

Module Systèmes Temps-Réels TP FreeRTOS

Capteur CO² - Température - Humidité



Protocoles de communication inter-puces

Source Matthias Puech CNAM

Comment communiquent des puces sur une même carte ?

Exemple

MCU ↔ ADC/DACs externes, LCD, EEPROM, Radio...

Les standards

Communication par messages selon le protocole établi:

- UART
- I²C
- SPI

Protocole UART

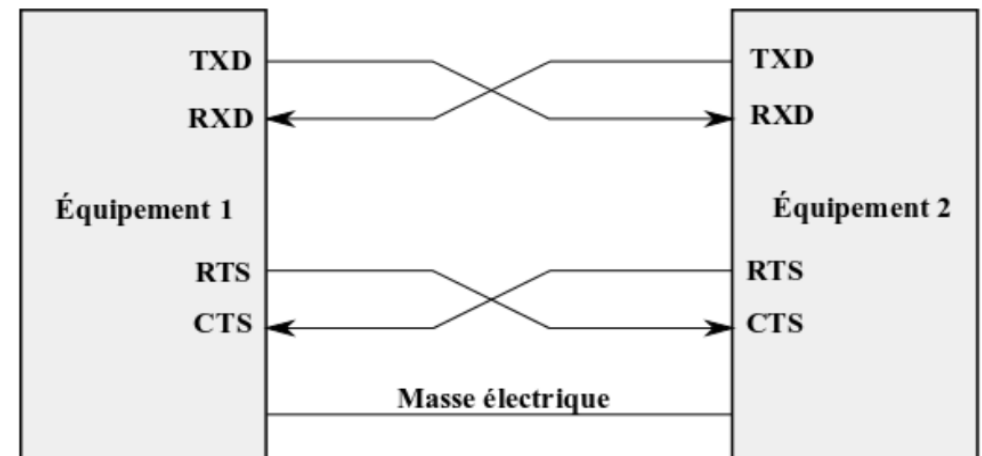
(Universal Asynchronous Receiver Transmitter)

“Composant” développé par Gordon BELL en 1959:

- 8 bits par trame.
- Vitesse de transmission (9600 ou 115200 bps).
- Asynchrone.
- Half-duplex (1 fil de données TX -> RX).
- Bit de parité (paire ou impaire ou rien).
- Bit de STOP (0.5, 1, 1.5 ou 2)
- 2 normes: RS 232 / RS 485



D : bit de début, P : bit de parité, F : bit de fin



Protocole I²C

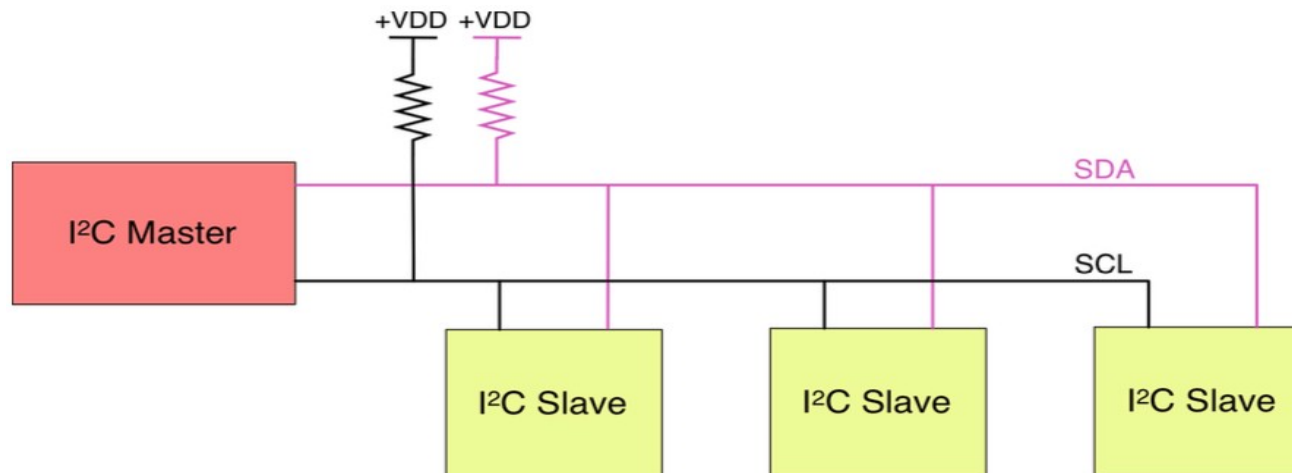
(Inter Integrated Circuit)

