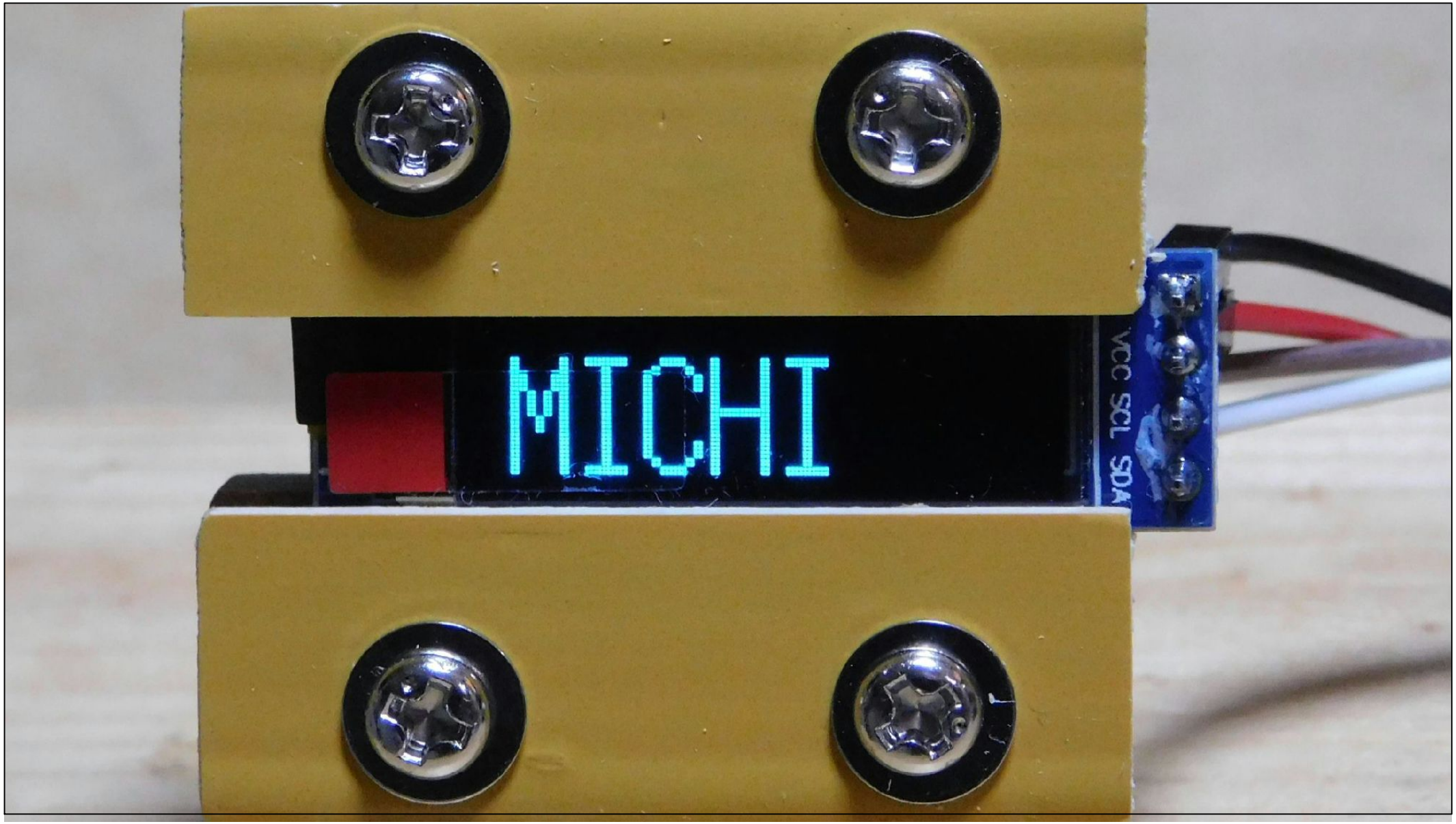
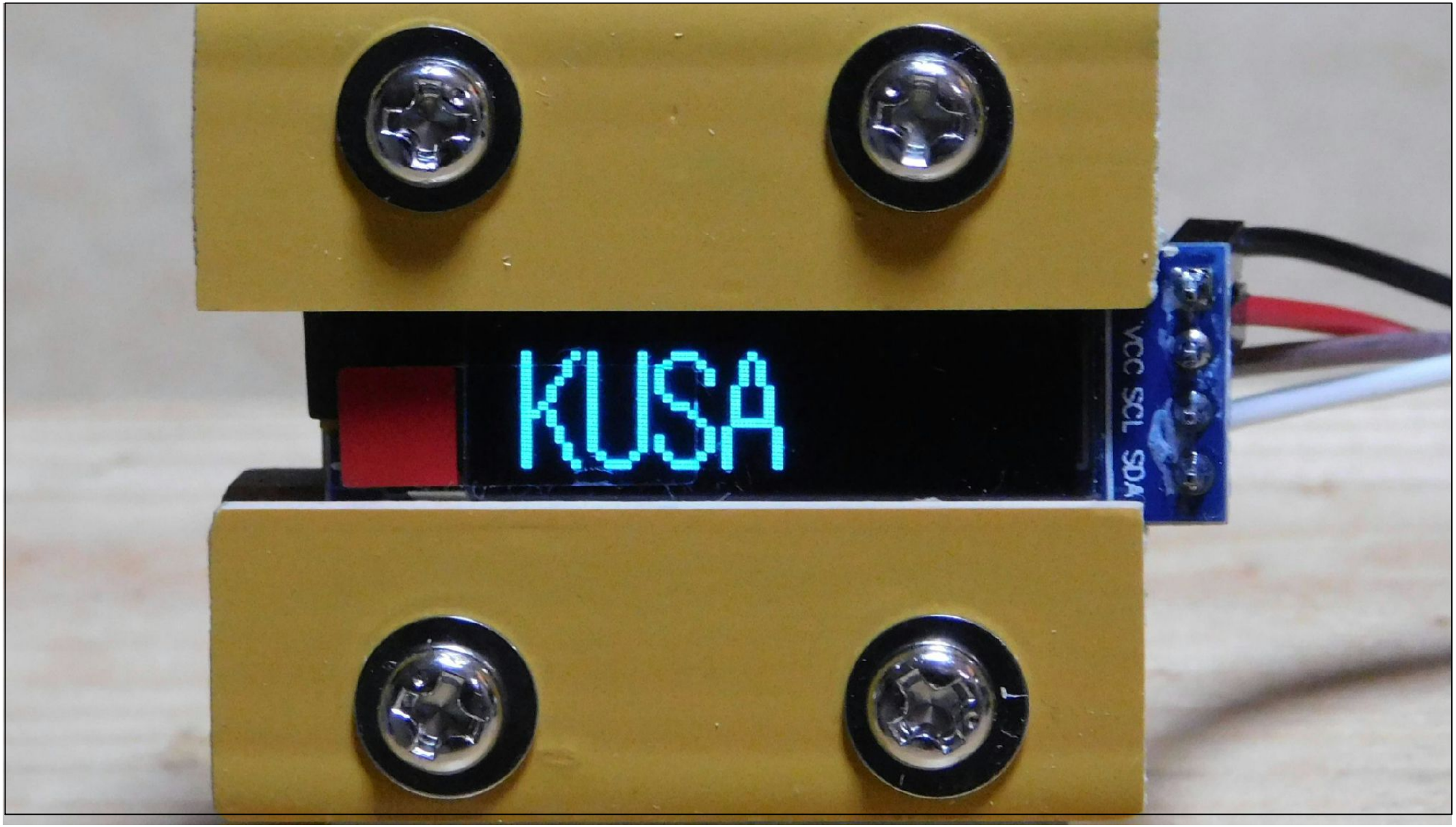
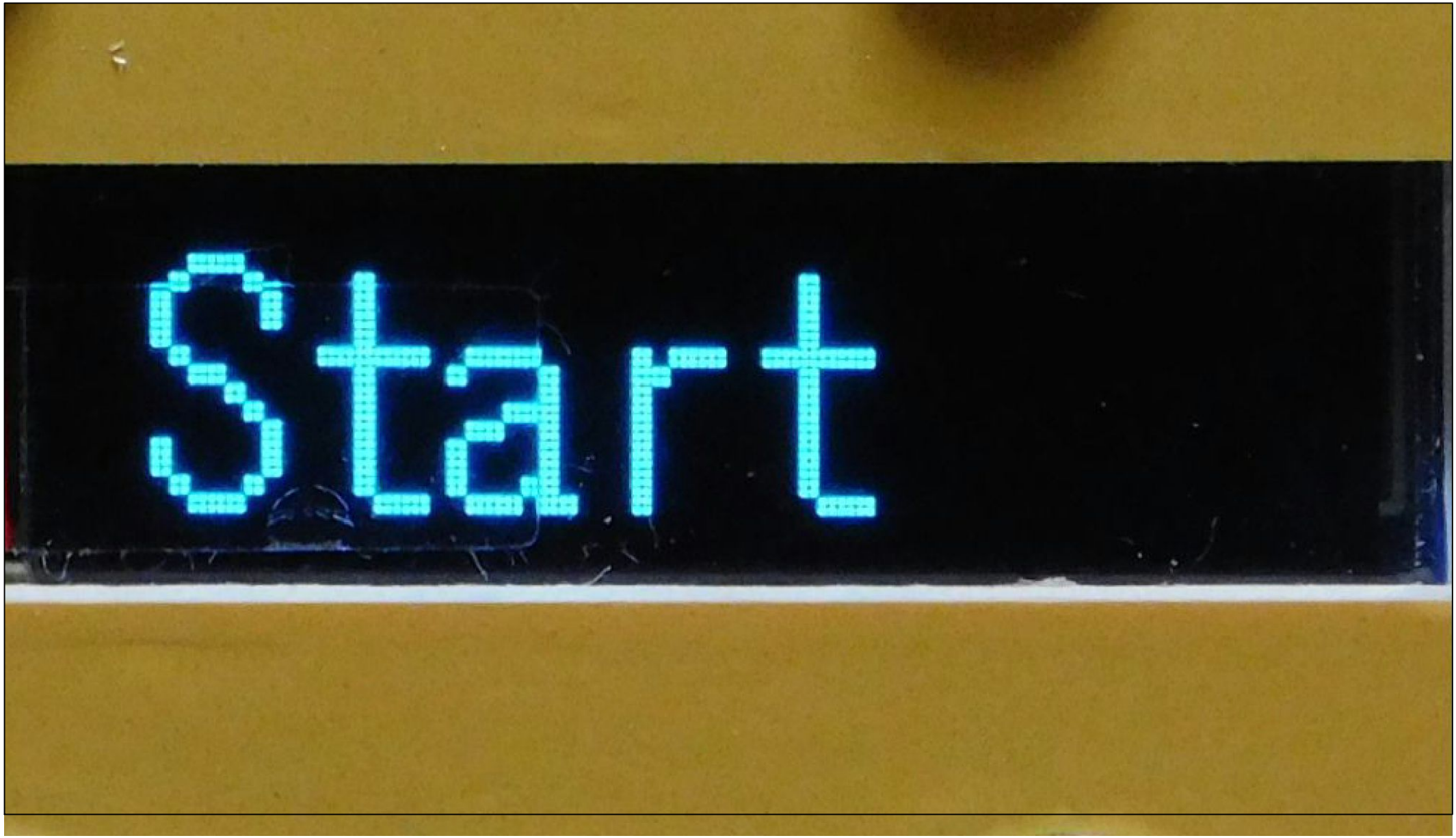

