Université Sidi Mohammed Ben Abdellah
Faculté des sciences Dhar El-Mehraz
Master smart industry

TP2 : Communication Modbus RTU entre deux Arduino

Relaisé par :

1. **Raheb Aref Mahyoub Saeed**
2. **Ahmed Karim**

Encadré Par :

Pr. Nabil Kabbaj
Pr. Yassine Boukhali

Année Universitaire: 2024 – 2025

Table des matières

I

introduction

Le protocole **Modbus RTU** est une norme de communication série très répandue dans le domaine de l'automatisation industrielle. Il permet l'échange de données entre un appareil **maître** et un ou plusieurs appareils **esclaves** via une liaison série, généralement sur un bus **RS-485**, reconnu pour sa fiabilité, sa résistance au bruit et sa capacité à couvrir de longues distances.

Ce travail pratique a pour objectif de mettre en œuvre une **communication Modbus RTU entre deux cartes Arduino**, afin de comprendre le fonctionnement du protocole dans une architecture maître-esclave, et de maîtriser la configuration matérielle et logicielle nécessaire à son implémentation. Le TP permet également d'apprendre à manipuler les **registres Modbus** (notamment les registres de bobine et registres de maintien ou *coil register and holding registers*), ainsi que de gérer les **erreurs de transmission** et la **synchronisation série**.

Dans notre réalisation, une carte Arduino agit en tant que **maître**, en envoyant des données périodiquement à une autre carte Arduino configurée comme **esclave**. Les deux cartes communiquent à l'aide de **modules RS-485** (MAX485), permettant une communication série différenciée. Ce projet se situe dans un contexte pédagogique, mais repose sur des bases technologiques directement applicables aux systèmes **SCADA**, aux automates programmables industriels (**PLC**) et aux réseaux de capteurs.

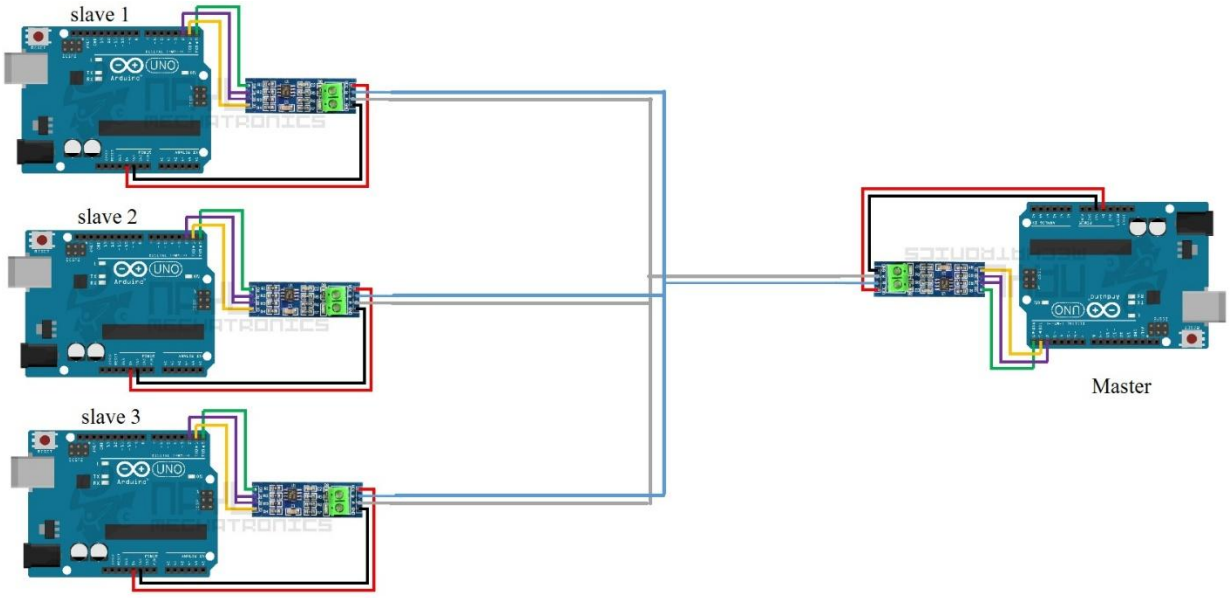

Figure 1 :schéma maître-esclave avec bus RS-485)

Présentation du TP

Ce travail pratique vise à concevoir et tester une communication série **Modbus RTU** entre deux cartes **Arduino**, en suivant une architecture **maître-esclave**. Cette structure est typique des systèmes industriels où un dispositif maître contrôle et interroge plusieurs esclaves connectés sur un bus série commun. Dans notre cas, le **maître Arduino Uno** envoie régulièrement des données (valeurs croissantes) à l'**esclave Arduino**, via un **bus RS-485**, utilisant les broches différentielles A et B pour transmettre les signaux.

