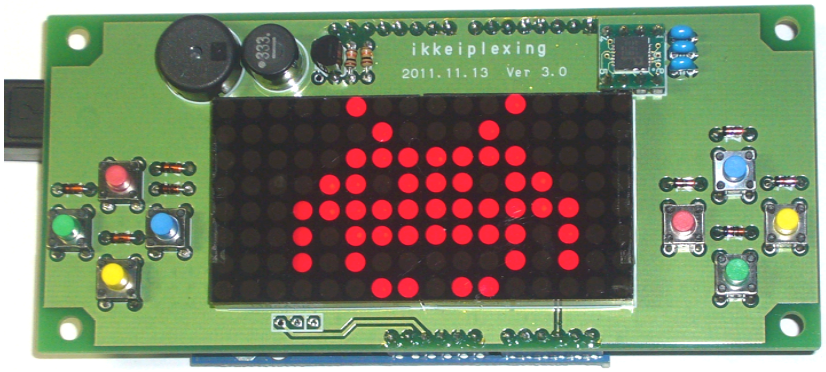
ikkeiplexing shield user's manual
ikkeiplexing シールド製作説明書



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、ハンダを付けるときに温度が下がりやすいので、
　1mm程度のもので良いでしょう。
・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、最近のコテ先は減りにくいように純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

・0.6～1mm位の太さが一番使い易いです。
・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おくと良い。

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線を
　当て、その上からハンダゴテを当ててハンダが浸みてくるのを待ちます。
・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、刃が傷むので切ってはいけません。

ピンセット

・挟んでもたわまない、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。
・チップ部品などを使用する場合は、くっつかないように、非磁性体のピンセットが良いです。

私の愛用品

VICTOR　110

HOZAN　P-892

電子工作の道具　４

ikkeiplexing シールド製作では使いませんが有れば便利なもの

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。AWG24～30

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。

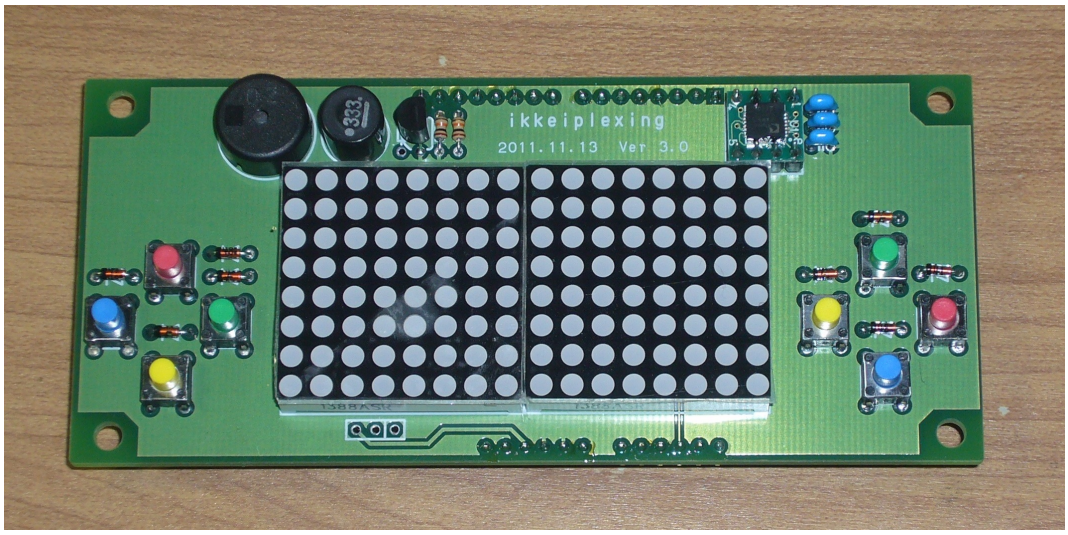
ラジオペンチ

・有れば便利ですが、あまり使わないのでとりあえず無くても良いです。

私の愛用品

1．ikkeipleing シールドの製作

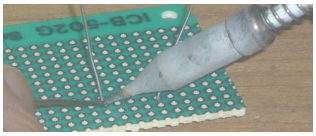
・専用基板に部品をハンダ付けするだけで、簡単に製作出来ます。
・加速度センサ、ブザー、スイッチも取り付けるので、ゲームなどに応用出来ます。



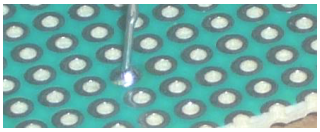
（1）ハンダ付けのコツ

1．コテ先をハンダを付けるところ（部品のリード線など）に当てます。

2．そこへハンダを持っていってコテ先に当てて溶かします。



3．ハンダが流れ、浸み込むのを見届けてからコテ先を離します。



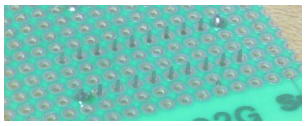
・コテ先を離したあとは息を吹いたりしないで、自然にさまします。
・ハンダを付けるときコテ先を銀色できれいな状態にします。
・新品のコテ先を使い始めるときは、温まったらすぐにハンダを付けておきます。
・余分なハンダはスポンジの角で落とします。
　たたき落とすようなことは絶対にしてはいけません。

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。

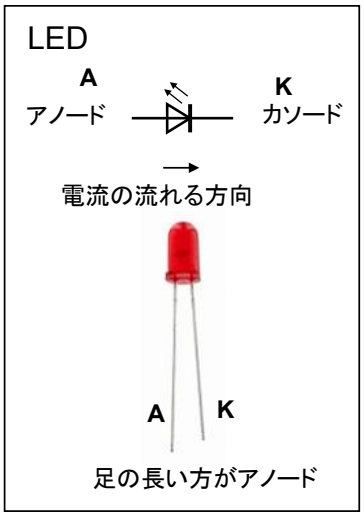
・余分なリード線はハンダ付けした後でニッパーで切断します。
その時、切れ端が飛ばないように注意します。短く切りすぎないように。

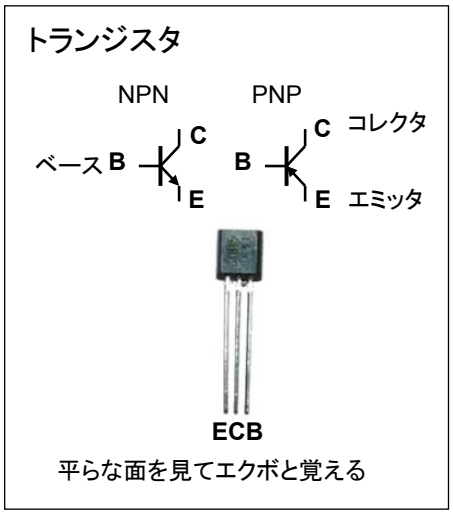
・端子がたくさんある部品は、始めに対角線上の2個の端子をハンダ付け
したあと、部品を押さえながらハンダを溶かして部品の浮きを無くします。



部品の向きに注意！

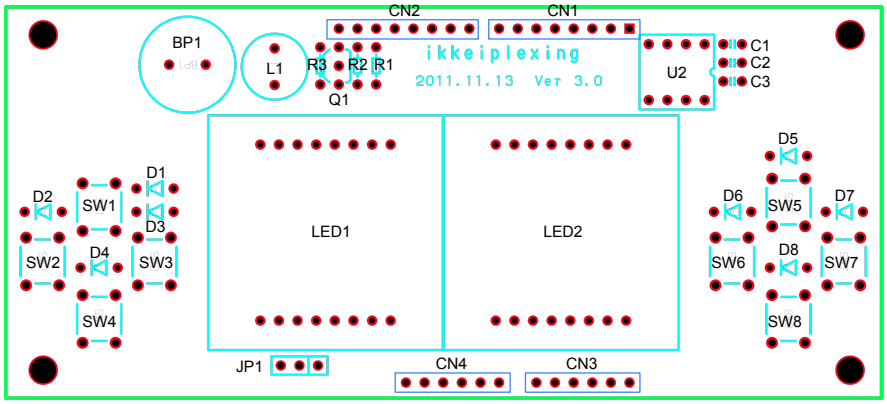
・部品の向きに注意して、挿してから再度向きを確認します。
ダイオードやICなどの部品には方向があります。
間違って逆にしてしまうと付け直すのは大変です。
念には念を入れて何度も確認しましょう。





(2) ハンダ付けの手順　1

部品番号



部品表

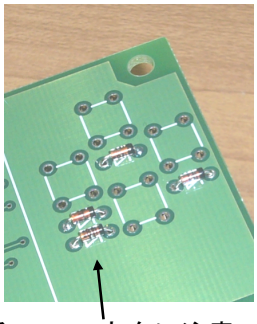
	部品番号	品名	品番／仕様
1	D1～D8	ダイオード	1N4148　スイッチング
2	R1～R3	抵抗	10KΩ
3	C1～C3	コンデンサ	0.1μF
4	BP1	圧電ブザー	Φ13mm
5	LED1,LED2	8x8 dot matrix LED	1388ASR　1.26"
6	SW1～SW8	タクトスイッチ	赤、黄、青、緑　各2個
7	Q1	トランジスタ	2SC1815　汎用
8	L1	インダクタ	10～33mH
9	U2	加速度センサ	KXM52-1050 または KXR94-2050
10	CN1～CN4	ヘッダピン	8P、6P　各2個

＊ L1を使用する場合R3は不要

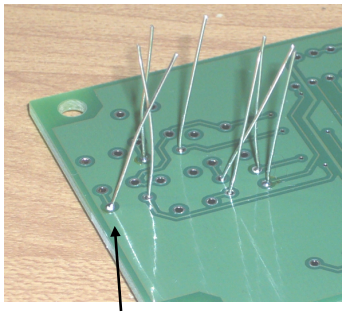
ハンダ付けの手順　2

1. ダイオード
   1. リード線を曲げて方向に注意して基板に挿します。
   2. まず片方だけハンダ付けします。
   3. 表側から見て基板から浮いていれば、ハンダを溶かして修正します。
   4. 残りのリード線もハンダ付けします。
   5. リード線をニッパーでカットします。
      その際、リード線を飛ばさないようにします。
      8本ともハンダ付けします。

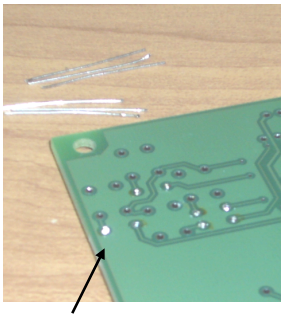
リード線を曲げる



方向に注意



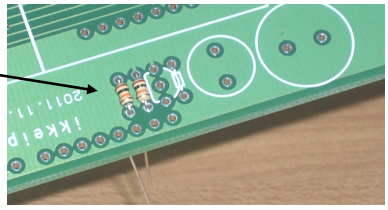
片方から



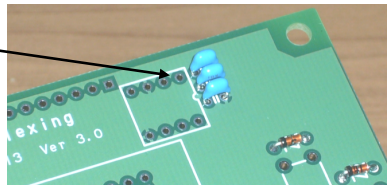
両方付けてカットする

ハンダ付けの手順　3

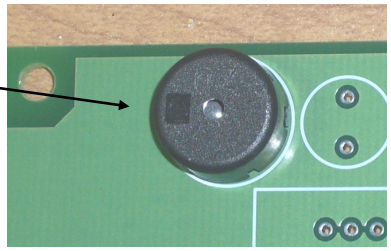
2. 抵抗
   ・R1,R2をダイオードと同じ要領で
   ハンダ付けします。
   R3については8. インダクタで説明します。



3. コンデンサ
   ・C1～C3をハンダ付けします。

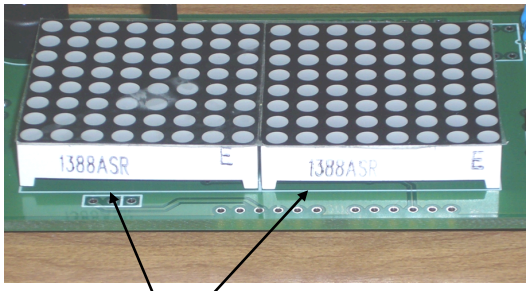


4. 圧電ブザー
   ・BP1をハンダ付けします。



ハンダ付けの手順　4

5. LED
   1. 部品名が手前になるようにLEDを挿し込みます。よく確認してください。
   2. やはり、対角の2本のピンを先にハンダ付けします。
   3. 表側から基板から浮いていないか、チェックします。
      浮いていれば、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。

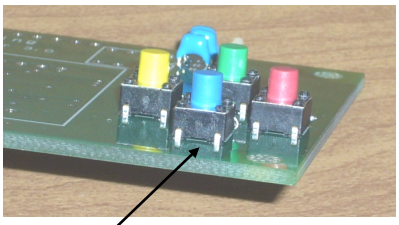


部品名を手前にする

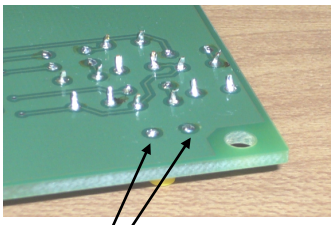
ハンダ付けの手順　５

６．タクトスイッチ

１．スイッチをしっかり押さえてはめ込みます。
２．やはり、まず対角の２本のピンをハンダ付けし、浮きをチェックします。
３．浮いていれば、押さえながらハンダを溶かして浮きを無くします。
４．全部ハンダ付けします。
５．DCプラグに干渉しないように、SW4の下側の２本の足をカットします。



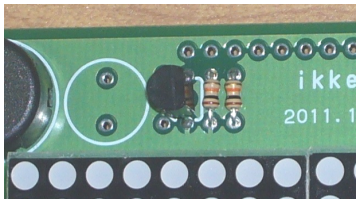
基板から浮いていないかチェックする



２本の足をカットする

７．トランジスタ

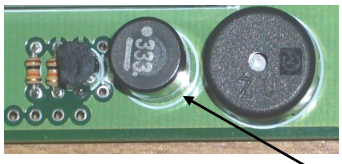
・向きに注意してQ1をハンダ付けします。

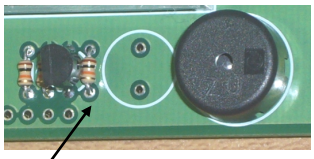


ハンダ付けの手順　６

８．インダクタ

・L1またはR3をハンダ付けします。
・L1の方が音が大きくなりますが入手出来ない場合はR3でも構いません。
・両方ハンダ付けしてしまっても問題ありません。

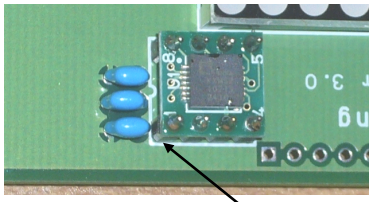




L1かR3のどちらかをハンダ付けします。

９．加速度センサ

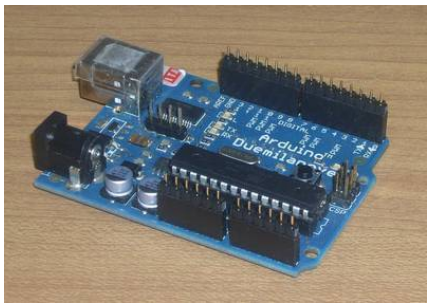
・1番ピンをコンデンサ側にしてU2をハンダ付けします。
加速度センサは直接ハンダ付けします。
ICソケットを使用しても構いませんが、LEDより高くなるため、抜け易くなってしまいます。



