

2004.2.12

情報工学ゼミナール研究

μITRON 仕様に基づく組み込みリアルタイム OS の研究

情報工学科 藤川雅男 (r00h0390)
多賀創 (r00h0211)

目次

0. 予定の変更

1. 環境構築

1.1. Cygwin 環境構築

1.2. Cygwin 開発環境(FreeBSDPress NO.14 を参照)から sample を実行するまで

2. 状態遷移を観察するための簡単なプログラム例

3. マイクロマウスへの ITRON の実装

4. 割り込みハンドラ

4.1. TOPPERS 割り込みの登録の仕方

5. 他の CPU への移植

5.1. 開発環境の構築

6. おわりに

7. 参考文献

0 . 予定の変更

当初「 μ ITRON 仕様に基づく組込みリアルタイム OS の開発」ということで作業を進めてきたが、時間の都合上それができなくなった。そのため、後期からは研究テーマを変更し、 μ ITRON 仕様 OS「TOPPERS」を使用する際の環境構築・実際に OS を使った場合のプログラムの書き方といったことについてまとめるのに専念することにした。

1 . 環境構築

ここでは Windows2000 マシンに cygwin をインストールし、その上の gcc で TOPPERS1.3 を利用したプログラムをコンパイルすることを想定している。

1 . 1 . Cygwin 環境構築 (Administrator 権限が必要)

1 . 1 . 1 . Cygwin を入れる。

- Install for を All Users にする
- Default Text File Type を Unix にする
- Select Packages で All の横の Default をクリックし Install にする(時間がかかり応答なしになることもある)

1 . 1 . 2 . 準備

以下のソースファイルを用意する。

アセンブラ・リンカ

binutils-2.10.tar.gz

<http://sources.redhat.com/binutils/>

<ftp://ftp.ring.gr.jp/pub/GNU/binutils>

C コンパイラ

gcc-2.95.2.tar.gz

<ftp://ftp.ring.gr.jp/pub/GNU/gcc/>

標準 C ライブラリ

newlib-1.8.2.tar.gz

<http://sources.redhat.com/pub/newlib/>

1 . 1 . 3 . binutils の作成

ソースを解凍

```
# cd /usr/local/src
```

```
# tar xzvf binutils-2.10.tar.gz
```

作業用ディレクトリ作成

```
# mkdir binutils-2.10/work
```

```
# cd binutils-2.10/work
```

コンフィグレーション

```
# ../configure --target=h8300-hms-coff --prefix=/usr/local/h8
```

コンパイル・インストール

```
# make
```

```
# make install
```

ここでインストールされたツールを実行できるようにするためにパスを通す。

```
# PATH=$PATH:/usr/local/h8/bin
```

1.1.4. gcc の作成

newlib のヘッダーファイルを使用するので解凍する

```
# cd /usr/local/src
# tar xzvf newlib-1.8.2.tar.gz
```

gcc をコンパイル

```
# tar xzvf gcc-2.95.2.tar.gz
# mkdir gcc-2.95.2/work
# cd gcc-2.95.2/work
# ../configure --target=h8300-hms-coff --prefix=/usr/local/h8
--with-newlib --with-headers=/usr/local/src/newlib-1.8.2/newlib/libc/include
# make LANGUAGES="c"
# make install LANGUAGES="c"
```

1.1.5. newlib の作成

```
# cd /usr/local/src/newlib-1.8.2
# mkdir work
# cd work
# ../configure --target=h8300-hms-coff --prefix=/usr/local/h8
# make
# make install
```

以上で以下の H8 プロセッサ用のツールが使用できるようになる。

h8300-hms-coff-as	アセンブラ
h8300-hms-coff-ld	リンカ
h8300-hms-coff-gcc	C コンパイラ
h8300-hms-coff-objcopy	オブジェクトファイル変換ツール
h8300-hms-coff-objdump	逆アセンブラ
h8300-hms-coff-nm	シンボル名表示
h8300-hms-coff-size	プログラムサイズ表示

短い名前で使用できるようにシンボリックリンクを張る。

```
# cd /usr/local/bin
# ln -s /usr/local/h8/bin/h8300-hms-coff-as h8-as
# ln -s /usr/local/h8/bin/h8300-hms-coff-ld h8-ld
# ln -s /usr/local/h8/bin/h8300-hms-coff-gcc h8-gcc
# ln -s /usr/local/h8/bin/h8300-hms-coff-objcopy h8-objcopy
# ln -s /usr/local/h8/bin/h8300-hms-coff-objdump h8-objdump
# ln -s /usr/local/h8/bin/h8300-hms-coff-nm h8-nm
# ln -s /usr/local/h8/bin/h8300-hms-coff-size h8-size
```

これで各ツールが h8-xxxx で使用できる。インストール後は /usr/local/src 以下は消しても良い。

1.1.6. 使い方

1.1.6.1. [リンカースクリプトの編集]

ターゲットボードのメモリ配置に合うように ldscript を編集。また、ROM 化に対応させる。

オリジナルは /usr/local/h8/h8300-hms-coff/lib/ldscript/h8300h.x にあるのでこれを元にして編集する。(サンプル参照)

1.1.6.2. [スタートアップルーチン]

ROM 化に対応するためのスタートアップルーチンを作成。

オリジナルは /usr/local/src/newlib-1.8.2/newlib/libc/sys/h8300hms/crt0.S にあるのでこれを元にする。

スタートアップルーチンではスタックポインタの初期化、ハードウェア初期化(メモリ領域、メモリアクセスウェイト設定等)、初期化データ領域(DATA セクション)の初期化、非初期化データ領域(BSS セクション)の 0 初期化を行った後 main() 関数を呼び出す。

(サンプル参照)

1.1.6.3. [newlib 用グルールーチン] (cygwin の場合)

newlib のターゲット依存部を作成する。

_read(), _write() にシリアル 1 文字入出力ルーチンを書くと printf(), scanf(), puts(), gets() 等でシリアル経由の入出力が出来るようになる。ただし、printf(), scanf() を使用するとファイルサイズが膨大になるので要注意。

(サンプルの glue_newlib.c 参照)

1.1.6.4. [コンパイル方法]

```
# h8-gcc -T -nostartfiles -mh -mrelax -O2 -o ...
```

オプションの意味

-T : リンカースクリプトファイルの指定
-nostartfiles : デフォルトのスタートアップルーチンを使用しない
-mh : H8/300H を指定
-mrelax : 絶対アドレッシングを最適化
-O2 : 最適化レベル 2
-o : COFF 形式出力ファイル名
: 使用するスタートアップルーチン
: C 言語、アセンブラソースファイルまたはオブジェクトファイル

1.1.6.5. [バイナリ出力ファイル変換]

COFF 形式のバイナリファイルをもとローラ S レコードフォーマットファイル(S2)に変換する

```
# h8-objcopy -O srec
```

-O srec : モトローラ S レコードフォーマットを指定
: COFF 形式入力ファイル名
: S フォーマットファイル出力ファイル名

binutils-2.10 では S2 レコードを出力する(binutils-2.9.1 以前では S3)

(以上、1 - 4 ~ 1 - 6 は以下のページから引用し、読みやすく編集した。

<http://www.ertl.ics.tut.ac.jp/%7Emuranaka/devel/h8-gcc.txt>)

1.1.7. Cygwin で X を使う

日本語を入力するには、[Ctrl] + [¥] キーを押す。

日本語化の仕方が書いてあるページ

<http://www.atmarkit.co.jp/flinux/special/cygwin2/cygwin01a.html>

バイナリ等が置いてあるページ

<http://www.on.ics.keio.ac.jp/%7EEmaru/cygwin-xfree-jp-supplement/?05171600>

1.1.8. Cygwin の消し方

1. Setup を実行し、全てアンインストールにする。

2. Cygwin フォルダを消す。

3. スタートメニュー ファイル名を指定して実行から Regedit で HKEY_LOCAL_MACHINE¥SOFTWARE¥Cygwin Solutions を消す。

1.2. Cygwin 開発環境 (FreeBSDPress NO.14 を参照) から sample を実行するまで

1. setup-2.249.2.5-jp.exe を実行し全てインストールする。(最新版の setup ではおかしくなったことがある。最新版がうまくいかなかったらこれを使う)

2. binutils-2.13.1.tar.gz と newlib-1.10.0.tar.gz と gcc-core-3.2.tar.gz と h8300-hms-gcc-3.1-1.patch を /usr/local に入れて解凍する

3. binutils ディレクトリに入り、objdir ディレクトリを作りそれに入る。

4. #./configure --prefix=/usr/local/h8300 - target=h8300-hms --disable-nls

5. #make CFLAGS="-O2 -fomit-frame-pointer" all

6. #make install

7. #PATH=\$PATH:/usr/local/h8300/bin

8. #export PATH

9. gcc フォルダに入り

#ln -s ../newlib-1.10.0/newlib .

10. #patch -p1 < ../h8300-hms-gcc-3.1-1.patch

11. objdir というディレクトリを作りそこに入る

12. #./configure --prefix=/usr/local/h8300 --target=h8300-hms --enable-languages=c --with-newlib

13. #make CFLAGS="-O2 -fomit-frame-pointer" all

14. #PATH=\$PATH:/usr/local/h8300/bin

15. #export PATH

16. #make install

17. #mkdir /usr/local/h8300/h8300-hms/bfd && cd /usr/local/h8300/h8300-hms/bfd
(2004.01.12cd うんぬん追加)

18. #cp /usr/local/h8300/binutils-2.13/include/ansidecl.h .

19. #cp /usr/local/h8300/binutils-2.13/include/libiberty.h .

20. #cp /usr/local/h8300/binutils-2.13/objdir/bfd/bfd.h .

21. #cp /usr/local/h8300/binutils-2.13/objdir/bfd/libbfd.a .

22. #cp /usr/local/h8300/binutils-2.13/objdir/libiberty/libiberty.a .

23. #cp /usr/local/h8300/binutils-2.13/include/symcat.h .

24. #cp /usr/local/h8300/binutils-2.13/objdir/intl/libintl.h .

25. <http://www.toppers.jp/oldjsp-download.html> から落としてきたカーネル 1.3 と
コンフィギュレータを解凍して cfg を jsp に入れる

26. #cd jsp

27. Release1.3 では make depend の時に呼び出している perl スクリプトの処理が

gcc3.x 系のプリプロセッサの出力フォーマットに対応していません。

・ /utils/makedep の書き換え

1 2 4 行目を次のように変更します。(TOPPERS1.3Web ページより)

変更前

```
if ($line =~ /^#\s*([0-9]+)\s*\$"([\^"]+)"$/) {
```

変更後

```
if ($line =~ /^#\s*([0-9]+)\s*\$"([\^"]<>+)"$/) {
```

2 8 . /jsp/config/h8/akih8_3048f/sys_config.h の 208 行目からを以下のように書き換える

```
##if 0
```

```
##define ENABLE_LOWER_DATA
```

```
#define ENABLE_P8_CS
```

```
(H8P8DDR_CS0|H8P8DDR_CS1|H8P8DDR_CS2|H8P8DDR_CS3)
```

```
##define ENABLE_PA_CS (H8PADDR_CS4|H8PADDR_CS5|H8PADDR_CS6)
```

```
##define ENABLE_PB_CS H8PBDDR_CS7
```

```
##endif /* of #if 0 */
```

```
#endif /* _SYS_CONFIG_H_ */
```

2 9 . /jsp/config/h8/akih8_3048f/release.ld の 14 行目を

```
ram : o = 0x20000, l = 0x80000
```

に書き換える。128K から 512K を有効にする為。(変えなくても動く)

3 0 . #mkdir obj (このあたり user.txt を見ること)

3 1 . #cd obj

3 2 . #../configure C h8 S akih8_3048f P /usr/local/h8300

3 3 . Makefile の書き換え。52 行目の Cygwin = true のコメントアウトを外し、58 行目の GDB_STUB = true をコメントアウトし、145 行目の all: \$(JSPOBJ) をコメントアウトし、147 行目の Jsp.bin のコメントアウトを外す。