T4



“Standard” développé par Philips en 1982:

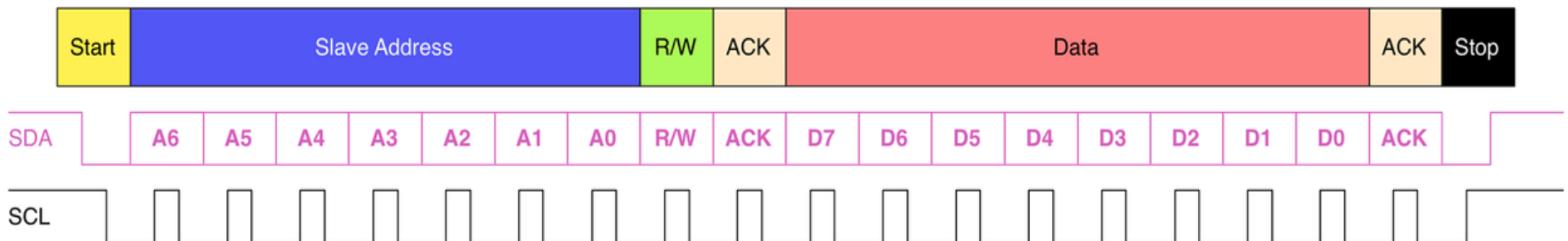
- Deux fils :
 - SDA (*Serial Data Line* : les données);
 - SCL (*Serial Clock Line*).
- Etat haut par défaut (+VDD).
- Chaque partie (maître/esclave) peut abaisser SDA ou SCL (communication bidirectionnelle).
- Chaque esclave a une adresse unique.
- Vitesse: 100kHz (standard) – 400 kHz (fast mode).



Messages

Tout message est initié et terminé et par le maître

- commence par START: front *descendant* sur SDA quand SCL est *haut*,
- finit par STOP: front *montant* sur SDA quand SCL est *haut*,
- constitué de plusieurs *frames* : (suite de 8 bits)
 - 1 *frame* d'adresse (7 bits + direction R/W),
 - 1 ou plusieurs *frames* de données (8 bits),
 - 1 ACK - > récepteur met SDA à 0.



Protocole SPI

(Serial Peripheral Interface)

T6



“Standard” développé par Motorola en 197x

- Nature des données = paquets d’octets en série.
- Topologie “bus” (+ 1 fil SS par esclave).
- Synchrone (fil clock).
- Full-duplex (2 fils de données).
- Symétrie un client, plusieurs serveurs.

Spécificités

- Beaucoup plus rapide que I²C(1-100MHz).
- Pas de spécification sur la structure des messages (pas de *frame*, pas d’en-tête).

Protocole SPI

(Serial Peripheral Interface)

