← Go Back

Learn

Arduino Ecosystem

Microcontrollers

Programming

Electronics

Communication

Inter-Integrated Circuit (I2C) Protocol

Arduino & Serial Peripheral Interface (SPI)

LPWAN (Low-Power Wide-Area Networks) 101

GPS NMEA 0183 Messaging Protocol 101

The Arduino Guide to LoRa® and LoRaWAN®

1-Wire Protocol

Arduino® & Modbus Protocol

Bluetooth® Low Energy

Universal Asynchronous Receiver-Transmitter (UART)

Home / Learn / Inter-Integrated Circuit (I2C) Protocol

Inter-Integrated Circuit (I2C) Protocol

Allows the communication between devices or sensors connected via Two Wire Interface Bus.

Author · Nicholas Zambetti, Karl Söderby, Jacob Hylén Last revision · 11/07/2024

Introduction

A good way of adding complexity of features to your projects without adding complexity of wiring, is to make use of the Inter-integrated circuit (I2C) protocol. The I2C protocol is supported on all Arduino boards. It allows you to connect several peripheral devices, such as sensors, displays, motor drivers, and so on, with only a few wires. Giving you lots of flexibility and speeding up your prototyping, without an abundancy of wires. Keep reading to learn about how it works, how it is implemented into different standards, as well as how to use the Wire Library to build your own I2C devices.

What Is I2C?

The I2C protocol involves using two lines to send and receive data: a serial clock pin **(SCL)** that the Arduino Controller board pulses at a regular interval, and a serial data pin **(SDA)** over which data is sent between the two devices.

In I2C, there is one controller device, with one or more peripheral devices connected to the controllers SCL and SDA lines.

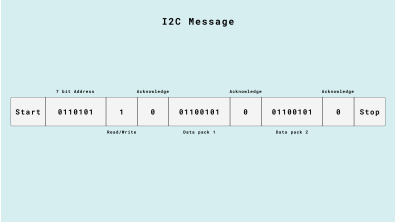
As the clock line changes from low to high (known as the rising edge of the clock pulse), a single bit of information is transferred from the board to the I2C device over the SDA line. As the clock line keeps pulsing, more and more bits are sent until a sequence of a 7 or 8 bit address, and a command or data is formed. When this information is sent - bit after bit - the called

it's data back - if required - to the board over the same line using the clock signal still generated by the Controller on SCL as timing.

Each device in the I2C bus is functionally independent from the controller, but will respond with information when prompted by the controller.

Because the I2C protocol allows for each enabled device to have it's own unique address, and as both controller and peripheral devices to take turns communicating over a single line, it is possible for your Arduino board to communicate (in turn) with many devices, or other boards, while using just two pins of your microcontroller.

An I2C message on a lower bit-level looks something like this:



An I2C Message

The controller sends out instructions through the I2C bus on the data pin (SDA), and the instructions are prefaced with the address, so that only the correct device listens.

Then there is a bit signifying whether the controller wants to read or write.

Every message needs to be acknowledged, to combat unexpected results, once the receiver has acknowledged the previous information it lets the controller know, so it can move on to the next set of bits.

8 bits of data

Another acknowledgement bit

8 bits of data

Another acknowledgement bit

But how does the controller and peripherals know where the address, messages, and so on starts and ends? That's what the SCL wire is for. It synchronises the clock of the controller with the devices, ensuring that they all move to the next instruction at the same time.

need to consider any of this, in the Arduino ecosystem we have the **Wire library** that handles everything for you.

Arduino I2C Pins

Below is a table that lists the different board form factors and what pins are for I2C.

Form factor	SDA	SCL	SDA1	SCL1	SDA2	SCL2
UNO	SDA/ A4	SCL/ A5				
Nano	A4	A5				
MKR	D11	D12				
GIGA	D20	D21	D102	D101	D9	D8
Mega & Due	D20	D21				

I2C Wiring

Below you'll find a couple ways to wire I2C breakout modules. Which way is best depends on each module, and your needs.