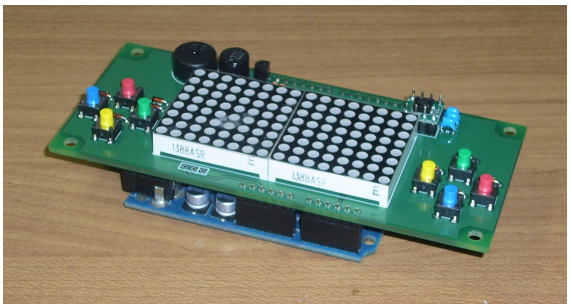
1番ピン

ハンダ付けの手順　7

10. 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。



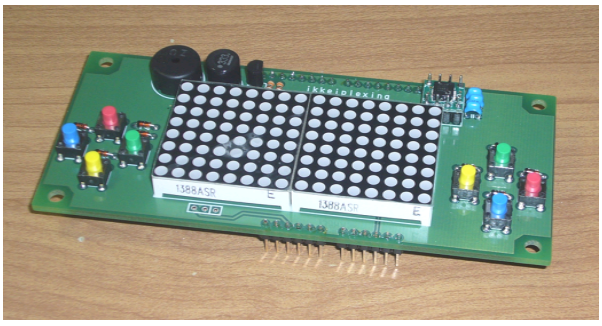
ヘッダピンはArduinoに挿す

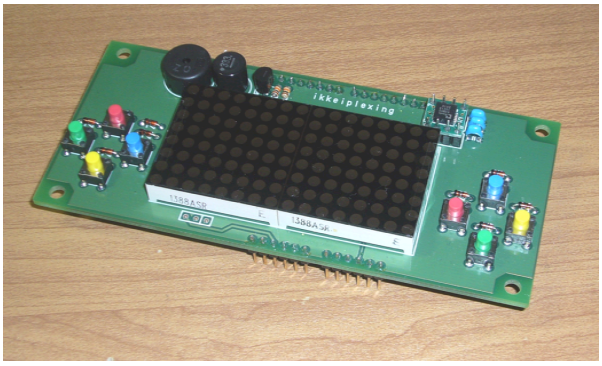


基板をかぶせる

ハンダ付けの手順　8

11. スモークフィルタ
    - コントラストを上げるため、スモークフィルタを貼ると良いです。
      例えば、塩ビ板 スモークなど



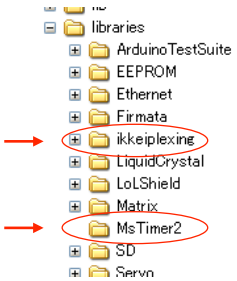


2. スケッチの説明

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

(0) スケッチのインストール

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



サンプルスケッチの選択:
スケッチ例
ikkeiplexing
exsamples for ikkeiplexing shield

メニューから選択する

サンプルスケッチを修正した場合は、
上書き保存が出来ないので、
他のところに保存します。

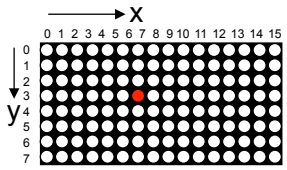
(1) LED チカチカ　L_chika

・L_chika

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

ikkeiplexing iLedSign;

void setup(){
  iLedSign.Init(IKKEIPLEXING, SINGLE_BUFFER);
}

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出力を反転

LED表示出力（ s=0:消灯 s=1:点灯）

(2) 反射　reflection

・reflection

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

void setup(){
  iLedSign.Init(IKKEIPLEXING, SINGLE_BUFFER);
}

void loop(){
  static int8_t x = 0;
  static uint8_t dx = 1;
  static int8_t y = 0;
  static uint8_t dy = 1;

  iLedSign.Set(x, y, 0); // turn off the previous dot
  x += dx;
  y += dy;
  iLedSign.Set(x, y, 1); // turn on the next dot

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

端を定義 （ x:0～15 y:0～7 ）

前に点灯させた点を消灯

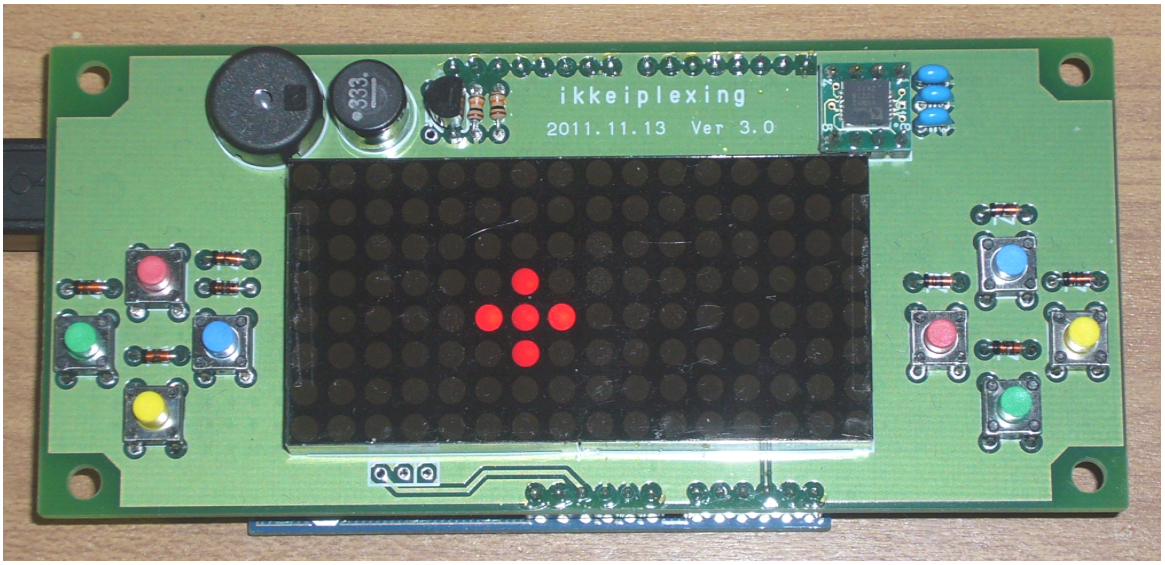
次の点の座標を計算

次の点を点灯

端まで来れば反転

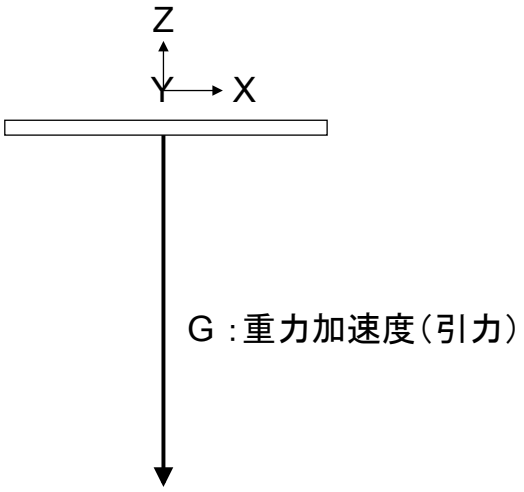
(3) LEDコロコロ　gravity

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。

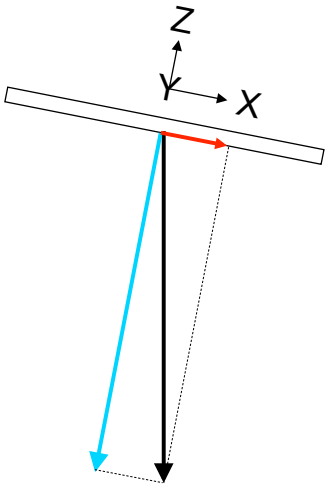


加速度センサについて

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。

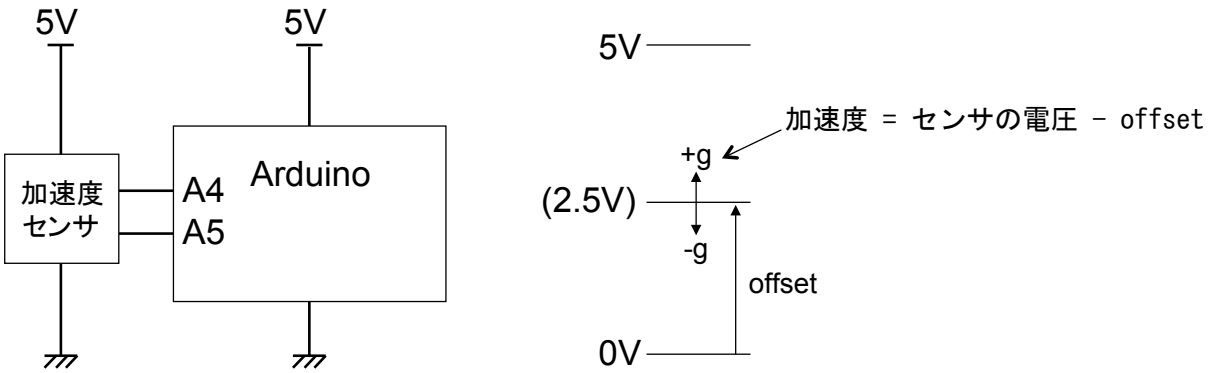


キャリブレーション

加速度ゼロの状態でもセンサの出力には誤差があるので、水平の時の出力値をゼロ状態として保存しておき、その値からの差を出力値とします。これをゼロ調整（キャリブレーション）と言います。

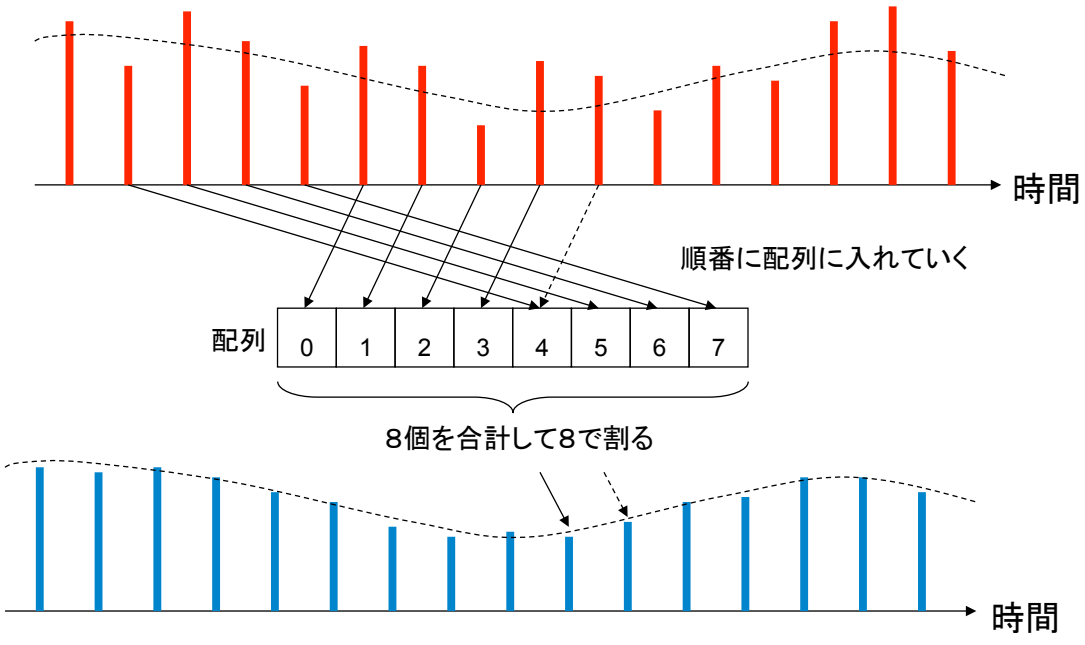
今回、起動時の状態を初期値としてoffset値とします。

加速度はセンサの電圧からoffsetを引いて求めます。



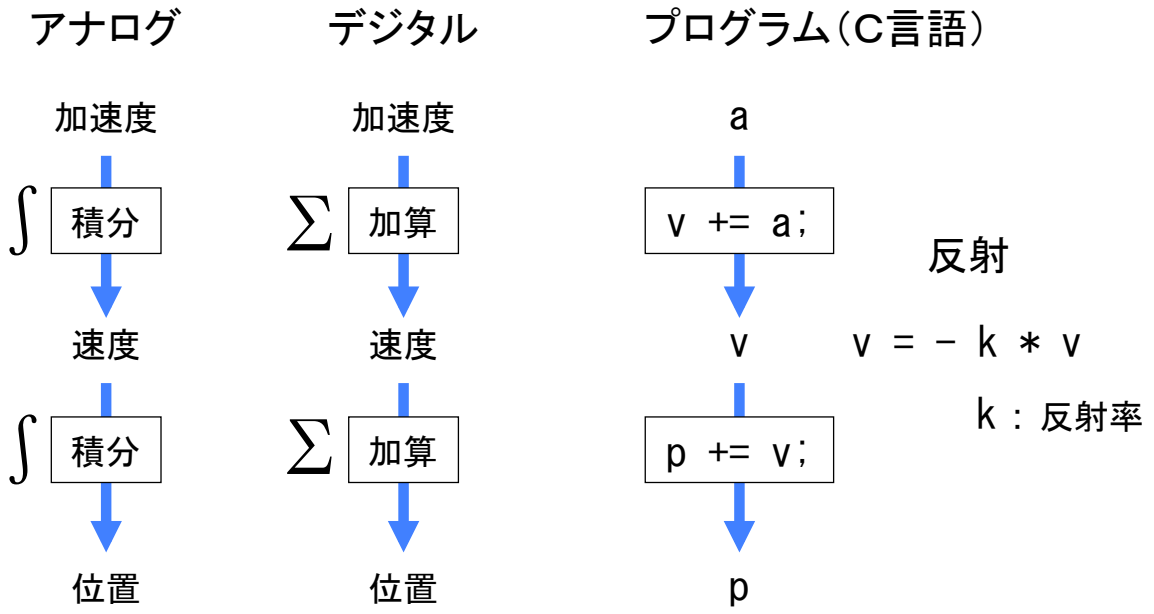
移動平均

加速度の値はかなりバラつくので平均を取ってならします。
測定値を移動させながら平均値を計算します。これを移動平均と言います。

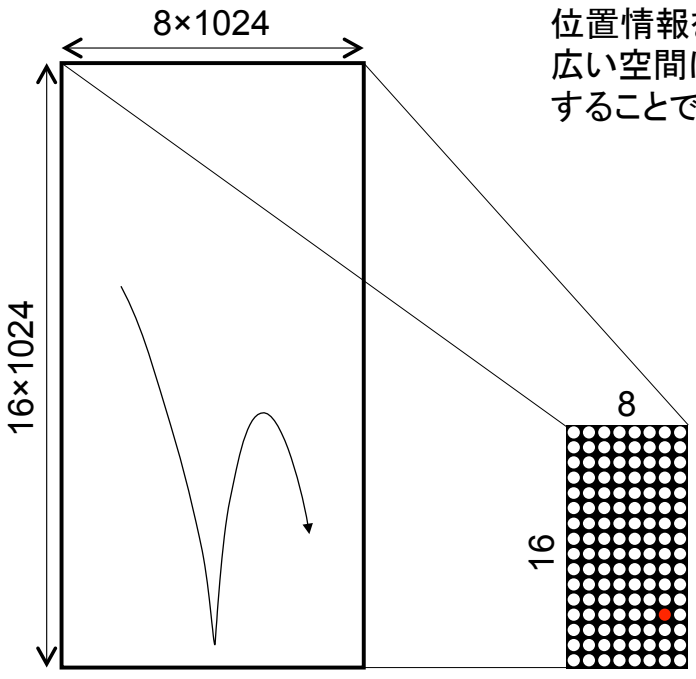


加速度、速度、位置

加速度を積分すると速度に、速度を積分すると位置になります。
ただ、積分と言ってもプログラムにすると下記のように1行になります。
また、端まで行ったら速度を反転すると跳ねます。ただし反射率を1より小さくしないと止まらなくなります。



