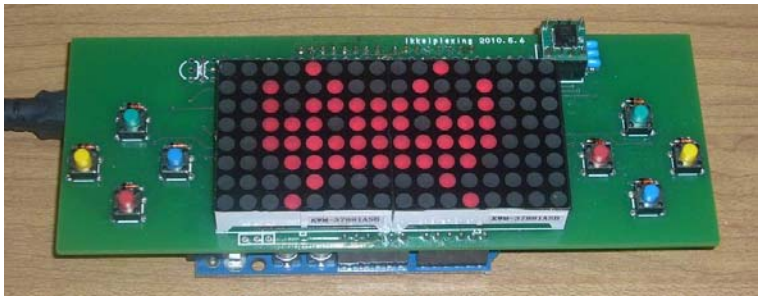
Arduinoを使った電子工作
8×16ドットマトリクスLEDで遊ぼう！



山西 一啓

ikkei

目次

0.　準備　電子工作の道具　1

ハンダゴテ

・ワット数は、電子工作なら15～20W程度で良いでしょう。
・コテ先はあまり細いと、温度が下がりやすいので、1mm程度で良い。
・コテ先は常にハンダで銀色に輝いているように保ちます。
　そのために、濡らしたスポンジの角で酸化物を取り除きます。
　また、最近のコテ先は鉄メッキされているので、絶対にヤスリで
　削ってはいけません。

私の愛用品

HOZAN HS-11

ANTEX製 18W

コテ台

スポンジを半分に切って角を使います。
合計16個の角が使えます。

電子工作の道具　2

ハンダ

・1mm位の太さが一番使いやすい。
・細かいチップ部品用など必要に応じて、もう少し細い物を用意して
　も良い。

ハンダ吸い取り線

・ハンダ付けの修正をするとき、ハンダを溶かしてこれで吸い取ります。
・フラックスが付いていてハンダがなじみ易いものを選びます。
・古いハンダやこびりついたような場合は、ハンダを足してから
　吸い取ります。
・あまり細いと吸わないので、2mm位の物が使いやすい。

私の愛用品

電子工作の道具　3

ニッパー

・刃の先端がピッタリ合うか確認しましょう。
・手に持って、握りやすい大きさを選びましょう。
・鉄などの固い物は、刃が傷むので切らないようにしましょう。

ピンセット

・挟んでも曲がらないように、厚みのある堅い物を選びましょう。
・先端はとがっていると曲がりやすいので、平らでピッタリ合う物を選びましょう。

私の愛用品

VICTOR　110

HOZAN　P-892

電子工作の道具　4

ワイヤストリッパー

・ビニール線などの被覆を剥くのに使います。
・ニッパーで剥くと、銅線を傷つける恐れがあります。
・AWGは数字の大きい方が細い線です。24～30

カッター

・基板のパターンをカットしたり、必要に応じて使います。

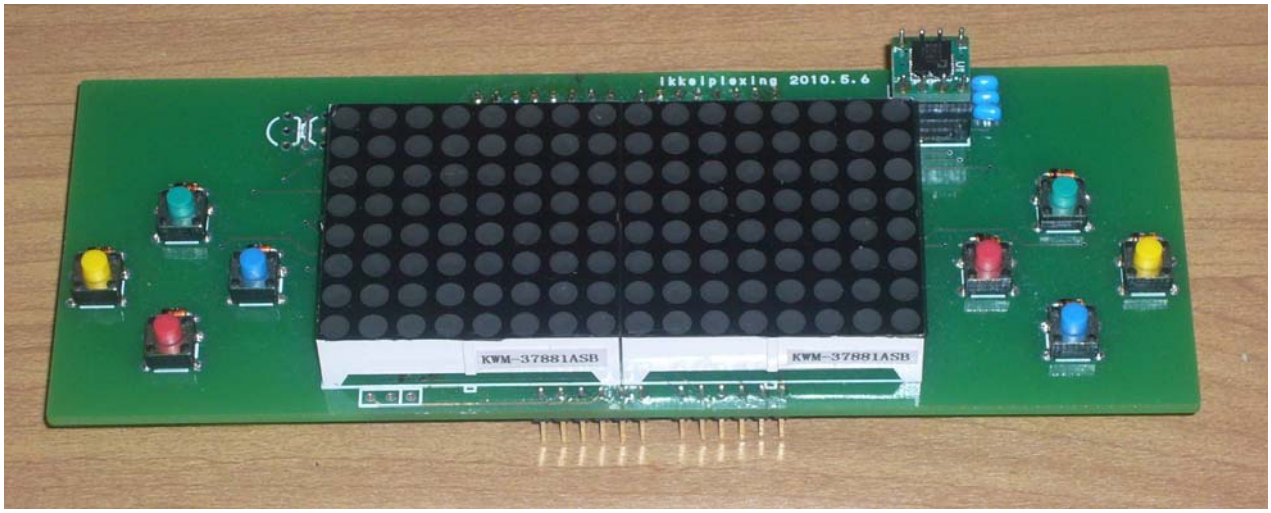
ラジオペンチ

・有れば便利だが、あまり使わないのでとりあえず無くても良い。

私の愛用品

1．　8x16ドットマトリクスLEDシールドの製作

・専用基板に部品をハンダ付けするだけで、簡単に製作出来ます。
・加速度センサやボタンスイッチも取り付けるので、ゲームなどに応用出来ます。

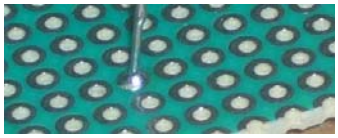


（1）ハンダ付けのコツ

1．コテ先を部品のリード線とパターンの両方に当てます。

2．そこへハンダを持っていって溶かします。

3．ハンダが流れ込むのを見届けてからコテ先を離します。



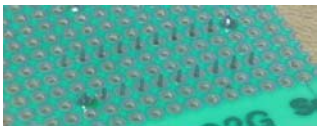
・コテ先を離したあとは息を吹いたりしないで、自然にさまします。
・コテ先は常にハンダが銀色できれいな状態にしておきます。
　くすんでいたらハンダを追加します。
・余分なハンダはスポンジの角で落とします。
　たたき落とすようなことは絶対にしてはいけません。

ハンダ付けの順番

・基本は**背の低い部品から**ハンダ付けしていきます。
・リード線が2本の部品は、まず1本ハンダ付けしたあとで、
部品の浮きを修正して、もう1本ハンダ付けします。

・余分なリード線はハンダ付けした後でニッパーで切断します。
その時、切れ端が飛ばないように注意します。短く切りすぎないように。

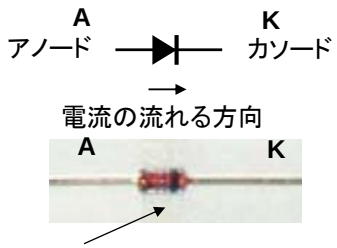
・端子がたくさんある部品は、始めに対角線上の2個の端子をハンダ付け
したあと、部品を押さえながらハンダを溶かして部品の浮きを無くします。



部品の向きに注意！

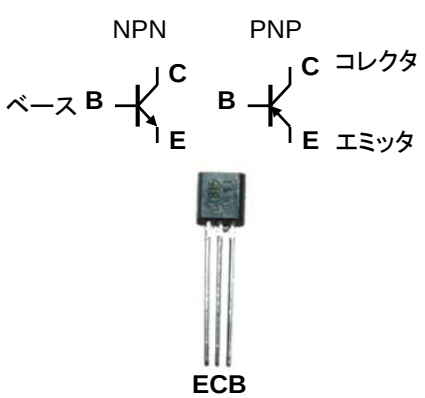
・**部品の向きに注意**して、挿してから再度向きを確認します。
ダイオードやICなどの部品には方向があります。
間違って逆にしてしまうと付け直すのは大変です。
念には念を入れて何度も確認しましょう。

ダイオード



LED

トランジスタ

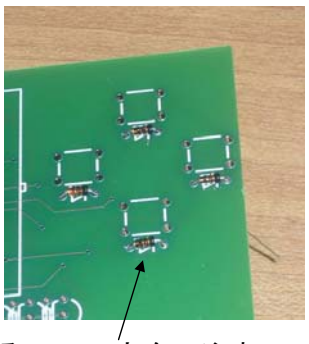


（2）ハンダ付けの手順　1

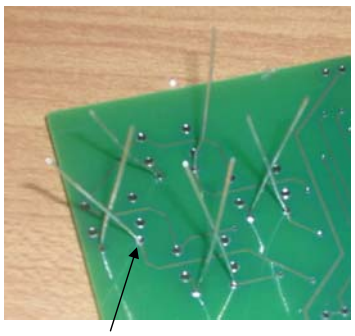
ダイオード

1. ダイオードのリード線を曲げてから基板に挿します。
2. ダイオードには方向があります。間違わないように確認します。
3. まず片方だけハンダ付けします。
4. 基板から浮いていれば、ハンダを溶かして修正します。
5. 残りのリード線もハンダ付けします。
6. リード線をニッパーでカットします。
   切ったリード線は飛ばさないように。
   これはジャンパーに使いますので、取っておきます。

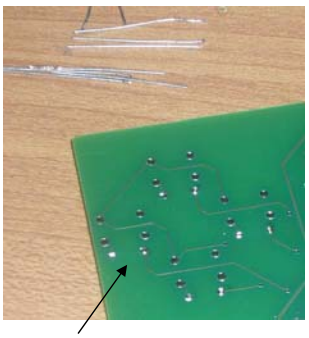
リード線を曲げる



方向に注意



片方から

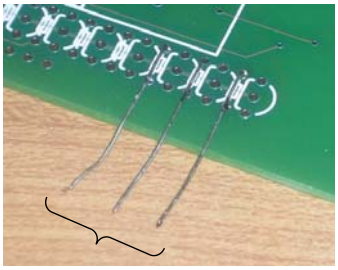


両方付けてカットする

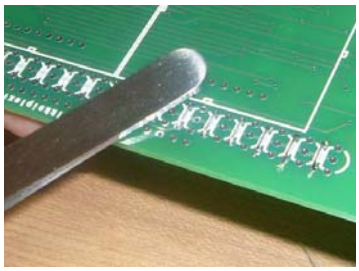
ハンダ付けの手順　2

ジャンパー

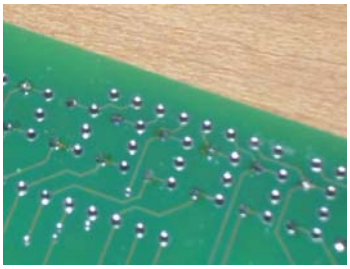
1. ダイオードのリード線をジャンパーに使用します。
2. 左側の3本はリード線を挿さないで上だけでハンダ付けします。
   ピンセットを使って位置を修正します。
3. 他のジャンパーはリード線を通し、ピンセットの持ち手で押さえます。
4. ジャンパーも片方からハンダ付けします。



3本だけは挿さずに付ける



挿してからピンセットで押さえる

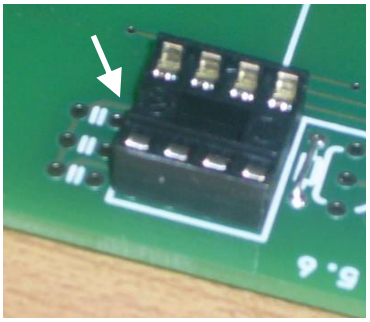


両方ハンダ付けしてカットする

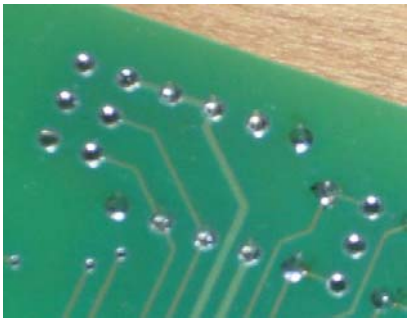
ハンダ付けの手順　3

8Pソケット

1. 向きを確認します。印刷のへこみにソケットのへこみを合わせます。
2. まず、対角の2本のピンをハンダ付けします。
3. ソケットを押さえながら1本ずつハンダを溶かして、
   浮きを無くします。
4. 全部のピンをハンダ付けします。



印刷のへこみにソケットのへこみを合わせる



まず対角の2ピンをハンダ付けする

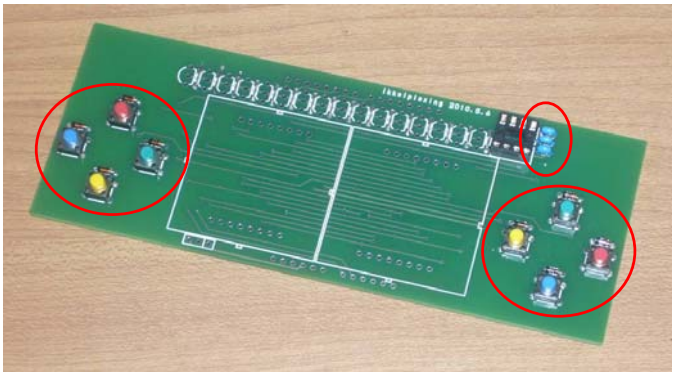
ハンダ付けの手順　4

コンデンサ

1. 3個をハンダ付けします。
2. これも片方のリード線からハンダ付けします。

スイッチ

1. まず、対角の2本のピンをハンダ付けします。
2. 押さえながら1本ずつハンダを溶かして浮きを無くします。
3. 全部ハンダ付けします。

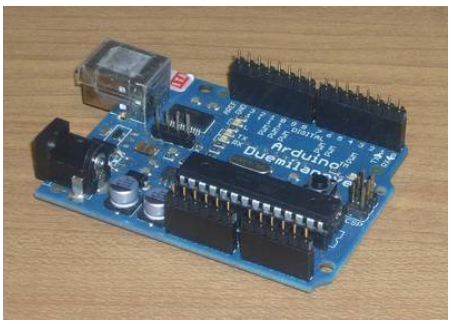


コンデンサとスイッチを取り付ける

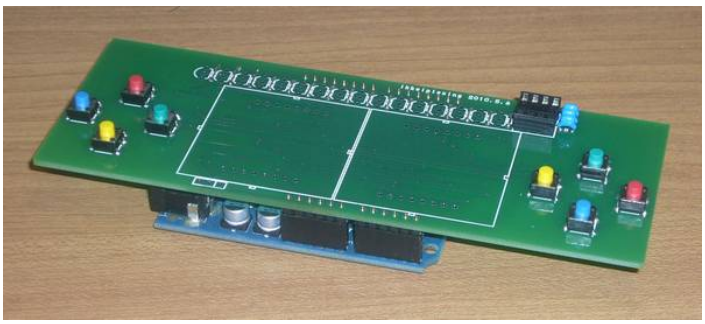
ハンダ付けの手順　５

ヘッダピン

1. Arduinoにヘッダピンを挿します。
2. その上から基板をかぶせます。
3. まずヘッダピンの両端の2本のピンをハンダ付けします。
4. 浮きが無いか確認します。
5. Arduinoにあまり熱が行かないように、
   手早くハンダを付けていきます。



ヘッダピンはArduinoに挿す

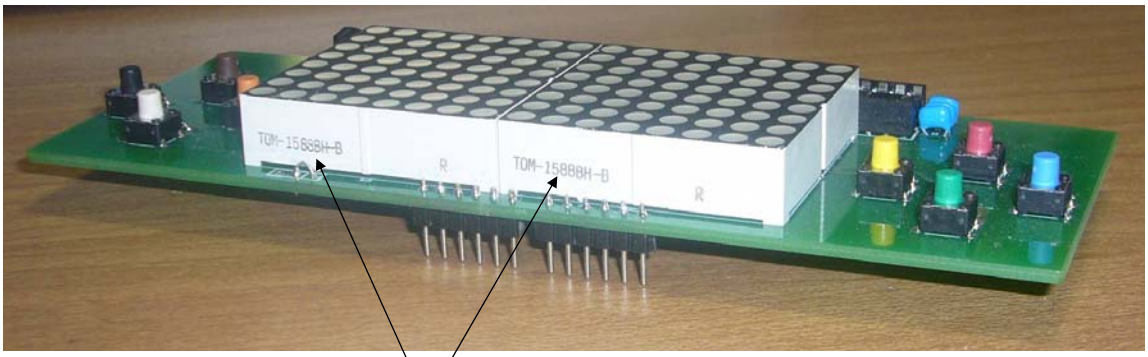


基板をかぶせる

ハンダ付けの手順　６

LED

1. LEDの向きを合わせて差し込みます。部品名が手前になります。
2. やはり、対角の2本のピンを先にハンダ付けします。
3. しっかり押さえながら、ハンダを溶かして浮きを無くします。
4. 全部のピンをハンダ付けします。



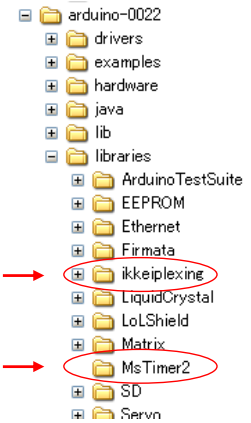
部品名を手前にする

2.　スケッチの説明

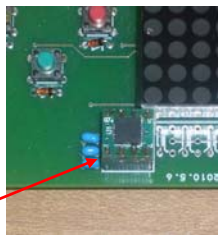
(1) LED チカチカ	プログラムの基本
(2) 反射	ドットを動かす基本
(3) LEDコロコロ	加速度センサの応用
(4) キッチンタイマ	数字表示と時間
(5) 電光掲示板	文字表示
(6) パターン表示	ドットパターンを表示
(7) ゲーム	ライブラリの移植