MICHI
VCC SCL SDA


KUSA
VCC SCL SDA


SYOKUNIN
VCC SCL SDA


Take
VCC SCL


Start

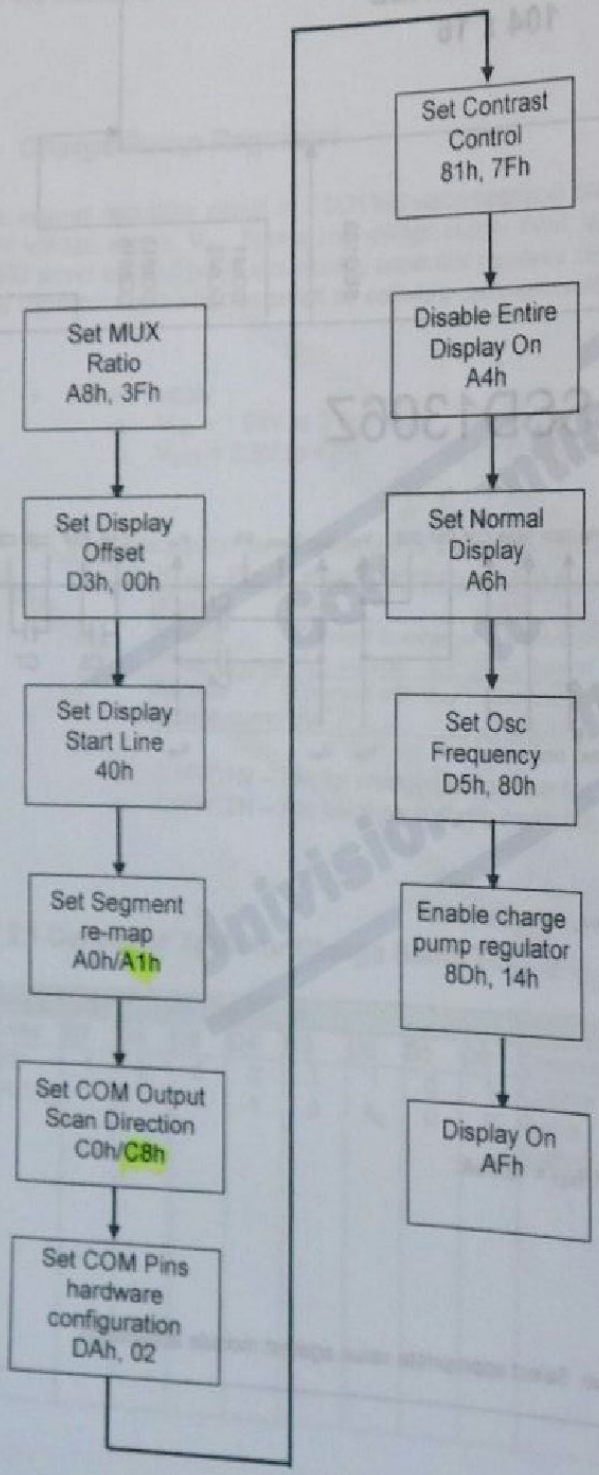

D IIC LCD Scre
Guide
ly before using it!
interface for Arduino,
ore things, such as disp
hermometer, and more
8*32 (0.91")
upply.
to Arduino
DSD TECH
Connecting the future
Quality, Responsibility, Openness
DSD TECH
Connecting the future
GND VCC SCL SDA
GND VCC SCL SDA


SOLOMON SYSTECH
SEMICONDUCTOR TECHNICAL DATA
SSD1306
今回のOLEDに使用されている
制御用IC SSD1306のデータシートです。
64ページあります。
Advance Information
128 x 64 Dot Matrix
OLED/PLED Segment/Common Driver with Controller

3 Software Configuration

SSD1306 has internal command registers that are used to configure the operations of the driver IC. After reset, the registers should be set with appropriate values in order to function well. The registers can be accessed by MPU interface in either 6800, 8080, SPI type with D/C# pin pull low or using I^2C interface. Below is an example of initialization flow of SSD1306. The values of registers depend on different condition and application.

