

## §2-I-3 レゴマインドストームの C 言語による制御（基礎）

(C language control of LEGO Mindstorms [Basic])

### 1. 実験の狙い

本実験では 1 年次で実施したメカトロニクスの実験で行ったレゴマインドストームの制御を C 言語ベースで行う。C 言語は 1 年次の C プログラミングで学んでいることを前提としているため、よく復習しておくこと。1 年次の実験ではビジュアルベースのプログラムを行っていたが、2 年次は C 言語を用いて機器を制御することで、文字ベースのプログラムによる機器制御の流れを習得する。

### 2. 解説

#### レゴマインドストーム EV3

本実験で使用するレゴマインドストーム EV3（以下 EV3）は、プロセッサとして ARM9 300 MHz が採用されており、3 色に光る 6 ボタン、モノクロディスプレイ、スピーカー、USB ポート、ミニ SD カードスロット、4 つの入力ポートと 4 つの出力ポートが内蔵されている（図 1）。また、1,000 回/秒のセンサ通信性能と最大 1,000 サンプル/秒のデータロギング性能を持つ。



図 1 インテリジェントブロック EV3

#### Cygwin

Cygwin は、Windows 上で UNIX ライクな安協を必要とするユーザのために開発されたソフトウェアである。Cygwin の Web サイトのトップにある「GNU+Cygnus+Windows=Cygwin」という式と、続く「Cygwin is a Linux-like environment for Windows」という文が示すとおり、GNU プロジェクトの成果物を利用し、Windows プラットフォーム上に Linux 風の操作環境を実現することを目的として、開始されたオープンソースの開発プロジェクトである。そのため、Cygwin は Windows に代わるシステムとはならず、Windows のシステムに上乗せする形で実装されているため、ハードウェアへのアクセスは Windows の機能（Win32 API）を経由しているなど、Windows とは不可分の存在である。UNIX のシステムコールを Windows の機能を利用してエミュレートしているため、

Linux や FreeBSD などのように独立して起動する OS と比較するとパフォーマンス面で不利なほか、システムの安定性が Windows に依存するという二重構造ゆえの問題点も抱えているが、PC-UNIX の機能を利用したいが Windows で作業する時間のほうが圧倒的に多い、もっと手軽に PC-UNIX の機能を使いたい、というユーザに適している。

Cygwin では UNIX 系 OS で広く利用されている開発ツールが提供されており、本実験では C/C++コンパイラである「gcc」を利用してソフトウェアの作成を行う。

### タスク処理

タスクとは、プログラムを実行する際の処理のことである。Toppers-C 環境では、起動された後、処理を最後まで実行したら終了するタスクと、一定時間ごとに呼び出されて処理を実行する周期タスクが用意されている。手順 2, 3 のプログラムでは `main_task()` と `run_task()` というタスクを使用し、プログラム起動と同時に `main_task()` が起動し、その中で `run_task()` を起動している。どちらのタスクも処理を最後まで実行したらタスクが終了するプログラムである（図 2）。

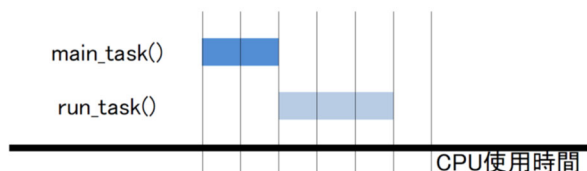


図2 プログラムの実行イメージ

Toppers-C 環境では、プログラムの処理をマルチタスクで実行している。このため、処理時間を非常に短い単位に分割し、複数の処理を並行して行うことが可能である。この時、タスクに優先度を設定し、優先度の高いものから処理を行っていく（図3）。

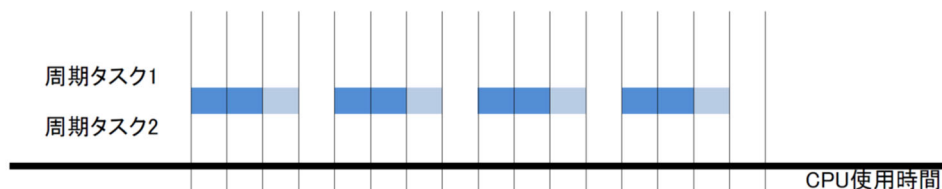


図3 マルチタスクの実行イメージ

周期タスクは、コンフィギュレーションファイルでタスクを生成する際に指定した周期ごとに起動して処理を実行する。他のタスク、または周期タスクがある場合は、マルチタスクで処理される（図4）。周期タスクは一定周期ごとに起動されるので、起動されてから次の起動までに処理が終了しないような重い処理（または短い周期）にしていると次に実行する処理の起動が遅れることになってしまい、プログラムが破綻する。

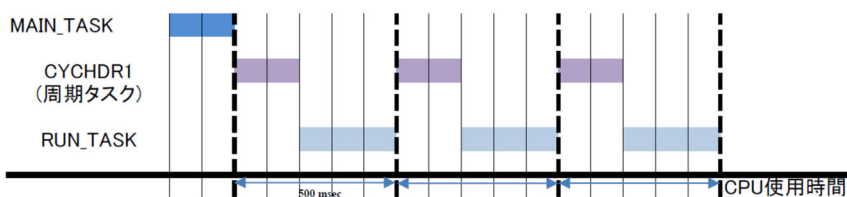


図4 周期タスクの実行イメージ

### 3. 実験の予習 \*予習レポートとして整理して提出すること

予習1： 下記の用語について，意味を調査し理解を進めておく

黒字は必ず．その他は余裕があれば．

- |           |                   |
|-----------|-------------------|
| a. C 言語   | b. レゴマインドストーム EV3 |
| c. Linux  | d. UNIX           |
| e. Cygwin | f. gcc            |
| g. GNU    | h. IP アドレス        |
| i. WiFi   | j. サーボモータ         |

予習2： C 言語に関して必ず復習しておくこと．

予習3： 実験で作成するプログラムのフローチャートを作成してくること．

1 年次のプログラムビジュアルプログラミング言語であったため作成したプログラムの流れが視覚的にわかりやすかったが，本実験では文字ベースであるためプログラムの流れを理解するためにも必ず作成して来ること．