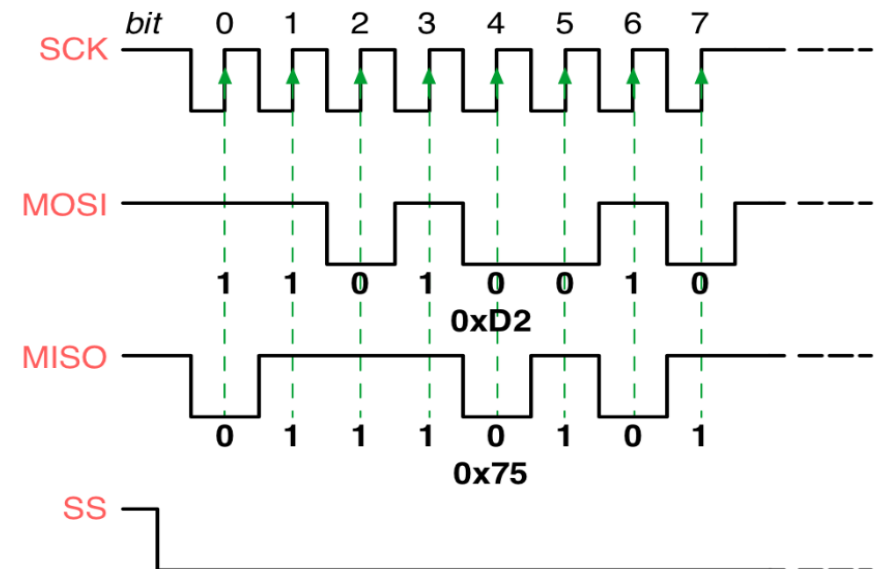
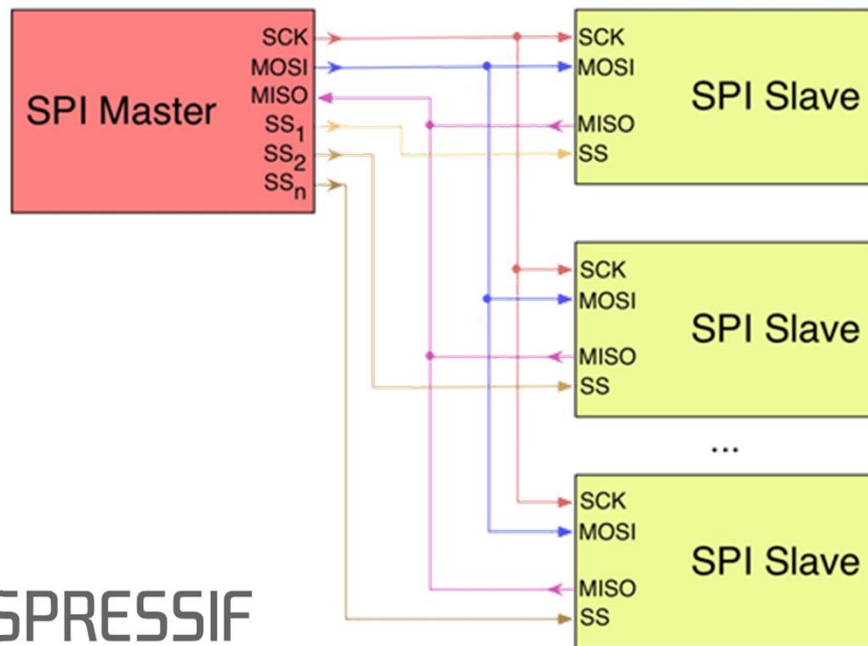
Fonctionnement:

- le maître met SS_i à 0 \rightarrow START
- les données transitent alors simultanément par paquets d'octets

MOSI maître vers esclave i

MISO esclave i vers maître (*most significant bit first* ou MSB)

- le signal est échantillonné à chaque front montant de SCK
- le maître met SS_i à 1 \rightarrow STOP



I²C

Vitesse de transmission

SPI

Même s'il existe des variations de l'I²C qui montent au dessus de 1MHz, la grande majorité des implémentations que l'on trouve utilisent généralement 100 ou 400 kHz

Pour le SPI il est possible de trouver certains composants au delà de 20 Mbits

I²C

Topologie

SPI

C'est un véritable protocole qui permet l'interconnexion de multiples boîtiers dans différentes configurations :
Maitre / Esclave, Maitre / Multiple esclaves, Multiple Maitres / Multiples esclaves

En général point à point, bien que l'on puisse connecter plusieurs esclaves mais il faut alors des lignes supplémentaires. Un seul maitre qui génère l'horloge.

