

한글 출력이 가능한 디스플레이 콘트롤러

- 이글의 최종 버전은 <https://cafe.naver.com/kpopenproject> 에서 확인하시기 바랍니다.
- 이 글은 전자, 전산 분야의 전문가가 아닌 비전공자의 경험으로 작성한 글입니다. 따라 잘못된 정보가 있을 수 있으므로 무작정 따라하지 마시고 검토 후 구현하시길 바랍니다. 그리고 잘못된 내용이 있을 경우 저에게 메일(khkim@mapescn.com)로 알려주시면 바로 수정 배포하도록 하겠습니다.

■ 서론

아두이노에 사용할 수 있는 출력장치는 많은 종류들이 있습니다.
가장 기본적으로는 처음 시작하면서 제일 먼저 사용해보는 LED부터
7-세그먼트, 4digit 7-세그먼트, LED MATRIX, CLCD, GLCD, OLED, TFT 출력
장치 등 제가 사용해본 출력 장치만 해도 6~7 종류가 되는 듯 합니다.
아두이노의 경우 대부분 라이브러리가 잘 만들어져 있어서
회로도를 참조하여 만들면 그리 어렵지 않게 만들 수가 있었습니다.

하지만 대부분의 라이브러리가 해외에서 만들어져서 한글을 지원을 하지 않
았습니다.

그러던 중 직접 빈 적은 없지만 인터넷으로 많은 활동 및 자료공개를 하시는
인터넷 카페의 선생님께서 한글 LCD에 대한 의견을 제안해 주셨습니다.

한글이 출력 가능한 LCD를 만들면 어떨지에 대해서...
물론 이 LCD는 현재 인터넷만 검색하셔도 개발되어 판매되고 있습니다.
하지만 대중화가 덜 되고 사용량이 적어서 인지 가격이 만만치 않았습니다.
그리고 꼭 모듈로 된 제품이 아닌 아두이노를 이용하여 직접 만드는 것도
꽤 의미있는 작업이 될 것 같습니다.

이번에는 한글 출력이 가능한 출력 장치 연결 보드를 만들어 보겠습니다.

■ 대상 디스플레이 장치

최종 구현에서 어느정도까지 완성될지 확신은 없지만
구상중인 기능은

GLCD(128x64)에 대한 한글 출력

OLED(128x64)에 대한 한글 출력(I2C통신 모듈)

OLED(128x64)에 대한 한글 출력(SPI통신 모듈)

1.8인치 TFT(128x160)에 대한 한글 출력

LED MATRIX(16x16, 16x32)에 대한 한글 출력

을 기준으로 시도해 보도록 하겠습니다.

해당 제품의 사진은 아래와 같습니다.

GLCD(128x64)



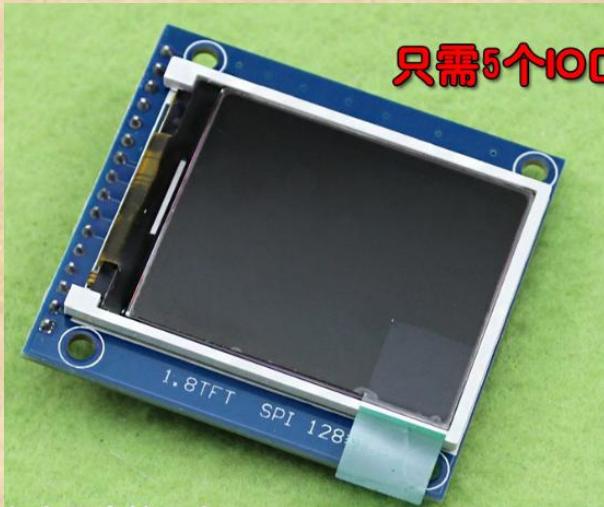


OLED(I2C) 방식



OLED(SPI) 방식

TFT(SPI) 방식



LED MATRIX(MATRIX를 4개 이상을 연결하여 16x16이상의 해상도로 구현)



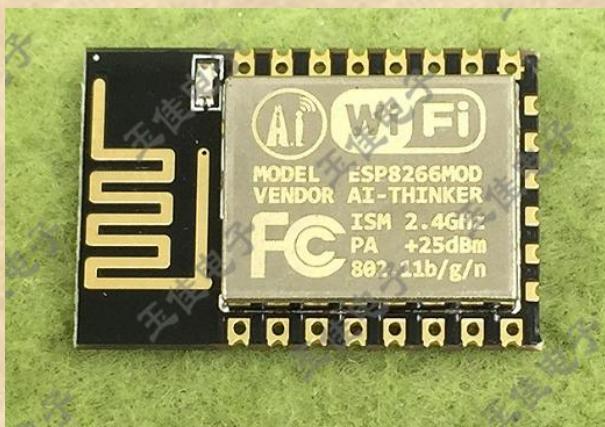
이 모든 표시 장치를 한개의 보드에서 처리 가능하도록 하면 좋겠다는 생각
이지만

실제 구현을 할 경우 보드에 모두 배치가 가능할지는 모르겠습니다.

좀 더 구현 후 한개의 보드에 모두 구현이 가능할지 확인해야 할 듯 합니다.
안되면 각각의 모듈을 위한 개별 보드로 구현하겠습니다.

■ MCU의 선정

한글을 출력하기 위해서는 영문 보다는 좀더 많은 데이터가 필요합니다. 우선 폰트 파일을 저장하고 있어야 하며, 완성형-조합형 한글 변환 테이블도 저장을 하여야 합니다. 또한 각 디스플레이 장치가 SPI통신 및 I2C통신을 하게 되고, 컨트롤러와 아두이노(메인 MCU)는 시리얼 통신을 할 계획입니다. 다양한 통신이 구현 가능하며, 저장 용량도 큰 MCU를 생각해보면 제일 적합한 MCU가 ESP-12이 아닌가 싶습니다. ESP-12 MCU의 경우 내부 SPIFFS가 4MByte로 이중 1MByte는 프로그램 저장 메모리로 사용이 되고 나머지 3MByte가량의 데이터를 저장할 수 있습니다. 이는 아두이노등을 사용할 경우 앞서 말한 폰트파일과 코드 변환테이블을 저장하기 위하여 SD카드등의 보조 저장장치가 있어야 하나 ESP-12모듈을 사용할 경우 내부 SPIFFS에 저장하면 되므로 가장 적합한 MCU 인 것 같습니다.



■ 한글 출력에 필요한 자료

한글 출력력을 하려면 필요한 자료들입니다.

한글 출력력을 하려면 기본 한글 FONT가 필요합니다.

요즘은 벡터방식의 폰트를 대부분 사용하지만

예전 1990년대에는 PC에도 DOS운영 체제에 비트맵 방식의 폰트를 사용했습니다.

(그 시절 DOS운영체제에 터보씨로 한글 라이브러리를 공부하던 기억이 새록새록 합니다.)

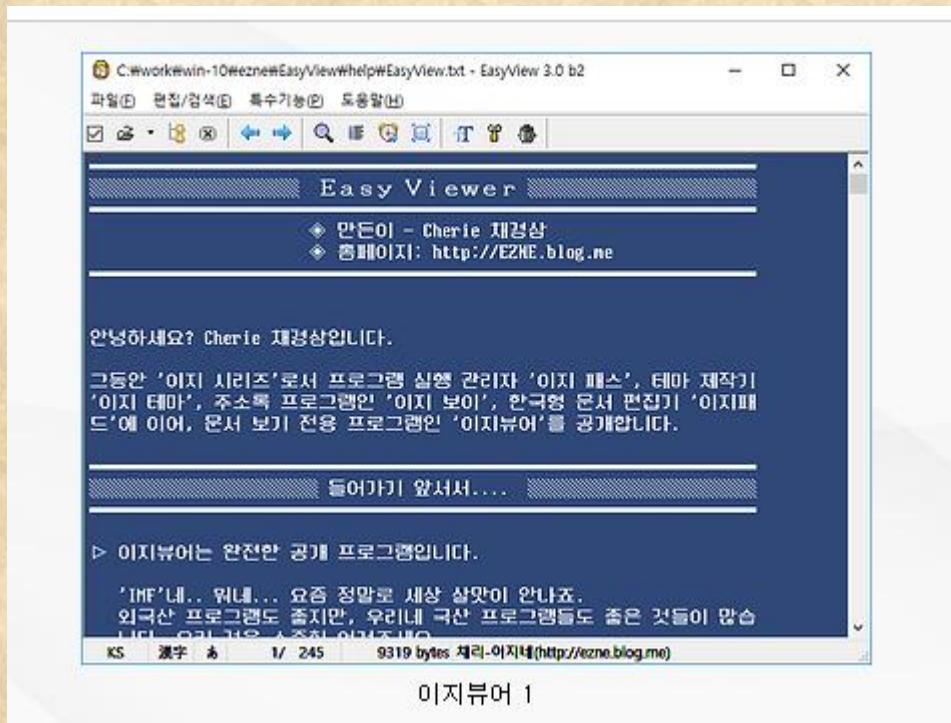
벡터 방식과 비트맵 방식 폰트의 차이점은 여기서 일일히 설명하기는 힘드므로

인터넷 검색을 활용하시기 바랍니다.

예전의 비트맵 폰트를 찾을 수 있을지 검색을 해보니 요즘 거의 찾을 수가 없네요.

열심히 폰트를 검색하던 중

아래와 같은 프로그램을 찾았습니다. 프로그램 이름은 이지뷰어이며 네이버 자료실에서 다운 받을 수 있는 프리웨어입니다.



이 프로그램은 조합형 비트맵 한글 글꼴을 사용하는 한글 뷰어입니다.

kProject 열정풀왕 프로젝트

이 프로그램을 설치하면 폰트파일도 같이 설치되므로 이 폰트 파일을 사용하면 될 것 같습니다.

프로그램을 설치하면 Font폴더 아래 아래와 같이 한글 폰트가 여러개 설치가 됩니다.

이름	수정한 날짜	유형	크기
Gray1.spc	1994-07-03 오후 ...	PKCS #7 인증서	4KB
Gray2.spc	1994-07-03 오후 ...	PKCS #7 인증서	4KB
H_Dmyong.han	1994-07-03 오후 ...	HAN 파일	12KB
H_Soft.han	1994-07-03 오후 ...	HAN 파일	12KB
H01.han	1994-07-03 오후 ...	HAN 파일	12KB
H02.han	1994-07-03 오후 ...	HAN 파일	12KB
H03.han	1994-07-03 오후 ...	HAN 파일	12KB
H04.han	1994-07-03 오후 ...	HAN 파일	12KB
H05.han	1994-07-03 오후 ...	HAN 파일	12KB
H06.han	1994-07-03 오후 ...	HAN 파일	12KB
H07.han	1994-07-03 오후 ...	HAN 파일	12KB
H08.han	1994-07-03 오후 ...	HAN 파일	12KB
H09.han	1994-07-03 오후 ...	HAN 파일	12KB
H10.han	1994-07-03 오후 ...	HAN 파일	12KB
H11.han	1994-07-03 오후 ...	HAN 파일	12KB
H12.han	1994-07-03 오후 ...	HAN 파일	12KB
H13.han	1994-07-03 오후 ...	HAN 파일	12KB
H14.han	1994-07-03 오후 ...	HAN 파일	12KB
H15.han	1994-07-03 오후 ...	HAN 파일	12KB
Han.han	1994-07-03 오후 ...	HAN 파일	12KB

파일의 사이즈를 보니 11520 BYTE입니다.

조합형 폰트의 경우 초성 8벌, 중성 4벌, 종성 4벌로 구성이 됩니다.

초성은 총 20자(1개 빈 글자 포함), 중성은 22자(1개 빈 글자 포함), 종성은 28자(1개 빈 글자 포함)입니다.

따라서 FONT 파일에 정의 되어야 하는 문자의 개수는

$$20 \times 8 + 22 \times 4 + 28 \times 4 = 360$$

총 360글자의 모양이 폰트파일에 정의되어야 하며

1글자는 16x16 픽셀의 사이즈를 가지며, 한열에 2Byte씩 총 16열을 가지게 됩니다.

따라서 1글자는 32Byte이고

총 폰트파일의 크기는 $360\text{자} * 32\text{Byte} = 11520\text{ Byte}$ 가 정의되어 있어야 합니다.

앞서 다운받은 이지뷰어 프로그램의 폰트 역시 11520 Byte이므로

초성 8벌, 중성 4벌, 종성 4벌 폰트일 것으로 예상이 됩니다.

정확한건 좀더 구현해 보면 결과가 나오리라 생각이 듭니다.

■ 조합형 한글 코드의 분석

조합형 한글은 2Byte 코드로 한글 1글자를 표시를 합니다.

ASCII 코드의 경우 1Byte가 영문 1글자를 표시하지만 한글의 경우 2Byte를 사용하여 표현하며 그 2Byte, 총 16비트의 코드는

초기 1bit는 한글인지 영문인지를 구분하는 코드로 사용하고

이후 5bit가 초성, 다음 5bit가 중성, 다음 5bit가 종성으로 총 16bit가 구성이 됩니다.

각 값에 대한 조합형 코드의 문자는 아래와 같습니다.

한글 조합형 코드표

비트값		초성	중성	종성
10 진	2 진			
0	00000			
1	00001	<채움>		<채움>
2	00010	ㄱ	<채움>	ㄱ
3	00011	ㄲ	ㅏ	ㄲ
4	00100	ㄴ	ㅐ	ㄴ
5	00101	ㄷ	ㅑ	ㄷ
6	00110	ㄸ	ㅒ	ㄸ
7	00111	ㄹ	ㅓ	ㅓ
8	01000	ㅁ		ㅁ
9	01001	ㅂ		ㅂ
10	01010	ㅃ	ㅔ	ㅔ
11	01011	ㅅ	ㅕ	ㅕ
12	01100	ㅆ	ㅘ	ㅆ
13	01101	ㅇ	ㅗ	ㅇ
14	01110	ㅈ	ㅙ	ㅙ
15	01111	ㅉ	ㅚ	ㅉ
16	10000	ㅊ		ㅊ
17	10001	ㅋ		ㅋ
18	10010	ㅌ	ㅚ	ㅌ
19	10011	ㅍ	ㅟ	ㅍ

20	10100	ㅎ	ㅌ	ㅍ
21	10101		ㅋ	ㅅ
22	10110		ㅖ	ㅆ
23	10111		ㅟ	ㅇ
24	11000			ㅈ
25	11001			ㅊ
26	11010		ㅠ	ㅋ
27	11011		ㅡ	ㅌ
28	11100		ㅡ	ㅍ
29	11101			ㅎ
30	11110			
31	11111			

자 이제 폰트파일의 구성을 확인해 보겠습니다.

폰트파일은 앞서 말한바와 같이 총 11520Byte입니다.

초성 8벌은 20문자 x 8벌 = 160글자이며 160자 x 32Byte = 5120 Byte

중성 4벌은 22문자 x 4벌 = 88글자이며 88자 x 32Byte = 2816 Byte

종성 4벌은 28문자 x 4벌 = 112글자이며 112자 x 32Byte = 3584 Byte입니다.

따라서 앞 0~5119바이트 위치의 5120Byte는 초성 8벌의 문자가 정의되어 있습니다.

이후 5120~7935의 2816Byte는 중성 4벌에 대한 문자가 정의됩니다.

이후 7936~11519의 3584Byte는 종성 4벌에 대한 문자가 정의됩니다.

자 이제는 각 초성 8벌, 중성 4벌, 종성 4벌이라고 되어 있는 벌에 대해서 확인해 보겠습니다.

한글로 초성 ‘ㄱ’을 표시한다고 할 경우 모든 ‘ㄱ’의 모양이 동일하지 않습니다.

예를 들자면 ‘그’ 자와 ‘가’의 ‘ㄱ’자의 모양이 모두 다릅니다.

더 확장해 보자면 다음글자 ‘가고구과궈각곡곽’에 사용된 8자의 초성 ‘ㄱ’의 모양은 모두 다릅니다.

그래서 초성 ‘ㄱ’에 대한 종류를 총 8개로 나누고 이 모양이 폰트파일에 모두 저장이 되어 있습니다.

이렇게 같은 초성에 대하여 모양을 구분해 둔걸 벌이라고 합니다.

따라서 초성의 경우 8개의 모양을 구분해 두고 중성은 4개 종성은 4개로 구분을 해 두었습니다.

어느분이 개발을 하셨는지 기막하게 잘 만드신 것 같습니다.

이와 같은 벌을 초성 중성 종성으로 정리하면 아래와 같습니다.

초성

중성

- 중성 1벌 : 받침없는 'ㄱㅋ' 와 결합
 중성 2벌 : 받침없는 'ㄱㅋ' 이외의 자음
 중성 3벌 : 받침있는 'ㄱㅋ' 와 결합
 충성 4벌 : 받침있는 'ㄱㅋ' 이외의 자음

종성

- | | |
|-------|--------------------|
| 종성 1벌 | : 중성 'ㅏ ㅑ ㅕ' 와 결합 |
| 종성 2벌 | : 중성 'ㅓ ㅕ ㅕ ㅕ ㅕ ㅕ' |
| 종성 3벌 | : 중성 'ㅔ ㅖ ㅖ ㅖ ㅖ ㅖ' |
| 종성 4벌 | : 충성 'ㅗ ㅜ ㅠ ㅡ' |

이제 한글 문자를 가지고 이 문자가 초성의 몇벌을 사용하는지 중성의 몇벌을 사용하는지 구분을 하여야 합니다.

초성의 경우 받침이 있는지와 중성의 문자에 따라 벌수가 결정이 되며,
중성의 경우 받침이 있는지와 초성이 ‘ㄱ’, ‘ㅋ’ 인지에 따라 벌수를 결정

중성의 경우 중성의 종류에 따라 벌수를 결정하게 됩니다

제 이론 전개하면 아래와

종성이 없을 경우(중성의 종류에 따라 초성의 벌수 결정)

ㅏ	ㅐ	ㅑ	ㅒ	ㅓ	ㅕ	ㅗ	ㅙ	ㅜ	ㅞ	ㅛ	ㅕ	ㅘ	ㅕ	ㅕ	ㅕ	ㅕ	ㅡ	ㅕ	ㅣ	
1	1	1	1	1	1	1	1	2	4	4	4	2	3	5	5	5	3	2	4	1

종성이 있을 경우(중성의 종류에 따라 초성의 벌수 결정)

ㅏ	ㅐ	ㅑ	ㅒ	ㅓ	ㅕ	ㅗ	ㅙ	ㅜ	ㅞ	ㅛ	ㅕ	ㅘ	ㅕ	ㅕ	ㅕ	ㅕ	ㅡ	ㅕ	ㅣ	
6	6	6	6	6	6	6	6	7	8	8	8	7	7	8	8	8	7	7	8	6

■ 중성의 벌수 결정

종성이 없을 경우

초성이 'ㄱ' 이거나 'ㅋ'인 경우 : 1벌

초성이 'ㄱ', 'ㅋ'이 아닌 경우 : 2벌

종성이 있을 경우

초성이 'ㄱ' 이거나 'ㅋ'인 경우 : 3벌

초성이 'ㄱ', 'ㅋ'이 아닌 경우 : 4벌

■ 종성의 벌수 결정

ㅏ	ㅐ	ㅑ	ㅒ	ㅓ	ㅕ	ㅇ	ㅈ	ㅗ	ㅙ	ㅘ	ㅚ	ㅛ	ㅞ	ㅜ	ㅞ	ㅙ	ㅘ	ㅚ	ㅡ	ㅕ	ㅣ
1	3	1	3	2	3	2	3	4	1	3	2	4	4	2	3	2	4	4	2	2	

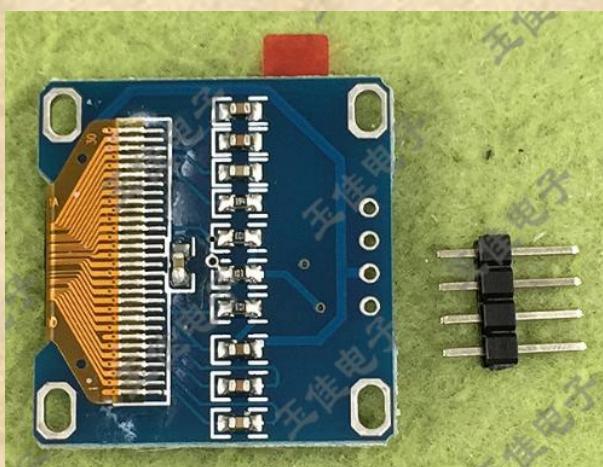
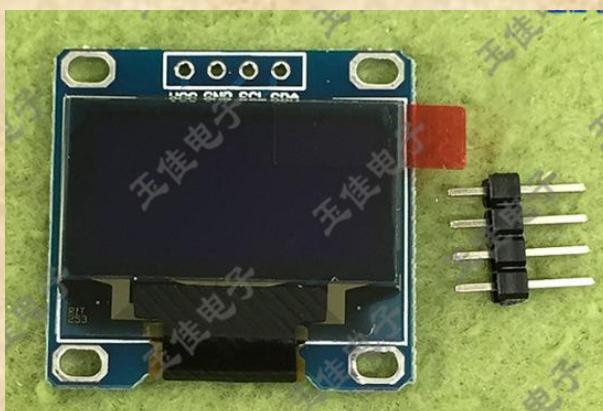
이후 문자의 표시는 한글 문자코드 16bit를 초성 중성 종성으로 나누고 각 초성, 중성, 종성에 대한 벌수를 구한후 해당 벌수에 대응하는 문자 형상을 FONT파일에서 읽어와 표시장치에 출력하면 되겠습니다.

자 여기까지 한글 조합형 비트맵 폰트(16x16)의 구성과 각 초성 중성 종성에 대한 벌수 계산 방법을 알아봤습니다.

■ 디스플레이 장치 출력하기(OLED – I2C)

현재글(한글 출력이 가능한 디스플레이 콘트롤러)와 직접적인 연관이 있는 건 아니지만 한글을 출력하려면 어쨌든 출력 장치가 있어야 합니다.
현재의 목표는 다양한 디스플레이 장치를 연결 할 수 있는 보드를 만들고자 하지만 한번에 할 수는 없으므로 우선 OLED 출력 장치를 이용하겠습니다.
OLED-I2C용을 우선 선정한 이유는 그냥 지금 제 책상에서 제일 쉽게 찾아져서 그렇습니다...^^

사용한 OLED 제품은 아래와 같습니다.

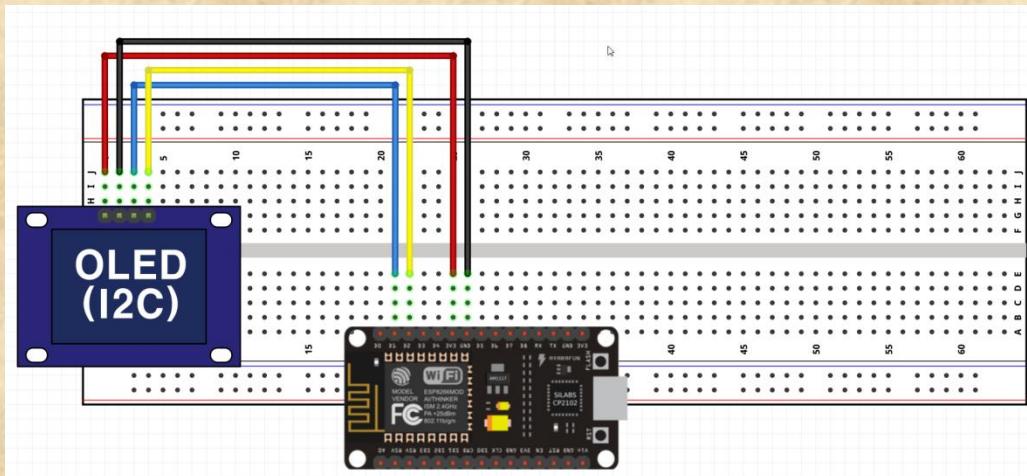


해상도는 128x64이며 I2C통신을 합니다.

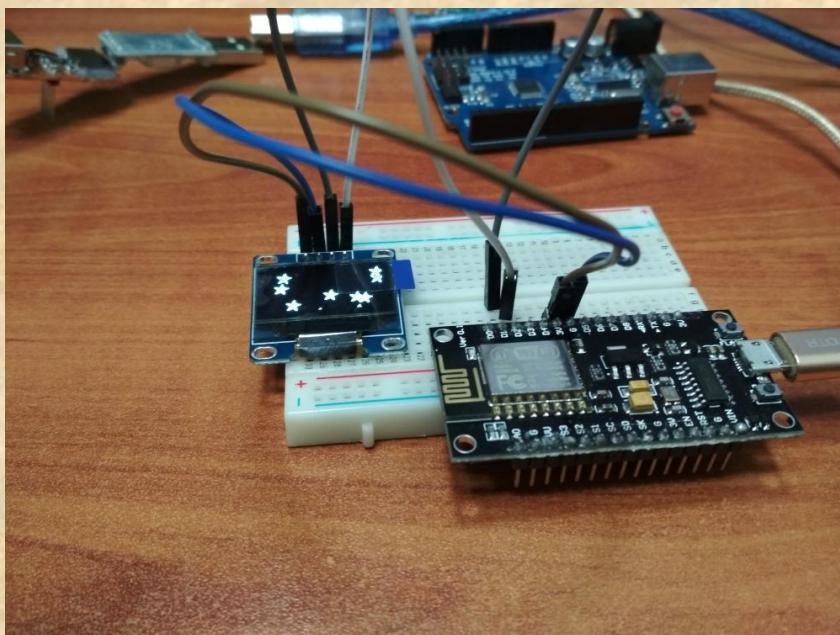
I2C용 OLED의 사용방법은 데이터시트를 이용하여 일일히 프로그램을 작성할 수도 있겠지만 지금 하고 있는 한글 출력 콘트롤러와 직접적인 연관이 없으므로 아두이노 라이브러리를 사용하도록 하겠습니다.

사용한 라이브러리는 https://github.com/adafruit/Adafruit_SSD1306 입니다.

라이브러리의 사용법은 본 글의 주제와 벗어나므로 생략하고
작동 테스트만 해 보도록 하겠습니다.



윗 회로도와 같이 연결하고 소스코드를 업로딩 합니다.
소스코드 및 회로도 다운로드 : [korean_display_1](#)



윗코드는 라이브러리의 기본 예제입니다.

이제 이 라이브러리를 이용하여 원하는 좌표에 점을 찍는 소스를 작성해 복니다.

별내용은 없고 x, y좌표를 이용하여 점을 찍는 소스코드입니다.

kProject 열정동창 프로젝트

소스코드 및 회로도 다운로드 : korean_display_2

■ SPIFF에 FONT 파일 업로드 하기

ESP8266 MCU의 제품에는 4MByte의 SPIFF가 장착되어 있습니다.

이중 1M가 프로그램 영역의 flash메모리로 사용이 되고 나머지 3M가 SPIFF 파일 시스템으로 사용이 됩니다.

SPIFF의 경우 일부를 EEPROM으로도 사용하기 때문에 3M중 EEPROM으로 할당한 양을 제외하면 SPIFF로 파일을 저장할 수 있습니다.

SPIFF란 SPI FLASH FILE SYSTEM으로 ESP8266의 데이터 파일들을 저장할 수 있는 영역입니다.

이 파일 시스템에는 자주 변경되지 않는 파일(Web페이지나, 설정값, 센서데이터)등을

저장하는데 사용할 수 있습니다.

여기에 저장된 자료는 리셋이 되어도 변경되지 않지만 속도가 느리기 때문에 자주 변경되는 자료의 저장에는 적합하지 않습니다.

폴더 역시 만들 수 없고 일반 파일명으로만 저장 가능합니다.

데이터 업로딩 하기

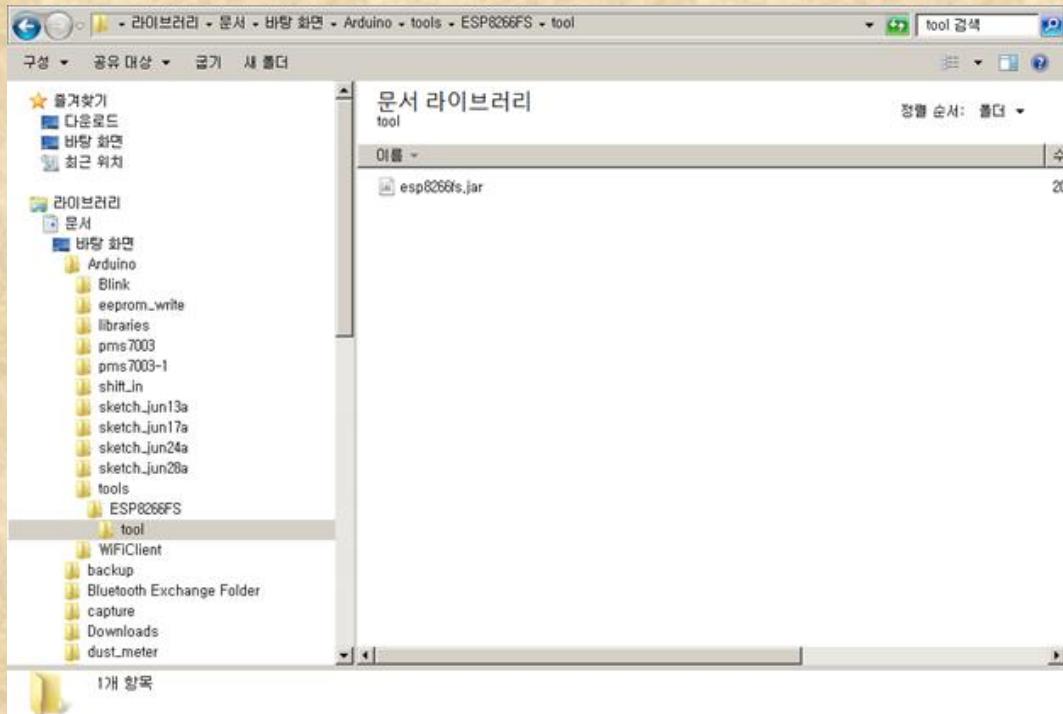
SPIFF영역에 파일을 업로딩하기 위해서는 별도의 툴을 설치하셔야 합니다.

<https://github.com/esp8266/arduino-esp8266fs-plugin/releases/download/0.1.3/ESP8266FS-0.1.3.zip>

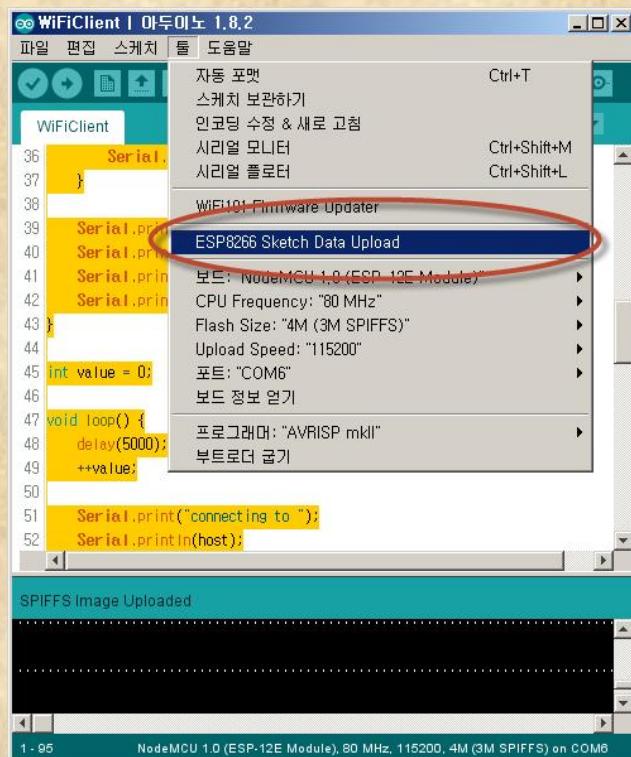
위 링크의 자료를 다운 받아 압축을 풀면 나오는 esp8266fs.jar 파일을 [home_dir]\Arduino\tools\ESP8266FS\tool\esp8266fs.jar에 복사해 넣습니다.

제 경우의 폴더는 아래 그림과 같았으며 컴퓨터마다 조금씩 다를 수 있습니다.

kProject 열정 풍랑 프로젝트



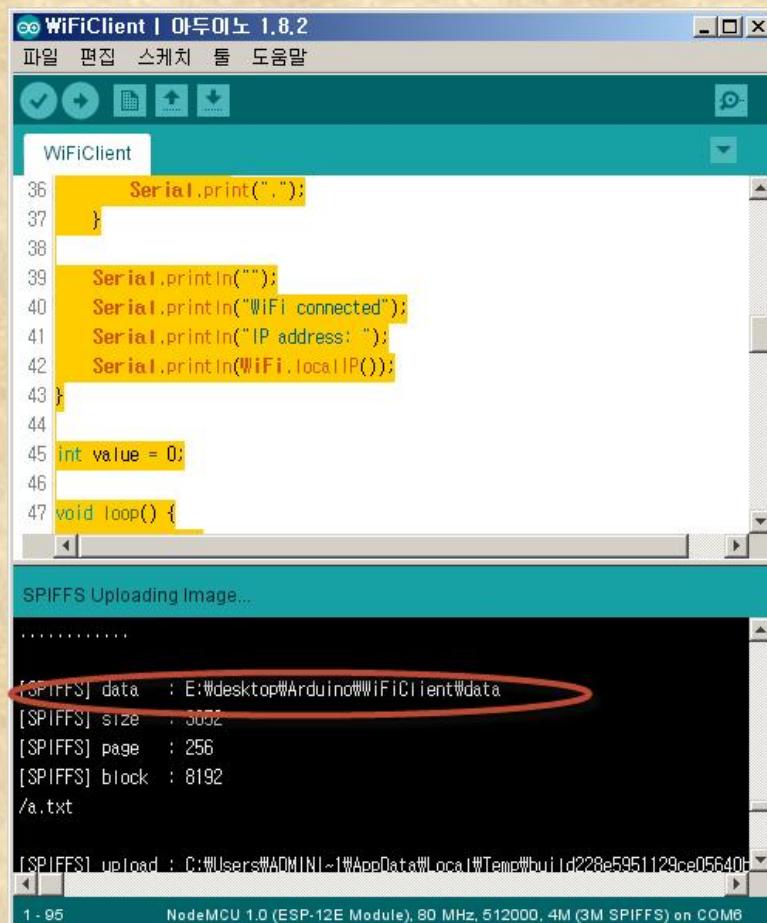
그리고 나서 아두이노 IDE를 다시 실행하면 다음과 같은 메뉴가 생성되 있는 걸 확인 할 수 있습니다.



이제 업로드할 파일을 아두이노 스케치가 저장되어 있는 폴더 아래 data 폴더를 만들고 파일을 복사

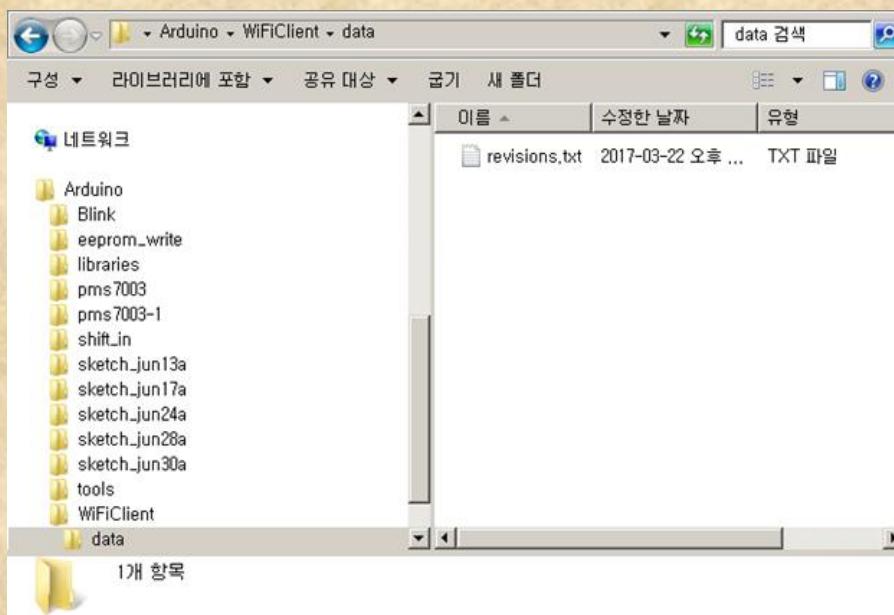
kProject 열정 풍랑 프로젝트

ESP8266 Sketch Data Upload를 선택하시면
해당 폴더에 있는 자료들이 업로딩이 됩니다.
폴더를 못 찾으시겠으면 업로딩을 실행하면 아래 창에 업로딩할 폴더가
출력이 되니 확인하시고 파일을 넣으시면 됩니다.



