Laboratoire 1 – La communication sérielle

Questionnement (objectifs) :

- Comment transmettre de l'information entre deux systèmes à microcontrôleur?
- Quels sont les standards de communication les plus courants?
- Qu'est-ce qu'un protocole de communication?
- Comment interconnecter deux systèmes à microcontrôleur pour établir un échange d'information?

Matériel requis :

- Plaque protoTPhys
- Module EIA-485
- Module translateur de niveau logique (« level-shifter »)
- Fil de liaison (câble téléphonique 4 brins)
- Module SHT20 modbus
- Outils de prototypage (pinces, coupe-fils, dégaineur, etc)
- Fils à breadboard et 4 fils à prises « dupont » mâle-femelle de 10cm

Mise en situation :

La communication sérielle est souvent celle de base disponible pour échanger des informations entre deux systèmes (ex : microcontrôleur et périphérique). Elle est à la base de beaucoup de systèmes de transmission : le disque dur de votre ordinateur, la souris via le USB, le réseau (Wifi ou Ethernet), entre les réseaux (internet), etc.

La situation du présent laboratoire est que vous allez étudier un système dans lequel deux microcontrôleurs, chacun relié à des périphériques distincts, doivent s'échanger de l'information et contrôler des états de leurs broches.

Ce travail de laboratoire devra être réalisé en équipe de 2.

Théorie associée :

- https://docs.arduino.cc/learn/communication/uart/
- https://microcontrollerslab.com/esp32-uart-communication-pins-example/
- https://microcontrollerslab.com/rs485-serial-communication-esp32-esp8266-tutorial/
- https://www.analog.com/en/resources/technical-articles/rs485-cable-specification-guide--maxim-integrated.html
- https://www.rohde-schwarz.com/fr/produits/test-et-mesure/essentials-test-equipment/digital-oscilloscopes/comprehension-uart_254524.html

Note : si vous avez une difficulté avec l'anglais, utiliser l'option de « Traduire en français » dans le menu contextuel (clic droit) du navigateur.

Vous devrez présenter à l'enseignant la démonstration de chaque partie pour avoir vos points.

Manipulation 1 – La communication RS-232 TTL de base (UART)

Lors de cette manipulation, vous aurez à interconnecter deux plaquettes ESP32 via une liaison sérielle de type RS-232 TTL pour échanger de l'information entre deux ordinateurs, via l'interface du moniteur série de Arduino. Le schématique de principe est celui de la Figure 1 et que vous devez réaliser. Bob et Alice doivent s'échanger des messages.

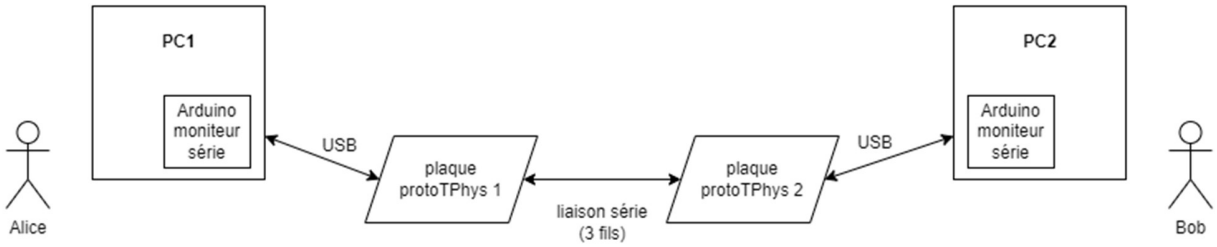

Figure 1 - Schématique des éléments de la communication série simple

Le développement à expérimenter est celui de l'envoi simple de messages entre deux (2) ESP32. La Figure 2 en présente un exemple schématisé de la communication, selon une ligne temporelle :

