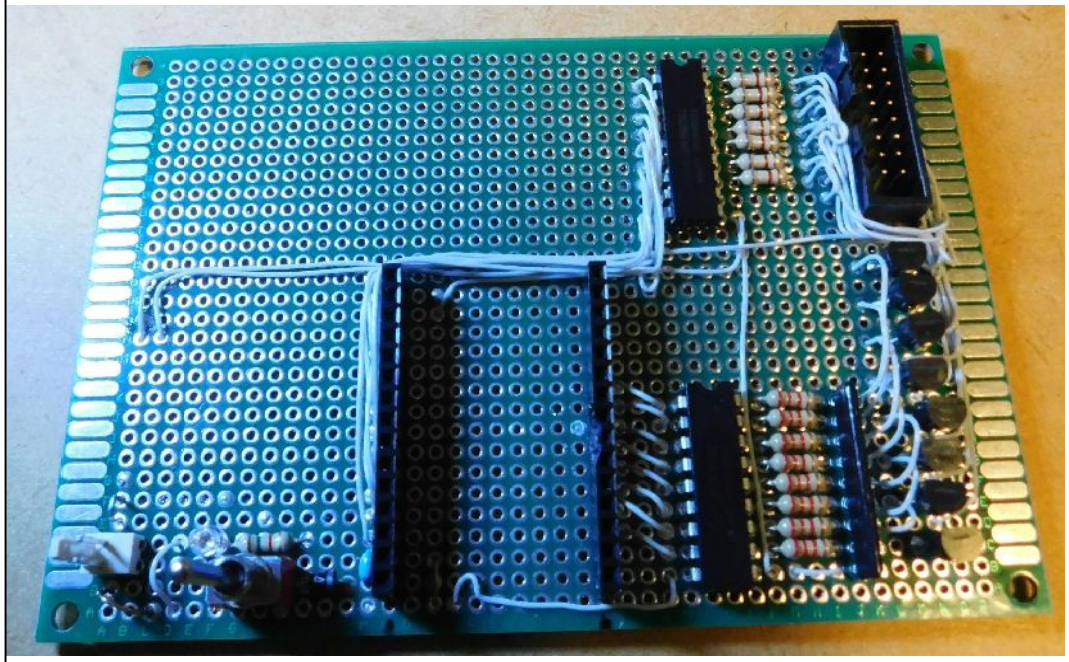
7セグメントLED ダイナミック点灯のソフト開発

ハードウェアを駆動するソフトを 作成する場合、まず対象となるハードウェアを理解する事から始めます。



上の画像は、今回作成したデジタル時計用のベース基板です。 中央下側の 縦2列に並んだピンソケットに ESP32モジュールが、挿入されます。 ESP32のピンソケットから、2つのトランジスタアレイ TD62083APに 接続されます。

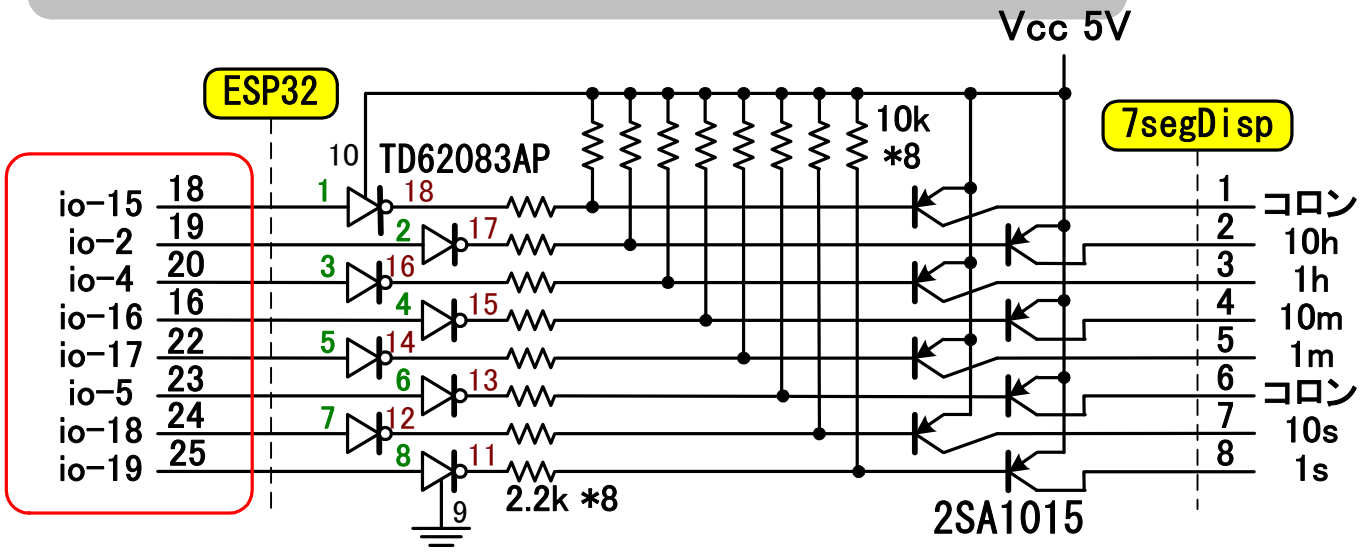
上側のトランジスタアレイが 7セグメントの カソード側 Aから Gの セグメント信号に接続されます。 このセグメント信号は 各桁の 同じセグメント名同士の信号が、結線されています。

