

# 赤/橙/緑 32×16ドット ドットマトリクス LEDパネル

## 参考資料

### 概要

- ◎ 表示が明るく見やすいドットマトリクスLEDパネル(32×16ドット)です。
- ◎ 表示データを書き込むだけで簡単に使用できます。表示動作はハードウェア的に自動で行われます。表示用バッファRAM(512ビット×2面)を内蔵していますので、データの書き込み操作は表示タイミングに関係なく行えます。
- ◎ 5V動作または3.3V動作のマイコンに接続して使用できます。
- ◎ 各ドットは赤/緑の2色LEDを使用しています。点灯させる組み合わせによって赤/橙(赤と緑を同時点灯)/緑の3色の表示ができます。
- ◎ 複数枚のパネルを連結接続して使用できます。(パネル裏面に連結接続用コネクタ搭載)
- ◎ 電源電圧は5V(DC)です。(駆動部、LED駆動部の2系統)

### 定格と主な仕様

#### (1) 絶対最大定格

	記号	絶対最大定格
駆動部電源電圧	VDD	-0.3~+5.5V DC
LED部電源電圧	VLED	+5.5V DC (最大)
信号入力電圧	VIN	-0.3~ VDD + 0.3V
動作温度範囲	Topr	-10~+60°C (結露なきこと)
保存温度範囲	Tstg	-25~+85°C (結露なきこと)

#### (2) 主な仕様

	記号	仕様値
表示色		赤/橙/緑
表示ドット数		32(よこ)×16(たて) ドット
表示面寸法		192(よこ)×96(たて) mm
デューティ比		1/16
スキャン周波数		330Hz (自動スキャン)
バッファRAM		512ビット(32ビット×16ワード)×2面
駆動部電源電圧	VDD	5V DC
LED部電源電圧	VLED	5V DC
LED部消費電流	ILED	最大約1.8A (全ドット点灯時※)
クロック周波数	fCLK	最大20MHz (単体動作時)
その他		M3ねじで取り付け可能

※ クロック周波数は単体動作時(パネル1枚のみ使用時)の最大値です。  
複数枚のパネルを連結接続して使用するときはこの値より若干低くなります。

オーディオ・マイコン・メカトロ・電子パーツ

年中無休・営業時間:AM10:00~PM8:00

**デジタル**

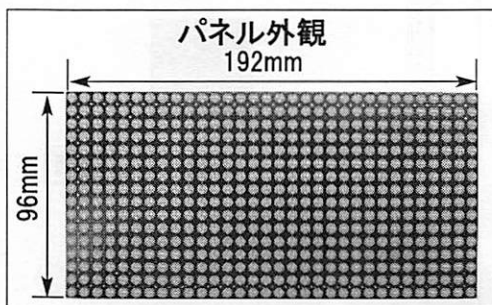
〒556-0005 大阪市浪速区日本橋4-6-7

TEL) 06-6644-4555 / FAX) 06-6644-1744

E-MAIL) http://digitryohitsu.com

Blog) http://blog.digit-parts.com (Twitter) @0666444555

電子工作向けの学習、実験、開発向けであり、資料等は参考用です。目安程度のもので差異や誤りがある場合があります。商品の性能等を保証するものではありません。各種設定、使用については自己責任でお願いいたします。いかなる事故、損失においても製造者、流通者、販売者は一切の責任を負いかねます。返品、交換、保証等の対応はしていません。



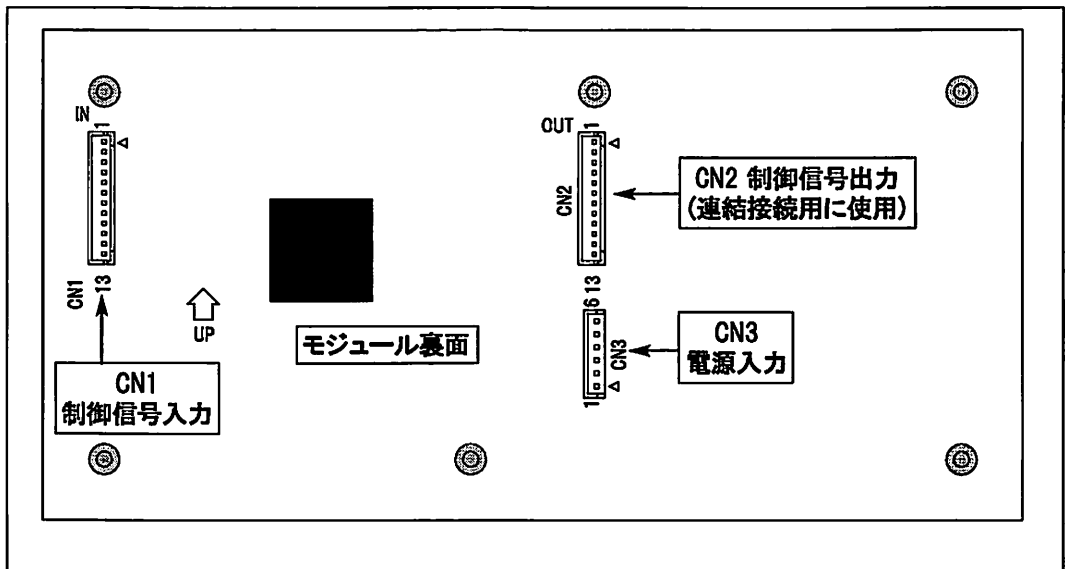
R G /  
0 0 -  
0 1 G  
1 0 R  
1 1 Orange