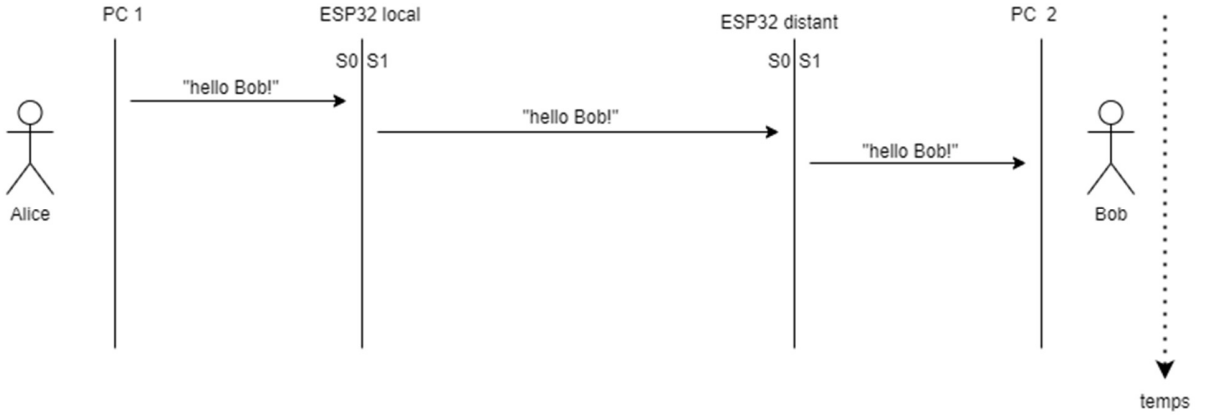

Figure 2 - Schéma de communication manipulation 1

Le EPS32 possède 3 ports sériels physiques en son cœur. Toutefois seulement deux sont exposés par défaut dans l'environnement Arduino (le 3^{e} par configuration). Ces ports physiques sont aussi désignés UART (« Universal Asynchronous Receiver Transmitter ») et constitués chacun d'une circuiterie logique et d'une mémoire temporisateur (un « buffer »). Notez que la communication est possible dans les deux sens et simultanément, soit dite « full-duplex ».

Puisque les deux microcontrôleurs sont perçus comme étant des DTE (Data Terminal Equipment), il faut utiliser un croisement des broches de communication, comme indiqué à la Figure 3 :

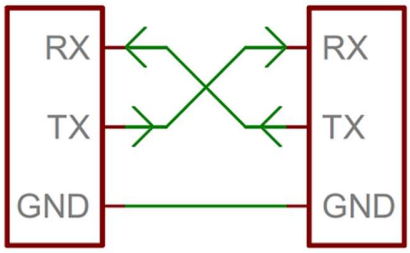

Figure 3 - Croisement des signaux entre deux DTE (ESP32)

Programmation

Pour le ESP32 : consulter cette façon de faire : https://microcontrollerslab.com/esp32-uart-communication-pins-example/

L'algorithme de chaque ESP32 de cette partie est symétrique et relativement simple : vous devez initialiser **deux** périphériques UART, l'un (S0) connectant l'ordinateur et le ESP32, l'autre (S1) connecte le ESP32 avec celui de votre voisin. Lors du fonctionnement normal, du point de vue du ESP32, lorsqu'un message est reçu sur l'un ou l'autre des ports série, le ESP32 doit retransmettre ce message sur l'autre port série. Notez que vous n'aurez qu'à développer qu'un (1) seul programme car il s'applique aux deux ESP32 (symétrique).

Indice : Consulter l'exemple de la référence docs.arduino.cc. Modifiez cet exemple à l'aide du contenu du lien de microcontrolerlabs.com et le tour joué.

Optionnel : au passage, il est possible de faire clignoter une DEL indiquant la réception/envoi d'une donnée. Par exemple : lors de la réception d'une donnée, changer l'état de la DEL.

- ⇨ **Question de réflexion**: est-ce possible d'ajouter un 3e participant à l'échange? Et un 4e? (faites votre recherche!). Détaillez votre réponse.

Observer et décoder des signaux du UART (RS-232) (optionel)

À l'aide du logiciel WaveForms et du Analog Discovery 2, tentez de prendre une trace figée à l'oscilloscope d'une transmission de 4 octets. Puis utiliser le mode analyseur logique (Logic Analyzer) de WaveForms pour capturer et décoder une transmissions numérique.

