Les systemes de communication

Un systeme de communication:

Un système de communication en Arduino est une configuration matérielle et logicielle qui permet à une carte Arduino d'échanger des données avec d'autres dispositifs, comme des capteurs, des actionneurs, d'autres microcontrôleurs ou même des ordinateurs. Ces systèmes utilisent divers protocoles et technologies pour transmettre des informations, en fonction des besoins du projet.

Les protocoles célèbres utilisés sont :

- **UART (Universal Asynchronous Receiver/Transmitter)** : Communication série asynchrone, où les données sont envoyées bit par bit sur un fil de transmission (TX) et reçues sur un fil deréception (RX).
- **I²C (Inter-Integrated Circuit)** : Un protocole de communication synchrone utilisant deux fils : SDA (données) et SCL (horloge).
- **SPI (Serial Peripheral Interface)** : Un protocole rapide et synchrone qui utilise plusieurs fils (MOSI, MISO, SCK, et parfois CS/SS).

Transformation des données avant transmission

Pour que les données soient transmises, elles doivent être converties dans un format compatible avec les systèmes numériques. Voici les étapes :

1- Conversion en ASCII :
Les informations, comme des caractères, sont d'abord transformées en leur équivalent ASCII (code numérique).

- Exemple : La lettre "A" est convertie en **65**.

2- Conversion en binaire :
L'équivalent ASCII est ensuite représenté en binaire.

- Exemple : **65** devient **01000001** (8 bits).

3- Encodage en signaux électriques :
Les données binaires sont transformées en un signal électrique (par exemple, une tension haute pour 1 et basse pour 0) qui peut être transmis à travers un fil.

4- Transmission via des fils ou un bus :

Fil de données (data) : Transporte les données codées.
Fil d'horloge (clock) : Synchronise l'envoi et la réception des bits dans les protocoles synchrones comme I²C ou SPI.

Protocole synchrone :

Un protocole synchrone est une méthode de communication dans laquelle les échanges de données sont étroitement coordonnés à l'aide d'un signal d'horloge partagé. Ce signal d'horloge détermine le moment où les données sont envoyées ou reçues, garantissant que les deux parties (émetteur et récepteur) travaillent à la même vitesse.
Caractéristiques principales :

- **Nécessite un signal d'horloge commun ou un mécanisme pour synchroniser l'émetteur et le récepteur.**
  - **Les données sont transmises de manière continue et ordonnée.**
  - **Convient aux communications en temps réel ou à débit constant.**

Exemples : communications sur une ligne téléphonique classique, SPI (Serial Peripheral Interface).

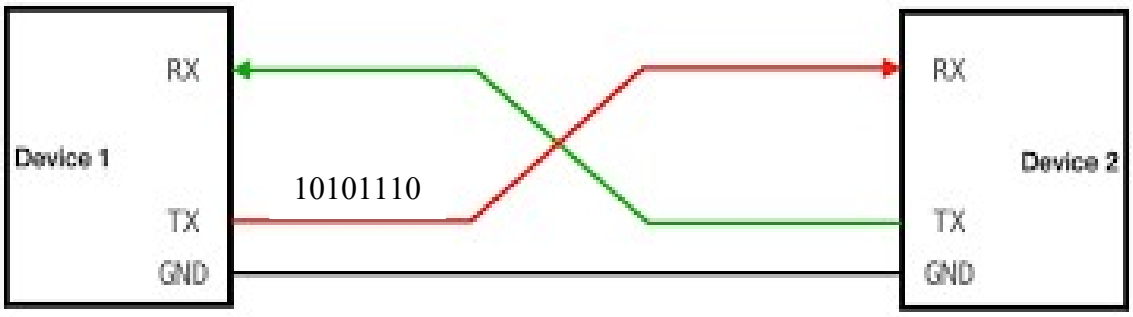
Protocole asynchrone :

Un protocole asynchrone, en revanche, n'utilise pas de signal d'horloge partagé. Les données sont transmises avec des bits de contrôle (par exemple, bits de start et de stop) pour indiquer le début et la fin de chaque paquet. Cela permet à l'émetteur et au récepteur de fonctionner de manière indépendante.
Caractéristiques principales :

- **Pas besoin d'un signal d'horloge commun.**
  - **Les données peuvent être envoyées à des intervalles irréguliers.**
  - **Plus adapté aux communications où la transmission est sporadique ou intermittente.**
  - **Exemples : UART (Universal Asynchronous Receiver-Transmitter), communication série RS-232.**

Après avoir compris comment les données sont converties de leur format habituel en signal électrique, approfondissons les protocoles de communication UART, I2C et SPI.