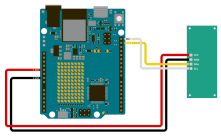
Breakout Boards

Some breakout board modules let you wire them directly, with bare wires on a breadboard. To connect a module like this to your Arduino board, connect it as follows:

- VCC* - 5V/3V3 pin (depending on breakout module)
- GND* - GND
- SDA - SDA
- SCL - SCL

*Pin name might vary depending on what module, VCC might be named "VIN", "+", etc. GND might be named "-".

Here's an example of how you might connect a sensor to an UNO R4 WiFi:

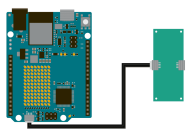


Bare I2C Wiring on UNO R4 WiFi

Qwiic & STEMMA QT

When delving into the market of breakout modules and sensors, you'll find that there are entire ecosystems, where standards are built around the I2C protocol. Examples of such standards are Qwiic, developed by Sparkfun, and STEMMA QT, developed by Adafruit. Both Qwiic and STEMMA QT use a 4-pin JST SH connector for I2C devices, making it easier for third parties to design hardware with vast compatibility. By having a standardized connector, you'll know that if you see the word Qwiic or STEMMA QT in association with an item, that it will work together with an Arduino board with a Qwiic or STEMMA QT connector, such as the UNO R4 WiFi.

Both Qwiic and STEMMA QT bundle together wires for power, ground, as well as the SDA and SCL wires for I2C, making it a complete kit, one cable that bundles everything together.



I2C on a Qwiic/STEMMA QT connector with UNO R4 WiFi

But what's the difference between the two?

Both Qwiic and STEMMA QT use I2C, and even when inspecting modules using the two standards up close, it may be difficult to tell what makes them unique from each other. But there is a difference! And it has some implications on how and for what you may use

Qwiic has level shifting and voltage regulation on the controller (but not on the peripherals). What this means is that Qwiic is 3.3 V logic **only**. This makes it easier to use, as for the end user, there is one less thing that can go wrong when designing and assembling your circuit.

STEMMA QT, on the other hand, doesn't have this. This lets you use both 3.3 V and 5 V logic for modules. This also means that there is one more thing you may need to consider when creating your circuit, but it also grants some more flexibility in power and logic requirements.

The pin order for STEMMA QT is designed to match the pin order for Qwiic, enabling cross-compatibility between the two standards.

Grove

Grove is another connector standard, this one developed by seeed studio. You can find a plethora of modules with a Grove connector, however only some of them use I2C. There are no Arduino boards that have a built in Grove connector, however you can use products such as the **MKR connector carrier**, **Nano Grove Shield**, or the Base Shield from the **Arduino Sensor Kit** to connect Grove sensors to your Arduino board.

Wire Library

The Wire library is what Arduino uses to communicate with I2C devices. It is included in all board packages, so you don't need to install it manually in order to use it.

To see the full API for the Wire library, visit its **documentation page**.

- `begin()` - Initialise the I2C bus
- `end()` - Close the I2C bus
- `requestFrom()` - Request bytes from a peripheral device
- `beginTransmission()` - Begins queueing up a transmission
- `endTransmission()` - Transmit the bytes that have been queued and end the transmission
- `write()` - Writes data from peripheral to

bytes available for retrieval

`read()` - Reads a byte that was transmitted from a peripheral to a controller.

`setClock()` - Modify the clock frequency

`onReceive()` - Register a function to be called when a peripheral receives a transmission

`onRequest()` - Register a function to be called when a controller requests data

`setWireTimeout()` - Sets the timeout for transmissions in controller mode

`clearWireTimeoutFlag()` - Clears the timeout flag

`getWireTimeoutFlag()` - Checks whether a timeout has occurred since the last time the flag was cleared.

Derived libraries

When you buy basically any breakout module that makes use of the I2C protocol, they will come with some library that helps you use the sensor. This library is more often than not built on top of the Wire library, and uses it under the hood. Adding functionality in order to make, for example, reading temperature easier.

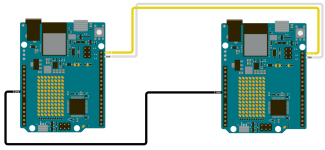
An example of this is if you want to use Adafruits MCP9808 sensor module, you download the Adafruit_MCP9808 Library from the IDEs library manager, which enables you to use functions such as `tempsensor.readTempC()` in order to read the sensors temperature data by requesting from the right address, and read the information returned with just a single line instead of writing the Wire code yourself.

