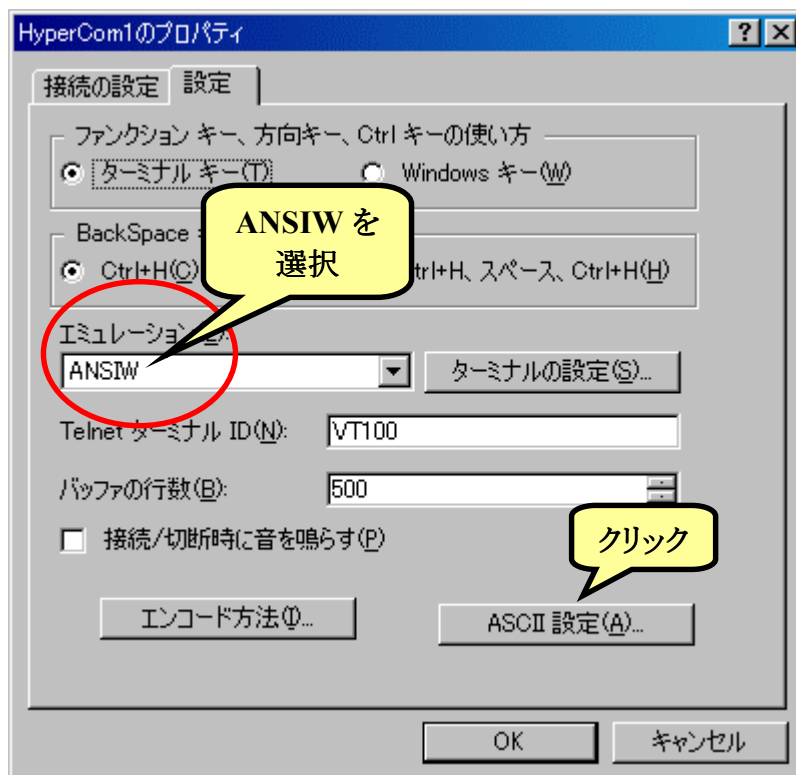
④ プロパティアイコンをクリック

ターミナル画面に切り替わりますので、ツールバーのプロパティアイコンをクリックしてプロパティダイアログを開きます。



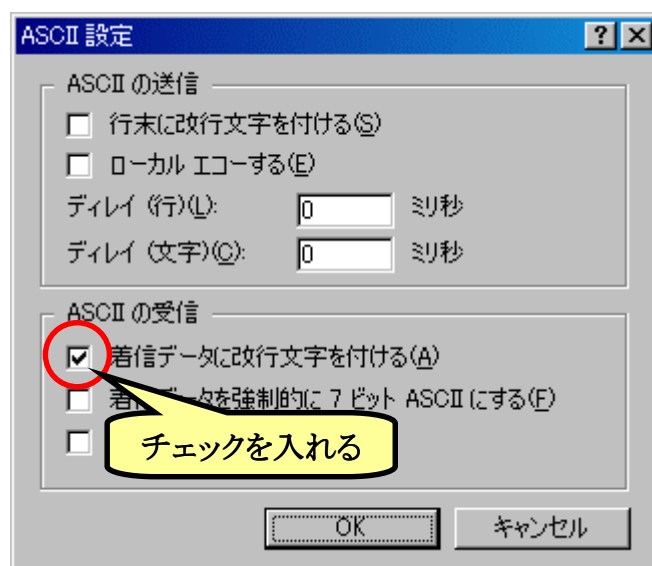
⑤ プロパティ

‘設定’タブをクリックして‘エミュレーション(E):’のプルダウンメニューから‘ANSIW’を選択し、**ASCII設定(A)...**をクリックします。



⑥ ASCII設定

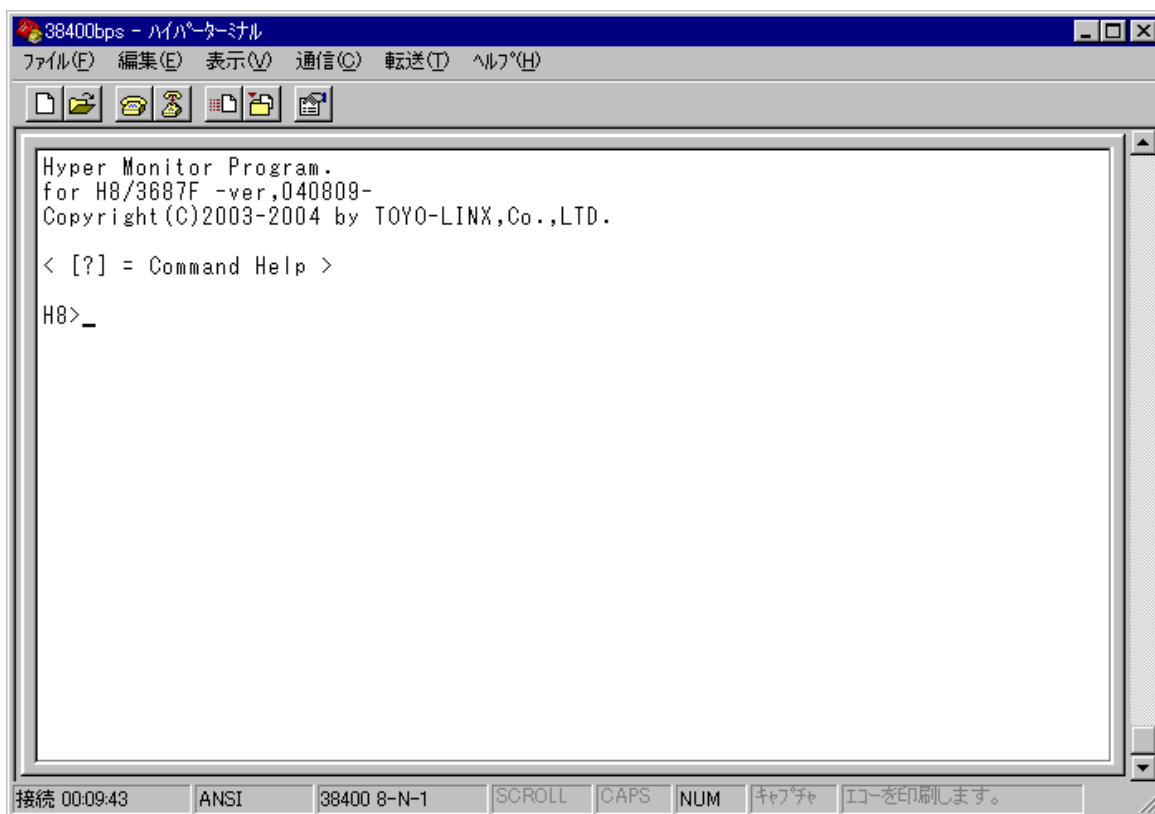
‘ASCIIの受信’の中の‘着信データに改行文字を付ける(A)’のチェックを入れて**OK**をクリックします。するとプロパティダイアログに戻りますので、もう一度**OK**をクリックしてターミナル画面に戻ります。



◆
これで設定は終了です。それでは電源をオンしてみましょ。ちゃんと動くでしょうか。

■ 電源オン!!

“Pirkus・R Type-02”に専用 AC アダプタ, もしくはバッテリーをつないでください。マイコン側の電源スイッチをオンにするとハイパーターミナルの画面に次のように表示されます。



The screenshot shows a Hyper Terminal window titled "38400bps - ハイパーターミナル". The window contains the following text:

```
Hyper Monitor Program.  
for H8/3687F -ver,040809-  
Copyright (C)2003-2004 by TOYO-LINX,Co.,LTD.  
  
< [?] = Command Help >  
  
H8>_
```

The status bar at the bottom of the window displays: 接続 00:09:43 ANSI 38400 8-N-1 SCROLL CAPS NUM キャッチャ エコーを印刷します。

ここまでくればマイコンの中身を自由に見ることができます。次は手始めにあらかじめ TK-3687mini に書き込まれているデモプログラムを実行してみましょう。

でも、その前に…(次のページを見てください)

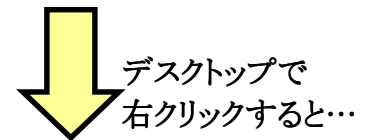
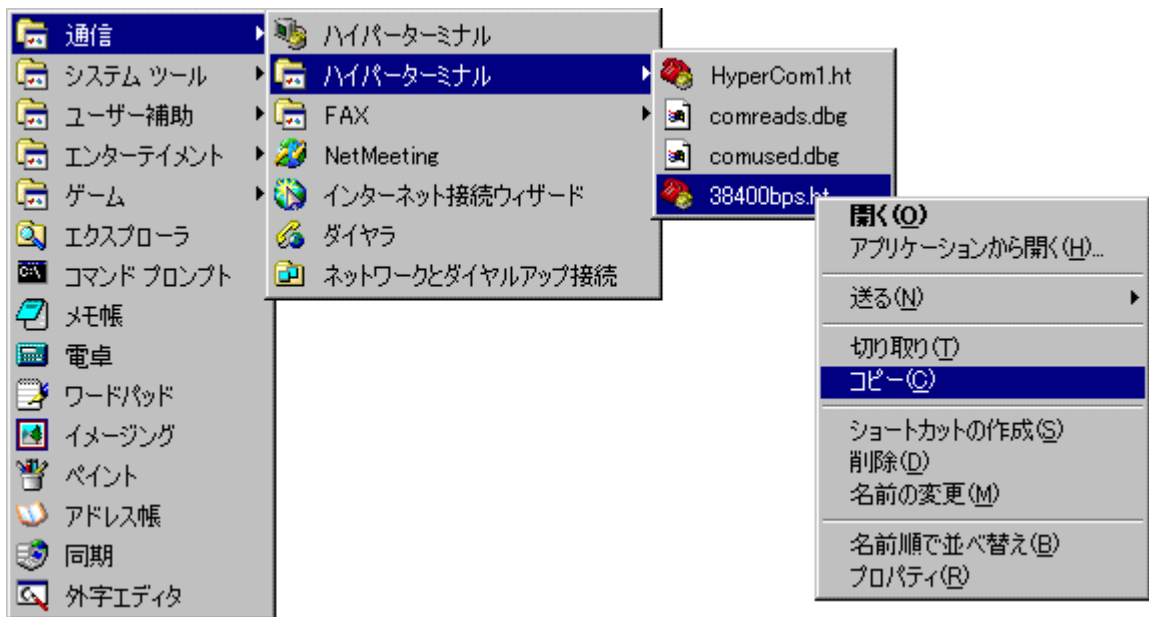
ハイパーターミナルを起動するたびに毎回設定を繰り返していたのでは面倒ですね。そこで、ハイパーターミナルの設定を保存しておきましょう。メニューバーの「ファイル(F)」→「上書き保存(S)」を選択して保存して下さい。



さらに、この設定のハイパーターミナルをすぐ呼び出せるように、デスクトップにショートカットを作成しましょう。スタートメニューから、

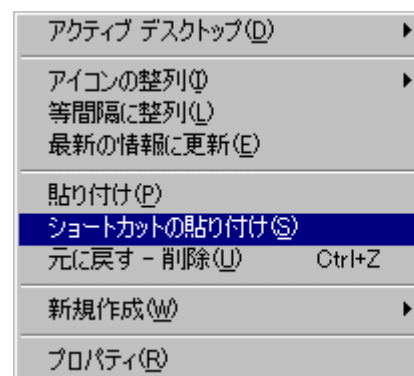


までカーソルを進め、右クリックします。プルダウンメニューの中の「コピー(C)」を選択してください。



デスクトップで再度右クリックし、「ショートカットの貼り付け(S)」を選択してショートカットを作成します。

なお、ここで示した方法は Windows2000 の場合です。この方法でショートカットが作成できない場合は、エクスプローラやファイルの検索を使ってデスクトップにショートカットを作ってください。



■ ハイパーH8 でプログラムを実行する

TK-3687mini にはデモ用に、また、基板チェックのために、いくつかのプログラムが ROM にあらかじめ書き込まれています。そのうちの一つを動かしてみましょう。ハイパーターミナルから 'G7200' と入力して 'Enter' キーを押します。すると、あっけないほど簡単にプログラムが動き出します。



```
38400bps - ハイパーターミナル
ファイル(F) 編集(E) 表示(V) 通信(C) 転送(T) ヘルプ(H)

Hyper Monitor Program.
for H8/3687F -ver,050120-
Copyright (C)2003-2005 by TOYO-LINX,Co.,LTD.

< [?] = Command Help >

H8>G7200
      Run Address [007200]
      Running...
```

このプログラムは、H8/3687 に内蔵されている時計機能 (RTC) を使って、23 時 59 分 30 秒から時計をスタートし、ハイパーモニタの画面に時間を表示します。



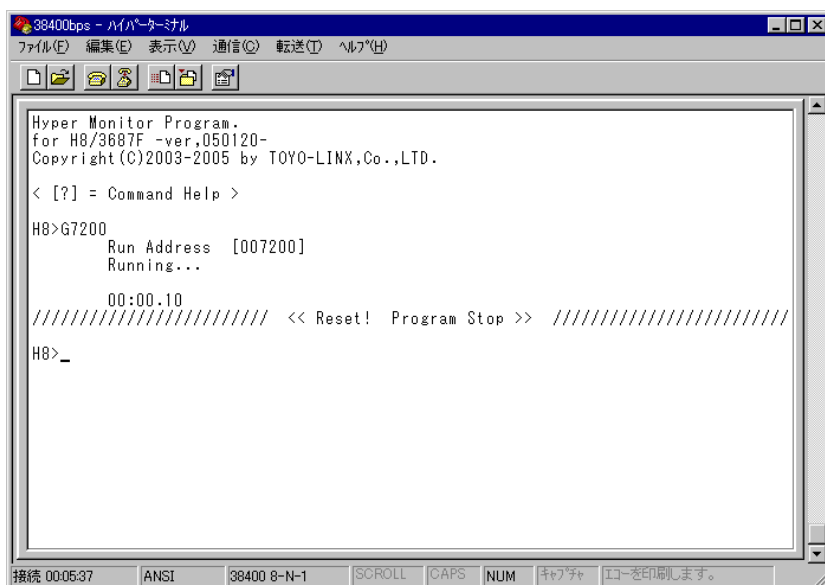
```
Hyper Monitor Program.
for H8/3687F -ver,050120-
Copyright (C)2003-2005 by TOYO-LINX,Co.,LTD.

< [?] = Command Help >

H8>G7200
      Run Address [007200]
      Running...
      23:59.35_
```

