LES LIAISONS SERIES

ARDUINO UNO

Les bits de données sont transmis les uns après les autres

Synchrone

Fil d'horloge supplémentaire permettant de synchroniser le récepteur.

Asynchrone

Un signal de synchronisation est généré par l'émetteur au début d'une séquence de bits données.

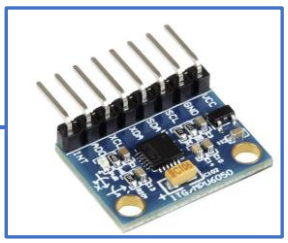
SIMPLEX circulation des données dans un seul sens

HALF-DUPLEX bidirectionnelle, 1 canal de transmission

FULL DUPLEX bidirectionnelle & simultanée

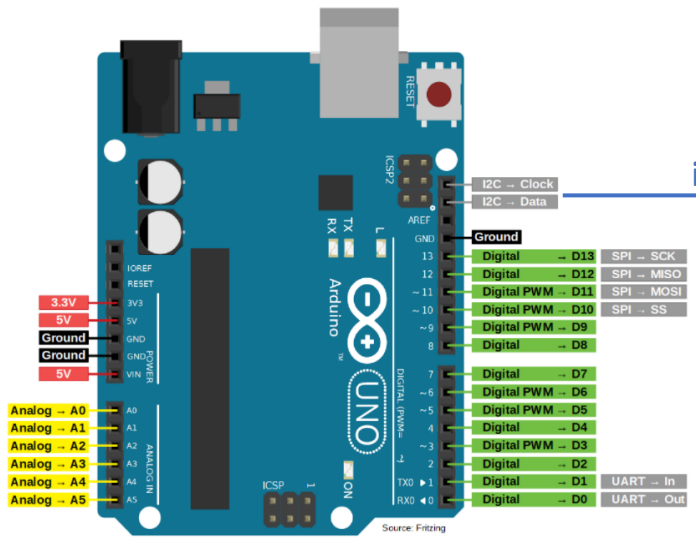
Vitesse de transmission

Exprimée en bit/s
1bit/s = 1baud



Centrale inertielle MPU6050

Arduino Uno Pinout

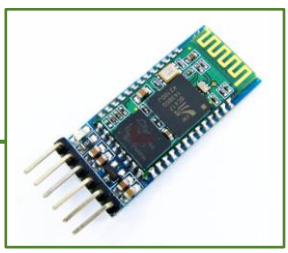


i2C

SPI

UART

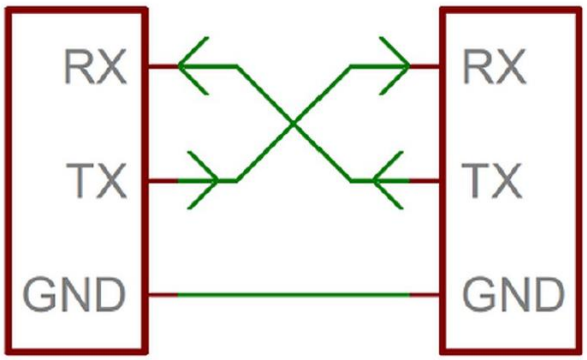
Joystick PMOD JSTK2



Module Bluetooth HC05

UART

Universal Asynchronous Receiver / Transmitter



Liaison point à point
(1 seul périphérique possible)

ASYNCHRONE

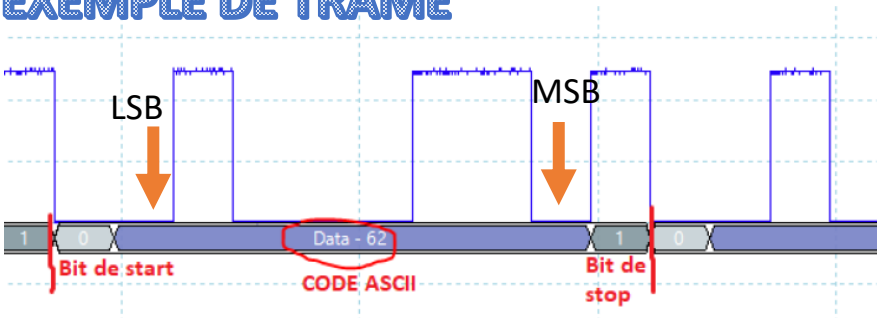
FULL DUPLEX

Code ASCII utilisé pour l'échange de données

Classiquement de 9600 à 115200 bit/s
(sous-multiple ou multiple de 9600)

