Arduino - I2C

Définition

Le bus I2C (Inter-Integrated Circuit) est un bus de communication série utilisé pour interconnecter plusieurs composants électroniques sur une carte ou un circuit imprimé. Il a été développé par Philips (aujourd'hui NXP) dans les années 1980 et est largement utilisé dans les systèmes embarqués, y compris les cartes Arduino.

Le bus I2C utilise deux lignes principales pour la communication :

1. SDA (Serial Data Line) : C'est la ligne de données série bidirectionnelle utilisée pour transmettre les informations entre les périphériques. Les données sont transmises sous forme de bits séquentiels sur cette ligne.
2. SCL (Serial Clock Line) : C'est la ligne d'horloge série utilisée pour synchroniser les transferts de données entre les périphériques. L'horloge est contrôlée par le maître et détermine le rythme des transferts de données sur le bus.

Le bus I2C permet de connecter plusieurs périphériques sur les mêmes lignes de données (SDA) et d'horloge (SCL). Chaque périphérique est identifié par une adresse unique, ce qui permet au maître de sélectionner spécifiquement le périphérique avec lequel il souhaite communiquer.

Les échanges ont toujours lieu entre un seul maître et un (ou tous les) esclave(s), toujours à l'initiative du maître (jamais de maître à maître ou d'esclave à esclave). Cependant, rien n'empêche un composant de passer du statut de maître à esclave et réciproquement.

Le bus I2C prend en charge des vitesses de transmission variables, généralement de 100 kbps (kilobits par seconde) à 400 kbps, bien que des vitesses plus élevées (jusqu'à 3.4 Mbps) soient également possibles avec certaines implémentations.

Le nombre maximum d'équipements est limité par le nombre d'adresses disponibles, 7 bits d'adressage et un bit R/W (lecture ou écriture), soit 128 périphériques, mais il dépend également de la capacité (CB) du bus (dont dépend la vitesse maximale du bus). Il faut savoir que des adresses sont réservées pour diffuser des messages en broadcast et que de nombreuses adresses sont déjà attribuées par les fabricants ce qui limite grandement le nombre d'équipements (une variante d'adressage sur 10 bits existe également).

Le bus I2C est couramment utilisé pour connecter des capteurs, des écrans LCD, des mémoires EEPROM, des convertisseurs analogique-numérique (CAN), des circuits

intégrés et d'autres périphériques à un microcontrôleur ou à un processeur. Il permet une communication simple et efficace entre les différents composants d'un système électronique, tout en utilisant un nombre réduit de lignes de connexion.

Utilisation

pour utiliser le bus I2C sur Arduino, il faut importer la bibliothèque **Wire.h**

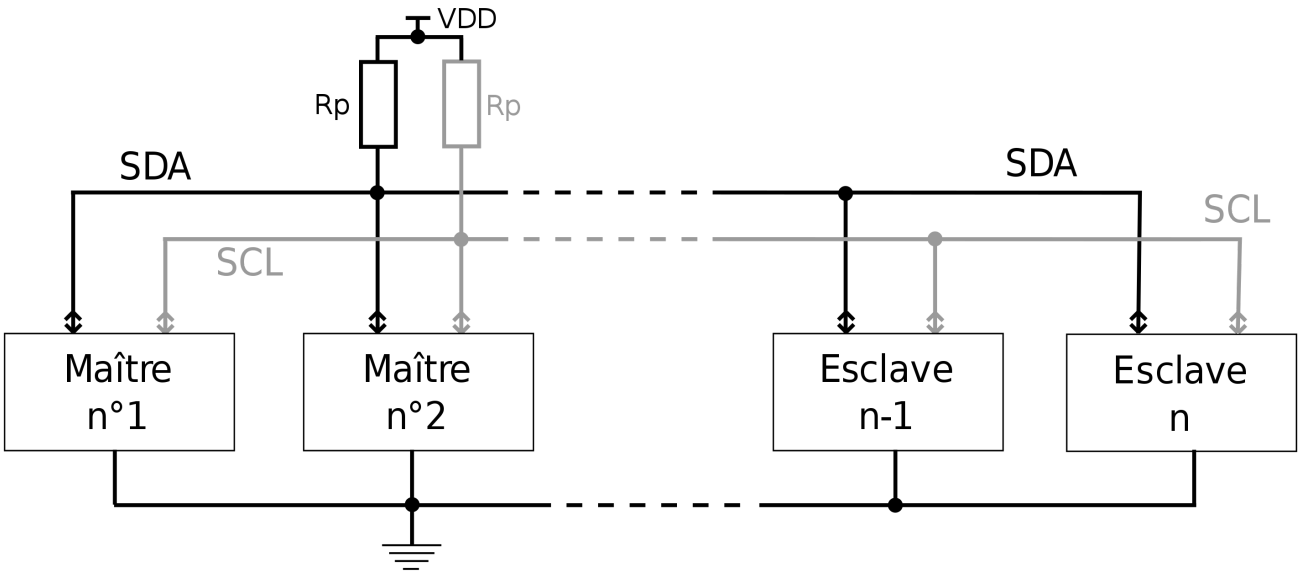
Câblage

L'I2C étant un bus, tout les participants doivent être relier sur les deux lignes SDA et SCL.
Les Arduino ont des broches dédiés pour cela.