時間の表示

TK-3687mini のリセットスイッチ (SW1) を押すと、実行中のプログラムは停止して、ハイパーH8 は右図のように入力待ちの状態になります。



```
Hyper Monitor Program.
for H8/3687F -ver,050120-
Copyright (C)2003-2005 by TOYO-LINX,Co.,LTD.

< [?] = Command Help >

H8>G7200
      Run Address [007200]
      Running...

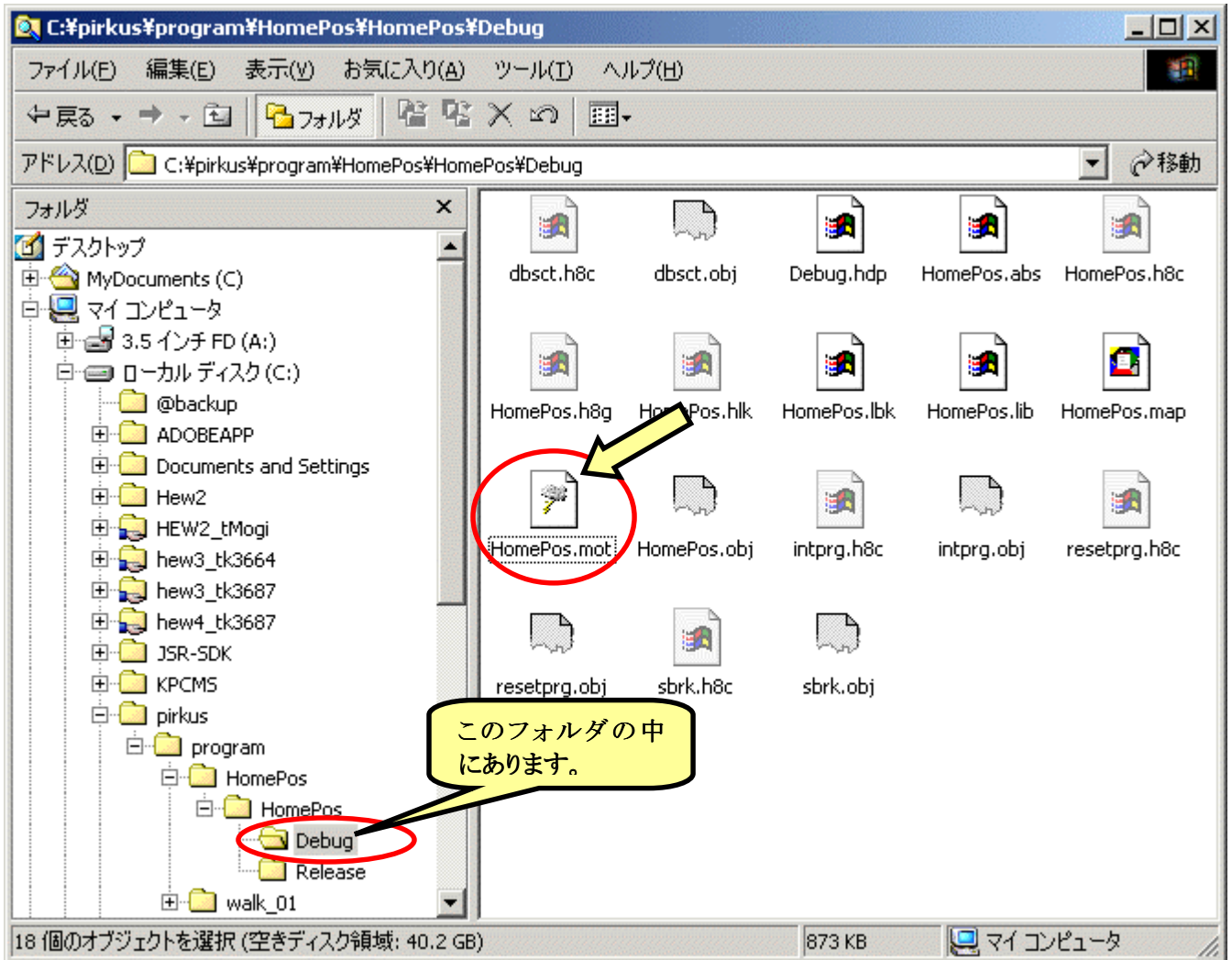
      00:00.10
      ////////////////////////////////// << Reset! Program Stop >> //////////////////////////////////
H8>_
```

接続 00:05:37 ANSI 38400 8-N-1 SCROLL CAPS NUM キーを押す エコーを印刷します。

■ ハイパーH8 でファイルをダウンロードし実行する

コンパイルすると拡張子が‘. mot’というファイルが作成されます。このファイルは「S タイプファイル」と呼ばれており、マシン語の情報が含まれています。ハイパーH8 は S タイプファイルをダウンロードすることができます。

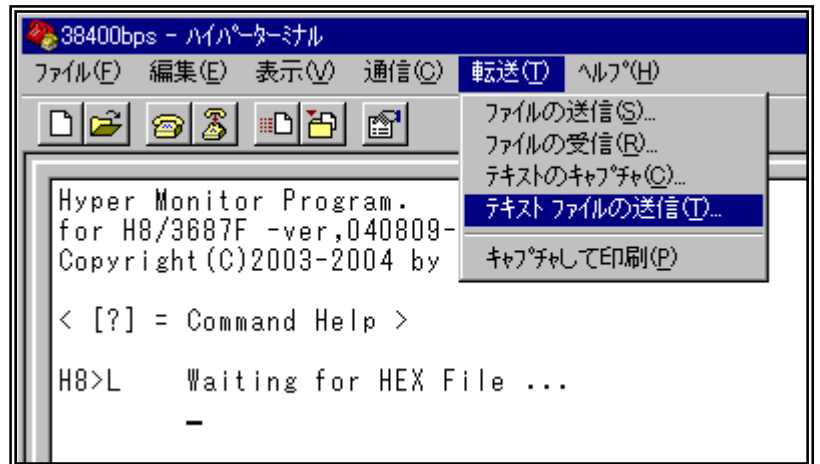
ここでは例として、マニュアルの「3 ホームポジションを作る」で使った‘HomePos. mot’をダウンロードし実行してみましょう。‘HomePos. mot’は次のフォルダ内に作られます。



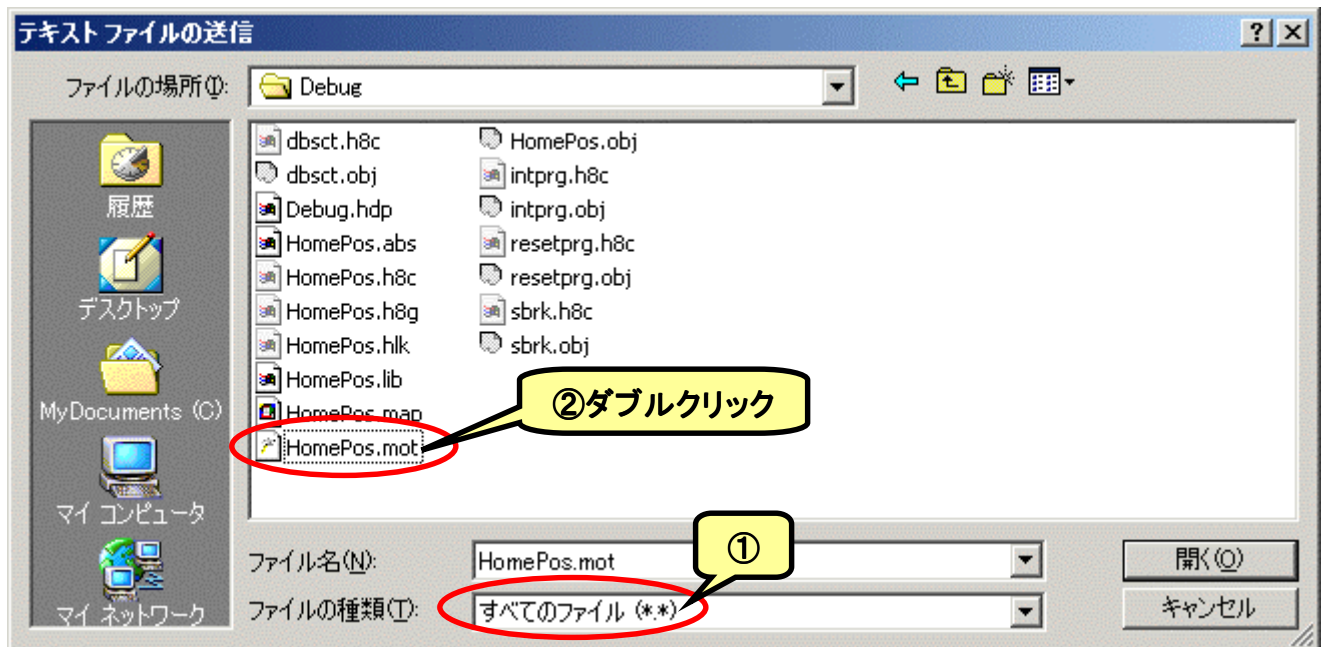
それでは、ハイパーH8 を起動して下さい。‘L’コマンドを使います。パソコンのキーボードから‘L’と入力して‘Enter’キーを押します。



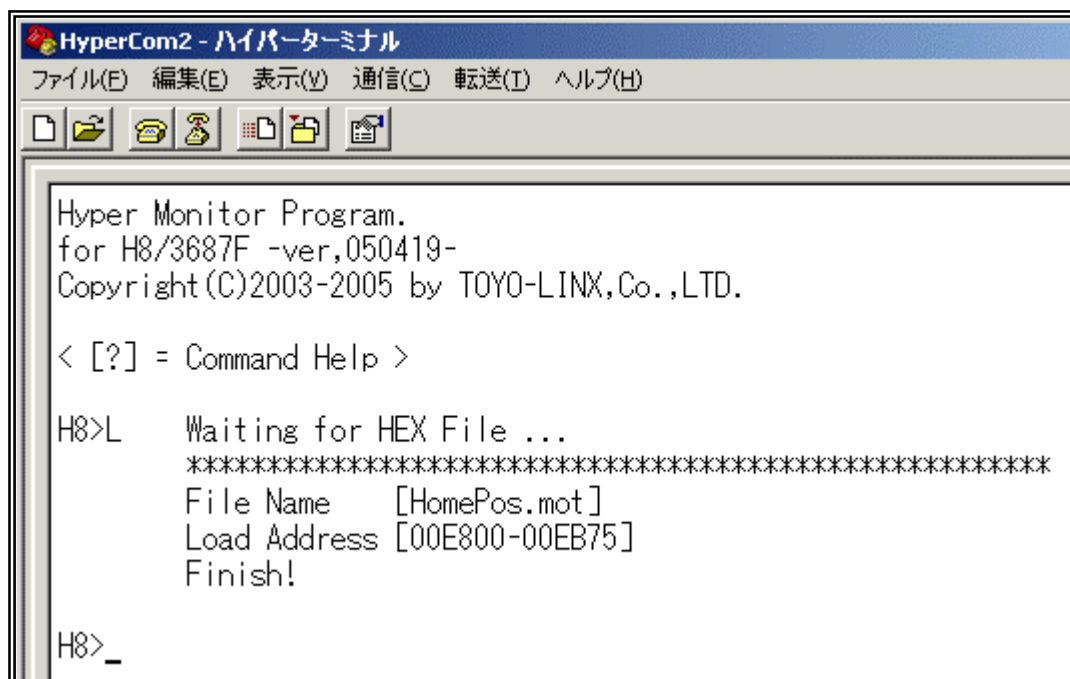
メニューから「テキストファイルの送信(T) ...」を選択します。



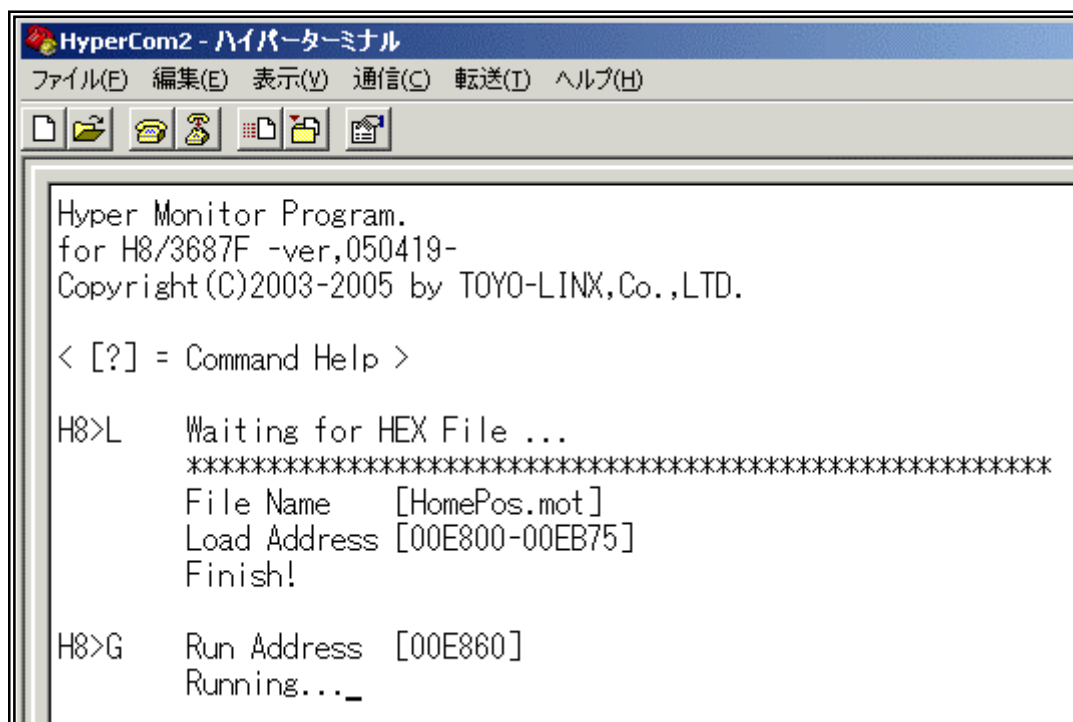
‘テキストファイルの送信’ウィンドウが開きます。①ファイルの種類を‘すべてのファイル’にして下さい。②‘HomePos. mot’をダブルクリックします。



ダウンロードが始まります。終了すると次のように表示されます。



では、ダウンロードしたプログラムを実行してみましょう。プログラムカウンタはダウンロードすると自動的に設定されますので、‘G’ ‘Enter’で実行できます。



どうでしょう。ちゃんと動きましたか？

ここでは‘L’、‘G’コマンドをそれぞれ入力しましたが、‘LG’とコマンドを連結して入力することもできます。このようにすると、プログラムをダウンロード後、直ちに実行することができます。

ハイパーH8 のコマンドを調べるには…

‘L’、‘G’コマンドを使いましたが、そのほかにもハイパーH8 には便利なコマンドがたくさん用意されています。詳しくはハイパーH8 のマニュアルを見ていただくとして、思い出しやすいようにコマンドヘルプがハイパーH8 には組み込まれています。キーボードから‘?’を入力して下さい。次の画面が表示されます。

```
Hyper Monitor Program.
for H8/3687F -ver.040809-
Copyright (C)2003-2004 by TOYO-LINX,Co.,LTD.

< [?] = Command Help >

H8?
**** Command Help ****
L = Load HEX File      Select HEX(.mot)File by Menu Bar.
G = Program Run        [G],[Gxxxx],[Gxxxx,xxxx],[G,xxxx]
T = Program Trace      [T],[TR],[Tn],[TRn]
S = Skip Trace          [S],[SR],[Sn],[SRn] (n=1-999d)
                        'Enter'=(Skip)Trace : 'R'=Register : '/'=Escape
D = Dump Data          [D],[Dxxxx],[DP],[D-]
W = Write Data         [W],[Wxxxx],[Wxxxx,dd]
R = Register View      [R],[RPC],[RSP],[RCCR],[Rr]
P = Module Register    [P],[Pr]
? = Command Help
Z = Memory Map Information
/ = Cancel              ( x:Address / r:Register / d:Data )
*****

H8>_

接続 01:41:44  ANSI  38400 8-N-1  SCROLL  CAPS  NUM  キャプチャ  エコーを印刷します。
```

ハイパーH8 は便利な道具なんですが…

ハイパーH8 は便利な道具ですが、多少の制限もあります。もっとも大きな制限は「ROM にデータを書き込むことができない」ということです。

この制限のため、ハイパーH8 でプログラムを入力する時は、RAM に入力しなければなりません。また、HEW を使ってアセンブルする時も、RAM 上にプログラムができるように Section を設定しなければなりません。

