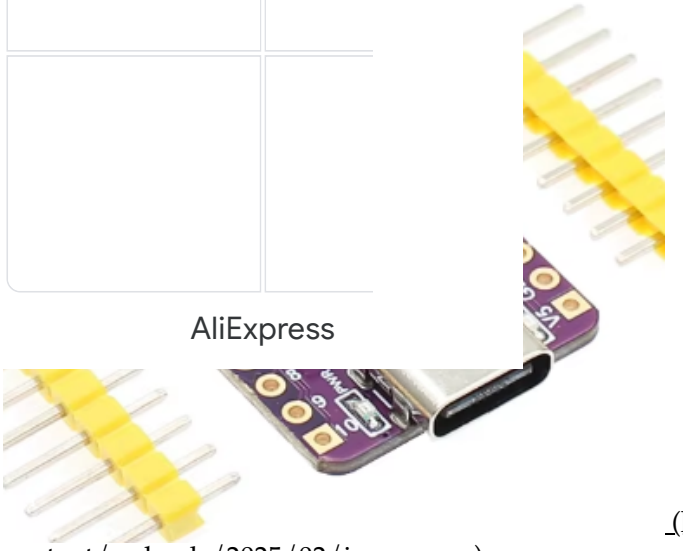
2025년 2월 12일 · 케빈



(https://emalliab.wordpress.com/wp-content/uploads/2025/02/image.png)

하지만 원본 참고 자료를 찾는 것이 쉽지는 않지만, 여러 출처에서 필요한 기본적인 내용들을 모아서 제대로 작동하게 할 수 있었던 것 같습니다. 그래서 몇 가지 메모해 두는 게 좋겠다고 생각했습니다.

"ABRobot ESP32-C3 0.42 OLED" 장치인 것 같습니다. 적어도 처음 전원을 켜면 "ABrobot"이라는 텍스트가 표시됩니다. 하지만 ABRobot 웹사이트에 직접 접속할 수 없는 것 같고, (아마도) 중국어로만 되어 있습니다.

원래 구매 사이트에는 다음과 같은 설명이 있습니다.

ESP32 C3 OLED 개발 보드는 ESP32C3FN4/FH4를 기반으로 설계 및 제작된 코어 보드입니다. 4M 플래시가 내장되어 있으며, WiFi와 Bluetooth 두 가지 모드를 지원하고, 세라믹 안테나를 사용하며, 0.42인치 OLED 화면을 탑재하고 USB 다운로드를 지원합니다.

- *전압: 3.3~6V*
- *WiFi: 802.11b/g/n 프로토콜, 2.4GHz*
- *블루투스: BT 5.0*
- *SPI 플래시: 4M 내장*
- *핀 인터페이스: 1xI2C, 1xSPI, 2xUART, 11xGPIO(PWM), 4xADC*
- *온보드 LED 파란색 표시등: GPO8 핀*
- *BO0: 다운로드 버튼, BO0 버튼을 길게 누르고 USB 케이블을 연결한 후 손을 떼면 "다운로드 모드"로 진입합니다.*
- *RST: 재설정 버튼, 프로그램 실행을 재시작하거나 재설정합니다.*
- *USB 인터페이스: 회로 기판의 전원 공급 장치로 사용하거나 PC와 ESP32C3를 연결하기 위한 통신 인터페이스로 사용할 수 있습니다.*
- *크기: 24.8mm * 20.45mm*

