PICによるカラーバーサライタの制作

小山工業高等専門学校　電子制御工学科

金野研究室

電子制御工学科　5年　桜井　裕亮

まえがき

本書は、筆者が構築した「画像ファイルを表示できる手動カラーバーサライタ」の制作過程をまとめたものである。

1 つの形として一応完成してはいるものの、未だ問題点は多く不完全な部分は多々あり、さらなる改良及び発展は必須である。「改良点」の項目で述べる問題以外にも、改善の余地があると思わしきところは各項目中で述べるようにしている。

本システムを構築する際は、本書の内容は参考にとどめ、システムの根本的なところから何らかの改良を加えた上で構築してほしい。本書はある種の失敗談として有効に活用していただければ幸いである。

2012 年 3 月 5 日　筆者

目次

1. はじめに

バーサライタは一列に並んだ光源を動かした残像によって文字などを空中に表現する装置である。あまり一般的に知られたものではないが、コンサート用の手で振るタイプや自転車の車輪や扇風機に取り付けられたタイプ、また「止まれ」などの文字を表示する誘導棒などが実際に販売されている。これら従来のものは、単色の光源を使用している故に単純な絵や文字しか表示できないものがほとんどである。(図１) しかし、光源としてフルカラー LED を用いればカラー画像なども表示することが可能となり、表現の幅と共に用途の幅も広まるものと考えられる。(図２)

本研究では、「画像ファイルを表示することの出来るバーサライタを PIC により制作することを目的とする。

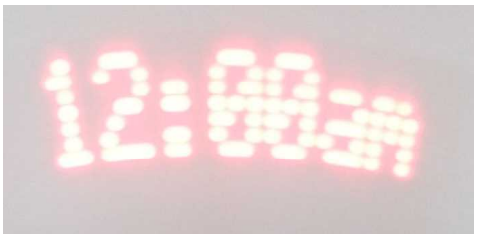


図１　一般的なバーサライタの例

図２　制作したカラーバーサライタの動作
(※上方の水色の線は赤の LED の不備によるもの)

2.　原理

まず、本システムの核となる２つの原理を簡単に説明する。

2.1.　残像効果

人は強い光を見ると、その光が消えた（移動した）後もその光がしばらくその空間に残っているように錯覚する。これが残像効果である。

この原理を利用し、反復運動する１列に並んだ光源を、動きに合わせて点滅させることで文字や絵を表示する装置がバーサライタである。

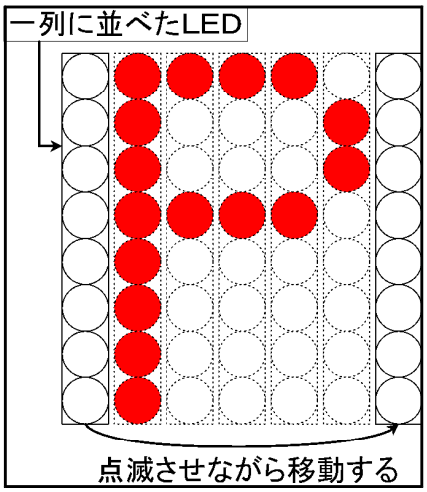


図３　バーサライタの原理

2.2.　光の三原色

人間の色覚は三色型色覚と呼ばれ、光刺激を三種類の錐体で受けとめ三次元の感覚情報として処理し、あらゆる光の色を三つの原色の混合比として捉える。つまりは、三つの原色（赤(R)、緑(G)、青(B)）の光源を用意し、それぞれの色の輝度（明るさ）を制御することができれば、その混合によってあらゆる色を表現することが可能なのである。テレビ等のディスプレイもこの原理を利用したもので、三色の輝度の段階を十分に作ることができればこれらと同等の色数を表現することも可能ということになる。

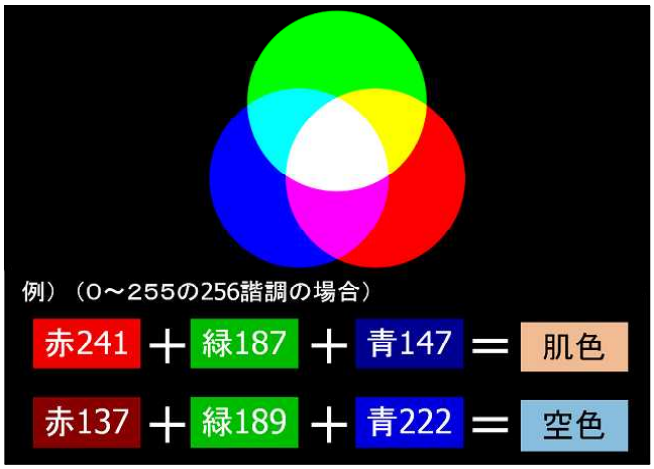


図４　光の三原色

3. 概要

今回、バーサライタは手動で振る物として構築した。図３に制作したバーサライタのシステムの概要図を示す。全体としては、加速度センサーがバーサライタ本体の動きを検知し、その情報を元に PIC がフルカラー LED の点滅を制御するという簡潔なものである。しかし、カラーの表現（LED の輝度制御）や、点灯位置の制御には難儀なところがあり、これらをどのようにして解決するかが本システムを製作する上で重要な点となる。

また、どのようにして画像ファイルを扱うかも重要となる。

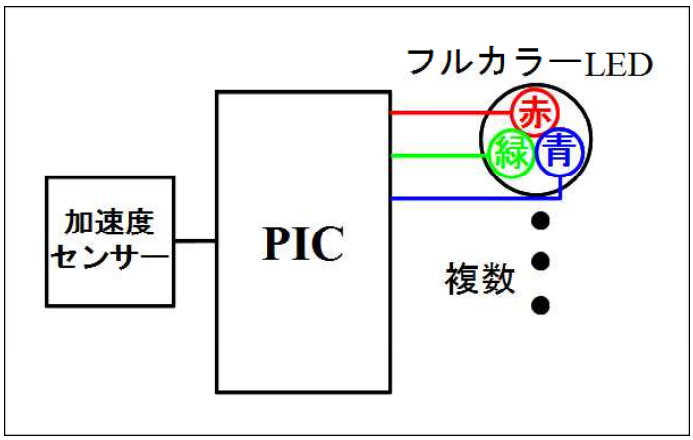


図５　システムの概要図

3.1. カラーの表現（LED の輝度の制御）

今回使用する「フルカラー LED」は、三原色である三色（赤・緑・青）の LED が１つにまとまったものである。これら三色は別々に制御することができるため、それぞれの輝度を制御することで、前述した原理（2.2.）によってカラーを表現できる。

今回製作したバーサライタは、輝度の制御にパルス幅変調（PWM）を用いている。人間は、認識できないほどの高速で点滅する光源を見たとき、周期内での明るさの平均値で常時点灯しているように錯覚する。この特性を利用し、LED を高速で点滅させその周期あたりの点灯時間（デューティ比）を調節することで、視覚的に明るさを変えるのである。

PWM 制御によって具体的にどのようにして明るさの濃淡の段階数（階調）を作っているかは、ソフトウェアの項（5.1.1）にて後述する。

3.2. 表示位置の制御

今回画像の表示位置の制御には加速度センサーを使用した。加速度センサーはその名の通り加速度を計測するセンサーで、これによってバーサライタ本体の動きに伴う加速度の変化を検知し、その情報を元に制御を行う。センサーから得た情報を、具体的にどのように扱って点灯位置を制御しているかは、ソフトウェアの項（5.1.2）にて後述する。

3.3.　画像ファイルの扱い

本研究の目的は画像ファイルを表示することの出来るバーサライタの制作である。画像ファイルをバーサライタで表示できるようにするまでの概要図を図６に示す。

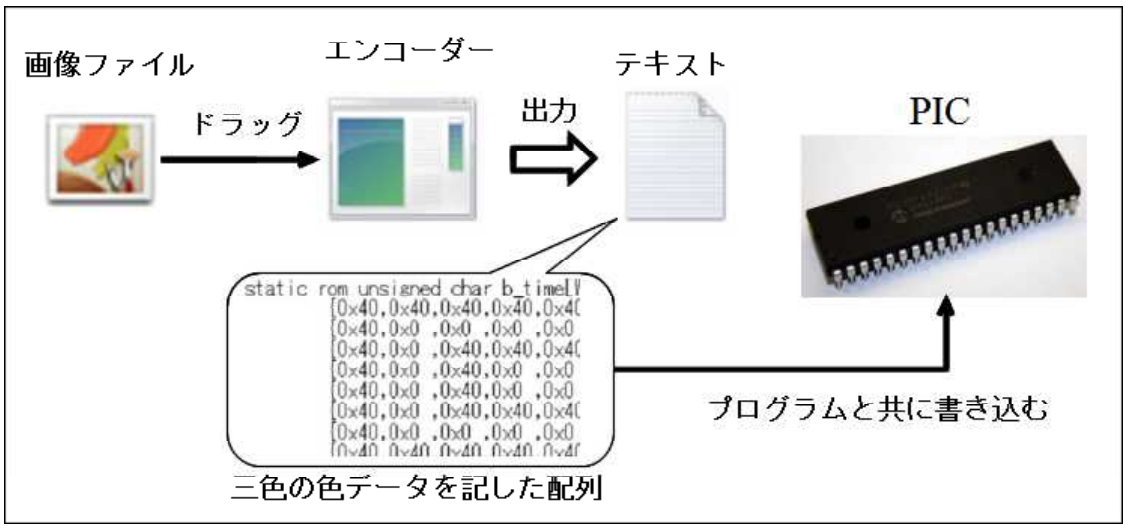


図６　画像ファイルの扱い

画像ファイルを PIC に直接入れることはできず、さらに PIC で画像ファイルのデータを解析し LED の輝度を制御するのでは動作速度に問題がでてしまう。

そこで、外部であらかじめ画像ファイルのデータを PIC で扱いやすいデータへと変換（エンコード）し、そのデータを PIC へと書き込むシステムとした。表示したい画像を一々エンコードして PIC へと書き込まなくてはならないといういささか面倒な手法ではあるが、この方法により一度書き込んだ画像データはバーサライタ本体ですぐに表示することができる。

詳細はソフトウェアの項（5.)、具体的な操作方法は使用方法の項（6.）にて後述する。

4. ハードウェア

4.1. 使用したデバイスなど

以下に、今回使用したデバイス・素子を列記する。

・PIC　（PIC18F452　× 3）
・フルカラー LED　（OSTA71A1D-A　× 32）
・加速度センサー　（KXM52-1050 モジュール）
・電源　（9V 型乾電池）
・3 端子レギュレータ 5V1.5A　（7805A）
・水晶発振器 40MHz　（VX-6231）
・抵抗、コンデンサ等　数本

4.1.1 PIC

PIC を選ぶ上で重要となるのがピン数（I/O ポート数）と最高クロック数である。フルカラー LED を制御するには赤・緑・青（RGB）三色の輝度をそれぞれ別に制御しなくてはならないため、フルカラー LED の数× 3（色）と、多くの信号を扱わなくてはならなく、その分の I/O ポートが必要となる。最高クロック数は PIC の動作速度に関係し、バーサライタの特性と PWM によって輝度制御することから、残像の間隔と階調の多さ、すなわち表示する画像の質に影響を与える。また、加速度センサーのアナログ出力を扱うこととなるので、A/D 変換機能がついているものが望ましい。

今回は、手元にあった PIC の中でもピン数の多かった PIC18F452 を使用した。PIC18F452 はピン数が 40 ピン、最大クロック数は 40MHz である。（図 7）