Conserver le programme de cette manipulation pour la remise.

Manipulation 2 – Protocole pour contrôle à distance

La première manipulation mettait en lumière la communication où le ESP ne faisait rien d'autre que de recopier l'information reçue d'un port série à l'autre port série (GIGO : « Garbage In, Garbage Out »).

Lors de cette seconde manipulation, le ESP32 écoutera les commandes provenant de l'usager via l'ordinateur PC et envoie un message « formatté » au ESP32 voisin afin que ce dernier interprète et agisse sur le message reçu. Ce message « formatté » est ce que l'on appelle un « protocole », c'est-à-dire un « contrat d'échange » entre les deux parties.

Le requis est le suivant :

1. L'usager peut commander (commande TGP_Decodeur) une des deux actions au ESP32 local, soient :
    - demander au ESP32 distant de changer l'état d'une DEL (action)
    - demander au ESP32 distant l'état du potentiomètre (« query » de la valeur numérique).
2. Le ESP32 local peut faire : répondre à l'usager local, commander de changer l'état d'une DEL ou questionner la lecture d'un potentiomètre à son homologue distant.
3. Vous devez <u>obligatoirement</u> utiliser la librairie TGP_Decodeur.

Le ESP32 local écoute donc, d'un côté l'usager, connecté via le port série 0 (USB, moniteur série Arduino) et de l'autre il envoie la commande correspondante à l'ESP32 distant (via un second port sériel).

Utiliser la DEL rouge comme DEL contrôlée par l'autre parti. Utiliser le potentiomètre de la plaquette pour fournir une lecture de tension (0-3.3v) convertie en mot numérique (0-4095), lorsque demandé.

La Figure 4 représente le diagramme des échanges, dans le temps, entre les systèmes impliqués.

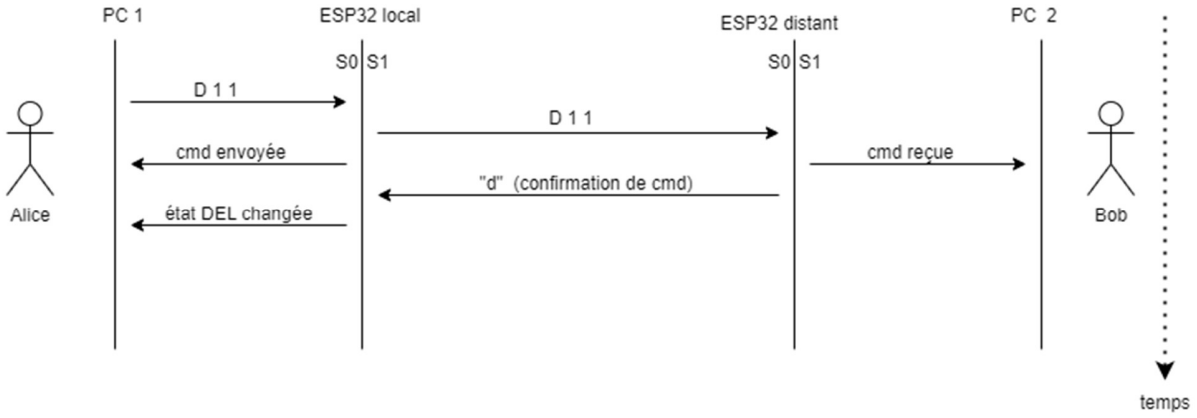

Figure 4 - Diagramme temporel des échanges du protocole TGP Decodeur

Programmation :

Vous devez compléter la programmation répondant au requis exposé plus haut en utilisant le code suivant : ESP32_RS232_protocole_tgpDecV2_Labo01_manip2_Vetudiant.ino

Conserver le code de cette manipulation pour la remise.