La communication repose sur l'utilisation de **modules RS-485 (type MAX485)**, connectés aux cartes Arduino à travers les broches **TX**, **RX** et **DE/RE** (permettant de gérer l'alternance émission/réception). Le maître utilise la bibliothèque **ModbusMaster** pour envoyer des données à l'adresse d'un registre spécifique de l'esclave. L'esclave, de son côté, utilise la bibliothèque **ModbusRTUSlave**, qui configure un tableau de **registres de maintien (holding registers)** accessibles en écriture par le maître. À chaque requête, les données sont enregistrées et un retour est affiché dans le moniteur série.

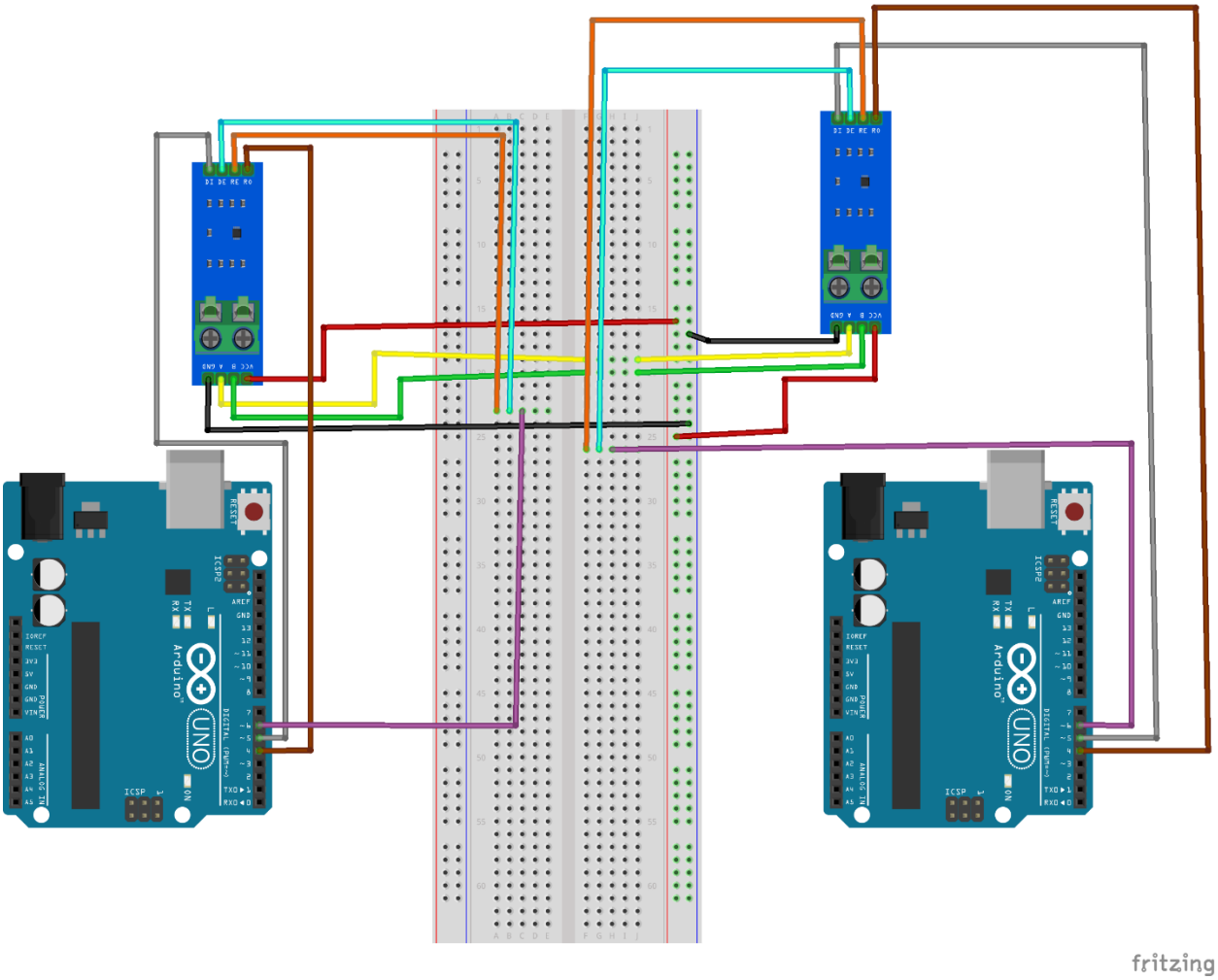