전 아두이노 폴더에 있는 revision.txt 파일을 위 데이터 폴더에 복사해 두었습니다.

kProject 열정 풍향 프로젝트



```
1 #include "FS.h"
2
3 void setup()
4 {
5     Serial.begin(115200);
6     bool result = SPIFFS.begin();
7     Serial.println("SPIFFS opened: " + result);
8
9     // opens the file "revision.txt" in read-mode
10    File f = SPIFFS.open("/revisions.txt", "r");
11
12    if (!f)
13    {
14        Serial.println("File doesn't exist!");
15        while (1);
16    }
17    else
```

kProject 열정 풍랑 프로젝트

```
18  {
19      Serial.println("#####START#####");
20      while (f.available())
21      {
22          String line = f.readStringUntil('\n');
23          Serial.println(line);
24      }
25      Serial.println("#####END#####");
26  }
27  f.close();
28 }
29
30 void loop()
31 {
32 }
```

소스코드 및 회로도 다운로드 : korean_display_3

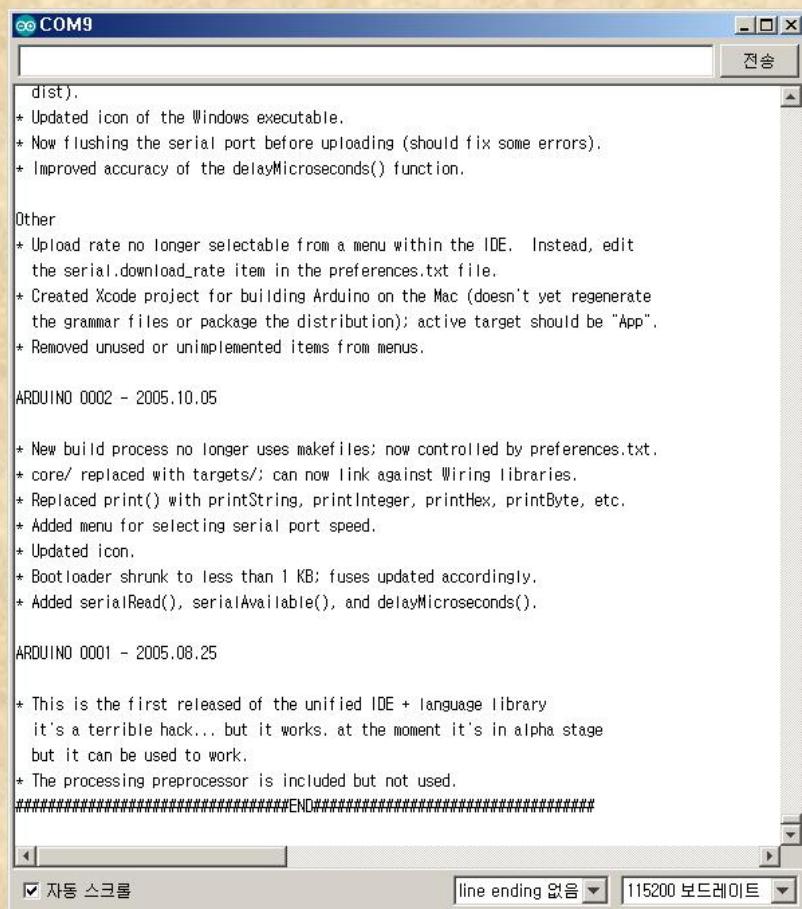
이제 위의 소스코드를 업로딩 합니다.

위의 코드는 SPIFF의 revisions.txt 파일을 읽어 시리얼 모니터에 출력하는 예입니다.

사용된 명령들이 복잡하지 않으므로 보시면 대충 감으로 사용 방법은 알 수 있을 것 같습니다.

이제 아두이노 IDE의 시리얼 모니터를 확인하시면 아래와 같이 SPIFF영역에 업로드 된 revisions.txt 파일의 내용이 출력되는 걸 보실 수 있습니다.

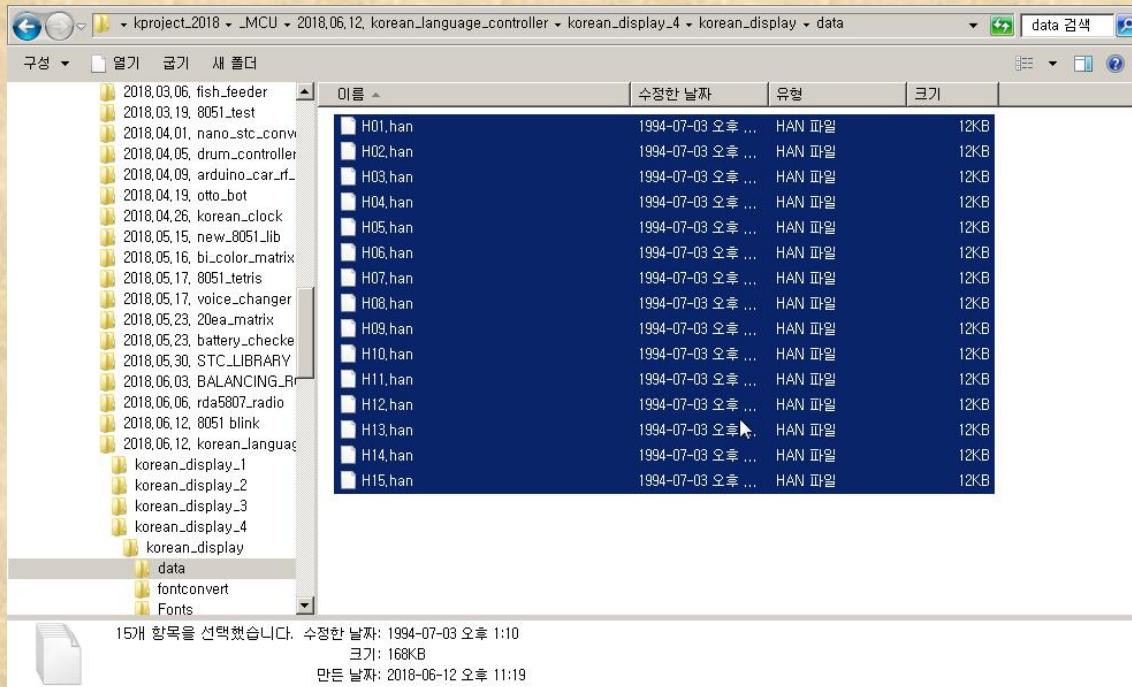
kProject 열정 풍랑 프로젝트



이제 많지 않은 양의 데이터라면 SD카드를 사용하지 않으셔도 저장할 수 있습니다.

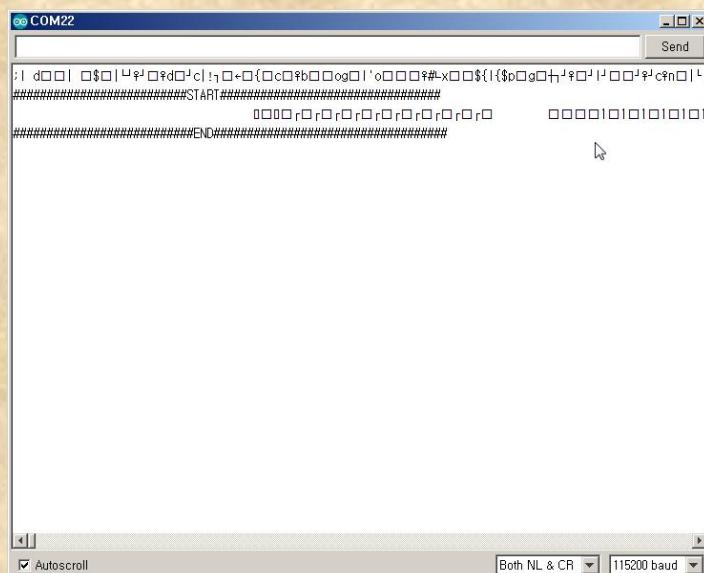
그럼 앞서 준비했던 한글폰트를 SPIFF에 올리도록 하겠습니다.

kProject 얼굴인식 프로젝트



데이터 폴더에 한글폰트를 넣고 SPIFF 영역에 파일을 업로딩 합니다.

이전과 같이 시리얼 모니터로 “ H01.han” 파일을 출력해 보면 폰트파일은 텍스트파일이 아닌 바이너리 파일이므로 깨진 문자들이 출력되는걸 확인하실 수 있습니다.



소스코드 및 회로도 다운로드 : [korean_display_4](#)

■ FONT 파일을 디스플레이하기

지금까지 적지 않은 과정을 거쳐 왔지만 화면에 보여진게 혹은 어떤 결과도
가시적인게 없습니다.

우선 갈 길은 멀지만 모라도 출력을 해 보겠습니다.

폰트파일을 읽어서 OLED 디스플레이 장치에 출력을 해 보도록 하겠습니다.

방법은 어렵지 않습니다. 이미 만들어진 소스코드를 조합해서 폰트파일을
읽어 OLED에 뿌려주기만 하면 됩니다.

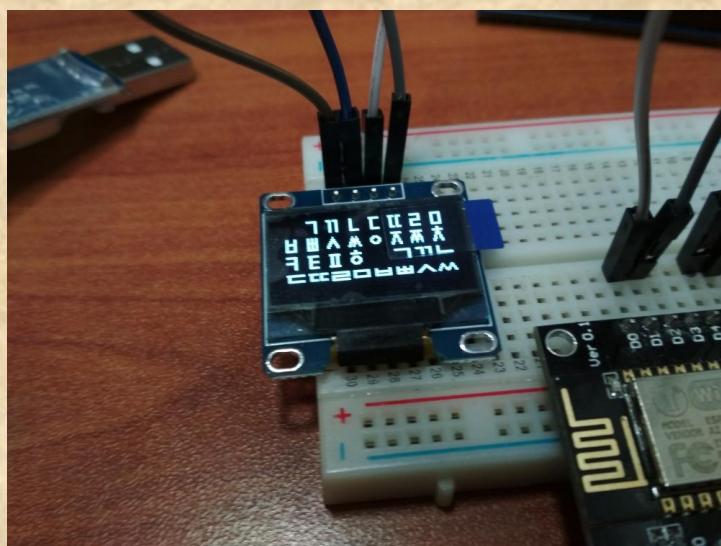
아래 소스코드는 폰트(" H01.han") 파일의 앞부분 32개 문자를 OLED에 출
력해 주는 코드입니다.

```
1234 12345678901234567890123456789012345678901234567890123456789012345678901234
 5 567890

1 #include <SPI.h>
2 #include <Wire.h>
3 #include "Adafruit_GFX.h"
4 #include "Adafruit_SSD1306.h"
5 #include "FS.h"
6
7 #define OLED_RESET LED_BUILTIN
8 Adafruit_SSD1306 display(OLED_RESET);
9 File f;
10
11 void setup()
12 {
13     Serial.begin(115200);
14     bool result = SPIFFS.begin();
15     Serial.println("SPIFFS opened: " + result);
16
17     f = SPIFFS.open("/H01.han", "r");
18
19     if (!f)
20     {
21         Serial.println("File doesn't exist!");
22         while (1);
23     }
24     display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
25     display.clearDisplay();
26 }
27
28 int index = 0;
29 unsigned char val;
30
31 void loop()
32 {
33     for (int j = 0; j < 4; j++)
34         for (int i = 0; i < 8; i++)
35         {
36             draw_font(i * 16, j * 16, j * 8 + i);
37         }
38     display.display();
39     while (1);
40 }
41
42 void draw_font(int x, int y, int index)
```

kProject 열정 풍랑 프로젝트

```
43  {
44      f.seek(index * 32, SeekSet);
45      for (int i = 0; i < 16; i++)
46      {
47          val = f.read();
48
49          for (int k = 0; k < 8; k++)
50          {
51              if (((val >> (7 - k)) & 0b00000001) != 0)
52              {
53                  display.drawPixel(x + k, y + i, WHITE);
54              }
55              else
56              {
57                  display.drawPixel(x + k, y + i, BLACK);
58              }
59          }
60          val = f.read();
61
62          for (int k = 0; k < 8; k++)
63          {
64              if (((val >> (7 - k)) & 0b00000001) != 0)
65              {
66                  display.drawPixel(x + k + 8, y + i, WHITE);
67              }
68              else
69              {
70                  display.drawPixel(x + k + 8, y + i, BLACK);
71              }
72          }
73      }
74  }
```



■ 한글 코드에서 초성 중성 종성 분리하기

이제는 한글 글자를 출력할 차례입니다.

한글을 출력하기 위해서는 먼저 초성과 중성 종성을 구분하여야 합니다.

한글코드의 경우 한자는 16비트로 이루어지며 첫번째 비트의 경우 1일 경우 한글 0일경우 영문으로 판단을 합니다.

다음 2~6까지의 비트가 초성 비트이며 7~11까지의 비트가 중성, 12~16비트가 종성을 나타냅니다.

그럼 한글 코드 2바이트로 부터 초성 중성 종성을 분리해 내는 코드는 다음과 같습니다.

■ 벌수 구하기

앞서 언급한 바와 같이 초성, 중성, 종성의 조합에 따라 각각의 초성 벌수, 중성 벌수, 종성 벌수가 결정이 됩니다.

이를 배열로 저장하고 초성, 중성 종성에 따른 벌수를 구하는 루틴은 다음과 같습니다.

■ 벌수에 따른 폰트 문자 위치 찾기

지금까지 한글코드 2Byte를 초성 중성 종성으로 나누고 각 초성, 중성, 종성에 따라 초성별수, 중성별수, 종성별수를 찾았습니다.

이제 이를 이용하여 폰트의 위치를 찾아서 출력을 하여야 합니다.

한글폰트에 초성의 글자 개수는 총 20자 x 8벌 = 160자 입니다.

중성의 글자 개수는 22자 x 4벌 = 88자 입니다.

종성의 글자 개수는 28자 x 4벌 = 112자입니다

총 $160 + 88 + 112 = 360$ 자이며 Byte로는 $360 \times 32\text{Byte} = 11520\text{Byte}$ 입니다.

이제 초성의 시작 위치는 0이며

중성의 시작위치는 160입니다.

종성의 시작위치는 248입니다.

이를 기준으로 FONT에서의 초,중,종성의 위치를 구하는 코드는 다음과 같습니다.

■ 글자 출력

글자의 출력은 앞서구한 초성 중성 종성의 폰트위치의 데이터를 출력해 주면 됩니다.

■ 한글 코드 문제

앞서 작성한 모든 코드는 조합형 한글을 기준으로 하고 있습니다.

한글 코드만 나오면 지긋지긋한 변화 싸움이 시작이 되는데요...

현재까지의 소스코드는 조합형을 사용하여야 하지만 아두이노 IDE에서 사용하는 한글은 조합형 코드가 아닙니다. (추후 확인해 보겠지만 UTF-8인것으로 보입니다.)

따라서 글자를 출력하기 위해서는 글짜를 조합하여 조합형 한글을 만들어야 합니다.

조금이나마 조합형 한글을 편리하게 조합하기 위하여 엑셀로 아래와 같이 조합할 수 있도록 만들었습니다.

kProject 엘정통령 프로젝트

현재 코드에서 한글 출력을 테스트하려면 윗 엑셀의 노란셀에 초성 중성 종성을 입력하고 아래 생성된 코드를 아래와 같이 소스코드에 입력하면 한글을 출력할 수 있습니다.

kProject 열정 풍랑 프로젝트

```
19     print_hangul(x, y, 0xD0, 0x60); //ㅎㅏ
20     x = x + 16;
21     print_hangul(x, y, 0xAD, 0x40); //ㅅㅓ
22     x = x + 16;
23     print_hangul(x, y, 0xB6, 0x60); //ㅇㅕ
24     x = x + 16;
25
26     display.display();
27
28     delay(1000);
29     f.close();
30     ind++;
31     if (ind > 9) ind = 0;
32 }
```

회로도 및 소스코드 : 첨부파일 참조

■ 아두이노 IDE의 한글을 사용하기

현재까지 생성된 코드는 모두 조합형 한글을 기준으로 작성하였습니다.
테스트 해본 것과 같이 현재까지의 결과는 정상적인 한글 출력이 가능하였습니다.
하지만 일일히 한글 글자에 대한 조합형 코드를 입력해야 하므로 매우 매우 불편합니다.
그러므로 편의를 위하여 아두이노 IDE에서 한글문자열을 입력하면 그걸 바로 디스플레이 장치에 출력할 수 있도록 하여야 합니다.

이제 지긋지긋한 한글 코드 변환과의 싸움이 시작되지 않을까 싶습니다...
(조합형, 완성형, UTF-8, UTF-16, 유니코드... 등등...)
우선 검색을 해보니 아두이노 IDE에서는 한글을 UTF-8을 사용을 하고 있습니다.

UTF-8의 경우 한글 한글자를 3BYTE로 할당하고 있습니다.
(UTF-8의 경우 영문은 1BYTE, 한글은 3BYTE를 사용합니다.)

우선 인터넷을 검색~~

기준에 UTF-8(아두이노 IDE)를 조합형으로 변환하는 루틴을 만들어 공개한 자료가 있는지 검색을 해 봅니다.

기존자료들이 많이 있는데 UTF-8을 조합형으로 직접 바꾸는 루틴은 찾질 못했습니다.

하지만 다행이도 UNI CODE를 초성, 중성, 종성으로 변환하는 루틴과 UTF-8을 UNI CODE로 변환하는 방법을 찾을 수 있었습니다.

그럼 직접 변환은 안되더라도 UTF-8을 UNI CODE로 변환하고 이를 초성 중성 종성으로 분리하여 출력하면 되것 같습니다.

우선 UTF-8을 UNI 코드로 변화하는 루틴은 다음과 같습니다.

utf-8의 3byte byte1, byte2, byte3의 1110xxxx, 10xxxxxx, 10xxxxxx의 부분만 가져오면 유니코드라고 합니다.

따라서 유니코드로 변환하는 코드는 아래와 같습니다.

111);

이제 유니코드에서 초성 중성 종성을 분리해 냅니다.

유니코드의 한글은 조합형이라고 합니다.

그래서 한글 유니코드 = (((초성 x 21)+ 중성)^{*} 28) + 종성 + 0xAC00
으로 구성이 된다고 합니다.

이를 역으로 분리해내면

초성 중성 종성의 값을 얻는 방법은

유니코드에서 0xAC00을 뺀다.

구학값을 28로 나누 나머지가 축성이며

28나눈값을 다시 21로 나눈 나머지가 중성. 나눈 값이 초성이랍니다.

이를 코드로 표현하면 다음과 같습니다

이렇게 각각 구해진 초성, 중성, 종성의 값을 조합형과 일치하도록 변환테이블로 변환을 하면 최종 조합형 한글코드의 초성, 중성, 종성 값을 구할 수 있습니다.

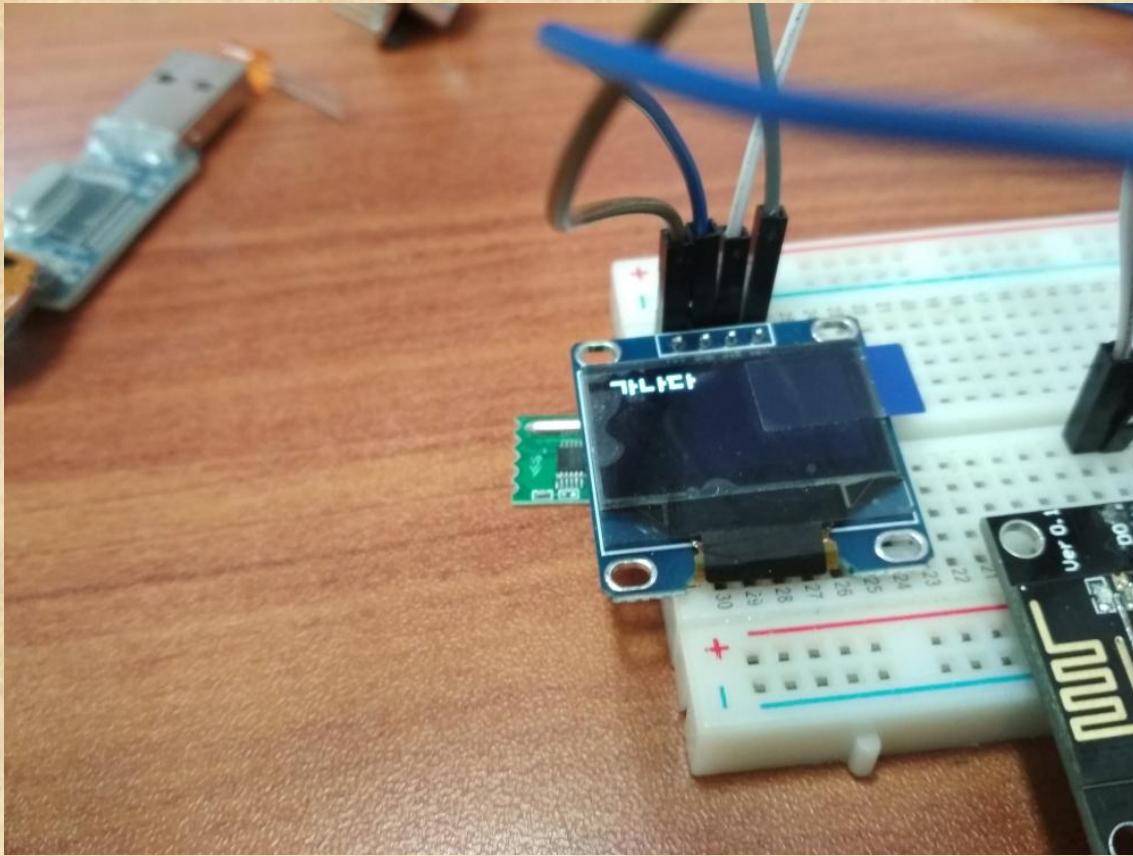
이렇게 만들어진 UTF-8용 한글 출력 함수는 다음과 같습니다.

kProject 열정풀왕 프로젝트

```
9     void print_hangul_utf8(int x, int y, unsigned char byte1, unsigned char byte2, unsigned char by
10    te3)
11    {
12        unsigned int utf16 = (byte1 & 0b00001111) << 12 | (byte2 & 0b00111111) << 6 | (byte3 & 0b00
13        111111);
14        unsigned int val = utf16 - 0xac00;
15
16        unsigned char jong = val % 28;
17        unsigned char jung = (val % (28 * 21)) / 28;
18        unsigned char cho = val / (28 * 21);
19
20        cho = _cho_ridx[cho];
21        jung = _jung_ridx[jung];
22        jong = _jong_ridx[jong];
23
24        print_hangul_cho_jung_jong(x, y, cho, jung, jong);
25    }
26
```

이제 아두이노 IDE에서 사용하는 UTF-8용 출력 함수가 만들어 졌으므로 아래와 같이 조금은 편하게 사용을 할 수가 있습니다.

```
1234 123456789012345678901234567890123456789012345678901234567890123456789012345678901234
5 567890
1 char str[] = "가나다";
2
3 void loop()
4 {
5     print_hangul_utf8(0, 0, str[0], str[1], str[2]);
6     print_hangul_utf8(16, 0, str[3], str[4], str[5]);
7     print_hangul_utf8(32, 0, str[6], str[7], str[8]);
8     display.display();
9     while (1);
10
11 }
```

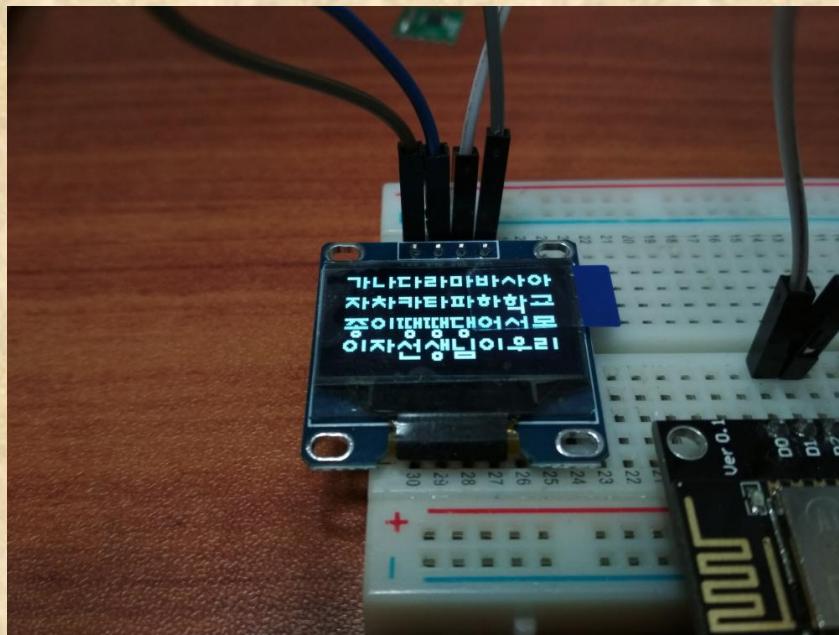


소스코드 다운로드 : 첨부파일 참조(korean_display_7)

■ 한글 문자열 출력하기

이제 아두이노를 이용하여 한글 문자열을 출력해 보겠습니다.

아직 영문출력에 대한 처리를 하지 않았기 때문에 영문이나 기호가 섞인 문자열을 출력할 수는 없지만 한글로만 이루어진 문자열을 출력해 봅니다.
소스는 아래와 같이 수정하였습니다.



소스코드 : korean_display_8

■ 영문 출력하기

이제 영문 및 기호에 대한 출력력을 할 차례입니다.

영문폰트 역시 비트맵폰트를 이지뷰어에서 가져왔습니다.

영문폰트의 사이즈는 4096 Byte입니다. 영문 한자는 16Byte를 차지하므로 총 $4096/16=256$, 총 256 ASCII 코드에 대한 폰트가 정의되어 있는 것으로 추정됩니다.

영문폰트를 샘플로 8개를 복사하여 ESP8266의 SPIFF영역에 업로드 합니다.

언로드 방법은 한글 폰트 언로드 방법과 동일합니다.

이름	수정한 날짜	유형	크기
E1.eng	1994-07-03 오후 ...	ENG 파일	4KB
E2.eng	1994-07-03 오후 ...	ENG 파일	4KB
E3.eng	1994-07-03 오후 ...	ENG 파일	4KB
E4.eng	1994-07-03 오후 ...	ENG 파일	4KB
E5.eng	1994-07-03 오후 ...	ENG 파일	4KB
E6.eng	1994-07-03 오후 ...	ENG 파일	4KB
E8.eng	1994-07-03 오후 ...	ENG 파일	4KB
H01.han	1994-07-03 오후 ...	HAN 파일	12KB
H02.han	1994-07-03 오후 ...	HAN 파일	12KB
H03.han	1994-07-03 오후 ...	HAN 파일	12KB
H04.han	1994-07-03 오후 ...	HAN 파일	12KB
H05.han	1994-07-03 오후 ...	HAN 파일	12KB
H06.han	1994-07-03 오후 ...	HAN 파일	12KB
H07.han	1994-07-03 오후 ...	HAN 파일	12KB
H08.han	1994-07-03 오후 ...	HAN 파일	12KB
H09.han	1994-07-03 오후 ...	HAN 파일	12KB
H10.han	1994-07-03 오후 ...	HAN 파일	12KB
H11.han	1994-07-03 오후 ...	HAN 파일	12KB
H12.han	1994-07-03 오후 ...	HAN 파일	12KB
H13.han	1994-07-03 오후 ...	HAN 파일	12KB

영문폰트를 업로딩하고 영문을 출력하기 위한 함수는 그대로 ASCII 코드로 폰트의 글자위치를 찾아 출력해 줍니다

폰트파일에서 원하는 글자의 위치는 한글자가 16byte를 차지하므로 코드값 x 16입니다

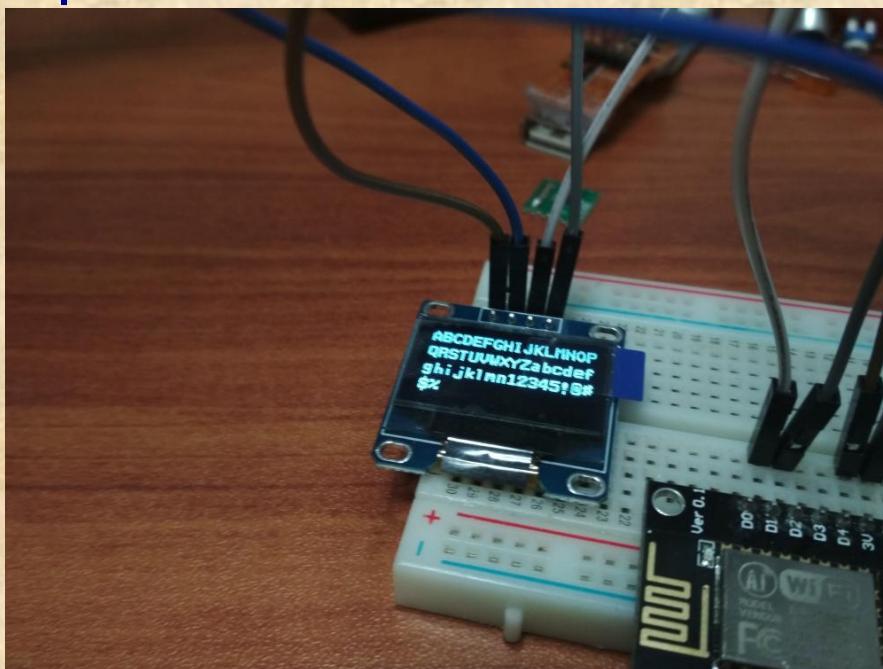
영문출력 학수는 다음과 같습니다

kProject 열정동창 프로젝트

```
10    for ( int k = 0; k < 8; k++ )
11    {
12        if ( ( ( val >> (7 - k) ) & 0b00000001 ) != 0 )
13        {
14            display.drawPixel(x + k, y + i, WHITE);
15        }
16    }
17 }
18 }
```

영문을 출력하기 위한 코드는 아래와 같습니다.

```
1 char str[] = "ABCDEFGHIJKLMNPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz j k l mn12345!@#$%";  
2  
3 void loop()  
4 {  
5     display.clearDisplay();  
6     int len = strlen(str);  
7     for( int i = 0; i < len; i+=1)  
8     {  
9         int pos_x = i;  
10        int pos_y = pos_x / 16;  
11        pos_x = pos_x % 16;  
12        draw_font_eng(pos_x*8, pos_y*16, str[i]);  
13    }  
14    display.display();  
15    delay(3000);  
16 }
```



소스코드 : korean_display_9

■ UTF-8의 문자가 차지하는 BYTE 알아내기

이제 한글과 영문이 조합되어 있는 문자열의 출력력을 할 차례입니다.

왜 이렇게 컴퓨터의 문자 출력력이 복잡해졌는지 모르겠지만 참 하기 싫은 영역입니다.

하지만 안 할 수 없는 부분이기에 우선 아두이노 IDE에서 사용하고 있는 UTF-8의 구조에 대해서 검색을 해 봅니다.

아래 링크의 사이트에 잘 설명이 되어 있네요..

<https://namu.wiki/w/UTF-8>

UTF-8은 한 문자를 출력하기 위하여 가변 길이를 사용을 하고 이에 따라 앞에서부터 차근차근 출력해 나가야 할 듯 합니다.

영문은 1BYTE를 사용하고 한글은 3BYTE를 사용합니다.

하지만 UTF-8은 1BYTE부터 6BYTE까지 한문자에 할당이 될수 있으며, 한글과 영문만 사용한다고 해서 무시해도되는건 아닙니다.

왜냐하면 한글-영문-한글-영문 이라고 되어 있는 문자를 예로들면(" A가B나" 와 같이...) A를 읽고 이게 영문인지 한글인지 판독 영문이므로 1BYTE를 이동 다시 값을 읽으면 가라는 문자가 나오므로 한글임을 판독 이후 1BYTE를 이동하는게 아닌 3BYTE를 이동해야 합니다.

하지만 문자열에 한글과 영문만 있다는 보장이 없으므로 값을 체크하여 몇바이트를 이동해야 하는지 찾아야 하므로 한글, 영문만 구분할 것이 아닌 1~6BYTE까지 몇 바이트를 차지하는 문자인지를 판독해야 합니다.

예전 조합형 시절 1BYTE는 영문 2BYTE는 한글 이게 편했는데.... 복잡해 졌네요..

문자열의 문자를 구분하는 루틴을 고민해 봅니다.

현재 문자열 포인터의 값을 가져옵니다.

그 값의 첫비트가 0 (0xxxxxxxx)이면 1Byte 문자입니다.

그 값의 앞 3개비트가 110(110xxxxx)이면 2Byte 문자입니다.

그 값의 앞 4개비트가 1110(1110xxxx)이면 3Byte 문자입니다.

그 값의 앞 5개비트가 11110(11110xxx)이면 4Byte 문자입니다.

그 값의 앞 6개비트가 111110(111110xx)이면 5Byte 문자입니다.

그 값의 앞 6개비트가 1111110(1111110x)이면 6Byte 문자입니다.

kProject 엘정통령 프로젝트

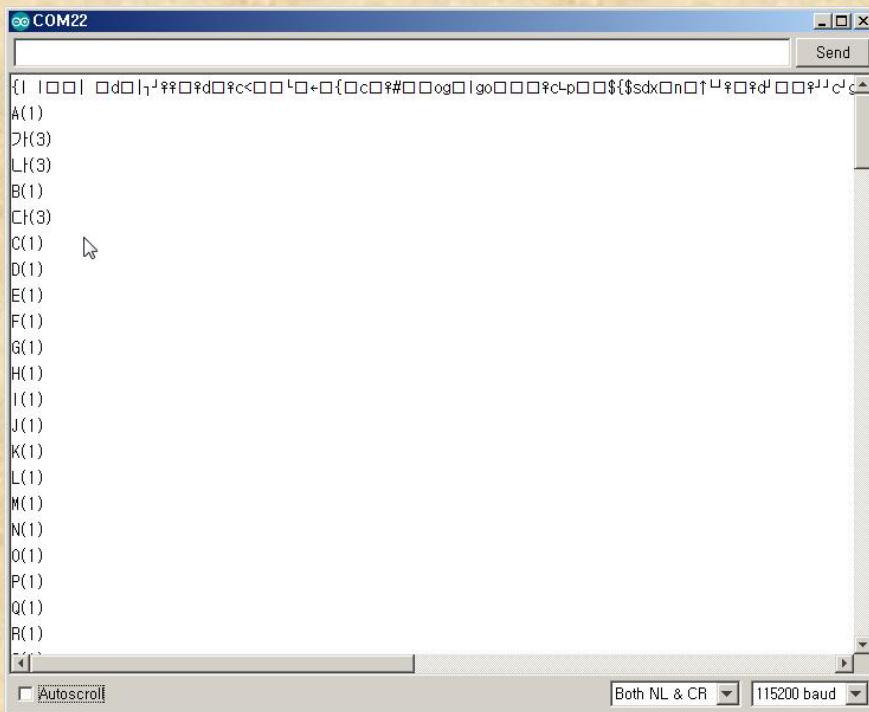
이렇게 적어놓고 보니 그리 복잡하진 않군요.

이제 소스로 정리하면 아래와 같습니다.

구동프로그램을 아래와 같이 작성하면 각 문자의 길이를 시리얼 모니터로 확인하실 수 있습니다.

kProject 엘정통령 프로젝트

18 }
19



소스코드 : korean_display_10

■ UTF-8 문자열 출력하기

이제 한글과 영문이 구분이 가능하므로 한글과 영문이 섞여 있는 문자열을 출력하도록 하겠습니다.

사용 가능한 문자가 한글과 영문만이므로 1Byte 문자는 영문으로 간주하고 3Byte 문자는 한글로 간주하여 출력하도록 합니다.

