マイクロマウス HM StarterKit パート３
『取扱説明書』
ソフトウェア解説編

株式会社アールティ

2018 年 7 月

目次

１．ソフトウェアの解説 3

1.1 Step1 Buzzer ... 3
1.2 Step2 Motor ... 6
1.3 Step3 光センサ ... 8
1.4 Step4 I2C と IO ... 11
1.5 Step5 SPI とエンコーダ ... 15
1.6 Step6 SPI とジャイロ ... 17
1.7 Step7 迷路走行 ... 18
1.9 Step9：迷路情報、センサ値を記憶しよう ... 25

2. 故障かな? と思ったら 26

4. 著作権について 27

5. ソフトウェアについて 27

6. 改訂履歴 27

製造元 27

製品に関するお問い合わせ 27

1．ソフトウェアの解説

HM_StarterKit パート 2 取扱い説明書ハードウェア製作編、パート 3 取扱い説明書環境構築編でプログラムを書き込める環境ができました。ここで 7 個のサンプルプログラムの解説を行います。サンプルプログラムの解説を読みながら、実際に動作させて、解説どおりに動くことを確認してください。確認ができたら次は自分でプログラムを書き加えていくことにより、思い通りに動作させることができるようになります。

ここで注意して頂きたいことがあります。プログラムの入ったフォルダの名前はなるべく日本語を使わないでください。特に括弧とスペースを付けますと CS+でコンパイル時にエラーが出る場合があります。またプログラム名も同様です。なるべくプロジェクト名、プログラム名、それらの保存フォルダの名前には英語で、スペースの代わりにアンダーバを使ってください。

また、本マニュアルではルネサス社の「RX63N グループ、RX631 グループ　ユーザーズマニュアル　ハードウェア編 Rev1.80」(以下 RX631 ハードウェアマニュアル)を一部引用しております。ページ数も記載しておりますが、リビジョン 1.80 であることにご注意ください。

ここでは STEP ごとに以下の流れで進めていきます。

概 要 大まかにその STEP で何をおこなうプログラムなのかを解説します。

プログラムソース 実際のプログラムの中からその STEP での重要なところを掲載します。
ライブラリ化しているファイル名については末尾に一覧を掲載します。

動作確認と解説 プログラムを実際に書き込んでみてどのように動作するのかを解説していきます。

1.1 Step1 Buzzer

概要

マイコンの基本機能である IO（インプット、アウトプット）ピンを使用してブザーを鳴らすサンプルプログラム。マイコンプログラムの基本であるレジスタアクセスと基本的な C 言語の文法について解説を行う。

Step1_Buzzer のメインプログラム

```
void main(void)
{
 /********************************************************
 I/Oピンの設定
                ブザー用IOポートの設定
 ********************************************************/
 //ブザー関連
 PORTB.PDR.BIT.B5 = 1;

 /********************************************************
 メインプログラム
                ブザーON
 ********************************************************/
 int i = 0;
 while(1){
                PORTB.PODR.BIT.B5 = 1;
                for(i = 0; i < 100*1000*1; i++);
                PORTB.PODR.BIT.B5 = 0;
                for(i = 0; i < 100*1000*1; i++);
 }
}
```

動作確認

このサンプルの CS＋「C:\ HM_StarterKit-master\Sample_Programm\step1_buzzer」フォルダ内の「HM_StarterKit.mtpj」です。それをビルドし、RFP を使って、C:\ HM_StarterKit-master\Sample_Programm\step1_buzzer\DefaultBuild」フォルダ内の「HM_StarterKit.mot」を書き込みます。

プログラムの解説

1.1.1 メインプログラムの動作

① ブザーの接続されている B5 ピンの機能設定を行う

② 一定秒数 B5 ピンの出力を High にする

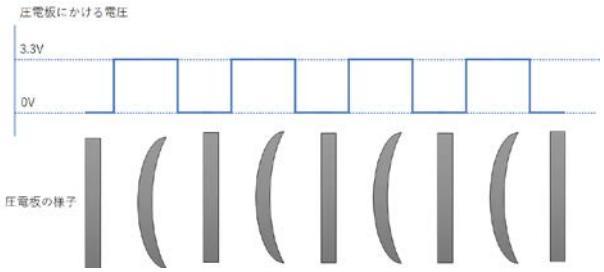
③ 一定秒数 B5 ピンの出力を Low にする

②、③を繰り返す

1.1.2 ブザーを鳴らす方法

今回使用しているブザーは圧電ブザーというものです。圧電ブザーは内部に圧電板というものが内蔵されています。電圧をかけると圧電板が歪みます。電圧をかけないと圧電板が元に戻ります。これを連続的に繰り返すことで、音を出すことが出来ます。

よってブザーにかける電圧を制御する方法を解説します。



1.1.3 ポート方向レジスタ(PDR)

ポート方向レジスタ（PDR：ポードディレクションレジスタ）は、ポートの入力か出力を決めるレジスタです。

ピンを入力設定にすると、マイコンのピンにかかっている電圧が High(3.3V)か Low(0V)かを知ることが出来ます。出力設定にすると、マイコンのピンから High か Low の電圧を出力することが出来ます。

入出力を設定するには、RX631 ハードウェア マニュアルの 692 ページ” 21 I/O ポート”を参照してください。RX631 の初期設定では、全てのポートは入力(=0)になっています。 PDR のレジスタで入出力を変更します。ポートを出力にしたい場合は、そのビットを”1”にします。今回、ブザーが接続されているピンは、ポートの 5 番（PB5）です。したがって、PB5 の PDR レジスタの値を出力(1)に設定します

PDR レジスタは次のような構造をしております。アクセスの仕方は iodefine.h に記述されています。

Iodefine.h の一部抜粋

```
struct st_portb {
 union {
            unsigned char BYTE;
            struct {
                        unsigned char B7:1;
                        unsigned char B6:1;
                        unsigned char B5:1;
                        unsigned char B4:1;
                        unsigned char B3:1;
                        unsigned char B2:1;
                        unsigned char B1:1;
                        unsigned char B0:1;
            } BIT;
          } PDR;

          ・・・（中略）・・・
};

・・・（中略）・・・

#define      PORTB     (*(volatile struct st_portb     __evenaccess *)0x8C00B)
```

st_portb という名前で構造体(struct)が定義されています。内部に PDR という名前で共用体(union)が定義されています。共用体は構造体とほぼおなじ記述で定義出来ます。構造体と共用体の大きな違いは、共用体では定義された変数は全て先頭アドレスを共有します。すなわち、PDR 内で定義された unsigned char 型の BYTE と構造体 BIT は同じ先頭アドレスで定義されます。unsigned char は 8bit で構造体 BIT は 8bit なので、どちらでアクセスしても同じアドレスにアクセスすることになります。

次にビットフィールドについて説明します。構造体 BIT の内部の記述について解説します。

```
Unsigned char B7:1;
```

これは unsigned char の型で B7 という名前で 1bit だけ領域を確保するという記述です。したがって、構造体 BIT では B7~B0 まで unsigned char という型で 1bit ずつ、合計 8bit の領域を確保しています。ビットフィールドでは 1bit ずつアクセス出来きます。

また、st_portb は PORTB という名前でアドレス定義されています。

では、実際に PDR レジスタにアクセスしてみましょう。

```
/*********************************************************
I/Oピンの設定
        ブザー用IOポートの設定
*********************************************************/
        //ブザー関連
    PORTB.PDR.BIT.B5 = 1;
```