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。

全体構成、加速度測定、移動平均

・gravity

```
void setup(){
  iLedSign.Init(IKKEIPLEXING, SINGLE_BUFFER);
  delay(1000);
  offset.x = analogRead( AX );
  offset.y = analogRead( AY );
  position.x = 1;
  position.y = 1;
}

void loop(){
  if ((iLedSign.GetPeriod() & FRAME) != 0){
    gravity();
  }
}
```

加速度センサの初期値をオフセット値にする

初期位置

フレームの検出（表示との同期をとる）

全体処理

```
void gravity( void ){
  // averaging acceleration sensor signal
  acceleration_in[ spt ].x = analogRead( AX ) - offset.x;
  acceleration_in[ spt ].y = analogRead( AY ) - offset.y;
  spt++;
  if ( spt >= AVE ){
    spt = 0;
  }
  acceleration.x = 0;
  acceleration.y = 0;
  for (byte i=0; i < AVE; i++){
    acceleration.x +=  acceleration_in[ i ].x;
    acceleration.y +=  acceleration_in[ i ].y;
  }
  acceleration.x /= AVE;
  acceleration.y /= AVE;
```

センサの電圧からオフセットを引いて加速度とし、配列に入れる

入れる場所を移動する

移動平均の計算

速度、位置の計算、反射、スケーリング

・gravity

```
 // integrating acceleration makes velocity
  velocity.x +=  acceleration.x;
  velocity.y += -acceleration.y;

  // integrating velocity makes position
  position.x += velocity.x;
  position.y += velocity.y;

  // reflection ( reverse velocity )
  // 0 < x < MAX.x
  if ( position.x >= MAG * MAX.x){
    position.x = MAG * MAX.x - 1;
    velocity.x = -( velocity.x * REF1 ) / REF2;
  }else if ( position.x < MAG ){
    position.x = MAG;
    velocity.x = -( velocity.x * REF1 ) / REF2;
  }
  // 0 < y < MAX.y
  if ( position.y >= MAG * MAX.y){
    position.y = MAG * MAX.y - 1;
    velocity.y = -( velocity.y * REF1 ) / REF2;
  }else if ( position.y < MAG ){
    position.y = MAG;
    velocity.y = -( velocity.y * REF1 ) / REF2;
  }

  present.x = position.x / MAG;
  present.y = position.y / MAG;
```

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

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

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データの消去と作成

・gravity

```
  // display limitter
  constrain(present.x, 1, MAX.x - 1);
  constrain(present.y, 1, MAX.y - 1);

  // turn off the last position
  iLedSign.Set(previous.x,     previous.y,     0);
  iLedSign.Set(previous.x +1, previous.y,     0);
  iLedSign.Set(previous.x -1, previous.y,     0);
  iLedSign.Set(previous.x,     previous.y +1, 0);
  iLedSign.Set(previous.x,     previous.y -1, 0);

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

  // store last position
  previous.x = present.x;
  previous.y = present.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　kitchen_timer

プログラムを設計図から作成します。

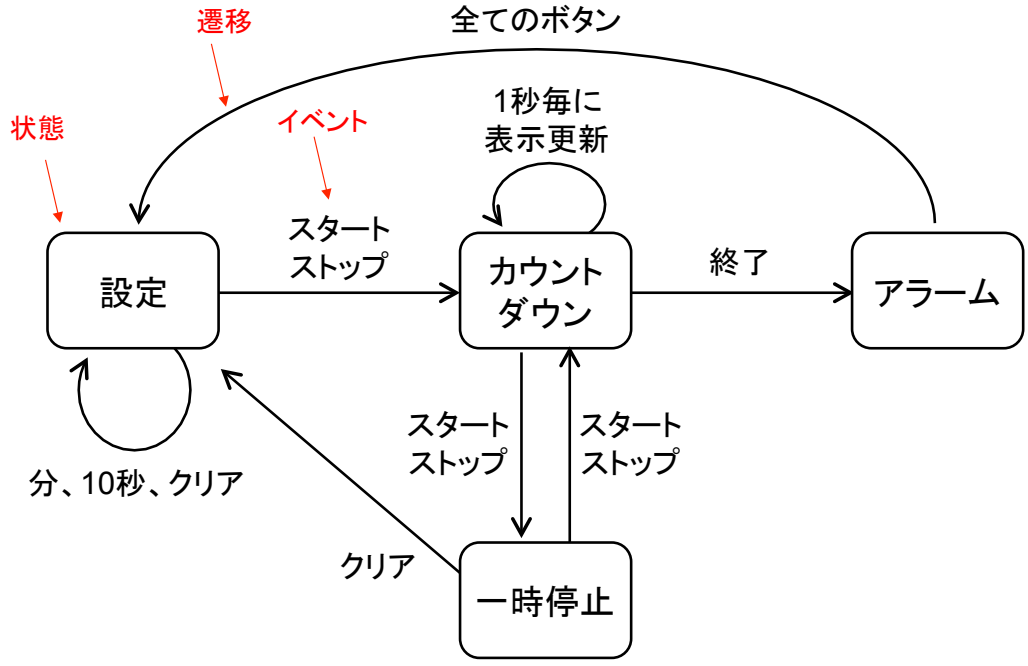
プログラムは適当に思い付いたコードを書いてバグを取って動いたらOKと思っていませんか？
プログラムもちゃんと設計図を描いてからコードを書くことによって信頼性を上げバグの出にくいコードを心がけましょう。

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。

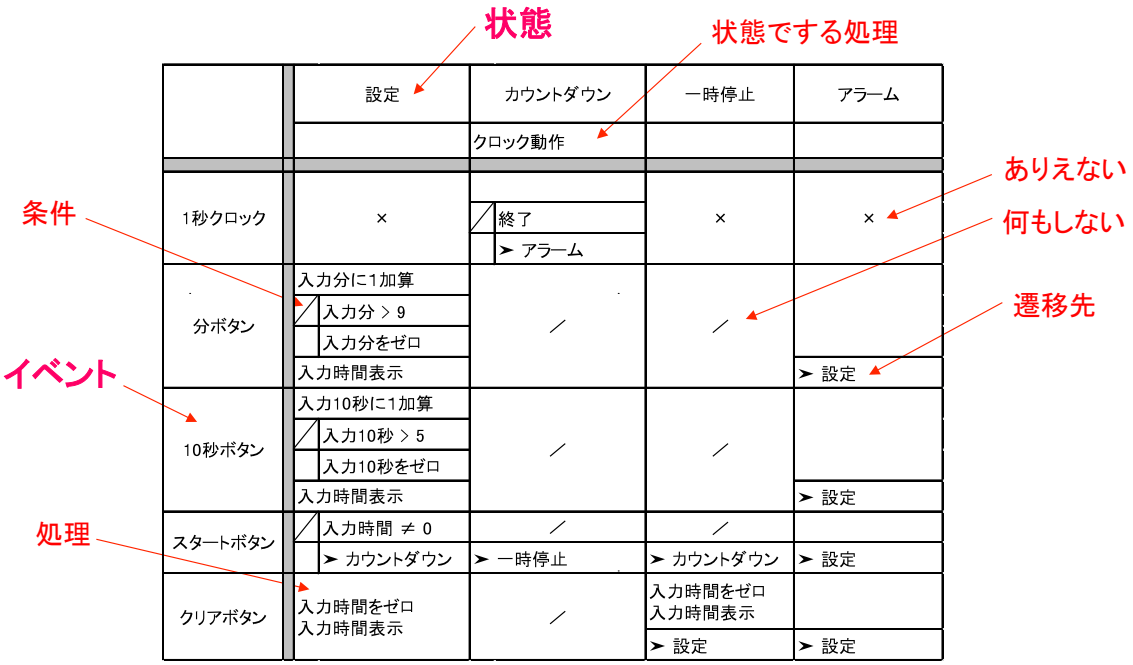
キッチンタイマの動作を状態遷移図で設計する

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

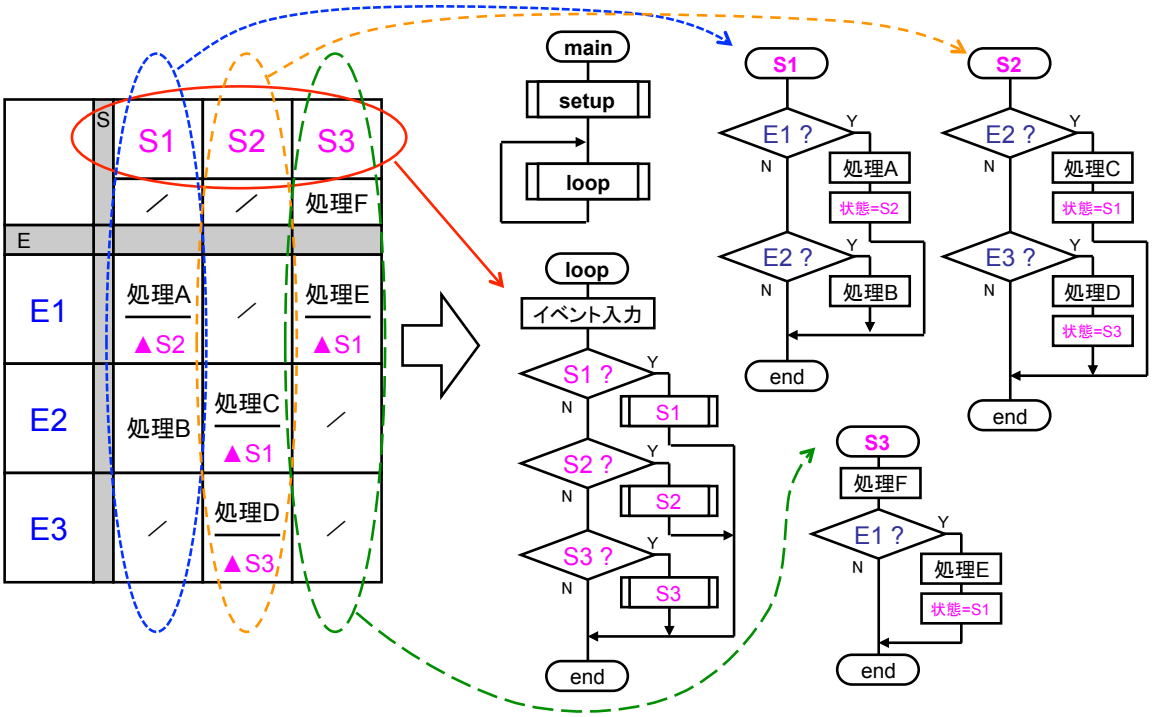
詳細設計：共通処理を状態のところに記入します。

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

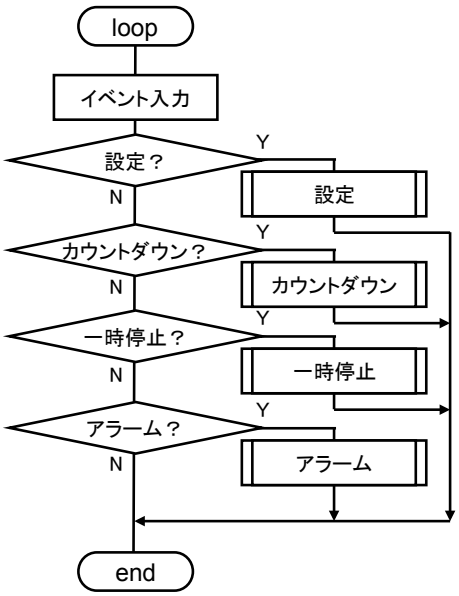
状態遷移表からソースコード作成

状態遷移表からプログラムコードへの変換は、下図の様に機械的に行えます。



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。

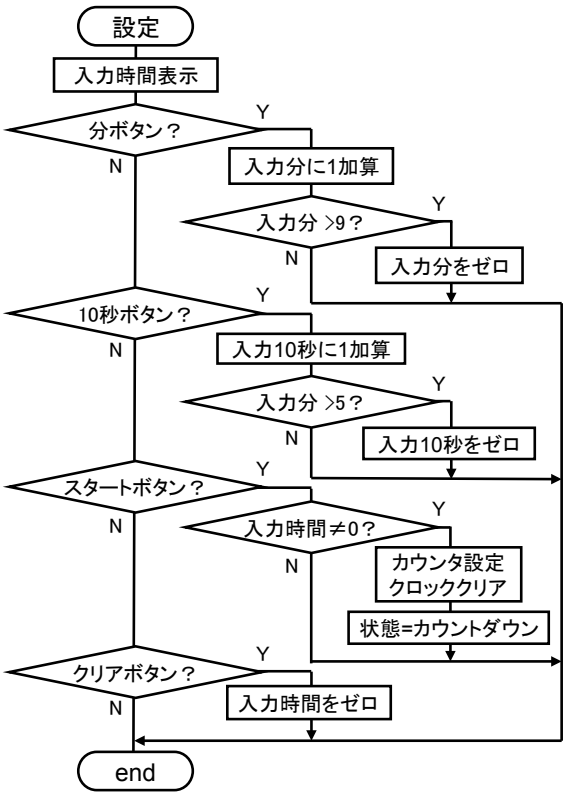


スイッチデータをエッジで取得する

10ms のインターバルをチェック

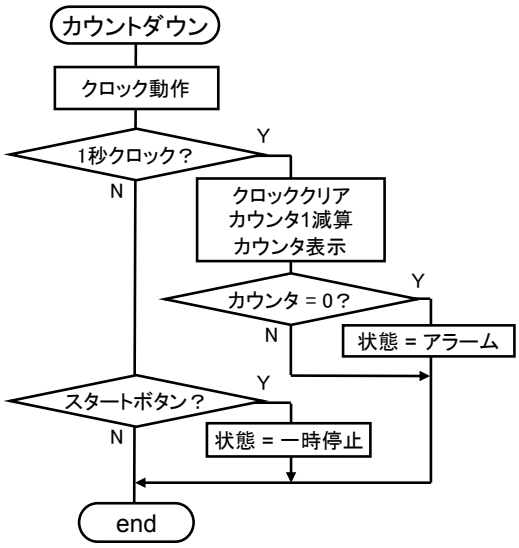
```
void loop(){
  if ((iLedSign::GetPeriod() & TENMS) != 0){
    event = iLedSign::GetSW(EDGE);
    switch (state){
      case set_time:    set_time_state();    break;
      case count_down:  count_down_state();  break;
      case pause:       pause_state();       break;
      case alarm:       alarm_state();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