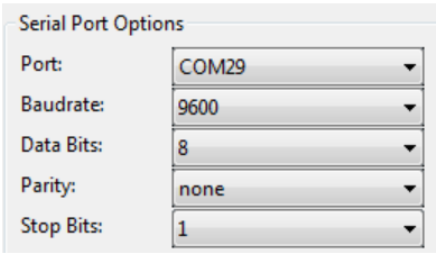
Le microcontrôleur et le périphérique doivent avoir la même configuration

EXEMPLE DE TRAME



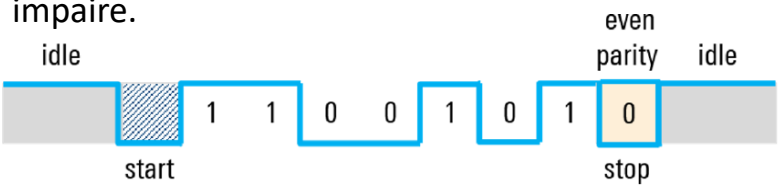
Mot envoyé : 0b01100010 = 0x62 = 'b'

CONFIGURATION



BIT DE PARITE

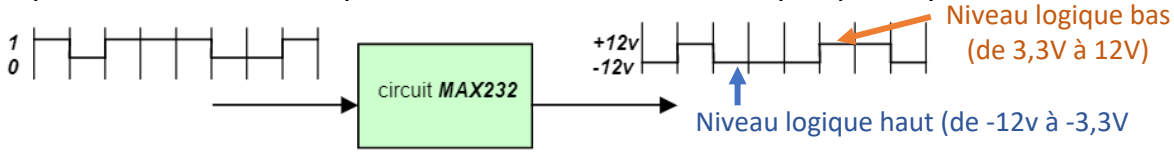
Le bit de parité qui est optionnel sert à détecter les erreurs. La valeur du bit de parité dépend du type de parité (impaire ou paire) utilisé.
Ce bit est réglé de manière à ce que le nombre total de 1 dans la trame soit paire ou impaire.



Ici ce code contient trois 0 et quatre 1. La parité utilisée est la parité paire, elle renvoie 0 car le nombre de 1 est paire.

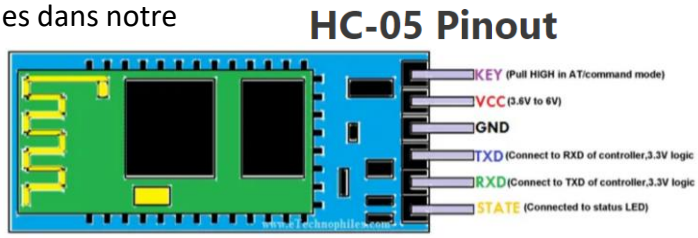
RS232

Norme « fréquemment » utilisée pour la transmission entre 2 périphériques

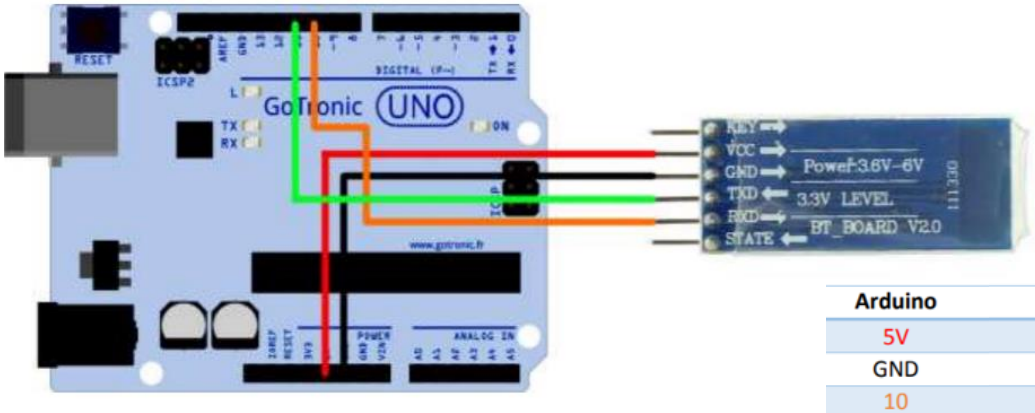


Exemple avec le module HC05

Ce module communique via une liaison UART avec une carte Arduino.
Cette liaison s'établit sur deux broches RX et TX définies dans notre programme en tant que broches 11 et 10.
La broche RX de la carte Arduino doit être raccordée à la broche TX du module Bluetooth HC-05.
La broche TX de la carte Arduino doit être raccordée à la broche RX du module HC-05.