※作成するプログラムは別資料を参照すること．

予習4： Color センサの実験ではインテリジェントブロック上の LCD パネルの表示がおかしくなる可能性がある．その理由と解決策を考えること．

ヒント: `sprintf` の処理がどのようなものかがわかれば解決する．

予習5： モータ制御の実験におけるコードを考えてくること．

※資料には回転ではなく直線に進むコードしか書かれていない．

## 4. 実験

### 4.1 実験の概要

- LED と LCD, サウンドの制御
- カラーセンサの閾値の確認
- サーボモータの制御
- タスク処理

### 4.2 実験で使用する機材

- ・ パーソナルコンピュータ
- ・ レゴマインドストーム(EV3)
- ・ microSD カード

**実験室の PC には（各自が持参した）USB を接続しないこ**

**と**

### 4.3 実験終了後の確認事項

実験終了後 TA に以下の項目を確認してもらうこと.

- 実験データ（カラーセンサ, サーボモータ）がまとめられているか.
- workspace 内に不要なファイルが残っていないこと.

実験データは各自が Moodle にて管理すること.

## 4.4 実験手順

### [1] 実験の進め方

～14:00	手順 1：LED の点灯制御 Cygwin の操作になれること。
～14:45	手順 2：カラーセンサの閾値の確認 カラーセンサからの値を取得するプログラムを作成する。 モード（色認識，反射光，周辺光）によりどのような値が得られるか確認する。
～15:30	手順 3：サーボモータの制御 サーボモータを制御するプログラムを作成する。 サーボモータをどのように制御すれば任意の角度に方向転換できるか調査する。
～16:00	手順 4：タスク処理 手順 2 のプログラムを基にプログラムを作成する。 センサ値をとりながら音が鳴ることを確認する。
～16:25	手順 4：TA による試問

## **5. 実験レポート**

### **検討すべき事項（考察事項）**

実験レポートでは、実験結果を踏まえて検討を行い、下記の項目に関してもよく考察をすること。

- ① カラーセンサの測定結果とモードに関して考察せよ。  
（どのような場合に用いると良いかや測定精度を上げる工夫など）
- ② サーボモータの制御結果に関して考察せよ。  
今回の制御結果より場面に応じてどのような制御が可能か？ 等

### **さらに調査すること**

次回は本実験をベースに行うため、しっかりと理解した上で予習を行ってこよう。

### **課題**

課題1： レゴマインドストーム(EV3)に付属している各種センサの仕様・特徴などを調べる。

※実験に使用していないものも調べてまとめよ。

課題2： ステッピングモータとサーボモータの違いについて調べよ。

課題3： タスク処理はどのようなところで利用されるか。例を挙げて説明せよ。

課題4： 各プログラムのフローチャートを修正せよ。

※事前課題で行っているため、修正がある場合は修正し、修正がない場合はそのままよい。

※修正を行う場合は予習課題のものを修正すること。

## **5. 参考文献**

- ・ レゴマインドストーム教本
- ・ 海上 忍 著「Windows で UNIX 環境を実現! 今すぐ使える」（技術評論社）

## LEGO EV3 の取り扱い概要

本実験では Windows 上で Linux 風の操作環境を実現した Cygwin を利用して、LEGO EV3 のプログラムの作成を行う。プログラムのコンパイルには、ARM コアで動作するプログラムをコンパイルするための、ARM 用の C コンパイラ（GCC ARM）を利用する。また、ブートローダから実行されるプログラムの実行ファイルを作成するため、U-Boot を利用する。

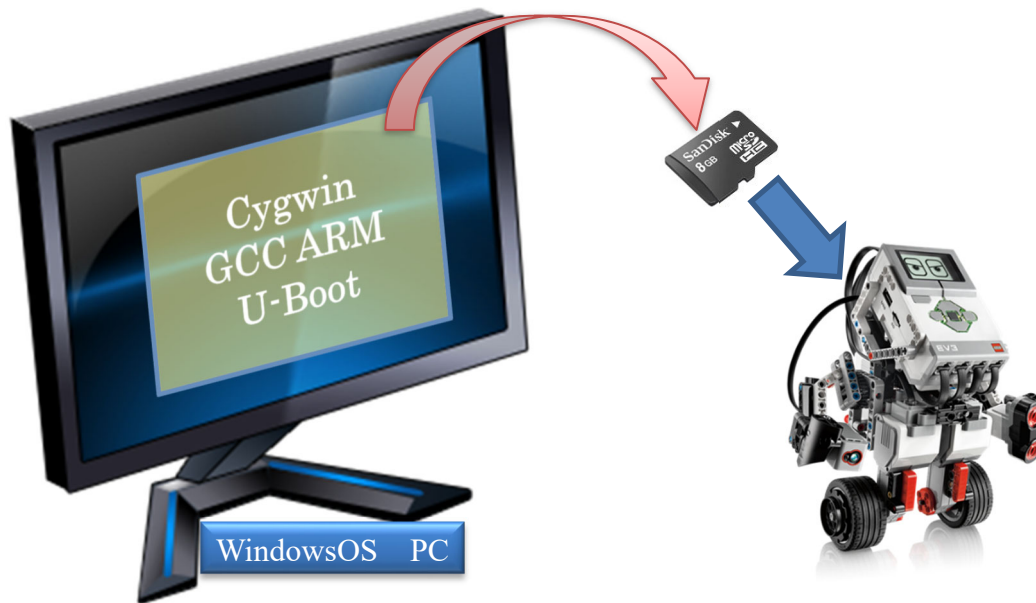


図 5 使用機器の構成

EV3 上で動作するプログラムを作成するため、以下の手順で操作を行う。

1. Windows PC の立ち上げ
2. プログラムの作成（テキストエディタ上）
3. プログラムのコンパイル（Cygwin 上）
4. microSD にプログラム（app）をコピーし、EV3（電源 OFF）に microSD を差し込む。
5. EV3 の電源を入れ、プログラムを確認する。

## 下準備

- Windows PC を立ち上げてログインする.

