第十四讲
总线与接口（II）
Lecture 14
Bus and Interface (II)

明玉瑞 Yurui Ming

yrming@gmail.com

声明
Disclaimer

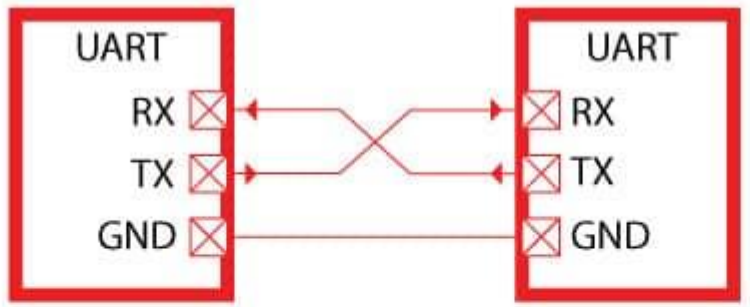
- 本讲义在准备过程中由于时间所限，所用材料来源并未规范标示引用来源。所引材料仅用于教学所用，作者无意侵犯原著者之知识版权，所引材料之知识版权均归原著者所有；若原著者介意之，请联系作者更正及删除。

  The time limit during the preparation of these slides incurs the situation that not all the sources of the used materials (texts or images) are properly referenced or clearly manifested. However, all materials in these slides are solely for teaching and the author is with no intention to infringe the copyright bestowed on the original authors or manufacturers. All credits go to corresponding IP holders. Please address the author for any concern for remedy including deletion.

UART 回顾
UART Recap

- 通用异步收发传输（UART）或串行通信是两个设备之间最简单的通信协议之一。它通过连接两根线缆在设备之间将数据以位的形式一个设备传输到另一个设备。

  Universal Asynchronous Receive Transmit (UART) or Serial communication is one of the most simple communication protocols between two devices. By connecting two wires between the devices, one is the transmission line while the other is the receiving line, it transfers data between devices in form of bits from one device to another.



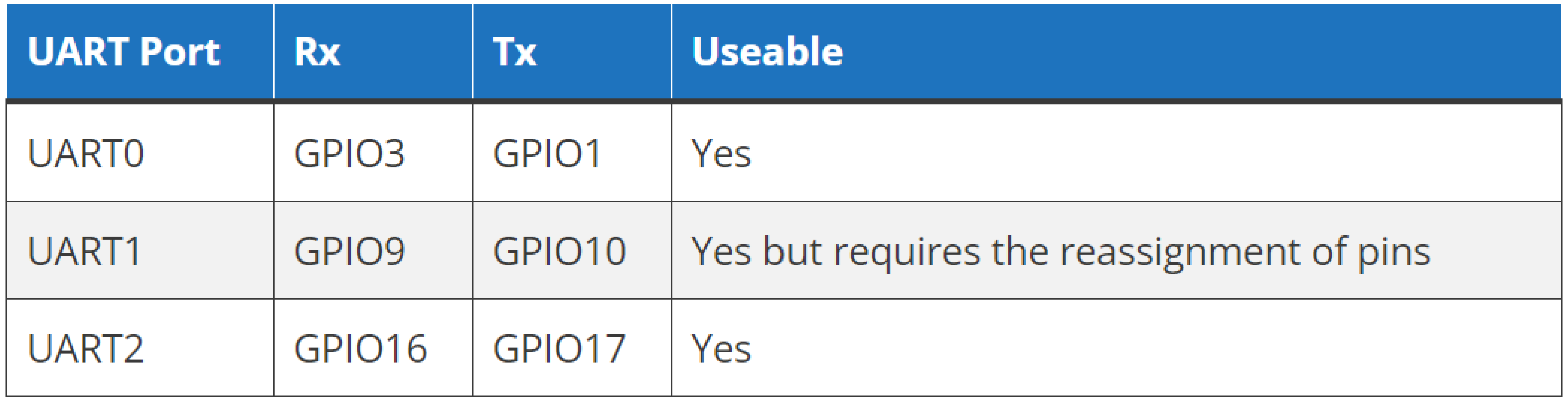
UART of ESP

- ESP32提供了三个通用异步收发器（UART）端口，分别是UART0、UART1和UART2，它们工作在3.3V TTL电平上。这三个串口接口都得到了硬件支持。每个串口都有4个引脚可用：RX、TX、RTS和CTS。

  ESP32 provides three universal asynchronous receivers and transmitter (UART) ports such as UART0, UART1, and UART2 that work at 3.3V TTL level. These three serial interfaces are hardware supported. Each of them exposes 4 pins: RX, TX, RTS and CTS. However, the Arduino IDE only uses RX and TX pins.