スケッチのインストールなど

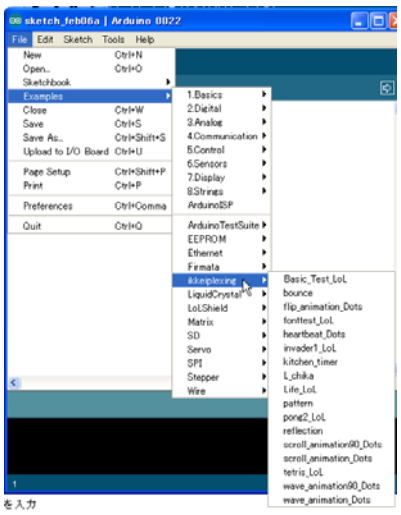
インストール
ikkeiplexing フォルダと
MsTimer2 フォルダを
arduino-0022¥libraries
フォルダ内にコピーする



秋月電子のKXM52-1050を方向に注意して、
写真の様に取り付けます。



スケッチの選択
File – Examples – ikkeiplexing
メニューから選択する



スケッチの保存
上書き出来ないので、他のところに保存する

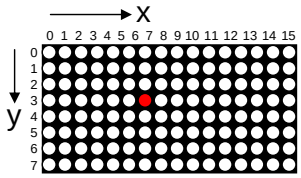
(1) LED チカチカ

・L_Chika

```
#include <MsTimer2.h>
#include <Charlieplexing.h>

void setup()
{
  iLedSign::Init();
}

void loop()
{
  static uint8_t x = 6;
  static uint8_t y = 3;
  static uint8_t s = 1;

  s ^= 1;
  iLedSign::Set(x, y, s);
  delay(500);
}
```

タイマ割り込み
LED表示処理
必須

LED表示の初期化



LED表示座標

LED表示出力(s=0:消灯 s=1:点灯)

(2) 反射

・reflection

```
const uint8_t X_MAX = 15;
const uint8_t Y_MAX = 7;

void setup()
{
  iLedSign::Init();
}

void loop()
{
  static uint8_t x = 0;
  static uint8_t dx = 1;
  static uint8_t y = 0;
  static uint8_t dy = 1;
  static uint8_t s = 1;

  iLedSign::Set(x, y, 0);
  x += dx;
  y += dy;
  iLedSign::Set(x, y, 1);

  if ((x <= 0)||(x >= X_MAX)){
    dx = -dx;
  }
  if ((y <= 0)||(y >= Y_MAX)){
    dy = -dy;
  }
  delay(50);
}
```

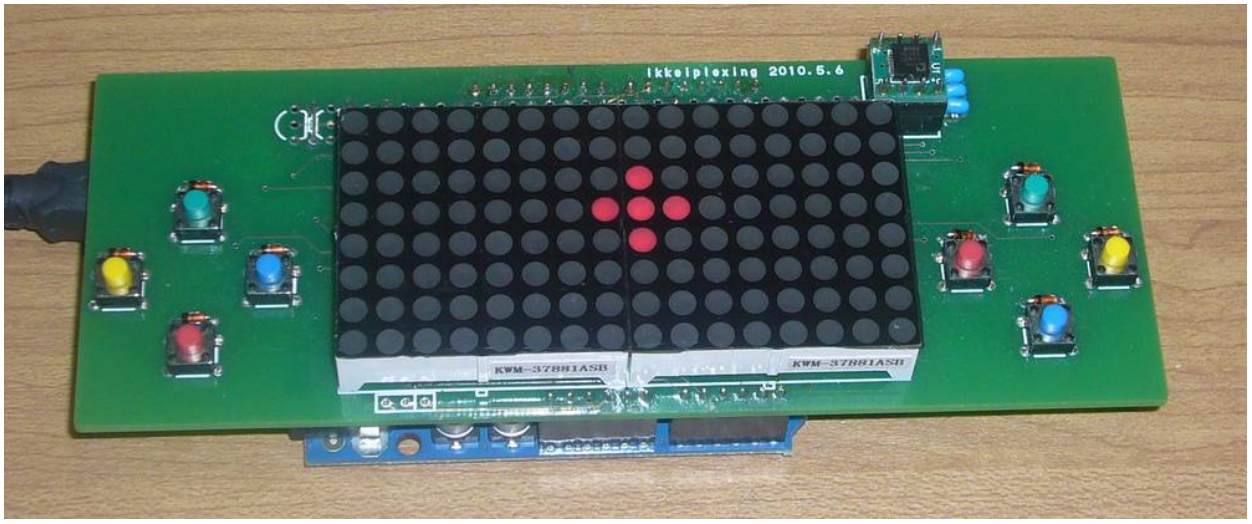
端を定義 (x:0～15 y:0～7)

前の点を消灯
次の点の座標を計算
次の点を点灯

端まで来れば反転

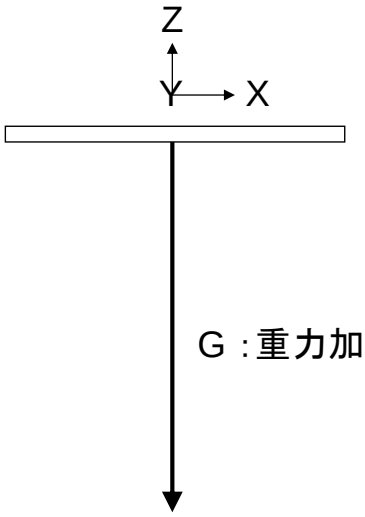
(3) LEDコロコロ

LEDをボールに見立てて、加速度センサでコロコロ動かします。
上下左右の端で跳ね返って、面白い動きをします。

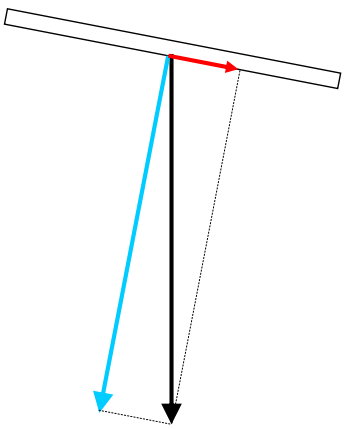


加速度センサについて

水平の場合、
X,Yの加速度はゼロ

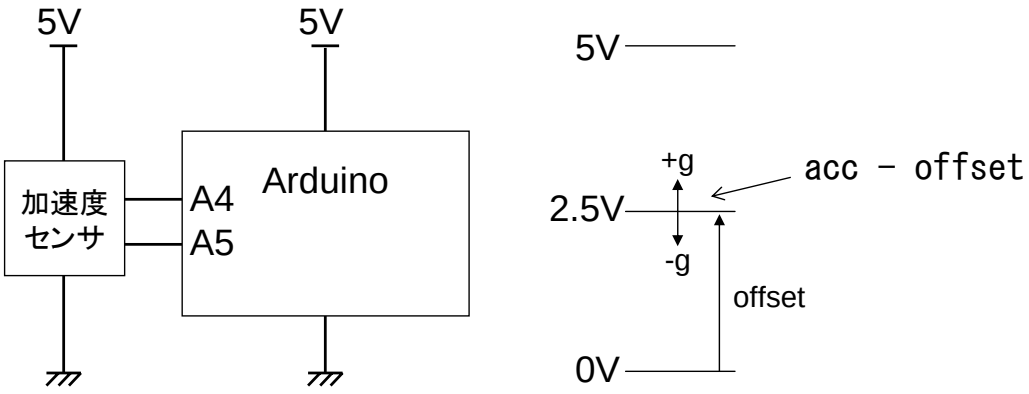


傾けると、
Gの分力が加速度センサのX,Yで測定出来ます。

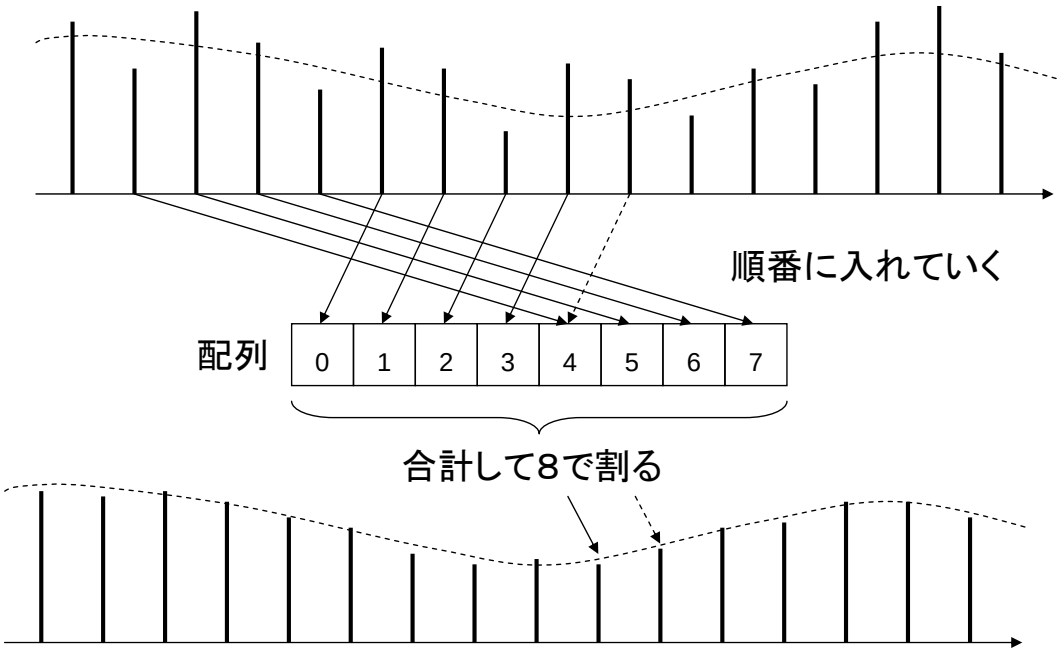


キャリブレーション

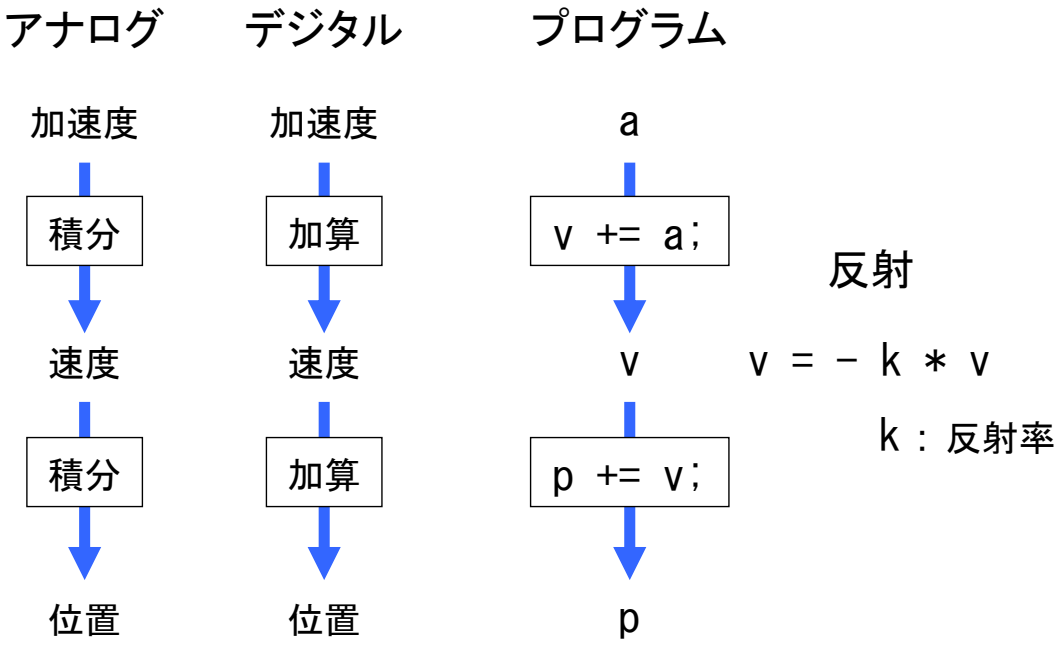
水平時にSW1 を押すと その時点の値をoffsetとし、
ゼロ調整（キャリブレーション）を行います。
同時にEEPROMにも保存します。
そして、次回の起動時にEEPROMからoffsetを読み出しますので、
毎回キャリブレーションしなくて済みます。



移動平均



加速度、速度、位置

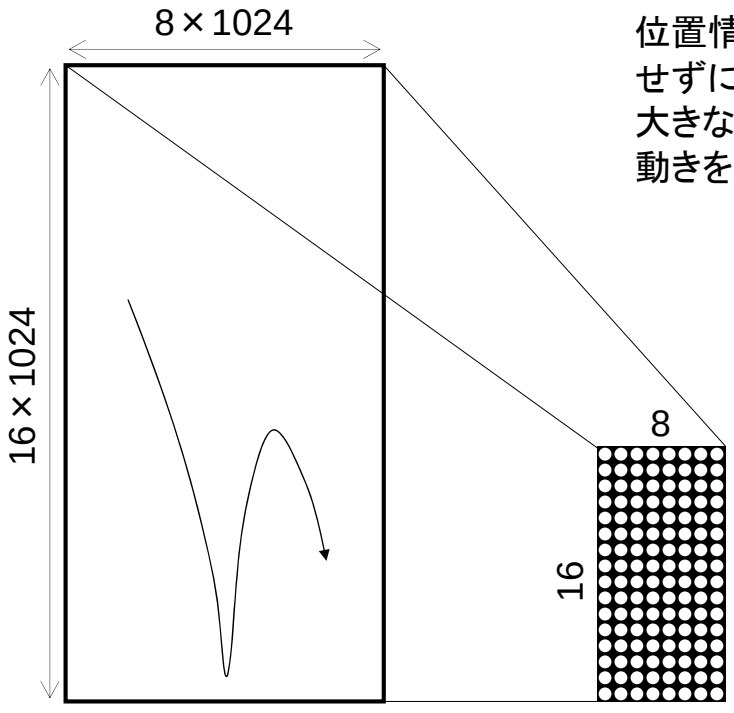


反射

v = − k ＊ v

k：反射率

スケーリング



位置情報を直接LEDに表示せずに、広い空間に配置し、大きな動きを縮小することで、動きをスムーズに見せます。

オフセットの処理

・bounce

```
#include <EEPROM.h>

void setup(){
  iLedSign::Init();
  offset_x = EEPROM.read(0)*256 + EEPROM.read(1);
  offset_y = EEPROM.read(2)*256 + EEPROM.read(3);
  pol_x = 1;
  pol_y = -1;
}

void loop(){
  if ((iLedSign::GetPeriod() & FRAME) != 0){
    uint16_t keydata = iLedSign::GetSW(SW_EDGE); // get switch edge
    calc_acc();

    if ((keydata & SW1) !=0 ){
      offset_x = acc_xin[ 0 ];
      offset_y = acc_yin[ 0 ];
      EEPROM.write(0, offset_x / 256);
      EEPROM.write(1, offset_x % 256);
      EEPROM.write(2, offset_y / 256);
      EEPROM.write(3, offset_y % 256);
    }
    if ((keydata & SW4) !=0 ){
      pol_x *= -1;
      pol_y *= -1;
    }
  }
}
```

EEPROMを使用する時インクルードする

オフセットデータをEEPROMから読み出す

フレームの検出

スイッチデータをエッジで取得する

加速度を取得して演算処理して点灯位置を移動させる

SW1なら
現在の加速度センサの直データをオフセット値とする

オフセット値(2バイト)を1バイト毎に分けて
EEPROMに書き込む

SW4なら
極性を反転する

移動平均処理と加速度、速度、位置の計算

・bounce

```
void calc_acc( void ){
  // averaging acceleration sensor signal
  acc_xin[ spt ] = analogRead( AX );
  acc_yin[ spt ] = analogRead( AY );
  spt++;
  if ( spt >= AVE ){
    spt = 0;
  }
  acc_x = 0;
  acc_y = 0;
  for ( i=0; i < AVE; i++){
    acc_x += ( acc_xin[ i ] - offset_x ) * pol_x;
    acc_y += ( acc_yin[ i ] - offset_y ) * pol_y;
  }
  acc_x /= AVE;
  acc_y /= AVE;

  // integrating acceleration makes velocity
  vel_x += acc_x * SENS;
  vel_y += acc_y * SENS;

  // integrating velocity makes position
  pos_x += vel_x;
  pos_y += vel_y;
```

加速度センサからデータを取得して配列に入れる

次のデータのために、配列のポインタを更新する

配列の内容からオフセット値を引いて合計し、個数で割る
ことによって平均値を算出する
その際、極性も掛けておく

加速度を積分して速度にする

速度を積分して位置にする

反射とスケーリング

・bounce

```
  // reflection ( reverse velocity )
  // 0 < x < X_MAX
  if ( pos_x >= MAG * X_MAX){
    pos_x = MAG * X_MAX - 1;
    vel_x = -( vel_x * REF1 ) / REF2;
  }else if ( pos_x < MAG ){
    pos_x = MAG;
    vel_x = -( vel_x * REF1 ) / REF2;
  }
  // 0 < y < 7
  if ( pos_y >= MAG * Y_MAX){
    pos_y = MAG * Y_MAX - 1;
    vel_y = -( vel_y * REF1 ) / REF2;
  }else if ( pos_y < MAG ){
    pos_y = MAG;
    vel_y = -( vel_y * REF1 ) / REF2;
  }

//  x = acc_x + 8;    // test acceleration value
//  y = acc_y + 4;
//  x = vel_x / 16 + 8;  // test velocity value
//  y = vel_y / 16 + 4;
  x = pos_x / MAG;
  y = pos_y / MAG;
```