## プログラム作成の流れ

- ① デスクトップ上の「2 年実験前期」フォルダ内にあるショートカット（workspace）を開く  
(ア) workspace はプログラムを保存するフォルダ  
ディレクトリ構造：C:\Cygwin64\hrp2\workspace\<プログラムディレクトリ>  
workspace 内部の以下のファイル・フォルダは消去しないこと
  - ① common [フォルダ]
  - ② gyroboy [フォルダ]
  - ③ OBJ [フォルダ]
  - ④ Makefile [ファイル]
- ② workspace 直下に作成するプログラムのフォルダを作成（半角英数字）
- ③ フォルダ内に以下のファイルを作成  
(ア) app.c：アプリケーションのデフォルトのソースファイル  
(イ) app.h：アプリケーションのデフォルトのヘッダファイル  
(ウ) app.cfg：アプリケーションのデフォルトのコンフィグレーションファイル
- ④ workspace 内のフォルダ<gyroboy>から以下のファイルをコピー  
(ア) Makefile.inc：全ての実行形式で共有するメイクファイル
- ⑤ ソースファイル，ヘッダファイル，コンフィグレーションファイル作成後 Cygwin を起動
- ⑥ プログラムをコンパイル（詳細は後述）
- ⑦ workspace 内に作成された app を microSD カードに保存

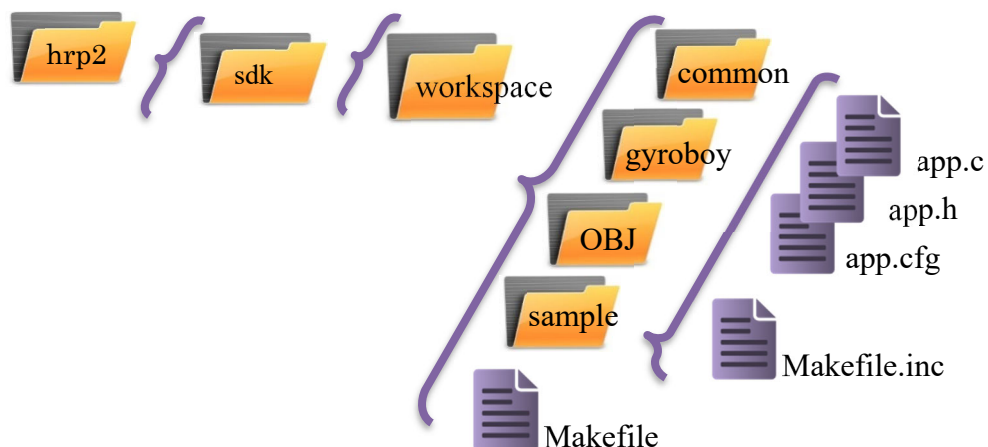


図 6 workspace のディレクトリ構造



## Cygwin を使用して app を作成する

ブートイメージを作成する流れを実験項目 1 LED の点灯制御を例にして説明する。

I-1. フォルダ、ファイルの作成、必要なファイルのコピー

I-2. Cygwin の起動

I-3. コンパイル（app の作成）

I-4. micorUSB に app をコピー，EV3 上でファイル実行

### I-1. フォルダ、ファイルの作成、ファイルのコピー

1. workspace 内にフォルダを作成する．実験班が火曜 03 班の場合，“T03\_LED”というフォルダを作成する．
2. フォルダ内に 3 つのファイル（app.c, app.h, app.cfg）を作成する．ファイルはテキストファイルを新規で作成し，拡張子を変更する．  
※1 それぞれのファイルのコードは，後述する実験項目に記載されている．  
※2 ファイル名は基本的に変更しないこと．
3. フォルダ<gyroboy>から 3 つのファイル（Makefile.inc）をコピー
4. 図 7 のような構成になっていることを確認する．

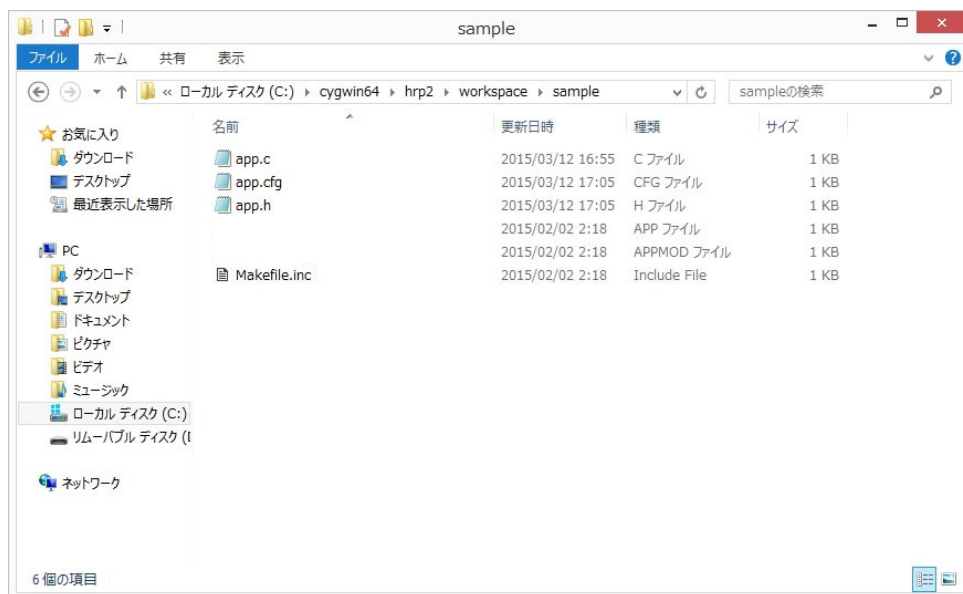



図 7 作成したフォルダの例

実験番号：2-I-3

実験タイトル：レゴマインドストームの C 言語による制御（基礎）

## I-2. Cygwin の起動

デスクトップ上の「2 年実験前期」フォルダ内にある Cygwin のアイコンをダブルクリックし、Cygwin を起動する。Cygwin が起動すると図 8 のような画面になる。