- 下表列出了ESP32可用的三个UART端口的RX和TX引脚对应关系：

  The table below specifies the RX and TX pins for each of the three UART ports available in ESP32:



UART Port	Rx	Tx	Useable
UART0	GPIO3	GPIO1	Yes
UART1	GPIO9	GPIO10	Yes but requires the reassignment of pins
UART2	GPIO16	GPIO17	Yes

UART of ESP

- 默认情况下，只有UART0和UART2可以使用。要使用UART1，我们需要重新定义引脚，因为UART1的默认引脚，如GPIO9和GPIO10，内部连接到SPI闪存存储器。此外，在一些ESP32开发板上，它们甚至没有暴露在引脚排针上。因此，在Arduino IDE中无法直接使用UART1，需要重新分配引脚。

  By default, only UART0 and UART2 can be used. To use UART1, we have to redefine the pins because default pins of UART1 such as GPIO9 and GPIO10 are internally connected to the SPI flash memory. Also, on some ESP32 boards, they are not even exposed on the pinout headers. Hence, we can not use UART1 directly without reassigning pins in Arduino IDE.

- 基于Arduino IDE使用硬件串口的示例如下：

  The paradigm of using hardware serial in Arduino IDE is as follows:

```
#include <HardwareSerial.h>

HardwareSerial SerialPort(2)

SerialPort.begin (BaudRate, SerialMode, RX_pin, TX_pin);

SerialPort.print(1); // write to it

if (SerialPort.available()) { char number = SerialPort.read(); } // read from it
```

SPI 回顾
SPI Recap

- SPI即串行外设接口，是微控制器用来与一个或多个外设通信的同步串行数据协议。例如，ESP32开发板可与支持SPI的传感器或另一个微控制器进行通信。在SPI通信中，控制器（也称为主机）控制外围设备（也称为从机），进行发送和接收数据。SPI是双工的，这意味着主机可以向从机发送数据，从机也可以同时向主机发送数据。

  SPI stands for Serial Peripheral Interface, and it is a synchronous serial data protocol used by microcontrollers to communicate with one or more peripherals. For example, your ESP32 board communicating with a sensor that supports SPI or with another microcontroller. In an SPI communication, there is always a controller (also called master) that controls the peripheral devices (also called slaves). Data can be sent and received simultaneously. This means that the master can send data to a slave, and a slave can send data to the master at the same time.

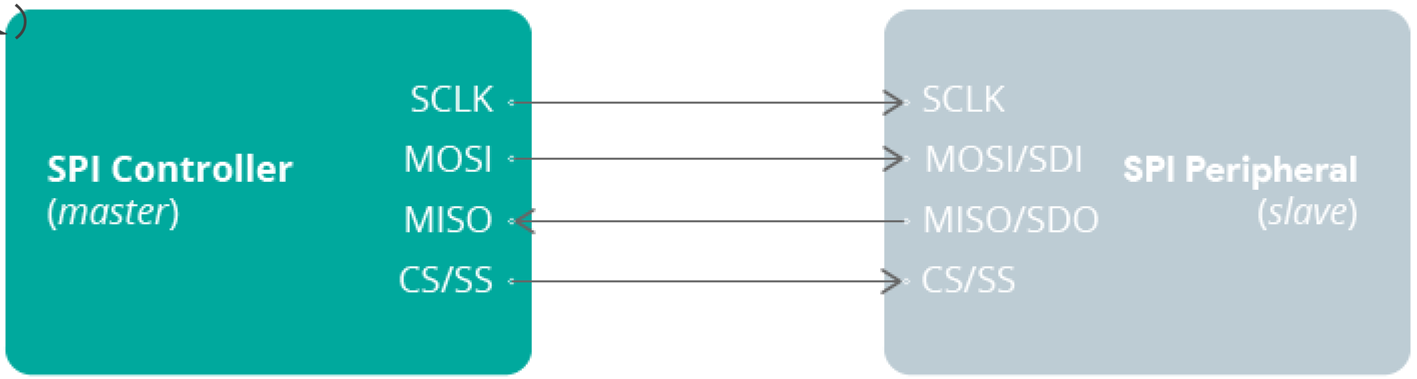
- 通常我们只有一个主机，如ESP32，但可以有多个从机。这意味着您可以将一个ESP32连接到多个从机，但同一个从机不能同时连接到多个ESP32开发板。

  You can have only one master, which will be a microcontroller (the ESP32), but you can have multiple slaves. This means you can have an ESP32 connected to multiple slaves, but the same slave can’t be connected to multiple ESP32 boards simultaneously.