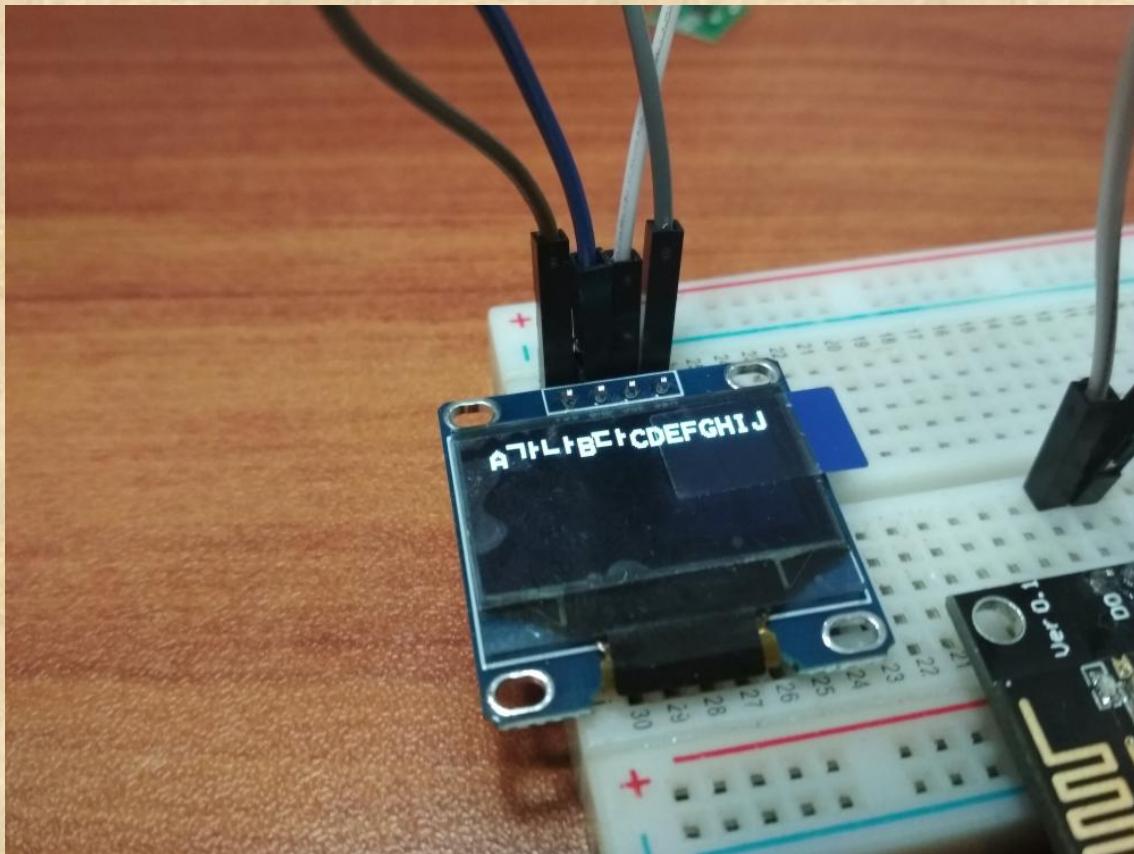
문자열을 출력하기 위한 print_string 함수를 작성합니다.

kProject 엘정통령 프로젝트

```
13     }
14     else
15     if ( bytes == 3 )
16     {
17         print_hangul_utf8(pos_x,y,str[ind],str[ind+1],str[ind+2]);
18         pos_x+=16;
19     }
20     ind += bytes;
21 }
22 di_spl ay.di_spl ay();
23 }
```

그리고 다음과 같이 구동합니다.

```
1 char str[] = "A 가나 B 다 CDEFGHI JKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz j kl mn12345!@#$%";  
2  
3 void loop()  
4 {  
5     display.clearDisplay();  
6     print_string(0, 0, str);  
7     delay(1000);  
8 }  
9
```



한글 영문 복합문자열도 잘 출력이 됩니다.

소스코드 : korean_display_11

■ FONT의 변경

이번에는 폰트변경을 해 보도록 하겠습니다.

제가 한글을 지원하는 아두이노용 디스플레이 장치를 직접 본적이 없습니다
만

아마도 폰트를 그리 많이 지원하지는 않을 것으로 보입니다.

디스플레이장치가 SD카드나 외부 저장장치를 가지고 있지 않다면 폰트파일
을 EEPROM에 저장하기에는 용량의 한계가 있을 것으로 보입니다.

한글 폰트가 11520Byte이며 영문 폰트가 4096Byte이므로 여러개의 폰트를
지원하게 되면 그 만큼 메모리 장치 부족의 압박이...

하지만 지금 사용하고 있는 ESP8266은 SPIFFS를 지원하여 SD카드처럼 메모
리를 내부에 가지고 있습니다. 그 사용할 수 있는 용량이 약 3MByte이므로
폰트를 저장하기에 충분한 용량입니다.

그럼 앞서 말한것처럼 비트맵 한글 및 영문 폰트를 이지뷰어라는 프로그램
에서 가져와 보겠습니다.

이지뷰어에 있는 한글과 영문 폰트를 모두 복사하니 한글 폰트가 142개 영
문 폰트가 74개입니다.

생각보다 많은 양의 폰트를 가지고 있네요..

일정량 이상의 많은 양은 의미가 없지만 어차피 가지고 있는 폰트이므로 모
두 정리하여 SPIFFS에 올리도록 하겠습니다.

kProject 엘정통령 프로젝트

이름	수정한 날짜	유형
131.han	1994-07-03 오후 ...	HAN 파일
132.han	1994-07-03 오후 ...	HAN 파일
133.han	1994-07-03 오후 ...	HAN 파일
134.han	1994-07-03 오후 ...	HAN 파일
135.han	1994-07-03 오후 ...	HAN 파일
136.han	1994-07-03 오후 ...	HAN 파일
137.han	1994-07-03 오후 ...	HAN 파일
138.han	1994-07-03 오후 ...	HAN 파일
139.han	1994-07-03 오후 ...	HAN 파일
140.han	1994-07-03 오후 ...	HAN 파일
141.han	1994-07-03 오후 ...	HAN 파일
142.han	1994-07-03 오후 ...	HAN 파일
001.eng	1994-07-03 오후 ...	ENG 파일
002.eng	1994-07-03 오후 ...	ENG 파일
003.eng	1994-07-03 오후 ...	ENG 파일
004.eng	1994-07-03 오후 ...	ENG 파일
005.eng	1994-07-03 오후 ...	ENG 파일
006.eng	1994-07-03 오후 ...	ENG 파일
007.eng	1994-07-03 오후 ...	ENG 파일
...	1994-07-03 오후 8:51	ENG 파일

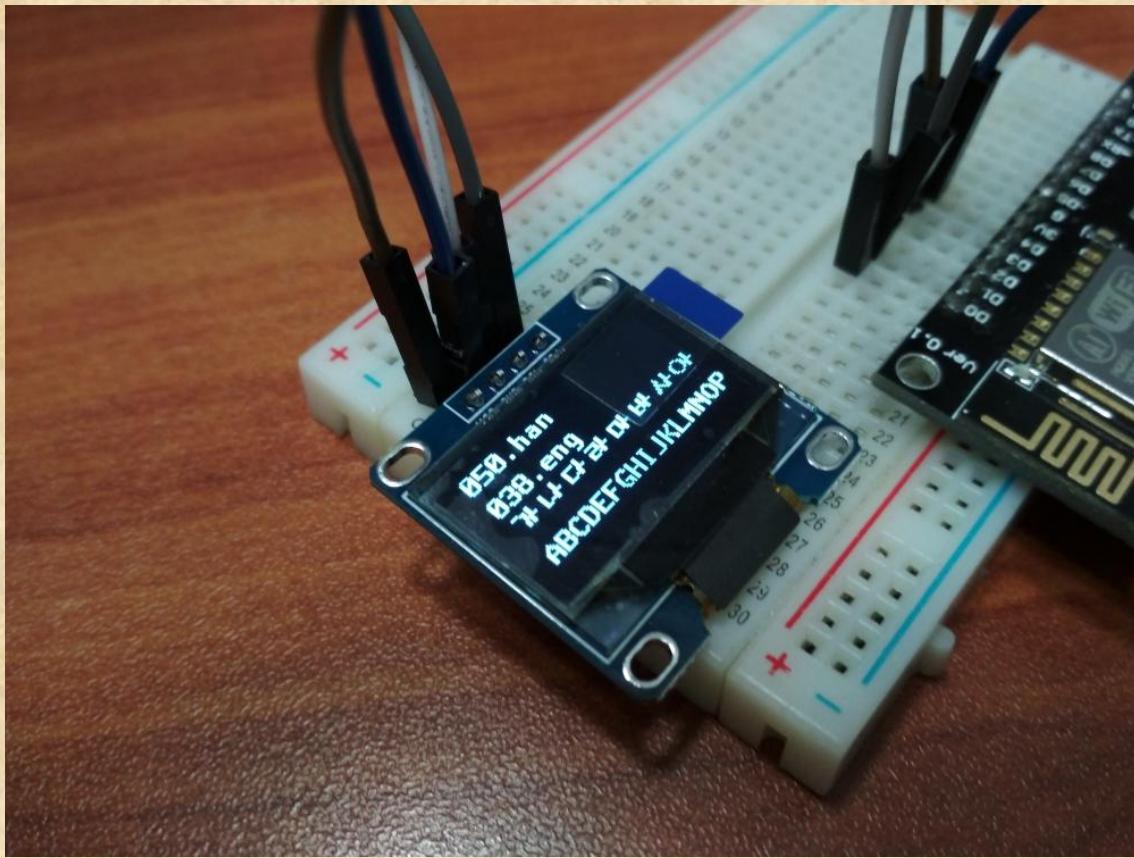
파일의 이름은 윗 그림과 같이
한글 폰트는 001.han ~ 142.han으로 하였으며
영문 폰트는 001.eng ~ 074.eng로 하였습니다.

그리고 폰트를 설정할 수 있는 함수를 제작하였습니다.

그리고 구동을 합니다.
구동프로그램은 모든 한글 및 영문 폰트를 변경해 가면서 글자를 출력하도록 하였습니다.

kProject 열정동창 프로젝트

구동결과를 보면 다양한 폰트로 출력이 되는걸 확인하실 수 있습니다.



소스코드 다운로드 : korean_display_12 (폰트파일은 저작권에 문제가 될 수 있어 포함하지 않았습니다. 직접 이지뷰어 프로그램에서 추출하셔도 되고 필요하시면 khkim@mapescn.com으로 요청하시면 메일로 보내드리도록 하겠습니다.)

■ 콘트롤러로 사용하기 위한 준비

지금까지 나름 많은 내용을 진행하여 왔지만 실제로 이 프로젝트의 목적은 디스플레이에 한글을 사용하기 위함이 아닌 한글을 지원하는 디스플레이 콘트롤러의 제작입니다.

즉, 아두이노에서 디스플레이에 직접 한글을 구동하기위한 소스코드를 입력하는게 아닌 아두이노에서 콘트롤러에 시리얼 통신으로 명령을 주면 콘트롤러가 알아서 출력을 설정해 주는 콘트롤러를 제작하고자 함 입니다.

그러기 위해서는 콘트롤러에 송신하기 위한 서로간의 규약이 필요하게 됩니다.

시리얼 통신을 이용하여 아두이노에서 콘트롤러에 신호를 보내면 콘트롤러는 수신된 바이트의 비트를 확인하여 첫 비트가 0일경우 영문으로 0이 아닌 경우 영문이 아닌 다른문자로(한글이나 그외의 문자) 인식하게 됩니다. (아두이노 IDE기준의 UTF-8기준입니다.)

하지만 콘트롤러는 단순 문자열만 입력을 받는게 아닌 제어 명령을 입력 받을 수 있어야 합니다.

예를 들자면 아두이노에서 콘트롤러에 x좌표와 y좌표 그리고 문자열을 송신 하여야 원하는 위치에 문자를 표시할 수 있습니다.

혹은 컬러 디스플레이 장치의 경우 색상을 지정할 수도 있어야 하며 아두이노에서 콘트롤로에 어떤 폰트로 출력 할지에 대한 명령등도 송신할 수 있어야 합니다.

하지만 전송문자는 이미 첫비트에 따라 0이면 영문 0이 아니면 다른 문자로 되어 있기 때문에 콘트롤로에서 문자가 아닌 제어 명령임을 인식할 수 있는 규약이 필요합니다.

이는 ASCII 문자중 사용하지 않는 문자를 이용하여 시리얼 통신으로 특정 ASCII 문자가 전송이 되면 그 이후의 전송데이터는 제어 변수로 인식하도록 하겠습니다.

그럼 ASCII에서 문자에 사용하지 않는 코드는 어떤게 있는지 확인해 보겠습니다.

다음이 ASCII의 코드표 입니다.

10진수	16진수	문자	10진수	16진수	문자	10진수	16진수	문자	10진수	16진수	문자
0	0x00	NULL	1	0x01	SOH	2	0x02	STX	3	0x03	ETX
4	0x04	EOT	5	0x05	ENQ	6	0x06	ACK	7	0x07	BEL
8	0x08	BS	9	0x09	HT	10	0x0A	LF	11	0x0B	VT
12	0x0C	FF	13	0x0D	CR	14	0x0E	SO	15	0x0F	SI
16	0x10	DEL	17	0x11	DCI	18	0x12	DC2	19	0x13	DC3
20	0x14	DC4	21	0x15	NAK	22	0x16	SYN	23	0x17	ETB
24	0x18	CAN	25	0x19	EM	26	0x1A	SUB	27	0x1B	ESC
28	0x1C	FS	29	0x1D	GS	30	0x1E	RS	31	0x1F	US
32	0x20	Space	33	0x21	!	34	0x22	"	35	0x23	#
36	0x24	\$	37	0x25	%	38	0x26	&	39	0x27	'
40	0x28	(41	0x29)	42	0x2A	*	43	0x2B	+
44	0x2C	,	45	0x2D	-	46	0x2E	.	47	0x2F	/
48	0x30	0	49	0x31	1	50	0x32	2	51	0x33	3
52	0x34	4	53	0x35	5	54	0x36	6	55	0x37	7
56	0x38	8	57	0x39	9	58	0x3A	:	59	0x3B	;
60	0x3C	<	61	0x3D	=	62	0x3E	>	63	0x3F	?
64	0x40	@	65	0x41	A	66	0x42	B	67	0x43	C
68	0x44	D	69	0x45	E	70	0x46	F	71	0x47	G
72	0x48	H	73	0x49	I	74	0x4A	J	75	0x4B	K
76	0x4C	L	77	0x4D	M	78	0x4E	N	79	0x4F	O
80	0x50	P	81	0x51	Q	82	0x52	R	83	0x53	S
84	0x54	T	85	0x55	U	86	0x56	V	87	0x57	W
88	0x58	X	89	0x59	Y	90	0x5A	Z	91	0x5B	[
92	0x5C	₩	93	0x5D]	94	0x5E	^	95	0x5F	-
96	0x60	'	97	0x61	a	98	0x62	b	99	0x63	c
100	0x64	d	101	0x65	e	102	0x66	f	103	0x67	g
104	0x68	h	105	0x69	i	106	0x6A	j	107	0x6B	k
108	0x6C	l	109	0x6D	m	110	0x6E	n	111	0x6F	o
112	0x70	p	113	0x71	q	114	0x72	r	115	0x73	s
116	0x74	t	117	0x75	u	118	0x76	v	119	0x77	w
120	0x78	x	121	0x79	y	122	0x7A	z	123	0x7B	{
124	0x7C		125	0x7D	}	126	0x7E	~	127	0x7F	DFL

ASCII의 127개 코드는 모두 문자를 나타내는게 아닌

데이터 전송 시작, 끝 등을 나타내는 제어 신호도 있습니다.

아스키 코드는 원래 하드웨어와 하드웨어, 하드웨어와 운영체제 사이에 주고 받는 신호들을 약속한 것이기 때문입니다.

이러한 이유로 프로그램에서 문자를 표현할 때 사용하지 않는 값들도 포함하고 있습니다.

다음은 아스키 코드의 특수 문자와 제어 신호들입니다.

◆ 아스키 코드의 특수 문자와 제어 신호

NUL : NULL

SOH : 데이터 전송 시작

STX : 본문 시작

ETX: 본문 종료

EOT: 전송 종료

ETB: 전송 블록 종료

ENQ: 응답 요구
ACK: 긍정 응답
BEL: 경고음
BS: Back Space
HT: 수평 탭
LF: 개행
VT: 수직 탭
FF: 다음 페이지
CR: Carriage Return
SO: 확장 문자 시작
SI: 확장 문자 종료
DFL: 전송 제어 확장
DC1: Device Control 1
DC2: Device Control 2
DC3: Device Control 3
DC4: Device Control 4
NAK: 부정 응답
SYN: Synchronous Idle
CAN: 취소
EM: 매체 종료
SUB: 치환
ESC: Escape
FS: 파일 경계
GS: 그룹 경계
RS: 레코드 경계
US: 장치 경계

여기서 저는 SOH(데이터 전송 시작) 코드를 제어명령 인식 코드로 사용하겠습니다.

즉, 시리얼 데이터로 전송된 데이터가 ASCII의 SOH(0x01)일 경우 이후 특정 바이트는 출력하기 위한 문자열이 아닌 제어 명령의 데이터로 사용하도록 하겠습니다.

■ 콘트롤러로 출력하기

우선은 제어 명령 이전에 기본 출력 기능을 사용해 보겠습니다.

시리얼 통신을 통해 수신된 데이터가 ASCII 코드 SOH(0x01)이 아닌 경우 문자를 디스플레이 장치에 출력을 합니다.

시리얼 통신 속도는 9600 baud rate를 사용하겠습니다.

아래 코드는 시리얼 통신을 통하여 입력받은 UTF-8코드를 디스플레이하는 소스입니다.

구동소스는 다음과 같습니다.

〈콘트롤러용 소스코드〉

kProject 엘정통령 프로젝트

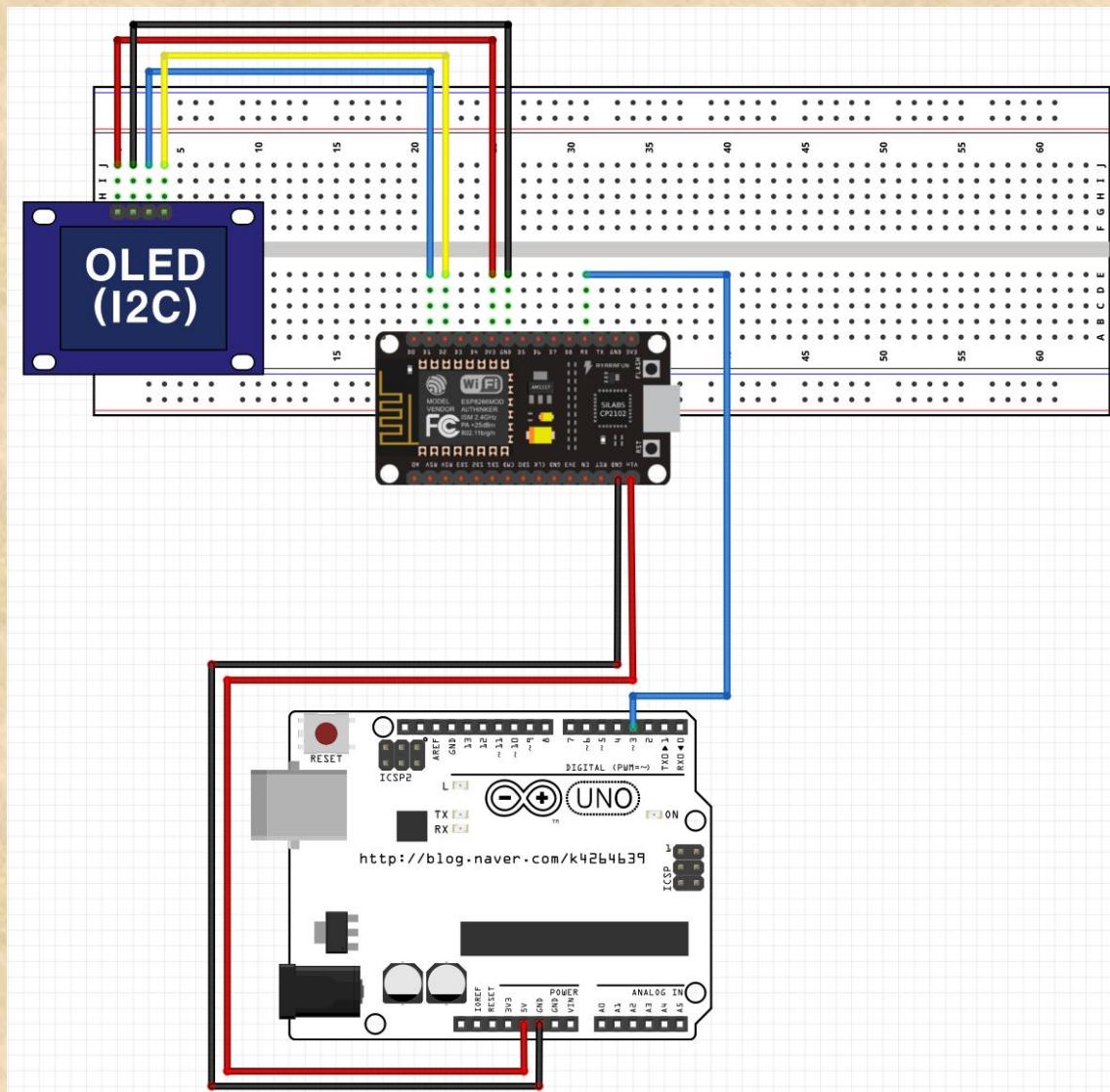
```
33 #endif f
34 }
35
36     if ( char_byte_type == 3 )
37     {
38         print_hangul_utf8(pos_x, pos_y, cha[0], cha[1], cha[2]);
39         display.display();
40         pos_x+=16;
41 #ifdef DEBUG
42         Serial.write(cha[0]);
43         Serial.write(cha[1]);
44         Serial.write(cha[2]);
45         Serial.println("");
46 #endif f
47     }
48 }
49
50 }
51 }
```

이제 아두이노 에서는 아래와 같이 소스를 작성합니다.

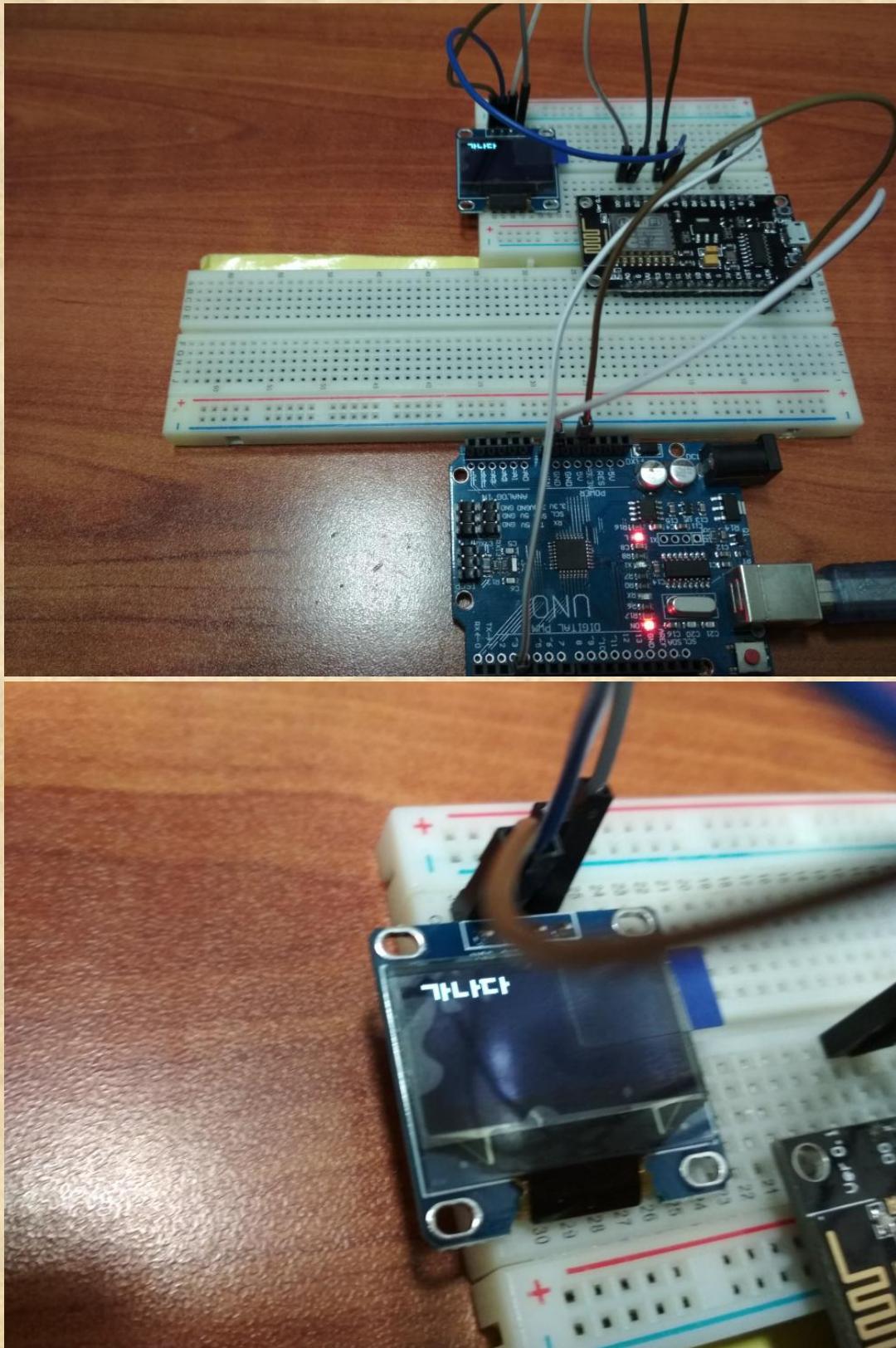
〈아두이노용 소스코드〉

회로도는 다음과 같습니다.

〈회로도〉



작동 화면



추가 사항

현재는 아두이노 우노의 시리얼 출력을 SoftwareSerial을 사용하고 있습니다.

SoftwareSerial을 사용하지 않고 0, 1번핀을 이용한 하드웨어 Serial을 사용해도 정상 작동을 합니다.

한글출력을 위한 디스플레이 콘트롤러를 사용하는 이유는 대부분 직접 아두이노 우노에서 한글 출력을 구현하여 구동하는 것보다 아두이노 우노의 리소스를 절약하여 다른 연산에 사용하기 위함일 것입니다.

하지만 하드웨어 시리얼을 사용할 경우 연결이 되어 있는 상태에서는 스케치 업로드가 되지 않는 단점이 있으며

SoftwareSerial을 사용할 경우 콘트롤러를 사용하는 이유가 리소스를 아끼기 위해서 이나 SoftwareSerial 자체가 적지 않은 리소스를 사용하기에 근본적인 해결 방법이 아니라고 생각합니다.

따라서 현재는 편의를 위해 SoftwareSerial을 사용하고 있지만 추후 자체 Serial 통신을 위한 TX루틴을 만들어 라이브러리화 할 예정입니다.

시리얼통신을 소프트웨어로 구현할 경우 많은 리소스를 차지하는 부분은 RX 부분입니다.

언제 입력될지 모르는 RX핀의 상태를 알아내기 위해서 지속적으로 RX핀을 검색해야 하기 때문입니다.

반면에 TX핀의 경우 필요할 경우에만 핀에 출력을 하기 때문에 TX만을 사용하는 SoftwareSerial 통신은 훨씬 적은 리소스를 사용하게 됩니다.

하지만 TX만 지원하는 Software Serial 라이브러리가 아두이노에 별도로 없기 때문에 추후 직접 제작할 예정입니다.

■ 디스플레이 장치 제어하기

지난 시간에 시리얼 통신을 이용하여 문자를 전송하는 부분까지 구현을 하였습니다.

이번에는 제어문을 통하여 화면을 지우고 문자를 출력할 커서(위치)를 수정하는 부분을 구현해 보도록 하겠습니다.

제어 문자는 ASCII 코드로 SOH(0x01)의 문자가 수신이 되면 이후 특정 바이트의 값을 제어 명령으로 입력받도록 하겠습니다.

화면을 지우는 명령부터 만들어 보겠습니다.

화면을 지우는 명령은 0x01의 코드가 온 후 0x00이 수신되면 화면을 지우도록 합니다.

그리고 커서의 위치를 옮기는 코드는 0x01코드가 수신된 이후 0x01이 수신되면 이후 2Byte는 x좌표, 이후 2Byte는 y좌표로 정의하겠습니다.

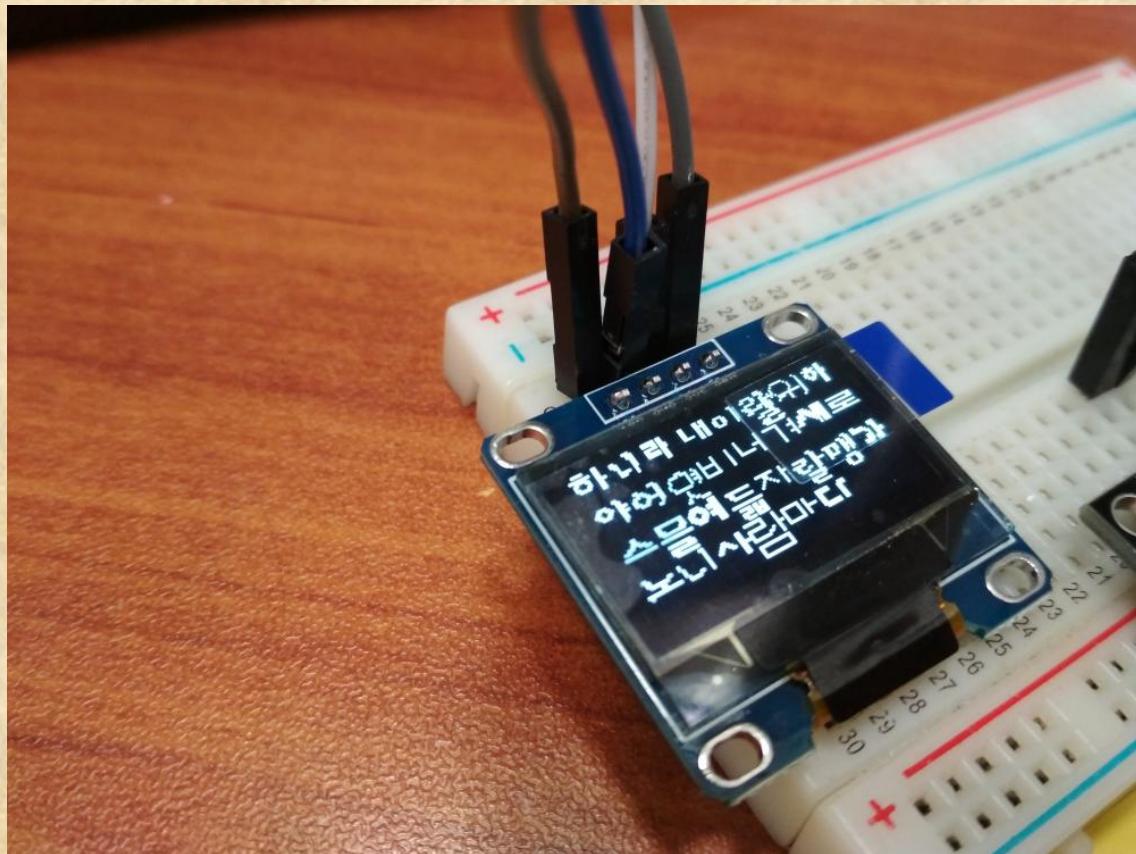
정리하면 아래 테이블과 같습니다.

제어코드	명령코드	데이터 1	데이터 2	데이터 3	데이터 4
0x01	0x01	-	-	-	-
0x01	0x02	x 좌표 상위바이트	x 좌표 하위 바이트	y 좌표 상위 바이트	y 좌표 하위 바이트
0x01	0x03	한글폰트번호 (1~142)	-	-	-
0x01	0x04	영문폰트번호 (1~74)	-	-	-

이를 구현한 아두이노 구동소스는 아래와 같습니다.

kProject 열정 풍랑 프로젝트

```
18     for ( int i = 0; i < 106; i++ )
19     {
20         di_splay_set_han_font(font_index);
21         font_index++;
22         if ( font_index > 142 ) font_index = 1;
23         ch[0] = hun[i * 3 + 0];
24         ch[1] = hun[i * 3 + 1];
25         ch[2] = hun[i * 3 + 2];
26         ch[3] = 0;
27         di_splay_set_cursor(pos_x, pos_y);
28         di_splay_print(ch);
29         pos_x += 16;
30         if ( pos_x > 112 )
31         {
32             pos_y += 16;
33             pos_x = 0;
34             if ( pos_y > 48 )
35             {
36                 pos_y = 0;
37                 di_splay_clear();
38             }
39         }
40         delay(500);
41     }
42 }
```



소스코드 다운로드 : 첨부파일 참조 - korean_display_14

■ TFT에 출력하기

아직 갈길이 멀긴 합니다만 I2C용 OLED의 제어는 여기에서 정리하고 이번엔 TFT에 출력하는 루틴을 작성하도록 하겠습니다.
I2C용 OLED는 추후 다시 진행하도록 하겠습니다.
제가 가지고 있는 TFT LCD는 다음 모델입니다.



다양한 TFT LCD가 있지만 아두이노에 가장 많이 사용하는 TFT가 아닌가 싶습니다.
크기가 1.8인치로 작기는 하지만 아두이노에서 이보다 큰 TFT에 출력은 속도면에서 무리가 있어 많이 사용하는 것 같습니다.
디스플레이의 해상도는 128x160입니다.

모듈에 나와있는 핀은 16개이나 실제로 사용되는 핀은 VCC, GND, RESET, A0, SDA, SCK, CS, LED+, LED-로 총 9개만 사용하면 TFT를 컨트롤 할 수 있습니다.
나머지 핀들은 사용하지 않거나 SD카드용이므로 화면표시에는 필요가 없습니다.

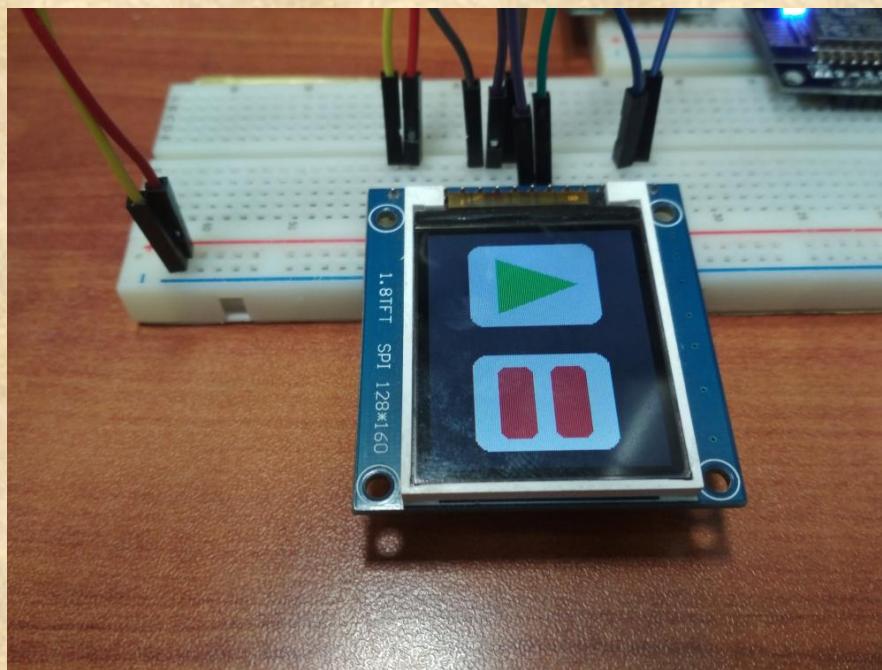
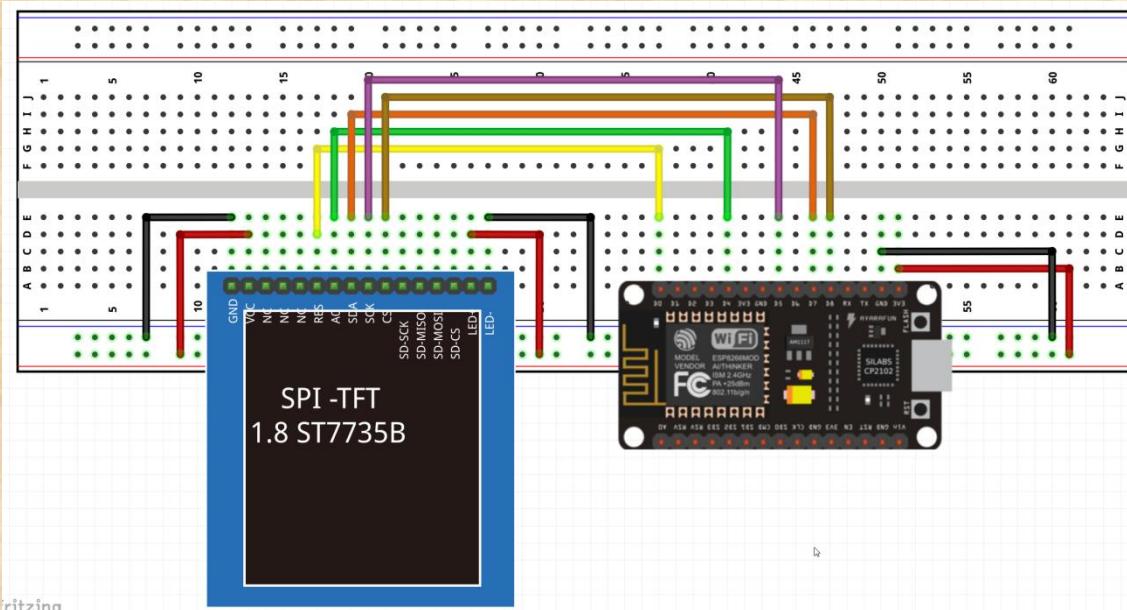
이 모듈은 SPI통신을 하며, 콘트롤 IC는 ST7735B입니다.
다행스럽게도 이 모듈역시 라이브러리가 공개되어 있습니다.