さらに、ROM に比べて RAM のサイズが小さいため、あまり大きなプログラムを実行することができない、という問題もおきます。

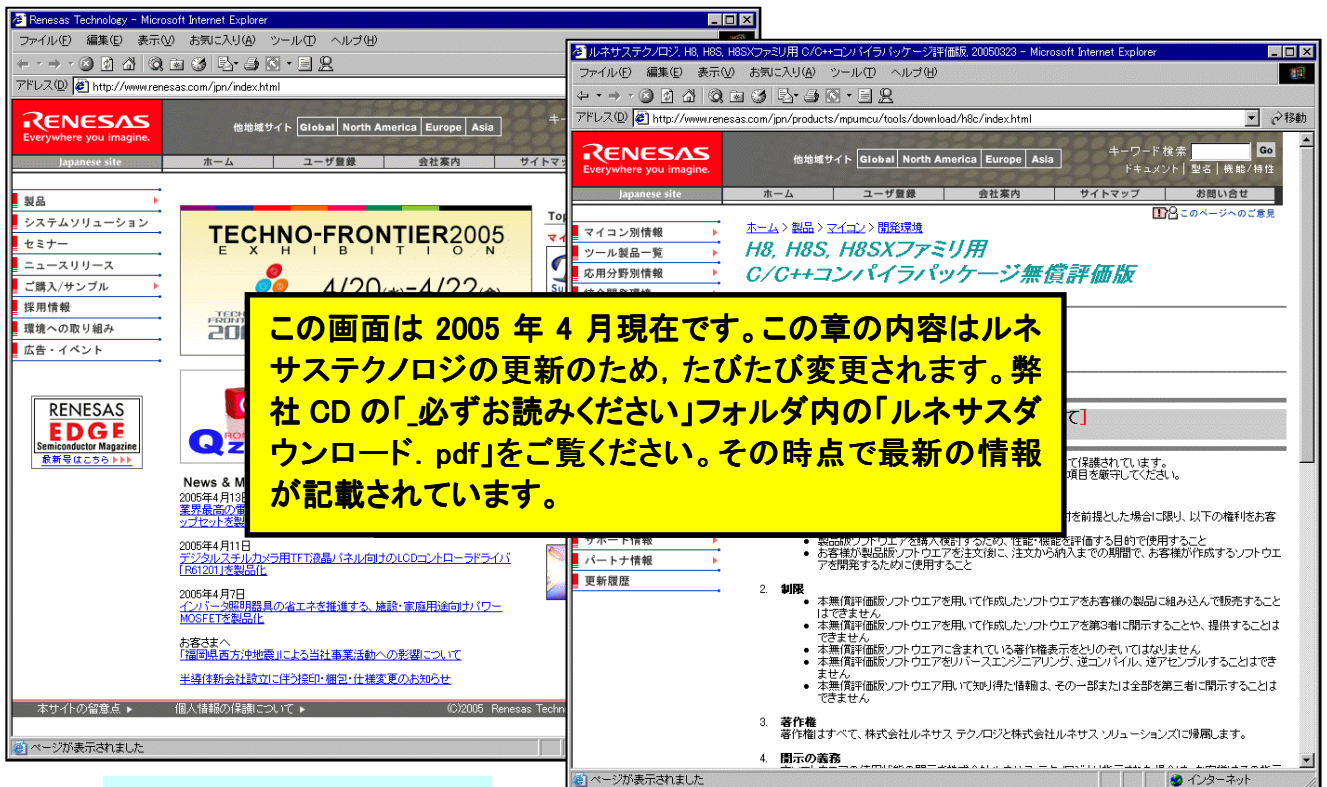
しかし、学習用と割り切って使う分には全く気にする必要はありません。なお、ROM にプログラムを書き込む場合は、FDT を使うことになります。また、デバッグまで行なう場合は‘E8’というエミュレータを購入して使うことになります。

HEW の使い方

ルネサステクノロジは現在、High-performancr Embedded Wprkshop V. 4(HEW4)に対応した無償評価版コンパイラを公開しています。無償評価版コンパイラは、はじめてコンパイルした日から 60 日間は製品版と同等の機能と性能のままで試用できます。61 日目以降はリンクサイズが 64K バイトまでに制限されますが、H8/3687 はもともとアクセスできるメモリサイズが 64K までバイトなので、この制限は関係ありません。また、無償評価版コンパイラは製品開発では使用できないのですが、H8/300H Tiny シリーズ(H8/3687 も含まれる)では許可されています。この項では無償評価版コンパイラのダウンロードからインストール、プログラムの入力とビルドまでを説明します。

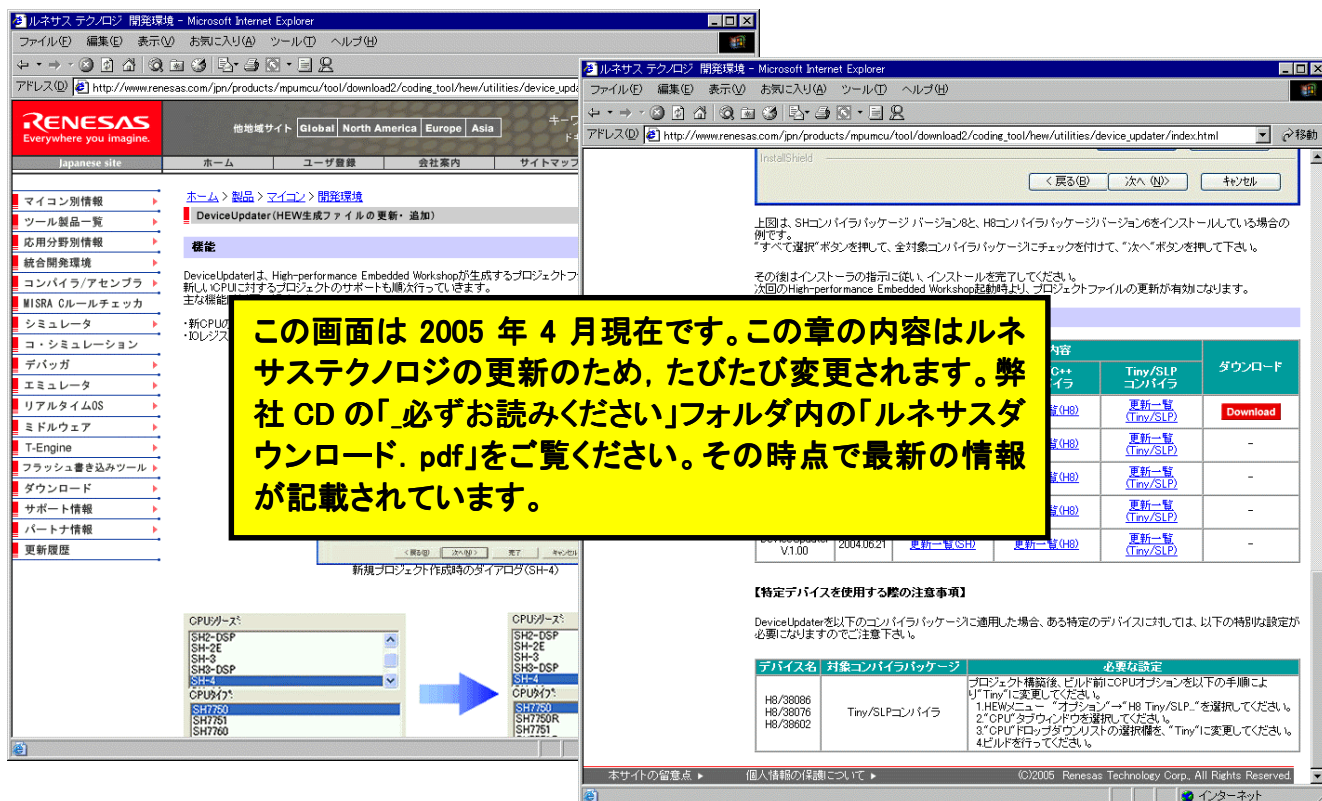
■ HEW の入手

HEW は株式会社ルネサステクノロジのホームページよりダウンロードします。ダウンロードサイトの URL は以下の通りです。



ダウンロードサイトの下の方にある「ダウンロードのページへ」をクリックして下さい。次のページで必須事項を入力してダウンロードを開始します。ダウンロード先はデスクトップにすると便利です。全部で 69.4MByte になりますので、ADSL か光回線でない、かなり大変なのが実情です。‘h8cv601r00.exe’ というファイルがダウンロードされます。

ところで、ここでダウンロードした無償評価版コンパイラには不具合があることが報告されています。それで、ルネサステクノロジが公開しているデバイスアップデートを使用して不具合を修正します。デバイスアップデートは下記の URL のサイトからダウンロードできます。



デバイスアップデート
ダウンロードサイト
http://www.renesas.com/jpn/products/mpumcu/tool/download2/coding_tool/hew/utilities/device_updata/index.html

ページの下の方にある「Download」をクリックしてください。ダウンロード先はデスクトップにすると便利です。全部で 3.55MByte になります。‘hew_du104.exe’ というファイルがダウンロードされます。

最新版の HEW を手に入れましょう

HEW は頻りにバージョンアップされます。HEW はルネサステクノロジのマイコン全てに対応しているため、H8 シリーズはもとより、R8 シリーズや SH シリーズなど、対応するマイコンが増えたとそのたびにマイナーチェンジされるようです。また、その際に報告されていた不具合を一緒に修正することもあります。そのため、このマニュアルの情報もすぐに古くなってしまい改訂が間に合わないのが実情です。

それで、ルネサステクノロジのホームページは定期的のぞいてみることをおすすめします。特にデバイスアップデートの情報は要注意です。

■ HEW のインストール

ダウンロードした ‘h8cv601r00.exe’ をダブルクリックしてください。すると、インストールが始まります。画面の指示に従ってインストールしてください。

次に、無償評価版コンパイラをアップデートします。ダウンロードした ‘hew_du104.exe’ をダブルクリックしてください。インストールが始まります。画面の指示に従ってインストールしてください。

■ メモリマップの確認

HEW を使うときのコツの一つは、メモリマップを意識する、ということです。プログラムがどのアドレスに作られて、データはどのアドレスに配置されるか、ちょっと意識するだけで、HEW を理解しやすくなります。

HEW がデフォルトで設定するメモリーマップは下記のとおりです。マップ中の“ユーザプログラムエリア”、“ユーザ RAM エリア”の範囲がユーザが自由に使用できるエリアで、H8/3687 の全てのメモリエリアを自由に使うことができます。ハイパーH8 を使わず、アプリケーションプログラムのみを ROM に書き込むときはこの設定にします。

0000 番地	割り込みベクタ			ROM/フラッシュメモリ (56K バイト)		
0400 番地	PResetPRG	リセットプログラム	ユーザプログラム エリア			
	PIntPRG	割り込みプログラム				
0800 番地	P	プログラム領域				
	C	定数領域				
	C\$DSEC	初期化データセクションのアドレス領域				
	C\$BSEC	未初期化データセクションのアドレス領域				
	D	初期化データ領域				
DFFF 番地						
E000 番地	未使用			未使用		
E7FF 番地						
E800 番地	B	未初期化データ領域 初期化データ領域 (変数領域)	ユーザ RAM エリア	RAM (2K バイト)		
	R					
EEFF 番地	S	スタック領域				
EF00 番地						
EFFF 番地						
F000 番地	未使用			未使用		
F6FF 番地						
F700 番地	I/O レジスタ			I/O レジスタ		
F77F 番地						
F780 番地	フラッシュメモリ書換用ワークエリア (使用禁止)			RAM (1K バイト)		
FB7F 番地						
FB80 番地			ユーザ RAM エリア	RAM (1K バイト)		
FF7F 番地						
FF80 番地	I/O レジスタ			I/O レジスタ		
FFFF 番地						

ハイパーH8 を使うときのメモリマップは次のとおりです。ROM はハイパーH8 が使用し、アプリケーションプログラムは RAM にロケーションします。

0000 番地 DFFF 番地	モニタプログラム 'ハイパーH8'		ROM/フラッシュメモリ (56K バイト)
E000 番地 E7FF 番地	未使用		未使用
E800 番地	ハイパーH8 ユーザ割り込みベクタ		RAM (2K バイト)
E860 番地	PResetPRG	リセットプログラム	
	PIntPRG	割り込みプログラム	
EA00 番地	P	プログラム領域	
	C	定数領域	
	C\$DSEC	初期化データセクションのアドレス領域	
	C\$BSEC	未初期化データセクションのアドレス領域	
	D	初期化データ領域	
FFFF 番地	ユーザ RAM エリア		
F000 番地 F6FF 番地	未使用		未使用
F700 番地 F77F 番地	I/O レジスタ		I/O レジスタ
F780 番地	B	未初期化データ領域 初期化データ領域 (変数領域)	RAM (1K バイト) フラッシュメモリ書換え用 ワークエリアのため、 FDT と E8 使用時は、 ユーザ使用不可
FB7F 番地	R		
FB80 番地			
FD80 番地	S	スタック領域	RAM (1K バイト)
FDFF 番地			
FE00 番地 FF7F 番地	ハイパーH8 ワークエリア		
FF80 番地 FFFF 番地	I/O レジスタ		I/O レジスタ

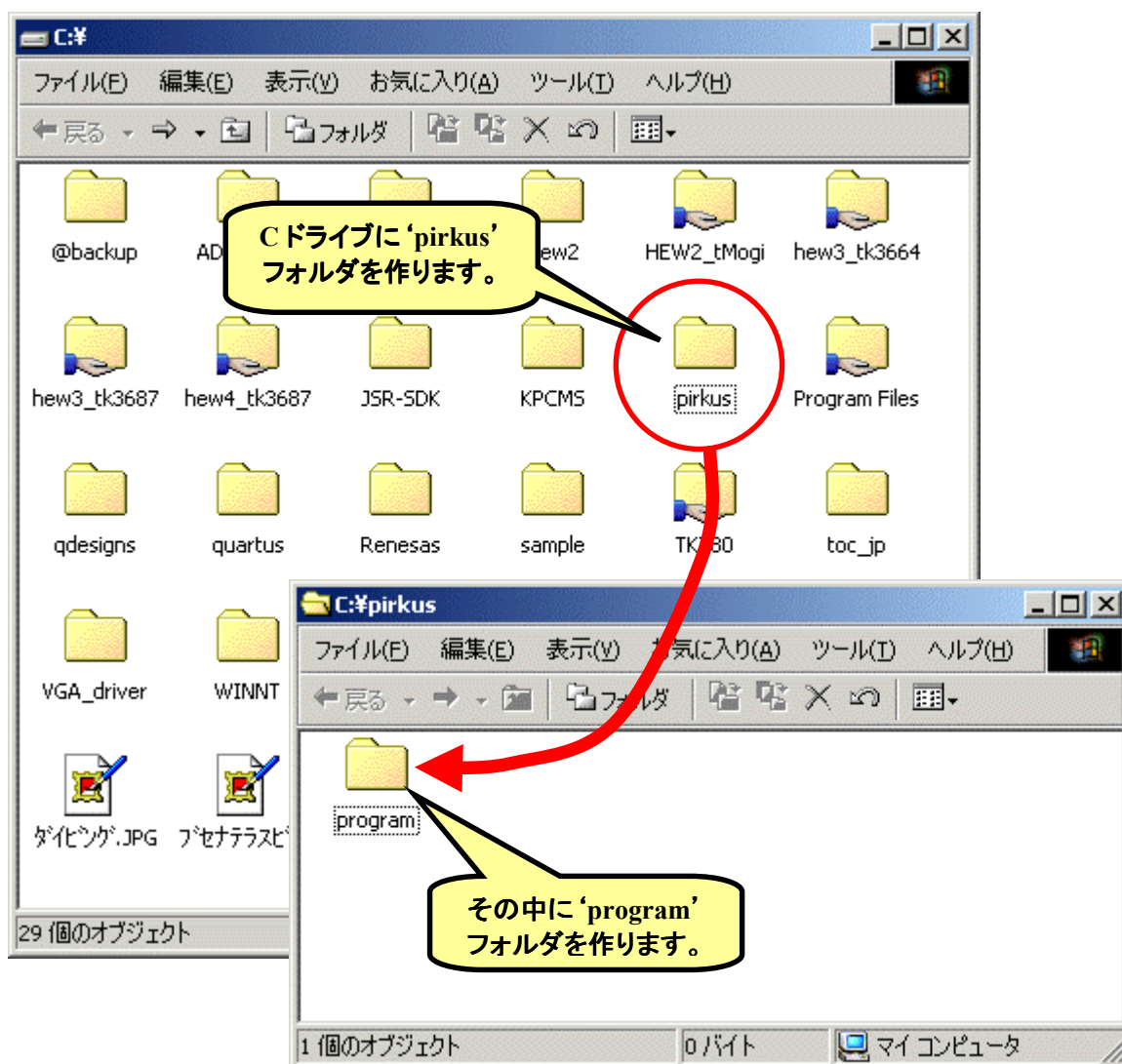
メモリマップのうちユーザ RAM エリアの部分だけが自由に使用できるエリアです。