TABLE 1-1: DEVICE FEATURES

Features	PIC18F242	PIC18F252	PIC18F442	PIC18F452
Operating Frequency	DC - 40 MHz	DC - 40 MHz	DC - 40 MHz	DC - 40 MHz
Program Memory (Bytes)	16K	32K	16K	32K
Program Memory (Instructions)	8192	16384	8192	16384
Data Memory (Bytes)	768	1536	768	1536
Data EEPROM Memory (Bytes)	256	256	256	256
Interrupt Sources	17	17	18	18
I/O Ports	Ports A, B, C	Ports A, B, C	Ports A, B, C, D, E	Ports A, B, C, D, E
Timers	4	4	4	4
Capture/Compare/PWM Modules	2	2	2	2
Serial Communications	MSSP, Addressable USART	MSSP, Addressable USART	MSSP, Addressable USART	MSSP, Addressable USART
Parallel Communications	—	—	PSP	PSP
10-bit Analog-to-Digital Module	5 input channels	5 input channels	8 input channels	8 input channels
RESETS (and Delays)	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST)	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST)	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST)	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST)
Programmable Low Voltage Detect	Yes	Yes	Yes	Yes
Programmable Brown-out Reset	Yes	Yes	Yes	Yes
Instruction Set	75 Instructions	75 Instructions	75 Instructions	75 Instructions
Packages	28-pin DIP 28-pin SOIC	28-pin DIP 28-pin SOIC	40-pin DIP 44-pin PLCC 44-pin TQFP	40-pin DIP 44-pin PLCC 44-pin TQFP

図 7　PIC18F452 の仕様

今回製作したバーサライタはフルカラー LED を 32 個使うものとして構築している。すなわち 32（個）× 3（色）＝"96"の信号を制御する必要があり、PIC18F452 のピン数 40 ピンでは全然足りない。すなわち、そのままだと一つの I/O ポートに一つずつ信号を割り当てていくことは不可能であり、何らかの解決策が必要となる。

そこでこの問題の解決策として、PIC18F452 の数を 3 つに増やし同時動作する方法をとった。それぞれの PIC が三色ある信号のうちの一色を、すなわち 32 の信号ずつを分担し、LED の輝度を制御するのである。（図８）詳細は回路の項にて後述する。（4.2.）

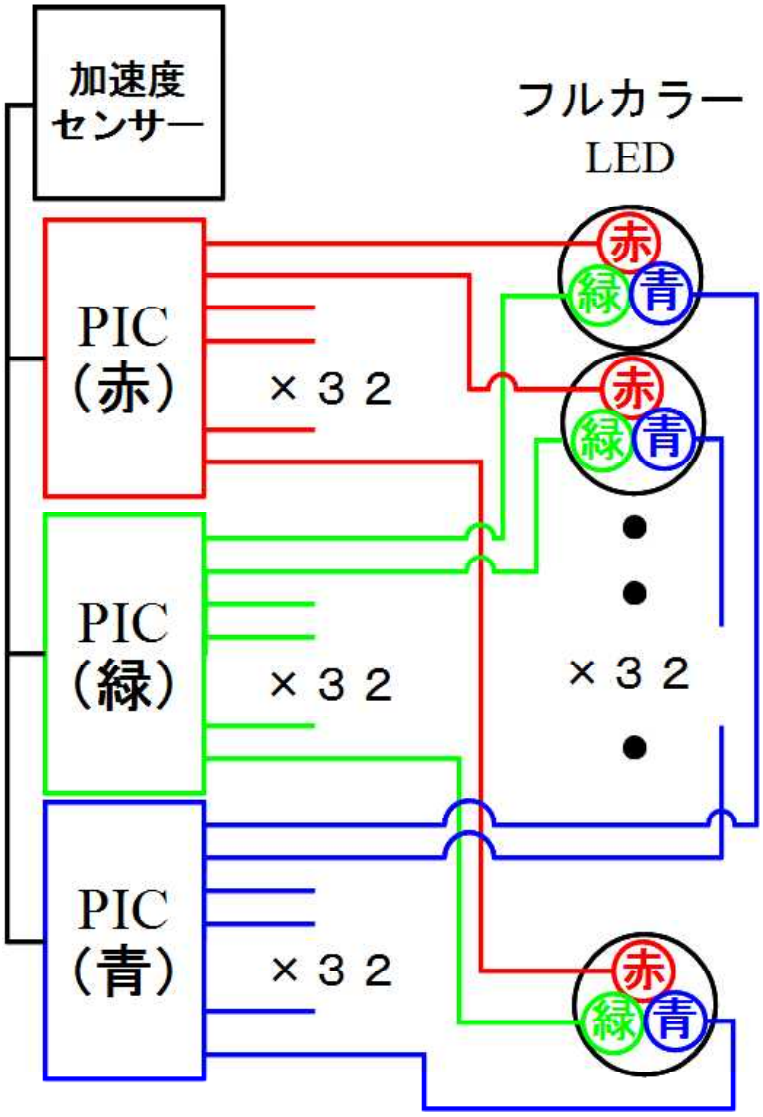


図８　３つの PIC によるフルカラー LED の制御

4.1.2　フルカラーLED

フルカラーLEDは前述したように三原色である三色（赤・緑・青）のLEDが1つにまとまったものである。

フルカラーLEDには、アノード（＋）側がまとまったアノードコモンのものと、カソード（－）側がまとまったカソードコモンのものがあるが、今回はアノードコモンのものを使用することを前提に構築している。理論的にはカソードコモンでも制作出来ないことはないが、その場合はLEDの制御プログラムの他、回路も多少異なってくる。また、カソードコモンの場合はアノード側をPICの出力で制御することとなるので、PIC出力の許容電流の関係からトランジスタによる電流増幅等が必要になる可能性がある。

用途にもよるがLEDの輝度はある程度大きいものが望ましい。バーサライタは原理の項で前述したように残像効果を利用したものなので、輝度が足りないと残像が上手く残らない可能性がある。

今回使用したOSTA71A1D-Aはアノードコモンの角型フルカラーLEDである。明るさ（光度）が赤3000，緑4200，青2180mcd（20mA）となかなか高輝度であり、バーサライタの明るさ、見やすさの面では問題を感じなかった。ただ、その分電流は大きくなるので、後述するように電源（4.1.4）や3端子レギュレータ（4.1.5）の面で多少であるが問題が出た。

なお、今回はフルカラーLEDを使用したが、要は三原色の光源が用意できればいいわけであり、三色のLEDを別々に用意してもよい。ただ、その場合、三原色の密集性が問題となり、離れて見ないとカラーが上手く見えない可能性がある。

図9　OSTA71A1D-A

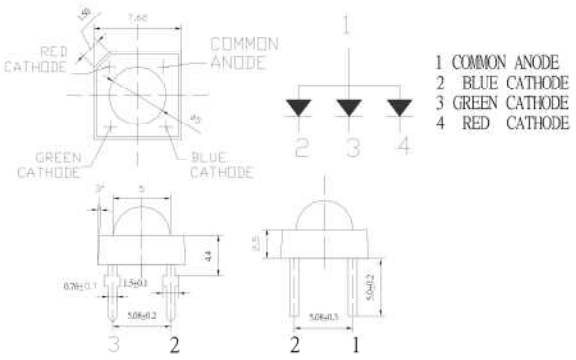
■Features

- High Luminous Output
- Common Anode
- Superior Weather-resistance
- Water Clear Type
- With 5mm Lens

■Applications

- Toys
- Games
- Audio

■Outline Dimension

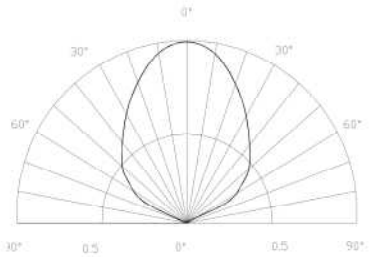


■Absolute Maximum Rating (Ta=25℃)

Item	Symbol	Value	Unit
DC Forward Current (RED)	I_F	50	mA
DC Forward Current (GREEN/BLUE)	I_F	30	mA
Pulse Forward Current*(RED)	I_{FP}	100	mA
Pulse Forward Current* (GREEN/BLUE)	I_{FP}	50	mA
Reverse Voltage	V_R	5	V
Power Dissipation	P_D	150	mW
Operating Temperature	Topr	-30 ~ +85	℃
Storage Temperature	Tstg	-30 ~ +100	℃
Lead Soldering Temperature	Tsol	260℃/ 5sec	-

*Pulse width Max.10ms Duty ratio max 1/10

■Directivity



■Electrical -Optical Characteristics (Ta=25℃)

Item	Symbol	Condition	Min.	Typ.	Max.	Unit
DC Forward Voltage	V_F (R)	I_F=20mA	1.8	2.0	2.4	V
	V_F (B/G)	I_F=20mA	2.8	3.2	3.6	V
DC Reverse Current	I_R	V_R=5V	-	-	10	μA
Domi. Wavelength	**λ_D(Red)**	I_F=20mA	620	625	630	nm
	λ_D(Green)	I_F=20mA	520	525	530	nm
	λ_D(Blue)	I_F=20mA	465	470	475	nm
Luminous Intensity	**Iv(Red)**	I_F=20mA	2180	3000	-	mcd
	Iv(Green)	I_F=20mA	3500	4200	-	mcd
	Iv(Blue)	I_F=20mA	1560	2180	-	mcd
50% Power Angle	$2\theta_{1/2}$	I_F=20mA	-	90	-	deg

図１０　OSTA71A1D-A の仕様

4.1.3　加速度センサー

前述したように、バーサライタ本体の動きに伴う加速度の変化を検知し、その情報を元にバーサライタの動きと点灯位置を合わせるためのものである。

今回使用した加速度センサー（KXM52-1050 モジュール）は、手元にあったのがこれしかなかったことから三軸の加速度センサーとなっているが、バーサライタの動きは、手動で扇状にあるいは左右へ水平に振った場合を想定しているため、一軸の加速度を測定できれば十分である。

今回はバーサライタを手動で振るものとして制作したため、本体の動きと LED の点滅を合わせるために加速度センサーを使用したが、自動でバーサライタを動かす何らかの装置を使う場合はそちらの動作を制御すれば良いので、加速度センサーは必要なくなるものと思われる。

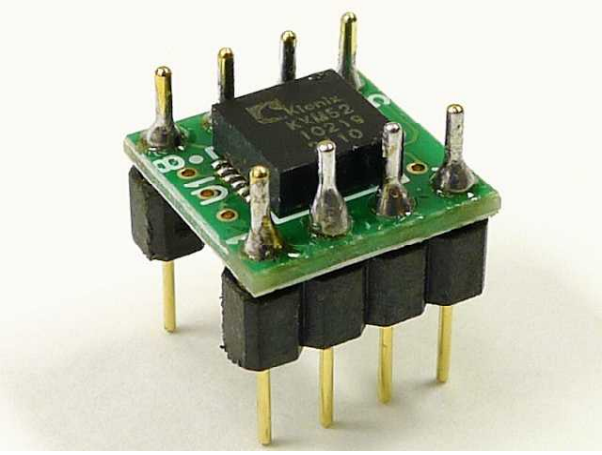
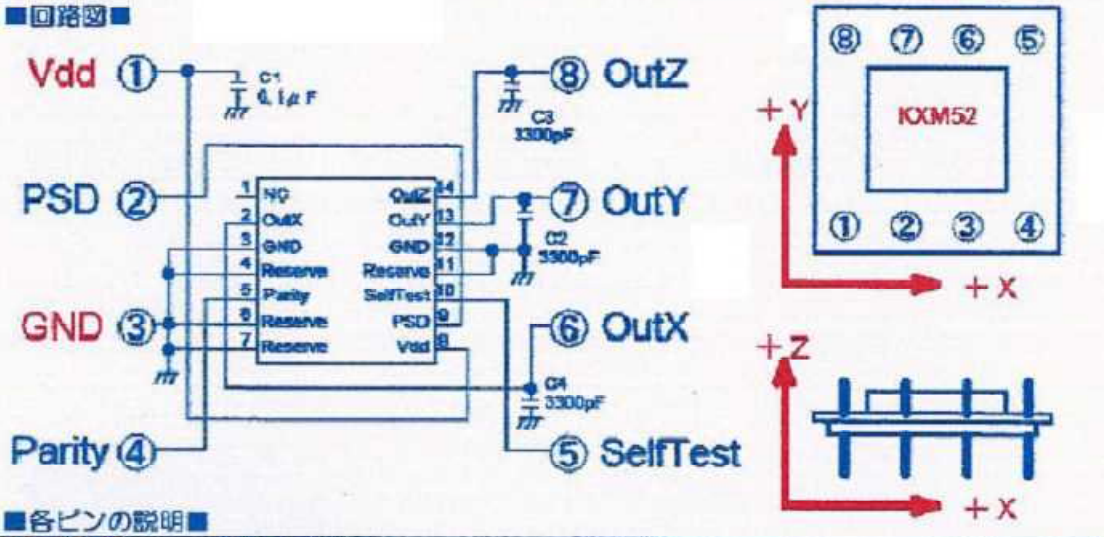


