

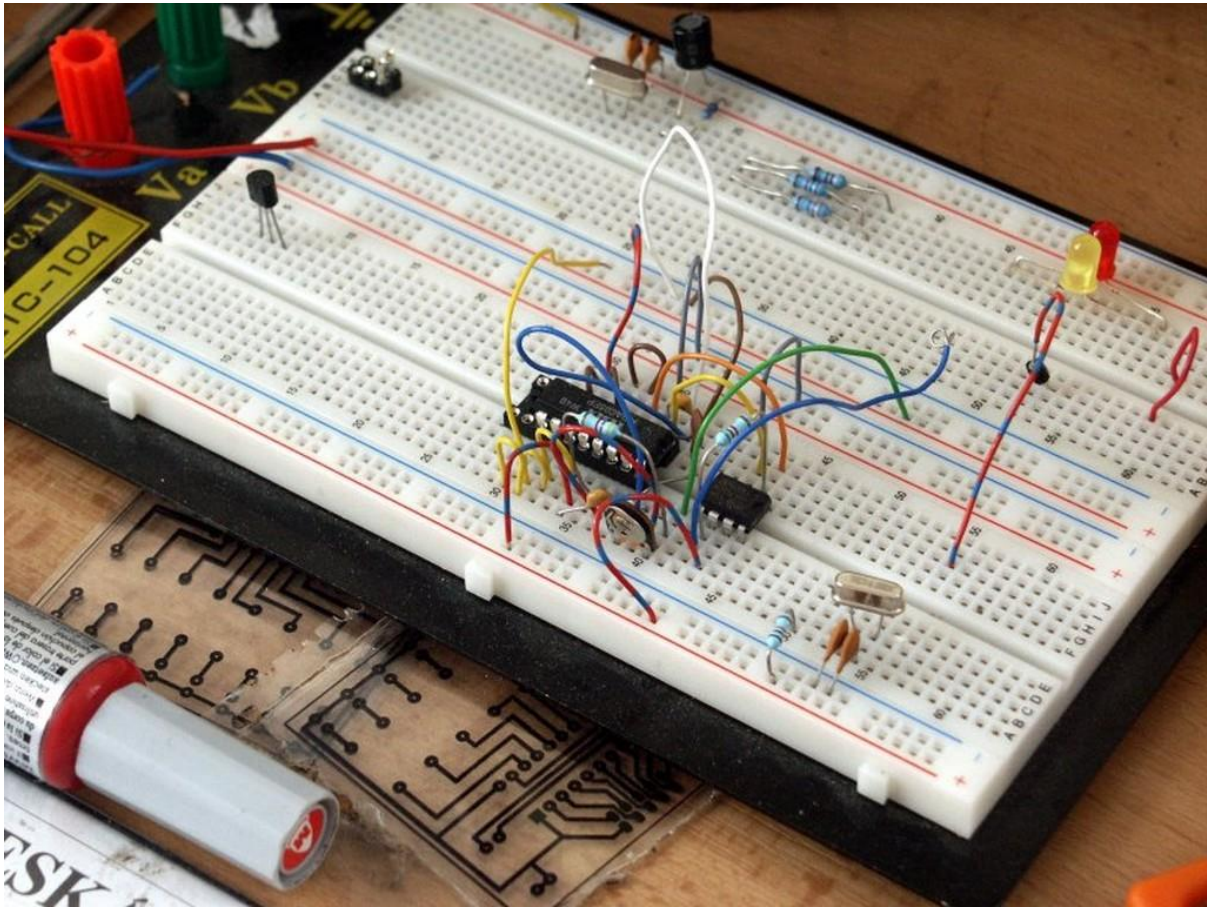


WIRE CONNECTION INTRODUCTION

Contents

CHAPTER 1: OVERVIEW	3
CHAPTER 2: UART.....	5
1. What is Uart?.....	5
a. Overview	5
b. How it work?	6
c. Advantages and disadvantages of Uarts	7
2. Usage.....	8
CHAPTER 3: I2C	10
1. What is I2C?	10
a. Overview	10
b. How it work?	11
c. Advantages and disadvantages of I2C.....	13
2. Usage:.....	14
CHAPTER 4: SPI.....	17
1. What is SPI?	17
a. Overview	17
b. How it works?	18
c. Advantages and disadvantages of SPI.....	19
2. Usage:.....	20
CHAPTER 5: DIFFERENCE BETWEEN UART, SPI, I2C	23

CHAPTER 1: OVERVIEW



Digital chips communicate with each other using two methods: serial or parallel. Serial communication means all of the data flows sequentially through a wire, versus parallel communication transmits data through multiple wires simultaneously.

Most chips interface using serial methods because of the need for less wires which simplifies the design. The downside of serial communication is it's not usually as fast as parallel communication.

For example, the connection between a microprocessor and any RAM memory must be very fast and therefore a parallel interface is typically required.

Universal-Asynchronous-Receive-Transit (UART) is one of the oldest and most common serial protocols. Asynchronous simply means no clock signal is used for timing purposes.

A UART interface needs at least two lines: receive and transmit. So data flows in only one direction on each of the two data lines.

I2C is a popular serial protocol that uses two wires: a clock signal and a bidirectional data line. I2C is considered synchronous since it uses a clock for timing. I2C is commonly used for interfacing all kinds of sensors to a microcontroller.

