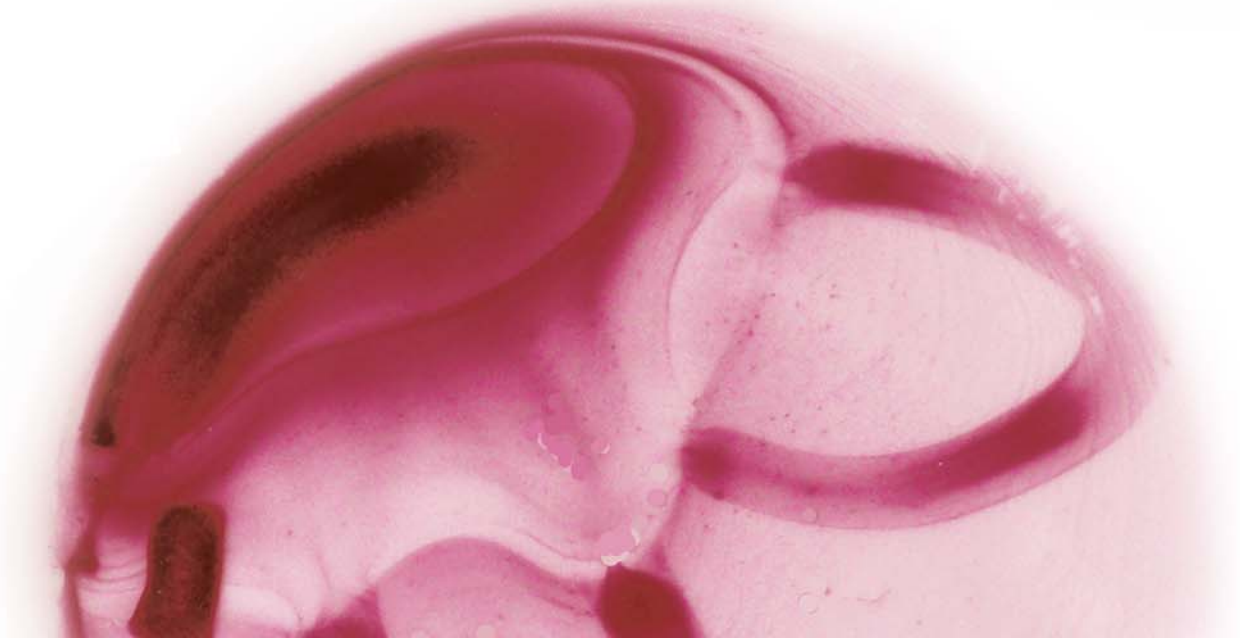
EAPL-Trainer™

Embedded Application development Trainer

mruby

FACTICS®



注意事項(必ず確認してください)

・本書は著作権上の保護を受けています。本書の一部あるいは全部について、株式会社アイ・エル・シーから文書による許諾を得ずに、いかなる方法においても無断で複写、複製することは禁じられています。

また、本書付属の CD-ROM(内部のソフトウェアなどを含みます)のご使用にあたり、以下の点に注意してください。

・CD-ROM は、特定のコンピュータでのみ使用可能です。

・CD-ROM に含まれるソフトウェアによって生成されたソースコードおよびソースコードを利用し作成した実行モジュールを、商用利用を目的に使用することはできません。

・株式会社アイ・エル・シーは、CD-ROM 内のソフトウェアのご使用に関する Q&A 等のサポートは原則行いません。

・CD-ROM の著作権は株式会社アイ・エル・シーが有するものであり、日本国著作権法により保護されています。

・お客様は、CD-ROM を複製することはできません。

・お客様は、株式会社アイ・エル・シーに無断で、第三者に対する CD-ROM の転載、配布、公開、公衆送信、譲渡、貸与、使用許諾その他一切の行為を行うことはできません。

・お客様は CD-ROM 内のソフトウェアについてリバース・エンジニアリング、逆コンパイル、逆アンセンブルすることはできません。

・株式会社アイ・エル・シーは、いかなる場合においても、CD-ROM の使用または使用不可能から生ずる損害に関しての責任の一切を負わないものとします。

・CD-ROM の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、株式会社アイ・エル・シーは一切その責を負いません。

・本書の内容に基づいて作成したアプリケーション及びサンプル提供アプリケーションを、商用利用を目的に使用することはできません。

本書に登場する製品名などは、一般に、各社の登録商標、または商標です。

なお、本文中に TM, ®, ©マークなどは特に明記しておりません。

目次

はじめに

(1) mrubyとは

日本発の純国産プログラム言語「Ruby」をベースとして、組込み開発の分野にも適用できるように作られた言語が「mruby」です。
Ruby は他の言語に比べて非常に開発効率が高いことが特長ですが、これまではパソコンやネットワークサーバーなどで主に使われてきました。
mruby は Ruby の特長を引き継ぎつつ、組込み機器でも動作できるように軽量・コンパクトに作られた言語です。
これからは、組込み開発の現場でも開発効率の高い mruby を使うことが可能になります。

EAPL-Trainer mruby (以下、本製品)は、日本発祥の組込み開発言語 mruby を習得するための学習教材です。

(2) 本製品の対象ユーザー

本製品では、同梱の液晶付きマイコンボードを使って組込みプログラムの基本を学ぶことができます。何らかのプログラム言語の経験がある人を対象として想定していますが、掲載しているプログラムは簡単なものですのでプログラムの初心者の方でも自分で動かすことができると思います。

(3) 本書の構成と使い方

第１章　付属マイコンボードの準備
付属マイコンボードを使うための準備の手順を記載します。

第２章　プログラムの作成：基礎編
付属マイコンボードを使って mruby プログラムを作成するために必要な基礎知識を記載します。
マイコンボードの基本的な使い方、mruby の基本的な文法、グラフィックを使ったプログラムの作り方、等、付属マイコンボードだけを使って実践できる内容が記載されています。
mruby を使ったマイコンプログラム作成の基礎を習得できます。

第３章　プログラムの作成：応用編
プログラム作成の応用として、外付けの基板と電子部品を使ったプログラムの作成について記載し

ます。
マイコンボードの入出力端子に色々な部品を接続して制御を行うことができます。

第４章　付録
プログラムがうまく動かない場合の対処方法を記載します。何らかの問題が発生したときにはこの章を参照してください。
また、マイコンボードに元々書き込まれているシステムプログラムを書きかえる場合の手順を記載します。

(4) 本書の表記法

本書で使う書体は以下の法則に従っています。

等幅(Courier)
　プログラムコードやプログラムコードに使用する語や出力結果などに使います。

等幅の太字(**Courier Bold**)
　コマンドラインで入力して実行するコマンド名やオプションなどに使います。

斜体(Italic)
　プログラム中に記述する項目の説明などに使います。

［途中のページは省略します］

第１章
付属マイコンボードの準備

本章では、付属のマイコンボードの使い方と学習に使う PC の準備の手順を説明します。
マイコンボードを使える状態にして、最も簡単なプログラムを作って動かします。

1.1 準備するもの

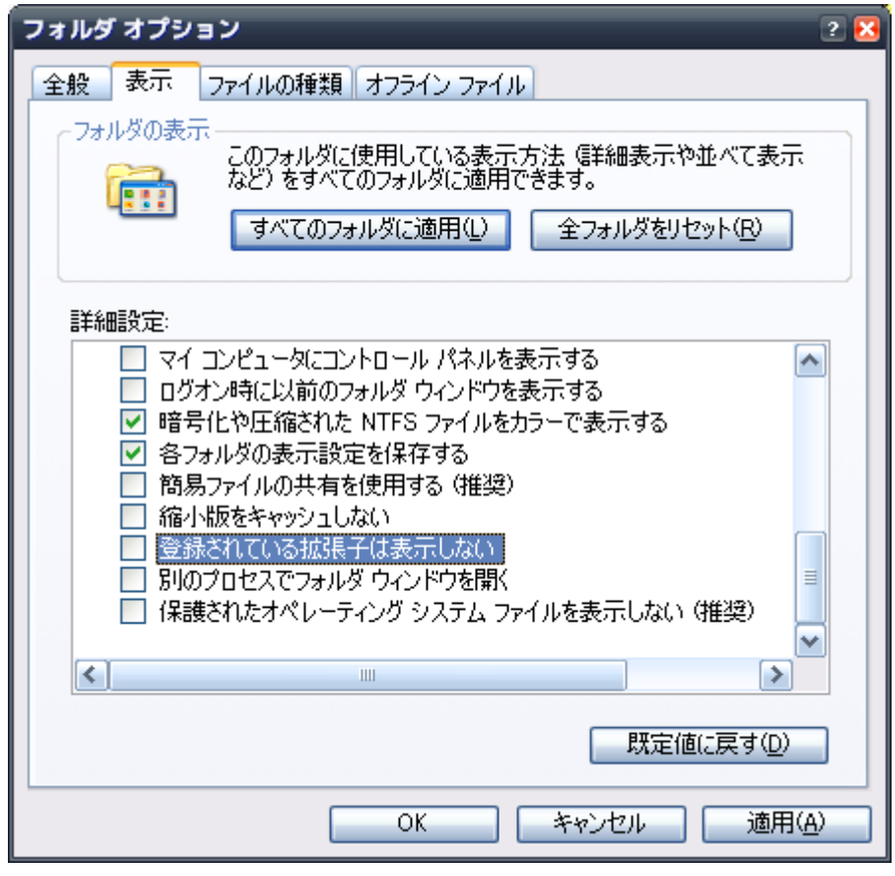
付属のマイコンボードを使うためには以下のものを準備します。

(1) プログラム開発用PC

Windows XP/Windows 7 が動作する PC です。
本書での説明はファイルの拡張子を表示した設定に合わせていますので、あらかじめファイルの拡張子を表示する設定としておいてください。

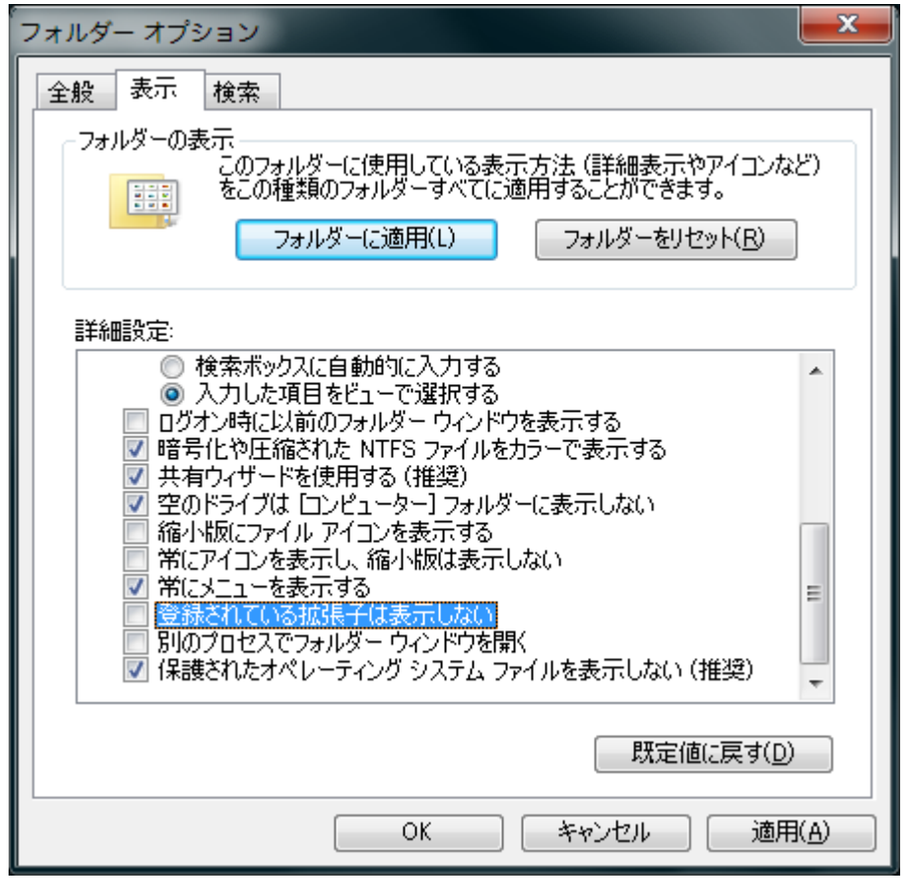
■Windows XP の場合の設定方法

エクスプローラのメニュー[ツール(T)] - [フォルダ オプション(O)]を選択してください。下記のダイアログが表示されますので、下記の「登録されている拡張子は表示しない」という項目のチェックボックスを外してください。



■Windows 7 の場合の設定方法

エクスプローラのメニュー[ツール(T)] - [フォルダー オプション(O)]を選択してください。下記のダイアログが表示されますので、下記の「登録されている拡張子は表示しない」という項目のチェックボックスを外してください。



(2) SDメモリカード

付属マイコンボードで動作させるプログラムを書きこみます。

対応する規格は下記となりますので、下記の表で"○"となっているものを用意してください。

	SD カード	miniSD カード	microSD カード
SD	○	○(*1)	○(*1)
SDHC	×	×	×
SDXC	×	×	×

(*1)SD カードサイズのアダプタが必要です。

SDHC/SDXC 規格のカードには対応していませんのでご注意ください。

(3) USBケーブル

PC と接続して付属マイコンボードに電源を供給します。

USB A オス <-> USB ミニ B オス のケーブルを用意してください。

USB A オス

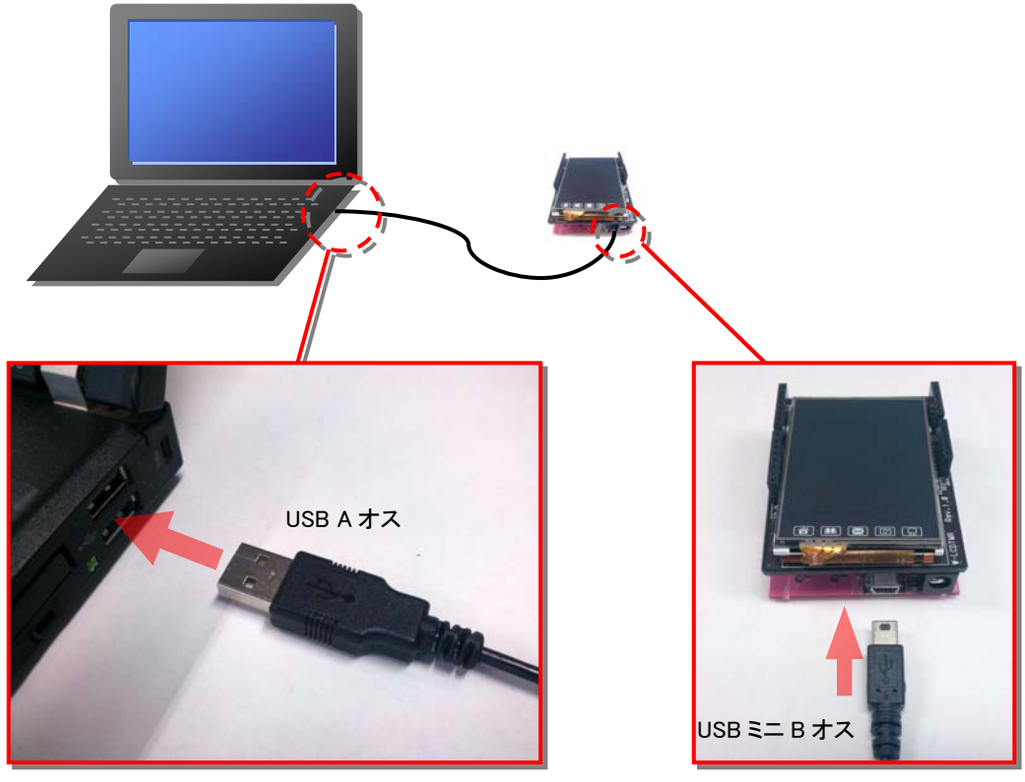
USB ミニ B オス

1.2 接続方法

付属のマイコンボードの電源としてPCのUSBポートが使えます。

1.1項で用意したUSBケーブルのUSB Aオスコネクタを動作中のPCに、USBミニBオスコネクタをマイコンボードに接続することによって、マイコンボードに電源が供給されてマイコンボードが起動します。

ケーブルの接続順序はPC側とマイコンボード側のどちらが先でも構いません。



電源を切る場合は、マイコンボードから USB ケーブルを外してください。

1.3 最初のプログラムの実行

(1) プログラムの作成

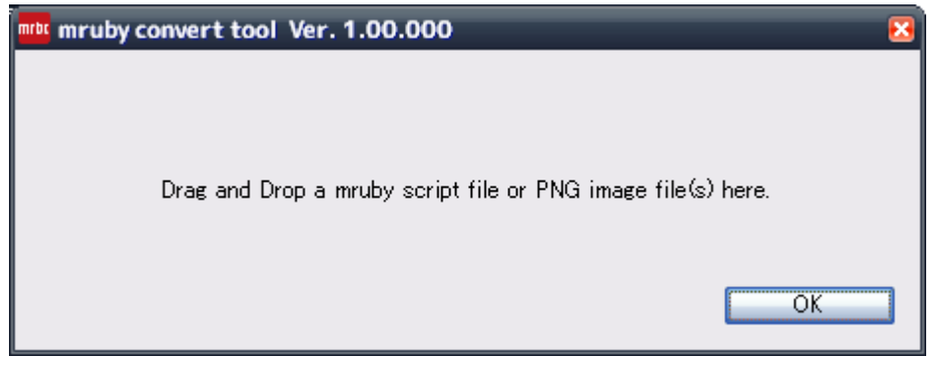
マイコンボード上にある LED を点滅させるだけの簡単なプログラムを作りましょう。
Windows PC 上で、メモ帳などを使用して下記のプログラムを作成してください。

```
def setup()
    gr_pinMode($PIN_LED0, $OUTPUT)
end

def loop()
    gr_digitalWrite($PIN_LED0, 1)
    gr_delay(100)
    gr_digitalWrite($PIN_LED0, 0)
    gr_delay(100)
end
```