キッチンタイマのソースコード作成　2



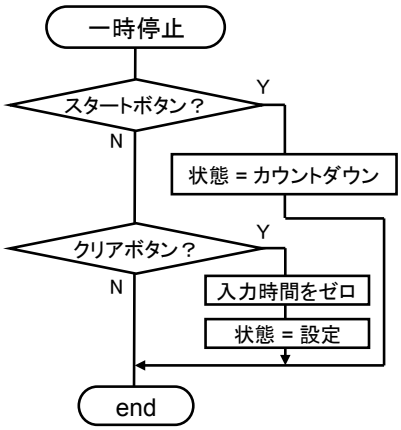

```
void set_time_state(void){
  DisplayLED(set_minute, set_tensec, 0);
  if ((event & SW_MINUTE) != 0){          ← 分ボタン
    set_minute++;
    if (set_minute > 9){
      set_minute = 0;
    }
  } else if ((event & SW_TENSEC) != 0){   ← 10秒ボタン
    set_tensec++;
    if (set_tensec > 5){
      set_tensec =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

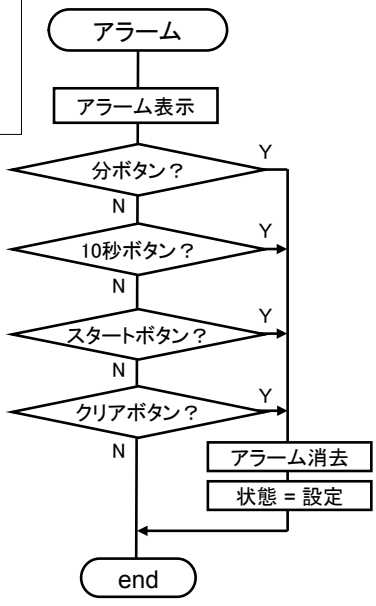



```
void count_down_state(void){
  clock++;
  if (clock >= 100){          ← 1 sec = 10ms X 100
    clock = 0;
    if (count_second > 0){
      count_second--;
    } else {
      count_second = 9;
      if (count_tensec > 0){
        count_tensec--;
      } else {
        count_tensec = 5;
        if (count_minute > 0){
          count_minute--;
        }
      }
    }                          } カウンタ1減算
    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



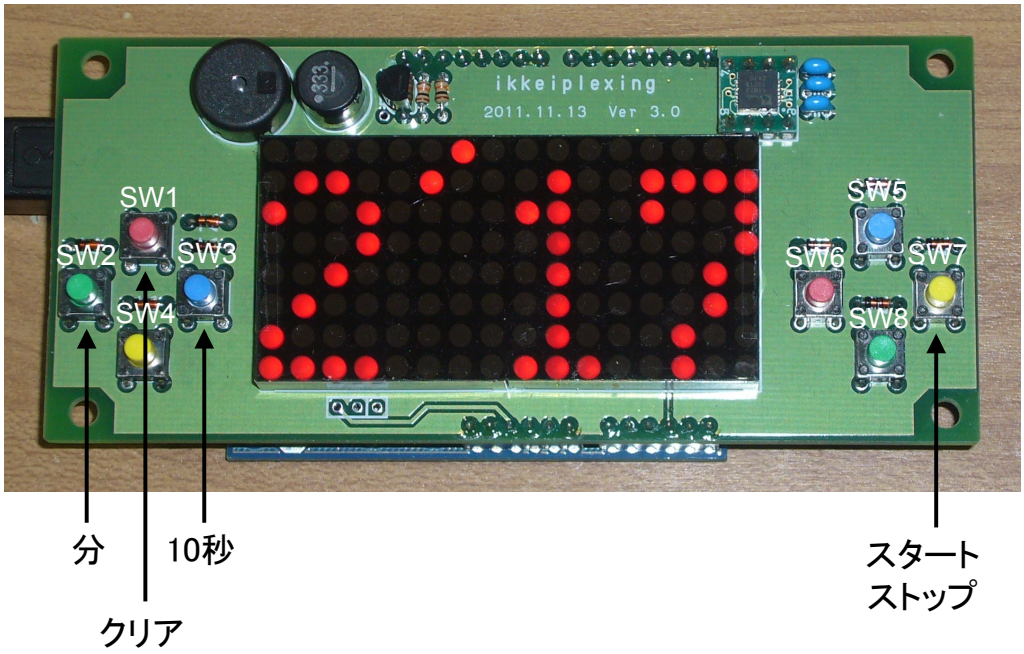

```
void pause_state(void){
  if ((event & SW_START) != 0){
    state = count_down;
  } else if ((event & SW_CLEAR) != 0){
    set_minute = 0;
    set_tensec = 0;
    state = set_time;
  }
}
```




```
void alarm_state(void){
  alarm_count--;
  if (alarm_count == 0){
    alarm_count = ALARM_PERIOD;
    alarm_flash ^= 1;          } アラーム表示
    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

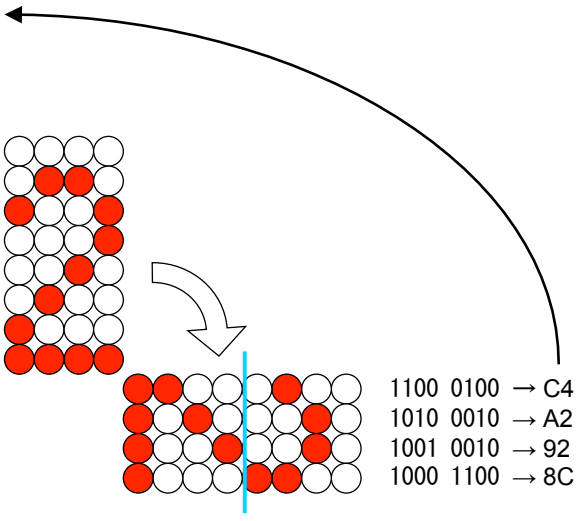
数字を表示して9分50秒までのタイマになります。
ラーメンを作るときに使えます。



数字のパターンの作成

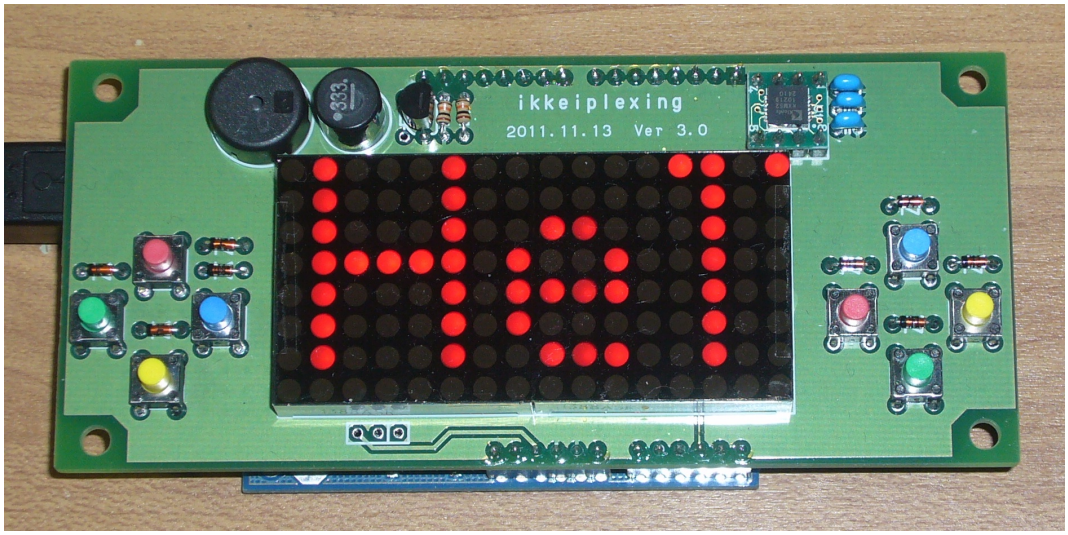
```
const byte FIG[10][4] = {
{ 0x7C, 0x82, 0x82, 0x7C},
{ 0x00, 0x84, 0xFE, 0x80},
{ 0xC4, 0xA2, 0x92, 0x8C},
{ 0x44, 0x82, 0x92, 0x6C},
{ 0x38, 0x24, 0xFE, 0x20},
{ 0x5E, 0x92, 0x92, 0x62},
{ 0x7C, 0x92, 0x92, 0x64},
{ 0x06, 0xC2, 0x32, 0x0E},
{ 0x6C, 0x92, 0x92, 0x6C},
{ 0x4C, 0x92, 0x92, 0x7C}
};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

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　scroll_text

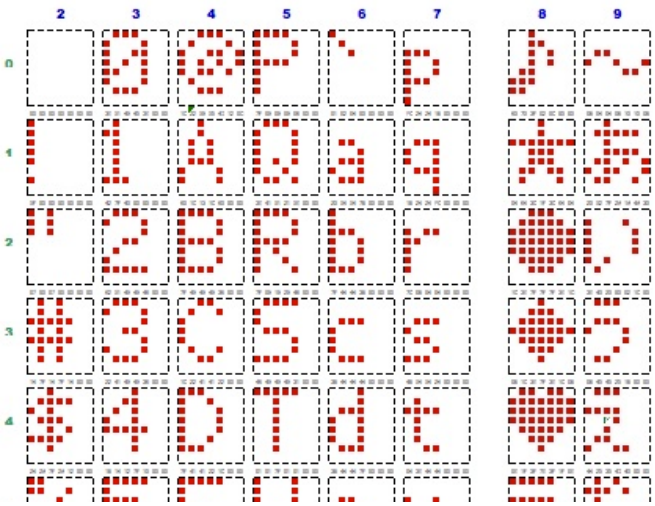
設定した文字列をスクロールしながら繰り返し表示します。

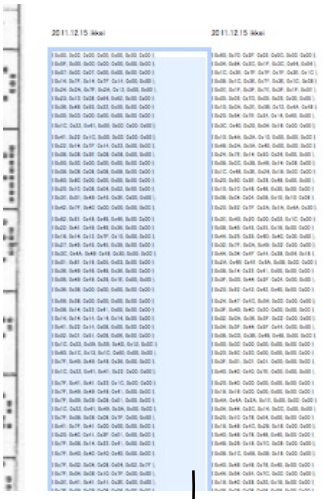


英字やカタカナの文字パターンからデータを作る

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 uint8_t dotfont[][7] PROGMEM = {
```

文字列の表示

・scroll_text

```
void loop(){
  const unsigned char text[]="Hello world! ¥xCA¥xDB¥xB0¥xDC¥xB0¥xD9¥xC4¥xDE! ¥xEA
  //const unsigned char text[]="¥xE1¥x8F¥xE1¥x8C¥x92¥xB0¥xE9 "; // Titchaino

  uint8_t d;
  int textsize = sizeof(text)/sizeof(text[0]);
  for(int16_t j=0;j<textsize;j++) {
    d = iFont.Width(text[j]);
    for(int i=0;i<=d;i++) {
      uint8_t data = iFont.Data(text[j],i);
      iLedSign.Vwrite(15, data);
      delay(80);
      iLedSign.ShiftLeft();
    }
  }
}
```

表示させる文字列
カタカナは16進数を使用する。

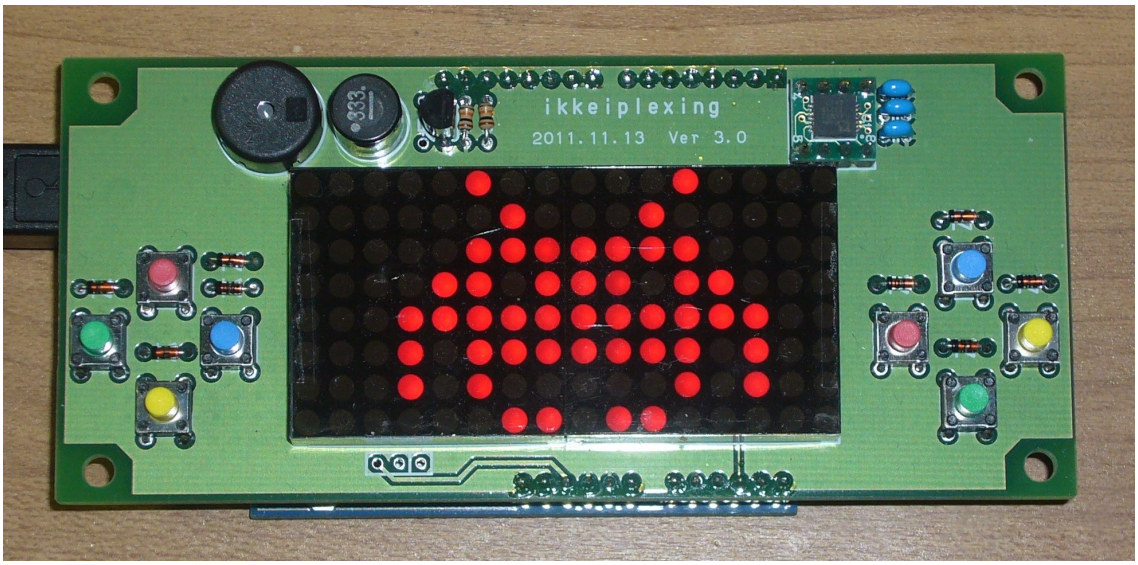
文字フォントの1列を抽出

最右端に1列書き込む

1ドット左スクロール

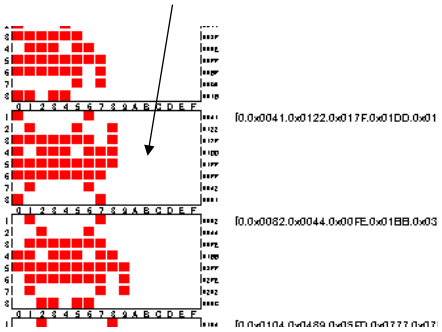
(6) パターン表示　pattern

設定したドットパターンを順々に表示させます。

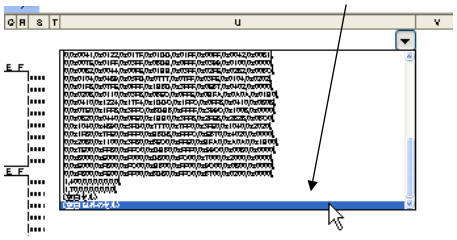


ドットパターンの作成

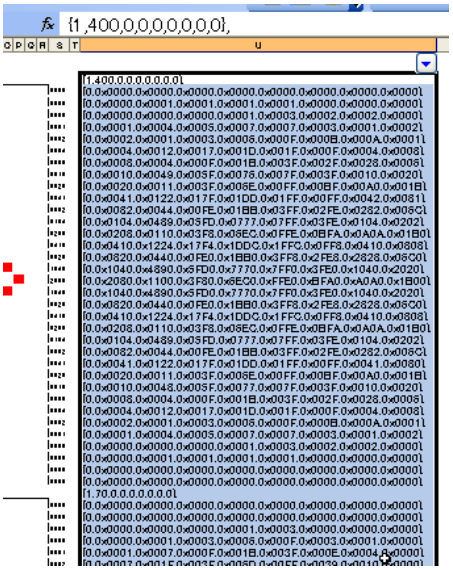
1. Excelでパターンを作成
点灯：1　消灯：0　を入れる。



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

・pattern

Flash ROM上に配置する。

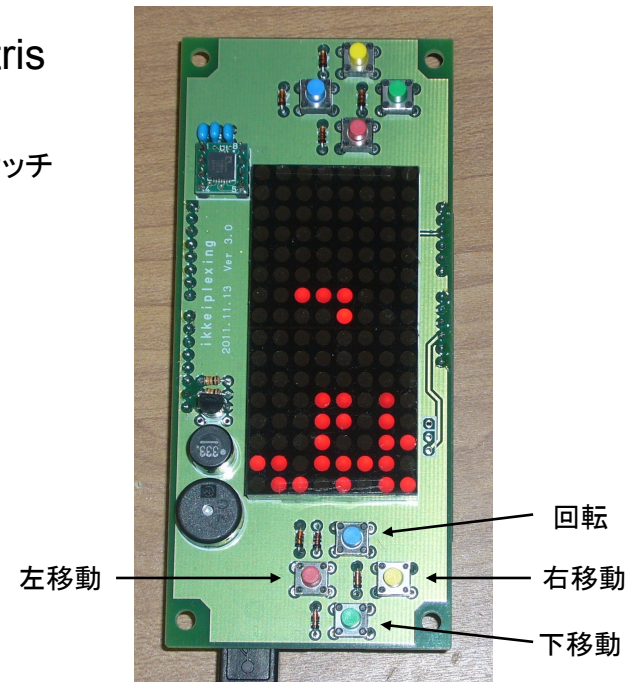
```
uint16_t BitMap[][9] PROGMEM = {
// first data 0:pattern data  1:delay time  2:terminator
{1,400,0,0,0,0,0,0,0},
{0,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000},
{0,0x0000,0x0001,0x0001,0x0001,0x0001,0x0000,0x0000,0x0000},
{0,0x0000,0x0000,0x0000,0x0001,0x0003,0x0002,0x0002,0x0000},
{0,0x0001,0x0004,0x0005,0x0007,0x0007,0x0003,0x0001,0x0002},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