■ プロジェクトの作成

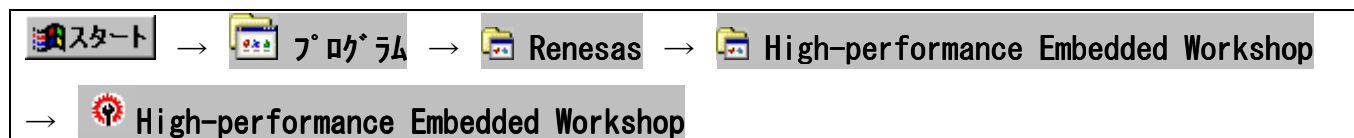
ここでは、本マニュアルの「4 二足歩行にチャレンジしよう」で作った‘walk_01.c’を例にします。

HEW ではプログラム作成作業をプロジェクトと呼び、そのプロジェクトに関連するファイルは1つのワークスペース内にまとめて管理されます。通常はワークスペース、プロジェクト、メインプログラムには共通の名前がつけられます。今回のプロジェクトは‘walk_01’と名付けます。以下に、新規プロジェクト‘walk_01’を作成する手順と動作確認の手順を説明します。

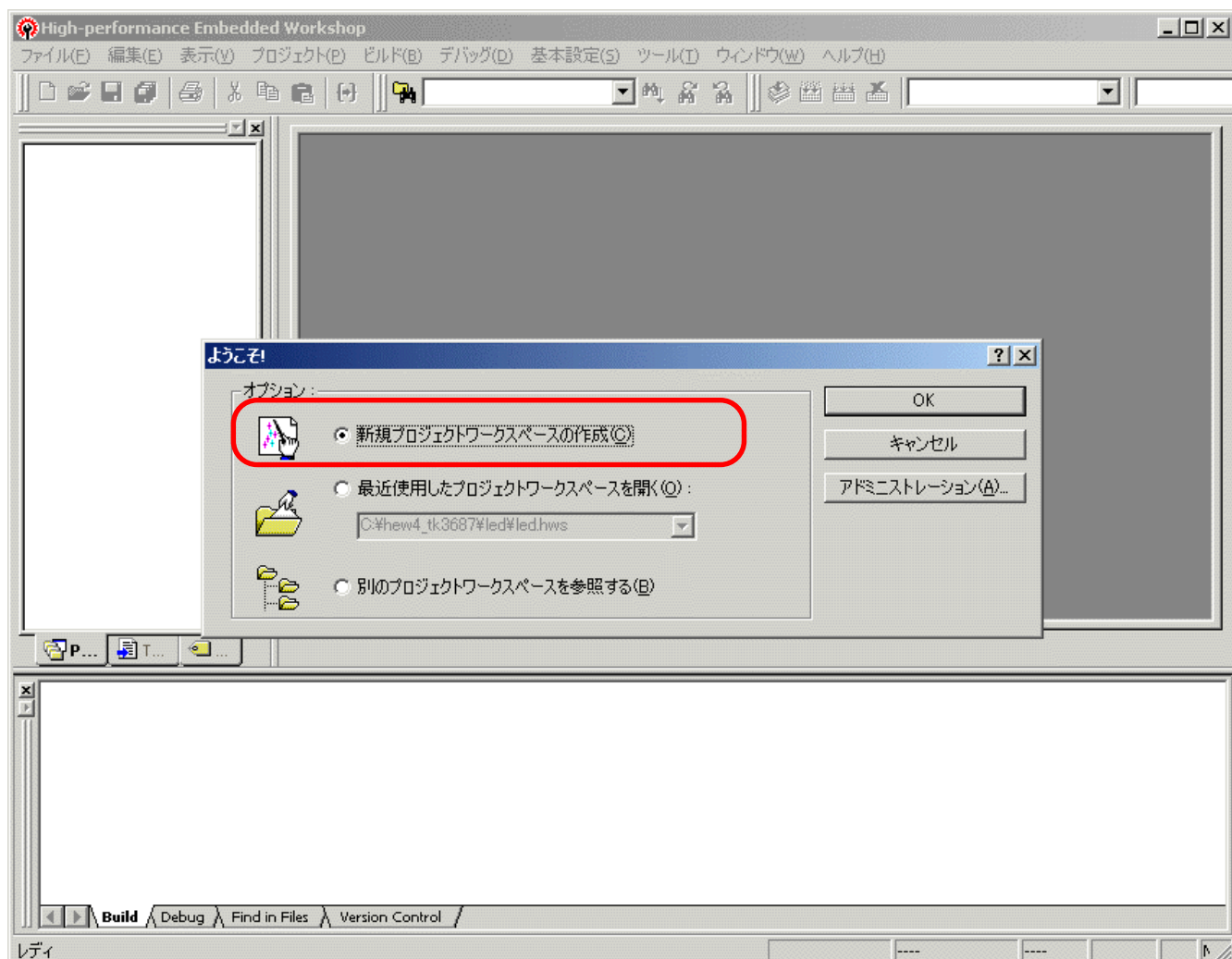
しかしその前に、Pirkus 専用作業フォルダを作っておきましょう。C ドライブに‘¥pirkus¥program¥’フォルダを作ってください。このマニュアルのプロジェクトは全てこのフォルダに作成します。



では、HEW を起動しましょう。スタートメニューから起動します。



HEW を起動すると下記の画面が現れるので、「新規プロジェクトワークスペースの作成」を選択して‘OK’をクリックします。



前に作ったプロジェクトを使うとき

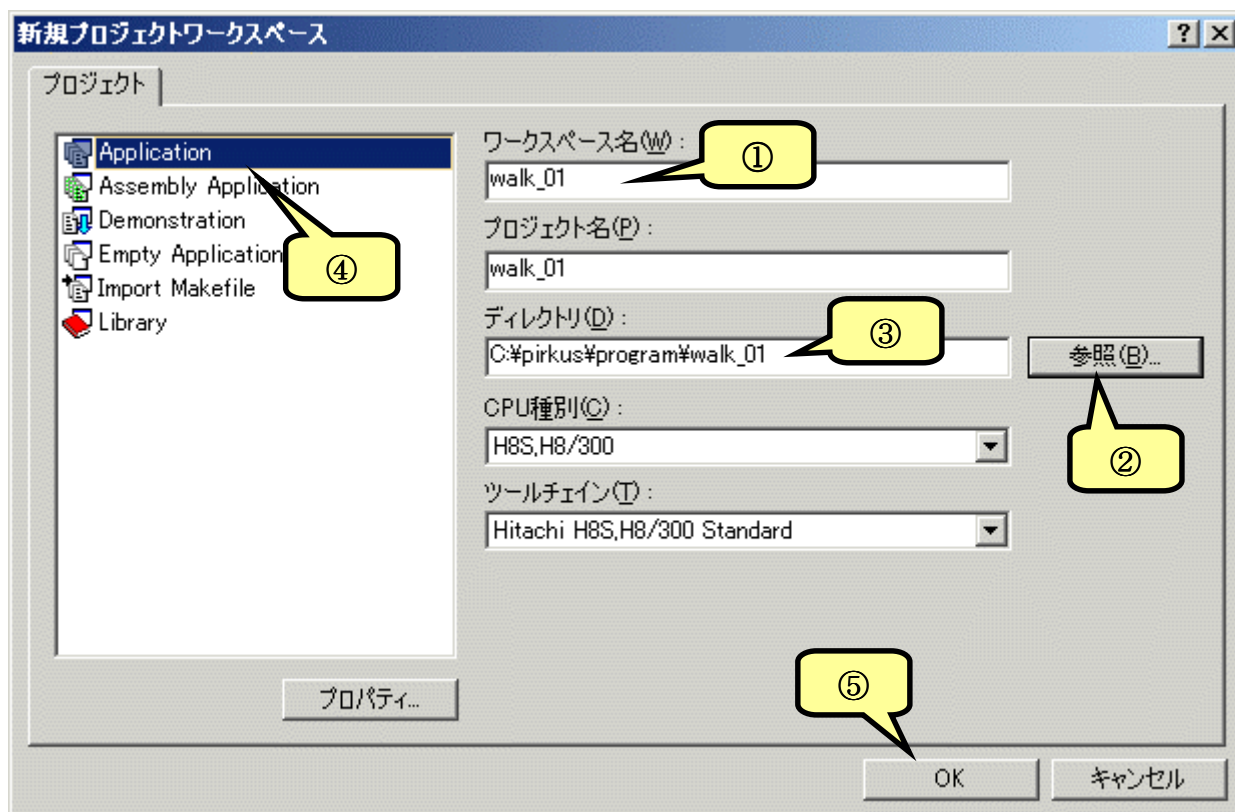
その場合は、「ようこそ!」ダイアログで「最近使用したプロジェクトワークスペースを開く」を選択して‘OK’をクリックします。そのプロジェクトの最後に保存した状態で HEW が起動します。

まず、①「ワークスペース名 (W)」(ここでは 'walk_01') を入力します。「プロジェクト名 (P)」は自動的に同じ名前になります。

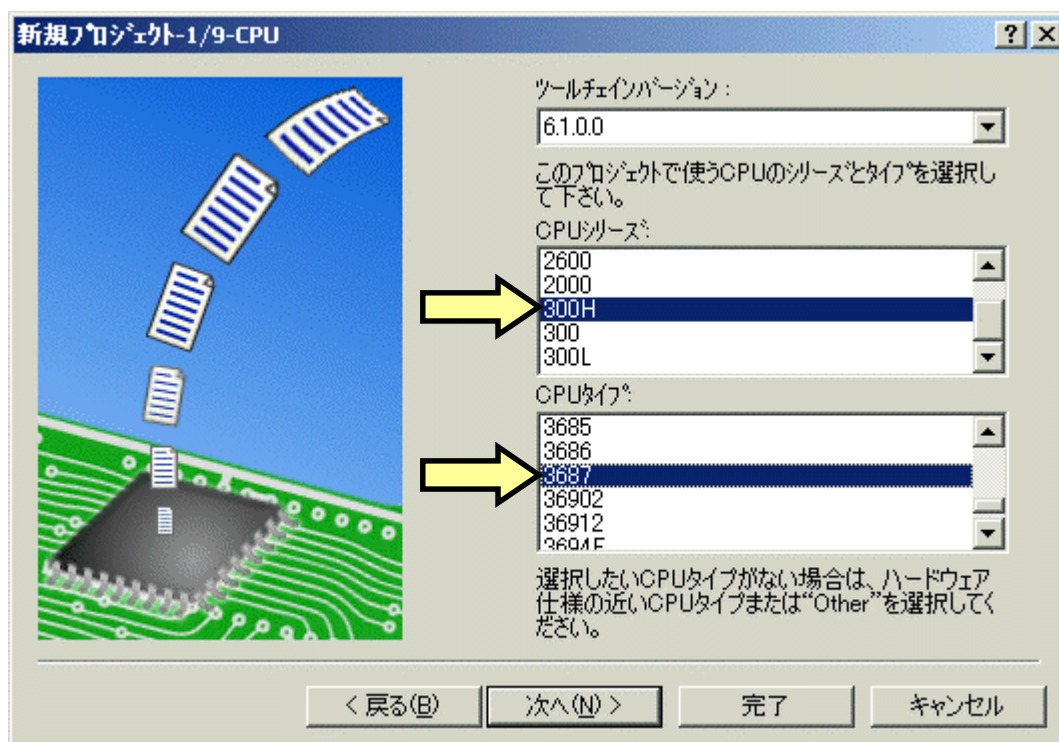
ワークスペースの場所を指定します。②右の「参照 (B) ...」ボタンをクリックします。そして、あらかじめ用意した HEW 専用作業フォルダ (ここでは C:\pirkus\program) を指定します。設定後、「ディレクトリ (D)」が正しいか確認して下さい。(③)

次にプロジェクトを指定します。今回は C 言語なので④「Application」を選択します。

入力が終わったら⑤「OK」をクリックして下さい。



「新規プロジェクト-1/9-CPU」で、使用する CPU シリーズ(300H)と、CPU タイプ(3687)を設定し、「次へ(N) >」をクリックします。



「新規プロジェクト-2/9-オプション」、「新規プロジェクト-3/9-生成ファイル」、「新規プロジェクト-4/9-標準ライブラリ」は変更しません。「次へ(N) >」をクリックして次の画面に進みます。

