

The Robot Can Talk

But, it chooses to type instead...

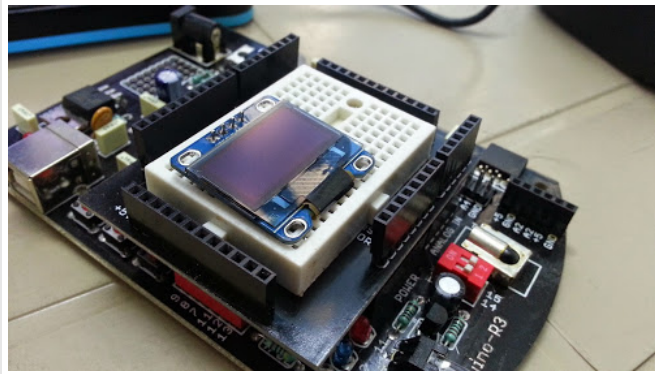
Home	Tower of Fun	Arduino	
----------------------	------------------------------	-------------------------	--

March 4, 2015

Interfacing the Arduino with an SSD1306 driven OLED Display - part 1

Interfacing the Arduino with an SSD1306 driven OLED Display - part 1

Who doesn't want a spiffy display for their Arduino!? An tiny OLED screen will let you do just that. I decided to get myself a display module for a intriguing project that I have in mind - emulating the CHIP-8 system on an Arduino Uno (atmega328). I decided to start by building the display interface.



Browsing around the web, I figured that this - [Adafruit Monochrome 0.96" 128x64 OLED graphic display](#) - fits the bill just right, aesthetically and technologically. The CHIP-8 has a resolution of 64x32, while it's successor, the SCHIP-8 (Super Chip-8) has a resolution of 128x64.

Interfacing the Arduino with an SSD1306 driven OLED Display - part 1

- [Purchasing the part](#)
- [First Impressions](#)
- [Shortcut](#)
- [Parsing the dreadful datasheet](#)
 - [I2C Communication](#)
 - [GDDRAM - Graphic Display Data RAM](#)
 - [Minimal Command Reference](#)
- [Arduino Sketch](#)
 - [setup](#)




You looking for something?

Topics

- [Tech](#) (25)
- [Life](#) (20)
- [OfficeMan](#) (9)
- [Arduino](#) (5)
- [TowerOfFun](#) (5)
- [Food](#) (3)
- [Fitness](#) (1)


Books!

Sonal's bookshelf: read







The Truth
by Terry Pratchett

★★★★★

page 1 of 4 


goodreads®

Subscribe To

-  [Posts](#) 
-  [Comments](#) 

Google+ Badge

Sonal Pinto

 [Follow](#)

About Me



loop



Sonal Pinto

 Follow

557

Just a regular Joe, who sometimes plays with electronics.

[View my complete profile](#)

Purchasing the part

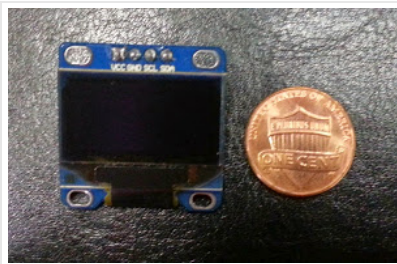
As usual, I couldn't find the original product among my trusted distributors in India (sigh...), at least without shoveling out tons of money for import costs - which defeats the point of an Arduino project. But! I did find a neat alternative at simplelabs.com (same place where I got my [Induino](#), a perfect low cost and feature rich Arduino Uno Rev3 Clone).

simplelabs 0.96" Blue 12C OLED Display Module

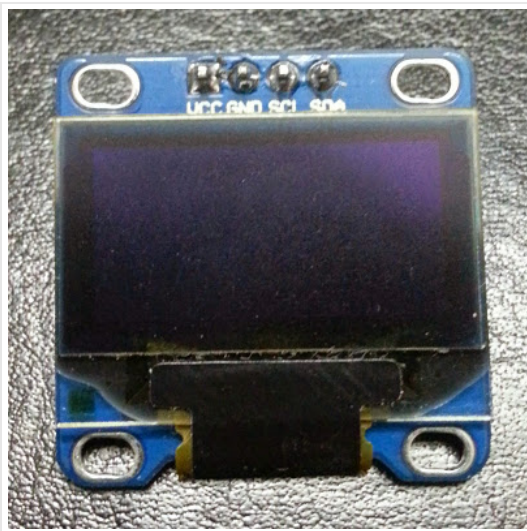
✂ *Bonus part* - simplelabs Arduino Prototyping Shield - not really required, but kinda helps keep things neat.

