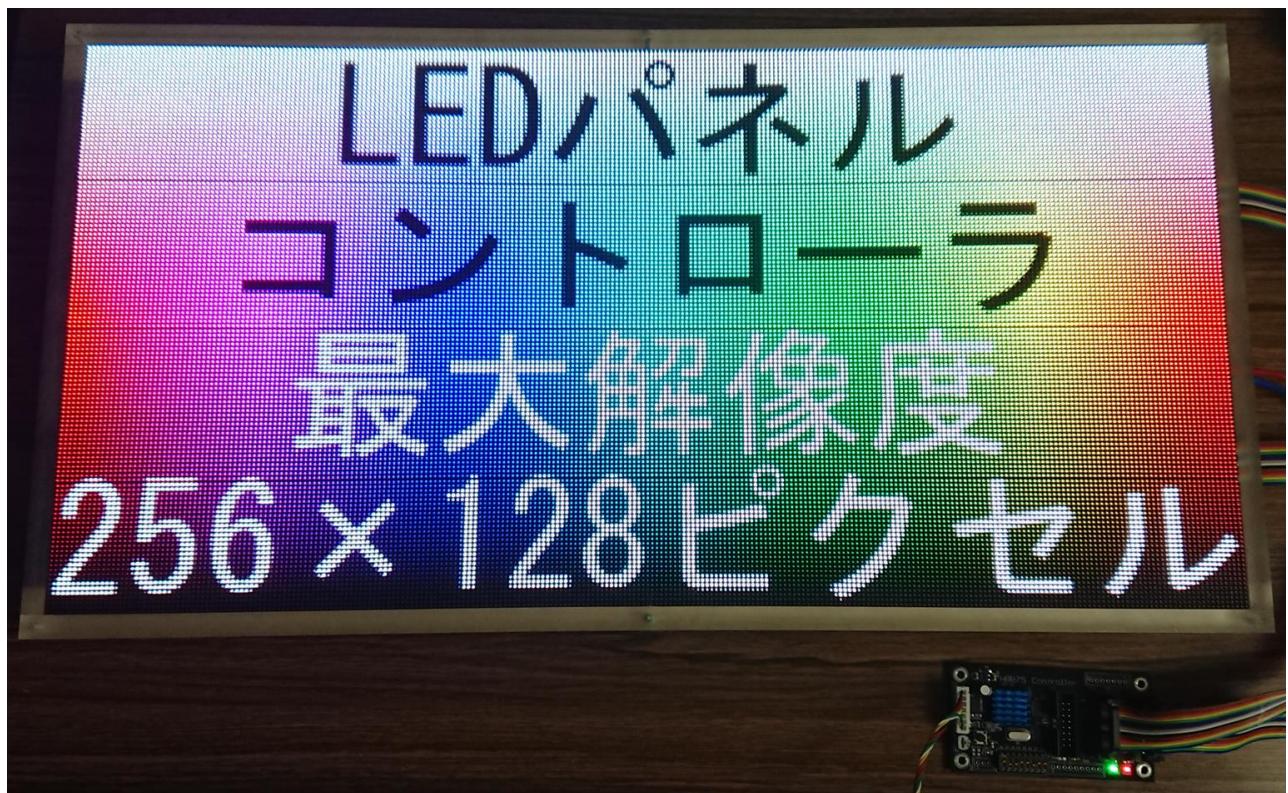


高性能版LEDパネルコントローラ

取扱説明書



1/29/2024

v0.90

目次

1 LEDパネルコントローラについて	4
1.1 概要	4
1.2 仕様	5
1.2.1 仕様一覧	5
1.2.2 機能一覧	6
1.2.3 回路図	12
1.2.4 部品リスト	14
1.2.5 外形寸法	15
1.2.6 各部の説明	16
1.2.7 ピンアサインと信号入力電圧	17
1.3 LEDパネルの接続	18
1.3.1 HUB75規格について	18
1.3.2 使用可能なLEDパネル	19
1.3.3 LEDパネルのピクセル座標の表し方	20
1.3.4 DIPスイッチについて	22
1.3.5 LEDパネルの解像度設定	23
1.3.6 HUB75出力ポート構成の設定	26
1.3.7 LEDパネルテスト機能	31
1.4 SPI通信	33
1.4.1 SPI信号タイミング	33
1.4.2 必要な待機時間	34
1.4.3 コマンド受信	35
1.4.4 データ受信	53
2 LEDパネルの点灯試験	55
2.1 必要なもの一覧	55
2.2 電源の選定	55
2.3 接続配線	56
2.3.1 全体配線図	56
2.3.2 LEDパネルの電源とHUB75のフラットケーブル接続箇所について	57
2.4 立ち上げ	58
3 Arduino Unoでの使用例	59
3.1 必要なもの一覧	59
3.2 接続配線	60
3.2.1 全体配線図	60
3.3 スケッチ例	61
3.3.1 ArduinoでのSPIの使用方法	61
3.3.2 データ送信のサンプルプログラム	62
3.3.3 LEDパネル全体を単色埋めする	64
3.3.4 コマンド送信のサンプルプログラム	67
3.3.5 LEDパネルの描画範囲をコマンドで指定する	68
3.3.6 LEDパネルONOFFと輝度調整コマンドを利用する	71
3.3.7 PWM分解能とガンマ補正值変更コマンドを利用する	74
3.3.8 描画支援機能を利用する	76
3.3.9 バッファ切り替え機能を利用する	80

3.3.10 画面コピー機能を利用する	84
4 その他	88
4.1 LEDパネルのリフレッシュタイミング(同期信号)を得るには	88
4.2 LEDパネルコントローラのリフレッシュタイミングを外部信号で制御したいときは	89
4.3 LEDパネルコントローラがBUSY状態であることを知るには	89
4.4 LEDパネルコントローラを外部から強制的にリセットするには	90

図の一覧

図 1.2.2-1 シングルバッファとダブルバッファ	10
図 1.2.3-1 回路図1	12
図 1.2.3-2 回路図2	13
図 1.2.5-1 外形寸法図 単位[mm]	15
図 1.2.7-1 ピンアサイン	17
図 1.3.1-1 HUB75Eコネクタのコネクタ形状とピンアサイン	18
図 1.3.3-1 LEDパネルの座標	20
図 1.3.3-2 LEDパネルの座標 2枚連結した場合	20
図 1.3.3-3 LEDパネルの座標 複数のHUB75ポートに接続した場合	21
図 1.3.5-1 可能なLEDパネルの構成例	24
図 1.3.5-2 不可能なLEDパネルの構成例	25
図 1.3.6-1 HUB75出力ポート設定とLEDパネル構成図	27
図 1.3.6-2 HUB75出力ポート設定とLEDパネル構成例1	28
図 1.3.6-3 HUB75出力ポート設定とLEDパネル構成例2	29
図 1.3.6-4 不可能なLEDパネルの構成例	30
図 1.3.7-1 テストパターンと表示される順番	31
図 1.4.1-1 SPIタイミングチャート	33
図 1.4.2-1 待機時間タイミングチャート	34
図 1.4.3-1 コマンド受信タイミングチャート	35
図 1.4.3-2 LEDパネルの描画範囲限定例	39
図 1.4.4-1 データ受信タイミングチャート	53
図 1.4.4-2 受信したデータと描画されるピクセルの位置関係	54
図 1.4.4-3 描画ピクセル位置をリセットする際のタイミングチャート	54
図 2.3.1-1 LEDパネル点灯試験の全体配線図	56
図 3.2.1-1 Arduino Unoとの接続	60
図 4.1-1 同期信号の出力端子	88
図 4.3-1 BUSY信号の出力端子	89
図 4.4-1 リセット入力端子	90

表の索引

表 1.2.1-1 仕様	5
表 1.2.4-1 部品リスト	14
表 1.2.7-1 ピンアサイン一覧	17
表 1.3.1-1 HUB75コネクタのピンの詳細	18

表 1.3.2-1	解像度、スキャンレート、バイナリデコーダ入力ピンの関係	19
表 1.3.4-1	DIPスイッチ機能一覧	22
表 1.3.5-1	DIPスイッチのONOFF組み合わせと解像度	23
表 1.3.6-1	DIPスイッチのONOFF組み合わせとHUB75出力ポート構成	26
表 1.4.2-1	必要な待機時間	34
表 1.4.3-1	CMD_NUM一覧	36
表 1.4.3-2	文字コード表	49

画像の一覧

画像 1.2.2-1	輝度調整機能	6
画像 1.2.2-2	PWM分解能の比較	7
画像 1.2.2-3	ガンマ補正值の比較	8
画像 1.2.2-4	色調補正の比較	9
画像 1.2.6-1	各部の説明	16
画像 1.3.4-1	DIPスイッチ	22
画像 2.3.2-1	LEDパネル裏面のケーブルの接続状況	57
画像 2.4-1	LEDパネル裏面のケーブルの接続状況	58
画像 3.3.2-1	サンプルプログラムが動作している様子	63
画像 3.3.3-1	LEDパネル単色埋めプログラムの動作している様子	65
画像 3.3.5-1	LEDパネルの描画範囲が限定されている様子	69
画像 3.3.5-2	描画範囲限定の拡大画像	70
画像 3.3.6-1	グラデーション輝度255	73
画像 3.3.8-1	描画支援機能サンプル	79
画像 3.3.9-1	バッファ1の表示	83
画像 3.3.9-2	バッファ2の表示	83
画像 3.3.10-1	バッファ1の表示	87
画像 3.3.10-2	バッファ2の表示	87

.1 LEDパネルコントローラについて

.1.1 概要

LEDパネルコントローラとは、HUB75規格のRGBドットマトリクスLEDパネルのハードウェアを隠蔽し、SPI接続で簡単にフルカラー表示を行えるようにしたものです。

HUB75規格のドットマトリクスLEDパネルは、単純にシフトレジスタとバイナリデコーダが載っているのみでフレームバッファを持っていないため、ドットマトリクスLEDパネルの全面を表示しているように見せるには常に高速でデータを送信し画面を更新し続けなければなりません。そのためマイコンにとってはかなり高負荷であり、高解像度なパネルを満足のいくリフレッシュレートと色深度で表示させるには多くのCPUリソースとメモリが必要になります。他の処理をさせる余裕が全く

ないばかりか、低スペックなマイコンではメモリ不足によりそもそもドットマトリクスLEDパネルを表示させることさえままならないでしょう。

またHUB75規格では、RGBの各LEDはONもしくはOFFのみでしか制御できないため、LEDの明るさを変えるにはソフトウェア的にパルス幅変調を実装しなければならず、かなり面倒です。

そこで、このLEDパネルコントローラを用いればそれらの面倒な処理をせずとも、SPIで各ピクセルの色を数値で送信するだけで簡単にドットマトリクスLEDパネルを表示できるようになります。

ドットマトリクスLEDパネルをいわばSPI接続のグラフィックディスプレイのように扱えるため、例えば時刻や温度湿度を表示させたり、オーディオスペクトラムアナライザを表示させたり、テトリスやブロック崩し、画像表示や動画再生などありとあらゆる応用が可能です。

画面のリフレッシュレートはおよそ60RPS以上を維持でき(PWM分解能8bit時)、ちらつきのない非常に滑らかな描画が可能です。また表示できる色数も約1543万色(PWM分解能12bit、ガンマ値2.2の時)あるため、グラデーションも美しく描画することができます。さらにLEDパネルコントローラ内部でガンマ補正をかけており、液晶モニタと比較しても遜色ない発色が可能になっています。最大総画素数が32768ピクセルあるため、64×64ピクセルのLEDパネルを最大8枚接続することができます。

.1.2 仕様

.1.2.1 仕様一覧

表 1.2.1-1 仕様

製品名	LEDパネルコントローラ
電源電圧	5V (4~6V)
消費電流	150mA程度
入力信号電圧	3.3Vもしくは5V(5Vトレント)
出力信号電圧	5V
出力信号規格	HUB75(E)
HUB75(E)ポート数	3ポート
最大横解像度	256ピクセル
最大縦解像度	128ピクセル
最大総画素数	32768ピクセル
最大色数	15,438,249色(PWM12bit、ガンマ2.2)
PWM分解能	1bit~12bit可変

ガンマ補正	0.5～5.0可変
色調補正	各色256段階
リフレッシュレート	60RPS以上(256×128、PWM8bit)
通信規格	SPI(受信のみ)
最大SPIクロック周波数	54MHz
外形寸法	90mm×47mm

表 1.2.1-1に記載されている消費電流は、LEDパネルを接続していないときに実測した値であり、あくまで目安値となります。条件によっては(電源の接続されていない、もしくは電源端子がショートされたLEDパネルを接続した場合や、LEDパネルコントローラに供給される電源電圧よりLEDパネルに供給される電源電圧が低い場合など)1 A近く流れる場合がございますのでご注意ください。

.1.2.2 機能一覧

⑩ 描画範囲指定機能

SPIからデータを受信する際、受信したデータをLEDパネルコントローラ内部のフレームバッファに書き込む範囲を、SPIからのコマンド受信によって指定できる機能です。

LEDパネルの一部分だけを描画したいとき、いくつかのコマンドと描画する範囲のデータ送信のみで済みます。毎回LEDパネル全体を描画するデータを送信する必要はありません。

⑩ LEDパネルON/OFF機能

本LEDパネルコントローラには後述の輝度調整機能が備わっていますが、輝度調整機能ではLEDパネルを完全に消灯させることはできません。LEDパネルを完全に消灯させたい場合はLEDパネルON/OFF機能を、SPIからのコマンド受信によって利用します。

LEDパネルON/OFF機能は、LEDパネルコントローラ内部のフレームバッファを書き換えることなくLEDパネルを消灯させますので、LEDパネルを再び点灯させれば以前の表示内容を即座に表示します。

⑩ 輝度調整機能

色再現性を損なうことなくLEDパネル全体の明るさを256段階で調整できる機能です。デフォルトではLEDパネルコントローラ上の半固定抵抗で明るさを可変できますが、SPIからのコマンド受信によって明るさを変更することもできます。

輝度:0

輝度:127

輝度:255



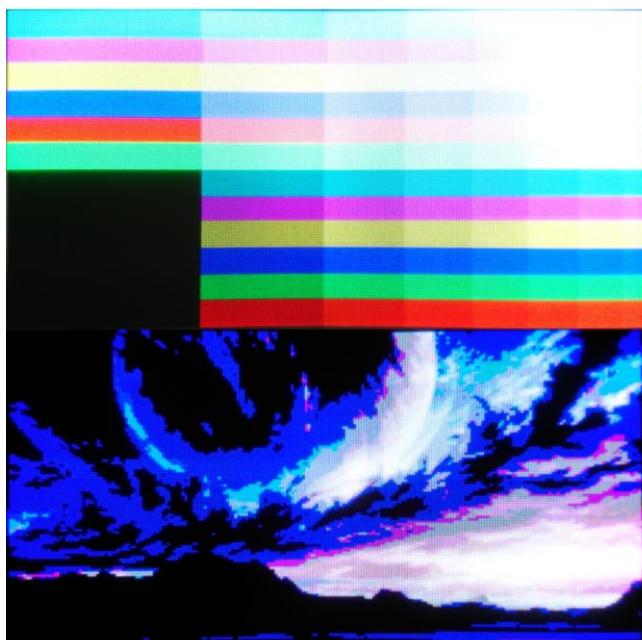
画像 1.2.2-1 輝度調整機能

⑩ PWM分解能変更機能

本LEDパネルコントローラでは、LEDパネルの各ピクセルにあたるRGBLEDの明るさをPWMによって可変することにより様々な色を表現しています。このPWMの分解能(明るさの段階数)を各色1bit(2段階)～12bit(4096段階)まで、SPIからのコマンド受信によって変更することができます。

基本的には、PWM分解能は高ければ高いほどより高画質になります。PWM分解能が低いとグラデーション部分が縞々になる現象(バンディングと言います)が発生し、これは特に暗い場面で顕著になります。しかし、PWM分解能が高いほどリフレッシュレートが低下してしまい、LEDパネルにちらつきを感じるようになります。本LEDパネルコントローラではデフォルト値

PWM:3bit



PWM:11bit



画像 1.2.2-2 PWM分解能の比較

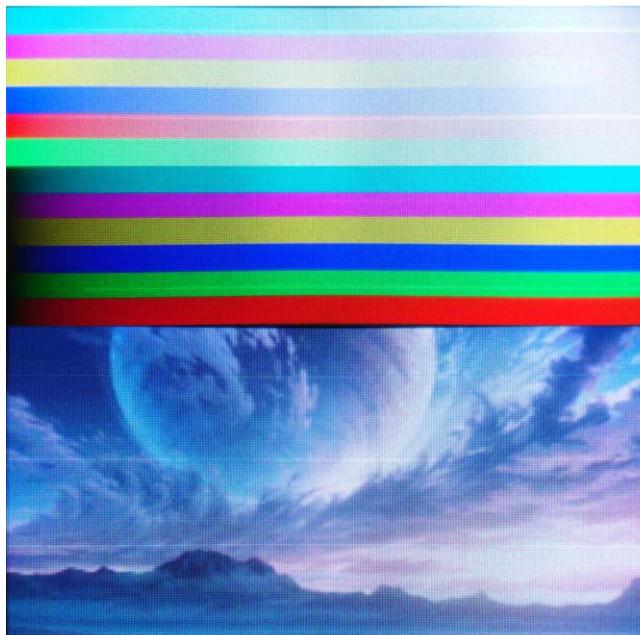
を8 bitとします。

⑩ ガンマ補正值変更機能

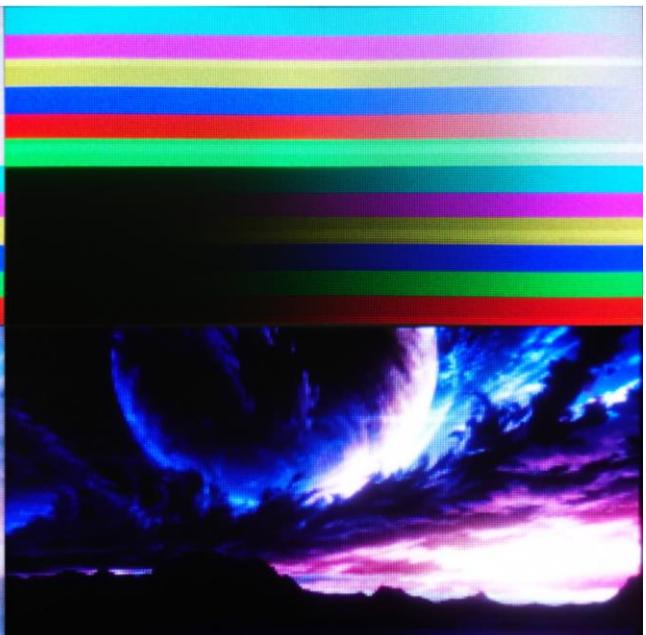
人間の目は暗いときは明るさの変化に敏感ですが、明るいときは明るさの変化に鈍感になるという特性があるため、実際の明るさと見た目の明るさはリニアに対応していません。そのような目の明るさの感じ方と表示器の明るさがリニアに対応するように補正することをガンマ補正と呼びます。本LEDパネルコントローラではこのガンマ補正值を0.5～5.0まで、SPIからのコマンド受信によって変更することができます。

ガンマ補正值が低いと全体的に白っぽくなりコントラストが低下します。
ガンマ補正值が高いとコントラストは高くなりますが暗い部分が潰れます。
Windowsの標準ではガンマ補正值は2.2と定められており、本LEDパネルコン

ガンマ:1.0



ガンマ:5.0



画像 1.2.2-3 ガンマ補正值の比較

トローラでもデフォルト値を2.2としています。

⑩ 色調補正機能

LEDパネルによっては、全体が緑がかった見えたりなど色のバランスがあまり良くないものが存在します。本LEDパネルコントローラはそういったLEDパネルの色調を、SPIからのコマンド受信によって各色256段階で補正す

色調補正前(255, 255, 255) 色調補正後(255, 127, 255)



画像 1.2.2-4 色調補正の比較

ることができます。

⑩ 表示・書き込みバッファ切り替え機能

本LEDパネルコントローラは、各ピクセルの色情報を格納するフレームバッファを2つ内蔵しており(バッファ1とバッファ2)、LEDパネルに表示するバッファ、データを書き込むバッファをSPIからのコマンド受信により自在に切り替えることが出来ます。

例えばティアリング軽減のためのダブルバッファとして使用する方法や、SPIの通信速度が遅い場合に2つのバッファに全く異なる内容の画面を書き込んでおき、表示するバッファを切り替えることで瞬時に画面が切り替わって