※ Cygwin のショートカットは「Cygwin64bit」となっている。



図 8 Cygwin の起動画面

## I-3. コンパイル（app の作成）

1. workspace ディレクトリに移動する。（図 9）

コマンド：cd /hrp2/sdk/workspace

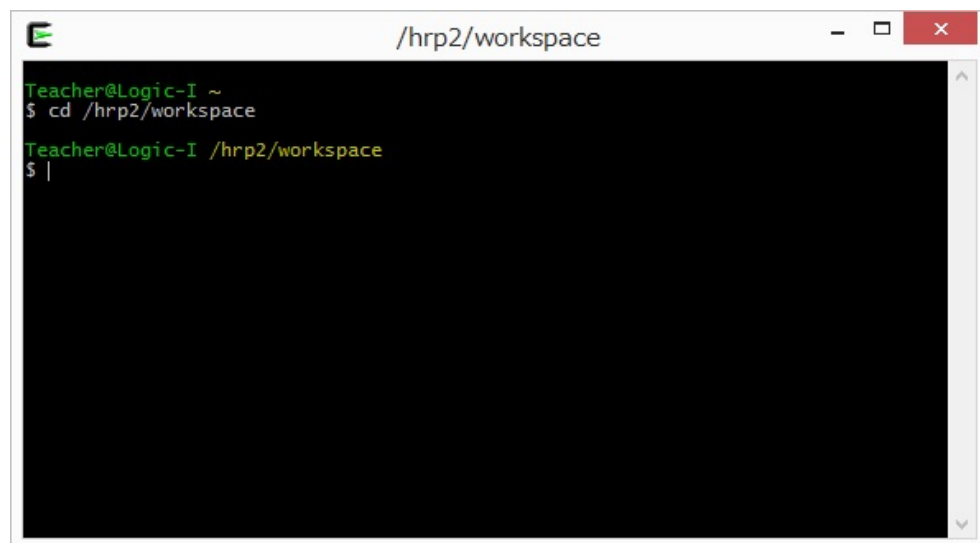


図 9 ディレクトリの移動

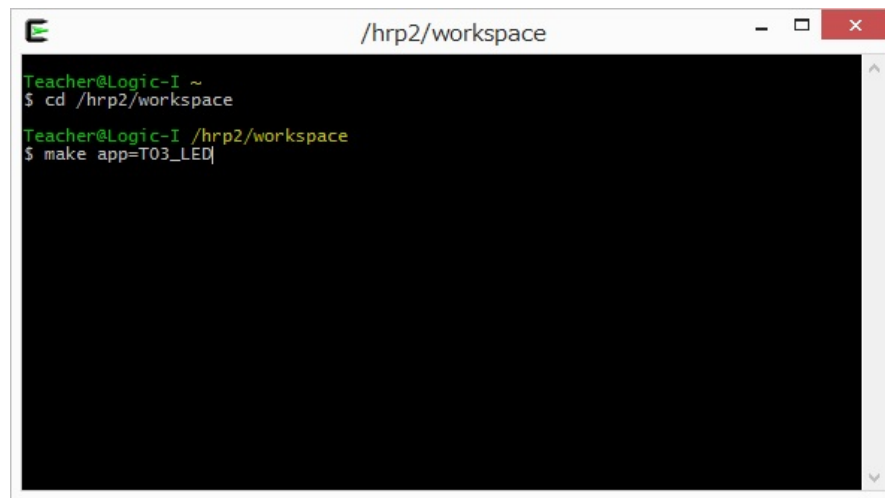
実験番号：2-I-3

実験タイトル：レゴマインドストームの C 言語による制御（基礎）

2. ビルドする．（図 10）

コマンド：make app=[プログラムのディレクトリ名]

例) make app=T03\_LED

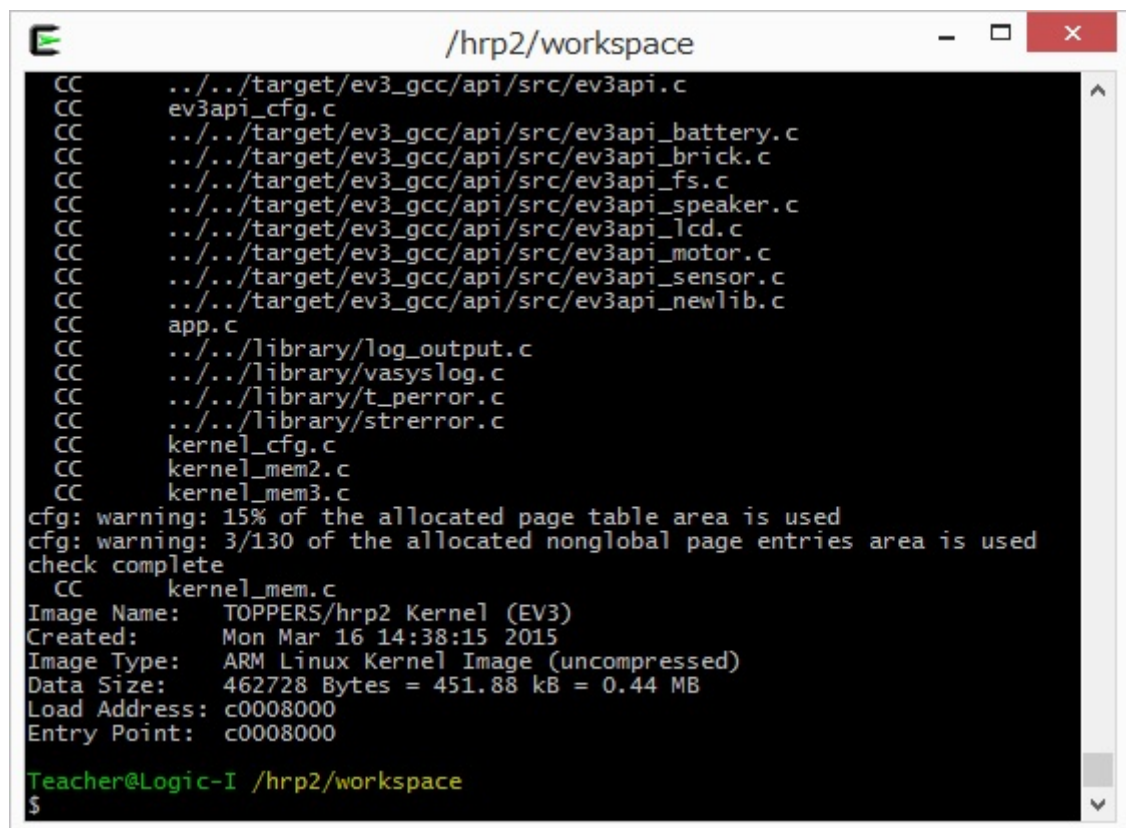
A terminal window titled "/hrp2/workspace" showing the execution of the 'make' command. The prompt is "Teacher@Logic-I ~". The user enters "\$ cd /hrp2/workspace". The prompt changes to "Teacher@Logic-I /hrp2/workspace". The user enters "\$ make app=T03\_LED".

```
Teacher@Logic-I ~  
$ cd /hrp2/workspace  
Teacher@Logic-I /hrp2/workspace  
$ make app=T03_LED
```