※ LED部消費電流は下記条件による  
おおよその実測値です。

- ◎ 周囲温度: 27°C
- ◎ 駆動部電源電圧: 5V DC
- ◎ LED部電源電圧: 5V DC
- ◎ 赤/緑ともに全ドット点灯

LED部消費電流以外の仕様はデータ  
シート上の値です。

## コネクタのピン配置



## CN1 制御信号入力 (PH12ピン)

ピン番号	信号名	機能概要
1	SE In	内部バッファRAM 切り替えモード選択 「H」:手動切り替え / 「L」:自動切り替え
2	A/BB In	RAM-A/RAM-B選択 (SE In = 「H」のとき有効) 「H」:RAM-Aを選択 / 「L」:RAM-Bを選択
3	A3 In	内部バッファRAMアドレス (データを書き込む行のアドレス)指定
4	A2 In	
5	A1 In	
6	A0 In	
7	Vss	駆動部 グラウンド
8	DG In	緑表示データ入力 (「H」:0 N/「L」:0 FF) 1行分のデータを上位ビットから順に入力
9	CLK In	データ取り込み用クロック入力 立ち上がりエッジでデータを取り込み
10	WE In	内部バッファRAMへの書き込み信号 (ALE In = 「H」のとき有効)
11	DR In	赤表示データ入力 (「H」:0 N/「L」:0 FF) 1行分のデータを上位ビットから順に入力
12	ALE In	アドレスラッチ有効 「H」でアドレスを取り込み

## CN3 電源入力 (EH6ピン)

ピン番号	信号名	機能概要
1	Vdd	駆動部 電源 (+5V DC)
2	GND	LED部 グラウンド
3	GND	
4	VLED	LED部 電源 (+5V DC 1.8A max.) ※ 消費電流は全点灯時のおよその実測値
5	VLED	
6	Vss	駆動部 グラウンド

## CN2 制御信号出力 (PH12ピン) 3.3V

ピン番号	信号名	機能概要
1	SE out	内部バッファRAM 自動切り替えモード出力
2	A/BB out	RAM-A/RAM-B選択信号出力
3	A3 out	内部バッファRAMアドレス出力
4	A2 out	
5	A1 out	
6	A0 out	
7	Vss	制御部 グラウンド
8	DG out	緑表示データ出力
9	CLK out	データ取り込み用クロック出力
10	WE out	内部バッファRAMへの書き込み信号出力
11	DR out	赤表示データ出力
12	ALE out	アドレスラッチ有効信号出力

※ CN2(制御信号出力)のロジックレベルは3.3V系 CMOSレベルです。

※CN1(制御信号入力)のCLK in信号の周波数は、最大20MHz(パネル1枚で使用する時)ですが、複数枚のパネルを連結して使用する場合はデータが正しく取り込めるよう、周波数を下げてください。

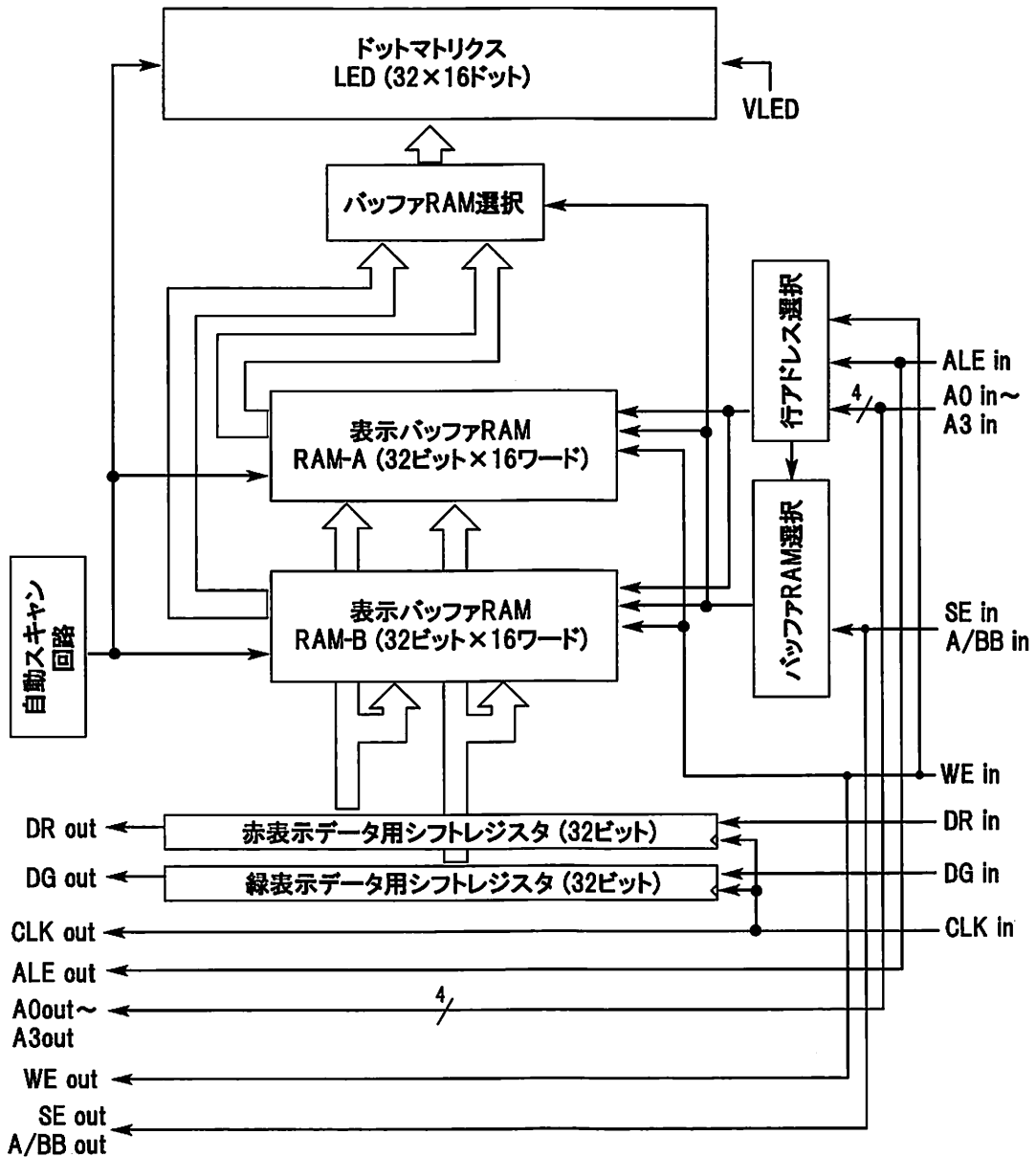
※データを送る必要のないときはCLK in信号を止めてもかまいません。(表示動作はパネル内部のクロックで自動的に行われます)