図１１　ＫＸＭ５２－１０５０モジュール

■ 測定レンジ　±2g
■ 感度　660mV/g（電源3．3V時）
■ 0g出力　1．65V　（電源3．3V時）
■ 電源電圧　2．7V～5．5V（標準3．3V）

■回路図■



■各ピンの説明■

番号	名称	接続、備考等
1	Vdd	電源入力　2.7V～5.5V
2	PSD	パワーシャットダウン　無接続または、GNDに接続すると シャットダウンになる　通常はVdd（1番ﾋﾟﾝ）に接続する。
3	GND	GND
4	Parity	パリティ　内部EEPROMのﾊﾟﾘﾃｨﾁｪｯｸ用 通常は、どこにも接続しない（無接続）
5	SelfTest	セルフテスト　Vddに接続すると、出力が1G増える 通常は通常はGND（3番ピン）に接続する。
6	OutX	X軸出力
7	OutY	Y軸出力
8	OutZ	Z軸出力

図１２　ＫＸＭ５２－１０５０モジュールの仕様

4.1.4　電源

今回電源は 9V 型乾電池の出力を、3 端子レギュレータで 5V　に安定化して使用している。

高輝度の LED を多数使用している故に作動時間あたりの消費電力が大きいため、電池の消耗は激しくすぐに切れてしまう。

4.1.5　3 端子レギュレータ

PIC、フルカラー LED、加速度センサー、これら全ては 5V で動作可能なので、前述したように、9V 型乾電池の出力を、3 端子レギュレータで 5V に安定化して使用している。

ここで問題となるのが電流であり、フルカラー LED の分量が大きく、32 個全ての 3 色を同時点灯すると 1 A を超える。そのため、最大電流が 1.5A と大きめのものとした。

なお、電圧降下による 3 端子レギュレータの発熱も大きく、放熱器は必須と思われる。

4.1.6　水晶発振器

PIC の外部発振として、PIC18F452 の最大クロック数である 40MHz の水晶発振器を使用している。

当初、セラミック発振子を使用していたが、LED の数を増やしていったところ動作が不安定となった。はっきりした原因は分からないが、セラミック発振子は電圧の変化に弱く最悪止まってしまうことがあるようで、LED に流れる電流の急激な変化に伴う電圧の変化が原因だったのでないかと考えている。

以降、水晶発振器に換えてからは、問題なく動作しているため、発振素子は水晶のものを推奨する。

4.2. 回路

図１３に本システムの回路図を示す。LED 部分の詳細は後述する。(4.2.2)

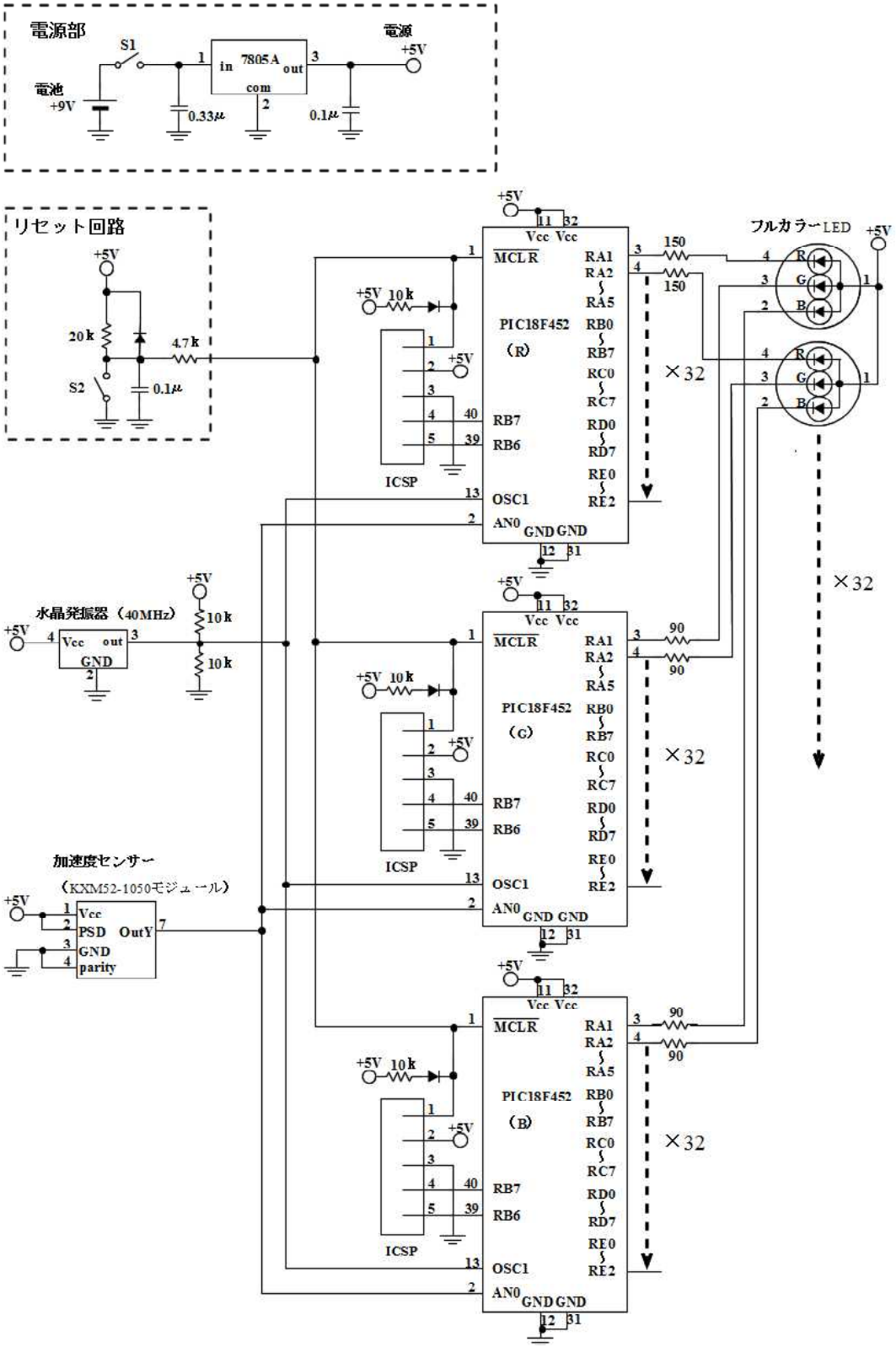


図１３　回路図

4.2.1　３つの PIC による回路

今回、PIC18F452 を赤用、緑用、青用として３つ使用しているが、それぞれの回路は対称的で似通っている。電源はもちろんとして、リセット回路、水晶発振器、加速度センサーの信号は全て同一のものを接続している。なお、PIC にプログラムを書き込むための ICSP だけは３つ別々に用意している。

注意すべきは LED で、信号の接続先が当然 RGB で異なり、さらに間の抵抗値も色によって異なる。抵抗値の求め方は次の項で説明する。

4.2.2　フルカラー LED の回路

フルカラー LED のアノード側は 32 個全て電源に固定している。

カソード側は色ごとに３つの PIC に振り分けている。LED と PIC のピン番号の対応関係は次の項で説明する。

LED のカソード側と PIC の端子との間にはそれぞれ電流制限用の抵抗を挟んでいる。抵抗値は LED の色により異なり、以下の式によって求められる。

$$\text{電流制限抵抗（Ω）}=\frac{\text{電源電圧（V）}-\text{LEDに加える電圧（V）}}{\text{LEDに流したい電流（A）}}$$

LED に流したい電流（A）は RGB ３色とも統一し 20mA とした。（※改良の余地あり→次ページ詳細）

LED に加える電圧（V）は、フルカラー LED、OSTA71A1D-A の仕様（図１０↵）より、赤が 3.2V、緑と青が 2.0V とする。

そして、電源電圧（V）は 5V であるため、20mA を流すための各色の電流制限抵抗の理論値は以下のようになる。

$$\text{赤の電流制限抵抗（Ω）}=\frac{5\ (\text{V})-3.2\ (\text{V})}{20\times 10^{-3}\ (\text{A})}=90\ (\Omega)$$

$$\text{緑・青の電流制限抵抗（Ω）}=\frac{5\ (\text{V})-2.0\ (\text{V})}{20\times 10^{-3}\ (\text{A})}=150\ (\Omega)$$

90 Ω及び 150 Ωの抵抗は丁度存在したため、理論値通りの抵抗を使用することとした。90 Ωの抵抗は赤用で３２個、150 Ωの抵抗は緑と青に必要なので６４個必要となる。

【改良点】

制作後に考察した内容であるため実際に試してはいないが、より良い構築のための参考として、調べた内容をまとめる。

今回、赤、緑、青の３色間の明るさの関係を考慮せず、LED に流したい電流は一律 20mA として抵抗値を設定したが、画像ファイルを本当に正しい色で表示するには３色の輝度の比率を考慮する必要があった。

今回用いる画像形式（ビットマップ）を含め、一般的に用いられている「RGB カラーモデル」は赤、緑、青の色成分（明度）をそれぞれ 0~255 の 8bit の数字で表しているが、これは LED の３色それぞれの明るさ（輝度）とは等しくないのである。三色それぞれの明度を等しく 255 とした場合の色を図１４に示す。

図１４　色による輝度の違い

緑が一番明るく、青が暗いことが目視でも見て取れる。この同じ明度における、色ごとの輝度の違いはおよそ（R：G：B）＝（３：６：１）の関係となるとされているようである。

参考として、コンポーネント映像信号に用いられる、RGB から輝度信号への変換式は以下のようになっている。

輝度: Y ＝ 0.299*R + 0.587*G + 0.114*B　（R,G,B：赤、緑、青の明度）

今回構築した、全色の LED に流す電流を 20mA とする設定における、LED の輝度は OSTA71A1D-A の仕様（図１０↵）より R：3000mcd　G：4200mcd　B：2180mcd となる。（cd（カンデラ）は光度の単位だが、輝度は単位面積あたりの光度のことなので比率は同じとする。）

先ほど述べた（R：G：B）＝（３：６：１）の比率と比べると、赤と特に青の輝度が大きいことがわかる。このことは、動作結果からも見て取れる。（図２などの写真を見ると、全体的に紫がかっている。）（動作結果（7.）も参照）

今回使用した LED では一律 20mA でも割合気にならないほどの差ではあったが、余裕があればこのような輝度の比率も視野に入れた上で、色ごとの LED に流す電流を（電流制限抵抗値を）設定した方が良い。

4.2.3　LED と PIC18F452 のピン番号との対応関係

フルカラー LED の数が３２個と多く扱いにくいため、８個ごとの ABCD ４つの組とし、１番目（バーサライタ本体の先端側）の LED から順に A1、A2、・・・D8 というように名前を付けた。32 個の LED と対応する PIC のピン番号は３つの PIC（３色の LED）全てで同じであり、配線しやすいようピンの並びに合わせ対応する LED を決めている。