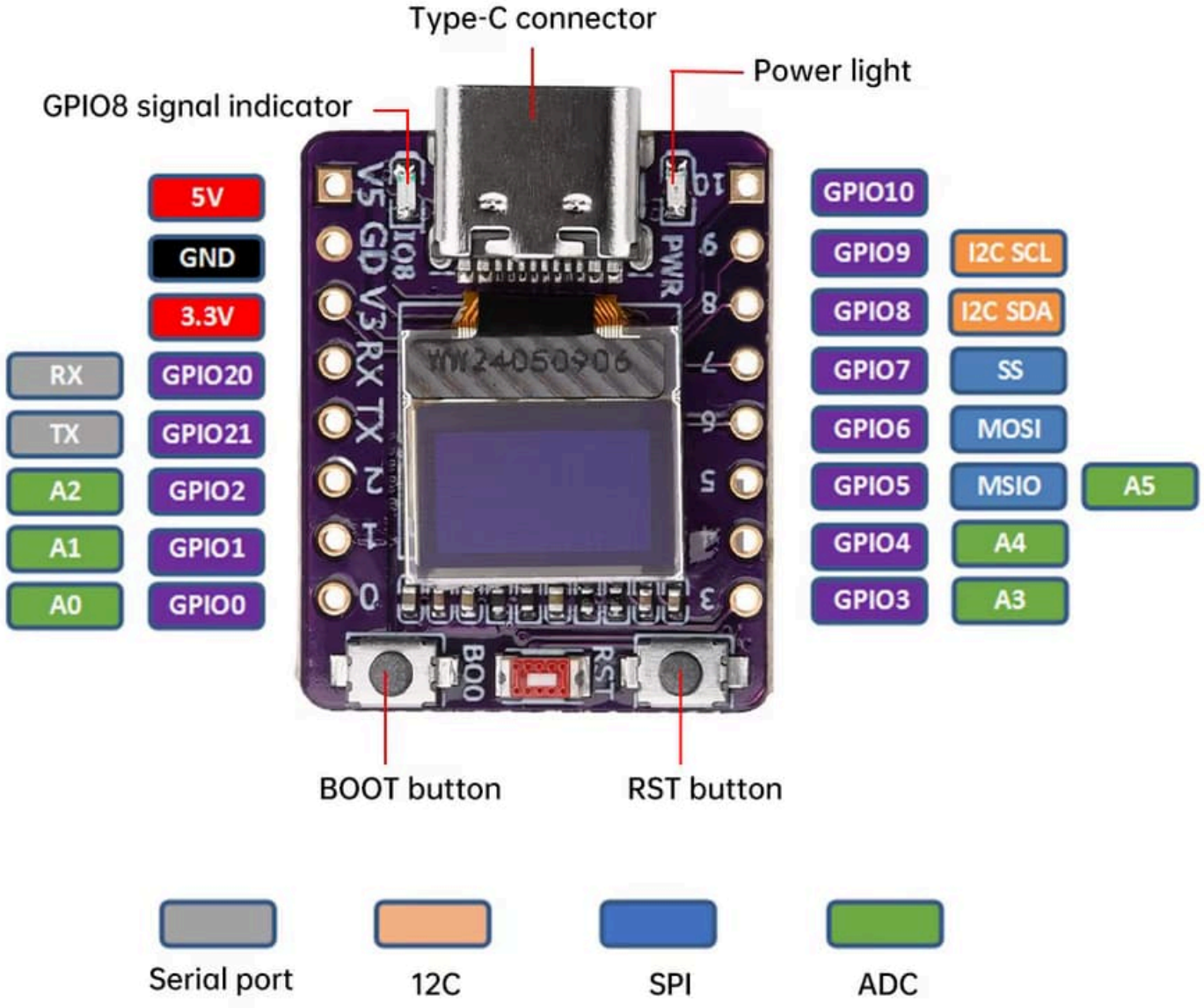
하지만 흥미로운 진술도 있습니다.

이 화면은 다른 0.42 인치 화면과 다릅니다. 화면의 시작점은 12864(13, 14) 입니다. 구매 전 주의하시기 바랍니다. 다른 0.42 인치 화면과 직접 교체해서는 안 됩니다.

좀 더 자세한 정보를 제공하는 추가 리뷰 코멘트가 있습니다.

ESP32-C3 를 탑재한 ABROBOT 개발 보드는 USB CDC 를 기본 지원하는 소형 IoT 플랫폼으로, 외부 컨버터가 필요 없습니다. 제대로 작동하려면 새 부트로더를 플래싱해야 합니다. 온보드 OLED 디스플레이는 유효 해상도가 72x40 픽셀이지만, U8g2 와 같은 라이브러리를 사용하려면 해상도를 128x64 로 설정해야 합니다. 디스플레이는 주소가 0x3C 인 GPIO5(SDA) 와 GPIO6(SCL) 을 사용하여 I2C 로 연결됩니다. (30, 12) 의 디스플레이 오프셋은 적절한 렌더링을 보장하며, 클리핑 방지를 위해 그리기는 72x40 픽셀로 제한됩니다. 기타 기능으로는 GPIO8 의 내장 LED, GPIO21 과 GPIO20 의 UART 통신, GPIO8 과 GPIO9 의 두 번째 I2C, SPI, ADC(GPIO0~GPIO5), 그리고 부팅 또는 리셋 기능을 위한 GPIO9 의 BOOT 버튼이 있습니다. 이 보드는 다음과 같은 IoT 프로젝트에 이상적입니다. 유연성과 소형 폼 팩터."

다음과 같은 핀아웃이 일치하는 것 같습니다.