SPI is another very common serial communication protocol. Like with UART, SPI uses two data unidirectional data lines, but it is instead synchronous with a clock signal.

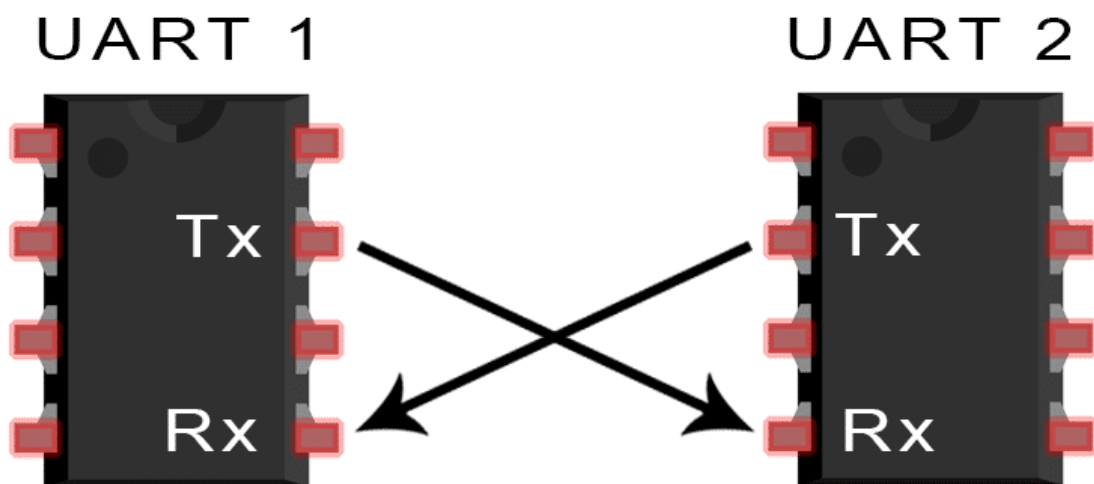
SPI tends to be the preferred choice when data speeds are more critical. For example, you may use SPI when interfacing a color display to a microcontroller. But you would likely use I2C to interface a temperature sensor to the microcontroller.

CHAPTER 2: UART

1. What is Uart?

a. Overview

In UART communication, two UARTs communicate directly with each other. The transmitting UART converts parallel data from a controlling device like a CPU into serial form, transmits it in serial to the receiving UART, which then converts the serial data back into parallel data for the receiving device. Only two wires are needed to transmit data between two UARTs. Data flows from the Tx pin of the transmitting UART to the Rx pin of the receiving UART:



UARTs transmit data *asynchronously*, which means there is no clock signal to synchronize the output of bits from the transmitting UART to the sampling of bits by the receiving UART. Instead of a clock signal, the transmitting UART adds start and stop bits to the data packet being transferred. These bits define the beginning and end of the data packet so the receiving UART knows when to start reading the bits.

When the receiving UART detects a start bit, it starts to read the incoming bits at a specific frequency known as the *baud rate*. Baud rate is a measure of the speed of data transfer, expressed in bits per second (bps). Both UARTs must operate at about the

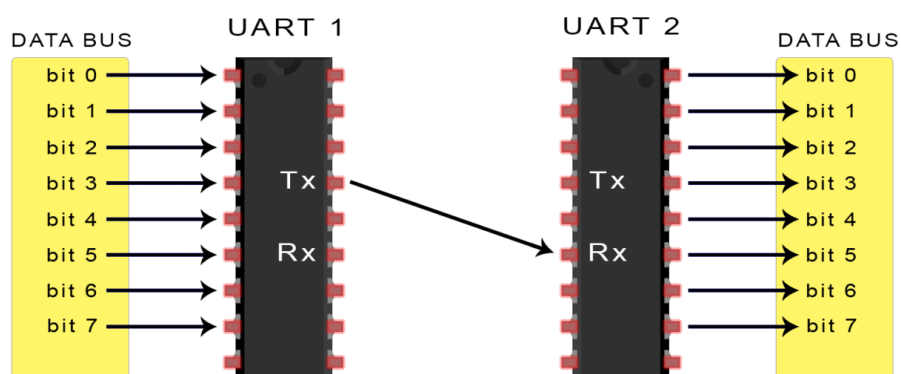
same baud rate. The baud rate between the transmitting and receiving UARTs can only differ by about 10% before the timing of bits gets too far off.

Both UARTs must also must be configured to transmit and receive the same data packet structure.