Figure 2 : Software Initialization Flow Chart



終わりの方のページに
OLED初期化のフローチャートが
あります。


初期化のプログラムを実行すると、OLEDが、点灯します。
GDDRAMは、初期化されない様で、ランダムな点が表示されてます。


この表示は、表示消去プログラムにて
消去パターーンを 00hではなくて0Bh
(00001011b)で初期化した画面です。
1byteの Dotが、縦に並んでいるので
このように表示されます。
GND VCC SCL SDA
表示Dot数は、X:128×Y:32です。


この表示は、表示消去プログラムにて
消去パターンを 00hから順次インクリメントした
データで、書き込み表示した画面です。
GND VCC SCL SDA

Graphic Display Data Ram (GDDRAM)

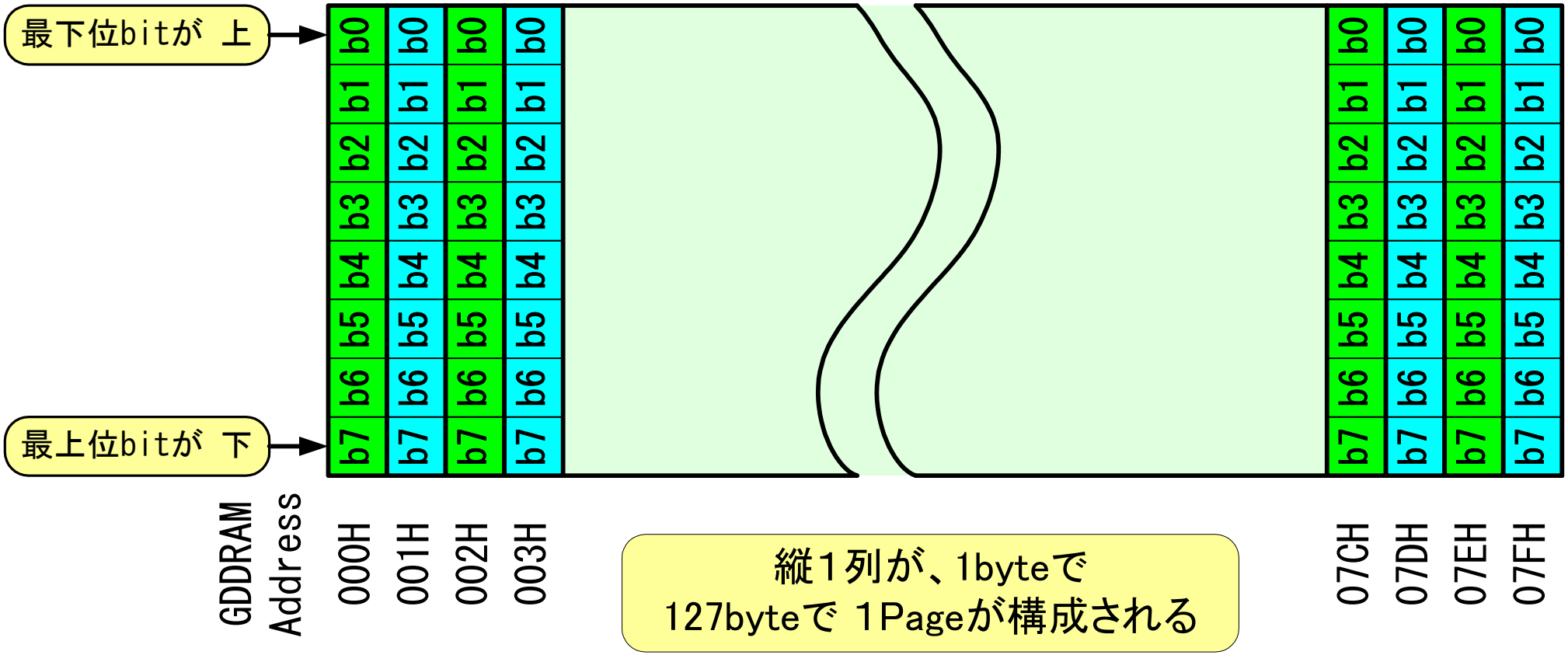
GDDRAM Address	(左上) 横Dot数：128							(右上)
00h ～ 7Fh	00h	01h	02h	～	Page0	～	7Dh	7Eh	7Fh
80h ～ FFh	80h	81h	82h	～	Page1	～	FDh	FEh	FFh
100h ～ 17Fh	100h	101h	102h	～	Page2	～	17Dh	17Eh	17Fh
180h ～ 1FFh	180h	181h	182h	～	Page3	～	1FDh	1FEh	1FFh
200h ～ 27Fh	200h	201h	202h	～	Page4	～	27Dh	27Eh	27Fh
280h ～ 2FFh	280h	281h	282h	～	Page5	～	2FDh	2FEh	2FFh
300h ～ 37Fh	300h	301h	302h	～	Page6	～	37Dh	37Eh	37Fh
380h ～ 3FFh	380h	381h	382h	～	Page7	～	3FDh	3FEh	3FFh

(左下) Page0 ～ Page7 全体で 1024 byte (右下)

128x32の OLED
使用メモリ範囲

128x64のOLEDでは
Page0～Page7まで
使用する

GDDRAMの Page内の構成　(Page0の例)



ASCII文字の BitMapFont Dataに関して

Windows Mac は、TrueTypeFontなので、今回の 用途に使えません。

BitMapFontのエディタを作って新規にデータを作る事も　可能ではありますが、94文字のデータを作るのは、結構面倒な作業です。

昔の MS-DOSでは、ASCII文字は 8x16 dotの BitMapFontでした。
大きさ的には、ちょうどいいです。

特に DOS/Vと呼ばれる MS-DOSがFontを ファイルで持っているので都合がいいです。

もう、４半世紀前の古い話で 若い人は MS-DOSは 知らないでしょうね。

で、どっかに MS-DOSシステムのバックアップが残ってないか探しました。

朝から夕方まで探した末、やっと見つけました。で、半角 8×16 dotのファイルを Getしました。
$JPNHN16.FNT というファイルです。

私も、MS-DOSに関して遥か記憶の彼方でしたが、まさか MS-DOSのファイルが、このような形で役に立つとは思いませんでした。

DOS/Vの 半角FONTファイル フォーマット

Byteデータを 16進 2桁と、ASCII文字で ダンプする FDUMP.EXEというユーティリティを昔作っていました。

これで、FONTファイルをダンプすると先頭から、単純に 16byte単位でFontDataが、入っているようです。

先頭は、ASCIIコード表の制御コードエリアの NULL です。 先頭から16x32Byteの制御コードエリアには、罫線のような記号が入ってました。

先頭から 512バイトの位置から 20hのスペースのフォントが入ってました。

その後に、ASCIIコードの 21h、22hと順次 16byte単位で文字フォントデータが並んでいる事を確認しました。

確認は、今回突貫で、ユーティリティプログラムを作成しました。

そして、そのユーティリティで、SSD1306の表示フォーマットに合う形に、BitMapを並べ直し新たなフォントファイルを作成しました。
（ File名 : BMF_8_16.dat ）

文字ばかりでは、分かりにくいので次に図で説明します。

MS-DOSの文字Font

Address	DATA							
0	b7	b6	b5	b4	b3	b2	b1	b0
1	b7	b6	b5	b4	b3	b2	b1	b0
2	b7	b6	b5	b4	b3	b2	b1	b0
3	b7	b6	b5	b4	b3	b2	b1	b0
4	b7	b6	b5	b4	b3	b2	b1	b0
5	b7	b6	b5	b4	b3	b2	b1	b0
6	b7	b6	b5	b4	b3	b2	b1	b0
7	b7	b6	b5	b4	b3	b2	b1	b0
8	b7	b6	b5	b4	b3	b2	b1	b0
9	b7	b6	b5	b4	b3	b2	b1	b0
10	b7	b6	b5	b4	b3	b2	b1	b0
11	b7	b6	b5	b4	b3	b2	b1	b0
12	b7	b6	b5	b4	b3	b2	b1	b0
13	b7	b6	b5	b4	b3	b2	b1	b0
14	b7	b6	b5	b4	b3	b2	b1	b0
15	b7	b6	b5	b4	b3	b2	b1	b0