SPI 回顾
SPI Recap

- 对于 SPI 通信，需要四根线：
  - MISO：主从输出
  - MOSI：主从输入
  - SCK：串行时钟
  - CS /SS：片选，用于在同一条SPI总线上使用多个外设时选择设备）
- 但在从机设备上，可能会用不同的术语：
  - MISO 可能被标记为 SDO（串行数据输出）
  - MOSI 可能被标记为 SDI（串行数据输入）

- For SPI communication you need four lines:
  - MISO: Master In Slave Out
  - MOSI: Master Out Slave In
  - SCK: Serial Clock
  - CS /SS: Chip Select, used to select the device when multiple peripherals are used on the same SPI bus
- On a slave-only device, like sensors, displays, and others, you may find a different terminology:
  - MISO may be labeled as SDO (Serial Data Out)
  - MOSI may be labeled as SDI (Serial Data In)



SPI of ESP32

- ESP32集成了4个SPI外设：SPI0、SPI1、SPI2（通常称为 HSPI）和 SPI3（通常称为VSPI）。SP0和SP1 在内部用于与内置闪存通信，不应将它们用于其他任务。用户可以使用 HSPI 和 VSPI 与其他设备进行通信。HSPI和VSPI有独立的总线信号，每条总线最多可以驱动三个SPI从机。

  The ESP32 integrates 4 SPI peripherals: SPI0, SPI1, SPI2 (commonly referred to as HSPI), and SPI3 (commonly referred to as VSPI). SP0 and SP1 are used internally to communicate with the built-in flash memory, and you should not use them for other tasks. You can use HSPI and VSPI to communicate with other devices. HSPI and VSPI have independent bus signals, and each bus can drive up to three SPI slaves.

- 许多ESP32开发板都预分配了默认的SPI引脚。通常，如果未指定，开发板将在使用默认设置，使用 VSPI 引脚初始化SPI通信。

  Many ESP32 boards come with default SPI pins pre-assigned. Usually, when not specified, the board will use the VSPI pins when initializing an SPI communication with the default settings.

SPI of ESP32

- 大多数板卡的引脚映射如下：

  The pin mapping for most boards is as follows:

SPI	MOSI	MISO	SCLK	CS
VSPI	GPIO 23	GPIO 19	GPIO 18	GPIO 5
HSPI	GPIO 13	GPIO 12	GPIO 14	GPIO 15

- 如果读者不清楚默认的SPI引脚，可以用下面代码找出，但要确保在“工具”>“开发板”中选择了正确的开发板。

  If you are not sure the default SPI pins for your board, you can find it using the script below. However, make sure you have the right board selected in Tools > Boards.

```
void setup() {
  Serial.begin(115200);
  Serial.print("MOSI: ");
  Serial.println(MOSI);
  Serial.print("MISO: ");
  Serial.println(MISO);
  Serial.print("SCK: ");
  Serial.println(SCK);
  Serial.print("SS: ");
  Serial.println(SS);
}
void loop() {
}
```

SPI of ESP32

- 当使用库与SPI外设连接时，使用自定义的SPI引脚通常很简单，将它们作为参数传递给库构造函数即可。例如，下面是使用Adafruit_BME280库与BME280传感器连接的示例。