Figure 2: Schéma Fritzing du montage maître-esclave via RS-485

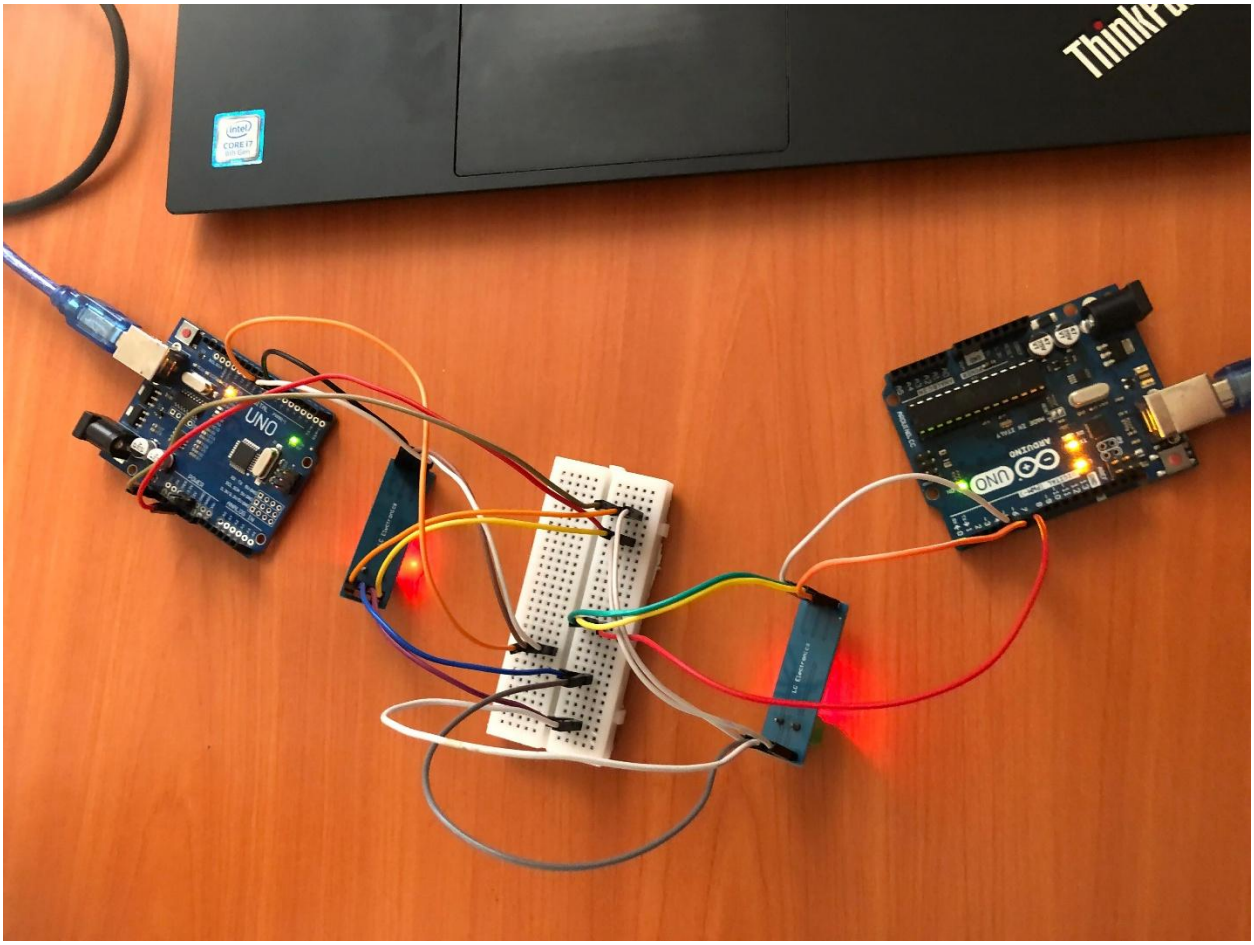

Figure 3Photo du montage réel sur breadboard

Le **maître** envoie une nouvelle valeur tous les 2 secondes via la fonction writeSingleRegister(), tandis que l'**esclave** affiche à intervalles réguliers l'état de ses registres. Cette configuration permet d'observer en temps réel la bonne réception des données et de tester la robustesse de la transmission. En plus du code, une gestion des erreurs est incluse, permettant au maître de détecter et afficher les problèmes tels que les délais de réponse ou les erreurs CRC.