構造体にアクセスするには”.”でつなげてアクセスします。ビットフィールドアクセスの場合はこのようになります。またサンプルプログラムでは記述されていませんが、BYTE アクセスする場合には次のように記述をします。

```
        PORTB.PDR.BYTE = 0x20;              //B5のみを1にする
```

1.1.4 ポート出力データレジスタ(PODR)

PODR レジスタは出力設定になっているポートの出力を High(3.3V)にするか Low(0V)にするかを設定できます。PODR の値を 1 にすると出力が High、0 にすると Low になります。

21.3.2　ポート出力データレジスタ（PODR）

アドレス　PORT0.PODR 0008 C020h、PORT1.PODR 0008 C021h、PORT2.PODR 0008 C022h、PORT3.PODR 0008 C023h、PORT4.PODR 0008 C024h、PORT5.PODR 0008 C025h、PORT6.PODR 0008 C026h、PORT7.PODR 0008 C027h、PORT8.PODR 0008 C028h、PORT9.PODR 0008 C029h、PORTA.PODR 0008 C02Ah、PORTB.PODR 0008 C02Bh、PORTC.PODR 0008 C02Ch、PORTD.PODR 0008 C02Dh、PORTE.PODR 0008 C02Eh、PORTF.PODR 0008 C02Fh、PORTG.PODR 0008 C030h、PORTJ.PODR 0008 C032h

	b7	b6	b5	b4	b3	b2	b1	b0
	B7	B6	B5	B4	B3	B2	B1	B0
リセット後の値	0	0	0	0	0	0	0	0

ビット	シンボル	ビット名	機能	R/W
b0	B0	Pm0出力データ格納ビット	0：Low出力 1：High出力	R/W
b1	B1	Pm1出力データ格納ビット		R/W
b2	B2	Pm2出力データ格納ビット		R/W
b3	B3	Pm3出力データ格納ビット		R/W
b4	B4	Pm4出力データ格納ビット		R/W
b5	B5	Pm5出力データ格納ビット		R/W
b6	B6	Pm6出力データ格納ビット		R/W
b7	B7	Pm7出力データ格納ビット		R/W

m=0 ～ 9、A ～ G、J

PODR も PDR と同様に iodefine.h に定義されています。同様に PODR の PB5 にアクセスして出力を High に設定します。

```
PORTB.PODR.BIT.B5 = 1;          //B5出力をHighに設定する
```

また、PB5 の出力を Low にする場合は PODR に 0 を入力します。

```
PORTB.PODR.BIT.B5 = 0;          //B5出力をLowに設定する
```

1.1.5　for 文

次に一定秒数 High にするために for 文を使って一定時間待つというプログラムを書きます。for 文の説明は省略致します。

```
for(i = 0; i < 100*1000*1; i++);　　//一定時間待機
```

これを繰り返すことでブザーを鳴らすことが出来ます。

1.2 Step2 Motor

概要

マイコンのタイマ機能である MTU を使用してモータを回すサンプルプログラム。
クロックの設定とタイマ機能の設定について説明を行う。

Step2_Buzzer のメインプログラム

```
void main(void)
{

	/************************************************************************
	クロック設定

	************************************************************************/
	SYSTEM.PRCR.WORD = 0xa50b; //クロックソース選択の保護の解除
	SYSTEM.PLLCR.WORD = 0x0F00;		/* PLL 逓倍×16 入力1分周 (12.000MHz * 16 = 192MHz)*/
	SYSTEM.PLLCR2.BYTE = 0x00;  /* PLL ENABLE */
	SYSTEM.PLLWTCR.BYTE = 0x0F;		/* 4194304cycle(Default) */

	// ICK    : 192/2 = 96MHz		// システムクロック CPU DMAC DTC ROM RAM
	// PCLKA : 192/2 = 96MHz		// 周辺モジュールクロックA ETHERC、EDMAC、DEU
	// PCLKB : 192/4 = 48MHz		// 周辺モジュールクロックB 上記以外 PCLKB=PCLK
/*
	SYSTEM.SCKCR.BIT.FCK=0x02;  //FCLK MAX 50MHz　192/4
	SYSTEM.SCKCR.BIT.ICK=0x01;  //ICLK MAX 100MHz 192/2
	SYSTEM.SCKCR.BIT.PSTOP1=0x01;		//BCLK 出力停止
	SYSTEM.SCKCR.BIT.PSTOP0=0x01;		//SDCLK 出力停止
	SYSTEM.SCKCR.BIT.BCK=0x02;  //BCLK MAX 100MHz ICLK以下にする必要がある192/4
	SYSTEM.SCKCR.BIT.PCKA=0x01;		//PCLKA MAX 100MHz 192/2
	SYSTEM.SCKCR.BIT.PCKB=0x02;		//PCLKB MAX 50MHz 192/4
	//上記の設定では正しくclock設定ができないため下記のように一括で設定すること
*/
	//FCK1/4 ICK1/2 BCLK停止 SDCLK停止 BCK1/4 PCLKA1/2 PCLKB1/4
	SYSTEM.SCKCR.LONG = 0x21C21211;
	SYSTEM.SCKCR2.WORD = 0x0032;			/* UCLK1/4 IEBCK1/4 */
	SYSTEM.BCKCR.BYTE = 0x01;		/* BCLK = 1/2 */

	SYSTEM.SCKCR3.WORD = 0x0400;			//PLL回路選択


	/************************************************************************
	I/O設定
		LEDの設定
	************************************************************************/
	//モータ系ピン設定
	//MOT_POWER
	PORTC.PDR.BIT.B6 = 1;//motor SLEEP (STBY)
	//MOT_CWCCW
	PORTC.PDR.BIT.B5 = 1;//Rmotor PH
	PORTB.PDR.BIT.B3 = 1;//Rmotor EN
	PORTC.PDR.BIT.B4 = 1;//Lmotor PH
	PORTB.PDR.BIT.B1 = 1;//Lmotor EN

	//機能ピン設定
	MPC.PWPR.BIT.B0WI=0;
	MPC.PWPR.BIT.PFSWE=1;
	MPC.PB1PFS.BIT.PSEL=1;		//PWM R MTIOC0C
	MPC.PB3PFS.BIT.PSEL=1;		//PWM L MTIOC0A
	MPC.PWPR.BYTE=0x80;
```

```
        //MTUのイニシャライズ
        SYSTEM.PRCR.WORD = 0xA502;
        MSTP(MTU) = 0;//MTUモジュールON
        SYSTEM.PRCR.WORD = 0xA500;

        //ピンや機能設定時にはタイマストップ
        MTU.TSTR.BYTE=0;  //タイマ動作ストップ

        //左右モータ用MTU0 PWM2 時定数τ=L/R=17uH/(1.07+0.5+0.3)=110kHz
        MTU0.TCR.BIT.TPSC=0;          //PCLK/1 48MHz
        MTU0.TCR.BIT.CCLR=6;          //PWM TGRDのコンペアマッチでTCNTクリア
        MTU0.TIORH.BIT.IOA=5;         //初期出力0コンペアマッチ0出力
        MTU0.TIORL.BIT.IOC=5;         //初期出力0コンペアマッチ0出力
        MTU0.TIORL.BIT.IOD=2;         //初期出力0コンペアマッチ1出力
        MTU0.TGRA = 0;                //4以下は動作しない
        MTU0.TGRC = 0;
        MTU0.TGRD = 240-1;  //周期  200kHz(48MHz/240 = 200kHz)
        MTU0.TMDR.BIT.MD=3;           //PWM2

        PORTB.PMR.BIT.B3=1;           //右PWM
        PORTB.PMR.BIT.B1=1;           //左PWM
        MTU0.TGRA = 24;               //左(Duty比10%)
        MTU0.TGRC = 24;               //右(Duty比10%)
        MTU.TSTR.BIT.CST0 =1;

        PORTC.PODR.BIT.B6 = 1;
        PORTC.PODR.BIT.B5 = 0;        //R_PH
        PORTC.PODR.BIT.B4 = 1;        //L_PH
        while(1);
}
```