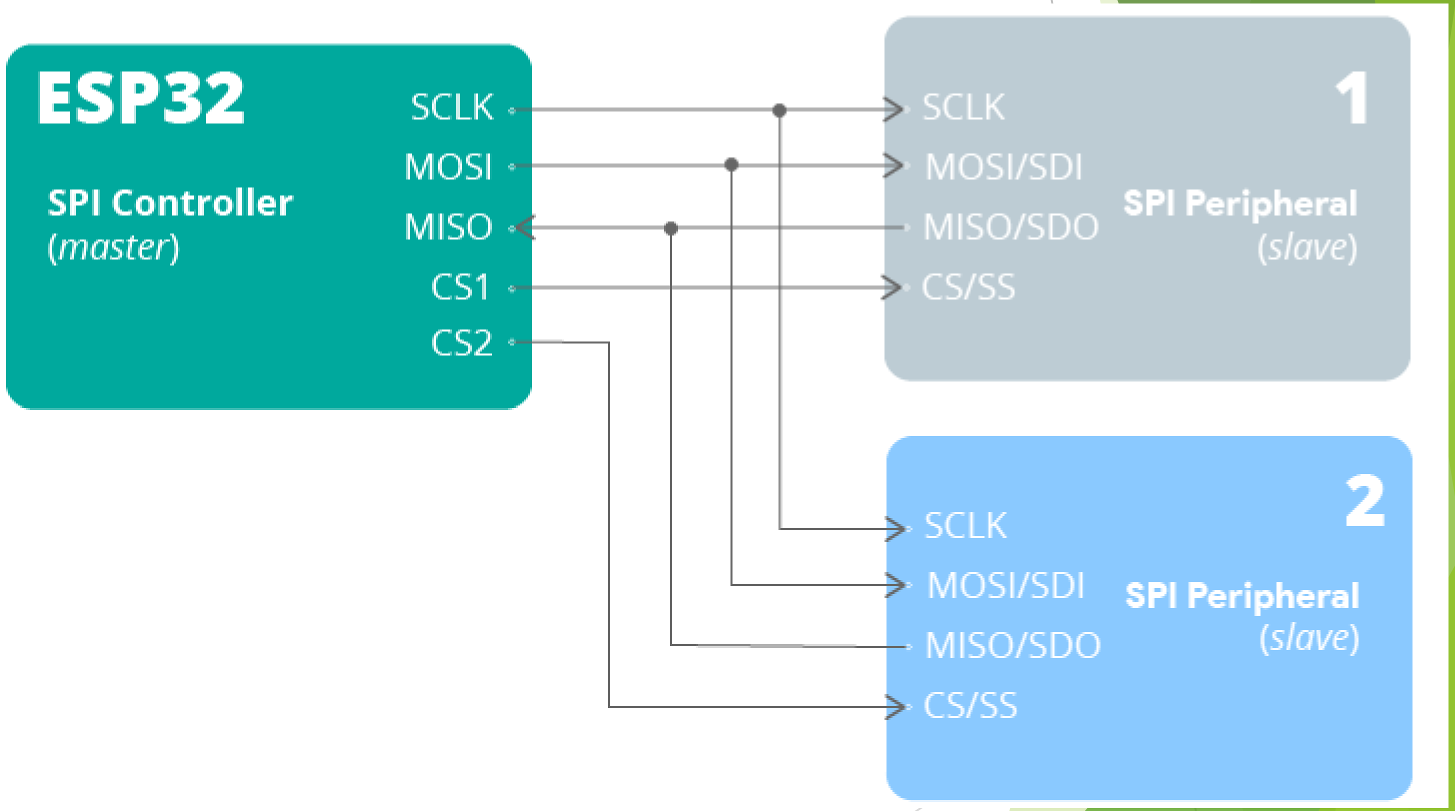
  When using libraries to interface with your SPI peripherals, it’s usually simple to use custom SPI pins because you can pass them as arguments to the library constructor. For example, the following example shows that to interface with a BME280 sensor using the Adafruit_BME280 library.

```
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>

#define BME_SCK 25
#define BME_MISO 32
#define BME_MOSI 26
#define BME_CS 33

Adafruit_BME280 bme(BME_CS, BME_MOSI, BME_MISO, BME_SCK);
```

SPI of ESP32

- 如前所述，可以使用ESP32上的两条不同的SPI总线，每条总线最多可以连接三个不同的外设，即我们最多可以将六个SPI设备连接到ESP32。当然，如果需要连接更多的设备，可以使用SPI多路复用器。

  As aforementioned, one can use two different SPI buses on the ESP32 and each bus can connect up to three different peripherals. This means that we can connect up to six SPI devices to the ESP32. Of course, if we need to use more, we can use an SPI multiplexer.



SPI of ESP32

- 当主机选择与之通信的外设时，会将其CS引脚设置为低电平。 例如，假设有外设1和外设2，要从外设1读取，则将其CS引脚（记为CS_1）设置为低电平：

  When the host selects the peripheral to communicate with, it will set the corresponding CS pin to LOW. For example, imagine you have peripheral 1 and peripheral 2. To read from peripheral 1, make sure its CS pin (here denoted as CS_1) is set to LOW:

```
digitalWrite(CS_1, LOW);
```