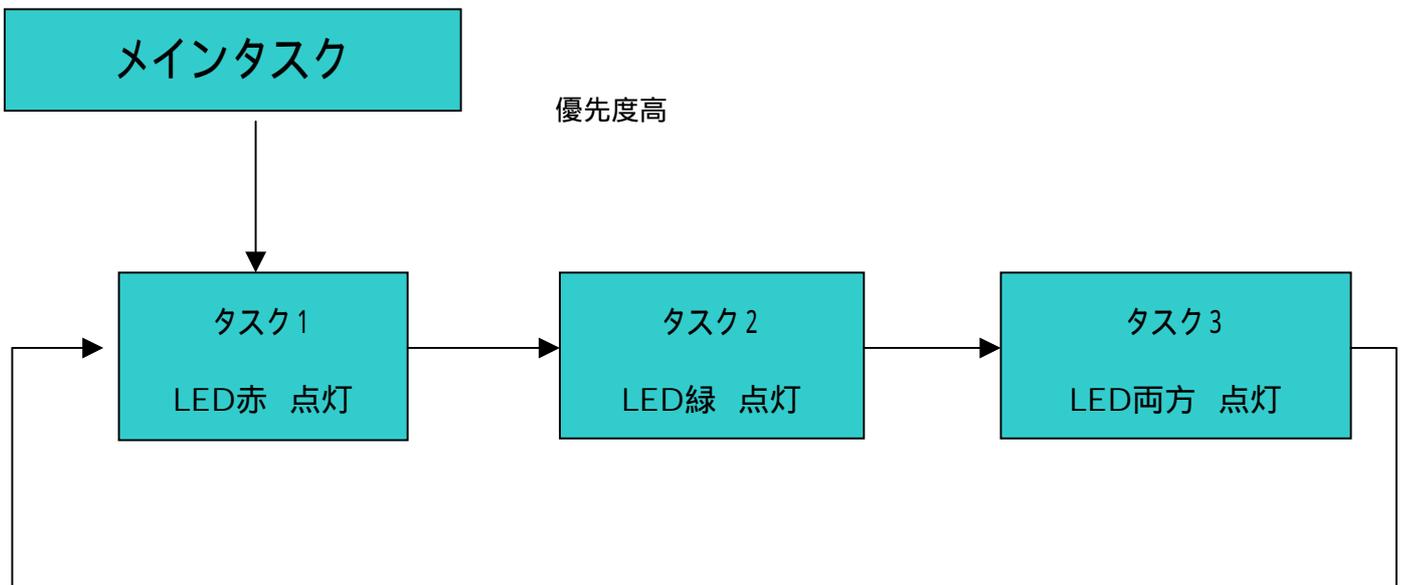
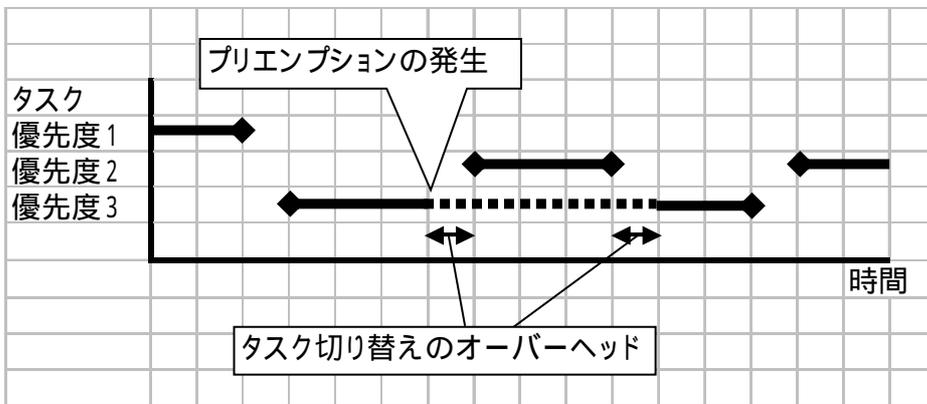
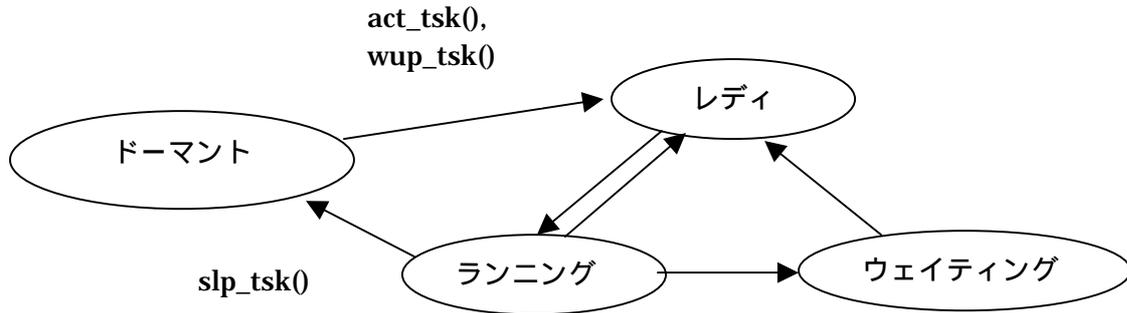
3 4 . #make depend

3 5 . #make

これでサンプルプログラムを jsp.bin というプログラムに収められたので、それをフラッシュライタ等で転送する。

3 6 .simpleterm・ハイパーターミナルなどの通信ソフトで RS232C 通信をする。(ボーレートは 9600bps)

2. 状態遷移を観察するための簡単なプログラム例



TRON を使用したプログラムにはメイン関数はないが、マシンが起動し、OS が起動された後まず何らかのプログラムが呼ばれないと何も起こらない。TOPPERS では main_task() という関数であらわされるタスク (以下メインタスク) がはじめに起動される。メインタスクはすべてのタスクの中で最高の優先度を持っている。そして、メインタスクから他のタスクが起動 (ドーマント状態からレディ状態に遷移、act_tsk()) され、プログラムがスタートする。

```
/*
 * タスクの起動
 */
act_tsk(TASK1);
act_tsk(TASK2);
act_tsk(TASK3);
/*
 * メインループ
 */
while(1){
    wup_tsk(TASK1); //TASK1 を起こすが、main_task のほうが優先度が高い
    //ので止まっている
    slp_tsk();      //main_task を眠らせると優先度の低いタスクが動き
    //始める
}
```

しかし、ほかのタスクを起動しても、メインタスクのほうが優先度が高いので何も起こらない。他のタスクに処理を行わせるためには、メインタスクが休止状態に（ランニング状態からドーマント状態に遷移）ならなければならない。

```

/*****/
プログラム例 (sample1.c)
三つのタスクが次々に入れ替わりながら起動する。
/*****/

#include <jsp_services.h>
#include "kernel_id.h"
#include "sample1.h"
//#define H8PBDDR          0xffffd4 //ポート B コントロール
//#define H8PBDR           0xffffd6

void task1(VP_INT exinf)
{
    volatile unsigned int counter;
    volatile int i;
    volatile char *ddr = (char *)H8PBDDR;
    volatile char *dr = (char *)H8PBDR;
    *ddr = 0xff;
    syslog(LOG_NOTICE, "task1 activated");
    while(1){
        slp_tsk();
        syslog(LOG_NOTICE, "task1 woke up");
        for(i=0; i<10; i++){
            *dr = 0x40;
            for(counter = 30000; counter>0; counter--);
            *dr = 0x00;
            for(counter = 60000; counter>0; counter--);
        }
        wup_tsk(TASK2);
    }
}

void task2(VP_INT exinf)
{
    volatile unsigned int counter;
    volatile int i;
    volatile char *ddr = (char *)H8PBDDR;
    volatile char *dr = (char *)H8PBDR;
    *ddr = 0xff;
    syslog(LOG_NOTICE, "task2 activated");
    while(1){
        slp_tsk();
        syslog(LOG_NOTICE, "task2 woke up");
        for(i=0; i<10; i++){
            *dr = 0x80;
            for(counter = 30000; counter>0; counter--);
            *dr = 0x00;
            for(counter = 60000; counter>0; counter--);
        }
    }
}

```

```

        }
        wup_tsk(TASK3);
    }
}
void task3(VP_INT exinf)
{
    volatile unsigned int counter;
    volatile int i;
    volatile char *ddr = (char *)H8PBDDR;
    volatile char *dr = (char *)H8PBDR;
    *ddr = 0xff;
    syslog(LOG_NOTICE, "task3 activated");
    while(1){
        slp_tsk();
        syslog(LOG_NOTICE, "task3 woke up");
        for(i=0;i<10;i++){
            *dr = 0xc0;
            for(counter = 30000;counter>0;counter--);
            *dr = 0x00;
            for(counter = 60000;counter>0;counter--);
        }
        wup_tsk(TASK1);
    }
}

/*
 * メインタスク
 */
void main_task(VP_INT exinf)
{
    ///////////////////////////////////////////////////////////////////

    volatile char *ddr = (char *)H8PBDDR;
    volatile char *dr = (char *)H8PBDR;
    volatile unsigned int counter;
    volatile int i;
    *ddr = 0xff;

    for(i = 0; i < 5; i++){
        *dr = 0x80;    //LED 緑のみ点灯
        for(counter = 60000;counter>0;counter--);
        *dr = 0x40;    //LED 赤のみ点灯
        for(counter = 60000;counter>0;counter--);
    }
    *dr = 0x00;        //消灯
    ///////////////////////////////////////////////////////////////////

    syslog_setmask(LOG_UPTO(LOG_INFO), LOG_UPTO(LOG_EMERG));

```

```

syslog(LOG_NOTICE, "Sample task starts (exinf = %d).", exinf);

serial_ioctl(0, (IOCTL_CRLF | IOCTL_RAW | IOCTL_IXON | IOCTL_IXOFF));

/*
 * タスクの起動
 */
act_tsk(TASK1);
act_tsk(TASK2);
act_tsk(TASK3);

/*
 * メインループ
 */
while(1){
    wup_tsk(TASK1); //TASK1 を起こすが、main_task のほうが優先度が高い
    //ので止まっている
    slp_tsk();      //main_task を眠らせると優先度の低いタスクが動き
    //始める
}

syslog(LOG_NOTICE, "Sample task ends.");
kernel_exit();
}

/*****          sample1.c   ここまで          *****/

```

```

/*****/
ヘッダファイル (sample1.h)
/*****/
#include "cpu_defs.h"
#include "sys_defs.h"

/*
 * 各タスクの優先度の定義
 */

#define MAIN_PRIORITY 5          /* メインタスクの優先度 */
                                /* HIGH_PRIORITY より高くすること */

#define HIGH_PRIORITY 9        /* 並列に実行されるタスクの優先度 */
#define MID_PRIORITY 10
#define LOW_PRIORITY 11

/*
 * ターゲット依存の定義 (CPU 例外ハンドラの起動方法など)
 */

#ifdef M68K

#define CPUEXC1 5              /* ゼロ除算例外 */
#define RAISE_CPU_EXCEPTION syslog(LOG_NOTICE, "zerodiv = %d", 10 / 0)

#elif defined(SH3)

#define CPUEXC1 224           /* ロードエラー例外 */
#define RAISE_CPU_EXCEPTION (*(volatile int *) 0xFFFFFEC1)

#elif defined(SH1)

#define CPUEXC1 9             /* CPU アドレスエラー例外 */
#define RAISE_CPU_EXCEPTION (*(volatile int *) 0xFFFFFEC1)
#ifdef TOKIWA_SH1
#define STACK_SIZE 512       /* タスクのスタックサイズ */
#endif /* TOKIWA_SH1 */

#elif defined(ARM7TDMI)

#define CPUEXC1 4             /* ロードエラー例外 */
#define RAISE_CPU_EXCEPTION (*(volatile int *) 0xFFFFFEC1)

#elif defined(V850)

#elif defined(H8)

```

```

#undef CPUEXC1 /* CPU 例外ハンドラをサポートしない */

#elif defined(H8S)

#undef CPUEXC1 /* CPU 例外ハンドラをサポートしない */
#define LOOP_REF 4000L /* 速度計測用のループ回数 */

#elif defined(MICROBLAZE)

#undef CPUEXC1 /* CPU 例外ハンドラをサポートしない */
#define STACK_SIZE 2048 /* タスクのスタックサイズ */

#elif defined(I386)

#define CPUEXC1 0 /* ゼロ除算例外 */
#define RAISE_CPU_EXCEPTION syslog(LOG_NOTICE, "zerodiv = %d", 10 / 0)
#define OMIT_VGET_TIM

#elif defined(TMS320C54X)

#undef CPUEXC1 /* CPU 例外ハンドラをサポートしない */
#define STACK_SIZE 320 /* タスクのスタックサイズ */
#define LOOP_REF 500000L /* 速度計測用のループ回数 */

#elif defined(LINUX)

#undef CPUEXC1 /* CPU 例外ハンドラをサポートしない */
#define OMIT_VGET_TIM
#define LOOP_REF 4000000 /* 速度計測用のループ回数 */

#endif

/*
 * ターゲットに依存する可能性のある定数の定義
 */

#ifndef STACK_SIZE
#define STACK_SIZE 8192 /* タスクのスタックサイズ */
#endif /* STACK_SIZE */

#ifndef LOOP_REF
#define LOOP_REF 1000000 /* 速度計測用のループ回数 */
#endif /* LOOP_REF */

/*
 * 関数のプロトタイプ宣言
 */
#ifndef _MACRO_ONLY