動作確認

このサンプルの CS＋「C:\HM_StarterKit-master\Sample_Programm\step2_motor」フォルダ内の「HM_StarterKit.mtpj」です。それをビルドし、RFP を使って、C:\HM_StarterKit-master\Sample_Programm\step2_motor\DefaultBuild」フォルダ内の「HM_StarterKit.mot」を書き込みます。

プログラムの解説

1.2.1　メインプログラムの動作

① クロックの設定を行う

② モータ用 IO ピンの設定

③ モータ用 MTU ピンの設定

④ PWM 出力用 MTU の設定

⑤ Duty の入力

⑥ MTU カウントスタート

1.2.2　クロックについて

本ロボットでは外部クロックに 12Mhz の発振器を使用しています。マイコン内部で 16 逓倍(クロックの周波数を 16 倍にすること)して 196MHz まで引き上げます。その後各モジュールに 2 分周か 4 分周(クロックの周波数を$\frac{1}{4}$倍すること)して 96MHz か 48MHz まで周波数を落として入力します。本サンプルでは、マイコンの動作周波数を 96MHz(最大入力周波数は 100MHz)としています。PWM 用クロックは 48MHz としています。

1.2.3　PWM 動作について

本サンプルでは MTU0(Multi Timer Unit　の 0 番ポート)を使用して PWM 出力を行っております。MTU0 の基準クロックは 48MHz に設定してあります。また、PWM 周期は以下のプログラムで 200kHz に設定してあります。

```
MTU0.TGRD = 240-1;  //周期  200kHz(48MHz/240 = 200kHz)
```

これは MTU0 のタイマを 240 カウントしたら 0 にリセットするという設定です。48MHz で 240 回カウントする。つまり 48MHz/240 で 200kHz が PWM の周期となります。また、1 周期で 240 カウントしているので Duty 比は 240 段階設定することが出来ます。

1.2.4　モータの使い方

● モータの回し方

モータを回すためにはモータドライバに対して信号を送る必要があります。モータドライバに入力された信号が　High　の間モータは動作し、Low の間モータは動作しません。モータの出力を制御するため、モータドライバの ENABLE ピンに PWM 信号を送ります。

● モータ出力の変更

モータに印加する電圧は PWM の Duty 比によって変更することが出来ます。モータに出力される電圧は次の式で表せます。

出力電圧(V) = 電源電圧(V) * MTU0.TGRA(または MTU0.TGRC) / 240

ただし、0 ≦ MTU0.TGRA ≦ 240

したがって、サンプルプログラムの　MTU0.TGRA、MTU0.TGRC　の値を変動させるとモータの出力が変動します。今回のサンプルでは 24/240=0.1 より電源電圧の 10%の電圧で駆動しています.

● モータの回転方向の変更

本ロボットに搭載されているモータドライバにはモータの回転方向を制御するフェーズ(Phase)ピンがあります。フェーズピンに入力された信号を変えるとモータの回転方向が変わります。本サンプルでは PC4,PC5 がそれぞれ左右のフェーズピンの信号(L_PH,R_PH)です。このピンの信号を切り替えるとモータの回転方向を変更できます。

回転方向を変える場合は PORTC.PODR.BIT.B5 と PORTC.PODR.BIT.B4 の値を変更してください。

1.3 Step3　光センサ

概要

AD 変換を用いて光センサの値を取得する。取得した AD 値を、シリアル通信を用いて PC へ送信する。その後、TeraTerm というシリアル通信ソフトを用いて PC 上で表示する。

Step3 光センサのメインプログラム

```
/**************************************************************************
I/O設定
        LEDの設定
**************************************************************************/

//赤外LEDのピン設定
PORTA.PDR.BIT.B3 = 1;           //PA3を出力用に設定
PORT1.PDR.BIT.B5 = 1;           //P15を出力用に設定
PORT1.PDR.BIT.B4 = 1;           //P14を出力用に設定
PORT3.PDR.BIT.B1 = 1;           //P31を出力用に設定
/**************************************************************************

その他イニシャライズ
        SCIとか
**************************************************************************/
```

```
//sciのイニシャライズ
init_sci();

/***********************************************************************************
A/DCの設定
        光センサ
***********************************************************************************/
//A/D変換用のピン設定
SYSTEM.PRCR.WORD = 0xA502;
MSTP_S12AD = 0;
SYSTEM.PRCR.WORD = 0xA500;
//PE1用設定
PORTE.PDR.BIT.B7 = 0;

//A/DポートのPMR設定
PORT4.PMR.BIT.B6=1;             //P46を周辺機器として使用
PORT4.PMR.BIT.B2=1;             //P42を周辺機器として使用
PORT4.PMR.BIT.B1=1;             //P41を周辺機器として使用
PORT4.PMR.BIT.B0=1;             //P40を周辺機器として使用
PORTE.PMR.BIT.B1=0;             //PE1を周辺機器として使用 電源 PE1
//A/DポートのPFS設定
MPC.PWPR.BYTE=0x00;             //プロテクト解除
MPC.PWPR.BYTE=0x40;             //プロテクト解除
MPC.P46PFS.BIT.ASEL=1;          //A/D SEN_FR        AN006を使用
MPC.P42PFS.BIT.ASEL=1;          //A/D SEN_R         AN002を使用
MPC.P41PFS.BIT.ASEL=1;          //A/D SEN_FR        AN001を使用
MPC.P40PFS.BIT.ASEL=1;          //A/D SEN_R         AN000を使用
MPC.PE1PFS.BIT.ASEL=1;          //A/D V_BAT         AN009を使用
MPC.PWPR.BYTE=0x80;             //プロテクト作動

//A/D変換(デフォルトでシングルモード)
//S12AD.ADCSR.BYTE = 0x0c;    //A/D変換クロックはPCLKB(48M[ha])
S12AD.ADCSR.BIT.CKS = 3;        //A/D変換のクロックをPCLKの1分周(48M[Hz])に設定
S12AD.ADCSR.BIT.ADCS = 0;       //シングルスキャンモードに設定

unsigned long i = 0;
while(1){
        S12AD.ADANS0.WORD = 0x0047; //A/D変換をAN000,1,2,6のみ許可する
        //センサ発光
        PORTA.PODR.BIT.B3 = 1;          //PA3をHighに設定
        PORT1.PODR.BIT.B5 = 1;          //P15をHighに設定
        PORT1.PODR.BIT.B4 = 1;          //P14をHighに設定
        PORT3.PODR.BIT.B1 = 1;          //P31をHighに設定
        for(i = 0; i < 100*1000/40; i++); //40で100usec
        S12AD.ADCSR.BIT.ADST = 1;       //A/D変換を開始
        while(S12AD.ADCSR.BIT.ADST);  //A/D変換が終わるまで待つ
        PORTA.PODR.BIT.B3 = 0;          //PA3をLowに設定
        PORT1.PODR.BIT.B5 = 0;          //P15をLowに設定
        PORT1.PODR.BIT.B4 = 0;          //P14をLowに設定
        PORT3.PODR.BIT.B1 = 0;          //P31をLowに設定
        //センサの値を表示
        SCI_printf("sensor_fr = %d\n\r",S12AD.ADDR6);
        SCI_printf("sensor_r = %d\n\r",S12AD.ADDR2);
        SCI_printf("sensor_fl = %d\n\r",S12AD.ADDR1);
        SCI_printf("sensor_l = %d\n\r",S12AD.ADDR0);
        //バッテリー電圧のA/D変換
        S12AD.ADANS0.WORD = 0x0200; //A/D変換をAN006のみ許可する
        S12AD.ADCSR.BIT.ADST = 1;       //A/D変換を開始
        while(S12AD.ADCSR.BIT.ADST);  //A/D変換が終わるまで待つ

        SCI_printf("V_BAT = %d\n\r",S12AD.ADDR9);
        for(i = 0; i < 100*1000*100; i++);


        //画面クリアシーケンス
        SCI_printf("\x1b[2J");                          //クリアスクリーン[CLS]
        SCI_printf("\x1b[0;0H");                        //カーソルを0,0に移動
}
```