- 与时同时，当需要从外设2读取时，应该通过将CS_1设置为高电平来禁用外设1，并通过将CS_2设置为低电平来启用外设2：

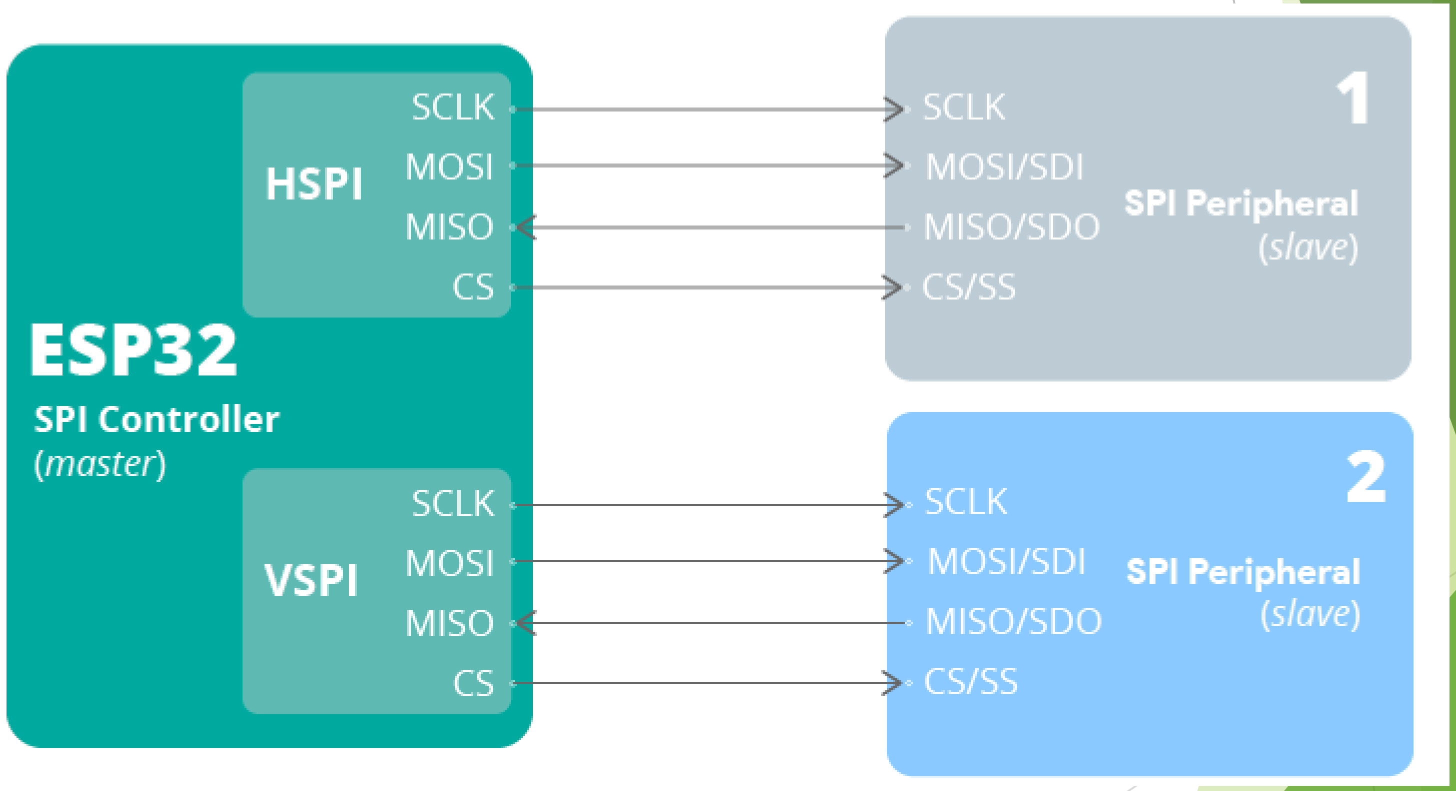
  Then at same point, you’ll want to read from peripheral 2. You should disable peripheral 1 CS pin by setting it to HIGH, and enable peripheral 2 CS pin by setting it to LOW:

```
digitalWrite(CS_1, HIGH);
digitalWrite(CS_2, LOW);
```

SPI of ESP32

- 要同时与多个 SPI 外设通信，可以使用ESP32的两条SPI总线（HSPI和VSPI），并且可以使用默认的 HSPI 和 VSPI 引脚或使用自定义引脚。

  To communicate with multiple SPI peripherals simultaneously, you can use the ESP32 two SPI buses (HSPI and VSPI). You can use the default HSPI and VSPI pins or use custom pins.