※CN3の2番/3番ピン(GND)、4番/5番ピン(VLED)はそれぞれ外部で並列に接続して使用してください  
(並列にしないと、コネクタの電流定格を超えることがあります)

## 内部ブロックダイアグラム(※独自調べによる)

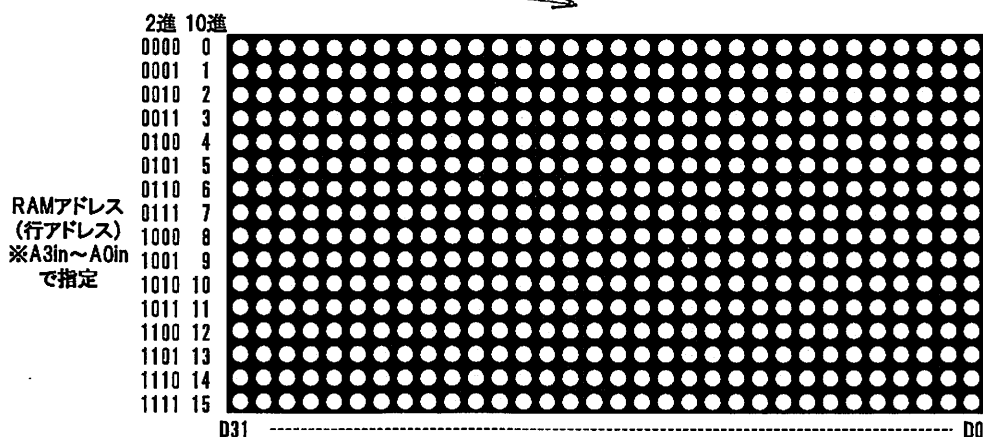
本LEDパネルの内部ブロックダイアグラム(※独自調べによる)は下図の通りです。

**!** このブロックダイアグラムは実際にパネルをマイコンでコントロールし、動作させて調べた独自調べのものです。  
 現物パネルの内部構造と異なる箇所がある可能性があります。(現物パネルの内部構造は公開されていないため、不明です)  
 マイコンでコントロールする際の考え方の参考として活用してください。

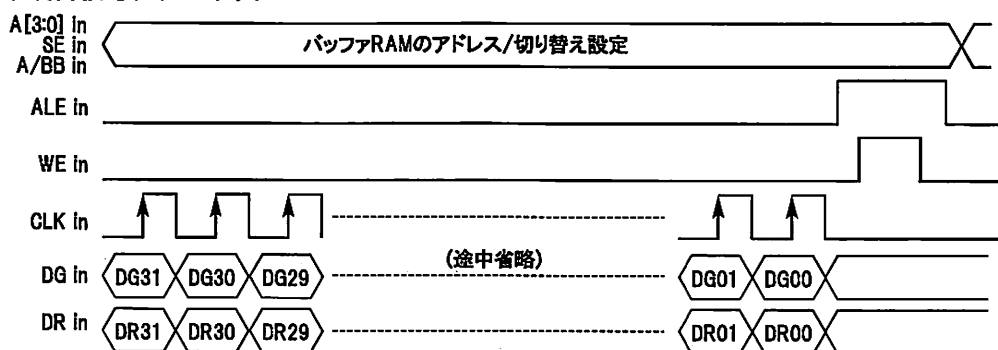


## 表示データの書き込みかた

## (1) ドット位置とアドレスの関係



## (2) 制御信号タイミング図



1行分(32ビット)の緑表示データ(DG31~DG00)と赤表示データ(DR31~DR00)をCLK in信号に合わせて送ります。DG in/DR inにセットしたデータはCLK inの立ち上がりエッジで内部シフトレジスタに取り込まれます。

最初に送った表示データ(DG31、DR31)がパネルの一番左端の列に対応します。

A3 in	A2 in	A1 in	A0 in	行アドレス
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15 (※)

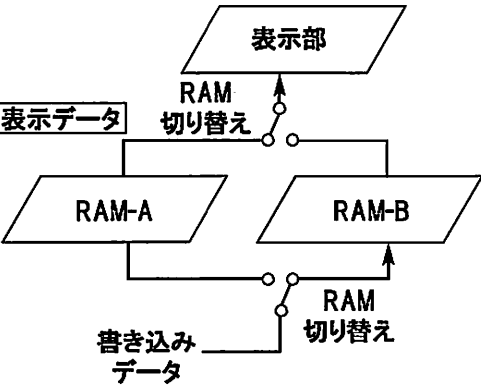
1行分(32ビット)のデータを送り終わったら、ALE inを「H」にして書き込み先のバッファRAMの行アドレス(A3 in~A0 in)をセットし、WE inを「H」にして送り込んだデータを内部バッファRAMに書き込みます。(書き込み信号(WE in)はアドレスラッチ信号(ALE in)が「H」のときのみ有効です。)

A3 in~A0 inの値と行アドレスの関係は左表の通りです。

※バッファRAMの15番地にデータを書き込むと、RAM自動切り替え機能が有効にしてある場合(SE in = 「L」)は、バッファRAMのページが自動的に切り替わり、書き込んだデータが表示に反映されます。  
(詳細は次のページを参照してください)