図 1.2.2-1 シングルバッファとダブルバッファ

いるように見せるなど、様々な応用が考えられます。

⑩ RGBピンスワップ機能

HUB75インターフェースのLEDパネルの中には、稀にRGBピンがHUB75規格通りになっておらず、入れ替わっているLEDパネルが存在しているようです。本LEDパネルコントローラはそういったLEDパネルに対応させるため、SPIからのコマンド受信によりHUB75コネクタのRGBピンをそれぞれ入れ替えることが出来ます。

⑩ 各種描画支援機能

例えばLEDパネル全体を任意の色で描画したいとき、LEDパネルが高解像度であればあるほど大量のデータを転送しなければならず描画速度が低下し、またデータ送信側のマイコンの負担も大きくなります。描画支援機能を使用すれば、描画範囲の大きさによらず数10バイトのコマンド送信のみでLEDパネルに数種類の図形を任意の色で描画することが出来ます。またコマンドで描画する図形の透過度を指定して背景とアルファブレンドすることも出来ます。描画支援機能を用いて描画できる図形は点、直線、四角形(枠のみ)、四角形(中埋め)、円(枠のみ)、円(中埋め)、文字の7種類です。

⑩ 画面コピー機能

本LEDパネルコントローラでは、SPIからのコマンド受信によりLEDパネルの任意の場所の矩形範囲を任意の場所にコピーすることができます。この機能を用いれば、

例えば画面のスクロールなどが簡単に行えるようになります。

⑩ 上下反転機能

本LEDパネルコントローラは通常LEDパネルの左上を原点としています
が、SPIからのコマンド受信により上下反転機能を有効にするとLEDパネル
の左下が原点になります。(数学的な直交座標と同じ座標系にすることが出
来ます。)

⑩ リフレッシュレート指定機能

LEDパネルのリフレッシュレートをSPIからのコマンド受信により数値で
指定することができます。より美しい画面スクロールを実現するために都合
がよい機能です。

⑩ 外部リフレッシュ同期機能

LEDパネルのリフレッシュタイミングを外部信号入力によりコントロール
することができる機能です。より美しい画面スクロールを実現するために都
合がよい機能です。

⑩ LEDパネルテスト機能

本LEDパネルコントローラはDIPスイッチを切り替えることにより、LEDパネルコントローラ内部で生成した複数のパターンをLEDパネルに表示させ、単体でLEDパネルの点灯試験を行うことができます。この機能を用いることで、接続しているLEDパネルが本LEDパネルコントローラに対応しているか、LEDパネルに点灯しないピクセルが無いか、HUB75インターフェースのRGBピンの並び順、最大消費電流などのチェックに役立ちます。

.1.2.3 回路図

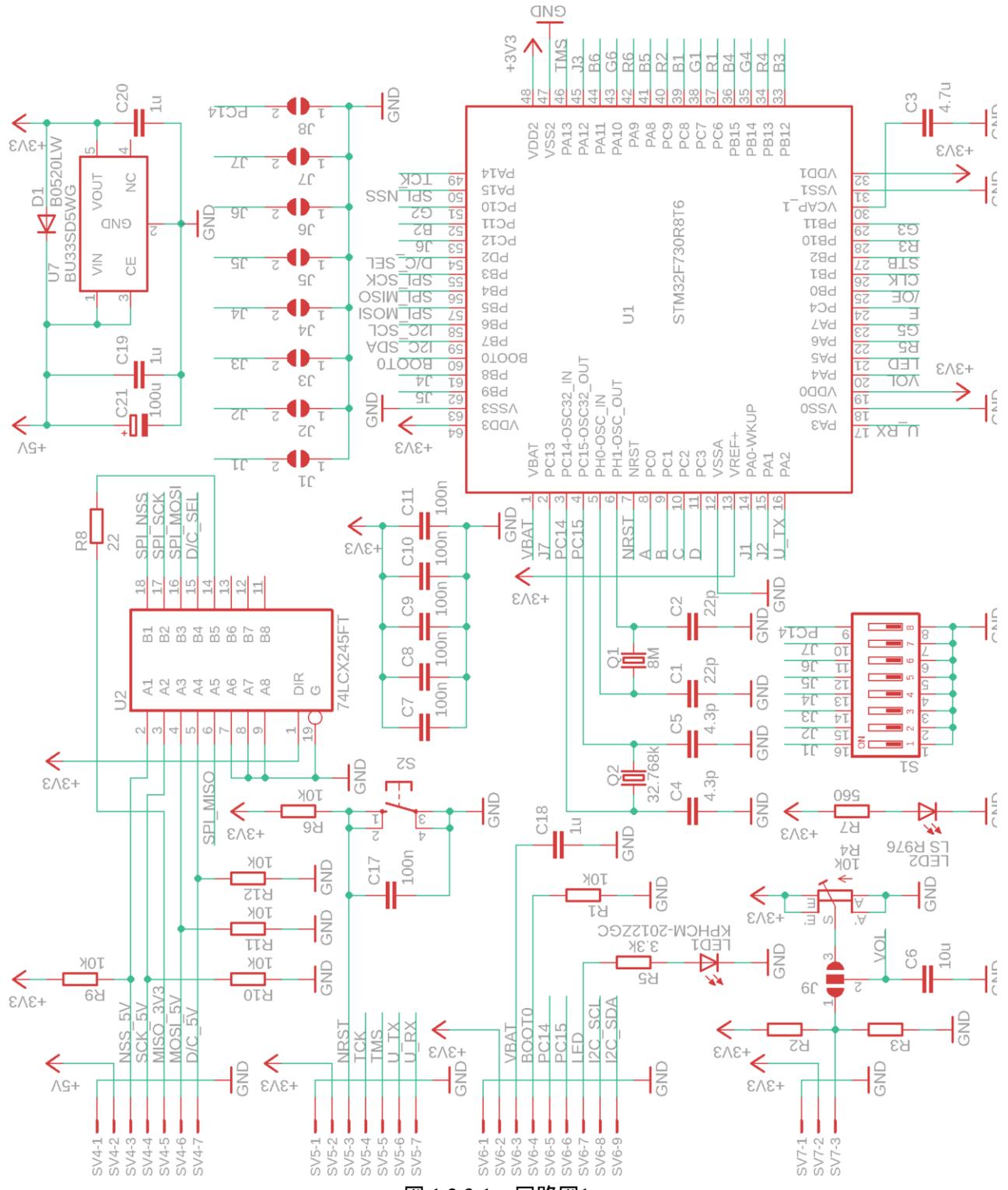


図 1.2.3-1 回路図1

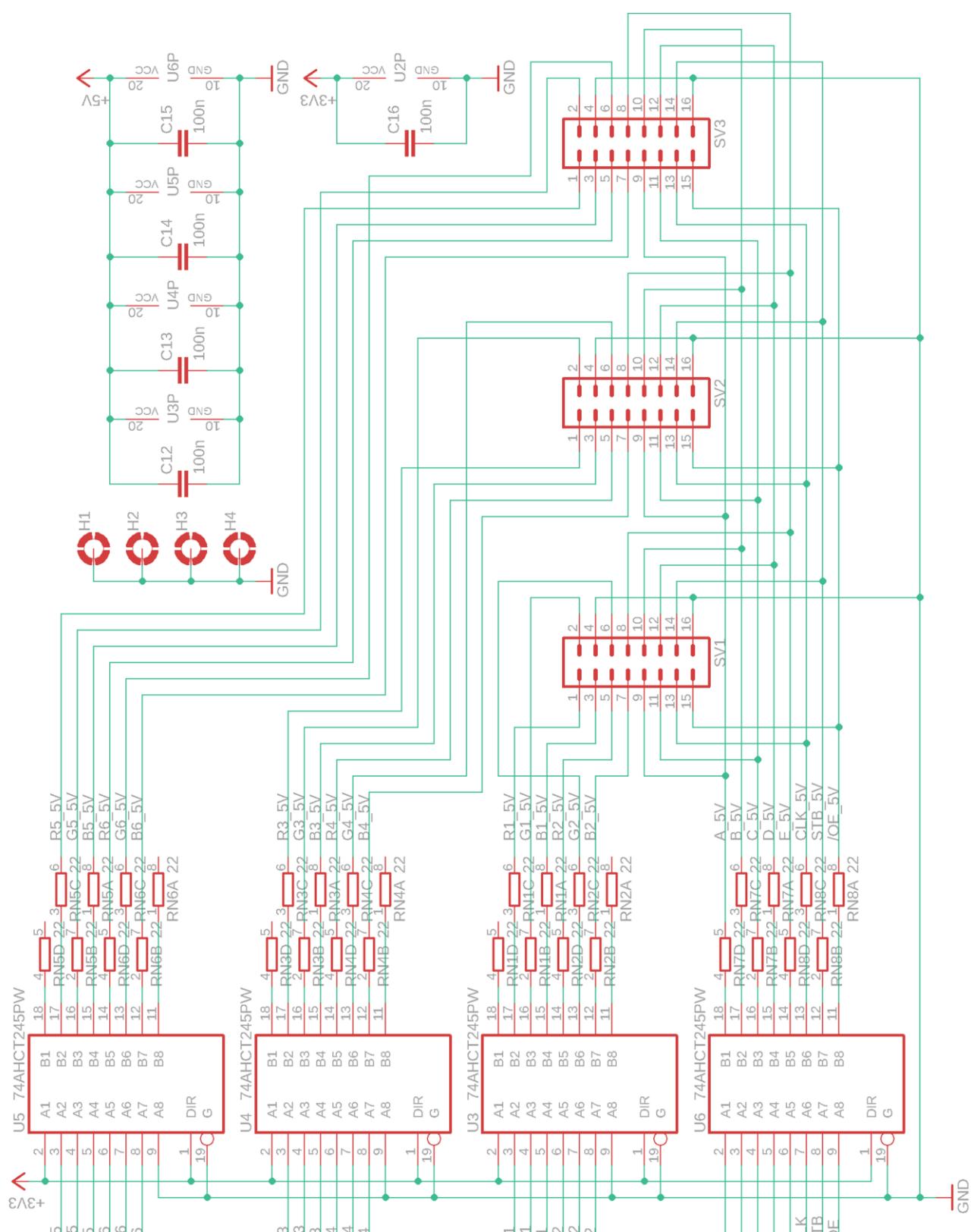


図 1.2.3-2 回路図2

1.2.4 部品リスト



表 1.2.4-

1 部品リスト

なお、実際のLEDパネルコントローラには同等品が使用されていたり、部品定数が異なる場合がございますが、動作には支障ないことを確認しておりますのでご了承ください。

.1.2.5 外形寸法

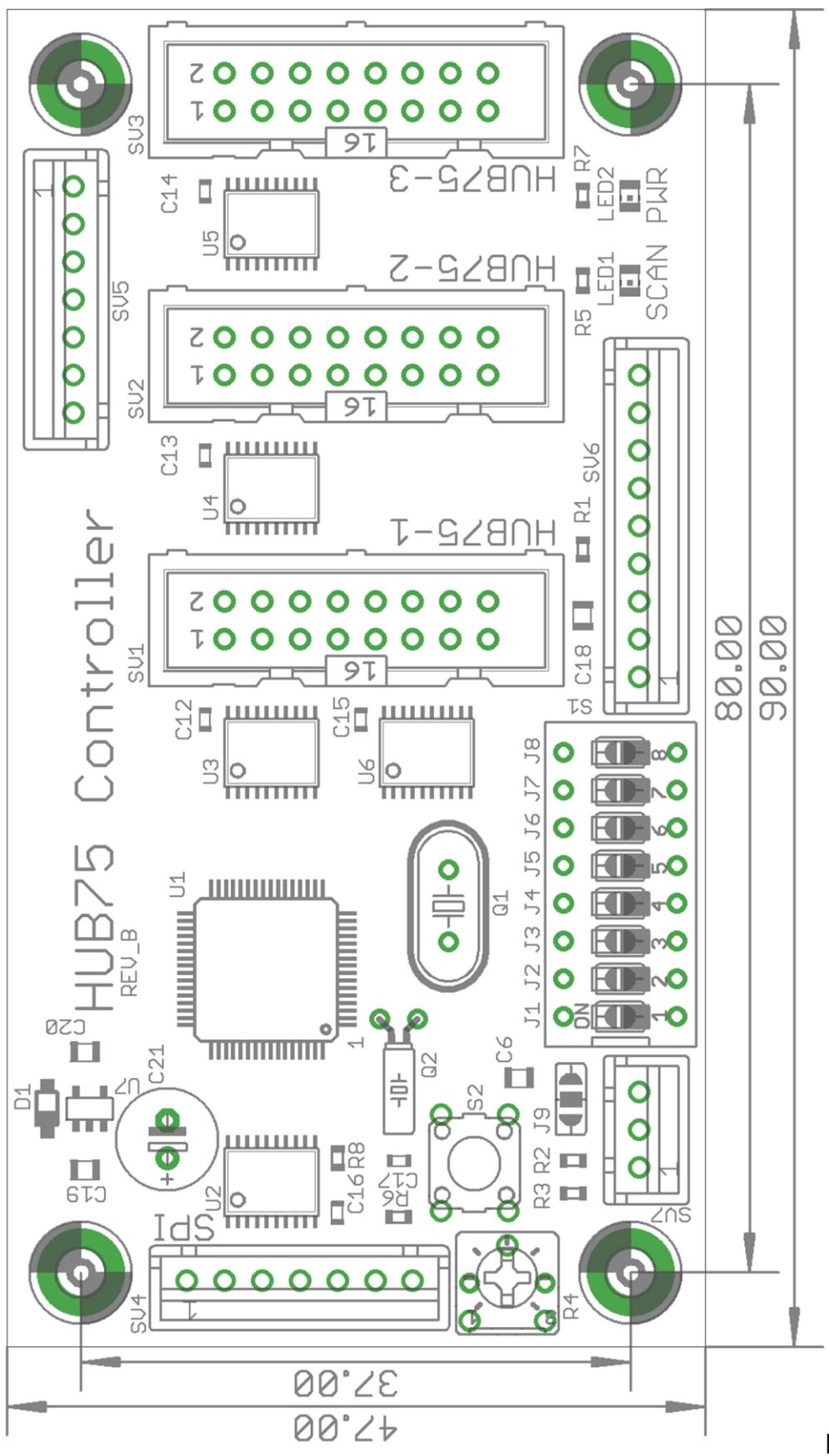
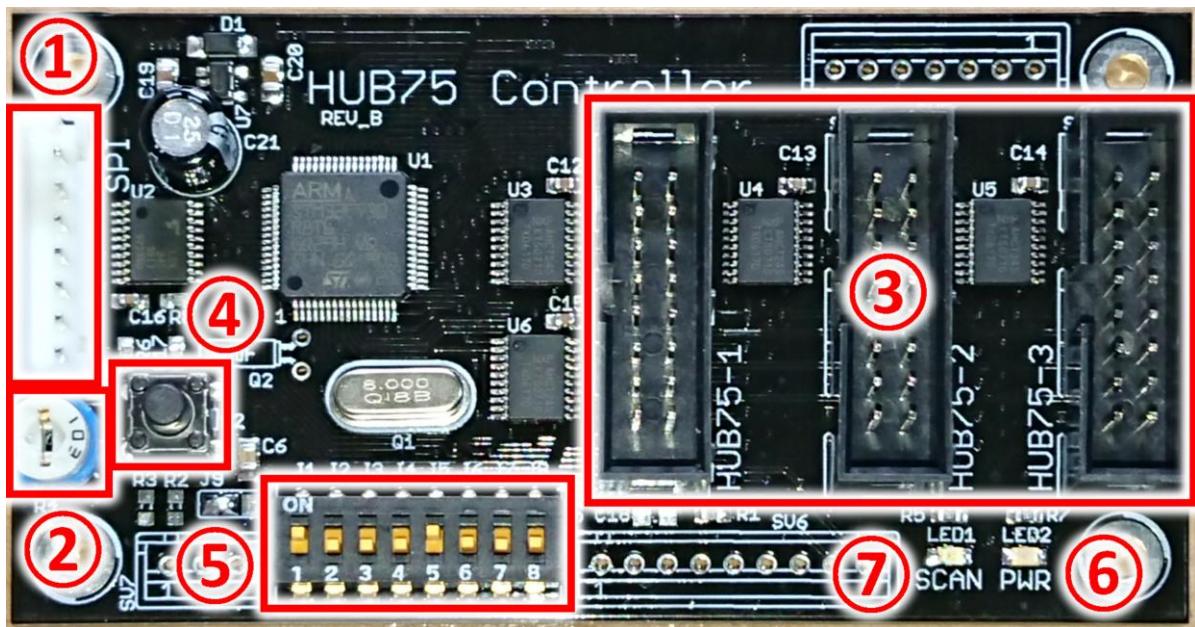


図 1.2.5-1 外形寸法図

単位[mm]

.1.2.6 各部の説明



画像

1.2.6-1 各部の説明

- ① マスター側のマイコンとSPI通信を行うためのコネクタです。
- ② LEDパネル全体の明るさを調整するための半固定抵抗です。左に回すと暗く、右に回すと明るくなります。
- ③ HUB75 規格のLEDパネルを接続するためのコネクタです。
- ④ LEDパネルコントローラをリセットするリセットスイッチです。
- ⑤ 接続するLEDパネルの解像度や、HUB75出力ポート構成、LEDパネルテスト機能を設定するためのDIPスイッチです。
- ⑥ 通電状態を示すLEDです。通電時は赤色に点灯します。
- ⑦ LEDパネルのスキャン状態を示すLEDです。LEDパネルをスキャンするたび緑色点灯と消灯を繰り返します。

.1.2.7 ピンアサインと信号入力電圧

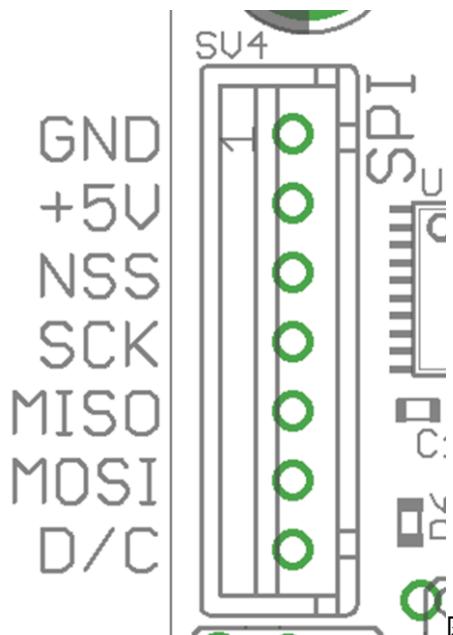


図 1.2.7-1 ピンアサイン

表 1.2.7-1 ピンアサイン一覧

端子名	詳細	種別
GND	電源のグランドです。	電源
+5V	5V電源入力です。	電源
NSS	SPIのスレーブセレクト端子です。チップセレクトとも呼ばれます。この端子がLOWになっている間のみ受信を受け付けます。	入力 プルアップ
SCK	SPIのクロック入力端子です。この端子は通常LOWで、立ち上がりエッジが発生したときにSPI_MOSIの信号を読み取ります。	入力 プルダウン
MOSI	SPIのマスターアウトスレーブイン端子です。この端子にデータもしくはコマンドを入力します。	入力 プルダウン
D/C	SPI_MOSIから入力された信号がピクセルの色情報(データ)なのか、LEDパネルコントローラの設定情報(コマンド)なのか判別するための端子です。この端子をLOWにするとデータ受信、HIGHにするとコマンド受信になります。	入力 プルダウン

MISO端子は受信オンリーであるため使用しません。そのため未接続としておいてください。

信号入力端子は5Vトレラントとなっており、3.3Vと5V両方の信号をそのまま入力することが出来ます。面倒なレベルシフトは必要ありません。

.1.3 LEDパネルの接続

.1.3.1 HUB75規格について

HUB75もしくはHUB75E規格とは、一般的に安く出回っているRGB ドットマトリクスLEDパネルの信号規格のことです。

コネクタには16ピンのボックスヘッダが用いられており、フラットケーブルを使用してLEDパネルコントローラと接続します。

また、LEDパネルからLEDパネルへと、フラットケーブルで数珠繋ぎ(デイジーチェーン接続)にして拡張させることができます。

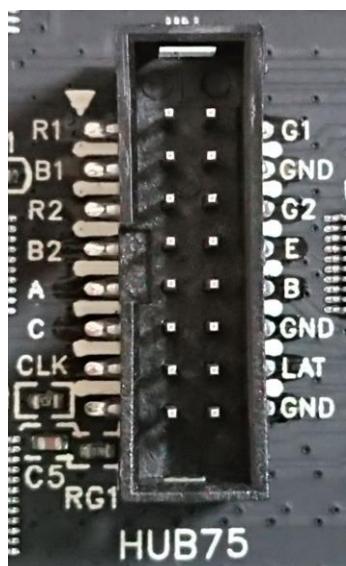


図 1.3.1-1 HUB75Eコネクタのコネクタ形状とピンアサイン

図 1.3.1-1に実物のコネクタ形状とピンアサインを示します。

表 1.3.1-1 HUB75コネクタのピンの詳細

ピン名	詳細
R1, G1, B1, R2, G2, B2	シフトレジスタのデータ入力ピンです。それぞれが赤、緑、青のLEDに対応しています。
A, B, C, D, E	バイナリデコーダの入力ピンです。これらのピンで表示させるLEDの行を選択します。
CLK	シフトレジスタのクロック入力ピンです。
LAT	データをシフトレジスタからラッチに転送するピンです。
/OE	出力イネーブルピンです。このピンをLにしている間だけLEDが点灯します。
GND	グランドです。LEDパネルの電源コネクタのGNDと共にっています。