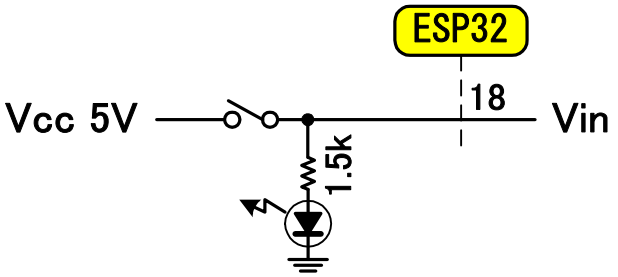
下が、7セグメントの アノード側で アノードコモンなので、各7セグメントから 論理的に1本の信号線が出てます。 表示桁数が、8桁であれば、8本の アノード側信号が 出てます。 アノード側なので、電圧を ＋側に引き上げる必要があります。 その関係で、トランジスタアレイの出力に 更に PNPトランジスタの 2SA1015を接続しています。

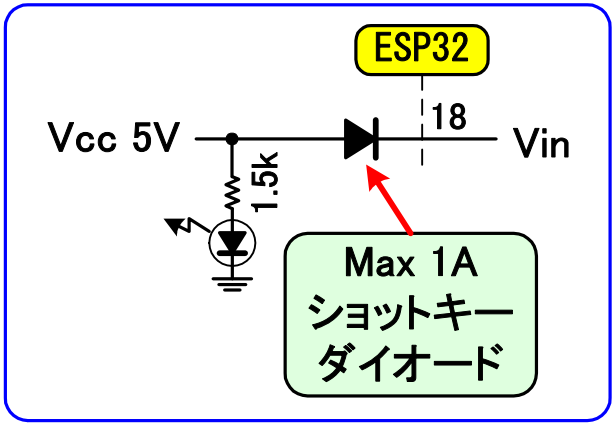
この回路構成にしたのは ESP32の 3.3V出力を5Vに 電圧変換する意味合いもあります。

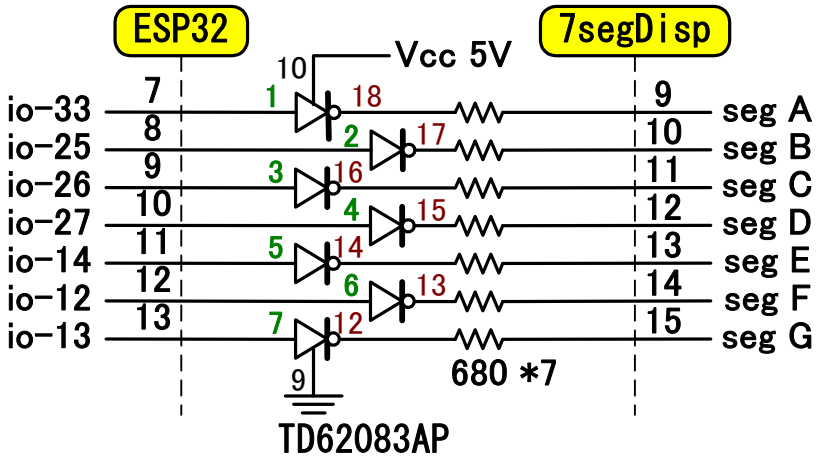
次ページに、変更した回路図を示します。

7セグメント表示基板　ドライブ回路図（改）









左側　赤枠内の ESP32側の信号線　割り付けは初期の物から 変更しています。
Vcc5Vを　ESP32の Vinに入れる回路は、上下どちらでも　かまいません。　スイッチを ON、OFFする手間が　無いのは、下の青枠内の回路です。