端まで来たら速度を反転する
その際、反射係数(REF1/REF2)を掛けておく
整数演算なのでREF1を掛けてからREF2で割る

加速度を表示したいときにコメントを外す

速度を表示したいときにコメントを外す

位置をスケーリング係数で割ってLEDの表示内に収める

LEDデータの作成と消去

・bounce

```
  // display limitter
  if ( x > (X_MAX - 1)) {
    x = X_MAX - 1;
  }else if ( x < 1 ){
    x = 1;
  }
  if ( y > (Y_MAX - 1)) {
    y = Y_MAX - 1;
  }else if (y < 1){
    y = 1;
  }

  // turn off the last position
  iLedSign::Set(x0, y0, 0);
  iLedSign::Set(x0+1, y0, 0);
  iLedSign::Set(x0-1, y0, 0);
  iLedSign::Set(x0, y0+1, 0);
  iLedSign::Set(x0, y0-1, 0);

  // turn on the next position
  iLedSign::Set(x, y, 1);
  iLedSign::Set(x+1, y, 1);
  iLedSign::Set(x-1, y, 1);
  iLedSign::Set(x, y+1, 1);
  iLedSign::Set(x, y-1, 1);

  // store last position
  x0 = x;
  y0 = y;
}
```

念のため、LEDの表示領域内に制限する

前回の位置のLEDデータを0にして消去する

今回の位置のLEDデータを1にする

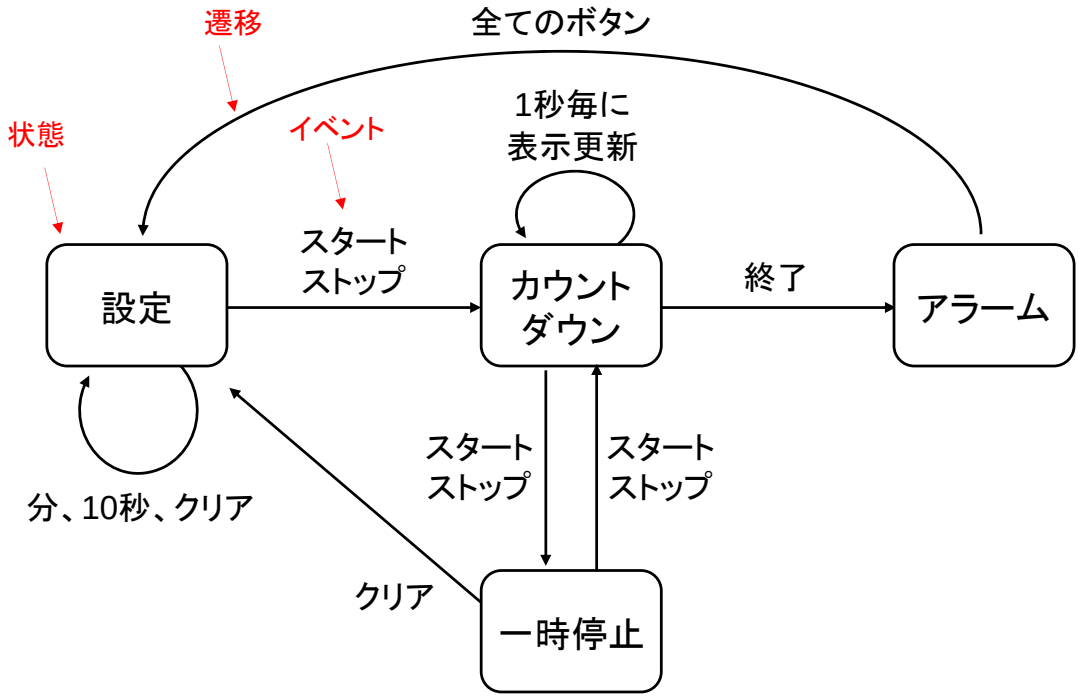
現在の位置を保存して、次回に前回の位置とする

(4) キッチンタイマ

仕様
- 「分」「10秒」ボタンで時間を設定する
- 「分」ボタンを押すと分表示が
  0から1ずつ増え、9の次は0に戻る。
- 「10秒」ボタンを押すと秒表示が
  00から10ずつ増え、50の次は00に戻る。
- 「クリア」ボタンで設定した時間を0分00秒にする。
- 「スタートストップ」ボタンで動作を開始し、
  表示が1秒ずつカウントダウンしていく。
- カウントダウン中に「スタートストップ」ボタンを
  押すと一時停止し、もう一度押すと再開する。
- 設定した時間が経過したらアラームが鳴る。
- アラームはどのボタンを押しても止まる。
- 止めた後は最初の状態に戻り、設定した時間を
  表示する。

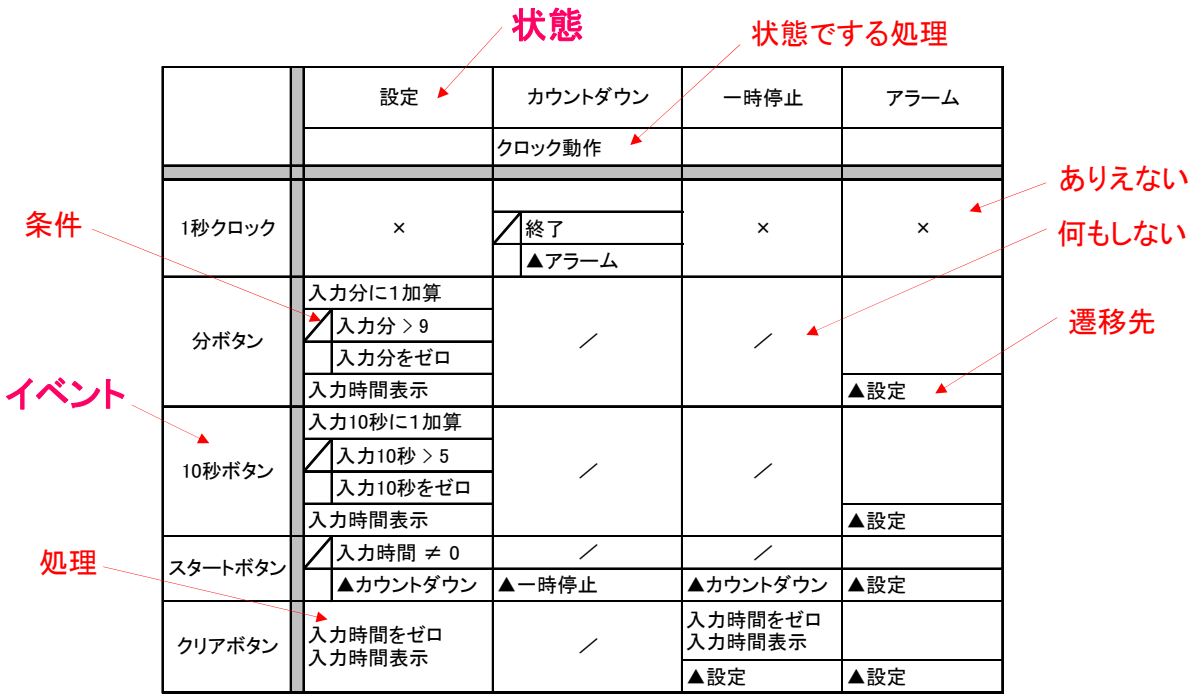
キッチンタイマの動作を状態遷移図で設計する

概要設計：仕様書から状態遷移図を作成します。



キッチンタイマの動作を状態遷移表で設計する

詳細設計：まずは状態遷移図から状態、イベント、遷移先などを表に入れます。



	設定	カウントダウン	一時停止	アラーム
		クロック動作		
1秒クロック	×	終了 ▲アラーム	×	×
分ボタン	入力分に1加算 入力分 > 9 入力分をゼロ 入力時間表示	／	／	▲設定
10秒ボタン	入力10秒に1加算 入力10秒 > 5 入力10秒をゼロ 入力時間表示	／	／	▲設定
スタートボタン	入力時間 ≠ 0 ▲カウントダウン	／ ▲一時停止	／ ▲カウントダウン	▲設定
クリアボタン	入力時間をゼロ 入力時間表示	／	入力時間をゼロ 入力時間表示 ▲設定	▲設定

キッチンタイマの動作を状態遷移表で設計する

詳細設計：終了条件を書き直し、詳細の処理を記入します。

	設定	カウントダウン	一時停止	アラーム
		クロック動作		
1秒クロック	×	カウンタ1減算 カウント表示 カウンタ == 0 アラーム表示 ▲アラーム	×	×
分ボタン	入力分に1加算 入力分 > 9 入力分をゼロ 入力時間表示	／	／	アラーム消去 入力時間表示 ▲設定
10秒ボタン	入力10秒に1加算 入力10秒 > 5 入力10秒をゼロ 入力時間表示	／	／	アラーム消去 入力時間表示 ▲設定
スタートボタン	入力時間 ≠ 0 カウンタ設定 クロッククリア ▲カウントダウン	／ ▲一時停止	／ ▲カウントダウン	アラーム消去 入力時間表示 ▲設定
クリアボタン	入力時間をゼロ 入力時間表示	／	入力時間をゼロ 入力時間表示 ▲設定	アラーム消去 入力時間表示 ▲設定

キッチンタイマの動作を状態遷移表で設計する

詳細設計：共通処理を状態のところに記入します。

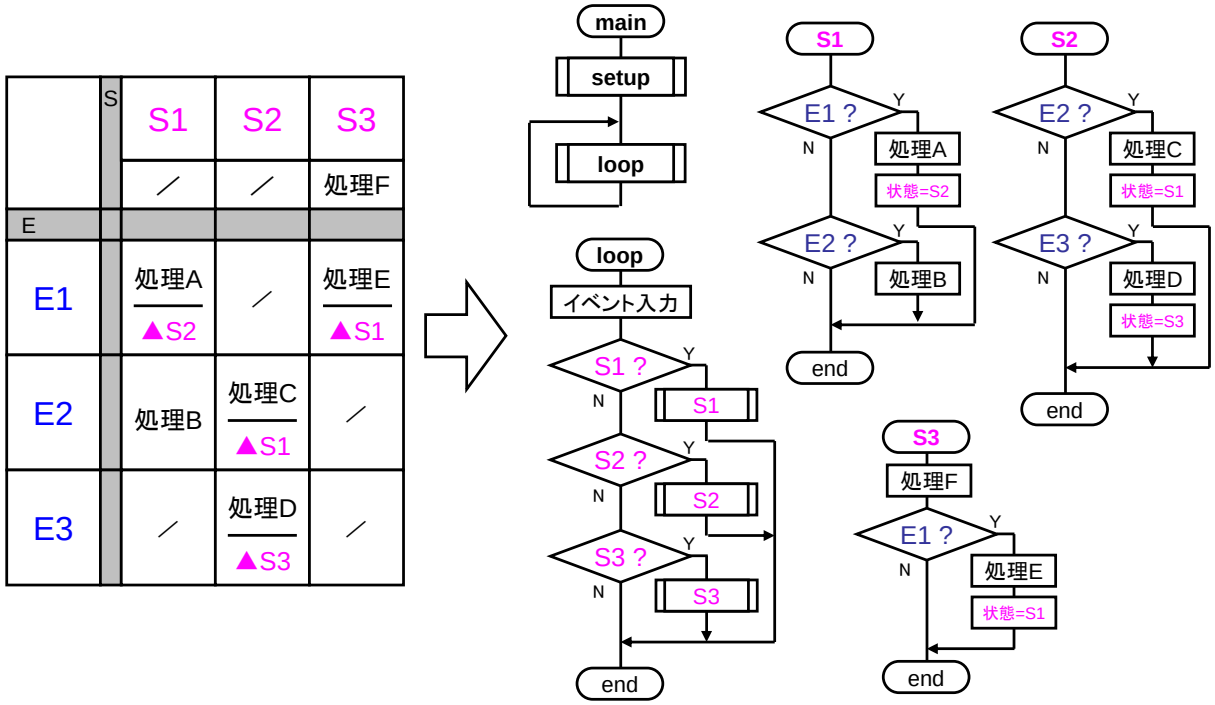
色分けすると分かり易くなります

	設定	カウントダウン	一時停止	アラーム
	入力時間表示	クロック動作	／	アラーム表示
1秒クロック	×	クロッククリア カウンタ1減算 カウンタ表示 ／ カウンタ == 0 ▲アラーム	×	×
分ボタン	入力分に1加算 ／ 入力分 > 9 入力分をゼロ	／	／	アラーム消去 ▲設定
10秒ボタン	入力10秒に1加算 ／ 入力10秒 > 5 入力10秒をゼロ	／	／	アラーム消去 ▲設定
スタートボタン	／ 入力時間 ≠ 0 カウンタ設定 クロッククリア ▲カウントダウン	／ ▲一時停止	／ ▲カウントダウン	アラーム消去 ▲設定
クリアボタン	入力時間をゼロ	／	入力時間をゼロ ▲設定	アラーム消去 ▲設定

状態遷移表からソースコード作成

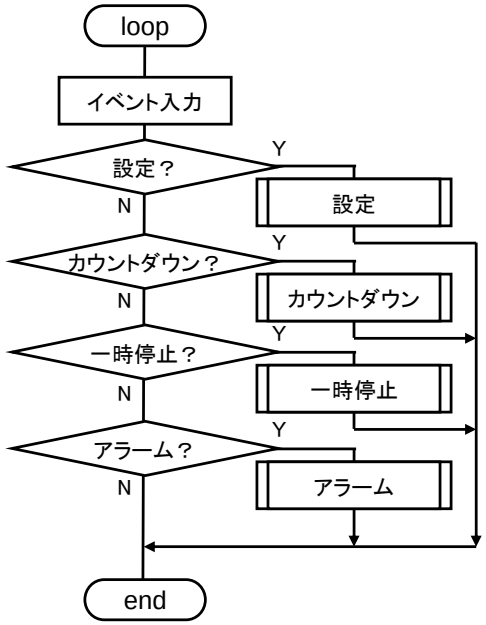
状態遷移表からプログラムコードへの変換は、下図の様に行います。

	S	S1	S2	S3
		／	／	処理F
E				
E1		処理A ▲S2	／	処理E ▲S1
E2		処理B	処理C ▲S1	／
E3		／	処理D ▲S3	／



キッチンタイマのソースコード作成　1

キッチンタイマの場合なら、下図のようにコードを書きます。

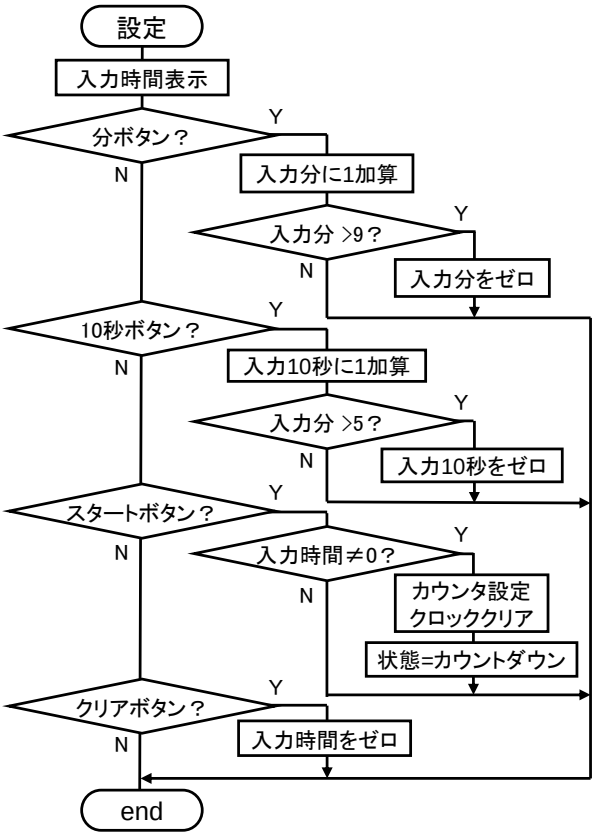


スイッチデータをエッジで取得する

10ms のインターバルをチェック

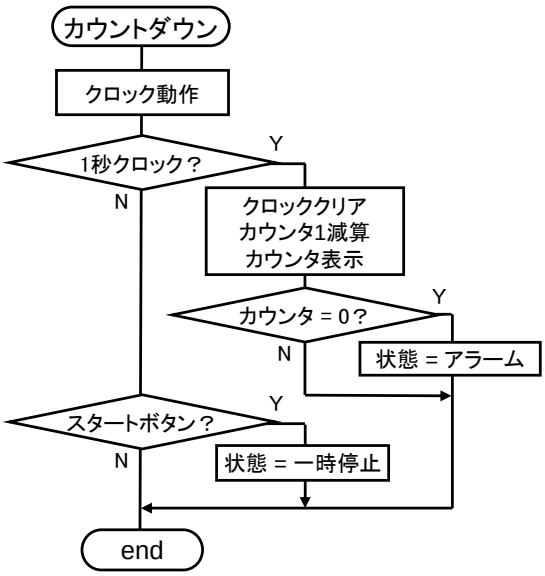
```
void loop(){
  if ((iLedSign::GetPeriod() & TENMS) != 0){
    event = iLedSign::GetSW(EDGE);
    switch (state){
      case set_time:    set_time_state();    break;
      case count_down:  count_down_state();  break;
      case pause:       pause_state();       break;
      case alarm:       alarm_state();       break;
    }
  }
}
```


キッチンタイマのソースコード作成　2



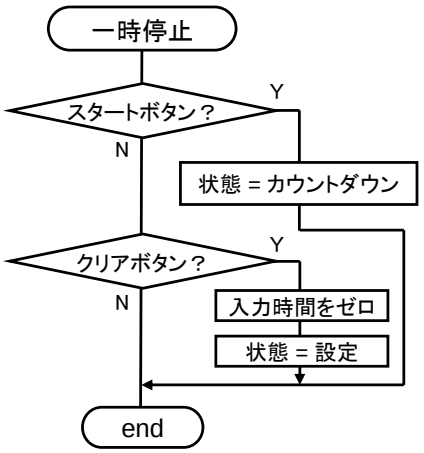

```
void set_time_state(void){
  DisplayLED(set_minute, set_tensec, 0);
  if ((event & SW_MINUTE) != 0){          ← 分ボタン
    set_minute++;
    if (set_minute > 9){
      set_minute = 0;
    }
  } else if ((event & SW_TENSEC) != 0){   ← 10秒ボタン
    set_tensec++;
    if (set_tensec > 5){
      set_tensec =0;
    }
  } else if ((event & SW_START) != 0){    ← スタートボタン
    if ((set_minute != 0)||(set_tensec != 0)){
      count_minute = set_minute;
      count_tensec = set_tensec;
      count_second = 0;
      clock = 0;
      state = count_down;
    }
  } else if ((event & SW_CLEAR) != 0){    ← クリアボタン
    set_minute = 0;
    set_tensec = 0;
  }
}
```