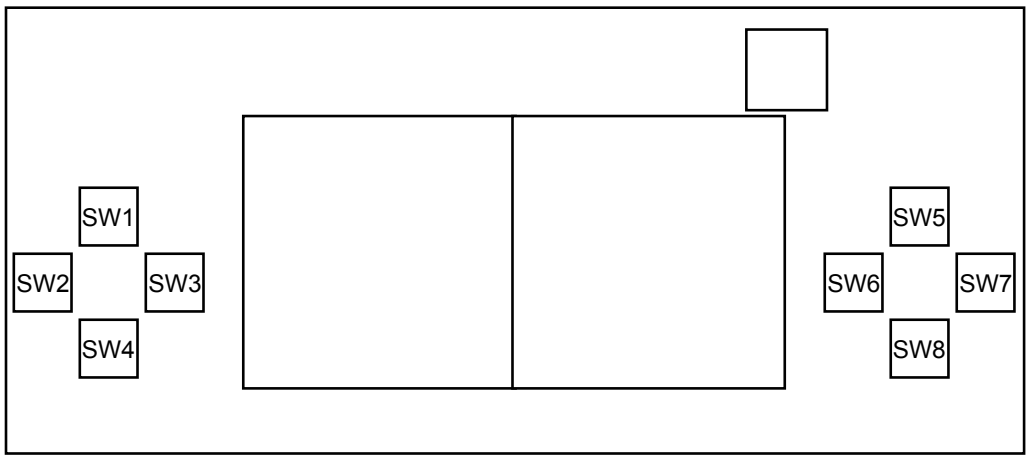
ライブラリを活用して、既存のゲームを動かしてみます。

tetris

LoL shield のスケッチからの移植です。



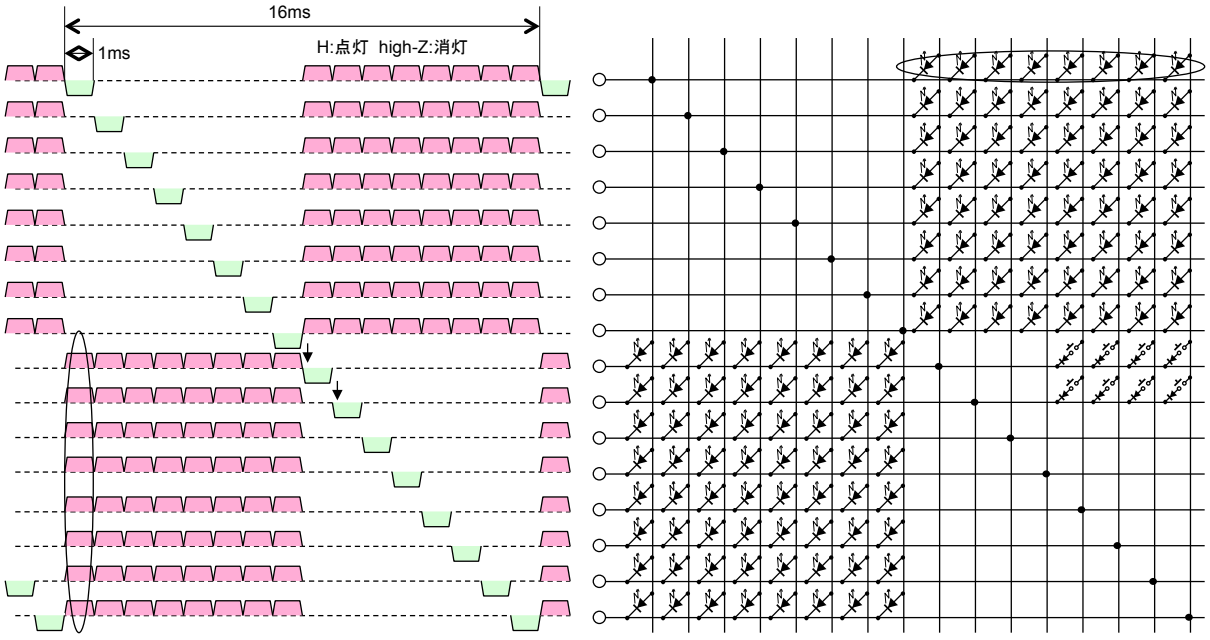
あとはあなたのアイデア次第！！



3.　ライブラリについて

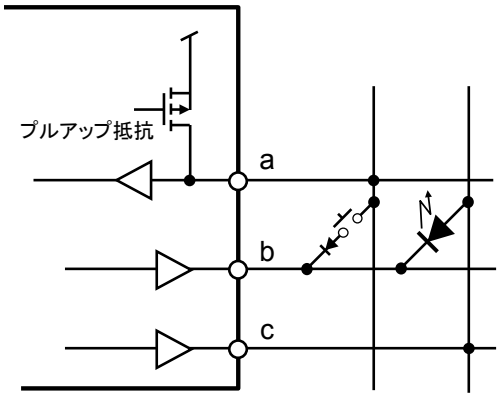
- LEDの点灯　タイミングチャート
- スイッチの取り込みタイミング
- 直結でも問題無いの？
- PWMで輝度補正　原理図
- PWMで輝度補正　実際のタイミング
- スキャン部のコード
- ドットをセットする部分とスイッチの取得

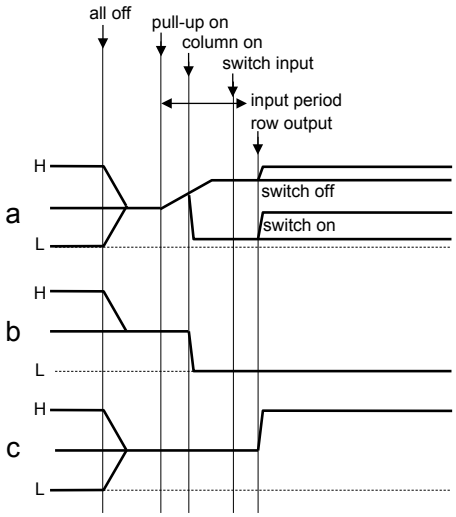
LEDの点灯　タイミングチャート



スイッチの取り込みタイミング

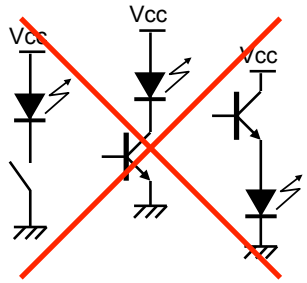
・スイッチの取り込みタイミングは列切り替えの狭間の
LEDが消灯している間に行うので、表示のスキャンには影響しない。



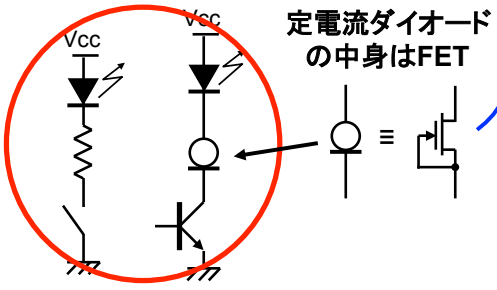


直結でも問題無いの？

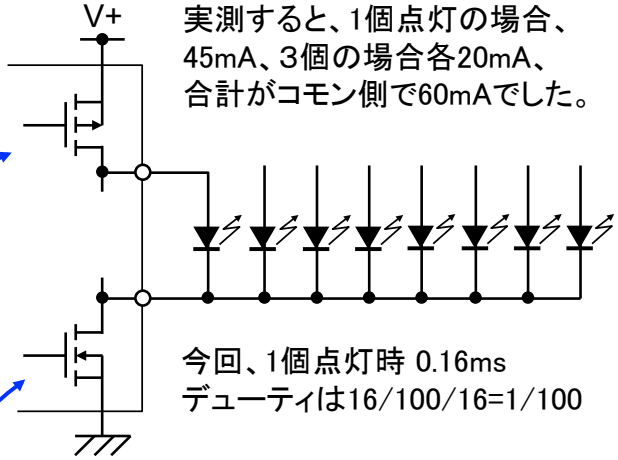
電流が流れ過ぎるので
もちろんこれはダメ！



電流制限しているので
これはOK！



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

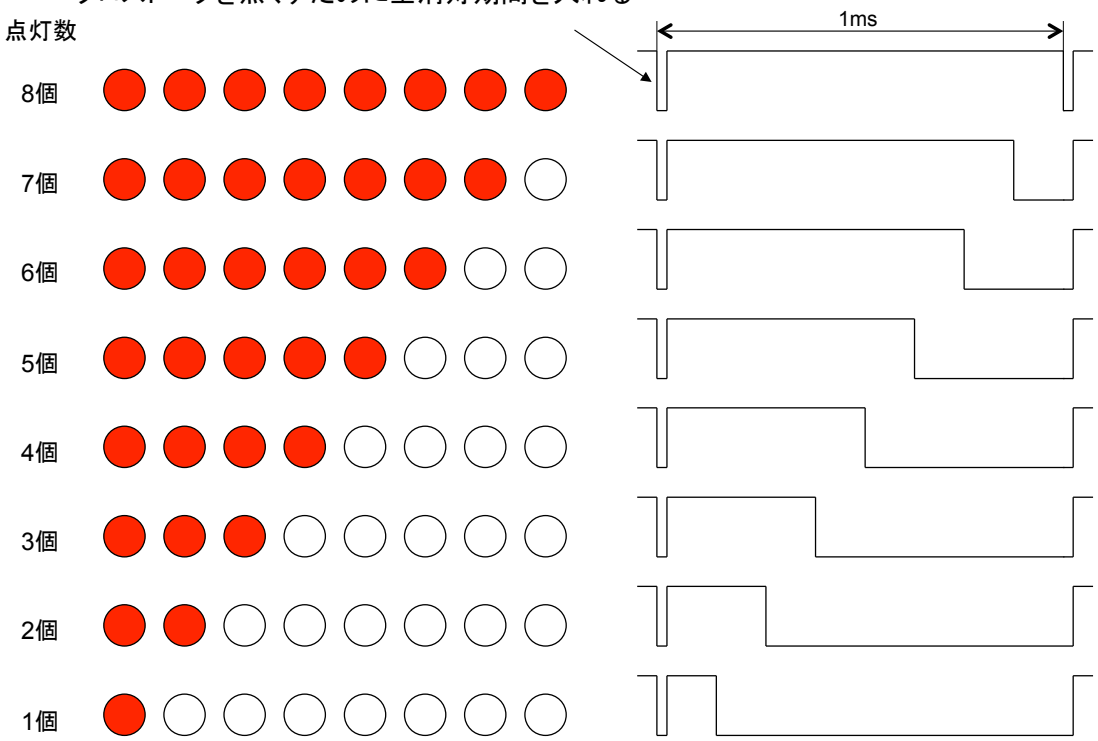
KWM-37881ASB（TOM-1588BH同等品）

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

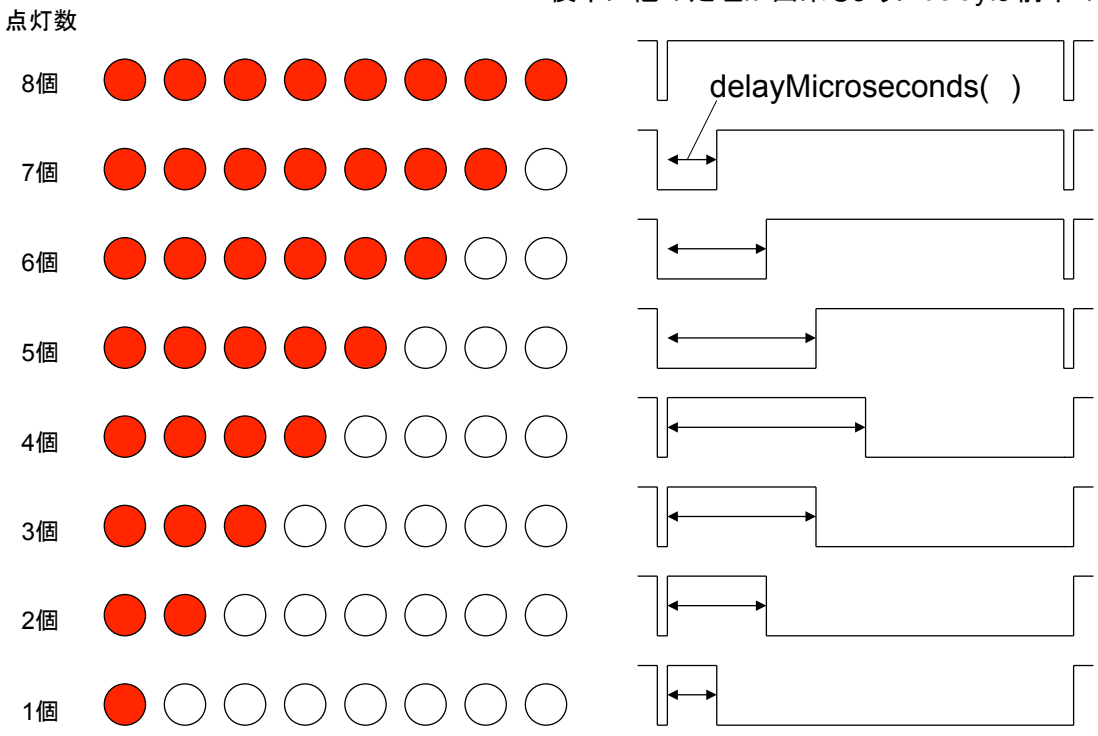
PWMで輝度補正　原理図

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

・ikkeiplexing.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

・ikkeiplexing.cpp

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

```
    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();
    }
```