(https://emalliab.wordpress.com/wp-content/uploads/2025/02/image-1.png)

하지만 주석에서 하드웨어 I2C가 이 다이어그램에 암시된 것처럼 8번과 9번 핀이 아니라 5번과 6번 핀에 있다고 언급되어 있는 것을 눈치채셨을 겁니다. 또한 디스플레이가 IO로 분리된 핀들을 사용하고 있다는 점도 흥미롭습니다. 그래서 번호 매기기가 정확하지 않은 건지, 아니면 칩/보드 변환 문제가 있는 건지 궁금합니다.

핀 5와 6은 예제 코드에서 특정 작업을 수행합니다(나중에 설명).

01Space에서 제작한, 밑면에 버튼이 있는 매우 유사한 보드가 있습니다. 이 보드에는 GitHub에 상당한 양의 문서와 회로도가 있지만, 저는 그것에 대해 아는 것이 없습니다. 제가 가지고 있는 보드는 확실히 위에 보이는 보드입니다.

여행 중에 Fritzing 포럼에서 이 게시물을 발견했는데, 여기에는 보드에 대한 Fritzing 부품이 포함되어 있었습니다: https://forum.fritzing.org/t/esp32-c3-oled-0-42-mini-board-part/25830 (https://forum.fritzing.org/t/esp32-c3-oled-0-42-mini-board-part/25830)

그리고 가장 중요한 건, stackexchange에 올라온 이 게시물인데, 이 게시물에는 디스플레이를 구동하는 샘플 코드가 포함된 리뷰 댓글이 있습니다! https://electronics.stackexchange.com/questions/725871/how-to-use-onboard-0-42-inch-oled-for-esp32-c3-oled-development-board-with-micro (https://electronics.stackexchange.com/questions/725871/how-to-use-onboard-0-42-inch-oled-for-esp32-c3-oled-development-board-with-micro)

이는 디스플레이를 SSD1306 디스플레이로 처리해야 하지만 화면에 사용 가능한 128x64 디스플레이 버퍼의 일부만 제공한다는 것을 의미합니다.

따라서 이를 실행하려면 다음이 필요합니다.

- Arduino에 ESP32 코어를 설치합니다.

- 저는 "ESP32C3 Dev Module" 보드 유형을 사용했습니다.
- U8g2 그래픽 라이브러리를 설치하세요(https://github.com/olikraus/u8g2 (https://github.com/olikraus/u8g2)) - Arduino 라이브러리 관리자에서 사용할 수 있습니다.
- 해당 리뷰 댓글/포럼 게시물에서 코드를 가져오세요(아래에 재생산했습니다).
- 빌드하고 업로드합니다.

참고 - 첫 번째 업로드 시에는 BOOT 버튼을 누른 상태에서 RESET 버튼을 눌렀다 놓아야 합니다. 이후 업로드 시에는 Arduino 업로드 과정의 일부로 자동으로 리셋/재부팅되는 것 같습니다.

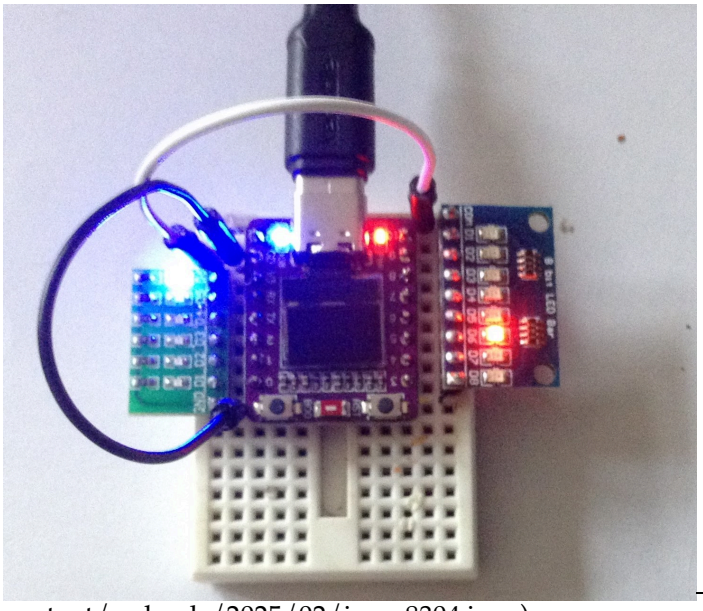
보드에 전원을 공급하는 가장 간단한 방법은 온보드 USB-C 연결을 사용하는 것이지만, 제가 본 사양에 따르면 3.7V~6V 전원을 5V 핀에 연결해도 작동할 것으로 보입니다... (저는 시도해 본 적이 없습니다).