우선 이 모듈을 ESP8266을 이용하여 구동하여 보도록 하겠습니다.
아래와 같이 회로를 연결하고 첨부 소스를 업로딩 합니다.
TFT의 RESET 핀은 ESP8266의 GPIO 16번에 연결하고 A0는 GPIO 2, CS는 GPIO 15에 연결하였습니다.

kProject 열정 풍랑 프로젝트

나머지 핀은 ESP8266의 SPI 핀을 사용하였으며 SCK는 GPIO 14번, MISO는 GPIO 12번(사용 안함), MOSI는 GPIO 13번이며 하드웨어 SPI는 핀이 고정이므로 소스코드에서 별도로 정의하지는 않습니다.
또한 MISO는 TFT 구동시 사용이 되지 않습니다.

회로도



소스코드 : 첨부파일 참조 - korean_display_15

■ TFT에 PIXEL 출력하기

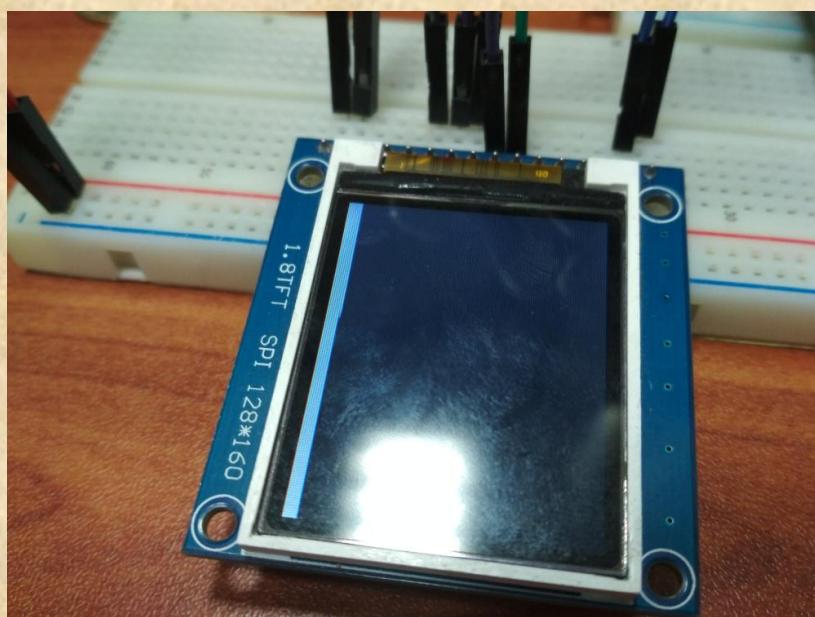
I2C용 OLED 사용과 마찬가지로 x, y좌표에 점을 찍는 루틴을 작성합니다.

```
1234 123456789012345678901234567890123456789012345678901234567890123456789012345678901234  
5 567890  
  
1 void tft_draw_pixel (int x, int y, uint16_t color)  
2 {  
3     tft.drawPixel (x, y, color);  
4 }
```

구동은 I2C용 OLED루틴과 마찬기자로 점을 찍어나가는 코드로 테스트 해 보았습니다.

```
1234 123456789012345678901234567890123456789012345678901234567890123456789012345678901234  
5 567890  
  
1 void loop()  
2 {  
3     for ( int i = 0; i < 128; i++)  
4         for ( int j = 0; j < 160; j++)  
5         {  
6             tft_draw_pixel (i, j, ST7735_WHITE);  
7             delay(10);  
8         }  
9     }  
10 }
```

소스코드 : 첨부파일 참조 - korean_display_16



■ TFT에 한글 출력하기

TFT용 PIXEL 출력 루틴이 만들어졌으므로 이제 나머지는 I2C와 동일하게 작성하면 되며 화면을 지우는 함수와 PIXEL을 찍는 함수만 변경을 하면 됩니다.

이제는 지금 만드는 콘트롤러가 I2C OLED만이 아닌 SPI TFT도 지원하므로 시작시 무조건 I2C 디스플레이 장치를 초기화하면 안됩니다.

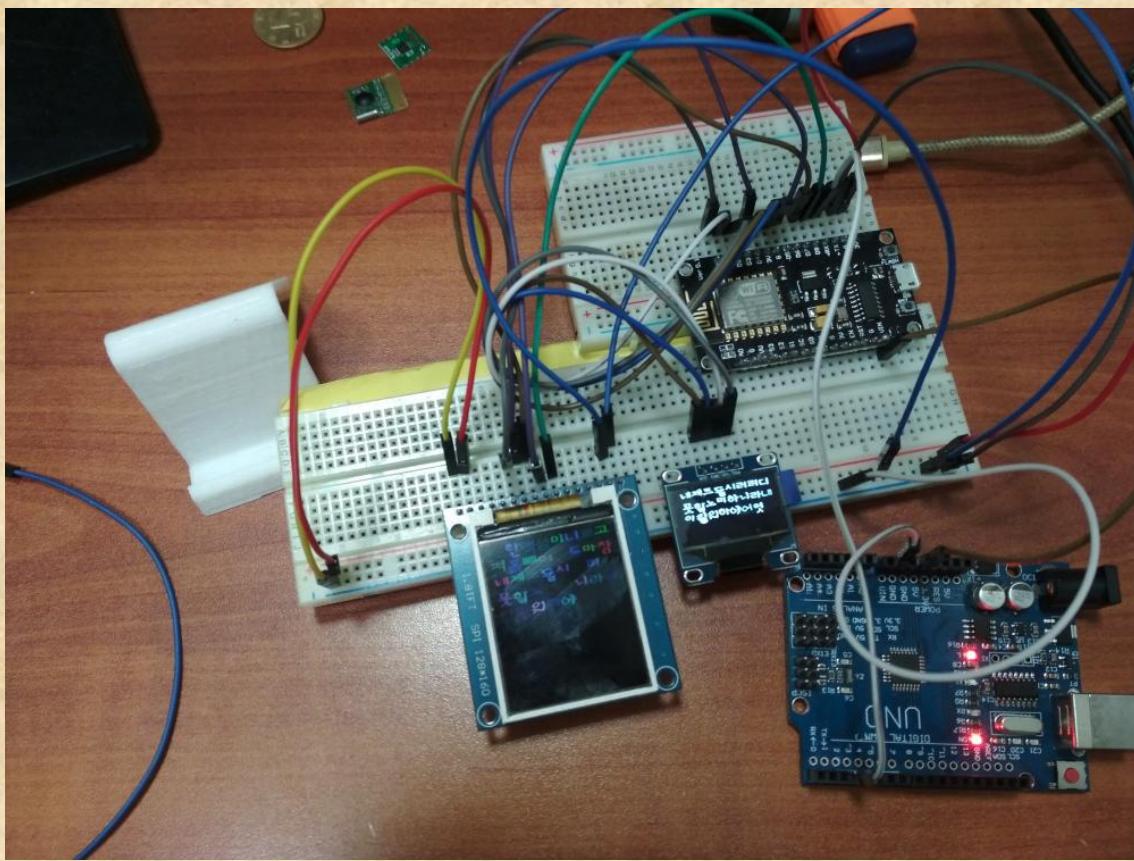
따라서 제어코드에 I2C, 혹은 SPI TFT를 초기화 할 수 있는 코드를 만듭니다.

명령코드는 0x05로 하였으며 데이터 1에 0일 경우 I2C OLED, 1 일 경우 TFT LCD로 하였습니다.

또한 I2C에 출력을 할지 TFT에 출력을 할지 활성화 시키는 명령을 0x06으로 할당하였습니다.

제어코드	명령코드	데이터 1	데이터 2	데이터 3	데이터 4
0x01	0x01 화면 초기화	-	-	-	-
0x01	0x02 Cursor 이동	x 좌표 상위바이트	x 좌표 하위 바이트	y 좌표 상위 바이트	y 좌표 하위 바이트
0x01	0x03 한글 폰트 설정	한글폰트번호 (1~142)	-	-	-
0x01	0x04 영문 폰트 설정	영문폰트번호 (1~74)	-	-	-
0x01	0x05 장치 초기화	디스플레이 장치 1 : OLED I2C 2 : TFT LCD SPI	-	-	-
0x01	0x06 장치 활성화	디스플레이 장치 0 : 활성화 되지 않음. 1 : OLED I2C 2 : TFT LCD SPI	-	-	-
0x01	0x07 색상 설정	색상 상위바이트	색상 하위바이트	-	-

활성화 장치는 ESP8266 펌웨어에서 active_display_index에 저장이 되며 이 값에 따라 출력명령을 I2C 혹은 TFT LCD에 출력하게 됩니다.



kProject 열정 풍향 프로젝트

이번 코드는 콘트롤러에 I2C용 OLED와 SPI 용 TFT를 모두 연결하여 두개를 모두 콘트롤 하는 예입니다.

구동 아두이노 UNO의 소스코드는 아래와 같습니다.

소스코드 : 첨부파일 참조 - korean_display_17

참고 : ESP8266의 코드는 TFT의 속도를 좀더 빠르게 하기 위하여 160MHz로

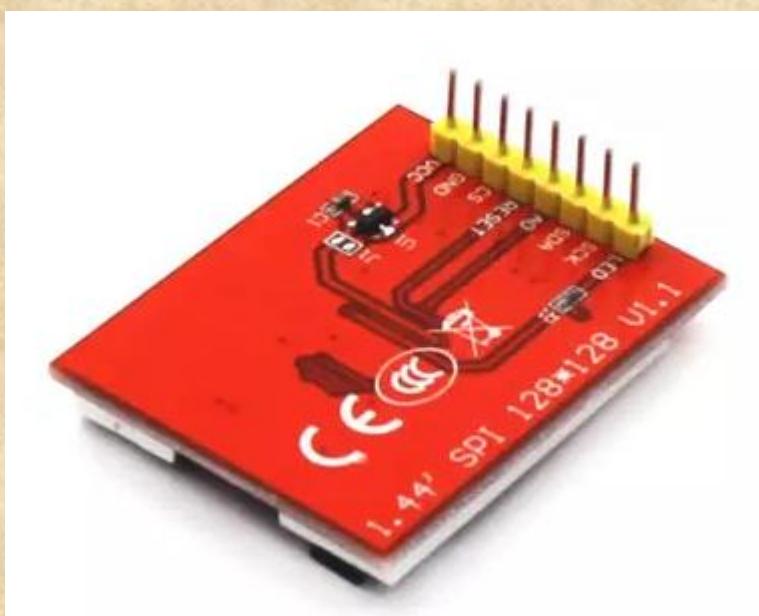
구동하도록 하였습니다.

■ TFT와 OLED에 동시 한글 출력하기

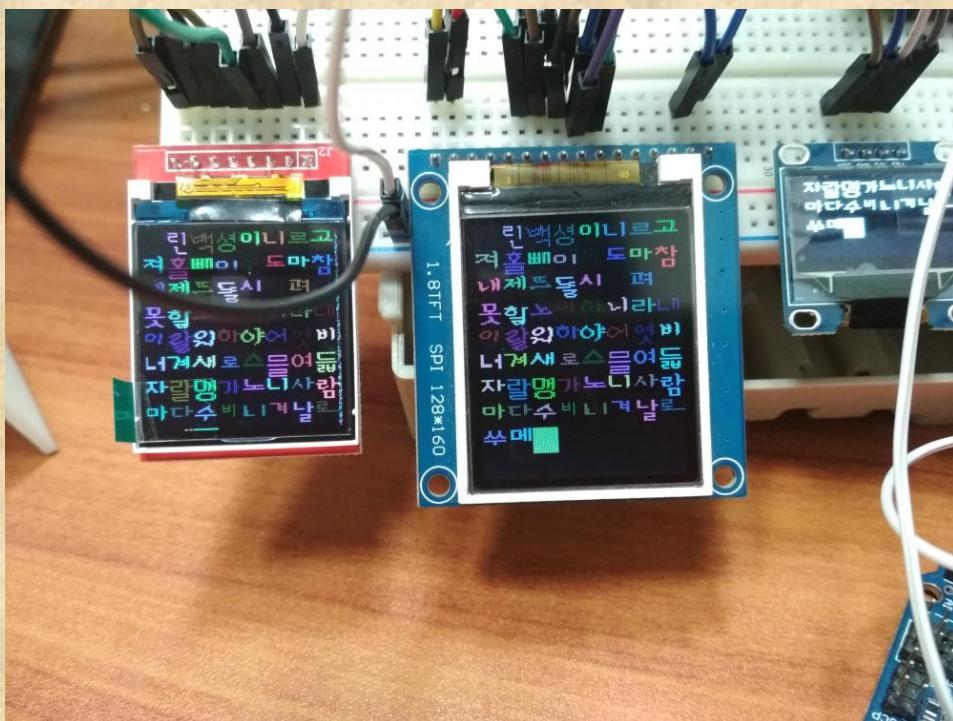
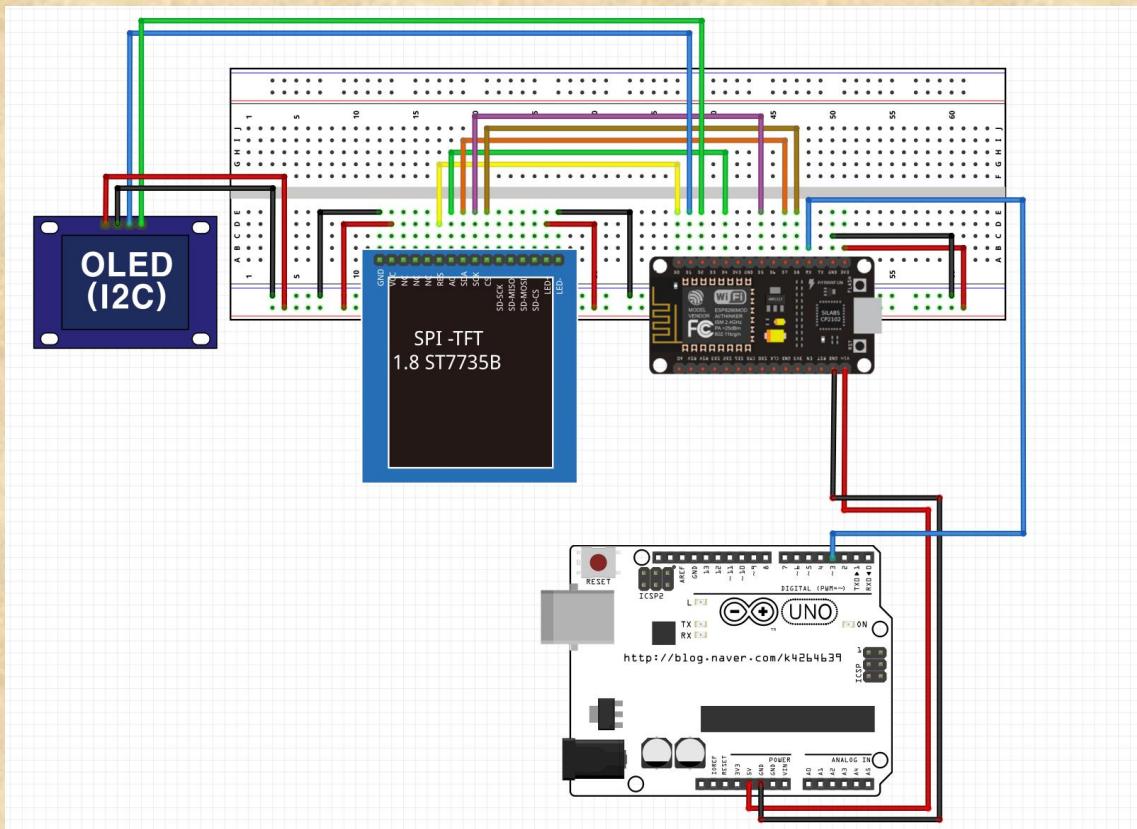
추가로 다른 TYPE의 TFT모듈이 있어 한개 더 연결하여 보았습니다.

TFT SPI의 핀들을 공유하도록 설정하였습니다.

제품은 아래 사진의 제품입니다.



kProject 열정풍향 프로젝트



소스코드 : 첨부파일 참조 - korean_display_17

■ GLCD(128x64)에 출력하기

한동안 연재를 쉬었습니다. 이유는 제가 GLCD 128x64 를 가지고 있지 않아서 주문하고 기다리는 기간이었습니다.
(제가 중국에 거주중인데 중국의 택배는 한국처럼 빠르지가 않습니다.
하지만 거리까지 감안한다면 중국이 워낙 크다보니 느리다고 불평할 만한 사항은 아니지요...)

드디어 주문한 GLCD가 도착을 하였습니다.

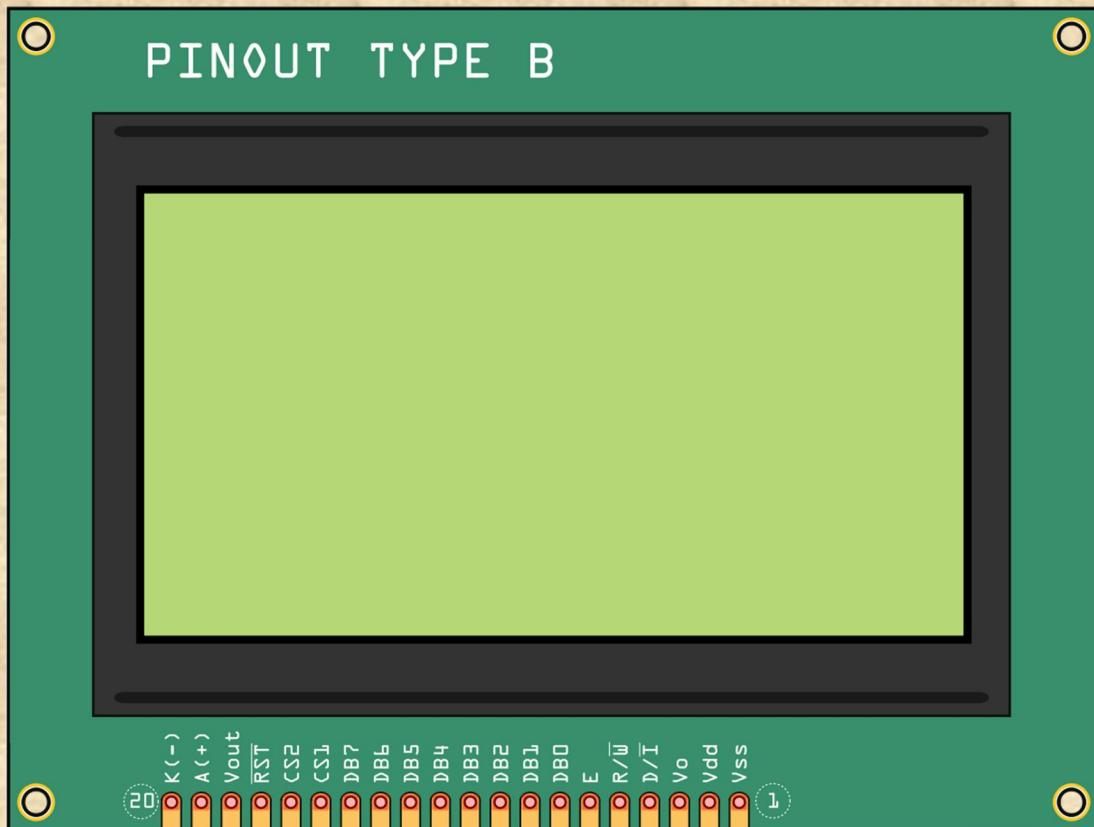
제가 주문한 GLCD는 아래 제품입니다.



가격이 다른 디스플레이 장치에 비해 조금은 비싼듯 한데...

막상 받아보니 사이즈도 생각보다는 꽤 큅니다.

제가 구매한 GLCD는 디스플레이 콘트롤러로 ST7920을 사용합니다.
사용하기 위한 핀은 총 20개 핀으로 핀맵은 FRITZING에 있는 GLCD와
동일한걸 확인했습니다.



구동은 라이브러리를 이용하겠지만 대충 데이터 시트를 훑어보니
기본 사용은 CLCD와 비슷하지만 Serial Mode로도 구동이 가능하다는 점이
특이점 인 듯 합니다.

제가 만들고 있는 한글 출력 장치는 콘트롤러를 MCU를 ESP8266을 사용하는
데 ESP8266의 단점이 출력핀이 모자라다는 점입니다.

다행이도 이 LCD는 Serial Mode를 지원하므로 좀더 적은 핀으로 구동을 할
수 있겠습니다.

만일 이 Serial Mode가 없었으면 아마도 74HC595 IC를 사용하여 출력핀 확
장을 했어야 할 듯 합니다.

Pin Description

Name	No.	I/O	Connects to	Function
XRESET	11	I	—	System reset input (low active)
PSB	23	I	—	Interface selection: 0: serial mode; 1: 8/4-bit parallel bus mode.
RS(CS*)	17	I	MPU	Parallel Mode: Register select. 0: Select instruction register (write) or busy flag, address counter (read); 1: Select data register (write/read). Serial mode: Chip select. 1: chip enabled; 0: chip disabled. When chip is disabled, SID and SCLK should be set as "H" or "L". Transient of SID and SCLK is not allowed.
RW(SID*)	18	I	MPU	Parallel Mode: Read/Write control. 0: Write; 1: Read. Serial Mode: Sserial data input.
E(SCLK*)	19	I	MPU	Parallel Mode: 1: Enable trigger. Serial Mode: Serial clock.
D4 to D7	28~31	I/O	MPU	Higher nibble data bus of 8-bit interface and data bus for 4-bit interface
D0 to D3	24~27	I/O	MPU	Lower nibble data bus of 8-bit interface.
CL1	12	O	Extension segment drv.	Latch signal for extension segment drivers.

ST7920 컨트롤러의 PSB핀이 LOW일 경우 Serial Mode이고 HIGH일 경우 Parallel bus Mode라고 합니다.

이 PSB핀은 GLCD의 CS1으로 표기되어 있는 15번 핀입니다.

Serial Mode로 구동시 사용되는 핀은 CS, SID, SCLK입니다.

여기서 CS는 Serial Mode시 chip enable을 설정하는 핀이므로 항상 HIGH로 설정하면 됩니다.

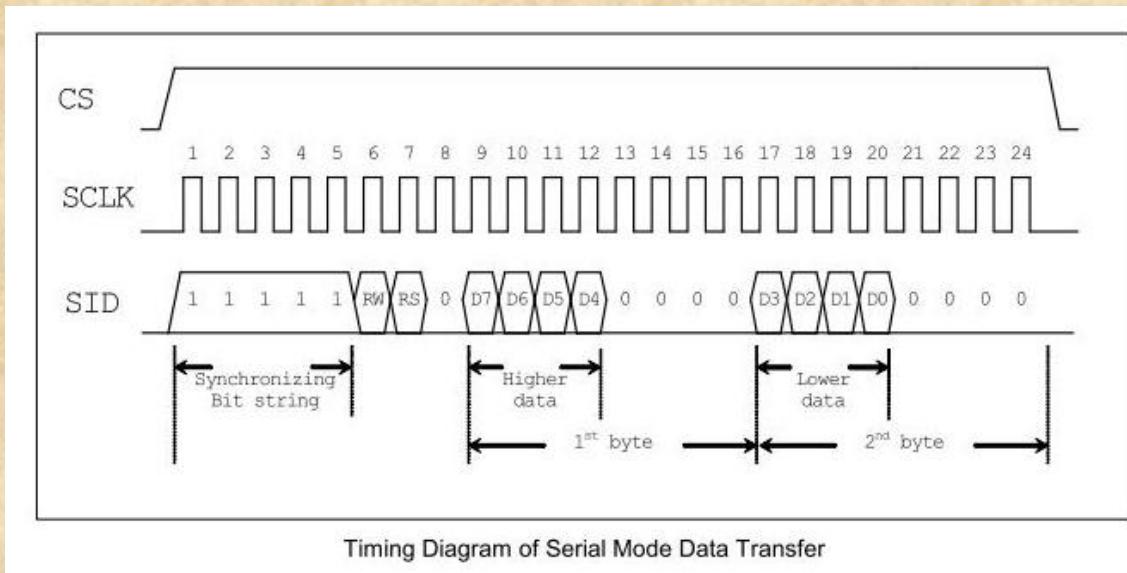
정리하면 ST7920컨트롤러를 이용한 GLCD를 Serial Mode로 구동하고 싶으면

PSB핀은 LOW로 CS핀은 HIGH로 고정하여 설정하면 되므로

PSB핀은 GND에 CS핀은 VCC에 연결을 하고

SID와 SCLK핀만을 이용하여 통신을 하면 됩니다.

통신규약은 데이터 시트에 아래와 같이 설명이 되어 있지만



여기서는 GLCD 의 구동이 목적이 아니라 한글 출력이므로
GLCD의 구동은 라이브러리를 사용하겠습니다.

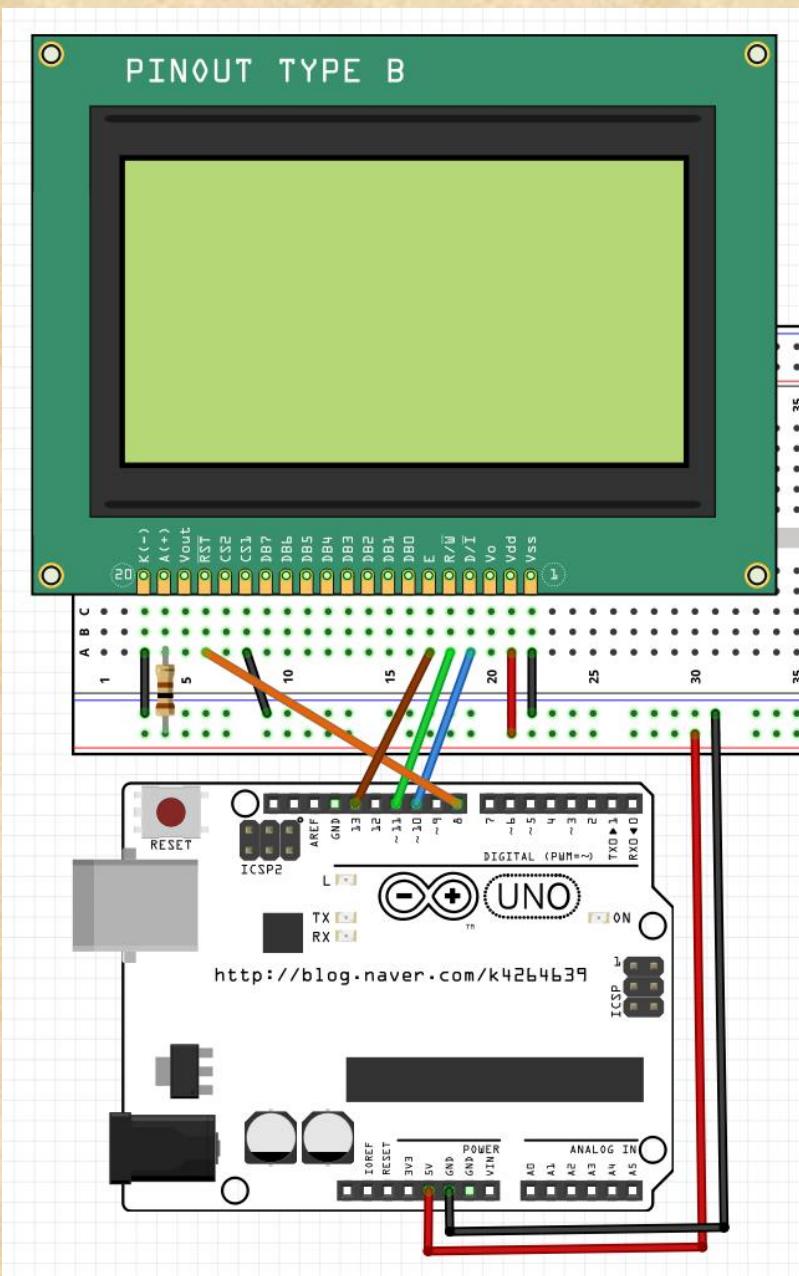
인터넷에 검색해 보니 아래와 같은 라이브러리를 찾을 수 있었습니다.

https://github.com/oli-kraus/U8g2_Arduino

이 라이브러리를 이용하여 구동해 보도록 하겠습니다.

우선은 가장 많이 사용하는 아두이노 UNO 기준으로 테스트를 해보도록 합니다.

회로의 구성은 아래와 같이 합니다.



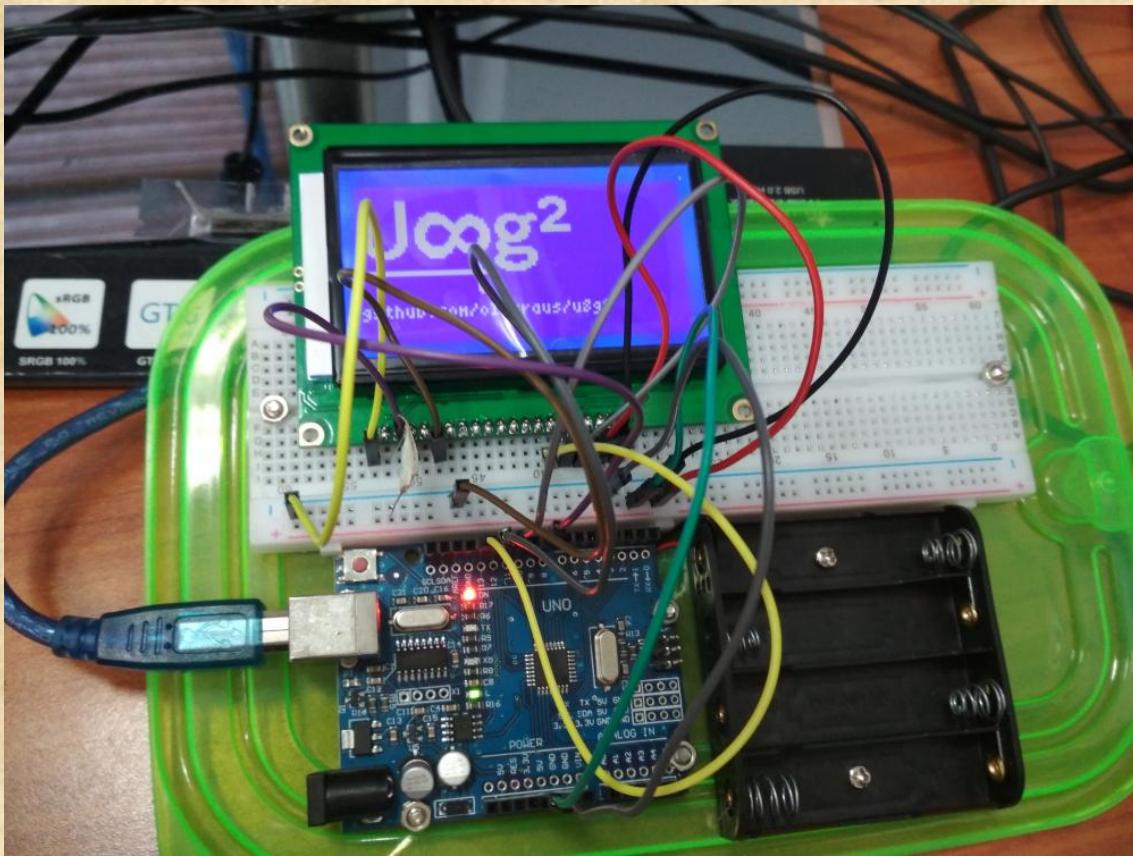
소스코드는 라이브러리의 많은 예제 중 U8g2Logo 예제를 사용하였습니다.
U8g2Logo.i no 파일의 소스 코드 중 아래 내용을 주석 해제 처리하여 줍니다.

```

1234 12345678901234567890123456789012345678901234567890123456789012345678901234
5 567890
1 U8G2_ST7920_128X64_F_HW_SPI u8g2(U8G2_R0, /* CS=*/ 10, /* reset=*/ 8);

```

그리고 업로딩을 하면 아래와 같이 작동하는 걸 확인할 수 있습니다.



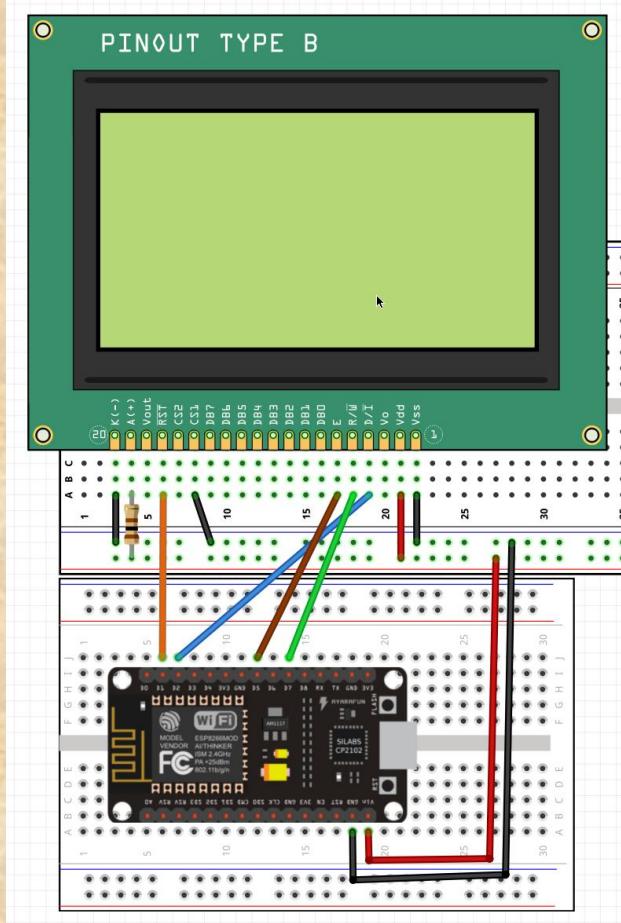
관련 자료 : [korean_display_18](#)

■ ESP8266으로 GLCD(128x64)에 출력하기

이제 아두이노 UNO에서 구현했던 것을 ESP8266에서 구현해 봅니다.
소스코드는 핀번만 수정해주고 연결을 수정하면 되어야 하겠습니다만...
작동하지 않습니다...
이런 일이 발생할 수 있을 듯 하여 ESP8266에 먼저 테스트 하지 않고
아두이노 UNO에서 먼저 테스트 해봤습니다.

인터넷 검색에서 u8g2 라이브러리가 esp8266에서 정상 작동하지 않는다는 걸 알았 보고 아두이노 UNO에서 먼저 테스트 해 보고 정상 작동하는 걸 확인하였습니다만
ESP8266에서 구동해 보니 역시 구동이 되지 않습니다.

테스트한 회로도는 아래와 같습니다.



소스코드는 아래와 같이 변경을 했습니다.

kProject 엘정풍향 프로젝트

어쨌든 u8g2라이브러리는 위에처럼 수정해도 ESP8266에서 작동하지 않습니다.

그래서 이번에는 인터넷 검색을 다시 시작...

프로젝트의 목표가 GLCD를 구동하기 였으면 데이터 시트보고 하나씩 만들어 보겠지만

목표가 GLCD의 구동이 아니므로 최대한 사용가능한 라이브러리를 검색해서 사용할 생각입니다.

인터넷에서 ESP8266에서 사용 가능한 다른 라이브러리를 찾았습니다.

기본 베이스는 u8g2라이브러리인데 ESP8266에서 구동 가능하도록 수정한 것 같습니다.

원본은 아래 사이트에서 확인하실 수 있습니다.