動作確認

このサンプルの CS＋「C:\ HM_StarterKit-master\Sample_Programm\step3_serial_light」フォルダ内の「HM_StarterKit.mtpj」です。それをビルドし、RFP を使って、C:\ HM_StarterKit-master\Sample_Programm\step3_serial_light \DefaultBuild」フォルダ内の「HM_StarterKit.mot」を書き込みます。

プログラムの解説

1.3.1 メインプログラムの動作

① クロックの設定を行う(メイン文では省略)

② 光センサ用 IO ピンの設定（LED の出力）

③ SCI 機能のイニシャライズ

④ 12bitA/D 変換機能の設定

⑤ LED 発光

⑥ A/D 変換開始

⑦ LED 消灯

⑧ A/D 変換結果の出力

1.3.2 SCI について

本サンプルから SCI(Serial Communication Interface:シリアルコミュニケーションインターフェース)という機能を用いて、PC との通信を行います。シリアル通信に関してはライブラリを用意したので本マニュアルでは使用方法のみを説明いたします。本サンプルではすでに説明する手順を行っています。SCI 機能を使用する場合は、同プロジェクト内にある sci.c 及び sci.h をインポートしてください。その後使用するファイルで sci.h をインクルードします。

```
#include "sci.h"
```

次に SCI 機能の初期化を行う関数を呼び出します。

```
init_sci();
```

最後に出力用関数 SCI_printf を呼び出すと、PC へシリアル出力をします。

```
SCI_printf("Test _NUM= %d\n\r",10);
```

引数や書式は一般的な printf と同じになっておりますので詳細はそちらを参照ください。

1.3.3 PC での表示について

シリアル通信ソフトである TeraTarm を利用します。まず PC と書き込み機、マウスを接続します。その後 TeraTarm を起動し、書き込み機の COM ポートと接続します。接続できない場合は書き込み機に使用している FT232RL 用のドライバがインストールされていない場合があります。その場合は FTDI Chip(http://www.ftdichip.com/Drivers/VCP.htm)からドライバをダウンロードしてインストールしてください。TeraTarm と書き込み機の接続が出来たら、マウスの電源を入ると通信が開始されます。TeraTarm と書き込み機を接続している間は RFP でマイコンに書き込みが出来ません。マイコンにプログラムを書き込む場合は、TeraTarm を落とすか、書き込み機との通信を切断してください。

1.3.4 AD 変換について

本サンプルで使用する ADC(Analog to Digital Converter：アナログデジタル変換器)は 12bitADC です。ADC についての説明は本サンプルでは省略させていただきます。センサの出力が 12bit、4096(2^{12} = 4096)分割で出力されます。リファレンス電圧である 3.3V と 0V が基準となるため出力された値は次の式で電圧に変換できます。

入力された電圧(V)　=　3.3(V) * AD 値　/　4096

AD 変換されたデータは S12AD.ADDRn レジスタに格納されます。n は使用している AD ポート番号を入力します。本サンプルでは各ポートの接続先は以下のようになっています。

アクセス先一覧

AD ポート	機能	アクセス先
0	左光センサ	S12AD.ADDR0
1	左前光センサ	S12AD.ADDR1
2	右光センサ	S12AD.ADDR2
6	右前光センサ	S12AD.ADDR6
9	電源電圧	S12AD.ADDR9

電源電圧はリファレンス電圧より高い(レギュレータで 3.3V に落とす前の電圧を計測している)ため、1kΩの抵抗で分圧しています。
よって、AD 値を二倍した値が電源電圧となるので注意して下さい。

1.4 Step4 I2C と IO

概要

I2C を用いて IO エキスパンダと通信することで、4 つの LED と 3 つのスイッチを制御する。このサンプルプログラムでは、4 つの LED を点灯させ、スイッチが押されたかどうかを TeraTearm で確認する。

Step4 i2c_io のメイン文

```
void main(void)
{
        /************************************************************************
        クロック設定

        ************************************************************************/
                (中略)
        /************************************************************************
        その他外部関数のイニシャライズ
        ************************************************************************/
        //sciのイニシャライズ
        init_sci();
        //I2Cのイニシャライズ
        init_I2C();

        //Main Program
        unsigned long i = 0;
        IOex_SETTING();

        while(1){
                IOex_LED(0xF0);
                IOex_SEITCH();
```

```
                for(i = 0; i < 100*1000*100; i++);
                //画面クリアシーケンス
                SCI_printf("\x1b[2J");                    //クリアスクリーン[CLS]
                SCI_printf("\x1b[0;0H");                        //カーソルを0,0に移動
        }
}
```

動作確認

このサンプルの CS＋「C:\HM_StarterKit-master\Sample_Programm\step4_i2c_io」フォルダ内の「HM_StarterKit.mtpj」です。それをビルドし、RFP を使って、C:\HM_StarterKit-master\Sample_Programm\step4_i2c_io\DefaultBuild」フォルダ内の「HM_StarterKit.mot」を書き込みます。

プログラムの解説

1.4.1 メインプログラムの動作

① クロックの設定を行う(省略)

② SCI 機能のイニシャライズ

③ I2C 機能の設定

④ IO エキスパンダの設定

⑤ LED 点灯

⑥ スイッチ入力の取得

⑦ スイッチ入力の表示

1.4.2 IO エキスパンダについて

本ロボットはピンアサインの関係上、ユーザーインターフェース用 IO ピンの数が足りません。そのため、IO エキスパンダという IC を用いて IO ピンの数を拡張しています。今回使用している IO エキスパンダは NXP 社の PCA9557 という IC を使用しています。

この IC は 8bit の IO ピンを制御できます。初めに拡張したピンを入力で使うか、出力で使うかの設定を行います。その後、設定したピンの入力状態を読み出したり、出力状態を変更したりできます。各ピンにアクセスするときは 8bit でアクセスします。

1.4.3 I2C について

デジタル通信方式の一つで、SCL、SDA の二本の信号線でデータをやりとりします。SCL がクロック、SDA がデータ信号です。SCL と SDA を同期させて通信します。詳しい説明は本マニュアルでは省略させていただきます。I2C を使用する方法のみ説明します。