Connection du module:



Arduino	Module HC-05
5V	Vcc
GND	GND
10	RXD
11	TXD

```
#include <SoftwareSerial.h>

#define rxPin 11 // Broche 11 en tant que RX, à raccorder sur TX du HC-05
#define txPin 10 // Broche 10 en tant que TX, à raccorder sur RX du HC-05

SoftwareSerial mySerial(rxPin, txPin);


void setup()
{
   // define pin modes for tx, rx pins:
   pinMode(rxPin, INPUT);
   pinMode(txPin, OUTPUT);
   mySerial.begin(9600);
   Serial.begin(9600);

}

void loop()
{
  int i = 0;
  char someChar[32] = {0};
  // when characters arrive over the serial port ...
  if(Serial.available()) {
    do{
      someChar[i++] = Serial.read();

     delay(3);

    }while (Serial.available() > 0);

    mySerial.println(someChar);
    Serial.println(someChar);
  }

  while(mySerial.available())
      Serial.print((char)mySerial.read());
}
```

Permet de créer une nouvelle liaison UART broches 10 et 11

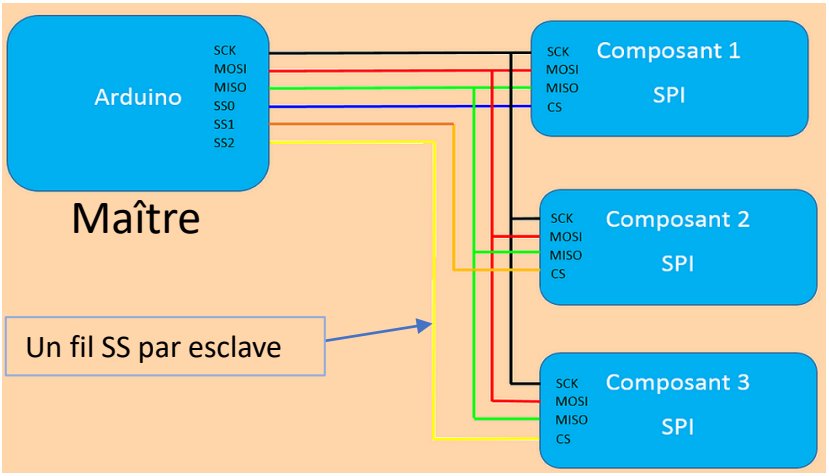
La vitesse de transmission est fournie, par défaut, 8bits, 1 bit de stop, pas de parité

La fonction ***available***, renvoie le nombre d'octets à lire.

La fonction ***read***, permet de récupérer un octet.

SPI

Serial Peripheral Interface



SYNCHRONE FULL DUPLEX

SCLK — **S**erial **Cl**oc**k** : Horloge.
(généré par le maître, quelques MHz, adaptée aux capacités de l'esclave)
MOSI — **M**aster **O**utput **S**lave **I**nput
(généré par le maître)
MISO — **M**aster **I**nput **S**lave **O**utput
(généré par l'esclave)
SS — **S**lave **S**elect, actif à l'état bas.
(généré par le maître)

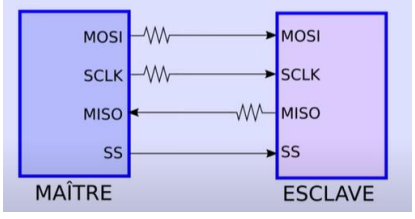
DIFFERENTS MODES

Mode SPI	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

Le **CPOL** détermine si au repos l'horloge est au niveau BAS (CPOL=0) ou HAUT (CPOL=1).