図 10 プログラムのビルド

3. 実行画面の確認

図 11 のような画面が出ればビルド成功.

A terminal window titled "/hrp2/workspace" showing the output of the 'make' command. The output lists various source files being compiled, followed by warnings about memory usage, and finally the creation of a kernel image. The prompt returns to "Teacher@Logic-I /hrp2/workspace".

```
CC      ../../target/ev3_gcc/api/src/ev3api.c  
CC      ev3api_cfg.c  
CC      ../../target/ev3_gcc/api/src/ev3api_battery.c  
CC      ../../target/ev3_gcc/api/src/ev3api_brick.c  
CC      ../../target/ev3_gcc/api/src/ev3api_fs.c  
CC      ../../target/ev3_gcc/api/src/ev3api_speaker.c  
CC      ../../target/ev3_gcc/api/src/ev3api_lcd.c  
CC      ../../target/ev3_gcc/api/src/ev3api_motor.c  
CC      ../../target/ev3_gcc/api/src/ev3api_sensor.c  
CC      ../../target/ev3_gcc/api/src/ev3api_newlib.c  
CC      app.c  
CC      ../../library/log_output.c  
CC      ../../library/vasyslog.c  
CC      ../../library/t_perror.c  
CC      ../../library/strerror.c  
CC      kernel_cfg.c  
CC      kernel_mem2.c  
CC      kernel_mem3.c  
cfg: warning: 15% of the allocated page table area is used  
cfg: warning: 3/130 of the allocated nonglobal page entries area is used  
check complete  
CC      kernel_mem.c  
Image Name:  TOPPERS/hrp2 Kernel (EV3)  
Created:     Mon Mar 16 14:38:15 2015  
Image Type:  ARM Linux Kernel Image (uncompressed)  
Data Size:   462728 Bytes = 451.88 kB = 0.44 MB  
Load Address: c0008000  
Entry Point: c0008000  
Teacher@Logic-I /hrp2/workspace  
$
```

図 11 ビルド成功画面

#### I - 4. micorUSB に app をコピー，EV3 上でファイル実行

workspace 内に作成された app を microSD にコピーする．

※app は常に上書きされるため，タイムスタンプなどで新しく作られたファイルであるかを確認すること．

microSD 内の階層を以下のように構築すること．uImage がいない場合は TA にもらうこと．

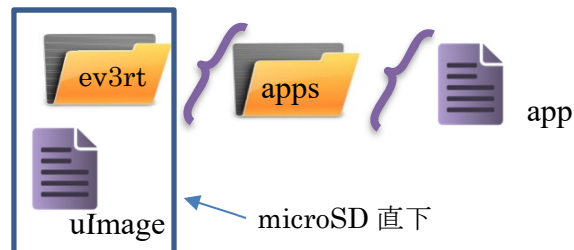


Fig. 12 microSD カード内の階層構造

EV3 の電源が切れていることを確認し，microSD を EV3 に挿入する．EV3 の電源を入れ正しい動作をするか確認する．

- 注意 -

#### EV3 の電源のプログラムの終了方法（電源を落とす方法）

EV3 の戻るボタン，左ボタン，右ボタンを同時に押す．



図 13 インテリジェントブロック EV3 のボタン

#### センサポートに関して

センサーポート 1: タッチセンサのみ使用可能

センサーポート 2～4: すべてのセンサを利用可能

## 1. 実験項目

- ① LED と LCD, サウンドの制御  
プログラムを作成し，実行し動作を確認する．
- ② カラーセンサの閾値の確認  
プログラムを作成し，各色パターンの読み取り値を確認する．
- ③ サーボモータの制御  
プログラムを作成し，任意の移動・回転運動制御が行えるようにする．
- ④ タスク処理  
プログラムを作成し，実行し動作を確認する．

## コード作成時の注意事項

- コードの最後に改行を入れておく．
- 全角スペースは利用しない．  
※コメントアウト内も利用しないこと

## ① LED と LCD，サウンドの灯制御

### <説明>

インテリジェントブロック上にあるボタンを利用して，LED と LCD，サウンドの制御を行う．

① 左ボタン：LED 赤，LCD：「Left Pressed」を表示，サウンド NOTE\_C4 を再生

② 右ボタン：LED 橙，LCD：「Right Pressed」を表示，サウンド NOTE\_D4 を再生

### <実験課題>

ボタンが押された際に，命令通りに LED と LCD の表示が変わることを確認。

app.c

```
/* LED, Button, LCD制御 */

#include "ev3api.h"
#include "app.h"

#ifdef BUILD_MODULE
#include "module_cfg.h"
#else
#include "kernel_cfg.h"
#endif

static void button_clicked_handler(intptr_t button) {
    switch (button) {
        case LEFT_BUTTON:
            ev3_led_set_color( LED_RED );
            ev3_speaker_play_tone(NOTE_C4, 500);
            ev3_lcd_draw_string("Left Pressed", 0, 0);
            tslp_tsk(500);
            break;
        case RIGHT_BUTTON:
            ev3_led_set_color( LED_ORANGE );
            ev3_speaker_play_tone(NOTE_D4, 500);
            ev3_lcd_draw_string("Right Pressed", 0, 0);
            tslp_tsk(500);
            break;
        default:
            break;
    }
}

void main_task(intptr_t unused) {
    /* 本体ボタンの設定 */
    ev3_button_set_on_clicked(LEFT_BUTTON, button_clicked_handler, LEFT_BUTTON);
    ev3_button_set_on_clicked(RIGHT_BUTTON, button_clicked_handler, RIGHT_BUTTON);
    ev3_speaker_set_volume(10);
    ev3_led_set_color( LED_OFF );
}
```



