C:\Users\jdn\Desktop\GithubJDN\Oled\OledMap_ino.asm

Program "OledMap.h" Assembly of the 7 Oled libs for 1k bitmap. 17061

MIT License

```
//==============
```

Routine "OledGenc.h"

```
#include <avr/pgmspace.h>
 const uint8_t didel[] PROGMEM = {0x82,0xf2,0xfe,0xfe,0x8e,0x82,0xc2,0x3c,0x00
                                  0xc0,0xf8,0xfd,0x3d,0x0d,0x00, \
                                  0x82,0xf2,0xfe,0xfe,0x8e,0x82,0xc2,0x3c,0x00, \
                                  0xc0,0xf8,0xfe,0xfe,0x96,0x92,0x82,0x02,0x00, \
                                  0xc0,0xf8,0xfe,0xfe,0x86,0x80,0x80,0x80,0x80 };
const uint8_t taNum[] PROGMEM = {
  0x00,0x00,0x00,0x00, 0x00,0xc0,0xde,0x00, // space !
  0x03,0x00,0x03,0x00, 0x74,0x2f,0x74,0x2f, // " #
  0x2c,0x49,0xff,0x39, 0x43,0x33,0x6c,0x63, // $ %
  0x6e,0x59,0x19,0x26, 0x00,0x05,0x03,0x00, // & '
  0x1c,0x22,0x41,0x00, 0x00,0x41,0x22,0x1c, // ( )
  0x2a,0x1c,0x1c,0x2a, 0x08,0x3e,0x08,0x08, // * +
  0x00,0xa0,0x60,0x00, 0x08,0x08,0x08,0x08, // , -
  0x00,0xc0,0xc0,0x00, 0x03,0x0c,0x30,0xc0, // . /
  0x3e,0x41,0x41,0x3e, 0x04,0x02,0x7f,0x00, // 0 1
  0x62,0x51,0x49,0x46, 0x22,0x41,0x49,0x36, // 2 3
  0x1c,0x12,0x79,0x10, 0x2f,0x49,0x49,0x31, // 4 5
  0x36,0x49,0x49,0x31, 0x41,0x31,0x0d,0x03, // 6 7
  0x36,0x49,0x49,0x36, 0x46,0x49,0x49,0x3e, // 8 9
  0x00,0x36,0x36,0x00, 0x00,0xd6,0x76,0x00, // : ;
  0x08,0x14,0x22,0x41, 0x24,0x24,0x24,0x24, // < =
  0x41,0x22,0x14,0x08, 0x02,0xc1,0xd9,0x06}; // > ?
const uint8_t taMaj[] PROGMEM = {
  0x39,0x48,0x71,0x41,0x3e, 0x7e,0x11,0x11,0x11,0x7e, // @ A
  0x7f,0x49,0x49,0x49,0x36, 0x3e,0x41,0x41,0x41,0x22, // B C
  0x7f,0x41,0x41,0x41,0x3e, 0x7f,0x49,0x49,0x41,0x41, // D,E
  0x7f,0x09,0x09,0x01,0x01, // F
  0x3e,0x41,0x49,0x49,0x3a, 0x7f,0x08,0x08,0x08,0x7f, // G,H
  0x00,0x00,0x7f,0x00,0x00, 0x21,0x41,0x41,0x41,0x3f, // I,J
  0x7f,0x08,0x14,0x22,0x41, 0x7f,0x40,0x40,0x40,0x40, // K,L
  0x7f,0x02,0x04,0x02,0x7f, 0x7f,0x04,0x08,0x10,0x7f, // M,N
  0x3e,0x41,0x41,0x41,0x3e, 0x7f,0x11,0x11,0x11,0x0e, // O,P
  0x3e,0x41,0x49,0x51,0x7e, 0x7f,0x09,0x11,0x29,0x46, // Q,R
  0x46,0x49,0x49,0x49,0x32, 0x01,0x01,0x7f,0x01,0x01, // S,T
  0x3f,0x40,0x40,0x40,0x3f, 0x07,0x18,0x60,0x18,0x07, // U,V
  0x1f,0x60,0x1e,0x60,0x1f, 0x63,0x14,0x08,0x14,0x63, // W,X
  0x07,0x04,0x78,0x04,0x07, 0x61,0x51,0x49,0x45,0x43, // Y,Z
  0x00,0x7f,0x41,0x41,0x00, 0x03,0x06,0x1c,0x30,0x60, // [ backsl
  0x00,0x41,0x41,0x7f,0x00, 0x04,0x02,0x01,0x02,0x04, // ] ^
  0x80,0x80,0x80,0x80,0x80}; // _

const uint8_t taMin [] PROGMEM = {
  0x00,0x03,0x06,0x00, 0x20,0x54,0x54,0x78, // ` a 0x38,0x44,0x44,0x7C,