First Impressions

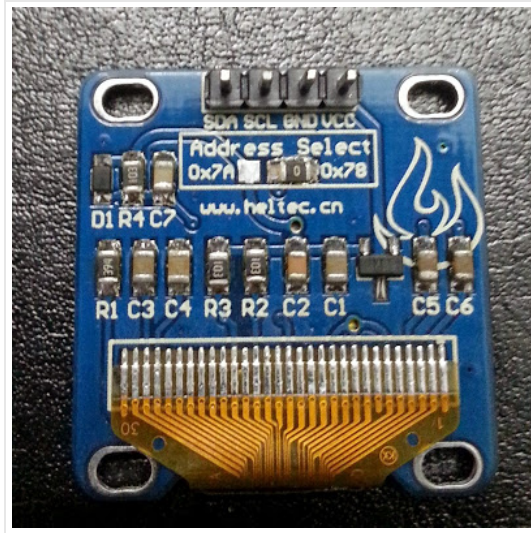
OLED vs penny



OLED front



OLED Back



Looking at the backside of the PCB it seems that this part is the same as this - [heltec 0.96 inch blue OLED](#). Another glaring fact is that this heltec OLED, though sporting the same controller, the `SSD1306`, only allows you to only interface with it via the I2C (TWI - Two Wire Interface). That would be the SDA and SCL - which would take only 2 pins from the Arduino (A4 and A5).

Secondly, there's no pin header for the Reset pin. The original Adafruit part has 8 pins with support for both 3-wire and 4-wire SPI along with I2C - and - a reset pin. Oh well... that's that.

Shortcut

The heltec OLED module (henceforth, simply called, the OLED) will work right off the bat with these libraries - `Adafruit_SSD1306` and `Adafruit-GFX`. Install them both (ie: unzip them and place them in your user libraries folder) and run the example sketch - `ssd1306_128x64_i2c.ino`. You'll need to modify this line (line#61):

```
display.begin(SSD1306_SWITCHCAPVCC, 0x3D);
```

to

```
display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
```

The I2C slave address is 0x3C, as the OLED address select jumpers have been soldered to 0x78 (I'll explain this part later).

The library is neat. But, I am masochistic when it comes to microcontrollers and I want to have optimal control and fully understand what I am doing. Plus, this project (the CHIP-8) will require a tight leash on execution speed and code length. Hence, I had no choice but to take the hard road...

Parsing the dreadful datasheet

鮟 *Datasheet - SSD1306 OLED Controller*

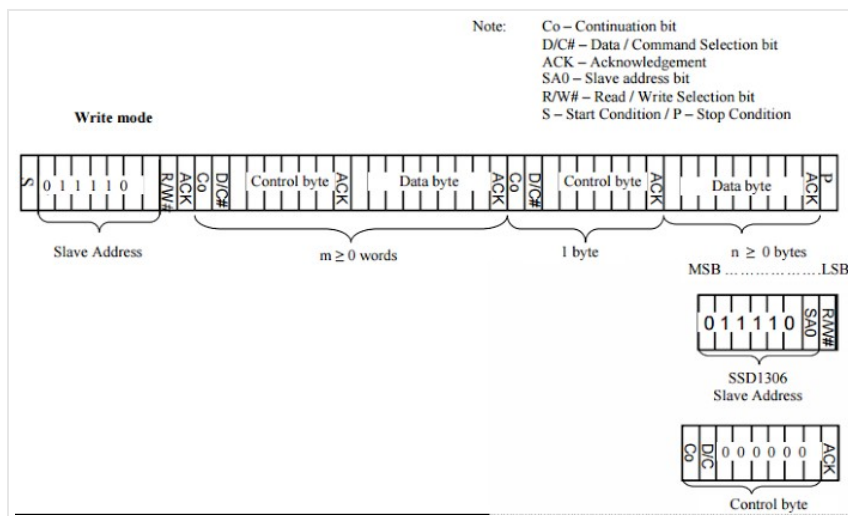
The datasheet says the the RES should be a 3us low pulse (pg.49). I wonder if there's an auto-reset circuit - for this case, RESET low (IC is reset with a low pulse) - it should be a capacitor to GND and a pullup resistor to VDD. It's probably there (looking at the components) - but, I can't really tell - as the heltec website doesn't provide any info.