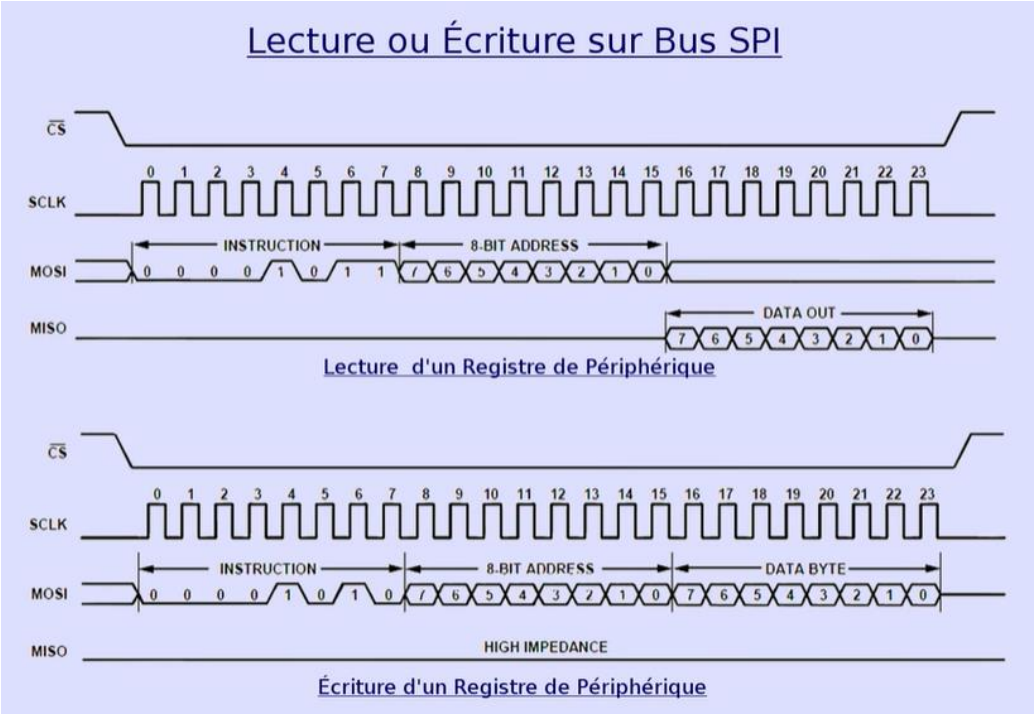
Le **CPHA** détermine à quel front de l'horloge les données sont transmises. CPHA=0 les données sont valides au premier front d'horloge, CPHA=1 elles sont valides au deuxième front.

BONNE PRATIQUE



Ajout de résistances de 33Ω

EXEMPLE DE TRAME



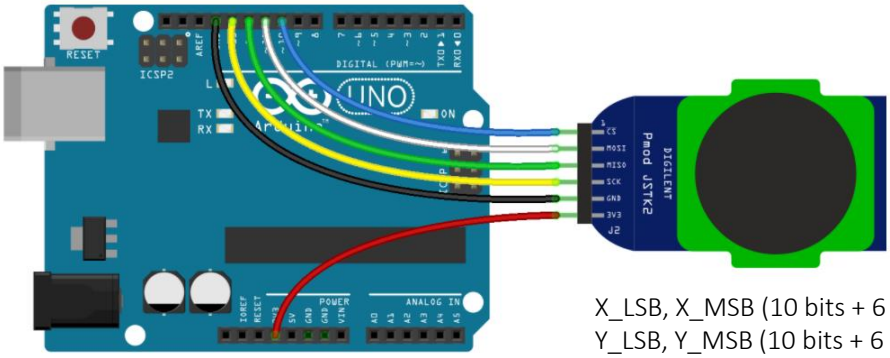
1- Le maître sélectionne l'esclave avec lequel il souhaite communiquer en mettant un niveau bas sur la ligne SS correspondante.

2- Le maître génère le signal d'horloge en fonction des capacités de l'esclave

3- A chaque coup d'horloge, le maître et l'esclave s'échangent un bit sur les lignes MOSI et MISO

Exemple : Joystick PMOD JSTK2

CONNECTION DU MODULE



- Manette résistive à deux axes
- Bouton central de la manette
- Bouton poussoir type gâchette
- LED RVB de 24 bits
- Connecteur Pmod 6 broches avec interface SPI (mode 0)

X_LSB, X_MSB (10 bits + 6 zéros non significatifs)
Y_LSB, Y_MSB (10 bits + 6 zéros non significatifs)

The 5 byte packet structure is provided in the image below:

	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
MOSI	COMMAND / 0	PARAM1 / DUMMY	PARAM2 / DUMMY	PARAM3 / DUMMY	PARAM4 / DUMMY
MISO	smpX (Low Byte)	smpX (High Byte)	smpY (Low Byte)	smpY (High Byte)	fsButtons

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
fsButtons	EXTPKT	0	0	0	0	0	TRIGGER	JOYSTICK

TRIGGER: Trigger Button Status Bit
1 = trigger button is currently pressed
0 = trigger button is not being pressed

JOYSTICK: Joystick Center Button Status Bit
1 = joystick center button is currently pressed
0 = joystick center button is not being pressed