Byte列は 90度回転した様になっていますが、同じ文字を表示できるようにbit単位で データを並べ直す必要があります。

この BitMapDataの並び変換は、パソコンで行い新たなフォントファイルを作成しました。

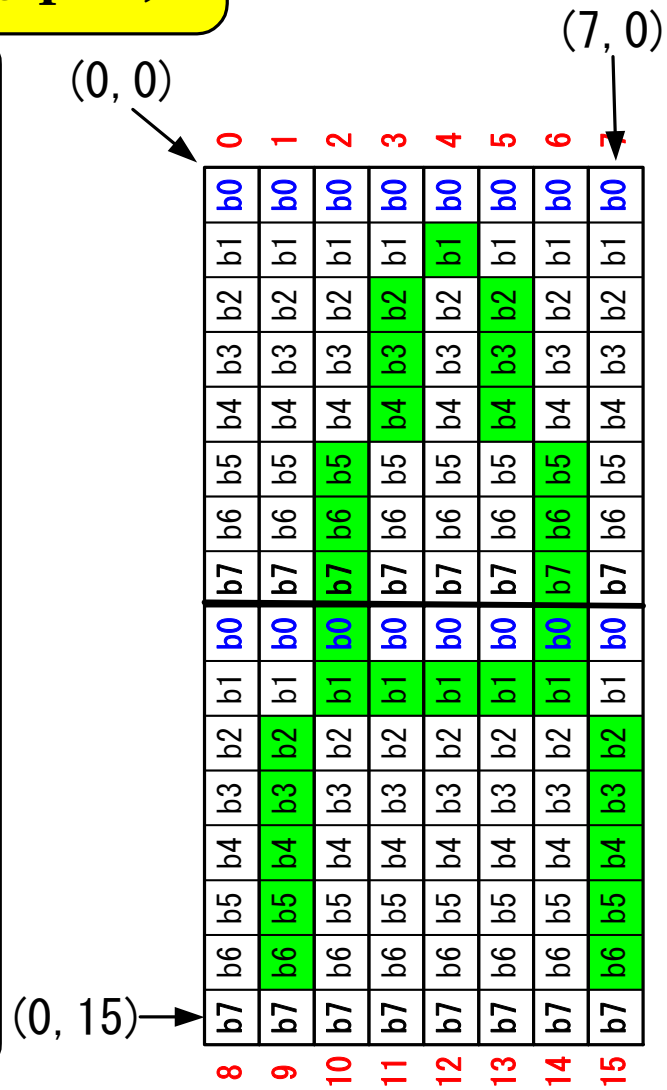
SSD1306仕様の文字Font

0	1	2	3	4	5	6	7
b0	b0	b0	b0	b0	b0	b0	b0
b1	b1	b1	b1	b1	b1	b1	b1
b2	b2	b2	b2	b2	b2	b2	b2
b3	b3	b3	b3	b3	b3	b3	b3
b4	b4	b4	b4	b4	b4	b4	b4
b5	b5	b5	b5	b5	b5	b5	b5
b6	b6	b6	b6	b6	b6	b6	b6
b7	b7	b7	b7	b7	b7	b7	b7
b0	b0	b0	b0	b0	b0	b0	b0
b1	b1	b1	b1	b1	b1	b1	b1
b2	b2	b2	b2	b2	b2	b2	b2
b3	b3	b3	b3	b3	b3	b3	b3
b4	b4	b4	b4	b4	b4	b4	b4
b5	b5	b5	b5	b5	b5	b5	b5
b6	b6	b6	b6	b6	b6	b6	b6
b7	b7	b7	b7	b7	b7	b7	b7

8	9	10	11	12	13	14	15

BitMapFont 並べ替えプログラム ソース(Delphi)

```
// バッファは、TForm1 クラス内にて、Privateで宣言している
// ------------------------------------------------------------
    buf:  array [0..15] of Byte;  // DOS/V仕様の BitMapData
    vbuf: array [0..15] of Byte;  // SSD1306仕様の BitMapData

//*********************************
//**  Vbuf ( SSD1306仕様 ) へ    **
//**   1Bit書き込み               **
//*********************************
procedure TForm1.setup_vbuf( x, y: Integer );
var
    a, b, p:   Integer;
begin
    a := 0;
    if y > 7  then  a := 8;
    a := a + x;    // 該当Dotのアドレス計算

    b := y mod 8;
    p := 1;
    p := p shl b;  // 該当DotのBit位置データ算出( p = p << b; )
    vbuf[a] := vbuf[a] OR p; // bitデータを加える
end;
```

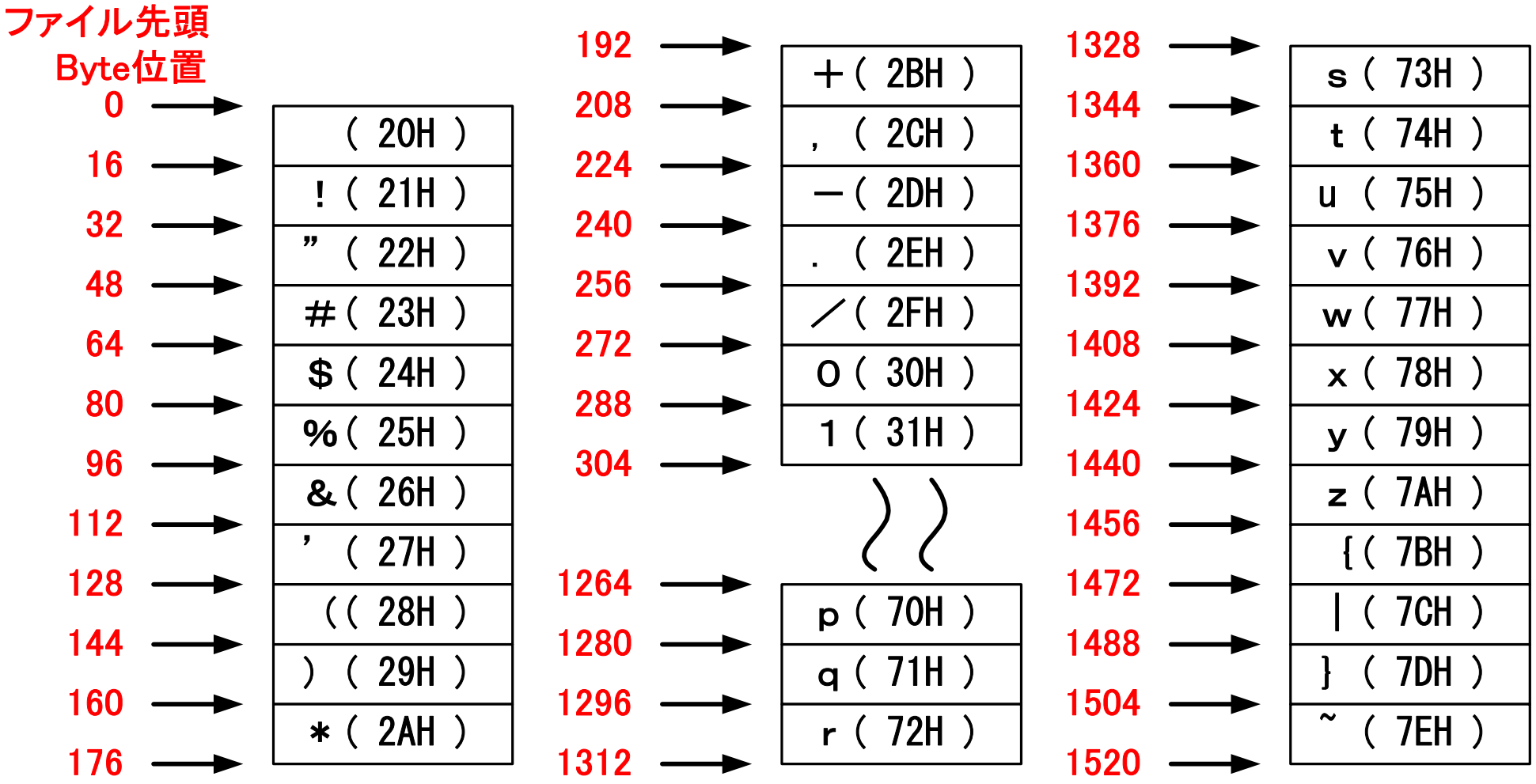


```
//*************************************************
//**  フォント グラフィック表示                   **
//** ------------------------------------------- **
//**  入力：  buf[] DOS/V仕様の BitMapData       **
//**  出力：  vbuf[] SSD1306仕様の BitMapData    **
//*************************************************
procedure  TForm1.font_disp;
var
    ptn:    Byte;
    i, j, x, y:    Integer;
begin
    for i:=0 to 15  do   vbuf[i] := 0;    // 出力バッファを一旦消去

    for i:=0 to 15  do    // Font byte数のループ（ 縦:16byteのループ ）
    begin
        y := i;
        ptn := buf[i];    // 元データから、1byteビットデータを取り出す
        for j:=0 to 7  do    // 横:8bitのループ
        begin
            x := j;
            if (ptn and $80) <> 0  then  // 該当bitが、１か確認
                setup_vbuf( x, y );    // １であれば、出力バッファに１を立てる
            ptn := ptn shl 1:    // ビット位置更新（ ptn を 1bit 左シフト ）
        end;
    end;
end;
```