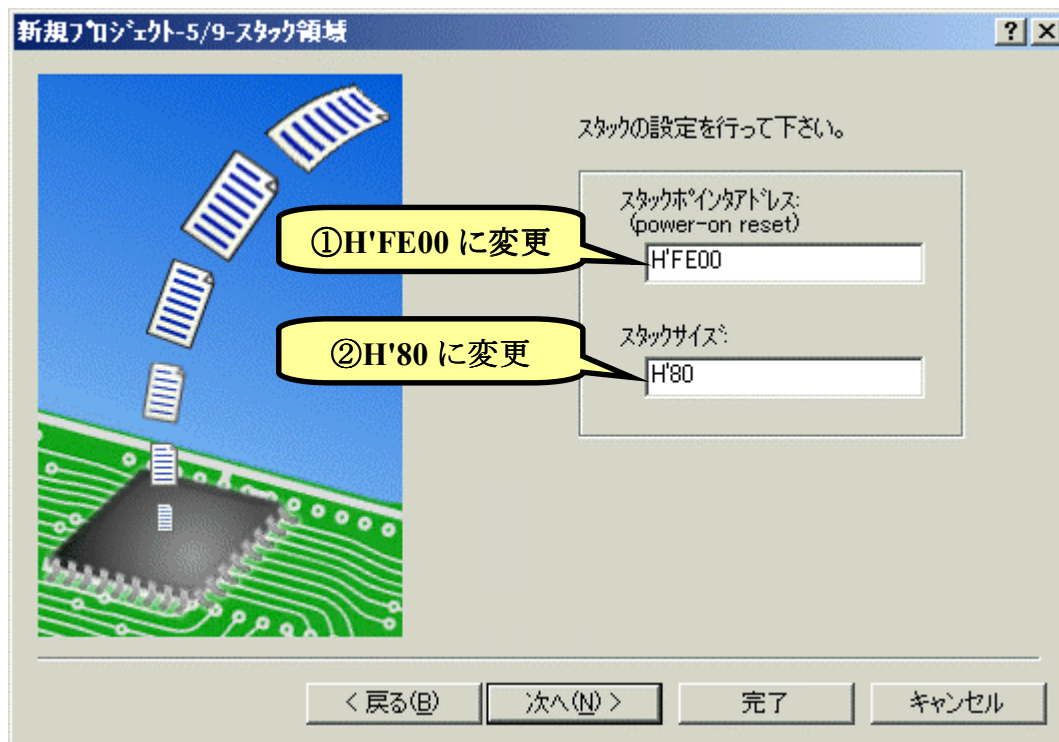


「新規プロジェクト-5/9-スタック領域」は変更しません。「次へ(N) >」をクリックして次の画面に進みます。

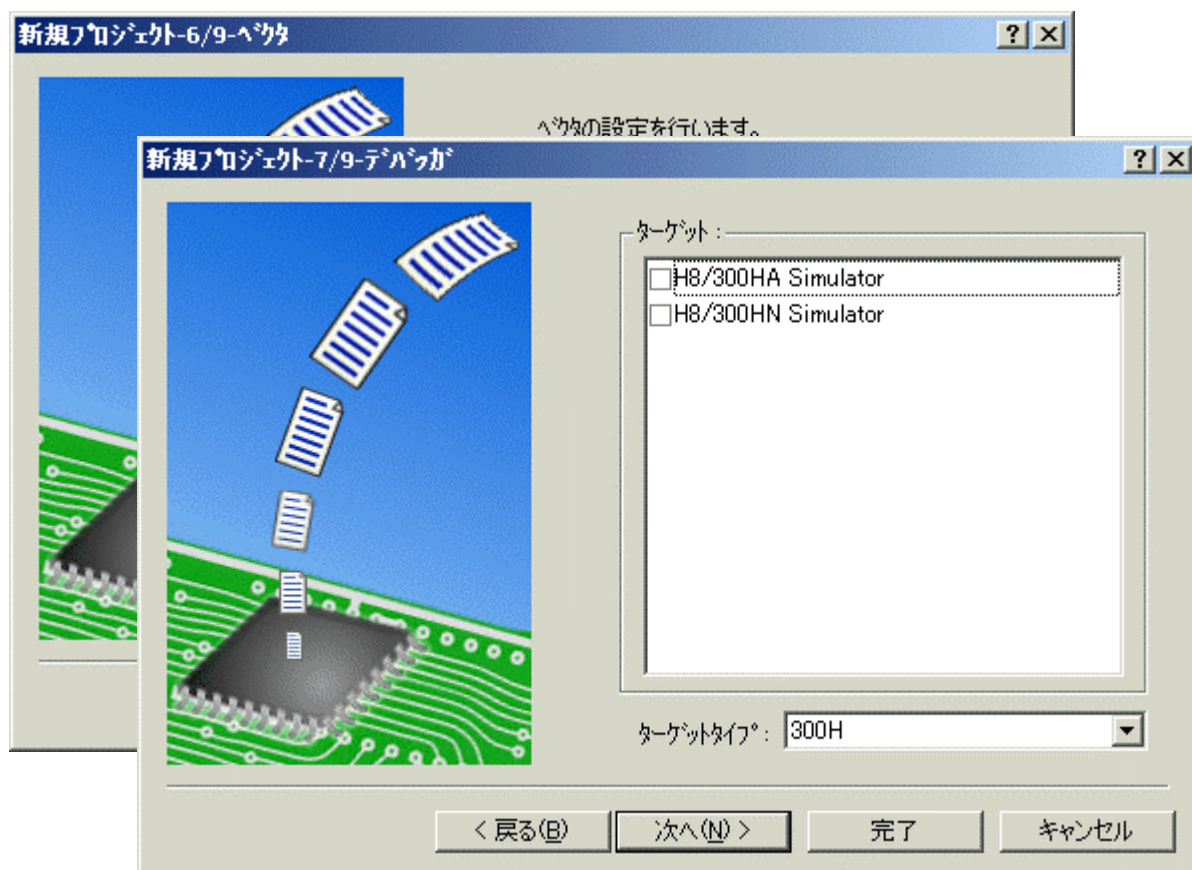


******* ハイパー-H8 を使用するとき(その 1) *******

「新規プロジェクト-5/9-スタック領域」でスタックのアドレスとサイズを変更します。①スタックポインタを H'FE00 に、②スタックサイズを H'80 にします。設定が終わったら「次へ(N) >」をクリックします。



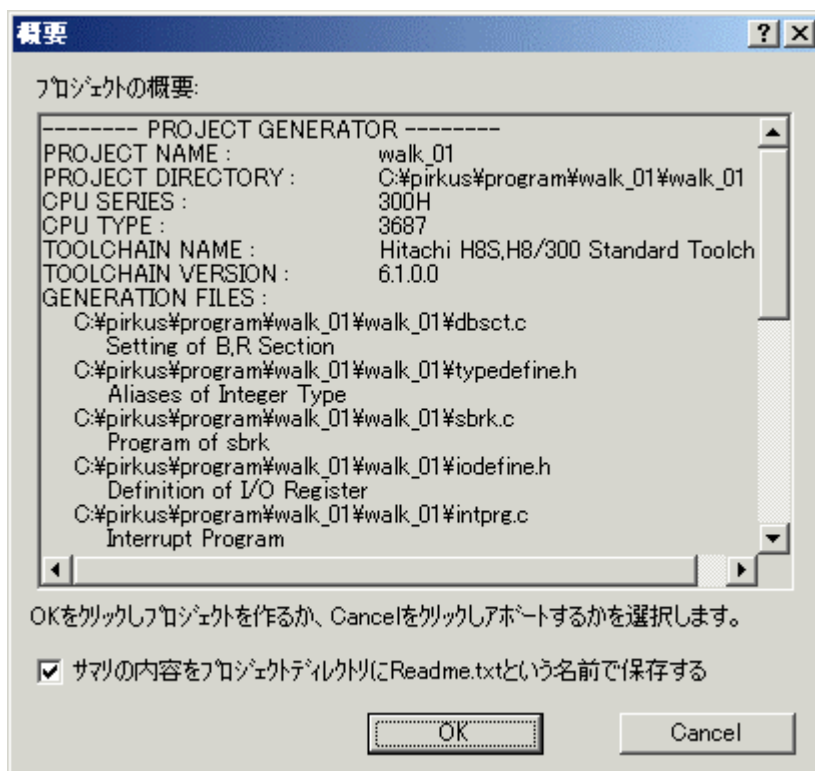
「新規プロジェクト-6/9-ベクタ」、「新規プロジェクト-7/9-デバッガ」は変更しません。「次へ(N) >」をクリックして順に次の画面に進みます。



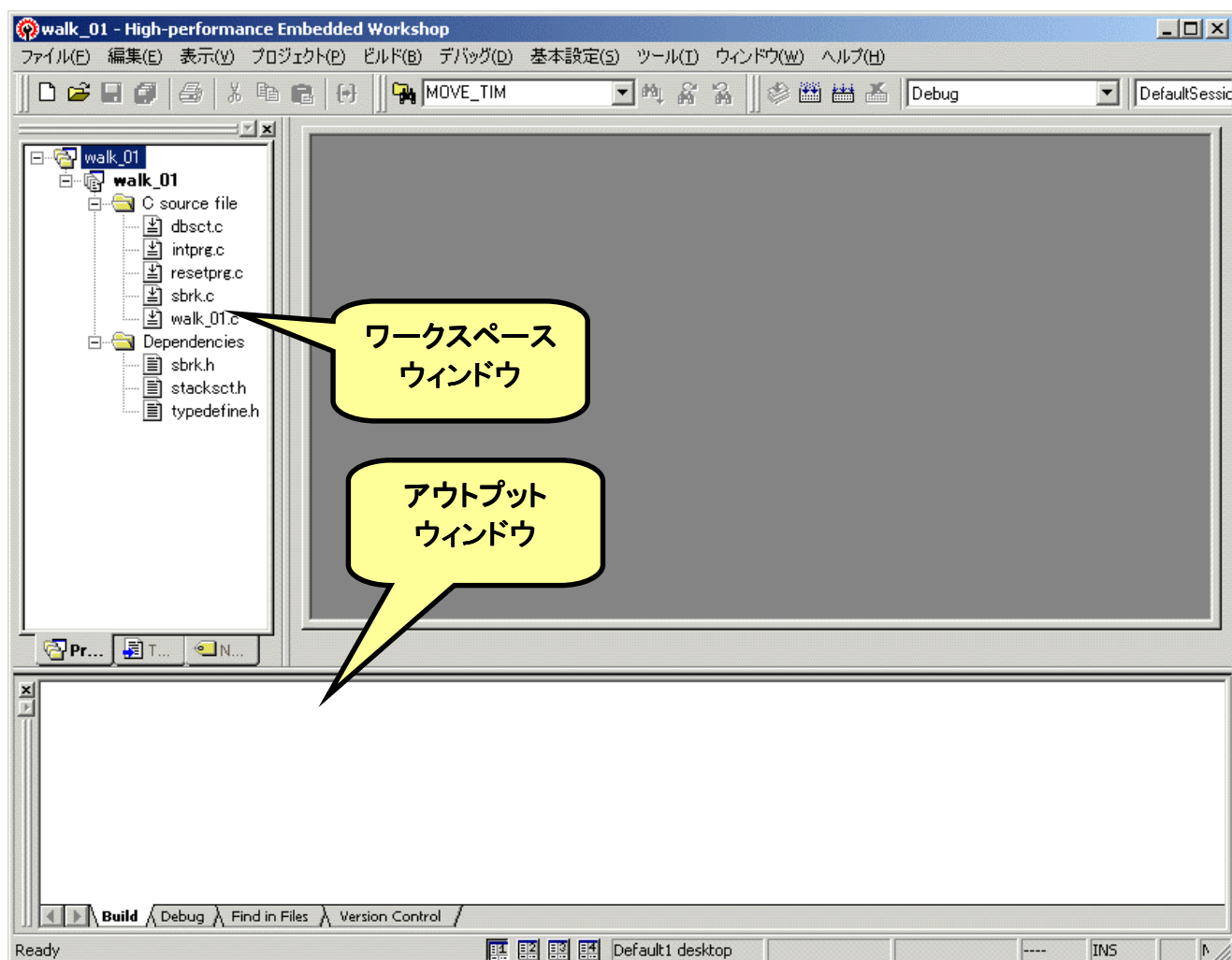
次は「新規プロジェクト-9/9-生成ファイル名」です。ここも変更しません。「完了」をクリックします。



すると、「概要」が表示されるので「OK」をクリックします。



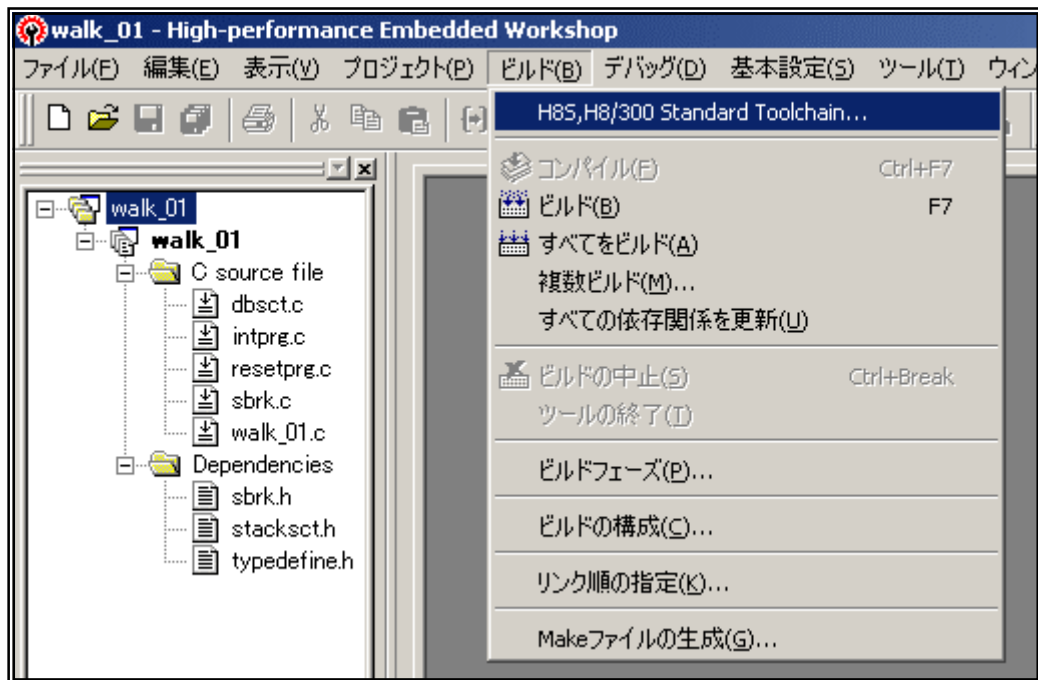
これで、プロジェクトワークスペースが完成します。HEW はプロジェクトに必要なファイルを自動生成し、それらのファイルは左端のワークスペースウィンドウに一覧表示されます。



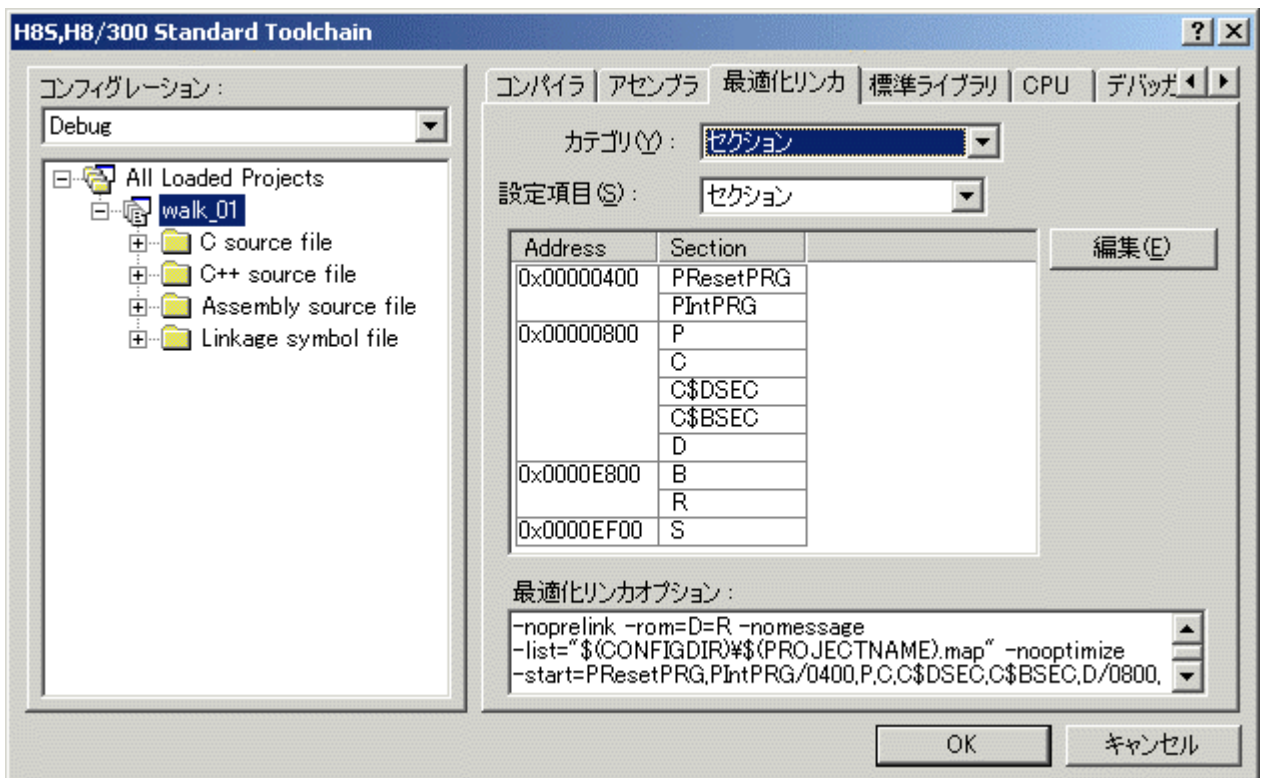
***** ハイパーH8 を使用するとき(その 2) *****

さて、これでプロジェクトは完成したのですが、ハイパーH8 を使うためにはセクションを変更してプログラムが RAM 上にできるようにします。

下図のように、メニューバーから「H8S,H8/300 Standard Toolchain...」を選びます。

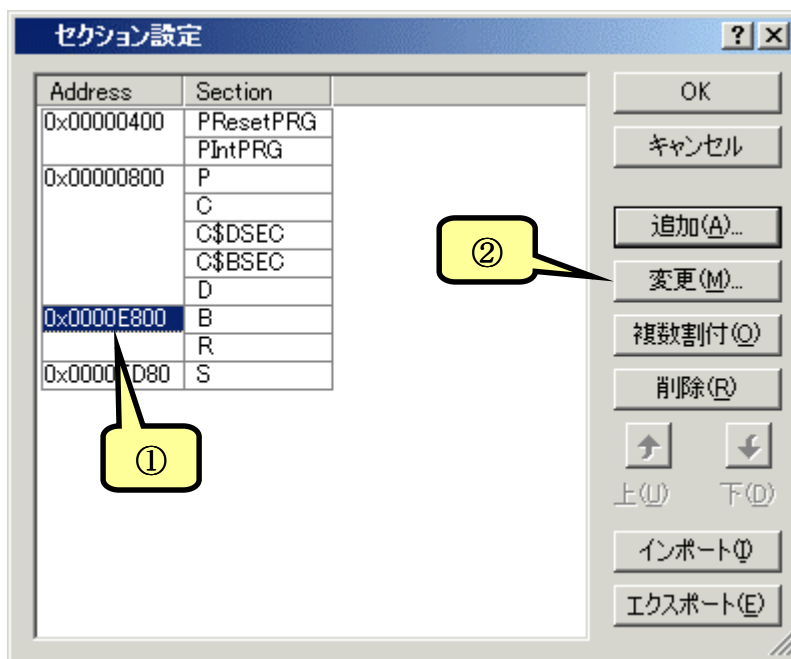


すると、「H8S, H8/300 Standard Toolchain」ウィンドウが開きます。「最適化リンカ」のタブを選び、「カテゴリ(Y)」のドロップダウンメニューの中から「セクション」を選択します。すると、下図のような各セクションの先頭アドレスを設定する画面になります。「編集(E)」ボタンをクリックしてください。