Manipulation 3 – Communication RS-485 (EIA-485)

Lors de la précédente manipulation, vous utilisiez une communication RS-232 TTL qui est sensible aux interférences induites sur le fil. Ainsi lorsque vous tenterez d'allonger les fils et continuer à échanger de l'information, il est fort probable que la communication soit corrompue par des influences extérieures. Vous avez alors besoin d'une liaison de communication plus fiable, résistant aux interférences. La solution réside dans l'utilisation d'un module transcepteur[1] (« transceiver » en anglais) dont la communication basée sur la norme EIA-485 (parfois aussi référé à « RS-485 ») est dotée. Toutefois la communication deviendra dans 1 seul sens à la fois, soit dite « half-duplex ».

Le module EIA-485 qui sera intercalé dans le montage est de type DCE (« Data Communication Equipment ») alors que le ESP32 est un DTE. Il faudra faire attention à la liaison entre le ESP32 et le module : pas de croisement de fil comme dans le cas de la communication DTE <-> DTE.

De plus, puisque le module fonctionne en mode « half-duplex », une broche supplémentaire est requise entre le ESP32 et le module RS-485 afin d'indiquer à ce dernier l'intention du ESP32 de transmettre. Par défaut le module est en mode de réception. Dès que l'un des deux 2 périphériques active sa transmission, il saisit le médium de communication, l'autre ESP32 demeure à l'écoute. Ce principe est le même que celui de la radio communication de type « walkie-talkie » : un émetteur à la fois, les autres écoutent.

Mentionnons également que le module fonctionne avec une alimentation à 5v. Sa logique numérique en est donc une à 5v, qui n'est directement pas compatible avec les niveaux logiques du ESP32 (3.3v). Il faudra donc utiliser un translateur de niveaux logiques pour adapter les signaux. Le schéma de principe de l'interconnexion pour ce montage est celui indiqué à la Figure 5 :

Figure 5 - Schéma de principe manipulation 3 (EIA-485)

Pour les détails de l'interconnexion, consulter le site https://microcontrollerslab.com/rs485-serial-communication-esp32-esp8266-tutorial/. Distinguez-bien le côté DTE-DCE du côté transmission/réception (A+, B-, COM).

Note : la broche COM est le niveau 0v = mise à la terre = ground = référence pour les niveaux des signaux voyageant sur les fils A+ et B-.

Programmation :

Pour vous permettre d'expérimenter simplement la liaison du protocole RS-485, vous reprendrez L'esprit de la manipulation 1. Il ne s'agit pas à 100% du même code, mais seulement quelques modifications. Prenez le temps de bien comprendre et mettre en place l'ajout du contrôle de la broche supplémentaire.

[1] Consulter la GDT : https://vitrinelinguistique.oqlf.gouv.qc.ca/fiche-gdt/fiche/8393130/emetteur-recepteur

À ce sujet, le ESP32 possède déjà la fonctionnalité, décrite dans la librairie HardwareSerial, consulter le fichier header (« .h »).

Attention, la communication est alors en mode « half-duplex », il faut que les usagers communiquent 1 à la fois.

Conserver le programme de cette manipulation pour la remise.

Manipulation 4 – Contrôleur et périphérique sur bus EIA-485

Pour cette manipulation, vous devrez prendre une copie de la manipulation 2 et y appliquer le principe ce que vous avez accompli lors de la manipulation 3. En somme vous aurez un montage ayant une architecture contrôleur-périphérique. Le contrôleur demande et commande les actions tandis que le périphérique fourni l'information demandée et exécute les actions, tout en réalisant ses tâches régulières (lire le potentiomètre, l'état d'un bouton, lire un capteur, etc). Le contrôleur initie la communication, il contrôle le bus de communication. Le périphérique demeure à la merci du contrôleur vis-à-vis le bus de communication.

Conserver le programme de cette manipulation pour la remise.

Manipulation 5 – Protocole Modbus

Le principe de communication EIA-485 ne définit en gros que la couche électrique de la communication[2]. Le protocole, c'est-à-dire ce qui détermine la forme des messages échangés sur le bus, la séquence des octets, peut être de toute sorte… mais pourquoi réinventer la roue lorsqu'il existe déjà une façon standardisée et performante d'utiliser le bus? C'est ici que nous introduisons le protocole Modbus[3].