32 個の LED と対応するピン番号を表１に示す。パターン図（図１６）の中にも書いてあるので参考にしてほしい。

LED の番号	LED の名前	ピン番号	ピンの名称
1	A1	40	RB7
2	A2	39	RB6
3	A3	38	RB5
4	A4	37	RB4
5	A5	36	RB3
6	A6	35	RB2
7	A7	34	RB1
8	A8	33	RB0
9	B1	30	RD7
10	B2	29	RD6
11	B3	28	RD5
12	B4	27	RD4
13	B5	26	RC7
14	B6	25	RC6
15	B7	24	RC5
16	B8	23	RC4
17	C1	3	RA1
18	C2	4	RA2
19	C3	5	RA3
20	C4	6	RA4
21	C5	7	RA5
22	C6	8	RE0
23	C7	9	RE1
24	C8	10	RE2
25	D1	15	RC0
26	D2	16	RC1
27	D3	17	RC2
28	D4	18	RC3
29	D5	19	RD0
30	D6	20	RD1
31	D7	22	RD3
32	D8	21	RD2

表１　LED と PIC のピン番号の対応

4.2.4　リセット回路

完成後、PIC が誤動作することが多かったために急遽後から取り付けた回路であるため、未だ不完全な回路である。

点滅させる LED の数を増やしていったところ、電源を入れてすぐに誤動作することがあまりに多かった。色々調べてみたところ、電源が立ち上がってからしばらくして PIC のリセットが入るようにしてみたところ誤動作はなくなった。そこでこのリセット回路を作ったのである。

しかし、遅延時間が足りなかったのか誤動作を起こすことが間々ある。スイッチ（回路図１３の S2）を切り替えることによって、手動でリセットを入れることは出来るので、これを現段階では使用しているが、これでは遅延回路が意味をなさない。

そもそも誤動作の原因ははっきりしておらず、電池の残量不足（電圧不足）や接触不良が原因だった可能性もあり、さらなる検証が必要である。

4.2.5　加速度センサー

今回用いた加速度センサーは三軸であるため、X、Y、Z の３つの出力があるが、今回は取り付けた向きの関係上 Y 軸の出力を用いている。

【改良点】

３つの PIC のアナログポートへ同じ出力信号をつなげているのだが、A/D 変換を各 PIC で行うこととなるため、各 PIC での値に誤差が生じてしまうという問題点を抱えている。３つの PIC で値が微妙に異なるために、３色間の点灯タイミングがずれてしまうことがあるのである。長く振っているとこれがたびたび起こるため、なんらかの改良が必要となる。

4.3.　組み立て

大した考察もなしに急いで作ったことと制作者の技量不足のために、とても雑な作りとなっており、不完全である。特に軸となる棒への固定方法は大きな問題を抱えているため、何らかの別の方法を考慮する必要がある。

本研究のメインはあくまでソフトウェアと回路と捉え、これから述べることは参考程度にとどめてもらいたい。

LED、スイッチ、リセット回路を除く回路は全てプリント基板で作成した。LED 部分は一列に並べるので長くなり、回路も単調なのでフレキシブル基板を用いた。LED と PIC の信号接続には、直径 0.2mm のポリウレタン銅線を用いている。スイッチとリセット回路は後付したため、フレキシブル基板による雑な作りとなっている。

振るための棒との固定は、軸棒の上にレール状の固定枠を作り、そこに基板などの本体を引き出しのように差し込むようにしている。

リセット回路は後付けしたためにフレキシブル基板となっているが、なるべくプリント基板上に作るべきである。

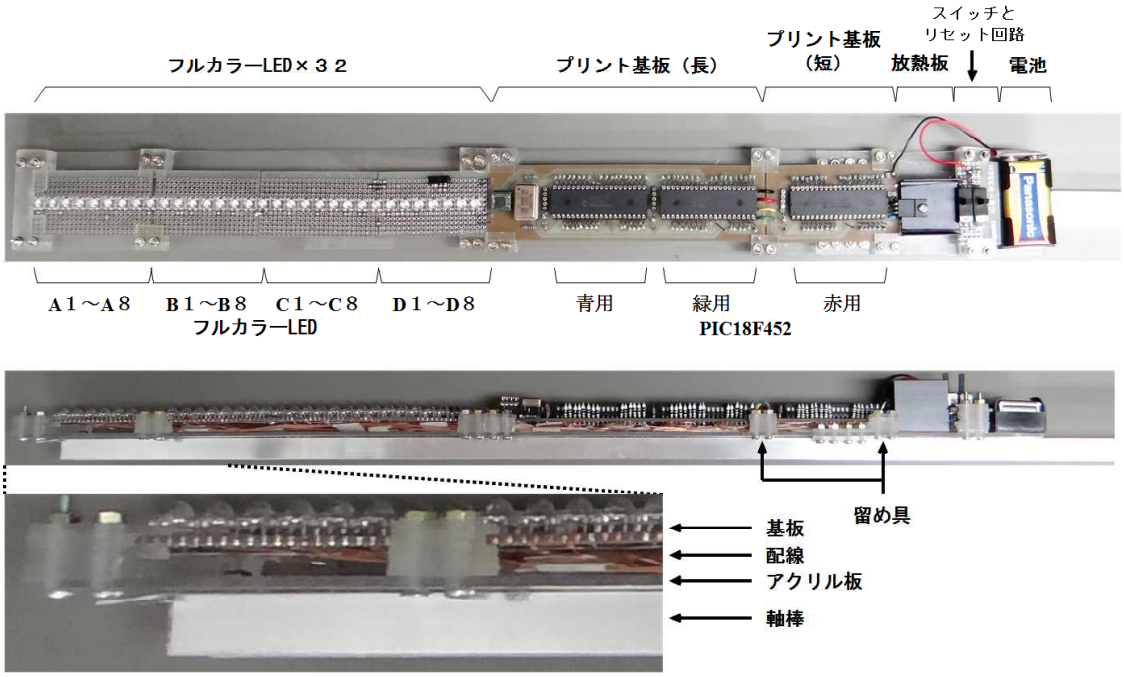


図１５　制作したバーサライタの部品配置図

4.3.1　基板

LED、スイッチ、リセット回路を除く回路は全てプリント基板で作成した。図１６にパターン図を示す。

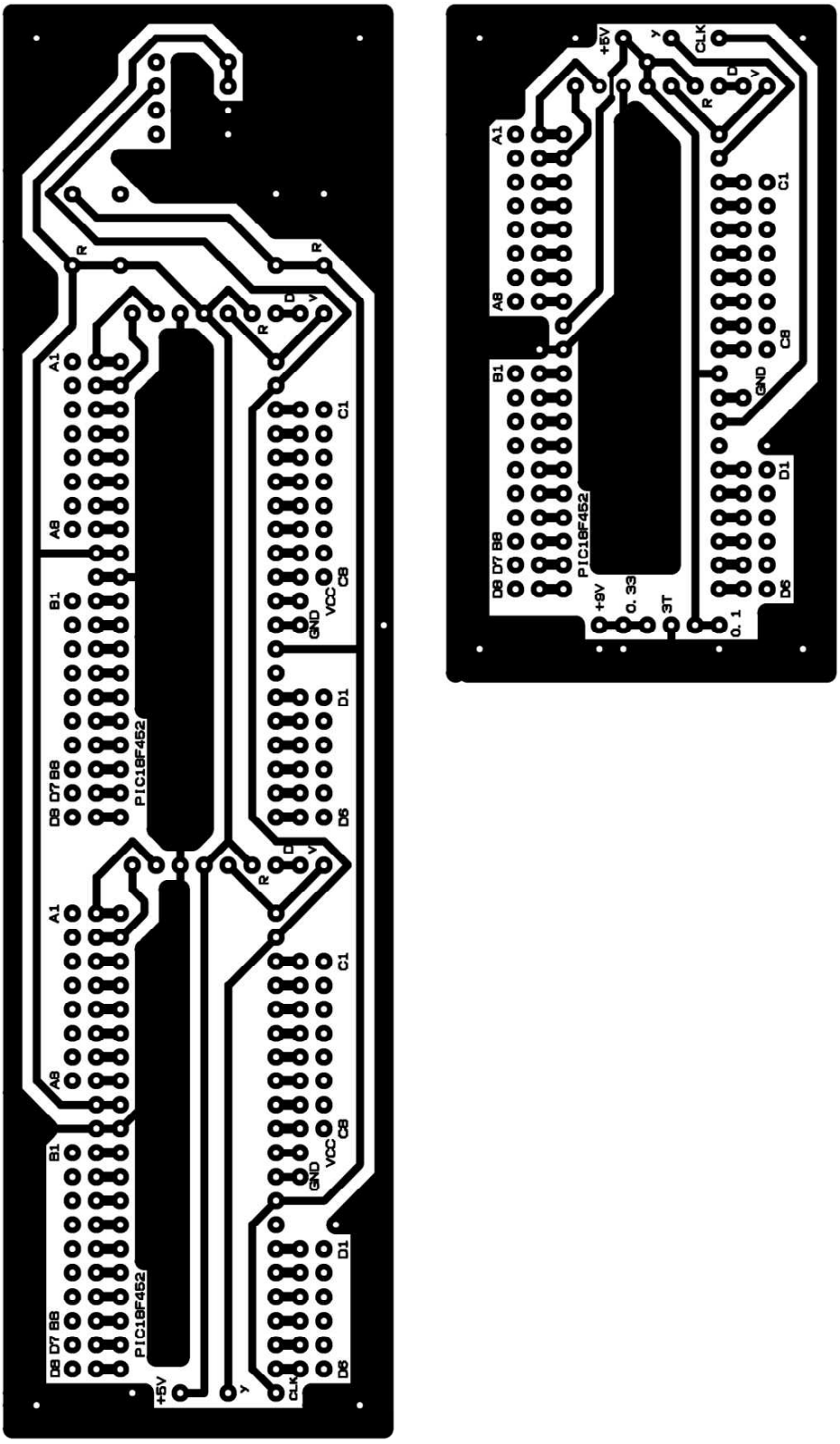


図１６　パターン図

細長くなるように設計した結果、150 × 100 のプリント基板では長さが足りなくなったため２分割した。図１６のパターン図において左下と右上がつながり、電源、GND、加速度センサーの出力、クロックの４つの信号を銅線などで接続する必要がある。（図１７）

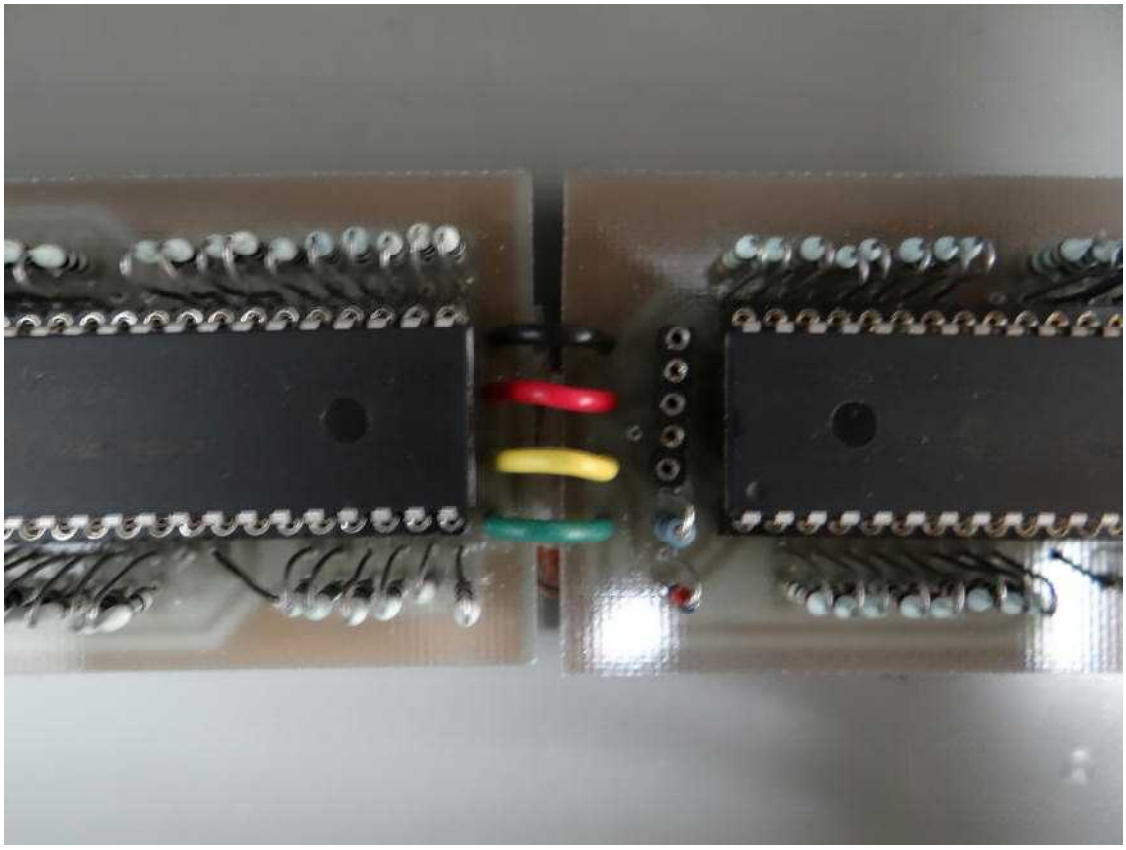


図１７　２つのプリント基板間の接続

黒：GND

赤：電源

黄色：加速度センサーの出力

緑：クロック

4.3.2　外枠

LED や基板などの回路本体を、振るための軸棒へと固定するために、軸棒の上に外枠（ケース）を取り付けている。外枠は引き出しのレールのようになっており、回路本体を引き出しのように差し込むことで固定できるようになっている。しかし、重大な欠点がいくつかあり、良い構築とはとても言えないことを先に述べておく。