最後に改行を入れること

## app.h

```
#include "target_test.h"

#ifndef STACK_SIZE
#define STACK_SIZE          4096          /* タスクのスタックサイズ */
#endif /* STACK_SIZE */

/* 関数のプロトタイプ宣言 */
#ifndef TOPPERS_MACRO_ONLY
extern void    main_task(intptr_t exinf);
#endif /* TOPPERS_MACRO_ONLY */
```

 最後に改行を入れること

## app.cfg

```
INCLUDE("app_common.cfg");

#include "app.h"

DOMAIN(TDOM_APP) {
  CRE_TSK(MAIN_TASK, { TA_ACT, 0, main_task, TMIN_APP_TPRI + 1, STACK_SIZE, NULL });
}

ATT_MOD("app.o");
```

 最後に改行を入れること

## LED の色

LED_OFF	LED を消灯
LED_RED	赤色
LED_GREEN	緑色
LED_ORANGE	橙色

## 本体のボタン

LEFT_BUTTON	左ボタン
RIGHT_BUTTON	右ボタン
UP_BUTTON	上ボタン
DOWN_BUTTON	下ボタン
ENTER_BUTTON	中央ボタン
BACK_BUTTON	戻るボタン
TNUM_BUTTON	ボタン数の最大値

## 使用した API

ev3_led_set_color(A)	LED を点灯・消灯させる． A に LED の色
ev3_button_set_on_clicked(A, B, C)	ボタン入力を受け付けるようにする． A, C: ボタンの種類, B: 呼び出される関数
ev3_speaker_set_volume(A)	サウンドの大きさ設定． A: 0 ～ 100
ev3_speaker_play_tone(A, B)	周波数と時間を指定してサウンド出力． A: 周波数 or 定数, B: 時間 [msec]
ev3_lcd_draw_string(A, x, y)	指定位置で文字列を描く． A: 文字列, x, y: 位置

## ② カラーセンサの閾値の確認

<説明>

EV3 に接続されたカラーセンサ値を読み取るプログラムを作成する。EV3 のカラーセンサは、インテリジェントブロック EV3 と UART 接続されている。専用のコマンドをカラーセンサに送ることにより、カラーセンサ値を取得している。カラーセンサには、3 種類のモードが存在する。REFLECT モードは反射光の強さを測定し、COLOR モードは対象の色を判別し、AMBIENT モードでは周囲の光の強さを測定する。測定されたデータはインテリジェントブロックの LCD 部分に表示され、Text ファイルに出力される。

<実験課題>

本項目では図 14 に示すようなカラーマップ(光沢紙と布で印刷されたもの)を 3 種類のモードを同時に測定する。測定されたデータは表などにまとめること。

app.c

```
/* カラーセンサの制御 */

#include "ev3api.h"
#include "app.h"

#ifdef BUILD_MODULE
#include "module_cfg.h"
#else
#include "kernel_cfg.h"
#endif

#define color_sensor    EV3_PORT_3    /* センサーのポートを設定 */

const uint32_t WAIT_TIME_MS = 100;    /* 待機時間の設定 [msec] */

void run_task(intptr_t unused) {
    int now_val = 0;
    int now_value = 0;
    char str[256], colorname[256];
    colorid_t now_color;
    FILE *fp;

    while (1) {
        fp = fopen("data.txt", "a");
        // REFLECTモード
        now_val = ev3_color_sensor_get_reflect(color_sensor);
        sprintf(str, "REFLECT=%d", now_val);
        ev3_lcd_draw_string(str, 0, 0);
        tslp_tsk(WAIT_TIME_MS);
        // COLORモード
        now_color = ev3_color_sensor_get_color(color_sensor);
        switch (now_color) {
            case COLOR_GREEN:
                ev3_lcd_draw_string("GREEN", 0, 10);
```



```

        sprintf(colname, "GREEN");
        break;
    case COLOR_YELLOW:
        ev3_lcd_draw_string("YELLOW", 0, 10);
        sprintf(colname, "YELLOW");
        break;
    case COLOR_RED:
        ev3_lcd_draw_string("RED", 0, 10);
        sprintf(colname, "RED");
        break;
    case COLOR_BLACK:
        ev3_lcd_draw_string("BLACK", 0, 10);
        sprintf(colname, "BLACK");
        break;
    case COLOR_BLUE:
        ev3_lcd_draw_string("BLUE", 0, 10);
        sprintf(colname, "BLUE");
        break;
    case COLOR_WHITE:
        ev3_lcd_draw_string("WHITE", 0, 10);
        sprintf(colname, "WHITE");
        break;
    case COLOR_BROWN:
        ev3_lcd_draw_string("BROWN", 0, 10);
        sprintf(colname, "BROWN");
        break;
    case COLOR_NONE:
        ev3_lcd_draw_string("NONE", 0, 10);
        sprintf(colname, "NONE");
        break;
    default:
        ev3_lcd_draw_string("NONE", 0, 10);
        sprintf(colname, "NONE");
        break;
}
// AMBIENTモード
now_value = ev3_color_sensor_get_ambient(color_sensor);
sprintf(str, "AMBIENT=%d", now_value);
ev3_lcd_draw_string(str, 0, 20);
fprintf(fp, "%d\t%s\t%d\t%r\n", now_val, colname, now_value);
fclose(fp);
tslp_tsk(WAIT_TIME_MS);
}
}

void main_task(intptr_t unused) {
    /* センサーの設定 */
    ev3_sensor_config( color_sensor , COLOR_SENSOR );

    /* タスクを開始する */
    act_tsk(RUN_TASK);
}