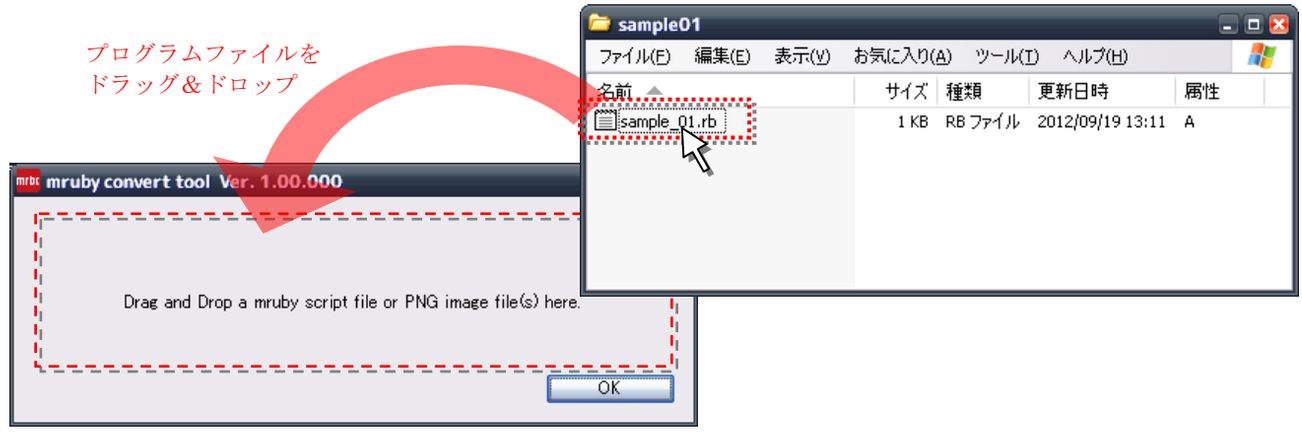
上記のプログラムを**"sample_01.rb"**という名前で保存します。
また、インストールフォルダ内の **sample¥sample01** フォルダに同じ内容のファイルが格納されています。

(2) プログラムのコンパイル

インストールフォルダ内の **tool** フォルダに格納されている **"mrbConvert.exe"**というプログラムを実行してください。実行するとプログラム変換ツール(下記の画面)が表示されます。

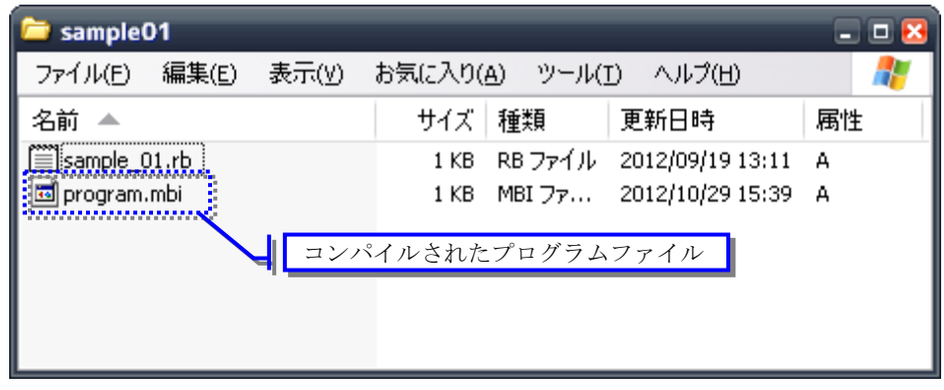


(1)で作成したプログラムファイルを上記の画面にドラッグ＆ドロップしてください。



正常にプログラムがコンパイルされると、下記のようにプログラムファイルと同じフォルダに"**program.mbi**"というファイルが生成されます。
何か問題が発生するとエラーダイアログが表示される場合があります。その場合は第４章 4.1.1を参照してください。

インストールフォルダ内の **sample¥sample01** フォルダに変換結果の"**program.mbi**"ファイルも格納されています。
プログラム変換ツールが表示されている間は、何度でもプログラムファイルをドラッグ＆ドロップすると"**program.mbi**"ファイルが生成されます。ただし、ファイル名は固定ですので同じプログラムをドラッグ＆ドロップすると"**program.mbi**"ファイルは上書きされます。



プログラム変換ツールの[OK]ボタンを押すと、プログラム変換ツールは終了します。

(3) プログラムの書き込みと実行

(2)で生成された"**program.mbi**"というファイルをSDカードにコピーしてください。
その際にSDカードにはフォルダを生成せずSDカードの直下にファイルをコピーしてください。また、ファイル名は"**program.mbi**"から変更しないでください。
※対応するSDカードの種別は「第１章 1.1(2)SDメモリカード」の項目を参照ください。

上記のプログラムをコピーしたSDカードを、マイコンボード液晶基板に搭載されているSDカードスロットに挿入して、マイコンボードの電源を入れるとプログラムを自動的に実行します。
既にマイコンボードの電源が入っている場合は、一度USBケーブルを外して電源を切断した後でSDカードを挿入して、マイコンボードの電源を入れてください。
※電源の入れ方は、「第１章 1.2 接続方法」の項目を参照ください。

プログラムが正常に実行されるとマイコンボードの下図の部分の LED が点滅します。

これで、最初のプログラムの動作確認は終わりです。
以後、本書で説明しているプログラムのコンパイル・書き込み・実行は同様の手順で行ってください。

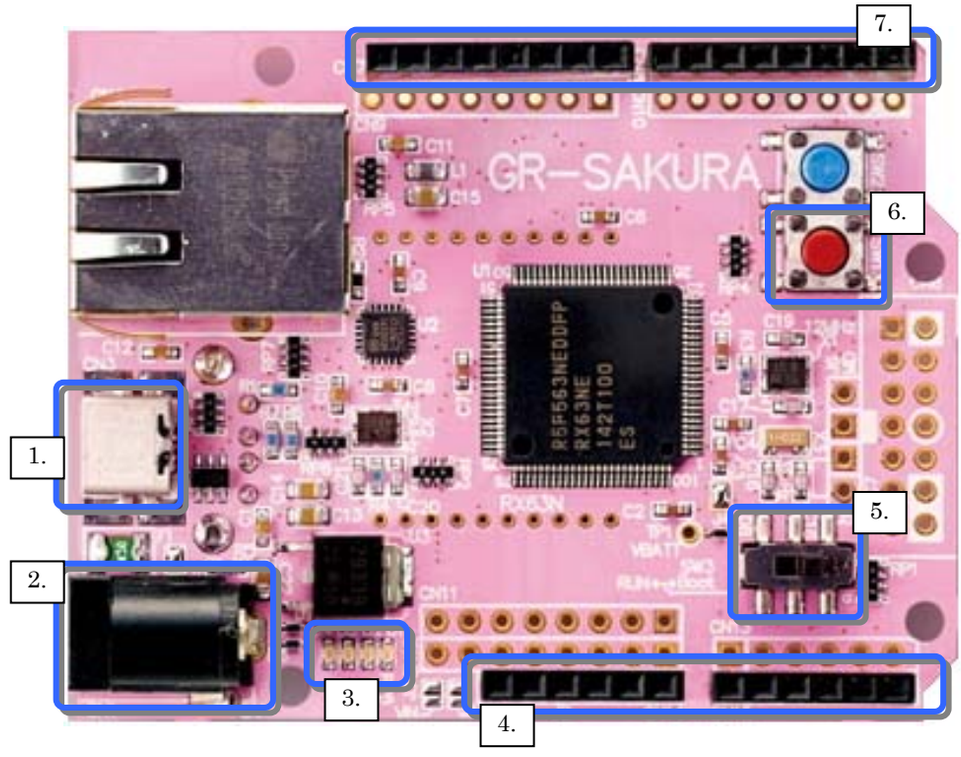
第２章
プログラム作成：基礎編

本章では、付属のマイコンボードで動作する簡単なプログラムの説明を行います。
mruby を使ってマイコンボードを動かすプログラムの作り方を習得できます。

2.1 マイコンボードの基本

2.1.1 マイコンボードの各端子と機能

マイコンボードの外観を以下に示します。



番号	説明
1.	USB 接続コネクタ。PC と接続してプログラムの転送、または、電源供給を行う。
2.	電源コネクタ。USB 接続コネクタに PC を接続している場合は、接続不要です。
3.	LED。ユーザ作成プログラムから点灯・消灯ができます。
4.	各種入出力端子。詳細は第３章で説明します。
5.	モード切換えスイッチ。常に"RUN"側にスイッチを入れてください。
6.	赤いボタンはリセットボタンです。
7.	各種入出力端子。詳細は第３章で説明します。

2.2 プログラム動作の説明

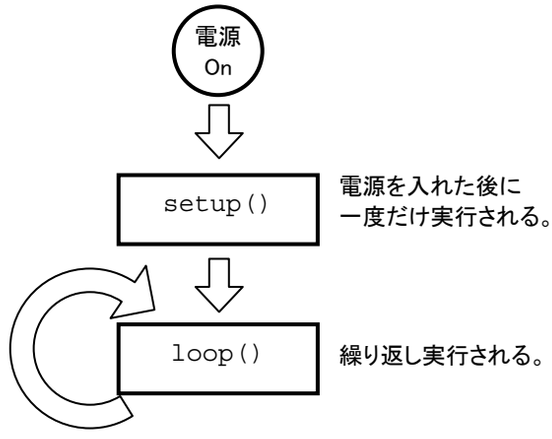
2.2.1 プログラムの作成

第一章で作ったプログラムについて説明します。

```
def setup()
    gr_pinMode($PIN_LED0, $OUTPUT)
end

def loop()
    gr_digitalWrite($PIN_LED0, 1)
    gr_delay(100)
    gr_digitalWrite($PIN_LED0, 0)
    gr_delay(100)
end
```

setup と loop という２つのキーワードを書いていますが、この２つのキーワードは必ず書く必要があります。

setup は電源を入れた後に一度だけ実行される処理です。loop は繰り返し実行される処理です。setup と loop の関係は以下のようになります。

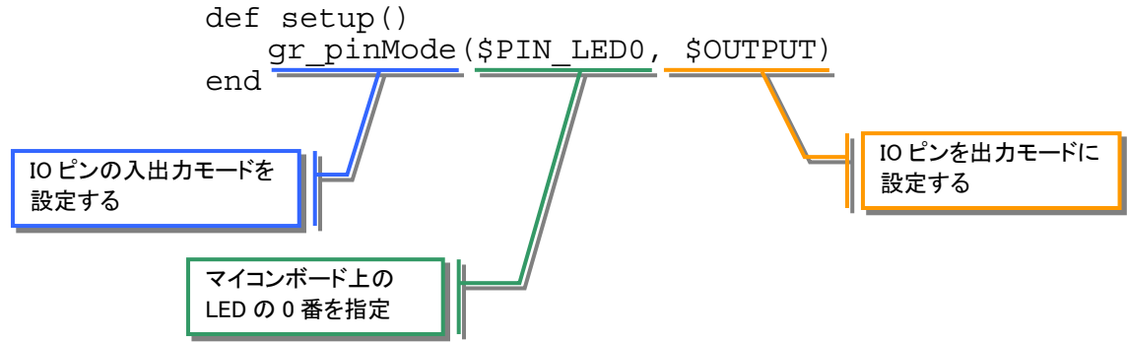


(1) setup処理

setup 処理の内容は下記となっています。

```
def setup()
   gr_pinMode($PIN_LED0, $OUTPUT)
end
```

setup 処理は電源 On の後に 1 回だけ実行されます。上記の setup 処理の意味は、「電源 On 後にマイコンボード上の LED の 0 番を出力モードに設定する」となります(下図参照)。

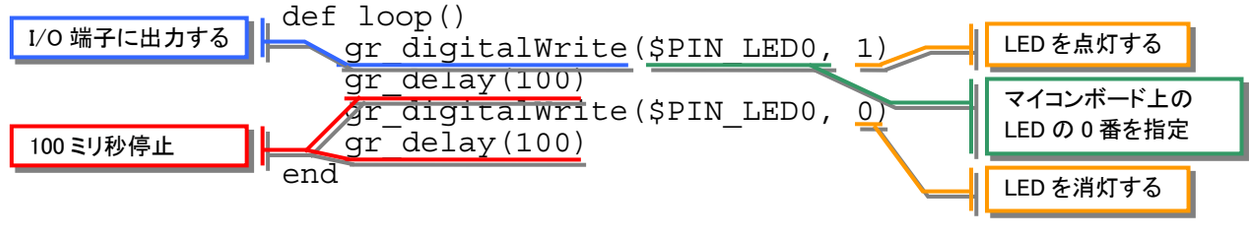


(2) loop処理

loop 処理の内容は下記となっています。

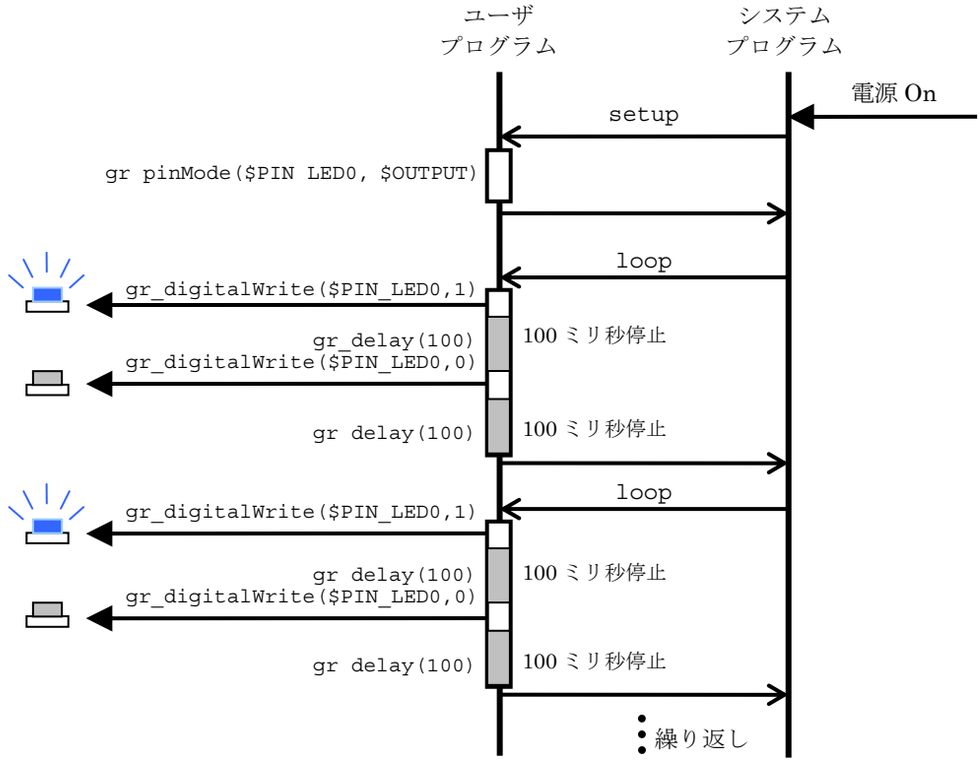
```
def loop()
   gr_digitalWrite($PIN_LED0, 1)
   gr_delay(100)
   gr_digitalWrite($PIN_LED0, 0)
   gr_delay(100)
end
```

loop 処理は繰り返し実行されます。上記の loop 処理の意味は、「マイコンボード上の LED の 0 番を点灯して 100 ミリ秒停止、LED の 0 番を消灯して 100 ミリ秒停止」となり、それを繰り返し実行することによって、LED を点滅させます(下図参照)。



(3) 全体の処理の流れ

全体の処理の流れは下記となります。



loop 処理が繰り返し実行されるため、LED が繰り返し点滅する動作となります。

2.2.2 動作を変えてみる

第１章で作成したプログラムを修正して下記のように変更します。

変更したプログラム：灰色の部分を変更

```
def setup()
   gr_pinMode($PIN_LED0, $OUTPUT)
end

def loop()
   gr_digitalWrite($PIN_LED0, 1)
   gr_delay(50)
   gr_digitalWrite($PIN_LED0, 0)
   gr_delay(50)
end
```

インストールフォルダ内の**sample¥sample02**フォルダに"**sample_01_modify.rb**"という名前で同じ内容のファイルが格納されています。

第１章と同じ手順でプログラムを実行してください。LED の点滅速度が速くなります。
このように、プログラムを修正することによってマイコンボードの動作を簡単に変えることができます。

2.3 mrubyプログラムの基本

ここまでで作成したプログラムは mruby というコンピュータ言語を使っています。本項では mruby の基本的な文法を説明します。

2.3.1 メソッド

mruby では、他の言語の関数やサブルーチンに相当するものをメソッドと呼びます。第１章で作成したプログラムの setup や loop もメソッドです。メソッドの書式は下記となります。

メソッドの書き方

```
def メソッドの名前（ メソッドの引数 ）
    メソッドで実行する処理
end
```

メソッドの引数が不要な場合は、引数の記述を省略することが可能です。

例)足し算を行うメソッド

```
def add(a, b)
   return a + b
end
```

上記の例では、メソッド名は add となります。引数は a と b の２つであり、a と b を足して返すという処理となります。

メソッドの使い方は下記となります。

```
def add(a, b)
   return a + b
end

c = add(10, 20)
```

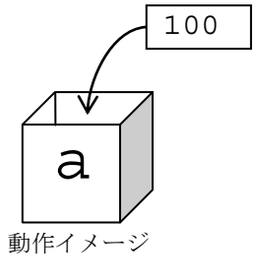
上記のプログラムを実行すると、add という名前のメソッドに引数として 10 と 20 を渡して呼び出します。add メソッドの内部でそれらを足し合わせて返し、その結果が c に入ります。
結果として、c の値は 30 となります。