先ほどの　外部電源の 5Vと　USB経由の 5Vの 切り替え回路ですが、ショットキーダイオードによる切り替えの動作がよく分からない方もいると思います。

その前に　定電圧電源回路に ついて簡単に説明しておきます。 定電圧電源回路内には、基準電圧発生回路が　あります。 この基準電圧に合わせる形で　出力電圧を制御します。
典型的なアナログフィードバック回路と なります。 基準電圧と比べて出力電圧が高ければ低くなるように制御し、逆に　基準電圧と比べて出力電圧が低ければ、高くなるように制御します。

基準電圧発生回路は、各電源毎に、多少バラつきがあります。

仮に　2つの 5V電源　Aと　Bがあり、Aが、5.1V出力で、Bが　5.0V出力の場合、その2つの電源出力を直接接続すると　Aは、B出力により電圧が引き落とされる方向に引っ張られ、逆にBは、A出力により電圧が引き上げられる事になります。 が、それぞれの電源のフィードバック回路により、自分の基準電圧に合わせようと制御します。

で、お互い引っ張り合いをして、やや大きい電流が相互に流れ　すぐに壊れる事は無くても、そのまま放置しておくと、お互い通常より発熱が、大きくなり　弱い電源の方が先に壊れるでしょう。

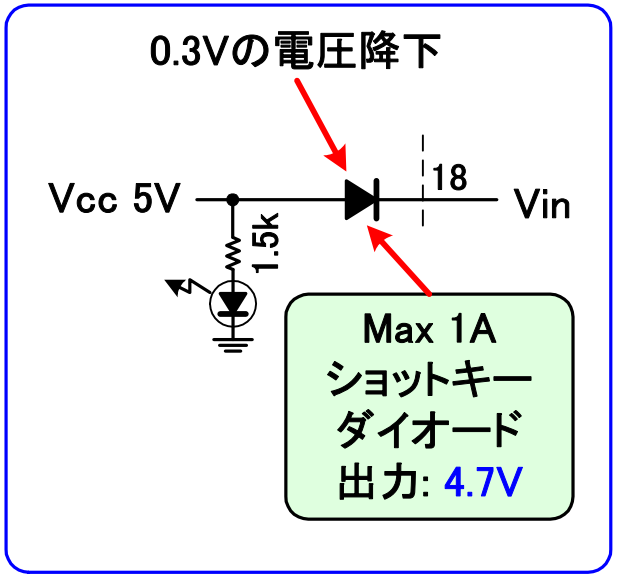
で、今回の場合　USB側は　いじる事が出来ないので、外部電源の 5Vに　小さいショットキーダイオードを入れて、VF=0.3Vぐらいで、ESP32側では、4.7Vぐらいになると思います。

USB電源が、5Vで、ショットキーダイオードのカソード出力が、4.7Vであれば、ショットキーダイオードからは、電流は流れ出ず、USBの電源が、ESP32に供給されます。

USBケーブルが引き抜かれると、ショットキーダイオードから、凡そ 4.7Vが　供給されます。

5Vよりは、やや低いですが、ESP32モジュール上の　3.3V三端子電源ICは、ロードロップアウトなので、三端子電源ICで、生じる電圧降下は 0.6Vぐらいと思われます。 よって、3.3V + 0.6V = 3.9V が、最低限の電圧と、なります。
4.7V − 3.9V = 0.8Vの電圧マージンとなります。

普通の 1A程度の整流ダイオードでも使用できますが、VF = 0.7Vなので、
0.4Vの電圧マージンと、なります。



今回の用途以外にも　ESP32を　使う用途で、3.3Vと　5Vを　混在して使う場合に、役に立てば幸いです。

ソフトの説明動画のはずが、ハードの説明に時間を　割いてしまいました。
次ページから、ソフトの方へ移行します。

ハードとソフトの接点　I/Oポートの番号

I/Oポートの番号とは、多数ある I/Oポートのうちの　1つを指定するための番号です。

今までのマイコンは　バイト単位で I/Oポートと呼んでおり、バイト単位でもアクセス出来ますが、その中のビットを　ビット単位でも　アクセス出来るようになっていました。

ESP32の場合は、I/Oポートの　バイト単位の概念は、無いようです。 もしかしたら有るのかもしれませんが、Arduino UNO等の I/Oポートアクセスの概念を 継承したのかもしれません。