		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
cmdSetLedRGB	(0x84)	1	0	0	0	0	1	0	0

Parameters
PARAM1 – Red LED duty cycle
PARAM2 – Green LED duty cycle
PARAM3 – Blue LED duty cycle
PARAM4 – ignored
Set the duty cycles for the Red, Green, and Blue LEDs.

```
#define CS 10 // CS pin

#include <SPI.h> // call library

void setup()
{
  SPI.begin(); // initialization of SPI port
  SPI.setDataMode(SPI_MODE0); // configuration of SPI communication in mode 0
  SPI.setClockDivider(SPI_CLOCK_DIV16); // configuration of clock at 1MHz
  pinMode(CS, OUTPUT);
}

void loop()
{
  float X, Y; //variables for X axis, Y axis and buttons
  int B;
  byte data[5];

  ////////////////// LECTURE DES DONNEES ////////////////

  digitalWrite(CS, LOW); // activation of CS line
  delayMicroseconds(15); // see doc: wait 15us after activation of CS line
  for (int i = 0; i < 5; i++) { // get 5 bytes of data
    data[i] = SPI.transfer(0);
    delayMicroseconds(10); // see doc: wait 10us after sending each data
  }
  digitalWrite(CS, HIGH); // deactivation of CS line
  delay(10);

  X = (data[1] << 8) | data[0]; //recunstruct 10-bit X value
  Y = (data[3] << 8) | data[2]; //recunstruct 10-bit Y value


  ////////ALLUMER LA LED RGB SI LE BOUTON EST APPUYE////////


  B = (data[4] & 1)
  if (B != 0) {
    //turn LED on
    digitalWrite(CS, LOW); // activation of CS line
    SPI.transfer(0x84);     //send next commands to LED
    delayMicroseconds(15);
    SPI.transfer(255 * (B & 1)); //R
    delayMicroseconds(15);
    SPI.transfer(255 * ((B & 2) >> 1)); //G
    delayMicroseconds(15);
    SPI.transfer(255 * ((B & 4) >> 2)); //B
    delayMicroseconds(15);
    digitalWrite(CS, HIGH); // deactivation of CS line
    delay(LED_time * 1000);
  }
  else
  {
    //turn LED OFF
```

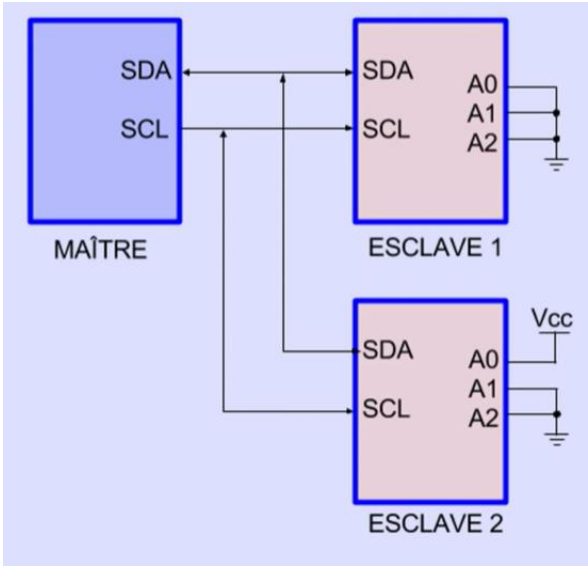
Charger la bibliothèque SPI.h

Configurer la liaison SPI

I2C
Inter Integrated Circuit

Le protocole est de type master slave, chaque circuit est identifié par son adresse et est soit transmetteur d'horloge (master) soit receveur(slave).

SYNCHRONE
HALF DUPLEX



Vitesse de transmission: 100kbit/s à 5 Mbit/s

SDA (Serial Data Line) : données bidirectionnelle,
SCL (Serial Clock Line) : horloge de synchronisation