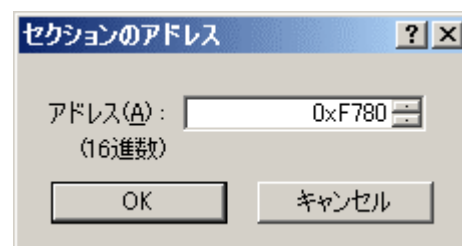


**** ハイパーH8を使用するとき(その3) ****

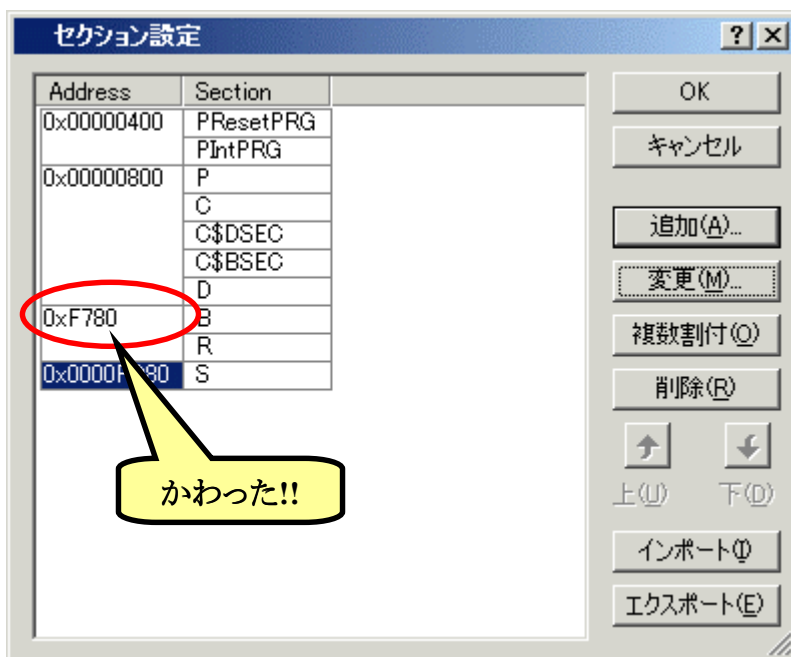
「セクション設定」ダイアログが開きます。それでは、「1. メモリマップの確認」で調べたメモリマップにあわせて設定していきましょう。最初に‘B’ Section のアドレスを変更します。デフォルトでは E800 番地になっていますね。①‘0x0000E800’というところをクリックして下さい。それから、②「変更(M)...」をクリックします。



そうすると、「セクションのアドレス」ダイアログが開きます。‘B’ Section は F780 番地から始まりますので、右のように入力して‘OK’をクリックします。



すると...

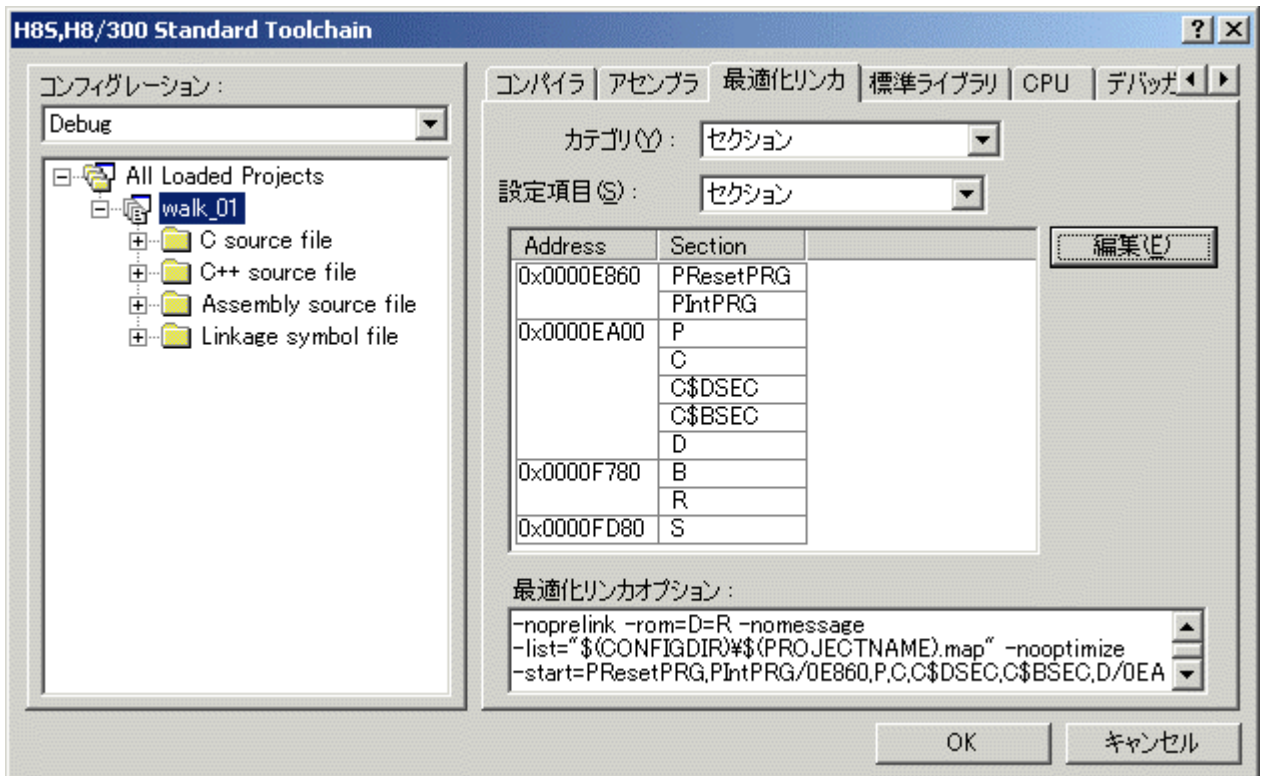
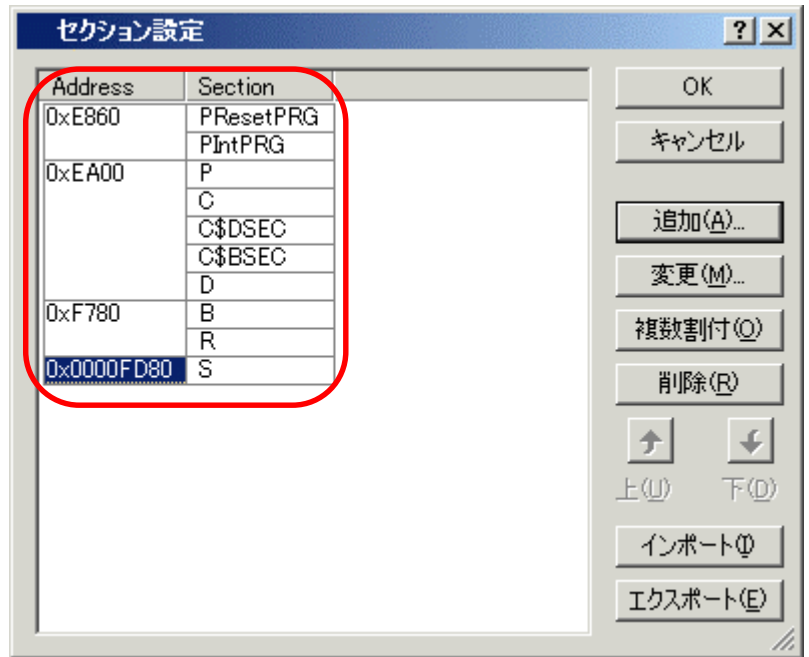


**** ハイパーH8を使用するとき(その4) ****

同じように、他のセクションも変更しましょう。メモリマップと同じようにSectionが指定されていることを確認します。ちゃんと設定されていたら「OK」をクリックします。

セクション設定の保存

次回のために今修正したセクション情報を保存することができます。下段の「エクスポート(E)」ボタンをクリックしてください。保存用のダイアログが開きますので好きな名前を付けて保存します。次回は「インポート(I)」ボタンをクリックすると保存したセクション設定を呼び出すダイアログが開きます。(おすすめ!!)



もう一度確認してから「OK」をクリックして‘H8S, H8/300 Standard Toolchain’ウィンドウを閉じます。

■ プログラムの入力

HEW のワークスペースウィンドウの 'walk_01. c' をダブルクリックしてください。すると、自動生成された 'walk_01. c' ファイルが開きます。

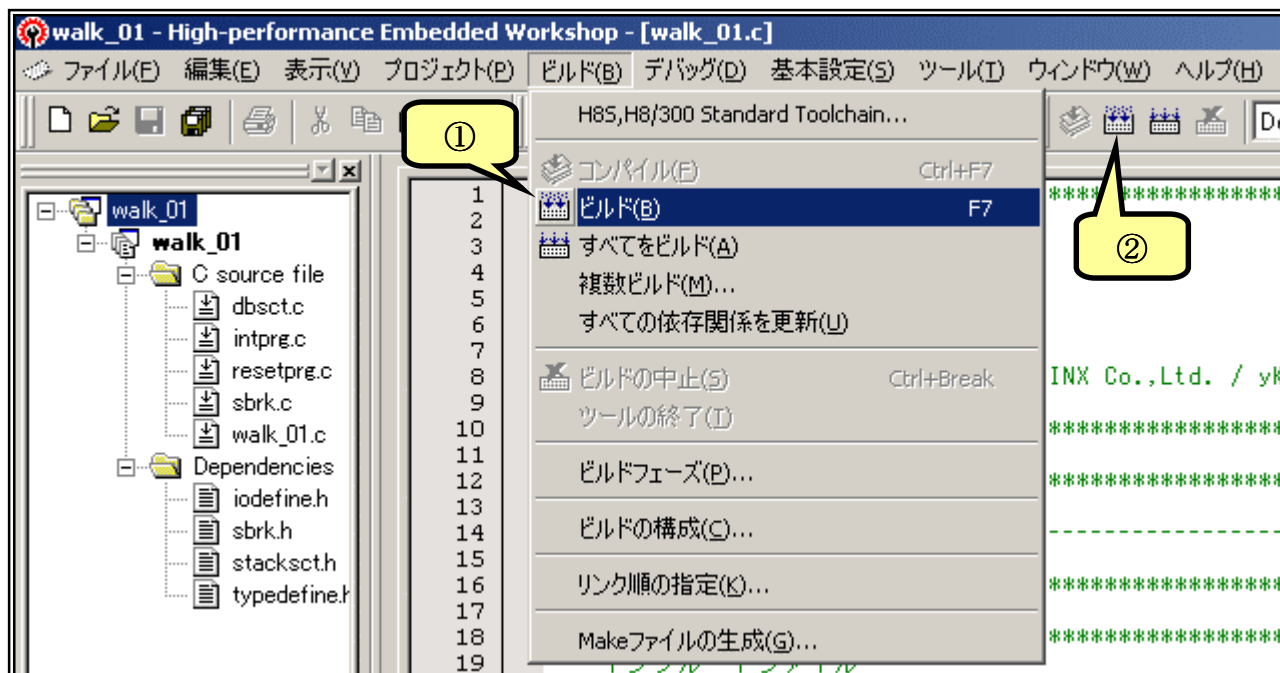
```
1  /*****  
2  /*  
3  /* FILE      :walk_01.c  
4  /* DATE      :Mon, Feb 13, 2006  
5  /* DESCRIPTION :Main Program  
6  /* CPU TYPE   :H8/3687  
7  /*  
8  /* This file is generated by Renesas Project Generator (Ver.4.0).  
9  /*  
10 /*****  
11  
12  
13  
14 #ifdef __cplusplus  
15 extern "C" {  
16 void abort(void);  
17 #endif  
18 void main(void);  
19 #ifdef __cplusplus  
20 }  
21 #endif  
22  
23 void main(void)  
24 {  
25  
26 }  
27  
28 #ifdef __cplusplus  
29 void abort(void)  
30 {  
31  
32 }  
33 #endif  
34
```

このファイルに追加・修正していきます。本マニュアルの「4 二足歩行にチャレンジしよう」のソースリストのとおり入力してみてください。なお、C 言語の文法については、HEW をインストールしたときに一緒にコピーされる「H8S, H8/300 シリーズ C/C++コンパイラ, アセンブラ, 最適化リンケージエディタ ユーザーズマニュアル」の中で説明されています。

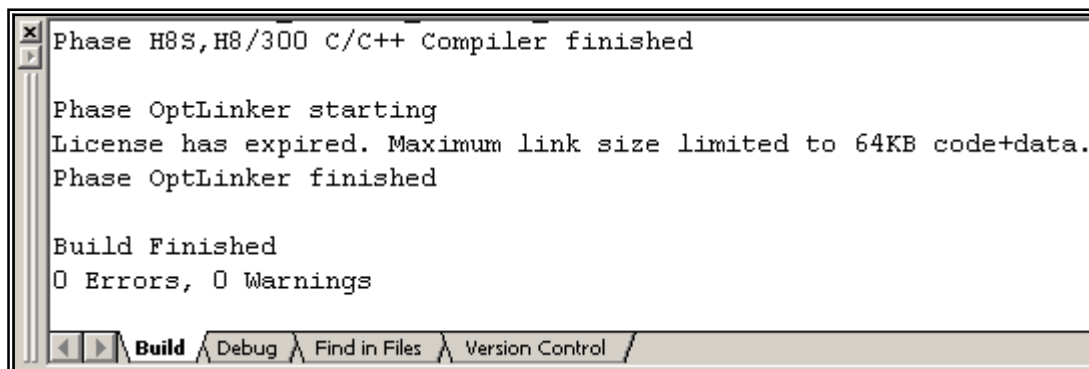
また、このプログラムはタイマ Z のオーバーフロー割込みを使っています。HEW のワークスペースウィンドウの 'intprg. c' をダブルクリックしてください。すると、自動生成された 'intprg. c' ファイルが開きますので追加・修正します。変更点は本マニュアルの「3 ホームポジションを作ろう」の 'intprg. c' と全く同じです。ソースリストどおりに入力して下さい。

■ ビルド!!