  0x7f,0x44,0x44,0x38, 0x38,0x44,0x44,0x28, // b c
  0x38,0x44,0x44,0x7f, 0x38,0x54,0x54,0x58, // d e
  0x00,0xfe,0x09,0x08, 0x9c,0xa2,0xa2,0x7c, // f g
  0x7f,0x04,0x04,0x78, 0x00,0x7d,0x00,0x00, // h i
  0x40,0x80,0x80,0x7d, 0x7e,0x10,0x28,0x44, // j k
  0x00,0x3f,0x40,0x00, 0x7c,0x08,0x08,0x7c, // l m
  0x7c,0x08,0x10,0x7c, 0x38,0x44,0x44,0x38, // n o
  0xfc,0x24,0x24,0x18, 0x38,0x44,0x44,0xf8, // p q
  0x7c,0x04,0x04,0x08, 0x48,0x54,0x54,0x24, // r s
  0x04,0x04,0x7c,0x04, 0x3c,0x40,0x40,0x3c, // t u
  0x1c,0x60,0x60,0x1c, 0x5c,0x20,0x20,0x5c, // v w
  0x64,0x18,0x18,0x64, 0x04,0x08,0x78,0x04, // x y
  0x64,0x54,0x4c,0x44, 0x08,0x08,0x36,0x41, // z {
  0x00,0x00,0xff,0x00, 0x41,0x36,0x08,0x08, // | }
  0x10,0x20,0x10,0x20, 0xff,0xff,0xff,0xff}; // ~ del
const uint8_t smile[] PROGMEM = {0x3c,0x42,0x95,0xa5,0xa1,0xa1, 0xa5,0x95,0x42
const uint8_t sad  [] PROGMEM = {0x3c,0x42,0xc5,0xa5,0xa1,0xa1, 0xa5,0xc5,0x42
const uint8_t didel[] PROGMEM = {0x82,0xf2,0xfe,0xfe,0x8e,0x82,0xc2,0x3c,0x00,
                                 0xc0,0xf8,0xfd,0x3d,0x0d,0x00, \
                                 0x82,0xf2,0xfe,0xfe,0x8e,0x82,0xc2,0x3c,0x00, \
                                 0xc0,0xf8,0xfe,0xfe,0x96,0x92,0x82,0x02,0x00, \
                                 0xc0,0xf8,0xfe,0xfe,0x86,0x80,0x80,0x80,0x80 };

//=============================================================
```

Routine "OledControlMap.h" 170429

```
// Initialisation and transfer
void  Cmd (byte cc) {
  Start(); Write (Adr); Write(0);Write (cc); Stop();
}
void  Cmd2 (byte aa,byte bb) {
  Start(); Write (Adr); Write(0);Write (aa); Write (bb);Stop();
}
void  Clear(); void Show();
byte taInitOled[]={ 0xae, 0xd5, 0x80, 0xa8, 63, \
        0xD3, 0x0, 0x40, 0x8d, 0x14, 0x20, 0x00, 0xa1, \
        0xc8, 0xDA, 0x12, 0x81, 0xcf, 0xd9, 0xf1, 0xdb, \
        0x40, 0xa4, 0xa6, 0x2e, 0xaf  };

uint16_t ptMap;
uint8_t taMap[1024];
void  Clear() {  //nn<128x8=1024
    for (int i=0; i<1024; i++) {taMap[i]=0;}
}
void  ClearRight() {  //nn<128x8=1024
    for (int i=0; i<1024; i++) {
      if (i&0x40) {taMap[i]=0;} // 64..127
    }
}
void  Fill(byte li,byte co,byte ll,byte dd) {
    ptMap = 128*li+co;
    for (int i=0; i<ll; i++) {taMap[ptMap+i]=0;}
}

void  Show () {  // include Initransfert
  Start(); Write (Adr); Write(0);
  Write (0x21); Write (0); Write (127);
  Write (0x22); Write (0); Write (7);
  Stop();
  Start(); Write (Adr); Write (0x40);
  for (int i=0; i<1024; i++)
   { Write (taMap[i]);}
  Stop();
  Cmd(0xC8);  // patch pour bb!