Le protocole Modbus est basé sur une architecture contrôleur-périphérique et présente des fonctionnalités spécifiques. Vous devez respecter l'ordre sur le bus. En gros le contrôleur commande le périphérique par un jeu bien défini de commandes (« function code »). Exemple de commande envoyée par le périphérique : « périphérique 02, allume la DEL 04 » ou encore « périphérique 05, donne-moi la lecture du registre 10 (qui est une température, par exemple) ». À chaque commande, le périphérique doit retourner une réponse dans un temps limité.

Pour simplifier la séquence des instructions que le ESP32 doit exécuter pour envoyer ou traiter les commandes, des librairies sont disponibles. Parmi la myriade, nous allons en utiliser deux : *ModbusRTUSlave* et *ModbusRTUMaster*. Notez que les noms des librairies empruntent une façon déprécié[4] de nommage des éléments : *Master* pour désigner un contrôleur et *Slave* pour désigner un périphérique. Les deux librairies sont disponibles dans l'environnement Arduino.

[2] Consulter : https://en.wikipedia.org/wiki/RS-485
[3] Consulter : https://en.wikipedia.org/wiki/Modbus, principalement l'introduction.
[4] Il faut lire : https://fr.wikipedia.org/wiki/Ma%C3%AEtre-esclave

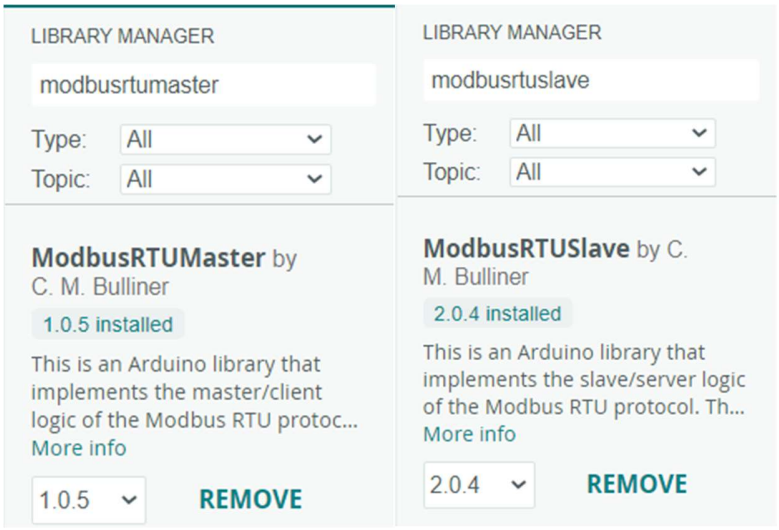

Figure 6 - Librairies ModbusRTU___ de CMB2

Pour cette manipulation, vous devez réaliser le montage selon ce que vous avez appris de la communication série : utiliser une interconnexion sérielle sur bus RS-485. Le module SHT20 doit être alimenté à +5V. Le code logiciel pour obtenir les données du module vous est fourni. Vous n'aurez qu'à installer la librairie ModbusRTUMaster.

La documentation du capteur SHT20 modbus est plutôt succincte et pas de très bonne qualité, vous en conviendrez. Consulter la fiche dans le document *sht20-modbus.png*. Vous y trouverez un tableau décrivant l'organisation des registres, comme suit :

MODBUS protocol:

Functional codes used in the product:
0x03: Read hold register
0x04: Read the input register
0x06: Write a single hold register
0x10: Write multiple hold registers

Register type	Register address	Data content	Number of bytes
Input register	0x0001	Temperature value	2
	0x0002	Temperature value	2
Hold register	0x0101	Device address (1-247)	2
	0x0102	Baud rate 0:9600 1:14400 2:19200	2
	0x0103	Temperature correction (/10)10.0~10.0	2
	0x0104	Humidity correction (/10)10.0~10.0	2

Figure 7 - Organisation des registres du module SHT20 modbus

Notez qu'à l'adresse 0x0002 la donnée est plutôt « Humidity value ».