(3) バッファRAM自動切り替え機能

本パネルには512ビット(32ビット×16ワード)のバッファRAMが2面(RAM-AとRAM-B)搭載されています。一方のRAMに書き込み中は他方のRAMの内容が表示されるようになっていいますので、表示中にRAMのデータを書き換えても表示がちらつきません。



左図のように、RAM-Aにデータを書き込み中のときはRAM-Bの内容が表示に反映され、RAM-Bにデータを書き込み中のときはRAM-Aの内容が表示に反映されます。  
(※パワーONリセット機能により、パワーON後バッファRAMを切り替えないかぎり表示はスタートしません)

RAM-AとRAM-Bのどちらにデータを書き込むかは手動でも選択できます(SE in=「H」のとき)が、自動切り替え機能を使用するとバッファRAMを意識せずにデータの書き換えが行えます。

バッファRAM自動切り替え機能は、SE in信号を「L」にしたとき有効になります。


※SE in信号とA/BB in信号は、データをメモリに書き込む時点で確定している必要があります。

◎ SE in = 「L」の場合(メモリ自動切り替えモード)

バッファRAMの15番地にデータを書き込むと、データを書き込んだ時点でバッファRAMが自動的に切り替わり、書き込んだデータが表示に反映されます。

	15番地への書き込み前		書き込み後
RAM-A	書き込み	→	表示
RAM-B	表示	→	書き込み
RAM-A	表示	→	書き込み
RAM-B	書き込み	→	表示

メモリ自動切り替えモードを使用することで、RAM-AとRAM-Bの切り替えを意識せずに表示データを書き込み、簡単に表示できます。

 15番地にデータを書き込まないとデータが表示に反映されません

◎ SE in = 「H」の場合(メモリ手動切り替えモード)

RAM-AとRAM-Bのどちらにデータを書き込むかを、アドレス指定時にA/BB in信号で選択します。RAM-Aを選択したときはRAM-Bのデータが、RAM-Bを選択したときはRAM-Aのデータが表示に反映されます。

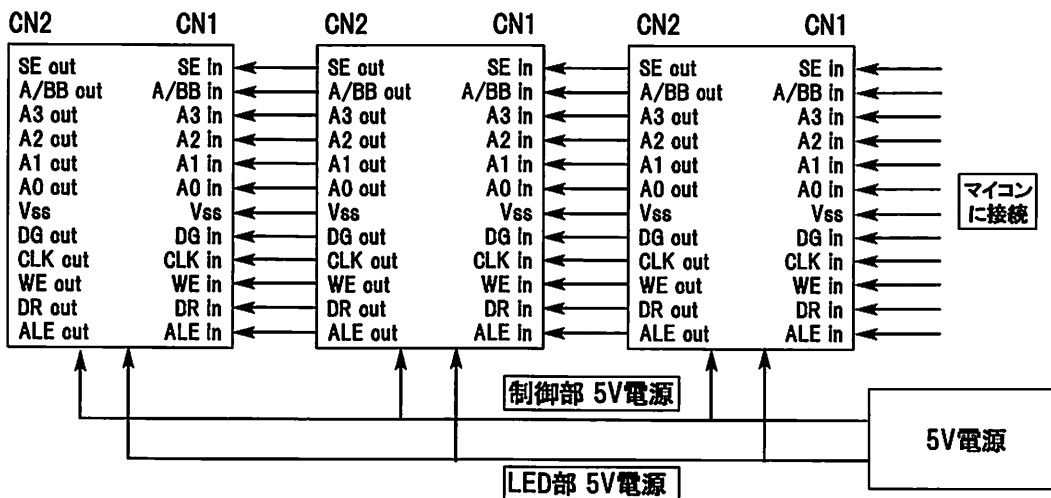
	A/BB in = 「H」	A/BB in = 「L」
RAM-A	書き込み	表示
RAM-B	表示	書き込み

手動切り替えはデータを一部分だけ書き換えたいときや、表示を素早く切り替えたいときに便利です。

## パネルを連結接続する場合

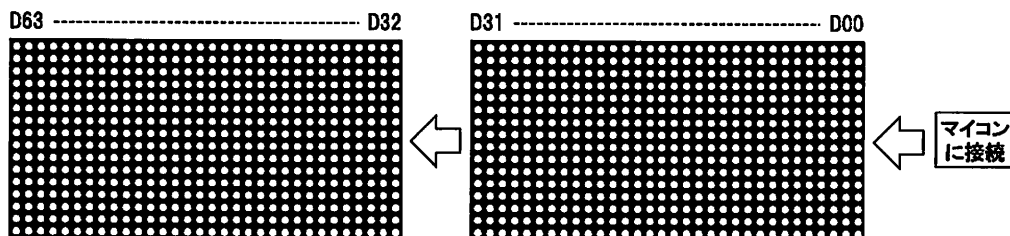
本ドットマトリクスLEDパネルは、複数枚連結接続することで、表示面積を簡単に増やすことができます。複数枚のパネルを連結して使用する場合は、PHR12-PHR12ストレートケーブルを使用して、下図のように接続します。

(図は表示面から見た状態で描いてあります)