}

// set pointer
byte saveLi,saveCol;
void  SetLine (byte cc) {
  saveLi=cc;
  ptMap = 128*saveLi+saveCol;
}
void  SetCol (byte cc) {
  saveCol=cc;
  ptMap = 128*saveLi+saveCol;
}
void  LiCol (byte li,byte cc) {
  saveLi=li;  saveCol=cc;
  ptMap = (128*saveLi)+saveCol;
}
void  GetLiCol () {
  saveLi=ptMap/128;  saveCol=ptMap%128;
}

#define  Sprite(tt) \
  for (byte i=0; i<sizeof tt; i++) \
   { taMap[ptMap++] = pgm_read_byte(&tt[i]);}

// For the character generators
#define Copy(tt) \
  for (byte i=0; i<sizeof tt/4; i++) { \
   for (byte j=0; j<4; j++) {taMap[ptMap++] = (pgm_read_byte(&tt[4*i+j]));} \
   taMap[i++]=0;
// Maj en 5 de large
#define CopyMaj(tt) \
  for (byte i=0; i<sizeof tt/5; i++) { \
   for (byte j=0; j<5; j++) {taMap[ptMap++] = (pgm_read_byte(&tt[5*i+j]));} \
   taMap[i++]=0;

void  SetupOledMap() {
  for (byte i=0; i<sizeof (taInitOled); i++) {
    Start(); Write (Adr); Write(0);
    Write (taInitOled[i]); Stop();
  }
  Clear(); //Sprite (smile);
  Show();
}

//=============================================================
```

Routine "OledCarMap.h" Copy car and text in buffer

OledI2C.bmp

```
void Error();

void DoubleH () {Cmd (0xda);Cmd (0x02);}

void Car(char cc) {
  cc&=0x7F;
  switch (cc/32) {
    case 0:
       if(cc=13) {SetLine(saveLi+1);SetCol(0);} // saut de ligne
       break;
    case 1:  // codes 32-
      for (byte i=0; i<4; i++)
        { taMap[ptMap++] = pgm_read_byte(&taNum[((cc-32)*4)+i]);}
      taMap[ptMap++] = 0; taMap[ptMap++] = 0;
          break;
    case 2:  // codes 64-
      for (byte i=0; i<5; i++)
        { taMap[ptMap++] = (pgm_read_byte(&taMaj[((cc-64)*5)+i]));}
      taMap[ptMap++] = 0;
          break;
    case 3:  //codes 96-
      for (byte i=0; i<4; i++)
      { taMap[ptMap++] = (pgm_read_byte(&taMin[((cc-96)*4)+i]));}
      taMap[ptMap++] = 0;
          break;
  }
}

#define Text(tt) \
  for (byte i=0; i<sizeof tt; i++) \
  { Car (tt[i]);}

 void Error() {
  LiCol(0,100); Car('e');Car('r');Car('r');Car('o');Car('r');
}
//=============================================================
```

Routine OledNum.h display non significative zeros

```
void Bin8 (byte bb) {
  for (byte i=0;i<8;i++) {
     if (bb&0x80) Car('1');
     else Car('0');
     bb <<= 1;
  }
  Car(' ');
}
char ConvNibble (byte nn) { // converti 4 bit hexa en Ascii
  byte cc;
  if (nn<10) {cc = nn + '0';}
  else {cc = nn-10 + 'A';}
  return cc;
}
void Hex8 (byte hh) {