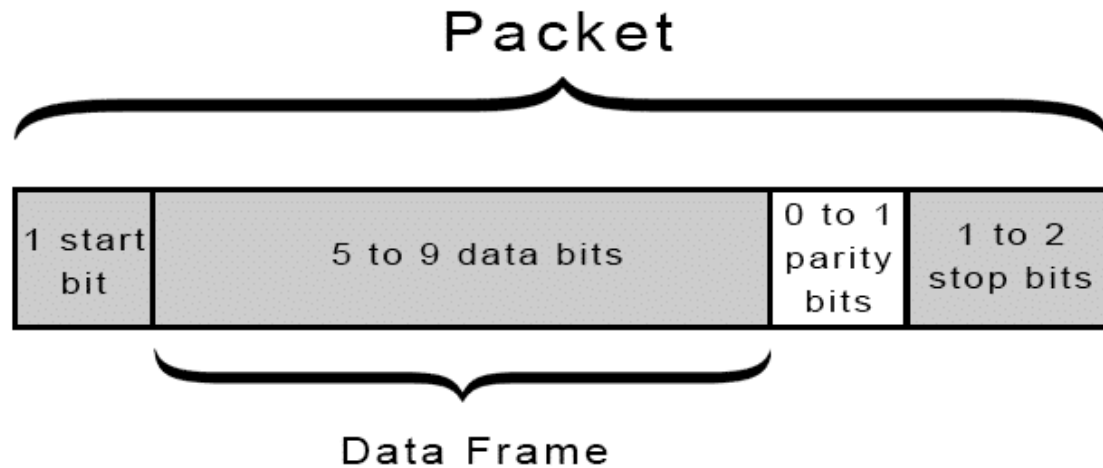
Wires Used	2
Maximum Speed	Any speed up to 115200 baud, usually 9600 baud
Synchronous or Asynchronous?	Asynchronous
Serial or Parallel?	Serial
Max # of Masters	1
Max # of Slaves	1

b. How it work?

The UART that is going to transmit data receives the data from a data bus. The data bus is used to send data to the UART by another device like a CPU, memory, or microcontroller. Data is transferred from the data bus to the transmitting UART in parallel form. After the transmitting UART gets the parallel data from the data bus, it adds a start bit, a parity bit, and a stop bit, creating the data packet. Next, the data packet is output serially, bit by bit at the Tx pin. The receiving UART reads the data packet bit by bit at its Rx pin. The receiving UART then converts the data back into parallel form and removes the start bit, parity bit, and stop bits. Finally, the receiving UART transfers the data packet in parallel to the data bus on the receiving end:



UART transmitted data is organized into *packets*. Each packet contains 1 start bit, 5 to 9 data bits (depending on the UART), an optional *parity* bit, and 1 or 2 stop bits:



c. Advantages and disadvantages of Uarts

No communication protocol is perfect, but UARTs are pretty good at what they do. Here are some pros and cons to help you decide whether or not they fit the needs of your project:

❖ Advantage:

- Only uses two wires
- No clock signal is necessary
- Has a parity bit to allow for error checking
- The structure of the data packet can be changed as long as both sides are set up for it
- Well documented and widely used method

❖ Disadvantages:

- The size of the data frame is limited to a maximum of 9 bits
- Doesn't support multiple slave or multiple master systems
- The baud rates of each UART must be within 10% of each other

2. Usage

This example will help you control LED on ESP8266 board by UART protocols.

a. Prepare hardware and software

❖ Hardware:

- + 2 x Expressif ESP8266 board
- + 2 x Micro USB cable
- + 2 x Female – female jumper wires

❖ Software:

- + Arduino IDE

b. Connect 2 ESP8266 board

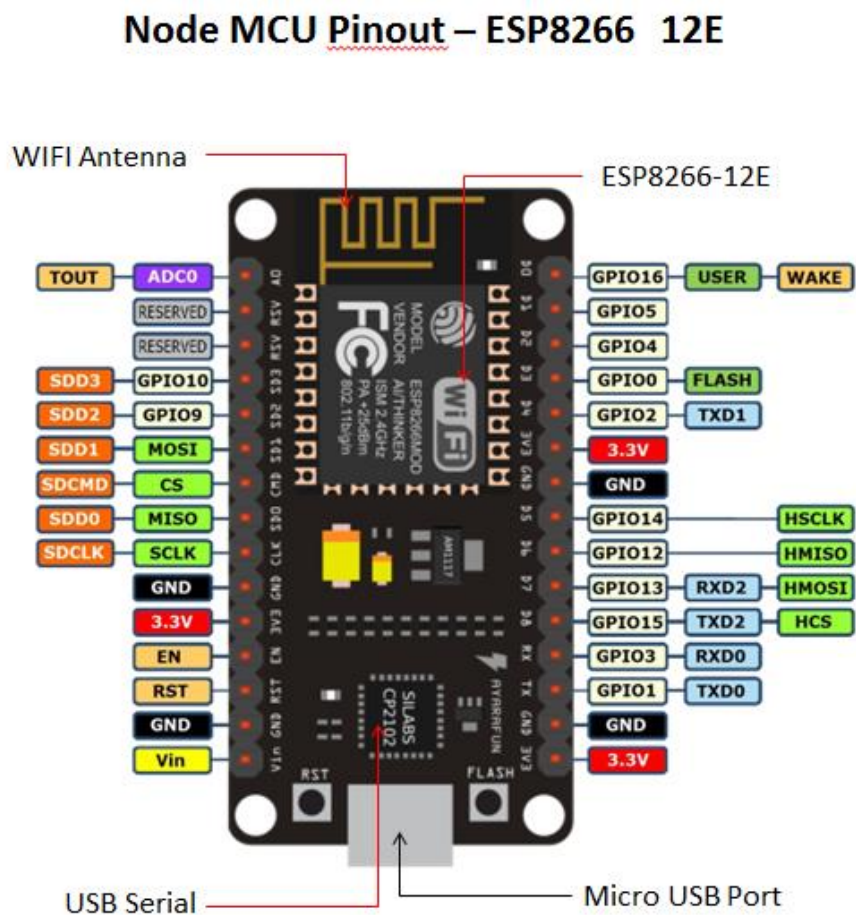


Figure 1: ESP8266 pinout

ESP8266 (Send)	ESP8266 (Receive)
GND	GND
TXD0	RXD0

c. Upload sketch

- Checkout materials and samples code from gitlab
- Remove TX RX jumper wires on 2 board before upload sketch.
- Compile and Upload sketch to 2 ESP8266 board.