SPI of ESP32

- 要同时使用HSPI和VSPI，我们通常按以下步骤进行：

  To use HSPI and VSPI simultaneously, we usually follow the procedures below:

- 1) 首先，确保在代码中包含 SPI 库。

  First, make sure you include the SPI library in your code.

```
#include <SPI.h>
```

- 2）初始化两个不同名称的SPIClass对象，一个在HSPI总线上，另一个在VSPI总线上。例如：

  Initialize two SPIClass objects with different names, one on the HSPI bus and another on the VSPI bus. For example:

```
vspi = new SPIClass(VSPI);
hspi = new SPIClass(HSPI);
```

- 3) 在这些对象上调用 begin() 方法。

  Call the begin() method on those objects.

```
vspi.begin(); hspi.begin();
```

SPI of ESP32

- 如果需要，可以将自定义引脚传递给 begin() 方法：

  You can pass custom pins to the begin() method if needed:

```
vspi.begin(VSPI_CLK, VSPI_MISO, VSPI_MOSI, VSPI_SS);
hspi.begin(HSPI_CLK, HSPI_MISO, HSPI_MOSI, HSPI_SS);
```

- 4)最后，我们还需要将 SS 引脚设置为输出。例如：

  Finally, you also need to set the SS pins as outputs. For example:

```
pinMode(VSPI_SS, OUTPUT);
pinMode(HSPI_SS, OUTPUT);
```

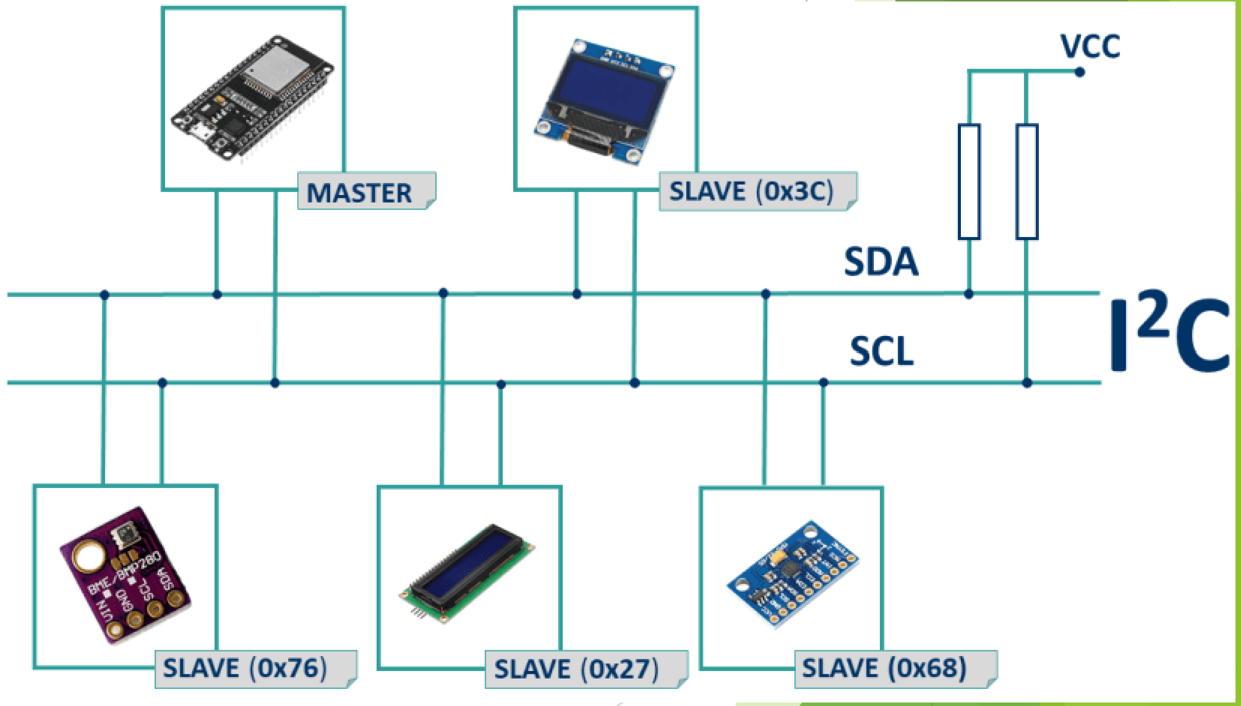
- 在此之后，我们可以使用传感器库或是 SPI 库方法，与 SPI 设备交互。

  Then, you can use a sensor library or the SPI library methods to interact with the SPI devices.