```

```
extern void    task1(VP_INT tskno);
extern void    task2(VP_INT tskno);
extern void    task3(VP_INT tskno);
extern void    main_task(VP_INT exinf);
#endif /* _MACRO_ONLY */
/*****      sample1.h   ここまで      *****/
```

```
/*
*****
コンフィギュレーションファイル (sample1.cfg)
*****
*/
/*
 * @(#) $Id: sample1.cfg,v 1.6 2002/04/10 10:56:46 hiro Exp $
 */

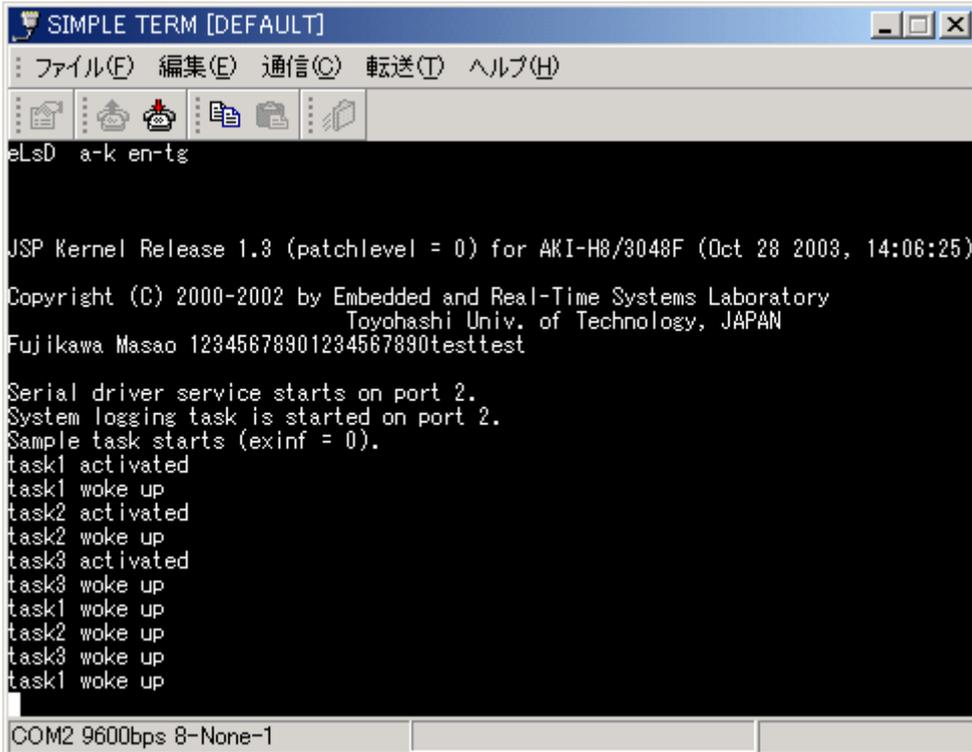
#define _MACRO_ONLY
#include "sample1.h"

INCLUDE("¥"sample1.h¥");
CRE_TSK(TASK1, { TA_HLNG, (VP_INT) 1, task1, MID_PRIORITY, STACK_SIZE, NULL });
CRE_TSK(TASK2, { TA_HLNG, (VP_INT) 2, task2, MID_PRIORITY, STACK_SIZE, NULL });
CRE_TSK(TASK3, { TA_HLNG, (VP_INT) 3, task3, MID_PRIORITY, STACK_SIZE, NULL });
CRE_TSK(MAIN_TASK, { TA_HLNG|TA_ACT, 0, main_task, MAIN_PRIORITY,
                    STACK_SIZE, NULL });

#include "../systask/timer.cfg"
#include "../systask/serial.cfg"
#include "../systask/logtask.cfg"

/*
***** sample1.cfg ここまで *****
*/
```

実行画面



```
SIMPLE TERM [DEFAULT]
ファイル(F) 編集(E) 通信(C) 転送(T) ヘルプ(H)
eLsD a-k en-tg

JSP Kernel Release 1.3 (patchlevel = 0) for AKI-H8/3048F (Oct 28 2003, 14:06:25)
Copyright (C) 2000-2002 by Embedded and Real-Time Systems Laboratory
Toyohashi Univ. of Technology, JAPAN
Fujikawa Masao 12345678901234567890testtest

Serial driver service starts on port 2.
System logging task is started on port 2.
Sample task starts (exinf = 0).
task1 activated
task1 woke up
task2 activated
task2 woke up
task3 activated
task3 woke up
task1 woke up
task2 woke up
task3 woke up
task1 woke up

COM2 9600bps 8-None-1
```

上記のプログラムを実行すると、まずメインタスク内の処理で二つのLEDがすばやく点滅した後タスク1の処理でLEDがゆっくり点滅する。処理が終わるとタスク1はタスク2を起床させる（ドーマント状態からレディ状態に遷移、`wup_tsk(TASK2)`）。ただしこの時はまだタスク2には処理は移らない。なぜなら、「最高度の優先順位のタスクのうち、一番初めにレディ状態になったものが実行権限を獲得する」という決まりがあるからである。その後タスク1が自ら休止状態に入り（`slp_tsk()`）、タスク2が実行権限を得、処理をはじめ。同様にタスク2はタスク3を起床させ、タスク3はタスク1を起床させるから、無限ループとなる。

RS-232C 通信も行っており、画面には上の図のようなメッセージが現われる。

コンパイルの手間を減らす方法

```
#make clean
#rm sample1.depend
#make depend
#make
```

は

```
#make realclean && make depend && make
```

ですむ。連続で処理を行い、途中でエラーがあった場合はそこで処理を中断してくれる。また、上のものをシェルスクリプトとして登録しておくことさらに楽になる。

GCC developer lite と同じスタイルでプログラムを書く方法

GCC developer lite の `3048.h` をインクルードすることで、よりわかりやすいプログラムを書くことができる。その際、ただコピーしただけだとエラーが出るので、インライン命令関係の部分を消すとよい。具体的には以下の部分を削除する。

```

/*-----*/
/* インライン命令 */
/*-----*/
__inline__ void sleep (void) __attribute__ ((always_inline));
__inline__ void nop (void) __attribute__ ((always_inline));
__inline__ void set_ccr (unsigned char mask) __attribute__ ((always_inline));
__inline__ unsigned char get_ccr (void) __attribute__ ((always_inline));
__inline__ void *get_sp (void) __attribute__ ((always_inline));

extern __inline__ void sleep (void)
{
    asm ("sleep");
}

extern __inline__ void nop (void)
{
    asm ("nop");
}

extern __inline__ void set_ccr (unsigned char mask)
{
    asm ("ldc %0l, ccr"::"r"(mask));
}

extern __inline__ unsigned char get_ccr (void)
{
    unsigned char value;
    asm ("stc ccr,%0l"::"g"(value):);
    return(value);
}

extern __inline__ void *get_sp (void)
{
    void *sp;
    asm ("mov.l r7,%0"::"g"(sp):);
    return(sp);
}

```

メモリの特定の場所に数値を書き込む方法について

(<http://www02.so-net.ne.jp/~morinaga/bbs/minibbs.cgi?log=log1> より引用)

【2310】レジスタ宣言

2003/11/19(水)03:02 - バーニング宮田 <mimii@ybb.ne.jp> 削除

谷さんこんばんは。

こうやって書いてしまうと

```

#define P5DDR 0xfffc8
P5DDR = 0x01;

```

コンパイラはこう解釈します。

```
#define P5DDR 0xfffc8
0xfffc8 = 0x01;
```

これだと定数に定数を代入する事になってエラーになっちゃいます。
なのでこう書くと

```
#define P5DDR (*(0xfffc8))
P5DDR = 0x01;
```

```
(*(0xfffc8)) = 0x01;
```

こんな感じになって 0xffc8 というアドレスに 0x01 を書くという意味になります。
しかしこれでは型が分からないので 1 バイトの 0x01 なのか 4 バイトの 0x00000001 なのか
分からないわけですね。というわけで

```
#define P5DDR (*(unsigned char)0xfffc8)
P5DDR = 0x01;
```

```
(*(unsigned char)0xfffc8) = 0x01;
```

という風を書くことで 1 バイト書くんだということになる訳です。

最後にこれがレジスタの宣言なら `volatile` を追加しておいた方が良いと思います。
コンパイラが最適化をしたときに処理を省かれずに済みます。

```
#define P5DR (*(volatile unsigned char *)0xFFFC4)
```

```
/* P5DR の 0bit 目が 0 になるまで待つ */
while(P5DR & 0x01);
```

こう書いたときに `volatile` 宣言がないと P5DR を普通の変数だと勘違いされ
P5DR という変数は値が変化しないのに while ループで見ても無駄だろう
とコンパイラが勝手に処理を省いてしまいます。

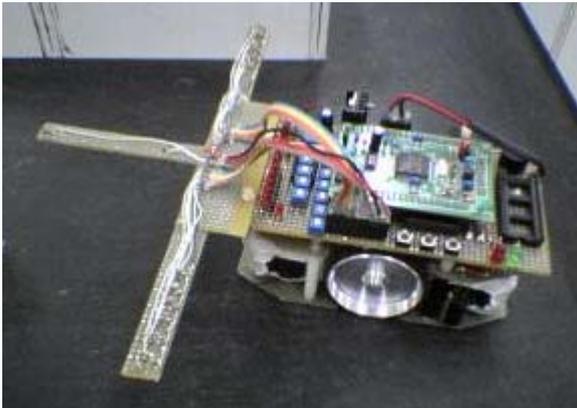
と、こんな感じです。
それでは。

3 . マイクロマウスへの ITRON の実装

2003 年の 11 月 22 日に、東京の「パナソニックセンター」で「マイクロマウス 2003」大会が開かれたので、同型機をもう一台作り、出場した。(多賀が OS なしのプログラムで出場し、藤川が OS ありのプログラムで出場した。当初同じ機体を使うつもりだったが、時間がバツィングする恐れが出てきたため、ほとんど同じ機体をもう一つ作った(協力: 高橋直樹、西庄一紘、多賀))。結果としてはエキスパートクラスで完走(記録: 58 秒、40 位)。

ニューテクノロジー振興財団マイクロマウス委員会の Web ページ
(<http://www.bekkoame.ne.jp/~ntf/mouse/mouse.html>)

外観



回路図は、センサの一部を除いて以前作ったもの(前期レポート参照)とほぼ同じなので省略する。

タイマ割り込みのプログラムがどうしてもうまくいかなかったため、OS の機能「周期ハンドラ」をタイマの代わりに用い、モータの速度調節を行っている。周期ハンドラは 1ms ごとにしかイベントを起こせないため、スピードはあまり上げることができなかった。また、左右の位置補正もなめらかなものは実現できなかった。しかし、危なげなく完走できるだけの性能があり、また、OS の機能を生かしたプログラム例として適当なため、取り上げることにする。

内容の詳細については後述する。

プログラム

teigi.h、sample1.cfg、sample1.h、sample1.c

```

/*****
      teigi.h
*****/

#define LEFTGO      1      //左モータ前進
#define LEFTBACK   0      //左モータ後進
#define RIGHTGO    0      //右モータ前進
#define RIGHTBACK  1      //右モータ後進

#define ON         1
#define OFF        0

#define HIDARI_GRA ITU0.GRA //左モータの GRA
#define MIGI_GRA   ITU1.GRA //右モータの GRA

#define GO_STEP 945          //一区間のステップ数 (180 * 200 * 2 * 2) / (46.9
* 3.14) = 977.822443741257316692243966699713
                        //式の意味 1区間の長さ トグル出力なので 1パルス
                        //0.9° 一周するのに必要なパルス数 タイヤの直径 円周率
#define TURN_STEP      288 //超信旋回(その場で旋回)ステップ数 ((58.5 + 5 *
2) * 3.14 * 200 / 4 * 2 * 2) / (46.9 * 3.14)=292.110874200426439232409381663113
                        //式の意味 左右のタイヤの間の長さ タイヤの太さ
                        //2個 円周率 一周するのに必要なパルス数 1/4周 トグル出力なので 1パルス 0.9°
                        //直径 円周率

#define HIDARI_GRA ITU0.GRA //左モータの GRA
#define MIGI_GRA   ITU1.GRA //右モータの GRA

//=====ポート番号設定
#define LSEN1 PB.DR.BIT.B0 //左側のセンサ
#define LSEN2 PB.DR.BIT.B1 // 0:壁なし 1:壁あり
#define LSEN3 PB.DR.BIT.B2
#define LSEN4 PB.DR.BIT.B3
#define LSEN5 PB.DR.BIT.B4
#define LSEN6 PB.DR.BIT.B5
#define LSEN (PB.DR.BYTE & 0x3f)

#define RSEN1 P7.DR.BIT.B0 //右側のセンサ
#define RSEN2 P7.DR.BIT.B1
#define RSEN3 P7.DR.BIT.B2
#define RSEN4 P7.DR.BIT.B3
#define RSEN5 P7.DR.BIT.B4
#define RSEN6 P7.DR.BIT.B5
#define RSEN (P7.DR.BYTE & 0x3f)

#define RSENIN (P7.DR.BYTE & 0x03)
#define LSENIN (PB.DR.BYTE & 0x30)

#define FSEN1 P7.DR.BIT.B6 //前のセンサ