2.3.2 変数

変数とは、名前の付いた箱のようなものです。その箱の中に数値などを入れることができます。

例）a という名前の変数に 100 という数値を入れるプログラム。

```
a = 100
```



動作イメージ

第１章で出てきた LED を点滅させるプログラムを題材に変数の使い方を説明します。

変更前：

```
def loop()
    gr_digitalWrite($PIN_LED0, 1)
    gr_delay(100)
    gr_digitalWrite($PIN_LED0, 0)
    gr_delay(100)
end
```

変更後：灰色の部分を変更

```
def loop()
    delayTime = 100
    gr_digitalWrite($PIN_LED0, 1)
    gr_delay(delayTime)
    gr_digitalWrite($PIN_LED0, 0)
    gr_delay(delayTime)
end
```

LED を点灯/消灯した後に 100 ミリ秒停止していました。この 100 という数値を delayTime という変数に置き換えています。
それによって、以下の２つの利点があります。

1. 処理の修正箇所が減る

停止時間を変更する場合、delayTime に入れている数値(100)を修正するだけで済みます。
変更前だと、100 という数値が２か所ありますので２か所修正する必要があります。

2. 数値の意味が明確になる

delayTime という変数に 100 という数値を入れて使うと、数値の意味がわかりやすくなります。
100 という数値がそのままプログラムに書いてあるとそれが何を意味するものかわかりにくいプログラムになります。

［途中のページは省略します］

2.4 タッチパネルを使ったプログラム

これまでは使用しなかったタッチパネル液晶画面を使ったプログラムの作り方を説明します。

2.4.1 LED点灯プログラム

(1) プログラムの動作内容

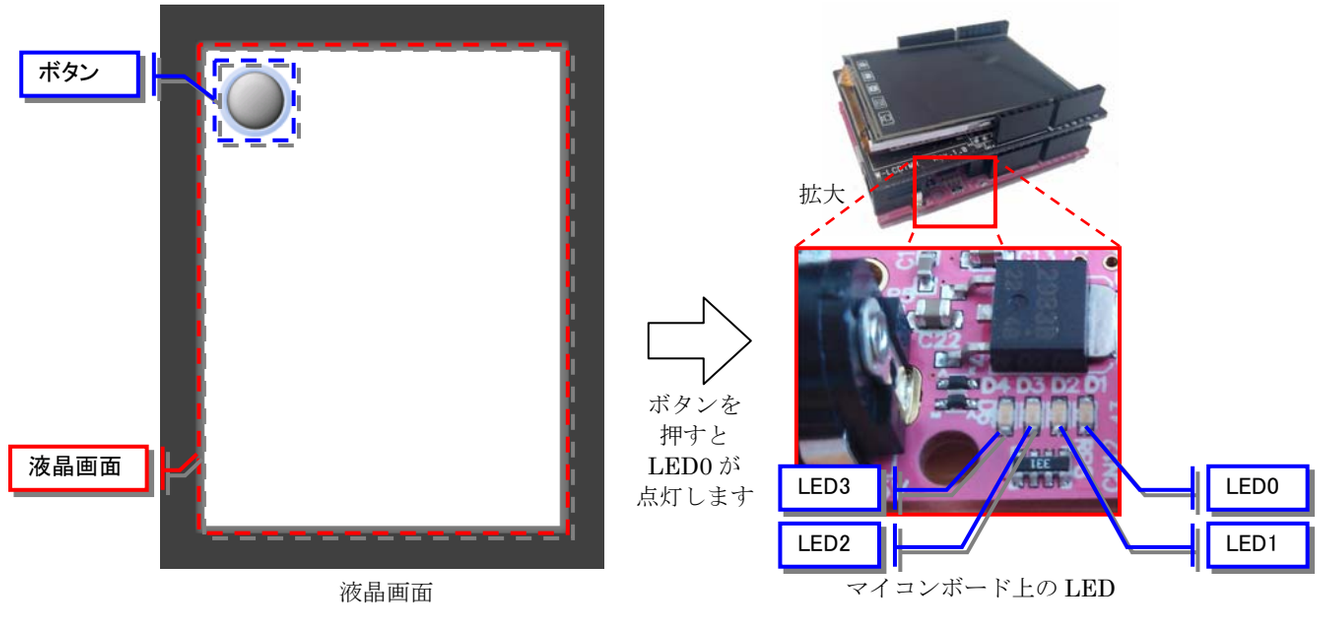
画面上にボタンを表示して、ボタンをタッチすると LED が点灯するプログラムを作ります。

(2) プログラムの作成

下記のプログラムを作成して実行してください。

```
def setup()
    # ボタンを作成
    $button1 = GNButton.new(10, 10, "onTouch1")

    gr_pinMode($PIN_LED0, $OUTPUT)
end

def onTouch1(param1, param2)
    # LED を On/Off
    gr_digitalWrite($PIN_LED0, param1)
end

def loop()
end
```

上記のプログラムを"**sample_03.rb**"という名前で保存します。インストールフォルダ内の **sample¥sample03** フォルダに同じ内容のファイルが格納されています。
上記のプログラムをコンパイルして SD カードに書き込み、実行すると下記のような画面が液晶に表示されます。



液晶画面に表示されたボタンを押すとマイコンボード上の LED が点灯して、離すと LED0 が消灯します。

(3) 処理内容の説明

処理内容の概要は下記となります。

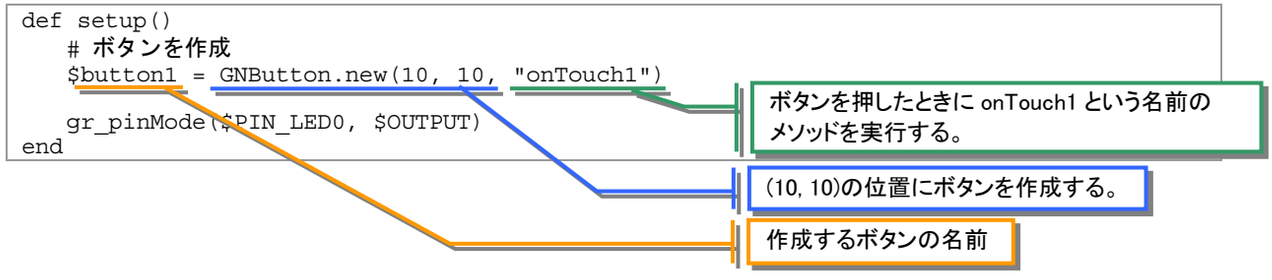
```
def setup()
    # ボタンを作成
    $button1 = GNButton.new(10, 10, "onTouch1")

    gr_pinMode($PIN_LED0, $OUTPUT)
end

def onTouch1(param1, param2)
    # LED を On/Off
    gr_digitalWrite($PIN_LED0, param1)
end

def loop()
end
```

液晶画面に配置する部品の設定。

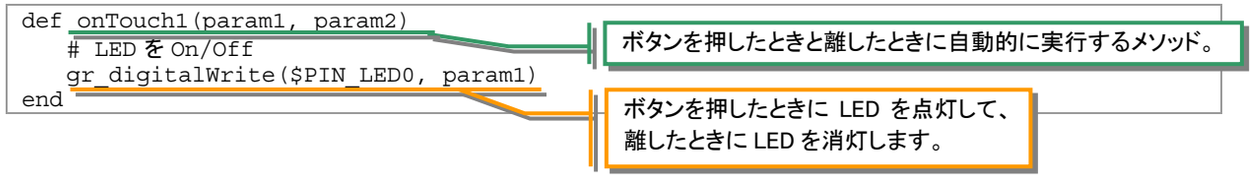
ボタンを押したときに実行する処理。

loop 処理は不要なので、メソッドの中身を空にしておきます。
loop メソッドそのものを削除すると正常に動作しません。

setup メソッドの中で、$button1 という名前のボタンを作成しています。
ボタンの作成には位置とメソッド名を設定します。この例では、画面上の(10, 10)の位置にボタンを配置して、ボタンを押したときには onTouch1 という名前のメソッドを実行する、という設定を行っています。

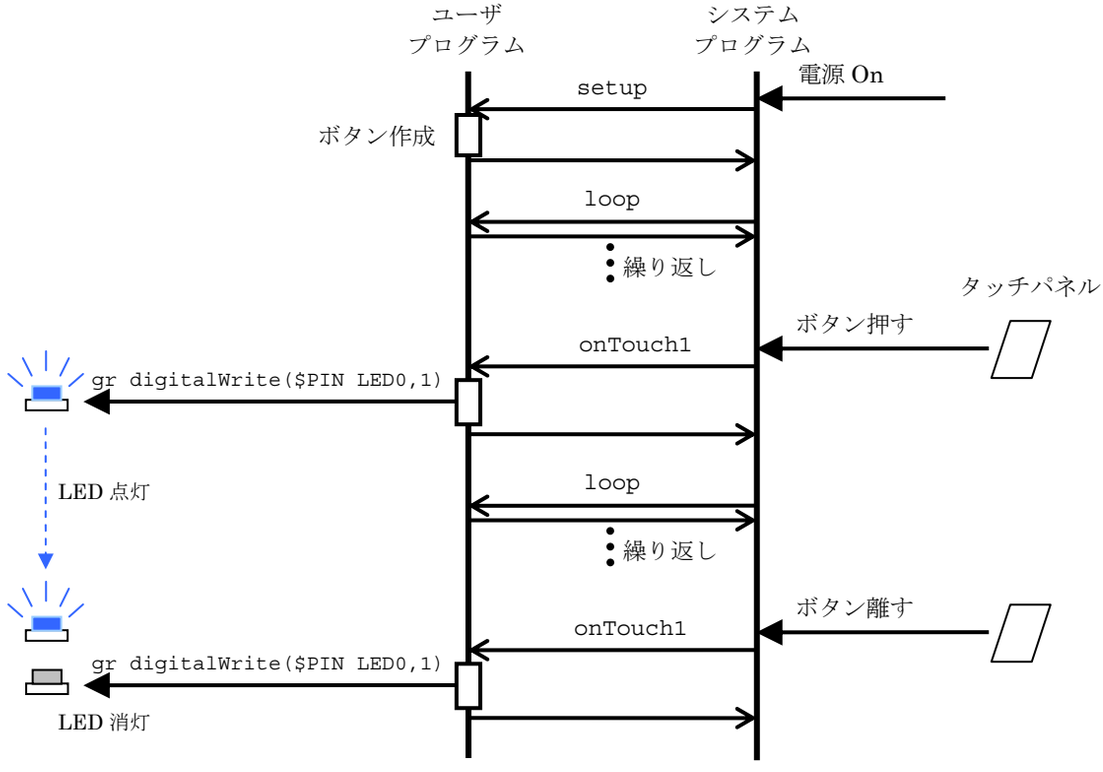


ボタンを押した場合に実行するメソッドには param1 と param2 という名前の引数があります。ボタンを押した場合には param1 の値は 1 となり、押したボタンを離した場合には param1 の値は 0 となります。
param1 の値を使って、押した場合と離した場合の処理を分けることが可能です。
この例では、param1 の値をそのまま LED 用のピンの出力値に使っているため、ボタンを押した場合は 1 を出力して LED が点灯します。押したボタンを離した場合は 0 を出力して LED が消灯します。



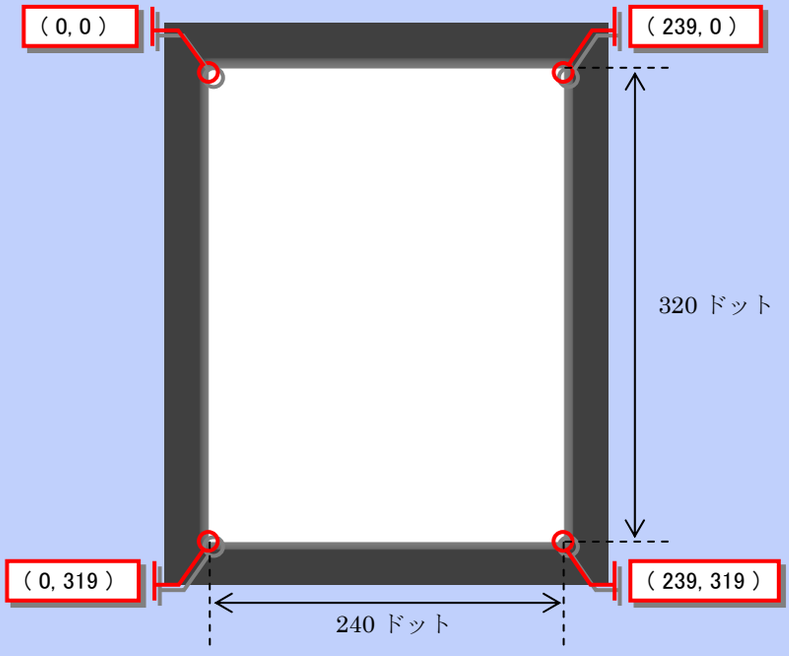
(4) 処理の流れ

全体の処理の流れは下記となります。loop メソッドが繰り返し実行されます。その途中で、ボタンを押したときと離したときだけ、onTouch1 メソッドが実行されます。
onTouch1 メソッドの中でボタンを押したときは LED0 を点灯、ボタンを離したときは LED0 を消灯します。それによって、ボタンを押している間は、LED0 は点灯したままとなります。

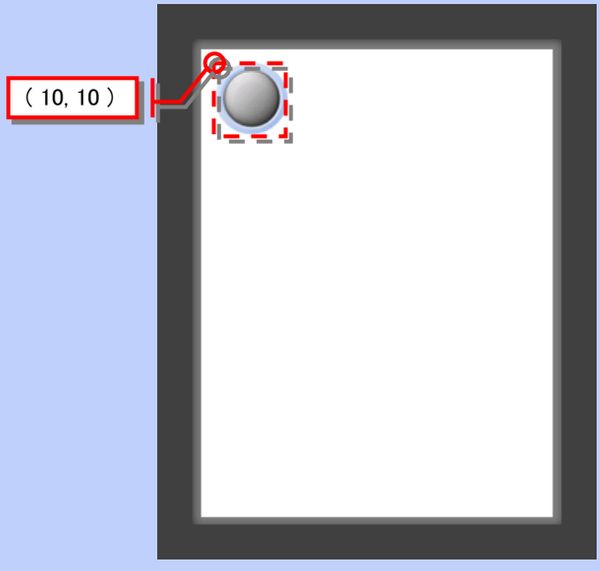


Memo ： 液晶画面の座標について

マイコンボードに搭載している液晶の表示画素数は横 240 ドット、縦 320 ドットです。
左上の座標が(0, 0)となりますので、画面上の座標は下記となります。



画面上に配置する部品の位置を指定するには、部品の左上の座標を設定します。
$button1 = GNButton.new(10, 10, "onTouch1") と記述した場合、下記のようにボタンの左上が画面上の(10, 10)の位置になります。



2.4.2 LED輝度変更プログラム

(1) プログラムの動作内容

前項では LED を点滅させるだけでしたが、付属のマイコンボードの LED は明るさを 256 段階で調整することができます。

ここでは、LED の明るさを連続的に変化させるプログラムを作成します。

(2) プログラムの作成

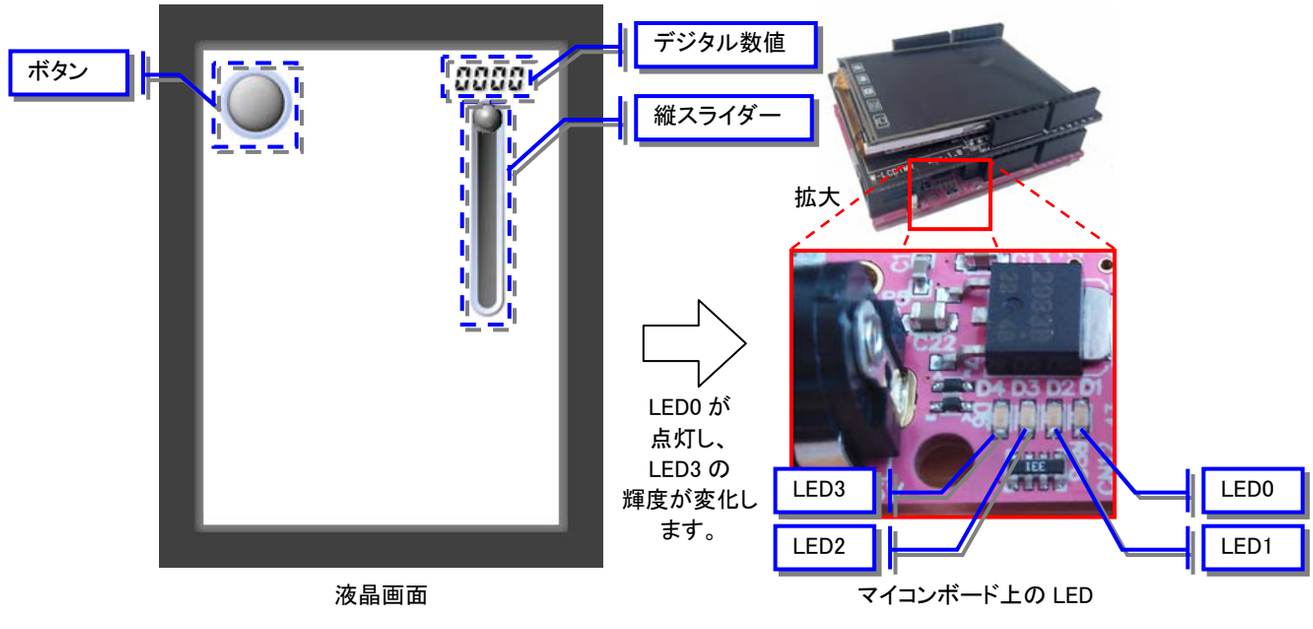
下記のプログラムを作成して実行してください。

```
def setup()
    $button1 = GNButton.new(10, 10, "onTouch1")
    $numbmp1 = GNDigitalNum.new(170, 10)
    $slider1 = GNVSlider.new(180, 30, 0, 255, "onTouch2")
    gr_pinMode($PIN_LED0, $OUTPUT)
    gr_pinMode($PIN_LED3, $OUTPUT)
end

def onTouch1(param1, param2)
    # LED を On/Off
    gr_digitalWrite($PIN_LED0, param1)
end

def onTouch2(param1, param2)
    gr_analogWrite($PIN_LED3, param1)
    $numbmp1.setInteger(param1)
end

def loop()
end
```