HUB75とHUB75Eの違いは、バイナリデコーダの入力ピンがA,B,C,DまでしかないものをHUB75と呼び、A,B,C,D,EまであるものをHUB75Eと呼ぶようです。

.1.3.2 使用可能なLEDパネル

LEDパネルコントローラで使用可能なLEDパネルは、次の条件を満たしている必要があります。

- ⑩ RGB ドットマトリクスLEDパネルであること(単色ではないこと)。
- ⑩ 信号規格がHUB75もしくはHUB75Eであること。
- ⑩ R1,G1,B1,R2,G2,B2,A,B,C,D,Eピンがすべて正論理入力であること。
- ⑩ OEピンが負論理入力であること。
- ⑩ CLK,LATピンが立ち上がりエッジ検出であること。
- ⑩ 総画素数が32768ピクセル以内であること。
- ⑩ 定電流LED ドライバICにFM6126Aなどの特殊なものが使用されていないこと。
- ⑩ 屋内用のLEDパネルであること。もしくは縦解像度64ピクセルの一部の屋外用パネルであること。
- ⑩ LEDパネルの解像度とスキャンレート、バイナリデコーダ入力ピンの本数の関係が表 1.3.6-1の通りになっていること。

表 1.3.2-1 解像度、スキャンレート、バイナリデコーダ入力ピンの関係

横解像度	縦解像度	スキャンレート	バイナリデコーダ入力ピン
32	16	1/8	A, B, C
32	32	1/16	A, B, C, D
64			
64	64	1/32	A, B, C, D, E
128			
64	64	1/16(屋外用)	A, B, C, D

これらの条件を満たしていないLEDパネルを接続した場合、正しく表示できませんのでご注意ください。また、これらの条件をすべて満たしていたとしても確実に正しく表示できることを保証するものではありません。特に屋外用LEDパネルは本LEDパネルコントローラで想定しているピクセルマップと異なる場合正常に表示できません。

.1.3.3 LEDパネルのピクセル座標の表し方

この取扱説明書では、上下反転機能を無効にしている状態において、LEDパネルの左上のピクセルを原点X0、Y0とし、右下に行くにしたがってX座標とY座標がそれぞれ増加していくこととします。(上下反転機能を有効にした場合は、左下から右上に行くにしたがってX座標とY座標が増加します。)

例えば、横解像度32ピクセル、縦解像度16ピクセルのLEDパネルでは図 1.3.3-1の

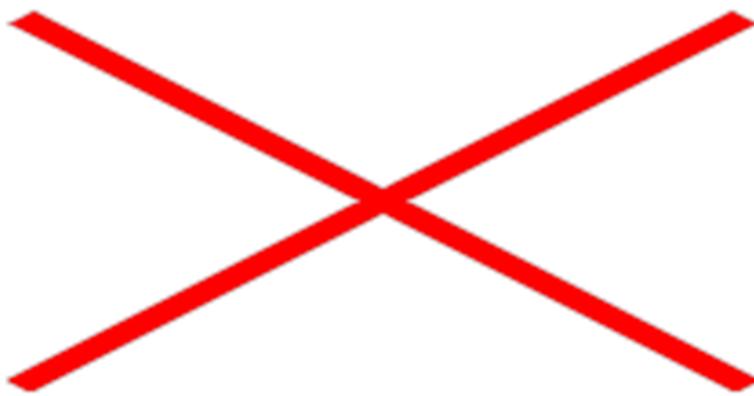


図 1.3.3-1 LEDパネルの座標

ようになります。

32×16ピクセルのLEDパネルを2枚連結した場合は図 1.3.3-2のようにX軸方向に増

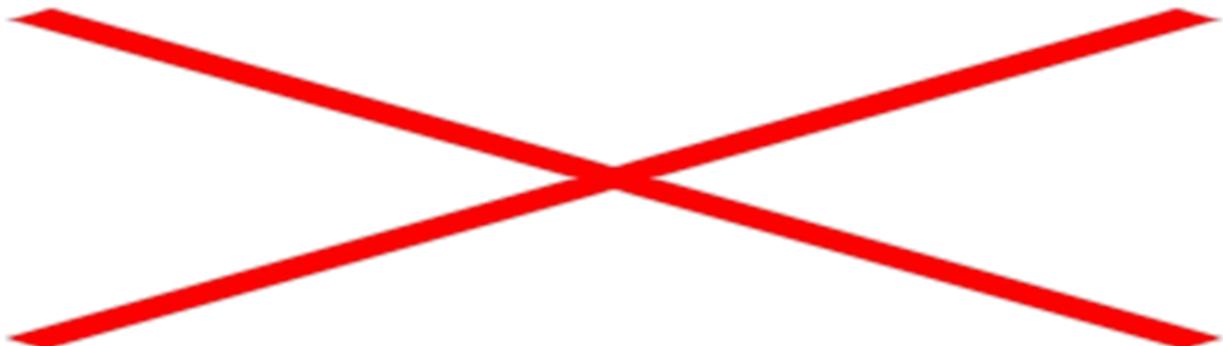


図 1.3.3-2 LEDパネルの座標 2枚連結した場合

加します。

32×16ピクセルのLEDパネルをLEDパネルコントローラの複数のHUB75ポートに2枚接続した場合は、HUB75ポートの番号の若いほうから数えて図 1.3.3-3のようにY軸方向に増加します。



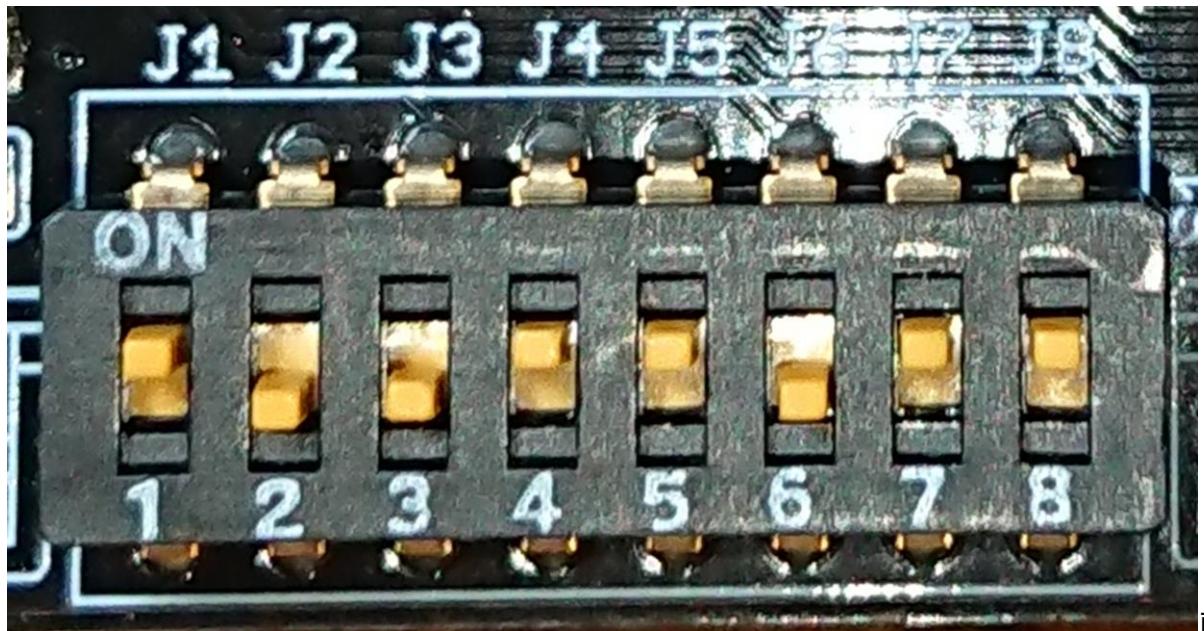
図 1.3.3-3 LEDパネルの座標 複数の
HUB75ポートに接続した場合

なお、LEDパネルコントローラの複数のHUB75ポートに複数のLEDパネルを接続する場合、それぞれのHUB75ポートに接続されるLEDパネルはすべて同じ解像度にする必要があります。詳細については「23 LEDパネルの解像度設定」と「26 HUB75出力ポート構成の設定」を参照してください。

.1.3.4 DIPスイッチについて

本LEDパネルは画像 1.2.6-1の⑤DIPスイッチにより、接続するLEDパネルの解像度を設定したり、HUB75出力ポートの構成の設定やLEDパネルテスト機能の有効無効を設定することができます。DIPスイッチは上にスライドさせるとON、下にスライドさせるとOFFになります。

DIPスイッチで設定した内容は、LEDパネルコントローラのリセット直後から有効になります。LEDパネルコントローラに電源が入っている状態で設定を変更した場合は、変更を反映させるために画像 1.2.6-1の④リセットスイッチを押下してLEDパネルコントローラをリセットしてください。



画像

1.3.4-1 DIPスイッチ

DIPスイッチの各接点に割り当てられている機能の一覧を表 1.3.4-1に示します。

表 1.3.4-1 DIPスイッチ機能一覧

接点の番号	機能
1	
2	
3	スイッチONOFFの組み合わせにより、LEDパネルの解像度を設定する。
4	
5	
6	スイッチONOFFの組み合わせにより、HUB75出力ポートの構成を設定する。
7	
8	LEDパネルテスト機能の有効無効を切り替える。

.1.3.5 LEDパネルの解像度設定

「22 DIPスイッチについて」に記載したように、本LEDパネルコントローラではDIPスイッチの接点1から接点5のONOFFの組み合わせにより、**单一のHUB75出力ポートに单一もしくは複数ディジーチェーン接続されるLEDパネルの解像度を設定**することができます。表 1.3.5-1にONOFFの組み合わせと解像度の一覧を記載します。



表 1.3.5-1 DIPスイッチの
ONOFF組み合わせと解像度

ここで、縦解像度とは単一のHUB75ポートに接続されるLEDパネルの縦解像度、横解像度とは単一のHUB75ポートにデイジーチェーン接続されるすべてのLEDパネルの横解像度の合計を意味します。

縦解像度が同じLEDパネルは、横解像度が異なっていてもデイジーチェーン接続することが可能ですが、縦解像度が異なるLEDパネルはデイジーチェーン接続することはできません。

図 1.3.5-1 可能なLEDパネルの構成例

図 1.3.5-1に可能なLEDパネルの構成例を示します。

図 1.3.5-2 不可能なLEDパネルの構成例

図 1.3.5-2に不可能なLEDパネルの構成例を示します。

.1.3.6 HUB75出力ポート構成の設定

「22 DIPスイッチについて」に記載したように、本LEDパネルコントローラではDIPスイッチの接点6から接点7のONOFFの組み合わせにより、HUB75出力ポート構成(画面の複製もしくは拡張)を設定することが出来ます。なお、各HUB75出力ポートに接続するLEDパネルの解像度はすべて同じ縦解像度、横解像度にしなければならず、また異なる内容を表示するLEDパネルの総画素数が32768ピクセル以内になるようにしなければなりません。エラー: 参照先が見つかりません。にONOFFの組み合わせと構成の一覧、図 1.3.6-1にHUB75出力ポート設定とLEDパネルの構成図を記載します。

表 1.3.6-1 DIPスイッチのONOFF組み合わせとHUB75出力ポート構成

DIPスイッチ 接点の番号		各HUB75出力ポート構成				
	6	7	HUB75-1	HUB75-2	HUB75-3	
0	OFF	OFF	A	A	A	1を2と3に複製
1	OFF	ON	A	A	B	1を2に複製、1を3に拡張
2	ON	OFF	A	B	B	1を2に拡張、2を3に複製
3	ON	ON	A	B	C	1を2と3に拡張



図 1.3.6-1 HUB75出力ポート設定
とLEDパネル構成図



図 1.3.6-2

HUB75出力ポート設定とLEDパネル構成例1

図 1.3.6-2と図 1.3.6-3にHUB75出力ポート設定とLEDパネル構成例を示します。



図 1.3.6-3

HUB75出力ポート設定とLEDパネル構成例2

図 1.3.6-4に不可能なLEDパネルの構成例を示します。



図 1.3.6-4

不可能なLEDパネルの構成例

.1.3.7 LEDパネルテスト機能

「22 DIPスイッチについて」に記載したように、本LEDパネルコントローラではDIPスイッチの接点8をONすることにより、LEDパネルテスト機能を有効にするこ

とができますが、予め「23 LEDパネルの解像度設定」と「26 HUB75出力ポート構成の設定」を参照して接続されたLEDパネルに合わせて設定を済ませておく必要があります。

なお、LEDパネルテスト機能が有効になっているときは、一切のコマンド受信、データ受信を受け付けません。輝度調整は画像 1.2.6-1の②半固定抵抗を用いて行います。



図

1.3.7-1 テストパターンと表示される順番

図 1.3.7-1に表示されるテストパターンと順番について記載します。

LEDパネルテスト機能が有効になっている場合は、図 1.3.7-1のパターンが一定時間ごとに順番に切り替わっていきます。

パターン1からパターン4では、LEDパネルのドット欠けがないかをチェックできます。

パターン1からパターン3の表示順番が入れ替わっている場合は、LEDパネルのHUB75ポートのRGBピンが入れ替わっています。RGBピンスワップ機能を用いることでピンアサインを修正できます。

パターン4ではLEDパネルの色調を確認できます。また、LEDパネル全面を白色で描画するため、LEDパネルが消費する最大電力を測定することができます。その場合、LEDパネルの明るさを最小にした状態から様子を見ながら徐々に明るさを上げていきます。

パターン5では対応しているLEDパネルか、RGBピンが入れ替わっていないかを確認できます。各グラデーションの帯の色が入れ替わっている場合は、LEDパネルのHUB75ポートのRGBピンが入れ替わっています。LEDパネルの横解像度によっては一番右端の白色のグラデーションの帯が存在しなかったり、白色のグラデーションの帯の右に白単色の帯が追加される場合がありますが、これは正常です。また、表示が安定せず常に変化している場合は信号タイミングの問題により正しく表示できないLEDパネルである可能性があります。

パターン6では横1本の白線が上から下まで1ピクセルずつ移動します。このパターンでは対応しているLEDパネルかどうかを確認することができます。もし、白線が2本かそれ以上表示されたり、白線が一部の範囲を飛ばして移動したり、横一直線ではなく途中で途切れたりする場合は、解像度の設定が間違っている可能性があります。もし、解像度の設定が正しいにも関わらず前述のような表示になる場合は、対応していないLEDパネルになります。

まとめると、図1.3.7-1のパターンがすべて正しく表示されるLEDパネルは対応しているLEDパネルであり、表示順番や色が入れ替わっている程度であれば、RGBピンスワップ機能を使用することで対応が可能です。表示が崩れる場合はまずは解像度設定が間違っていないかを確認し、解像度設定が正しいにもかかわらず表示が崩れる場合は非対応のLEDパネルです。

1.4 SPI通信

1.4.1 SPI信号タイミング

図1.4.1-1にSPIのタイミングチャートを示します。MSBFIRSTのSPI_MODE0フォーマットとなっております。

図1.4.1-1 SPIタイミングチャート

一般的なSPIと異なるのは、D/C端子があることです。これはMOSIに送られてきた信号がデータなのかコマンドなのか判別する端子になります。

データもしくはコマンドを送信している途中でD/C端子をHIGHもしくはLOWに変化させることはできません。

SPIのクロック周波数は54MHzまでは動作することが確認できています。

.1.4.2 必要な待機時間

本LEDパネルコントローラでは、D/C端子が変化した後、コマンド4バイト分を受信した後、指定分のデータを受信した後、電源投入直後、それぞれ待機時間が必要になります。表 1.4.2-1に必要な待機時間の一覧、図 1.4.2-1にタイミングチャートを記載します。

表 1.4.2-1 必要な待機時間

名前	タイミング	最小待機時間
ta	D/C端子変化後	10 μs
tb	コマンド4バイト受信後	コマンドによって変化
tc	指定された描画範囲分のデータ受信後	60 μs
td	電源投入後	20 ms

図 1.4.2-1 待機時間タイミングチャート

taは10 μs、tcは60 μs、tdは20 msで固定ですが、tbはコマンドの内容によって必要な待機時間が変化します。

これらの待機時間を守らなかった場合、正常に動作しない可能性があります。

.1.4.3 コマンド受信

図 1.4.3-1にコマンドを受信する際の入力信号のタイミングチャートを示します。

図 1.4.3-1 コマンド受信タイミングチャート

CMD_NUMやCMD_0などは8ビットの符号無し整数データを表します。

コマンドを送信するときはD/C端子をHIGHにし、ta時間以上待機しつつNSS端子をLOWにしてコマンドの送信を開始してください。

CMD_NUM	コマンド番号になります。この値によってのちに続くCMD_0からCMD_2の解釈が変化します。
CMD_0	コマンドの設定値になります。
CMD_1	

CMD_2

コマンドは4バイト固定長となっており、必ず4バイト連續で送信しなければなりません。各バイト間に待機時間は必要ありませんが、コマンド4バイト受信後はtb時間以上待機しなければD/C端子の切り替え、コマンド受信、データ受信を受け付けることができません。

表 1.4.3-1 CMD_NUM一覧

表 1.4.3-1にコマンド番号とその設定値の一覧を示します。

以下に各コマンドの詳細を記載します。

⑩ コマンド番号:0x00 描画範囲指定機能_X座標指定コマンド

このコマンドの待機時間tbはコマンドの設定内容によらず $5\ \mu s$ 一定です。

また設定値は待機時間経過後から反映されます。

	名前	初期値	設定範囲	詳細
CMD_0	描画開始X座標	0	0～ LEDパネル 横解像度-1	データを受信したときに、 描画を開始するピクセルのX 座標
CMD_1	描画終了X座標	LEDパネル 横解像度-1	0～ LEDパネル 横解像度-1	データを受信したときに、 描画を終了するピクセルのX 座標
CMD_2	ダミー	0x00	0x00	ダミーデータ

もし受信したコマンドが設定範囲内になかった場合、描画開始X座標、描画終了X座標のコマンドはすべて無視され、以前の設定値のままで動作し続けます。

描画開始X座標と描画終了X座標の設定範囲は、**描画開始Y座標 = 描画終了Y座標の時は例外として、描画開始X座標 ≤ 描画終了X座標** に制限されます。

よって以前に送信されたコマンドが、**描画開始Y座標 = 描画終了Y座標**となっていた場合、先に **描画終了X座標 ≤ 描画開始X座標** となっているコマンドを送信すると解釈されません。そのため、**いかなる場合でも必ず描画範囲指定機能_Y座標指定コマンドを先に送信したのち、描画範囲指定機能_X座標指定コマンドを送信するようにしてください。**

⑩ コマンド番号:0x02 描画範囲指定機能_Y座標指定コマンド

このコマンドの待機時間tbはコマンドの設定内容によらず5μs一定です。

また設定値は待機時間経過後から反映されます。

	名前	初期値	設定範囲	詳細
CMD_0	描画開始Y座標	0	0～ 描画終了Y 座標	データを受信したときに、 描画を開始するピクセルのY 座標
CMD_1	描画終了Y座標	LEDパネル 縦解像度-1	描画開始Y 座標 ～ LEDパネル 縦解像度-1	データを受信したときに、 描画を終了するピクセルのY 座標
CMD_2	ダミー	0x00	0x00	ダミーデータ

もし受信したコマンドが設定範囲内になかった場合、描画開始Y座標、描画終了Y座標のコマンドはすべて無視され、以前の設定値のままで動作し続けます。

描画範囲指定機能_X座標指定コマンドと描画範囲指定機能_Y座標指定コマンドでLEDパネルの描画範囲をある程度限定することができます。例として、 16×16 ピクセルのLEDパネルを使用したときに、描画開始X座標=13、描画終了X座標=8、描画開始Y座標=4、描画終了Y座標=12というコマンドを受信した後にデータ受信を行った際のLEDパネルに描画される範囲を図 1.4.3-2 に示します。