```

app.h

```
#include "target_test.h"

#ifndef STACK_SIZE
#define STACK_SIZE          4096          /* タスクのスタックサイズ */
#endif /* STACK_SIZE */

/* 関数のプロトタイプ宣言 */
#ifndef TOPPERS_MACRO_ONLY
extern void    main_task(intptr_t exinf);
extern void run_task(intptr_t exinf);
#endif /* TOPPERS_MACRO_ONLY */
```

app.cfg

```
INCLUDE("app_common.cfg");

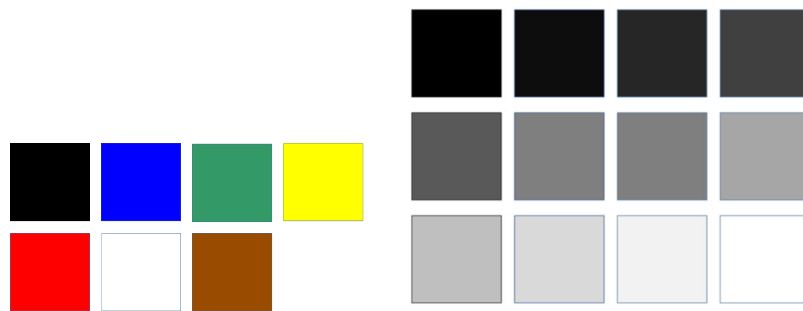
#include "app.h"

DOMAIN(TDOM_APP) {
  CRE_TSK( RUN_TASK, { TA_NULL, 0, run_task,  TMIN_APP_TPRI,    STACK_SIZE, NULL });
  CRE_TSK(MAIN_TASK, { TA_ACT, 0, main_task, TMIN_APP_TPRI + 1, STACK_SIZE, NULL });
}

ATT_MOD("app.o");
```

実験番号：2-I-3

実験タイトル：レゴマインドストームの C 言語による制御（基礎）



a) 色

b) グレースケール

図 14 カラーマッピングの例

カラーID (ColorID)

COLOR_NONE	不明	COLOR_RED	赤色
COLOR_BLACK	黒色	COLOR_WHITE	白色
COLOR_BLUE	青色	COLOR_BROWN	茶色
COLOR_GREEN	緑色	TNUM_COLOR	カラーID 数の 最大値
COLOR_YELLOW	黄色		

使用した API

ev3_color_sensor_get_color(A)	COLOR モードで値を取得する． A: ポート
ev3_color_sensor_get_reflect(A)	REFLECT モードで値を取得する． A: ポート
ev3_color_sensor_get_ambient(A)	AMBIENT モードで値を取得する． A: ポート

## ③ サーボモータの制御

## &lt;説明&gt;

EV3 に接続されたモータを制御し、3 秒間前進した後、スタート位置まで戻るプログラムを作成する。スタート位置に戻る際は、3 秒間後進するのではなく、3 秒間の全身でモータが回転した回転角度を取得し、回転角度を指定して後進させる。

## &lt;実験課題&gt;

下記リストを実行し、上記動作をしていることを確認する。

次に、EV3 の片側だけモータを動かし 10 度、15 度、30 度、45 度、90 度、180 度、360 度回転させるにはどのようなパラメータにすればよいかを実験を通して確認すること。また、この値を表にまとめてみる。

app.c

```
/* モータ制御 */

#include "ev3api.h"
#include "app.h"

#ifdef BUILD_MODULE
#include "module_cfg.h"
#else
#include "kernel_cfg.h"
#endif

/* モータのポートを設定 */
#define R_motor          EV3_PORT_B
#define L_motor          EV3_PORT_C

/* 待機時間の設定 [msec] */
const uint32_t RUN_TIME_MS = 3000;
const uint32_t WAIT_TIME_MS = 500;

void run_task(intptr_t unused) {
    int power;
    static int32_t cnt = 0;
    ev3_motor_reset_counts(L_motor);          // 回転角度をリセット
    power = 20;          // 前進

    ev3_motor_steer(L_motor, R_motor, power, 0 );
    tslp_tsk(RUN_TIME_MS);

    // 停止
    ev3_motor_stop(L_motor, true);
    ev3_motor_stop(R_motor, true);
    tslp_tsk(WAIT_TIME_MS);

    // 後進
    cnt = ev3_motor_get_counts(L_motor) * -1;
```

```

        ev3_motor_rotate(L_motor, cnt, power, false );
        ev3_motor_rotate(R_motor, cnt, power, false );
    }

void main_task(intptr_t unused) {
    /* モータの設定 */
    ev3_motor_config( L_motor , LARGE_MOTOR );
    ev3_motor_config( R_motor , LARGE_MOTOR );

    act_tsk(RUN_TASK);      /* タスクを開始する */
}

```

app.h

```

#include "target_test.h"

#ifndef STACK_SIZE
#define STACK_SIZE          4096          /* タスクのスタックサイズ */
#endif /* STACK_SIZE */

/* 関数のプロトタイプ宣言 */
#ifndef TOPPERS_MACRO_ONLY
extern void    main_task(intptr_t exinf);
extern void run_task(intptr_t exinf);
#endif /* TOPPERS_MACRO_ONLY */

```

app.cfg

```

INCLUDE("app_common.cfg");

#include "app.h"

DOMAIN(TDOM_APP) {
    CRE_TSK( RUN_TASK, { TA_NULL, 0, run_task,  TMIN_APP_TPRI,    STACK_SIZE, NULL });
    CRE_TSK(MAIN_TASK, { TA_ACT, 0, main_task, TMIN_APP_TPRI + 1, STACK_SIZE, NULL });
}

ATT_MOD("app.o");

```

## モータのポート

EV3_PORT_A	ポート A
EV3_PORT_B	ポート B
EV3_PORT_C	ポート C
EV3_PORT_D	ポート D
TNUM_MOTOR_PORT	モータのポート数の最大値

## モータのタイプ

NONE_MOTOR	モータ接続なし
MEDIUM_MOTOR	M モータ
LARGE_MOTOR	L モータ
UNREGULATED_MOTOR	未制御のモータ
TNUM_MOTOR_TYPE	モータタイプ数の最大値