Okay... okay... let's take it from the top.

I2C Communication

The SSD1306 is a 128x64 Dot Matrix Driver/Controller manufactured by Solomon Systech (based in HK). The sheet says that the VDD for the IC is supposed to be 3.3V max - but the OLED module will work with a 5V as well because there seems to be a levelshifter (3-pin LDO with 662K inscribed) on board. Since, the OLED only has an I2C interface, I shall focus on those specs, which begin on pg.19.

I2C Bus Data Format



Write Mode - is the only mode I care about. So, the Arduino is going to be the I2C master and the OLED, the I2C slave. As per the protocol, after the start condition, the Slave Address (SLA) needs to be sent.

SSD1306 Slave Address (SLA) is 0x3C

The back of the PCB shows that the 0x78 jumper has been soldered. The sheet says that the slave address is a 7-bit code that can be either 0x3C (011-1100) or 0x3D (011-11001), based on the SA0 bit (LSB of the address). The SA0 bit can be controlled by the D/C# pin of the SSD1306 (not to be confused with the D/C# bit of the control byte in the above image!). I guess, heltec must have soldered the pin to GND. Moreover, since, the OLED will always be interfaced in WRITE mode, the I2C First Byte will be the 7-bit SLA and the WRITE mode bit (0) - which becomes the byte, 0x78. Ha!

After sending the SLA+Mode byte, in order to do anything with the OLED, a Control Byte needs to be sent. The Control Byte determines whether the upcoming bytes would be treated as Data (which is written directly to the GDDRAM) or as Commands (to the internal MCU). Also controls the length of the upcoming bytes - single or stream (multiple bytes to be received by the SSD1306 until I2C Stop condition).

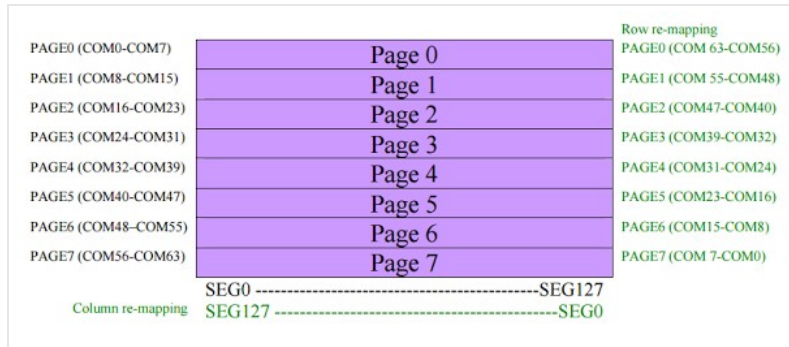
Control Byte

Deciphering this took some trial and error and real sloooooow reading of section 8.1.5.2 (Writing mode for I2C) point 5. The Control Byte has these

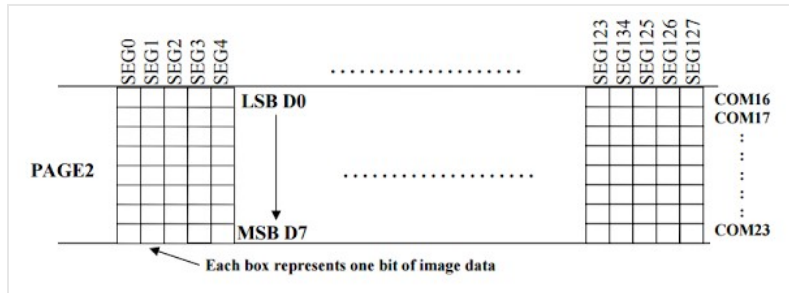
C_0 : bit 8 : Continuation Bit
 * **1** = no-continuation (only one byte to follow)
 * **0** = the controller should expect a stream of bytes.
 $D/C\#$: bit 7 : Data/Command Select bit
 * **1** = the next byte or byte stream will be Data.
 * **0** = a Command byte or byte stream will be coming up next.
 Bits 6-0 will be all zeros.
Usage:
 0×80 : Single Command byte
 0×00 : Command Stream
 $0 \times C0$: Single Data byte
 0×40 : Data Stream