今、Arduino IDEの　環境で プログラム開発をしているので、このビット単位の 番号で　アクセスする事に します。

で、今回の　7セグメントLEDの　アクセスは 7セグメント側と　一桁に 1つのアノード側というか、こちら側を　カラム側と呼ぶ事にします。

よって　7セグメント側 7本と、カラム側 8本の I/Oポートの　ビットを使用しています。

これらを　表に しておきます。

7セグメント側　7本	
信号名	I/O番号
seg_A	33
seg_B	25
seg_C	26
seg_D	27
seg_E	14
seg_F	12
seg_G	13

カラム側　8本	
信号名	I/O番号
colon_L	15
colon_R	5
time_H10	2
time_H1	4
time_M10	16
time_M1	17
time_S10	18
time_S1	19

この場合の 信号名は、
ソフトで使用する定数名と　解釈して下さい。

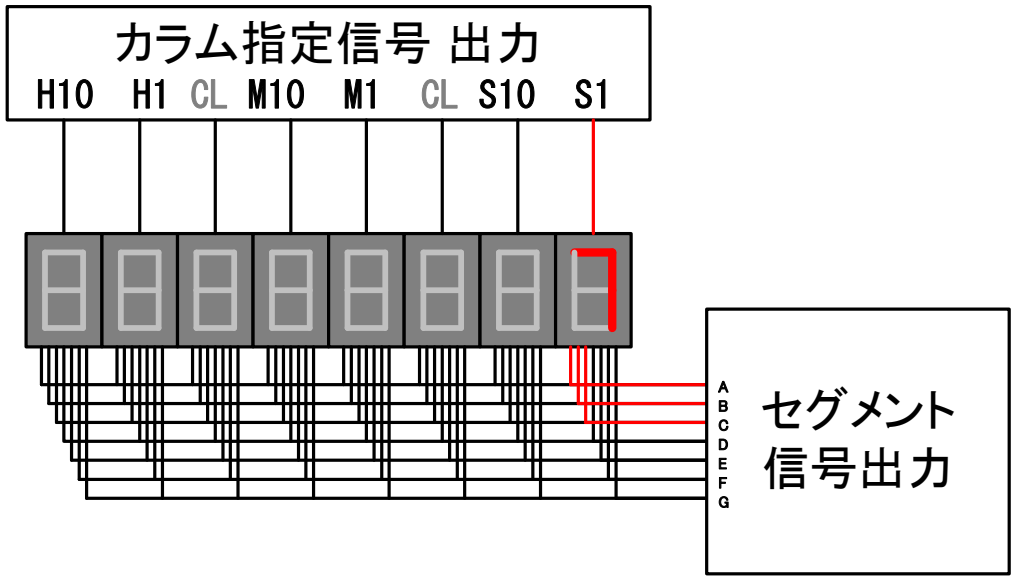
８桁の　７セグメントLEDアクセスは、
カラム側　8本と　セグメント側　7本に　明確に分けてアクセスする事になります。

カラム側、セグメント側 共に　ハイアクティブです。I/Oポートに HIGHを　設定すれば 点灯、LOWを　設定すれば、消灯を　意味します。

① カラム側の設定：
8本の 信号線を １本ずつ順番に　HIGHにします。２つ以上同時に HIGH にする事はありません。

② セグメント側の設定：
1個の ７セグメントLEDに　指定された数字を表示するための　点灯するセグメントをHIGHにします。　よって 同時に複数点灯有りです。

結線図にすると、以下のようになります。



上の図では、カラム選択信号の　S1 を 選択してます。 そして、セグメント信号出力は、 7の文字を 表示するので、ＡＢＣの　３セグメントを有効に してます。

１文字 出力するための　一連の処理としてはこうなります。

7セグメントLED ダイナミック点灯を実現する 関数の仕様

ソフト構築の やり方として、トップダウン式とボトムアップ式が ありま す。 今回は、ボトムアップ式で 行います。

まず、ハードウェアとの接点となる I/Oポートの宣言です。 #define を 用いて宣言します。

```
//   I/O Port 宣言
// ------------------------------
#define colon_L  15   // 左コロン
#define colon_R   5   // 右コロン
#define time_H10  2   // 時 2桁目
#define time_H1   4   // 時 1桁目
#define time_M10 16   // 分 2桁目
#define time_M1  17   // 分 1桁目
#define time_S10 18   // 秒 2桁目
#define time_S1  19   // 秒 1桁目
```

```
#define seg_A   33    // Segment A
#define seg_B   25    // Segment B
#define seg_C   26    // Segment C
#define seg_D   27    // Segment D
#define seg_E   14    // Segment E
#define seg_F   12    // Segment F
#define seg_G   13    // Segment G
```