タイトルがフォントグラフィック表示となってますが、ビット並び替え以外に グラフィック表示機能も実装していました。

余分な表示機能があると分かりにくいと思ったので、ビット並び替え機能だけにソースを整理しました。

新たに作成したSSD1306仕様の BMF_8_16.datのフォーマット



BMF_8_16.datのデータをR8Cマイコンのアセンブラソースにしたデータ

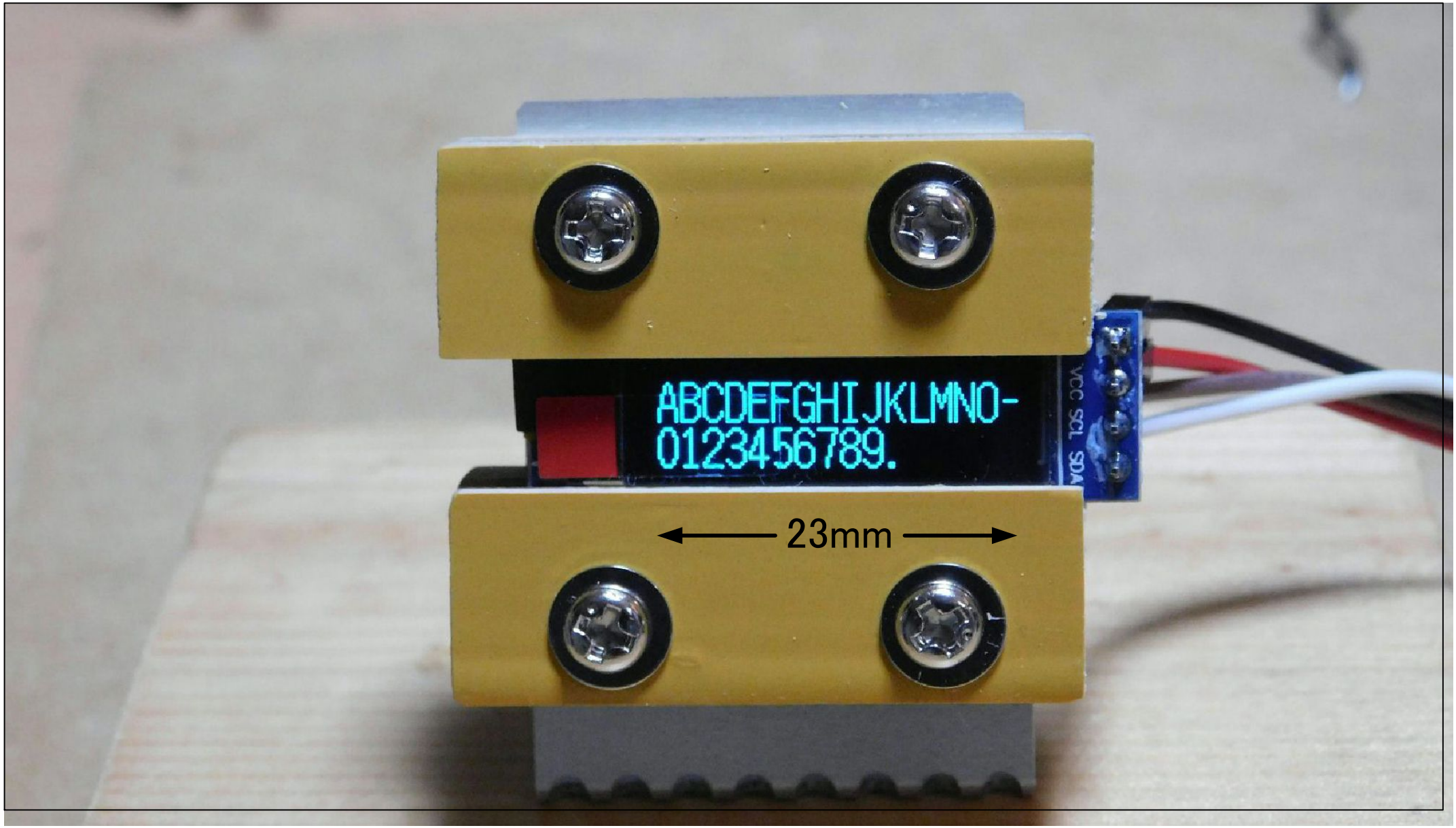
先頭4文字分だけ切り出しました。

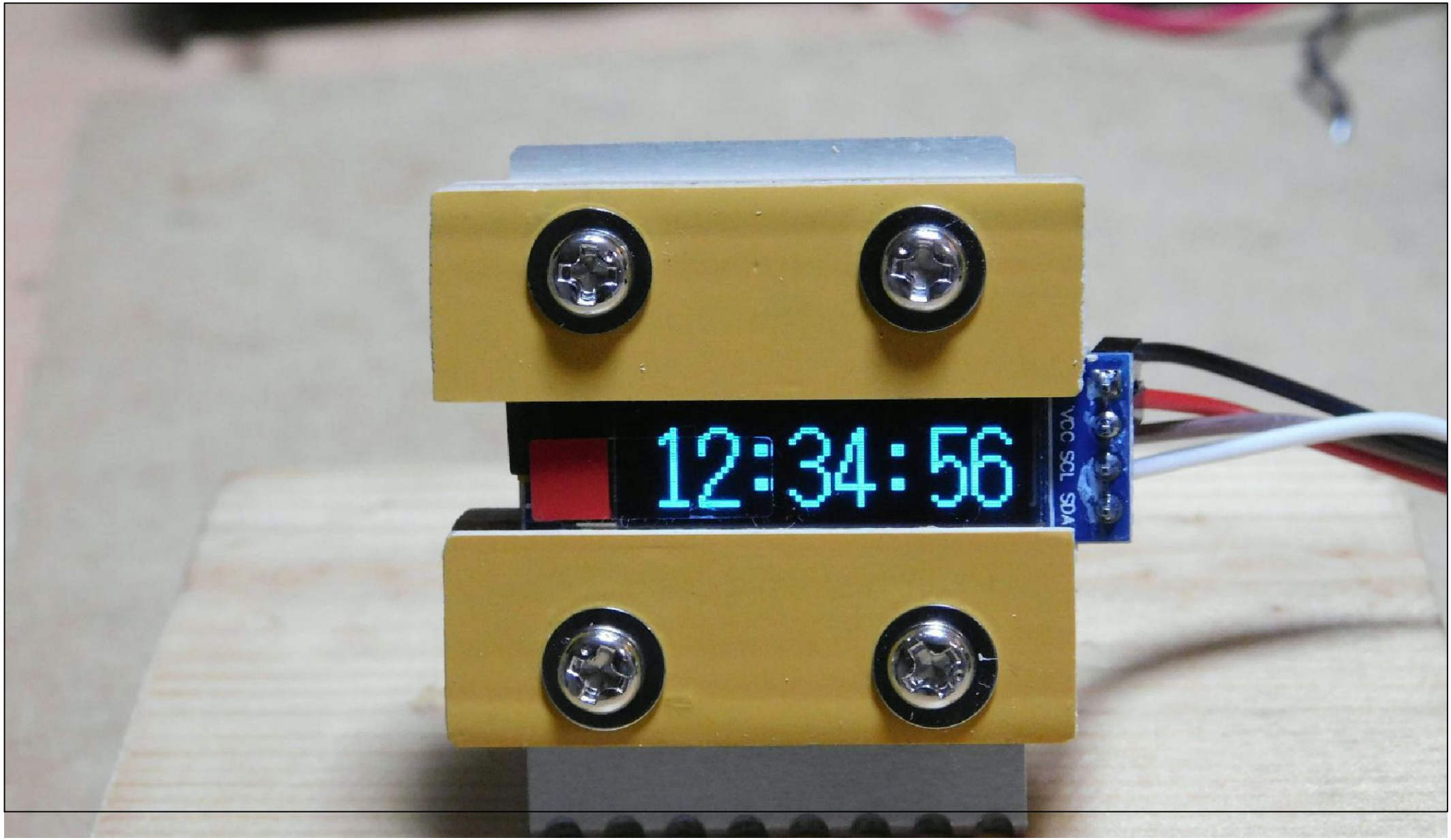
```
;    ***  BitMapFont 8x16  *** ( BMF_8x16.inc )
;    ----------------------------------------
bmf_8x16_asc:
;    Code = 20
     .byte   000h, 000h, 000h, 000h, 000h, 000h, 000h, 000h
     .byte   000h, 000h, 000h, 000h, 000h, 000h, 000h, 000h
;    Code = 21
     .byte   000h, 000h, 000h, 03Ch, 0FEh, 03Ch, 000h, 000h
     .byte   000h, 000h, 000h, 020h, 073h, 020h, 000h, 000h
;    Code = 22
     .byte   000h, 000h, 003h, 00Fh, 000h, 00Fh, 003h, 000h
     .byte   000h, 000h, 000h, 000h, 000h, 000h, 000h, 000h
;    Code = 23
     .byte   000h, 040h, 0FCh, 020h, 020h, 0FEh, 010h, 000h
     .byte   000h, 008h, 07Fh, 004h, 004h, 03Fh, 002h, 000h
```

R8Cマイコン OLED関数

dsd_OLED_sub.h の一部です。
（ dsd_OLED_sub.cも参照の事 ）

```
#define OLED_ADDR 0x3C      // IIC Address

//   ＊＊＊    関数プロトタイプ宣言    ＊＊＊
// ------------------------------------------------------------
BYTE  *get_dsd_data( WORD no );       // コマンドデータの取り出し
BYTE  *get_bmf_8x16_asc( WORD c );    // ASCIIコードのフォントデータ取得
WORD  exp2_pattern( BYTE p );         // WORD <-- BYTEパターンを２倍にする

void  dt_oled1_init( void );          // DSD TECH OLED の 初期化
void  dt_oled1_fill( BYTE ptn );      // 画面フィルインコマンド