図 1.4.3-

2 LEDパネルの描画範囲限定例

注意すべきは、矩形で描画範囲を限定できるわけではないという点です。LEDパネルコントローラのフレームバッファは、単純に各ピクセルのRGB値がLEDパネルの座標(0,0)から横並びになった構造をしており、描画を開始、終了するピクセルというのはこのバッファのある特定のアドレスから始まり特定のアドレスで終わる範囲にデータを書き込むという意味になっています。

また、LEDパネルコントローラに使用しているマイコンの制約により、
21845ピクセル以上を描画範囲として指定することができません。もし、
21845ピクセル以上の範囲を描画する場合は、複数回に分けて描画する必要
があります。

例:横256ピクセル、縦128ピクセルの範囲(合計32768ピクセル)を2回に分け
て描画する際の手順

- ① 描画開始Y座標を0、描画終了Y座標を63として描画範囲指定機能_Y座
標指定コマンドを送信する。
- ② 描画開始X座標を0、描画終了X座標を255として描画範囲指定機能_X座
標指定コマンドを送信する。
- ③ 座標(0,0)から座標(255,63)までのデータを送信する。(合計16384ピクセ
ル)
- ④ 描画開始Y座標を64、描画終了Y座標を127として描画範囲指定機能_Y
座標指定コマンドを送信する。
- ⑤ 描画開始X座標を0、描画終了X座標を255として描画範囲指定機能_X座
標指定コマンドを送信する。
- ⑥ 座標(0,64)から座標(255,127)までのデータを送信する。(合計16384ピク
セル)

⑩ コマンド番号:0x04 LEDパネルONOFFと輝度調整コマンド

このコマンドの待機時間tbはコマンドの設定内容によらず5 μs一定です。

また設定値は待機時間経過後から反映されます。

	名前	初期値	設定範囲	詳細
CMD_0	LEDパネル ONOFF	0x01	0x00,0x01	0x00:LEDパネルOFF 0x01:LEDパネルON
CMD_1	輝度ソース 切り替え	0x00	0x00～ 0x02	0x00:LEDパネルコントローラ上の半固定抵抗により輝度調整 0x01:コマンドにより輝度調整 0x02:半固定抵抗値とコマンド値の乗算
CMD_2	輝度値	0	0～255	輝度調整コマンド

⑩ コマンド番号:0x05 PWM分解能とガンマ補正值変更コマンド

このコマンドの待機時間tbはコマンドの設定内容によらず1.5 ms一定です。

また設定値は待機時間経過後から反映されます。

	名前	初期値	設定範囲	詳細
CMD_0	PWM分解能	8	1～12	PWM分解能
CMD_1	ガンマ補正值	22	5～50	ガンマ補正值 実際の補正值は設定値の1/10になります
CMD_2	ダミー	0x00	0x00	ダミーデータ

もし受信したコマンドが設定範囲内になかった場合、PWM分解能とガンマ補正值のコマンドはすべて無視され、以前の設定値のままで動作し続けます。

PWM分解能とガンマ補正值変更コマンドを受信すると、LEDパネルコントローラは変換テーブルの再生成を行います。この変換テーブルの再生成には1.5 msほどかかるため、LEDパネルに一瞬ちらつきが発生する場合があります。

PWM分解能とガンマ補正值変更コマンドはLEDパネルコントローラに接続されているすべてのLEDパネル全体、またバッファ1とバッファ2の両方に適用されます。特定の範囲のみ設定値を変更したり、バッファ1とバッファ2で異なる設定値を適用することはできません。

⑩ コマンド番号:0x06 色調補正コマンド

このコマンドの待機時間tbはコマンドの設定内容によらず1 ms一定です。

また設定値は待機時間経過後から反映されます。

	名前	初期値	設定範囲	詳細
CMD_0	赤輝度補正	255	0~255	赤輝度補正
CMD_1	緑輝度補正	255	0~255	緑輝度補正
CMD_2	青輝度補正	255	0~255	青輝度補正

色調補正コマンドを受信すると、LEDパネルコントローラは変換テーブルの再生成を行います。この変換テーブルの再生成には1 msほどかかるため、LEDパネルに一瞬ちらつきが発生する場合があります。

色調補正コマンドはLEDパネルコントローラに接続されているすべてのLEDパネル全体、またバッファ1とバッファ2の両方に適用されます。特定の範囲のみ設定値を変更したり、バッファ1とバッファ2で異なる設定値を適用することはできません。

⑩ コマンド番号:0x07 バッファ切り替えコマンド

このコマンドの待機時間tbはコマンドの設定内容によらず5 μs一定です。

また表示バッファ切り替えコマンドはコマンドを受信し、LEDパネル全領域スキャン終了後の次のスキャンから反映されます。書き込みバッファ切り替えコマンドは待機時間経過後から反映されます。

	名前	初期値	設定範囲	詳細
CMD_0	表示バッファ	0x00	0x00,0x01	LEDパネルに表示するバッファ 0x00:バッファ1選択 0x01:バッファ2選択
CMD_1	書き込みバッファ	0x00	0x00,0x01	データ受信の書き込み先バッファ 0x00:バッファ1選択 0x01:バッファ2選択
CMD_2	ダミー	0x00	0x00	ダミーデータ

バッファ切り替えコマンドは、最下位1ビットのみを解釈しています。

書込バッファ、表示バッファはLEDパネルコントローラに接続されているすべてのLEDパネル全体に適用されます。例えばバッファ1をLEDパネルの半分に表示、バッファ2をLEDパネルのもう半分に表示するなどといった使い方はできません。

ダブルバッファ機能を用いても、LEDパネル1枚あたりの縦解像度の半分の位置(スキャンレートがLEDパネル縦解像度の半分の場合)に発生するティアリングは解消することが出来ません。これはLEDパネルのスキャンパター

ンの性質上致し方ないです。PWM分解能を下げてリフレッシュレートを高めることにより目立たなくすることは可能ですが、根本的な解決は不可能です。

また、バッファの切り替え周期はLEDパネルのスキャン周期より長くなければダブルバッファリングの恩恵を受けることができません。

LEDパネルの輝度がバッファ1を表示させているときとバッファ2を表示させているときで異なる場合があります。これはマイコンのSRAM上に配置されているフレームバッファの位置により、CPUからメモリにアクセスするときの速度がバッファ1とバッファ2で異なるために発生する現象です。PWM分解能を高めることにより目立たなくすることは可能ですが、根本的な解決は不可能です。

⑩ コマンド番号:0x08 RGBピンスワップコマンド

このコマンドの待機時間tbはコマンドの設定内容によらず5 μs一定です。

また設定値は待機時間経過後から反映されます。

	名前	初期値	設定範囲	詳細
CMD_0	赤ピンスワップ	0	0~2	0:Rピンに赤データ出力 1:Rピンに緑データ出力 2:Rピンに青データ出力
CMD_1	緑ピンスワップ	1	0~2	0:Gピンに赤データ出力 1:Gピンに緑データ出力 2:Gピンに青データ出力
CMD_2	青ピンスワップ	2	0~2	0:Bピンに赤データ出力 1:Bピンに緑データ出力 2:Bピンに青データ出力

もし受信したコマンドが設定範囲内になかった場合、RGBピンスワップコマンドは無視され、以前の設定値のままで動作し続けます。

RGBピンスワップコマンドはLEDパネルコントローラに接続されているすべてのLEDパネル全体、またバッファ1とバッファ2の両方に適用されます。特定の範囲のみ設定値を変更したり、バッファ1とバッファ2で異なる設定値を適用することはできません。

⑩ コマンド番号:0x09 上下反転コマンド

このコマンドの待機時間tbはコマンドの設定内容によらず5 μs一定です。

また設定値は待機時間経過後から反映されます。

	名前	初期値	設定範囲	詳細
CMD_0	上下反転	0x00	0x00,0x01	0x00:Y座標はLEDパネルの上から下に向かって増加 0x01:Y座標はLEDパネルの下から上に向かって増加

CMD_1	ダミー	0x00	0x00	ダミーデータ
CMD_2	ダミー	0x00	0x00	ダミーデータ

上下反転コマンドは、最下位1ビットのみを解釈しています。

上下反転コマンドはLEDパネルコントローラに接続されているすべてのLEDパネル全体、またバッファ1とバッファ2の両方に適用されます。特定の範囲のみ設定値を変更したり、バッファ1とバッファ2で異なる設定値を適用することはできません。

⑩ コマンド番号:0x0a リフレッシュレート指定コマンド

このコマンドの待機時間tbはコマンドの設定内容によらず5μs一定です。

また設定値は待機時間経過後から反映されます。

	名前	初期値	設定範囲	詳細
CMD_0	RPS×1	255	1～255	リフレッシュレートの1の位 単位はRPS
CMD_1	RPS×0.01	0	0～99	リフレッシュレートの0.01の位 単位はRPS
CMD_2	外部リフレッシュ ユ	0x00	0x00, 0x01	0x00:外部リフレッシュ無効 0x01:外部リフレッシュ有効

リフレッシュレートの値は固定小数点方式で設定します。CMD_0がリフレッシュレートの1の位、CMD_1は0.01の位になります。実際に設定されるリフレッシュレートは、CMD_0+CMD_1×0.01となります。設定範囲は1以上となります。もし設定値が1未満だった場合は1として動作します。

通常は設定値通りのリフレッシュレートで動作しますが、LEDパネルの解像度設定、PWM分解能、LEDパネルの明るさなどの設定により指定されたリフレッシュレートを維持できない場合は設定値よりも低下する場合があります。

リフレッシュレートはLEDパネルコントローラの水晶振動子の誤差などによって設定値に対して必ず誤差が発生します。またデータ受信やコマンド受信、LEDパネルコントローラ内部の処理によってリフレッシュレートは常に変動します。外部リフレッシュが無効の場合、LEDパネルコントローラが指定されたリフレッシュレートに従ってLEDパネルのリフレッシュタイミングを生成します。

外部リフレッシュが有効の場合、LEDパネルコントローラの外部リフレッシュ信号入力ピンに立ち上がりもしくは立下りエッジが入力されたタイミングでLEDパネルをリフレッシュします。詳細に関しては、「87 LEDパネルコントローラのリフレッシュタイミングを外部信号で制御したいときは」を参照してください。ただし、リフレッシュレートは255RPSを超えることはできません。またLEDパネルの解像度設定、PWM分解能、LEDパネルの明るさなどの設定によっては、入力された外部リフレッシュ信号と同じタイミングでLEDパネルをリフレッシュできない場合があります。

リフレッシュレート指定コマンドはLEDパネルコントローラに接続されているすべてのLEDパネル全体、またバッファ1とバッファ2の両方に適用されます。特定の範囲のみ設定値を変更したり、バッファ1とバッファ2で異なる設定値を適用することはできません。

⑩ コマンド番号:0x20 描画支援機能_左上座標指定コマンド

このコマンドの待機時間tbはコマンドの設定内容によらず5 μs一定です。

また設定値は待機時間経過後から反映されます。

	名前	初期値	設定範囲	詳細
CMD_0	左上X座標	0	0～255	描画支援機能を使って図形を描画する際の左上X座標 上下反転機能を有効にしているときは左下のX座標となる
CMD_1	左上Y座標	0	0～255	描画支援機能を使って図形を描画する際の左上Y座標 上下反転機能を有効にしているときは左下のY座標となる
CMD_2	ダミー	0x00	0x00	ダミーデータ

描画支援機能_左上座標指定コマンドを受信した時点では、その設定値で正しく描画できるかは評価されず、受信した設定値がそのまま書き込まれます。

⑩ コマンド番号:0x21 描画支援機能_右下座標指定コマンド

このコマンドの待機時間tbはコマンドの設定内容によらず5 μs一定です。

また設定値は待機時間経過後から反映されます。

	名前	初期値	設定範囲	詳細
CMD_0	右下X座標	0	0～255	描画支援機能を使って図形を描画する際の右下X座標 上下反転機能を有効にしているときは右上のX座標となる
CMD_1	右下Y座標	0	0～255	描画支援機能を使って図形を描画する際の右下Y座標 上下反転機能を有効にしているときは右上のY座標となる
CMD_2	ダミー	0x00	0x00	ダミーデータ

描画支援機能_右下座標指定コマンドを受信した時点では、その設定値で正しく描画できるかは評価されず、受信した設定値がそのまま書き込まれます。

⑩ コマンド番号:0x22 描画支援機能_中心座標と半径指定コマンド

このコマンドの待機時間tbはコマンドの設定内容によらず5 μs一定です。

また設定値は待機時間経過後から反映されます。

	名前	初期値	設定範囲	詳細
CMD_0	中心X座標	0	0~255	描画支援機能を使って図形を描画する際の中心X座標
CMD_1	中心Y座標	0	0~255	描画支援機能を使って図形を描画する際の中心Y座標
CMD_2	半径	0	0~255	描画支援機能を使って図形を描画する際の半径

描画支援機能_中心座標と半径指定コマンドを受信した時点では、その設定値で正しく描画できるかは評価されず、受信した設定値がそのまま書き込まれます。

⑩ コマンド番号:0x23 描画支援機能_描画色指定コマンド

このコマンドの待機時間tbはコマンドの設定内容によらず5 μs一定です。

また設定値は待機時間経過後から反映されます。

	名前	初期値	設定範囲	詳細
CMD_0	描画色赤	0	0~255	描画支援機能を使って図形を描画する際の赤色成分
CMD_1	描画色緑	0	0~255	描画支援機能を使って図形を描画する際の緑色成分
CMD_2	描画色青	0	0~255	描画支援機能を使って図形を描画する際の青色成分

⑩ コマンド番号:0x25 描画支援機能_透過度指定コマンド

このコマンドの待機時間tbはコマンドの設定内容によらず5 μs一定です。

また設定値は待機時間経過後から反映されます。

	名前	初期値	設定範囲	詳細
CMD_0	描画色α値	0	0~255	描画支援機能を使って図形を描画する際の透過度
CMD_1	ダミー	0x00	0x00	ダミーデータ
CMD_2	ダミー	0x00	0x00	ダミーデータ

描画色α値を指定すると、背景色(書き込み先バッファの元の色)と設定された値に従ってαブレンドを行います。

⑩ コマンド番号:0x30 描画支援機能_描画開始コマンド

このコマンドの待機時間tbはコマンドの設定内容によって変化します。この待機時間は描画するピクセル数にほぼ比例して長くなります。待機時間tbを15 ms確保すればほとんどの場合正常に描画できますが、もっと高速化したい場合は試行錯誤しながら待機時間を減らしていく必要があります。

	名前	初期値	設定範囲	詳細
CMD_0	図形選択	-	0~6	0:点描画 1:直線描画 2:四角形描画(枠のみ) 3:四角形描画(中埋め) 4:円描画(枠のみ) 5:円描画(中埋め) 6:1文字描画
CMD_1	書込先バッファ	0	0~3	0:バッファ切り替えコマンド(CMD_NUM=0x07)で表示バッファに指定したバッファに書き込む 1:バッファ切り替えコマンド(CMD_NUM=0x07)で書込みバッファに指定したバッファに書き込む 2:バッファ1に書き込む 3:バッファ2に書き込む
CMD_2	文字コード	-	0x20~0xdf	図形選択で1文字描画を選択した際、この値の文字コードに相当する文字を描画する

受信した図形選択コマンドが設定範囲内になかった場合は何も描画しません。

書込み先バッファコマンドが設定範囲内になかった場合は、以前の設定値のまま選択された図形を描画します。

図形選択コマンドに1文字描画を選択していた際に、文字コードが設定範囲内になかった場合は何も描画しません。図形選択コマンドに1文字描画以外を選択していた場合は文字コードに設定された値は無視されます。

1文字描画で描画される文字は横5ピクセル、縦8ピクセルです。

描画ピクセル数が極端に多いと、描画に時間がかかるためLEDパネルにちらつきが発生する場合があります。

点を描画する手順

- ①描画支援機能_左上座標指定コマンド(CMD_NUM=0x20)で点の座標を指定します。

- ②描画支援機能_描画色指定コマンド(CMD_NUM=0x23)で描画する点の色を指定します。
- ③描画支援機能_透過度指定コマンド(CMD_NUM=0x25)で描画する点の透過度を指定します。
- ④描画支援機能_描画開始コマンド(CMD_NUM=0x30)の図形選択を0とし、書込み先バッファを任意に選択し、図形描画を開始します。文字コードは無視します(適当に0としておけばよいです)。

直線を描画する手順

- ①描画支援機能_左上座標指定コマンド(CMD_NUM=0x20)で直線の始点座標を指定します。
- ②描画支援機能_右下座標指定コマンド(CMD_NUM=0x21)で直線の終点座標を指定します。
- ③描画支援機能_描画色指定コマンド(CMD_NUM=0x23)で描画する直線の色を指定します。
- ④描画支援機能_透過度指定コマンド(CMD_NUM=0x25)で描画する直線の透過度を指定します。
- ⑤描画支援機能_描画開始コマンド(CMD_NUM=0x30)の図形選択を1とし、書込み先バッファを任意に選択し、図形描画を開始します。文字コードは無視します(適当に0としておけばよいです)。

四角形(枠のみ)もしくは四角形(中埋め)を描画する手順

- ①描画支援機能_左上座標指定コマンド(CMD_NUM=0x20)で四角形の左上座標を指定します。
- ②描画支援機能_右下座標指定コマンド(CMD_NUM=0x21)で四角形の右下座標を指定します。
- ③描画支援機能_描画色指定コマンド(CMD_NUM=0x23)で描画する四角形の色を指定します。
- ④描画支援機能_透過度指定コマンド(CMD_NUM=0x25)で描画する四角形の透過度を指定します。
- ⑤描画支援機能_描画開始コマンド(CMD_NUM=0x30)の図形選択を2もしくは3とし、書込み先バッファを任意に選択し、図形描画を開始します。文字コードは無視します(適当に0としておけばよいです)。

円(枠のみ)もしくは円(中埋め)を描画する手順

- ①描画支援機能_左上座標指定コマンド(CMD_NUM=0x22)で円の中心座標と半径を指定します。
- ②描画支援機能_描画色指定コマンド(CMD_NUM=0x23)で描画する円の色を指定します。
- ③描画支援機能_透過度指定コマンド(CMD_NUM=0x25)で描画する円の透過度を指定します。
- ④描画支援機能_描画開始コマンド(CMD_NUM=0x30)の図形選択を4もしくは5とし、書込み先バッファを任意に選択し、図形描画を開始します。文字コードは無視します(適当に0としておけばよいです)。

1文字を描画する手順

- ①描画支援機能_左上座標指定コマンド(CMD_NUM=0x20)で文字の左上座標を指定します。
- ②描画支援機能_描画色指定コマンド(CMD_NUM=0x23)で描画する文字の色を指定します。
- ③描画支援機能_透過度指定コマンド(CMD_NUM=0x25)で描画する文字の透過度を指定します。
- ④描画支援機能_描画開始コマンド(CMD_NUM=0x30)の図形選択を6とし、書込み先バッファを任意に選択し、文字コードに表 1.4.3-2の文字コード表を参



表 1.4.3-2 文

字コード表

照して描画したい文字を指定し、図形描画を開始します。

⑩ コマンド番号:0x40 画面コピー機能_コピー元左上座標指定コマンド

このコマンドの待機時間tbはコマンドの設定内容によらず5 μs一定です。

また設定値は待機時間経過後から反映されます。

	名前	初期値	設定範囲	詳細
CMD_0	コピー元 左上X座標	0	0～ コピー元 右下X座標	コピー元の左上X座標 上下反転機能を有効にして いるときは左下のX座標とな る
CMD_1	コピー元 左上Y座標	0	0～ コピー元 右下Y座標	コピー元の左上Y座標 上下反転機能を有効にして いるときは左下のY座標とな る
CMD_2	ダミー	0x00	0x00	ダミーデータ

画面コピー機能_コピー元左上座標指定コマンドを受信した時点では、そ
の設定値で正しくコピーできるかは評価されず、受信した設定値がそのまま
書き込まれます。

⑩ コマンド番号:0x41 画面コピー機能_コピー元右下座標指定コマンド

このコマンドの待機時間tbはコマンドの設定内容によらず5 μs一定です。

また設定値は待機時間経過後から反映されます。

	名前	初期値	設定範囲	詳細
CMD_0	コピー元 右下X座標	0	コピー元 左上X座標 ～ LEDパネル 横解像度-1	コピー元の右下X座標 上下反転機能を有効にして いるときは右上のX座標とな る
CMD_1	コピー元 右下Y座標	0	コピー元 左上Y座標 ～ LEDパネル 縦解像度-1	コピー元の右下Y座標 上下反転機能を有効にして いるときは右上のY座標とな る
CMD_2	ダミー	0x00	0x00	ダミーデータ