また、ここで説明する使用方法は、サンプルプログラムにおいて既に行われています。

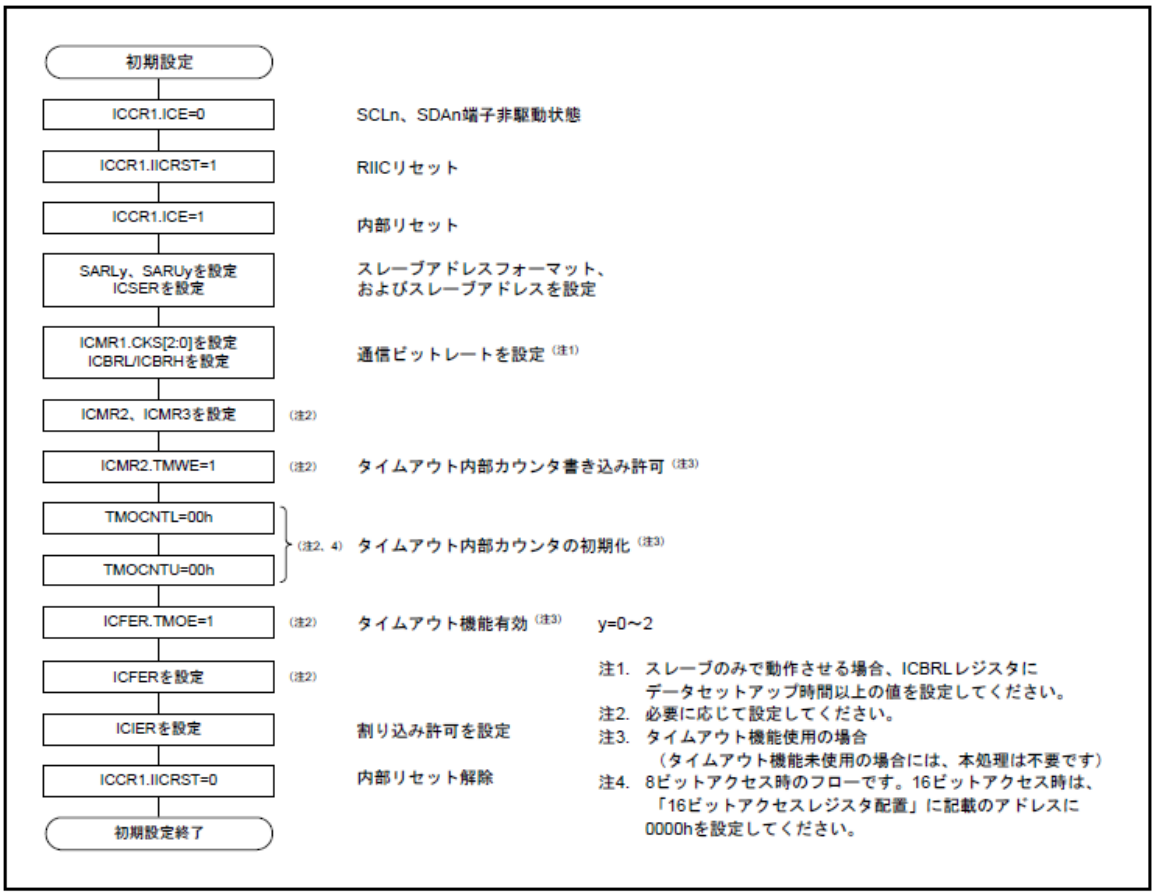
まず、i2c.c をプロジェクトにインポートします。次に使用したいソースファイル内で次の関数を定義します。

```
extern void init_I2C(void);
extern void IOex_SEITCH(void);
extern void IOex_SETTING(void);
extern void IOex_LED(short led_num);
```

次に I2C 機能を初期化する関数を呼び出します。

```
init_I2C();
```

この関数の中身に関して，RX631 ハードウェアマニュアルの 1519 ページ「36.3.2 初期設定」より以下のフローチャートの通りとなります．



IO エキスパンダにアクセスして初期設定を行います。

```
IOex_SETTING();
```

また，内部はこのように記述されています。

```
void IOex_SETTING(void){
//ポーリングによる設定
//*
I2C_START();//Startbit 発生
I2C_PUT((0x18<<1));//アドレス  write
I2C_PUT(0x03);//config
I2C_PUT(0xF0);//sleep 解除
I2C_STOP();
}
```

I2C_START();によって I2C 通信が行われる合図，すなわちスタートビットが送信されます．さらにデータの宛先を示すスレーブアドレス，及び Read or Write の指令を送信します．ここで，回路図を参照すると IO エキスパンダの A0～A2 ピンはすべて 0V(LOW)となってなので I2C アドレスは 0x18 となります。

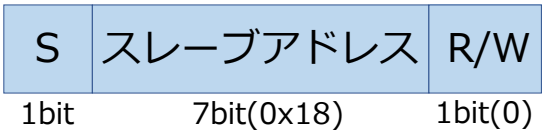
ここまでのフォーマットを図に示すと以下のようになります．

Table 2. Address Reference

INPUTS			I²C BUS SLAVE ADDRESS
A2	A1	A0	
L	L	L	24 (decimal), 18 (hexadecimal)
L	L	H	25 (decimal), 19 (hexadecimal)
L	H	L	26 (decimal), 1A (hexadecimal)
L	H	H	27 (decimal), 1B (hexadecimal)
H	L	L	28 (decimal), 1C (hexadecimal)
H	L	H	29 (decimal), 1D (hexadecimal)
H	H	L	30 (decimal), 1E (hexadecimal)
H	H	H	31 (decimal), 1F (hexadecimal)

アドレスは0x18

その後任意のデータを送信します．0x03 は IO エキスパンダの設定モードに入ることを示し，0xF0 は IO ポートの上位 4bit を論理反転させることを意味しています．(回路上では SW を押すと LOW が入力されるが，ソフトウェア上では HIGH 入力として扱える．)



これで初期設定は終了です。LED を点灯、消灯させたい場合は次の関数を呼び出します。

```
IOex_LED(0xF0);
```

この関数の引数の下四桁が LED の点灯、消灯に関係しています。入力する数値に関しては下記表を参照してください。LED は左から順番に D3,D4,D5,D6(シルクの表記)です。

点灯＝○,消灯＝×

入力	D4	D3	D6	D5
0xF0	○	○	○	○
0xF1	×	○	○	○
0xF2	○	×	○	○
0xF3	×	×	○	○
0xF4	○	○	×	○
0xF5	×	○	×	○
0xF6	○	×	×	○
0xF7	×	×	×	○
0xF8	○	○	○	×
0xF9	×	○	○	×
0xFA	○	×	○	×
0xFB	×	×	○	×
0xFC	○	○	×	×
0xFD	×	○	×	×
0xFE	○	×	×	×
0xFF	×	×	×	×

この関数の中身は以下のようになっています．基本的に設定のときと同じ要領です．I2C_PUT(0x01);は出力モードを示し，引数の値を送信するとその値が IO ポートに出力されます．今回の回路では HIGH 出力で LED が消灯します．

```
void IOex_LED(short led_num){
//ポーリングによる設定
//*
I2C_START();//Startbit 発生
I2C_PUT((0x18<<1));//アドレス  write
I2C_PUT(0x01);//OUTPUT
I2C_PUT(0x00 ¦ led_num);//sleep 解除
I2C_STOP();
}
```