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

・**ikkeiplexing.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

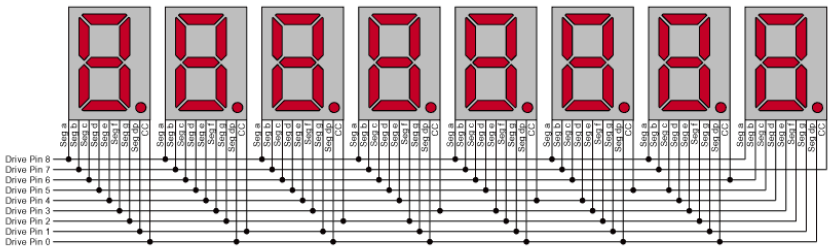
```
uint8_t iLedSign::GetSW(uint8_t edge)
{
    static uint8_t previous_swdata;
    uint8_t flag = swdata;
    if (edge != 0){
        flag &= ~previous_swdata;
        previous_swdata = swdata;
    }
    return (flag);
}
```

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

APPLICATION NOTE 1880

Charlieplexing - Reduced Pin-Count LED Display Multiplexing

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

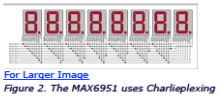
This application note discusses how to "Charlieplex" a display driver circuit to reduce pin-count. This un multiplex technique is used by the MAX6950, MAX6951, MAX6954, MAX6955, MAX6958, and MAX6959 display drivers. Charlieplexing reduces driver pin-count by using some pins alternately as cathode and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

The standard connection for multiplexing common cathode (CC) LED digits uses a separate pin for each CC connection, while the anode segment connections are communed across all the digits. Similarly, the connection for multiplexing common anode (CA) LED digits uses a separate pin for each digit's CA conn while the cathode segment connections are communed across all the digits. The number of connections can be calculated as being 1 for every digit used, plus 1 for every segment within a digit. Therefore an segment multiplex driver typified by the MAX7219 and MAX7221 CC drivers uses 8 cathode drive pins a 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 di output is actually in use. The other digit drives are high-impedance, ensuring that no drive current flow these undriven digits. By making the LED drive pins alternate duty between driving digits and segments pins can be used to drive n digits each with n-1 segments. Charlie Allen originally championed this tech internally at Maxim, and so the shorthand name "Charlieplexing" came into use to distinguish reduced multiplexing from the traditional method. The first Maxim product to use Charlieplexing is the Maxim M 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.

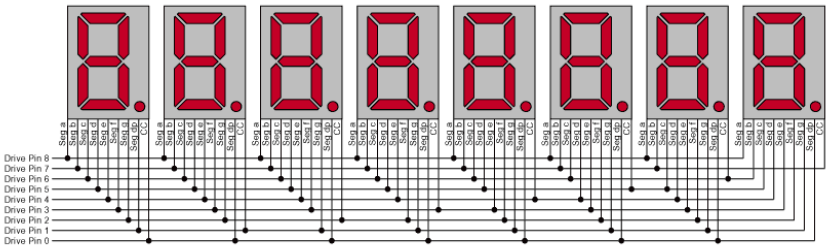
To understand how Charlieplexing works, first examine the pin connections shown in Table 1. There is

2003年2月10日

MAXIM社　AN1880

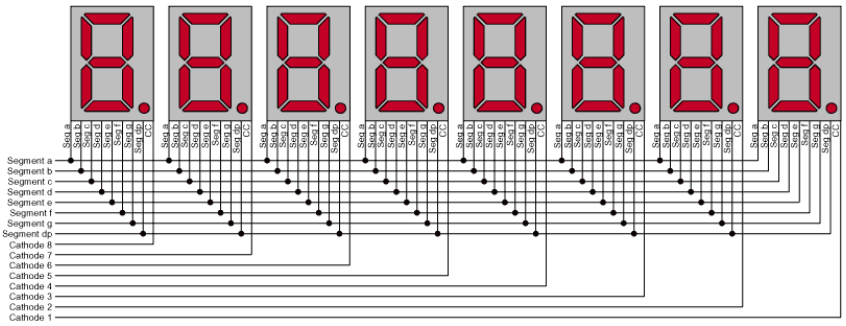
7セグメントLEDに適用した場合

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。

MAXIM社　AN1880

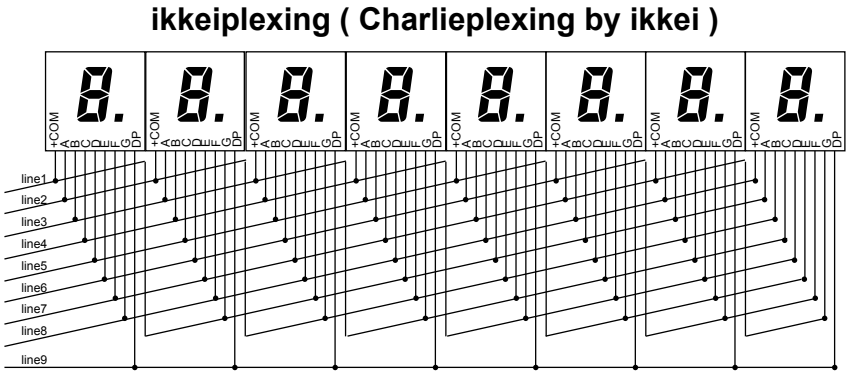
	DIG0/SEG0 Pin 6	DIG1/SEG1 Pin 5	DIG2/SEG2 Pin 4	DIG3/SEG3 Pin 3	DIG4/SEG4 Pin 14	DIG5/SEG5 Pin 13	DIG6/SEG6 Pin 12	DIG7/SEG7 Pin 11	SEG8 Pin 10
LED Digit 0	CC0	SEG dp	SEG g	SEG f	SEG e	SEG d	SEG c	SEG b	SEG a
LED Digit 1	SEG dp	CC1	SEG g	SEG f	SEG e	SEG d	SEG c	SEG b	SEG a
LED Digit 2	SEG dp	SEG g	CC2	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、これでは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、各桁毎に少しずつ異なっていた配線を全桁の配線が同じになるようにしてみました。

	line 1	line 2	line 3	line 4	line 5	line 6	line 7	line 8	line 9
LED Digit 0	CC0	SEG g	SEG f	SEG e	SEG d	SEG c	SEG b	SEG a	SEG dp
LED Digit 1	SEG a	CC1	SEG g	SEG f	SEG e	SEG d	SEG c	SEG b	SEG dp
LED Digit 2	SEG b	SEG a	CC2	SEG g	SEG f	SEG e	SEG d	SEG c	SEG dp
LED Digit 3	SEG c	SEG b	SEG a	CC3	SEG g	SEG f	SEG e	SEG d	SEG dp
LED Digit 4	SEG d	SEG c	SEG b	SEG a	CC4	SEG g	SEG f	SEG e	SEG dp
LED Digit 5	SEG e	SEG d	SEG c	SEG b	SEG a	CC5	SEG g	SEG f	SEG dp
LED Digit 6	SEG f	SEG e	SEG d	SEG c	SEG b	SEG a	CC6	SEG g	SEG dp
LED Digit 7	SEG g	SEG f	SEG e	SEG d	SEG c	SEG b	SEG a	CC7	SEG dp

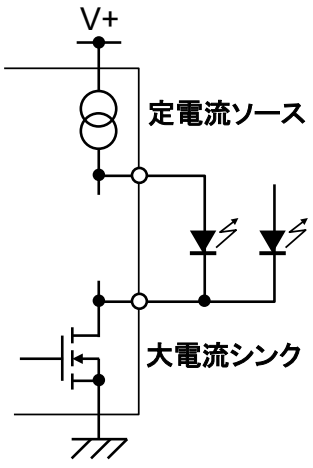
ikkeiplexing

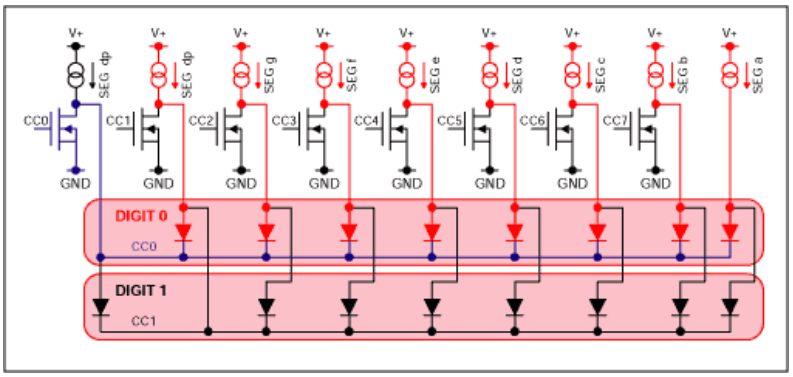
その回路図がこれです。



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。



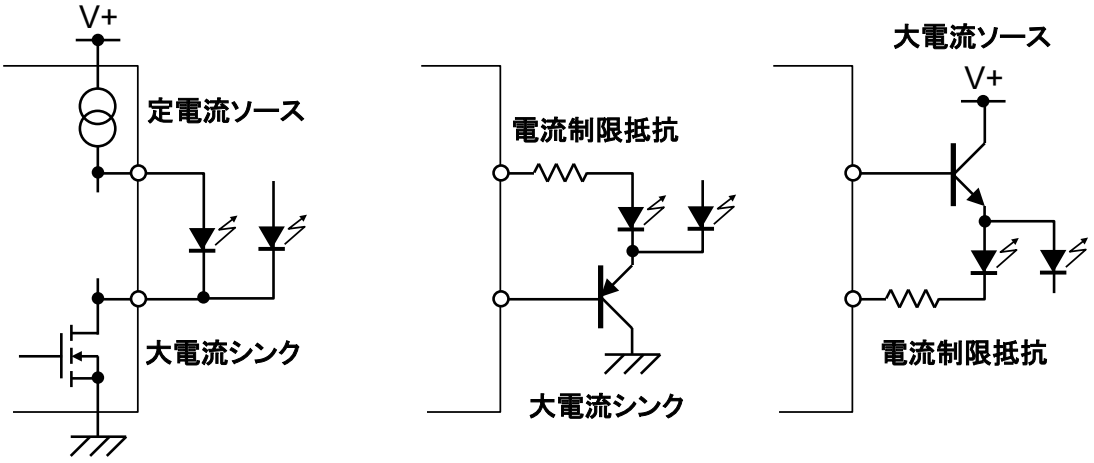


MAXIM社　AN1880

カソードコモン

電流ブーストトランジスタ

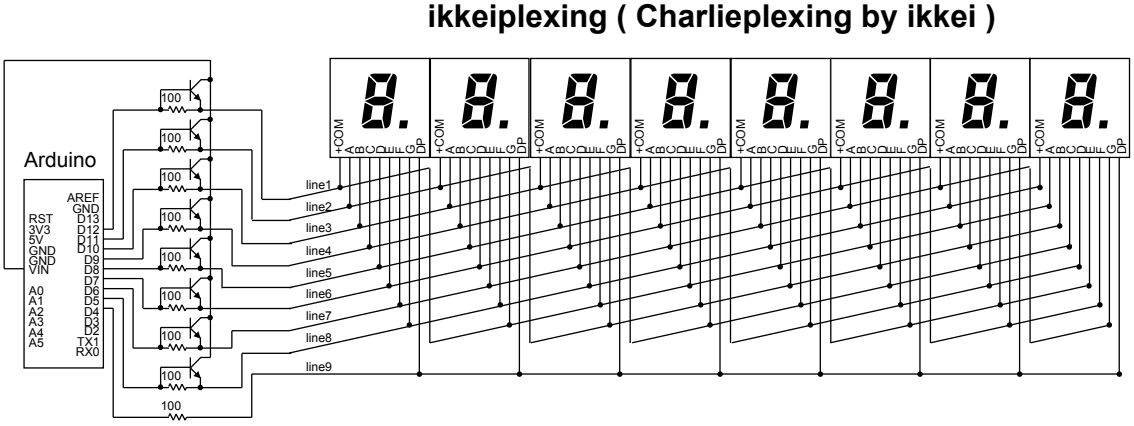
でも、一般のマイコンで駆動する場合は、そうはいきません。
そこで私は、一般のマイコンの出力につなぐために電流制限抵抗と、
コモン線をドライブするために電流をブーストするトランジスタを追加しました。
このトランジスタはエミッタフォロアなので、ベース抵抗は要りません。



カソードコモン　　カソードコモン　　アノードコモン

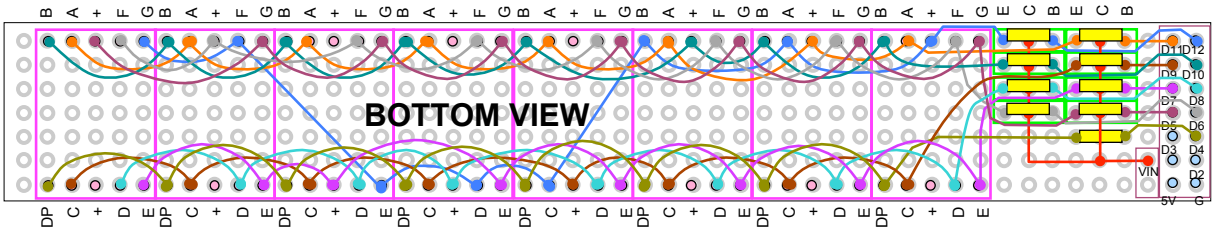
ikkeiplexing

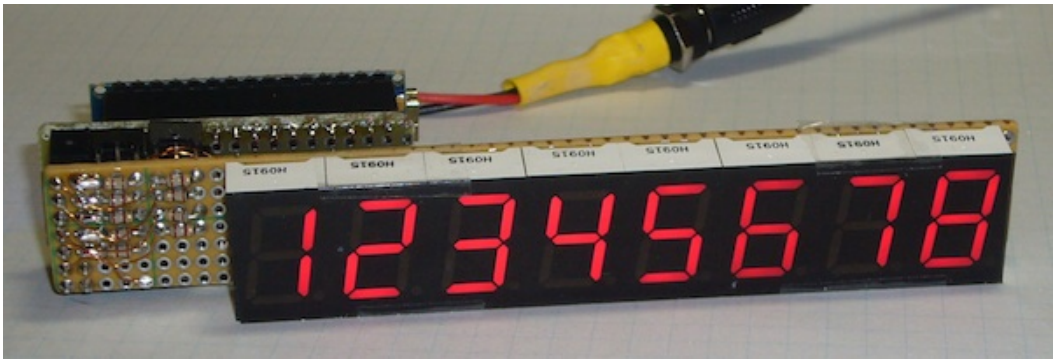
これが全体の回路です。
アノードコモンの場合だとVccより高い電圧をトランジスタのコレクタに
掛けることが出来ます。