上記のプログラムを"sample_04.rb"という名前で保存します。

インストールフォルダ内の sample¥sample04 フォルダに同じ内容のファイルが格納されています。

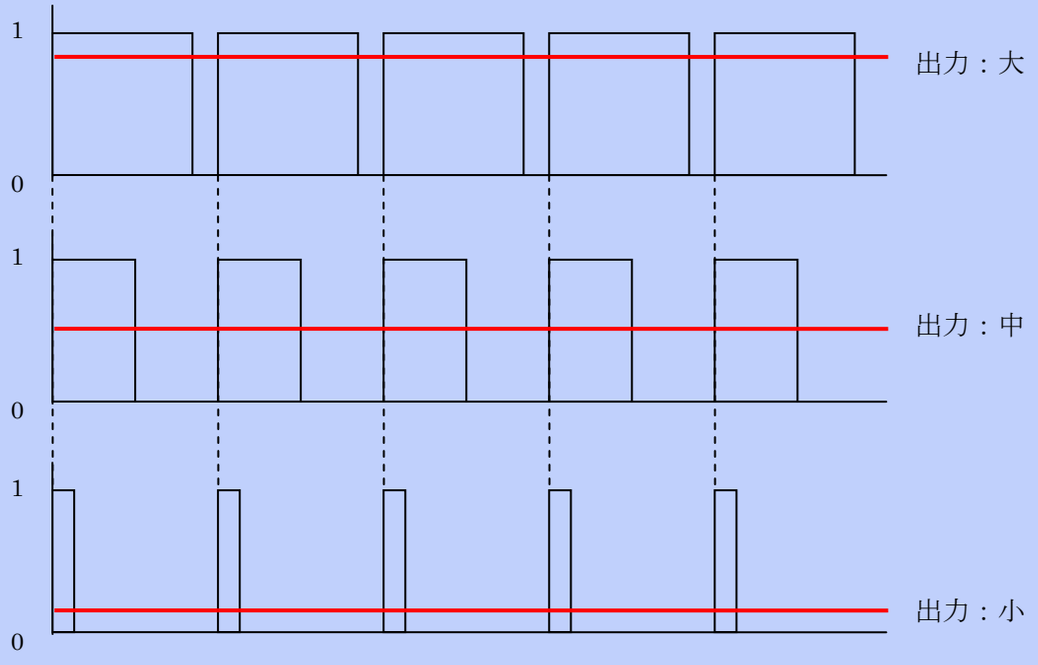
上記のプログラムをコンパイルして SD カードに書き込み、実行すると下記のような画面が液晶に表示されます。



［途中のページは省略します］

Memo ： PWM 制御について

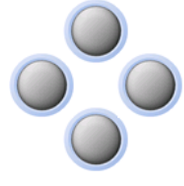
LED0～LED3 を含めてマイコンボードに搭載しているデジタル I/O ピンの出力値は基本的には 0 か 1 のデジタル値です。しかし、本項の例のように LED の明るさをアナログ値で設定することもできます。
これは、PWM 制御と呼ばれる方式で出力値を変化させています。
PWM(Pulse Width Modulation)とは、デジタル値出力を小刻みに 0 と 1 で切り換えて出力電圧の平均値を変化させるという方式です。
出力を 1 にする時間を長くすれば出力の平均値は高くなり、短くすれば出力の平均値が低くなります。



2.5 画面表示部品クラス

2.5.1 画面に表示可能な部品

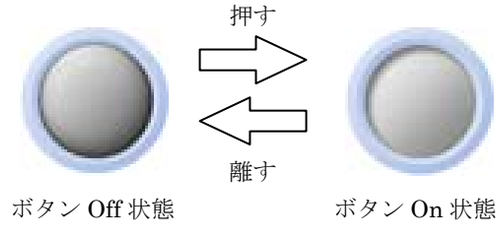
これまで、ボタンや縦スライダーなどの部品を液晶画面に表示して使ってきました。
ここでは、画面に表示して使うことができる部品について説明を行います。表示部品は mruby のクラス機能を使って実現しています。
従って、部品を使用する場合には表示部品クラスをもとにインスタンスを生成する必要があります。
液晶画面に表示して使用できる部品の一覧は下記となります。

表示部品名	クラス名	表示	機能
ボタン	GNButton		押している間だけ何かを実行したい場合に使います。
スイッチ	GNSwitch		押す度に On と Off を切換えることができます。
縦スライダー	GNVSlider		縦方向にスライドさせて連続的に数値を変化させます。
横スライダー	GNHSlider		横方向にスライドさせて連続的に数値を変化させます。
デジタル数値	GNDigitalNum		４桁の数値を表示します。
テキストボックス	GNTextbox		任意の英数字を表示します。
ピクチャ	GNPicture		SD カード上の画像を表示することができます。
デジタルパッド	GNDigitalPad		４方向のボタン入力を行うことができます。

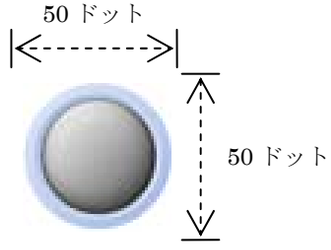
2.5.2 ボタン

(1) 表示

画面上に押しボタンを表示します。タッチパネルを押すとボタンが On になり、離すと Off に戻ります。



画面上の大きさは下記となります。



(2) ボタンの作り方

ボタンの作り方は下記となります。

ボタンの作り方

ボタンの名前 = GNButton.new(*x 座標* , *y 座標*, *"メソッド名"*)

ボタンの名前	：	*作成するボタンの名前*
x 座標	：	*ボタンの表示位置 x 座標*
y 座標	：	*ボタンの表示位置 y 座標*
メソッド名	：	*ボタンを操作したときに実行するメソッドの名前*

例）(40, 40)の位置に$button1 という名前のボタンを作成して、ボタン操作時に onTouch1 というメソッドを実行します。

```
$button1 = GNButton.new(40, 40, "onTouch1")
```

(3) ボタンを操作したときの処理

ボタンを押したときと離したときに実行します。ボタンを押し続けている間は実行しません。
ボタンを操作したときに実行するメソッドは以下の形となります。

ボタンを操作したときの処理の書き方

```
def メソッド名 (param1, param2)
end
```

メソッド名　　：ボタンを作成したときに設定したメソッド名
param1　　：ボタンの操作情報１
ボタンを押した場合は１となります
ボタンを離した場合は０となります
param2　　：ボタンの操作情報２
常に０となります

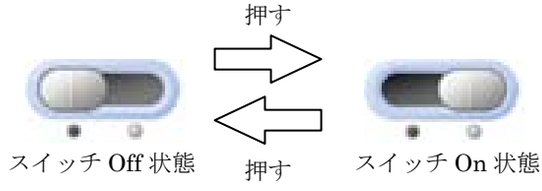
例）ボタンを押している間だけ LED を点灯します。

```
def onTouch1(param1, param2)
   # LED を On/Off
   gr_digitalWrite($PIN_LED0, param1)
end
```

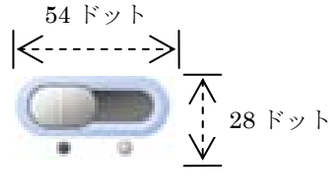
2.5.3 スイッチ

(1) 表示

画面上に切換えスイッチを表示します。タッチパネルを押すたびにスイッチの状態が On と Off を繰り返します。



画面上の大きさは下記となります。



(2) スイッチの作り方

スイッチの作り方は下記となります。

スイッチの作り方

```
def スイッチの名前 = GNSwitch.new(x 座標 , y 座標, "メソッド名" )
```

スイッチの名前　　：　作成するスイッチの名前
x 座標　　：　スイッチの表示位置 x 座標
y 座標　　：　スイッチの表示位置 y 座標
メソッド名　　：　スイッチを操作したときに実行するメソッドの名前

例）(80, 40)の位置に$switch1 という名前のスイッチを作成して、スイッチ操作時に onTouch2 というメソッドを実行します。

```
$switch1 = GNSwitch.new(80, 40, "onTouch2")
```

(3) スイッチを操作したときの処理

スイッチを押したときと離したときに実行します。スイッチを押し続けている間は実行しません。
スイッチを操作したときに実行するメソッドは以下の形となります。

スイッチを操作したときの処理の書き方

```
def メソッド名 (param1, param2)
end
```

メソッド名　　：スイッチを作成したときに設定したメソッド名
param1　　：スイッチの状態
スイッチが Off の場合は 0 となります
スイッチが On の場合は 1 となります
param2　　：スイッチの操作情報 2
常に 0 となります

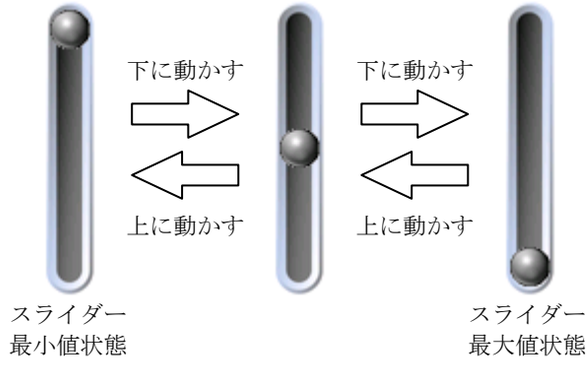
例）スイッチが On のときは LED を点灯して、スイッチが Off のときは LED を消灯します

```
def onTouch2(param1, param2)
   # LED を On/Off
   gr_digitalWrite($PIN_LED0, param1)
end
```

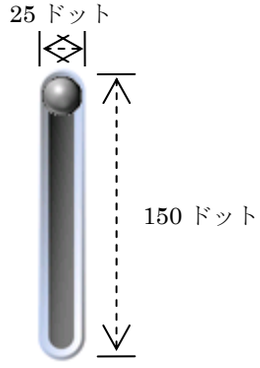
2.5.4 縦スライダー

(1) 表示

画面上に縦方向のスライダーを表示します。タッチパネル押しながら滑らかに上下に動かすことができます。



画面上の大きさは下記となります。



(2) 縦スライダーの作り方

縦スライダーの作り方は下記となります。

縦スライダーの作り方

縦スライダーの名前 = GNVSlider.new(*x 座標* , *y 座標*, *最小値*, *最大値*, "*メソッド名*")

縦スライダーの名前 ： *作成する縦スライダーの名前*
x 座標 ： *ボタンの表示位置 x 座標*
y 座標 ： *ボタンの表示位置 y 座標*
最小値 ： *スライダー最小状態の数値*
最大値 ： *スライダー最大状態の数値*
メソッド名 ： *縦スライダーを操作したときに実行するメソッドの名前*

例）(60, 40) の位置に$slider1 という名前の縦スライダーを作成して、最小値を 0、最大値を 255 とする。スライダー操作時に onTouch3 というメソッドを実行します。

```
$slider1 = GNVSlider.new(60, 40, 0, 255, "onTouch3")
```

［途中のページは省略します］

2.6 画面描画メソッド

2.6.1 画面描画機能

表示部品を使わずに直接液晶画面にグラフィック表示を行う機能もあります。
画面表示部品との違いは、クラスを使用しないためインスタンスを生成する必要はありません。メソッドを呼ぶだけで画面描画を行うことができます。

液晶画面に描画を行うメソッドの一覧は下記となります。

名前	メソッド名	機能
画像表示	GNDrawImage	SD カード上の画像ファイルを表示します。
矩形表示	GNDrawRect	画面上に指定した色の矩形を表示します。

2.6.2 画像表示

(1) 機能概要

マイコンボード上に搭載した SD カード上の画像データを画面上に表示します。

(2) 画像データの変換

表示可能な画像データはピクチャ部品で使用するものと同じものとなります。画像データの作成方法は 2.5.8(3)項を参照してください。

(3) メソッドの使い方

メソッドの使い方を下記に示します。

画像表示メソッドの書き方

GNDrawImage(*画像表示ｘ座標*, *画像表示ｙ座標*, *画像データファイル名*)

例）画面上の(50, 100)の位置に"test.gbf"という名前の画像ファイルを表示する。

```
GNDrawImage(50, 100, "test.gbf")
```

2.6.3 矩形表示

(1) 機能概要

画面上の任意の位置に指定した色の矩形を表示します。

(2) メソッドの使い方

メソッドの使い方を下記に示します。

矩形表示メソッドの書き方

```
GNDrawRect(矩形表示ｘ座標，矩形表示ｙ座標，矩形幅，矩形高さ，表示色Ｒ，表示色Ｇ，表示色Ｂ)
```

表示色の指定は、光の三原色である赤(R),緑(G),青(B)の値で行います。それぞれ 0～31 の 32 段階の数値を指定可能です。

例）画面上の(50, 100)の位置に幅 30、高さ 20 の青色の矩形を表示する。

```
GNDrawRect(50, 100, 30, 20, 0, 0, 31)
```

2.7 写真表示プログラム

2.7.1 プログラムの動作内容

液晶画面を使って自分の好きな写真を表示するプログラムを作成します。

2.7.2 画像データの作成

自分で用意した画像ファイルを液晶画面に表示するためには、まず最初に画像データの作成処理を行います。

画像データを作成するためには、PNG 形式の画像ファイルを用意する必要があります。ここでは、下記の名前の PNG 形式画像ファイルを用意します。

bluesky.png
flower.png
green.png
leaf.png
tile.png

上記の PNG 形式画像ファイルは、インストールフォルダ内の **sample¥sample05** フォルダに格納されています。
画像データを変換するためにインストールフォルダ内の **tool** フォルダに格納されている**″mrbConvert.exe″**というプログラムを実行してください。起動した画面に、上記のファイルを全てドラッグ＆ドロップすると拡張子が**″.gbf″**形式のファイルに変換されます。

変換した結果のファイルも、インストールフォルダ内の **sample¥sample05** フォルダに格納されています。

2.7.3 プログラムの作成

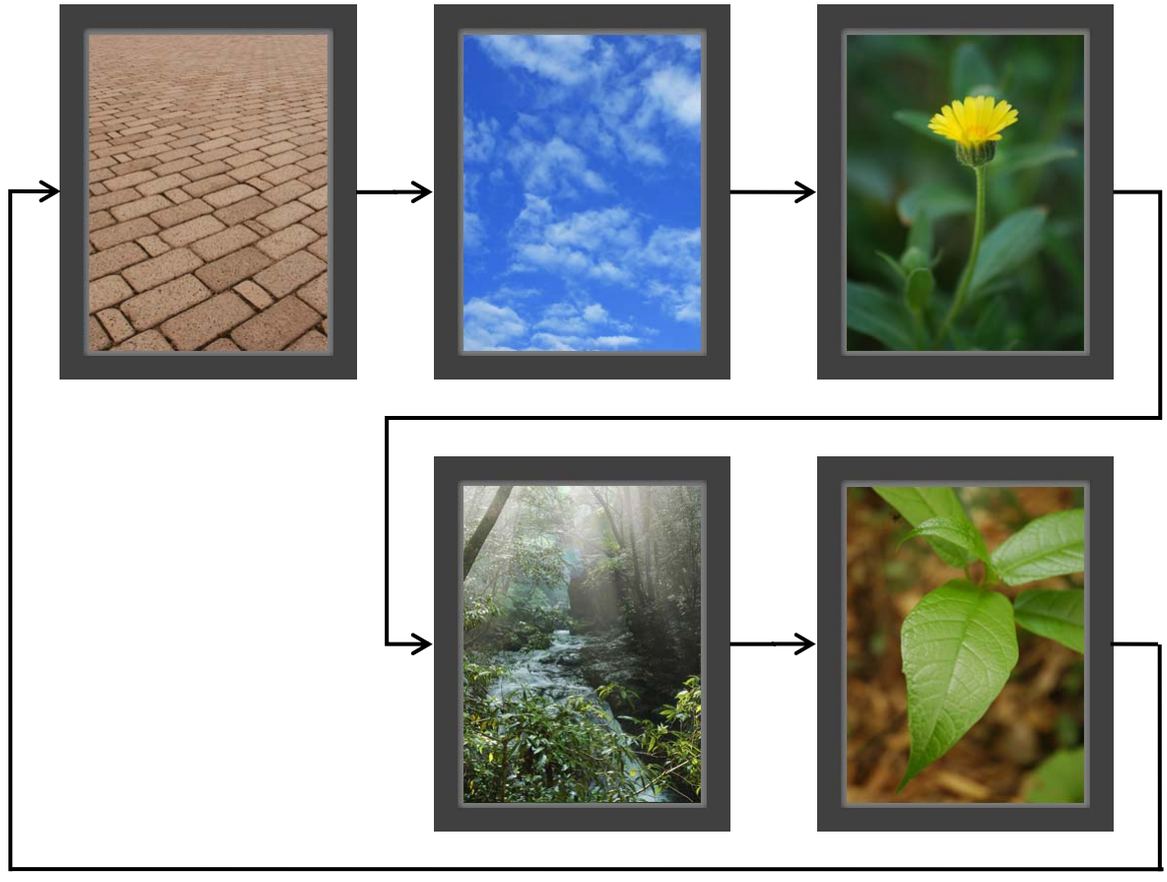
下記のプログラムを作成して実行してください。

```
#================================================================================
# メイン初期化
#--------------------------------------------------------------------------------
def setup()
    $counter = 0
    $photoNo = 0
end

# メインループ
#--------------------------------------------------------------------------------
def loop()
    if $counter == 0 then
        case $photoNo
            when 0 then
                GNDrawImage(0, 0, "tile.gbf")
            when 1 then
                GNDrawImage(0, 0, "bluesky.gbf")
            when 2 then
                GNDrawImage(0, 0, "flower.gbf")
            when 3 then
                GNDrawImage(0, 0, "green.gbf")
```

```
          when 4 then
             GNDrawImage(0, 0, "leaf.gbf")
       end
    end


   $counter += 1
    if $counter >= 50000 then
       $counter = 0
       $photoNo += 1
       if $photoNo >= 5 then
          $photoNo = 0
       end
    end
end
```