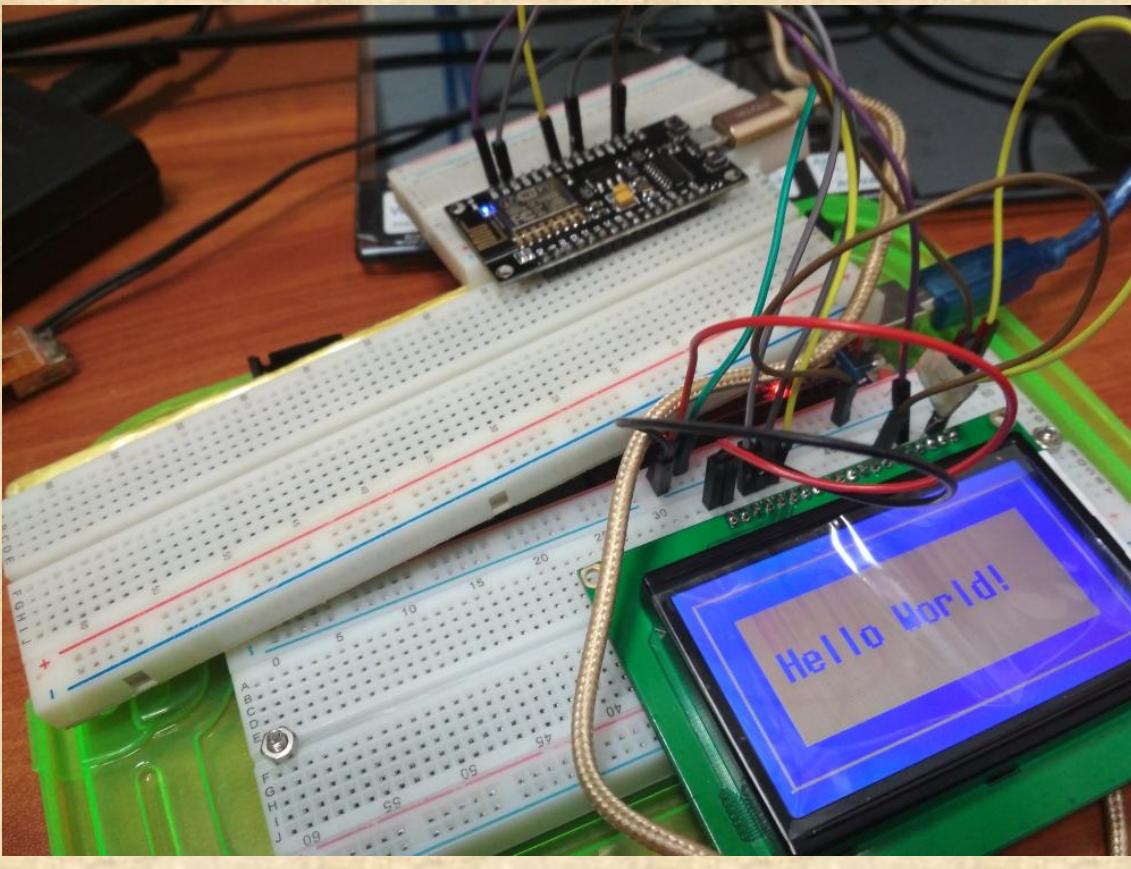
http://cpre.kmutnb.ac.th/esl/learning/index.php?article=esp8266_demo_program-3

원문이 영어도 아니고 태국어인지? 무슨 글인지... 그림과 기호로 추정해 보니 아래에 ST7920용 소스코드가 있어 구동시켜 봅니다.

회로도는 앞선 회로도와 동일하며 소스코드에서의 핀 설정은 아래와 같이 수정하시면 됩니다.

이제 구동을 해보면 정상적으로 작동하는 걸 확인하실 수 있습니다.
테스트 코드는 esp8266_st7920_u8g2_demo.ino 입니다.

kProject 열정 풍랑 프로젝트



전체적인 구동 절차를 스스로 확인해 보겠습니다.

소스코드는 아래와 같습니다.

```
1  #include <Arduino.h>
2  #include <SPI.h>
3
4  #include "U8g2lib.h" // use: https://github.com/olikraus/u8g2/
5          // see: https://github.com/olikraus/u8g2/wiki/setup_tutorial
6
7  /**
8   * E  <--> GPIO-14 (D5) [SCK]
9   * RW <--> GPIO-13 (D7) [MOSI]
10  * RS <--> GPIO-4 / D2
11  * RST <--> GPIO-5 / D1
12  */
13
14 // use Software SPI
15 //U8G2_ST7920_128X64_1_SW_SPI u8g2( U8G2_R0, 14 /*SCK*/, 13 /*MOSI*/, 4 /*CS*/, 5 /*RST*/ );
16
17 // use Hardware SPI
18 U8G2_ST7920_128X64_1_HW_SPI u8g2( U8G2_R0, 4 /*CS*/, 5 /*RST*/ );
19
20 int w,h;
21 int disp = 0;
22
23 void setup(void) {
24     Serial.begin(115200);
```

kProject 열정동향 프로젝트

```
24     Serial.println("\n\n\n\n");
25     Serial.flush();
26     delay(5000);
27
28     u8g2.begin();
29     u8g2.setContrast( 0x10 );
30     u8g2.setDisplayRotation( U8G2_R2 );
31     u8g2.setColorIndex(1);
32     w = u8g2.getWidth();
33     h = u8g2.getHeight();
34     randomSeed( analogRead(0) );
35 }
36
37 void draw_page0() {
38     u8g2.setFont( u8g2_font_7x14B_tf );
39     u8g2.drawFrame( 0, 0, w, h );
40     u8g2.drawBox( 10 /*x*/, 10 /*y*/, w-20 /*w*/, h-20 /*h*/ );
41     u8g2.setColorIndex(0); // pixel off
42     u8g2.drawString( 16, 35, "Hello World!" );
43     u8g2.setColorIndex(1); // pixel on
44 }
45
46 void draw_page1() {
47     u8g2.setFont( u8g_font_6x10r );
48     u8g2.drawRFrame( 0, 0, w, h, 4 ); // draw rounded frame
49     u8g2.drawString( 4 /*x*/, 10 /*y*/, "128x64 LCD (ST7920)" );
50     u8g2.drawString( 4 /*x*/, 20 /*y*/, "ESL @ KMUTNB" );
51     u8g2.drawString( 4 /*x*/, 30 /*y*/, "Date: 2016-12-05" );
52     u8g2.drawString( 4 /*x*/, 40 /*y*/, "Temperature: 25.6 C" );
53 }
54
55 int bar_level = 0;
56
57 void draw_page2() {
58     u8g2.drawFrame( 0, (h/2)-6, w, 12 );
59     u8g2.drawBox( 4, (h/2)-4, bar_level, 8 );
60 }
61
62 void draw() {
63     switch (disp) {
64     case 0: draw_page0(); break;
65     case 1: draw_page1(); break;
66     case 2: draw_page2(); break;
67     default: break;
68     }
69 }
70
71 void loop(void) {
72     u8g2.firstPage();
73     do {
74         draw();
75     } while ( u8g2.nextPage() );
76     if (disp==2) {
77         bar_level = (bar_level + 10) % 120;
78     }
79     disp = (disp+1) % 3;
80     delay(3000);
81 }
82 }
```

28번 줄에서 begin으로 시작후 setDisplayRotation로 화면의 방향을 결정하네요..

kProject 월정뚱땅 프로젝트

이 후 색상을 설정하고 좌표로 그리는 함수를 실행하면 되는 간단한 구조입니다.

여기서 u8g2변수 정의시 주의할 부분이 변수형명에 F를 선언하셔야 합니다.
즉, 위의 예제에서는 변수 선언시 17번째 줄에서

U8G2_ST7920_128X64_1_HW_SPI

와 같이 선언을 하였으나

일반적인 사용시에는

`U8G2_ST7920_128X64_F_HW_SPI`로 선언하여야 합니다.

위와 같이 U8G2_ST7920_128X64_1_HW_SPI 로 선언을 할 경우 Page buffer 모드로 구동이 되며

이는 실제 화면에 표시되지 않는 가상의 한페이지를 모두 그리고 이 가상의 페이지를 화면에 표시되는 페이지로 전환하는 기법으로
화면의 깜빡거림을 없앨수 있지만
속도상의 저하를 감수하셔야 합니다.

우리는 실시간으로 화면에 직접 그림 혹은 글짜를 표현하고 싶은 경우이므로 이를 사용하지 않고 “ 1 ” 대신 F를 선언하여

U8G2_ST7920_128X64_F_HW_SPI

로 선언을 하여야 합니다.

라이브러리 설명서에서는 이 방법을 “full buffer” 모드라고 정의하고 있습니다.

이 모드로 구동을 할 경우 그리기 학수를 실행한 이후에

u8g2.sendBuffer();

를 해 주어야만 화면에 반영이 됩니다.

이제 이를 응용하여 원하는 좌표에 점을 찍도록 해 보겠습니다.

kProject 얼굴 풍성 프로젝트

```
12     u8g2.setDisplayRotation( U8G2_R2 );
13     u8g2.setColOrIndex(1);
14     u8g2.clearBuffer();
15 }
16
17 int x = 0;
18 int y = 0;
19
20 void loop(void)
21 {
22     u8g2.drawPixel(x, y);
23     x++;
24     if (x >= 128)
25     {
26         x = 0;
27         y++;
28         if (y >= 64)
29         {
30             x = 0;
31             y = 0;
32         }
33     }
34     u8g2.sendBuffer();
35 }
36 }
```

간단하게 모든 화면에 점을 찍는 코드입니다.

하지만 한점씩을 찍고 매번 sendBuffer 함수를 실행하게되어 속도는 생각보다 느리게 출력이 됩니다.

현재는 기능 구현을 우선적으로 해야 하므로 속도는 나중에 최적화 할 예정입니다.

관련 자료 : [korean_display_19](#)

■ GLCD(128x64)에 한글 출력하기

이제 한글 출력 컨트롤러로 돌아가서 GLCD에서도 한글이 출력되도록 하겠습니다.

이미 앞서 GLCD에 출력하는 루틴을 확인했으므로 초기화와 draw_pixel 함수 그리고 display_clear 함수만 업데이트하면 GLCD에서도 한글이 출력이 가능합니다.

GLCD장치의 초기화는 장치 초기화 명령을 받았을 때의 처리 루틴을 추가하면 됩니다.

```
1234 12345678901234567890123456789012345678901234567890123456789012345678901234  
5 567890  
  
1 case 0x05: // 장치 초기화  
2 {  
3     unsigned char val = read_serial_data();  
4     if (val == 1)  
5     {  
6         init_oled();  
7     }  
8     else  
9     if (val == 2)  
10    {  
11        init_spi_tft();  
12    }  
13    else  
14    if (val == 3)  
15    {  
16        init_glcd();  
17    }  
18    return;  
19 }  
20 break;  
21
```

draw_pixel 함수와 display_clear 함수는 아래와 같이 수정하였습니다.

```
1234 12345678901234567890123456789012345678901234567890123456789012345678901234  
5 567890  
  
1 void draw_pixel (int x, int y, uint16_t color)  
2 {  
3     switch (active_display_index)  
4     {  
5         case 0: // 디스플레이 장치 활성화 안됨  
6             break;  
7         case 1: // OLED I2C 인 경우  
8             display.drawPixel (x, y, color != 0);  
9             break;
```

kProject 열정 풍향 프로젝트

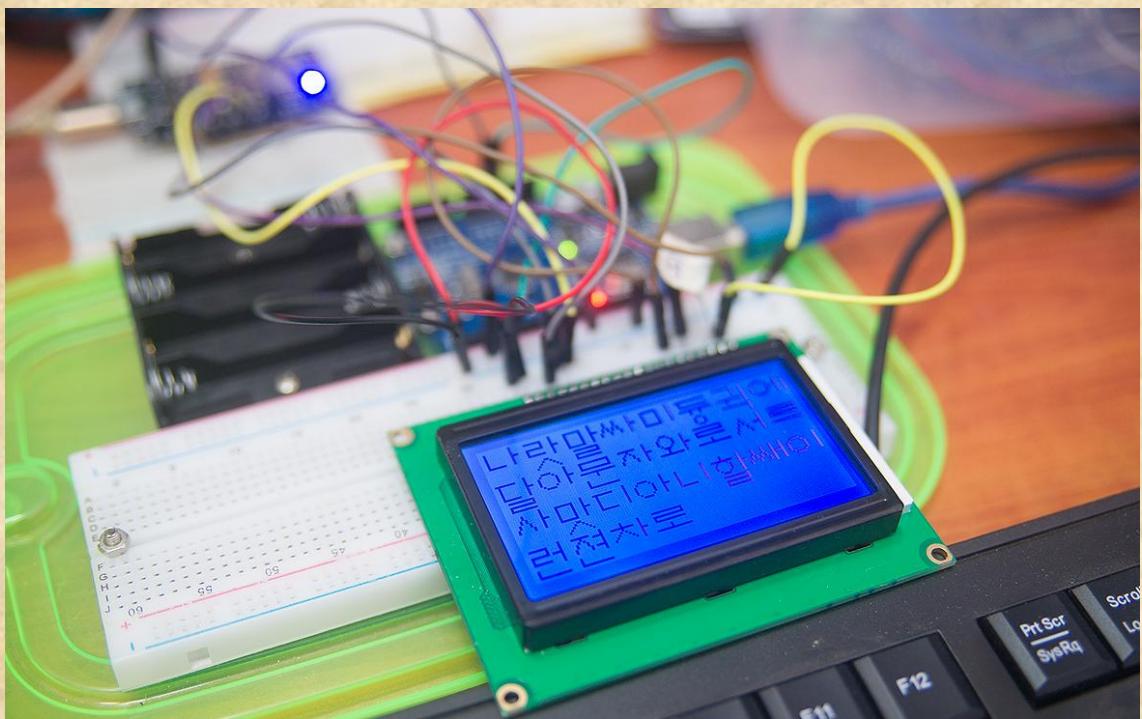
```
10 case 2: // TFT SPI LCD 인 경우
11     tft.drawPixel(x, y, color);
12     break;
13 case 3: // GLCD 인 경우
14     if (x >= 128) return;
15     if (x < 0) return;
16     if (y >= 64) return;
17     if (y < 0) return;
18     {
19         u8g2.setColOrIndex(1);
20         u8g2.drawLine(x, y);
21         u8g2.sendBuffer();
22     }
23     break;
24 }
25 }
26
27 void display_clear()
28 {
29     switch (active_display_index)
30     {
31         case 0: // 디스플레이 장치 활성화 안됨
32             break;
33         case 1: // OLED I2C 인 경우
34             display.clearDisplay();
35             display.display();
36             break;
37         case 2: // TFT SPI LCD 인 경우
38             tft.fillScreen(ST7735_BLACK);
39             break;
40         case 3: // GLCD 인 경우
41             u8g2.clearBuffer();
42             u8g2.sendBuffer();
43             break;
44     }
45 }
```

kProject 얼굴인식 프로젝트

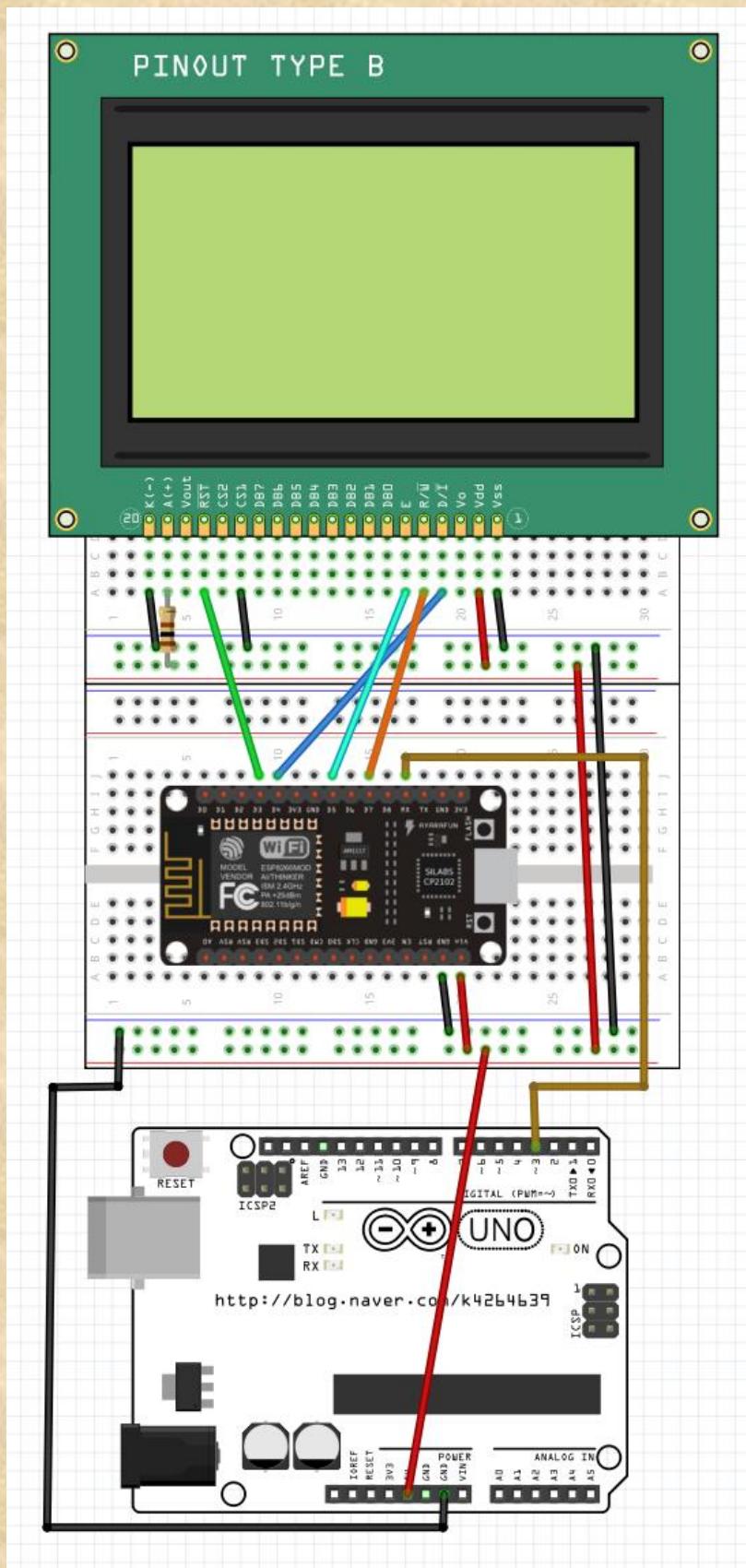
```
15     di_splay_init(3);
16     delay(100);
17     di_splay_activate(3);
18     delay(100);
19     di_splay_clear();
20 }
21
22 char ch[4];
23
24 int i2c_pos_x = 0;
25 int i2c_pos_y = 0;
26 int spi_tft_pos_x = 0;
27 int spi_tft_pos_y = 0;
28
29 int font_index = 1;
30
31 void loop()
32 {
33     di_splay_set_han_font(font_index);
34     di_splay_set_color(1);
35     di_splay_activate(3);
36     for (int i = 0; i < 106; i++)
37     {
38         ch[0] = hun[i * 3 + 0];
39         ch[1] = hun[i * 3 + 1];
40         ch[2] = hun[i * 3 + 2];
41         ch[3] = 0;
42         di_splay_set_cursor(i2c_pos_x, i2c_pos_y);
43         di_splay_print(ch);
44         i2c_pos_x += 16;
45
46         if (i2c_pos_x >= 128)
47         {
48             i2c_pos_y += 16;
49             i2c_pos_x = 0;
50             if (i2c_pos_y >= 64)
51             {
52                 i2c_pos_y = 0;
53                 di_splay_clear();
54             }
55         }
56
57         font_index++;
58         if (font_index > 142) font_index = 1;
59
60         delay(3000);
61
62     }
63 }
64 }
```

GLCD의 장치번호는 3번입니다.

구동화면은 아래와 같이 정상적으로 작동이 됩니다만
현재 최적화가 되어 있지 않은 상태이므로 매우 느리게 출력이 됩니다.



관련 자료 : korean_display_20



■ 현재까지의 핀 정리

지금까지 다양한 모듈에 대하여 출력이 가능한 한글 디스플레이 컨트롤러의 기본 기능등을 구현하였습니다.

방식이 다른 여러가지 출력 모듈이 사용되고 MCU로 ESP8266을 사용하다보니 동시에 여러가지 모듈에 출력을 할 경우 다른 모듈과 핀이 중복사용이 될 수도 있으므로 현재까지의 모듈들이 사용하는 핀을 정리해 보고자 합니다.

OLED(I2C)

SCL - GPIO 5

SDA - GPIO 4

TFT(SPI)

RES - GPIO 16

A0 - GPIO 2

SDA - GPIO 13(MOSI)

SCK - GPIO 14(SCK)

CS - GPIO 15

GLCD(SPI)

RST - GPIO 0

D/I - GPIO 2

E - GPIO 14(SCK)

R/W - GPIO 13(MOSI)

여기서 보면 GPIO-2가 TFT(SPI)일 경우와 GLCD(SPI)일 경우 중복되는 걸 알 수 있습니다.

동시에 두가지의 모듈을 모두 연결하였을 경우 TFT에 신호를 보내면서 동시에 GLCD에 신호를 보내는 경우는 없습니다.

순차적으로 TFT에 신호를 보내고 종료 후 GLCD에 신호를 보냅니다.

이때 현재 사용중인 모듈을 선택하는 CS핀은 TFT의 경우 GPIO 15번 핀이며 GLCD의 경우 D/I(GPIO2)입니다. 따라서 TFT(SPI)를 사용중인 경우 A0(GPIO 2)의 핀상태는 계속 변경이 될 것이며, 이에 따라 GLCD(SPI)의 D/I 핀의 변경에 따라 GLCD의 Chip Enabled와 Disabl ed가 변경되어 작동에 영향을 줄 것으로 보입니다.

따라서 현재의 설계에서 D/I를 GPIO-0로 핀 연결을 변경하고 RST핀은 연결

을 하지 않도록 하겠습니다.
수정된 핀 배치는 아래와 같습니다.

OLED(I2C)

SCL - GPIO 5
SDA - GPIO 4

TFT(SPI)

RES - GPIO 16
A0 - GPIO 2
SDA - GPIO 13(MOSI)
SCK - GPIO 14(SCK)
CS - GPIO 15

GLCD(SPI)

D/I - GPIO 0
E - GPIO 14(SCK)
R/W - GPIO 13(MOSI)

■ GLCD 한글 출력 속도 향상 시키기

현재 만들어진 GLCD에 한글 출력 기능의 경우 속도가 매우 느립니다. 이유는 GLCD에 1개 DOT를 출력 후 전체 화면을 갱신하는 과정을 거치기 때문입니다.

기존 코드 작성 시 우선 기능을 수행할 수 있도록 하기 위하여 이부분의 좀 더 효율적인 방법은 고려하지 않고 한글이 출력 되기 위한 방법만을 고려했습니다.

이제 이부분의 코드를 수정하여 좀더 효율적으로 한글을 출력할 수 있도록 해 보도록 하겠습니다.

속도의 향상 방법은 한개 DOT 출력 후 화면을 갱신하는게 아닌 한글 1개 글짜를 출력 후 화면을 갱신하는 방법으로 하겠습니다.

이전의 1개 DOT를 출력하고 화면을 갱신할 경우 속도도 느리지만 한개 점씩 출력되는게 인식이 가능한 점이 있었습니다.

하지만 한개 글자를 출력 후 화면을 갱신할 경우에는 한개의 점 출력하는게 인식이 되지는 않지만 한개 글자가 출력되고 다음 글자가 출력되는 과정이 인식이 되게 됩니다.

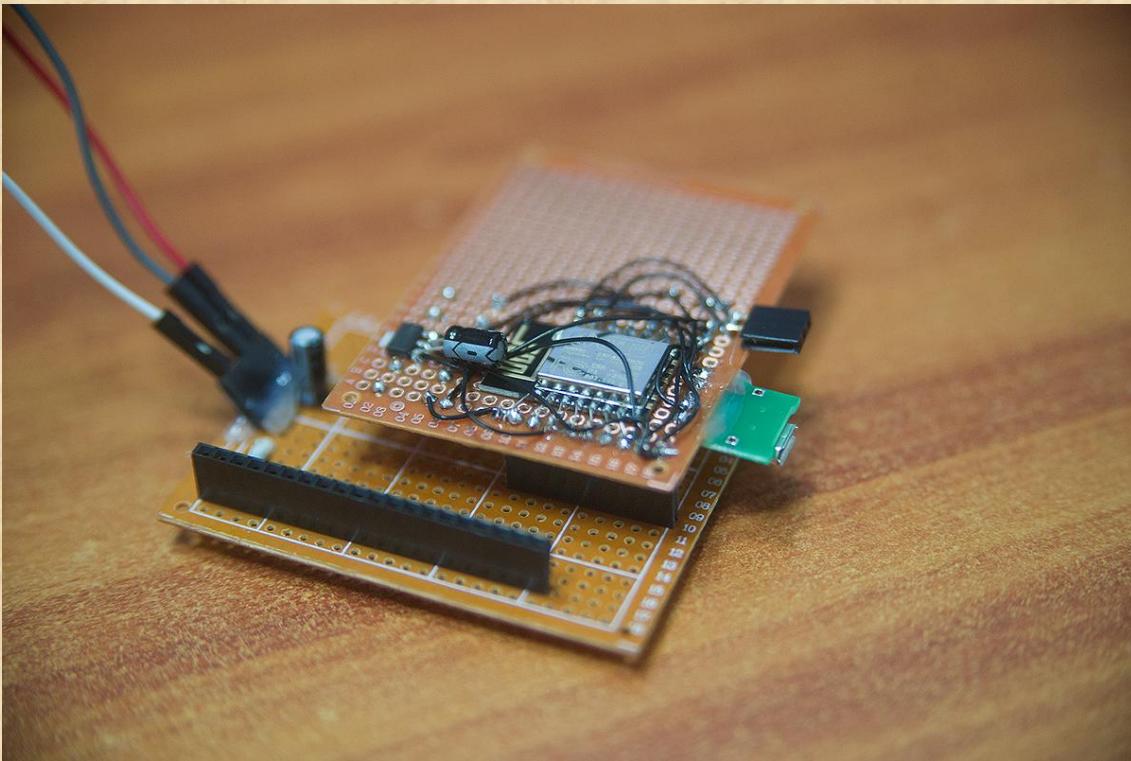
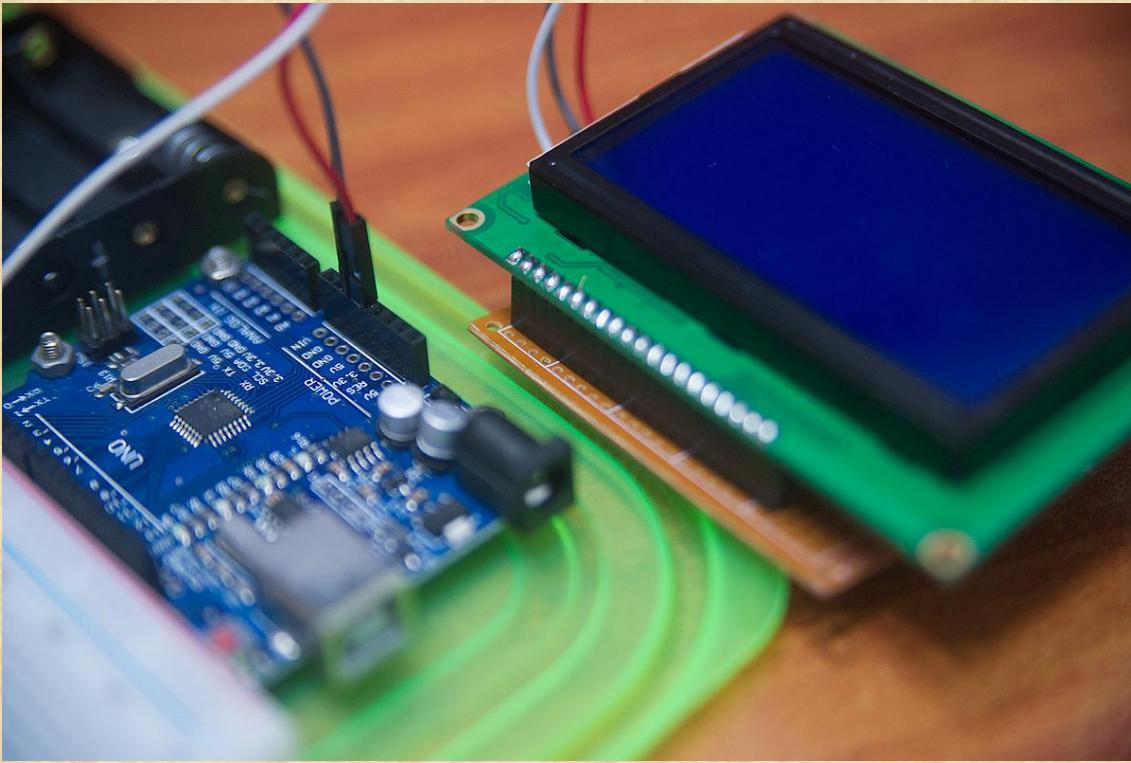
즉, 전체 화면에 모든 글자가 출력이 되는게 아닌 화면에 한글자가 출력되고 다시 다음 글자가 출력되는게 인식이 됩니다.

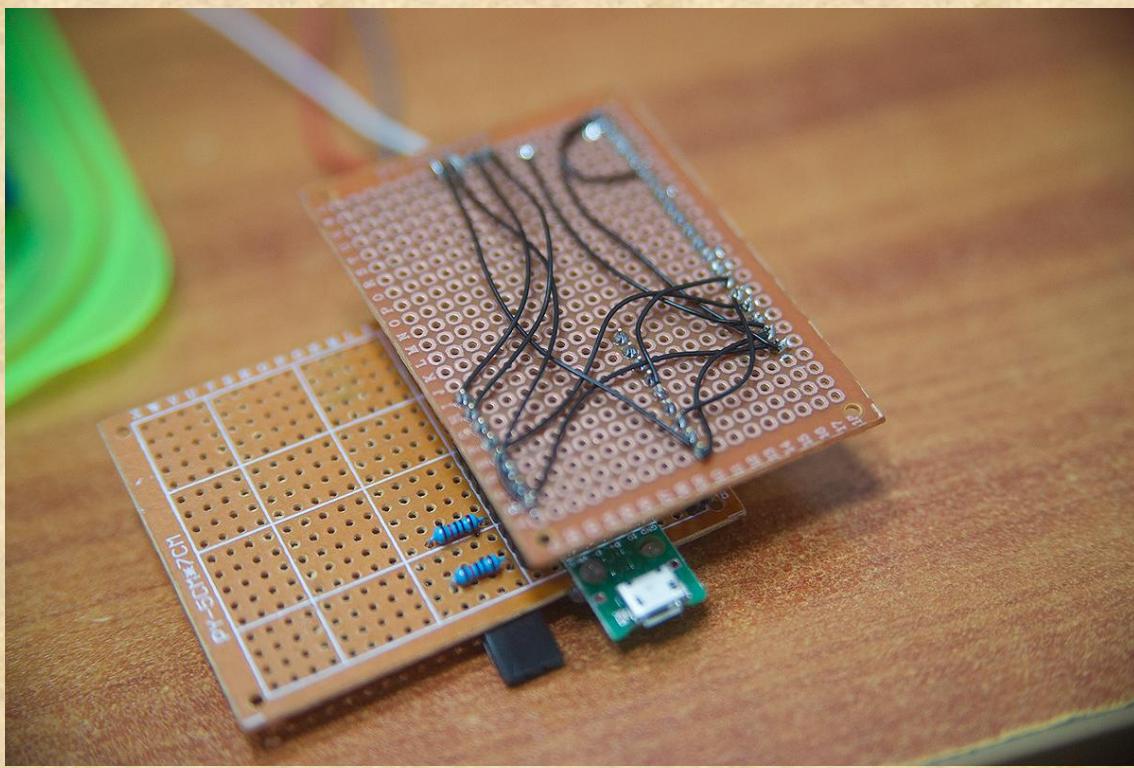
좀더 향상시키려면 전체화면에 모든 글자를 출력 후 한번에 화면을 갱신하면 출력 과정이 보이지 않고 한번에 글자들을 출력 할 수 있습니다.

이 부분은 추후 고려 예정이며 지금은 한글자씩만 갱신하도록 합니다.

그리고 현재까지 만들던 브레드보드의 회로는 관리가 너무 힘들어 이를 만능기판에 납땜하여 만들었습니다.

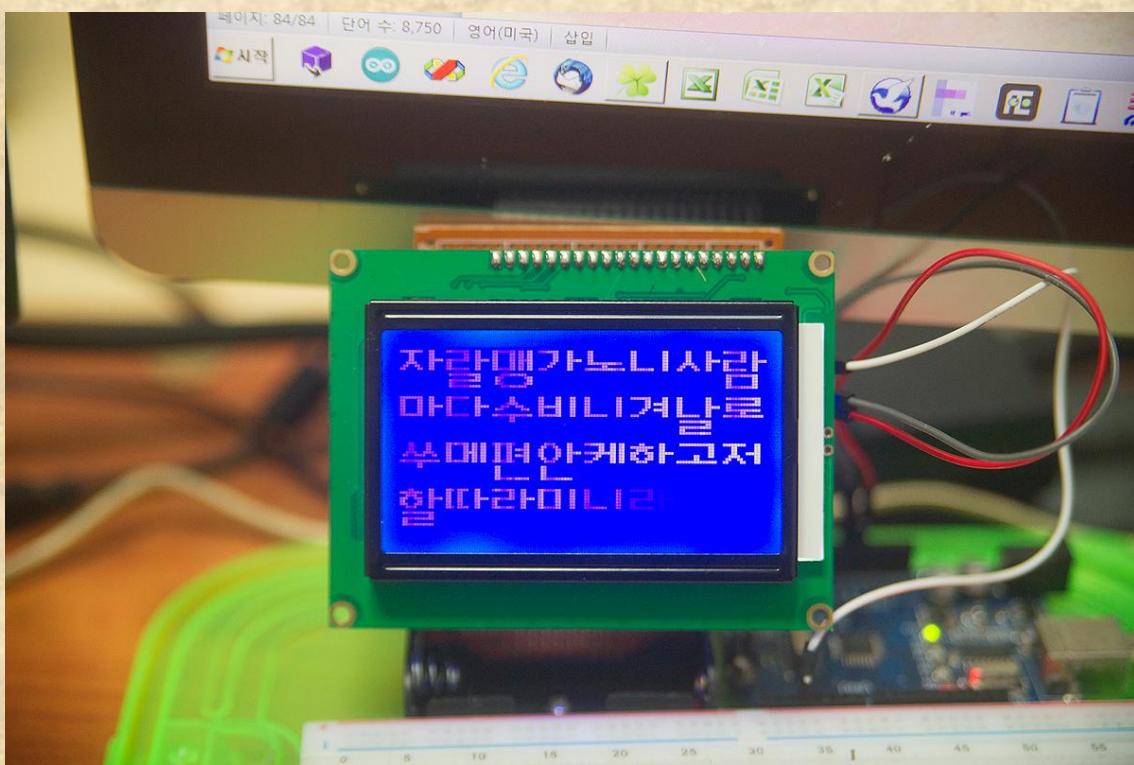
만능기판에 만든 상태는 아래 그림과 같습니다.





회로도는 기존 브레드보드와 동일 합니다.

구동화면은 다음과 같습니다.



관련 자료 : [korean_display_21](#)

■ 출력 방식의 변경

현재까지의 한글 출력 방식은 시리얼 통신을 이용하여 출력할 문자가 전달이 되면 LCD모듈은 즉시 LCD에 출력하도록 하고 있습니다.

이 방법을 사용을 하게 되면 LCD에 화면을 출력시 한번에 화면이 출력 되는
것이 아닌 한글자씩 연속적으로 출력이 되면서
동영상 처럼 출력 되는 현상이 발생합니다.

이 현상을 제거하기 위하여 출력 방식을 우선 시리얼 통신을 이용하여 전달된 출력 데이터를 바로 LCD에 반영하지 않고 우선 별도의 메모리에 저장하고 있다가 화면에 출력하라는 명령이 접수가 되면 화면에 반영하는 방식으로 변경을 하겠습니다.

이제는 명령이나 데이터가 전달이되면 바로 LCD에 반영이 되지 않습니다.
글자를 시리얼 통신을 이용하여 전달한 후 화면 갱신 명령을 주어야 화면이
변경이 됩니다.

이제 기존의 명령중 display_clear은 명령을 송신해도 바로 화면이 지워지지 않습니다.