キッチンタイマのソースコード作成　3

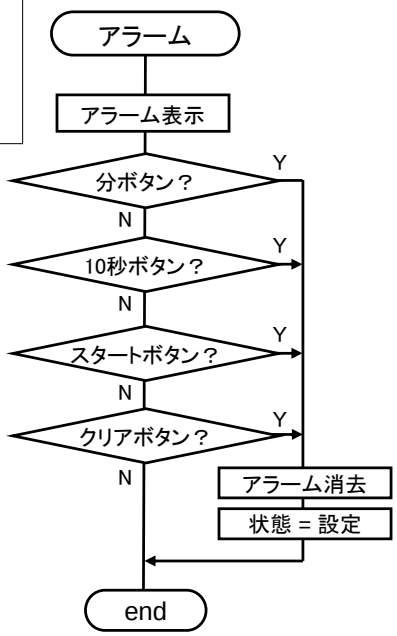



```
void count_down_state(void){
  clock++;
  if (clock >= 100){                          ← 1 sec = 10ms X 100
    clock = 0;
    if (count_second > 0){
      count_second--;
    } else {
      count_second = 9;
      if (count_tensec > 0){
        count_tensec--;                       } カウンタ1減算
      } else {
        count_tensec = 5;
        if (count_minute > 0){
          count_minute--;
        }
      }
    }
    DisplayLED(count_minute, count_tensec, count_second);
    if ((count_second == 0)&&(count_tensec == 0)&&(count_minute == 0)){
      alarm_count = ALARM_PERIOD;
      state = alarm;
    }
  }
  if ((event & SW_START) != 0){
    state = pause;
  }
}
```


キッチンタイマのソースコード作成　4



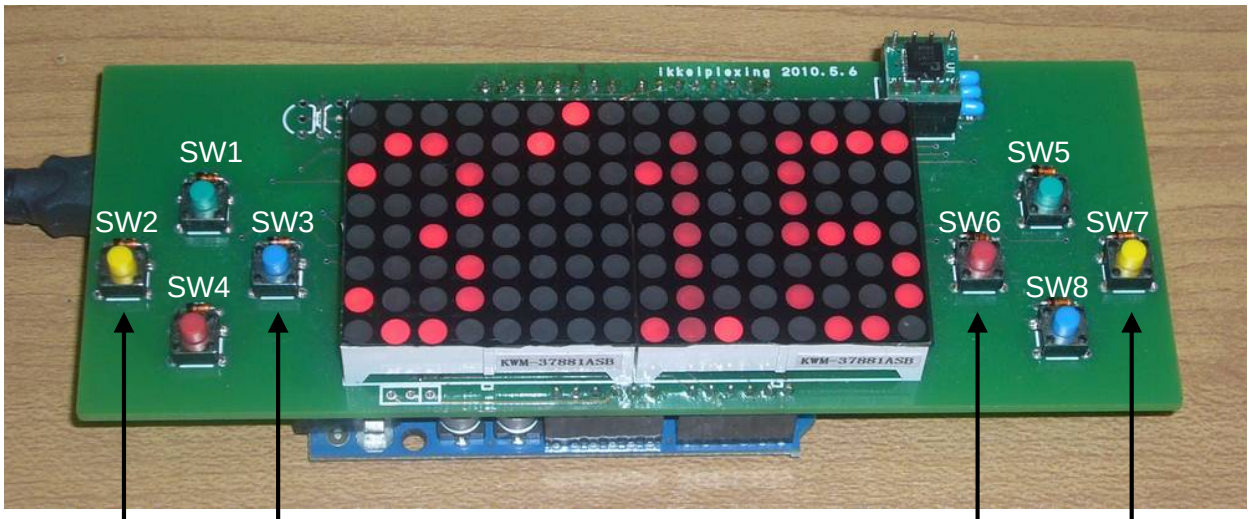

```
void pause_state(void){
  if ((event & SW_START) != 0){
    state = count_down;
  } else if ((event & SW_CLEAR) != 0){
    set_minute = 0;
    set_tensec = 0;
    state = set_time;
  }
}
```




```
void alarm_state(void){
  alarm_count--;
  if (alarm_count == 0){
    alarm_count = ALARM_PERIOD;               } アラーム表示
    alarm_flash ^= 1;
    DisplayAlarm(alarm_flash);
  }
  if (((event & SW_MINUTE) != 0)||((event & SW_TENSEC) != 0)||
      ((event & SW_START) != 0)||((event & SW_CLEAR) != 0)){
    iLedSign::Clear();
    state = set_time;
  }
}
```

キッチンタイマ

数字を表示して9分50秒までのタイマになります。
ラーメンを作るときに使えます。

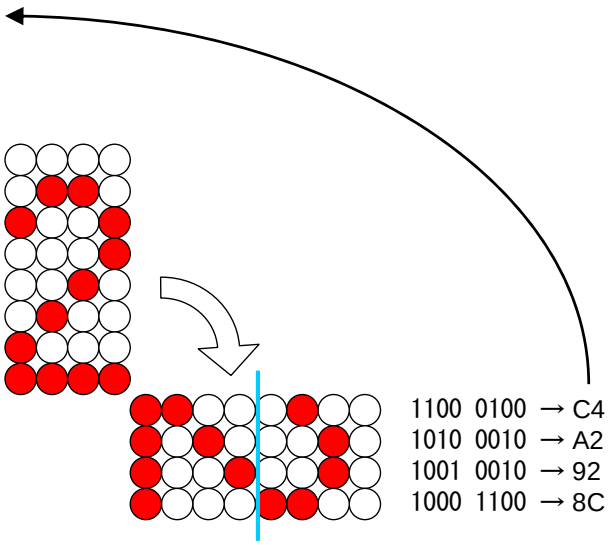


分　　10秒　　スタート　クリア
ストップ

数字のパターンの作成

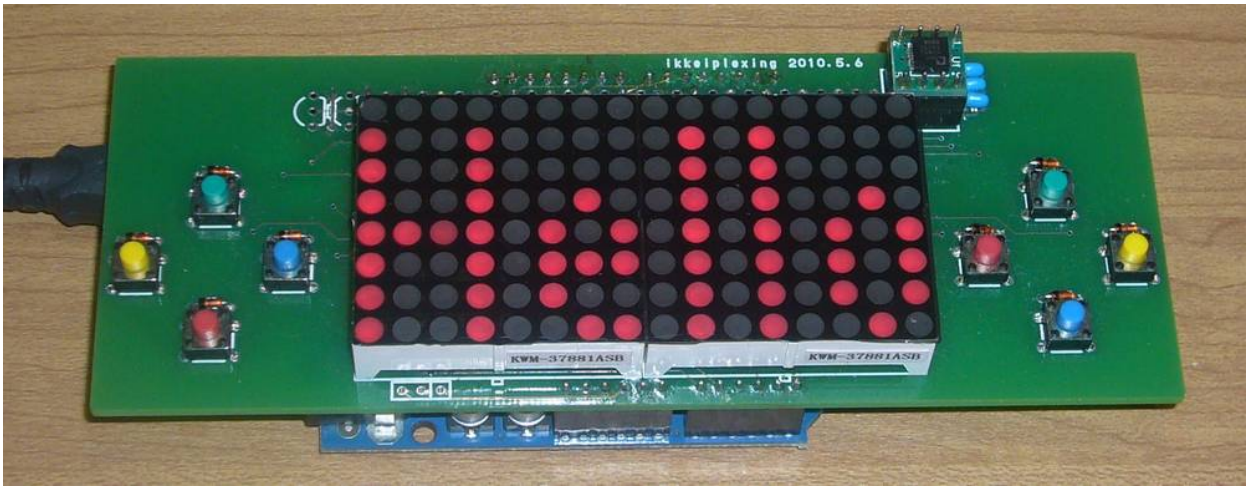
```
const byte FIG[10][4] = {
{ 0x7C, 0x82, 0x82, 0x7C},
{ 0x00, 0x84, 0xFE, 0x80},
{ 0xC4, 0xA2, 0x92, 0x8C},
{ 0x44, 0x82, 0x92, 0x6C},
{ 0x38, 0x24, 0xFE, 0x20},
{ 0x5E, 0x92, 0x92, 0x62},
{ 0x7C, 0x92, 0x92, 0x64},
{ 0x06, 0xC2, 0x32, 0x0E},
{ 0x6C, 0x92, 0x92, 0x6C},
{ 0x4C, 0x92, 0x92, 0x7C}
};
```

ビットパターンから16進数にして、
配列に入れます。
10個なので、手作業で作りました。



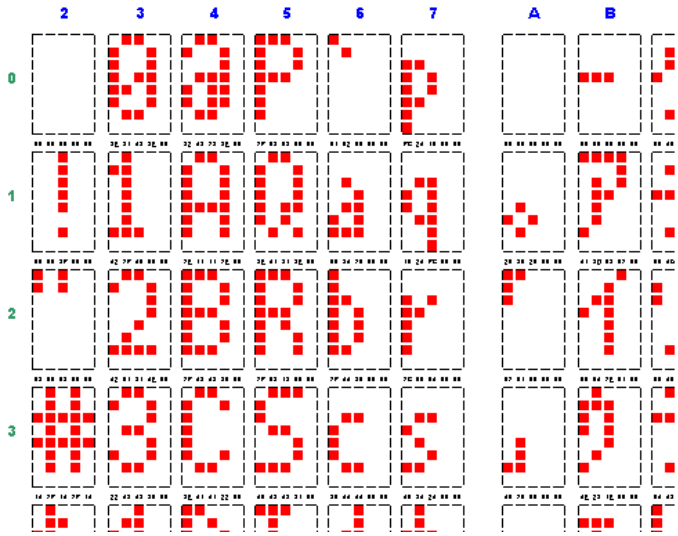
(5) 電光掲示板

設定した文字列をスクロールしながら繰り返し表示します。



英字やカタカナの文字パターンからデータを作る

1. Excelでパターンを作成
点灯:1　消灯:0　を入れる。



2. コードをコピーし、iFont.cppの
配列に貼り付ける。

Flash ROM上に配置する。

```
const uint16_t BitMap[][9] PROGMEM = {
```

文字列の表示

```
void loop()
{
  //const unsigned char test[]="HELLO WORLD!   ¥xCA¥xDB¥xB0¥xDC¥xB0¥xD9¥xDE¥xC4!   ";
  const unsigned char test[]="Hello world!   ¥xCA¥xDB¥xB0¥xDC¥xB0¥xD9¥xC4¥xDE!   ";
  //const char test[]="HELLO WORLD!   ";

  int16_t x=0;
  for(int16_t j=15;j>-150;j--) {
    x=j;
    iLedSign::Clear();
    for(int i=0;i<24;i++) {
      x+= iFont::Draw(test[i],x,0);
      //x+= iFont::Draw90(test[i],x,0);
      if (x>=15) break;
    }
    delay(80);
  }
  //delay(1000);
}
```

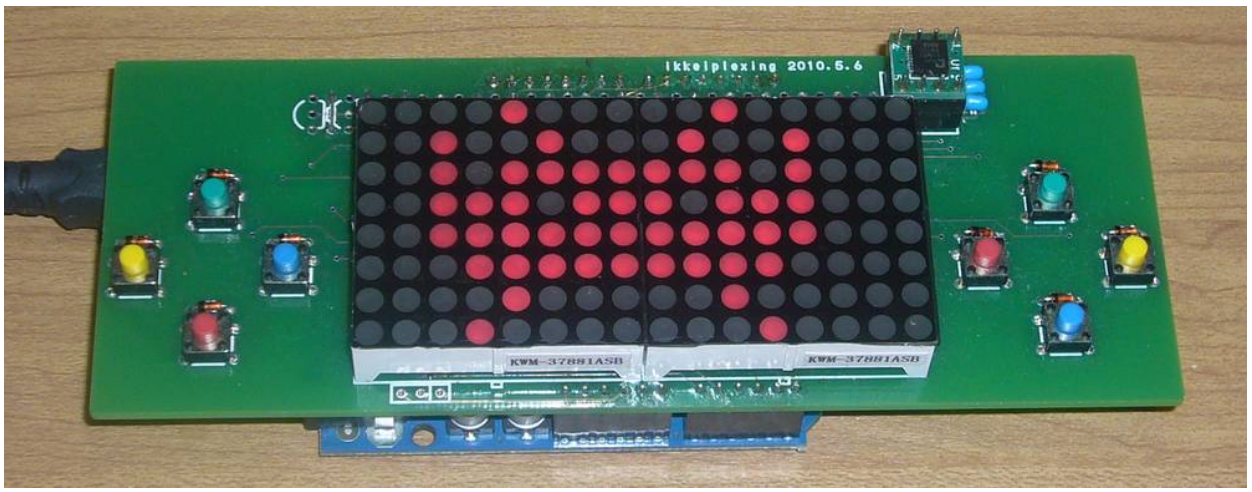
表示させる文字列
カタカナは16進数を使用する。

文字列全体のドットから
表示位置をずらしていく
方法なので、文字列の
長さなどに制限がある。
あまり良くないやり方。
あくまでテスト用。

縦スクロールの場合

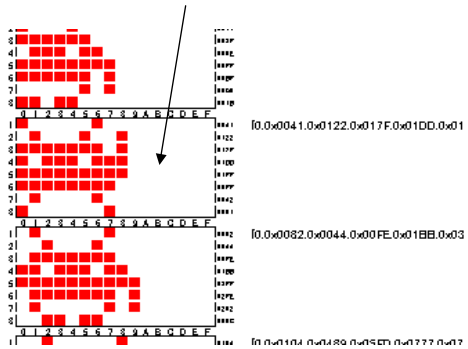
(6) パターン表示

設定したドットパターンを順々に表示させます。

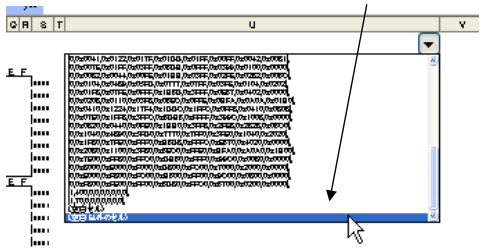


ドットパターンの作成

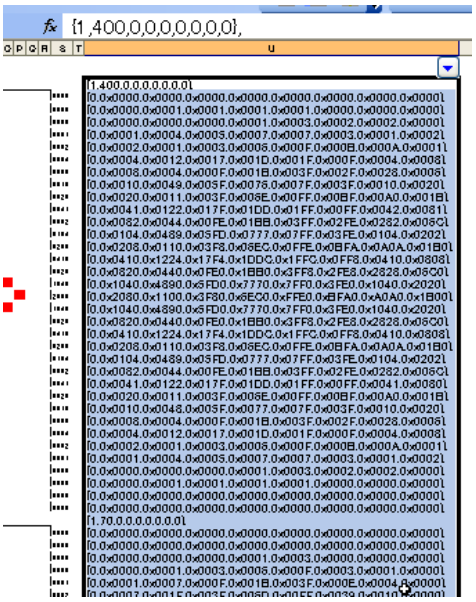
1. Excelでパターンを作成
点灯：1　消灯：0　を入れる。



2.「空白以外のセル」を選ぶ



3. コードをコピーし、スケッチに貼り付ける。



4. 元に戻すには「すべて」を選ぶ

1フレームの時間の設定

最初の数字でデータの種別をしている。
0：1フレーム分のビットパターン（16ビット×8）
1：1フレーム表示の時間（ms単位）
2：データの終わり

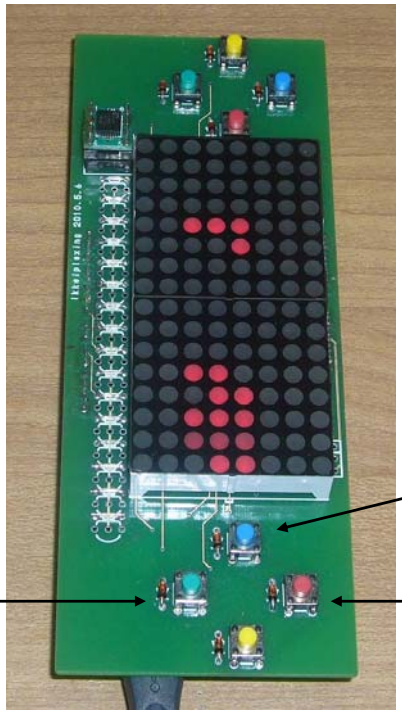
Flash ROM上に配置する。

```
uint16_t BitMap[][9] PROGMEM = {
// first data 0:pattern data  1:delay time  2:terminator
{1, 400, 0, 0, 0, 0, 0, 0, 0},
{0, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000},
{0, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0000, 0x0000, 0x0000},
{0, 0x0000, 0x0000, 0x0000, 0x0001, 0x0003, 0x0002, 0x0002, 0x0000},
{0, 0x0001, 0x0004, 0x0005, 0x0007, 0x0007, 0x0003, 0x0001, 0x0002},
{0, 0x0002, 0x0001, 0x0003, 0x0006, 0x000F, 0x000B, 0x000A, 0x0001},
                                    ⋮
{0, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000},
{0, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000},

{2, 0, 0, 0, 0, 0, 0, 0, 0},
};
```