d. Check result

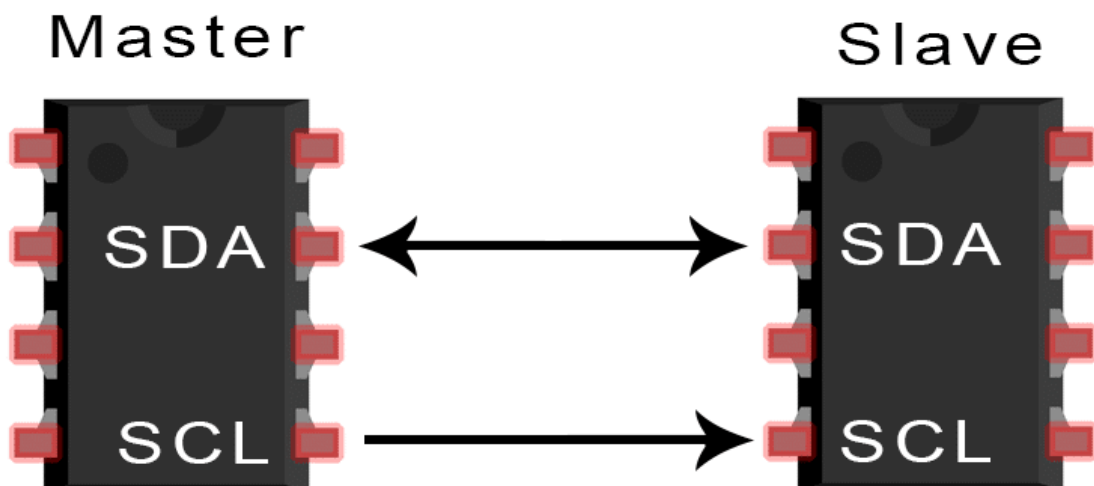
CHAPTER 3: I2C

1. What is I2C?

a. Overview

I2C combines the best features of SPI and UARTs. With I2C, you can connect multiple slaves to a single master (like SPI) and you can have multiple masters controlling single, or multiple slaves. This is really useful when you want to have more than one microcontroller logging data to a single memory card or displaying text to a single LCD.

Like UART communication, I2C only uses two wires to transmit data between devices:



SDA (Serial Data) – The line for the master and slave to send and receive data.

SCL (Serial Clock) – The line that carries the clock signal.

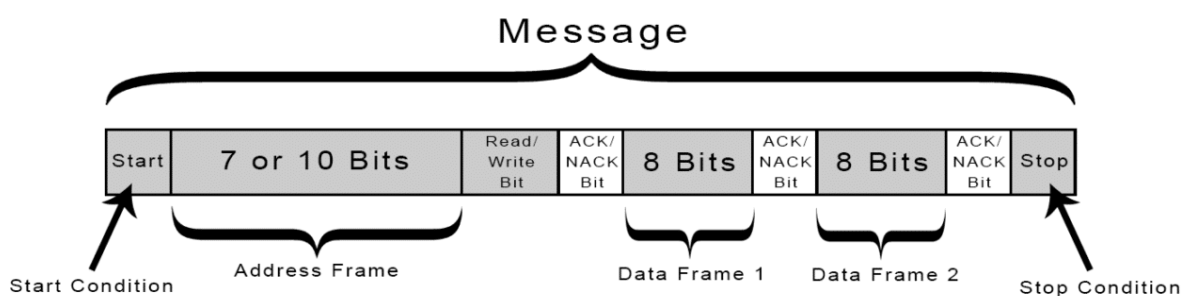
I2C is a serial communication protocol, so data is transferred bit by bit along a single wire (the SDA line).

Like SPI, I2C is synchronous, so the output of bits is synchronized to the sampling of bits by a clock signal shared between the master and the slave. The clock signal is always controlled by the master.

Wires Used	2
Maximum Speed	Standard mode= 100 kbps
	Fast mode= 400 kbps
	High speed mode= 3.4 Mbps
	Ultra fast mode= 5 Mbps
Synchronous or Asynchronous?	Synchronous
Serial or Parallel?	Serial
Max # of Masters	Unlimited
Max # of Slaves	1008