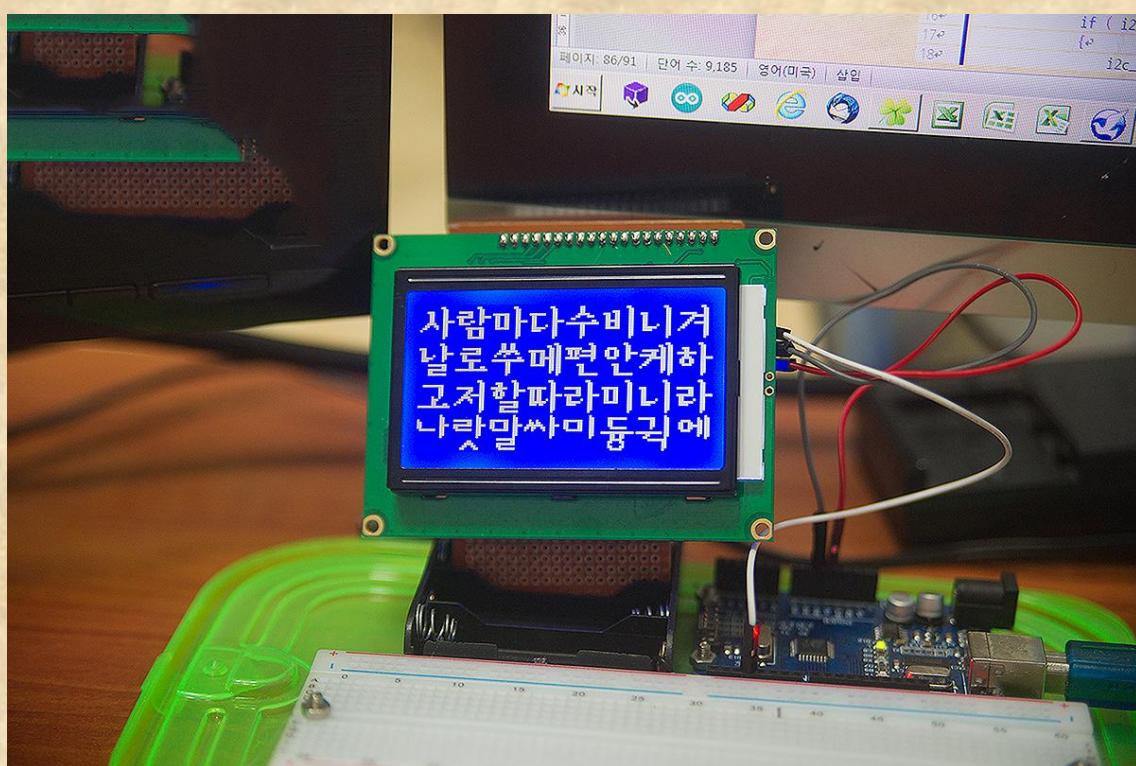
이후 `display_refresh` 명령을 전송하여 화면이 지워지게 됩니다.

또한 화면에 글자 출력도 마찬가지로 송신 즉시 화면에 출력이 되는게 아닌
di spl ay_refresh명령을 전송하여야 출력이 됩니다.

다음은 화면에 글자를 한페이지씩 출력 하는 예입니다.

kProject 열정 풍랑 프로젝트

```
19     i2c_pos_x = 0;
20     if ( i2c_pos_y >= 64 )
21     {
22         display_refresh();
23         i2c_pos_y = 0;
24         display_clear();
25     }
26 }
27 }
28 font_index++;
29 if ( font_index > 142 ) font_index = 1;
30 }
31 }
```

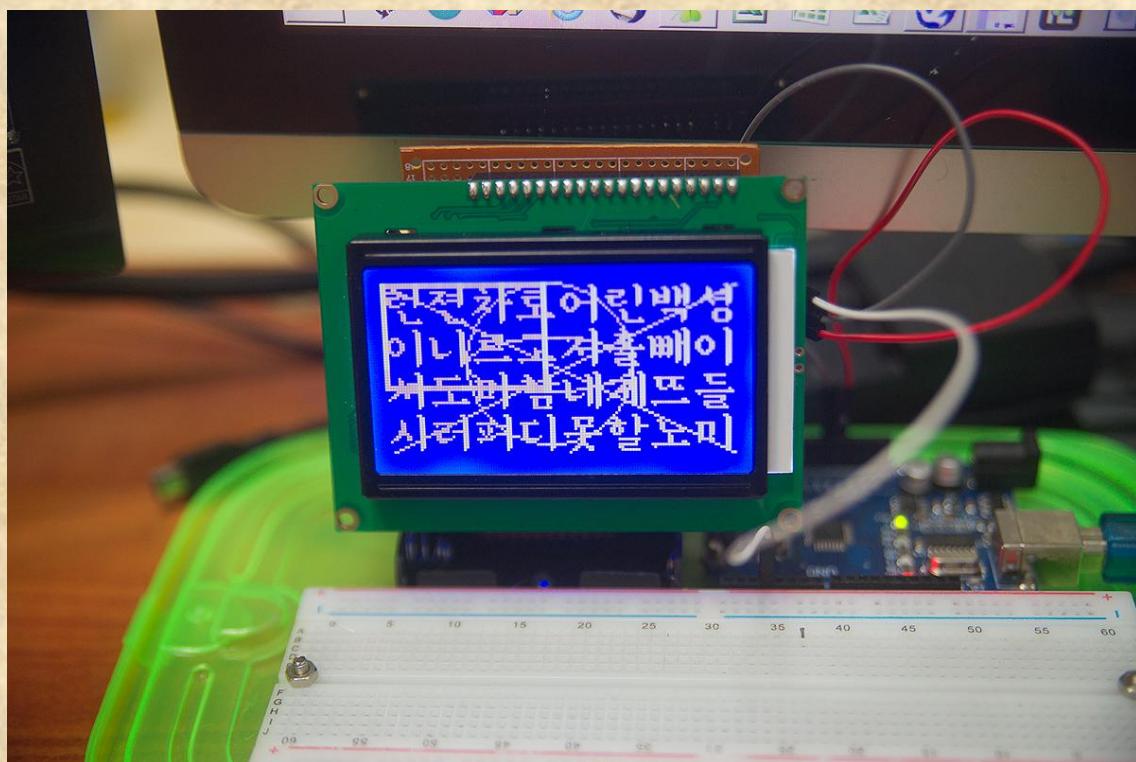


관련 자료 : korean_display_23

■ GLCD 그래픽 출력 기능 추가

지금까지 한글을 출력하기 위한 기능만 추가하고 있었습니다.
하지만 사용하는 출력 장치가 문자용이 아닌 그래픽 용이므로
이제 나머지 그래픽 기능들도 출력 할 수 있도록 하겠습니다.
우선은 GLCD에 대해서 그래픽 기능을 추가하고
나머지 OLED와 TFT등에 대해서는 추후 기능 추가를 하도록 하겠습니다.
추가한 그래픽 함수는 아래와 같습니다.

함수의 사용 예는 아래 참조 바랍니다.



kProject 엘정풍향 프로젝트

■ 한글 문자열의 전송

지금까지는 문자열이 한글이라는 가정하에서 콘트롤러에 문자열을 전송하도록 하였습니다.

이는 시리얼 통신을 이용하여 데이터를 전송할 때 1문자에 해당하는 바이트를 연속으로 보내야 하기 때문에 송신시 한문자에 대한 바이트를 한번에 보내야 하며

1개 BYTE씩 송신을 하고 명령을 송신하는 방식으로 처리하면 안됩니다.

즉 1개 BYTE씩 송신을 하며 중간에 명령을 넣을 경우 한문자가 모두 전송되지 않은 상태에서 다른 명령이 전달되면서 명령의 데이터가 문자 데이터로 인식되는 오류가 발생할 수 있습니다.

따라서 모든 문자를 한글로 처리하게 하여 UTF8에서 한글 한문자가 3BYTE이므로

3개 바이트씩 송신을 하여 문자열에서 한문자만을 보낼 수 있도록 하였습니다만

이는 테스트를 위하여 한글만 송신한다는 가정으로 작성한 부분이었습니다.
이번에는 한글 영문이 섞여 있는 문자열을 전달하는 명령을 만들었습니다.

```
1 void display_print_message(char *message)
2 {
3     int index = 0;
4     int len = strlen(message);
5     while (index < len)
6     {
7         int bytes = get_char_byte(message + index);
8         if (bytes == 1)
9         {
10             soft_serial.write(message[index]);
11         }
12         else if (bytes == 3)
13         {
14             soft_serial.write(message[index]);
15             soft_serial.write(message[index+1]);
16             soft_serial.write(message[index+2]);
17         }
18         index += bytes;
19     }
20 }
```

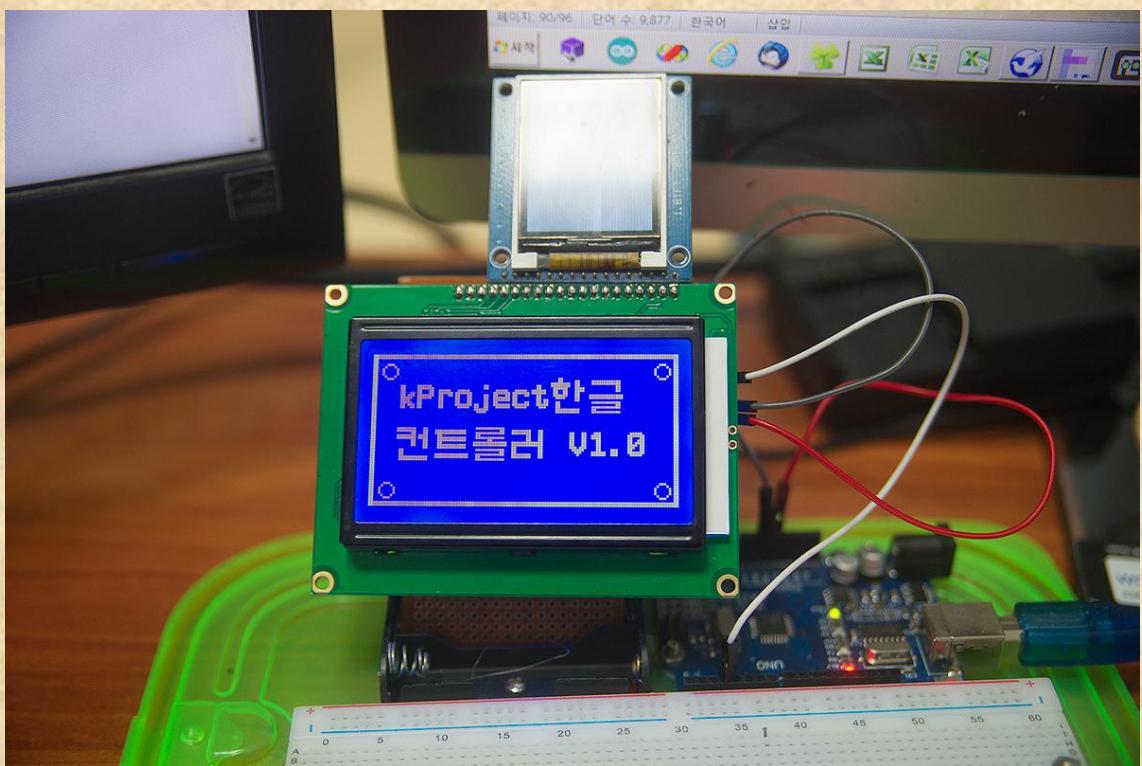
이제 사용 예제를 확인해 보겠습니다.

```
1234      1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234  
5      567890  
  
1      #include "kp_korean_display.h"  
2  
3      char test_message1[] = "kProject 한글";  
4      char test_message2[] = "컨트롤러 V1.0";
```

kProject 열정동향 프로젝트

```
5 void setup()
6 {
7     soft_serial.begin(9600);
8     display_init(3);
9     display_activate(3);
10 }
11
12 void loop()
13 {
14     display_set_han_font(1);
15     display_set_han_font(1);
16     display_set_color(1);
17
18     draw_back();
19
20     display_set_cursor(10, 10);
21
22     display_print_message(test_message1);
23     display_set_cursor(10, 30);
24     display_print_message(test_message2);
25     display_refresh();
26
27     while(1);
28 }
29
30
31 void draw_back()
32 {
33     display_clear();
34     display_set_cursor(0, 0);
35     display_circle(7, 7, 3);
36     display_circle(128-7, 64-7, 3);
37     display_circle(128-7, 7, 3);
38     display_circle(7, 64-7, 3);
39     display_rect(0, 0, 128, 64);
40     display_rect(1, 1, 126, 62);
41 }
42
43 }
```

kProject 얼굴인식 프로젝트



관련 자료 : [korean_display_24](#)

■ GLCD와 SPI 표시 장치의 동시 구동(실패)

한글 표시장치의 구동은 현재 GLCD와 TFT(SPI 통신), OLED(I2C통신)을 지원하고 있습니다.

원래의 목적은 이 세가지 모듈을 모두 연결하여 동시에 구동할 수 있도록 할 예정이었습니다.

I2C와 SPI의 구동에는 간섭이 없으며 SPI통신은 CS핀의 상태에 따라 해당 모듈만 구동할 수 있기 때문에 가능하리라 생각했습니다.

하지만 테스트 결과 GLCD와 SPI통신을 동시에 연결했을 경우 GLCD가 정상적으로 작동하지 않는 현상이 발생을 했습니다.

현재 GLCD의 연결은 아래와 같이 연결되어 있습니다.

GLCD(SPI)

D/I - GPIO 0

E - GPIO 14(SCK)

R/W - GPIO 13(MOSI)

여기서 D/I 핀이 Chip Enable와 Chip Disable을 선택하는 핀입니다.

SPI를 통해 GLCD가 아닌 다른 모듈을 구동할 때는 Chip Disable 상태로 설정하고 다른 모듈을 구동하고 GLCD를 구동할 때만 D/I 핀을 Chip Enable로 설정할 예정이었습니다.

하지만 실제 구동 시 정상 작동하지 않아 데이터시트를 확인해 보니 다음과 같은 문구가 있습니다.

				Parallel mode: 1: 8/4-bit parallel bus mode.
RS(CS*)	17	I	MPU	Parallel Mode: Register select. 0: Select instruction register (write) or busy flag, address counter (read); 1: Select data register (write/read). Serial mode: Chip select. 1: chip enabled; 0: chip disabled. When chip is disabled, SID and SCLK should be set as "H" or "L". Transient of SID and SCLK is not allowed.
				Parallel Mode: Read/Write control.

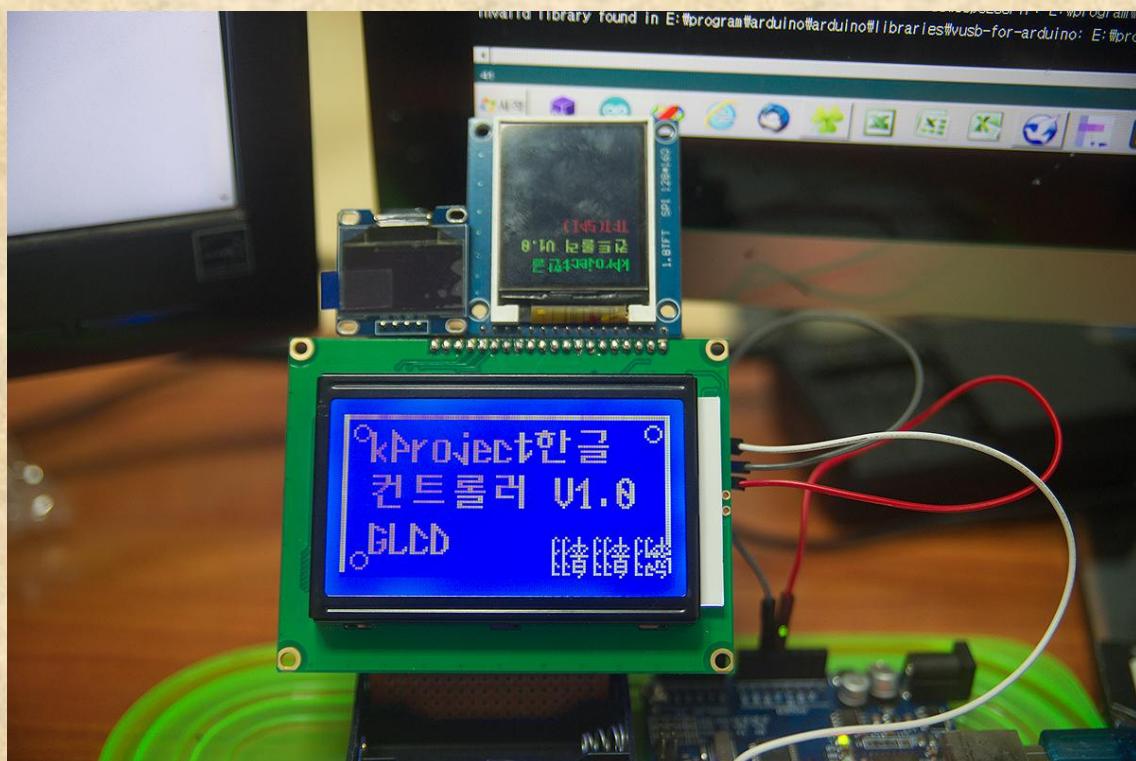
Chip Disable일 경우 SID는 “H”어야 하며 SCLK는 “L”로 설정이 되어 있어야 한답니다.

kProject 얼굴인식 프로젝트

즉, SPI통신을 공통핀으로 사용하지 못한다는 결론이 나오는군요....
따라서 SPI통신을 사용하는 디스플레이 모듈과 GLCD와의 동시 사용은 불가합니다.

GPIO핀의 여유가 있으면 Software SPI라도 구현을 하겠지만
이미 GPIO는 여유가 없어 모두 사용이 되고 있으므로
이번 버전에서의 동시 사용은 포기를 하고
한개의 기판에 여러가지 디스플레이장치를 여러개가 아닌 한개씩 연결할 수
있도록 만들도록 하겠습니다.

아래 사진을 보시면 GLCD의 글자가 깨진것을 알 수 있습니다.



관련 자료 : [korean_display_26](#)

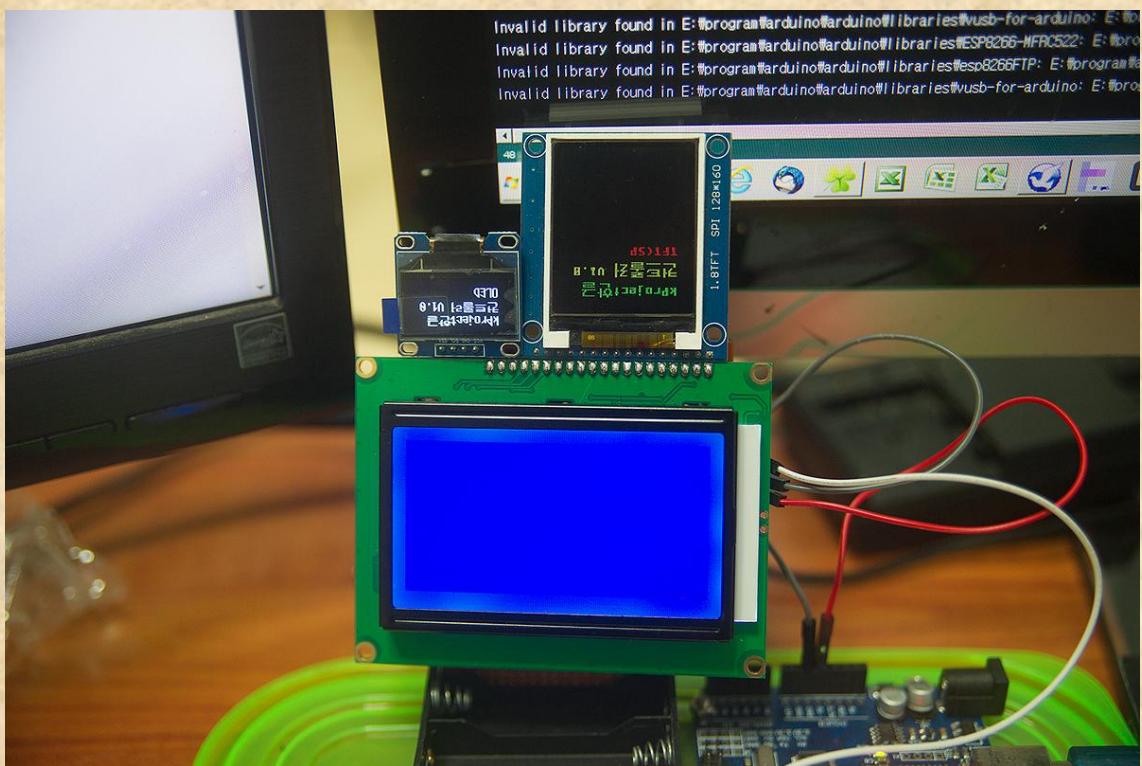
■ TFT(SPI)와 OLED(I2C) 출력 장치의 동시 구동

GLCD는 TFT(SPI)와의 동시 구동이 불가하므로 TFT(SPI) OLED(I2C)의 동시 구동을 확인해 보았습니다.

kProject 열정동향 프로젝트

```
53     di_splay_print_message(test_message4);
54     di_splay_set_cursor(10, 23);
55     di_splay_print_message(test_message5);
56     di_splay_set_cursor(10, 41);
57     di_splay_print_message(test_message6);
58     di_splay_refresh();
59 */
60     di_splay_activate(1);
61     di_splay_clear();
62     draw_back();
63     delay(200);
64     di_splay_set_cursor(10, 5);
65     delay(200);
66     di_splay_print_message(test_message7);
67     delay(200);
68     di_splay_set_cursor(10, 23);
69     delay(200);
70     di_splay_print_message(test_message8);
71     delay(200);
72     di_splay_set_cursor(10, 41);
73     delay(200);
74     di_splay_print_message(test_message9);
75     delay(200);
76     di_splay_refresh();
77
78     delay(1000);
79     han_font_index++;
80     eng_font_index++;
81     if (han_font_index > 142) han_font_index = 1;
82     if (eng_font_index > 74) eng_font_index = 1;
83 }
84
85 void draw_back()
86 {
87     di_splay_clear();
88     delay(200);
89     di_splay_set_cursor(0, 0);
90     delay(200);
91     di_splay_circle(7, 7, 3);
92     delay(200);
93     di_splay_circle(128-7, 64-7, 3);
94     delay(200);
95     di_splay_circle(128-7, 7, 3);
96     delay(200);
97     di_splay_circle(7, 64-7, 3);
98     delay(200);
99     di_splay_rect(0, 0, 128, 64);
100    delay(200);
101    di_splay_rect(1, 1, 126, 62);
102    delay(200);
103 }
104
105 }
```

kProject 열정 풍랑 프로젝트



관련 자료 : korean_display_27

■ GLCD와 OLED(I2C) 출력 장치의 동시 구동

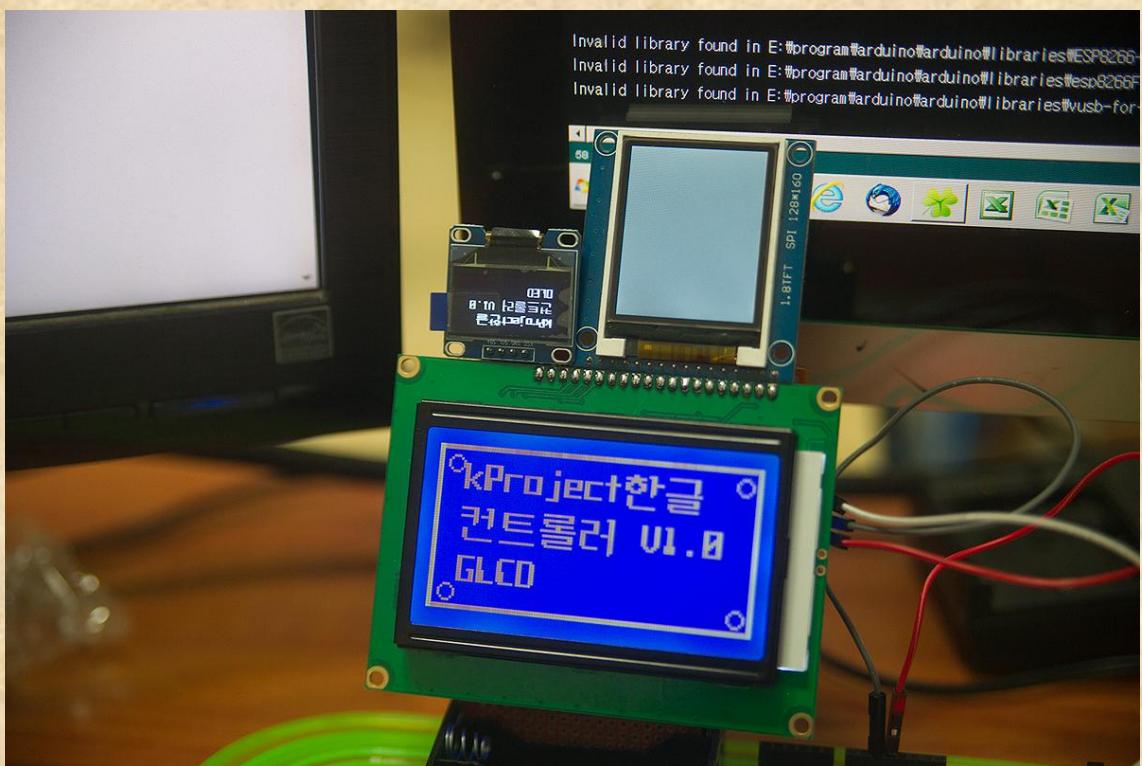
GLCD 와 OLED(I2C)의 동시 구동을 확인해 보았습니다.

```
1 #include "kp_korean_display.h"
2
3 char test_message1[] = "kProject 한글";
4 char test_message2[] = "컨트롤러 V1.0";
5 char test_message3[] = "TFT(SPI)";
6
7 char test_message4[] = "kProject 한글";
8 char test_message5[] = "컨트롤러 V1.0";
9 char test_message6[] = "GLCD";
10
11 char test_message7[] = "kProject 한글";
12 char test_message8[] = "컨트롤러 V1.0";
13 char test_message9[] = "OLED";
14
15 int han_font_index = 1;
16 int eng_font_index = 1;
17
18 void setup()
19 {
20     soft_serial.begin(9600);
21     display_init(1);
22     delay(1000);
23 //    display_init(2);
24 //    delay(2000);
25     display_init(3);
26     delay(100);
27 }
28
29 void loop()
30 {
31     display_set_han_font(han_font_index);
32     display_set_eng_font(eng_font_index);
33 /*
34     display_activate(2);
35     display_set_color(0b0000011111000000);
36
37     draw_back();
38     display_set_cursor(10, 10);
39     display_print_message(test_message1);
40     display_set_cursor(10, 30);
41     display_set_color(0b1110011111000000);
42     display_print_message(test_message2);
43     display_set_cursor(10, 50);
44     display_set_color(0b1111100000000000);
45     display_print_message(test_message3);
46     display_refresh();
47 */
48     display_activate(3);
49     display_clear();
50     draw_back();
51     display_set_cursor(10, 5);
52     display_print_message(test_message4);
```

kProject 열정동향 프로젝트

```
53     diplay_set_cursor(10, 23);
54     diplay_print_message(test_message5);
55     diplay_set_cursor(10, 41);
56     diplay_print_message(test_message6);
57     diplay_refresh();
58
59     diplay_activate(1);
60     diplay_clear();
61     draw_back();
62     delay(200);
63     diplay_set_cursor(10, 5);
64     delay(200);
65     diplay_print_message(test_message7);
66     delay(200);
67     diplay_set_cursor(10, 23);
68     delay(200);
69     diplay_print_message(test_message8);
70     delay(200);
71     diplay_set_cursor(10, 41);
72     delay(200);
73     diplay_print_message(test_message9);
74     delay(200);
75     diplay_refresh();
76
76
77     delay(1000);
78     han_font_index++;
79     eng_font_index++;
80     if (han_font_index > 142) han_font_index = 1;
81     if (eng_font_index > 74) eng_font_index = 1;
82 }
83
84 void draw_back()
85 {
86     diplay_clear();
87     delay(200);
88     diplay_set_cursor(0, 0);
89     delay(200);
90     diplay_circle(7, 7, 3);
91     delay(200);
92     diplay_circle(128-7, 64-7, 3);
93     delay(200);
94     diplay_circle(128-7, 7, 3);
95     delay(200);
96     diplay_circle(7, 64-7, 3);
97     delay(200);
98     diplay_rect(0, 0, 128, 64);
99     delay(200);
100    diplay_rect(1, 1, 126, 62);
101    delay(200);
102 }
```

kProject 얼굴인식 프로젝트



관련 자료 : korean_display_28

■ PCB 설계

지금까지 브레드보드, 그리고 만능기판에 납땜하여 모듈을 설계하고 있습니다.

하지만 매번 GLCD, TFT(SPI), OLED(I²C)를 번갈아 가며 연결하기도 힘들고 하니 PCB로 기판을 만들어 버리겠습니다.

이미 ESP8266의 모든 GPIO를 사용해 버려서 더이상의 하드웨어 기능 추가는 불가능하고 펌웨어만 더 업데이트가 있을 예정이므로 현재 상태의 기능으로 기판을 만들어도 큰 문제가 되지 않을 듯 합니다.

초기의 계획했던 기능들의 추가는 아직 많은 부분이 미진합니다.

하지만 ESP8266의 모든 핀을 사용하였기 때문에

여기서 하드웨어적인 기능 추가를 하려면 핀 확장을 위한 74HC595 등의 IC를 사용하여야 하며

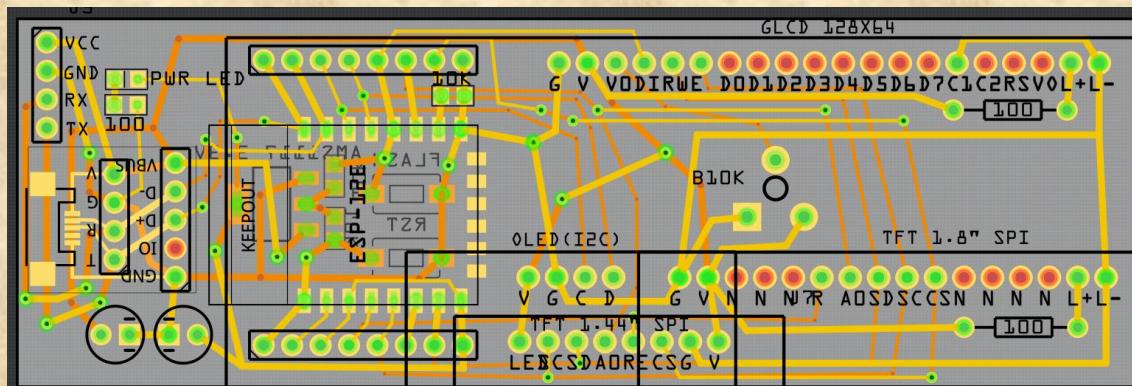
이 경우 현재 사용중인 라이브러리들을 전반적으로 수정하여야 하므로 그러한 추가 기능은 시간을 두고 차츰 업데이트 하도록 하겠습니다.

너무 이 프로젝트에 시간을 할애하여 다른 프로젝트들이 기다리고 있는 중입니다.

기본적인 기능만으로 V1.0을 만들고 이후 보강하여 초기 제가 원했던 기능들을 모두 보완하는 걸로 하겠습니다.

PCB의 설계는 FRITZING을 이용하여 하였습니다.

설계한 PCB는 아래와 같습니다.



디스플레이 장치는

GLCD 128x64

TFT(1.8 인치 , 1.44 인치) - SPI 방식

OLED(0.96 인치) - I2C 방식

입니다.

MCU는 ESP-12를 사용하며 FLASH 버튼과 RESET 버튼을 넣었습니다.

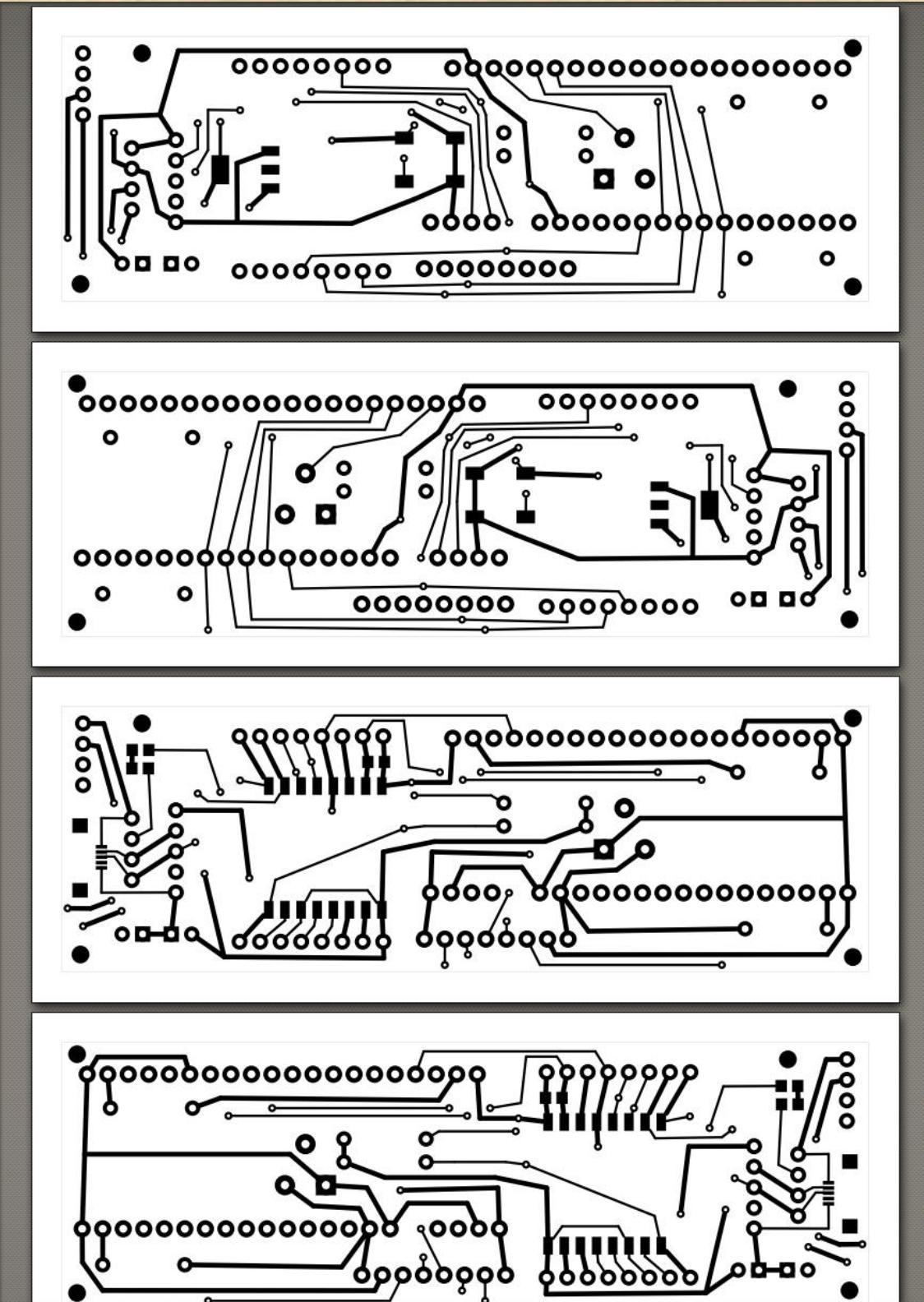
전원은 5V를 입력 받으며 이를 AMS1117-3.3V를 이용하여 ESP-12의 전원으로 공급합니다.

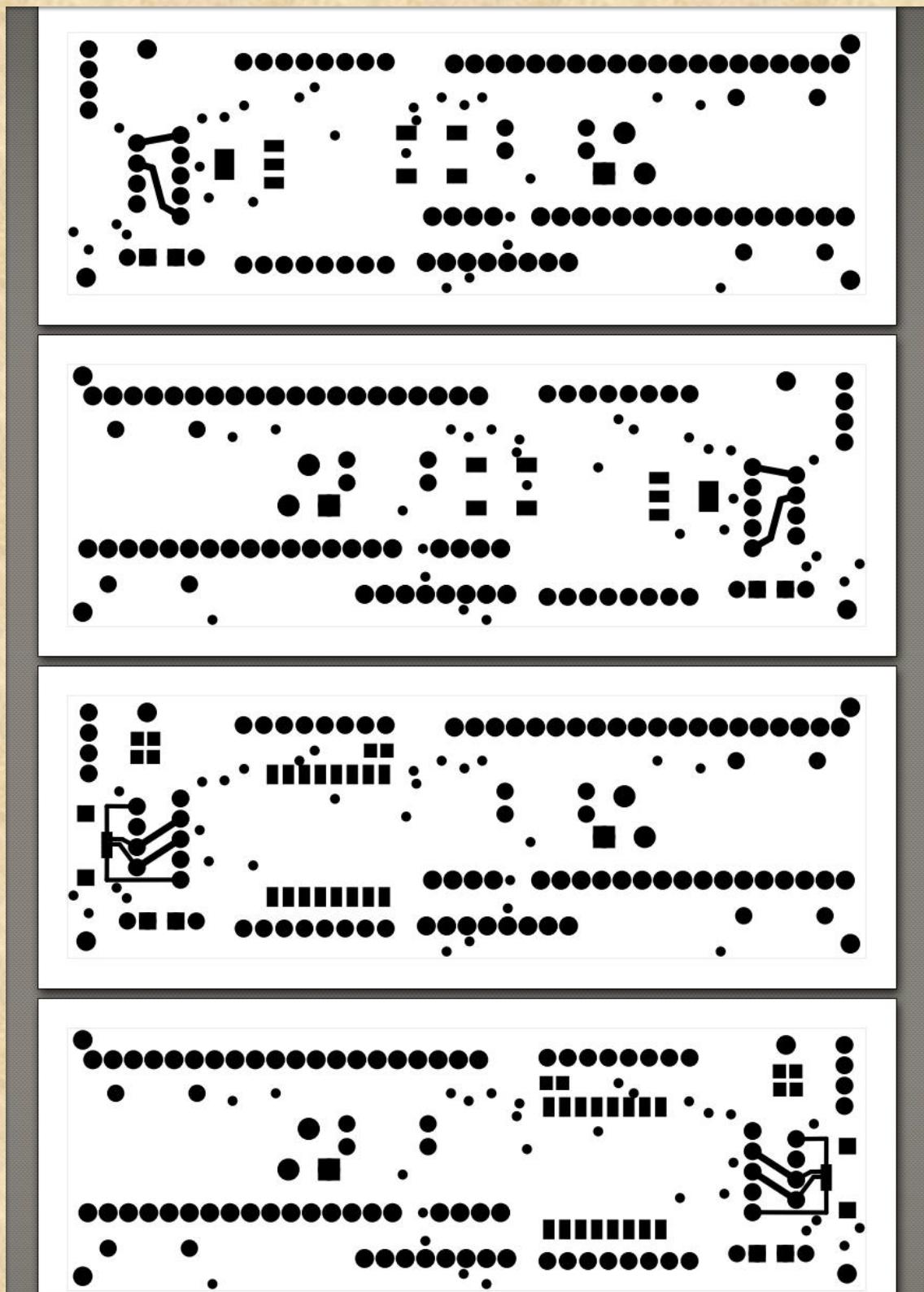
디스플레이 장치(TFT, OLED, GLCD)에는 입력된 5V를 공급합니다.