画面コピー機能_コピー元右下座標指定コマンドを受信した時点では、そ
の設定値で正しくコピーできるかは評価されず、受信した設定値がそのまま
書き込まれます。

⑩ コマンド番号:0x42 画面コピー機能_コピー先左上座標指定コマンド

このコマンドの待機時間tbはコマンドの設定内容によらず5μs一定です。

また設定値は待機時間経過後から反映されます。

	名前	初期値	設定範囲	詳細
CMD_0	コピー先 左上X座標	0	0～ LEDパネル 横解像度-1	コピー先の左上X座標 上下反転機能を有効にして いるときは左下のX座標とな る
CMD_1	コピー先 左上Y座標	0	0～ LEDパネル 縦解像度-1	コピー先の左上Y座標 上下反転機能を有効にして いるときは左下のY座標とな る
CMD_2	ダミー	0x00	0x00	ダミーデータ

画面コピー機能_コピー先左上座標指定コマンドを受信した時点では、その設定値で正しくコピーできるかは評価されず、受信した設定値がそのまま書き込まれます。

コピー元範囲とコピー先が重複していたとしても問題なくコピーできます。またコピー先の範囲がLEDパネルからはみ出していた場合は、はみ出でていない範囲のみをコピーします。

⑩ コマンド番号:0x50 画面コピー機能_コピー開始コマンド

このコマンドの待機時間tbはコマンドの設定内容によって変化します。この待機時間はコピーするピクセル数にほぼ比例して長くなります。待機時間tbを3 ms確保すればほとんどの場合正常にコピーできますが、もっと高速化したい場合は試行錯誤しながら待機時間を減らしていく必要があります。

	名前	初期値	設定範囲	詳細
CMD_0	コピー元バッファ	0	0~3	0:バッファ切り替えコマンド(CMD_NUM=0x07)で表示バッファに指定したバッファからコピーする 1:バッファ切り替えコマンド(CMD_NUM=0x07)で書込みバッファに指定したバッファからコピーする 2:バッファ1からコピーする 3:バッファ2からコピーする
CMD_1	コピー先バッファ	0	0~3	0:バッファ切り替えコマンド(CMD_NUM=0x07)で表示バッファに指定したバッファにコピーする 1:バッファ切り替えコマンド(CMD_NUM=0x07)で書込みバッファに指定したバッファにコピーする 2:バッファ1にコピーする 3:バッファ2にコピーする
CMD_2	ダミー	0x00	0x00	ダミーデータ

コピー元、コピー先バッファコマンドが設定範囲内になかった場合は、以前の設定値のままコピーします。

もし画面コピー機能_コピー元左上座標指定コマンドと画面コピー機能_コピー元右下座標指定コマンド、画面コピー機能_コピー先左上座標指定コマンドにて設定した設定値が範囲内になかった場合、コピーは実行されません。

コピーするピクセル数が極端に多いと、コピーに時間がかかるためLEDパネルにちらつきが発生する場合があります。

画面をコピーする手順

- ①画面コピー機能_コピー元左上座標指定コマンド(CMD_NUM=0x40)でコピー元の左上座標を指定します。
- ②画面コピー機能_コピー元右下座標指定コマンド(CMD_NUM=0x41)でコピー元の右下座標を指定します。

- ③画面コピー機能_コピー先左上座標指定コマンド(CMD_NUM=0x42)でコピー先の左上座標を指定します。
- ④画面コピー機能_コピー開始コマンド(CMD_NUM=0x50)のコピー元、コピー先バッファを任意に選択し、画面コピーを開始します。

.1.4.4 データ受信

図 1.4.4-1 データ受信タイミングチャート

図 1.4.4-1にデータを受信する際の入力信号のタイミングチャートを示します。

R0、G0、B0などは符号無し整数8ビットのデータを表します。

データを送信するときはD/C端子をLOWにし、ta時間以上待機しつつNSS端子をLOWにしてデータの送信を開始してください。

	名前	初期値	設定範囲	詳細
Rn	赤輝度値	0	0~255	1ピクセルの赤輝度値
Gn	緑輝度値	0	0~255	1ピクセルの緑輝度値
Bn	青輝度値	0	0~255	1ピクセルの青輝度値

データはRGBの3バイトで1ピクセルを描画します。最初に3バイトのデータを受信すると、コマンド送信で設定した描画開始ピクセルを描画し、X座標が1増加します。次の3バイトで次の1ピクセルを描画します。X座標がLEDパネル横解像度-1になると、X座標は0になり、Y座標が1増加した場所のピクセルを描画します。そうしてコマンド送信で設定した描画終了ピクセルまで描画し終わると再び描画開始ピクセルの座標まで戻り、繰り返し描画し続けます。

一度のデータ送信で必ず描画範囲指定機能によって指定した描画範囲分のデータをすべて送信しなければなりません。データ送信が途中で中断された場合、データ化けが発生しLEDパネルが正しく描画されない可能性があります。

また、LEDパネルコントローラに使用しているマイコンの制約により、65535バイト(21845ピクセル)以上のデータを一度に受信することができません。65535バイト(21845ピクセル)以上のデータを送信する場合は、描画範囲指定機能を用いて複数回に分けてデータ送信を行う必要があります。詳細については、「33 コマンド受信」を参照願います。

仮に8×8のLEDパネルを用いて、描画開始X座標=5、描画終了X座標=2、描画開始Y座標=3、描画終了Y座標=5というコマンドを受信した後に、R0、G0、B0、R1、G1、B1、・・・、R13、G13、B13という順番でデータ受信を行った際にLEDパネルに描画されるピクセルの位置を図 1.4.4-2に示します。

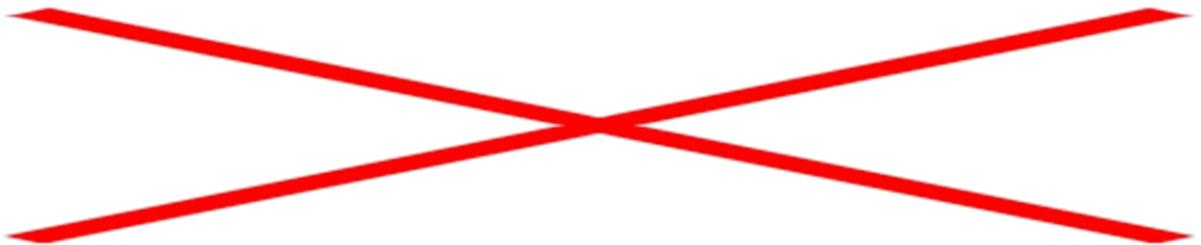


図 1.4.4-2 受信したデータと描画されるピクセルの位置関係

R13、G13、B13までデータ送信したのちにtc時間以上待機してR14、G14、B14というデータを受信すると、座標(5,3)に描画されます。

データ受信の場合、データを受信し続ける限り繰り返しピクセルを描画し続けることができますが、稀にデータを取りこぼすなどして1バイト欠けるとそれ以降のデータがすべて1バイトずれた状態でLEDパネルに描画されてしまいます。

これを防ぐため、コマンド送信で設定したピクセル描画範囲分のデータを送信し終わった後は、コマンド送信をする必要がなくともいったんtc時間以上待機したのちD/C端子をHIGHにしてコマンド受信モードにした後、ta時間以上待機してコマンドは送信せず、そのままD/C端子をLOWにしてta時間以上待機した後データ送信を

図 1.4.4-3 描画ピクセル位置をリセットする際のタイミングチャート

再開させることをお勧めします。タイミングチャートを図 1.4.4-3に示します。

こうすることでたとえ受信すべきデータを取りこぼしても、描画されるピクセルの位置が以前にコマンド送信した描画開始ピクセルまで戻されるため、正常に描画し続けることができます。

この方法は、例えばLEDパネルで動画を再生する場合など大量のデータを高速で頻繁に送信する場合に特に有効です。

.2 LEDパネルの点灯試験

.2.1 必要なもの一覧

LEDパネルの点灯試験を行うにあたって必要なものを以下に列挙します。

1. LEDパネル(20 使用可能なLEDパネルの項目にすべて適合しているもの)
2. LEDパネルコントローラ
3. LEDパネル用電源ケーブル

4. HUB75用フラットケーブル
5. 十分な電流を供給できる5V電源
6. LEDパネルコントローラ用6ピンケーブル

.2.2 電源の選定

LEDパネルは大量のLEDを使用しているため、大きな電流を消費します。もし電流容量が不足している電源を使用した場合、過負荷によって電源を壊してしまいかねません。そのため、LEDパネルに使用する電源は適切に選定しなければなりません。

LEDパネルには通常、1m²当たりの最大消費電力と平均消費電力が定められており、この値を用いて電源に必要な電流容量を求めることができます。

電流容量を求める式を式2.2に示します。

$$I = \frac{P \times W \times D \times 10^{-6}}{V} \quad \dots \text{式2.2} \quad I: \text{電流容量[A]}$$

P: LEDパネル1m²当たりの最大消費電力[W/m²]

W: LEDパネル横幅[mm]

D: LEDパネル奥行き[mm]

V: LEDパネルの電源電圧[V]

LEDパネルの電源電圧については記載がない場合がありますが通常はDC5Vです。

例として、P=800[W/m²]、W=320[mm]、D=160[mm]、V=5[V]として計算すると、

$$I = \frac{800 \times 320 \times 160 \times 10^{-6}}{5} \approx 8.2[A] \text{となります。}$$

しかし、実際には消費電力が分からないLEDパネルも多く、そういう場合は十分大きな電流容量の電源を用いてLEDパネルが全面白色で最高輝度で発光しているときの電流を実測したほうが良いかもしれません。

.2.3 接続配線

配線接続を行う前に、「23 LEDパネルの解像度設定」と「26 HUB75出力ポート構成の設定」を参照して使用するLEDパネルに合わせて適切に解像度とHUB75出力ポート構成を設定しておいてください。また「30 LEDパネルテスト機能」を参照してLEDパネルテスト機能を有効にしておいてください。

.2.3.1 全体配線図

図 2.3.1-1に全体の配線図を示します。

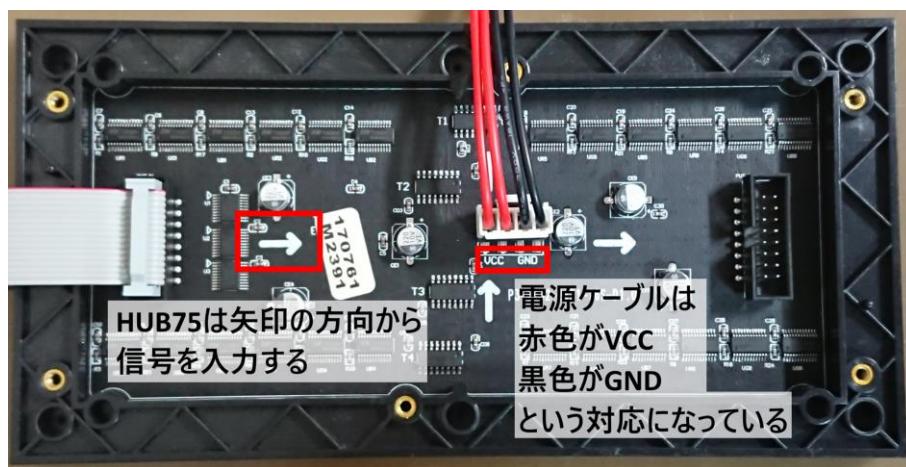


図 2.3.1-1 LEDパネル点灯試験の全体配線図

図 2.3.1-1のように配線を行います。

2.3.2 LEDパネルの電源とHUB75のフラットケーブル接続箇所について

画像 2.3.2-1にLEDパネル裏面の電源ケーブルとHUB75フラットケーブルの接続状



画像 2.3.2-1 LEDパネル裏面

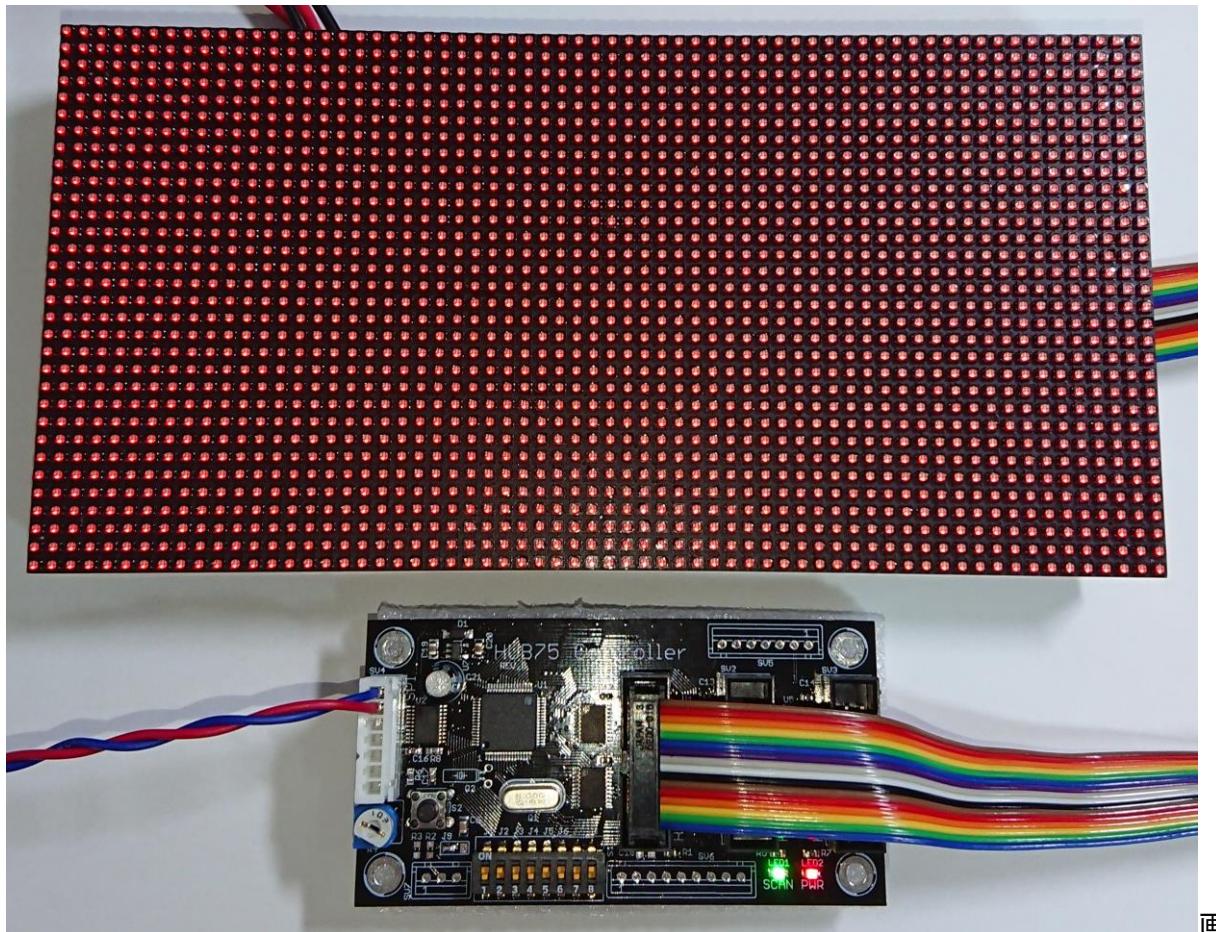
のケーブルの接続状況

況を示します。

画像 2.3.2-1のようにLEDパネルにHUB75フラットケーブルと電源ケーブルが正しく接続されていることをよく確認してください。また、LEDパネルの電源とスイッチング電源を接続する箇所もプラスとマイナスが正しいことをよく確認してください。間違えるとLEDパネルが壊れます。

.2.4 立ち上げ

解像度選択ジャンパが適切に設定され、間違いなく配線ができていることを確認



画像

2.4-1 LEDパネル裏面のケーブルの接続状況

したのち電源を投入してください。

図 1.3.7-1のようにLEDパネルの表示が一定時間ごとにテストパターンを表示していれば問題ありません。この状態で、LEDパネルコントローラ上の半固定抵抗を回してLEDパネル全体の明るさが変化することも確認しておきましょう。

もし正常に表示できない場合は、「30 LEDパネルテスト機能」を参照してください。

.3 Arduino Unoでの使用例

LEDパネルコントローラをArduino Unoから制御する場合の使用例です。なお、ArduinoIDEの導入方法やArduino Unoの使用方法、ArduinoIDEでのプログラミング方法や言語まではこの取扱説明書では解説していませんので他のサイトや書籍などを参考にしてください。最低限、Arduino UnoでLチカができるレベルの人を対象としています。

.3.1 必要なもの一覧

以下に必要なものを列挙します。

1. LEDパネル(「20 使用可能なLEDパネル」の項目にすべて適合しているもの)
2. LEDパネルコントローラ
3. LEDパネル用電源ケーブル
4. HUB75用フラットケーブル
5. 十分な電流を供給できる5V電源
6. Arduino Uno
7. LEDパネルコントローラ用6ピンケーブル
8. Arduino Unoと5V電源をつなぐケーブル
9. Arduino UnoとパソコンをつなぐUSBケーブル

.3.2 接続配線

.3.2.1 全体配線図

図 3.2.1-1にArduino Unoと接続するときの全体配線図を示します。



Arduino Unoとの接続

図 3.2.1-1

図 3.2.1-1のように配線を行います。

SPI通信が不安定で上手く動かないときは、Arduinoに限らずマスター側マイコンとLEDパネルコントローラを接続している線をツイストする、もしくはシールドケーブルを使用する(シールドはGNDに接続する)と動作が安定します。SPIクロック周波数が高ければ特に有効です。

.3.3 スケッチ例

.3.3.1 ArduinoでのSPIの使用方法

LEDパネルコントローラではSPI通信を用いてマスター側のマイコンからデータやコマンドを受け取ります。そのため、マスター側のマイコンではSPI通信が使用できるようにペリフェラルを設定する必要があります。

といつても、ArduinoでSPI通信を行うのはとても簡単です。以下にサンプルコードを示します。

```
#include <SPI.h>

#define SS 10 //スレーブセレクトピン
#define DC 9 //データ/コマンド選択ピン
//ボーレート8Mbps
SPISettings mySPISettings = SPISettings(8000000, MSBFIRST, SPI_MODE0);

void setup() {
    pinMode(SS, OUTPUT);
    digitalWrite(SS, HIGH);
    pinMode(DC, OUTPUT);
    digitalWrite(DC, LOW);
    /*スレーブセレクトピンとデータコマンド選択ピンは出力にした後それぞれHIGHとLOW
    出力にしておいてください*/
    SPI.begin();
    SPI.beginTransaction(mySPISettings);

    delay(20);
}

void loop() {
    // put your main code here, to run repeatedly:
}
```