（7）ゲーム

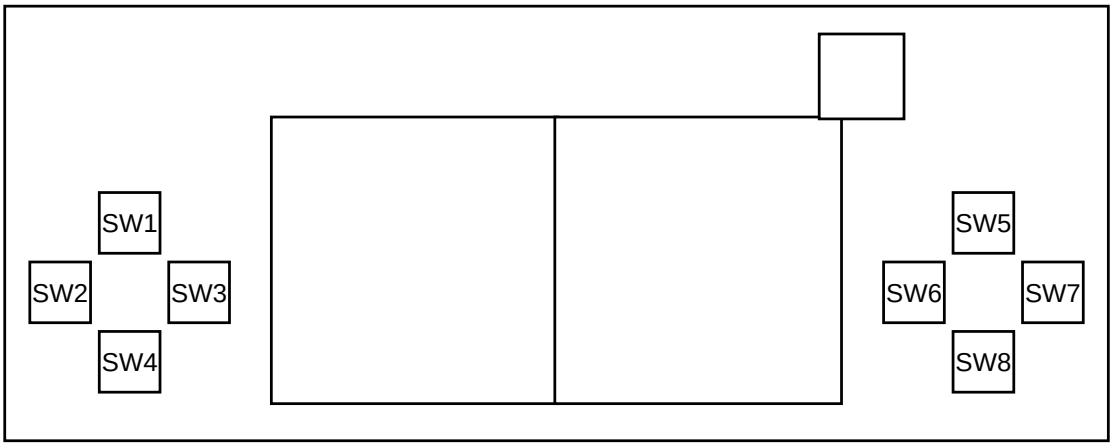
ライブラリを活用して、既存のゲームを動かしてみます。

tetris



tetrisはarduino-0022で
エラーが出ます。
arduino-0021を
使用して下さい。

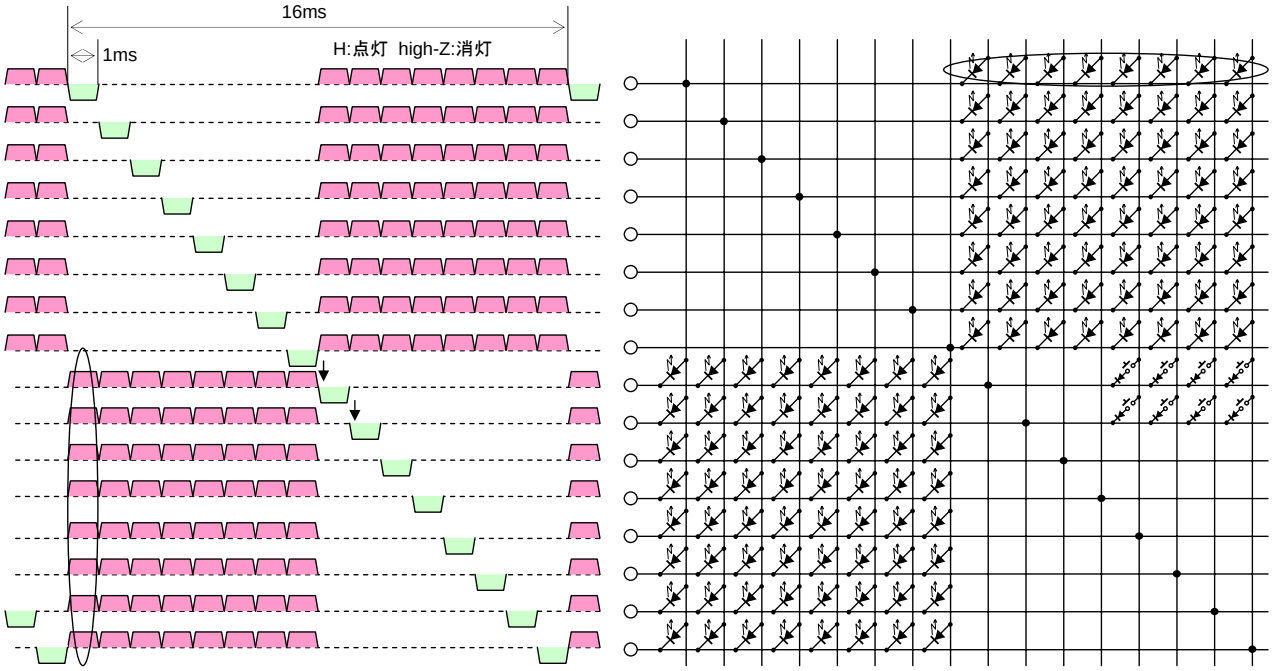
あとはあなたのアイデア次第！！



3.　ライブラリについて

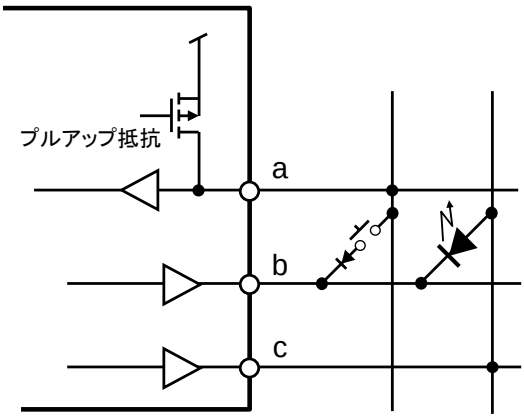
・LEDの点灯　タイミングチャート
・スイッチの取り込みタイミング
・直結でも問題無いの？
・PWMで輝度補正　原理図
・PWMで輝度補正　実際のタイミング
・スキャン部のコード
・ドットをセットする部分とスイッチの取得

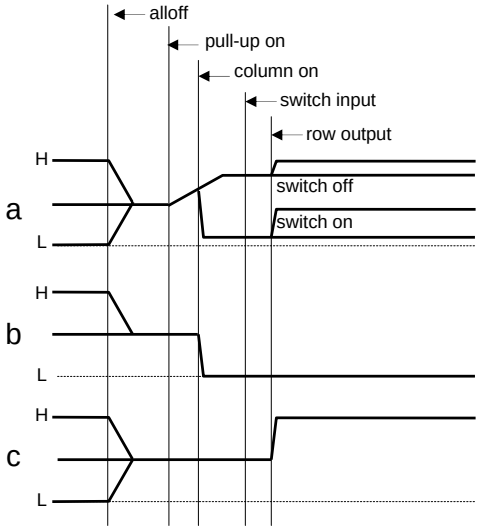
LEDの点灯　タイミングチャート



スイッチの取り込みタイミング

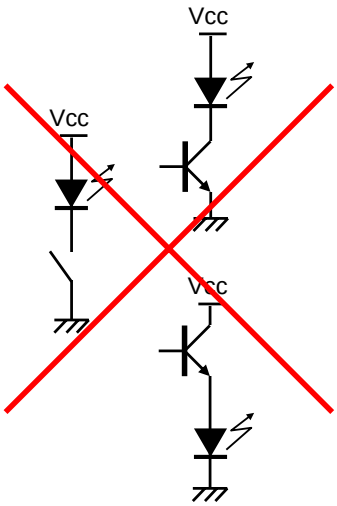
・スイッチの取り込みタイミングは列切り替えの狭間の
LEDが消灯している間に行うので、表示のスキャンには影響しない。



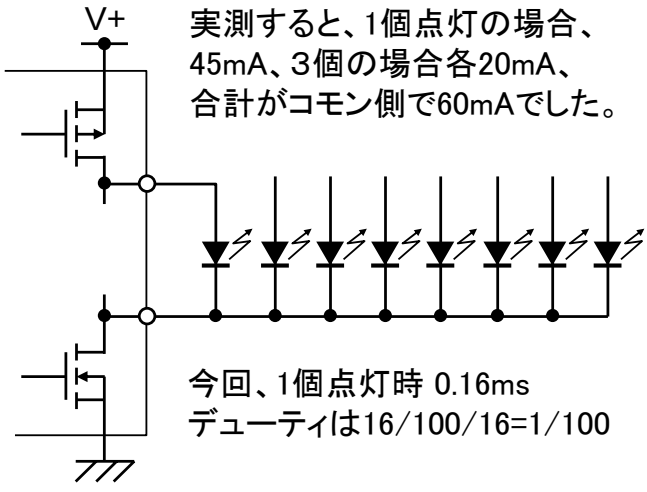


直結でも問題無いの？

もちろんこれはダメ！



出力ポートのFETが電流制限をするので大丈夫



実測すると、1個点灯の場合、
45mA、3個の場合各20mA、
合計がコモン側で60mAでした。

今回、1個点灯時 0.16ms
デューティは16/100/16=1/100

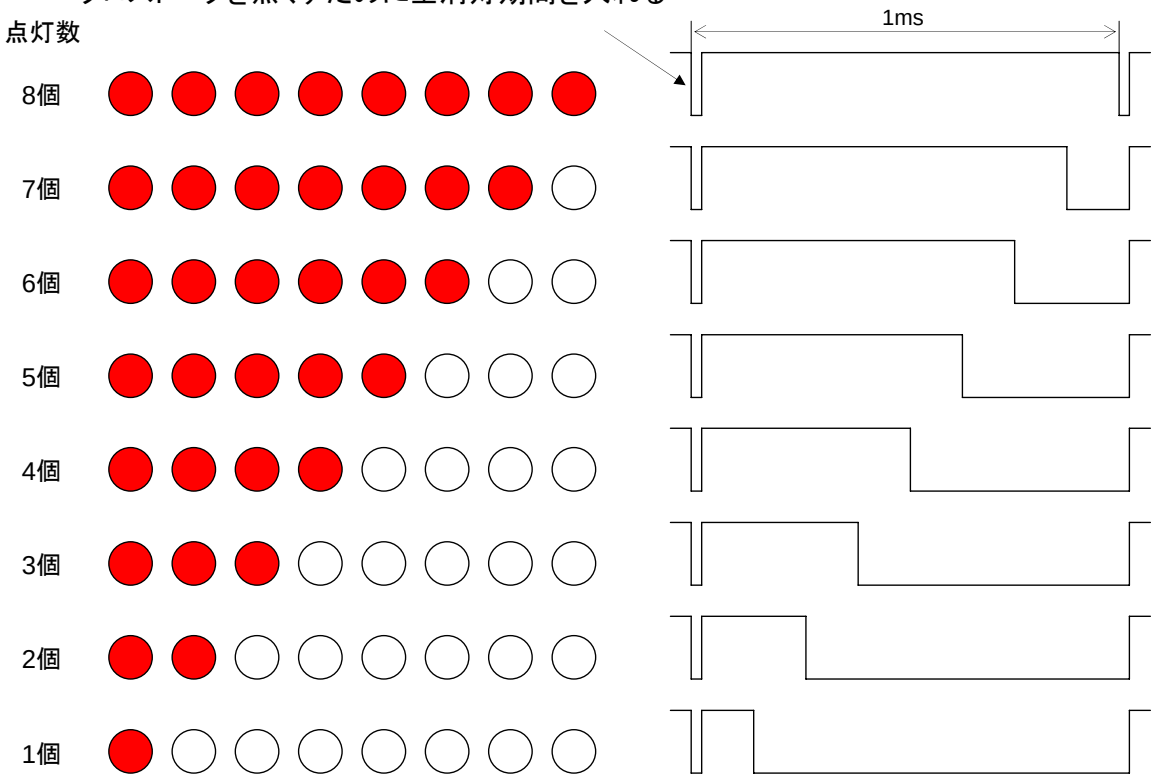
KWM-37881ASB（TOM-1588BH同等品）

Absolute Maximum Ratings 絶対最大定格

Peak Forward Current (1/10 Duty Cycle, 0.1ms Pulse Width)	100	mA
Continuous Forward Current	40	mA

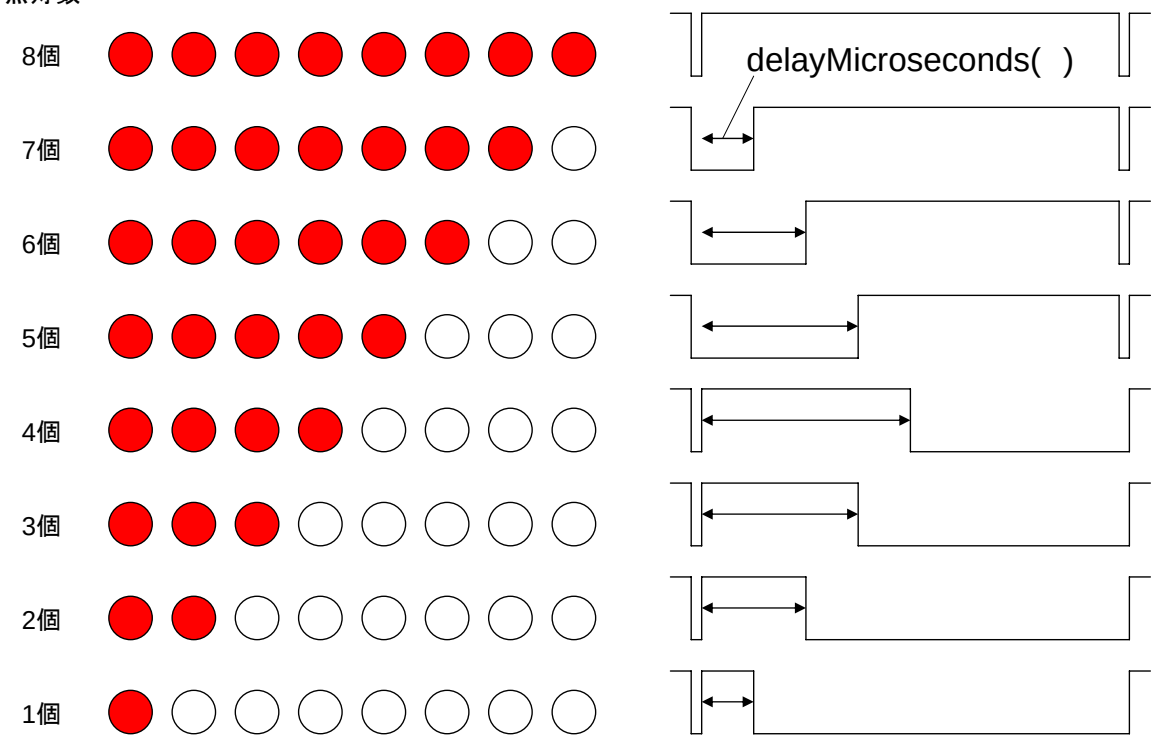
PWMで輝度補正　原理図

クロストークを無くすために全消灯期間を入れる



PWMで輝度補正　実際のタイミング

後半に他の処理が出来るようにdelayは前半のみ行う



スキャン部のコード　1

・Charlieplexing.cpp

```
void scanLED(){

    alloff(); // all output high-Z (input)

    if ((scan==8)||(scan==9)){ // switch common lines
        PORTD |= 0x28;   // pull up register of SW4 & SW2 or SW8 & SW6
        PORTB |= 0x02;   // pull up register of SW3 or SW7
        PORTC |= 0x01;   // pull up register of SW1 or SW5
    }

    uint8_t n = ledMap[ scan ]; // scan column output LOW for cathode
    if (n < 8) {
        DDRD  = _BV(n);
    } else if (n < 14 ){
        DDRB  = _BV(n-8);
    } else {
        DDRC  = _BV(n-14);
    }

    //### count ON LED numbers
    uint8_t on_count = 0;
    for (int j=0; j<8; j++){
        if ((displayBuffer[scan*3] & 1<<j ) != 0){
            on_count++;
        }
    }
    for (int j=0; j<6; j++){
        if ((displayBuffer[scan*3+1] & 1<<j ) != 0){
            on_count++;
        }
        if ((displayBuffer[scan*3+2] & 1<<j ) != 0){
            on_count++;
        }
    }
```

クロストークを防ぐために、全出力オフにする。

スイッチ入力のためのプルアップ抵抗オン。

スキャン順に応じた列出力を端子の配列から取得、
その番号に対応するポートを出力ポートにする。
この時の行データはあり得ないのでデータは0
従って、LOW出力となる。

スキャン順に応じた行出力データ
を全部チェックしてその時の
LEDの点灯数をカウントする。
入力のためのディレイを兼ねる。

スキャン部のコード　2

・Charlieplexing.cpp

```
    if (scan==8){ // get SW1,SW2,SW3,SW4
        sw1234 = (digitalRead(sw1)*8 + digitalRead(sw2)*4 + digitalRead(sw3)*2 + digitalRead(sw4)) ^ 0x0F;
    } else if (scan==9){ // get SW5,SW6,SW7,SW8
        previous_sw = current_sw;
        current_sw = (sw1234 << 4) | ((digitalRead(sw5)*8 + digitalRead(sw6)*4 + digitalRead(sw7)*2 + digitalRead(sw8)) ^ 0x0F);
        if (previous_sw == current_sw){
            swdata = current_sw;
        }
    } // swdata: SW1,SW2,SW3,SW4,SW5,SW6,SW7,SW8

    //### equalization pre-delay
    if ( (on_count >= MID_LED) && (on_count < MAX_LED) ){
        delayMicroseconds( 1000 - dd[on_count] );
    }

    // output row data
    DDRD  |= displayBuffer[scan*3];
    PORTD  = displayBuffer[scan*3];
    DDRB  |= displayBuffer[scan*3+1];
    PORTB  = displayBuffer[scan*3+1];
    DDRC  |= displayBuffer[scan*3+2];
    PORTC  = displayBuffer[scan*3+2];

    //### equalization post-delay
    if ( on_count < MID_LED ){
        delayMicroseconds( dd[on_count] );
        alloff();
    }
```