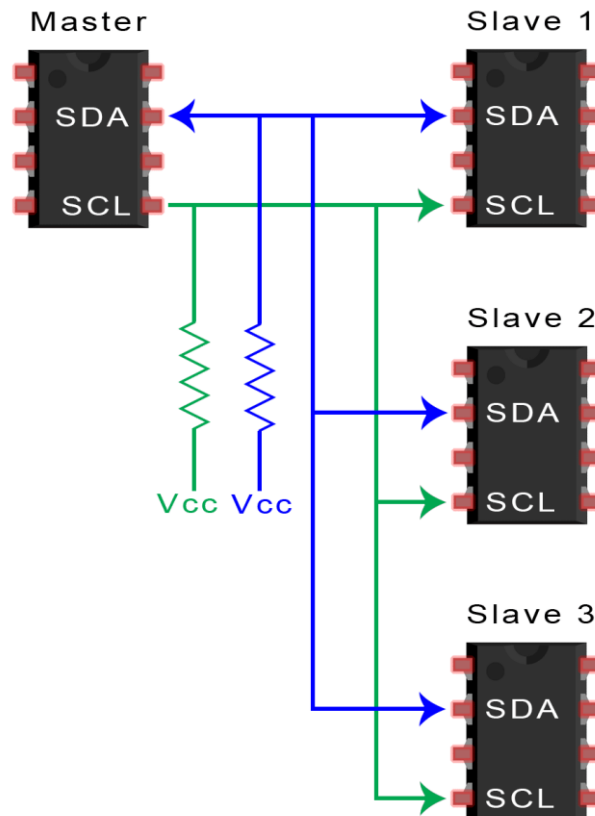
b. How it work?

With I2C, data is transferred in *messages*. Messages are broken up into *frames* of data. Each message has an address frame that contains the binary address of the slave, and one or more data frames that contain the data being transmitted. The message also includes start and stop conditions, read/write bits, and ACK/NACK bits between each data frame:



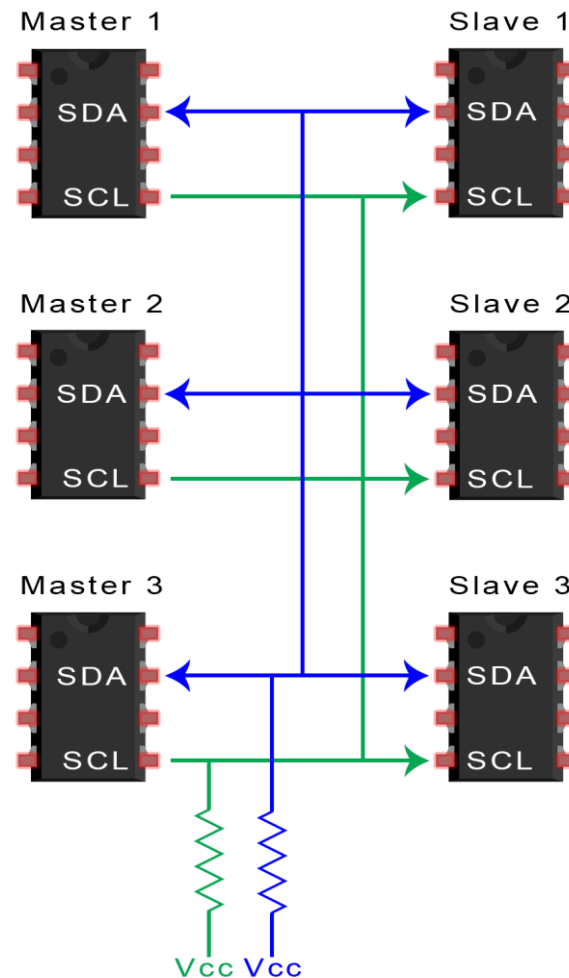
❖ Single master with multiple slaves

Because I2C uses addressing, multiple slaves can be controlled from a single master. With a 7 bit address, 128 (2^7) unique address are available. Using 10 bit addresses is uncommon, but provides 1,024 (2^{10}) unique addresses. To connect multiple slaves to a single master, wire them like this, with 4.7K Ohm pull-up resistors connecting the SDA and SCL lines to Vcc:



❖ Multiple masters with multiple slaves

Multiple masters can be connected to a single slave or multiple slaves. The problem with multiple masters in the same system comes when two masters try to send or receive data at the same time over the SDA line. To solve this problem, each master needs to detect if the SDA line is low or high before transmitting a message. If the SDA line is low, this means that another master has control of the bus, and the master should wait to send the message. If the SDA line is high, then it's safe to transmit the message. To connect multiple masters to multiple slaves, use the following diagram, with 4.7K Ohm pull-up resistors connecting the SDA and SCL lines to Vcc:



c. Advantages and disadvantages of I2C

There is a lot to I2C that might make it sound complicated compared to other protocols, but there are some good reasons why you may or may not want to use I2C to connect to a particular device:

❖ Advantages:

- Only uses two wires
- Supports multiple masters and multiple slaves
- ACK/NACK bit gives confirmation that each frame is transferred successfully
- Hardware is less complicated than with UARTs
- Well known and widely used protocol

❖ **Disadvantages:**

- Slower data transfer rate than SPI
- The size of the data frame is limited to 8 bits
- More complicated hardware needed to implement than SPI

2. Usage:

This example will help you to read the information of RFID tags by SPI protocols.

a. Prepare hardware and software

❖ **Hardware:**

- + 1 x Expressif ESP32 board
- + 1 x Micro USB cable
- + 1 x RFID – MFRC522 module
- + 1 x RFID tag or label
- + 7 x Female – female jumper wires