今後はすべてこのサンプルプログラムをテンプレートにしてコードを追加していきます。

.3.3.2 データ送信のサンプルプログラム

データを送信する際は図 1.4.4-1のように、データ/コマンド選択ピンをLOWにしてから10us待機した後スレーブセレクトピンをLOWにし、データを転送します。また、1つのピクセルはRGBの3バイトで描画されるため、データ転送する際は3色分を一気に送ります。以下に1ピクセルを赤色に描画するデータを1秒ごとに送信する場合のサンプルプログラムを示します。

```
#include <SPI.h>

#define SS 10 //スレーブセレクトピン
#define DC 9 //データ/コマンド選択ピン

unsigned char color_red, color_green, color_blue;

//ボーレート8Mbps
SPISettings mySPISettings = SPISettings(8000000, MSBFIRST, SPI_MODE0);

void setup() {
    pinMode(SS, OUTPUT);
    digitalWrite(SS, HIGH);
    pinMode(DC, OUTPUT);
    digitalWrite(DC, LOW);
```

```
/*スレーブセレクトピンとデータコマンド選択ピンは出力にした後それぞれHIGHとLOW
出力にしておいてください*/
SPI.begin();
SPI.beginTransaction(mySPISettings);

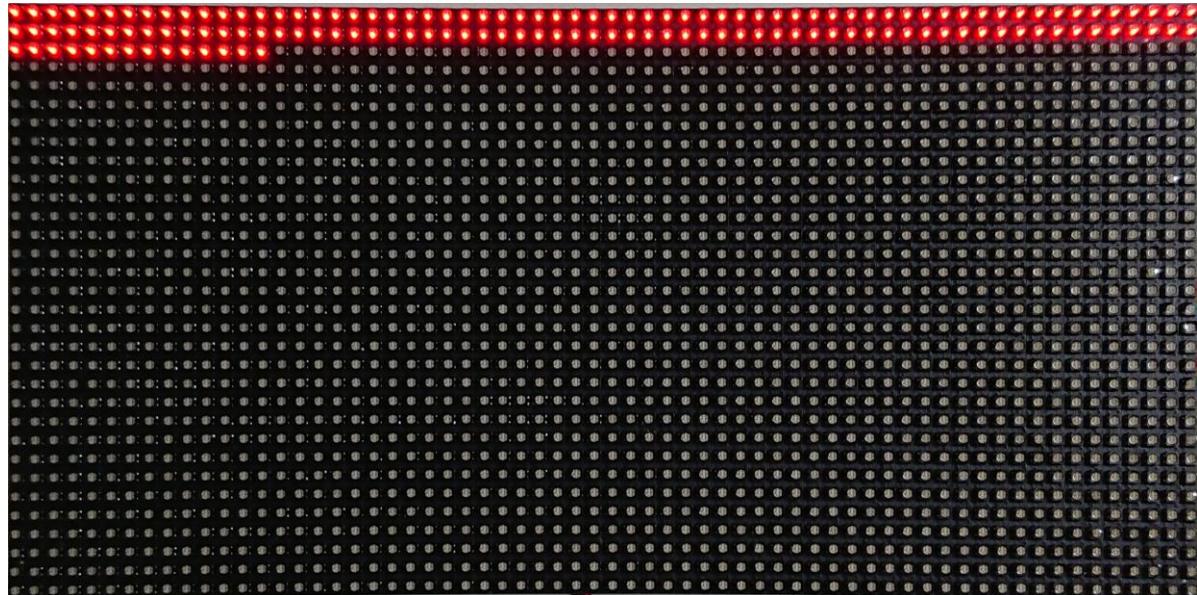
delay(20);
}

void loop() {
/*ここでピクセルを描画する色を決定*/
color_red = 255;
color_green = 0;
color_blue = 0;

digitalWrite(DC, LOW); //データ送信なのでデータ/コマンド選択ピンはLOW
delayMicroseconds(10);
digitalWrite(SS, LOW);
/*1ピクセル分のデータ送信*/
SPI.transfer(color_red);
SPI.transfer(color_green);
SPI.transfer(color_blue);
digitalWrite(SS, HIGH);

delay(1000);
}
```

上記プログラムをArduinoに書き込んだ後、LEDパネルの電源を入れてください。最初はLEDパネルの点灯試験の時と同様にLEDパネル全面が白色に発光しますが、1秒ごとに一番左上のピクセルから右に赤色で描画されていきます。そして、LEDパネルの一番右まで描画されると一段下がってまた左から赤色に描画されていくのが分かると思います。また、LEDパネルコントローラのリセットスイッチ（「17 各部の説明」を参照）を押すと全面が白色に戻り、また左上ピクセルから赤色に描画されていきます。実際に動作させている様子を画像 3.3.2-1 に示します。



画像

3.3.2-1 サンプルプログラムが動作している様子

ここで、

```
void loop() {
    /*ここでピクセルを描画する色を決定*/
    color_red = 255;
    color_green = 0;
    color_blue = 0;
```

の部分の数値を書き換えて描画色が変化することも確認しましょう。

3.3.3 LEDパネル全体を単色埋めする

「61 データ送信のサンプルプログラム」では、1ピクセル送るごとにデータ/コマンド選択ピンをLOWにし、10us待ってスレーブセレクトをLOWにして3バイトのピクセルデータを送信していましたが、もっと広い範囲を描画したい場合、3バイトのピクセルデータを複数のピクセル分一気に送信することもできます。以下に1秒ごとにLEDパネル全体をランダムな色で単色埋めするサンプルプログラムを示します。

```
#include <SPI.h>

#define SS 10 //スレーブセレクトピン
#define DC 9 //データ/コマンド選択ピン

#define MATRIXLED_Y_COUNT 32 //LEDパネルの縦解像度
#define MATRIXLED_X_COUNT 64 //LEDパネルの横解像度
```

```

unsigned char color_red, color_green, color_blue;
int x_loop, y_loop;

// ポーレート 8Mbps
SPISettings mySPISettings = SPISettings(8000000, MSBFIRST, SPI_MODE0);

void setup() {
    pinMode(SS, OUTPUT);
    digitalWrite(SS, HIGH);
    pinMode(DC, OUTPUT);
    digitalWrite(DC, LOW);
    /* スレーブセレクトピンとデータコマンド選択ピンは出力にした後それぞれHIGHとLOW
    出力にしておいてください */
    SPI.begin();
    SPI.beginTransaction(mySPISettings);

    delay(20);
}

void loop() {
    /* ここでピクセルを描画する色を決定 */
    color_red = rand() % 256;
    color_green = rand() % 256;
    color_blue = rand() % 256;
    /* 描画開始するピクセル位置をリセット */

    digitalWrite(DC, HIGH);
    delayMicroseconds(10);

    digitalWrite(DC, LOW); // データ送信なのでデータ/コマンド選択ピンはLOW
    delayMicroseconds(10);
    digitalWrite(SS, LOW);
    /* パネル全体のデータ送信 */
    for(y_loop = 0; y_loop < MATRIXLED_Y_COUNT; y_loop++){
        for(x_loop = 0; x_loop < MATRIXLED_X_COUNT; x_loop++){
            SPI.transfer(color_red);
            SPI.transfer(color_green);
            SPI.transfer(color_blue);
        }
    }
    digitalWrite(SS, HIGH);
    delay(1000);
}

```

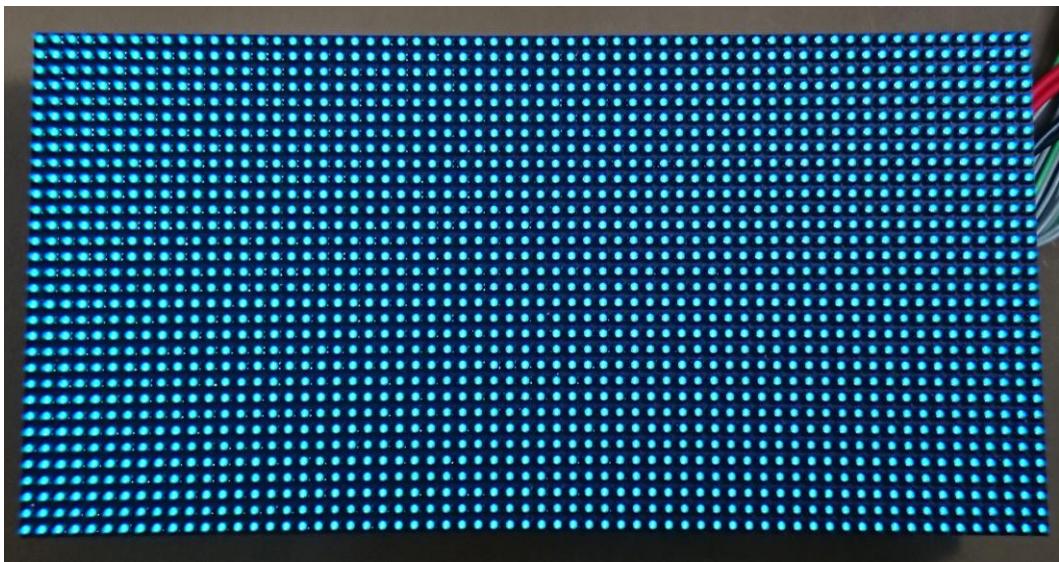
まず、上記プログラムをArduinoに書き込んで実行する前に、以下の部分を現在使用しているLEDパネルの解像度に合わせて書き換えてください。サンプルプログラムでは縦の解像度が32ピクセル、横の解像度が64ピクセルのLEDパネルを使用しています。

```

#define MATRIXLED_Y_COUNT 32 // LEDパネルの縦解像度
#define MATRIXLED_X_COUNT 64 // LEDパネルの横解像度

```

書き換えが終わったらArduinoに書き込み実行してみてください。LEDパネル全体が一秒ごとにランダムな色を表示すると思います。実際に動作している様子を画像 3.3.3-1 に示します。



画像 3.3.3-1 LED

パネル単色埋めプログラムの動作している様子

画像では水色っぽい色ですが、色はランダムなので水色になるとは限りません。では、サンプルプログラムの解説を行っていきます。

```
/*ここでピクセルを描画する色を決定*/
color_red = rand() % 256;
color_green = rand() % 256;
color_blue = rand() % 256;
```

ここでは、LEDパネルに表示する色をランダムに決定しています。rand()がランダムな数値を生成する関数で、その戻り値を256で割った余りを出すことによって0から255までのランダムな数値を得ています。

```
/*描画開始するピクセル位置をリセット*/
digitalWrite(DC, HIGH);
delayMicroseconds(10);
```

これは、LEDパネル全体を描画する前にLEDパネルのピクセル描画位置をリセットするためのものです。詳細については、「54 データ受信」を参照してください。

```

/*パネル全体のデータ送信*/
for(y_loop = 0;y_loop < MATRIXLED_Y_COUNT;y_loop++){
    for(x_loop = 0;x_loop < MATRIXLED_X_COUNT;x_loop++){
        SPI.transfer(color_red);
        SPI.transfer(color_green);
        SPI.transfer(color_blue);
    }
}

```

内側のforループでLEDパネルの横一列のピクセルを描画し、さらにそれを外側のforループでLEDパネルの縦解像度ぶん繰り返すことによってLEDパネルの全体を単色で描画しています。

.3.3.4 コマンド送信のサンプルプログラム

LEDパネルコントローラでは、コマンドを受信することによってさまざまな機能を利用できるようになります。機能の詳細については「5 機能一覧」を、コマンドの詳細については「33 コマンド受信」を参照してください。

コマンド送信は今後頻繁に使用するため、関数化しておきます。

では、サンプルプログラムを以下に示します。

```

void send_command(unsigned char command_number,
                  unsigned char command_1,
                  unsigned char command_2,
                  unsigned char command_3){
    digitalWrite(DC, HIGH); //コマンド送信なのでデータ/コマンド選択はHIGH
    delayMicroseconds(10);
    digitalWrite(SS, LOW);
    SPI.transfer(command_number);
    SPI.transfer(command_1);
    SPI.transfer(command_2);
    SPI.transfer(command_3);
    digitalWrite(SS, HIGH);
    delayMicroseconds(10);
}
}
}

```

引数については以下のようになっております。

- 引数1:コマンド番号(CMD_NUM)
- 引数2:コマンド1(CMD_1)
- 引数3:コマンド2(CMD_2)
- 引数4:コマンド3(CMD_3)

これ以降のサンプルプログラムでは、この関数を呼び出して使用することになるため「63 LEDパネル全体を単色埋める」のところまで書いたプログラムの一番下にそのまま追記しておいてください。

.3.3.5 LEDパネルの描画範囲をコマンドで指定する

それでは、「66 コマンド送信のサンプルプログラム」で追記したコマンド送信用の関数を呼び出し、LEDパネルの描画範囲を指定してみましょう。

描画開始地点のピクセルのX座標を4、Y座標を4、描画終了地点のX座標を8、Y座標を8とし、10ミリ秒ごとに1ピクセルをランダムな色に描画するデータを送信するときのサンプルプログラムを以下に示します。

```
#include <SPI.h>

#define SS 10 //スレーブセレクトピン
#define DC 9 //データ/コマンド選択ピン

#define MATRIXLED_Y_COUNT 32 //LEDパネルの縦解像度
#define MATRIXLED_X_COUNT 64 //LEDパネルの横解像度

unsigned char color_red, color_green, color_blue;
int start_x, start_y, end_x, end_y;

//ボーレート8Mbps
SPISettings mySPISettings = SPISettings(8000000, MSBFIRST, SPI_MODE0);

void setup() {
    pinMode(SS, OUTPUT);
    digitalWrite(SS, HIGH);
    pinMode(DC, OUTPUT);
    digitalWrite(DC, LOW);
    /*スレーブセレクトピンとデータコマンド選択ピンは出力にした後それぞれHIGHとLOW
    出力にしておいてください*/
    SPI.begin();
    SPI.beginTransaction(mySPISettings);

    delay(20);

    /*描画範囲をコマンドで指定*/
    start_x = 4;
    start_y = 4;
    end_x = 8;
    end_y = 8;
    /*描画範囲を設定*/
    set_draw_range(start_x, start_y, end_x, end_y);
}

void loop() {
    /*ここでピクセルを描画する色を決定*/
    color_red = rand() % 256;
    color_green = rand() % 256;
    color_blue = rand() % 256;

    digitalWrite(DC, LOW); //データ送信なのでデータ/コマンド選択ピンはLOW
    delayMicroseconds(10);
    digitalWrite(SS, LOW);
    /*1ピクセル分のデータ送信*/
    SPI.transfer(color_red);
    SPI.transfer(color_green);
    SPI.transfer(color_blue);
    digitalWrite(SS, HIGH);

    delay(10);
}

void send_command(unsigned char command_number,
```

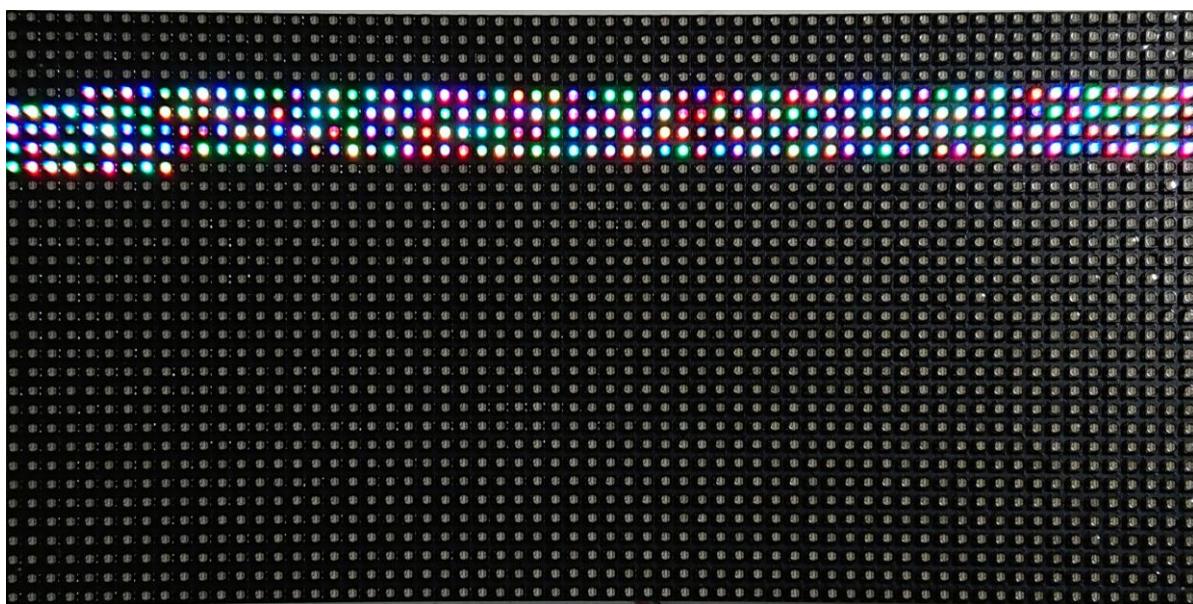
```

        unsigned char command_1,
        unsigned char command_2,
        unsigned char command_3){
digitalWrite(DC, HIGH); //コマンド送信なのでデータ/コマンド選択はHIGH
delayMicroseconds(10);
digitalWrite(SS, LOW);
SPI.transfer(command_number);
SPI.transfer(command_1);
SPI.transfer(command_2);
SPI.transfer(command_3);
digitalWrite(SS, HIGH);
delayMicroseconds(10);
}

void set_draw_range(unsigned char draw_start_pixel_x,
                    unsigned char draw_start_pixel_y,
                    unsigned char draw_end_pixel_x,
                    unsigned char draw_end_pixel_y){
//描画範囲Y座標指定
send_command(0x02, draw_start_pixel_y, draw_end_pixel_y, 0x00);
//描画範囲X座標指定
send_command(0x00, draw_start_pixel_x, draw_end_pixel_x, 0x00);
}

```

では、サンプルプログラムをArduinoに書き込んで実行してみてください。画像



画像

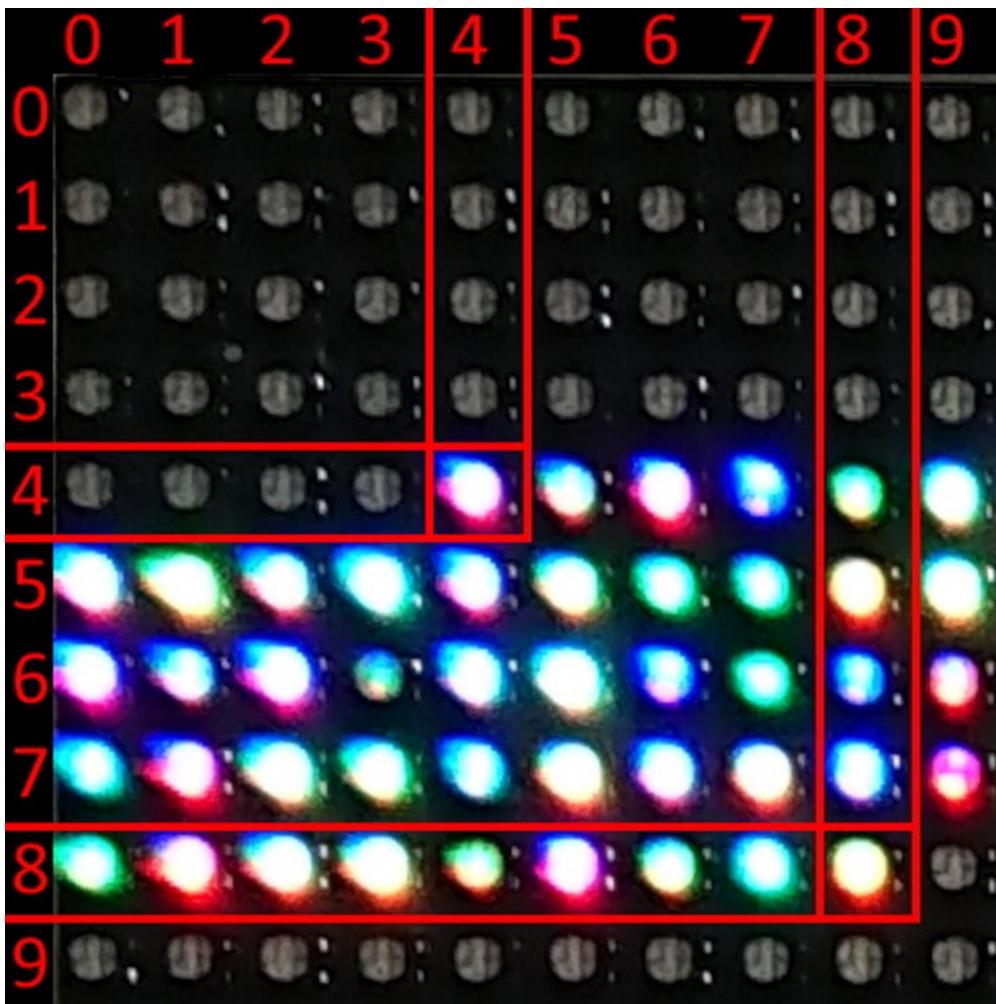
3.3.5-1 LEDパネルの描画範囲が限定されている様子

3.3.5-1に実際に動作している様子を示します。

もしうまくいかない場合は、まずLEDパネルコントローラのリセットスイッチを押し続け、次にArduino Unoのリセットスイッチを押し続け、LEDパネルコントローラのリセットスイッチを離してからArduino Unoのリセットスイッチを離してください。

ださい。コマンド送信はArduinoがリセットされた直後の1回のみしか送信されないため、このような操作が必要になります。

さて、画像 3.3.5-1の右上を拡大してみましょう。拡大画像を画像 3.3.5-2に示します。



画像 3.3.5-2 描画範囲

限定の拡大画像

確かに、X座標4、Y座標4のピクセルから始まりX座標8、Y座標8のピクセルまで描画されていることが分かります。このように、矩形で描画範囲を限定できるわけではないことに注意してください。

サンプルプログラムでは、`set_draw_range`という関数で描画範囲の指定を行っていますが、その内部はこのようになっております。

```
void set_draw_range(unsigned char draw_start_pixel_x,
                    unsigned char draw_start_pixel_y,
                    unsigned char draw_end_pixel_x,
                    unsigned char draw_end_pixel_y){
    //描画範囲Y座標指定
    send_command(0x02, draw_start_pixel_y, draw_end_pixel_y, 0x00);
    //描画範囲X座標指定
    send_command(0x00, draw_start_pixel_x, draw_end_pixel_x, 0x00);
}
```