La connexion est réalisée par l'intermédiaire de deux lignes :

- SDA (Serial Data Line) : ligne de données bidirectionnelle,
- SCL (Serial Clock Line) : ligne d'horloge de synchronisation bidirectionnelle.

Il ne faut également pas oublier **la masse qui doit être commune aux équipements.**
Les 2 lignes sont tirées au niveau de tension VDD à travers des résistances de pull-up (RP). Ceci n'est pas obligatoire pour les Arduino mais cela peut provoqué des instabilité du bus



source: https://fr.wikipedia.org/wiki/I2C#/media/Fichier:I2C_Architecture_2.svg

Pins

Voici un tableau récapitulatif des broches SDA et SCL pour différents modèles d'Arduino et les modules ESP :

Modèle	SDA	SCL
Arduino Uno, Nano, Mini	A4	A5

Modèle	SDA	SCL
Arduino Mega 2560, Mega ADK	20	21
Arduino Leonardo, Micro	2	3
Arduino Due	20	21
Arduino Zero, MKR1000, MKRZero	SDA	SCL
ESP8266 (NodeMCU, Wemos D1)	D2	D1
ESP32 (NodeMCU-32S, ESP-WROOM-32)	21	22

Initialisation et adresse

La première chose à faire est d'initialisé le bus sur chaque composants programmable (les composant comme les écrans ont eux une valeur fixe le plus souvent)
Un appel de la fonction `Wire.begin()` permet de le faire, sans argument l'Arduino sera le **maitre**, sinon, en mettant comme paramètre une adresse: `Wire.begin(address)` , celui-ci sera un **esclave**.
Les ESP ne peuvent jouer que le rôle de maître.
L'adresse est une valeur comprise en **0** et **127** (7 bits).

Pour connaitre l'adresse d'un composants non programmable comme un capteur, le programme I2C scanner, inclus dans la bibliothèque wire.h permet de connaitre toute les adresse utilisé actuellement sur le bus, et donc par extension de vérifier la bonne connexion du composant.

```c
// --------------------------------------
// i2c_scanner
//
// Version 1
//    This program (or code that looks like it)
//    can be found in many places.
//    For example on the Arduino.cc forum.
//    The original author is not known.
// Version 2, Juni 2012, Using Arduino 1.0.1
//     Adapted to be as simple as possible by Arduino.cc user
Krodal
// Version 3, Feb 26  2013
//    V3 by louarnold
// Version 4, March 3, 2013, Using Arduino 1.0.3
//    by Arduino.cc user Krodal.
//    Changes by louarnold removed.
//    Scanning addresses changed from 0...127 to 1...119,
//    according to the i2c scanner by Nick Gammon
//    https://www.gammon.com.au/forum/?id=10896
// Version 5, March 28, 2013
//    As version 4, but address scans now to 127.
//    A sensor seems to use address 120.
// Version 6, November 27, 2015.
//    Added waiting for the Leonardo serial communication.
//
//
// This sketch tests the standard 7-bit addresses
// Devices with higher bit address might not be seen
properly.
//

#include <Wire.h>

void setup() {
  Wire.begin();

  Serial.begin(9600);
  while (!Serial); // Leonardo: wait for Serial Monitor
  Serial.println("\nI2C Scanner");
}

void loop() {
```

```
  int nDevices = 0;

  Serial.println("Scanning...");

  for (byte address = 1; address < 127; ++address) {
    // The i2c_scanner uses the return value of
    // the Wire.endTransmission to see if
    // a device did acknowledge to the address.
    Wire.beginTransmission(address);
    byte error = Wire.endTransmission();

    if (error == 0) {
      Serial.print("I2C device found at address 0x");
      if (address < 16) {
        Serial.print("0");
      }
      Serial.print(address, HEX);
      Serial.println("  !");

      ++nDevices;
    } else if (error == 4) {
      Serial.print("Unknown error at address 0x");
      if (address < 16) {
        Serial.print("0");
      }
      Serial.println(address, HEX);
    }
  }
  if (nDevices == 0) {
    Serial.println("No I2C devices found\n");
  } else {
    Serial.println("done\n");
  }
  delay(5000); // Wait 5 seconds for next scan
}
```