```

```
#define FSEN2 P7.DR.BIT.B7
#define FSEN (P7.DR.BYTE & 0xc0)

#define SW1 P6.DR.BIT.B0 //ボタン
#define SW2 P6.DR.BIT.B1 // 0:押されてない 1:押されている
#define BUZZER P6.DR.BIT.B2 //ブザー

#define MOTOR_EN PA.DR.BIT.B0 //モーターOnOff
#define MOTOR_R PA.DR.BIT.B1 //右モーター 前向きか後ろ向きか
#define MOTOR_L PA.DR.BIT.B3 //左モーター 前向きか後ろ向きか

#define LED1 PB.DR.BIT.B6
#define LED2 PB.DR.BIT.B7
```

```

/*****
      sample1.cfg
*****/
/*
 *  @(#) $Id: sample1.cfg,v 1.6 2002/04/10 10:56:46 hiro Exp $
 */

#define _MACRO_ONLY
#include "sample1.h"

INCLUDE("¥"sample1.h¥");

CRE_TSK(TANSAKU_TASK1, { TA_HLNG, (VP_INT) 5, tansaku_task, LOWER_PRIORITY,
STACK_SIZE, NULL });
CRE_TSK(TANSAKU_TASK2, { TA_HLNG, (VP_INT) 10, tansaku_task, LOWER_PRIORITY,
STACK_SIZE, NULL });
CRE_TSK(CHALLENGE_TASK1, { TA_HLNG, (VP_INT) 9, challenge_task, LOWER_PRIORITY,
STACK_SIZE, NULL });
CRE_TSK(CHALLENGE_TASK2, { TA_HLNG, (VP_INT) 11, challenge_task, LOWER_PRIORITY,
STACK_SIZE, NULL });
CRE_TSK(CHALLENGE_TASK3, { TA_HLNG, (VP_INT) 12, challenge_task, LOWER_PRIORITY,
STACK_SIZE, NULL });

CRE_TSK(KUKAN_TASK, { TA_HLNG, (VP_INT) 1, kukan_task, LOW_PRIORITY, STACK_SIZE,
NULL });
CRE_TSK(TURN_R_TASK, { TA_HLNG, (VP_INT) 6, turn_r_task, LOW_PRIORITY, STACK_SIZE,
NULL });
CRE_TSK(TURN_L_TASK, { TA_HLNG, (VP_INT) 7, turn_l_task, LOW_PRIORITY, STACK_SIZE,
NULL });
CRE_TSK(TURN_U_TASK, { TA_HLNG, (VP_INT) 8, turn_u_task, LOW_PRIORITY, STACK_SIZE,
NULL });

CRE_TSK(SHUSEI_TASK, { TA_HLNG, (VP_INT) 4, shusei_task, HIGH_PRIORITY, STACK_SIZE,
NULL });
CRE_TSK(MOT_L_TASK, { TA_HLNG, (VP_INT) 2, mot_l_task, HIGHER_PRIORITY, STACK_SIZE,
NULL });
CRE_TSK(MOT_R_TASK, { TA_HLNG, (VP_INT) 3, mot_r_task, HIGHER_PRIORITY, STACK_SIZE,
NULL });
CRE_TSK(MAIN_TASK, { TA_HLNG|TA_ACT, 0, main_task, MAIN_PRIORITY, STACK_SIZE,
NULL });

CRE_CYC(MOT_CYCHDR, { TA_HLNG, 0, mot_cyclic_handler, 1, 0 });
CRE_CYC(SHUSEI_CYCHDR, { TA_HLNG, 0, shusei_cyclic_handler, 5, 0 });
CRE_CYC(SHUSEI_CYCHDR2, { TA_HLNG, 0, shusei_cyclic_handler2, 20, 0 });
CRE_CYC(LED_CYCHDR, { TA_HLNG, 0, led_cyclic_handler, 250, 0 });

CRE_SEM(1, {TA_TFIFO, 1, 1});      /* マイクロマウス自身を排他制御するセマフォ */

```

```
CRE_FLG(SHURYO_FLG, { TA_TFIFO | TA_CLR, 0});  
#include "../systask/timer.cfg"  
#include "../systask/serial.cfg"  
#include "../systask/logtask.cfg"
```

```

/*****
    sample1.h
*****/
#include "cpu_defs.h"
#include "sys_defs.h"

/*
 * 各タスクの優先度の定義
 */

#define MAIN_PRIORITY 5          /* メインタスクの優先度 */
                                /* HIGH_PRIORITY より高くすること */
#define HIGHEST_PRIORITY 7      /* 並列に実行されるタスクの優先
度 */
#define HIGHER_PRIORITY 8       /* 並列に実行されるタスクの優先度 */
#define HIGH_PRIORITY 9         /* 並列に実行されるタスクの優先度 */
#define MID_PRIORITY 10
#define LOW_PRIORITY 11
#define LOWER_PRIORITY 12
#define LOWEST_PRIORITY 13

/*
 * ターゲット依存の定義 (CPU 例外ハンドラの起動方法など)
 */

#ifdef M68K

#define CPUEXC1 5                /* ゼロ除算例外 */
#define RAISE_CPU_EXCEPTION syslog(LOG_NOTICE, "zerodiv = %d", 10 / 0)

#elif defined(SH3)

#define CPUEXC1 224              /* ロードエラー例外 */
#define RAISE_CPU_EXCEPTION (*((volatile int *) 0xFFFFFEC1))

#elif defined(SH1)

#define CPUEXC1 9                /* CPU アドレスエラー例外 */
#define RAISE_CPU_EXCEPTION (*((volatile int *) 0xFFFFFEC1))
#ifdef TOKIWA_SH1
#define STACK_SIZE 512          /* タスクのスタックサイズ */
#endif /* TOKIWA_SH1 */

#elif defined(ARM7TDMI)

#define CPUEXC1 4                /* ロードエラー例外 */
#define RAISE_CPU_EXCEPTION (*((volatile int *) 0xFFFFFEC1))

```

```

#elif defined(V850)

#elif defined(H8)

#undef CPUEXC1 /* CPU 例外ハンドラをサポートしない */

#elif defined(H8S)

#undef CPUEXC1 /* CPU 例外ハンドラをサポートしない */
#define LOOP_REF 4000L /* 速度計測用のループ回数 */

#elif defined(MICROBLAZE)

#undef CPUEXC1 /* CPU 例外ハンドラをサポートしない */
#define STACK_SIZE 2048 /* タスクのスタックサイズ */

#elif defined(I386)

#define CPUEXC1 0 /* ゼロ除算例外 */
#define RAISE_CPU_EXCEPTION syslog(LOG_NOTICE, "zerodiv = %d", 10 / 0)
#define OMIT_VGET_TIM

#elif defined(TMS320C54X)

#undef CPUEXC1 /* CPU 例外ハンドラをサポートしない */
#define STACK_SIZE 320 /* タスクのスタックサイズ */
#define LOOP_REF 500000L /* 速度計測用のループ回数 */

#elif defined(LINUX)

#undef CPUEXC1 /* CPU 例外ハンドラをサポートしない */
#define OMIT_VGET_TIM
#define LOOP_REF 4000000 /* 速度計測用のループ回数 */

#endif

/*
 * ターゲットに依存する可能性のある定数の定義
 */

#ifndef STACK_SIZE
#define STACK_SIZE 8192 /* タスクのスタックサイズ */
#endif /* STACK_SIZE */

#ifndef LOOP_REF
#define LOOP_REF 1000000 /* 速度計測用のループ回数 */
#endif /* LOOP_REF */

```

```
/*
 * 関数のプロトタイプ宣言
 */
#ifndef _MACRO_ONLY

extern void    tansaku_task(VP_INT tskno);
extern void    challenge_task(VP_INT tskno);
extern void    kukan_task(VP_INT tskno);
extern void    shusei_task(VP_INT tskno);
extern void    mot_l_task(VP_INT tskno);
extern void    mot_r_task(VP_INT tskno);
extern void    main_task(VP_INT exinf);
extern void    turn_r_task(VP_INT exinf);
extern void    turn_l_task(VP_INT exinf);
extern void    turn_u_task(VP_INT exinf);
extern void    mot_cyclic_handler(VP_INT exinf);
extern void    shusei_cyclic_handler(VP_INT exinf);
extern void    shusei_cyclic_handler2(VP_INT exinf);
extern void    led_cyclic_handler(VP_INT exinf);

extern void Init(void);
extern void mouse_search(int tx,int ty,int mode);
extern void make_smap(int gx,int gy,int mode);
extern void make_map_data(void);
extern void clear_map(void);
#endif /* _MACRO_ONLY */
```

```

/*****
    sample1.c
*****/
#include <jsp_services.h>
#include "kernel_id.h"
#include "sample1.h"
#include "3048.h"    //ポートなど定義
#include "teigi.h"   //ポート設定

#define NORMAL_WAIT    1
#define LONG_WAIT      2
#define MOUSE_SEM     1

typedef unsigned char uchar; //型宣言

unsigned int wait_l = 1;
unsigned int wait_r = 1;
unsigned int kukan_pulse;

#define      KEY_OFF          50          /* スイッチにチャタリン
グキャンセル時間          */
#define      S_MODE          0          /* 未探索区間は壁が無い
ものとして扱う          */
#define      T_MODE          1          /* 未探索区間は壁が有る
ものとして扱う          */

uchar      map[16][16];    /* MAP データ
                        */
uchar      smap[16][16];  /* 等高線 データ
                        */

uchar      head;          /* マウスの進行方向 0:N 1:E 2:S
3:W
                        */
uchar      my;            /* マウスのX座標
                        */
uchar      mx;            /* マウスのY座標
                        */

uchar      F_SEN,LS_SEN,RS_SEN;
uchar      shusei_mode = 0;

/*****
*/
/*          タスク
*/
*****/
/*****
*/

```

```

/*-----*/
/*          第1走行（探索走行）
          */
/*-----*/
void tansaku_task(VP_INT exinf)
{
    //マイクロマウス自身を共有資源と考える
    wai_sem(MOUSE_SEM);          /* ID が1のセマフォを - 1、すでに
0 なら待つ*/
    mx=0;my=0;head=0;          /* マウスの座標を初期化する
          */
    LED1 = 0; LED2 = 1;
    mouse_search(7,7,S_MODE);    /* ゴールまで探索しながら走行する
          */
    mouse_search(0,0,S_MODE);    /* スタートまで探索しながら走行する
          */
    LED1 = 0; LED2 = 0;
    sta_cyc(LED_CYCHDR);        //LED を点滅しながら走る
    dly_tsk(3000);              //3 秒間停止
    stp_cyc(LED_CYCHDR);
    sig_sem(MOUSE_SEM);        /* ID が1のセマフォを + 1*/
    ext_tsk();
}

/*-----*/
/*          第2走行（チャレンジ走行）
          */
/*-----*/
void challenge_task(VP_INT exinf)
{
    wai_sem(MOUSE_SEM);          /* ID が1のセマフォを - 1、すでに
0 なら待つ*/
    mx=0;my=0;head=0;          /* マウスの座標を初期化する
          */
    LED1 = LED2 = 0;          //LED は消したまま走る
    mouse_search(7,7,T_MODE);    /* ゴールまで最短距離で走行する
          */
    mouse_search(0,0,S_MODE);    /* スタートまで探索しながら走行する
          */
    dly_tsk(3000);              //3 秒間停止
    sig_sem(MOUSE_SEM);        /* ID が1のセマフォを + 1*/
    ext_tsk();
}

/*
void kaiseki_task(VP_INT exinf)
{
    //右手法