まず描画範囲指定機能_Y座標指定コマンドで描画開始Y座標と描画終了Y座標を送信し、その後描画範囲指定機能_X座標指定コマンドで描画開始X座標と描画終了X座標を送信しています。LEDパネルコントローラでコマンドを解釈する際に設定不可能な値ははじかれるため、この関数では特に引数のチェックは行っておりません。

詳細については「33 コマンド受信」を参照してください。

.3.3.6 LEDパネルONOFFと輝度調整コマンドを利用する

本LEDパネルコントローラでは、コマンドによってLEDパネルをONOFFしたり、輝度を調整することができます。

この動作を確認するため、LEDパネルにグラデーションを表示したのち1秒ごとにLEDパネル輝度255→LEDパネル輝度0→LEDパネルOFFを繰り返すサンプルプログラムを以下に示します。

```
#include <SPI.h>

#define SS 10 //スレーブセレクトピン
#define DC 9 //データ/コマンド選択ピン

#define MATRIXLED_Y_COUNT 32 //LEDパネルの縦解像度
#define MATRIXLED_X_COUNT 64 //LEDパネルの横解像度

#define OFF 0x00
#define ON 0x01
#define SRC_VOLUME 0x00
#define SRC_COMMAND 0x01

unsigned char color_red, color_green, color_blue;

//ボーレート8Mbps
SPISettings mySPISettings = SPISettings(8000000, MSBFIRST, SPI_MODE0);

void setup() {
    pinMode(SS, OUTPUT);
    digitalWrite(SS, HIGH);
    pinMode(DC, OUTPUT);
    digitalWrite(DC, LOW);
    /*スレーブセレクトピンとデータコマンド選択ピンは出力にした後それぞれHIGHとLOW
    出力にしておいてください*/
    SPI.begin();
    SPI.beginTransaction(mySPISettings);

    delay(20);

    /*LEDパネル全体をグラデーション描画*/
    digitalWrite(DC, LOW); //データ送信なのでデータ/コマンド選択ピンはLOW
    delayMicroseconds(10);
    digitalWrite(SS, LOW);
    //縦解像度分繰り返し
    for(long loop_y = 0;loop_y < MATRIXLED_Y_COUNT;loop_y++){
        //横一列分繰り返し
        for(long loop_x = 0;loop_x < MATRIXLED_X_COUNT;loop_x++){
            //1ピクセル分のデータ送信
            SPI.transfer((unsigned char)
                         map(loop_y, 0, MATRIXLED_Y_COUNT - 1, 255, 0));
            SPI.transfer((unsigned char)
                         map(loop_y, 0, MATRIXLED_Y_COUNT - 1,
                             map(loop_x, 0, MATRIXLED_X_COUNT - 1, 255, 0), 0));
            SPI.transfer((unsigned char)
                         map(loop_y, 0, MATRIXLED_Y_COUNT - 1,
                             map(loop_x, 0, MATRIXLED_X_COUNT - 1, 255, 0), 0));
        }
    }
}
```

```

    }
}

digitalWrite(SS, HIGH);

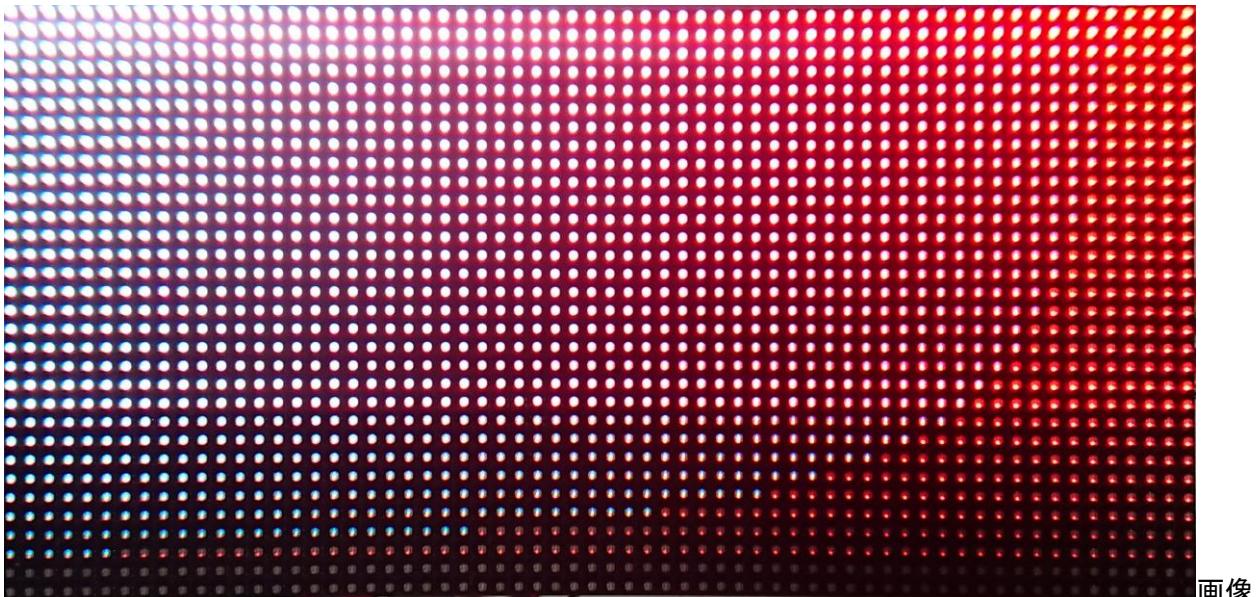
}

void loop() {
    //LEDパネルON、輝度255
    set_brightness_onoff(SRC_COMMAND, 255, ON);
    delay(1000);
    //LEDパネルON、輝度0
    set_brightness_onoff(SRC_COMMAND, 0, ON);
    delay(1000);
    //LEDパネルOFF
    set_brightness_onoff(SRC_COMMAND, 0, OFF);
    delay(1000);
}

void send_command(unsigned char command_number,
                  unsigned char command_1,
                  unsigned char command_2,
                  unsigned char command_3){
    digitalWrite(DC, HIGH); //コマンド送信なのでデータ/コマンド選択はHIGH
    delayMicroseconds(10);
    digitalWrite(SS, LOW);
    SPI.transfer(command_number);
    SPI.transfer(command_1);
    SPI.transfer(command_2);
    SPI.transfer(command_3);
    digitalWrite(SS, HIGH);
    delayMicroseconds(10);
}

void set_brightness_onoff(unsigned char brightness_source,
                         unsigned char brightness,
                         unsigned char onoff){
    //LEDパネルONOFFと輝度調整コマンドを送信
    send_command(0x04, onoff, brightness_source, brightness);
}

```



3.3.6-1 グラデーション輝度255

このサンプルプログラムを実行した様子を画像 3.3.6-1に示します。

LEDパネルの輝度が0でもLEDパネルが完全に消灯しないこと、LEDパネルをONOFFしてもバッファの内容が書き換わらないことがお分かりいただけたと思います。また、LEDパネルの輝度調整をコマンドにて行っているため、LEDパネルコントローラ上の半固定抵抗で輝度を変えなくなっていることも確認できます。

詳細については、「33 コマンド受信」を参照してください。

また、白いグラデーション部分が他と分離していることがお分かりいただけるかと思いますが、これがいわゆるバンディングという現象になります。

3.3.7 PWM分解能とガンマ補正值変更コマンドを利用する

「71 LEDパネルONOFFと輝度調整コマンドを利用する」にて、バンディングという現象が発生することを確認しましたが、これはPWM分解能を大きくすることによって解消できます。

PWM分解能の大きさの違いとガンマ補正值の大きさの違いを確認するため、LEDパネルにグラデーションを表示したのち1秒ごとにPWM分解能4bitガンマ補正值2.2→PWM分解能12bitガンマ補正值2.2→PWM分解能12bitガンマ補正值1→PWM分解能12bitガンマ補正值5を繰り返すサンプルプログラムを以下に示します。

```
#include <SPI.h>

#define SS 10 //スレーブセレクトピン
#define DC 9 //データ/コマンド選択ピン

#define MATRIXLED_Y_COUNT 32 //LEDパネルの縦解像度
#define MATRIXLED_X_COUNT 64 //LEDパネルの横解像度

//ボーレート8Mbps
```

```

SPISettings mySPISettings = SPISettings(8000000, MSBFIRST, SPI_MODE0);

void setup() {
    pinMode(SS, OUTPUT);
    digitalWrite(SS, HIGH);
    pinMode(DC, OUTPUT);
    digitalWrite(DC, LOW);
    /*スレーブセレクトピンとデータコマンド選択ピンは出力にした後それHIGHとLOW
    出力にしておいてください*/
    SPI.begin();
    SPI.beginTransaction(mySPISettings);

    delay(20);

    /*LEDパネル全体をグラデーション描画*/
    digitalWrite(DC, LOW); //データ送信なのでデータ/コマンド選択ピンはLOW
    delayMicroseconds(20);
    digitalWrite(SS, LOW);
    //縦解像度分繰り返し
    for(long loop_y = 0;loop_y < MATRIXLED_Y_COUNT;loop_y++){
        //横一列分繰り返し
        for(long loop_x = 0;loop_x < MATRIXLED_X_COUNT;loop_x++){
            //1ピクセル分のデータ送信
            SPI.transfer((unsigned char)
                map(loop_y, 0, MATRIXLED_Y_COUNT - 1, 255, 0));
            SPI.transfer((unsigned char)
                map(loop_y, 0, MATRIXLED_Y_COUNT - 1,
                    map(loop_x, 0, MATRIXLED_X_COUNT - 1, 255, 0), 0));
            SPI.transfer((unsigned char)
                map(loop_y, 0, MATRIXLED_Y_COUNT - 1,
                    map(loop_x, 0, MATRIXLED_X_COUNT - 1, 255, 0), 0));
        }
    }
    digitalWrite(SS, HIGH);
}

void loop() {
    //PWM分解能4bitガンマ補正值2.2
    set_pwm_gamma(4, 22);
    delay(1000);
    //PWM分解能12bitガンマ補正值2.2
    set_pwm_gamma(12, 22);
    delay(1000);
    //PWM分解能12bitガンマ補正值1
    set_pwm_gamma(12, 10);
    delay(1000);
    //PWM分解能12bitガンマ補正值5
    set_pwm_gamma(12, 50);
    delay(1000);
}

void send_command(unsigned char command_number,
                  unsigned char command_1,
                  unsigned char command_2,
                  unsigned char command_3){
    digitalWrite(DC, HIGH); //コマンド送信なのでデータ/コマンド選択はHIGH
}

```

```

delayMicroseconds(10);
digitalWrite(SS, LOW);
SPI.transfer(command_number);
SPI.transfer(command_1);
SPI.transfer(command_2);
SPI.transfer(command_3);
digitalWrite(SS, HIGH);
delayMicroseconds(10);
}

void set_pwm_gamma(unsigned char pwm,
                   unsigned char gamma){
    //PWM分解能とガンマ補正值変更コマンドを送信
send_command(0x05, pwm, gamma, 0x00);
delay(2);
}

```

LEDパネルの明るさは最高にすると一番わかりやすいです。サンプルプログラムを実行すると、PWM分解能が4bitの時はバンディングがはっきりと目立っていたのがPWM分解能を12bitにするとバンディングがほぼ解消されているのがお分かりいただけますかと思います。また、ガンマ補正值が1の時はコントラストが低下し全体的に白っぽくなり、ガンマ補正5の時はコントラストは上がるが全体的に暗くなっていることがお分かりいただけますかと思います。

詳細については、「33 コマンド受信」を参照してください。

.3.3.8 描画支援機能を利用する

本LEDパネルコントローラでは、コマンド送信のみでLEDパネルに図形を描画できる描画支援機能が実装されています。描画支援機能を用いてコマンド送信のみでLEDパネルにDVDスクリーンセーバーを描画するときのサンプルプログラムを以下に示します。

```

#include <SPI.h>

#define SS 10 //スレーブセレクトピン
#define DC 9 //データ/コマンド選択ピン

#define MATRIXLED_Y_COUNT 32 //LEDパネルの縦解像度
#define MATRIXLED_X_COUNT 64 //LEDパネルの横解像度

#define DOT 0
#define LINE 1
#define SQUARE_FRAME 2
#define SQUARE_FILL 3
#define CIRCLE_FRAME 4
#define CIRCLE_FILL 5
#define CHARACTER 6

#define FONT_SIZE_X 5 //フォントデータの横解像度
#define FONT_SIZE_Y 8 //フォントデータの縦解像度

#define DISPLAY_BUFFER 0
#define WRITING_BUFFER 1

```

```

#define BUFFER_1 2
#define BUFFER_2 3

unsigned char color_red, color_green, color_blue;
char text_buffer[20];
int pos_x = 0, pos_y = 0;
int diff_x = 1, diff_y = 1;

//ボーレート8Mbps
SPISettings mySPISettings = SPISettings(8000000, MSBFIRST, SPI_MODE0);

void setup() {
    pinMode(SS, OUTPUT);
    digitalWrite(SS, HIGH);
    pinMode(DC, OUTPUT);
    digitalWrite(DC, LOW);
    /*スレーブセレクトピンとデータコマンド選択ピンは出力にした後それぞれHIGHとLOW
    出力にしておいてください*/
    SPI.begin();
    SPI.beginTransaction(mySPISettings);

    delay(20);

    color_red = random(0, 255);
    color_green = random(0, 255);
    color_blue = random(0, 255);
}

void loop() {
    sprintf(text_buffer, "DVD");

    //LEDパネル全体に黒色四角形を透明度64で描画
    draw_shape(SQUARE_FILL,
               0, 0,
               MATRIXLED_X_COUNT - 1, MATRIXLED_Y_COUNT - 1,
               0, 0, 0, 64,
               0,
               DISPLAY_BUFFER);
    delay(10); //描画待機時間tb

    //ランダムな色で文字列を描画
    draw_text(text_buffer,
              pos_x, pos_y,
              color_red, color_green, color_blue, 255,
              DISPLAY_BUFFER);

    //文字列の位置を動かす
    pos_x += diff_x;
    pos_y += diff_y;

    //文字列がLEDパネルの端にぶつかったときは跳ね返らせる
    if(pos_x > MATRIXLED_X_COUNT - ((FONT_SIZE_X+1)*strlen(text_buffer))){
        diff_x = -1;
    }else if(pos_x <= 0){
        diff_x = 1;
    }
    if(pos_y > MATRIXLED_Y_COUNT - FONT_SIZE_Y){

```

```

diff_y = -1;
}else if(pos_y <= 0){
    diff_y = 1;
}

//文字列がLEDパネルの端にぶつかったとき色を変える
if((pos_x > MATRIXLED_X_COUNT - ((FONT_SIZE_X+1)*strlen(text_buffer))) ||
   (pos_x <= 0) ||
   (pos_y > MATRIXLED_Y_COUNT - FONT_SIZE_Y) ||
   (pos_y <= 0)){
    color_red = random(0, 255);
    color_green = random(0, 255);
    color_blue = random(0, 255);
}

delay(50);
}

void send_command(unsigned char command_number,
                  unsigned char command_1,
                  unsigned char command_2,
                  unsigned char command_3){
    digitalWrite(DC, HIGH); //コマンド送信なのでデータ/コマンド選択はHIGH
    delayMicroseconds(10);
    digitalWrite(SS, LOW);
    SPI.transfer(command_number);
    SPI.transfer(command_1);
    SPI.transfer(command_2);
    SPI.transfer(command_3);
    digitalWrite(SS, HIGH);
    delayMicroseconds(10);
}

void draw_shape(unsigned char shape,
                unsigned char left_top_x,
                unsigned char left_top_y,
                unsigned char right_bottom_x,
                unsigned char right_bottom_y,
                unsigned char color_r,
                unsigned char color_g,
                unsigned char color_b,
                unsigned char color_alpha,
                unsigned char character,
                unsigned char draw_buffer){

    //左上座標指定
    send_command(0x20, left_top_x, left_top_y, 0x00);
    //右下座標指定
    send_command(0x21, right_bottom_x, right_bottom_y, 0x00);
    //色指定
    send_command(0x23, color_r, color_g, color_b);
    //透過度指定
    send_command(0x25, color_alpha, 0x00, 0x00);
    //図形と書込み先バッファと文字コードを指定し描画開始
    send_command(0x30, shape, draw_buffer, character);
}

void draw_text(char *chara_array,
               unsigned char left_top_x,

```

```
    unsigned char left_top_y,
    unsigned char color_r,
    unsigned char color_g,
    unsigned char color_b,
    unsigned char color_alpha,
    unsigned char draw_buffer){

int loop_count = 0;
int pixel_pos_x = 0;

pixel_pos_x = left_top_x;

//文字列を連続して描画
while((chara_array[loop_count] |= '\0') &&
    (pixel_pos_x < MATRIXLED_X_COUNT)){
    draw_shape(CHARACTER,
        pixel_pos_x, left_top_y,
        0x00, 0x00,
        color_r, color_g, color_b, color_alpha,
        chara_array[loop_count],
        draw_buffer);
    delayMicroseconds(100); //描画待機時間tb
    pixel_pos_x += FONT_SIZE_X + 1;
    loop_count++;
}
}
```

このサンプルプログラムを実行させた様子を画像 3.3.8-1に示します。



画像

3.3.8-1 描画支援機能サンプル

なお、文字色と表示位置は変化するため、画像の通りになるとは限りません。

このサンプルプログラムでは、DVDという文字を表示させるために描画支援機能の1文字描画を、残像を表現するために描画支援機能の四角形描画(中埋め)を黒色の透明度64で描画しています。

詳細については、「33 コマンド受信」を参照してください。

3.3.9 バッファ切り替え機能を利用する

本LEDパネルコントローラにはフレームバッファが2つ内蔵されており、データを書き込むバッファとLEDパネルに表示するバッファを任意に切り替えることができます。この動作を確認するため、電源投入後に描画支援機能を用いてバッファ1に"BUFFER1"、バッファ2に"BUFFER2"と書込み、1秒ごとにLEDパネルに表示するバッファを切り替えるサンプルプログラムを以下に示します。

```
#include <SPI.h>

#define SS 10 //スレーブセレクトピン
#define DC 9 //データ/コマンド選択ピン

#define MATRIXLED_Y_COUNT 32 //LEDパネルの縦解像度
#define MATRIXLED_X_COUNT 64 //LEDパネルの横解像度

#define DOT 0
#define LINE 1
#define SQUARE_FRAME 2
#define SQUARE_FILL 3
#define CIRCLE_FRAME 4
#define CIRCLE_FILL 5
#define CHARACTER 6
```

```

#define FONT_SIZE_X 5 //フォントデータの横解像度
#define FONT_SIZE_Y 8 //フォントデータの縦解像度

#define DISPLAY_BUFFER 0
#define WRITING_BUFFER 1
#define BUFFER_1 2
#define BUFFER_2 3
//ボーレート8Mbps
SPISettings mySPISettings = SPISettings(8000000, MSBFIRST, SPI_MODE0);

void setup() {
    pinMode(SS, OUTPUT);
    digitalWrite(SS, HIGH);
    pinMode(DC, OUTPUT);
    digitalWrite(DC, LOW);
    /*スレーブセレクトピンとデータコマンド選択ピンは出力にした後それぞれHIGHとLOW
    出力にしておいてください*/
    SPI.begin();
    SPI.beginTransaction(mySPISettings);

    delay(20);

    //バッファ1にBUFF1と書き込み
    draw_text("BUFF1",
              0, 0,
              255, 255, 255,
              BUFFER_1);

    //バッファ2にBUFF2と書き込み
    draw_text("BUFF2",
              MATRIXLED_X_COUNT - ((FONT_SIZE_X+1)*strlen("BUFF2")) + 1,
              MATRIXLED_Y_COUNT - FONT_SIZE_Y + 1,
              255, 255, 255,
              BUFFER_2);
}

void loop() {
    //表示バッファをバッファ1に、書き込み先バッファをバッファ2にする
    send_command(0x07, 0, 1, 0);
    delay(1000);

    //表示バッファをバッファ2に、書き込み先バッファをバッファ1にする
    send_command(0x07, 1, 0, 0);
    delay(1000);
}

void send_command(unsigned char command_number,
                  unsigned char command_1,
                  unsigned char command_2,
                  unsigned char command_3){
    digitalWrite(DC, HIGH); //コマンド送信なのでデータ/コマンド選択はHIGH
    delayMicroseconds(10);
    digitalWrite(SS, LOW);
    SPI.transfer(command_number);
    SPI.transfer(command_1);
    SPI.transfer(command_2);
    SPI.transfer(command_3);
}