기타 설정:

- USB 직렬 포트(예: 직렬 모니터용)가 필요한 경우 "부팅 시 USB-CDC 활성화"를 "활성화"로 설정해야 합니다.
- 나머지는 모두 "ESP32C3 Dev Module"과 함께 제공된 기본값으로 두었습니다.

IO 핀 확인

다음 코드는 각 GPIO 핀을 차례로 테스트합니다. 이 코드를 사용하는 가장 좋은 방법은 아래와 같이 얻을 수 있는 "LED 막대"를 사용하는 것입니다.

 (https://emalliab.wordpress.com/wp-content/uploads/2025/02/img_8304.jpeg)

다음 코드는 각 GPIO 핀을 출력으로 순환합니다. 핀 0은 세 번 나열되어 있으므로 각 시퀀스가 시작될 때 세 번 깜빡여 확인됩니다.

위의 모든 숫자가 확인되는 것과 더불어 주의해야 할 점은 GPIO 8이 HIGH일 때 온보드 LED가 꺼진다는 것입니다.

```
#define NPINS 15
int pins[NPINS] = {0,0,0,1,2,3,4,5,6,7,8,9,10,20,21};

void setup() {
  int i=0; i<NPINS; i++ {
    pinMode(pins[i],OUTPUT);
    digitalWrite(pins[i], LOW);
  }
}

void loop() {
  int i=0; i<NPINS; i++ {
    digitalWrite(pins[i], HIGH);
    delay(500);
    digitalWrite(pins[i], LOW);
    delay(500);
  }
}
```

기타 코드/환경 참고 사항

"ESP32C3 Dev Module"을 사용할 때 LED_BUILTIN이 정의된 것으로 나타나지만 LED에 연결된 GPIO(GPIO 8)에 매핑되지 않습니다.

IO 핀이 사용되지 않을 때 다음 핀은 기본적으로 HIGH로 표시됩니다.

- 21 – 텍사스
- 5 – I2C에서 OLED로
- 6 – I2C에서 OLED로
- 8 – 기본 I2C
- 9 – 기본 I2C

직렬/USB 직렬/UART

Arduino ESP32C3 코어 구성에서 UART0(RX0, TX0)은 20번, 21번 핀에, UART1(RX1, TX1)은 18번, 19번 핀에 설정되어 있습니다. 즉, 시리얼 통신이 가능하며, USB CDC가 활성화된 경우 USB 포트로 매핑됩니다.

업데이트: 회로도를 찾았는데(게시물 끝 부분 참조), 실제로 USB가 18, 19에 매핑되어 있는 것을 볼 수 있었습니다(실제로 이것들은 ESP32C3의 USB D+/D- 핀입니다). 그러면 20, 21은 USB가 아닌 하드웨어 UART로 비어 있게 됩니다.

그럼, 직렬 포트에 대해 간단히 살펴보겠습니다.

ESP32 Arduino 코어의 ESP32C3 정의 (https://github.com/espressif/arduino-esp32/blob/master/cores/esp32/HardwareSerial.h#L369) 를 보면 직렬 포트에 대한 다양한 옵션이 있으며, 특히 ESP32C3의 기본 장치는 다음 중 하나일 수 있습니다.

- USBSerial – Arduino 기본 USB 드라이버?
- HWCDCSerial – USB CDC 드라이버?
- Serial0 – 항상 존재하는 기본 직렬 포트, 핀 20, 21의 UART0.
- Serial1 – 두 번째 UART, 핀 18, 19의 UART1이 가능합니다.

솔직히 말해서 두 가지 USB 모드에 대해 전적으로 확신할 수는 없지만, USBSerial을 사용하려고 하면 컴파일되지 않지만 부팅 시 USB CDC를 활성화하면 HWCDCSerial은 정상적으로 컴파일된다는 것을 알고 있습니다.