入力したスイッチデータをビットに割り当てて
0x0FとEX-ORして反転する。
スイッチ入力は負論理（スイッチオン:LOW）のため

前回取り込んだスイッチデータと同じだったら
swdataに結果を入れる。（チャタリング除去）

輝度補正のための前半ディレイ
この間全消灯のまま

行出力:
データが1のビットを出力ポートにし、
さらに1を出力してLEDを点灯させる。

輝度補正のための後半ディレイ
その後は全出力オフにして消灯する。

スキャン部のコード　3

・Charlieplexing.cpp

```
    period = 0;
    msec++;
    if (msec >= 10){
        msec = 0;
        period |= TENMS;
    }

    scan++;
    if (scan >= 16) {
        scan = 0;
        period |= FRAME;

        // If the page should be flipped, do it here.
        if (videoFlipPage && displayMode == DOUBLE_BUFFER)
        {
            // TODO: is this an atomic operation?
            videoFlipPage = false;

            uint8_t* temp = displayBuffer;
            displayBuffer = workBuffer;
            workBuffer = temp;
        }
    }
}
```

1msのスキャンを10回カウントして
10ms毎のフラグを立てる

スキャンが1巡したら
フレームのフラグを立てる

ダブルバッファの場合、
描画バッファを入れ替える

ドットをセットする部分とスイッチの取得

```
void iLedSign::Set(uint8_t x, uint8_t y, uint8_t c)
{
    uint8_t pin_high;
    if ( x >= 8 ){
        pin_high = ledMap[ y ];
    } else {
        pin_high = ledMap[ y + 8 ];
    }
    if (c == 1) {
        if ( pin_high < 8 ){
            workBuffer[ x*3     ] |= _BV( pin_high );
        }else if ( pin_high < 14 ){
            workBuffer[ x*3 + 1 ] |= _BV( pin_high - 8 );
        }else{
            workBuffer[ x*3 + 2 ] |= _BV( pin_high - 14 );
        }
    } else {
        if ( pin_high < 8 ){
            workBuffer[ x*3     ] &= ~_BV( pin_high );
        }else if ( pin_high < 14 ){
            workBuffer[ x*3 + 1 ] &= ~_BV( pin_high - 8 );
        }else{
            workBuffer[ x*3 + 2 ] &= ~_BV( pin_high - 14 );
        }
    }
}
```

y 座標に応じたピン番号を取得する
x 座標が8以上の時は右側のLED
x 座標が8未満の時は左側のLEDになる

点灯させる場合
y 座標のピン番号のビット位置を1にする

消灯させる場合
y 座標のピン番号のビット位置を0にする

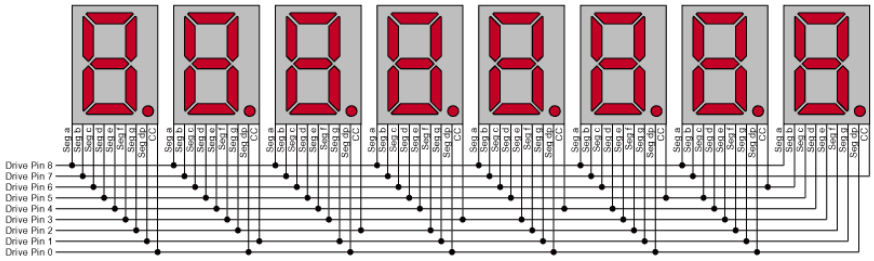
```
uint8_t iLedSign::GetSW(uint8_t edge)
{
    static uint8_t previous_swdata;
    uint8_t flag = swdata;
    if (edge != 0){
        flag &= ~previous_swdata;
        previous_swdata = swdata;
    }
    return (flag);
}
```

レベル検出の場合、swdataそのものを返す

エッジ検出の場合
前回のデータが0で、今回が1のビットを返す

4.　Charlieplexingとは

Charlieplexing とは、LEDの結線を工夫して、駆動する線を節約する方法です。
例えば、7セグメントLEDを8個使用する場合、DPを含めて9本で点灯できます。
このようになります。



MAXIM社　AN1880

オリジナル

Charlieplexingのオリジナルは MAXIM社のこの資料(AN1880)です。
”Charlieplexing - Reduced Pin-Count LED Display Multiplexing”
http://japan.maxim-ic.com/app-notes/index.mvp/id/1880

MAXIM

Maxim > App Notes > AUTOMOTIVE　DISPLAY DRIVERS

Keywords: charlie, charlieplex, charlieplexing, led driver, display drivers, MAX6950, MAX6951, MAX6954, MAX6955, MAX6958, MAX6959, LED drivers　Feb 10, 2003

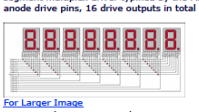
APPLICATION NOTE 1880

Charlieplexing - Reduced Pin-Count LED Display Multiplexing

Abstract: This application note discusses "Charlieplexing" -- a pin-count reducing multiplex technique used by the MAX6950, MAX6951, MAX6954, MAX6955, MAX6958, and MAX6959 LED display drivers.

This application note discusses how to "Charlieplex" a display driver circuit to reduce pin-count. This unusual multiplex technique is used by the MAX6950, MAX6951, MAX6954, MAX6955, MAX6958, and MAX6959 LED display drivers. Charlieplexing reduces driver pin-count by using some pins alternately as cathode and anode drivers. This differs from the standard LED multiplex connection, which uses separate driver pins for anodes and cathodes.

The standard connection for multiplexing common cathode (CC) LED digits uses a separate pin for each digit's CC connection, while the anode segment connections are communed across all the digits. Similarly, the standard connection for multiplexing common anode (CA) LED digits uses a separate pin for each digit's CA connection, while the cathode segment connections are communed across all the digits. The number of connections required can be calculated as being 1 for every digit used, plus 1 for every segment within a digit. Therefore an 8-digit, 8-segment multiplex driver typified by the MAX7219 and MAX7221 CC drivers uses 8 cathode drive pins and 8 anode drive pins, 16 drive outputs in total (**Figure 1**).



For Larger Image

Figure 1. The MAX7219 and MAX7221 use standard connections - 16 pins to drive 8 digits.

A more pin-efficient scheme relies on the fact that during the multiplex operation, only one CC or CA digit drive output is actually in use. The other digit drives are high-impedance, ensuring that no drive current flows into these undriven digits. By making the LED drive pins alternate duty between driving digits and segments, n drive pins can be used to drive n digits each with n-1 segments. Charlie Allen originally championed this technique internally at Maxim, and so the shorthand name "Charlieplexing" came into use to distinguish reduced pin count multiplexing from the traditional method. The first Maxim product to use Charlieplexing is the Maxim MAX6951 LED driver, which drives 8 numeric digits with only 9 pins (**Figure 2**).

For Larger Image

Figure 2. The MAX6951 uses Charlieplexing - only 9 pins to drive 8 digits.

To understand how Charlieplexing works, first examine the pin connections shown in Table 1. There is one row per multiplexed digit, and each row contains 9 columns. For each digit, one column corresponds to the CC digit drive, and the remaining 8 columns correspond to the 8 anode (segment) drives.

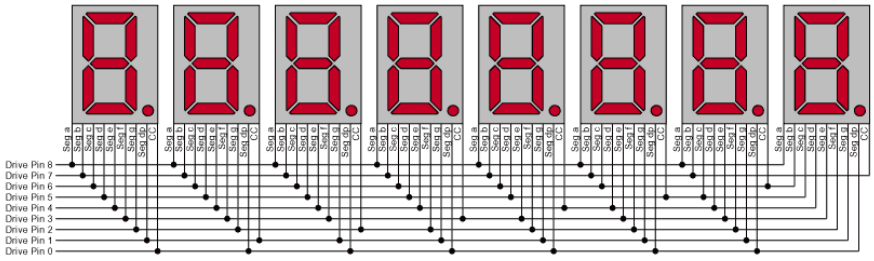
Table 1. The MAX6951 Driver to LED Display Connections

2003年2月10日

MAXIM社　AN1880

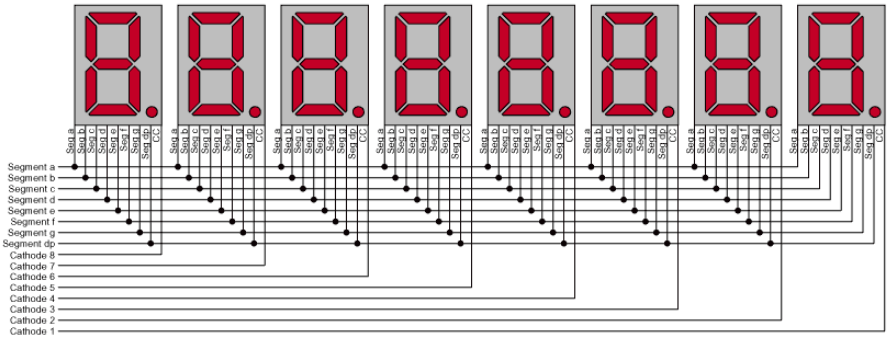
7セグメントLEDに適用した場合

ここには、例として7セグメントLEDに適用した回路図があります。9本の線で8桁。
http://www.maxim-ic.com/images/appnotes/1880/DI217Fig02.gif



MAXIM社　AN1880

こちらは従来方式。8桁の場合は8＋8＝16本必要。
見比べると、上図はコモンの線をセグメントの線と兼用していることが分かります。



MAXIM社　AN1880

セグメントとデジットの対応表

さらに、セグメントとデジットの対応表があります。
表示桁に応じて、セグメントの線を交代していることが分かります。

Original (The MAX6951 Driver to LED Display Connections)　MAXIM社　AN1880

	DIG0/SEG0	DIG1/SEG1	DIG2/SEG2	DIG3/SEG3	DIG4/SEG4	DIG5/SEG5	DIG6/SEG6	DIG7/SEG7	SEG8
	Pin 6	Pin 5	Pin 4	Pin 3	Pin 14	Pin 13	Pin 12	Pin 11	Pin 10
LED Digit 0	**CC0**	SEG dp	SEG g	SEG f	SEG e	SEG d	SEG c	SEG b	SEG a
LED Digit 1	SEG dp	**CC1**	SEG g	SEG f	SEG e	SEG d	SEG c	SEG b	SEG a
LED Digit 2	SEG dp	SEG g	**CC2**	SEG f	SEG e	SEG d	SEG c	SEG b	SEG a
LED Digit 3	SEG dp	SEG g	SEG f	**CC3**	SEG e	SEG d	SEG c	SEG b	SEG a
LED Digit 4	SEG dp	SEG g	SEG f	SEG e	**CC4**	SEG d	SEG c	SEG b	SEG a
LED Digit 5*	SEG dp	SEG g	SEG f	SEG e	SEG d	**CC5**	SEG c	SEG b	SEG a
LED Digit 6*	SEG dp	SEG g	SEG f	SEG e	SEG d	SEG c	**CC6**	SEG b	SEG a
LED Digit 7*	SEG dp	SEG g	SEG f	SEG e	SEG d	SEG c	SEG b	**CC7**	SEG a

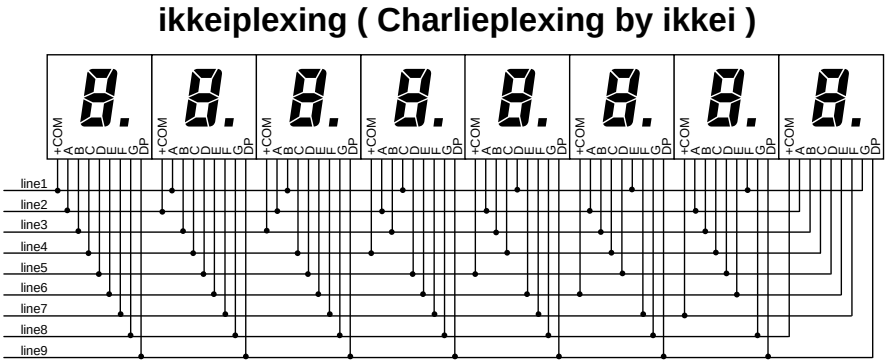
でも、このままではDPを使わない場合も9本必要になります。
そこで、私はセグメントを入れ替え、DPを使わないときは8本で表示出来る
ようにし、さらに、なるべくセグメントと線の関係を変えないようにしてみました。

Minimum Difference Connections (Close to Conventional Connections)

	DIG0/SEG0	DIG1/SEG1	DIG2/SEG2	DIG3/SEG3	DIG4/SEG4	DIG5/SEG5	DIG6/SEG6	DIG7/SEG7	SEG8
LED Digit 0	**AC0**	SEG a	SEG b	SEG c	SEG d	SEG e	SEG f	SEG g	SEG dp
LED Digit 1	SEG a	**AC1**	SEG b	SEG c	SEG d	SEG e	SEG f	SEG g	SEG dp
LED Digit 2	SEG b	SEG a	**AC2**	SEG c	SEG d	SEG e	SEG f	SEG g	SEG dp
LED Digit 3	SEG c	SEG a	SEG b	**AC3**	SEG d	SEG e	SEG f	SEG g	SEG dp
LED Digit 4	SEG d	SEG a	SEG b	SEG c	**AC4**	SEG e	SEG f	SEG g	SEG dp
LED Digit 5	SEG e	SEG a	SEG b	SEG c	SEG d	**AC5**	SEG f	SEG g	SEG dp
LED Digit 6	SEG f	SEG a	SEG b	SEG c	SEG d	SEG e	**AC6**	SEG g	SEG dp
LED Digit 7	SEG g	SEG a	SEG b	SEG c	SEG d	SEG e	SEG f	**AC7**	SEG dp

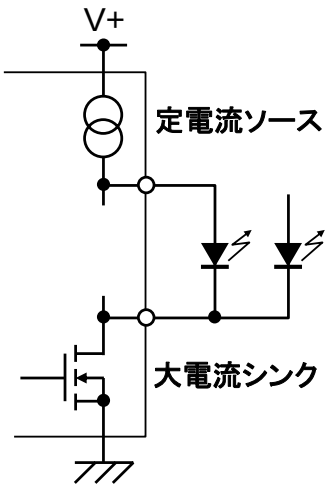
ikkeiplexing

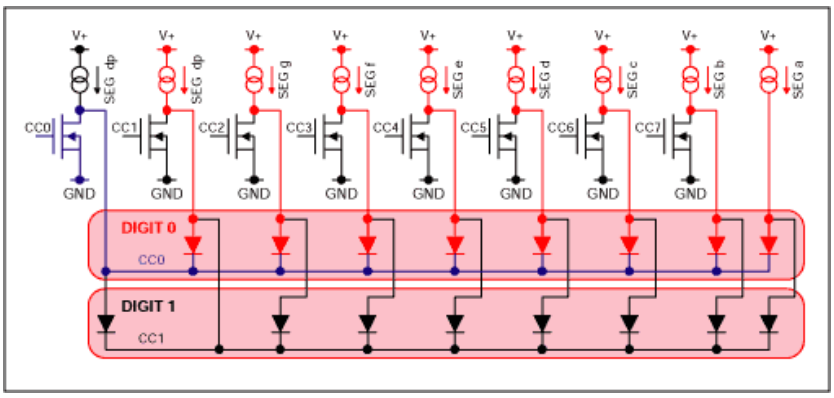
その回路図がこれです。



オリジナルデバイス

さらに、オリジナルでは下記のような専用デバイスを使用しているので、
LEDを直結することが出来ます。



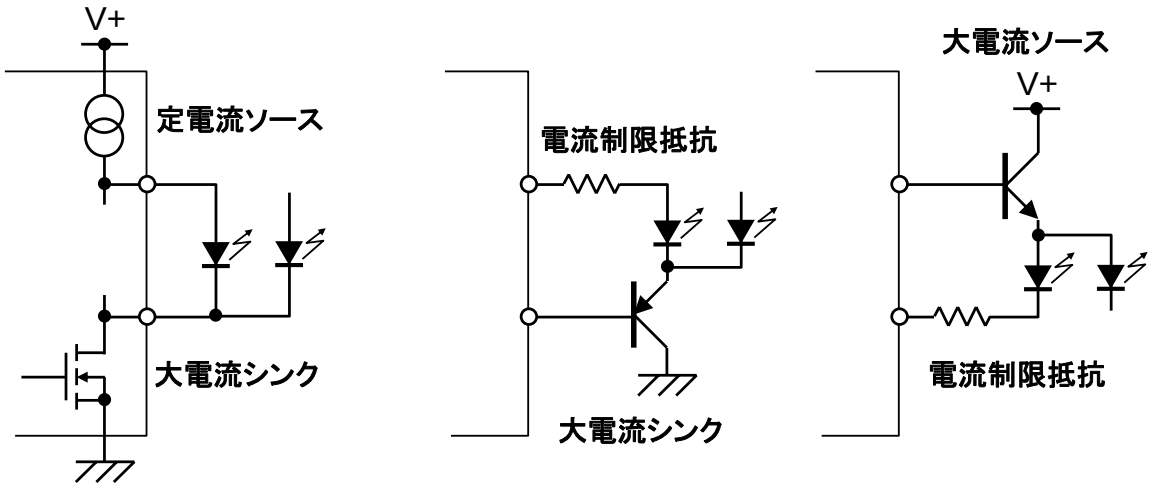


MAXIM社　AN1880

カソードコモン

電流ブーストトランジスタ

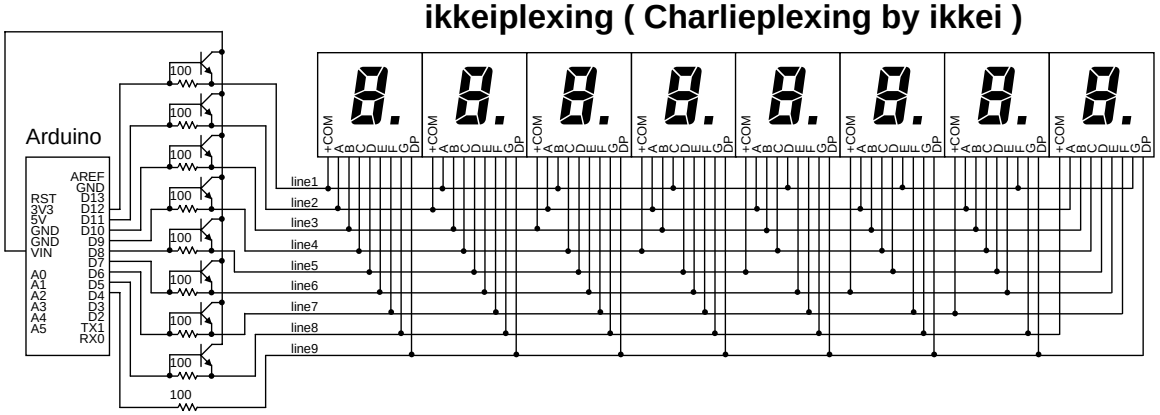
でも、一般のマイコンで駆動する場合は、そうはいきません。
そこで私は、一般のマイコンの出力につなぐために電流制限抵抗と、
コモン線をドライブするために電流をブーストするトランジスタを追加しました。
このトランジスタはエミッタフォロアなので、ベース抵抗は要りません。