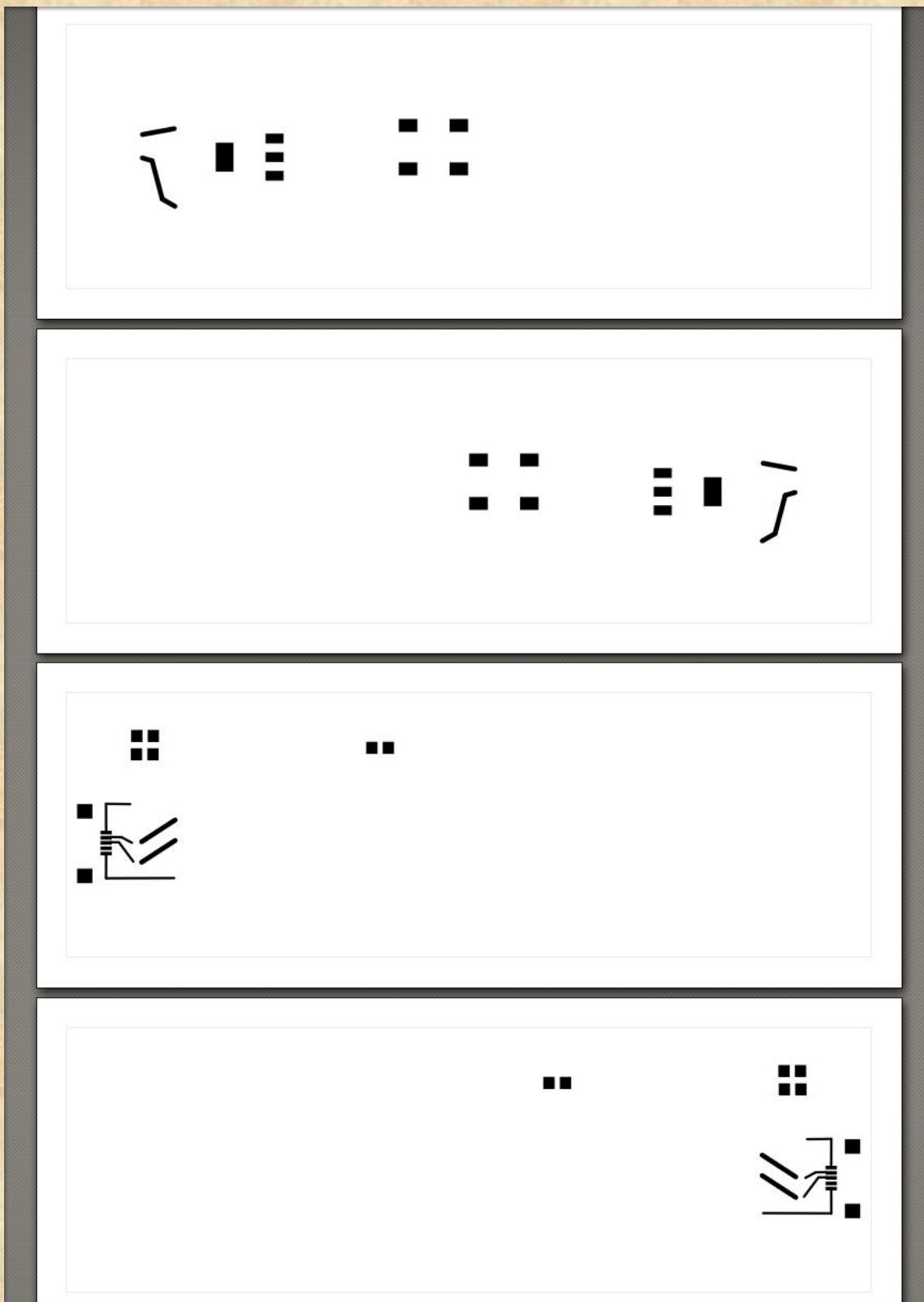
외부와의 통신을 위한 RXD, TXD 단자를 외부로 노출 하였습니다.

설계한 PCB는 아래와 같습니다.

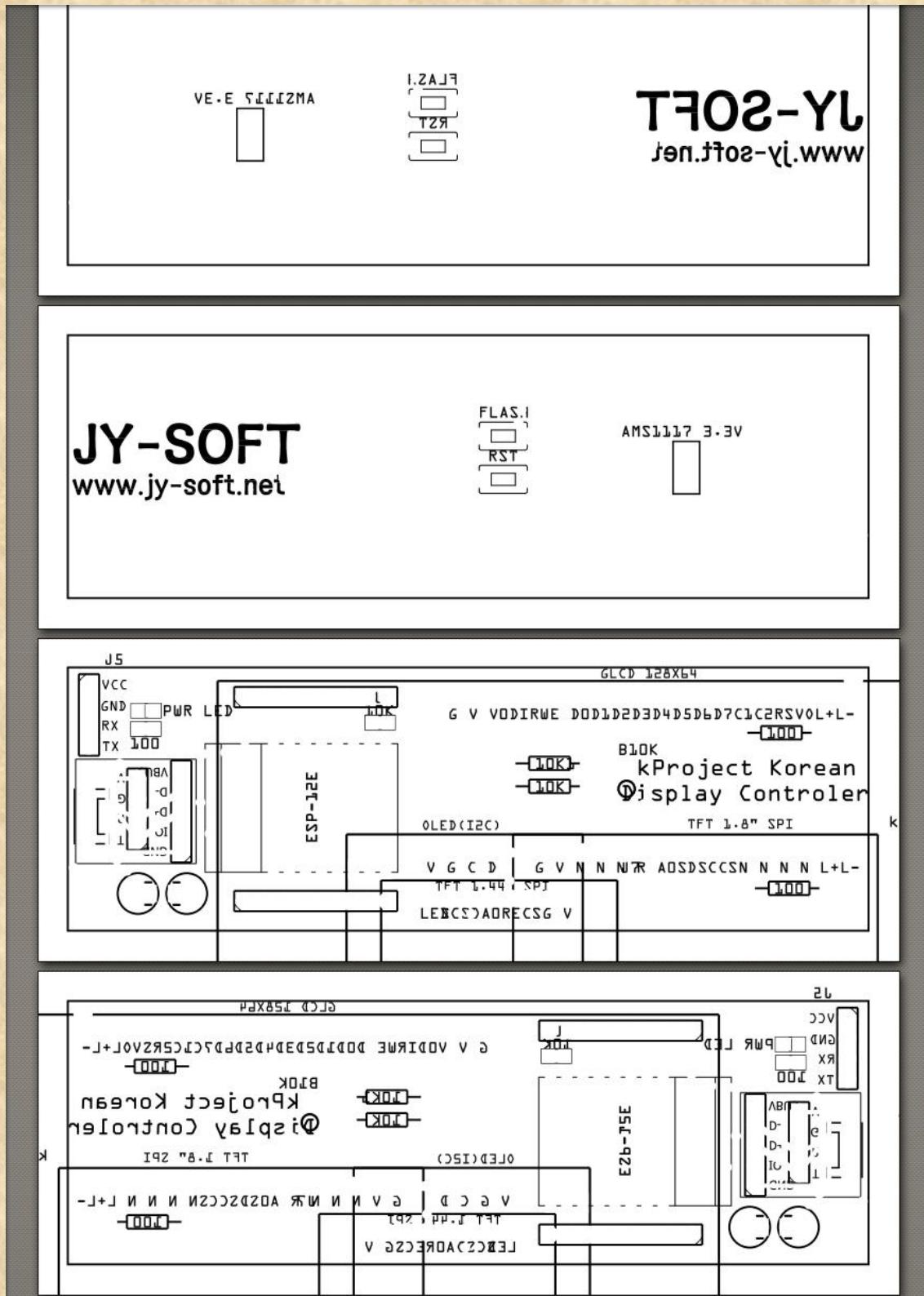
관련 자료 : [korean_display_29](#)







kProject 열정풍향 프로젝트



■ 사용 라이브러리

사용된 라이브러리입니다.

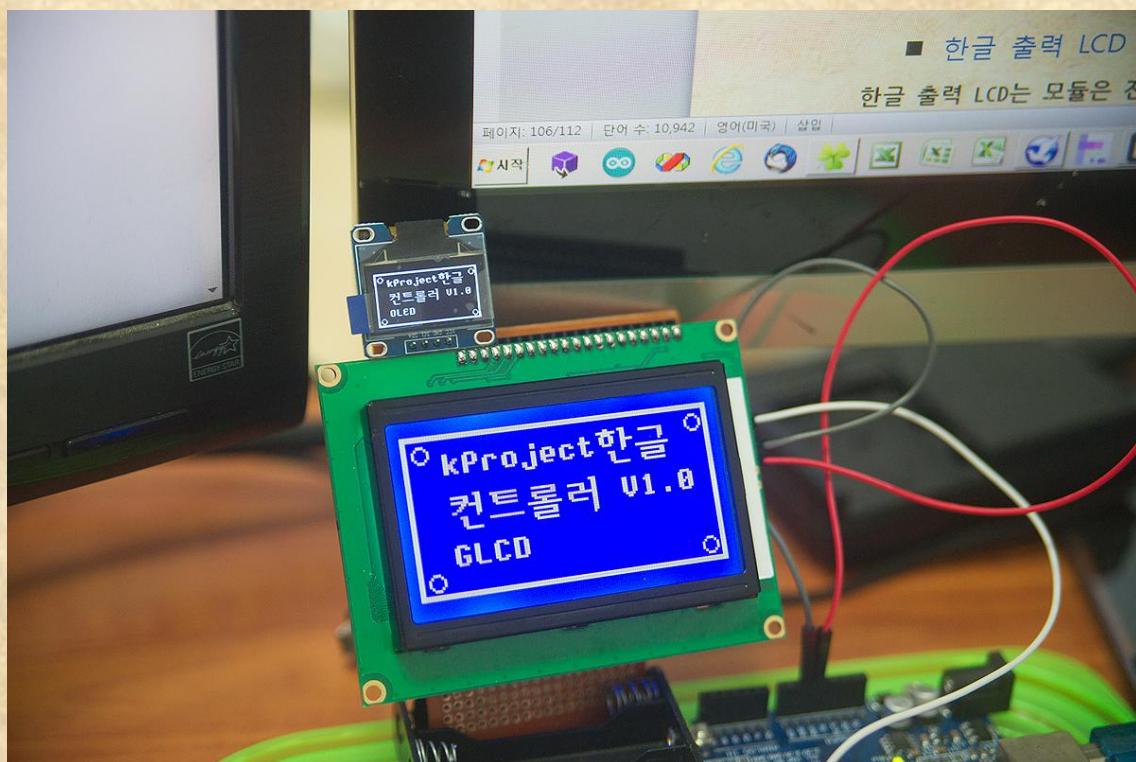
이전까지는 OLED I2C를 구동하기 위하여 adafruit의 라이브러리를 사용
TFT SPI를 구동하기 위하여 adafruit의 라이브러리를 사용
GLCD SPI를 구동하기 위하여 u8g2 라이브러리를 사용하고 있습니다.
U8G2 의를 사용하는 GLCD의 경우 GRAPHIC 함수를 만들었지만
ADAFRUIT 라이브러리를 사용하는 OLED 와 TFT 의 경우 GRAPHIC 함수를 아직
만들지는 못했습니다.

현재까지 사용된 총 3개의 라이브러리중 U8G2 라이브러리는 OLED I2C를
지원하므로 adafruit의 OLED I2C를 제거하고 GLCD에 사용된
U8G2 라이브러리를 사용하도록 통합하였습니다.
U8G2 라이브러리의 경우 단색만 지원하므로 컬러로 구동되는 TFT의 경우
U8G2 라이브러리는 구동이 불가합니다.

TFT의 그래픽 함수는 이후 프로그램 업데이트 하도록 할 예정입니다.

이제 수정된 소스코드는 U8G2를 사용하므로

GLCD 와 OLED 는 그래픽 함수의 사용이 가능합니다.



kProject 열정동향 프로젝트

구동코드

```
1234 1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234  
5 567890  
  
1 #include "kp_korean_display.h"  
2  
3 char test_message4[] = "kProject 한글";  
4 char test_message5[] = "컨트롤러 V1.0";  
5 char test_message6[] = "GLCD";  
6  
7 char test_message7[] = "kProject 한글";  
8 char test_message8[] = "컨트롤러 V1.0";  
9 char test_message9[] = "OLED";  
10  
11 int han_font_index = 1;  
12 int eng_font_index = 1;  
13  
14 void setup()  
15 {  
16     soft_serial.begin(9600);  
17     display_init(1);  
18     display_init(3);  
19 }  
20  
21 void loop()  
22 {  
23     display_set_han_font(han_font_index);  
24     display_set_eng_font(eng_font_index);  
25  
26     display_activate(3);  
27     display_clear();  
28     draw_back();  
29     display_set_cursor(15, 5);  
30     display_print_message(test_message4);  
31     display_set_cursor(15, 23);  
32     display_print_message(test_message5);  
33     display_set_cursor(15, 41);  
34     display_print_message(test_message6);  
35     display_refresh();  
36  
37     display_activate(1);  
38     display_clear();  
39     draw_back();  
40     display_set_cursor(15, 5);  
41     display_print_message(test_message7);  
42     display_set_cursor(15, 23);  
43     display_print_message(test_message8);  
44     display_set_cursor(15, 41);  
45     display_print_message(test_message9);  
46     display_refresh();  
47  
48     han_font_index++;  
49     eng_font_index++;  
50     han_font_index = random(141)+1;  
51     eng_font_index = random(73)+1;  
52     if (han_font_index > 142) han_font_index = 1;  
53     if (eng_font_index > 74) eng_font_index = 1;  
54 }  
55  
56 void draw_back()  
57 {
```

kProject 얼정뚱땅 프로젝트

```
58     diplay_clear();
59     diplay_set_cursor(0, 0);
60     diplay_circle(7, 7, 3);
61     diplay_circle(128-7, 64-7, 3);
62     diplay_circle(128-7, 7, 3);
63     diplay_circle(7, 64-7, 3);
64     diplay_rect(0, 0, 128, 64);
65     diplay_rect(1, 1, 126, 62);
66 }
67
68
```

관련 자료 : korean_diplay_30

■ TFT 모듈 그래픽 가능 추가

OLED와 GLCD의 경우 U8G2라이브러리를 사용하였으므로 동일한 그래픽 함수를 사용하게 되어 그래픽함수가 구현 하였습니다.

하지만 TFT의 경우 adafruit의 라이브러리를 사용하면서 그래픽 함수를 넣지 못했었는데 이번에 입력해 넣었습니다.

이 과정에서 GLCD와 OLED의 경우 단색이므로 버퍼를 별도로 만들어 사용함으로 버퍼에 그림을 미리 그리고(화면에 반영이 되지 않는 상태)

화면 간신 명령을 송신하여 버퍼의 그림을 화면에 반영하면서 화면의 깜빡임이 발생하지 않는 방식을 사용하였습니다.

OLED와 GLCD의 경우 128x64픽셀이기 때문에 버퍼의 사이즈는 128x64비트 즉, $128 \times 64 / 8 = 1024$ Byte로 버퍼 설정이 가능했습니다.

하지만 TFT의 경우 RGB의 데이터를 가지고 있으므로 필요한 Buffer의 사이즈는 $3 \times 128 \times 160 = 61440\text{Byte}$, 즉 61KByte가 필요하게 됩니다.

이는 MCU에 사용하기에는 너무 큰 데이타 입니다.

따라서 TFT의 경우 별도의 BUFFER에 저장하고 갱신(REFRESH) 명령에 의해 화면에 그리는게 아닌

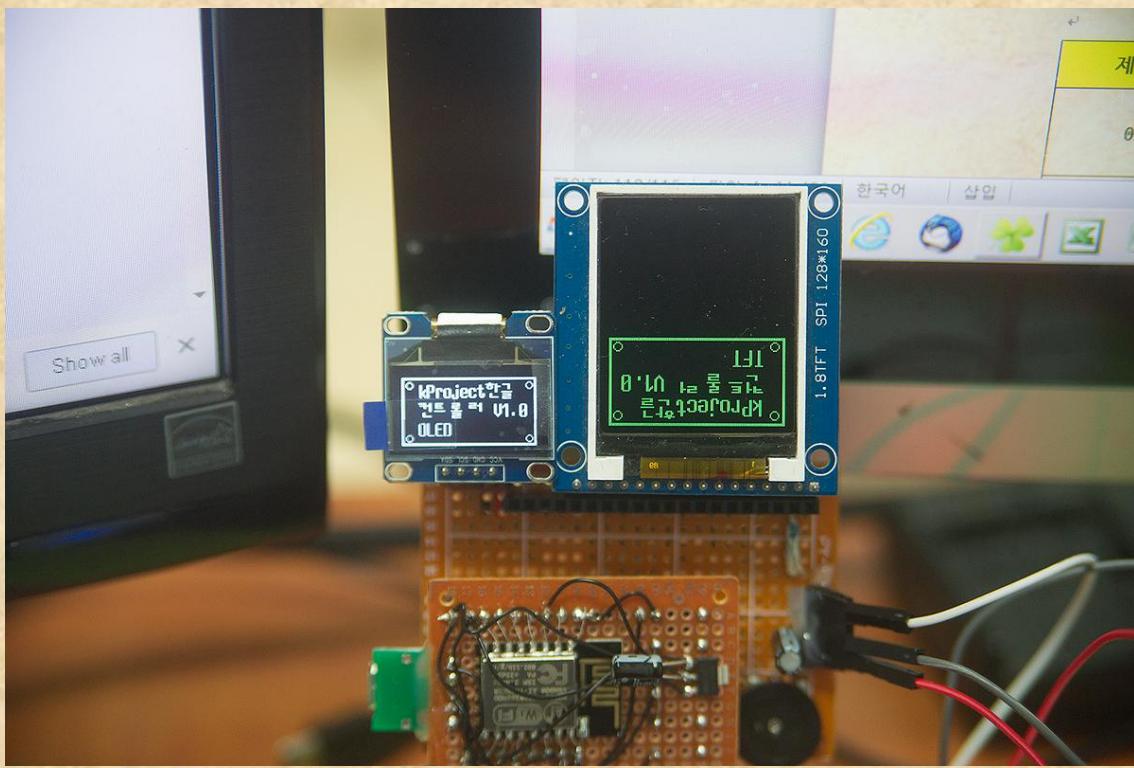
그리는 명령이 입력되면 바로 화면에 반영을 하여 별도의 BUFFER 없이 작동하도록 하였습니다.

관련 자료 : korean display 31

kProject 열정 풍랑 프로젝트

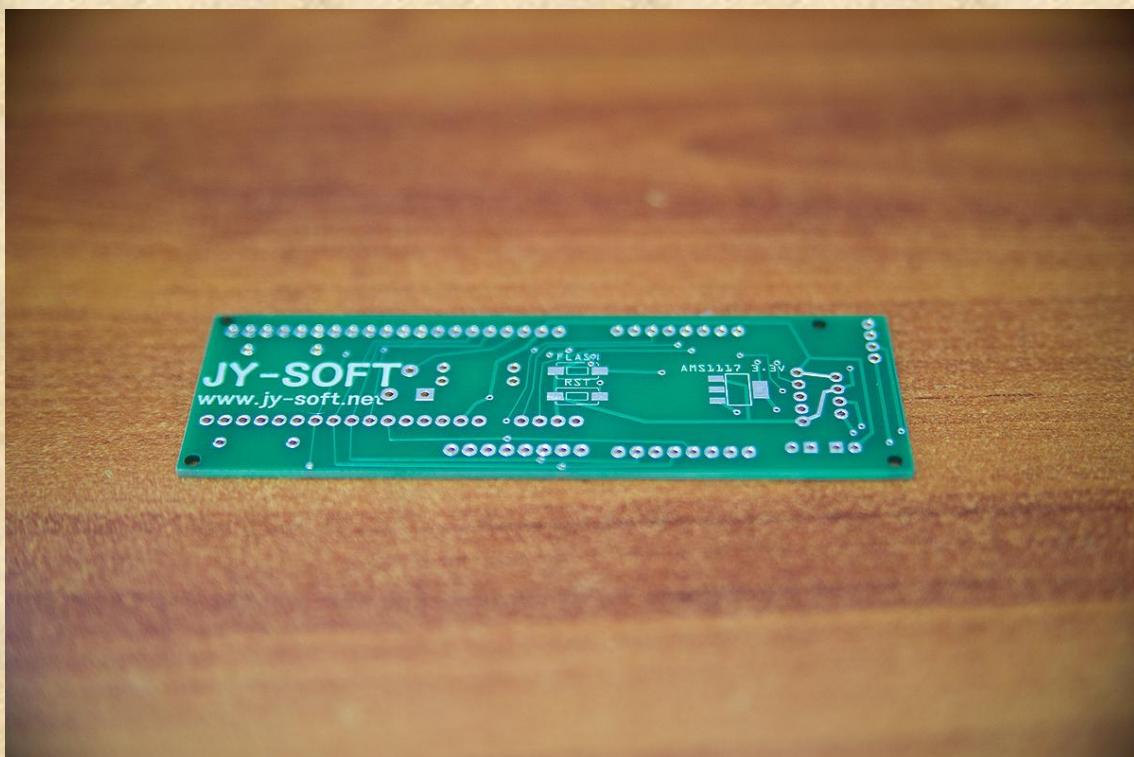
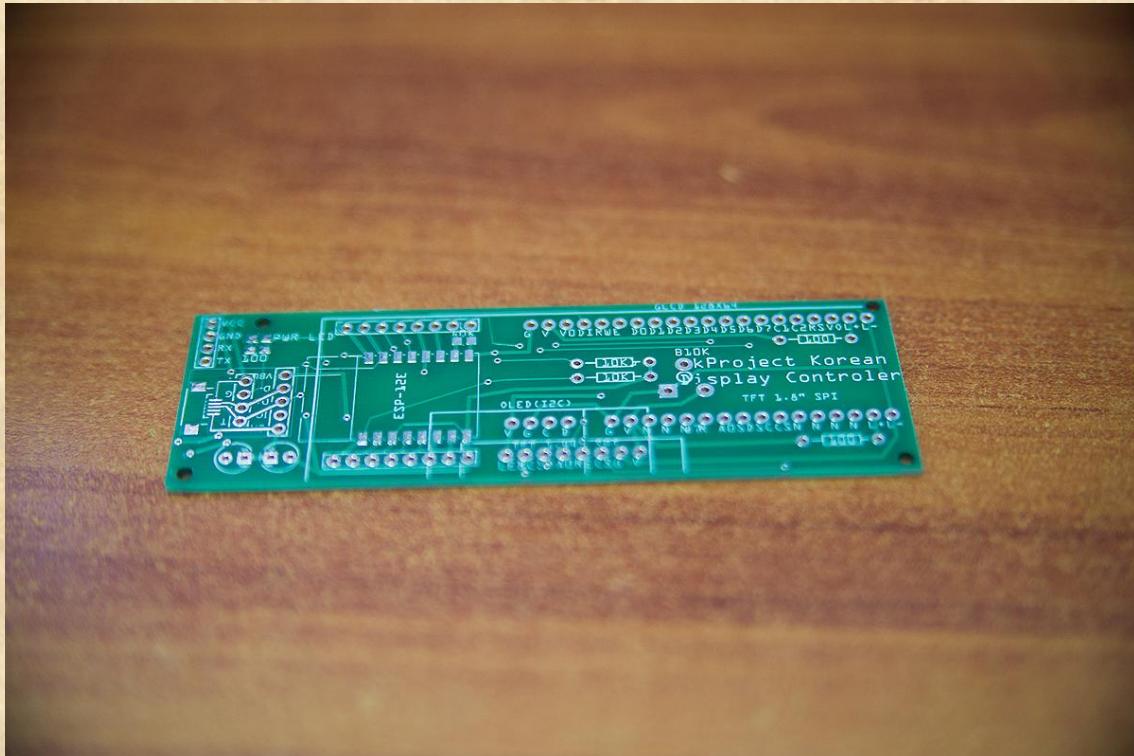
```
17     di_splay_init(1);
18     delay(100);
19     di_splay_init(2);
20     delay(2000);
21 }
22
23 void loop()
24 {
25     di_splay_set_han_font(han_font_index);
26     di_splay_set_eng_font(eng_font_index);
27
28     di_splay_activate(1);
29     di_splay_clear();
30     draw_back();
31     di_splay_set_cursor(15, 5);
32     di_splay_print_message(test_message4);
33     di_splay_set_cursor(15, 23);
34     di_splay_print_message(test_message5);
35     di_splay_set_cursor(15, 41);
36     di_splay_print_message(test_message6);
37     di_splay_refresh();
38
39     di_splay_set_color(di_splay_color_rgb(0, 63, 0)); // r : 0~31 , g : 0~63 , b : 0~31
40     di_splay_activate(2);
41     draw_back();
42     di_splay_set_cursor(15, 5);
43     di_splay_print_message(test_message7);
44     di_splay_set_cursor(15, 23);
45     di_splay_print_message(test_message8);
46     di_splay_set_cursor(15, 41);
47     di_splay_print_message(test_message9);
48
49     delay(500);
50     han_font_index++;
51     eng_font_index++;
52     han_font_index = random(141)+1;
53     eng_font_index = random(73)+1;
54     if ( han_font_index > 142 ) han_font_index = 1;
55     if ( eng_font_index > 74 ) eng_font_index = 1;
56 }
57
58 void draw_back()
59 {
60     di_splay_clear();
61     di_splay_circle(7, 7, 3);
62     di_splay_circle(128-7, 64-7, 3);
63     di_splay_circle(128-7, 7, 3);
64     di_splay_circle(7, 64-7, 3);
65     di_splay_rect(0, 0, 128, 64);
66     di_splay_rect(1, 1, 126, 62);
67 }
68
69 }
```

kProject 열정 풍랑 프로젝트



■ PCB 제작

설계하여 주문한 PCB가 제작이 완료 되었습니다.



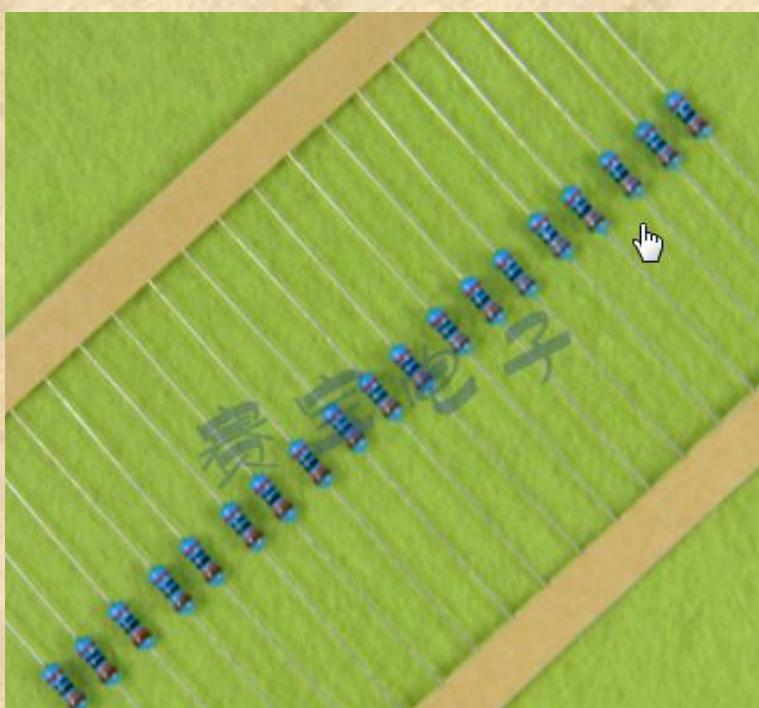
■ 사용 부속 정리

기판에 실장하여야 하는 부속들을 정리해 봤습니다.

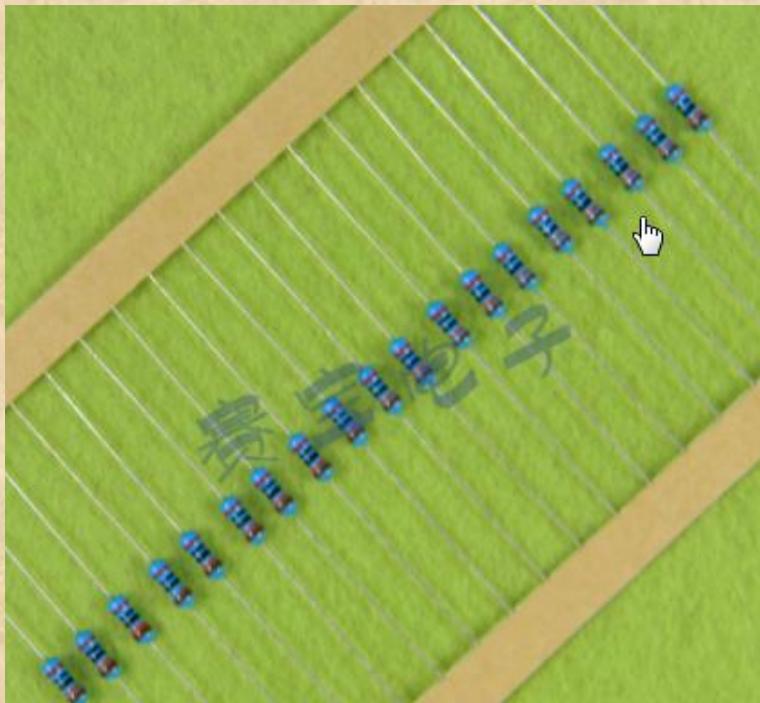
ESP-12 MCU



저항 10K옴 2개



저항 100옴 2개



칩저항 1개 10K옴



AMS1117 3.3V



가변 저항 B10K



버튼 2개



핀헤더 여러개



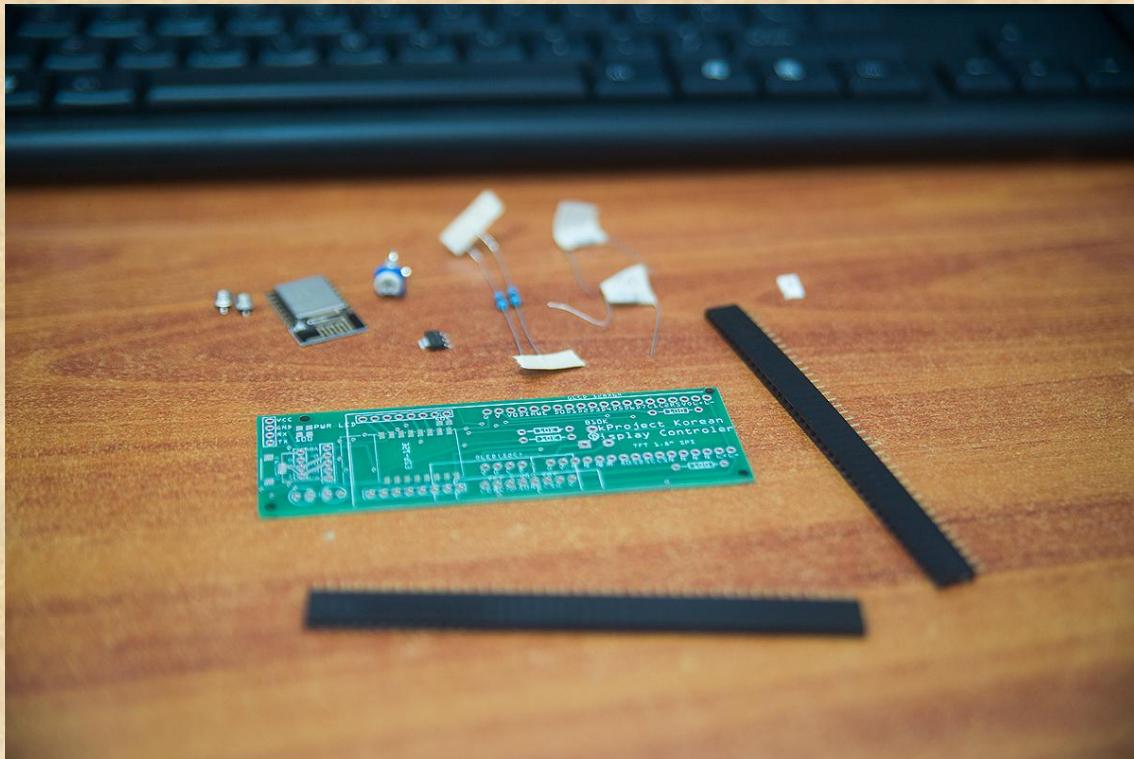
칩저항 1개 100옴(옵션)



칩 LED 1개(옵션)

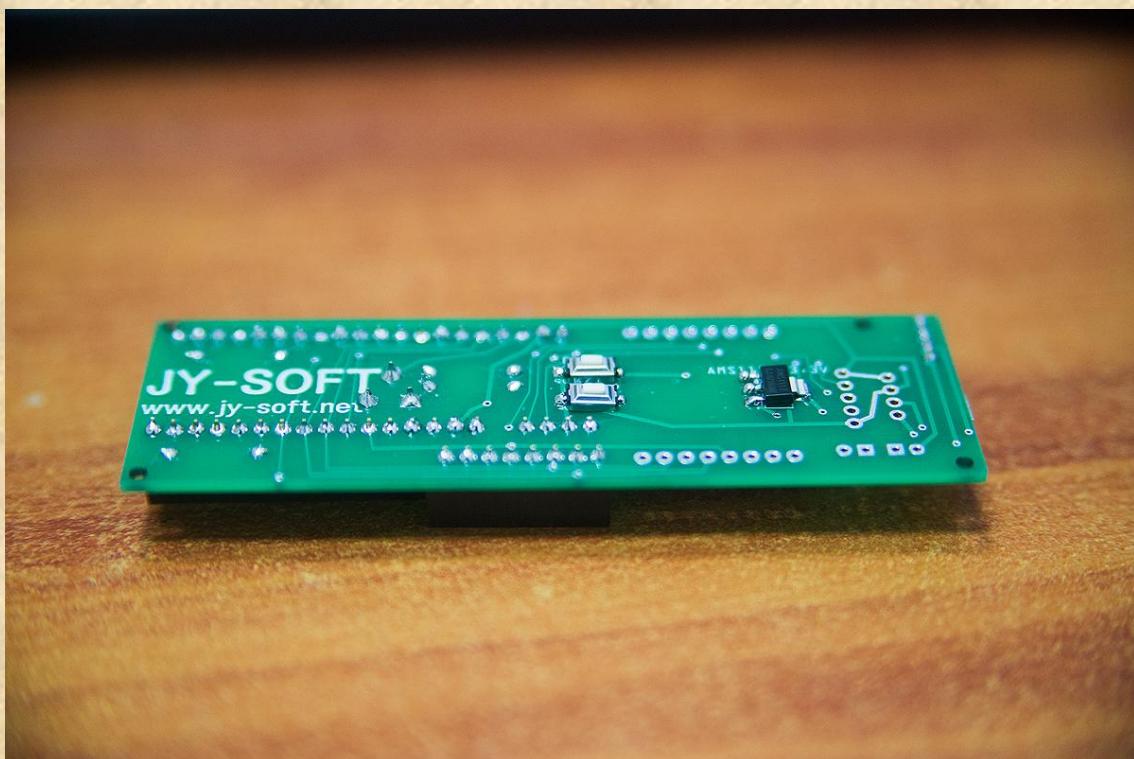
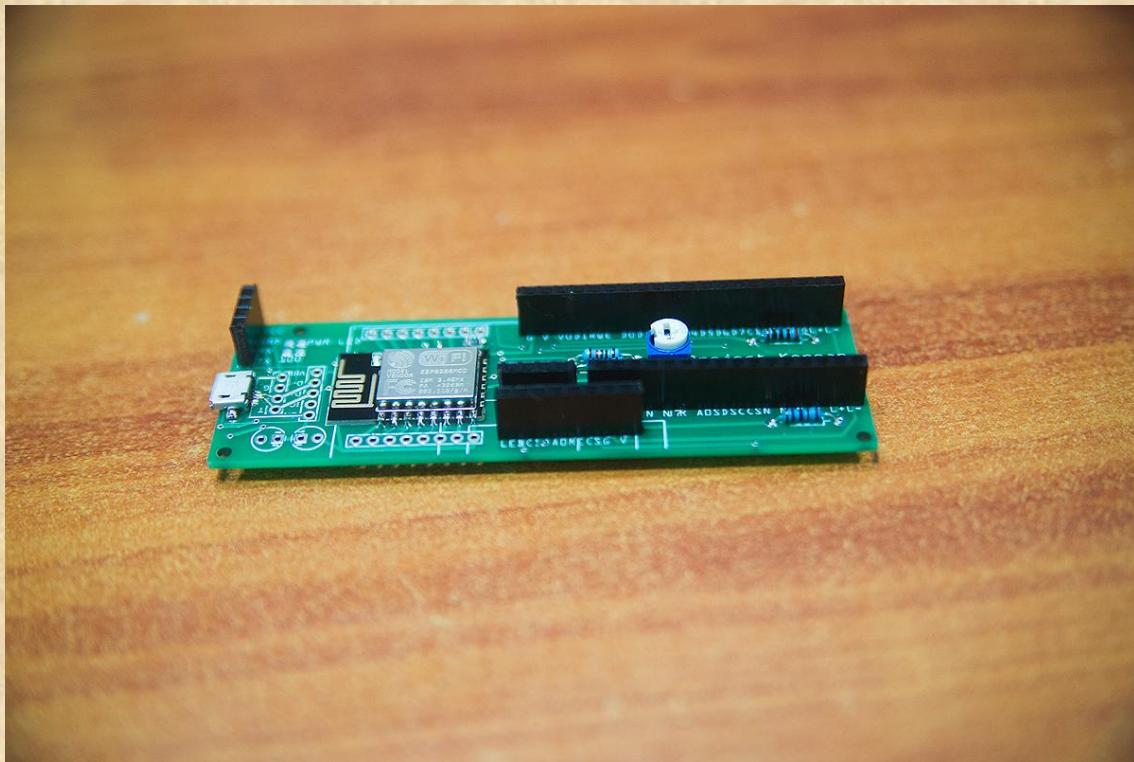


사용될 부속을 모아보면 아래 정도입니다.



kProject 열정 풍향 조호젝트

이제 납땜을 하여 부속을 실장 합니다.
필요한 부속을 모두 실장하면 다음과 같습니다.