코드는 다양한 하드웨어와 컴파일 시간 옵션에 따라 항상 "Serial"을 유용한 것으로 정의하므로 무슨 일이 일어나고 있는지 알아내기 위해 다음 코드를 사용했습니다.

```
void setup() {
  Serial.begin(9600);
  HWCDCSerial.begin(9600);
  Serial0.begin(9600);
  //Serial1.begin(9600);
}

unsigned cnt;
void loop() {
  Serial.print("시리얼: ");
  Serial.println(cnt);
  HWCDCSerial.print("HWCDCSerial: ");
  HWCDCSerial.println(cnt);
  Serial0.print("시리얼0: ");
  Serial0.println(cnt);
  //Serial1.print("시리얼1: ");
  //Serial1.println(cnt);
  delay(500);
  cnt++;
}
```

흥미롭게도 위의 코드는 잘 작동하고 컴파일되고 실행되며 다음과 같이 직렬 모니터에 출력이 표시됩니다.

```
직렬: 0
HWCDCSerial: 0
직렬: 1
HWCDCSerial: 1
직렬: 2
HWCDCSerial: 2
직렬: 3
HWCDCSerial: 3
직렬: 4
HWCDCSerial: 4
직렬: 5
HWCDCSerial: 5
```

Serial1을 포함하면 코드는 컴파일되지만 연결된 컴퓨터에서는 USB 직렬 포트를 전혀 인식하지 못합니다. 이것이 Espressif IDF와 ESP32 Arduino 코어 간의 상호 작용에 약간의 버그가 있는 것 같습니다.

IDF:soc/esp32c3/include/soc/soc_caps.h는 ESP32C3에 두 개의 UART가 있다고 선언합니다.

```
#SOC_UART_NUM(2)을 정의합니다.
#SOC_UART_HP_NUM(2)을 정의합니다.
```

하지만 ESP32C3 데이터시트에는 단일 UART와 USB 인터페이스만 나열되어 있습니다.

25	GPIO18	USB_D-				GPIO18	GPIO18	GPIO18	GPIO18	GPIO18	GPIO18	GPIO18	GPIO18	GPIO18
26	GPIO19	USB_D+				GPIO19	GPIO19	GPIO19	GPIO19	GPIO19	GPIO19	GPIO19	GPIO19	GPIO19
27	U0RXD					U0RXD	GPIO20	GPIO20	GPIO20	GPIO20	GPIO20	GPIO20	GPIO20	GPIO20
28	U0TXD					U0TXD	GPIO21	GPIO21	GPIO21	GPIO21	GPIO21	GPIO21	GPIO21	GPIO21

(https://emalliab.wordpress.com/wp-content/uploads/2025/02/image-8.png)

27	GPIO20	U0RXD	I1	GPIO20	I/O/T		
28	GPIO21	U0TXD	O	GPIO21	I/O/T		

(https://emalliab.wordpress.com/wp-content/uploads/2025/02/image-9.png)

25	GPIO18	USB_D-	
26	GPIO19	USB_D+	

(https://emalliab.wordpress.com/wp-content/uploads/2025/02/image-10.png)

그러나 ESP32 Arduino 코어에서는 SOC_UART_HP_NUM을 사용하여 Serial1이 있는지 여부를 확인합니다.

```
#SOC_UART_HP_NUM > 1이면
#define RX1 (gpio_num_t)18
#define TX1 (gpio_num_t)19
#endif

외부 HardwareSerial Serial0;
#SOC_UART_HP_NUM > 1이면
외부 HardwareSerial Serial1;
#endif
```

18/19가 실제로 차동 USB 신호라면 Serial1을 RX/TX가 있는 일반 직렬 포트로 처리하려고 하면 USB 링크가 실제로 엉망이 될 것입니다. 여기에는 USB 컨트롤러가 없고, 핀 18/19는 본질적으로 USB 소켓에 직접 연결되어 있습니다.

모든 것이 컴파일되고 실행되더라도 Serial0의 출력을 전혀 볼 수 없다는 점에 유의하세요. 이는 USB에 관련되지 않은 하드웨어 직렬 포트라는 생각과 일치합니다.

이 모든 것을 종합해 보면 Serial0은 실제로 하드웨어 UART0이고 Serial1은 하드웨어 UART1이라는 결론을 내릴 수 있습니다. 하지만 USB는 하드웨어 UART1에 연결되어 있고 Serial은 HWCDCSerial에 매핑됩니다. 적어도 부팅 시 USB CDC 모드가 활성화된 경우에는 그렇습니다.

즉, USB 직렬 인쇄를 방해하지 않고 핀 20과 21의 UART0을 애플리케이션에 사용할 수 있습니다.

I2C 포트에서

또 하나 주목할 점은 ESP32C3의 pins_arudino.h 파일에서 I2C가 핀 8과 9에 있다고 실제로 지정되어 있다는 것입니다: https://github.com/espressif/arduino-esp32/blob/master/variants/esp32c3/pins_arduino.h (https://github.com/espressif/arduino-esp32/blob/master/variants/esp32c3/pins_arduino.h)

하지만 ESP32C3에는 I2C 주변 장치가 하나뿐이므로 기본 빌드에서는 일반적으로 핀 8과 9에 I2C 풀업을 배치하지만, 이 특정 보드에서는 핀 5와 6에 추가 풀업을 추가하고 대신 OLED를 해당 핀에 연결했는지 궁금합니다.