振るための軸棒にはアルミのコの字レールを使用している。この軸棒に回路本体よりも大きめのアクリル板を固定し、そこにアクリル板と接着剤で作ったレール状（コの字）の留め具を、ボルトで付けている。この留め具にある程度の高さを設けることで、LED と PIC 間の配線が基板とアクリル板の間に入るようにしている。（図１５）

コの字レールとなっている留め具の凹の部分に丁度基板を差し込めるようになっており、全部差し込んだ後抜けないようにするために電池ボックスを最後部にふたをするように取り付けている。

図１８　回路本体（上）と外枠（下）

最大の欠点は重さ・大きさである。LED を一列に並べるためだけに、フレキシブル基板に固定しさらにアクリル板で軸棒に固定している。結果無駄に大きくなり、先端部分でもあるため振るときにとても重みがかかる。

さらに、そもそも回路を取り外ししやすいように抜き差し型にしたにも関わらず、とても抜き差ししにくい。基板同士のつながりがとても華奢なことや、レールの雑さが原因と思われるが、これでは本末転倒である。

他にも、作るのが割と大変なことなどもあり、あまりお勧めできない作りである。軸棒と回路本体の固定には何らか他の方法を考えていく必要がある。

4.3.3　LED のぼかし

LED に白色のビニール袋をかぶせることで、光をぼかしてみた。これによってドットがぼかされ、ふんわりとした自然な感じになるのがわかる。

今回は仮止めによって試しに動作してみただけであるが、これによって比較的近くから見ても画像が見やすくなる効果などもあるため、さらに調査をすすめ改良を加えていく価値があるように思う。

図１９　ビニール袋の仮止め

図２０　ぼかし無し

図２１　ぼかしあり

5. ソフトウェア

プログラムは全て C 言語によって記述している。PIC に入れるプログラムとは別に、画像ファイルを C 言語の配列とするプログラム（エンコーダ）があり、これは PIC に入れずパソコンで使用する。

両プログラムソースは本書の最後に付録として記述した。

5.1. PIC 用プログラム

PIC は３つ使用しているが、全て同じプログラムを入れており、異なるのは 3 色の色情報を示した配列のみである。

プログラムの主たる部分について解説する。

5.1.1 LED の輝度の階調

デューティ比と輝度の関係は、LED の特性や LED に流す電流、さらには見る人によって異なってくるようであり、具体的な尺度はないようである。見る人による違い、いわゆる個人差はそれほど大きくないものと思われるため気にしなくてもよいが、LED の種類による違いは各自で考慮する必要が出てくる。

明るさの濃淡の段階数（階調）が多いほど表現できる色の数も多くなるので、輝度の段階は細かく設定できるに越したことはない。人間の目が認識できる限界は 256 階調（8 ビット）とされているためこれが理想であり、現代のディスプレイでは当たり前となっている約 1677 万色（256 階調（R）× 256 階調（G）× 256 階調（B））が表現できることとなる。しかし、PWM 制御でそこまでの階調を作るのは難しいところがある。

PWM による制御ということで、while 文により周期を作る。特定回数のループが 1 周期に相当するようにし、サイクル数でそのドットでの点灯時間、すなわちドットの横幅を調整できる。この周期が大きいほど輝度の段階は細かく設定できるようになるが、あまり大きすぎると点滅速度が遅くなるのでちらつきの原因となる。つまり、PWM 制御ではこの部分で階調の数に限界ができるのである。

デューティ比と階調の関係は参考となる資料をほとんど見つけられず、明るさを測定するような装置もなかったことから、試行錯誤で目測により作ることとした。

試していて分かったことが、デューティ比（点灯時間側の比率）が大きくなるにつれ輝度の変化は小さくさるようで、デューティ比 50%以上では明るさの違いがあまり感じられなくなった。逆にデューティ比が小さくなるほどデューティ比の違いによる輝度の変化は大きく感じられ、デューティ比 1%とデューティ比 2%での明るさの違いも十分に感じ取れた。

このことより、輝度の段階が１つ下がるとデューティ比は半分となる単純な対数関係として、輝度の段階を構築した。周期の制限により段階数は８となっている。

輝度レベル	0	1	2	3	4	5	6	7
デューティ比	0	1/64	1/32	1/16	1/8	1/4	1/2	1

表２　輝度レベルとデューティ比の関係

5.1.2　表示位置の制御

今回点灯位置の制御には加速度センサーを使用した。加速度センサーはその名の通り加速度を計測するセンサーで、これによってバーサライタ本体の動きに伴う加速度の変化を検知し、その情報を元に制御を行う。

加速度センサーの出力はアナログであるため、PIC で扱えるよう A/D 変換する必要がある。A/D 変換には PIC に備わった機能を使用している。PIC18F452 はピン番号 2 ～ 5、7 ～ 10 の 8 ポートがアナログポートとして使用可能で、そのうちの 1 つをアナログポートとして加速度センサーの出力を入力し、残りのポートはデジタルポートとして LED の制御へと回している。

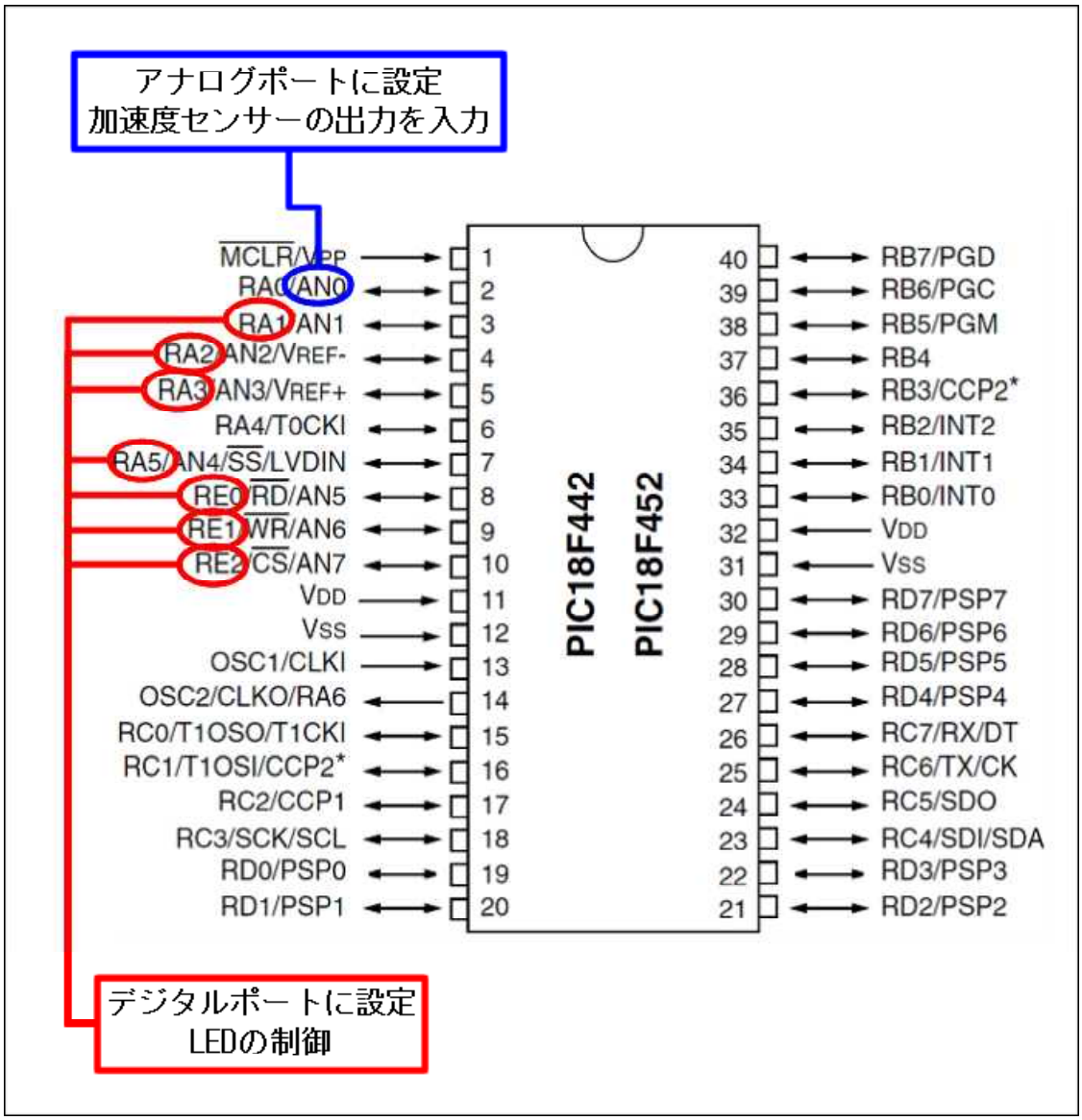


図２２　アナログポートの設定

バーサライタの動きは、手動で扇状にあるいは左右へ水平に振った場合を想定しているため、その振る方向一軸の加速度を計測し得た情報を元に制御を行うこととなる。加速度センサーは重力による加速度も計測するため、扇状に振るとその傾きにより重力の影響を受けることとなるが、微少であるため考えないこととする。

振ったときの画像の表示位置を毎回合わせるには、加速度の変化の特性より今の位置を示す毎回安定した箇所を指定し、その地点を軸として遅延関数などにより表示位置を丁度良い場所へと合わせる必要がある。

加速度センサーは単振動のような運動をすることとなるため、その進行方向への速度は折り返し地点で 0 となり中間地点で最速となる sin 波のようになると仮定できる。加速度は速度の時間についての微分であるため cos 波、すなわち折り返し地点で加速度は最大となると考えられるが、実際には振るときの力のかけ方によって歪むこととなるため、正確にはどのように加速度が変化しているのか予測がつかない。実際に振ったときの加速度の値の変化を測定できれば良いのだが、加速度センサーの出力を数値として出力する手段が思いつかなかったため、試行錯誤でいろいろ試すこととした。

十分な検証するには至らなかったが、結論として、数値を指定し加速度がその数値に達した地点を軸とする方法が最も安定した。前述したように左右に移動するバーサライタの折り返し地点では、正方向あるいは負方向への加速度が最大となると考えられる。正及び負の加速度の数値をあらかじめ設定し、その数値に達した地点を軸として遅延により表示位置を合わせるのである。正か負のどっちの数値に達したかによって、これからの進行方向が分かり、表示する順番（画像を左から右へと表示するか右から左へと表示するか）を合わせることができる。