void  put_string_8x16( int x, int y, char *txt );  // 8x16 文字列出力
void  put_char_8x16( int x, int y, char code );    // 8x16 ASCII１文字出力

void  put_string_16x32( int x, int y, char *txt ); // 16x32 文字列出力
void put_char_16x32( int x, int y, char code );    // 16x32 ASCII文字出力
```

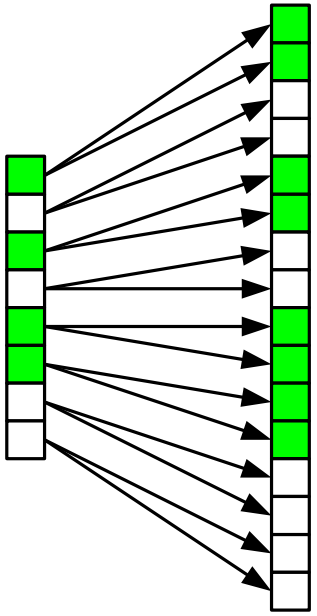

ABCDEFGHIJKLMNO-
0123456789.
VCC SCL SDA
23mm


12:34:56
VCC SCL SDA

R8Cマイコン側で行っている フォントの縦横２倍拡張処理 （ 1/2 ）

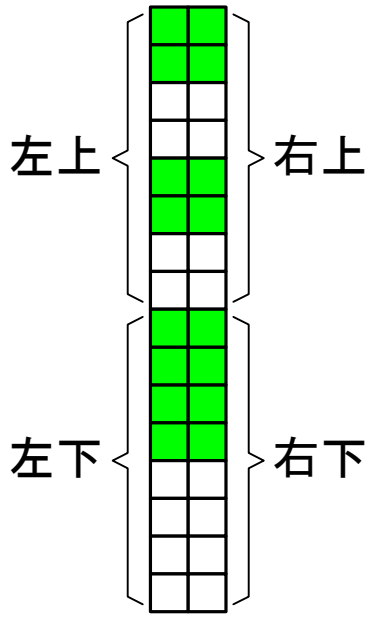
```
; ************************************************
; **  ○  8bit --> 16bit パターン２倍化処理     **
; ** ------------------------------------------ **
; **  引数：  R1L：  Byte Pattern               **
; **  関数値：  R0 ：Word Pattern               **
; ************************************************
    .glb   $exp2_pattern          ; グローバル宣言
$exp2_pattern:                    ; ラベル（ 関数エントリアドレス ）
        push.w  r2                ; R2 退避
        mov.w   #8, r2            ; R2 Loopカウンタ初期値 = 8
p003:
        shl.w   #2, r0            ; R0 <-- R0 << 2
        tst.b   #80h, r1l         ; R1L.b7 は Zero か ？
        jz      p004              ; Zero で あれば
        or.w    #3, r0            ; Zero でなければ R0 = R0 + 3
p004:
        shl.b   #1, r1l           ; R1L <-- R1L << 1
        sub.w   #1, r2            ; R2 = R2 - 1  Loopカウンタデクリメント
        jnz     p003              ; Zeroで無ければ、P003へ行く
        pop.w   r2                ; R2 復帰
        rts                       ; リターン
```