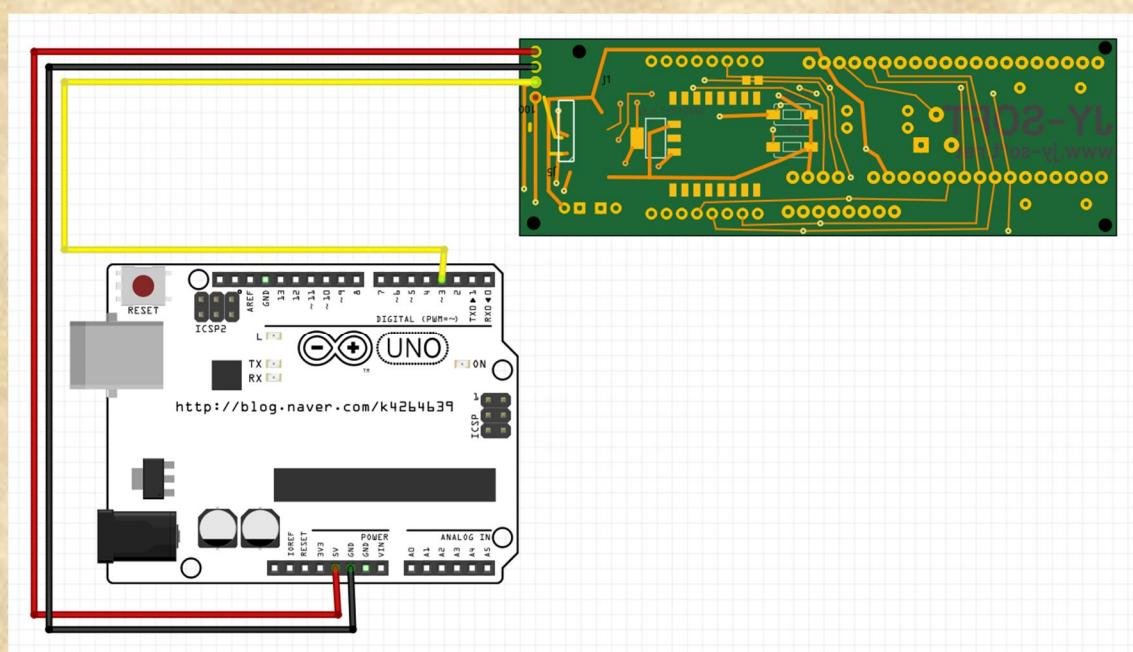


■ 1차 완성

이제 1차 완성이 되었습니다. 초기 생각했던 모든 기능들을 구현하지는 못 했지만 우선 여기서 1차 프로젝트 종료를 하도록 하겠습니다.
언제가 될지 모르겠지만 초기 생각했던 기능을 위한 추가 업데이트는 반드시 하도록 하겠습니다.

PCB 조립 및 납땜이 완성되면 완성된 기판에

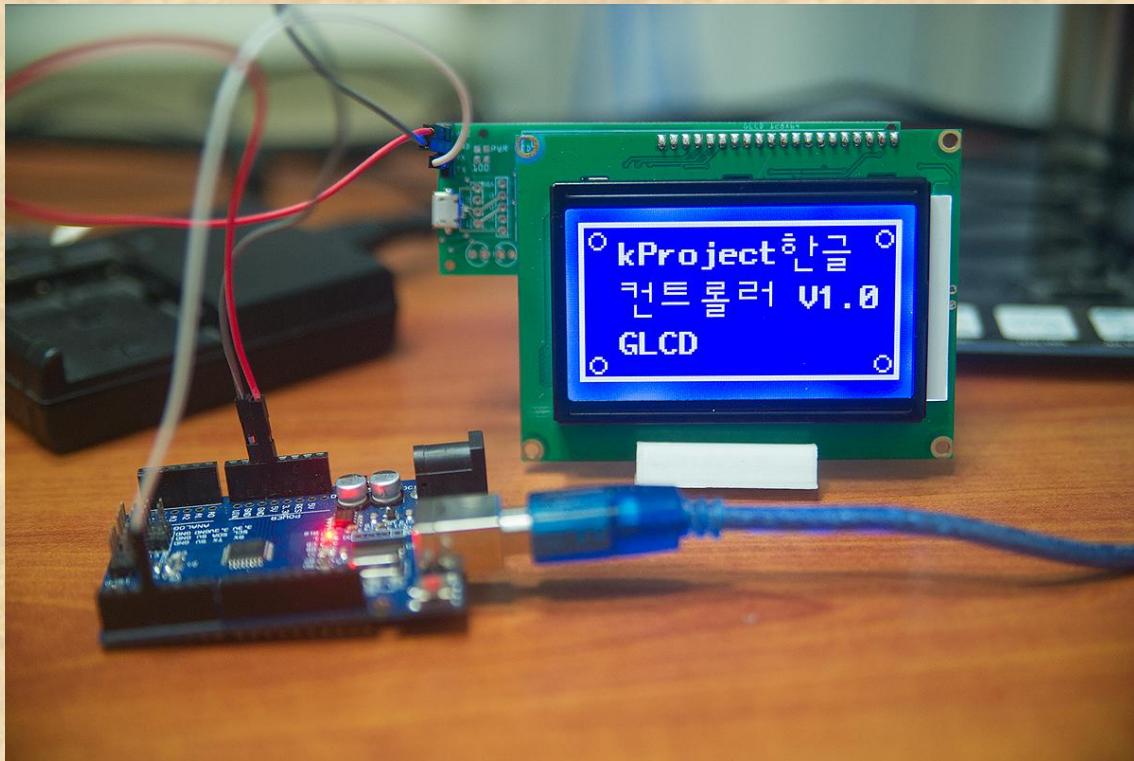
- ESP8266의 SPIFFS 영역에 폰트 파일을 업로딩 합니다. (업로딩 방법은 이전 글을 참조 바랍니다.)
 - 그리고 ESP8266에 펌웨어를 업로딩 합니다.
 - 아두이노 UNO를 이용하여 컨트롤러를 구동하기 위한 회로도는 아래와 같습니다.



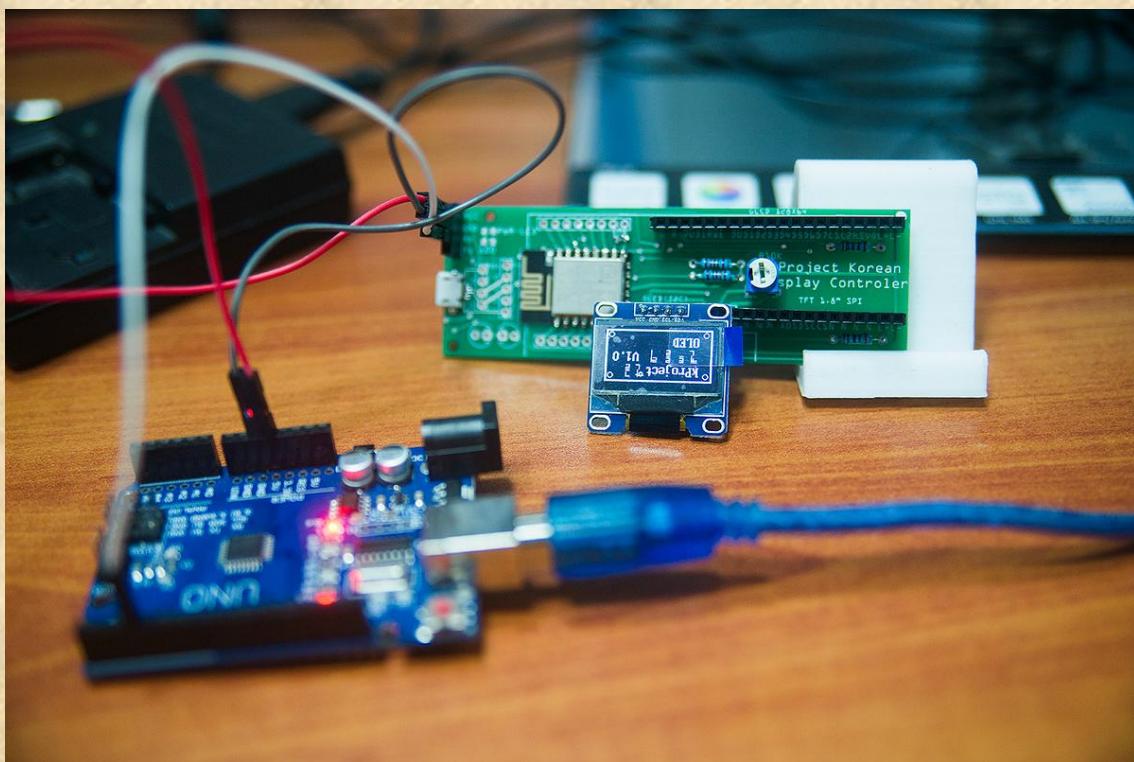
- 다음 사진은 각 디스플레이 장치에 작동하는 모습입니다.

kProject 열정 풍랑 프로젝트

GLCD(128x64)

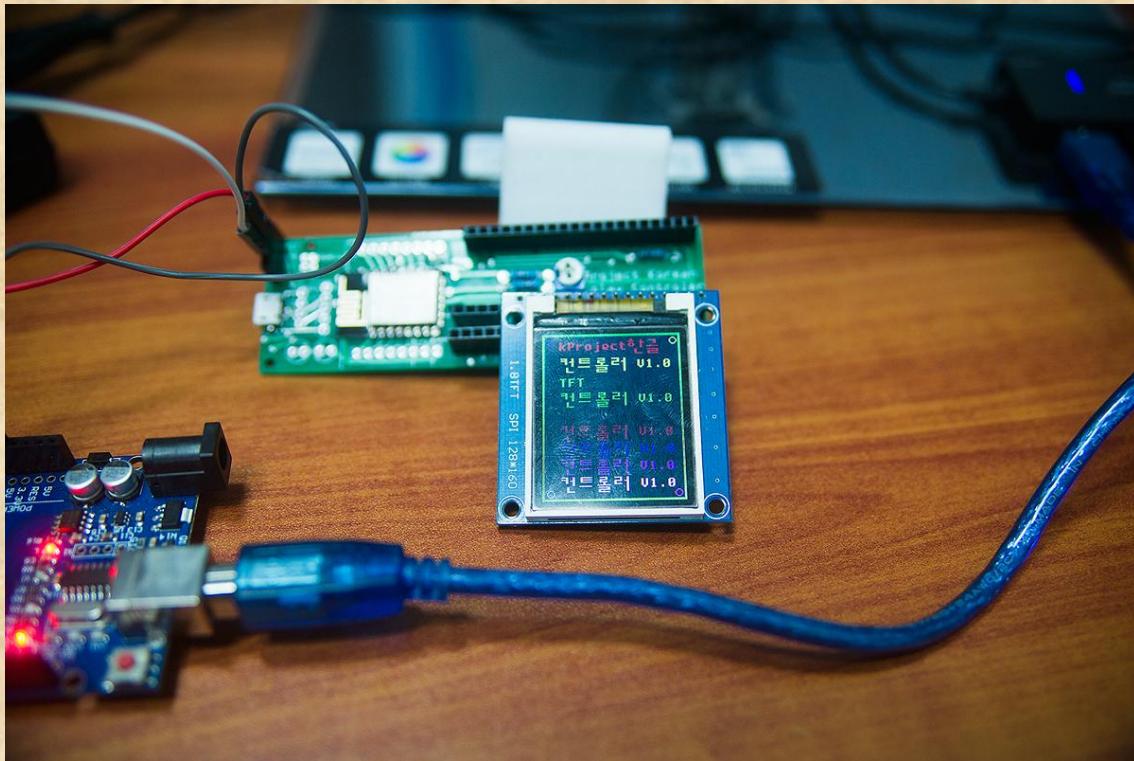


OLED(128x64)



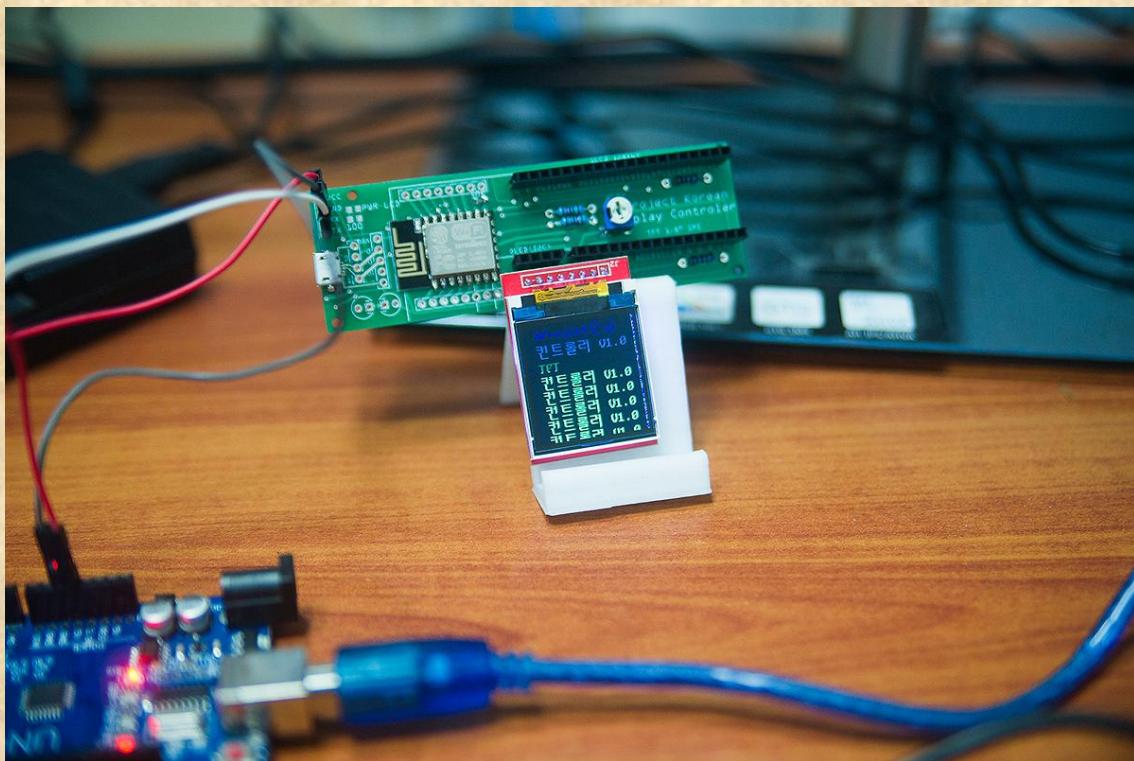
kProject 열정 풍랑 프로젝트

TFT(128x160)



TFT(128x128)

이건 색상이 이상하게 나오는 군요



그래서 다시 확인해 보니 이 LCD 모듈은 ST7735 칩을 사용한게 아닌
ILI9163C 칩을 사용한 모듈입니다.

이번 글을 마지막으로 1차 제작을 마무리 하려고 했는데..
한번더 글을 써야 할 듯 합니다...

■ ILI9163C 칩을 위한 수정

앞서 발생한 |LI9163C칩의 경우 색상의 출력이 잘못 나오는 부분이 있었습니다.

이 수정을 위하여 ILLI9163라이브러리를 추가하여 수정하여 보았으나 기존 ST7735라이브러리와 다르게 속도가 느리게 작동하는 현상이 발생하였습니다.

검색해 보면 ILI9163과 STT7735의 경우 명령셋이 거의 동일하다고 하고 실제 구동시에도 색상의 문제만 아니면 정상 작동하고 있었습니다.

그래서 색상의 차이를 보니 ILI9163과 ST7735의 R과 B가 서로 바뀌어 있는 걸 확인하였습니다.

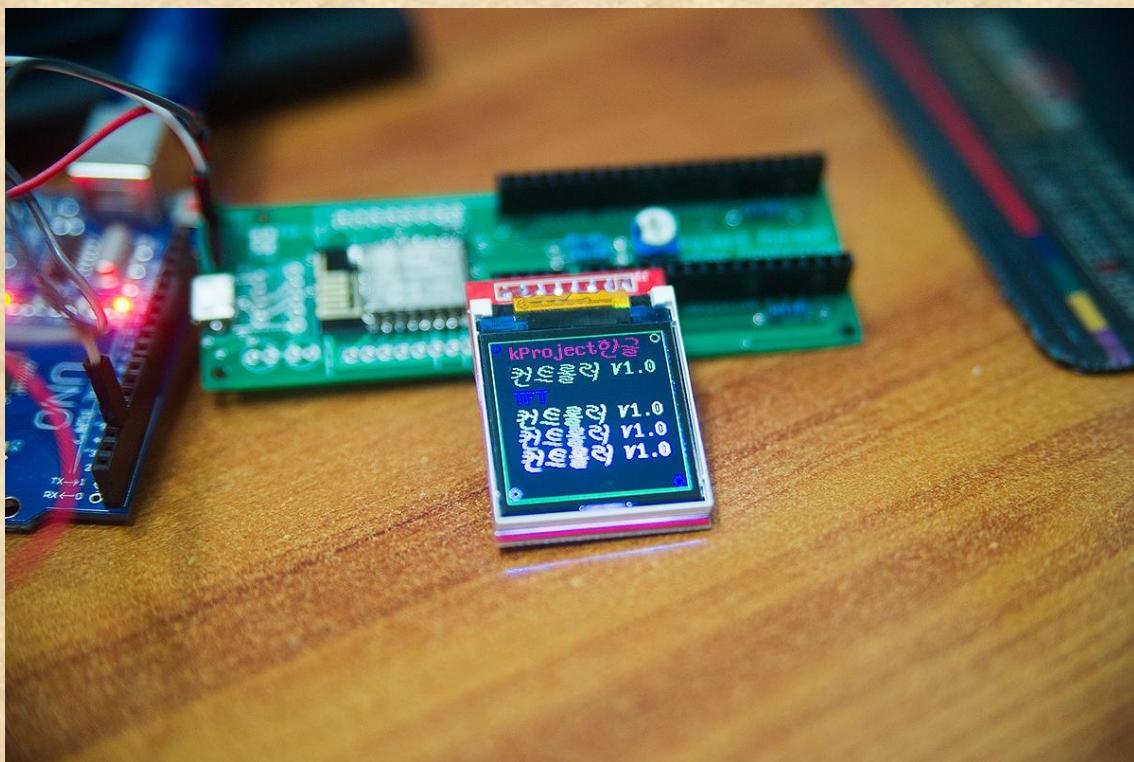
따라서 새로운 ILLI9163 라이브러리를 추가하지 않고 기존 ST7735 라이브러리에서 ILLI9163일 경우 색상만 변경하여 사용하는 것으로 수정하였습니다.

kProject 열정동향 프로젝트

```
33     di_splay_print_message(test_message8);
34     di_splay_set_cursor(15, 41);
35     di_splay_set_color(di_splay_color_rgb(0, 0, 31)); // r : 0~31 , g : 0~63 , b : 0~31
36     di_splay_print_message(test_message9);
37     di_splay_set_cursor(15, 57);
38     di_splay_set_color(di_splay_color_rgb(31, 63, 0)); // r : 0~31 , g : 0~63 , b : 0~31
39     di_splay_print_message(test_message8);
40     di_splay_set_cursor(15, 73);
41     di_splay_set_color(di_splay_color_rgb(0, 63, 31)); // r : 0~31 , g : 0~63 , b : 0~31
42     di_splay_print_message(test_message8);
43     di_splay_set_cursor(15, 89);
44     di_splay_set_color(di_splay_color_rgb(31, 63, 31)); // r : 0~31 , g : 0~63 , b : 0~31
45     di_splay_print_message(test_message8);

46
47     delay(1000);
48     han_font_index++;
49     eng_font_index++;
50     han_font_index = random(141)+1;
51     eng_font_index = random(73)+1;
52     if ( han_font_index > 142 ) han_font_index = 1;
53     if ( eng_font_index > 74 ) eng_font_index = 1;
54 }

55
56 void draw_back()
57 {
58     di_splay_clear();
59     di_splay_set_color(di_splay_color_rgb(0, 0, 31)); // r : 0~31 , g : 0~63 , b : 0~31
60     di_splay_circle(7, 7, 3);
61     di_splay_set_color(di_splay_color_rgb(31, 0, 31)); // r : 0~31 , g : 0~63 , b : 0~31
62     di_splay_circle(128-7, 128-7, 3);
63     di_splay_set_color(di_splay_color_rgb(31, 63, 0)); // r : 0~31 , g : 0~63 , b : 0~31
64     di_splay_circle(128-7, 7, 3);
65     di_splay_set_color(di_splay_color_rgb(0, 63, 31)); // r : 0~31 , g : 0~63 , b : 0~31
66     di_splay_circle(7, 128-7, 3);
67     di_splay_set_color(di_splay_color_rgb(0, 63, 0)); // r : 0~31 , g : 0~63 , b : 0~31
68     di_splay_rect(0, 0, 128, 128);
69     di_splay_rect(2, 2, 126, 126);
70 }
71
72 }
```



정상적으로 색상이 출력이 됩니다.

이로서 한글 디스플레이 콘트롤러의 1 차 제작을 완료 합니다.
감사합니다.

APPENDIX

■ 기능 요약

kProject 한글 출력장치 컨트롤러 기능 요약입니다.

1. 기존 디스플레이 장치에 한글을 표현할 수 있습니다.
해외에서 만들어진 대부분의 장치에 한글의 표현이 어렵습니다.
이 모듈은 좀더 쉽게 디스플레이 장치에 한글을 출력 할 수 있습니다.
2. 자체 한글 폰트 142개 내장 , 자체 영문 폰트 74개 내장
자체 내부에 많은 양의 한글 및 영문 폰트를 내장하여 다양한 폰트를 사용하실 수 있습니다.
3. 시리얼 통신을 이용한 디스플레이 장치 구동
시리얼 통신(BAUD RATE 9600)으로 아두이노와 같은 Master 모듈과 통신을 하므로 Master 모듈의 디스플레이 장치 구동을 위한 부하가 걸리지 않습니다.
4. 다양한 디스플레이 장치 지원(OLED-I2C, GLCD12864, TFT128160)
다양한 디스플레이 장치를 지원합니다.
지원하는 디스플레이 장치는 OLED 모듈, GLCD 모듈, TFT 모듈을 지원합니다.
또한 OLED와 TFT모듈의 동시 구동, OLED와 GLCD모듈의 동시 구동이 가능합니다. GLCD와 TFT모듈의 동시 구동은 지원하지 않습니다.
5. 1개 통신핀을 이용한 간단한 연결
MASTER모듈과 단, 1개의 핀으로 연결하여 구동할 수 있으므로 기존 복잡한 핀 배열을 사용할 필요가 없으며
MASTER모듈의 GPIO 및 입출력 핀을 좀더 많이 확보 할 수 있습니다.
MASTER모듈은 Serial 통신의 TX핀만을 사용합니다.
6. 빠른 출력(아두이노 UNO대비)
MASTER모듈이 느릴 경우 화면 출력은 많은 부하가 걸리게 되며 화면의 표시는 매우 느려지게 됩니다.
하지만 이 모듈은 32bit 160MHz Processor을 사용하므로 좀더 빠르게 화면을 출력 할 수 있습니다.

■ 한글 출력 LCD 모듈 시리얼 통신 명령

한글 출력 LCD는 모듈은 전달된 명령을 기본 문자열로 인식하여 화면에 영문 및 한글을 출력 합니다.

하지만 전달된 시리얼 데이터가 ASCII 코드 0x01일 경우 이후의 일정 데이터를 명령 코드 및 데이터 자료로 사용을 하게 됩니다.

각 제어코드에 대한 설명은 아래 자료를 참조하시기 바랍니다.

제어코드	명령코드	데이터 1	데이터 2	데이터 3	데이터 4
0x01	0x01 화면 초기화	-	-	-	-
0x01	0x02 Cursor 이동	x 좌표 상위바이트	x 좌표 하위 바이트	y 좌표 상위 바이트	y 좌표 하위 바이트
0x01	0x03 한글 폰트 설정	한글폰트번호 (1~142)	-	-	-
0x01	0x04 영문 폰트 설정	영문폰트번호 (1~74)	-	-	-
0x01	0x05 장치 초기화	디스플레이 장치 1 : OLED I2C 2 : TFT LCD SPI 3: GLCD 4 : TFT(ILI9163)	-	-	-
0x01	0x06 장치 활성화	디스플레이 장치 0 : 활성화 되지 않음. 1 : OLED I2C 2 : TFT LCD SPI 3 : GLCD 4 : TFT(ILI9163)	-	-	-
0x01	0x07 색상 설정	색상 상위바이트	색상 하위바이트	-	-

kProject 얼굴인식 프로젝트

0x01	0x08 화면 갱신 (GLCD, OLED Onl y)	-	-	-	-
0x01	0x09 Draw Point	-	-	-	-
0x01	0x10 Draw Line	x1 좌표 상위바이트	x1 좌표 하위 바이트	y1 좌표 상위 바이트	y1 좌표 하위 바이트
		x2 좌표 상위바이트	x2 좌표 하위 바이트	y2 좌표 상위 바이트	y2 좌표 하위 바이트
0x01	0x11 Draw Circle	x 좌표 상위바이트	x 좌표 하위 바이트	y 좌표 상위 바이트	y 좌표 하위 바이트
		원의 반지름 상위바이트	원의 반지름 하위바이트	-	-
0x01	0x12 Draw Rect.	x 좌표 상위바이트	x 좌표 하위 바이트	y 좌표 상위 바이트	y 좌표 하위 바이트
		Width 상위바이트	Width 하위 바이트	height	height
0x01	0x13 Draw Ellipse	x 좌표 상위바이트	x 좌표 하위 바이트	y 좌표 상위 바이트	y 좌표 하위 바이트
		rx 상위바이트	rx 하위 바이트	ry	ry

■ 한글 폰트 리스트

- 001: 나랏말싸미|등|국|에|달|아|문|자|와|로|서|르|사|맛|디|아|니|할|쌔|이|린|전|
002: 나랏말싸미|등|국|에|달|아|문|자|와|로|서|르|사|맛|디|아|니|할|쌔|이|린|전|
003: 나랏말싸미|등|국|에|달|아|문|자|와|로|서|르|사|맛|디|아|니|할|쌔|이|린|전|
004: 나랏말싸미|등|국|에|달|아|문|자|와|로|서|르|사|맛|디|아|니|할|쌔|이|린|전|
005: 나랏말싸미|등|국|에|달|아|문|자|와|로|서|르|사|맛|디|아|니|할|쌔|이|린|전|
006: 나랏말싸미|등|국|에|달|아|문|자|와|로|서|르|사|맛|디|아|니|할|쌔|이|린|전|
007: 나랏말싸미|등|국|에|달|아|문|자|와|로|서|르|사|맛|디|아|니|할|쌔|이|린|전|
008: 나랏말싸미|등|국|에|달|아|문|자|와|로|서|르|사|맛|디|아|니|할|쌔|이|린|전|
009: 나랏말싸미|등|국|에|달|아|문|자|와|로|서|르|사|맛|디|아|니|할|쌔|이|린|전|
010: 나랏말싸미|등|국|에|달|아|문|자|와|로|서|르|사|맛|디|아|니|할|쌔|이|린|전|
011: 나랏말싸미|등|국|에|달|아|문|자|와|로|서|르|사|맛|디|아|니|할|쌔|이|린|전|
012: 나랏말싸미|등|국|에|달|아|문|자|와|로|서|르|사|맛|디|아|니|할|쌔|이|린|전|
013: 나랏말싸미|등|국|에|달|아|문|자|와|로|서|르|사|맛|디|아|니|할|쌔|이|린|전|
014: 나랏말싸미|등|국|에|달|아|문|자|와|로|서|르|사|맛|디|아|니|할|쌔|이|린|전|
015: 나랏말싸미|등|국|에|달|아|문|자|와|로|서|르|사|맛|디|아|니|할|쌔|이|린|전|
016: 나 랏 말 싸 미 등 국 에 달 아 문 자 와 로 서 르 사 맛 디 아 니 할 쟤 이 린 전
017: 나랏말싸미|등|국|에|달|아|문|자|와|로|서|르|사|맛|디|아|니|할|쌔|이|린|전|
018: 나랏말싸미|등|국|에|달|아|문|자|와|로|서|르|사|맛|디|아|니|할|쌔|이|린|전|
019: 나랏말싸미|등|국|에|달|아|문|자|와|로|서|르|사|맛|디|아|니|할|쌔|이|린|전|
020: 나랏말싸미|등|국|에|달|아|문|자|와|로|서|르|사|맛|디|아|니|할|쌔|이|린|전|

- 021: 나랏말싸미등록에달아문자와로서르사맛디아니 할쌔이런전
022: 나랏말싸미등록에달아문자와로서르사맛디아니 할쌔이런전
023: 나랏말싸미등록에달아문자와로서르사맛디아니 할쌔이런전
024: 나랏말싸미등록에달아문자와로서르사맛디아니 할쌔이런전
025: 나랏말싸미등록에달아문자와로서르사맛디아니 할쌔이런전
026: 나랏말싸미등록에달아문자와로서르사맛디아니 할쌔이런전
027: 나랏말싸미등록에달아문자와도서드사맛디아니 할쌔이런전
028: 나랏말싸미등록에달아문자와로서르사맛디아니 할쌔이런전
029: 나랏말싸미등록에달아문자와로서르사맛디아니 할쌔이런전
030: 나랏말싸미등록에달아문자와로서르사맛디아니 할쌔이런전
031: 나랏말싸미등록에달아문자와로서르사맛디아니 할쌔이런전
032: 나랏말싸미등록에달아문자와로서르사맛디아니 할쌔이런전
033: 나랏말싸미등록에달아문자와로서르사맛디아니 할쌔이런전
034: 나랏말싸미등록에달아문자와로서르사맛디아니 할쌔이런전
035: 나랏말싸미등록에달아문자와로서르사맛디아니 할쌔이런전
036: 나랏말싸미등록에달아문자와로서르사맛디아니 할쌔이런전
037: 나랏말싸미등록에달아문자와로서르사맛디아니 할쌔이런전
038: 나랏말싸미등록에달아문자와로서르사맛디아니 할쌔이런전
039: 나랏말싸미등록에달아문자와로서르사맛디아니 할쌔이런전
040: 나랏말싸미등록에달아문자와로서르사맛디아니 할쌔이런전

- 121: 나랏말싸미 등 곡에 달아 문자와로서 르사 맛디 아니 할 쐐이 런전
122: 나랏말싸미 등 곡에 달아 문자와로서 르사 맛디 아니 할 쐐이 런전
123: 나랏말싸미 등 곡에 달아 문자와로서 르사 맛디 아니 할 쐐이 런전
124: 나랏말싸미 등 곡에 달아 문자와로서 르사 맛디 아니 할 쐐이 런전
125: 나랏말싸미 등 곡에 달아 문자와로서 르사 맛디 아니 할 쐐이 런전
126: 나랏말싸미 등 곡에 달아 문자와로서 르사 맛디 아니 할 쐐이 런전
127: 나랏말싸미 등 곡에 달아 문자와로서 르사 맛디 아니 할 쐐이 런전
128: 나랏말싸미 등 곡에 달아 문자와로서 르사 맛디 아니 할 쐐이 런전
129: 나랏말싸미 등 곡에 달아 문자와로서 르사 맛디 아니 할 쐐이 런전
130: 나랏말싸미 등 곡에 달아 문자와로서 르사 맛디 아니 할 쐐이 런전
131: 나랏말싸미 등 곡에 달아 문자와로서 르사 맛디 아니 할 쐐이 런전
132: 나랏말싸미 등 곡에 달아 문자와로서 르사 맛디 아니 할 쐐이 런전
133: 나랏말싸미 등 곡에 달아 문자와로서 르사 맛디 아니 할 쐐이 런전
134: 나랏말싸미 등 곡에 달아 문자와로서 르사 맛디 아니 할 쐐이 런전
135: 나랏말싸미 등 곡에 달아 문자와로서 르사 맛디 아니 할 쐐이 런전
136: 나랏말싸미 등 곡에 달아 문자와로서 르사 맛디 아니 할 쐐이 런전
137: 나랏말싸미 등 곡에 달아 문자와로서 르사 맛디 아니 할 쐐이 런전
138: 나랏말싸미 등 곡에 달아 문자와로서 르사 맛디 아니 할 쐐이 런전
139: 나랏말싸미 등 곡에 달아 문자와로서 르사 맛디 아니 할 쐐이 런전
140: 나랏말싸미 등 곡에 달아 문자와로서 르사 맛디 아니 할 쐐이 런전

141: 나랏말싸미 등 곡에 달아 문자와 압으로 서르사 맛디 아니 할 쐐이 런전
142: 나랏말싸미 등 곡에 달아 문자와 압으로 서르사 맛디 아니 할 쐐이 런전

■ 영문 폰트 리스트

001: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
002: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
003: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
004: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
005: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
006: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
007: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
008: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
009: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
010: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
011: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
012: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
013: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
014: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
015: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
016: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
017: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
018: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
019: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
020: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/

021: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
022: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
023: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
024: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
025: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
026: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
027: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
028: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
029: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
030: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
031: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
032: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
033: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
034: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
035: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
036: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
037: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
038: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
039: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
040: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/

041: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
 042: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
 043: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
 044: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
 045: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
 046: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
 047: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
 048: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
 049: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
 050: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
 051: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
 052: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
 053: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
 054: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
 055: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
 056: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
 057: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
 058: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
 059: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
 060: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/

061: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
 062: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
 063: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
 064: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
 065: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
 066: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
 067: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
 068: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
 069: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
 070: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
 071: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
 072: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
 073: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/
 074: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890~!@#\$%^&*()-=+/