問題としては、数値で指定しているため振る力が弱ければその数値に達せず光らないことなりり、逆に強く振りすぎてはその数値に達するの地点が折り返し地点とずれてしまい表示位置が安定しない。すなわち振り方が結構シビアであり、慣れないと上手く表示されない。

表示位置の制御方法は先に述べたように十分な検証が出来ておらず､未だ不完全である。簡単に安定して表示されるようにするためには、もっと加速度の変化パターンを検証し、制御方法を改良していく必要がある。

5.2. エンコーダ

本システムは画像ファイルを表示できるバーサライタということで、外部であらかじめ画像ファイルのデータを PIC で扱いやすいデータへと変換（エンコード）し、そのデータを PIC へと書き込むシステムとなっている。

この画像ファイルを C 言語の配列とするプログラム（エンコーダ）は PIC に入れるプログラムとは別であり、これは PIC に入れずパソコンで使用する。

このエンコーダも C 言語によって作っており、付録として本書最後に載せているプログラムをコンパイルすることで生成される。コンパイルによって生成された exe ファイルに対応する画像ファイルをドラッグすることで、画像データを配列としたものが txt ファイルで生成される。使い方の詳細は次の使用方法の項（6.）で述べる。

生成された txt に記される配列は３つあり、それぞれ赤、緑、青の三色のドットごとの輝度情報が示されており、これをそれぞれ対応する色用の PIC に書き込むこととなる。

変換後の配列データは、ただ画像ファイルの色情報を３色に分けただけではなく、PIC の中でそのまま扱える数値、順番となっている。これによって PIC 内での無駄な処理を軽減している。

具体的には、画像ファイルに含まれているドットごとの明るさの情報（明度：0~255 の 256 段階）を、今回扱える限界である８段階へと減らし、それを今回 PIC 内で直接扱う「」周期あたりの消灯時間」へと直している。すなわち、配列に書かれている数値（0x00~0x40）は８段階の周期あたりの消灯時間を 16 進数で示したものである。

配列の数値の順番も PIC 用プログラムで扱いやすい順番としてあるので、元の画像と順番が異なっている。（図２３）

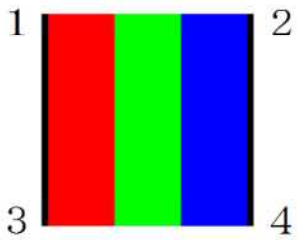


図２３　配列データの順番

6. 使用方法

以下に、画像ファイルをエンコードして PIC へと書き込み、実際にバーサライタ本体で表示するまでの手順を説明する。

6.1. 画像ファイルの準備

まず、バーサライタで表示したい画像ファイルを準備する。対応する画像ファイルの条件は以下の２つである。

・24bit ビットマップファイル（bmp ファイルで一般的な形式）
・幅 64 ピクセル以下、高さ 32 ピクセル

拡張子が.bmp であり、画像ファイルのプロパティを見て、概要タブ（windows 7 では詳細タブ）の「ビットの深さ」が"24"、「幅」が"64 ピクセル"以下、「高さ」が"32 ピクセル"となっていれば大丈夫である。

画像がこの条件を満たしていない場合は、画像編集ソフトを使うことで対応させることが可能である。

フリーソフト pixia（ver.5.40h）を使う場合を例に手順を説明する。

・メニューから「ファイル」→「開く」を選択し、編集したい画像を開く。
　↓
・「画像」→「大きさを変えて複製」を選択し、"変更後の高さ"が 32 ピクセルに、"変更後の幅"が 64 ピクセル以下になるように設定する。
　↓
・「ファイル」→「名前を付けて保存」を選択し、"ファイルの種類"を windows ビットマップ[*.bmp]にして、ファイル名を付けて保存する。

以上の手順で、条件に合っていない画像ファイルを条件に合わせることができる。

6.2.　エンコード

画像ファイルを PIC に書き込めるようにするために､C 言語の配列へとエンコードする。

エンコードには、独自に C 言語で作成したエンコーダを使用するわけだが、本書の最後に付録として C 言語のプログラムソースを公開しているので、これを各自でコンパイルしてほしい。

コンパイルによって生成された実行ファイル(.exe)に、画像ファイル(.bmp)をドラッグする。

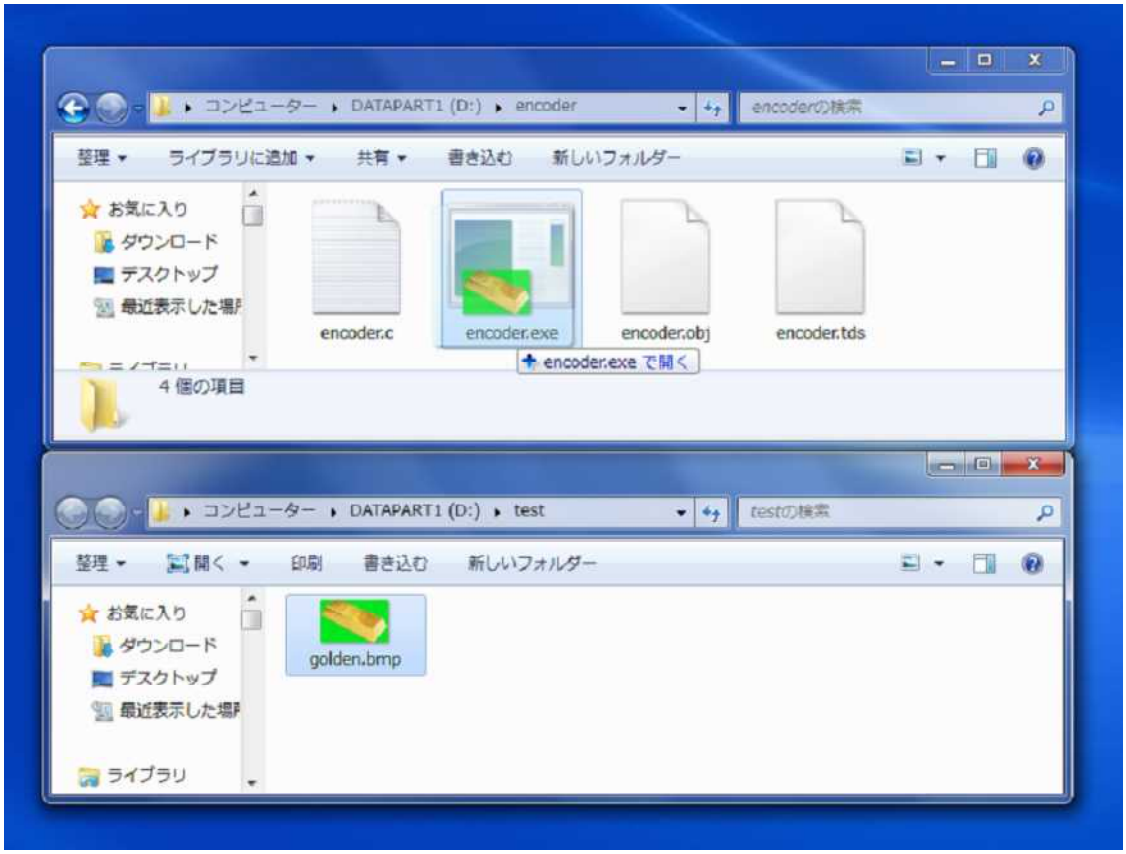


図２４　エンコードの手順１

すると、画像ファイルと同じ名前のテキストファイル(.txt)が画像ファイルと同じフォルダに生成される。

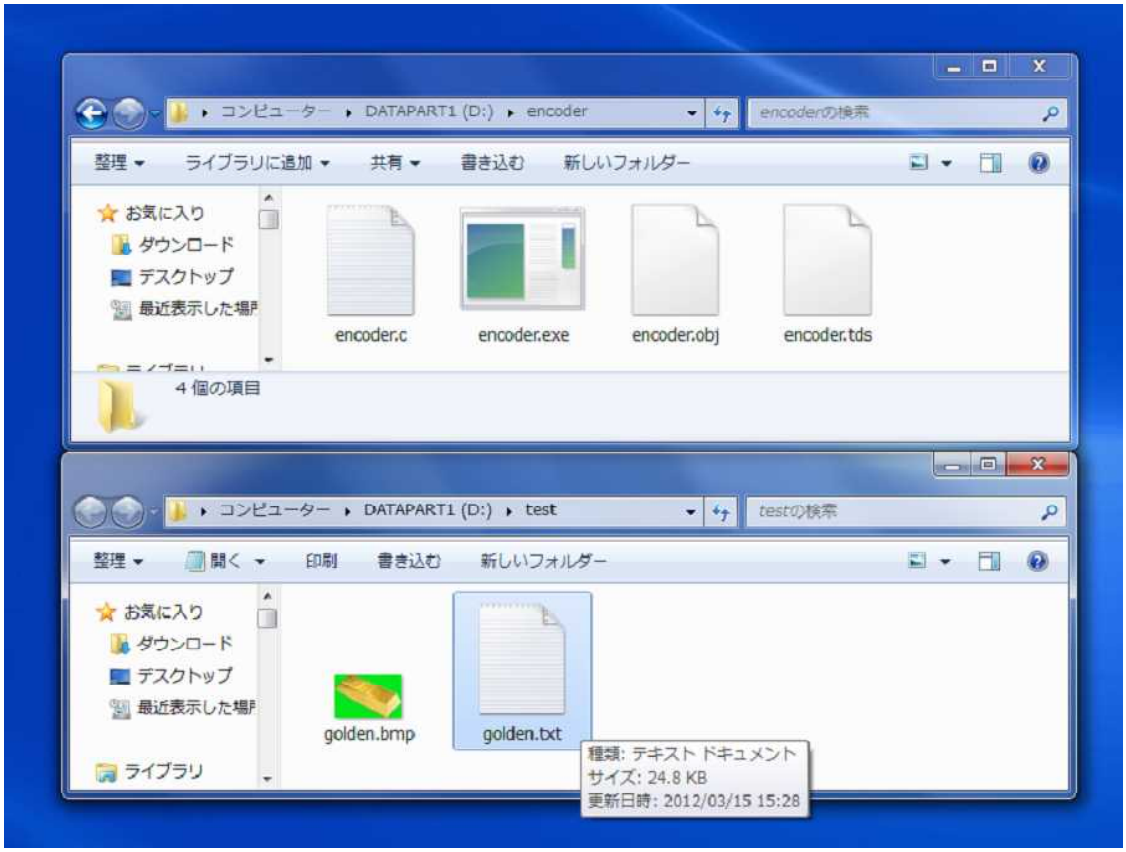


図２５　エンコードの手順２

これでエンコード完了である。

※エンコードする画像ファイルのパスに"bmp"の文字列が入っていないこと。
　またパスの中に日本語が入っていると失敗することがある。

6.3.　PIC への書き込み

MPLAB 等で PIC 用プログラム（付録のプログラムソース（PIC））を PIC へと書き込めるように準備する。

エンコードによって生成されたテキストファイルを開く。エンコードによって生成されたテキストファイルには、赤、緑、青三色それぞれの、ドットごとの輝度情報を書き表した C 言語の配列が記されている。