カソードコモン　　カソードコモン　　アノードコモン

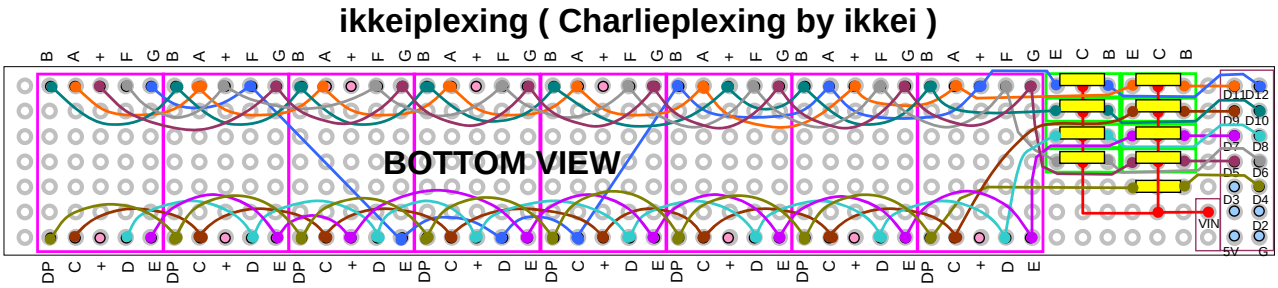
ikkeiplexing

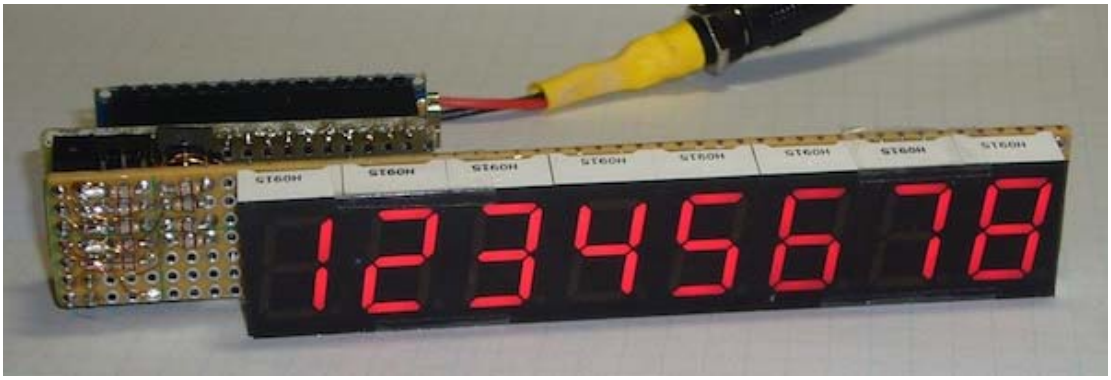
これが全体の回路です。
アノードコモンの場合だとVccより高い電圧をトランジスタのコレクタに
掛けることが出来ます。



実体配線図と実物

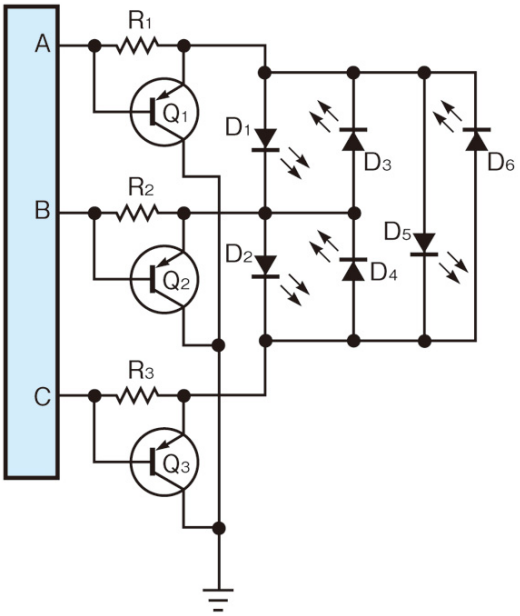
実体配線図と実物です。





EDNの記事

このブースト回路は、調べてみるとEDNの記事に有るものと同じでした。
「少ないI/Oで多数のLEDを制御、同時点灯も可能」
http://ednjapan.rbi-j.com/issue/2009/3/22/2983
こちらはカソードコモンの場合です。



EDN

[issued: 2009年3月号]

Guillermo Jaquenod
アルゼンチン

Wikipedia

それでは、なぜこのような回路が可能になるのかを原理から考察します。
これはWikipediaにも書いてあります。
http://en.wikipedia.org/wiki/Charlieplexing
ここには1995年始めにMAXIM社のCharlie Allen さんが提唱したとあります。

WIKIPEDIA
The Free Encyclopedia

navigation
- Main page
- Contents
- Featured content
- Current events
- Random article

search
Go Search

interaction
- About Wikipedia
- Community portal
- Recent changes
- Contact Wikipedia
- Donate to Wikipedia
- Help

toolbox

Try Beta Log in / create account

article | discussion | edit this page | history

Charlieplexing

From Wikipedia, the free encyclopedia

Charlieplexing is a technique proposed in early 1995 by Charlie Allen at Maxim Integrated Products for driving a multiplexed display in which relatively few I/O pins on a microcontroller are used to drive an array of LEDs. The method utilizes the tri-state logic capabilities of microcontrollers in order to gain efficiency over traditional multiplexing. Although it is more efficient in its use of IO, there are issues that cause it to be more complicated to design and render it impractical for larger displays. These issues include duty cycle, current requirements and the forward voltages of the LEDs.

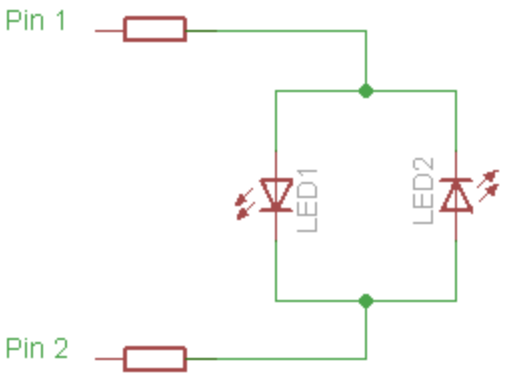
Contents [hide]

1 Traditional multiplexing
2 Charlieplexing
  2.1 Complementary drive
  2.2 Expanding: tri-state logic
3 Problems with charlieplexing
  3.1 Refresh rate
  3.2 Peak current
  3.3 Requirement for tristate
  3.4 Complexity
  3.5 Forward voltage
  3.6 LED failure
4 Input data multiplexing

A Charlieplexed digital clock which controls 90 LEDs with 10 pins of a PIC16C54 microcontroller. Click to enlarge picture.[1]

原理

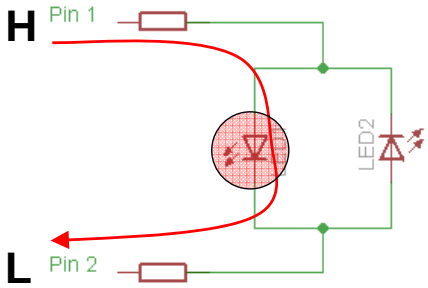
まず、LEDが2個の場合を考え、このように配線します。

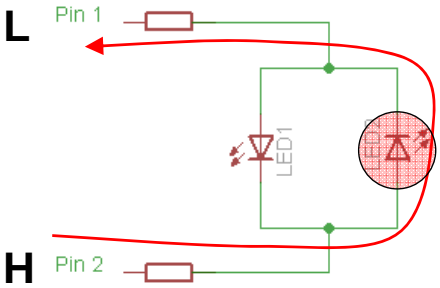


Wikipedia

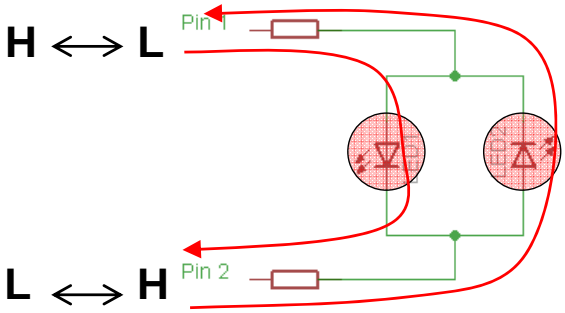
2個のLEDの点灯方法

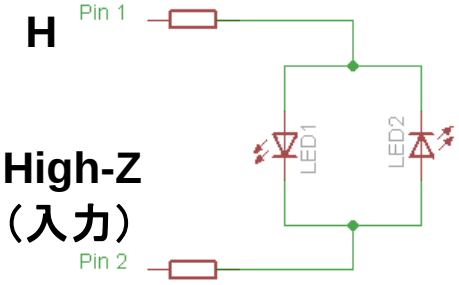
それぞれのLEDを点灯させるには、出力をH，LまたはL，Hにします。





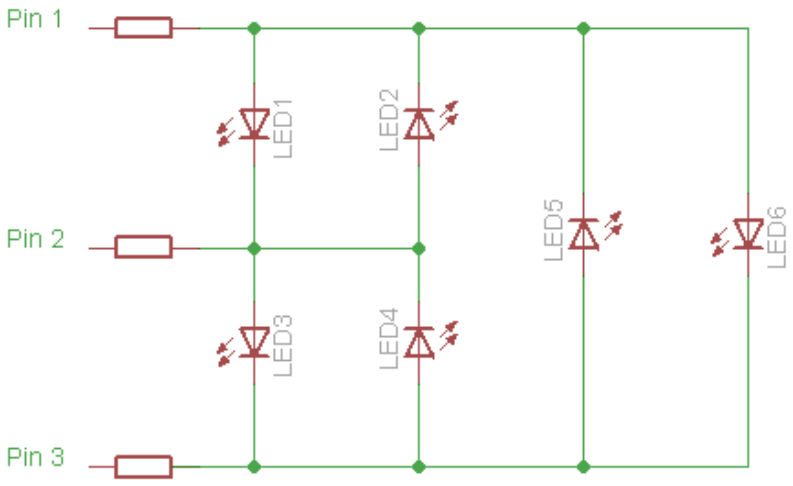
両方点灯させるには高速に切り替えます。片方をHigh-Zにすると両方消灯になります。





3本線の場合6個のLED

2本線でLED2個だとメリットが無いのですが、
3本になるとLEDは3個ではなく、6個制御できます。
つまりn本の線だとn×(n-1)個のLEDを制御できる事になります。
従来方式だとnが偶数の場合$n^2/4$ですからnが大きいほど効果があることが分かります。

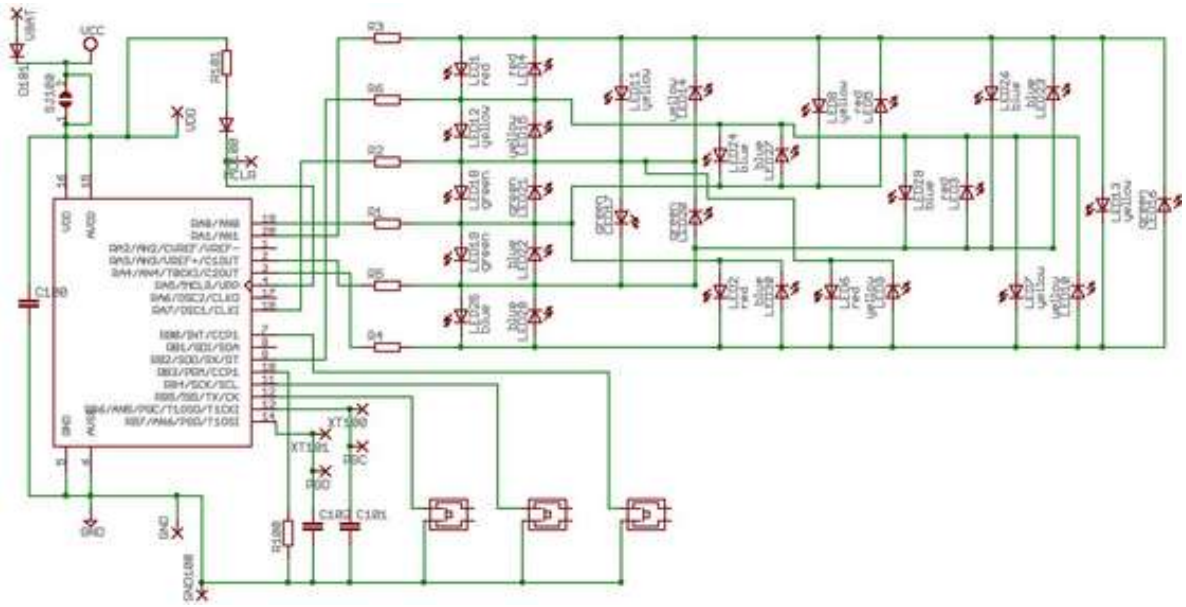


Wikipedia

n	n×(n-1)	$n^2/4$
4	12	4
6	30	9
8	56	16
10	90	25
12	121	36
14	182	49
16	240	64

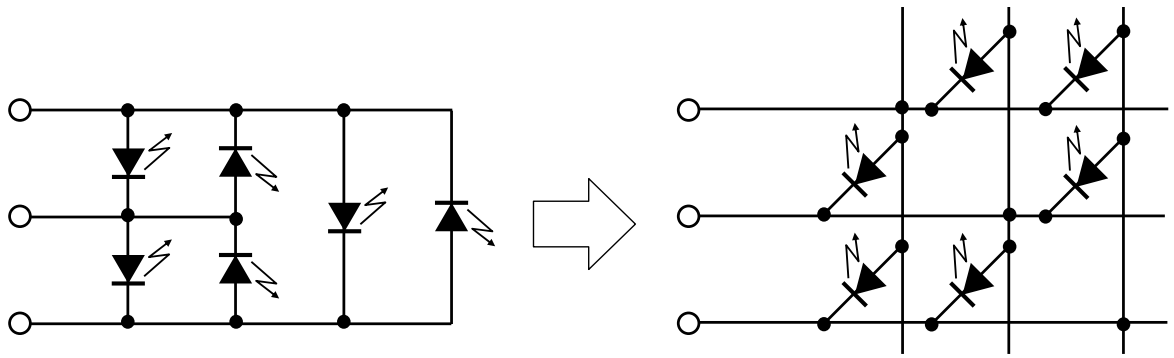
6本線では30個のLED

nが6だと30個ものLEDを制御できます。総当たり方式で書くと下図になります。
http://www.instructables.com/id/Charlieplexing-LEDs--The-theory/step4/Finallya-Charlieplex-matrix/
しかし、このような書き方でLEDの数を増やそうとすると、書くのが大変です。