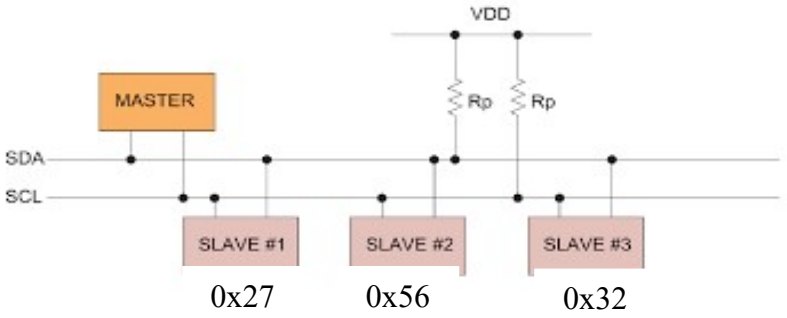
➢ PROTOCOLE UART (UNIVERSAL ASYNCHRONOUS RECEIVER-TRANSMITTER)



UART est une méthode de communication série **asynchrone**, utilisée pour l'échange de données entre microcontrôleurs et périphériques. Son fonctionnement repose sur un émetteur (TX) et un récepteur (RX), sans horloge partagée.

- **Transmission des données** :
  - **Bit de départ (Start bit)** : Indique le début de la transmission.
  - **Données** : Généralement 7 ou 8 bits.
  - **Bits d'arrêt (Stop bits)** : Indiquent la fin du message.
- **Avantages** :
  - Simple à mettre en œuvre.
  - Nécessite peu de connexions (seulement 2 fils).
- **Inconvénients** :
  - Débit relativement limité.
  - Ne convient pas aux communications nécessitant un haut débit ou la gestion de multiples périphériques.

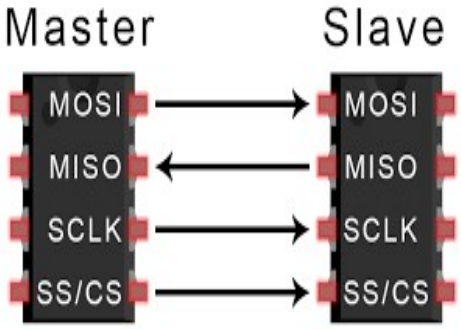
➢PROTOCOLE I2C (INTER-INTEGRATED CIRCUIT)

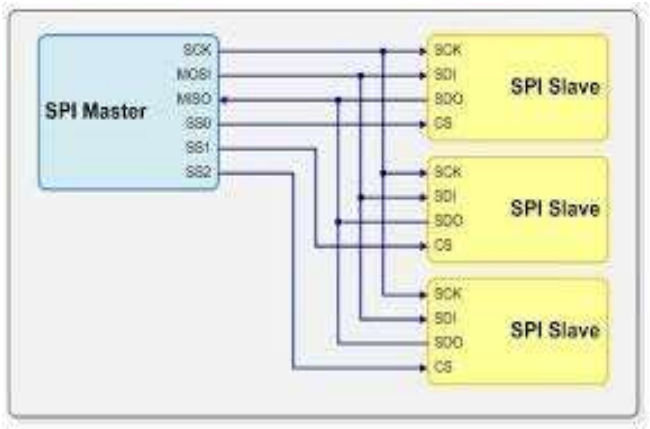


I2C est un protocole de communication **synchrone**, conçu pour connecter plusieurs périphériques sur un même bus via seulement deux fils :

- **SDA (Serial Data)** : Ligne pour l'échange de données.
- **SCL (Serial Clock)** : Ligne d'horloge qui synchronise la transmission.
- **Caractéristiques principales** :
  - **Adressage unique** : Chaque périphérique a une adresse spécifique permettant au maître de le sélectionner.
  - **Mode maître-esclave** : Un maître contrôle la communication avec un ou plusieurs esclaves.
  - **Possibilité de multimaster** : Plusieurs maîtres peuvent coexister sur le même bus.
- **Avantages** :
  - Utilisation efficace des connexions (seulement 2 fils).
  - Prise en charge de plusieurs périphériques sans nécessiter de nombreuses lignes de communication.
- **Inconvénients** :
  - Vitesse de transmission inférieure à celle du SPI.
  - Plus complexe à implémenter qu'UART

➢ PROTOCOLE SPI (SERIAL PERIPHERAL INTERFACE)





Principe : SPI est un protocole de communication série synchrone utilisé pour la communication entre un maître et plusieurs esclaves. Contrairement à I2C, qui utilise un bus partagé pour les données, SPI utilise plusieurs fils pour des fonctions spécifiques, ce qui permet des débits plus élevés.