これら赤、緑、青３つある配列のうち、赤用の部分だけコピーし PIC 用プログラムの下図の場所に貼り付ける。

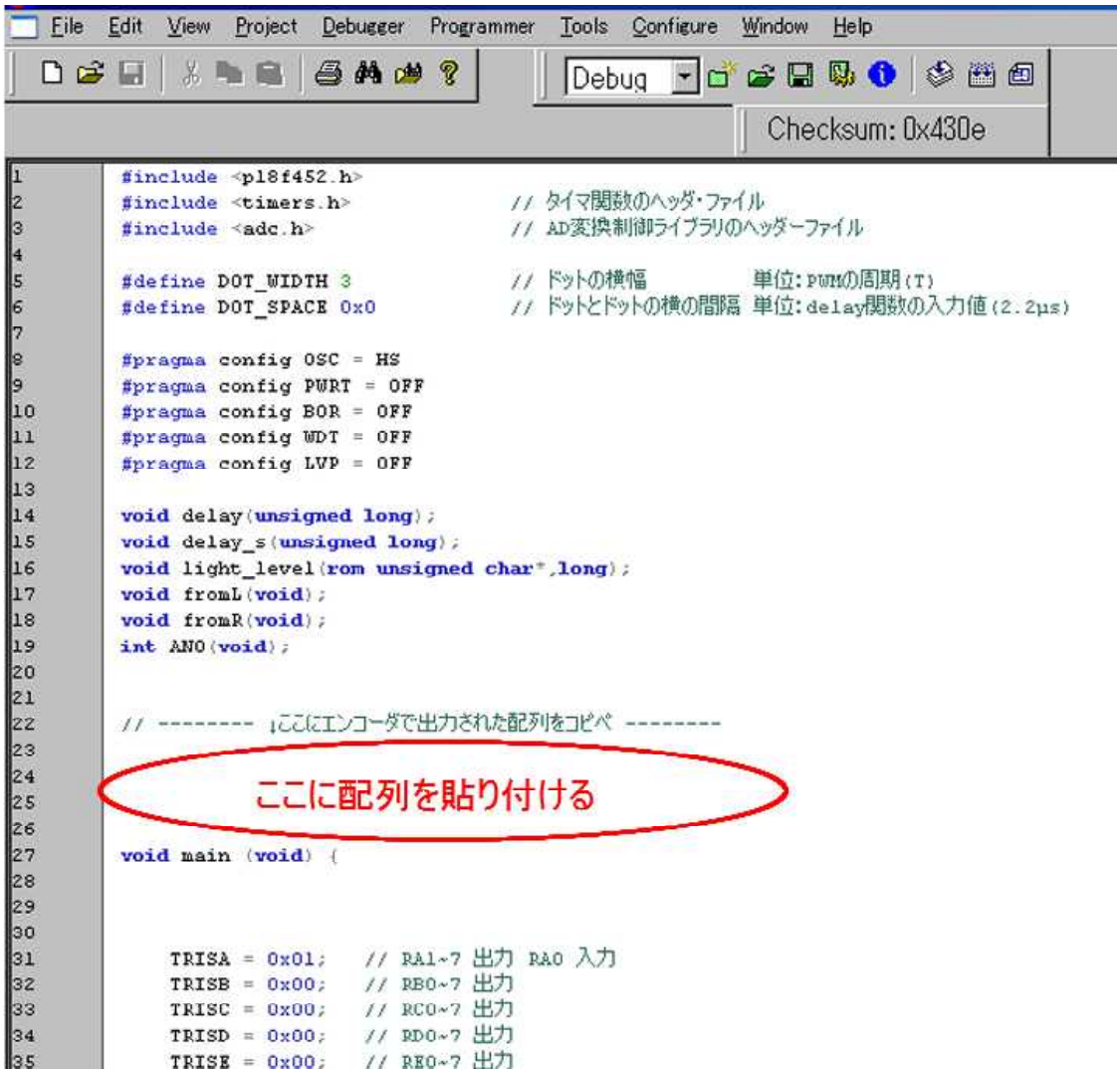


図２６　配列を張り付ける箇所（PIC 用プログラム）

貼り付けたらコンパイルして PIC（赤用）に書き込む。

同様にして青用、緑用の配列もプログラムと共にそれぞれの PIC へと書き込む。

これで準備は完了である。

6.4.　バーサライタの使い方

バーサライタのスイッチ及び振り方について説明する。

スイッチには電源スイッチ（S1）と別にリセット回路スイッチ（S2）がある。

・リセット回路スイッチが ON（導通）になっているのを確認し、電源スイッチを ON にする。

・１、２秒たってから続いてリセット回路スイッチを OFF（遮断）に切り替える。

電源スイッチを ON にした状態でリセット回路スイッチを ON から OFF に切り替えると PIC のリセット端子に電流が流れ、PIC が動作するようになる。

（本来リセット回路は、電源が立ち上がってすぐの不安定な状態の時にリセットが入らないように遅延する回路だが、遅延が不十分だったようなのでこのような仕様となってしまっている。（要修正））

これで電源は入ったのでバーサライタを一定のリズムで左右に振る。画像が表示されれば成功である。

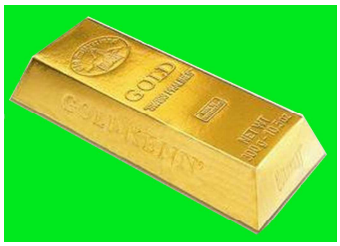
（上手く表示するには慣れが必要。）

7. 動作結果

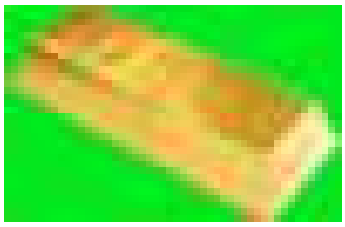
前述したぼかしありで色々な画像を表示してみた。

※上部に 2 本ある不自然な横線はフルカラー LED の赤の不備によるもの（要修正）

金野研究室ロゴ



元画像



表示する画像
（50 × 32）

バーサライタによる表示

サンプルピクチャ（ペンギン）

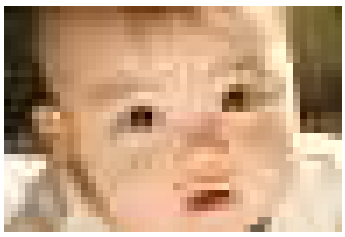
元画像

表示する画像
（42 × 32）

バーサライタによる表示

人物写真（赤ちゃん）

元画像



表示する画像
（48 × 32）

バーサライタによる表示

8. 改良点

改良が必要な点は各項目中で述べてきたが、特に改良が必要と思われるものをここでもう一度述べる。

8.1. 軽量化及び小型化

今回制作したバーサライタは、手動で振るには大きく重い物となってしまった。基板を固定するためのアクリル板が想定していたよりも重かったのが第一の原因だと考えられる。改善策としては振るための棒と LED の固定の方法や材料を変えることで、もっと軽量にすることが可能と思われる。また QFP タイプの PIC を使用することで基盤のサイズを小さくすれば小型化及び軽量化ができると思われる。

8.2. 表示位置のずれの改善

表示がやや不安定で時に光らないことがあることや、表示位置のずれが気になるところである。ある程度は振る方で合わせなくてはならないが、ソフトウェアの項でも述べたように表示位置を制御するプログラムを改良する必要があると思われる。

また、原因は回路にもあると思われる。詳しくは各項目を参照してほしい。

8.3. 画像を入れる手間の改善

表示したい画像を一々エンコードして PIC へと書き込まなくてはならないといういささか面倒な手法となってしまっている。

システムの根本に関わる部分だが、USB メモリなどの記憶媒体を PIC で扱うなどの方法を用いることで、今より大分楽に画像を表示させることが出来ると考えられる。

参考文献

［1］Microchip Technology Inc.,『 PIC18f452DataSheet 』

［2］3-TERMINAL POSITIVE VOLTAGE REGULATOR DataSheet

［3］KXM52 Series Accelerometers and Inclinometers Analog Output.

［4］OSTA71A1D-A DataSheet

［5］電子工作の実験室

http://www.picfun.com/index.html

［6］INE［Information Network Environment］

http://www.ine.sie.dendai.ac.jp/wiki/index.php?FrontPage

［7］ＰＩＣで遊ぶ電子工作

http://homepage3.nifty.com/mitt/pic/index.html#toc

付録

プログラムソース（PIC 用プログラム）

```
#include <p18f452.h>
#include <timers.h>                    // タイマ関数のヘッダ・ファイル
#include <adc.h>                       // AD 変換制御ライブラリのヘッダーファイル

#define DOT_WIDTH 3                    // ドットの横幅              単位：PWM の周期(T)
#define DOT_SPACE 0x0                  // ドットとドットの横の間隔  単位：delay 関数の入力値(2.2 μ s)

#pragma config OSC = HS
#pragma config PWRT = OFF
#pragma config BOR = OFF
#pragma config WDT = OFF
#pragma config LVP = OFF

void delay(unsigned long);
void delay_s(unsigned long);
void light_level(rom unsigned char*,long);
void fromL(void);
void fromR(void);
int AN0(void);


// -------- ↓ここにエンコーダで出力された配列をコピペ --------


void main（void）{
      TRISA = 0x01;    // RA1~7 出力  RA0 入力
      TRISB = 0x00;  // RB0~7 出力
      TRISC = 0x00;  // RC0~7 出力
      TRISD = 0x00;  // RD0~7 出力
      TRISE = 0x00;  // RE0~7 出力
      PORTA = 0xfe;
      PORTB = 0xff;
      PORTC = 0xff;
      PORTD = 0xff;
      PORTE = 0x07;
```

```
    // A/D 変換モジュールの動作モードを設定
    OpenADC(ADC_FOSC_64 &       //  クロック選択                Fosc/64(~40MHz)
            ADC_RIGHT_JUST &    //  結果の詰め方                右詰め
            ADC_1ANA_0REF,      //  使用チャンネル数とリファレンス    AN0 のみ/電源基準
            ADC_CH0 &           //  チャンネル指定              AN0
            ADC_INT_OFF);       //  割り込み                    無し


    while(1) {
        int data;

        data = AN0();
        if(data>0x3f0) {
            delay(20000);
            fromL();
        }else
        if(data<0x10) {
            delay(20000);
            fromR();
        }
    }
}

int AN0() {
    int data;
    SetChanADC(ADC_CH0); //  チャンネル再選択
    ConvertADC();        //  A/D 変換スタート
    while(BusyADC());    //  A/D 変換が終わるまで待機
    data = ReadADC();    //  結果を変数に移す
    return data;         //  戻り値として結果を返す
}
```

```
void fromL() {
	int i=0;
	while(1) {
		light_level(&lightsout_time[i][0],DOT_WIDTH);
		delay(DOT_SPACE);
		i++;
		if(i==WIDTH)break;
	}
}

void fromR() {
	int i=WIDTH-1;
	while(1) {
		light_level(&lightsout_time[i][0],DOT_WIDTH);
		delay(DOT_SPACE);
		i--;
		if(i==-1)break;
	}
}

void light_level(rom unsigned char *p,long t) {
	int count=0;
	while(1) {
		if(t==0)break;
		count++;
		if(count==0x00 || count==0x20 || count==0x30 || count==0x38 || count==0x3c || count==0x3e ||
count==0x39) {
			if(count>=*(p+0))	{LATBbits.LATB7 = 0;}	//01 個目の LED
			if(count>=*(p+1))	{LATBbits.LATB6 = 0;}	//02
			if(count>=*(p+2))	{LATBbits.LATB5 = 0;}	//03
			if(count>=*(p+3))	{LATBbits.LATB4 = 0;}	//04
			if(count>=*(p+4))	{LATBbits.LATB3 = 0;}	//05
			if(count>=*(p+5))	{LATBbits.LATB2 = 0;}	//06
			if(count>=*(p+6))	{LATBbits.LATB1 = 0;}	//07
			if(count>=*(p+7))	{LATBbits.LATB0 = 0;}	//08

			if(count>=*(p+8))	{LATDbits.LATD7 = 0;}	//09
			if(count>=*(p+9))	{LATDbits.LATD6 = 0;}	//10
			if(count>=*(p+10))	{LATDbits.LATD5 = 0;}	//11
			if(count>=*(p+11))	{LATDbits.LATD4 = 0;}	//12
```

```
            if(count>=*(p+12))   {LATCbits.LATC7 = 0;}    //13
            if(count>=*(p+13))   {LATCbits.LATC6 = 0;}    //14
            if(count>=*(p+14))   {LATCbits.LATC5 = 0;}    //15
            if(count>=*(p+15))   {LATCbits.LATC4 = 0;}    //16

            if(count>=*(p+16))   {LATAbits.LATA1 = 0;}    //17
            if(count>=*(p+17))   {LATAbits.LATA2 = 0;}    //18
            if(count>=*(p+18))   {LATAbits.LATA3 = 0;}    //19
            if(count>=*(p+19))   {LATAbits.LATA4 = 0;}    //20
            if(count>=*(p+20))   {LATAbits.LATA5 = 0;}    //21
            if(count>=*(p+21))   {LATEbits.LATE0 = 0;}    //22
            if(count>=*(p+22))   {LATEbits.LATE1 = 0;}    //23
            if(count>=*(p+23))   {LATEbits.LATE2 = 0;}    //24

            if(count>=*(p+24))   {LATCbits.LATC0 = 0;}    //25
            if(count>=*(p+25))   {LATCbits.LATC1 = 0;}    //26
            if(count>=*(p+26))   {LATCbits.LATC2 = 0;}    //27
            if(count>=*(p+27))   {LATCbits.LATC3 = 0;}    //28
            if(count>=*(p+28))   {LATDbits.LATD0 = 0;}    //29
            if(count>=*(p+29))   {LATDbits.LATD1 = 0;}    //30
            if(count>=*(p+30))   {LATDbits.LATD3 = 0;}    //31
            if(count>=*(p+31))   {LATDbits.LATD2 = 0;}    //32
        }
        if(count==0x40)
        {
            count=0;
            PORTA = 0xfe;
            PORTB = 0xff;
            PORTC = 0xff;
            PORTD = 0xff;
            PORTE = 0x07;
            t--;
        }
    }
}
```

```
// 遅延関数

// 約 2.2 μ s（OSC:40MHz）
// wait［μ s］= 88 / OSC［MHz］
void delay（unsigned long ms）｛
        unsigned long i;
        for  （i=0 ; i<ms ; i++）{}
}

// 約 0.1s（OSC:40MHz）
void delay_s（unsigned long t）｛
        unsigned long i;
        for  （i=0 ; i<t ; i++）{delay（45000）;}
}
```