Et on vous fournit un exemple d'interrogation par le Contrôleur (ici noté « Host ») ainsi que le résultat… et enfin comment interpréter le résultat. Notez que la séquence d'interrogation est réalisée par l'appel de la fonction de la librairie « *readInputRegisters* ».

MODBUS command frame:

Host reads temperature command frame (0x04)

From machine address	Function code	Register address High byte	Register address low byte	Register number High byte	Register number low byte	CRC High byte	CRC low byte
0x01	0x04	0x00	0x01	0x00	0x01	0x60	0x0a

Slave responds to data frames

From machine address	Function code	Number of bytes	Temperature High byte	Temperature low byte	CRC High byte	CRC low byte
0x01	0x04	0x02	0x01	0x31	0x79	0x74

Temperature value =0x131, converted to decimal305, the actual temperature value = 305/10 = 30.5℃
Note: Temperature is a signed hexadecimal number, temperature value =0xFF33, converted to decimal -205, the actual temperature = -20.5℃;

Figure 8 - Exemple de "framing" d'interrogation du capteur et sa réponse

Si tout est monté et configuré correctement, vous devriez obtenir sur le moniteur série ce qui suit :

```
Cmd: READ Type= IR sID=1 addr=1 len=2
Read 2 input registers(s) at ID 0x1 at address 0x1
      IR[0] : 212 (0xd4)
      IR[1] : 347 (0x15b)
```

Ceci représente le contenu de deux Input registers. L'index 0 est la température tandis que l'index 1 est l'humidité. Les valeurs décimales et hexadécimales sont affichées.

Vous devez modifier ce code pour afficher la donnée en format « humain », c'est-à-dire la valeur de la température affichée en degré Celsius, au dixième de degré et de même pour l'humidité.

Consulter la documentation des librairies :

- https://github.com/CMB27/ModbusRTUMaster
- https://github.com/CMB27/ModbusRTUSlave

Comprendre Modbus de manière brève :
https://arduinogetstarted.com/tutorials/arduino-modbus

Conclusion

Consignes pour le résumé de laboratoire.

- Le résumé de laboratoire est avant tout un recueil de vos notes, observations et résultats de vos travaux. Il a pour but de forger en vous une bonne pratique professionnelle, en tant que futur technologue, de consigner et de décrire vos travaux sur des mandats qui vous seront confiés.
- Je veux avoir, sur peu de pages, sections bien identifiées :

  Une **page couverture** vous identifiant, titre du travail, présenté à, la date de remise, lieu de remise (CÉGEP A-L).

  Introduction :

  Date des manipulations accompagné d'une brève description des travaux réalisés pour chaque période.

  Rappel des objectifs du laboratoire (pourquoi faire ce labo?).

  Développement :

  Détailler sommairement pour chacune des manipulations :

  - Comment vous avez accompli le travail.
  - Détailler de façon simple l'algorithme de votre code logiciel.
  - Indiquer les source consultées, s'il y a lieu.

  Conclusion :

  Discussion sur les résultats obtenus, notes et observations pertinentes.

  Ce que vous avez appris ou découvert (ou peut-être pas).

  Une critique constructive de cet exercice de laboratoire. Votre appréciation de ce laboratoire. Soyez honnête.

- Consigne pour la remise
  - Rapport format Word
  - Remettre le programme Arduino de chacune des manipulations
  - **SVP: Aucun fichier archive (zip)**

Références

- Comprendre Modbus de manière brève : https://arduinogetstarted.com/tutorials/arduino-modbus
- https://docs.arduino.cc/learn/communication/uart/
- https://microcontrollerslab.com/esp32-uart-communication-pins-example/
- https://microcontrollerslab.com/rs485-serial-communication-esp32-esp8266-tutorial/
- https://www.analog.com/en/resources/technical-articles/rs485-cable-specification-guide--maxim-integrated.html
- https://www.rohde-schwarz.com/fr/produits/test-et-mesure/essentials-test-equipment/digital-oscilloscopes/comprehension-uart_254524.html
- https://arduinogetstarted.com/tutorials/arduino-modbus