I2C 回顾
I2C Recap

- I²C是一种同步的、多主多从的通信协议，可以将多个从设备连接到一个主设备，或将多个主设备控制同一个从设备。其用两根线来共享信息。其中一根用于时钟信号（SCL），另一根用于发送和接收数据（SDA）

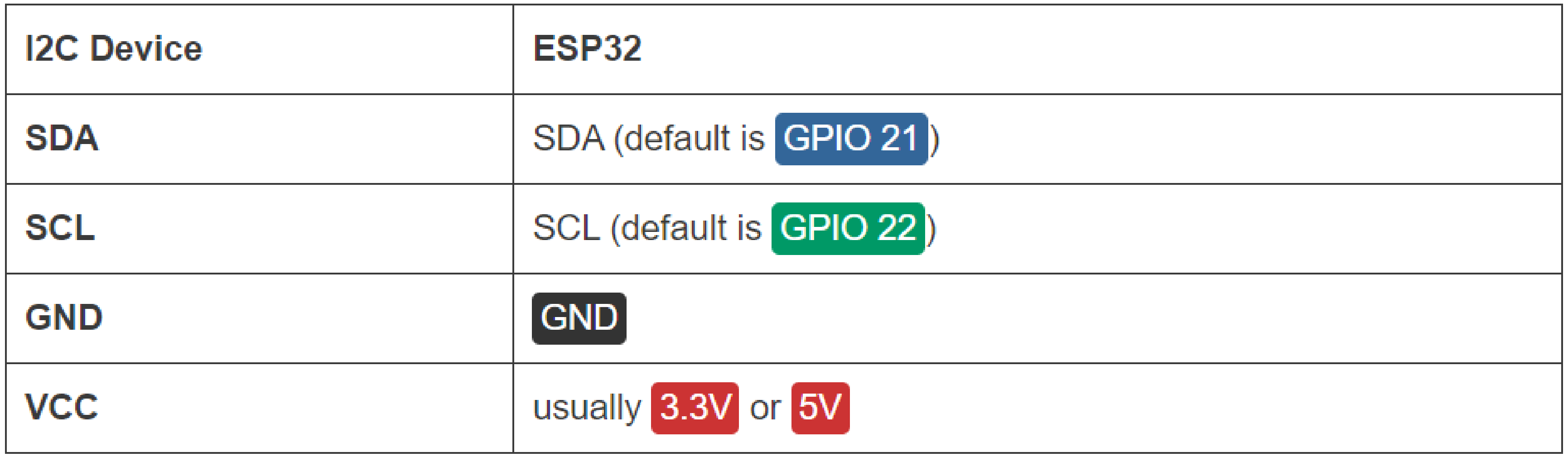
  I²C means Inter Integrated Circuit (it’s pronounced I-squared-C), and it is a synchronous, multi-master, multi-slave communication protocol. It can be used to connect multiple slaves to one master or multiple masters controlling the same slave. It uses two wires to share information. One is used for the clock signal (SCL) and the other is used to send and receive data (SDA).



I2C of ESP32

- ESP32通过其I2C总线接口支持I2C通信，这些接口可以根据用户的配置作为I2C主设备或从设备。将一个I2C设备连接到ESP32通常只需将GND与GND相连，SDA与SDA相连，SCL与SCL相连，然后将ESP32开发板上的电源正极连接到外设，通常是3.3V（但具体取决于使用的模块）。

  The ESP32 supports I2C communication through its I2C bus interface that can serve as I2C master or slave, depending on the user’s configuration. Connecting an I2C device to an ESP32 is normally as simple as connecting GND to GND, SDA to SDA, SCL to SCL and a positive power supply of the ESP32 board to a peripheral, usually 3.3V (but it depends on the module you’re using).