スイッチが押されたかを判断するには次の関数を呼び出してください。

```
IOex_SWITCH ();
```

この関数を呼び出すとスイッチが押されたかどうかを TeraTarm 上に表示します。確認する際は TeraTarm と接続してください。

1.5 Step5 SPI とエンコーダ

概要

SPI を用いてエンコーダと通信をするサンプルプログラム。TeraTearm を使って左右のエンコーダの値を確認する。

Step5 spi_enc メイン文

```
void main(void)
{
  /**********************************************************************************
  クロック設定

  **********************************************************************************/
                (中略)
  /**********************************************************************************
  I/O設定
              LEDの設定
  **********************************************************************************/
  //LEDのピン設定

  //sciのイニシャライズ
  init_sci();
  //spiのイニシャライズ
  init_spi_enc();

  unsigned long i = 0;
  int data = 0;
  for(i = 0; i < 100*1000*10; i++);
            //Encoderの初期化
  RSPI0.SPCMD0.BIT.SSLA = 0x02; //SSL信号アサート設定(SSL2を使う)
  preprocess_spi_enc(0x7E40);
  for(i = 0; i < 100*1000*10; i++);
  preprocess_spi_enc(0x5040);
  for(i = 0; i < 100*1000*10; i++);

  //Encoderの初期化
```

```
    RSPI0.SPCMD0.BIT.SSLA = 0x00; //SSL信号アサート設定(SSL0を使う)
    preprocess_spi_enc(0x7E40);
    for(i = 0; i < 100*1000*10; i++);
    preprocess_spi_enc(0x5040);
    for(i = 0; i < 100*1000*10; i++);
    while(1){

                RSPI0.SPCMD0.BIT.SSLA = 0x02;           //SSL信号アサート設定(SSL2を使う)
                preprocess_spi_enc(0xFFFF);                     //Read Angle
                data = Get_enc_r_data();                        //エンコーダ値取得
                SCI_printf("R_data_H,%d\n\r",data >> 8);    //上位ビット表示
                SCI_printf("R_data_L,%d\n\r",data & 0x00FF);        //下位ビット表示

                RSPI0.SPCMD0.BIT.SSLA =     0x00;       //SSL信号アサート設定(SSL0を使う)
                preprocess_spi_enc(0xFFFF);                     //Read Angle
                data = Get_enc_r_data();                        //エンコーダ値取得
                SCI_printf("L_data_H,%d\n\r",data >> 8);    //上位ビット表示
                SCI_printf("L_data_L,%d\n\r",data & 0x00FF);        //下位ビット表示

                for(i = 0; i < 100*1000*1; i++);
                //画面クリアシーケンス
                SCI_printf("\x1b[2J");                  //クリアスクリーン[CLS]
                SCI_printf("\x1b[0;0H");                        //カーソルを0,0に移動
    }
}
```

動作確認

このサンプルの CS＋「C:\ HM_StarterKit-master\Sample_Programm\step5_spi_encoder」フォルダ内の「HM_StarterKit.mtpj」です。それをビルドし、RFP を使って、C:\ HM_StarterKit-master\Sample_Programm\step5_spi_encode\DefaultBuild」フォルダ内の「HM_StarterKit.mot」を書き込みます。

プログラムの解説

1.5.1 メインプログラムの動作

① クロックの設定を行う(省略)

② SCI 機能のイニシャライズ

③ SPI 機能の設定

④ 右のエンコーダの初期設定

⑤ 左のエンコーダの初期設定

⑥ 右のエンコーダの値の取得

⑦ 右のエンコーダの値の表示

⑧ 左のエンコーダの値の取得

⑨ 左のエンコーダの値の表示

1.5.2　エンコーダについて

回転角度を出力するセンサです。本ロボットに搭載されているのは磁気式エンコーダです。搭載されている IC は MA700GQ というエンコーダです。エンコーダには IC によって出力方式が異なります。パルス信号を出力したり、デジタル信号で出力したりします。今回使用している IC は SPI というデジタル通信で車軸位置を出力します。360 度を 12bit、4096 分割した出力が返ってきます。

1.5.3　SPI について

デジタル通信方式の一つで、MISO(マスターインプットスレーブアウトプット)、MOSI(マスターアウトプットスレーブインプット)、SCK(シリアルクロック)、SS(スレーブセレクト)の 4 本の信号線でデータをやりとりします。SPI 通信の詳しい説明は本マニュアルでは省略させていただきます。SPI を使用する方法のみ説明します。また、ここで説明する使用方法は、サンプルプログラムにおいて既に行われています。

まず、spi.c,spi.h をプロジェクトにインポートします。次に SPI 機能の初期設定を行う関数を呼び出します。

```
init_spi_enc();
```

左右エンコーダの初期設定を行います。左右のエンコーダは同じ SPI 通信ポートに接続されているので、スレーブセレクトピンを設定してどちらのエンコーダ IC と通信するかを選択する必要があります。スレーブセレクトの設定は次のレジスタの値を書き換えて行います。

```
RSPI0.SPCMD0.BIT.SSLA = 0x02; もしくは　RSPI0.SPCMD0.BIT.SSLA = 0x00;
```

スレーブセレクトの設定を行った後に、それぞれのエンコーダの初期化処理を行います。この初期化処理を行わないと、エンコーダの値が変わる可能性があるので必ず初期化を行ってください。

```
 preprocess_spi_enc(0x7E40);
for(i = 0; i < 100*1000*10; i++);
preprocess_spi_enc(0x5040);
for(i = 0; i < 100*1000*10; i++);
```

初期化処理が終わったら、エンコーダから車軸の角度情報を読み込みます。角度情報を読み込むときは次のように記述します。

```
preprocess_spi_enc(0xFFFF);                    //Read Angle
data = Get_enc_r_data();                    //エンコーダ値取得
```

この二行を処理すると、data 変数の中に 12it の角度情報が入力されます。それを SCI_printf で出力し、値を確認します。

1.6 Step6 SPI とジャイロ

概要

SPI を用いてジャイロセンサと通信をするサンプルプログラム。TeraTearm を使ってジャイロセンサの値を確認する。

Step6 spi_gyro メイン文

```
void main(void)
{
        /**********************************************************************************
        クロック設定

        **********************************************************************************/
                (中略)

        /**********************************************************************************
        SCI_SPIの設定        **********************************************************************************/

        //sciのイニシャライズ
        init_sci();
```

```
        //spiのイニシャライズ
        init_spi_gyro();

        unsigned long i = 0;
        long data = 0;
        for(i = 0; i < 100*1000*10; i++);
        preprocess_spi_gyro(0x062100);
        data = read_gyro_data();
        for(i = 0; i < 100*1000*10; i++);

        while(1){
                SCI_printf("data_H,%d\n\r",(data & 0xFF0000) >> 16);
                SCI_printf("data_M,%d\n\r",(data & 0x00FF00) >> 8);
                SCI_printf("data_L,%d\n\r",(data & 0x00FF));

                for(i = 0; i < 100*1000*10; i++);
                for(i = 0; i < 100*1000*100; i++);
                //画面クリアシーケンス
                SCI_printf("\x1b[2J");                          //クリアスクリーン[CLS]
                SCI_printf("\x1b[0;0H");                        //カーソルを0,0に移動
                preprocess_spi_gyro(0xB70000);
                data = read_gyro_data();
        }
}
```

動作確認

このサンプルの CS＋「C:\ HM_StarterKit-master\Sample_Programm\step6_spi_gyro」フォルダ内の「HM_StarterKit.mtpj」です。それをビルドし、RFP を使って、C:\ HM_StarterKit-master\Sample_Programm\step6_spi_gyro\DefaultBuild」フォルダ内の「HM_StarterKit.mot」を書き込みます。