では、ビルドしてみましょう。ファンクションキーの[F7]を押すか、図のように①メニューバーから‘ビルド’を選ぶか、②ツールバーのビルドのアイコンをクリックして下さい。



ビルドが終了するとアウトプットウィンドウに結果が表示されます。文法上のまちがいがなければ「0 Errors」と表示されます。



エラーがある場合はソースファイルを修正します。アウトプットウィンドウのエラー項目にマウスカーソルをあててダブルクリックすると、エラー行に飛んでいきます(このあたりの機能が統合化環境の良いところですね。)ソースファイルと前のページのリストを比べてまちがいをなく入力しているかももう一度確認して下さい。

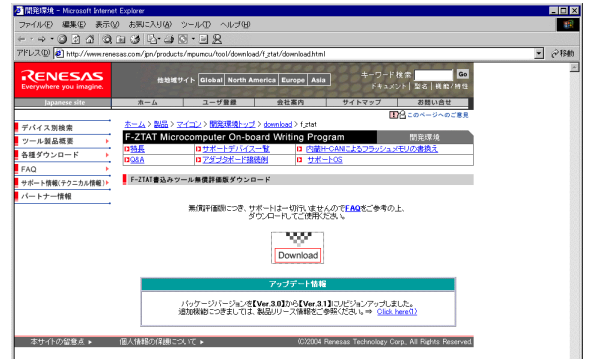
さて、Error ではなく Warning の場合、何も問題ないケースも多いのですが、中には動作に影響を与えるものもあります。「H8S, H8/300 シリーズ C/C++コンパイラ, アセンブラ, 最適化リンケージエディタ ユーザーズマニュアル」の 539 ページからコンパイラのエラーメッセージが、621 ページから最適化リンケージエディタのエラーメッセージが載せられていますので、問題ないか必ず確認して下さい。


FDT によるプログラムの書き込み手順

ハイパーモニタやユーザが作成したプログラムを flashROM に書き込むには“FDT(Flash Development Toolkit)”を使用します。無償版の FDT がルネサステクノロジから提供されています。

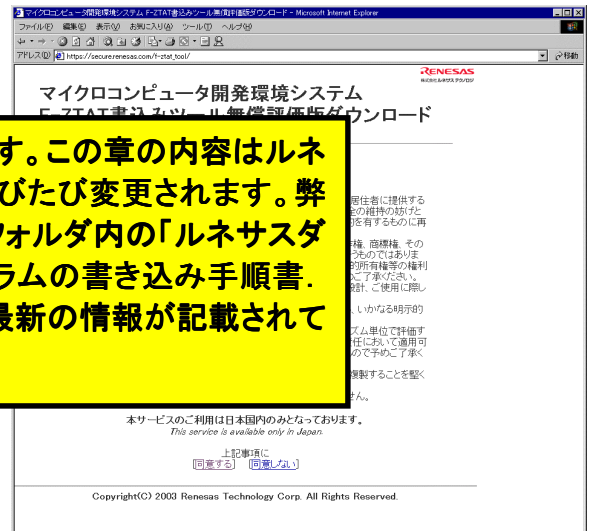
■ FDT のダウンロード

1. FDT は以下のサイトからダウンロードして下さい。
http://www.renesas.com/jpn/products/mpumcu/tool/download/f_ztat/download.html
(2004 年 4 月現在)

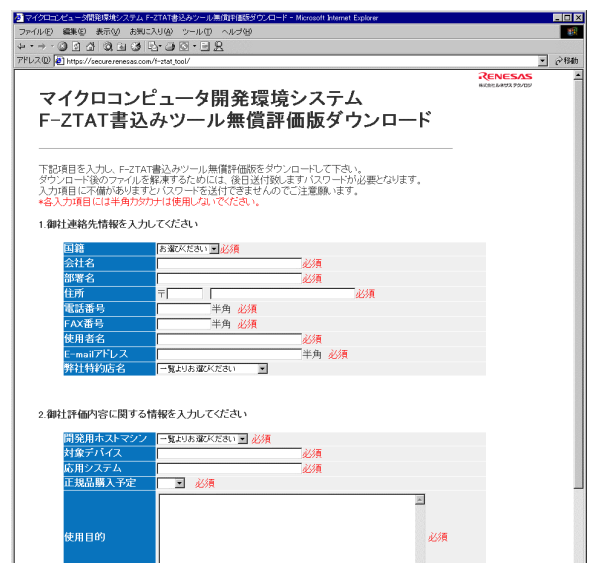


2. ダウンロードするには  をクリックします。
3. 注意事項が記されたページへ移動するので内容を読み、同意した上で[同意する]をクリックします。

この画面は 2004 年 4 月現在です。この章の内容はルネサステクノロジの更新のため、たびたび変更されます。弊社 CD の「必ずお読みください」フォルダ内の「ルネサスダウンロード.pdf」と「モニタプログラムの書き込み手順書.pdf」をご覧ください。その時点で最新の情報が記載されています。

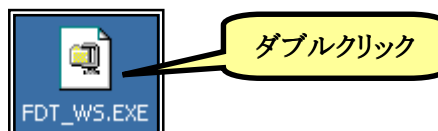


4. ユーザ情報を入力し、プログラムをダウンロードします。入力したメールアドレスに、プログラムを解凍する際必要なパスワードが送られてくるので入力事項に間違いが無い様、注意して下さい。
5. ダウンロード先はデスクトップにすると便利です。デスクトップに“FDT_WS.EXE”がダウンロードされたか確認しましょう。

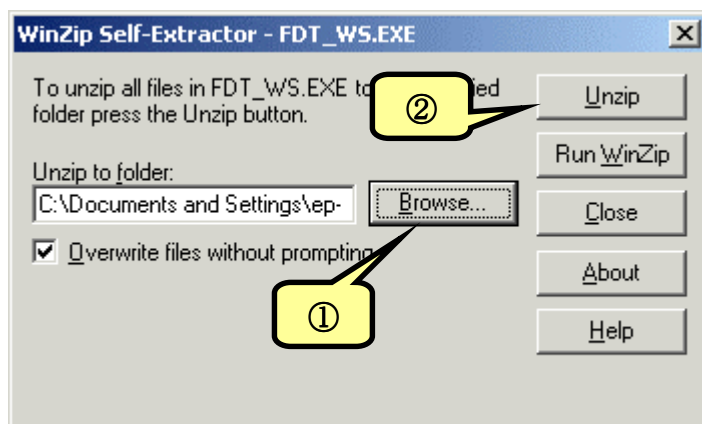


■ FDT のインストール

1. ダウンロードした“FDT_WS.EXE”をダブルクリックします。



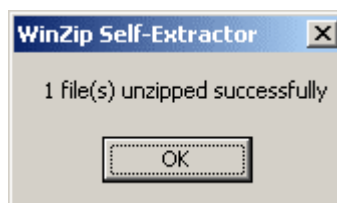
2. 右のようなダイアログが開きますので、①“Browse”をクリックして解凍先のフォルダを指定します。デスクトップにすると便利です。指定したら②“Unzip”をクリックします。



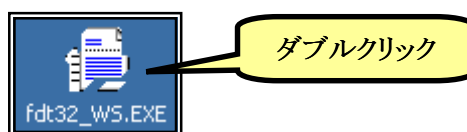
3. パスワードの入力ダイアログが開きます。ルネサステクノロジからメールで送られてきたパスワードを入力して“OK”をクリックしてください。



4. 解凍が始まります。右のダイアログが表示されたら成功です。“OK”をクリックしてください。さらに“Close”をクリックして全てのダイアログを閉じます。



5. デスクトップに“fdt32_WS.EXE”ができています。このファイルをダブルクリックすればインストールが始まります。あとは画面の指示に従ってインストールしてください。



■ プログラムの書き込み手順

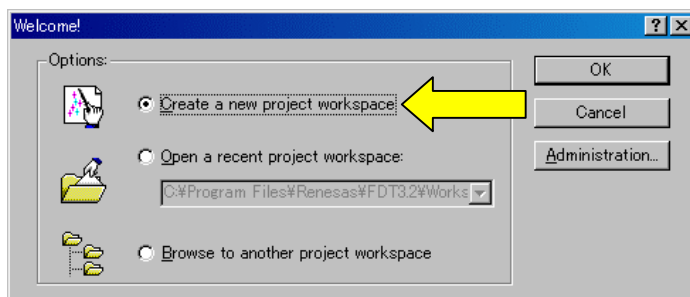
H8 書き込みツール“Flash Development Toolkit (FDT)”を用いて FDT のセッティングからプログラム書き込みまで、順を追って説明していきます。ここではハイパーモータ‘ハイパーH8’の書き込みを例にします。

FDT のセッティング(ワークスペースとプロジェクトの立ち上げ)

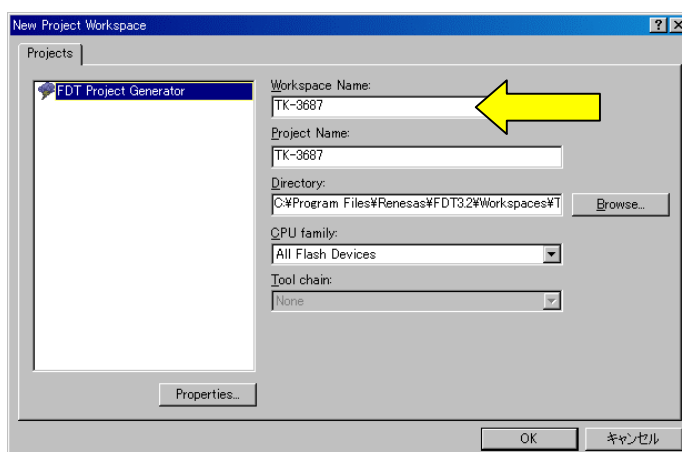
1. スタートメニューから“**Flash Development Toolkit 3.2**”を起動します。



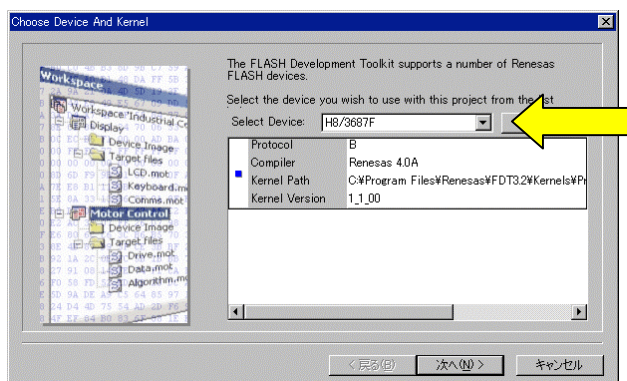
2. 右図のようなダイアログが開くので、“**Create a new project workspace**”を選択して **OK** をクリックします。



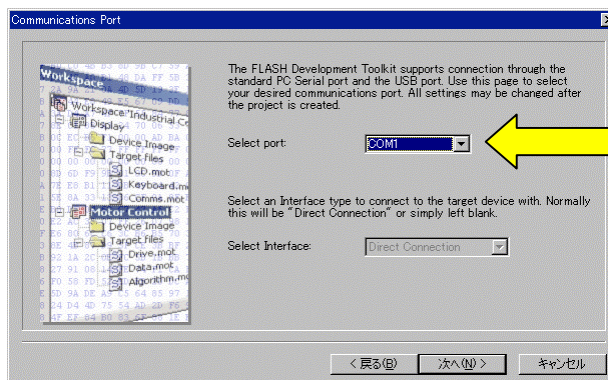
3. “**Workspace Name**”を決定します。名前は自由に決めて結構です(ここでは TK-3687 としています)。またワークスペースを作成するディレクトリを指定したい場合は“**Directory:**”の **Browse...** をクリックしディレクトリを指定して下さい。よければ **OK** をクリックし次へ進みます。




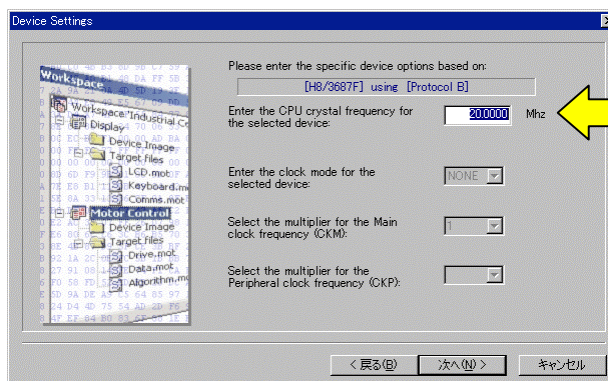
4. デバイスを選択します。“**Select Device:**”の欄で“**H8/3687F**”を選択し、**次へ(N) >** をクリックします。




5. 使用する Com ポートを選択します。“**Select port:**”で接続する Com ポートを選択し、をクリックします。




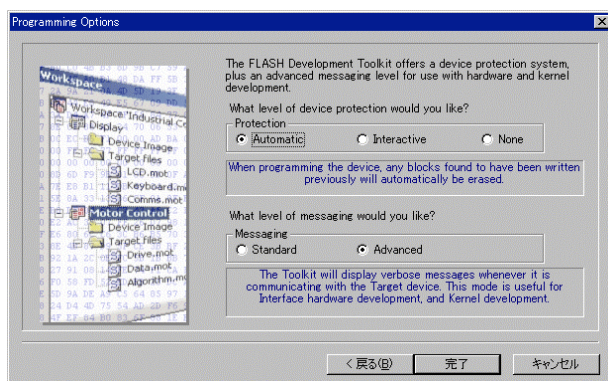
6. CPU のクロックを入力します。“**Enter the CPU crystal frequency ...**”の欄に実装されているクロックの周波数“**20.00**” MHz を入力し、をクリックします。



7. この後出てくる項目は入力・変更の必要はないので  をクリックします。

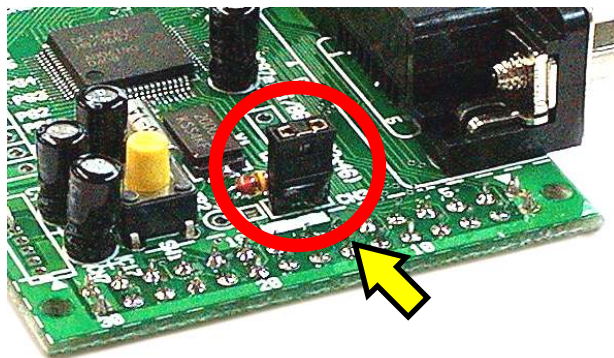


ここでも変更は無いので  をクリックします。以上でワークスペースとプロジェクトの立ち上げは完了です。

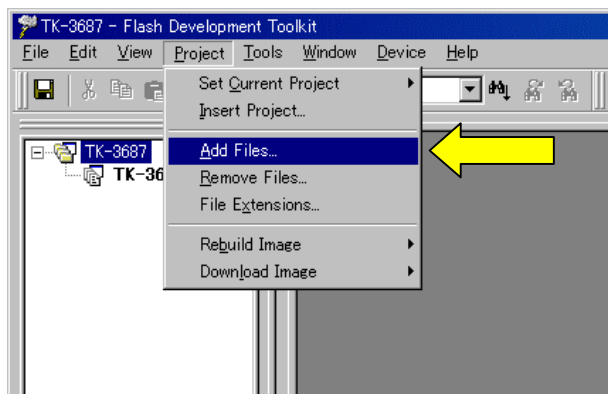


モニタファイルのダウンロード

1. まず TK-3687mini とパソコンとを接続します。基板上のジャンパ・**JP1** を付属のジャンパソケットでショートさせ、RS-232C ケーブルでパソコンと接続し電源を投入します。ファイルをダウンロードする為に CPU をブートモードで起動しなくてはならないのですが、この TK-3687mini にはブートモードで起動する為に必要な P85 のプルアップ抵抗が入っていません。そこでブートモードで起動する為に電源を入れたら2、3回リセットスイッチを押して下さい。



2. 次にダウンロードするファイルをプロジェクトに追加します。メニューバーから“**Project > Add Files...**”を選択します。




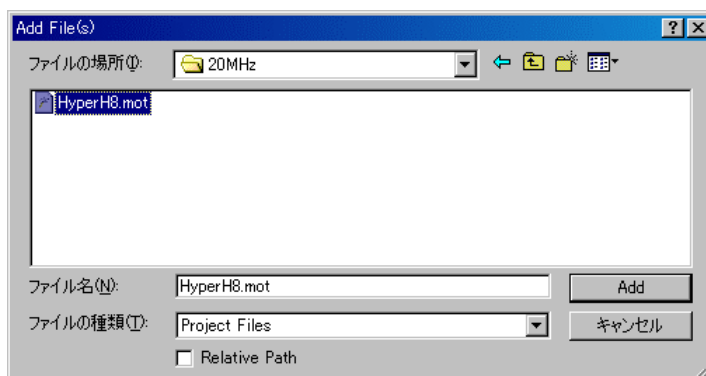
3. モニタファイル‘HyperH8. mot’を選択します。モニタファイルは製品に付属している CD-ROM に収録されています。