  byte cc;
  cc = ConvNibble (hh >> 4) ; // ne modifie pas hh
  Car(cc);
  cc = ConvNibble (hh & 0x0F) ;
  Car(cc);
  Car(' '); // space
}
void Hex16 (uint16_t hh) {
  byte cc;
  cc = ConvNibble (hh >> 12) ; Car(cc);
  cc = ConvNibble ((hh >> 8)&0x0F) ; Car(cc);
  cc = ConvNibble ((hh >> 4)&0x0F) ; Car(cc);
```

```
  cc = ConvNibble (hh & 0x0F) ;  Car(cc);
  Car(' ');
}
void Hex12 (uint16_t hh) {
  byte cc=0;
  cc = ConvNibble ((hh >> 8)&0x0F) ; Car(cc);
  cc = ConvNibble ((hh >> 4)&0x0F) ; Car(cc);
  cc = ConvNibble (hh & 0x0F) ;  Car(cc);
  Car(' ');
}
uint16_t BinDec8 (uint8_t bb) {
  uint16_t dd=0;
   for (byte i=0; i<8; i++) {
    if ((dd & 0x0F)>0x04) {dd += 0x03;}
    if ((dd & 0xF0)>0x40) {dd += 0x30;}
    dd=dd<<1;
    if ((bb & 0x80)) {dd += 1;} //inject bit
    bb=bb<<1; // prepare next bit
  }
  return dd;
}
void Dec8 (byte hh) {
  Hex12(BinDec8(hh));
}
uint16_t BinDec9999 (uint16_t bb) { //0x270F max
  uint16_t dd=0;
 for (byte i=0; i<16; i++) {
    if ((dd & 0x000F)>0x0004) {dd += 0x0003;}
    if ((dd & 0x00F0)>0x0040) {dd += 0x0030;}
    if ((dd & 0x0F00)>0x0400) {dd += 0x0300;}
    if ((dd & 0xF000)>0x4000) {dd += 0x3000;}
      dd=dd<<1;
      if ((bb & 0x8000)) {dd += 1; } //inject bit
      bb<<=1; // prepare next bit
  }
  return dd;
}
void Dec9999 (uint16_t hh) {  // limitï¿½ ï¿½ 0x2703
  if (hh>9999) { Car('?'); Car('?'); Car('?'); Car('?'); }
  else Hex16(BinDec9999(hh));
}
//============================================================
```

Routine "OledGraMap.h" Dots and simple lines in buffer 170426

```
void Dot(byte xx,byte yy) {  // yy 0-64 --> 0-7
  byte coly, nob;
  coly = yy/8; nob = yy%8;
  saveLi = yy/8; saveCol = xx+1;
  ptMap = (128*saveLi)+saveCol;
//  GetLiCol();
  taMap[(coly*128)+xx] |= (1<<nob);
}
void NoDot(byte xx,byte yy) {  // yy 0-64 --> 0-7
  byte coly, nob;
  coly = yy/8; nob = yy%8;
  saveLi = yy/8; saveCol = xx+1;
  ptMap = (128*saveLi)+saveCol;
//  GetLiCol();
  taMap[(coly*128)+xx] &= ~(1<<nob);
}
void DDot(byte xx,byte yy) {  // yy0-64 --> 0-7
  byte coly, nob; // on ajoute un point en dessous
  coly = yy/8; nob = yy%8;
  taMap[(coly*128)+xx] |= (1<<nob);
  if ((nob==7)&&(coly<7)) { coly++; nob=0; }
  else { nob++; }
  taMap[(coly*128)+xx] |= (1<<nob);
}
void Hline (byte yy) {
  for (byte i=0; i<128; i++) {
//    taMap[(yy*128)+i] |= (1<<(yy%8));
     Dot(i,yy);
  }
}
void Hseg (byte xx,byte yy,byte ll) {
  for (byte i=0; i<ll; i++) {