プログラムの解説

1.6.1 メインプログラムの動作

① クロックの設定を行う(省略)

② SCI 機能のイニシャライズ

③ SPI 機能の設定

④ ジャイロセンサの初期設定

⑤ ジャイロセンサの値取得

⑥ ジャイロセンサの値表示

1.7 Step7 迷路走行

概要

このプログラムは今までの step1～6 までの機能を統合し、迷路を走行出来るようにしたサンプルソフトです。走行アルゴリズムとして、北優先の足立法を実装。最短経路は歩数最短を計算しています。また、走行ログも一部取得をしており、ログ出力も可能です。

動作確認

このサンプルの CS+プロジェクトは「C:\ HM_StarterKit-master\Sample_Programm\step7_ maze」フォルダ内の「HM_StarterKit.mtpj」です。それをビルドし、RFP を使って、「C:\ HM_StarterKit-master\Sample_Programm\step7_ maze \DefaultBuild」フォルダ内の「HM_StarterKit.mot」を書き込みます。

プログラムの解説

1.7.1　メインプログラムの動作

① 各機能の初期設定を行う

② モード選択開始

③ モード決定

④ 決定されたモードのプログラムを実行

⑤ へ戻る

1.7.2　モード説明

本サンプルプログラムに入っているモードの解説をします。

メインモード

1. 左手法
2. 足立法探索(マップ保存)
3. 最短走行
4. 空きモード
5. 空きモード
6. 空きモード
7. 空きモード
8. 空きモード
9. 空きモード
10. 空きモード
11. 空きモード
12. 空きモード
13. 空きモード
14. 空きモード
15. 調整モードへ移行

モードの概要

●左手法

- 左手法を用いて迷路を走行します。このプログラムではマップは E2 フラッシュに保存されません。

●足立法探索

- 足立法という探索アルゴリズムで迷路を走行します。走行時に迷路をマッピングし、ゴールした時と、スタートに帰ってきたときの二回、迷路情報をフラッシュに保存します。

●最短走行

- 足立法探索によってマッピングされた迷路情報をもとに最短走行を行います。

●調整モードへ移行

- 車体調整を行うための調整モードへ移行します。

調整モード

1. 車体状態の出力
2. 一区画前進(ログ保存)
3. 90 度右に旋回(ログ保存)
4. 空きモード
5. 空きモード
6. E2 フラッシュへ保存されているマップの表示
7. ログ出力

モードの概要

●車体状態の出力

- 各種センサ値をシリアル通信ソフトに出力します。光センサの調整や、各種センサの値の確認が出来ます。

●一区画前進(ログ保存)

- 一区画(90mm)前進し停止します。この時、走行距離や、車体速度などの走行ログを RAM に保存します。電源を切らずに 7 番の「ログ出力」を実行すると、ログが出力されます。

●90 度右に旋回(ログ保存)

- 90 度右に超信地旋回をします。この時、走行距離や、車体速度などの走行ログを RAM に保存します。電源を切らずに 7 番の「ログ出力」を実行すると、ログが出力されます。

●E2 フラッシュへ保存されているマップの表示

- 足立法探索によって保存されたマップをシリアル通信ソフトに出力します。赤色が未探索の壁、白色が探索済みの壁、空白が探索済みの壁のない場所となります。

●ログの出力

- 走行ログを出力します。(ログ保存)と書かれたプログラムを実行するとログが保存されます。

各種ファイルの説明

init.c：初期設定全般が入ったファイル

動作概要

各モジュールやセンサの初期化処理を行うファイル。

電源投入時に 1 度呼び出される。

動作内容

クロックの初期設定

CMT（コンペアマッチタイマ）の初期設定

IO の初期設定（光センサ用赤外 LED とブザ）

ADC の初期設定（光センサ用 AD ポートと電源監視用ポートの設定）

MTU2 の初期設定（左右モータの PWM 出力設定）

構造体、変数の初期化

迷路情報の初期化

SPI の初期化処理呼び出し

I2C の初期化処理呼び出し

ジャイロ、エンコーダの初期化処理

intrrupt.c：割り込み処理全般が入ったファイル

動作概要

CMT による割り込み処理を記述している。

それぞれ機能ごとに CMT を用いている。主に速度制御、姿勢制御、センサ割り込みが存在する。

CMT は 3 つ使用している。

動作内容

走行制御に関する割り込み

CMT0

車体の目標速度生成

車体の目標角速度生成

車体の目標角度生成

車体の姿勢制御（フィードバック制御）

モータ出力

タイマカウント

CMT1：AD 変換と走行ログ取得に関する割り込み

光センサ値取得

バッテリ電圧監視

走行ログ取得

CMT2：デジタル通信系割り込み

SPI 通信による左右エンコーダ値取得

車体速度計算

SPI 通信によるジャイロセンサ値取得

車体角度計算

車体角速度計算

parameters.h：各種パラメータを設定ファイル

概要

走行用パラメータやセンサ用パラメータ、ゴール座標の設定などをするためのプログラムです。走行速度を変えたいときやゴール座標を変えたい場合はこのプログラムを見ましょう。

search.c：迷路探索アルゴリズムの入ったファイル

概要

迷路を走破するための肝になっているプログラムです。歩数マップ生成関数、壁情報の保存、優先度の取得、足立法、左手法の関数などが入っています。詳しくは、別紙の”マイクロマウス HM_StarterKit　パート 1「基礎知識編」の 3.迷路解析の手法と考え方”を見てください。

fast.c：最短走行用ファイル

概要

最短走行用関数 fast_run が入ったファイルです。詳しくは、別紙の”マイクロマウス HM_StarterKit　パート 1「基礎知識編」の 3.7 最短経路の求め方”を見てください。

HM_StarterKit.c：main 関数の入っているファイルです

概要

モード 1 に左手法、モード 2 に足立法でゴールを目指し、帰りにまた足立法でスタートまで帰り探索をするプログラムになっています。モード 3 には最短走行プログラムになっています。

portdef.h：アプリケーション層とデバイスドライバ層をつなぐヘッダファイル

動作概要

各マイコンや回路固有のレジスタをアプリケーション層で使用するための中間層。

センサ LED やモータのイネーブル制御用 GPIO などの定義をしています。

定義内容

センサ LED 制御用マクロの定義

モータイネーブル制御用マクロの定義

モータの回転方向制御用マクロの定義

モータの出力制御用マクロの定義

Parameter.h：車体のパラメータ等を記述するヘッダファイル

定義概要

車体の物理的なパラメータや、センサリファレンス値を記述するファイル。

マウスの調整では主にこのファイルを編集する。

Static_parameter.h：静的な値を記述するヘッダファイル

定義概要

静的（変動しない）値を記述してあるヘッダファイル。主に区画の距離や迷路の外形などが定義されている。

Misc.c：wait 関数用ファイル

動作概要

Wait 関数が記述されたファイル。

Wait 関数を別ファイルで exclude することで、指定した時間待機する wait_ms が使えるようになる。

Spi.c：SPI 用プログラム

動作概要

SPI を使ったセンサ系のプログラム。初期化シーケンスや通信シーケンスはここに記述されている。

ジャイロセンサと磁気式エンコーダが SPI で通信している。

run.c：走行制御用プログラム

動作概要

直進、旋回の指令を行う。制御自体は割り込みで行うが、割り込みで使用する変数を計算したり、初期値を代入したり、制御系の切り替えなどを行う。

動作内容

Stragith 関数

概要

直進の指令を出す関数

引数

距離[mm]，加速度[m/s^2]，最高速度[m/s]，最終速度[m/s]

