OledI2C and OledI2Cbb

OledI2C and OledI2Cbb are stripped down libraries required by the OledMap and OledPix graphic libraries for the Oled SSD1306. This Oled needs only simple write transfers that take less than 150 bytes of code. Using the 2k Wire Arduino library is of course possible.

OledI2C.h uses the TWI functionality of the AVR328. Similar libraries for other processors are welcomed. We wrote in CALM assembler the 16F882 SSP functions that need 42 lines, that is only 42 14-bit program words.

OledI2Cbb.h defines several macros for controlling the I2C lines. A short delay allows to adjust the timings. I2C timing on the SSD1306 is not critical; 100 kHz clock cycle is easy to reach with a 4MHz processor, as we tested using a Microchip PIC 10F202.

In both cases, the primitives to be defined are

SetupI2C(); Initialise the hardware, control registers, clock rates, pins, power pin if used
Start(); Macro or function, include ready test
Stop(); Macro or function, include ready test
Write(dd); Send 8 bits and clock the acknowledge bit.

That's all. The acknowledge is ignored, handling errors add a layer of complexity to say through a different channel what you see immediately!
Frequently Start() and Write(AdrOled) are merged in a function and 7-bit addresses are used. If you look at SSD1306 functions, it would not add clarity. Oled control uses rather long sequences of Write, first Write is for the address.

Source code can be found on https://github.com/nicoud/Oled and on the many program examples of the documentation in French listed in www.didel.com/Oled.html
Of course, the danger, and big advantage, of include files is they are easy to be modified and taylored to the application. Our documentation developped over several months, there may have been minor name changes, ask if you are confused.

Other use of OledI2C

Adding ReadI2C(); function allows to handle any I2C circuit. The Start() is written in such a way it can be used as a Restart. ReadI2C is however not as easy, frequently the peripheral needs a negative acknowledge for the last byte and this must be planned.

Source code OledI2C (AVR328)

```
// "OledI2C.h"  Minimal lib for SSD1306
#include <avr/io.h>
#define Adr 0x78  //(0x3C*2)
void SetupOledI2C() {
  TWSR = 1;    //0 for 400kHz 1 for 160k
  TWBR = 0x0C; // bitrate
  TWCR = (1<<TWEN); // other bits = 0
}

void  Start () {
  TWCR = (1<<TWINT)|(1<<TWSTA)|(1<<TWEN);
  while (!(TWCR & (1<<TWINT))) {}
}
void  Stop () {
  TWCR = (1<<TWINT)|(1<<TWSTO)|(1<<TWEN);
  for (volatile int i=0; i<20; i++) ;
  }
void  Write (byte ab) {  // addr8 ou data8
  TWDR = ab;
  TWCR = (1<<TWINT)|(1<<TWEN);
  while (!(TWCR & (1<<TWINT))) {}
;  status= TWSR & 0xF8;  // acknowledge bit
}
```

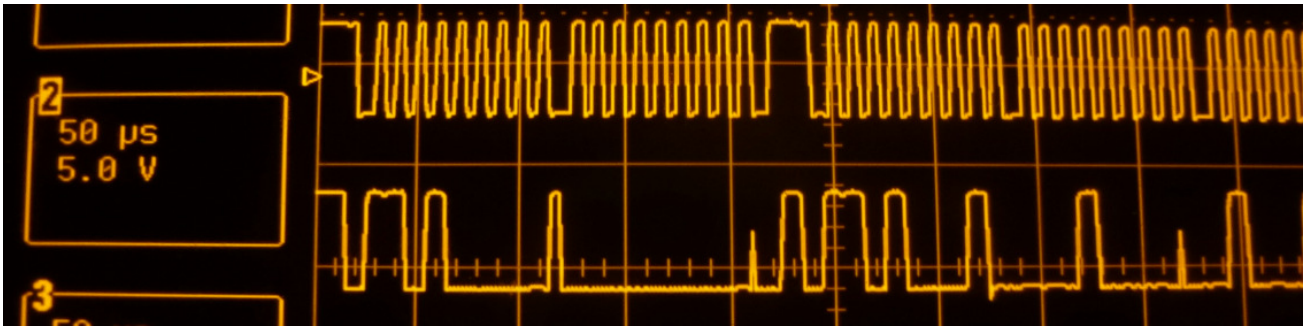
Source code OledI2Cbb (any microcontroller, SCK SDA same port)

```
// OledI2Cbb.h
#define Adr 0x78  //(0x3C*2)
void DelI2C() {
  for (volatile byte j=0; j<10; j++)  // 12 12us period
     {  asm("nop"); }
}
 #define Ddr  DDRD
 #define Port PORTD
 #define Pin  PIND
 #define bGnd 7
 #define bVcc 6
 #define bCk 5
 #define bDa 4

 #define Ck1  bitClear (Ddr,bCk)
 #define Ck0  bitSet (Ddr,bCk)
 #define Da1  bitClear (Ddr,bDa)
 #define Da0  bitSet (Ddr,bDa)
 void SetupI2Cbb() {
  Ddr  |=  (1<<bGnd|1<<bVcc|1<<bCk|1<<bDa) ;
  Port |=  (1<<bVcc) ;
  Port &= ~(1<<bGnd|1<<bCk|1<<bDa) ;
 }

// ---- Macros Start() et Stop()
#define Start() Da0; DelI2C(); Ck0; DelI2C()
#define Stop() Da0; DelI2C(); Ck1; DelI2C(); Da1

// ---- Fonction  Write (addr ou data); // ecrit 8 bits
void Write (byte aa) {
  for (int i=0; i<8; i++) {
    if (aa&0x80) Da1;
    else Da0;  // out addr
    DelI2C();  Ck1;  DelI2C();
    aa <<= 1;
    Ck0;  DelI2C();
  }
  Da1;  DelI2C(); // input Ack here if required
  Ck1;  DelI2C();
  Ck0;  DelI2C();
  DelI2C();
}
```



jdn 170611