いきなり 33とか 25とか ポート番号で 都度指定するより、ソース先頭で このように宣言してポートを指定する時 seg_A とか seg_B と 指定する方が、意味が分かりやすいですし、何らかの理由で ポート番号が 変っても先頭の #define の 宣言を変えるだけで 対応出来ます。

次は、カラムの宣言と セグメントの宣言をバイトデータで 指定できるようにします。

まずは、カラムの選択ですが、カラムは　同時に複数のカラムを　選択する事は無いので、バイト整数で 引数を渡し　0 ～ 7 の値で　カラムを　指定します。　今回は　col_select(char n) という関数を用意しました。

```
//  カラムの選択
void  col_select( char n )
{
      switch( n )
      {
        case  0:  digitalWrite( time_H10, HIGH );  break;
        case  1:  digitalWrite( time_H1, HIGH );   break;
        case  2:  digitalWrite( time_M10, HIGH );  break;
        case  3:  digitalWrite( time_M1, HIGH );   break;
        case  4:  digitalWrite( time_S10, HIGH );  break;
        case  5:  digitalWrite( time_S1, HIGH );   break;
        case  6:  digitalWrite( colon_L, HIGH );   break;
        case  7:  digitalWrite( colon_R, HIGH );   break;
      }
}
```

因みに　左の関数の
カラム位置の 番号は、

時の 2桁目が 0、1桁目が 1 で
分の 2桁目が 2、1桁目が 3 で
秒の 2桁目が 4、1桁目が 5 で
左から右に　時:分:秒のイメージで　番号が　付けられています。

時と　分の間のコロンが 6 で
分と　秒の間のコロンが 7です。

時計の場合、コロンは固定的に扱うので、最後の 6 と　7 に持って来ました。

次は　7セグメントの信号を出力する関数を用意します。
7セグメントの信号は、同時に　複数の信号を アクティブにするので、バイトデータの各ビットに　セグメント信号を対応させます。

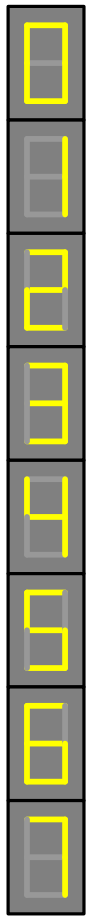
こうする事により、"0" を表示する時は、
seg7_out(0x3F); になります。

"2" を　表示する時は
seg7_out(0x5B); になります。
次ページにて 表示する数字に対応する セグメントデータの一覧を　表示します。

b7	b6	b5	b4	b3	b2	b1	b0
NC	seg_G	seg_F	seg_E	seg_D	seg_C	seg_B	seg_A

```
void  seg7_out( char d )
{
    if((d & 0x01) != 0 )
        digitalWrite( seg_A, HIGH );   // Segment_A  HIGH
    if((d & 0x02) != 0 )
        digitalWrite( seg_B, HIGH );   // Segment_B  HIGH
    if((d & 0x04) != 0 )
        digitalWrite( seg_C, HIGH );   // Segment_C  HIGH
    if((d & 0x08) != 0 )
        digitalWrite( seg_D, HIGH );   // Segment_D  HIGH
    if((d & 0x10) != 0 )
        digitalWrite( seg_E, HIGH );   // Segment_E  HIGH
    if((d & 0x20) != 0 )
        digitalWrite( seg_F, HIGH );   // Segment_F  HIGH
    if((d & 0x40) != 0 )
        digitalWrite( seg_G, HIGH );   // Segment_G  HIGH
}
```

表示数字と セグメントパターンデータの対応



G	F	E	D	C	B	A

b7	b6	b5	b4	b3	b2	b1	b0	16進数
0	0	1	1	1	1	1	1	0x3F

b7	b6	b5	b4	b3	b2	b1	b0	16進数
0	0	0	0	0	1	1	0	0x06

b7	b6	b5	b4	b3	b2	b1	b0	16進数
0	1	0	1	1	0	1	1	0x5B

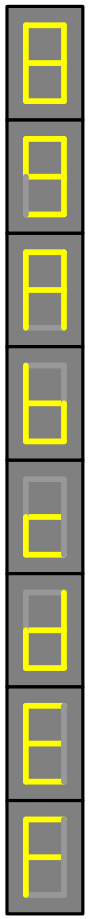
b7	b6	b5	b4	b3	b2	b1	b0	16進数
0	1	0	0	1	1	1	1	0x4F