インストールフォルダ内の **sample¥sample05** フォルダに"**photoFrame.rb**"という名前で同じ内容のファイルと、そのコンパイル結果のファイル"**program.mbi**"が格納されています。

実行すると下記のような画面が順番に液晶画面に表示されます。

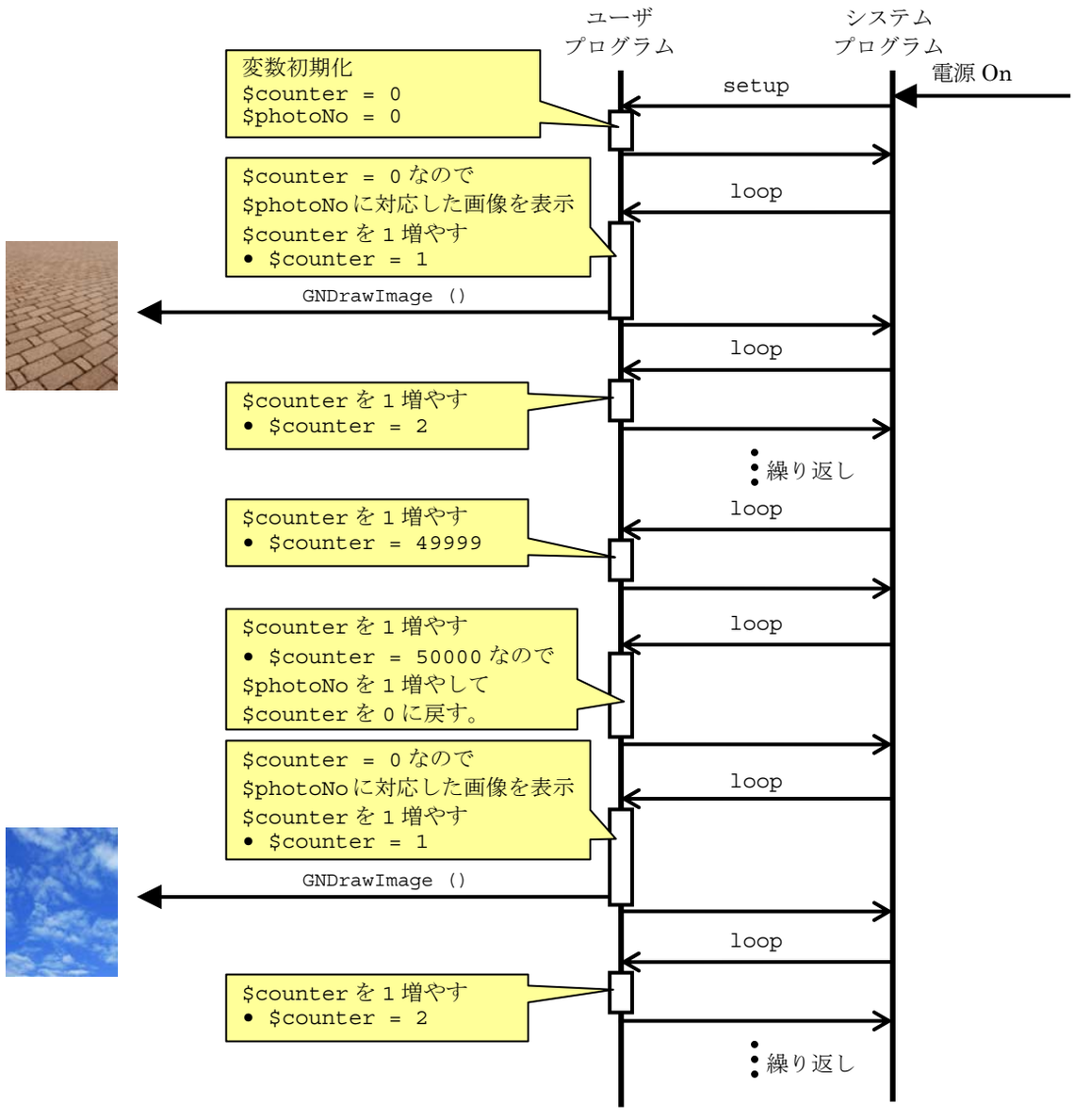


2.7.4 プログラム内容の説明

時間経過とともに 5 種類の画像を順番に表示します。表示する画像を指定するために$photoNo という変数を使用します。

また、一定時間毎に$photoNo を切換えるために周期処理の中でループの回数をカウントします。loop メソッドを実行するたびに変数$counter を 1 ずつ増加させます。$counter の値が 50000 以上になると$photoNo の値を 1 増やして、$counter を 0 に戻します。

これを繰り返すことによって、一定時間ごとに写真を繰り返して表示します。



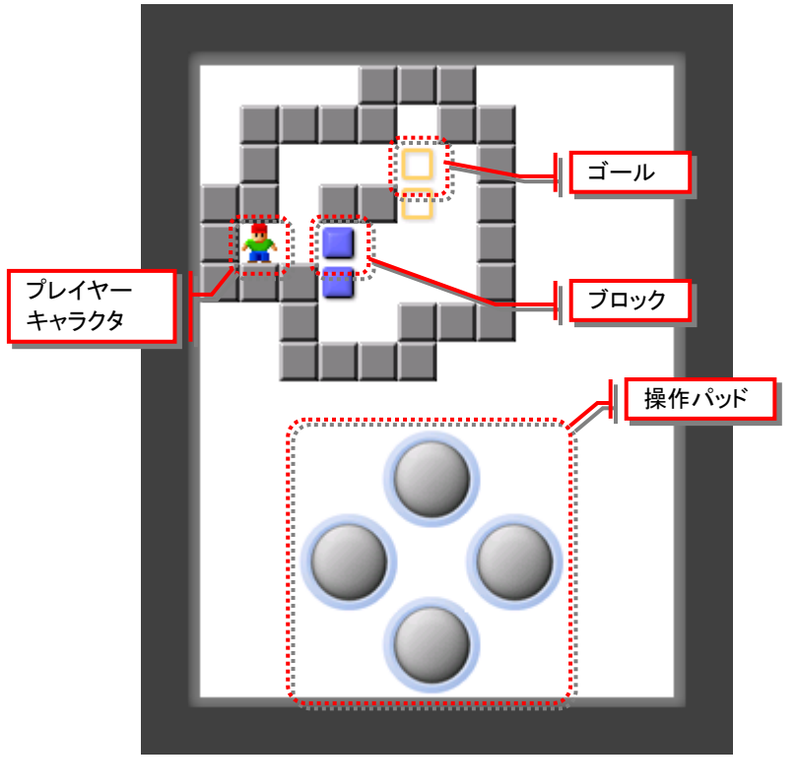
2.8 タッチパネルを使ったゲーム

2.8.1 プログラムの動作内容

本項では、LCD 画面のタッチパネルを使った簡単なゲームを動かす例を説明します。

画面上に４方向ボタンとパズル画面が表示されます。
４方向ボタンを操作してプレイヤーキャラクタを移動させて、画面上の青いブロックをゴールまで運びます。全てのブロックをゴールまで運ぶとクリアとなります。
プレイヤーキャラクタは青いブロックを押すことはできますが、引くことはできません。また、一度に押すことができるブロックは１つだけです。

全てのブロックをゴールまで移動させるとステージクリアとなり、次のステージに進みます。今回はサンプルですのでステージ数は２とします。２つめのステージをクリアすると最初のステージに戻ります。



2.8.2 画像データの準備

画面に表示するキャラクタの画像データを準備します。ここでは、下記の名前の PNG 形式画像ファイルを用意します。

	ファイル名	画像	大きさ
プレイヤーキャラクタ	player.png		20x20
ブロック	block.png		20x20
ゴール	goal.png		20x20
壁	wall.png		20x20
床	floor.png	 （白色の画像）	20x20
クリア時メッセージ	clear.png		80x20

上記の PNG 形式画像ファイルは、インストールフォルダ内の **sample¥sample06** フォルダに格納されています。
また、画像変換した結果の拡張子が"**.gbf**"のファイルも、同じフォルダに格納されています。

2.8.3 プログラムの作成

下記のプログラムを作成して実行してください。
同じ内容のプログラムファイルがインストールフォルダ内の sample¥sample06 フォルダに"blockgame.rb"という名前で格納されています。

```
# 状態遷移の定義
$ST_CREATE = 0
$ST_PLAY   = 1
$ST_CLEAR  = 2

# ステージデータの定義
$DT_FLOOR  = 0
$DT_WALL   = 1
$DT_PLAYER = 2
$DT_BLOCK  = 3
$DT_GOAL   = 4

# 画面上の表示物の大きさ
$UNIT_SCALE = 20

# ステージデータ
$totalStage = 2
$stageData = [
    [
        0, 0, 1, 1, 1, 1, 0, 0,
```

［途中のページは省略します］

2.8.4 プログラム内容の説明

(1) クラス定義

画面上で移動する物体を管理するための画面表示物クラスを作成します。

画面表示物クラスは下記を管理するために使用します。

・プレイヤーキャラクタ

・ブロック

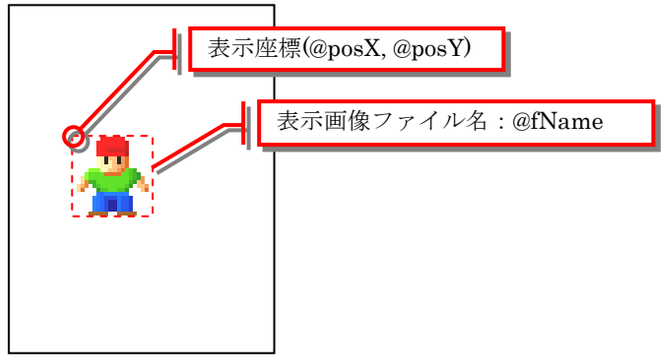
```
# 画面表示物クラス
class DisplayObject
    attr_accessor :posX, :posY

    # 初期化処理
    def initialize(x, y, fName)
        @fName = fName
        setPos(x, y)
    end

    # 位置設定
    def setPos(x, y)
        @posX = x
        @posY = y
        GNDrawImage(@posX * $UNIT_SCALE, @posY * $UNIT_SCALE, @fName)
    end

    # 移動
    def moveTo(x, y)
        # 移動元を描画
        if getStageData(@posX, @posY) == $DT_GOAL then
            drawGoal(@posX, @posY)
        else
            drawFloor(@posX, @posY)
        end

        # 現在位置を移動、移動先に画像を描画
        setPos(x, y)
    end
end
```

attr_accessor に続けて *:インスタンス変数名* を記述することによって、インスタンス変数を直接外部から読み書き可能となります。

画面表示物クラスに関する説明を以下に記述します。

■インスタンス変数
インスタンス変数として下記の３つを持ちます。

@posX： *表示ｘ座標*
@posY： *表示ｙ座標*
@fName： *表示画像ファイル名*



■インスタンスメソッド
インスタンスメソッドは下記となります。

initialize メソッド

```
initialize(x, y, fName)
        x       : 表示ｘ座標
        y       : 表示ｙ座標
        fName   : 表示画像ファイル名(gbf 形式)
```

初期化処理です。インスタンス生成時に引数として渡された x, y 座標と表示画像ファイル名をインスタンスメソッドに格納します。

表示位置設定メソッド

```
setPos(x, y)
        x       : 表示ｘ座標
        y       : 表示ｙ座標
```

表示位置設定処理です。引数で指定した座標に移動し、そこに画像ファイルの内容を表示します。

移動処理メソッド

```
moveTo(x, y)
        x       : 表示ｘ座標
        y       : 表示ｙ座標
```

移動処理です。移動前の位置の表示を消去して、移動後の位置に表示を行います。
移動前の位置にゴールがあればゴールを表示して、そうでなければ床を表示します。

(2) 状態遷移

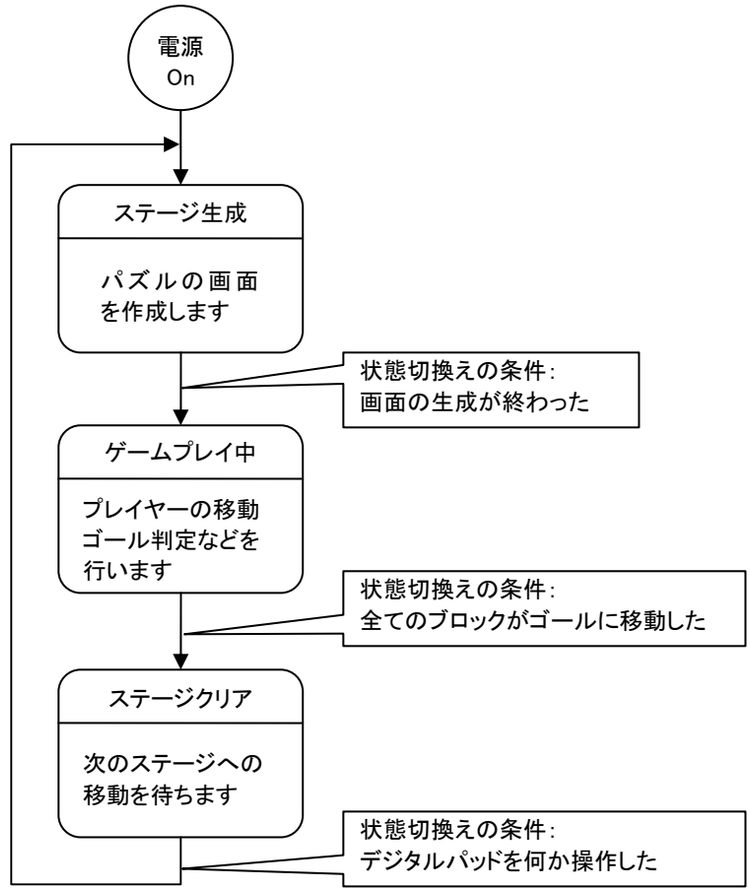
パズルゲームを作るにあたって、ゲームの状態を管理します。
その時のゲームの状態によって処理内容を決めます。また、条件によって状態を切換えます。これを、状態遷移設計と呼びます。
今回のゲームでは、下記の３つのゲーム状態を持つことにします。

・ステージ生成状態
パズルゲームのステージを生成する状態です。ステージの生成が終わるとゲームプレイ中状態に切換えます。

・ゲームプレイ中状態
パズルゲームをプレイ中の状態です。デジタルパッドを操作してプレイヤーキャラクタを移動させることができます。
すべてのブロックをゴールまで移動させると、ステージクリア状態に切換えます。

・ステージクリア状態
ステージクリア時の状態です。ステージクリアメッセージを表示し、デジタルパッドを操作するとステージ生成状態に切換えます。

上記の状態の関連図を以下に示します。



プログラムの中ではゲーム状態を以下の変数に格納して管理します。

```
$state
```

ゲーム状態の値は以下の変数を使います。

```
$ST_CREATE = 0     ステージ生成状態
$ST_PLAY   = 1     ゲームプレイ中状態
$ST_CLEAR  = 2     ステージクリア状態
```

［途中のページは省略します］

第３章
プログラム作成：応用編

本章では、付属のマイコンボードと色々な電子部品を組み合わせて動作するプログラムの例を説明します。必要な電子部品の一覧も記載しますので、別途それらの部品を購入いただければ実際に動作させることができます。

3.1 本章の使い方

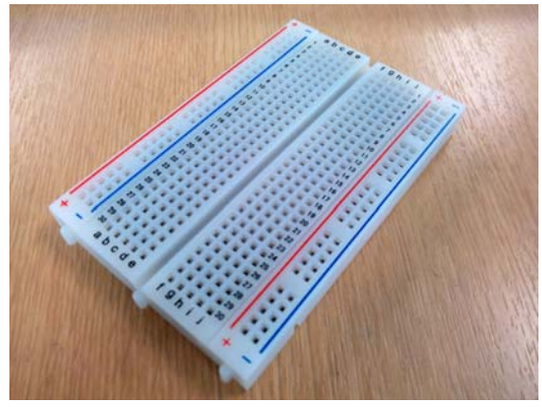
本章では、マイコンボードの各端子と色々な電子部品を接続して電子回路を作成する方法を説明します。
色々な電子回路について、必要な部品一覧と接続方法、プログラムソースコードを記載しますので気に入ったものがあれば実際に作ってみてください。
また、それらを応用して自分だけの電子回路を作って動かしてみましょう。