Chaque élément a une adresse spécifique

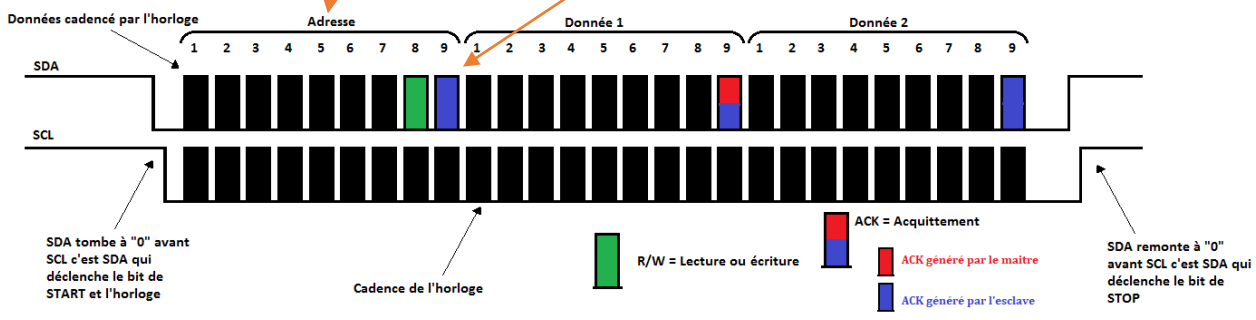
L'adresse est envoyée par le maître pour indiquer à quel composant est destiné le message

Les 2 lignes sont tirées au niveau de tension V_{CC} à travers des résistances de pull-up (4 à 5kΩ)

TRAME

Tous les esclaves décodent l'adresse pour savoir si le message leur est destinés.

L'esclave concerné par le message acquitte la bonne réception



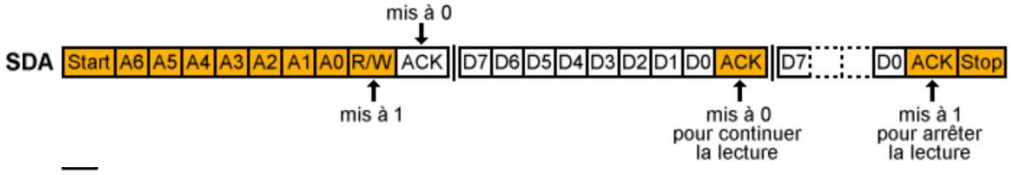
ECRITURE D'UNE DONNEE

R/W = 0
Le maître transmet un ou plusieurs octets de données. L'esclave acquitte à chaque octet.

état imposé par le maître

état imposé par l'esclave

LECTURE D'UNE DONNEE

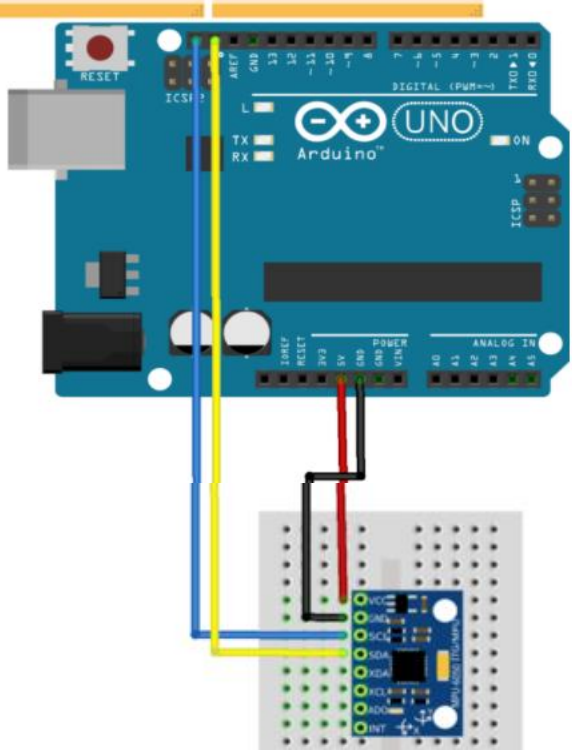


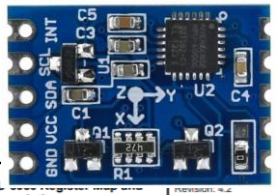
R/W = 1
L'esclave transmet un ou plusieurs octets de données. Le maître acquitte à chaque octet.

Exemple : Accéléromètre + Gyroscope MPU6050

SCL pin (you can also use A5)

SDA pin (you can also use A4)





InvenSense — MPU-6000/MPU Register Map and Descriptions — Release Date: 08/19/2013