CD-ROM¥TK-3687¥モニタプログラム ¥20MHz

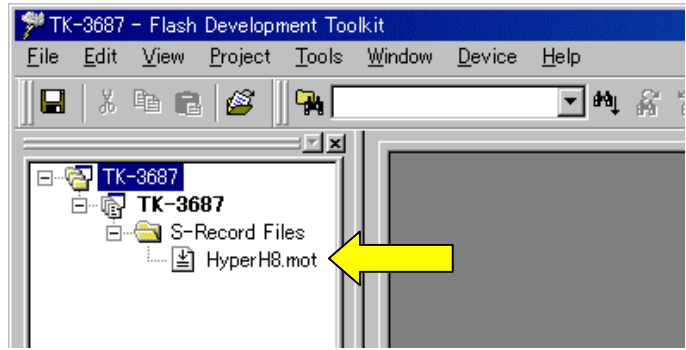
但し CD-ROM に収録されているモニタファイルはご購入時でのバージョンですので、最新版を web からダウンロードする事をお勧めします。弊社ホームページよりダウンロードして下さい。

<http://www2.u-netsurf.ne.jp/~toyolinx/program/tk3687/HyperH8.mot>

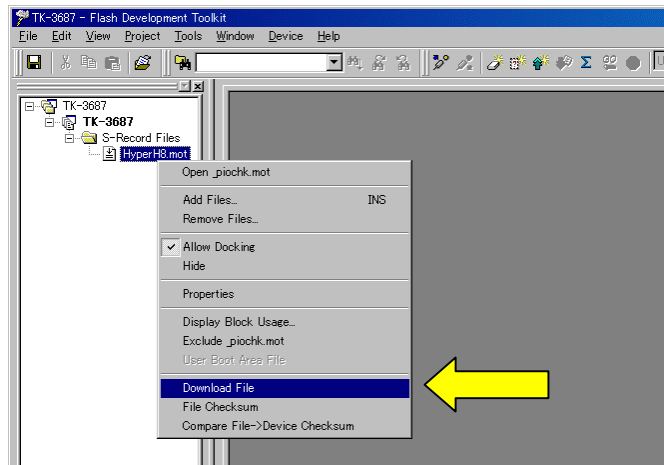
モニタファイルを選択したら  をクリックして下さい。



4. 以上でファイルが追加されました。画面左のルートディレクトリ内“S-Record Files”に選択したモニタファイルが追加されたのを確認して下さい。



5. モニタファイルをデバイスへダウンロードします。追加されたモニタファイルを右クリックし、“Download File”を選択すると、ダウンロードを開始します。



6. 右図の“Image successfully written to device”のメッセージが表示されれば終了です。先程ショートしたジャンパ・JP1を外し、リセットスイッチを押して下さい。ダウンロードしたプログラムが走り始めます(通常モード)。

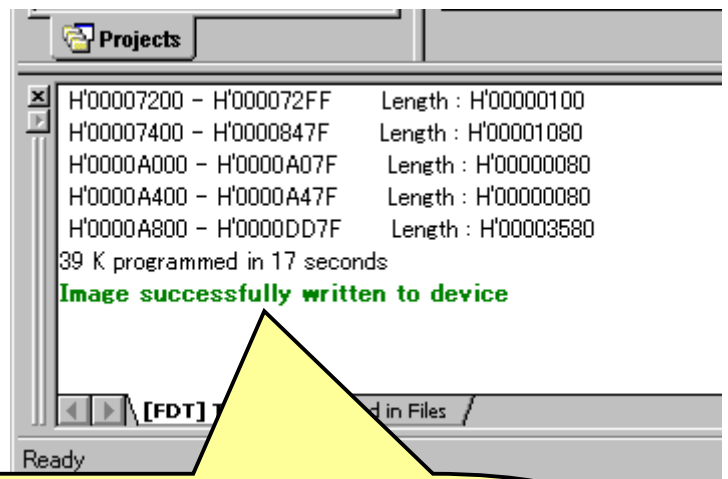


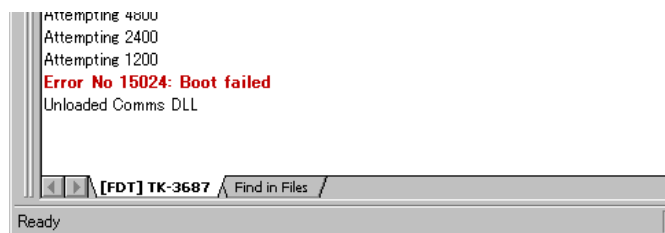
Image successfully written to device

このメッセージが出ればダウンロード完了！！

7. 次回はワークスペースを作成したディレクトリ内にある“TK-3687. AWS”をダブルクリックすれば、ここで設定した状態で起動します。

■ うまく書き込めないときは

書き込み完了のメッセージが出ず右図のような“Boot failed”が表示された場合は次の事を確認して下さい。

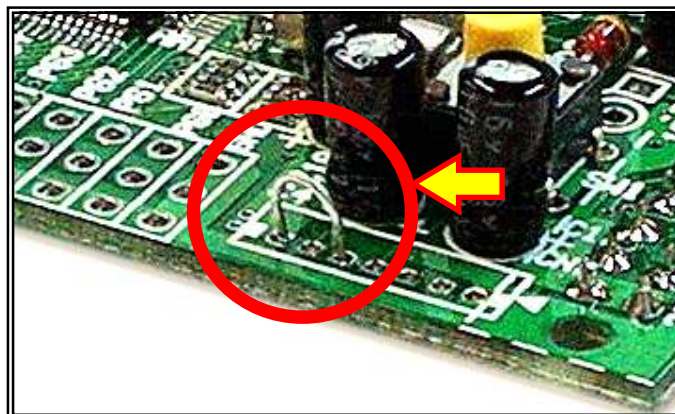


1. ハンダ付けした部品の確認

取り付けした部品をもう一度確認しましょう。部品の極性やハンダ付けが上手にできているかよく確認して下さい。特にレギュレータを逆に取り付けてしまうと全く動きません。また、電源コネクタの向きも注意しましょう。コネクタは逆になっていませんか？

2. 部品、ハンダ付けの確認で問題なければ次の手順を試して下さい。

“Boot failed”が表示された場合は再度リセットスイッチを押して、9頁の操作を行なって下さい。繋がるまでリセットと9頁の操作を行ないます。もし4～5回行なっても繋がらない場合は次の処置を行なって下さい。まず、一旦電源を外します。次に基板右下にあるCN7の5番と7番を抵抗のリードなどを差し込んでショートさせます(三角印のある方が1番・右図参照)。後で外すのでハンダ付けは不要です。



差し込み終わったら再度電源を入れ、9頁の操作を行なって下さい。“Image successfully written to device”のメッセージが表示されれば終了です。先程ショートしたジャンパ・JP1とCN7・5番－7番を外し、リセットスイッチを押して下さい。ダウンロードしたプログラムが走り始めます(通常モード)。

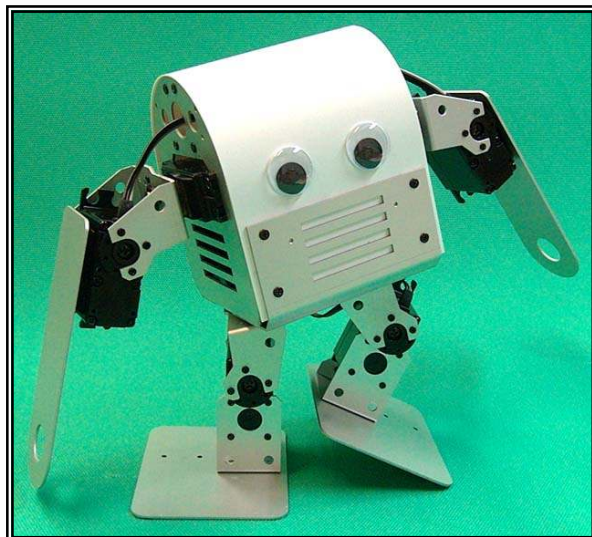
以上の事を行なっても動作しない場合は、弊社までご連絡願います(連絡先は巻末に掲載しています)。

ロボット制御とモーションエディタの製作

モーションエディタの製作には、①仕様の決定、②マイコンのプログラム、③パソコンのプログラム、が関連する。モーションエディタ全体に関連するスキルを身に付けるためのカリキュラム案。(但し、HEW や H8/3687, VBA についての基本的な知識はあるものとする。)

■ 仕様の決定

- ・必要な機能の分析。
- ・必要なコマンドを検討、プロトコルの決定。
 - ① RC サーボモータの角度を指定(チャンネル毎), 即移行, モーションデータ作成時に使用。
 - ② RC サーボモータのホームポジションを指定(チャンネル毎)。
 - ③ モーションデータを送信(モーション番号毎), RC サーボの角度と移行時間。
 - ④ モーション数を指定。
 - ⑤ モーションデータをEEPROM にセーブ。
 - ⑥ モーションデータをEEPROM からロード。
 - ⑦ モーションの再生。
 - ⑧ 現在位置からモーションデータの位置に移行。
 - ⑨ パソコンとロボットがつながっているか確認する。
- ・仕様の決定。



■ マイコンのプログラム

- ・Type-02, TK-3687mini, 統合環境HEW, C 言語で作成する。
- ・モーションデータのデータ構造。構造体の理解。
- ・RC サーボモータの動かし方。タイマ Z の使い方。PWM。スムージング処理。角度指定。
 - ① データは角度だがタイマ Z はカウント値で指定するので、角度→カウント値の換算が必要。
 - ② RC サーボを変更しても、その特性にあわせてすぐに調整できるようにしておく。
 - ③ モーションからモーションの移行はスムージング処理を加える。
 - ④ 移行時間はモーションデータに含め、各モーション毎に指定できるようにする。
- ・EEPROM の使い方。I²C バス。EEPROM 内のモーションパターンデータによって動作。
- ・シリアルポートの使い方。割り込み処理。
 - ① キュー(リングバッファ)の理解。
- ・プロトコルに基づき、コマンドに対応する動作をプログラミングする。

この時点ではパソコンプログラムがないので、ハイパーターミナルでコマンドを入力し、マイコンのデバッグを行なう。この状態でマイコンプログラムは完成させる。→コマンドはアスキーコードにする。

- ・パソコンなしでも、電源オンで EEPROM からデータをロードし、自動的にモーションを再生するようしておく。

■ VBAによるパソコンのプログラム(ステップ 1)

- ・この時点ではマイコンプログラムが完成しているので、Type-02 を動かしながらパソコンプログラムをデバッグしていく。
- ・VB でもよいが、ほとんどのパソコンにインストールされている Excel&VBA を使用する。
- ・モーションデータを Excel の表として作成、保存する。例えば以下のようにする

	Servo-0	Servo-1	Servo-2	Servo-3	Servo-4	Servo-5	Servo-6	Servo-7
モーション-0	XX.X	XX.X	XX.X	XX.X	XX.X	XX.X	XX.X	XX.X
モーション-1	XX.X	XX.X	XX.X	XX.X	XX.X	XX.X	XX.X	XX.X
モーション-2	XX.X	XX.X	XX.X	XX.X	XX.X	XX.X	XX.X	XX.X
モーション-17	XX.X	XX.X	XX.X	XX.X	XX.X	XX.X	XX.X	XX.X
モーション-18	XX.X	XX.X	XX.X	XX.X	XX.X	XX.X	XX.X	XX.X
モーション-19	XX.X	XX.X	XX.X	XX.X	XX.X	XX.X	XX.X	XX.X

- ・通信プログラムの作り方を理解。
MSCOMM32 のインストール。標準では用意されていないので、Google などで検索して入手する。
- ・ユーザインタフェースとコマンド送信。

■ VBAによるパソコンのプログラム(ステップ 2)

- ・第 2 段階として、スライダコンポーネントなどを使用して、入力しやすいインタフェースを目指す。
- ・ロボットの写真を利用して、直感的に操作できるようにする。
- ・ひとつのボタンにひとつのコマンドを割り付けるのではなく、ひとつのボタンに一連のコマンドを割り付けることで、より効果的なインタフェースになる可能性を考慮する。ボタンをコマンドマクロとして捕らえる。

■ VBA以外の言語で作る

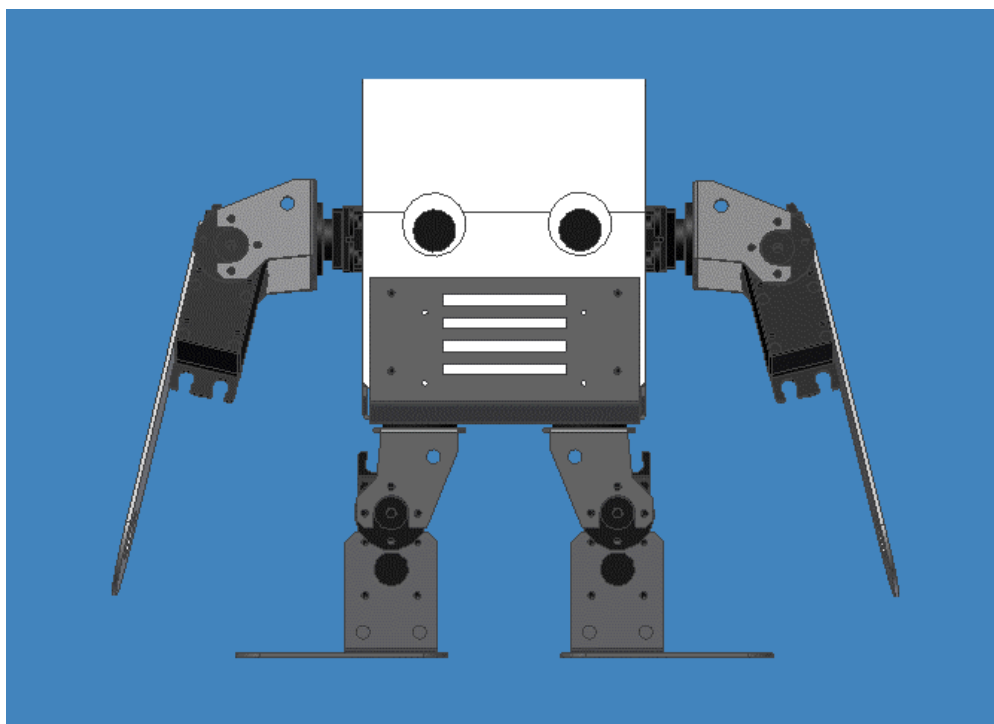
- ・VB, VC++, JAVAで、マン・マシンインタフェースの改善を図る。

参考:市販のモーションエディタのコマンドの一部を示す。

① モーションエディタ & PCLINK : JinSato さんのプログラム

RC サーボモータの相対角度を指定			マイコン→PC(動作完了後, 送信)		
PC→マイコン			マイコン→PC(動作完了後, 送信)		
's'	73h	コマンド-1	'\$'	24h	
'r'	72h	コマンド-2	'0'	4Fh	
','	2Ch	区切り	'K'	4Bh	
○	30h-32h	サーボ番号 (0-23)	cr	0Dh	
○	30h-39h				
','	2Ch	区切り			
'-'	2Dh	負のときだけ追加			
○	30h-39h	相対角度 (○○○.○°)			
○	30h-39h				
○	30h-39h				
○	30h-39h				

② PODTerm & PODLink : Pirkus が紹介しているオープンソース(現在調査中)



株式会社東洋リンクス

※ご質問はメール, または FAX で…
ユーザーサポート係(月～金 10:00～17:00, 土日祝は除く)
〒102-0093 東京都千代田区平河町 1-2-2 朝日ビル
TEL:03-3234-0559
FAX:03-3234-0549
E-mail:toyolinx@va.u-netsurf.jp
URL:<http://www2.u-netsurf.ne.jp/~toyolinx>

20091222