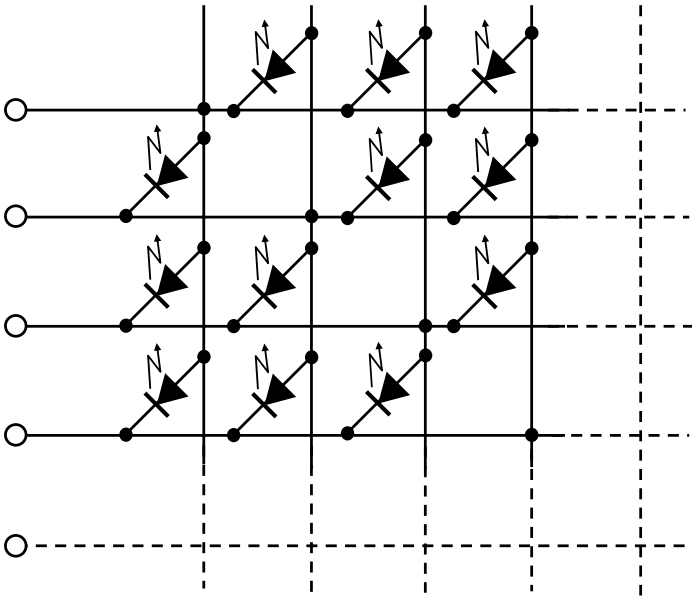
3本線のマトリクス表記

でも、
書き方を変えて、3本の場合は右の様に書けば、拡張するのは簡単です。



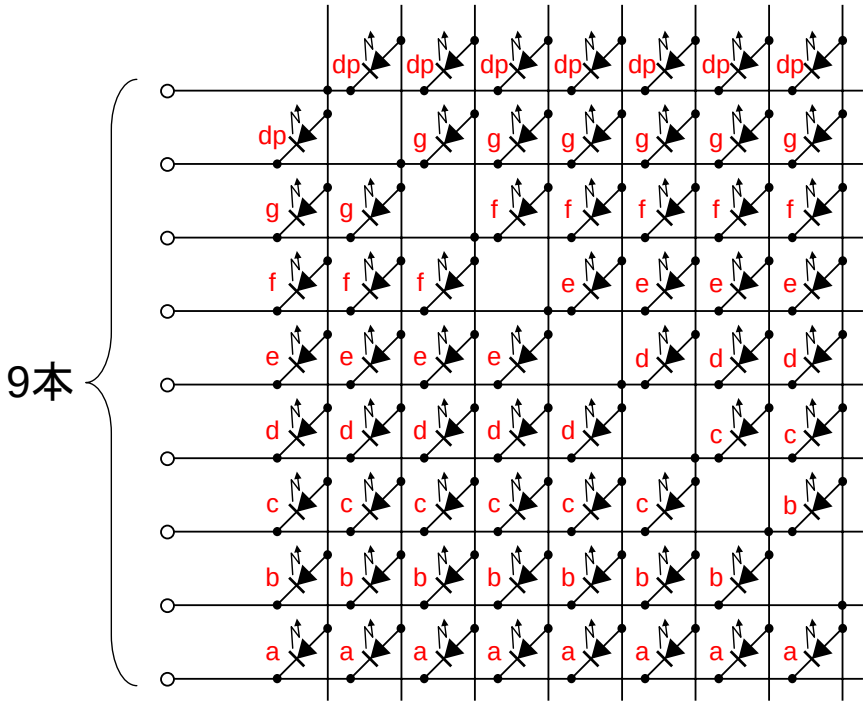
多くのLEDへ拡張

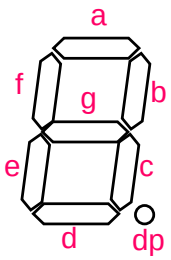
つまり、このように拡張できます。
これを９×８に拡張してみます。



多くのLEDへ拡張

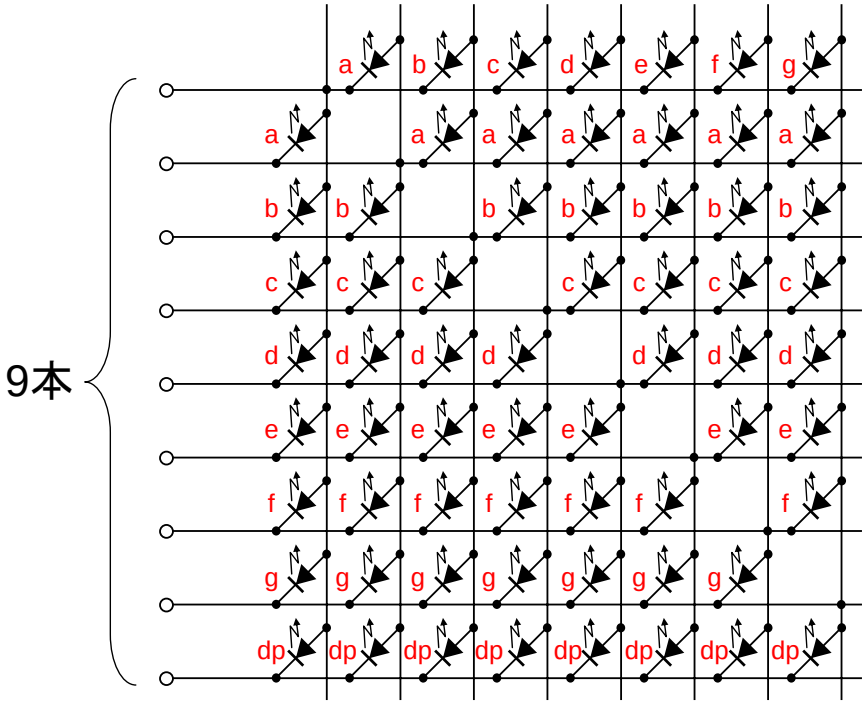
すると、始めのCharlieplexingの回路になります。

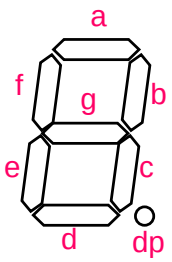




拡張の一工夫

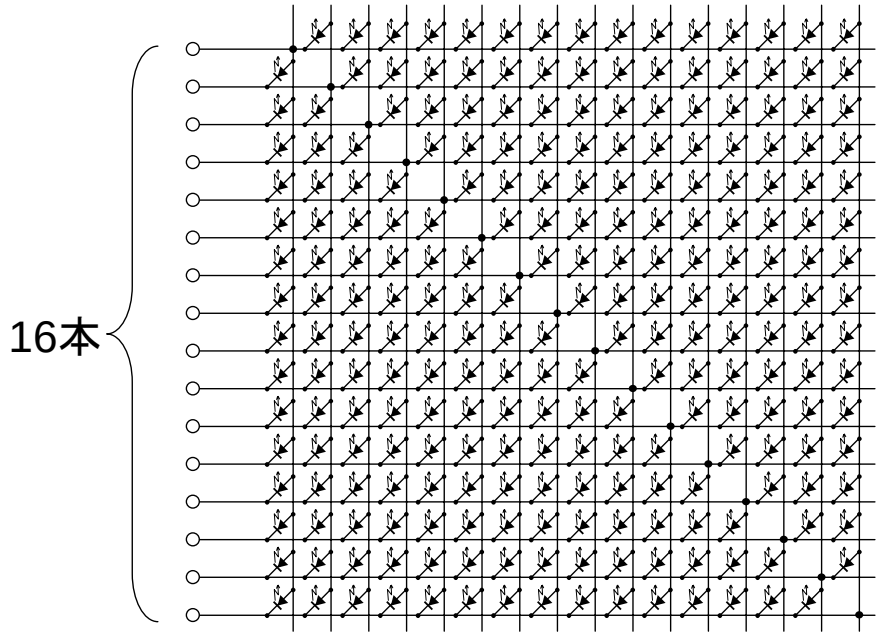
セグメントの配置を逆順にして、dpのみを分離し、
同じセグメントをなるべく同じラインにしたのがikkeiplexingというわけです。





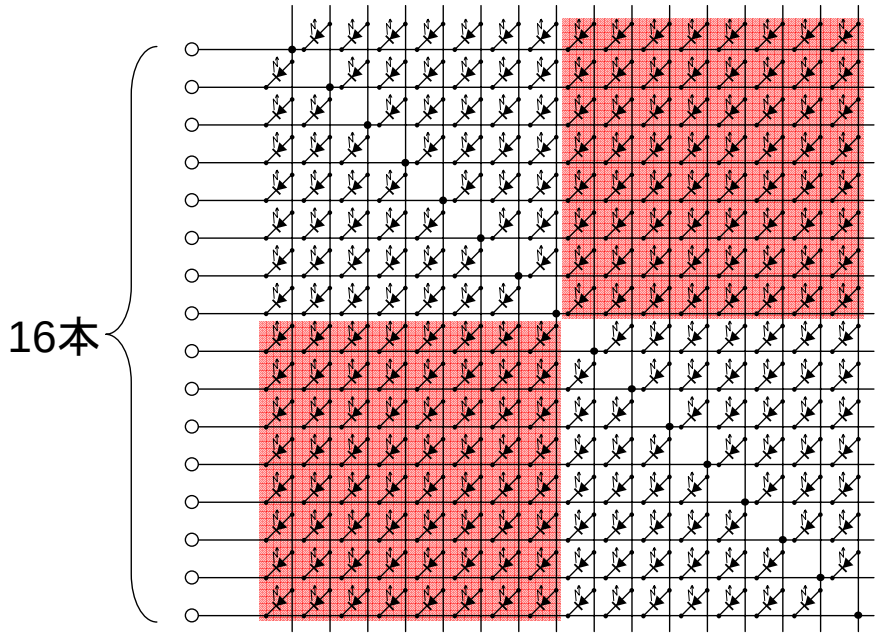
16本線の場合

さらに16本に拡張した場合は240個ものLEDを駆動することができます。
よく見ると、この回路の中に8x8の部分があるのが分かります。



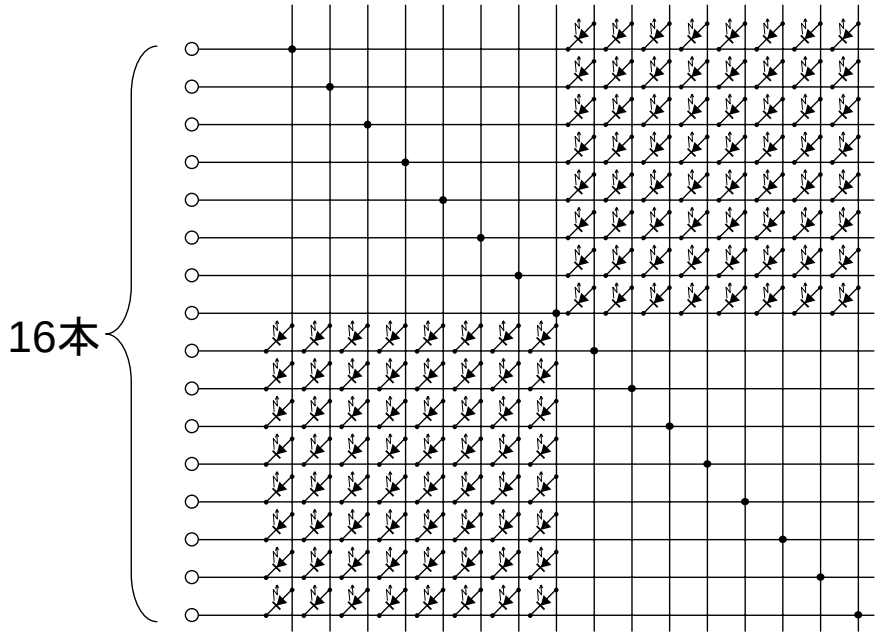
16本線の場合

この赤い部分が8x8のマトリクスになっています。



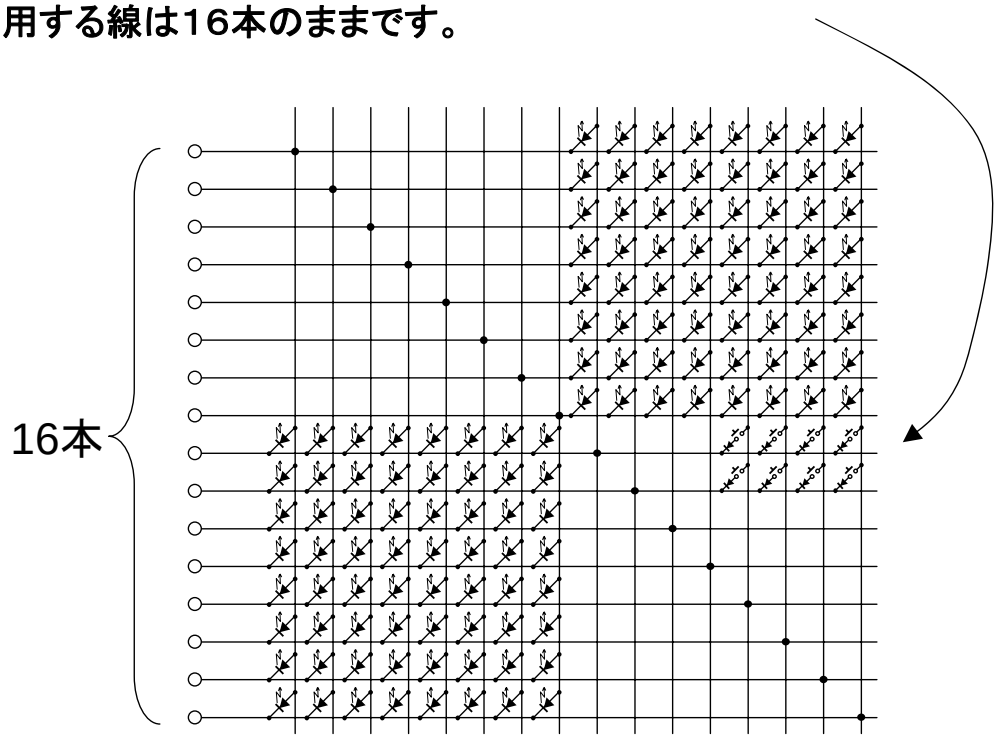
16本線の一部分

この部分だけを残すとこのようになります。
しかし、LEDの表示だけだと操作が出来ません。 しかも残りの端子は
2，3個しか残っていません。なんとかたくさんのスイッチを付けたいものです。



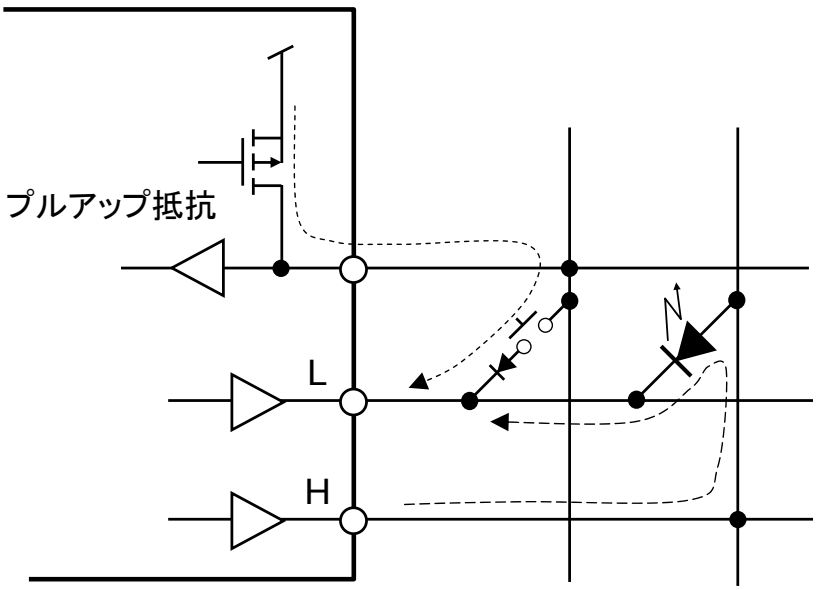
スイッチの追加

と言うことで、空いているところにLEDの代わりにスイッチを追加しました。
使用する線は16本のままです。

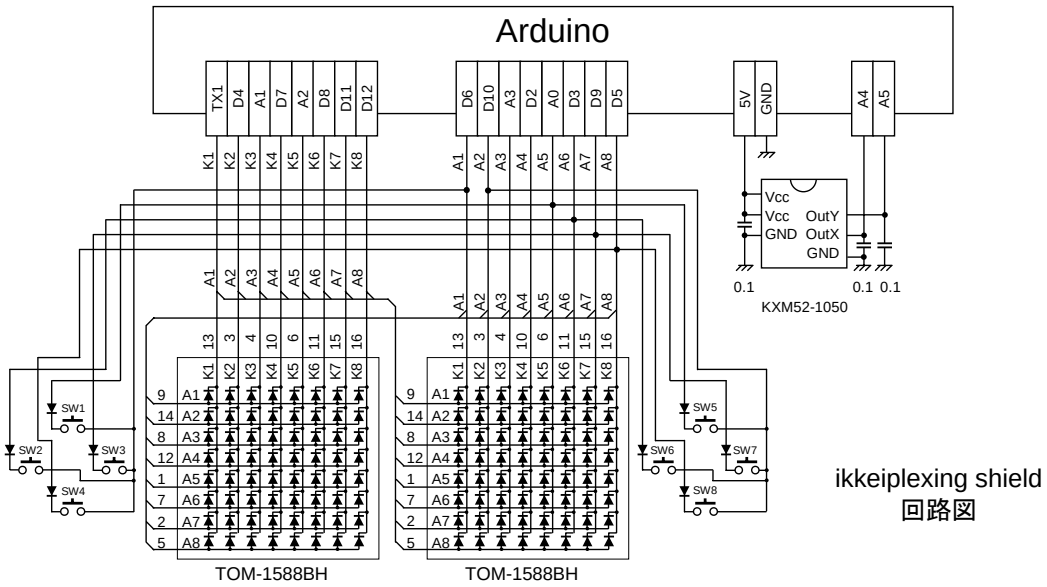


スイッチの入力方法

スイッチ入力をする場合、プルアップ抵抗を使ってスイッチのON/OFFを読み取ります。しかし、この時に同時にLEDを点灯させてしまうと、L出力の電圧が上がってしまうので、スイッチが読み取れなくなります。
従って、全消灯のタイミングでスイッチを読み取ります。

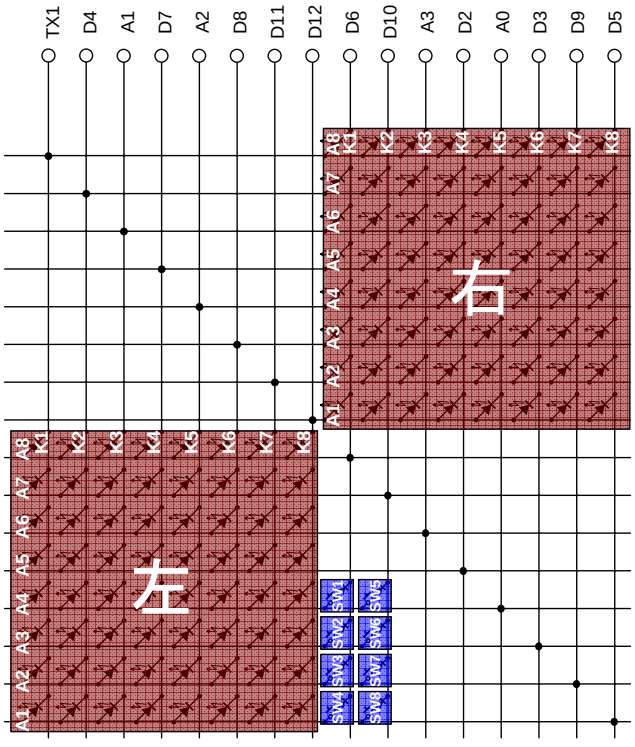


ikkeiplexing shield 回路図



ikkeiplexing shield
回路図

マトリクスとLED/SWの配置



お疲れ様でした

ikkei blog

http://blog.goo.ne.jp/jh3kxm

ikkei Electronics

http://web.mac.com/kxm_ikkei/