The slave address of the MPU-60X0 is b110100X

Register (Hex)	Register (Decimal)	Bit7 ···	···Bit0
3B	59	ACCEL_XOUT[15:8]	
3C	60	ACCEL_XOUT[7:0]	
3D	61	ACCEL_YOUT[15:8]	
3E	62	ACCEL_YOUT[7:0]	
3F	63	ACCEL_ZOUT[15:8]	
40	64	ACCEL_ZOUT[7:0]	
41	65	TEMP_OUT[15:8]	
42	66	TEMP_OUT[7:0]	
43	67	GYRO_XOUT[15:8]	
44	68	GYRO_XOUT[7:0]	
45	69	GYRO_YOUT[15:8]	
46	70	GYRO_YOUT[7:0]	
47	71	GYRO_ZOUT[15:8]	
48	72	GYRO_ZOUT[7:0]	

données sur 16bits

```arduino
#include "Wire.h" // This library allows you to communicate with I2C devices.

const int MPU_ADDR = 0x68; // I2C address of the MPU-6050. If AD0 pin is set to HIGH, the I2C address will be 0x69.

int16_t accelerometer_x, accelerometer_y, accelerometer_z; // variables for accelerometer raw data
int16_t gyro_x, gyro_y, gyro_z; // variables for gyro raw data
int16_t temperature; // variables for temperature data

void setup() {
  Serial.begin(9600);
  Wire.begin();
  Wire.beginTransmission(MPU_ADDR); // Begins a transmission to the I2C slave (GY-521 board)
  Wire.write(0x6B); // PWR_MGMT_1 register
  Wire.write(0); // set to zero (wakes up the MPU-6050)
  Wire.endTransmission(true);
}
void loop() {
  Wire.beginTransmission(MPU_ADDR);
  Wire.write(0x3B); // starting with register 0x3B (ACCEL_XOUT_H) [MPU-6000 and MPU-6050 Register Map and Descriptions Revision 4.2, p.40]
  Wire.endTransmission(false); // the parameter indicates that the Arduino will send a restart. As a result, the connection is kept active.
  Wire.requestFrom(MPU_ADDR, 7*2, true); // request a total of 7*2=14 registers

  // "Wire.read()<<8 | Wire.read();" means two registers are read and stored in the same variable
  accelerometer_x = Wire.read()<<8 | Wire.read(); // reading registers: 0x3B (ACCEL_XOUT_H) and 0x3C (ACCEL_XOUT_L)
  accelerometer_y = Wire.read()<<8 | Wire.read(); // reading registers: 0x3D (ACCEL_YOUT_H) and 0x3E (ACCEL_YOUT_L)
  accelerometer_z = Wire.read()<<8 | Wire.read(); // reading registers: 0x3F (ACCEL_ZOUT_H) and 0x40 (ACCEL_ZOUT_L)
  temperature = Wire.read()<<8 | Wire.read(); // reading registers: 0x41 (TEMP_OUT_H) and 0x42 (TEMP_OUT_L)
  gyro_x = Wire.read()<<8 | Wire.read(); // reading registers: 0x43 (GYRO_XOUT_H) and 0x44 (GYRO_XOUT_L)
  gyro_y = Wire.read()<<8 | Wire.read(); // reading registers: 0x45 (GYRO_YOUT_H) and 0x46 (GYRO_YOUT_L)
  gyro_z = Wire.read()<<8 | Wire.read(); // reading registers: 0x47 (GYRO_ZOUT_H) and 0x48 (GYRO_ZOUT_L)
    .
  .
  .
  .
  //ECRITURE d'une donnée de config
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x1D);// starting with register 0x1D (ACCEL_CONFIG)
  Wire.write(0xF0)
  Wire.endTransmission(true);
```

Active la liaison i2C

Nous positionne sur le registre souhaité (RW=1 => lecture)

Ecriture de la valeur 0xF0 dans le registre 0x1C (R/W = 0 =>écriture)

…G_SEL	Full Scale Range
0	± 2g
1	± 4g
2	± 8g
3	± 16g

5 Register 28 – Accelerometer Configuration

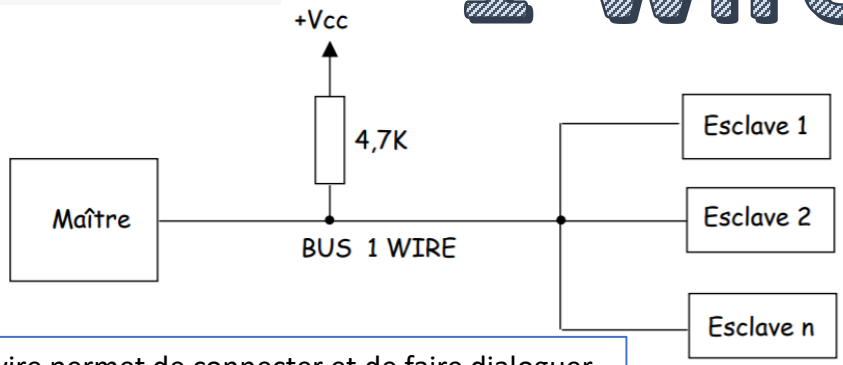
CCEL_CONFIG

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1C	28	XA_ST	YA_ST	ZA_ST	AFS_SEL[1:0]			-	