GDDRAM - Graphic Display Data RAM

OLED has an 128x64 SRAM driven display with 64 rows divided as 8 Pages and 128 Columns.

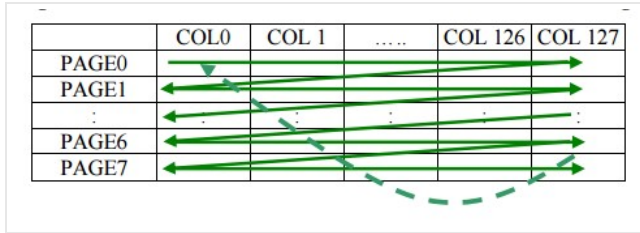


Each **PAGE** will have 8 rows (COM pins) and 128 byte-sized column fractions called **SEGMENT**. The whole mapping is fully customizable, ie: You can rearrange the COM/COL assignment to have the xy(0,0) at any corner. The GDDRAM is divided into Pages to allow easy writing of character sized sprites. That means you can also use this OLED as a 8-line character display by default.



But, I wanna use this as a generic buffered display. Hence, among the three memory addressing modes (pg.34-35), Horizontal seemed to be perfect for me. Starting from (PAGE0,COL0), the address line will auto-shift to the next segment after each data byte write, and will shift to the next page when all 128 segments of a page are written onto. Kinda exactly like the KS108 based graphic LCD, without the need to select the segment driver.

Horizontal Addressing Mode



Minimal Command Reference

I parsed section 9 (pg28-32 and pg62) of the sheet to collect these commands that would be most pertinent to interface with the OLED as a generic buffered display.

```
// SLA (0x3C) + WRITE_MODE (0x00) = 0x78 (0b01111000)
#define OLED_I2C_ADDRESS 0x3C

// Control byte
#define OLED_CONTROL_BYTE_CMD_SINGLE 0x80
#define OLED_CONTROL_BYTE_CMD_STREAM 0x00
#define OLED_CONTROL_BYTE_DATA_STREAM 0x40

// Fundamental commands (pg.28)
#define OLED_CMD_SET_CONTRAST 0x81 // follow with 0x7F
#define OLED_CMD_DISPLAY_RAM 0xA4
#define OLED_CMD_DISPLAY_ALLON 0xA5
#define OLED_CMD_DISPLAY_NORMAL 0xA6
#define OLED_CMD_DISPLAY_INVERTED 0xA7
#define OLED_CMD_DISPLAY_OFF 0xAE
#define OLED_CMD_DISPLAY_ON 0xAF

// Addressing Command Table (pg.30)
#define OLED_CMD_SET_MEMORY_ADDR_MODE 0x20 // follow with 0x00 = HORZ mode = Be
have like a KS108 graphic LCD
#define OLED_CMD_SET_COLUMN_RANGE 0x21 // can be used only in HORZ/VERT mod
e - follow with 0x00 and 0x7F = COL127
#define OLED_CMD_SET_PAGE_RANGE 0x22 // can be used only in HORZ/VERT mod
e - follow with 0x00 and 0x07 = PAGE7

// Hardware Config (pg.31)
#define OLED_CMD_SET_DISPLAY_START_LINE 0x40
#define OLED_CMD_SET_SEGMENT_REMAP 0xA1
#define OLED_CMD_SET_MUX_RATIO 0xA8 // follow with 0x3F = 64 MUX
#define OLED_CMD_SET_COM_SCAN_MODE 0xC8
#define OLED_CMD_SET_DISPLAY_OFFSET 0xD3 // follow with 0x00
#define OLED_CMD_SET_COM_PIN_MAP 0xDA // follow with 0x12
#define OLED_CMD_NOP 0xE3 // NOP

// Timing and Driving Scheme (pg.32)
#define OLED_CMD_SET_DISPLAY_CLK_DIV 0xD5 // follow with 0x80
#define OLED_CMD_SET_PRECHARGE 0xD9 // follow with 0xF1
#define OLED_CMD_SET_VCOMH_DESELCT 0xDB // follow with 0x30

// Charge Pump (pg.62)
#define OLED_CMD_SET_CHARGE_PUMP 0x8D // follow with 0x14
```