❖ **Software:**

- + Arduino IDE
- + RFID RC522 library ()

b. Connect RFID module with ESP32

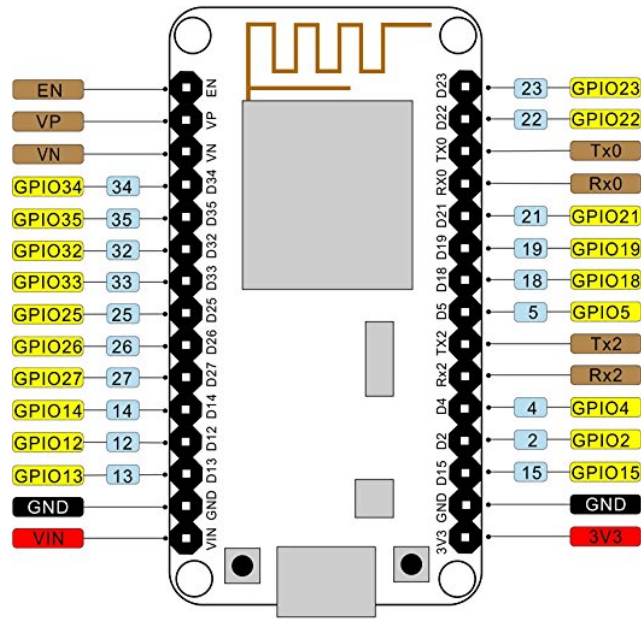


Figure 2: ESP32 pinout



Figure 3: RC522 pinout

RC522	ESP32
SS	5
SCK	18
MOSI	23
MISO	19
IRQ	NC
GND	GND
RST	0(EN)
VCC	3v3

c. Upload RFIDReader sketch

- Checkout materials and samples code from gitlab

- Navigate to Samples code and open **RFIDReader** sketch
- Compile and Upload sketch to ESP32 board.

d. Check result

- Use RFID tag to touch the RFID module you can see the informations of RFID tag in Serial Monitors.

CHAPTER 4: SPI

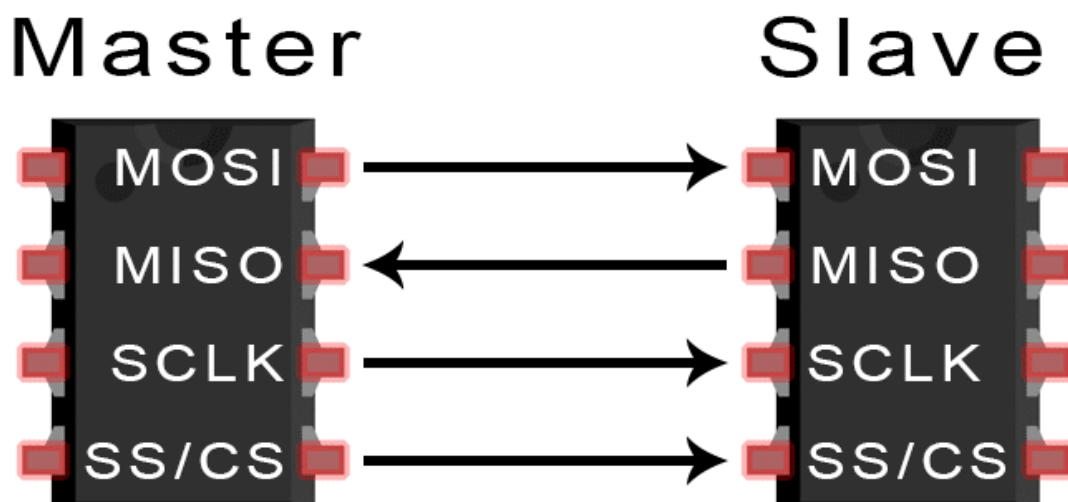
1. What is SPI?

a. Overview

SPI is a common communication protocol used by many different devices. For example, , , and all use SPI to communicate with microcontrollers.

One unique benefit of SPI is the fact that data can be transferred without interruption. Any number of bits can be sent or received in a continuous stream. With I2C and UART, data is sent in packets, limited to a specific number of bits. Start and stop conditions define the beginning and end of each packet, so the data is interrupted during transmission.

Devices communicating via SPI are in a master-slave relationship. The master is the controlling device (usually a microcontroller), while the slave (usually a sensor, display, or memory chip) takes instruction from the master. The simplest configuration of SPI is a single master, single slave system, but one master can control more than one slave (more on this below).



MOSI (Master Output/Slave Input) – Line for the master to send data to the slave.

MISO (Master Input/Slave Output) – Line for the slave to send data to the master.

SCLK (Clock) – Line for the clock signal.

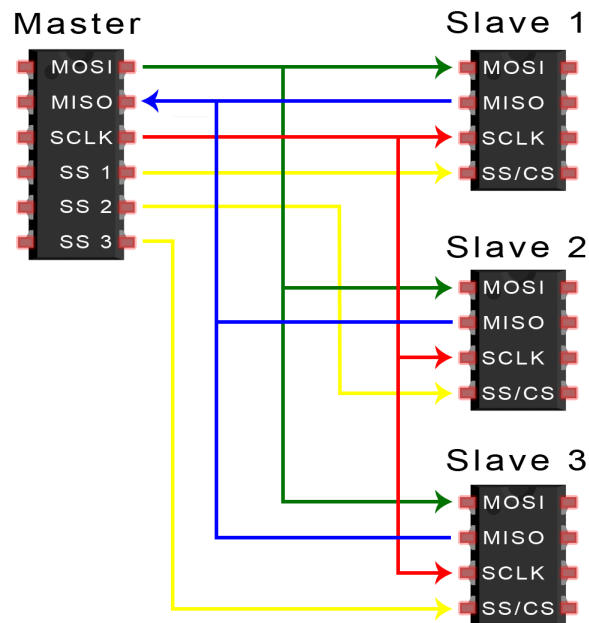
SS/CS (Slave Select/Chip Select) – Line for the master to select which slave to send data to.