  if((xx+i)>127) break;
     Dot(xx+i,yy);
  }
}
void Vline (byte xx) {
  for (byte i=0; i<8; i++) {
    taMap[(i*128)+xx] = 0xff;
  }
}
void Vseg (byte xx,byte yy,byte hh) {
  for (byte i=0; i<hh; i++) {
    if((yy+i)>63) break;
    NoDot (xx,yy+i);
  }
    for (byte i=0; i<hh; i++) {
    if((yy+i)>63) break;
    Dot (xx,yy+i);
  }
}

//===========
```

Routine "OledBig.h" Double size single Ascii character and numbers

```
// Table to double nibble to byte
byte nToB[]={0,3,0xc,0xf,0x30,0x33,0x3c,0x3f,0xc0,0xc3,0xcc,0xcf,0xf0,0xf3,0xf
void Big(byte cc) {
  //utilise saveLi -/>  saveCol -->+8 ou+10 selon cc&(1<<5)
  cc&=0x7F; byte tmp,i,k;
//if (cc&(1<<5)) k=4; else k=5; // taille
  k=5;
    // on s'occupe des nible sup
  SetLine (--saveLi); // saveCol dÃ©fini au premier car
  for (byte i=0; i<k; i++) {
   switch (cc/32) {
    case 0:
      Error();
      break;
    case 1:  // codes 32-
      tmp = pgm_read_byte(&taNum[((cc-32)*4)+i]);
      tmp &= 0x0f; // low byte
      tmp = nToB[tmp];
      if (i>3) tmp=0;
      // on Ã©crit ce byte sur 2 colonnes
      taMap[ptMap++]=(tmp); taMap[ptMap++]=(tmp);
      break;
    case 2:  // codes 64-
      tmp = pgm_read_byte(&taMaj[((cc-64)*5)+i]);
      tmp &= 0x0f; // low byte
      tmp = nToB[tmp];
      taMap[ptMap++] = (tmp); taMap[ptMap++] = (tmp);
      break;
    case 3:  // codes 96-
     tmp = pgm_read_byte(&taMin[((cc-96)*4)+i]);
      tmp &= 0x0f; // low byte
      tmp = nToB[tmp];
      // on Ã©crit ce byte sur 2 colonnes
      taMap[ptMap++] = (tmp); taMap[ptMap++] = (tmp);
      break;
   }  // end switch
  } // end for
  // On aloute 2 espaces
  taMap[ptMap++] = (0);taMap[ptMap++] = (0);
     // on s'occupe des nible inf
  SetLine (++saveLi); SetCol(saveCol);
  for (byte i=0; i<k; i++) {
   switch (cc/32) {
    case 0:
      break;
  case 1:  // codes 32-
      tmp = pgm_read_byte(&taNum[((cc-32)*4)+i]);
      tmp = (tmp&0xf0)>>4 ; // high byte
      tmp = nToB[tmp];
      if (i>3) tmp=0;
      // on Ã©crit ce byte sur 2 colonnes
      taMap[ptMap++] = (tmp); taMap[ptMap++] = (tmp);
      break;
    case 2:  // codes 64-
      tmp = pgm_read_byte(&taMaj[((cc-64)*5)+i]);
      tmp = (tmp&0xf0)>>4 ; // high byte
      tmp = nToB[tmp];
      // on Ã©crit ce byte sur 2 colonnes
      taMap[ptMap++]=(tmp); taMap[ptMap++]=(tmp);
      break;
    case 3:  // codes 96-
      tmp = pgm_read_byte(&taMin[((cc-96)*4)+i]);