Fonctions de contrôle du bus

Voici une liste des fonctions utilisées pour travailler avec le bus I2C sur Arduino:

Fonction	Description
`Wire.begin()`	Initialise la bibliothèque Wire pour utiliser le bus I2C.
`Wire.beginTransmission(address)`	Démarre une transmission vers un dispositif esclave spécifié par `address`.
`Wire.write(data)`	Envoie des données vers le dispositif esclave pendant une transmission en cours.
`Wire.endTransmission()`	Termine la transmission et libère le bus I2C.
`Wire.requestFrom(address, qty)`	Demande `qty` octets du dispositif esclave spécifié par `address` et prépare la lecture de ces octets.
`Wire.available()`	Renvoie le nombre d'octets disponibles à la lecture après une demande de données.
`Wire.read()`	Lit un octet reçu du bus I2C après une demande de données.

D'autre fonctions supplémentaires `onReceive` et `onRequest` associées à la communication I2C sur Arduino peuvent être utilisé sur un **esclave** pour associé à une demande de données ou une réception une fonction qui gérera cet événement.
Ces fonctions seront traiter comme des **interruptions**.

Fonction	Description
`Wire.onReceive(callback)`	Définit une fonction `callback` qui sera appelée lorsque des données sont reçues par l'Arduino via le bus I2C. Les données reçues peuvent être lues avec `Wire.read()`.
`Wire.onRequest(callback)`	Définit une fonction `callback` qui sera appelée lorsque l'Arduino reçoit une demande de données via le bus I2C. Cette fonction doit envoyer des données en utilisant `Wire.write()`.

Ces fonctions sont utilisées pour implémenter la communication bidirectionnelle avec d'autres périphériques I2C. La fonction `onReceive` est appelée lorsqu'un autre périphérique envoie des données à l'Arduino, tandis que la fonction `onRequest` est appelée lorsque l'Arduino reçoit une demande de données d'un autre périphérique.

Voici un exemple de leur utilisation :

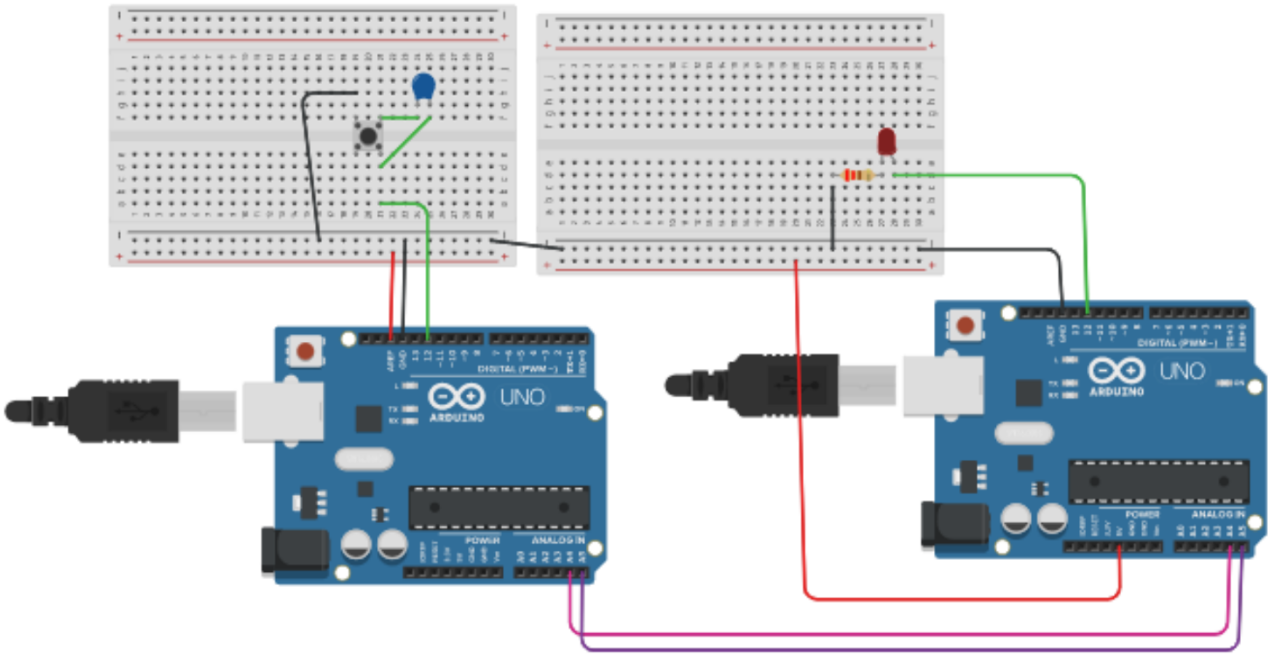
```c
#include <Wire.h>

void setup() {
  Wire.begin(8); // Initialise l'Arduino en tant qu'esclave I2C avec l'adresse 8
  Wire.onReceive(receiveEvent);
  Wire.onRequest(requestEvent);
}

void loop() {
  // Votre code principal
}

void receiveEvent(int numBytes) {
  while (Wire.available()) {
    // Lire les données reçues
    char receivedData = Wire.read();
    // Faites quelque chose avec les données
    // ...
  }
}

void requestEvent() {
  // Envoyer des données en réponse à une demande
  Wire.write("Hello from Arduino!");
}
```