**!** LED部の5V電源には十分余裕のあるものを使用してください  
(LEDを全部点灯した状態で1枚あたり最大1.8A消費します)

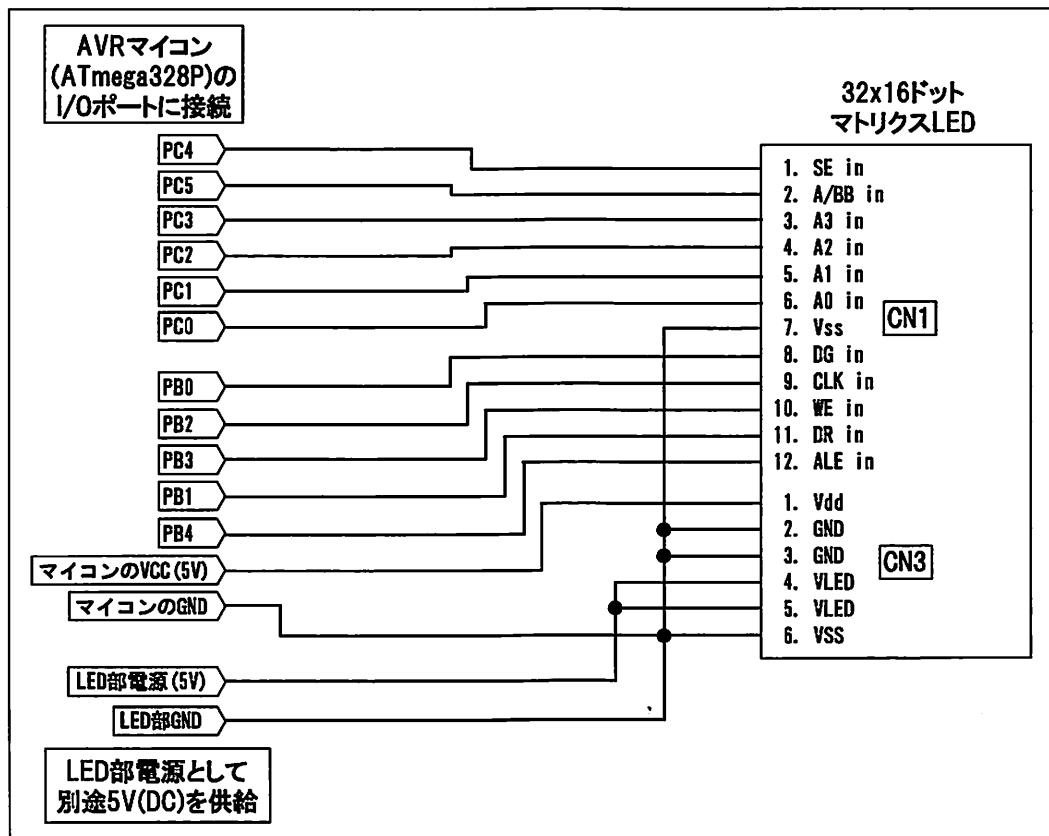
複数枚連結した場合、連結した枚数分の1行分のデータ(2枚なら64ビット、3枚なら96ビット)を送ってからアドレスを指定して書き込みを行います。



複数枚連結した場合でも、一番最初に送ったビットが一番左端の表示に対応します。

## 接続のしかた (AVRマイコン ATmega328Pとの接続例)

下の接続図は、制御用のマイコンとしてAVRマイコン(ATmega328P)を使用した場合の接続例です。



## 表示データの送り方

- ◎ 以下のプログラムはAtmel Studio 6上のGCCで作成しています。動作は確認していますが、プログラムの動作について保証はいたしかねます。実験用の参考として使用してください。
- ◎ プログラム中の「CLK」「DG」「DR」などの名前はプログラム冒頭で定義してあります。サンプルプログラムの全リストについては12ページを参照してください。
- ◎ 使用しているマイコンはATmega328Pです。(クロックは内部RC発振、8MHzに設定してください)

次ページのサブルーチンプログラムは、配列に入れた1行分の赤表示データと緑表示データをドットマトリクスLEDパネルに転送し、行アドレスを指定してパネル内部のバッファRAMに書き込むプログラムです。