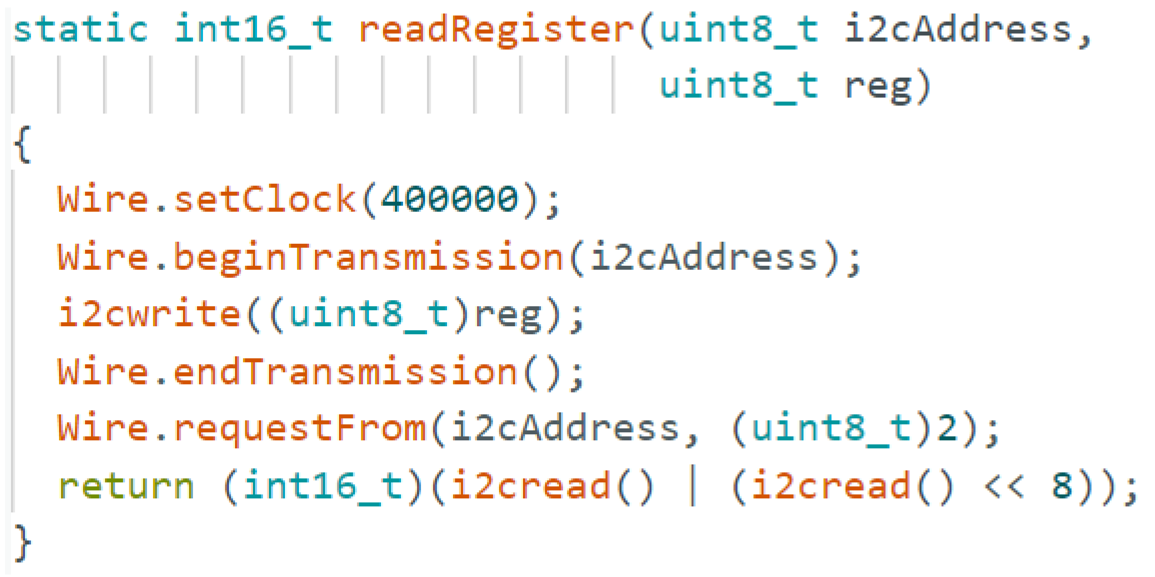


I2C Device	ESP32
SDA	SDA (default is GPIO 21)
SCL	SCL (default is GPIO 22)
GND	GND
VCC	usually 3.3V or 5V

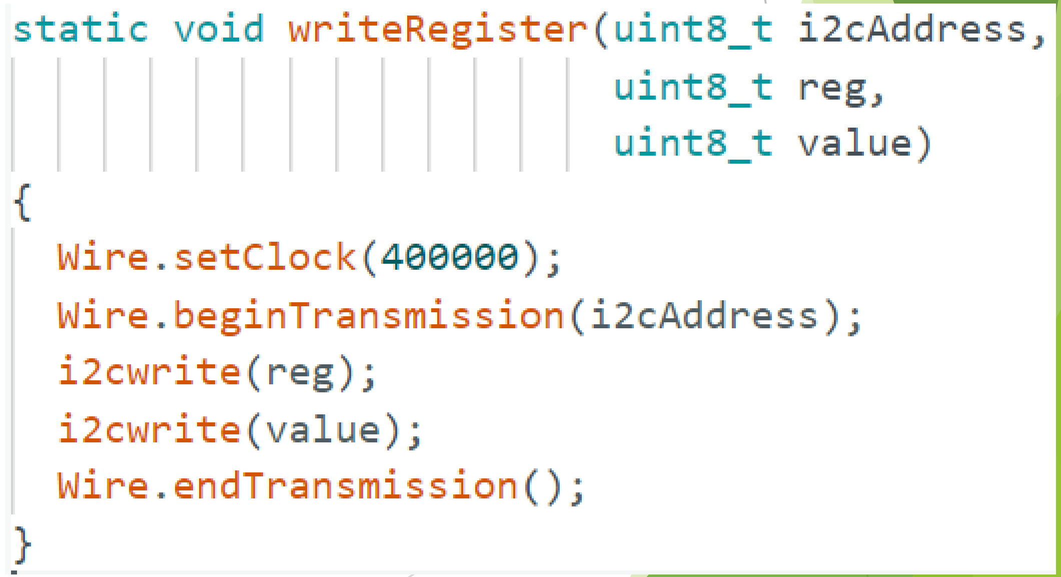
I2C of ESP32

- 在I2C通信中，总线上的每个从设备都有自己的地址，其是一个十六进制数，允许ESP32与每个设备进行通信。I2C地址通常可以在组件的数据手册中找到。基于Arduino IDE使用I2C编程的示例如下：

  With I2C communication, each slave on the bus has its own address, a hexadecimal number that allows the ESP32 to communicate with each device. The I2C address can be usually found on the component’s datasheet. The paradigm of using I2C in Arduino IDE is as follows:



```
static int16_t readRegister(uint8_t i2cAddress,
                            uint8_t reg)
{
  Wire.setClock(400000);
  Wire.beginTransmission(i2cAddress);
  i2cwrite((uint8_t)reg);
  Wire.endTransmission();
  Wire.requestFrom(i2cAddress, (uint8_t)2);
  return (int16_t)(i2cread() | (i2cread() << 8));
}
```



```
static void writeRegister(uint8_t i2cAddress,
                          uint8_t reg,
                          uint8_t value)
{
  Wire.setClock(400000);
  Wire.beginTransmission(i2cAddress);
  i2cwrite(reg);
  i2cwrite(value);
  Wire.endTransmission();
}
```