## 使用した API

ev3_motor_config(A, B)	モータポートを設定. モータポートに接続しているモータのタイプを設定 A: ポート      B: モータタイプ
ev3_motor_steer(A, B, C, D)	2 つのモータでステアリング操作を行う. A: 左モータのポート      B: 右モータのポート C: パワー -100~100 D: ステアリングの度合い (ーは左, +は右回転) -100~100
ev3_motor_set_power(A, B)	power を指定してモータを回転させる. A: ポート      B: power -100~100
ev3_motor_stop(A, B)	モータをストップする A: ポート      B: ブレーキ(true) or 惰性走行(false)
ev3_motor_get_counts(A)	モータの回転数を取得する. A: ポート
ev3_motor_reset_counts(A)	モータの回転角度をゼロにリセットする. A: ポート
ev3_motor_rotate(A, B, C, D)	モータを指定した角度だけ回転させる. A: ポート      B: 回転角度, ーは逆回転 C: 回転速度 (0~100) D: true=回転が完了してからリターン false=回転完了を待たずにリターン

## ④ タスク処理

&lt;説明&gt;

以下のリストでは、「カラーセンサの閾値の確認」で使用したプログラムを基に、1 秒ごとに起動される周期タスク（音が鳴るタスク）を新たに加えたプログラムを作成する。

&lt;実験課題&gt;

下記リストを実行し、上記動作をしていることを確認する。

app.c

```

/* カラーセンサの制御に周期タスクを追加 */

#include "ev3api.h"
#include "app.h"

#ifdef BUILD_MODULE
#include "module_cfg.h"
#else
#include "kernel_cfg.h"
#endif

#define color_sensor    EV3_PORT_3    /* センサーのポートを設定 */

const uint32_t WAIT_TIME_MS = 100;    /* 待機時間の設定 [msec] */

void cyclic_task(intptr_t exinf)
{
    ev3_speaker_set_volume(5);
    ev3_speaker_play_tone( NOTE_C5, 200);
    tslp_tsk(200);
}

void run_task(intptr_t unused) {
    int now_val = 0;
    int now_value = 0;
    char str[256];
    colorid_t now_color;

    while (1) {
        // REFLECTモード
        now_val = ev3_color_sensor_get_reflect(color_sensor);
        sprintf(str, "REFLECT=%d", now_val);
        ev3_lcd_draw_string(str, 0, 0);
        ev3_lcd_draw_string(str, 0, 0);
        tslp_tsk(WAIT_TIME_MS);
        // COLORモード
        colorid_t now_color = ev3_color_sensor_get_color(color_sensor);
        ev3_lcd_draw_string(str, 0, 10);
        switch (now_color) {
            case COLOR_GREEN:
                ev3_lcd_draw_string("GREEN", 0, 10);

```

```

        break;
    case COLOR_YELLOW:
        ev3_lcd_draw_string("YELLOW", 0, 10);
        break;
    case COLOR_RED:
        ev3_lcd_draw_string("RED", 0, 10);
        break;
    case COLOR_BLACK:
        ev3_lcd_draw_string("BLACK", 0, 10);
        break;
    case COLOR_BLUE:
        ev3_lcd_draw_string("BLUE", 0, 10);
        break;
    case COLOR_WHITE:
        ev3_lcd_draw_string("WHITE", 0, 10);
        break;
    case COLOR_BROWN:
        ev3_lcd_draw_string("BROWN", 0, 10);
        break;
    case COLOR_NONE:
        ev3_lcd_draw_string("NONE", 0, 10);
        break;
    default:
        ev3_lcd_draw_string("NONE", 0, 10);
        break;
    }
    // AMBIENTモード
    now_value = ev3_color_sensor_get_ambient(color_sensor);
    sprintf(str, "AMBIENT=%d", now_value);
    ev3_lcd_draw_string("                ", 0, 20);
    ev3_lcd_draw_string(str, 0, 20);
    tslp_tsk(WAIT_TIME_MS);
}

void main_task(intptr_t unused) {
    /* センサーの設定 */
    ev3_sensor_config( color_sensor , COLOR_SENSOR );

    /* タスクを開始する */
    ev3_sta_cyc(CYCHDR1);
    act_tsk(RUN_TASK);
}

```

app.h

```

#include "target_test.h"

#ifndef STACK_SIZE
#define STACK_SIZE          4096          /* タスクのスタックサイズ */
#endif /* STACK_SIZE */

```



```

/* 関数のプロトタイプ宣言 */
#ifndef TOPPERS_MACRO_ONLY
extern void      main_task(intptr_t exinf);
extern void run_task(intptr_t exinf);
extern void cyclic_task(intptr_t exinf);
#endif /* TOPPERS_MACRO_ONLY */

```

app.cfg

```

INCLUDE("app_common.cfg");

#include "app.h"

DOMAIN(TDOM_APP) {
  CRE_TSK( RUN_TASK, { TA_NULL, 0, run_task,  TMIN_APP_TPRI,      STACK_SIZE, NULL });
  CRE_TSK(MAIN_TASK, { TA_ACT, 0, main_task, TMIN_APP_TPRI + 1, STACK_SIZE, NULL });
  EV3_CRE_CYC( CYCHDR1, { TA_NULL, 0, cyclic_task, 1000, 0 });
}

ATT_MOD("app.o");

```

## タスク制御 API

EV3_CRE_CYC(A, {B, C, D, E, F})	<p>周期タスクを生成</p> <p>A: 周期タスク ID                      B: 周期タスク属性</p> <p>C: 周期タスクの拡張情報      D: 周期タスクの起動番地</p> <p>E: 周期タスクの起動周期      F: 周期タスクの起動位相</p>
CRE_TSK(A, {B, C, D, E, F, G, H, I})	<p>タスクを生成</p> <p>A: タスク ID                          B: タスク属性</p> <p>C: タスクの拡張情報      D: タスクの起動優先度</p> <p>E: タスクのスタック領域のサイズ</p> <p>F: タスクのスタック領域の先頭番地</p> <p>G: タスクのシステムスタック領域のサイズ</p> <p>H: タスクのシステムスタック領域の先頭番地</p>