使用するパネルの枚数は、「#define NUM\_PANEL」文で指定します。

このサブルーチン呼び出す前に、1行分の表示データ(赤表示データと緑表示データ)を入れた配列(32ビット符号なし整数型)を用意してください。※配列の要素数はパネルの枚数分必要です。

```

/* データを1行分書き込む
void send_data(int iaddr_y, unsigned long *idata_r, unsigned long *idata_g);
iaddr_y : データを書き込む行アドレス (0~15)
*idata_r : 赤表示データ (符号なし32ビット整数)の配列先頭要素のアドレス
*idata_g : 緑表示データ (符号なし32ビット整数)の配列先頭要素のアドレス
※表示データの配列の要素数はパネルの枚数と同じです。
*/

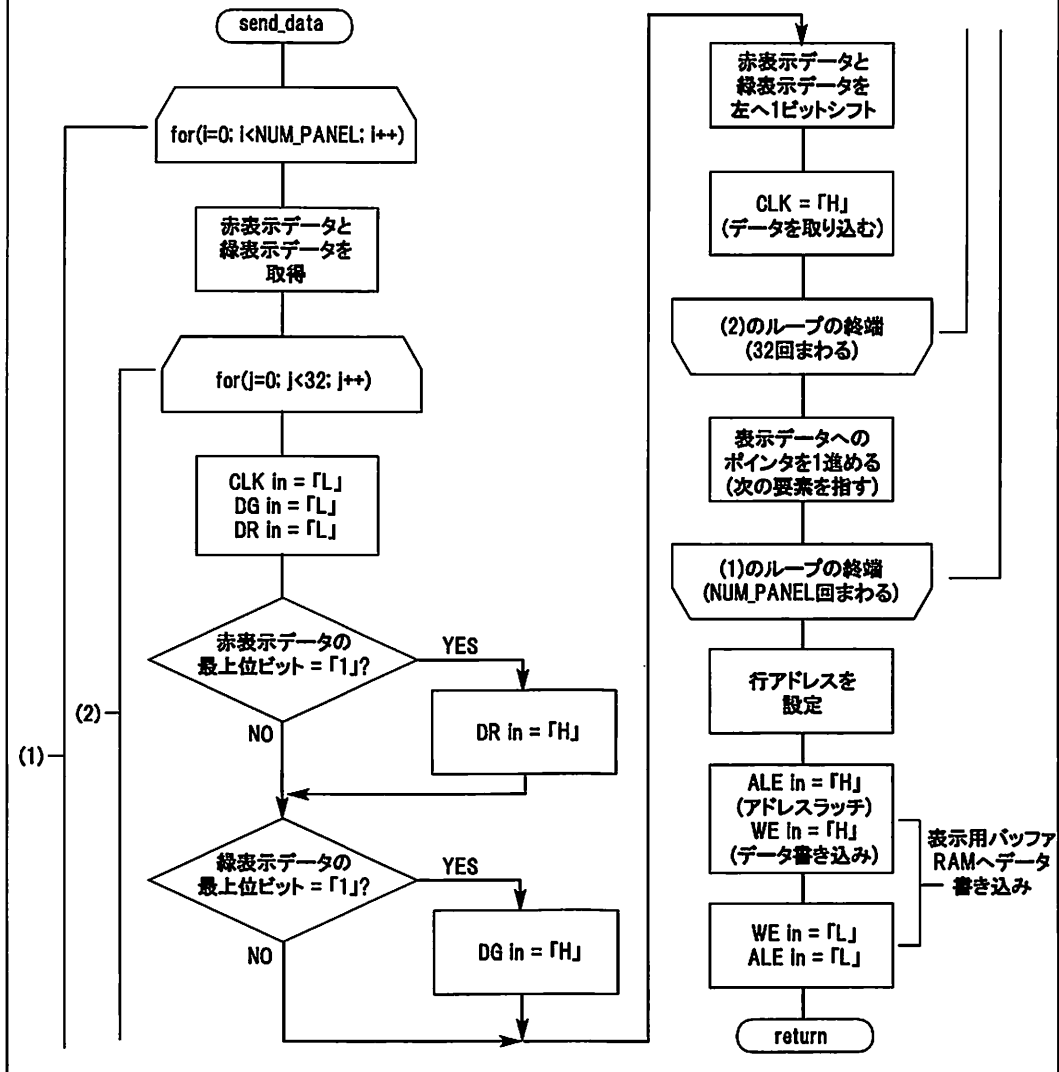
#define NUM_PANEL 1          //パネルの枚数(1枚)

void send_data(int iaddr_y, unsigned long *idata_r, unsigned long *idata_g);
void send_data(int iaddr_y, unsigned long *idata_r, unsigned long *idata_g)
{
    volatile int i, j;
    unsigned long m, n;
    for (i=0; i<NUM_PANEL; i++)
    {
        m = *idata_r;          // 配列に入った表示データを取得
        n = *idata_g;
        /* 32ビットの表示データを、下位ビットから順番に送る */
        for (j=0; j<32; j++)
        {
            PORTB = PORTB & ~(1<<CLK);          // CLK = L
            PORTB = PORTB & ~((1<<DR)|(1<<DG));    // データビット(DR/DG)をセット
            if ((m & 0x80000000UL) != 0)
            {
                PORTB = PORTB | (1<<DR);
            }
            if ((n & 0x80000000UL) != 0)
            {
                PORTB = PORTB | (1<<DG);
            }
            m = m <<1;
            n = n <<1;
            /* シフトクロックを送る(立ち上がりエッジで取り込み) */
            PORTB = PORTB | (1<<CLK);          // CLK = H
        }
        idata_r = idata_r + 1;                  // 次の配列データを指すようにする
        idata_g = idata_g + 1;
    }
    /* 書き込むRAMのアドレスをセットする */
    PORTC = PORTC & 0b11110000;
    PORTC = PORTC | (iaddr_y & 0x0f);          // PC[3:0] = A[3:0]
    PORTB = PORTB | (1<<ALE);                  // ALE = Hでアドレスセット
    /* 書き込み信号を与える */
    PORTB = PORTB | (1<<WE);                  // WE = Hでデータ書き込み
    PORTB = PORTB & ~(1<<WE);                 // WE = L
    PORTB = PORTB & ~(1<<ALE);                 // ALE = L
}

```



## フローチャート



## 1画面分の表示を行わせる

前ページのsend\_data()サブルーチンプログラムを使用して、1画面分の表示を行わせます。プログラムリストは次のページを見てください。

send\_data()サブルーチンプログラムを16回繰り返し呼び出し、一番上の行(0番地)から順番に表示データを書き込みます。プログラムを簡単にするため、SEin信号を「L」レベルに固定してバッファRAM自動切り替え機能を有効にしていますので、一番下の行(15番地)にデータを書き込んだ時点で、書き込んだデータが表示に自動的に反映されます。