PCB의 흔적만으로는 알아내기 어렵습니다. 많은 연결부가 디스플레이 아래에 있고, 표면 실장 부품이 솔더 패드를 조사하기에는 너무 작기 때문입니다.

업데이트: 회로도를 찾아본 결과, 8번과 9번 핀에 풀업 핀이 있는 데에는 다른 이유가 있는 것 같습니다. 따라서 ESP32 코어가 기본 I2C라고 "생각하는" 값을 무시하고 이 보드에서는 GPIO 5와 6을 그대로 받아들이는 것이 좋을 것 같습니다. 물론 이렇게 되면 위의 핀 배치 다이어그램은 틀리게 됩니다...

결론은 보드 자체는 꽤 깔끔하지만, I2C 주변 장치가 디스플레이로 다시 라우팅되어 외부 I2C 및 SPI를 사용할 수 없다는 것입니다. 대신 SPI 주변 장치 중 하나(Fast SPI "FSPI")의 핀 2개를 사용합니다(이것들은 JTAG 디버깅 핀이기도 하지만, 지금은 관련이 없을 것입니다).

다른 SPI 주변 장치는 일반적으로 제공된 SPI 플래시 메모리에 연결하는 데 사용되는 ESP GPIO 12-17에 매핑됩니다.

OLED 코드 예시

따라서 기본적으로 이 코드에서 일어나는 일은 U8g2 그래픽 라이브러리 내에서 128×64 SSD1306 객체를 초기화하지만, 디스플레이의 픽셀을 지정할 때는 항상 행/열을 오프셋하여 128×64 디스플레이의 중앙에 있는 72×40 픽셀 창만 사용하도록 하는 것입니다.

이 모든 것은 아래의 너비, 높이, xOffset, yOffset 초기화에서 설정되고, xOffset과 yOffset은 모든 그래픽 작업을 위해 x, y 좌표에 추가됩니다.

이는 나머지 라이브러리 사용을 더 쉽게 만들기 위해 몇 개의 매크로나 인라인 함수로 정의하는 것이 가치 있는 종류의 것입니다.

사용된 생성자는 "U8G2_SSD1306_128X64_NONAME_F_HW_I2C"이며 여기에 정의되어 있습니다: https://github.com/olikraus/u8g2/wiki/u8g2setupcpp#ssd1306-128x64_noname-1 (https://github.com/olikraus/u8g2/wiki/u8g2setupcpp#ssd1306-128x64_noname-1) 그리고 매개변수는 다음과 같습니다:

- U8G2_SSD1306_128X64_NONAME_F_HW_I2C(회전, [재설정 [, 클럭, 데이터]]) [전체 프레임버퍼, 크기 = 1024바이트]

따라서 이 경우에는 회전(U8G2_R0) 없음, RESET 핀 없음, SCL=6, SDA=5 및 기본 프레임 버퍼 1024를 사용합니다.

```
#include <U8g2lib.h>
#include <Wire.h>

// u8g2에는 72x40 생성자가 없으므로 72x40 화면은
// SSD1306 컨트롤러의 132x64 픽셀 버퍼 중앙에 매핑됩니다
U8G2_SSD1306_128X64_NONAME_F_HW_I2C u8g2(U8G2_R0, U8X8_PIN_NONE, 6, 5);
int width = 72;
int height = 40;
int xOffset = 30; // = (132-w)/2
int yOffset = 12; // = (64-h)/2

void setup(void)
{
     delay(1000);
     u8g2.begin();
     u8g2.setContrast(255); // 대비를 최대로 설정
     u8g2.setBusClock(400000); //400kHz I2C
     u8g2.setFont(u8g2_font_ncenB10_tr);
}

void loop(void)
{
    u8g2.clearBuffer(); // 내부 메모리를 지웁니다.
    u8g2.drawFrame(xOffset+0, yOffset+0, width, height); // 테두리 주위에 프레임을 그립
니다.
    u8g2.setCursor(xOffset+15, yOffset+25);
    u8g2.printf("%dx%d", width, height);
    u8g2.sendBuffer(); // 내부 메모리를 디스플레이로 전송합니다
. }
```