3.2 ブレッドボードの使い方

3.2.1 ブレッドボードとは

ここでは手軽に電子回路を作るためにブレッドボードというものを使います。

ブレッドボードとは、たくさんの穴があいたボードです。電子部品やリード線を直接挿し込むことによって電子回路を作ることができるので、ハンダ付けが不要です。

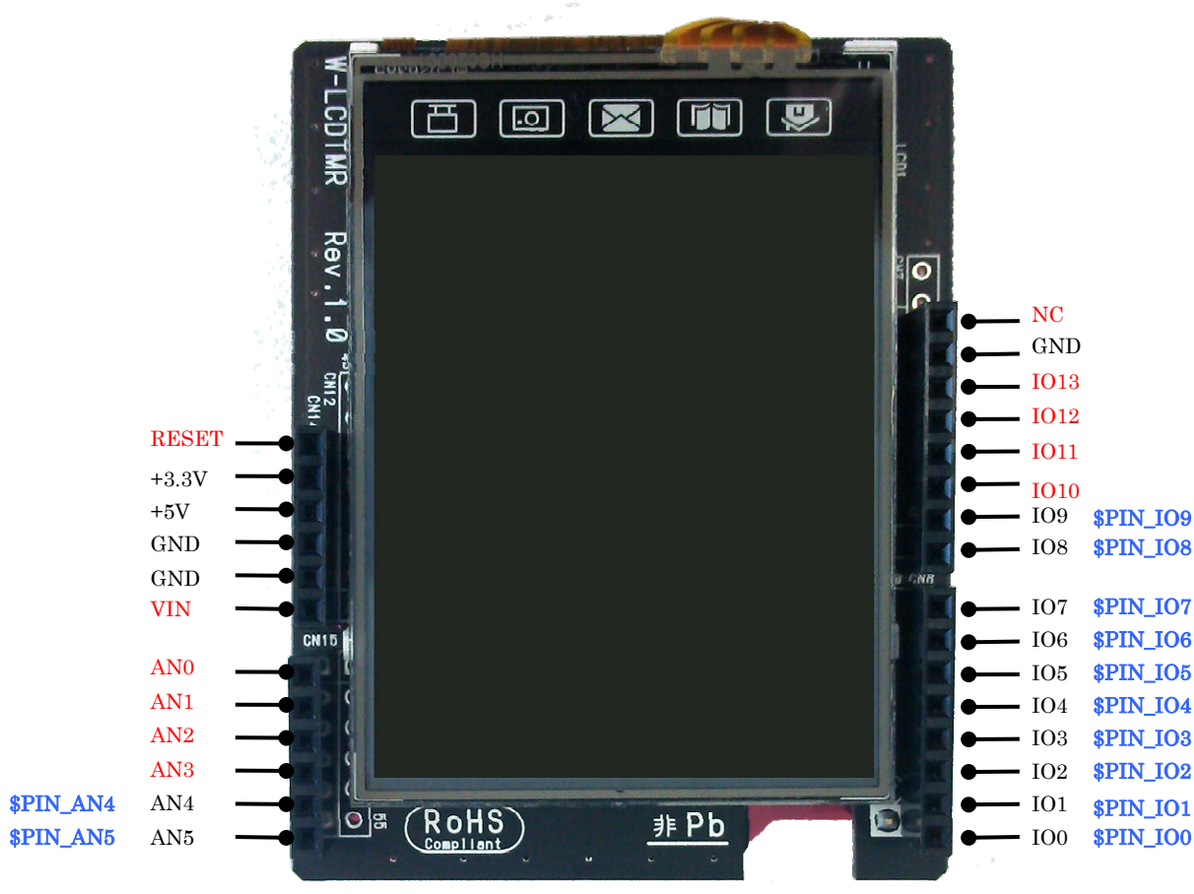


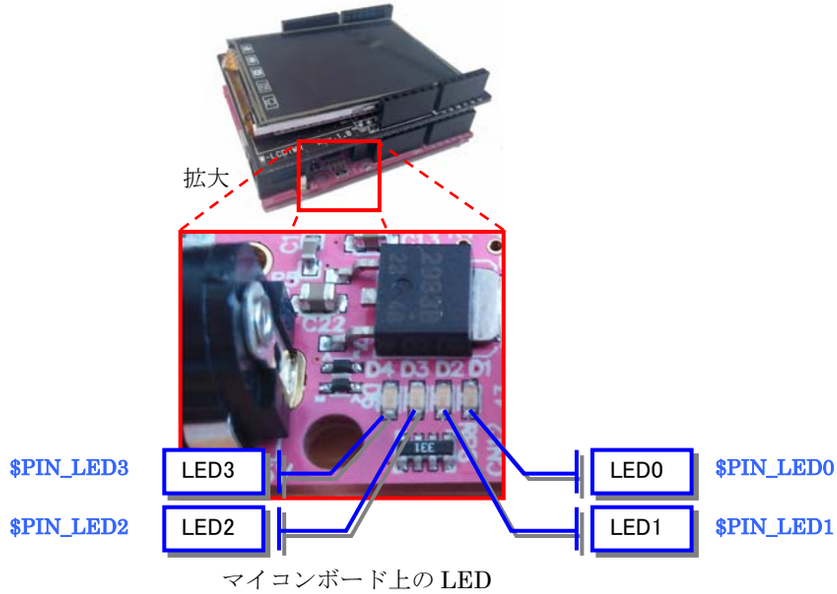
表面にある穴はブレッドボード内部で電気的に接続されています。規則的に接続されているので、それを考慮した上で電子部品を配置することで電子回路を作成することができます。

［途中のページは省略します］

3.3 マイコンボードの端子と変数名の対応

入出力端子の詳細を以下に示します。赤字で表示している端子は使用しません。
プログラムから使用するときは青い文字のグローバル変数名で端子を指定します。





マイコンボード上の LED

端子名称	用途
RESET	使いません
+3.3V	3.3V の電源が得られます
+5V	5V の電源が得られます
GND	グラウンドとなります
VIN	使いません
AN0〜AN5	アナログ入力端子です。AN0〜AN3 は使用できません。
IO0〜IO13	デジタル入出力端子です。IO10〜IO13 は使用できません。
NC	使いません
LED0〜LED3	LED です。点灯させることができます。

3.4 マイコンボードの端子の使い方

マイコンボード上の端子 IO0～IO9 および AN4～AN5 をプログラムから使うことができます。

(1) デジタル入出力端子

IO0～IO9 の 10 本の端子はデジタル入出力端子です。設定によって入力にも出力にも使用することができます。
また、出力を行う場合はデジタル(On/Off)の出力と PWM 制御によるアナログ出力を行うことができます。

デジタル入出力端子に関して使用可能なメソッドの一覧は下記となります。

名前	メソッド名	機能
入出力切換え	gr_pinMode	指定した端子の入出力動作を設定します。
デジタル入力	gr_digitalRead	端子からデジタル値を読み込みます。
デジタル出力	gr_digitalWrite	端子へデジタル値を出力します。
アナログ出力	gr_analogWrite	端子へ PWM 制御によるアナログ値を出力します。
矩形波出力	gr_tone	端子へ指定周波数の矩形波を出力します。 圧電ブザーで音を鳴らすのに便利です。
矩形波出力停止	gr_noTone	矩形波出力を停止します。

■入出力切換え

デジタル入出力端子を使う前に、その端子を入力として使用するか出力として使用するかを指定する必要があります。

入出力切換えメソッドの書き方

gr_pinMode(*端子番号, 入出力モード*)

端子番号　：　端子番号(端子の対応は 3.3 項の図を参照してください)
入出力モード　　：　入力の場合：$INPUT
出力の場合：$OUTPUT

例）端子 IO4 を出力モードで使用する場合

```
gr_pinMode($PIN_IO4, $OUTPUT)
```

端子 IO3 を入力モードで使用する場合

```
gr_pinMode($PIN_IO3, $INPUT)
```

LED0 を出力モードで使用する場合(LED を入力モードで使用することはできません)

```
gr_pinMode($PIN_LED0, $OUTPUT)
```

■デジタル入力

入出力モードを入力に設定した端子からの入力値を取得します。

デジタル入力メソッドの書き方

gr_digitalRead(*端子番号*)

端子番号　：　端子番号(端子の対応は 3.3 項の図を参照してください)
戻り値：　HIGH の場合　：　1
LOW の場合　：　0

例）端子 IO3 の入力を取得する場合

```
value = gr_digitalRead($PIN_IO3)
```

変数 value に入力値が格納されます。

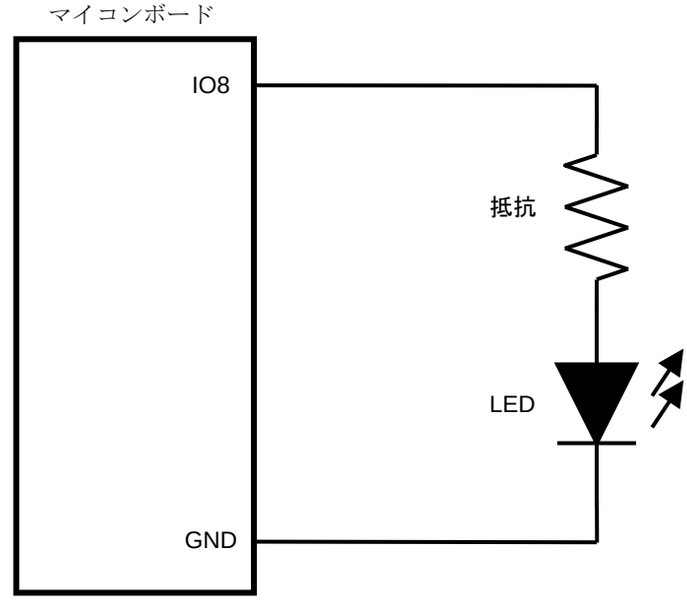
［途中のページは省略します］

3.5 LEDを点滅させる

実際にブレッドボードを使って簡単な電子回路を作ってみます。
マイコンボードを使って最初に作った LED 点滅プログラムを応用して、ブレッドボード上に配置した LED を点滅させます。

必要な部品一覧
・ブレッドボード　　１個
・LED　　１個
・抵抗(220～470Ω程度)　１個
・ジャンパワイヤ線　　２本

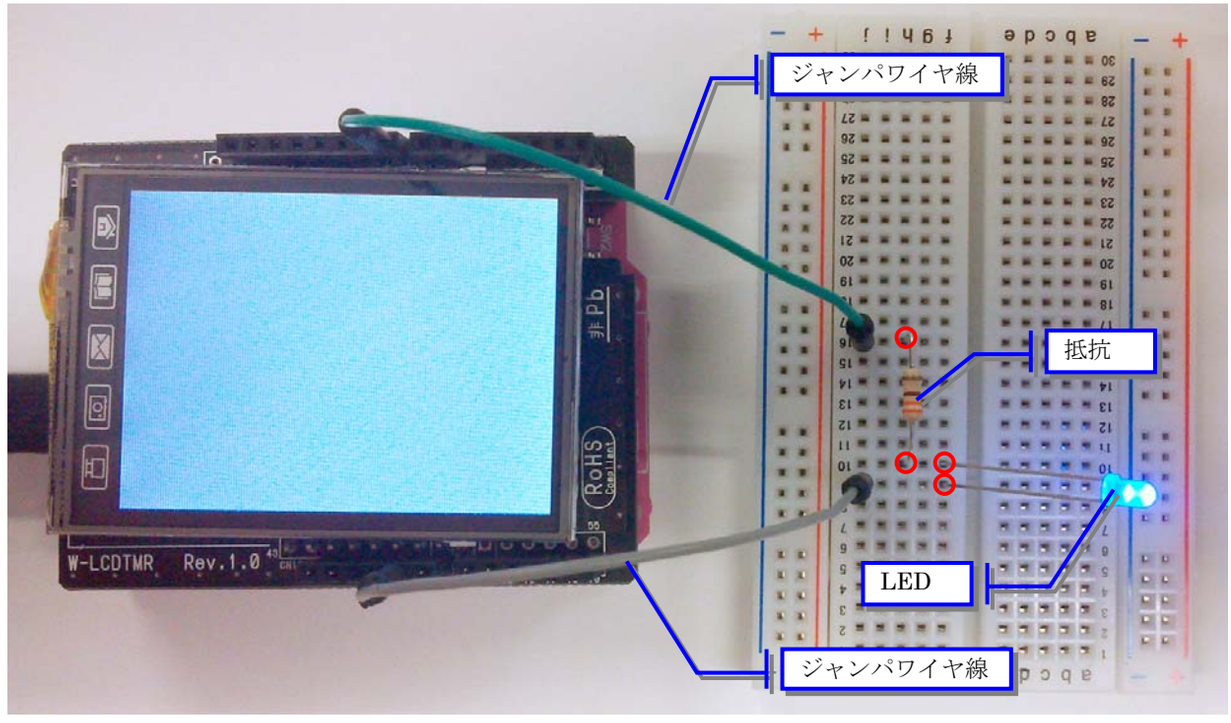
(1) 回路図



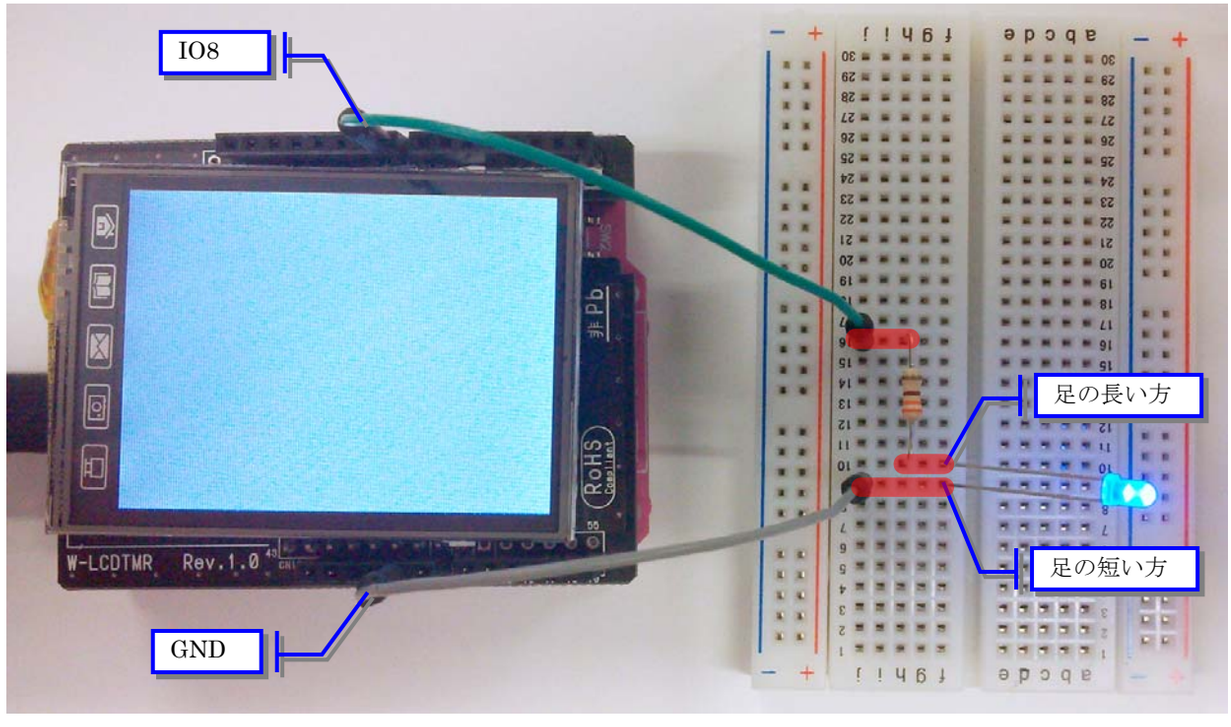
マイコンボードの IO8 番ピンの出力を On/Off 切換えを行って LED を点滅させます。
マイコンボードのピン出力を直接 LED に接続すると電圧が高すぎて LED が壊れる可能性があるため、抵抗を間に接続します。

(2) 接続図

赤丸部分に抵抗と LED の足を差し込んでいます。



赤い線の部分がブレッドボード内で接続されています。
LED の２本の足は長い方を+電源(PIN8)側に接続してください。



(3) プログラムの作成

下記のプログラムを作成して実行してください。

同じ内容のプログラムファイルがインストールフォルダ内の **sample¥sample07** フォルダに"**led.rb**"という名前で格納されています。

```
# メイン初期化
#--------------------------------------------------------------------------------
def setup()
    # IO8 を出力設定とする
    gr_pinMode($PIN_IO8, $OUTPUT)
end

# メインループ
#--------------------------------------------------------------------------------
def loop()
    # IO8 の出力を On とする
    gr_digitalWrite($PIN_IO8, 1)
    gr_delay(100)

    # IO8 の出力を Off とする
    gr_digitalWrite($PIN_IO8, 0)
    gr_delay(100)
end
```