I²C

Consommation

SPI

du à la configuration collecteur/ drain ouvert sur les 2 lignes de transmission (SDA + SCL), consommation relativement élevée

signaux de type TTL/CMOS donc consommation faible

Avantages/Inconvénients

- ✓ Si on doit interconnecter plusieurs boîtiers et que la vitesse n'est pas un problème, préférer l'I²C car c'est un protocole.
- ✓ Si on veut de la vitesse le SPI est loin devant...
- ✓ Implémentation logicielle sur des E/S : Il est BEAUCOUP plus facile (et cela prend moins de ressources) de faire du SPI par logiciel sur des broches d'E/S que de l'I²C dû à la machine d'état.
- ✓ Mise en œuvre : l'I²C est plus compliqué.
- ✓ L'interconnexion de plusieurs boîtiers est également plus délicate avec l'I²C car il faut prendre en compte les impédances de chacun des boîtiers pour calculer les résistances de rappels.

Affichage OLED:

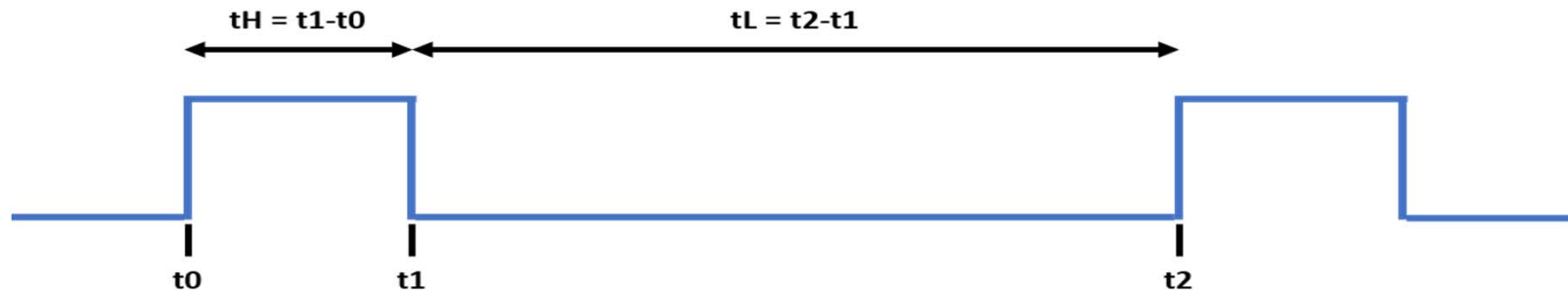
- Communication ESP32 <-> Afficheur OLED via bus I²C
 - Adresse = 0x3C / SDA = 5 / SCL = 4
 - Bibliothèque ESP32 OLED driver for SSD1306 (Framework Arduino)
 - Programme SSD1306SimpleDemo
 - Include <SSD1306Wire.h>
 - Initialisation: *Display.init();*
 - Effaçage: *Display.clear();*
 - Affichage: *Displays.display();*
- **Réaliser une tâche qui affiche un compteur en s au centre de l'écran (RAZ modulo 60)**

Capteur Température – Humidité : DHT 22

- Communication ESP32 <-> DHT 22 via 1-Wire bus:
 - **Input pin = 15**
 - Bibliothèque DHT sensor library for ESPx (Framework Arduino)
 - Programme DHT_ESP32
 - Initialisation: `dht.setup(dhtPin, DHTesp::DHT12);`
- **Réaliser une tâche qui affiche la température et le taux d'humidité toutes les 5 secondes (Utilisation des « Tickers » non obligatoire)**

Capteur CO²: MH-Z19B

- Communication ESP32 <-> MH-Z19B via UART / PWM / Analog:
 - Serial1 pour init + lecture: **TXPin = 17 / RXPIn = 16**
 - PWM (Pulse Width Modulation) pour lecture: **PWMPin = 23**



CO2 concentration: $C_{ppm} = \text{Range} \times (t_H - 2 \text{ ms}) / (t_H + t_L - 4 \text{ ms})$

- **Réaliser une tâche qui affiche la concentration en CO² toutes les 4 secondes.**
- **Comparer les mesures obtenues par liaison série et PWM.**