b7	b6	b5	b4	b3	b2	b1	b0	16進数
0	1	1	0	0	1	1	0	0x66

b7	b6	b5	b4	b3	b2	b1	b0	16進数
0	1	1	0	1	1	0	1	0x6D

b7	b6	b5	b4	b3	b2	b1	b0	16進数
0	1	1	1	1	1	0	1	0x7D

b7	b6	b5	b4	b3	b2	b1	b0	16進数
0	0	0	0	0	1	1	1	0x07



G	F	E	D	C	B	A

b7	b6	b5	b4	b3	b2	b1	b0	16進数
0	1	1	1	1	1	1	1	0x7F

b7	b6	b5	b4	b3	b2	b1	b0	16進数
0	1	1	0	1	1	1	1	0x6F

b7	b6	b5	b4	b3	b2	b1	b0	16進数
0	1	1	1	0	1	1	1	0x77

b7	b6	b5	b4	b3	b2	b1	b0	16進数
0	1	1	1	1	1	0	0	0x7C

b7	b6	b5	b4	b3	b2	b1	b0	16進数
0	1	0	1	1	0	0	0	0x58

b7	b6	b5	b4	b3	b2	b1	b0	16進数
0	1	0	1	1	1	1	0	0x5E

b7	b6	b5	b4	b3	b2	b1	b0	16進数
0	1	1	1	1	0	0	1	0x79

b7	b6	b5	b4	b3	b2	b1	b0	16進数
0	1	1	1	0	0	0	1	0x71

セグメントパターンによる数字表示

バイトデータの 下位 4bit データを 数値として扱い seg7_val_out(char d) 関数に 渡します。 これにより 7セグメントのその数値に対応する 文字の形を セグメントパターンとして 表示します。

右の例の場合 10進数の "0" ～ "9" 及び、16進数表示で使う "A" ～ "F"を表示します。

表示する 桁位置は col_select(char n) 関数にて 指定します。

```
void  seg7_val_out( char d )
{
    switch( d )
    {
      case  0x00:  seg7_out( 0x3F );   break;
      case  0x01:  seg7_out( 0x06 );   break;
      case  0x02:  seg7_out( 0x5B );   break;
      case  0x03:  seg7_out( 0x4F );   break;
      case  0x04:  seg7_out( 0x66 );   break;
      case  0x05:  seg7_out( 0x6D );   break;
      case  0x06:  seg7_out( 0x7D );   break;
      case  0x07:  seg7_out( 0x07 );   break;
      case  0x08:  seg7_out( 0x7F );   break;
      case  0x09:  seg7_out( 0x6F );   break;
      case  0x0A:  seg7_out( 0x77 );   break;
      case  0x0B:  seg7_out( 0x7C );   break;
      case  0x0C:  seg7_out( 0x58 );   break;
      case  0x0D:  seg7_out( 0x5E );   break;
      case  0x0E:  seg7_out( 0x79 );   break;
      case  0x0F:  seg7_out( 0x71 );   break;
    }
}
```

7セグメントLEDによる 数値表示サブルーチンの 最後の関数で、
カラム位置指定と、数値文字の表示を一つにまとめた seg7_put 関数です。

時計の時刻表示のカウンタ変数を バイト整数を 使って行う場合、時分秒を
例えば char hh, mm, ss; で行うとします。
ss を 1秒毎にインクリメントして
ssが 60になったら ss = 0 を 行い、
mmを インクリメントします。
mmが 60になったら mm = 0 を 行い
hhを インクリメントします。
hhが 24になったら hh = 0 に します。
(24時間制の場合) で、この時刻カウンタの歩進動作を行った後に 時刻を 7セグメントLEDに 表示します。

```
//**************************************
//**  ７セグLED １文字  出力メイン  **
//** ---------------------------- **
//**  c :  カラム位置                **
//**  d :  表示する数値              **
//**************************************
void  seg7_put( char c, char d )
{
    col_select( c );       // カラム指定
    seg7_val_out( d );     // 数値文字の表示
}
```

時刻カウンタから、7セグメントLEDに表示する値に変更するには、時分秒の 各要素を 10 で割り、割った答えが 2桁目になり、余りが 1桁目になります。

今回は、ダイナミック点灯の 7セグメントLEDの１文字表示を メインに説明したので、時計の歩進と連携した表示処理は、また 改めて行います。