      tmp = (tmp&0xf0)>>4 ; // high byte
      tmp = nToB[tmp];
      // on Ã©crit ce byte sur 2 colonnes
      taMap[ptMap++]=(tmp); taMap[ptMap++]=(tmp);
      break;
   }  // end switch
  } // end for
  // On ajoute 2 espaces
  taMap[ptMap++] = (0);taMap[ptMap++] = (0);
  //if (cc&(1<<5)) saveCol+=10; else saveCol+=12; // prep car suivant
  saveCol+=12;
  SetCol(saveCol);
}
void Space (byte nn){ saveCol+=nn; SetCol(saveCol); }

void BigBin8 (byte bb) {
  for (byte i=0;i<8;i++){
     if (bb&0x80) Big('1');
     else Big('0');
     bb <<= 1;
  }
//  Big(' ');
}
void BigHex8 (byte hh) {
  byte cc;
  cc = ConvNibble (hh >> 4) ; // ne modifie pas hh
  Big(cc);
  cc = ConvNibble (hh & 0x0F) ;
  Big(cc);
//  Big(' '); // space
}
void BigHex16 (uint16_t hh) {
  byte cc;
  cc = ConvNibble (hh >> 12) ;  Big(cc);
  cc = ConvNibble ((hh >> 8)&0x0F) ; Big(cc);
  cc = ConvNibble ((hh >> 4)&0x0F) ; Big(cc);
  cc = ConvNibble (hh & 0x0F) ; Big(cc);
//  Big(' ');
}
void BigHex12 (uint16_t hh) {
  byte cc=0;
  cc = ConvNibble ((hh >> 8)&0x0F) ; Big(cc);
  cc = ConvNibble ((hh >> 4)&0x0F) ; Big(cc);
  cc = ConvNibble (hh & 0x0F) ;  Big(cc);
//  Big(' ');
}

void BigDec8 (byte hh) { BigHex12(BinDec8(hh)); }

void BigDec9999 (uint16_t hh) {  // limitÃ© Ã  0x270F
  if (hh>9999) { Big('?'); Big('?'); Big('?'); Big('?'); }
  else BigHex16(BinDec9999(hh));
}

//=========================
```

Routine "OledLineCircle.h" 170603

```
int sx,sy;
void Line(int8_t x0,int8_t y0,int8_t x1,int8_t y1){
  int8_t dx =  abs(x1-x0), sx = x0<x1 ? 1 : -1;
  int8_t dy = -abs(y1-y0), sy = y0<y1 ? 1 : -1;
  int8_t err = dx+dy, e2; /* error value e_xy */
  while(1){
    Dot(x0,y0);
    if (x0==x1 && y0==y1) break;
```

```
    e2 = 2*err;
    if (e2 > dy) { err += dy; x0 += sx; } /* e_xy+e_x > 0 */
    if (e2 < dx) { err += dx; y0 += sy; } /* e_xy+e_y < 0 */
  }
}  // End Line

void Circle(uint8_t x0,uint8_t y0,uint8_t radius) {
    int8_t f = 1 - radius;
    int8_t ddF_x = 0;
    int8_t ddF_y = -2 * radius;
    uint8_t x = 0;
    uint8_t y = radius;
    Dot(x0, y0 + radius);
    Dot(x0, y0 - radius);
    Dot(x0 + radius, y0);
    Dot(x0 - radius, y0);
    while(x < y) {
      if(f >= 0) {
        y--; ddF_y += 2; f += ddF_y;
      }
      x++; ddF_x += 2; f += ddF_x + 1;
      Dot(x0+x,y0+y); Dot(x0-x,y0+y);
      Dot(x0+x,y0-y); Dot(x0-x,y0-y);
      Dot(x0+y,y0+x); Dot(x0-y,y0+x);
      Dot(x0+y,y0-x); Dot(x0-y,y0-x);
    }
  }  // end Circle
```