```

```

digitalWrite(SS, HIGH);
delayMicroseconds(10);
}

void draw_shape(unsigned char shape,
               unsigned char left_top_x,
               unsigned char left_top_y,
               unsigned char right_bottom_x,
               unsigned char right_bottom_y,
               unsigned char color_r,
               unsigned char color_g,
               unsigned char color_b,
               unsigned char color_alpha,
               unsigned char character,
               unsigned char draw_buffer){
    //左上座標指定
    send_command(0x20, left_top_x, left_top_y, 0x00);
    //右下座標指定
    send_command(0x21, right_bottom_x, right_bottom_y, 0x00);
    //色指定
    send_command(0x23, color_r, color_g, color_b);
    //透過度指定
    send_command(0x25, color_alpha, 0x00, 0x00);
    //図形と書込み先バッファと文字コードを指定し描画開始
    send_command(0x30, shape, draw_buffer, character);
}

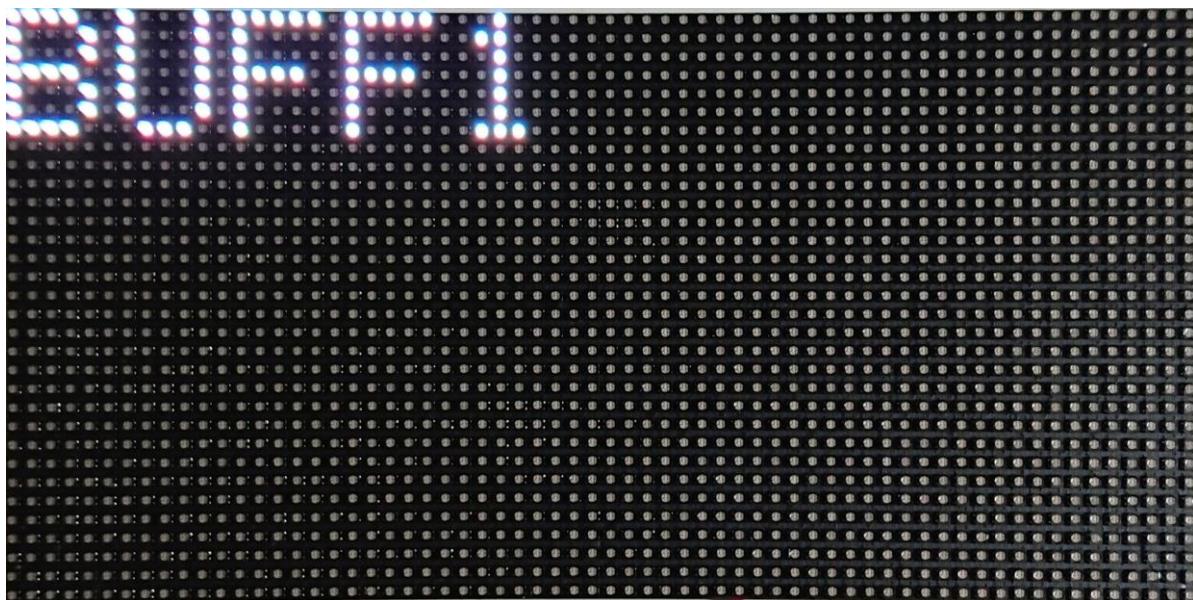
void draw_text(char *chara_array,
              unsigned char left_top_x,
              unsigned char left_top_y,
              unsigned char color_r,
              unsigned char color_g,
              unsigned char color_b,
              unsigned char color_alpha,
              unsigned char draw_buffer){
    int loop_count = 0;
    int pixel_pos_x = 0;

    pixel_pos_x = left_top_x;

    //文字列を連続して描画
    while((chara_array[loop_count] |= '\0') &&
          (pixel_pos_x < MATRIXLED_X_COUNT)){
        draw_shape(CHARACTER,
                   pixel_pos_x, left_top_y,
                   0x00, 0x00,
                   color_r, color_g, color_b, color_alpha,
                   chara_array[loop_count],
                   draw_buffer);
        delayMicroseconds(100); //描画待機時間tb
        pixel_pos_x += FONT_SIZE_X + 1;
        loop_count++;
    }
}

```

このサンプルプログラムを実行させた様子を画像 3.3.9-1と画像 3.3.9-2に示しま



画像

3.3.9-1 バッファ1の表示

す。



画像

3.3.9-2 バッファ2の表示

3.3.10 画面コピー機能を利用する

本LEDパネルコントローラはバッファの任意の範囲をコピーし任意の位置に張り付けることができる画面コピー機能が実装されています。この動作を確認するため、バッファ1に英数字を画面一面に書き込み、コピー元範囲を四角形の枠で囲っ

たのちにバッファ2にコピーし、1秒ごとにLEDパネルに表示するバッファを切り替えるサンプルプログラムを以下に示します。

```
#include <SPI.h>

#define SS 10 //スレーブセレクトピン
#define DC 9 //データ/コマンド選択ピン

#define MATRIXLED_Y_COUNT 32 //LEDパネルの縦解像度
#define MATRIXLED_X_COUNT 64 //LEDパネルの横解像度

#define DOT 0
#define LINE 1
#define SQUARE_FRAME 2
#define SQUARE_FILL 3
#define CIRCLE_FRAME 4
#define CIRCLE_FILL 5
#define CHARACTER 6

#define FONT_SIZE_X 5 //フォントデータの横解像度
#define FONT_SIZE_Y 8 //フォントデータの縦解像度

#define DISPLAY_BUFFER 0
#define WRITING_BUFFER 1
#define BUFFER_1 2
#define BUFFER_2 3
//ボーレート8Mbps
SPISettings mySPISettings = SPISettings(8000000, MSBFIRST, SPI_MODE0);

void setup() {
    pinMode(SS, OUTPUT);
    digitalWrite(SS, HIGH);
    pinMode(DC, OUTPUT);
    digitalWrite(DC, LOW);
    /*スレーブセレクトピンとデータコマンド選択ピンは出力にした後それぞれHIGHとLOW
    出力にしておいてください*/
    SPI.begin();
    SPI.beginTransaction(mySPISettings);

    delay(20);

    //バッファ1に文字列書き込み
    draw_text("0123456789",
              0, 0,
              random(0, 255), random(0, 255), random(0, 255), 255,
              BUFFER_1);
    draw_text("ABCDEFGHIJ",
              0, 8,
              random(0, 255), random(0, 255), random(0, 255), 255,
              BUFFER_1);
    draw_text("JKLMNOPQRST",
              0, 16,
              random(0, 255), random(0, 255), random(0, 255), 255,
              BUFFER_1);
    draw_text("UVWXYZabcd",
              0, 24,
              random(0, 255), random(0, 255), random(0, 255), 255,
```

```

        BUFFER_1);

//分かりやすいようにコピー元の範囲を四角形の枠で囲む
draw_shape(SQUARE_FRAME,
            3, 3,
            28, 28,
            255, 255, 255, 63,
            0,
            BUFFER_1);
delay(15);

//バッファ1の(3,3)から(28,28)までの範囲をバッファ2の(35,3)にコピー
copy_display(3, 3,
              28, 28,
              BUFFER_1,
              35, 3,
              BUFFER_2);
delay(15);
}

void loop() {
    //表示バッファをバッファ1に、書き込み先バッファをバッファ2にする
    send_command(0x07, 0, 1, 0);
    delay(1000);

    //表示バッファをバッファ2に、書き込み先バッファをバッファ1にする
    send_command(0x07, 1, 0, 0);
    delay(1000);
}

void send_command(unsigned char command_number,
                  unsigned char command_1,
                  unsigned char command_2,
                  unsigned char command_3){
    digitalWrite(DC, HIGH); //コマンド送信なのでデータ/コマンド選択はHIGH
    delayMicroseconds(10);
    digitalWrite(SS, LOW);
    SPI.transfer(command_number);
    SPI.transfer(command_1);
    SPI.transfer(command_2);
    SPI.transfer(command_3);
    digitalWrite(SS, HIGH);
    delayMicroseconds(10);
}

void draw_shape(unsigned char shape,
                unsigned char left_top_x,
                unsigned char left_top_y,
                unsigned char right_bottom_x,
                unsigned char right_bottom_y,
                unsigned char color_r,
                unsigned char color_g,
                unsigned char color_b,
                unsigned char color_alpha,
                unsigned char character,
                unsigned char draw_buffer){
    //左上座標指定
    send_command(0x20, left_top_x, left_top_y, 0x00);
}

```

```

//右下座標指定
send_command(0x21, right_bottom_x, right_bottom_y, 0x00);
//色指定
send_command(0x23, color_r, color_g, color_b);
//透過度指定
send_command(0x25, color_alpha, 0x00, 0x00);
//図形と書込み先バッファと文字コードを指定し描画開始
send_command(0x30, shape, draw_buffer, character);
}

void draw_text(char *chara_array,
               unsigned char left_top_x,
               unsigned char left_top_y,
               unsigned char color_r,
               unsigned char color_g,
               unsigned char color_b,
               unsigned char color_alpha,
               unsigned char draw_buffer){
int loop_count = 0;
int pixel_pos_x = 0;

pixel_pos_x = left_top_x;

//文字列を連続して描画
while((chara_array[loop_count] |= '\0') &&
      (pixel_pos_x < MATRIXLED_X_COUNT)){
    draw_shape(CHARACTER,
               pixel_pos_x, left_top_y,
               0x00, 0x00,
               color_r, color_g, color_b, color_alpha,
               chara_array[loop_count],
               draw_buffer);
    delayMicroseconds(100); //描画待機時間tb
    pixel_pos_x += FONT_SIZE_X + 1;
    loop_count++;
}
}

void copy_display(unsigned char src_left_top_x,
                  unsigned char src_left_top_y,
                  unsigned char src_right_bottom_x,
                  unsigned char src_right_bottom_y,
                  unsigned char src_buffer,
                  unsigned char dest_left_top_x,
                  unsigned char dest_left_top_y,
                  unsigned char dest_buffer){
//コピー元左上座標指定
send_command(0x40, src_left_top_x, src_left_top_y, 0x00);
//コピー元右下座標指定
send_command(0x41, src_right_bottom_x, src_right_bottom_y, 0x00);
//コピー先左上座標指定
send_command(0x42, dest_left_top_x, dest_left_top_y, 0x00);
//バッファ指定してコピー開始
send_command(0x50, src_buffer, dest_buffer, 0x00);
}

```

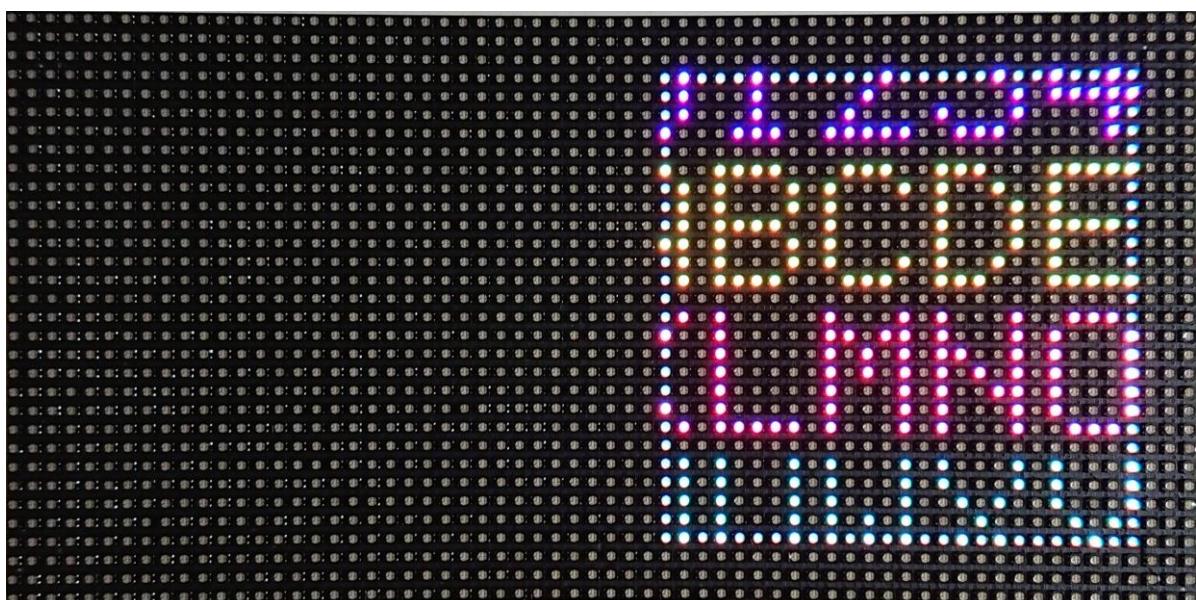
このサンプルプログラムを実行させた様子を画像 3.3.10-1と画像 3.3.10-2に示します。

なお、文字色はランダムなため、必ずしも画像通りの色になるとは限りません。



画像

3.3.10-1 バッファ1の表示



画像

3.3.10-2 バッファ2の表示

.4 その他

.4.1 LEDパネルのリフレッシュタイミング(同期信号)を得るには

LEDパネルのリフレッシュが終了する度に3.3Vもしくは0Vに変化する信号がコネクタの実装されていないSV6の7ピン目に出力されています。図 4.1-1に示します。

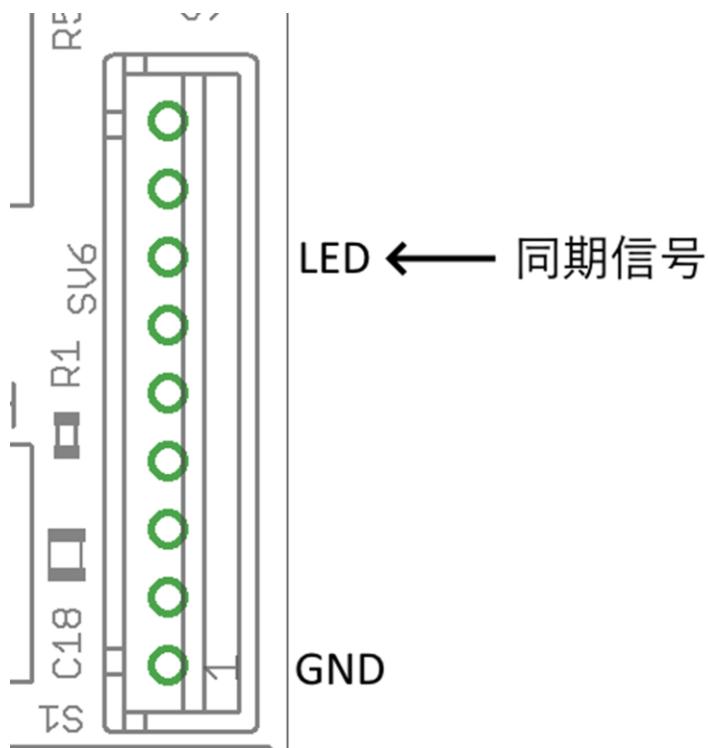


図 4.1-1 同期信号の出力端子

この端子に周波数カウンタ、もしくはオシロスコープを接続して、出力される信号の周波数を測定することでLEDパネルのリフレッシュレートを知ることができます。リフレッシュレートは測定された周波数の2倍になります。

また、この同期信号をマスター側マイコンの外部割込み端子に入力し、立上りと立下りの両エッジで割り込みをかけることで、LEDパネルのリフレッシュタイミングに同期してデータ転送などが行えるようになります。

.4.2 LEDパネルコントローラのリフレッシュタイミングを外部信号で制御したいときは

4.3 LEDパネルコントローラがBUSY状態であることを知るには

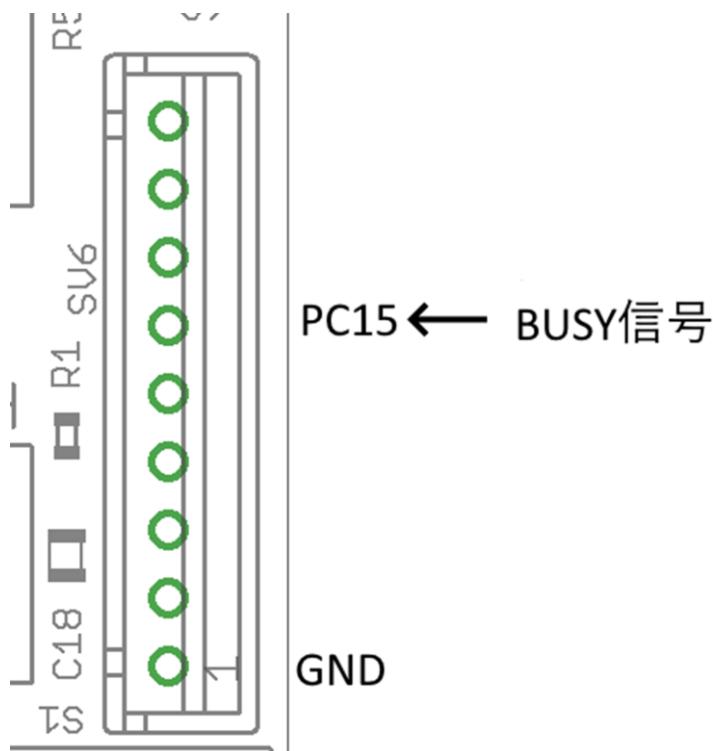


図 4.3-1 BUSY信号の出力端子

LEDパネルがREADY状態の時は3.3V、BUSY状態の時は0Vになる信号がコネクタの実装されていないSV6の6ピン目に出力されています。図 4.3-1に示します。

電源ON直後の待機時間 t_d 、D/C端子変化後の待機時間 t_a 、コマンド受信後の待機時間 t_b 、データ受信後の待機時間 t_c が発生しているときにはこのBUSY信号は0Vを出力します。このBUSY信号が0Vの時はデータ受信、コマンド受信、D/C端子を変化させることはできません。

オシロスコープでBUSY信号が0Vになっているときのパルス幅を測定することで、待機時間が変化するコマンド(コマンド番号0x30、コマンド番号0x50)の待機時間を知ることができます。

.4.4 LEDパネルコントローラを外部から強制的にリセットするには

LEDパネルコントローラをリセットする信号が、コネクタの実装されていないSV5の3ピン目に接続されています。図 4.4-1に示します。

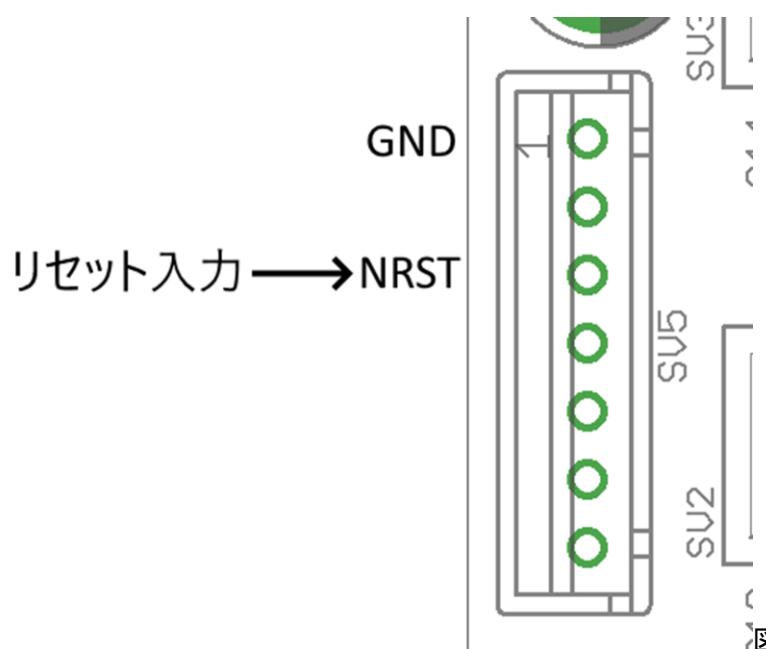


図 4.4-1 リセット入力端子

この端子をGNDに落とすことでLEDパネルコントローラがリセットされます。なお、リセット入力の最大入力電圧は3.3Vです。5Vトレラントではないため5Vの信号を直接入力することはできません。

このLEDパネルコントローラについての仕様や使用方法の解説は以上になります。

いかがだったでしょうか。もし分からぬ点、分かりにくい点、間違っている所や改善すべき点など、何かありましたら下記メールアドレス宛にお願いいたします。
strm6543@yahoo.co.jp

本文中に記載したサンプルプログラムやもう少し応用的なサンプルプログラムはこちらで公開しています。ぜひご覧になってみてください。

https://drive.google.com/drive/folders/1kmFrQVXpnWaGmdmbP23e2_CmO2kA-zAd