To learn how to install libraries, check out our **guide to installing libraries**.

Examples

The remainder of this article is a collection of examples that can get you off the ground with I2C.

Controller Reader



Arduino Boards connected via I2C

In some situations, it can be helpful to set up two (or more!) Arduino boards to share information with each other. In this example, two boards are programmed to communicate with one another in a Controller Reader/ Peripheral Sender configuration via the **I2C synchronous serial protocol**. Several functions of Arduino's **Wire Library** are used to accomplish this. Arduino 1, the Controller, is programmed to request, and then read, 6 bytes of data sent from the uniquely addressed Peripheral Arduino. Once that message is received, it can then be viewed in the Arduino Software (IDE) serial monitor window.

Controller Reader Sketch

COPY

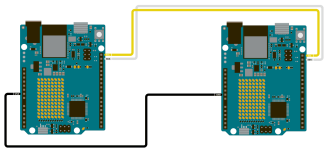
```
// Wire Controller Reader
// by Nicholas Zambetti [http://ww

// Demonstrates use of the Wire li
// Reads data from an I2C/TWI peri
// Refer to the "Wire Peripheral S

// Created 29 March 2006

// This example code is in the pub


#include <Wire.h>

void setup() {
  Wire.begin();        // join i2c
  Serial.begin(9600);  // start se
}

void loop() {
  Wire.requestFrom(8, 6);    // re

  while (Wire.available()) { // pe
    char c = Wire.read(); // recei
    Serial.print(c);         // pr
  }

  delay(500);
}
```

Peripheral Sender Sketch

COPY

```
// Wire Peripheral Sender
// by Nicholas Zambetti [http://ww

// Demonstrates use of the Wire li
// Sends data as an I2C/TWI periph
// Refer to the "Wire Master Reade

// Created 29 March 2006

// This example code is in the pub


#include <Wire.h>

void setup() {
  Wire.begin(8);                //
  Wire.onRequest(requestEvent); //
}

void loop() {
  delay(100);
}

// function that executes whenever
// this function is registered as
void requestEvent() {
  Wire.write("hello "); // respond
  // as expected by master
}
```

Controller Writer



Arduino Boards connected via I2C

In some situations, it can be helpful to set up two (or more!) Arduino boards to share information with each other. In this example, two boards are programmed to communicate with one another in a Controller Writer/ Peripheral Receiver configuration via the **I2C synchronous serial protocol**. Several functions of Arduino's **Wire Library** are used to accomplish this. Arduino 1, the Controller, is programmed to send 6 bytes of data every half second to a uniquely addressed Peripheral

viewed in the Peripheral board's serial monitor window opened on the USB connected computer running the Arduino Software (IDE).

Controller Writer Sketch

COPY

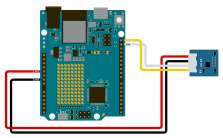
```
// Wire Master Writer
// by Nicholas Zambetti [http://ww

// Demonstrates use of the Wire li
// Writes data to an I2C/TWI Perip
// Refer to the "Wire Peripheral R

// Created 29 March 2006

// This example code is in the pub


#include <Wire.h>

void setup()
{
  Wire.begin(); // join i2c bus (a
}

byte x = 0;

void loop()
{
  Wire.beginTransmission(4); // tr
  Wire.write("x is ");        // s
  Wire.write(x);              // s
  Wire.endTransmission();    // st

  x++;
  delay(500);
```

Peripheral Receiver Sketch

COPY

```
// Wire Peripheral Receiver
// by Nicholas Zambetti [http://ww

// Demonstrates use of the Wire li
// Receives data as an I2C/TWI Per
// Refer to the "Wire Master Write

// Created 29 March 2006

// This example code is in the pub


#include <Wire.h>

void setup()
{
  Wire.begin(4);                //
  Wire.onReceive(receiveEvent); //
  Serial.begin(9600);           //
}

void loop()
{
  delay(100);
}

// function that executes whenever
// this function is registered as
void receiveEvent(int howMany)
{
```

Accelerometer



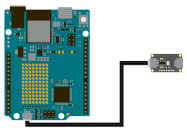
Grove IMU over I2C

This code lets you read accelerometer data from a **Grove 6-Axis Accelerometer module** using the **seeed arduino LSM6DS3 library**.