```

```

while(1){
    act_tsk(KUKAN_TASK);
    wai_flg(SHURYO_FLG,1,TWF_ANDW,NULL);
    if(migi == 0)
        act_tsk(TURN_R_TASK);
    else{
        if(mae == 0)
            set_flg(SHURYO_FLG,1); //終了を知らせるためイベ
ントフラグに1を書き込む
        else{
            if(hidari == 0)
                act_tsk(TURN_L_TASK);
            else
                act_tsk(TURN_U_TASK);
        }
    }
    wai_flg(SHURYO_FLG,1,TWF_ANDW,NULL);
}
}
*/
void kukan_task(VP_INT exinf)
{
    kukan_pulse = GO_STEP + GO_STEP;
    MOTOR_R = RIGHTGO;           //前向き
    MOTOR_L = LEFTGO;           //前向き モーターが逆についている為
    F_SEN = RS_SEN = LS_SEN = 0;
    sta_cyc(MOT_CYCHDR); //モータをまわすための周期ハンドラスタート
    sta_cyc(SHUSEI_CYCHDR); //センサで軌道修正の周期ハンドラスタート
    MOTOR_EN = ON;              //モーターON
    ext_tsk();
}

void turn_r_task(VP_INT exinf){
    kukan_pulse = TURN_STEP + TURN_STEP;
    MOTOR_R = RIGHTBACK;        //後ろ向き
    MOTOR_L = LEFTGO;          //前向き
    sta_cyc(MOT_CYCHDR); //モータをまわすための割り込み
    MOTOR_EN = ON;            //モーターON
    ext_tsk();
}

void turn_l_task(VP_INT exinf){
    kukan_pulse = TURN_STEP + TURN_STEP;
    MOTOR_R = RIGHTGO;         //前向き
    MOTOR_L = LEFTBACK;       //後ろ向き
    sta_cyc(MOT_CYCHDR); //モータをまわすための割り込み
    MOTOR_EN = ON;            //モーターON
    ext_tsk();
}

```

```

void turn_u_task(VP_INT exinf){
    kukan_pulse = TURN_STEP + TURN_STEP + TURN_STEP + TURN_STEP;
    MOTOR_R = RIGHTBACK;          //後ろ向き
    MOTOR_L = LEFTGO;             //前向き
    sta_cyc(MOT_CYCHDR);         //モータをまわすための割り込み
    MOTOR_EN = ON;                //モーターON
    ext_tsk();
}
void shusei_task(VP_INT exinf)
{
    while(1){
        slp_tsk();
        if(LSENIN) wait_l = LONG_WAIT; //左より過ぎ
        else if(RSENIN) wait_r = LONG_WAIT; //右より過ぎ
        else{
            if(LSEN){
                if(LSEN > 0x0c) wait_l = LONG_WAIT;
                //右より過ぎ
                else if(LSEN < 0x0c) wait_r = LONG_WAIT;
                //左より過ぎ
            }
            else if(RSEN){
                if(RSEN > 0x0c) wait_l = LONG_WAIT;
                //右より過ぎ
                else if(RSEN < 0x0c) wait_r = LONG_WAIT;
                //左より過ぎ
            }
        }
        if(FSEN != 0) F_SEN = 1;
        if(FSEN2) kukan_pulse = 1;
    }
}
void mot_l_task(VP_INT exinf)
{
    while(1){
        slp_tsk();
        PA.DR.BIT.B4 = ~PA.DR.BIT.B4; //トルグル出力
        wait_l = NORMAL_WAIT;
        kukan_pulse--;
        if(kukan_pulse == GO_STEP + GO_STEP / 5){
            shusei_mode = 1;
        }
        if(kukan_pulse == GO_STEP+10){
            if(RSEN != 0) RS_SEN = 1;
            else RS_SEN = 0;
            if(LSEN != 0) LS_SEN = 1;
            else LS_SEN = 0;
        }
    }
}

```

```

    }
    if(kukan_pulse == GO_STEP){
        if(RSEN != 0 && RS_SEN == 1)    RS_SEN = 1;
        else RS_SEN = 0;
        if(LSEN != 0 && LS_SEN == 1) LS_SEN = 1;
        else LS_SEN = 0;
    }
    if(kukan_pulse == GO_STEP + GO_STEP - GO_STEP * 4 / 5){
        shusei_mode = 0;
    }
    if(kukan_pulse == 0){
        stp_cyc(MOT_CYCHDR);    //モータをまわすためのハンドラ停
止
        stp_cyc(SHUSEI_CYCHDR); //センサで軌道修正停止
        set_flg(SHURYO_FLG,1); //終了を知らせるためイベントフラ
グに 1 を書き込む
    }
}
void mot_r_task(VP_INT exinf)
{
    while(1){
        slp_tsk();
        PA.DR.BIT.B6 = ~PA.DR.BIT.B6;
        wait_r = NORMAL_WAIT;
        kukan_pulse--;
        if(kukan_pulse == GO_STEP + GO_STEP / 5){
            shusei_mode = 1;
        }
        if(kukan_pulse == GO_STEP+10){
            if(RSEN != 0)    RS_SEN = 1;
            else RS_SEN = 0;
            if(LSEN != 0) LS_SEN = 1;
            else LS_SEN = 0;
        }
        if(kukan_pulse == GO_STEP){
            if(RSEN != 0 && RS_SEN == 1)    RS_SEN = 1;
            else RS_SEN = 0;
            if(LSEN != 0 && LS_SEN == 1) LS_SEN = 1;
            else LS_SEN = 0;
        }
        if(kukan_pulse == GO_STEP + GO_STEP - GO_STEP * 4 / 5){
            shusei_mode = 0;
        }
        if(kukan_pulse == 0){
            stp_cyc(MOT_CYCHDR);    //モータをまわすためのハンドラ停
止
            stp_cyc(SHUSEI_CYCHDR); //センサで軌道修正停止

```

```

        set_flg(SHURYO_FLG,1); //終了を知らせるためイベントフラ
グに 1 を書き込む
    }
}

/*****
*/
/*          サブルーチン
*/
/*****
*/
/* ===== */
/*          イニシャライズ          */
/* ===== */
void Init(void)
{
    PA.DDR = 0xff;          //全ビット出力設定   モーター関係
    P6.DDR = 0x04;          //全ビット入力設定   ボタン   一つだけブザー用出力
    PB.DDR = 0xc0;          //センサは入力   LED は出力に設定

    MOTOR_EN = OFF;        //モーターOff
    MOTOR_R = RIGHTGO;     //前向き
    MOTOR_L = LEFTGO;      //前向き   モーターが逆についている為
    LED1 = LED2 = 0;
}

/* ===== */
/*          迷路探索
*/
/*          マウスの現在位置 (mx,my,head)   HEAD(0:N 1:E 2:S 3:W)
*/
/*          モードが 0 の場合は未探索区間は壁無しと想定して(tx,ty)へ向かう
*/
/*          モードが 1 の場合は未探索区間は壁有りとして(tx,ty)へ向かう
*/
/*          壁データは map[][]に記録される
*/
/*          壁データ記録方法
*/
/*          壁の位置 0bit:上 1bit:右 2bit:下 3bit:左          1:壁有り 0:壁無し
*/
/*          壁の探索 4bit:上 5bit:右 6bit:下 7bit:左          1:済み 0:未探索
*/
/* ===== */

```

```

void mouse_search(int tx,int ty,int mode)
{
    short    temp0;
    unsigned short  s0,s1,head0 = 0;

    while(1){
        /* ---- 直進開始 -----
        */
        if          (head == 0) my = my +1; /* 座標更新
        */
        else if (head == 1) mx = mx +1;
        else if (head == 2) my = my -1;
        else if (head == 3) mx = mx -1;

        act_tsk(KUKAN_TASK);
        make_smap(tx,ty,mode);          /* 等高線マップを作る
        */
        wai_flg(SHURYO_FLG,1,TWF_ANDW,NULL);
        make_map_data();                /* 壁データから
迷路データを作る          */

        /* 回りの4区間の内、その方向に壁がなくて、目的地に一番近い
        */
        /* 区間に移動する。ただし、移動できる一番近い区間が複数ある
        */
        /* 場合は(未探索区間,直進)(未探索区間,旋回)(既探索区間,直進)
        */
        /* (既探索区間,旋回)の順で選択する
        */
        temp0 = map[my][mx];
        s0 = 1024;
        if ((temp0 & 1)==0){                /* 北方向の
区間の確認          */
            s1 = smap[my+1][mx]*4+4;
            if ((map[my+1][mx]&0x0f0)!=0x0f0)    s1 = s1 - 2;
            if (head == 0)
                s1 = s1 - 1;
            if (s1<s0) {s0 = s1;head0 = 0;}
        }
        if ((temp0 & 2)==0){                /* 東方向の
区間の確認          */
            s1 = smap[my][mx+1]*4+4;
            if ((map[my][mx+1]&0x0f0)!=0x0f0)    s1 = s1 - 2;
            if (head == 1)
                s1 = s1 - 1;
            if (s1<s0) {s0 = s1;head0 = 1;}
        }
        if ((temp0 & 4)==0){                /* 南方向の

```

```

区間の確認          */
                    s1 = smap[my-1][mx]*4+4;
                    if ((map[my-1][mx]&0x0f0)!=0x0f0)      s1 = s1 - 2;
                    if (head == 2)
s1 = s1 - 1;
                    if (s1<s0) {s0 = s1;head0 = 2;}
                }
                if ((temp0 & 8)==0){                          /*      西方向の
区間の確認          */
                    s1 = smap[my][mx-1]*4+4;
                    if ((map[my][mx-1]&0x0f0)!=0x0f0)      s1 = s1 - 2;
                    if (head == 3)
s1 = s1 - 1;
                    if (s1<s0) {s0 = s1;head0 = 3;}
                }

head0 = (head0 - head) & 3;                                  /* 移動する方向
を決定              */

/* ----- 決定した方向に移動する -----
*/
if ((mx==tx)&&(my==ty)){/* ----- 目的地の場合 ----- */
    /* 180度反転      */
    act_tsk(TURN_U_TASK);
    wai_flg(SHURYO_FLG,1,TWF_ANDW,NULL);
    head = (head + 2) & 3;
/* 進行方向更新      */
    return;
}else if (head0 == 0){ /* ----- そのまま前進 ----- */
    set_flg(SHURYO_FLG,1); //終了を知らせるためイベントフラ
グに1を書き込む
}else if (head0 == 1){ /* ----- 右に旋回する ----- */
    /* 右90度旋回      */
    act_tsk(TURN_R_TASK);
}else if (head0 == 3){ /* ----- 左に旋回する ----- */
    /* 左90度旋回      */
    act_tsk(TURN_L_TASK);
}else if (head0 == 2){ /* ----- 反転して戻る ----- */
    /* 180度反転      */
    act_tsk(TURN_U_TASK);
}
wai_flg(SHURYO_FLG,1,TWF_ANDW,NULL);
head = (head + head0) & 3;
/* 進行方向更新      */
}
}

```

```

/* ===== */
/*      M A P                                     */
/*                                                     */
/* 壁データ記録方法                                     */
/*                                                     */
/* 壁の位置 0bit:上 1bit:右 2bit:下 3bit:左      1:壁有り 0:壁無し */
/*                                                     */
/* 壁の探索 4bit:上 5bit:右 6bit:下 7bit:左      1:済み 0:未探索 */
/*                                                     */
/* ===== */

/* ===== */
/* マップデータ初期化                                     */
/*                                                     */
/* ===== */
void clear_map()
{
    short  x,y;
    char    d;

    /* すべてのマップデータを未探索状態にする */
    for(y=0;y<16;y++){
        for(x=0;x<16;x++){
            d = 0x0000;
            if      ((x == 0)&&(y == 0))      d = 0xfe;
            else if ((x == 1)&&(y == 0))      d = 0xcc;
            else if ((x == 15)&&(y == 0))     d = 0x66;
            else if ((x == 0)&&(y == 15))    d = 0x99;
            else if ((x == 15)&&(y == 15))   d = 0x33;
            else if (x == 0)                 d = 0x88;
            else if (x == 15)                d = 0x22;
            else if (y == 0)                 d = 0x44;
            else if (y == 15)                d = 0x11;
            map[y][x] = d;
        }
    }
}

/* ===== */
/* 等高線作成モジュール                                     */
/*                                                     */
/* ===== */
void make_smap(int gx,int gy,int mode)
{
    uchar  pt0,pt1,ct;
    short  x,y,z;