- Communication ESP32 <-> MH-Z19B via UART: `#include "driver/uart.h"`
- Init: TXPin = 17 / RXPIN = 16

```
const uart_config_t uart_config = {
    .baud_rate = 9600,
    .data_bits = UART_DATA_8_BITS,
    .parity = UART_PARITY_DISABLE,
    .stop_bits = UART_STOP_BITS_1,
    .flow_ctrl = UART_HW_FLOWCTRL_DISABLE,
    .source_clk = UART_SCLK_APB,
};
// We won't use a buffer for sending data.
uart_driver_install(UART_NUM_1, RX_BUF_SIZE * 2, 0, 0, NULL, 0);
uart_param_config(UART_NUM_1, &uart_config);
uart_set_pin(UART_NUM_1, TXD_PIN, RXD_PIN, UART_PIN_NO_CHANGE, UART_PIN_NO_CHANGE);
```

- Lecture:

```
uint8_t* data = (uint8_t*) malloc(RX_BUF_SIZE+1);
rxBytes = uart_read_bytes(UART_NUM_1, data, RX_BUF_SIZE, 1000 / portTICK_RATE_MS);
```

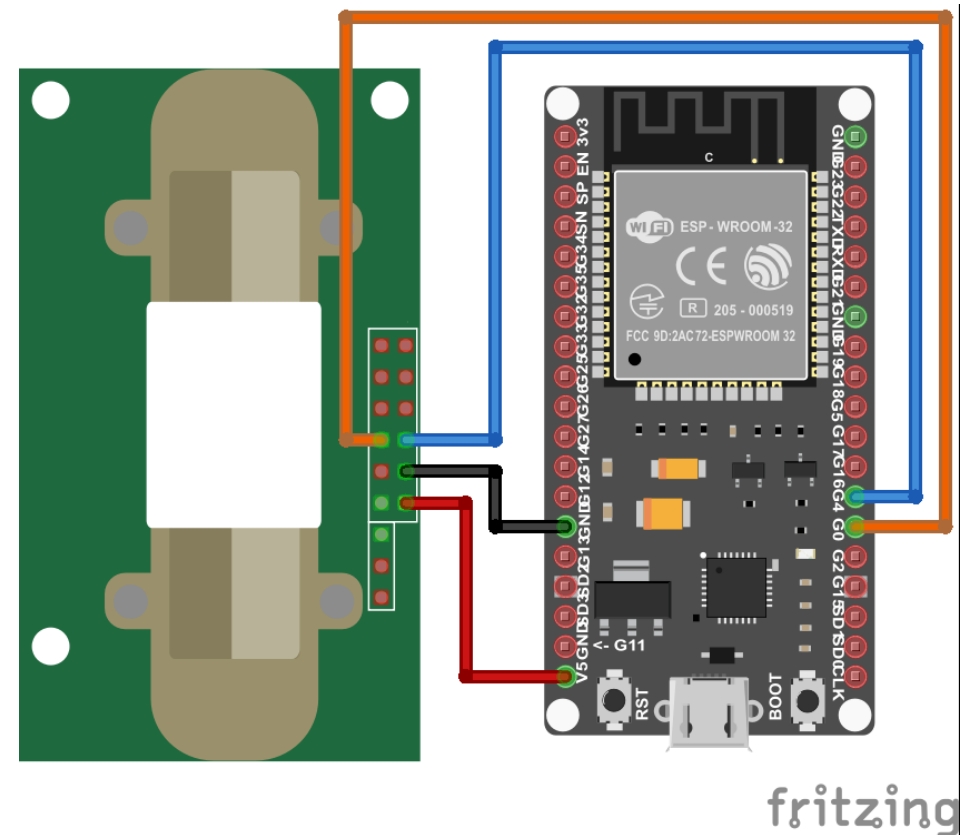
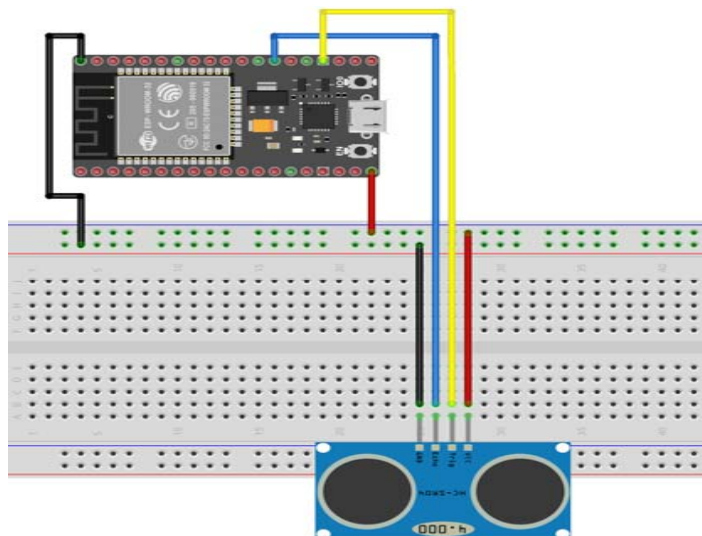
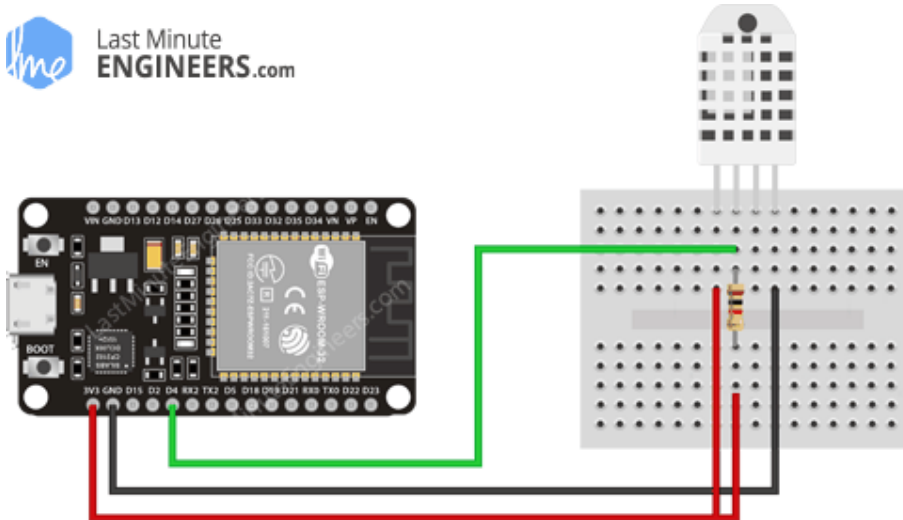
- Ecriture:

```
txBytes = uart_write_bytes(UART_NUM_1, data, len);
```

Capteur Ultrason: HC-SR04

- Communication ESP32 <-> MH-Z19B via UART / PWM / Analog:
 - Trigger: **Pin = 33**
 - Echo: **Pin = 32**
- **Réaliser une tâche qui se déclenche sur un front montant d'interruption et affiche la distance de détection.**
- **Régler pour que détection se fasse à 50cm.**

Montages:



Projet:

Réalisation d'un capteur / afficheur de CO² - température –
hygrométrie, sur détection de personnes

- Tester les capteurs individuellement.
- Utiliser l'ESP-32 OLED pour la température (Framework Arduino).
- Utiliser l'ESP32 pour le CO² (Framework IdF)
- Synchroniser les 2 ESP (communication par RS232 : Pins 25 - 26).
- Option: déclencher par détection de présence (HC – SR04).
- Mettre en œuvre le modèle Producteurs Multiples (CO²/ Temp/Hum) –
Consommateur unique (Affichage)
- Utiliser « à bon escient » les outils d'exclusion mutuelle et de synchronisation.
- Rédiger un CR détaillant les tâches et leur liens (avec modèle SART si possible)
- Déposer le projet sur Arche

Projet:

Réalisation d'un capteur / afficheur de CO² - température –
hygrométrie, sur détection de personnes