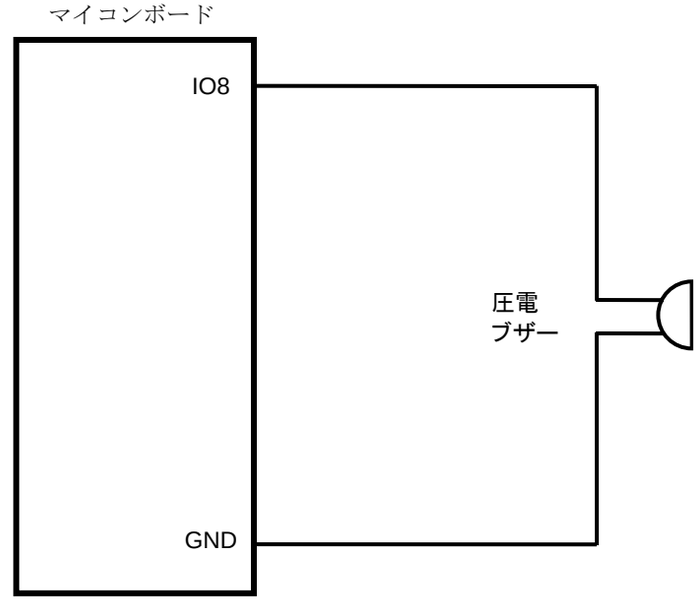
プログラムの動作内容は第１章で作成した最初のプログラムと同様となります。

3.6　電子ピアノを作る

本項では、圧電スピーカーを接続して音を鳴らす例を説明します。LCD 画面上のボタンを押すとボタンに対応した音階の音が鳴ります。

必要な部品一覧
・ブレッドボード　　１個
・圧電ブザー　　１個
・ジャンパワイヤ線　　２本

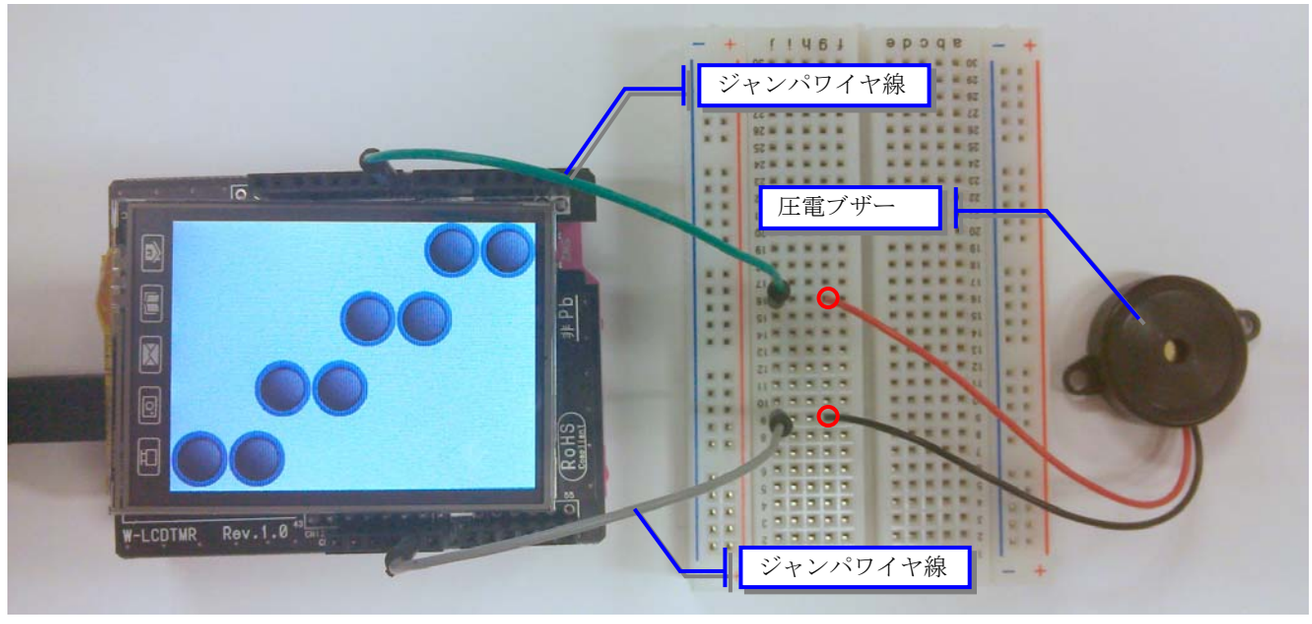
(1)　回路図



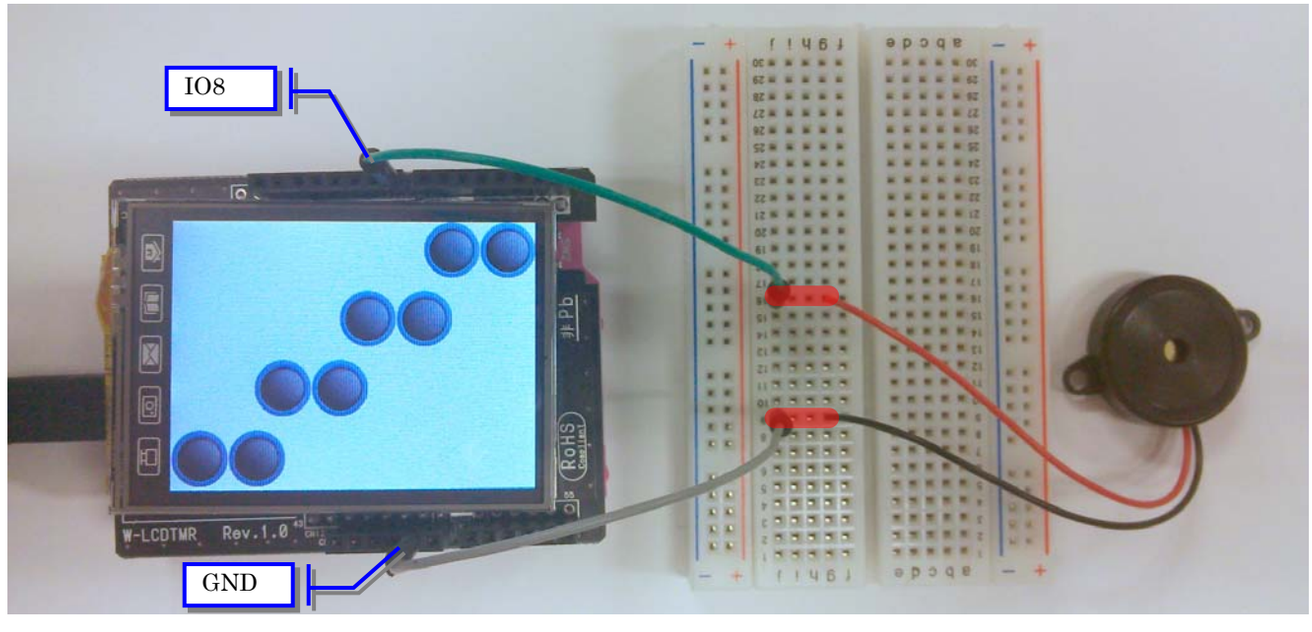
マイコンボードの IO8 番ピンの出力を PWM によるアナログ出力を行って圧電ブザーを鳴らします。液晶画面に表示したボタンによって圧電ブザーを鳴らす周波数を決めます。

(2) 接続図

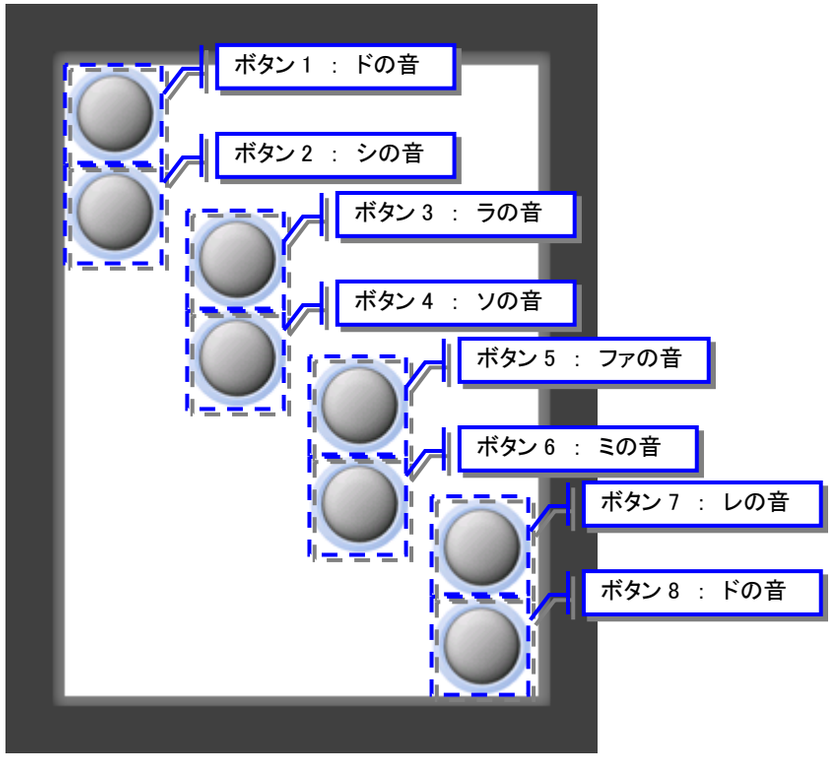
赤丸部分に圧電ブザーの線を差し込んでいます。



赤い線の部分がブレッドボード内で接続されています。



(3) 液晶画面表示


ボタン１ ： ドの音
ボタン２ ： シの音
ボタン３ ： ラの音
ボタン４ ： ソの音
ボタン５ ： ファの音
ボタン６ ： ミの音
ボタン７ ： レの音
ボタン８ ： ドの音

(4) プログラムの作成

下記のプログラムを作成して実行してください。

同じ内容のプログラムファイルがインストールフォルダ内の **sample¥sample08** フォルダに"**piano.rb**"という名前で格納されています。

```
# メイン初期化
#--------------------------------------------------------------------------------
def setup()
    $button1 = GNButton.new(  0,   0, "onTouch1")
    $button2 = GNButton.new(  0,  50, "onTouch2")
    $button3 = GNButton.new( 62,  74, "onTouch3")
    $button4 = GNButton.new( 62, 124, "onTouch4")
    $button5 = GNButton.new(124, 148, "onTouch5")
    $button6 = GNButton.new(124, 198, "onTouch6")
    $button7 = GNButton.new(186, 220, "onTouch7")
    $button8 = GNButton.new(186, 270, "onTouch8")

    gr_pinMode($PIN_IO8, $OUTPUT)
end

# メインループ
#--------------------------------------------------------------------------------
def loop()
end

def onTouch1(param1, param2)
    if param1 == 1 then
        # ドの音
        gr_tone($PIN_IO8, 1046, 0)
    else
        gr_noTone($PIN_IO8)
    end
end

def onTouch2(param1, param2)
    if param1 == 1 then
        # シの音
        gr_tone($PIN_IO8, 986, 0)
    else
        gr_noTone($PIN_IO8)
    end
end

def onTouch3(param1, param2)
    if param1 == 1 then
        # ラの音
        gr_tone($PIN_IO8, 880, 0)
    else
        gr_noTone($PIN_IO8)
    end
end

def onTouch4(param1, param2)
    if param1 == 1 then
        # ソの音
        gr_tone($PIN_IO8, 783, 0)
    else
        gr_noTone($PIN_IO8)
    end
end

def onTouch5(param1, param2)
    if param1 == 1 then
        # ファの音
        gr_tone($PIN_IO8, 698, 0)
    else
        gr_noTone($PIN_IO8)
    end
end
```

［途中のページは省略します］

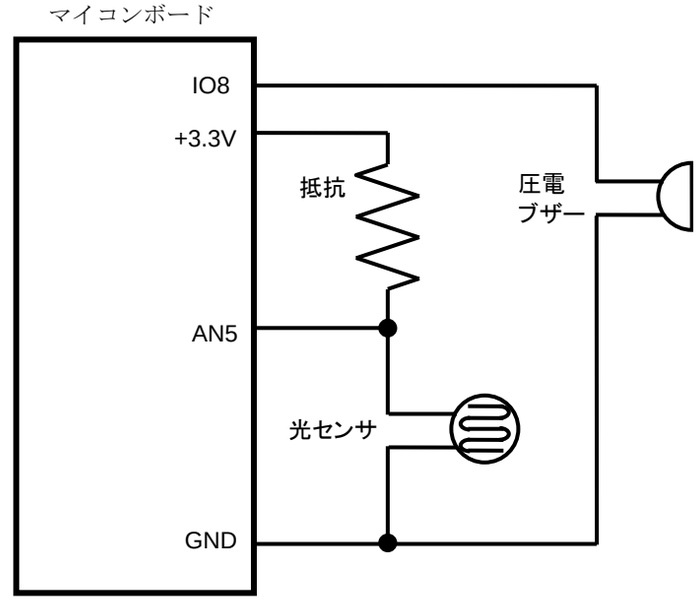
3.7　電子オルゴールを作る

本項では、光センサを使って明るさを検知する例を説明します。ここでは光センサとして CdS と呼ばれるセンサを使います。

CdS とは周囲の明るさによって抵抗値が変化する素子です。抵抗値の変化に伴って変化する電圧の値をマイコンボードで読み取ることによって周囲の明るさを知ることができます。
それを応用して、周囲が明るくなったら音楽を鳴らして周囲が暗くなったら音楽を止めるオルゴールを作ってみます。

必要な部品一覧
・ブレッドボード　　１個
・圧電ブザー　　１個
・抵抗(10KΩ程度)　　１個
・CdS(光センサ)　　１個
・ジャンパワイヤ線　　６本

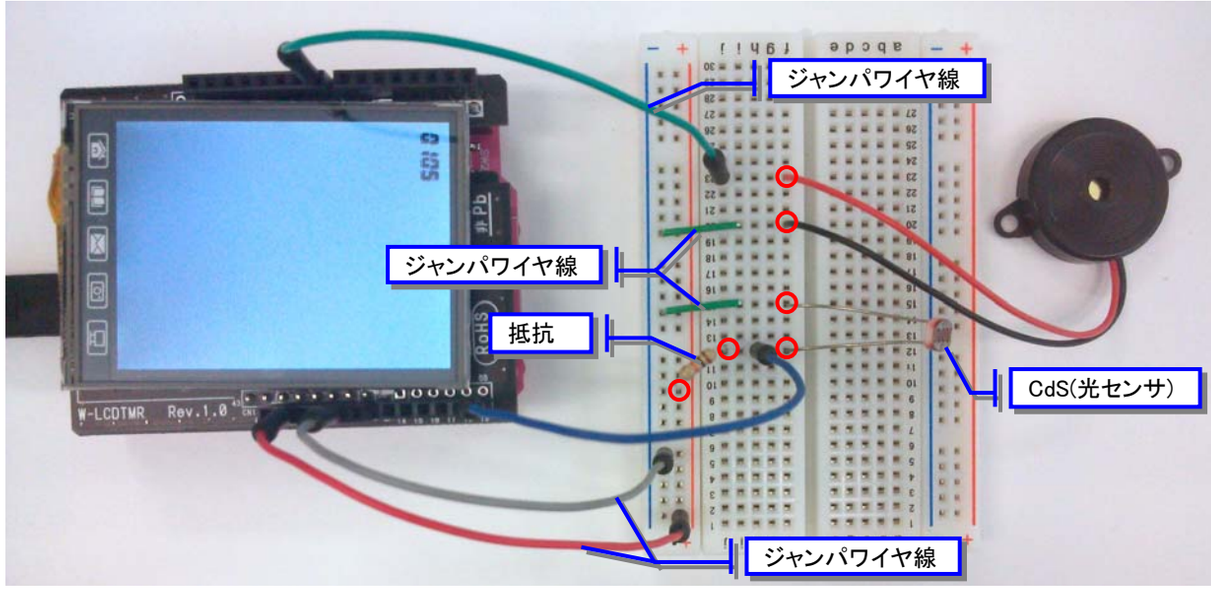
(1) 回路図



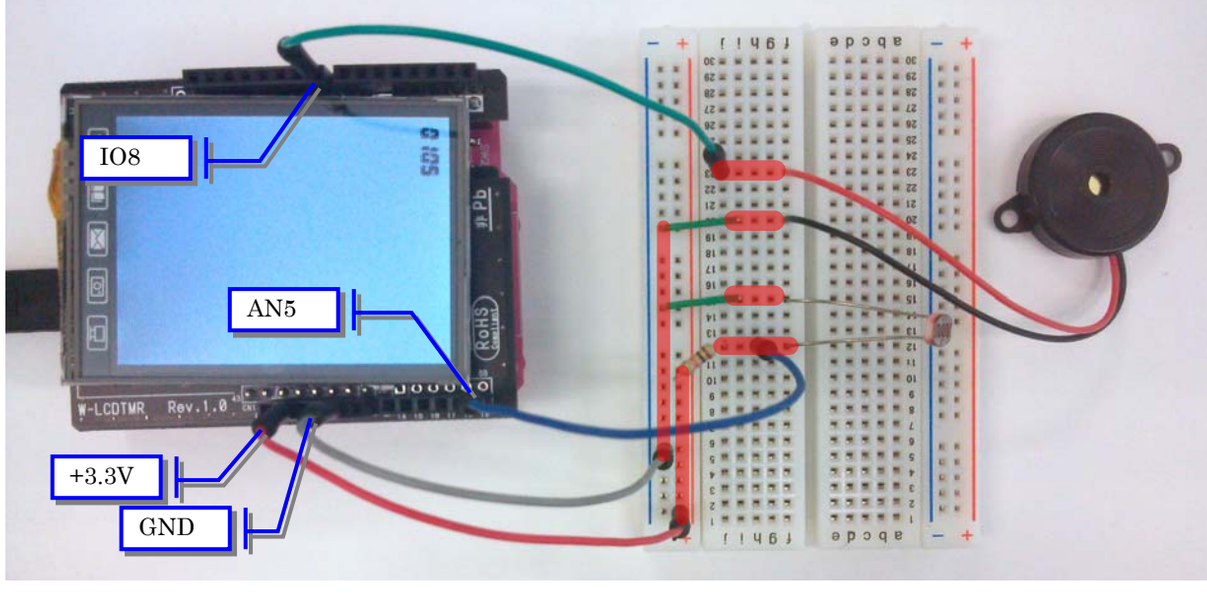
CdS は明るさによって抵抗値が変化しますが、マイコンボードの入力ポート(上記 AN5)が検出できるのは電圧値の変化です。抵抗値の変化を電圧値に変換するために、固定抵抗と CdS の直列回路を作成しています。