Wires Used	4
Maximum Speed	Up to 10 Mbps
Synchronous or Asynchronous?	Synchronous
Serial or Parallel?	Serial
Max # of Masters	1
Max # of Slaves	Theoretically unlimited*

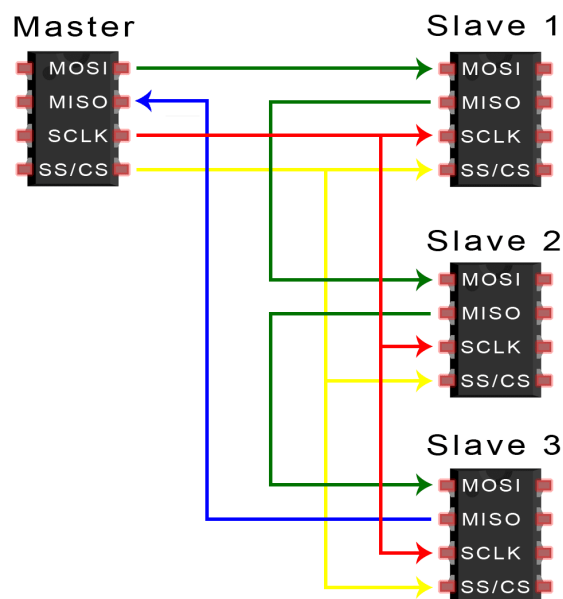
*In practice, the number of slaves is limited by the load capacitance of the system, which reduces the ability of the master to accurately switch between voltage levels.

b. How it works?

SPI can be set up to operate with a single master and a single slave, and it can be set up with multiple slaves controlled by a single master. There are two ways to connect multiple slaves to the master. If the master has multiple slave select pins, the slaves can be wired in parallel like this:



If only one slave select pin is available, the slaves can be daisy-chained like this:



c. Advantages and disadvantages of SPI

There are some advantages and disadvantages to using SPI, and if given the choice between different communication protocols, you should know when to use SPI according to the requirements of your project:

❖ Advantages:

- No start and stop bits, so the data can be streamed continuously without interruption
- No complicated slave addressing system like I2C
- Higher data transfer rate than I2C (almost twice as fast)
- Separate MISO and MOSI lines, so data can be sent and received at the same time

❖ **Disadvantages:**

- Uses four wires (I2C and UARTs use two)
- No acknowledgement that the data has been successfully received (I2C has this)
- No form of error checking like the parity bit in UART
- Only allows for a single master

2. Usage:

This example will help you use I2C protocol to connect OLED screen..

a. Prepare hardware and software

❖ **Hardware:**

- + 1 x Expressif ESP32 board
- + 1 x Micro USB cable
- + 1 x
- + 4 x Female – female jumper wires

❖ **Software:**

- + Arduino IDE
- + library ()