Caractéristiques :

- **Synchronisé** : Utilise un signal d'horloge (SCK) pour synchroniser la transmission des données.
- **Full-Duplex** : Les données peuvent être envoyées et reçues simultanément.
- **Lignes de communication** :
  - **MOSI (Master Out Slave In)** : Ligne de données allant du maître vers l'esclave.
  - **MISO (Master In Slave Out)** : Ligne de données allant de l'esclave vers le maître.
  - **SCK (Serial Clock)** : Signal d'horloge généré par le maître.
  - **SS (Slave Select)** : Utilisé pour sélectionner quel esclave communiquer.

Avantages :

- Très rapide comparé à UART et I2C.
- Simple à implémenter avec une transmission en full-duplex.

Inconvénients :

- Nécessite plus de lignes de connexion (au moins 4 fils, plus un pour chaque esclave supplémentaire).
- Moins efficace que l'I2C lorsqu'il s'agit de connecter plusieurs périphériques

Critere	UART (Universal Asynchronous Receiver-Transmitter)	I2C (Inter-Integrated Circuit)	SPI (Serial Peripheral Interface)
Type de communication	Série asynchrone	Série synchrone	Série synchrone
Nombre de fils requis	2 (TX, RX)	2 (SDA, SCL)	4 minimum (MOSI, MISO, SCLK, SS)
Vitesse de transmission	Jusqu'à 1 Mbps (variable selon l'implémentation)	Standard : 100 kHz, Fast : 400 kHz, Ultra-fast : 3,4 MHz	Très rapide (jusqu'à plusieurs dizaines de MHz)
Mode de communication	Full-duplex (mais unidirectionnel par ligne)	Half-duplex	Full-duplex
Nombre de maîtres	Un seul maître et un seul esclave (1:1)	Un ou plusieurs maîtres (1:N ou N:N)	Un seul maître avec plusieurs esclaves (1:N)
Adressage	Pas d'adressage, chaque périphérique doit être connecté sur un port dédié	Adresse unique pour chaque périphérique (7 ou 10 bits)	Chaque esclave est sélectionné via une ligne SS dédiée
Complexité matérielle	Faible, simple à implémenter	Moyenne (résistances pull-up nécessaires)	Élevée (plus de connexions physiques)
Gestion des erreurs	Bit de parité optionnel	Confirmation par ACK/NACK	Pas de mécanisme de détection d'erreur intégré
Consommation d'énergie	Moyenne	Faible (idéale pour l'embarqué)	Élevée (à cause des signaux rapides et multiples fils)
Utilisation courante	Communication série simple (ex: USB, Bluetooth, GPS)	Capteurs, EEPROM, RTC, gestion de petits périphériques	Affichages, cartes SD, mémoires Flash, ADC rapides
Avantages	Facile à utiliser, peu de fils nécessaires	Connexion multi-périphérique sur un seul bus	Très rapide, efficace pour la transmission de gros volumes de données
Inconvénients	Vitesse limitée, nécessite une configuration identique des baud	Plus lent que SPI, nécessite des résistances de pull-up	Nécessite plus de fils, plus complexe pour gérer plusieurs esclaves

Salut, Je m'appelle arduino , voulez-vous de pratique

Ok !!