1-wire

DALLAS SEMICONDUCTOR — maxim integrated

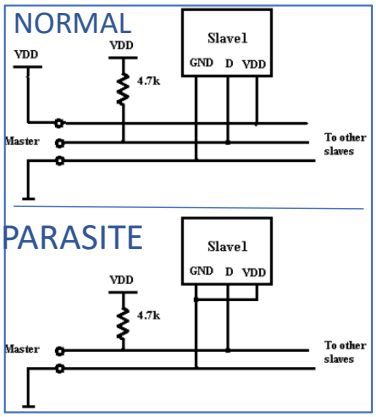


ASYNCHRONE HALF DUPLEX

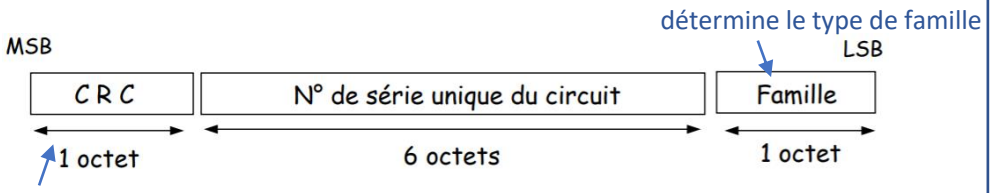
Le fil unique du bus doit être tiré au +Vcc par une résistance de 4,7KΩ. L'état repos du bus est donc un état haut.

1-wire permet de connecter et de faire dialoguer entre eux des circuits sur un seul fil (+GND). Toutes les commandes et données sont envoyées avec le bit LSB en tête.

2 modes d'alimentation



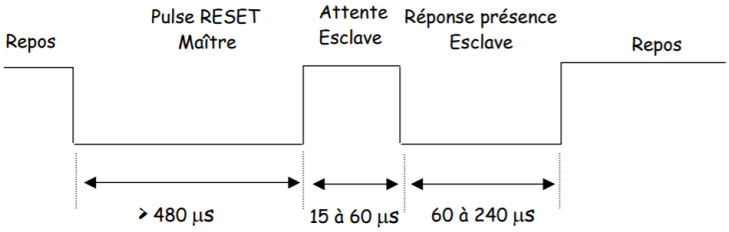
Chaque circuit a une adresse (64 bits) physique unique.



CRC : octet de contrôle calculé à partir des 56 bits précédents.

PROTOCOLE DE COMMUNICATION

Toute transaction entre un maître et un ou plusieurs esclaves, débute par une initialisation (RESET) :



Le maître impose l'état bas plus de 480 µs
Tous les composants sur le bus sont remis à zéro.

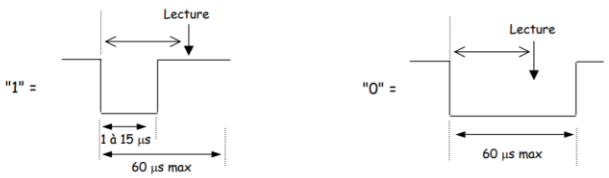
Après un délai de 15 à 60 µs, le ou les esclaves raccordés, forcent le bus à l'état bas pendant 60 à 240 µs pour signaler leur présence.

Le maître doit ensuite envoyer une commande de type ROM qui est propre au protocole 1 Wire, et que tous les circuits de ce type vont reconnaître. Cela va permettre entre autre de sélectionner un circuit parmi les différents esclaves qui ont répondu présents au pulse de Reset. Le dialogue et l'échange de données pourra ensuite commencer, entre le maître et l'esclave sélectionné.

ÉMISSION D'UN BIT

Du maître vers l'esclave

Le maître force le bus à "0" pendant 1 à 15 µs. L'esclave va lire le bus entre 15 et 45 µs après le front descendant (valeur typique 30 µs)



De l'esclave vers le maître

Le maître force le bus à "0" pendant au moins 1 µs. Si l'esclave veut émettre un "1", il laisse le bus libre donc tiré à "1". Pour émettre un "0", l'esclave doit tirer le bus à "0" pendant 15 µs au minimum.