```



```

0x08)==0x00)&&(x!=0)){

    if (smap[y][x-1]==255){smap[y][x-1]=pt1;ct++;}}
        }
    }
    }
    pt0 = pt0+1;
}while(ct!=0);
}

/* ===== */
/* センサ情報から迷路データを作りマップに書き込むモジュール
   */
/* ===== */
void make_map_data()
{
    uchar wall;

    /* 走行中にセンサから得た壁情報をM A Pデータに書きこむ */
    if ((mx==0) && (my==0)) wall = 0x0fe; else wall = (uchar)get_wall_data();
    map[my][mx] = wall;

    /* 隣の区間のM A Pデータも更新する */
    if (mx !=15) map[my][mx+1]=(map[my][mx+1]&0x77)|0x80|((wall<<2)&0x08);
    if (mx != 0) map[my][mx-1]=(map[my][mx-1]&0xdd)|0x20|((wall>>2)&0x02);
    if (my !=15) map[my+1][mx]=(map[my+1][mx]&0xbb)|0x40|((wall<<2)&0x04);
    if (my != 0) map[my-1][mx]=(map[my-1][mx]&0xee)|0x10|((wall>>2)&0x01);
}

int get_wall_data()
{
    short wall;

    /* センサデータの入力し閾値と比較し壁の有無を判定する */
    wall = 0;

    if (F_SEN) wall = wall | 0x11;
    if (LS_SEN) wall = wall | 0x88;
    if (RS_SEN) wall = wall | 0x22;
    /* マウスの進行方向にあわせてセンサデータを移動し壁データとする */
    if (head == 1) wall = wall >> 3;
    else if (head == 2) wall = wall >> 2;
    else if (head == 3) wall = wall >> 1;

    /* 探索済みフラグを立てる */
    return(wall | 0xf0);
}

```

```

}

/*****
*/
/*          ハンドラ
*/
/*****
*/

void mot_cyclic_handler(VP_INT exinf) //1ms ごとに割り込み発生
{
    wait_l--;
    wait_r--;
    if(wait_l == 0) iwup_tsk(MOT_L_TASK);
    if(wait_r == 0) iwup_tsk(MOT_R_TASK);
}

void shusei_cyclic_handler(VP_INT exinf) //5ms ごとにセンサで軌道修正
{
    if(shusei_mode == 0) iwup_tsk(SHUSEI_TASK);
}
void shusei_cyclic_handler2(VP_INT exinf) //20ms ごとにセンサで軌道修正
{
    if(shusei_mode == 1) iwup_tsk(SHUSEI_TASK);
}
void led_cyclic_handler(VP_INT exinf) //500ms ごとに LED 点滅
{
    LED1 = ~LED1;
    LED2 = ~LED2;
    BUZZER = ~BUZZER;
}
/*
 * メインタスク
 */
void main_task(VP_INT exinf)
{
    //////////////////////////////////////

    volatile unsigned int counter;
    volatile int i;
    Init();

    for(;!SW1); //スイッチ 1 押したらスタート
    LED1 = 1; LED2 = 0; //LED 赤のみ点灯
    for(i = 0; i < 10; i++){
        for(counter = 0; counter < 30000; counter++);
        LED1 = ~LED1; LED2 = ~LED2;
    }
}

```

```
LED1 = LED2 = 0;           //消灯
////////////////////

/*
 * タスクの起動
 */

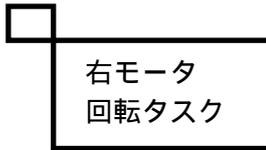
act_tsk(SHUSEI_TASK);
act_tsk(MOT_L_TASK);
act_tsk(MOT_R_TASK);

act_tsk(TANSAKU_TASK1); //以下の5つのタスクは順番に起動される(セマフォ
                        //を利用している)
act_tsk(CHALLENGE_TASK1);
act_tsk(TANSAKU_TASK2);
act_tsk(CHALLENGE_TASK2);
act_tsk(CHALLENGE_TASK3);

slp_tsk(); //main_task を眠らせると優先度の低いタスクが動き始める

syslog(LOG_NOTICE, "Sample task ends.");
kernel_exit();
}
```

解説



タスク処理は通常の間数と同じように記述する。

典型的な書き方

```
void task(){
    (やりたい処理)
    ext_tsk();          //タスクの終了
}
```

典型的な書き方

```
void task(){
    while(1){          //無限ループ
        slp_tsk();    //タスクを自ら休止状態にする
        (やりたい処理)
    }
    ext_tsk();        //タスクの終了
}
```

書き方 は、一度しか行わない処理に適している。他のタスクから act_tsk()によって起動されると、すぐに処理が行われる。メモリは食わないが、起動まで時間がかかる。

書き方 は、繰り返し行う処理に適している。他のタスクから act_tsk()によって起動されると、無限ループに入り、休止状態になる。そして、他のタスクから wup_tsk()や iwup_tsk()によって起床させられると処理を行い、再び休止状態になる。この方法ならば敏速な処理ができるが、タスクを起動したままになるのでメモリを食う。

右モータ回転タスクは、1ms ごとの周期ハンドラ(周回ハンドラ、サイクリックハンドラ)のイベントから起床される。タスクが起動されるごとにモータドライブ素子とつながっている足からトグル出力が行われ、これによってモータは回転する。タスク内では出力の回数も数えており、ある回数まで数えると(つまり、ロボットがある位置までくると)、左右の壁の有無を調べるなどのイベントを起こす。そして、規定の距離分進んだところで周期ハンドラを停止し、イベントフラグに1をセットして他のタスクにその旨を伝える。

sample1.cfg

```
CRE_TSK(TURN_R_TASK, { TA_HLNG, (VP_INT) 6, turn_r_task, LOW_PRIORITY, STACK_SIZE, NULL });
```

sample1.c

```
void mot_r_task(VP_INT exinf)
{
    while(1){          //無限ループ
        slp_tsk();    //休止状態になる
        PA.DR.BIT.B6 = ~PA.DR.BIT.B6; //トグル出力
        wait_r = NORMAL_WAIT;
```

```

kukan_pulse--; //規定パルス数 1 減らす
(中略)
if(kukan_pulse == GO_STEP){ //あるパルス数まできたら壁データ
    サンプリング
        if(RSEN != 0 && RS_SEN == 1) RS_SEN = 1;
        else RS_SEN = 0;
        if(LSEN != 0 && LS_SEN == 1) LS_SEN = 1;
        else LS_SEN = 0;
    }
    (中略)
    if(kukan_pulse == 0){
        止
            stp_cyc(MOT_CYCHDR); //モータをまわすためのハンドラ停止
            (中略)
            set_flg(SHURYO_FLG,1); //終了を知らせるためイベントフラグに 1 を書き込む
        }
    }
}

```

左モータ回転タスクもほぼ同じであるため、説明は省略する。



軌道修正タスクはセンサの値を見て、右により過ぎていたら左モータを遅くし、左により過ぎていたら右モータを遅くする。



周期ハンドラは、1ms 単位で周期的なタイマ割り込みと同じようなイベントを起こすことができる。コンフィギュレーションファイルで定義した関数に定期的を起こしたい処理を書くだけですむ。

ここでは 1ms ごとの周期ハンドラを用いて左右のモータの制御を、5ms ごとと 20ms ごとの周期ハンドラを用いて左右の位置補正を行っている。(もともとすべて 5ms ごとに修正動作を行っていたが、振動が激しいため柱周辺では 5ms ごと、それ以外の区間では 20ms ごとの位置補正を行っている。)

sample1.cfg

```
CRE_CYC(MOT_CYCHDR, { TA_HLNG, 0, mot_cyclic_handler, 1, 0 });  
CRE_CYC(SHUSEI_CYCHDR, { TA_HLNG, 0, shusei_cyclic_handler, 5, 0 });
```

sample1.c

左右のモータ制御

```
void mot_cyclic_handler(VP_INT exinf) //1ms ごとに割り込み発生  
{  
    wait_l--;  
    wait_r--;  
    if(wait_l == 0) iwup_tsk(MOT_L_TASK);  
    if(wait_r == 0) iwup_tsk(MOT_R_TASK);  
}
```

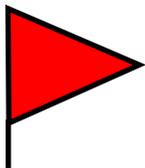
位置補正

```
void shusei_cyclic_handler(VP_INT exinf) //5ms ごとにセンサで軌道修正  
{  
    if(shusei_mode == 0) iwup_tsk(SHUSEI_TASK);  
}
```

周期ハンドラをスタートするには sta_cyc()、停止するには stp_cyc()を用いる。

(例)

```
sta_cyc(MOT_CYCHDR); //モータをまわすための周期ハンドラスタート  
stp_cyc(MOT_CYCHDR); //モータをまわすためのハンドラ停止
```



イベントフラグ

イベントフラグは、タスク同士の通信に用いられる手段の一つである。タスクはこのフラグが希望のパターンになるまで wai_flg() で待ち状態に入ることができる。いくつかのイベントフラグの「すべて」や「いずれか」を待つこともできる。

イベントフラグを作るときはコンフィギュレーションファイルに以下のように記述する。

sample1.cfg

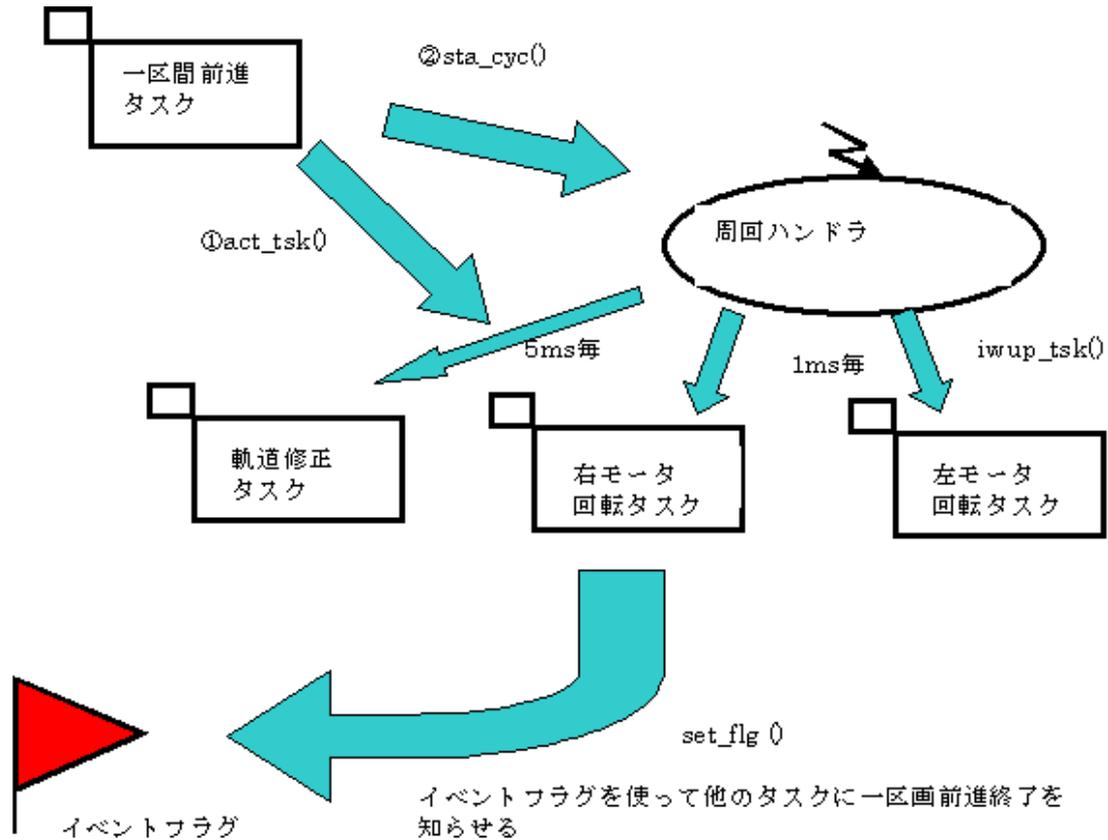
```
CRE_FLG(SHURYO_FLG, { TA_TFIFO | TA_CLR, 0});
```

フラグの書き込みや待ちは以下のようにする。

sample1.c

```
set_flg(SHURYO_FLG, 1); //終了を知らせるためイベントフラグに 1 を書き込む  
wai_flg(SHURYO_FLG, 1, TWF_ANDW, NULL); //イベントフラグに 1 が書き込まれるまで待ち状態に入る
```