(2) 接続図

赤丸部分に抵抗と圧電ブザーと光センサの足を差し込んでいます。



赤い線の部分がブレッドボード内で接続されています。



(3) 液晶画面表示

(4) プログラムの作成

下記のプログラムを作成して実行してください。

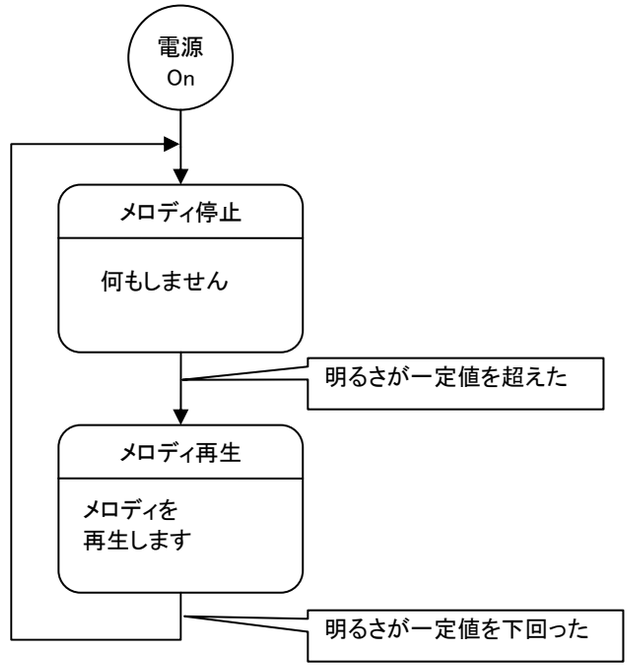
同じ内容のプログラムファイルがインストールフォルダ内の **sample¥sample09** フォルダに"**musicbox.rb**"という名前で格納されています。

```
# 状態の定義
$ST_STOP = 0
$ST_PLAY = 1

# 1:ド 2:レ 3:ミ 4:ファ 5:ソ 6:ラ 7:シ 8:ド
$musicData = [
    1, 1, 5, 5, 6, 6, 5, 0, 4, 4, 3, 3, 2, 2, 1, 0
]
$musicLength = $musicData.size

# 再生と停止を切換える値(周囲の明るさによって調整してください)
$light_val = 150

# メイン初期化
#--------------------------------------------------------------------------------
def setup()
    # 状態の初期化
    $state = $ST_STOP

    # カウンタの初期化
    $tic = 0
    $playCount = 0

    # 明るさ表示部品の生成
    $numbmp1 = GNDigitalNum.new(10, 10)

    gr_pinMode($PIN_IO8, $OUTPUT)
end

# メインループ
#--------------------------------------------------------------------------------
def loop()
    # 明るさの取得
    light = gr_analogRead($PIN_AN5)
    $numbmp1.setInteger(light)

    case $state
    when $ST_STOP
        # 停止状態の場合
        if light < $light_val then
            $state = $ST_PLAY
            $tic = 0
            $playCount = 0
        end
    when $ST_PLAY
        # 再生状態の場合
        if light >= $light_val then
            $state = $ST_STOP
        end
    end

    # 再生状態ならばメロディの再生を行います
    if $state == $ST_PLAY then
        $tic += 1
        if $tic >= 300 then
            $tic = 0

            playMelody($musicData[$playCount])
            $playCount += 1
            if $playCount >= $musicLength then
                $playCount = 0
            end
        end
    end
```

［途中のページは省略します］

(5) 状態遷移

周囲の明るさによって状態遷移を行います。



プログラムの中ではゲーム状態を以下の変数に格納して管理します。

```
$state
```

ゲーム状態の値は以下の変数を使います。

```
$ST_STOP = 0      メロディ停止状態
$ST_PLAY = 1      メロディ再生状態
```

(6)明るさの取得と判定

明るさの取得はメインループで行っています。

```
# 再生と停止を切換える値(周囲の明るさによって調整してください)
$light_val = 150

（中略）

# メインループ
#--------------------------------------------------------------------------------
def loop()
   # 明るさの取得
   light = gr_analogRead($PIN_AN5)
    $numbmp1.setInteger(light)

   case $state
   when $ST_STOP
      # 停止状態の場合
      if light < $light_val then
         $state = $ST_PLAY
         $tic = 0
         $playCount = 0
      end
   when $ST_PLAY
      # 再生状態の場合
      if light >= $light_val then
         $state = $ST_STOP
      end
   end

（中略）

end
```

電圧値を端子 AN5 から取得して、
取得した値を画面に表示します。

メロディ停止状態の時に、電圧値が一定値を
下回った(明るくなった)ならば、状態をメロディ
再生状態に切換えます。

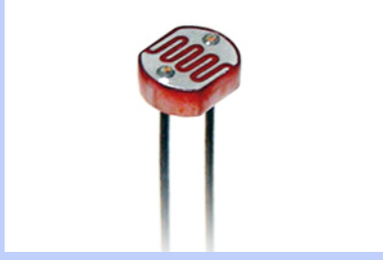
メロディ再生状態の時に、電圧値が一定値を
上回った(暗くなった)ならば、状態をメロディ
停止状態に切換えます。

状態を切換えるための判定値は$light_val という変数に入っています。この値を境界値として
メロディの再生と停止の判定を行います。
再生と停止がうまく切り換わらない場合は$light_val の値を調整してください。

画面上に現在の電圧値が表示されていますので、明るくした場合と暗くした場合に表示される値を
確認して、それらの中間の値を$light_val に設定するようにプログラムを修正してください。

Memo ： CdS とは

CdS 光導電セルとは、光を当てることで半導体内の電流の流れに変化が生じ抵抗値が下がる、という性質をもった素子です。



3.8 傾きを検知する

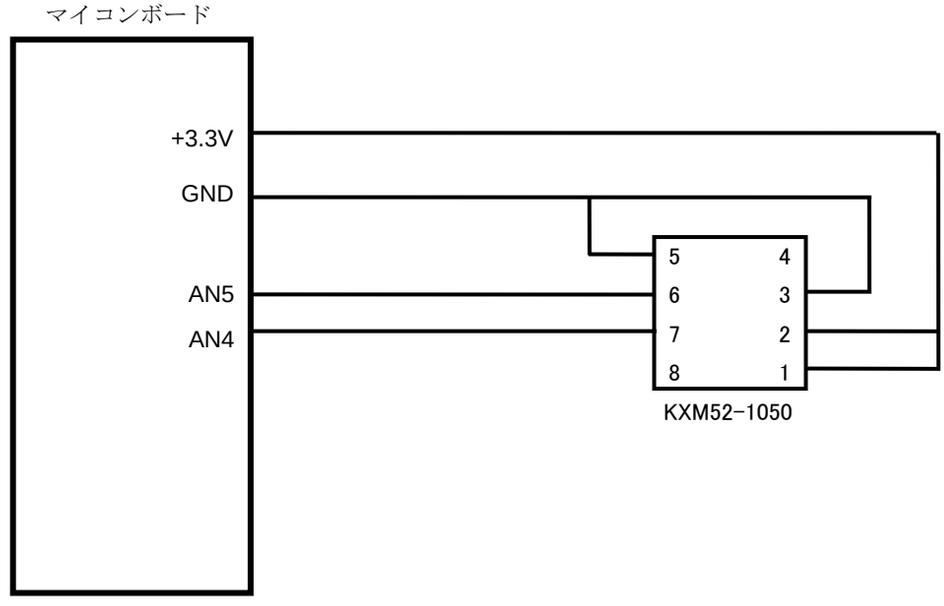
本項では、加速度センサーを使って傾きを検知する例を説明します。ここでは、加速度センサーとしてKXM52-1050(秋月電子通商 製)という名前のセンサーを使います。

加速度センサーはX, Y, Zの３軸の傾きを検出することができます。ここでは、X, Yの２軸の傾きを検出して、液晶画面上のグラフィックを動かすプログラムを作成します。

必要な部品一覧

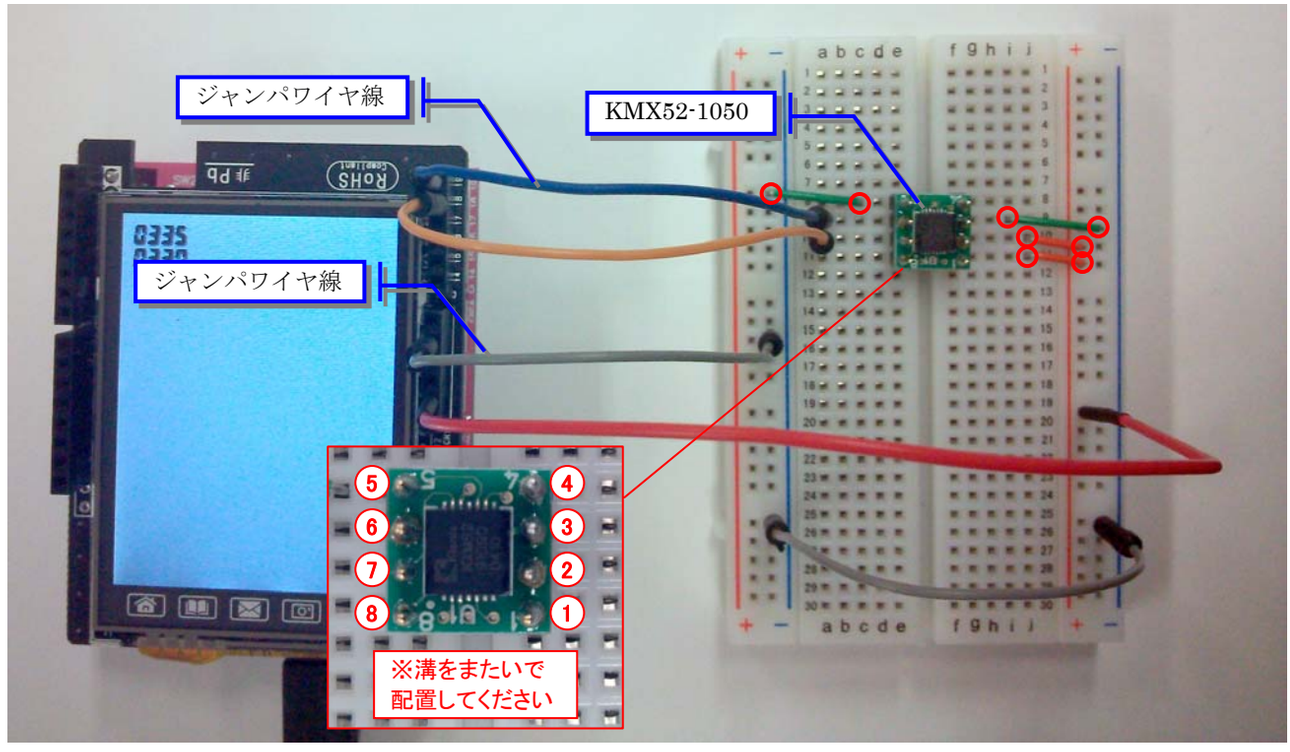
・ブレッドボード　１個
・加速度センサー(KXM52-1050)　１個
・ジャンパワイヤ線　９本

(1) 回路図

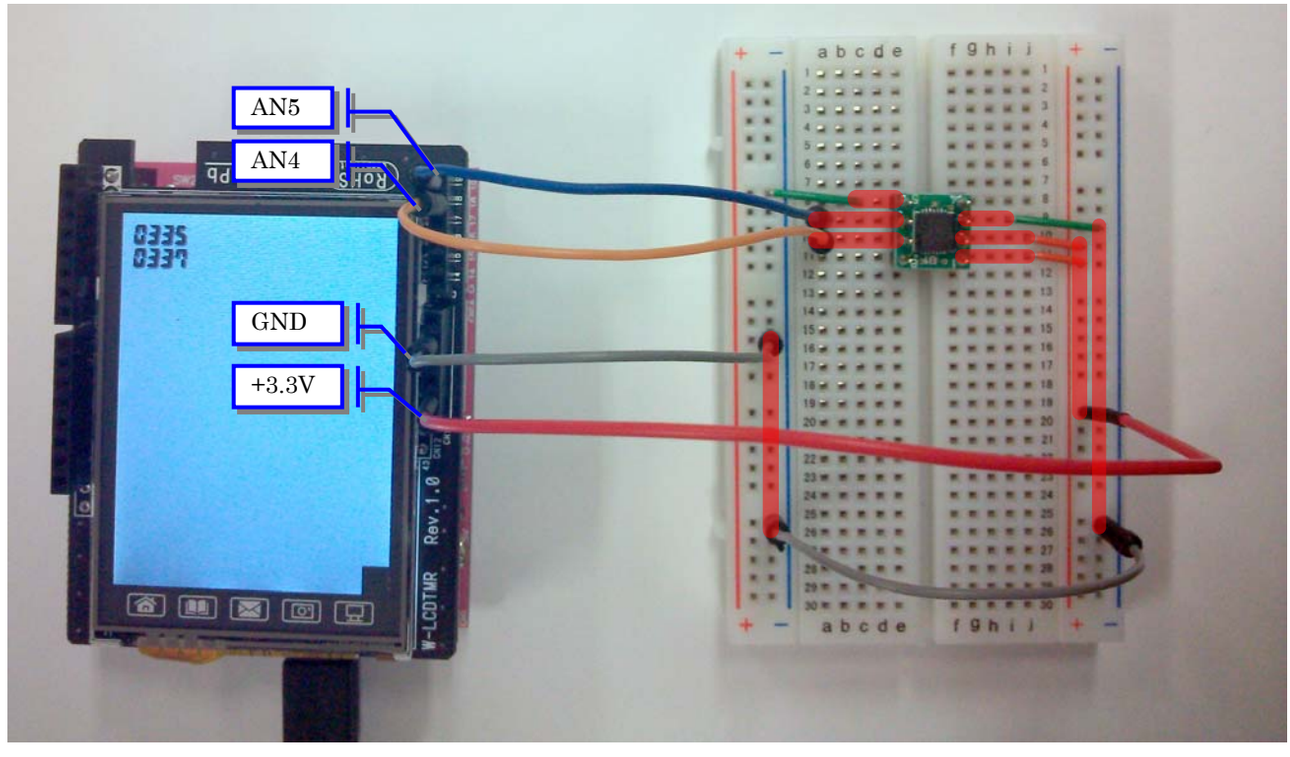


(2) 接続図

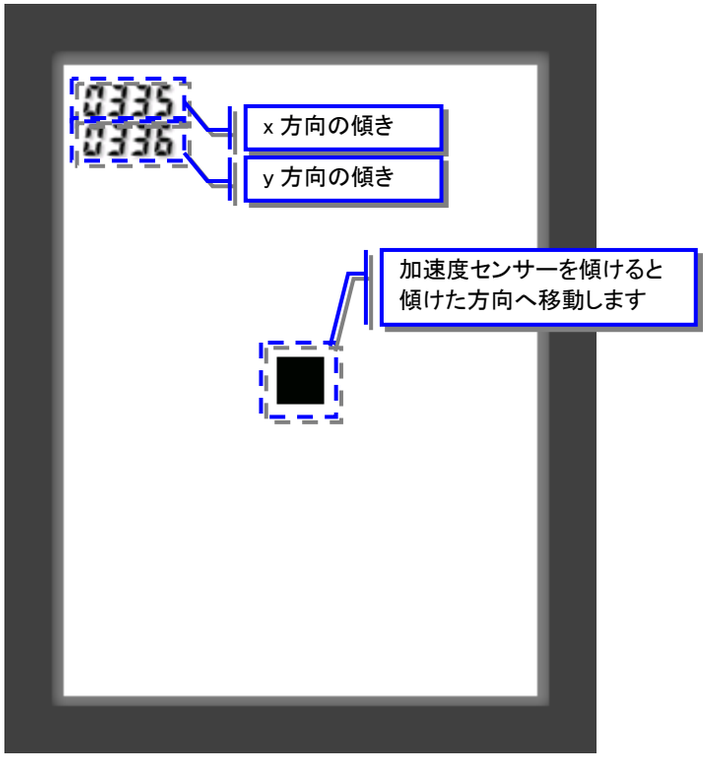
赤丸部分にジャンパワイヤ線を差し込んでいます。KMX52-1050 の配置と向きに注意してください。



赤い線の部分がブレッドボード内で接続されています。



(3) 液晶画面表示



(4) プログラムの作成

下記のプログラムを作成して実行してください。

同じ内容のプログラムファイルがインストールフォルダ内の **sample¥sample10** フォルダに"**accelometer.rb**"という名前で格納されています。

```
# 傾きの中心値の設定(センサを水平な場所に置いて調整してください)
$centerX = 336
$centerY = 336
$margin  = 4

# 表示する矩形の大きさ
$sizeX = 24
$sizeY = 24

# メイン初期化
#--------------------------------------------------------------------------------
def setup()
    # 傾きの値を表示する部品を配置
    $numbmp1 = GNDigitalNum.new(10, 10)
    $numbmp2 = GNDigitalNum.new(10, 30)

    # 表示位置
    $posX = 120
    $posY = 160
    $newX = 120
    $newY = 160

    # 移動範囲
    $posXmin = 0
    $posYmin = 0
    $posXmax = 240 - $sizeX
    $posYmax = 320 - $sizeY
end

# メインループ
#--------------------------------------------------------------------------------
def loop()
    # x方向傾きの取得
    axisX = gr_analogRead($PIN_AN4)
    $numbmp1.setInteger(axisX)

    # y方向傾きの取得
    axisY = gr_analogRead($PIN_AN5)
    $numbmp2.setInteger(axisY)

    # 移動量を一旦0にします
    velX = 0
    velY = 0

    # x方向の傾きによってx方向の移動量を決めます
    if axisX > ($centerX + $margin) then
       velX = 1
    end
    if axisX < ($centerX - $margin) then
       velX = -1
    end

    # y方向の傾きによってy方向の移動量を決めます
    if axisY > ($centerY + $margin) then
       velY = 1
    end
    if axisY < ($centerY - $margin) then
       velY = -1
    end

    # 新しいx座標を計算します
    $newX = $posX + velX
```

［途中のページは省略します］

著者　　株式会社アイ・エル・シー
〒732-0824
広島市南区的場町 1 丁目 3 番 6 号 広島的場ビル 9F
http://www.ilc.co.jp

EAPL-Trainer™

Embedded Application development Trainer

mruby

INTERNATIONAL LABORATORY CORPORATION