Arduino Sketch

源 [SOURCE: oled_test](#)

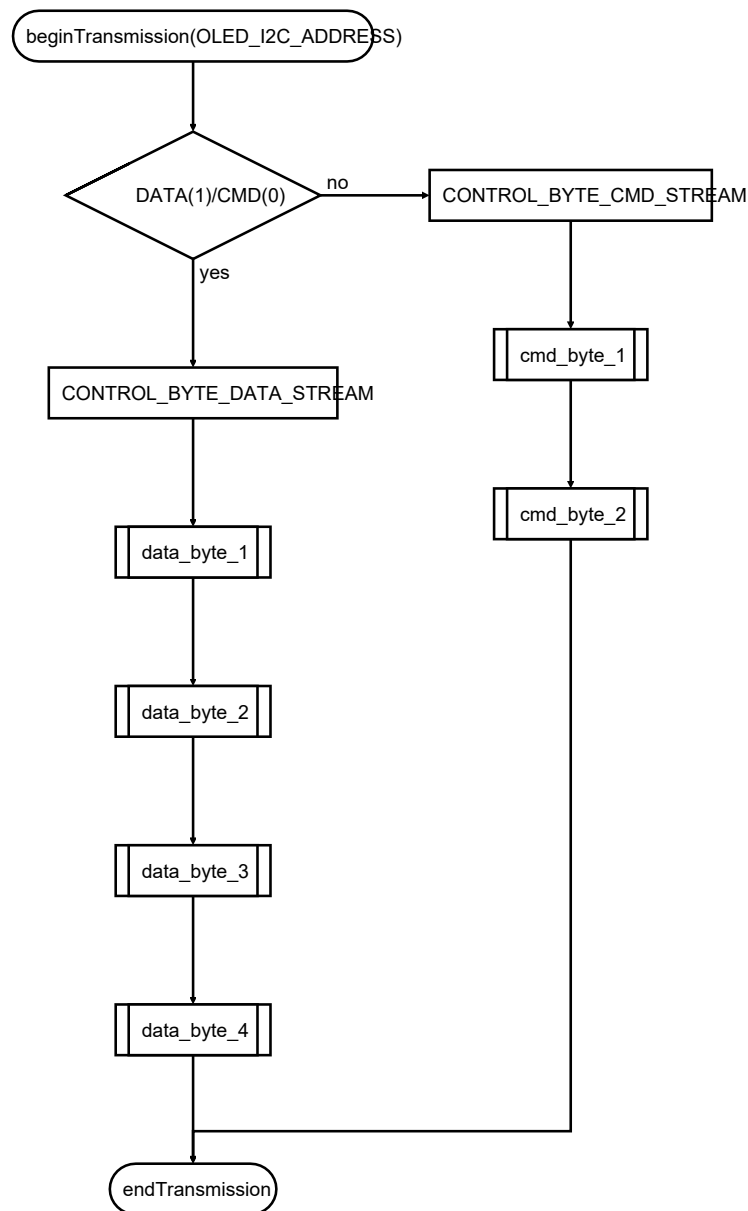
The [oled_test](#) sketch is flash-ready for the Arduino plugged to a SSD1306 OLED over I2C - and - it's well commented .