一区画前進タスク



一区画前進タスクの構造は上のようになっている。

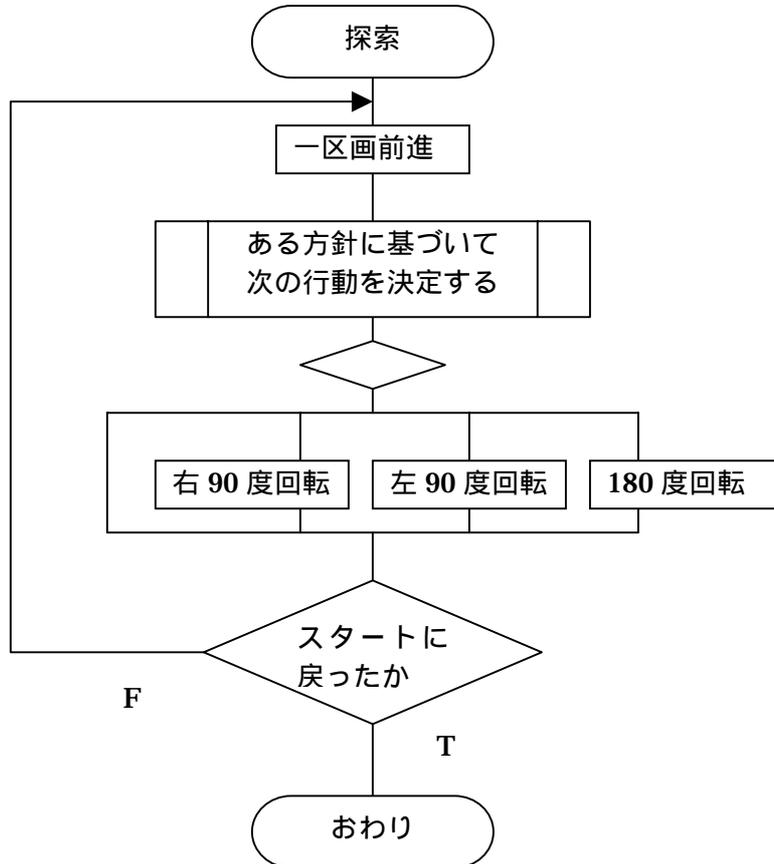
まず、一区画前進タスクは、モータを回転させるための周期ハンドラ（上図では周回ハンドラ、周期 1ms、左右それぞれ 1 つずつ）と、軌道修正をするための周期ハンドラ（周期 5ms）を起動する。各周期ハンドラは他のタスクに停止させられるまでこの後自動的に一定周期ごとにそれぞれのタスクを起床させ続ける。

左右のモータ回転タスクは、モータを回転させるパルスを出力し、その数を数え続ける（つまり前進した距離を測り続ける）。数がある値になると（一区画分の距離を前進したことがわかると）、イベントフラグによってほかのタスクに「一区画前進が終了しました」という旨を伝える。

右 90 度回転タスク・左 90 度回転タスク・180 度回転タスクは、軌道修正タスクを起動しない以外は一区画前進タスクとほぼ同じである。

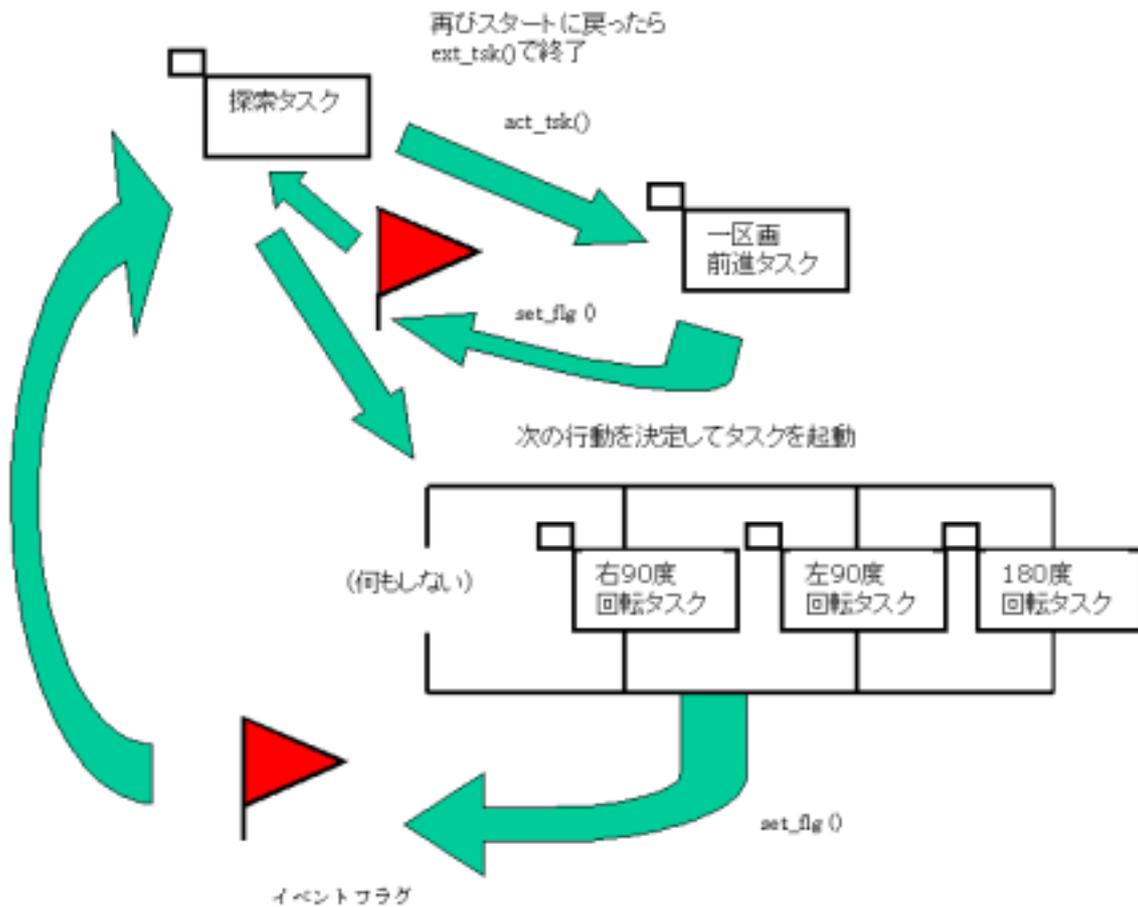
探索タスク

ここでは以下のような処理を行いたい。



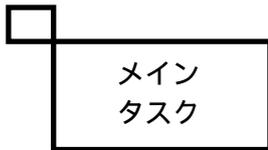
しかし、単純にこのような順番でタスクを起動しても、ただたくさんのタスクが起動されるだけで思うような結果を得ることはできない。あるタスクが終了した後に別のタスクを起動したいときは、タスクの終了を知る工夫が必要になる。そこで、ここではイベントフラグを用いる。(イベントフラグについてはすでに述べた)

タスク同士の関係を示したのが以下の図である。

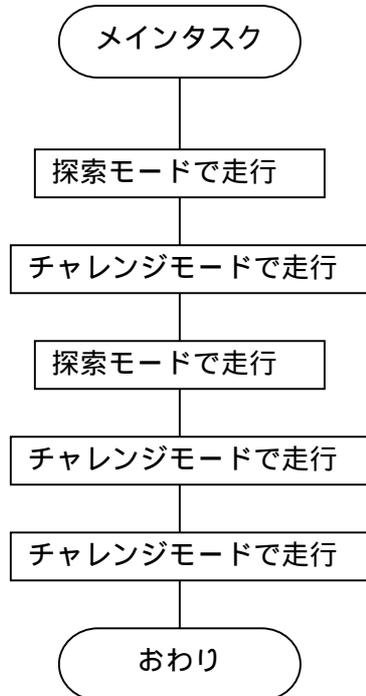


探索タスクはまず一区間前進タスクを起動し、イベントフラグ待ちに入る。一区間前進タスク（正確には一区画前進タスクから起動されたタスク）は、終了するときにイベントフラグをセットする。探索タスクはイベントフラグを見て一区画前進タスクの終了を知り、次の行動を決定した後それにあったタスクを起動し、再びイベントフラグ待ちに入る。それぞれのタスクは終了時にイベントフラグをセットする。これをくりかえす。スタート位置に戻ると探索タスクは終了する。

ロボットの走り方には探索モードとチャレンジモードの二つがあるが、チャレンジタスクは探索タスクとほぼ同じ構造なので説明は省略する。



メインタスクでは、マウスの五回のトライアルを順番に実行する。



単純にこのような順番でタスクを起動しても良いが、今回は練習のため排他制御を使うことにする。

イベントフラグを使っても良いが、ここではセマフォによる制御を考える。

上の状態は「5つのタスクがマウスという1つの資源を奪い合うために排他制御を行わなければならない状態」とも考えることができる。そこで、バイナリセマフォを使って排他制御を行う。

セマフォとは、棒にひっかけてある輪のようなものである。輪が棒にかかっているとき、資源は使用できる状態にある。輪がないときは輪が戻ってくるまで待つ。資源を使いたいときは、まず輪を棒から外して取り（P操作）、使用権利を獲得する。資源の使用が終了すると、輪を棒に戻す（V操作）。

各タスクは、はじめにセマフォ待ち状態にある。セマフォが使用できる状態になったら、セマフォに対しP操作を行い、使用が終わったらV操作を行う。起動順にセマフォの操作権を得るというモードにすると、メインタスクで起動した順番にセマフォを獲得することになる。モードは優先度順にすることもできる。

プログラム例は以下のようになる。

sample1.cfg

```
CRE_SEM(1, {TA_TFIFO, 1, 1}); /* マウス自身を排他制御するセマフォ */
```

P 操作と V 操作は以下のようにする。

sample1.c

```
wai_sem(MOUSE_SEM);
```

```
/* P 操作 */
```

```
sig_sem(MOUSE_SEM);
```

```
/* ID が 1 のセマフォを - 1、すでに 0 なら待
```

```
/* ID が 1 のセマフォを + 1 ( V 操作 ) */
```

4 . 割り込みハンドラ

3章までで実現したプログラムは、リアルタイム性があるプログラムとはいえない。タイマなどの「割り込み」を実現していないからである。モータを回転させることを考えると、滑らかな加速・減速の実現はタイマ割り込みを使用しないと難しい。

タイマ割り込みを使用するためにはその CPU 独自の設定を変更しなければならない。そのためコンフィギュレーションファイル・ヘッダファイル・C ソースファイル以外にもいじらなければならない場所があり、実現は難航した。以下に H8/3048F でタイマ割り込みを使うための方法を記す。

割り込みハンドラは最高優先度のタスクのようなものであり、実行中は他のすべてのタスクは停止する。そのため、処理時間はできるだけ短くする必要がある。時間がかかる処理は、タスクに任せて自らはすぐに終了しなければならない。

4 . 1 . TOPPERS 割り込みの登録の仕方

割り込みの使い方 (ITU0 (SYSTEM_TIMER) の IMIA0 をシステムが使っているのもそれを参考にした)

1. config¥h8¥akih8_3048f¥sys_support.S

```
.long _timer_handler_entry          /* 24, 0x0060: IMIA0    */
```

を参考にして 151 行目を

```
.long _imia1_handler_entry          /* 28, 0x0070: IMIA1    */
```

のように書き換えて登録した。こうすると imia1_handler という関数が割り込みのたびに呼ばれる。

2. config¥h8¥cpu_support.S を

218 行目に _imia1_handler (関数名) _enable_int を書く

```
/*
```

```
 * ハードウェア割り込み許可
```

```
*/
```

```
/*
```

```
 * タイマ -- 何もしない。
```

```
*/
```

```
.globl _imia1_handler_enable_int
```

```
_imia1_handler_enable_int:
```

```
    rts
```

を追加し、

261 行目に _imia1_handler (関数名) _disable_int を書く

```
/*
```

```
 * ハードウェア割り込み禁止
```

```
*/
```

```
/*
```

```
 * タイマ -- Compare Match フラグをクリアーする。
```

```
*/
```

```
.globl _imia1_handler_disable_int
```

```

_imia1_handler_disable_int:
    mov.l    #ITU1_TIMER_IFR, er0
    bclr    #ITU1_TIMER_IF_BIT, @er0
    rts