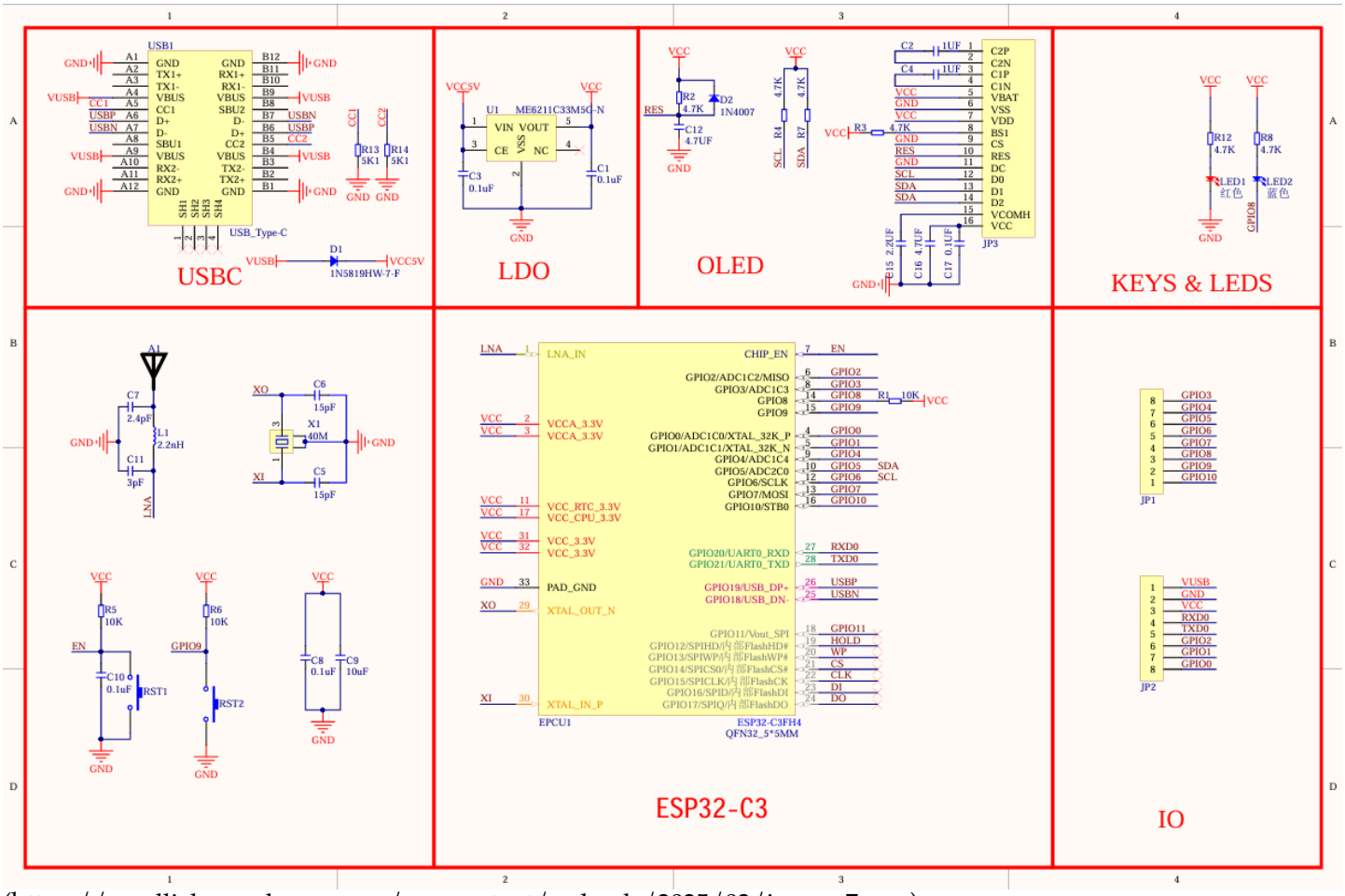
가능한 개략도

저는 보드에 대한 매우 낮은 해상도의 개략도를 발견했고, 그것을 바탕으로 제가 할 수 있는 모든 것을 조각해 보려고 시도하기 시작했습니다.

… 그런데 운이 좋았어요. 원래 프로젝트 페이지에는 접근할 수 없었지만, DuckDuckGo에서 해당 URL을 검색해 보니 (Google에서 아무리 찾아도 나오지 않았어요) 결국 해당 보드와 관련이 있는 것 같은 GitHub 저장소가 나타났고, 거기에 제가 찾은 퍼지 회로도 원본이 들어 있었어요!