b. Connect OLED module with ESP32

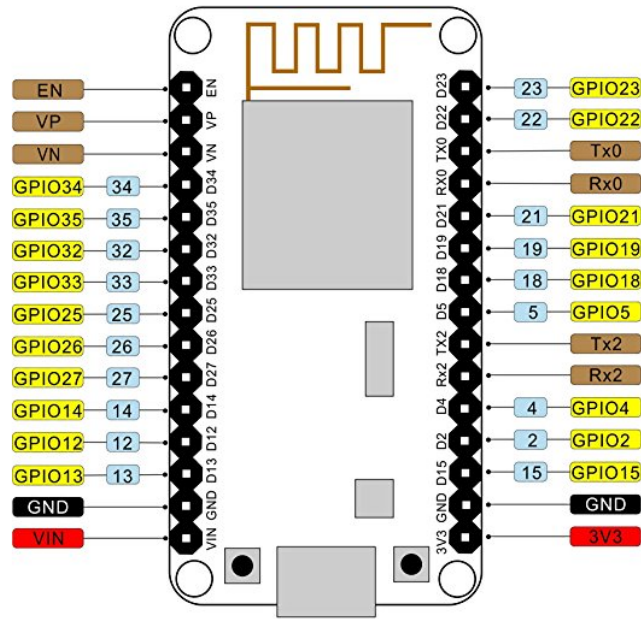


Figure 4: ESP32 pinout

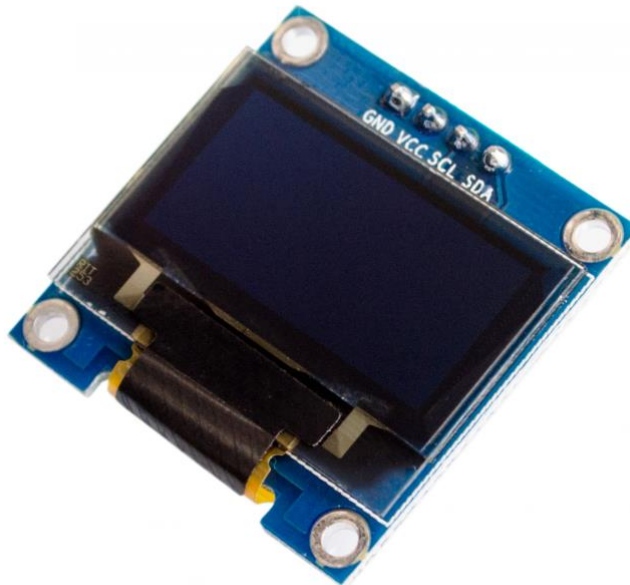


Figure 5: OLED

OLED	ESP32
VCC	VIN
GND	GND
SCL	22

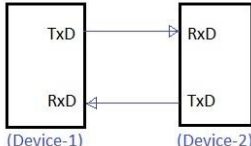
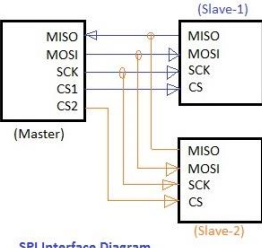
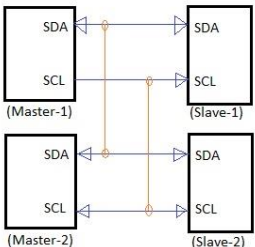
SDA	21
-----	----

c. Upload SSD1306SimpleCode sketch

- Navigate to Samples code and open **SSD1306SimpleCode** sketch.
- Edit “SSD1306 display(0x3c, D3, D5);” -> “SSD1306 display(0x3c, 21, 22);”
- Compile and Upload sketch to ESP32 board.

d. Check result

CHAPTER 5: DIFFERENCE BETWEEN UART, SPI, I2C

Features	UART	SPI	I2C
Full Form	Universal Asynchronous Receiver/Transmitter	Serial Peripheral Interface	Inter-Integrated Circuit
Interface Diagram	 <p style="text-align: center;"><u>UART Interface Diagram</u></p>	 <p style="text-align: center;"><u>SPI Interface Diagram</u></p>	 <p style="text-align: center;"><u>I2C Interface Diagram</u></p>
Pin Designations	TxD: Transmit Data RxD: Receive Data	SCLK: Serial Clock MOSI: Master Output, Slave Input MISO: Master Input, Slave Output SS: Slave Select	SDA: Serial Data SCL: Serial Clock
Data rate	As this is asynchronous communication, data rate between two devices wanting to communicate should be set to equal value. Maximum data rate supported is about 230 Kbps to 460kbps.	Maximum data rate limit is not specified in SPI interface. Usually supports about 10 Mbps to 20 Mbps	I2C supports 100 kbps, 400 kbps, 3.4 Mbps. Some variants also supports 10 Kbps and 1 Mbps.
Distance	Lower about 50 feet	highest	Higher

Type of communication	Asynchronous	Synchronous	Synchronous
Number of masters	Not Application	One	One or more than One
Clock	No Common Clock signal is used. Both the devices will use there independent clocks.	There is one common serial clock signal between master and slave devices.	There is common clock signal between multiple masters and multiple slaves.
Hardware complexity	lesser	less	more
Protocol	For 8 bits of data one start bit and one stop bit is used.	Each company or manufacturers have got their own specific protocols to communicate with peripherals. Hence one needs to read datasheet to know read/write protocol for SPI communication to be established. For example we would like SPI communication between microcontroller and EPROM. Here one need to go through read/write operational diagram in the EPROM data sheet.	It uses start and stop bits. It uses ACK bit for each 8 bits of data which indicates whether data has been received or not. Figure depicts the data communication protocol.
Software addressing	As this is one to one connection between two devices,	Slave select lines are used to address any particular slave connected with the	There will be multiple slaves and multiple masters and all masters can communicate with all the slaves. Upto 27 slave

	addressing is not needed.	master. There will be 'n' slave select lines on master device for 'n' slaves.	devices can be connected/addressed in the I2C interface circuit.
Advantages	<ul style="list-style-type: none"> • It is simple communication and most popular which is available due to UART support in almost all the devices with 9 pin connector. It is also referred as RS232 interface. 	<ul style="list-style-type: none"> • It is simple protocol and hence so not require processing overheads. • Supports full duplex communication. • Due to separate use of CS lines, same kind of multiple chips can be used in the circuit design. • SPI uses push-pull and hence higher data rates and longer ranges are possible. • SPI uses less power compare to I2C 	<ul style="list-style-type: none"> • Due to open collector design, limited slew rates can be achieved. • More than one masters can be used in the electronic circuit design. • Needs fewer i.e. only 2 wires for communication. • I2C addressing is simple which does not require any CS lines used in SPI and it is easy to add extra devices on the bus. • It uses open collector bus concept. Hence there is bus voltage flexibility on the interface bus. • Uses flow control.
Disadvantages	<ul style="list-style-type: none"> • They are suitable for communication between only two devices. • It supports fixed data rate agreed upon between devices initially before communication otherwise data will be garbled. 	<ul style="list-style-type: none"> • As number of slave increases, number of CS lines increases, this results in hardware complexity as number of pins required will increase. • To add a device in SPI requires one to add extra CS line and changes in software for particular device addressing is 	<ul style="list-style-type: none"> • Increases complexity of the circuit when number of slaves and masters increases. • I2C interface is half duplex. • Requires software stack to control the protocol and hence it needs some processing overheads on microcontroller/microprocessor

		<p>concerned.</p> <ul style="list-style-type: none">•Master and slave relationship can not be changed as usually done in I2C interface.•No flow control available in SPI.	
--	--	--	--