```

を追加する

3. ¥config¥h8¥akih8_3048f¥sys_config.h に#define で ITUなどを定義する。190 行目付近

```

#define SYSTEM_TIMER_STR  H8TSTR_STR0
#define SYSTEM_TIMER_STR_BIT    H8TSTR_STR0_BIT
#define SYSTEM_TIMER_IE          H8TIER_IMIEA          /* interrupt mask
*/
#define SYSTEM_TIMER_IE_BIT      H8TIER_IMIEA_BIT
#define SYSTEM_TIMER_IF          H8TSR_IMIFA          /* match flag */
#define SYSTEM_TIMER_IF_BIT      H8TSR_IMIFA_BIT

////////////////////////////////////ここから追加
#define ITU1_TIMER              H8ITU1

#define ITU1_TIMER_CNT          (ITU1_TIMER + H8TCNT)
#define ITU1_TIMER_TCR          (ITU1_TIMER + H8TCR)
#define ITU1_TIMER_TIOR         (ITU1_TIMER + H8TIOR)
#define ITU1_TIMER_IER          (ITU1_TIMER + H8TIER)
#define ITU1_TIMER_IFR          (ITU1_TIMER + H8TSR)
#define ITU1_TIMER_TSTR         H8ITU_TSTR
#define ITU1_TIMER_GR           (ITU1_TIMER + H8GRA)

#define ITU1_TIMER_STR          H8TSTR_STR0
#define ITU1_TIMER_STR_BIT      H8TSTR_STR0_BIT
#define ITU1_TIMER_IE           H8TIER_IMIEA          /* interrupt mask
*/
#define ITU1_TIMER_IE_BIT      H8TIER_IMIEA_BIT
#define ITU1_TIMER_IF           H8TSR_IMIFA          /* match flag */
#define ITU1_TIMER_IF_BIT      H8TSR_IMIFA_BIT

#define ITU1_TIMER_TCR_BIT(H8TCR_CCLR0      |      H8TCR_TPSC1      |
H8TCR_TPSC0)
#define ITU1_TIMER_TIOR_BIT          (0)

```

4. ¥obj (作業ディレクトリ) の.cfg ファイルに

```

ATT_INI({TA_HLNG, 0, itu1_timer_initialize});
DEF_INH(28,{TA_HLNG, imia1_handler});

```

を書く

5. sample.h の 141 行目付近に

```

extern void imia1_handler(void);

```

を書く

6.sample1.c に itu1_timer_initialize を定義する

```
void itu1_timer_initialize()
{
    UW addr = ITU1_TIMER_TSTR;

    /* タイマ停止 */
    /*outb(SYSTEM_TIMER_TSTR,          inb(SYSTEM_TIMER_TSTR)          &
~SYSTEM_TIMER_STR);*/

    #define BITCLR(bit)  Asm("bclr #" bit ", @%0" :: "r"(addr))
        BITCLR(_TO_STRING(ITU1_TIMER_STR_BIT));
    #undef BITCLR

    /* GRA コンペアマッチでカウンタをクリア、分周比設定 */
    outb(ITU1_TIMER_TCR, 0x21);

    /* GRA コンペアマッチによる割り込み要求を許可 */
    outb(ITU1_TIMER_IER, 0x1);

    /* GRA コンペアマッチによる端子出力禁止 */
    outb(ITU1_TIMER_TIOR, 0x0);

    /* GRA レジスタ設定 (カウンタ目標値) */
    outw(ITU1_TIMER_GR, 60000);

    outw(ITU1_TIMER_CNT, 0);          /* カウンタをクリア */
    /* GRA コンペアマッチの割り込み要求フラグをクリア */
    /*outb(SYSTEM_TIMER_IFR,          inb(SYSTEM_TIMER_IFR)          &
~SYSTEM_TIMER_IF);*/
    UW addr2 = ITU1_TIMER_IFR;
    #define BITCLR(bit)  Asm("bclr #" bit ", @%0" :: "r"(addr2))
        BITCLR(_TO_STRING(ITU1_TIMER_IF_BIT));
    #undef BITCLR

    /* タイマスタート */
    /*outb(SYSTEM_TIMER_TSTR,          inb(SYSTEM_TIMER_TSTR)          |
SYSTEM_TIMER_STR);*/

    #define BITSET(bit)  Asm("bset #" bit ", @%0" :: "r"(addr))
        BITSET(_TO_STRING(ITU1_TIMER_STR_BIT));
    #undef BITSET
}
```

レジスタ (ITU0.GRA など) に値を入れるときは outb か outw を使わないといけない
かもしれない

7. imia1_handler に割り込み処理を書く

```
void imia1_handler(void)
{
    割り込み処理
}
```

8. H8ITU_TSTR の 0x2 を入れると ITU1 の割り込みが有効になる。

注意することは TRON が ITU0 を使っているので勝手に有効にしたり、無効にしたりするとおかしくなる。

それなので H8ITU_TSTR に 0x2 を入れるには

H8ITU_TSTR |= 0x2

のようにする。

ITU1 に 0 を入れたければ

H8ITU_TSTR &= 0xd

のようにする。

5 . 他の CPU への移植

OS を使うメリットの一つとして、移植性が高まることがあげられる。ハードウェアの違いを OS が吸収してくれるため、プログラムの全面的な書き直しがいらぬからである。

ターゲットシステムとして、ここではゲームボーイアドバンスを使用することにする。

ゲームボーイアドバンスの開発風景



ゲームボーイアドバンス（以下 GBA）は CPU として ARM 7（ARM7TDMI）を使用している。TOPPERS（Ver.1.3）は ARM7TDMI を使用したテスト用ボード「Evaluator7t」に対応しているので、これを改造することによって GBA への対応も可能である。

以下は、GBA で三つのタスクを遷移させ、画面にメッセージを出力させている様子を示したものである。



5.1 開発環境の構築

開発環境の構築は、H8/3048F の場合とほぼ同じ感覚でできる。具体的には以下のようにする。

1. 以前のものを参考にして環境依存 (CONFIG ディレクトリ) 以外の所を直す。
2. `jsp/config/arm7tdmi` ディレクトリに `evaluator_7t` ディレクトリをコピーして、`gba` という名前をつける
3. `./cfg/fc_bfd.cpp` の 99 行目の
`boolean result;`
を
`bool result;`
に直す。c++では `boolean` は定義されていないので `bool` に直す。
4. `jsp/config/arm7tdmi/gba/sys_defs.h` の 46 行目を
`#define EVALUATOR_7T`
から
`#define GBA`
に変更。
5. `jsp/config/arm7tdmi/gba/sys_config.h` の 48 行目を
`#define TARGET_NAME"Evaluator-7T"`

から

```
#define TARGET_NAME "GBA"
```

に変更し、66 行目の

```
#define STACKTOP 0x00074000 /* 非タスクコンテキスト用のスタックの初期  
値 */
```

から

```
#define STACKTOP 0x203fff /* 非タスクコンテキスト用のスタックの初期  
値 */
```

に変更。

6 . jsp/config/arm7tdmi/gba/Makefile.config の 17・18 行目の

```
TEXT_START_ADDRESS=0x00008000
```

```
DATA_START_ADDRESS=0x0003A000
```

を

```
TEXT_START_ADDRESS=0x02000000
```

```
DATA_START_ADDRESS=0x02010000
```

に変更。

7 . jsp/config/arm7tdmi/start.S の 116・117 行目をコメントアウトする

8 .後は configure や makefile の書き換えをして GBA の点を表示などのソースを main に書いて make などをして転送すると実行できる。

ただこの方法だとタイマ割り込み等の移植はうまくいかず、タイマを使う機能（周期ハンドラなど）を使うことはできなかった。しかし、2人とも ARM / GBA のプログラム開発は未経験であったにもかかわらず、マルチタスクプログラムの開発が2週間あまりでできるようになった。そのほとんどが GNU 開発環境などの構築に費やされたことを考えると、はじめから OS を使った開発環境が与えられた場合の作業効率はきわめてよいだろうと考えられる。

6 . おわりに

当初の予定では前期のうちに OS を乗せる予定だったが、できなかった。前期の前半のロボット製作までは予定通りに進めることができた。その後メモリの増設から遅れ始め、OS を乗せるのは大幅に遅れ後期の前半ぐらまでかかった。遅れた理由としては、時間の見積もりの甘さと経験不足である。まず見積もりの甘さというのは、具体的なスケジュールを立てずおおざっぱにしか立てていないということ。次の経験不足というのは、そのまま経験不足ということで、例えば Cygwin を使ったが、その使い方は linux 風で今までそれほど linux を使っていなかったのだからそこらつまづいた。作業を進めるとスタートアップルーチンというのが必要で、環境によって一部書き換えないといけないことがわかった。スケジュール見積もりにスタートアップルーチンは含まれておらず、cygwin の操作の習得も含まれていなかった。そういう意味では見積もりの甘さとも言えるが、経験がなかったので見積もることができなかったとも言える。これからは経験のないものは余裕を持ってスケジュールを立てることにしたい。

H8 と arm という 2 種類の CPU に TOPPERS を乗せることで、いろいろわかったことがあった。1 つ目は環境に依存する部分と環境に依存しない部分がわかった。具体的には環境に依存する部分は/jsp/config にまとめられている。その下に CPU 名のディレクトリがあり、その更に下にシステム名のディレクトリがあり、それぞれ CPU 依存の部分と、システム依存の部分定義されている。ドキュメントにもそう書いてあったが、最初はよくわからなかったが、実際に手を動かしてみたら初めてわかった。arm の場合は、CPU は同じでシステムだけ違ったので、システム部分だけ書き換えることで動かすことができた。それと 2 つのシステム共通で書き換えないといけない所があったが、それは環境に依存せずおそらく他の CPU でも書き換えないといけないだろう。逆に 2 つのシステム共通でない所は CPU 固有で書き換えないといけない所と考えられるので、他の CPU を使えば違うように書き換えないといけないと考えられる。

環境構築に予想より大幅に時間がかかり TOPPERS を十分に使う時間がなかった。それなので OS を使ったそれらしいプログラムを書くことができなかった。今のところ一応動いているという感じで OS をつんだメリットがあまりない。

講義などによって OS について知識としてだけ知っていたことを実感することができた。例えば、タスクの遷移の実行状態や実行待ち状態などが実際に動いている所が見れた。それに付随して、Linux の使い方や、タスクや C、リンクスクリプトとは何かといったことについて理解が深まった。

感想としては、組み込みシステムについての知識があまりなく、概念を理解するのに時間がかかった。例えばクロスコンパイラで、PC 用のコンパイラとターゲットシステム用のコンパイラがあって、どのようにして使い分けしているのかわからなかった。実際には実行ファイル名の前にターゲット名がつくからそれで判断できる。他にもスタートアップルーチンはその存在自体知らなかった。それなのでそれが何をするものなのかもわかっていなかった。今までは知らなかっただけで、PC でもスタートアップルーチンは必要で、自動的に生成していた。普段 PC を使ってプログラミングしているときは自動的に処理されて意識することがないことが、PC より低いレベルで機器を制御しないといけないことでプログラムを始める前の準備が増えた。

今回は H8 プロセッサについて主に研究したが、他の CPU についてももっと調べてみたかった。また、RT-Linux など、他の OS も試してみたかった。

7 . 参考文献

H8

H8 マイコン完全マニュアル (藤沢幸穂、オーム社)
‘99・’00年技術交流の記録 (マイクロマウス委員会東日本支部)
初心者のためのロボットの作り方第六版 (マイクロマウス委員会東日本支部)
マイコン技術教科書 H8 編 (今野金顕、CQ 出版社)
H8 ビギナーズガイド (電気通信大学出版局)

リアルタイム OS

μITRON4.0 標準ガイドブック (パーソナルメディア社)
実用 組込み OS 構築技法 情報通信を支える基礎技術 RTOS 入門 (永井 正武 (著), 権藤
正樹 (著), 沢田 勉 (著))
組込みネット <http://www.kumikomi.net/>
組込み Linux 入門 (TECH I 16号、CQ 出版)
FreeBSD Press14・15・16号
Design Wave 2003年4月号・5月号
HI シリーズテキスト (日立半導体セミナー)
Embedded UNIX 1 ~ 3 (CQ 出版)
TOPPERS 付属ドキュメント
リアルタイム / マルチタスクシステムの徹底研究 (TECH I 15号、CQ 出版)
リアルタイム OS と組込み技術の基礎 (TECH I 17号、CQ 出版)

ゲームボーイアドバンスと ARM プロセッサ

Linux から目覚めるぼくらのゲームボーイ! (西田互、SOFTBANK)
ARM プロセッサ入門 (TECH I 18号、CQ 出版)
きまぐれ遊戯少年進歩 <http://vsync.org/agb/>
http://users.kyoto-kcg.ac.jp/~r00h0646/cincs/irukaichigou_advance/address_map.html

マイクロマウス

マイクロマウス工房 - 森永 - <http://www02.so-net.ne.jp/~morinaga/>
全日本マイクロマウス大会 <http://www.bekkoame.ne.jp/~ntf/mouse/taikai/taikai.html>