자세한 내용은 여기를 참조하세요: https://github.com/zhuhai-esp/ESP32-C3-ABrobot-OLED/tree/main (https://github.com/zhuhai-esp/ESP32-C3-ABrobot-OLED/tree/main)

하지만 다시 잃어버릴 경우를 대비해 이 도면을 사용하세요.



(https://emalliab.wordpress.com/wp-content/uploads/2025/02/image-7.png)

따라서 여기서는 핀 5와 6이 디스플레이에 대한 I2C 인터페이스이고 HIGH로 설정되었다는 것이 분명합니다.

하지만 GPIO 8이 저항 R1을 통해 HIGH로 설정되어 있는 것을 볼 수 있습니다. 또한, GPIO 8은 또 다른 저항(R8)을 통해 온보드 사용자 LED에 연결되어 있습니다. 역방향 논리 설정에서는 LED를 켜려면 GPIO 8을 LOW로 설정해야 합니다.

그렇다면 GPIO 9는 어떨까요? 당장은 명확하지 않지만, 왼쪽 하단의 리셋 회로를 보면 GPIO 9가 R6을 통해 VCC에 HIGH로 연결되어 있고 RST2에 연결되어 있는 것을 볼 수 있습니다. RST1이 실제 리셋 버튼(EN에 연결되어 있으므로)이고, RST2는 BOOT(제 보드에서는 "BOO")일 것이고, 아마도 사용자 버튼 역할도 할 것으로 예상됩니다.

그 외에도 LDO 섹션에서 5V 헤더 핀은 실제로 VUSB이고, 이 핀 자체는 1N5819 다이오드를 통해 VCC5V에 연결되어 있으며, 이 VCC5V는 시스템에 필요한 3V3을 생성하는 ME6211C33 LDO 레귤레이터의 입력입니다.

제 제한된 이해력으로는 5V 헤더 핀이 USB가 연결된 동안 5V가 인가되는 것을 방지하기 위한 다이오드의 반대편에 있을 것이라고 생각했을 겁니다... 하지만 제가 이런 것들이 작동하는 방식을 잘못 이해하고 있는 것일 수도 있습니다.

하지만 계측기로 확인해보니 다이오드의 USB 쪽은 5.1V를 나타내고 핀 헤더의 5V 라인도 마찬가지입니다. 반면 다이오드의 일반 쪽은 레귤레이터의 핀 중 하나와 마찬가지로 약 4.8V를 나타냅니다. 따라서 회로도가 정확하다고 생각합니다.

결론

보통은 끈기 있게 검색하면 필요한 정보를 찾을 수 있지만, 이 작은 보드들은 의외로 이해하기가 어려웠습니다. 그 흐릿한 회로도를 해독하려고 한참을 노력하던 중, 마침내 운이 좋게도 그 정보를 찾아냈습니다.

핀아웃이 실제로 잘못되었다는 것을 알고 있으니, 다른 핀, 특히 ADC의 라벨링이 궁금합니다. 나중에 ESP32-C3 데이터시트를 확인해서 확인해 보겠습니다.

하지만 이제는 충분히 활용할 수 있을 것 같아요.

그 작은 디스플레이는 정말 멋진데, 벌써 몇 가지 다른 용도가 떠오르네요.

케빈

Kevin 이 게시함

인터넷을 떠도는 또 다른 영혼일 뿐입니다... *Kevin의 모든 게시물 보기*

아두이노 **ESP32C3** **OLED**

“ ESP32-C3 0.42 OLED ” 에 대한 4가지 생각

1. Pingback: ESP32C3 OLED 미니 MIDI 모니터 – 간단한 DIY 전자 음악 프로젝트
2. *마체 자신*
   말한다:
   2025년 5월 1일 오전 11시 39분
   정말 아름답게 그려내셨네요! 정말 감사합니다! 저도 좀 특이한 보드를 가지고 있는데, 이 블로그 글이 정말 큰 도움이 됩니다. 감사합니다! 😀

   회신하다
   1. *케빈*
      말한다:
      2025년 5월 2일 오후 7시 4분

좋아요! 알려주셔서 감사합니다 🙂
보드가 너무 귀여워요!

케빈

회신하다

3. *미구엘프라테스*

말한다:
2025년 6월 5일 오후 10시 34분
안녕하세요, Kevin 님. 모든 설명 감사합니다.
OLED를 사용하는 코드에서 U8G2_SSD1306_72X40_ER_F_HW_I2C 클래스는 오프셋을 사용하지 않고도 완벽하게 작동합니다.

회신하다