Connections

Arduino A4 to OLED SDA

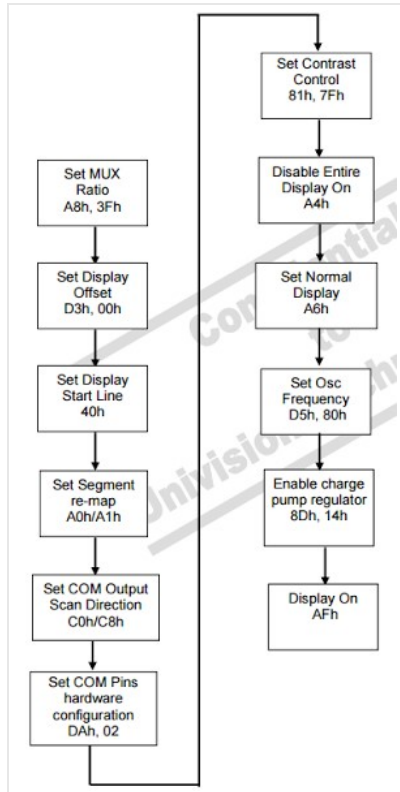
Arduino A5 to OLED SCL

The I2C control of the OLED is pretty simple. To send a command to the OLED, you'd need to transmit a Command Control Byte first, and similarly to write to the GDDRAM, you'd need to transmit a Data Control Byte.



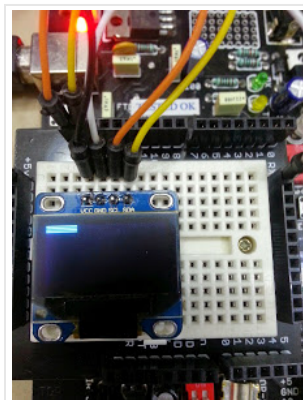
setup

The entire I2C comm is handled by the [Arduino Wire](#) library and is easy to setup. The `OLED_init()` function follows the software config instructions detailed on pg.64 of the datasheet. The various command bytes are sent over as a stream. Most of the values are set as the default `RESET` state values as described in the Command Table (Section 9 in the sheet). The addressing mode is set to `Horizontal`.



As I had said before, the ROW and COL mapping is completely customizable to any corner. This is described in the Hardware Config table on pg.31 (`Set-Segment-Remap` and `Set-COM-Output-Scan-direction`). For example, if the board pins are north, then by setting those two to `0xA1` and `0xC8` , you'd have the xy(0,0) at top-right. Whereas, if you place the board such that the pins are south, you'd need to transmit `0xA0` and `0xC0` - which is the default setting after a *reset*.

board placed with pins facing north



board placed with pins facing south



loop

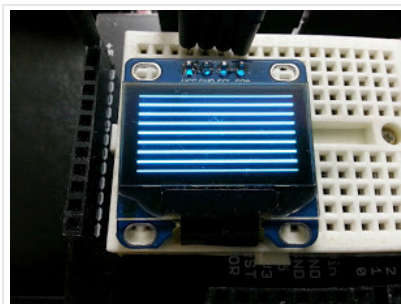
At every frame, I reset the column and page address to the full range, ie: COLUMN from 0 to 127 (0x7F) and PAGE from 0 to 7. This sets the address line at (seg0,page0) - the top-right.

```
Wire.beginTransmission(OLED_I2C_ADDRESS);
Wire.write(OLED_CONTROL_BYTE_CMD_STREAM);
Wire.write(OLED_CMD_SET_COLUMN_RANGE);
Wire.write(0x00);
Wire.write(0x7F);
Wire.write(OLED_CMD_SET_PAGE_RANGE);
Wire.write(0);
Wire.write(0x07);
Wire.endTransmission();

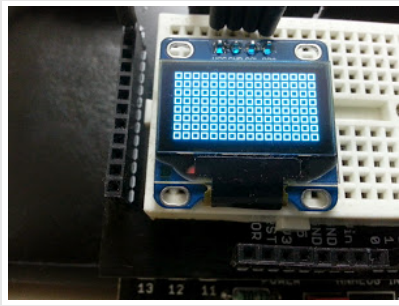
for(uint16_t i=0;i<1024;i++){
  Wire.beginTransmission(OLED_I2C_ADDRESS);
  Wire.write(OLED_CONTROL_BYTE_DATA_STREAM);
  for (uint8_t x=0; x<16; x++) {
    Wire.write(0x81);
    i++;
  }
  i--;
  Wire.endTransmission();
}
```

128x64 is 8192 bits. If I am to transfer them in terms of bytes, then I would need to send 1024 bytes. In the above loop, this is done in 16 byte burst transfers. In the inner loop, you can set what type of byte you'd want to send over. I have detailed two patterns as well - pattern1 and pattern2. Ensure that there's only one write command in the inner loop.