内容

直進の距離と加速度、現在速度等から台形加減速を行う。

Turn 関数

概要

超信地旋回の指令を出す関数

引数

回転数[degree],角加速度[rad/ss],最高角速度[rad/s],回転方向

内容

角速度に対して台形加減速を行い、超信地旋回する関数。

Grab_var.h：グローバル変数を定義するヘッダー

概要

詳細はファイル内定義を参照

Interface.c：ユーザーインターフェース用関数を定義するファイル

概要

ブザーや LED、スイッチなどの UI 関連を定義するファイル。

Dataflash.c：E2 フラッシュ関連のファイル

概要

E2 フラッシュにマップの書き込みを行ったり、E2 フラッシュからマップの展開を行うファイル。

詳細

E2 フラッシュとは RX631 に搭載されている、内臓フラッシュメモリです。

データフラッシュの詳細は、RX631 ハードウェアマニュアルの”47. フラッシュメモリ”を見てください。RX631 のフラッシュメモリには ROM と E2 データフラシュの二種類あります。ROM の方にはマイクロマウスのプログラムを書き込んで使用しています。E2 データフラッシュは、電源を OFF にしてもデータを保存したいものを書き込む領域です。この E2 データフラッシュ(以降データフラッシュと省略)の書き込みは 2 バイト、イレース(消去)は 32 バイトと小さいサイズで書き込み、イレースが可能となっています。迷路の map データは最大 32x32=1024 バイトの領域が必要です。このデータをすべてデータフラッシュに書き込むには、書き込みでは 512 回(1024/2byte)、イレースでは 32 回(1024/32byte)必要です。データを書き換えるには、一度イレースしてから書き込む必要があります。

USB ブートモード(USB でプログラムを書き換えるモード)では、データフラッシュに書き込んだデータをプロテクトすることができません。USB ブートモードにするとデータフラッシュ内がすべてイレースされます。データフラッシュにあるデータをプロテクトするには、UART によるブートモードにする必要があります。

データフラッシュに保存したマップ情報は最短走行時(モード 3)の最初に map_copy()関数でデータフラッシュから RAM にあるマップ情報に上書きされます。

迷路を走らせるには

HM StarterKit のプログラムには左手法か足立法でゴールする二種類の関数が用意されています。このうち足立法でゴールする場合は main 関数のモード 2 に書いてあるように

1. スタート地点の座標を入力(座標の初期化)
2. スタート地点で自分の向いている方向を入力(方角の初期化)
3. 足立法関数の呼び出し

この手順で設定して頂ければマウスは足立法で迷路のゴールを目指してくれます。

上記のプログラムではゴールからの帰りにも探索をするように探索用関数 search_adaci()で目指す座標をスタートの 0,0 にすることでゴールからスタートまで探索しながら帰ってきます。

また最短走行の場合は最短走行関数 fast_run()を使用して、加速度を与える以外は探索時と同じ手順で最短走行ができます。

これで全ての解説を終わります。マイクロマウスは迷路をゴールしてからが始まりです。

これからはアルゴリズムを改良したり、より速く、安定して走れるようにするなど自分なりに工夫してマイクロマウスを楽しんでください。できることならば、この HM_StarterKit で学んだことを生かし新しく自分で設計、製作したマイクロマウスで大会に参加するなどして、楽しんで頂けたら幸いです。

1.9 Step9：迷路情報、センサ値を記憶しよう

プログラムの解説

モード 2 の足立法で探索し、ゴールに入ると LED が点滅すると思います。ゴールに入る goal_appeal()関数が呼ばれ、その関数内で LED の点滅をさせています。その関数内で”map_write();”の関数を呼び出し、探索したマップデータをデータフラッシュメモリに書く作業をしています。

DataFlash.c から抜粋

```
void map_write(void)
{
	short i;
	unsigned short *map_add;
	map_add = (unsigned short *)&wall;

	//DataFlashイレース
	for(i=0;i<32;i++){
		erase((unsigned short *)(MAP_ADD+i*32));
	}
	//マップデータをDataFlashに書き込む
	for(i=0;i<512;i++){
		write_eeflash((unsigned short *)(MAP_ADD+i*2),*map_add);
		map_add++;
	}
}
```

DataFlash.c から抜粋

```
void map_copy(void)
{
	short i;
	unsigned short *map_add;
	map_add = (unsigned short *)&wall;
	//マップデータをRAMにコピー
	for(i=0;i<512;i++){
		*map_add = *(unsigned short *)(MAP_ADD+i*2);
		map_add++;
	}
}
```

2. 故障かな？　と思ったら

以下に主に起こりうる故障についての解決策をあげます。どうしても直らない、解決しない場合は弊社までお問い合わせください。

●電源スイッチを入れても何も反応しない	
電池をコネクタにきちんと接続していますか？	意外と忘れてしまうときがあります。落ち着いてもう一度接続してみましょう。
電池のコネクタがとれていませんか？	何度もコネクタの抜き差しを行うと取れてしまう場合があります。
電池の容量は充分にありますか？	充電をしなおしてみてください。
●センサの調子がおかしい	
プログラムにミスはないですか？	絶対に入らないような論理式を立てていたりしていないか、無限ループに入っていないかなどを確認しましょう。
閾値が高すぎたり、低すぎたりしていませんか？	まずはシリアル通信でセンサの情報を調べてみましょう。
●まっすぐ進まない、動かない	
基板を支えるねじが緩んでいませんか？	ねじが緩んでいると正確に力が伝わらないことがあります。
ゴムタイヤがずれていませんか？	ゴムリングが車輪からずれてしまうことがあります。確認してみてください。
姿勢制御や走行距離のパラメータは適正ですか？	さまざまな要因によりずれが生じてしまうことがあります。パラメータを調整してみましょう。
プログラムにミスはないですか？	コンパイルが成功していたとしても、動作しない場合があります。もう一度プログラムを確認してみましょう。
テフロンシートの貼りつけ位置は合っていますか？	土台の一番下に張っているテフロンシートの位置を貼り直して調整してみてください。

4. 著作権について

本取扱説明書で紹介、または記載されている会社名、製品名は、各社の登録商標または商標です。

本取扱説明書に掲載している文書、写真、イラストなどの著作物は、日本の著作権法及び国際条約により、著作権の保護を受けています。

5. ソフトウェアについて

●ルネサスエレクトロニクス製ソフトウェアについて

ダウンロードしたルネサスエレクトロニクス製ソフトウェアは、サポート対象製品ではありません。サポートは一切行われませんので、あらかじめご了承ください。

●すべての収録ファイルについて

ダウンロードしたすべての収録ファイル対して，その使用にあたって生じたトラブル等は、ルネサスエレクトロニクス(株)、および(株)アールティは一切の責任を負いません。

インターネット等の公共ネットワーク、構内ネットワーク等へのアップロードなどは、ルネサスエレクトロニクス(株)および(株)アールティの許可無く行うことはできません。

6. 改訂履歴

発行日	ページ	改訂内容
2018/7	-	新規発行

製造元

株式会社アールティ

〒101-0021 東京都千代田区外神田 3-2-13 山口ビル 3F TEL 03-6666-2566

URL https://www.rt-net.jp/

製品に関するお問い合わせ

本製品に関するお問い合わせは、下記までお願いします。

お問い合わせは電子メールにて受け付けております。

E-mail: support@rt-net.jp