ARDUINO UNO + HC-05 Bluetooth Module (UART)

Contrôlons une LED par Bluetooth :

Le but de cette expérience est de contrôler une led par arduino à distance via Bluetooth à travers notre téléphone

Matériels requis :

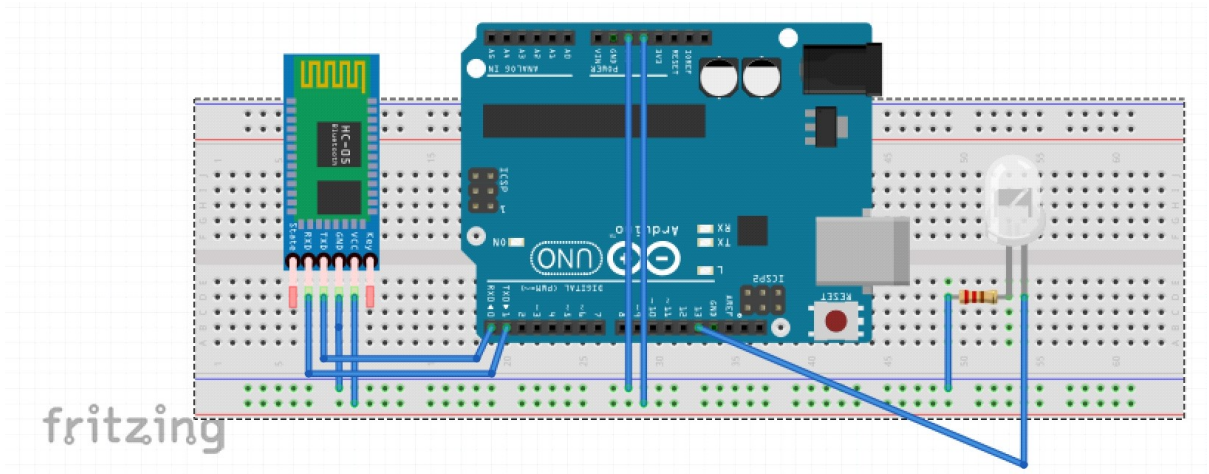
Arduino (uno – nano – Leonardo – méga …)

Bluetooth module (HC-05)

Led et résistance

Cette plateforme (Remote XY) nous aide à concevoir l'interface utilisateur de commande.

Le schéma du projet :



Pour le code et l'interface nous visitons le site remote xy : https://remotexy.com

Le code pour cette interface :

```
/*
   -- New project --

   This source code of graphical user interface
   has been generated automatically by RemoteXY editor.
   To compile this code using RemoteXY library 3.1.13 or later version
   download by link http://remotexy.com/en/library/
   To connect using RemoteXY mobile app by link http://remotexy.com/en/download/
     - for ANDROID 4.15.01 or later version;
     - for iOS 1.12.1 or later version;

   This source code is free software; you can redistribute it and/or
   modify it under the terms of the GNU Lesser General Public
   License as published by the Free Software Foundation; either
   version 2.1 of the License, or (at your option) any later version.
*/

//////////////////////////////////////////////
//        RemoteXY include library          //
//////////////////////////////////////////////

// you can enable debug logging to Serial at 115200
//#define REMOTEXY__DEBUGLOG

// RemoteXY select connection mode and include library
#define REMOTEXY_MODE__SOFTSERIAL

#include <SoftwareSerial.h>

// RemoteXY connection settings
#define REMOTEXY_SERIAL_RX 2
#define REMOTEXY_SERIAL_TX 3
#define REMOTEXY_SERIAL_SPEED 9600


#include <RemoteXY.h>

// RemoteXY GUI configuration
#pragma pack(push, 1)
uint8_t RemoteXY_CONF[] =   // 43 bytes
  { 255,1,0,0,0,36,0,19,0,0,0,0,31,2,106,200,200,84,1,1,
  1,0,2,36,50,23,52,78,25,44,22,0,2,26,31,31,79,78,0,79,
  70,70,0 };

// this structure defines all the variables and events of your control interface
struct {

    // input variables
  uint8_t switch_01; // =1 if switch ON and =0 if OFF

    // other variable
  uint8_t connect_flag;  // =1 if wire connected, else =0

} RemoteXY;
#pragma pack(pop)

//////////////////////////////////////////////
//           END RemoteXY include           //
//////////////////////////////////////////////

#define PIN_SWITCH_01 13
```

```
void setup()
{
  RemoteXY_Init ();

  pinMode (PIN_SWITCH_01, OUTPUT);

  // TODO you setup code

}

void loop()
{
  RemoteXY_Handler ();

  digitalWrite(PIN_SWITCH_01, (RemoteXY.switch_01==0)?LOW:HIGH);

  // TODO you loop code
  // use the RemoteXY structure for data transfer
  // do not call delay(), use instead RemoteXY_delay()


}
```