プログラムソース（エンコーダ）

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define WIDTH 64                /*最大画像サイズ（横）*/
#define HEIGHT 32               /*最大画像サイズ（縦）*/

/*ファイルサイズ取得用関数*/
int getFileSize(FILE *fp) {
        fpos_t tmpSeek = 0;
        fpos_t fileSize = 0;

        tmpSeek = ftell(fp);          /*位置保存*/
        fseek(fp, 0L, SEEK_SET); /*先頭にセット*/
        fseek(fp, 0L, SEEK_END);/*サイズ取得*/
        fgetpos(fp,&fileSize);

        fseek(fp, tmpSeek, SEEK_SET);         /*位置復元*/

        return (int)fileSize;
}

/*置換関数*/
int strrep(char *buf, char *mae, char *ato)
{
      char *mituke;
      size_t maelen, atolen;

      maelen = strlen(mae);
      atolen = strlen(ato);
      if (maelen == 0 || (mituke = strstr(buf, mae)) == NULL) return 0;
      memmove(mituke + atolen, mituke + maelen, strlen(buf) - (mituke + maelen - buf ) + 1);
      memcpy(mituke, ato, atolen);
      return 1;
}
```

```
int main(int argc, char *argv[]){
    FILE *readFile;                 /*読み込みようファイル*/
    FILE *writeFile;                /*書き込み用ファイル*/
    unsigned char *buffer = NULL;   /*データの格納用*/
    unsigned char *tmpBuffer = 0;   /*データの操作用*/
    unsigned int fileSize = 0;      /*ファイルサイズの格納用*/
    int bmpWidth = 0;               /*ビットマップの横幅格納用*/
    int bmpHeight = 0;              /*ビットマップの縦幅格納用*/
    int paddingByte = 0;            /*埋める分のバイト数を格納*/
    int i = 0, j = 0;               /*ループ用*/
    char buf[1][50];                /*置換用*/
    int lightsout_time_b[WIDTH][HEIGHT];    /*消灯時間（青）*/
    int lightsout_time_g[WIDTH][HEIGHT];    /*消灯時間（緑）*/
    int lightsout_time_r[WIDTH][HEIGHT];    /*消灯時間（赤）*/

   strcpy(buf[0], argv[1]);
   while（strrep(buf[0], "bmp", "txt")）;

    /*引数の確認*/
    if（argc != 2）{
        printf("USAGE:argv[0] ReadFile WriteFile¥n");
        exit(1);
    }

    /*バイナリ読み込みモードでオープン*/
    if（(readFile = fopen(argv[1], "rb")) == NULL）{
        printf("ERR:ReadFile¥n");
        exit(1);
    }
    /*バイナリ書き込みモードでオープン*/
    if（(writeFile = fopen(buf[0], "w")) == NULL）{
        printf("ERR:WriteFile¥n");
        fclose(readFile);
        exit(1);
    }
    fileSize = getFileSize(readFile);
    /*メモリ確保*/
    if（(buffer =（unsigned char *)malloc(sizeof(unsigned char）* fileSize)) == NULL）{
        printf("ERR:Memory¥n");
```

```
        fclose(readFile);
        fclose(writeFile);
        exit(1);
}

/*ファイルからバッファにコピー*/
fread(buffer, sizeof(unsigned char), fileSize, readFile);

/*データから画像の幅と高さを得る*/
/*Windows24 ビットのビットマップファイルの場合*/
/*先頭から数えて 19 バイト目から 22 バイト目までに画像の横幅が*/
/*先頭から数えて 23 バイト目から 26 バイト目までに画像の縦幅が*/
/*ピクセル単位で格納されている*/
/*高さの値が負の場合左上から右下にかけて表示するタイプになるが*/
/*あまり一般的でないため今回は全部正のみとする*/
/*ので左下から右上に描画するタイプのみとする*/
memcpy(&bmpWidth, buffer + (sizeof(unsigned char) * 18), sizeof(unsigned char) * 4);
memcpy(&bmpHeight, buffer + (sizeof(unsigned char) * 22), sizeof(unsigned char) * 4);
printf("width = %d¥n", bmpWidth);
printf("height = %d¥n", bmpHeight);

/*24 ビットビットマップは 1 ピクセル 24 ビット(3 バイト)*/
/*で色を表示する(リトルエンディアンなので BGR の順番)*/
/*ビットマップは横幅が 4 バイト単位になるようになっているので*/
/*4 バイト単位にならない場合は余分にデータを追加されている*/
/*例)3x3 ピクセルの場合　3 バイト x3 ピクセル＝ 9 バイトなので*/
/*3 バイトのデータを余分に追加する*/

/*画像の横幅を割って 4 バイト境界になっているかどうか調べ*/
/*飛ばすバイト数を計算する*/
paddingByte = (bmpWidth * 3) % 4;
if (paddingByte != 0) {
        paddingByte = 4 - paddingByte;
}
```

```
/*RGB 三色の消灯時間をテキストファイルに配列で書き出す*/

/*BLUE*/
/*ヘッダを読み飛ばすため buffer に 54 足したアドレスを tmpBuffer に格納する*/
tmpBuffer = buffer + 54;

/*lightsout_time_b 配列の作成*/
for（i = 0; i < bmpHeight; i++）{
    for（j = 0; j < bmpWidth; j++）{
        lightsout_time_b[j][bmpHeight-1-i] = 0x40-(0x1<<（*tmpBuffer>>5）>>1);
        tmpBuffer = tmpBuffer+3;
    }
    tmpBuffer += paddingByte; /*余分なデータを読み飛ばす*/
}

/*lightsout_time_b 配列の書き出し*/
fprintf(writeFile,"//　↓青の配列（これを PIC 用プログラムにコピペして青用の PIC に書き込み）
¥n¥n",bmpWidth);
fprintf(writeFile,"#define WIDTH %d¥n",bmpWidth);
fprintf(writeFile,"#define HEIGHT %d¥n",bmpHeight);
fprintf(writeFile,"static rom unsigned char lightsout_time[WIDTH][HEIGHT] = {¥n");
for（i = 0; i < bmpWidth; i++）{
        fprintf(writeFile,"¥t{");
    for（j = 0; j < bmpHeight; j++）{
        fprintf(writeFile,"0x%-2x",lightsout_time_b[i][j]);
        if(j!=bmpWidth-1)　fprintf(writeFile,",");
    }
    fprintf(writeFile,"},¥n");
}
fprintf(writeFile,"};¥n¥n");
fprintf(writeFile,"// ↑青の配列¥n¥n¥n");
/*BLUE*/
```

```
	/*GREEN*/
	/*ヘッダを読み飛ばすため buffer に 54 足したアドレスを tmpBuffer に格納する*/
	tmpBuffer = buffer + 54;
	/*GREEN のデータ最初にするために BLUE のデータを読み飛ばす*/
	tmpBuffer = tmpBuffer + 1;

	/*lightsout_time_g 配列の作成*/
	for（i = 0; i < bmpHeight; i++）{
		for（j = 0; j < bmpWidth; j++）{
			lightsout_time_g[j][bmpHeight-1-i] = 0x40-(0x1<<（*tmpBuffer>>5）>>1);
			tmpBuffer = tmpBuffer+3;
		}
		tmpBuffer += paddingByte; /*余分なデータを読み飛ばす*/
	}

	/*lightsout_time_g 配列の書き出し*/
	fprintf(writeFile,"//　↓緑の配列（これを PIC 用プログラムにコピペして緑用の PIC に書き込み)
¥n¥n",bmpWidth);
	fprintf(writeFile,"#define WIDTH %d¥n",bmpWidth);
	fprintf(writeFile,"#define HEIGHT %d¥n",bmpHeight);
	fprintf(writeFile,"static rom unsigned char lightsout_time[WIDTH][HEIGHT] = {¥n");
	for（i = 0; i < bmpWidth; i++）{
			fprintf(writeFile,"¥t{");
		for（j = 0; j < bmpHeight; j++）{
			fprintf(writeFile,"0x%-2x",lightsout_time_g[i][j]);
			if(j!=bmpWidth-1)　fprintf(writeFile,",");
		}
		fprintf(writeFile,"},¥n");
	}
	fprintf(writeFile,"};¥n¥n");
	fprintf(writeFile,"// ↑緑の配列¥n¥n¥n");
	/*GREEN*/
```

```
/*RED*/
/*ヘッダを読み飛ばすため buffer に 54 足したアドレスを tmpBuffer に格納する*/
tmpBuffer = buffer + 54;
/*RED のデータ最初にするために BLUE,GREEN のデータを読み飛ばす*/
tmpBuffer = tmpBuffer + 2;

/*lightsout_time_r 配列の作成*/
for（i = 0; i < bmpHeight; i++）{
    for（j = 0; j < bmpWidth; j++）{
        lightsout_time_r[j][bmpHeight-1-i] = 0x40-(0x1<<（*tmpBuffer>>5）>>1);
        tmpBuffer = tmpBuffer+3;
    }
    tmpBuffer += paddingByte; /*余分なデータを読み飛ばす*/
}

/*lightsout_time_r 配列の書き出し*/
fprintf(writeFile,"//　↓赤の配列（これを PIC 用プログラムにコピペして赤用の PIC に書き込み）
¥n¥n",bmpWidth);
fprintf(writeFile,"#define WIDTH %d¥n",bmpWidth);
fprintf(writeFile,"#define HEIGHT %d¥n",bmpHeight);
fprintf(writeFile,"static rom unsigned char lightsout_time[WIDTH][HEIGHT] = {¥n");
for（i = 0; i < bmpWidth; i++）{
        fprintf(writeFile,"¥t{");
    for（j = 0; j < bmpHeight; j++）{
        fprintf(writeFile,"0x%-2x",lightsout_time_r[i][j]);
        if(j!=bmpWidth-1)　fprintf(writeFile,",");
    }
    fprintf(writeFile,"},¥n");
}
fprintf(writeFile,"};¥n¥n");
fprintf(writeFile,"// ↑赤の配列¥n¥n¥n");
/*RED*/
```

```
    /*メモリの解放とファイルのクローズ*/
    free(buffer);
    fclose(writeFile);
    fclose(readFile);
    return 0;
}
```