実体配線図と実物

実体配線図と実物です。





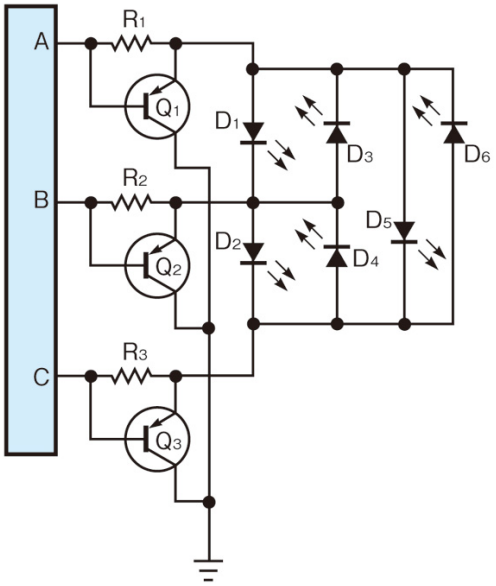
EDNの記事

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

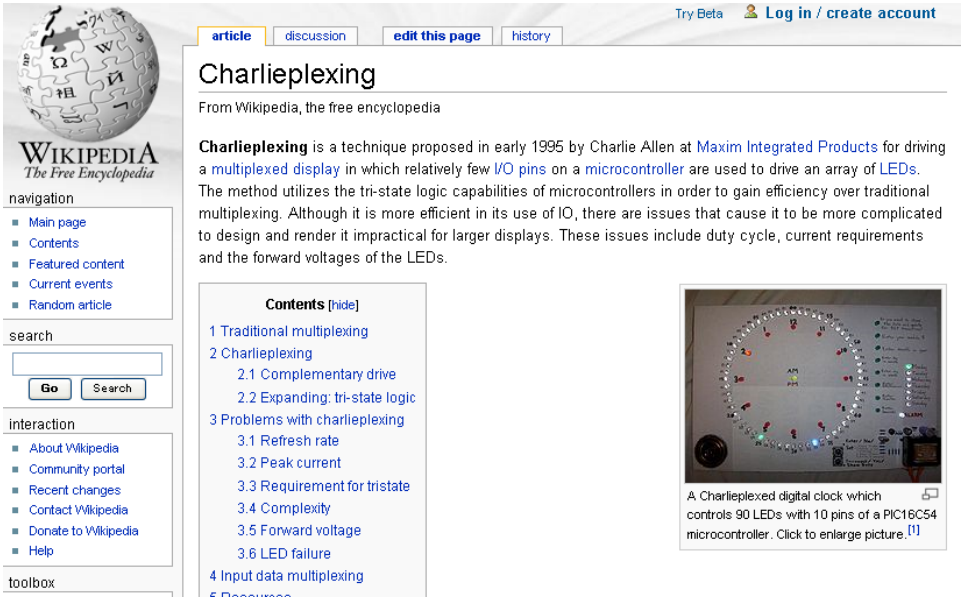
[issued: 2009年3月号]