Dans cet exemple, la fonction `receiveEvent` est appelée lorsque des données sont reçues par l'Arduino via le bus I2C. La fonction `requestEvent` est appelée lorsque l'Arduino reçoit une demande de données d'un autre périphérique. Vous pouvez effectuer des opérations spécifiques dans ces fonctions en fonction de vos besoins, par exemple lire les données reçues ou envoyer des données en réponse à une demande.

Exemple

Login | Tinkercad

https://www.tinkercad.com/things/bEchq0FXGPf-test-i2c



code Arduino maitre:

```c
#include <Wire.h>

// C++ code
//https://www.redohm.fr/2018/02/tutoriel-arduino-communication-i2c-1-maitre-2-esclaves/
//liens avec les fonctions i2c

#define i2C_slave_addr 1  //declaration de l'adresse

const int bouton = 12;

void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(bouton, INPUT_PULLUP);
  Serial.begin(9600);

  Wire.begin();  //initialisation du bus i2c en master
}

void loop() {

  bool etat_bouton = digitalRead(bouton);
  //Serial.print(etat_bouton);
  if (etat_bouton == 1)
    digitalWrite(LED_BUILTIN, HIGH);
  else {
    digitalWrite(LED_BUILTIN, LOW);
  }
  Wire.beginTransmission(i2C_slave_addr);
  Wire.write(etat_bouton);
  Wire.endTransmission();
  Wire.requestFrom(1, 4);
  while (Wire.available()) {
    char recep = Wire.read();
    Serial.print(recep);
  }
  Serial.println("");
  delay(1000);  // Wait for 1000 millisecond(s)
}
```

code arduino esclave:

C

```c
#include <Wire.h>

// C++ code
//
#define i2c_slave_addr 1
#define pin_led 12
//bool receivedI2CSignal = false;
char hexa[4] = {'H', 'e', 'x', 'b'};

void setup()
{
  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(pin_led, OUTPUT);
  Serial.begin(9600);
  Wire.begin(i2c_slave_addr);
  Wire.onReceive(i2c_event);
  Wire.onRequest(i2c_req);
}



void i2c_event(int howMany) {
  if (Wire.available() > 0) {
    // Enregistre le fait qu'au moins une valeur a été reçue
sur le bus I2C, ce qui désactive
    // totalement l'effet du bouton.
    //receivedI2CSignal = true;

    // Lit la donnée suivante, disponible sur le bus I2C.
    int i2c_valeur = Wire.read();

    if(i2c_valeur == 0){
      digitalWrite(pin_led, HIGH);
    }
    else
      digitalWrite(pin_led, LOW);

    // Affiche un message d'information.
    Serial.print("Valeur lue sur le bus I2C : '");
    Serial.print(i2c_valeur);
    Serial.println("'.");
  }
}
```

```
void i2c_req(){
   for(int i=0;i<4;i++) {
        Wire.write(hexa[i]);
  }
}

void loop()
{
  digitalWrite(LED_BUILTIN, HIGH);
  delay(1000); // Wait for 1000 millisecond(s)
  digitalWrite(LED_BUILTIN, LOW);
  delay(1000); // Wait for 1000 millisecond(s)
}
```

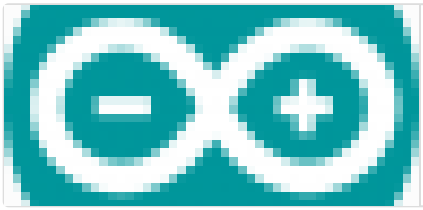
Sources

I2C — Wikipédia

https://fr.wikipedia.org/wiki/I2C

A Guide to Arduino & the I2C Protocol (Two Wire) | Ardui…

Allows the communication between devices or sensors connected via Two Wire Interface Bus.
https://docs.arduino.cc/learn/communication/wire?…



Wire - Arduino Reference

The Arduino programming language Reference, organized into Functions, Variable and Constant, and Structure keywords.
https://www.arduino.cc/reference/en/language/functions/communicat…