COPY

```
#include "LSM6DS3.h"
#include "Wire.h"

//Create instance of Accelerometer
LSM6DS3 accelerometer(I2C_MODE, 0x

void setup() {
    // put your setup code here, t
    Serial.begin(9600);
    while (!Serial);

    if (accelerometer.begin() != 0
        Serial.println("LSM6DS3 nc
    } else {
        Serial.println("LSM6DS3 fc
    }
}

void loop() {
    //Gyroscope
    Serial.print("\nGyroscope:\n")
    Serial.print(" X1 = ");
    Serial.println(accelerometer.r
    Serial.print(" Y1 = ");
    Serial.println(accelerometer.r
    Serial.print(" Z1 = ");
    Serial.println(accelerometer.r

    //Accelerometer
    Serial.print("\nAccelerometer:
```

I2C BMP280



Qwiic BMP280 module

This code example lets you read the temperature over I2C from a BMP280 breakout module from Adafruit:

DOCS

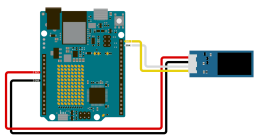
/

COPY

```
#include <Wire.h>
#include <Adafruit_BMP280.h>

//Create an instance of the BMP280
Adafruit_BMP280 bmp;

void setup() {
  Serial.begin(9600);

  // Start the sensor, and verify
  if (!bmp.begin()) {
    Serial.println("Sensor not fou
    while (1){}
  }

}

void loop() {
  // Read the values
  float temperature = bmp.readTemp

  // Print to the Serial Monitor
  Serial.print("Temperature: ");
  Serial.print(temperature);
  Serial.println(" C");

  Serial.println();
  delay(2000);
}
```

I2C OLED

Was



Grove OLED over I2C

This code example draws a version of the Arduino logo on a 128x64 I2C Grove OLED:

DOCS

/

COPY

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

Adafruit_SSD1306 display(128, 64,

// The Arduino Logo in a bitmap fo
const uint8_t arduinoLogo[] = {
  0x00, 0x00, 0x00, 0x00, 0x00, 0x
  0x00, 0x00, 0x00, 0x00, 0x07, 0x
  0x00, 0x00, 0x00, 0x00, 0x3f, 0x
  0x00, 0x00, 0x00, 0x01, 0xff, 0x
  0x00, 0x00, 0x00, 0x07, 0xff, 0x
  0x00, 0x00, 0x00, 0x0f, 0xff, 0x
  0x00, 0x00, 0x00, 0x3f, 0xff, 0x
  0x00, 0x00, 0x00, 0x7f, 0xff, 0x
  0x00, 0x00, 0x00, 0xff, 0xff, 0x
  0x00, 0x00, 0x01, 0xff, 0xfc, 0x
  0x00, 0x00, 0x01, 0xff, 0xc0, 0x
  0x00, 0x00, 0x03, 0xff, 0x80, 0x
  0x00, 0x00, 0x07, 0xfe, 0x00, 0x
  0x00, 0x00, 0x07, 0xfc, 0x00, 0x
  0x00, 0x00, 0x0f, 0xf8, 0x00, 0x
  0x00, 0x00, 0x0f, 0xf0, 0x00, 0x
  0x00, 0x00, 0x0f, 0xf0, 0x00, 0x
  0x00, 0x00, 0x1f, 0xe0, 0x00, 0x
  0x00, 0x00, 0x1f, 0xe0, 0x00, 0x
  0x00, 0x00, 0x1f, 0xe0, 0x00, 0x
  0x00, 0x00, 0x1f, 0xc0, 0x00, 0x
  0x00, 0x00, 0x1f, 0xc0, 0x1f, 0x
```

Suggest changes

The content on docs.arduino.cc is facilitated through a public GitHub repository. If you see anything wrong, you can edit this page here.

Need support?

Help Center
Ask the Arduino Forum
Discover Arduino Discord

License