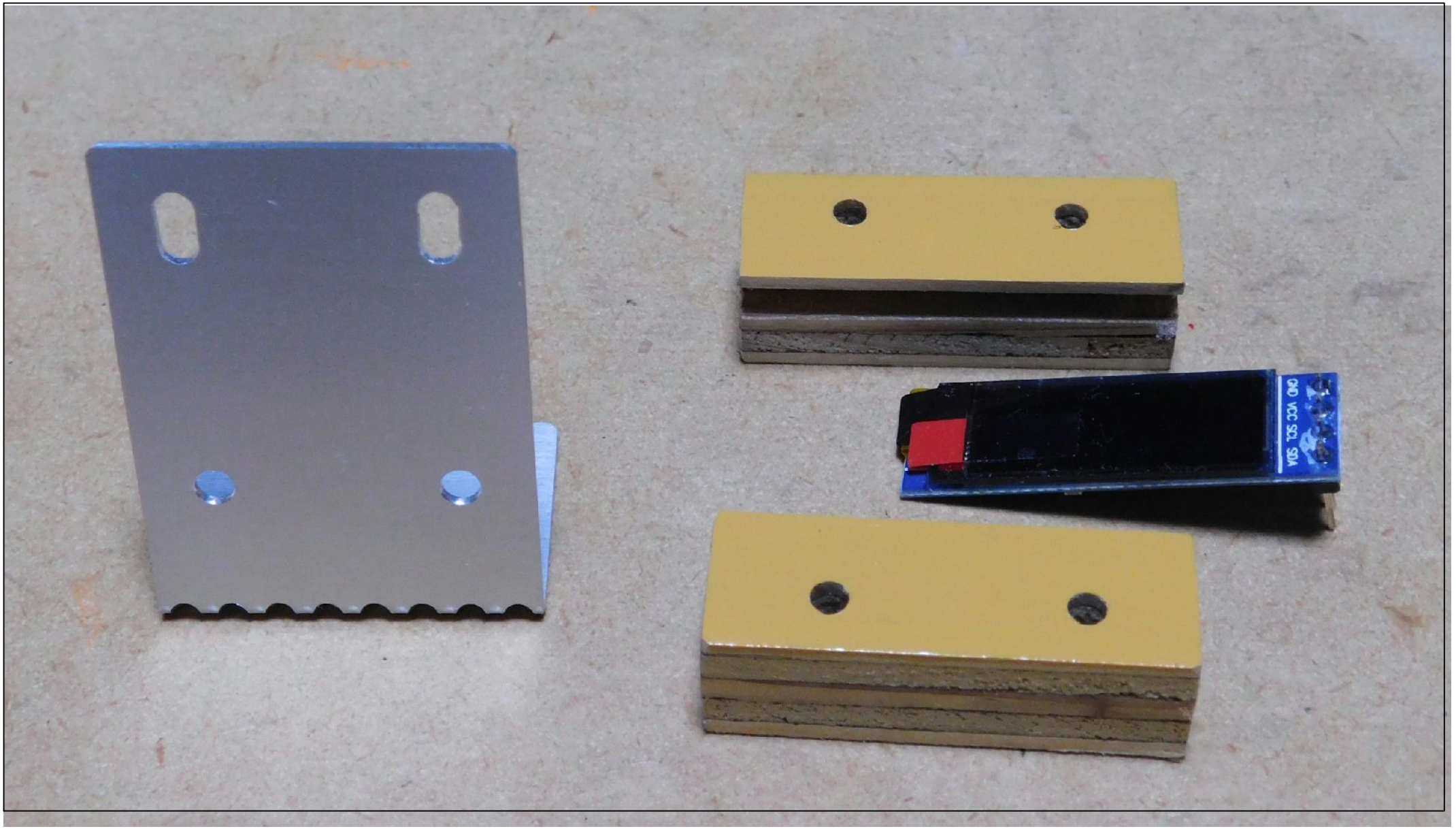
Byte --> Word 拡張のイメージ

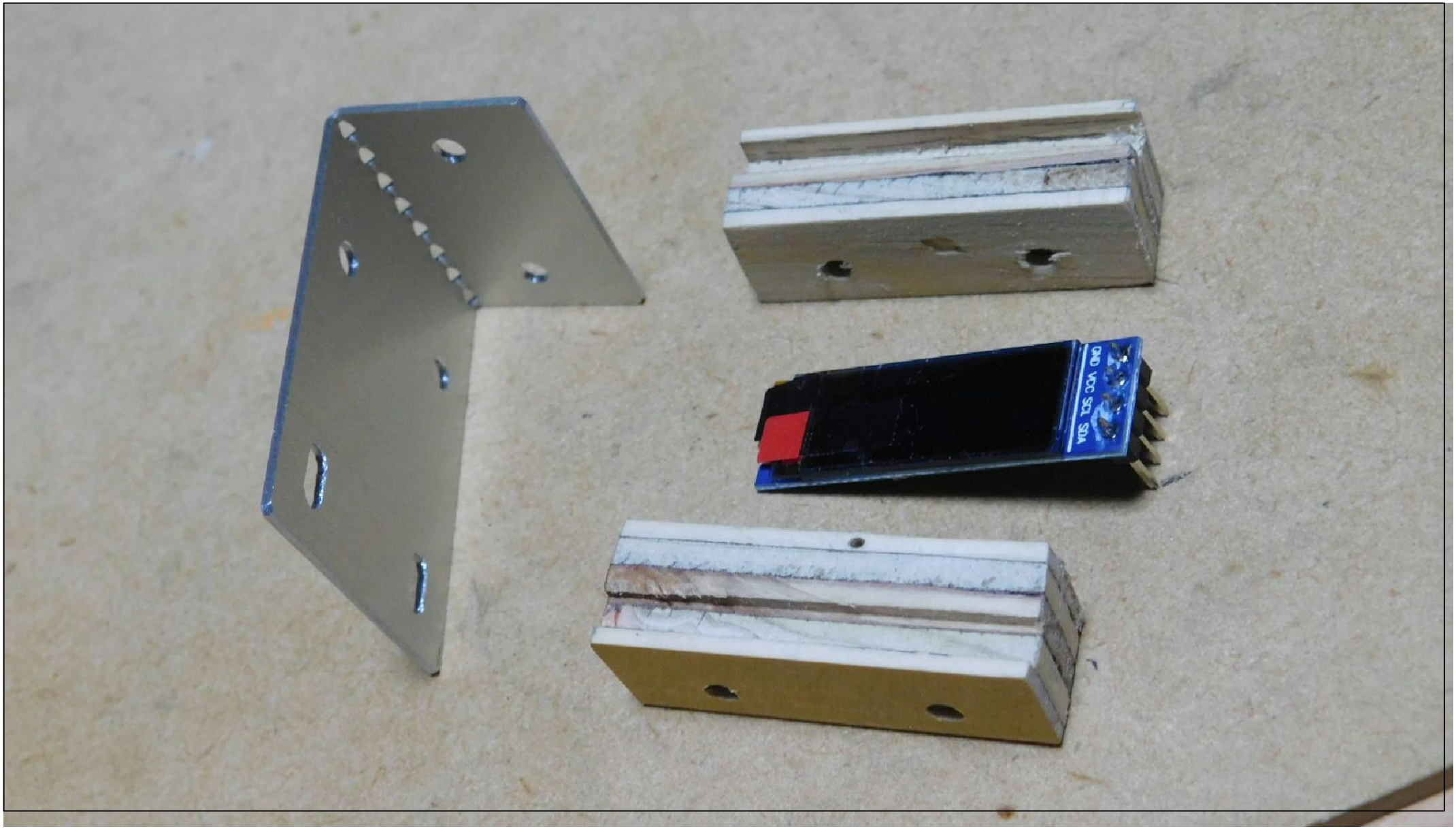


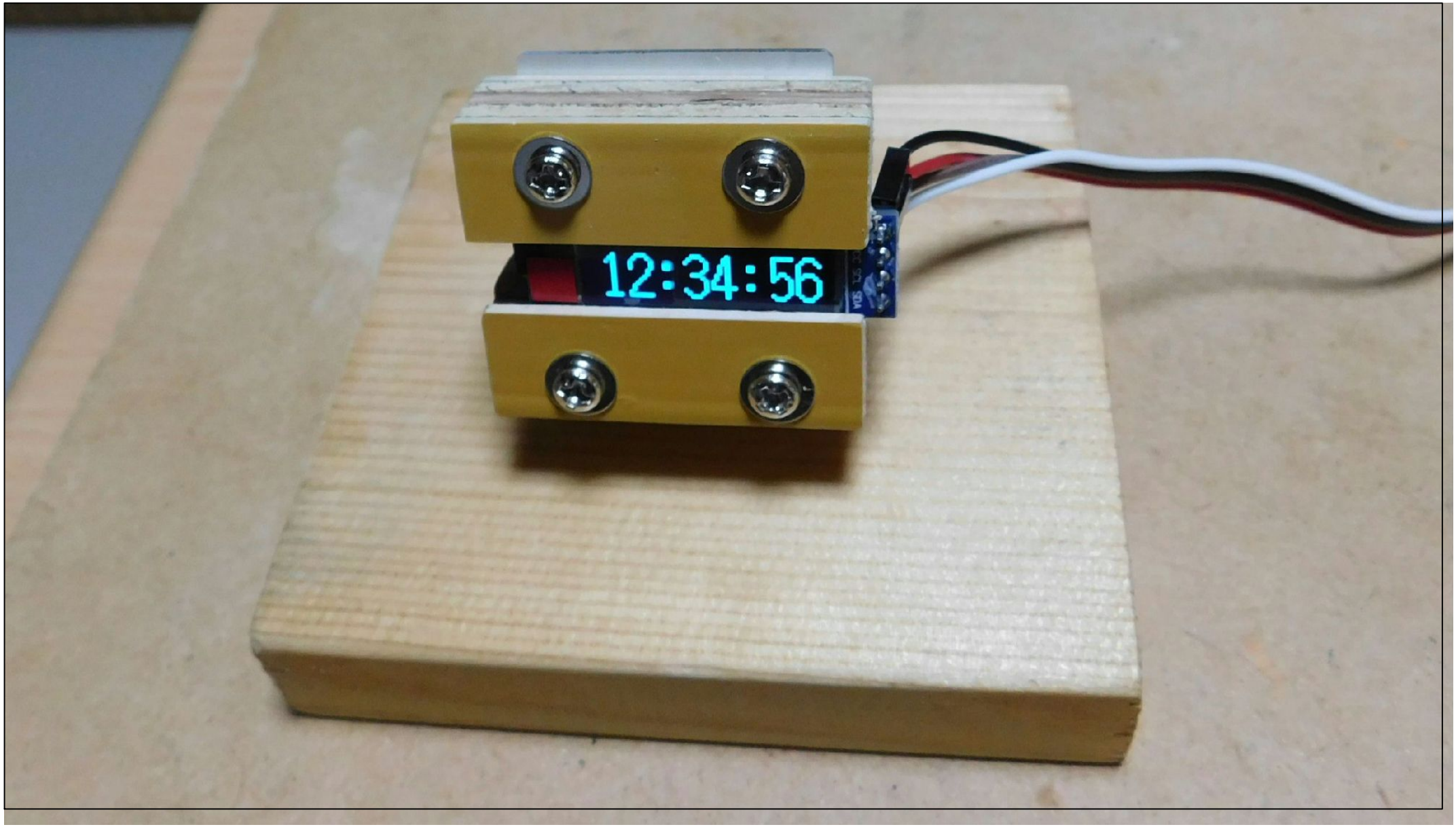
R8Cマイコン側で行っている フォントの縦横２倍拡張処理 (2/2)

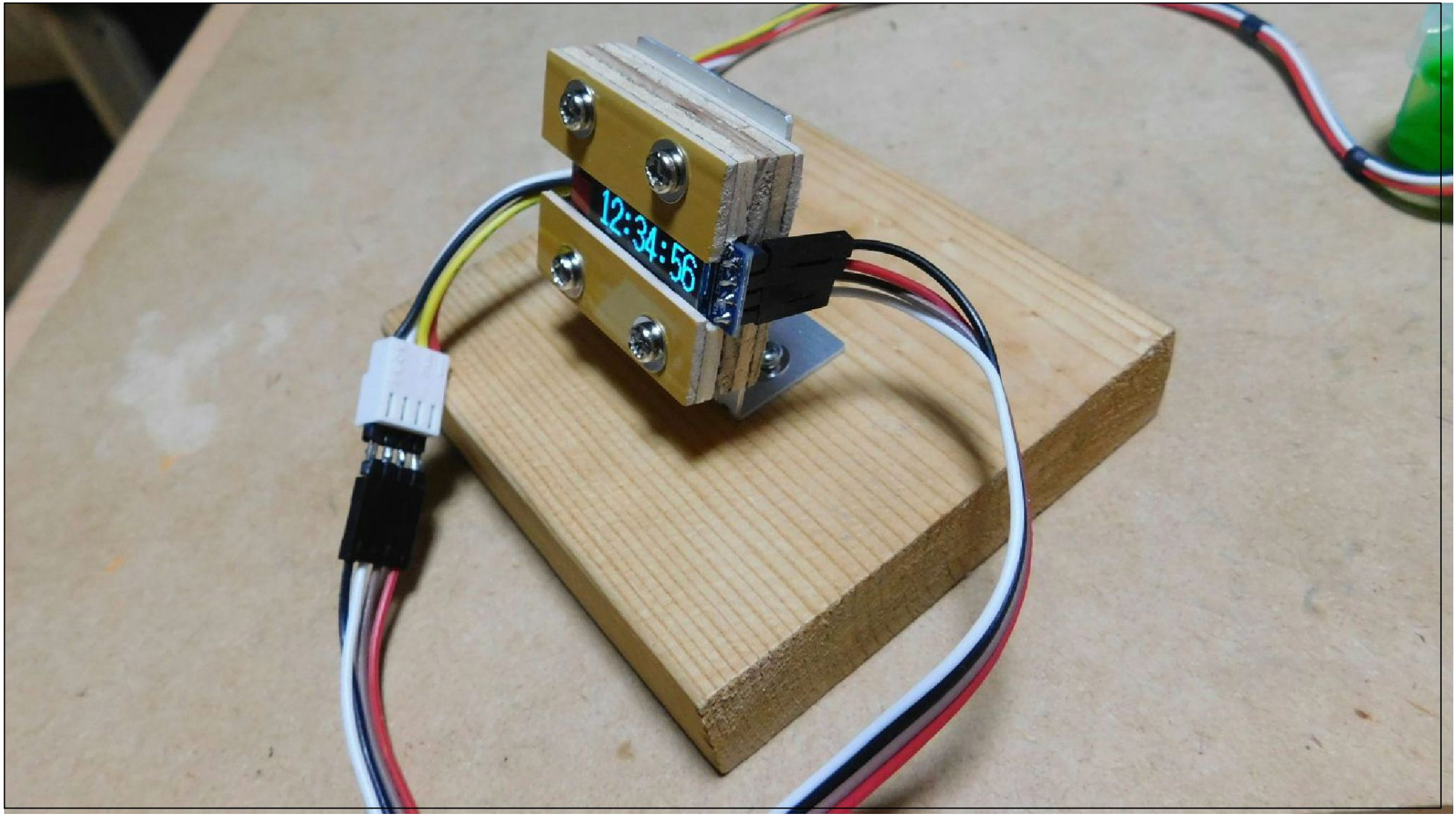
```
union  Word_Byte {
    WORD   w;    // 2byte変数１個
    BYTE   b[2];// 1byte変数２個
};
static   union Word_Byte  Wb;   // Word Byteの共用体
static   BYTE   Bufx4[64];      // 縦横２倍サイズのバッファ

//  ★★★     関数の一部分の切り出し     ★★★
    for( i=0; i<16; i++ )
    {
        Wb.w = exp2_pattern( ptr[i] );
                              // Byte pattern -> Word pattern変換
        if( i < 8 )  q1 = i * 2;     // (前半 8byte)左上Byte位置
        else         q1 = 16 + i * 2;// (後半 8byte)左上Byte位置
        q2 = q1 + 1;     // 右上Byte位置
        q3 = q1 + 16;    // 左下Byte位置
        q4 = q3 + 1;     // 右下Byte位置
        Bufx4[q1] = Wb.b[0];    // 左上 Pattern data 格納
        Bufx4[q2] = Wb.b[0];    // 右上 Pattern data 格納
        Bufx4[q3] = Wb.b[1];    // 左下 Pattern data 格納
        Bufx4[q4] = Wb.b[1];    // 右下 Pattern data 格納
    }
```




GND VCC SCL SDA


GND VCC SCL SDA


12:34:56


12:34:56