(このサンプルプログラムでは、データを書き込むと「快速」の文字が橙色で表示されます)

```

/* 電光掲示板 フォントのデータ */
const PROGMEM unsigned int font_tbl_r[2][16] =
{
    {0x0000, 0x2040, 0x2844, 0x2BFC, 0xA844, 0xA844, 0xA044, 0xA7FE,
     0xA040, 0xA040, 0x20A0, 0x20A0, 0x2110, 0x2208, 0x2406, 0x0000}, //快
    {0x0000, 0x6044, 0x37FC, 0x0040, 0x07FC, 0x0444, 0xE444, 0x27FC,
     0x2140, 0x2260, 0x2450, 0x2848, 0x2044, 0x5000, 0x8FFE, 0x0000}, //遅
};

const PROGMEM unsigned int font_tbl_g[2][16] =
{
    {0x0000, 0x2040, 0x2844, 0x2BFC, 0xA844, 0xA844, 0xA044, 0xA7FE,
     0xA040, 0xA040, 0x20A0, 0x20A0, 0x2110, 0x2208, 0x2406, 0x0000}, //快
    {0x0000, 0x6044, 0x37FC, 0x0040, 0x07FC, 0x0444, 0xE444, 0x27FC,
     0x2140, 0x2260, 0x2450, 0x2848, 0x2044, 0x5000, 0x8FFE, 0x0000}, //遅
};

/* 表示用バッファメモリ */
unsigned int disp_buf_r[2][16];
unsigned int disp_buf_g[2][16];

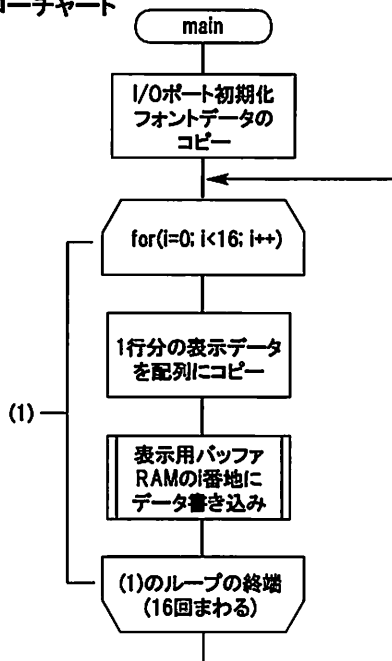
#define NUM_PANEL 1 //パネルの枚数(1枚)

int main(void)
{
    int i;
    // 32ビット符号なし整数型の配列をパネルの枚数分用意
    unsigned long m[NUM_PANEL], n[NUM_PANEL];
    // I/Oポートの初期化
    PORTB = 0b00000000;

    _delay_ms(100); // パネルのパワーONリセット待ち
    while(1)
    {
        /* フォントデータをRAM上にコピー */
        for (i=0; i<16; i++)
        {
            disp_buf_r[1][i] = pgm_read_word(&font_tbl_r[0][i]);
            disp_buf_g[1][i] = pgm_read_word(&font_tbl_g[0][i]);
            disp_buf_r[0][i] = pgm_read_word(&font_tbl_r[1][i]);
            disp_buf_g[0][i] = pgm_read_word(&font_tbl_g[1][i]);
        }
        for (i=0; i<16; i++) //1画面分の表示データ転送
        {
            m[0] = disp_buf_r[1][i];
            m[0] = (m[0] <<16) + disp_buf_r[0][i];
            n[0] = disp_buf_g[1][i];
            n[0] = (n[0] <<16) + disp_buf_g[0][i];
            send_data(i, &m, &n); // 1行分の表示データを書き込む
        }
    }
}

```

## フローチャート



表示するフォントのデータはプログラムメモリ上に置いてありますので、一度データSRAM領域にコピーしたあと、コピーしたものを表示データとして使用しています。

(1)のループの16回目のデータ書き込み(i = 15)が終わった時点で、表示バッファRAMに書き込んだデータが表示に反映されます。

※このプログラムでは表示データは固定ですが、(1)のループをまわる度にデータを変えることで、表示を変化させることができます。

## (参考) 表示バッファRAMの手動切り替え

このプログラムでは使用していません(表示バッファRAMの自動切り替え機能を有効にしています)が、表示バッファRAMを手動で切り替えるときは、SE\_in信号を「H」にした状態で次のサブルーチンプログラムを呼び出すことで、表示バッファRAMをプログラムから切り替えることができます。

send\_data()サブルーチンプログラムを呼び出して表示バッファRAMへデータを書き込んだあと、次のサブルーチンプログラムを呼び出して表示バッファRAMを切り替えると、書き込んだデータが表示に反映されます。

```

/* 書き込み先RAMの選択 */
#define RAMA 0
#define RAMB 1

/* 表示用バッファRAMの選択
   バッファRAM手動切り替え時に使用
   (このプログラムでは使用していません)
   void select_ram(int iram);
   iram = 0のときRAM-Aに書き込み
   iram = 1のときRAM-Bに書き込み
*/
void select_ram(int iram);
void select_ram(int iram)
{
    if(iram == RAMA)
    {
        PORTC = PORTC | (1<<A_BB);
    }
    else
    {
        PORTC = PORTC & ~(1<<A_BB);
    }
}

```

## プログラム例 (ATmega328P用)

この説明書で解説した表示サンプルプログラム(ATmega328P用)の全リストは次の通りです。  
ドットマトリクスLEDの接続については、プログラム冒頭のコメント欄にI/Oポートの割り当て表がありますので、それに従ってください。

```

/*
 * Denkou_sample_140916.c
 * 16x32ドットマトリクスLED (HD-0158) 説明用サンプルプログラム
 * ATmega328P / IntRC 8MHz
 */

/* ***** I/Oポート割り当て *****
   ----- PORTD -----
PD7 :
PD6 :
PD5 :
PD4 :
PD3 :
PD2 :
PD1 :
PD0 :

   ----- PORTC -----
PC5 : A/BB in
PC4 : SE in
PC3 : A3 in
PC2 : A2 in
PC1 : A1 in
PC0 : A0 in

   ----- PORTB -----
PB7 :
PB6 :
PB5 :
PB4 : ALE in
PB3 : WE in
PB2 : CLK in
PB1 : DR in (赤データ)
PB0 : DG in (緑データ)
*/

#include <avr/io.h>
#include <avr/pgmspace.h>
#include <util/delay.h>

#define F_CPU 8000000

/* 電光掲示板 フォントのデータ */
const PROGMEM unsigned int font_tbl_r[2][16] =
{
    {0x0000, 0x2040, 0x2844, 0x2BFC, 0xA844, 0xA844, 0xA044, 0xA7FE,
     0xA040, 0xA040, 0x20A0, 0x20A0, 0x2110, 0x2208, 0x2406, 0x0000}, //快
    {0x0000, 0x6044, 0x37FC, 0x0040, 0x07FC, 0x0444, 0xE444, 0x27FC,
     0x2140, 0x2260, 0x2450, 0x2848, 0x2044, 0x5000, 0x8FFE, 0x0000}, //速
};

const PROGMEM unsigned int font_tbl_g[2][16] =
{
    {0x0000, 0x2040, 0x2844, 0x2BFC, 0xA844, 0xA844, 0xA044, 0xA7FE,
     0xA040, 0xA040, 0x20A0, 0x20A0, 0x2110, 0x2208, 0x2406, 0x0000}, //快
    {0x0000, 0x6044, 0x37FC, 0x0040, 0x07FC, 0x0444, 0xE444, 0x27FC,
     0x2140, 0x2260, 0x2450, 0x2848, 0x2044, 0x5000, 0x8FFE, 0x0000}, //速
};