Guillermo Jaquenod
アルゼンチン



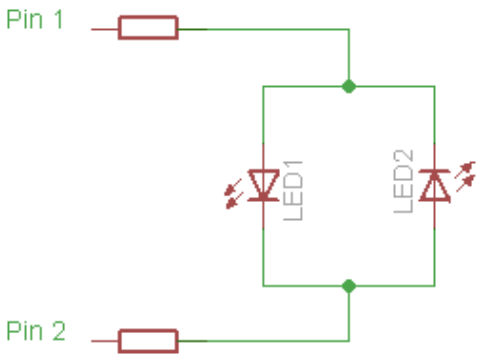
Wikipedia

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。



原理

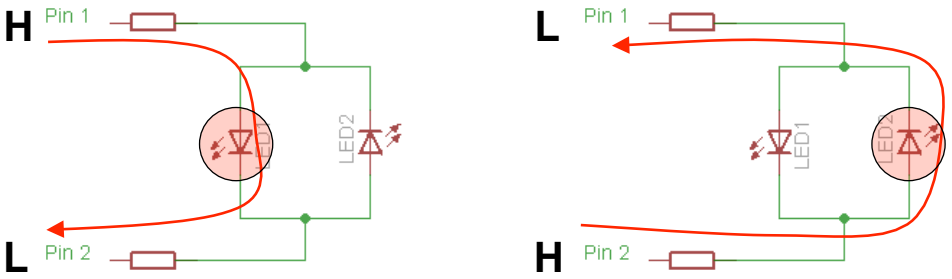
まず、LEDが2個の場合を考え、このように配線します。



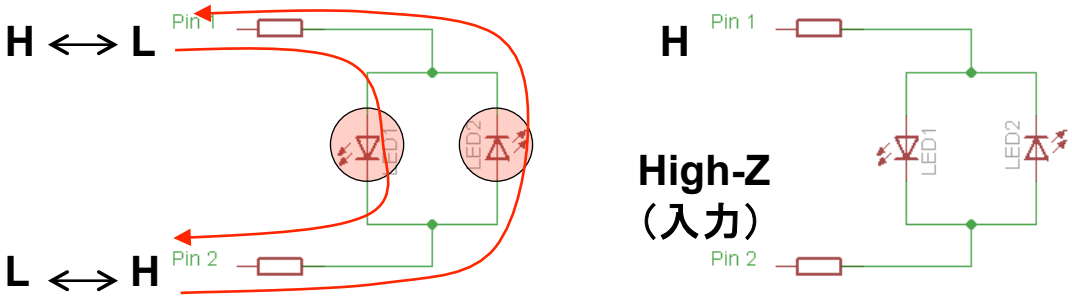
Wikipedia

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。

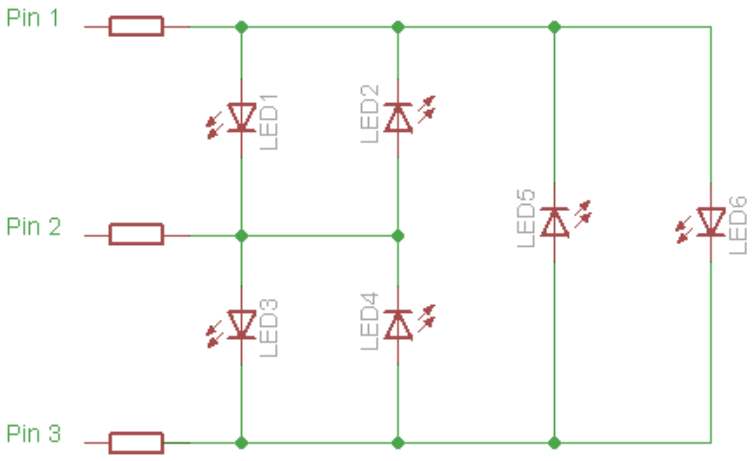


両方点灯させるには高速に切り替えます。片方をHigh-Zにすると両方消灯になります。



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。

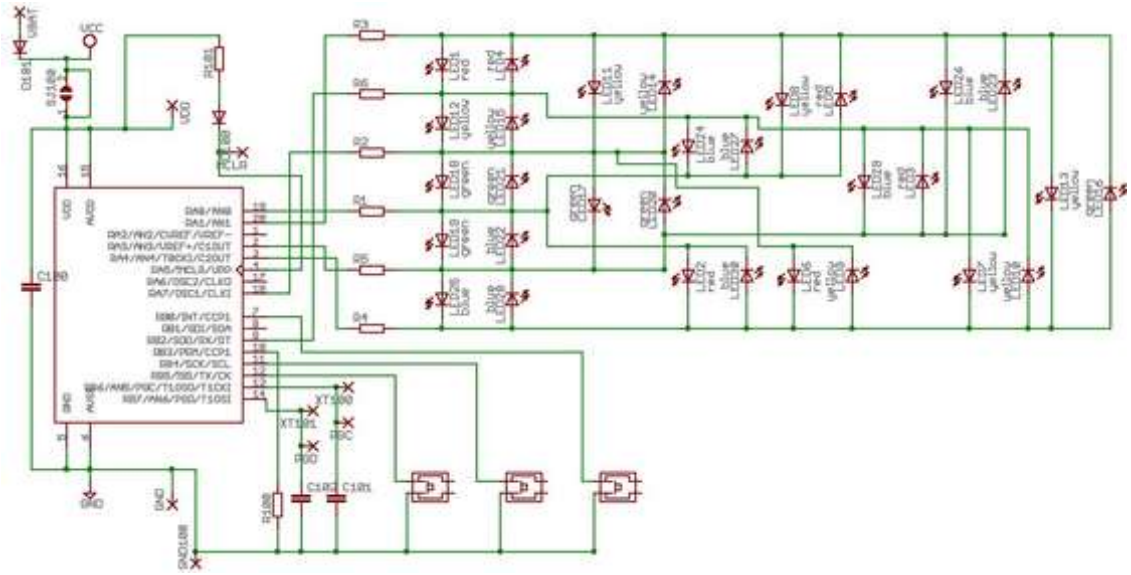


Wikipedia

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

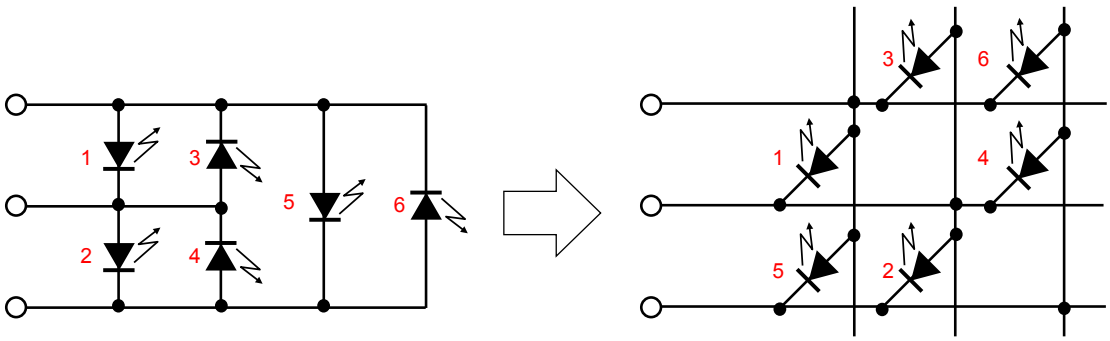
6本線では30個のLED

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

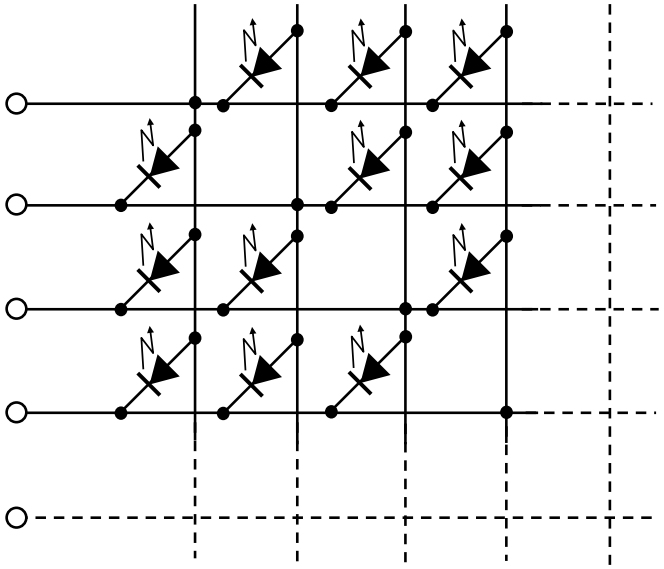
でも、
書き方を変えて、3本の場合は右の様に書けば、拡張するのは簡単です。



見え方は違うが配線は同じ

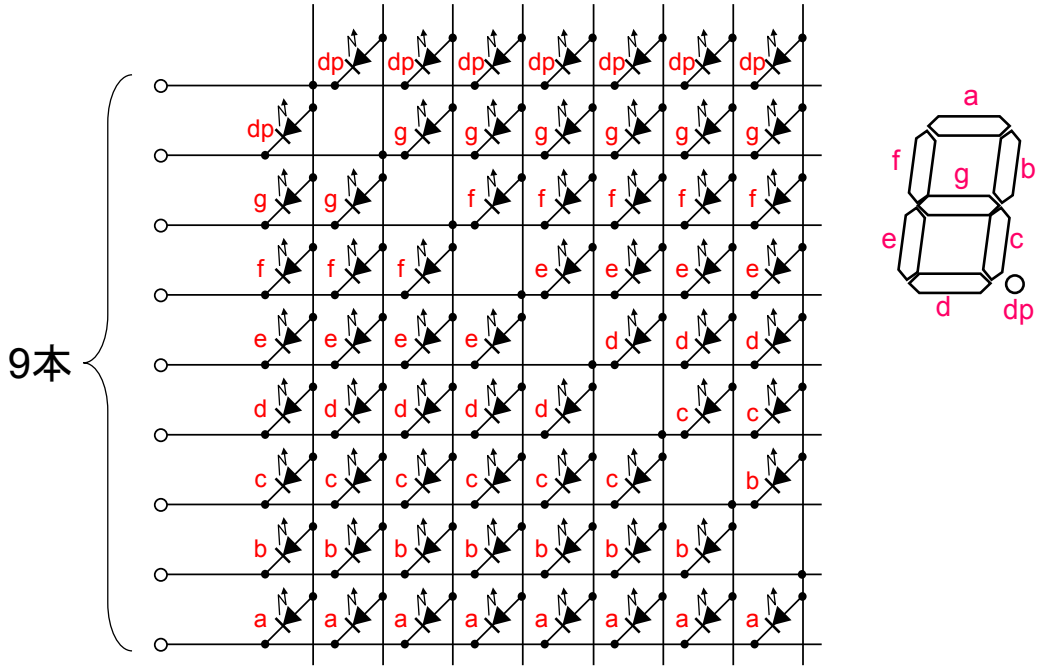
多くのLEDへ拡張

つまり、このように簡単に拡張できます。
これを9×8に拡張してみます。



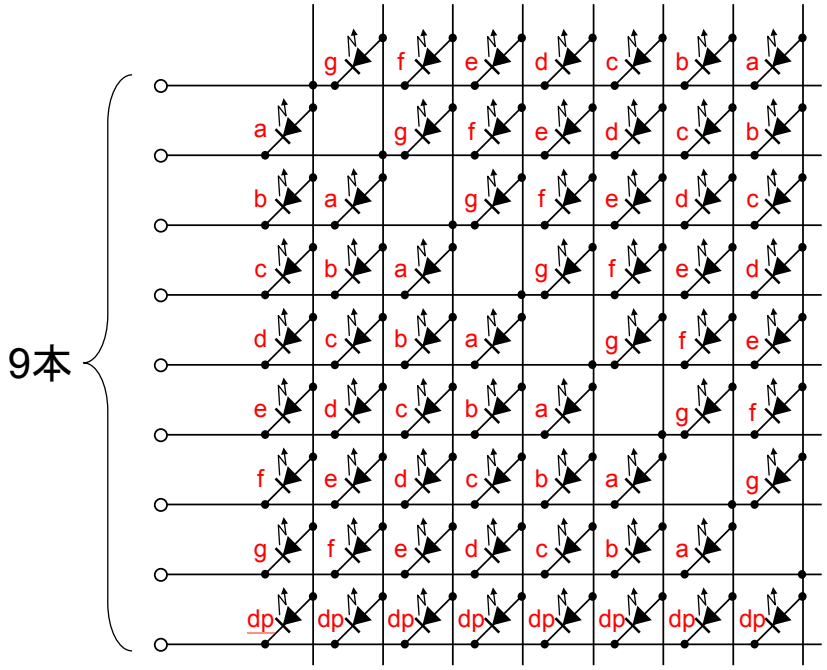
多くのLEDへ拡張

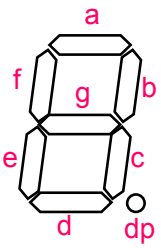
すると、始めのCharlieplexingの回路になります。



拡張の一工夫　ikkeiplexing

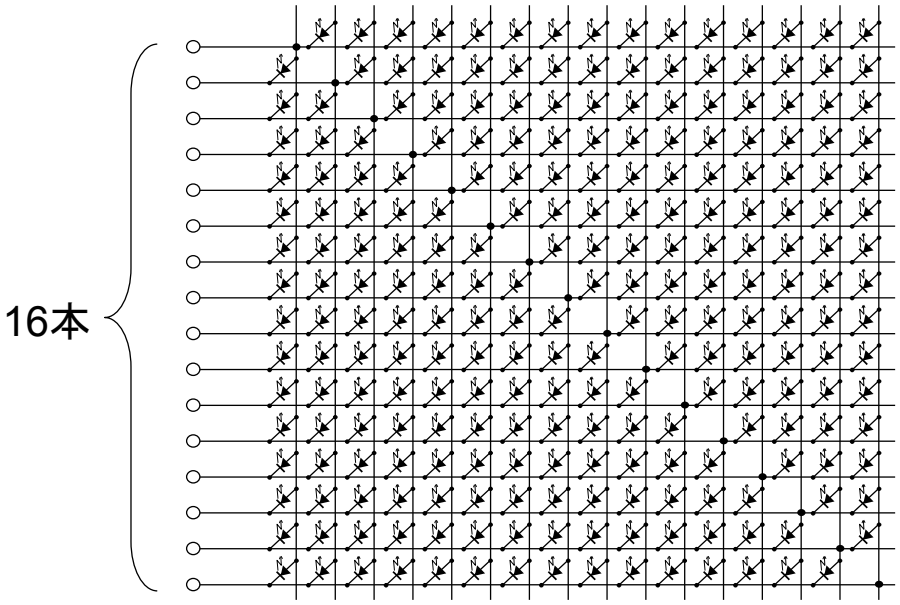
セグメントの配置を逆順にして、dpを分離し、
桁間の配線を同じにしたのがikkeiplexingというわけです。





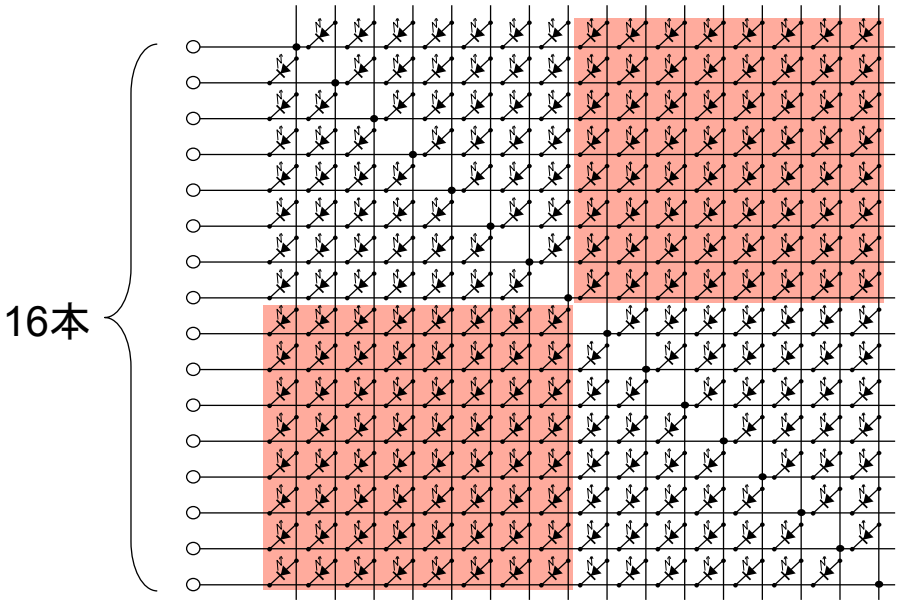
16本線の場合

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。



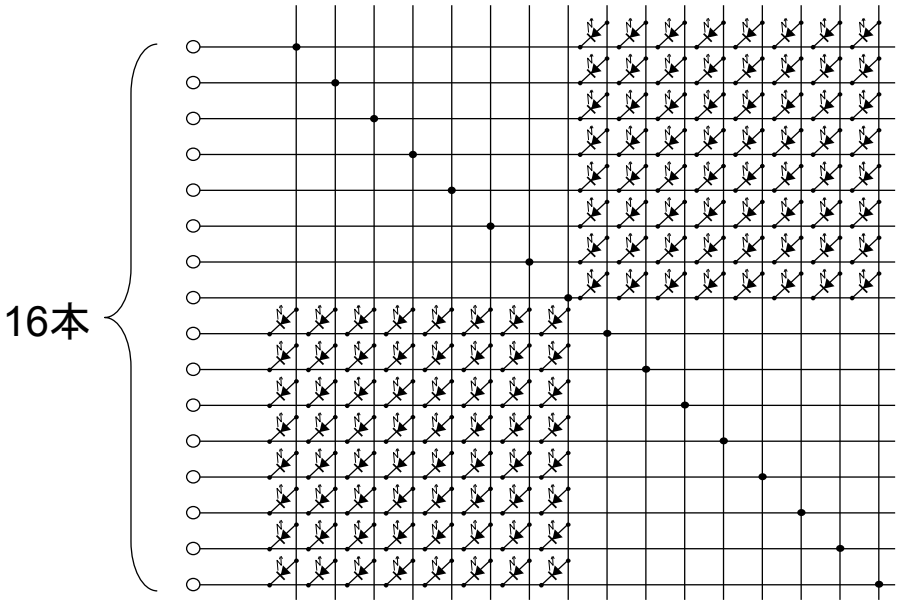
１６本線の場合

この赤い部分が8x8のマトリクスになっています。



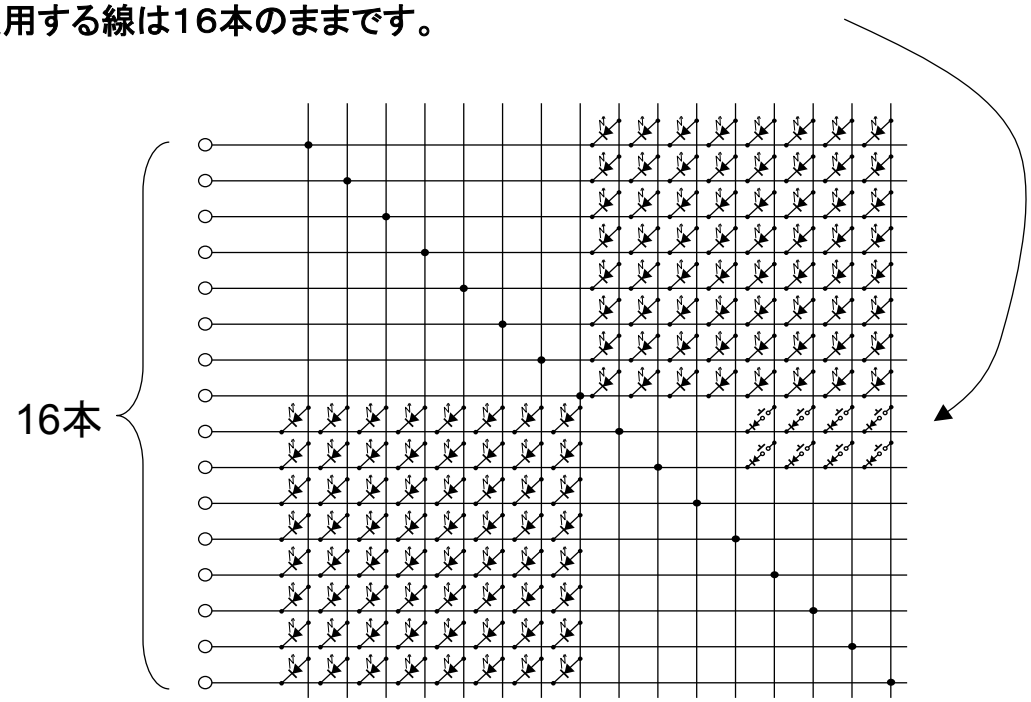
１６本線の一部分

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。

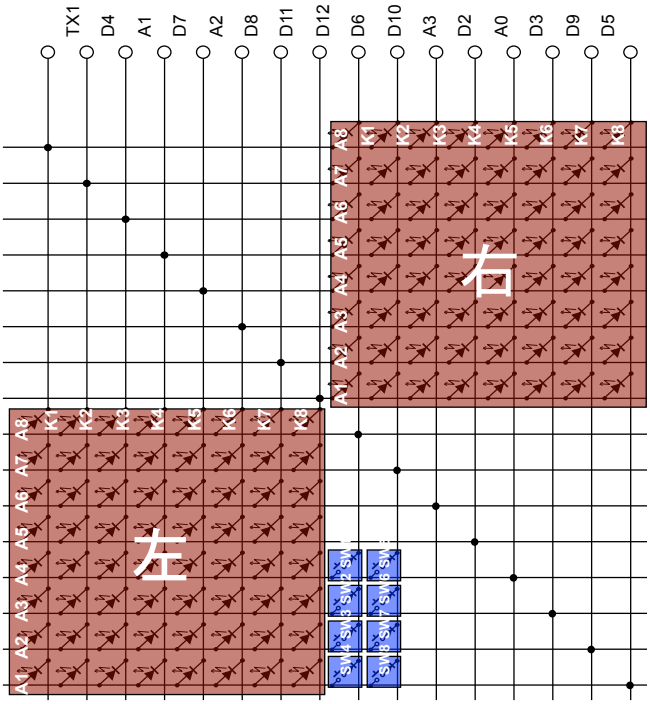


スイッチの追加

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。

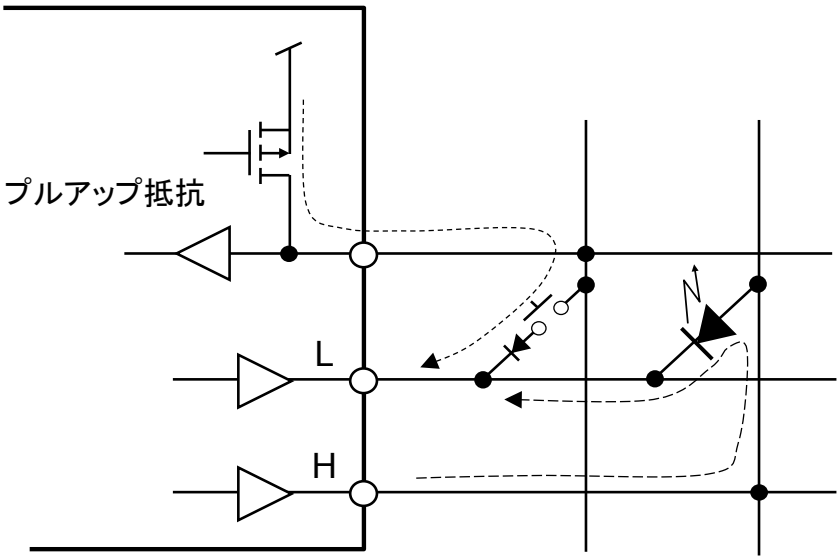


マトリクスとLED/SWの配置

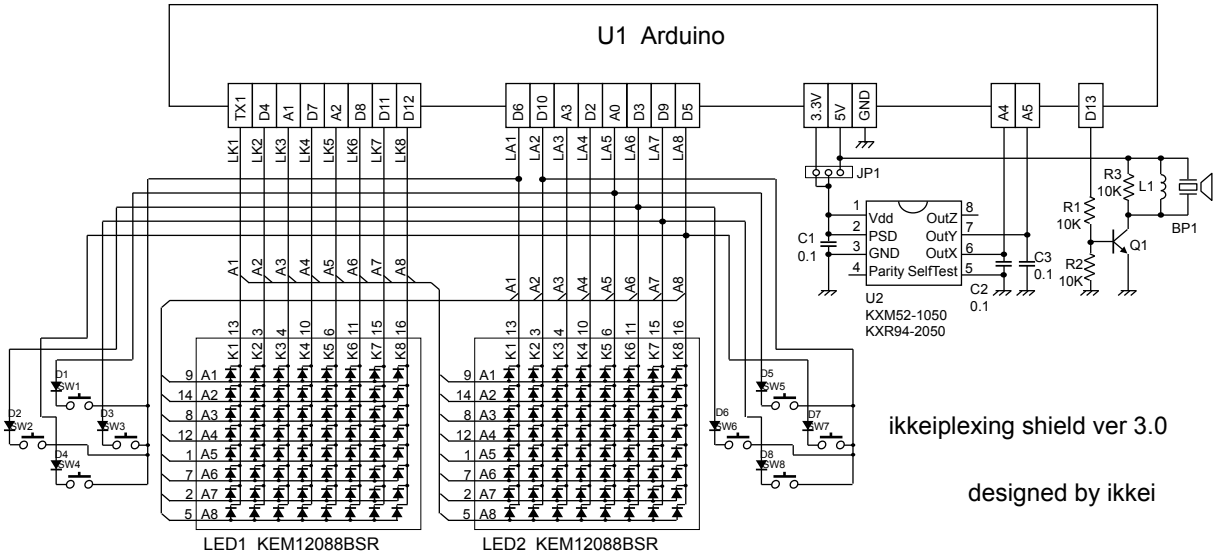


スイッチの入力方法

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 ver 3.0 回路図



お疲れ様でした

ikkei blog

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

ikkei Electronics

hhttp://ikkei.akiba.coocan.jp/ikkei_Electronics