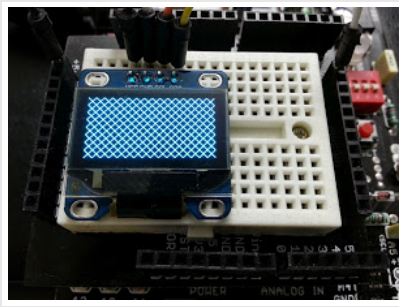
```
Wire.write(0x81)
```



```
Wire.write(pattern1[x])
```



```
Wire.write(pattern2[x])
```



Cool, huh? But, this simply shows how to interface the SSD1306 OLED to an Arduino.

Next up, true implementation of a buffered display on the SSD1306 OLED with higher FPS.

PART II - Arduino and SSD1306

Scripted by Sonal Pinto



Labels: Arduino, Tech

14 comments



Add a comment

Top comments

**RCSTART** 2 years ago · Shared publicly

Hey you are a genius regrading the address staff. I already was ready to decide a buy a broken display when I find your page and put the right address. Now is working OK. Thanks a lot !!!

+1 1 · Reply

**Sonal Pinto** 2 years ago

Nice! Glad to be of help. Maybe when you are done with your project, you could show me a photo or two :)

**Pradeep Wagre** 7 months ago · Shared publicly

How can I display bigger fonts on ssd1306?

1 · Reply

**The Robot Can Talk via Google+** 2 years ago · Shared publicly**Interfacing the Arduino with an SSD1306 driven OLED Display - part 1**

Interfacing the Arduino with an SSD1306 driven OLED Display - part 1 Who doesn't want a spiffy display for their Arduino!? An tiny OLED screen will let you do just that. I decided to get myself a display module for a intriguing project that I have in mind -...

1 · Reply

**Dipak Bhagat** 9 months ago · Shared publicly

Hi, I am Dipak. Recently i brought digispark attiny85 and 0.96" OLED ssd1306. when I interfaced with arduino uno were it was quite easy. BUT i found very difficulty while interfacing with digispark attiny85 module. Please help me with providing me the code and library.

+1 1 · Reply

**code beat** 7 months ago

Got the same problem on a digispark, what I do, what I try, no display. In USI_TWI_Master.h of TinyWire there is a port and pin configuration. The library use PINB0 and PINB2 after define `_AVR_ATtiny85_`. Used this to connect display and doesn't work. Download any examples, doesn't work. Tried another display (white one

**J Yang** 11 months ago · Shared publicly

Thank you for this awesome post. Your information on "I2C slave address" saved a lot of time for me.

1 · Reply

**Octavio Francisco Salcido Rocha** 1 year ago · Shared publicly

Hi! My wife's car is a 2002 pontiac grand am and the shift indicator (P,R,N,D,3,2,1) is damaged and very difficult to find the parts here, my question is if could be possible to use one of this chips to do the job of show the shift letters??? I was trying to use a 27c64 eprom plus clock plus decoders to make the project but find this article, may somebody help me???

1 · Reply

**victor oropeza** 5 months ago

Just buy a new car. You can buy an 02' grand anything for about 2 grand. lol. Come

[Newer Post](#)

[Home](#)

[Older Post](#)

[Subscribe to: Post Comments \(Atom\)](#)

Blog Archive

► [2016](#) (2)

▼ [2015](#) (3)

▼ [March](#) (2)

[Interfacing the Arduino with an SSD1306 driven OLE...](#)

[Interfacing the Arduino with an SSD1306 driven OLE...](#)

► [February](#) (1)

► [2014](#) (11)

► [2013](#) (18)

► [2012](#) (21)

Simple theme. Theme images by [fpm](#). Powered by [Blogger](#).