```

次のページに続きます

## 前のページの続きです

```

/* 表示用バッファメモリ */
unsigned int disp_buf_r[2][16];
unsigned int disp_buf_g[2][16];

#define ADDR_PORT PORTC    // A[3:0]inの接続先ポート
#define ADDR_DDR DDRC
#define CTRL_PORT PORTB    // データ線、クロックなどの接続先ポート
#define CTRL_DDR DDRB
#define A_BB 5              // PC5 : RAM-A/RAM-B切り替え (手動切り替えのとき有効)
#define SEin 4              // PC4 : RAM切り替えモード選択 (「H」:手動/「L」:自動)
#define A3 3                // PC3~PC0 : 書き込み先行アドレス指定
#define A2 2
#define A1 1
#define A0 0
#define ALE 4               // PB4 : アドレスラッチイネーブル(「H」でアドレス取り込み)
#define WE 3                // PB3 : メモリへの書き込み信号(ALE = 「H」のとき有効)
#define CLK 2               // PB2 : 転送クロック
#define DR 1                // 赤表示データ
#define DG 0                // 緑表示データ

/* 書き込み先RAMの選択 */
#define RAMA 0
#define RAMB 1

/* 表示用バッファRAMの選択
   バッファRAM手動切り替え時に使用
   (このプログラムでは使用していません)
   void select_ram(int iram);
   iram = 0のときRAM-Aに書き込み
   iram = 1のときRAM-Bに書き込み
*/
void select_ram(int iram);
void select_ram(int iram)
{
    if(iram == RAMA)
    {
        PORTC = PORTC | (1<<A_BB);
    }
    else
    {
        PORTC = PORTC & ~(1<<A_BB);
    }
}

/* データを1行分書き込む
   void send_data(int iaddr_y, unsigned long *idata_r, unsigned long *idata_g);
   iaddr_y : データを書き込む行アドレス (0~15)
   *idata_r : 赤表示データ (符号なし32ビット整数) の配列先頭要素のアドレス
   *idata_g : 緑表示データ (符号なし32ビット整数) の配列先頭要素のアドレス
   ※表示データの配列の要素数はパネルの枚数と同じです。
*/

#define NUM_PANEL 1        //パネルの枚数(1枚)

void send_data(int iaddr_y, unsigned long *idata_r, unsigned long *idata_g);
void send_data(int iaddr_y, unsigned long *idata_r, unsigned long *idata_g)
{
    volatile int i, j;
    unsigned long m, n;

```

## 次のページに続きます

## 前のページの続きです

```

for(i=0;i<NUM_PANEL;i++)
{
    m = *idata_r;          // 配列に入った表示データを取得
    n = *idata_g;
    /* 32ビットの表示データを、下位ビットから順番に送る */
    for(j=0;j<32;j++)
    {
        PORTB = PORTB & ~(1<<CLK);          // CLK = L
        PORTB = PORTB & ~((1<<DR)|(1<<DG));    // データビット(DR/DG)をセット
        if((m & 0x80000000UL)!=0)
        {
            PORTB = PORTB | (1<<DR);
        }
        if((n & 0x80000000UL)!=0)
        {
            PORTB = PORTB | (1<<DG);
        }
        m = m <<1;
        n = n <<1;
        /* シフトクロックを送る(立ち上がりエッジで取り込み) */
        PORTB = PORTB | (1<<CLK);          // CLK = H
    }
    idata_r = idata_r + 1; // 次の配列データを指すようにする
    idata_g = idata_g + 1;

    /* 書き込むRAMのアドレスをセットする */
    PORTC = PORTC & 0b11110000;
    PORTC = PORTC | (iaddr_y & 0x0f);          // PC[3:0] = A[3:0]
    PORTB = PORTB | (1<<ALE);                  // ALE = Hでアドレスセット
    /* 書き込み信号を与える */
    PORTB = PORTB | (1<<WE);                  // WE = Hでデータ書き込み
    PORTB = PORTB & ~(1<<WE);                 // WE = L
    PORTB = PORTB & ~(1<<ALE);                 // ALE = L
}

int main(void)
{
    int i;
    // 32ビット符号なし整数型の配列をパネルの枚数分用意
    unsigned long m[NUM_PANEL], n[NUM_PANEL];
    // I/Oポートの初期化
    PORTB = 0b00000000;
    PORTC = 0b00000000;
    PORTD = 0b00000000;
    DDRB = 0b00111111;
    DDRC = 0b00111111;
    DDRD = 0b10000000;
    _delay_ms(100);          // パネルのパワーONリセット待ち
    while(1)
    {
        /* フォントデータをRAM上にコピー */
        for(i=0;i<16;i++)
        {
            disp_buf_r[1][i] = pgm_read_word(&font_tbl_r[0][i]);
            disp_buf_g[1][i] = pgm_read_word(&font_tbl_g[0][i]);
            disp_buf_r[0][i] = pgm_read_word(&font_tbl_r[1][i]);
            disp_buf_g[0][i] = pgm_read_word(&font_tbl_g[1][i]);
        }
        for(i=0;i<16;i++)
        {
            m[0] = disp_buf_r[1][i];
            m[0] = (m[0] <<16) + disp_buf_r[0][i];
            n[0] = disp_buf_g[1][i];
            n[0] = (n[0] <<16) + disp_buf_g[0][i];
            send_data(i, &m, &n);          // 1行分の表示データを書き込む
        }
    }
}

```