Ce TP permet donc de mettre en œuvre une communication industrielle fiable entre deux microcontrôleurs, de configurer une ligne RS-485, et d'analyser le comportement de chaque appareil selon le protocole Modbus RTU. Il constitue une base solide pour aborder les systèmes de contrôle distribués en milieu professionnel.

Matériel et composants utilisés

Pour la réalisation de cette communication Modbus RTU entre deux cartes Arduino, nous avons utilisé un ensemble de composants électroniques permettant d'assurer à la fois la transmission série, l'alimentation stable, et l'observation des résultats. Voici les éléments essentiels:

Tableau 1: présenter l'ensemble des composants posés pour le montage

Composant	**Quantité**	**Rôle dans le TP**
Arduino Uno	2	Microcontrôleurs utilisés comme maître et esclave
Module RS-485 MAX485	2	Interface de communication différée via bus série RS-485
Fils Dupont (Mâle-Mâle)	~15	Connexions entre Arduino, modules RS-485, et breadboard
Breadboard	1	Support de câblage temporaire pour le montage du circuit
Alimentation USB	2	Fourniture d'énergie aux deux cartes Arduino
Câble RS-485 (ligne A et B)	1	Ligne de communication série entre les deux modules
Ordinateur avec Arduino IDE	1	Programmation des cartes et surveillance via le moniteur série

L'ensemble du montage est entièrement alimenté via USB, ce qui simplifie le câblage et permet une programmation directe depuis l'environnement Arduino IDE. La configuration matérielle respecte les contraintes classiques de l'implémentation RS-485 en point-à-point, avec une gestion logicielle de la direction de transmission par une broche dédiée (DE/RE).

Tous les composants sélectionnés sont compatibles avec une communication série à **9600 bauds**, ce qui convient pour des tests pédagogiques tout en maintenant une bonne fiabilité du lien.

Schéma du circuit

La communication série Modbus RTU entre les deux cartes Arduino est réalisée à l'aide de **modules RS-485 (type MAX485)** connectés via les lignes différentielles **A** et **B**. Le câblage a été réalisé conformément aux exigences du protocole demi-duplex RS-485 et à la configuration des bibliothèques utilisées, notamment la bibliothèque ModbusRTUSlave pour l'esclave.

Le schéma de connexion, illustré dans la **Figure 1**, a été réalisé avec l'outil **Fritzing**. Il montre les interconnexions entre les deux cartes Arduino, les modules RS-485, et les liaisons de contrôle nécessaires au bon fonctionnement de la communication série.

Les connexions du module RS-485 du côté esclave sont configurées comme suit :

*Tableau 2: Connection d'Arduino avec Module **RS-485***

RO (Receiver Output)	pin 4 (RX SoftwareSerial de l'Arduino)
DI (Driver Input)	pin 5 (TX SoftwareSerial de l'Arduino)
DE/RE (contrôle ligne)	pin 6 (broche de commande de direction)
VCC	5V
GND	GND
A/B	lignes de données différentielles vers le maître

La Figure 2 illustre le montage physique du circuit tel qu'il a été réalisé sur breadboard. On y distingue clairement les modules RS-485, les cartes Arduino, les câbles de liaison et les connexions d'alimentation. Ce montage a permis de tester efficacement la communication série dans des conditions proches de celles d'un environnement industriel simplifié.

Ce câblage respecte les recommandations techniques du protocole Modbus RTU, avec gestion logicielle de la direction de transmission (via la broche DE/RE), séparation claire des lignes de communication, et compatibilité avec les vitesses de transmission standards (ici 9600 bauds).

Description du code

La communication Modbus RTU mise en œuvre dans ce TP repose sur deux programmes distincts : l'un pour le **maître**, chargé d'envoyer des données, et l'autre pour l'**esclave**, chargé de recevoir et enregistrer ces données dans des registres. Ces deux programmes sont conçus pour fonctionner avec des bibliothèques Arduino compatibles avec le protocole Modbus RTU, respectant une communication série demi-duplex via des modules RS-485.

1. Code du maître

Le maître est une carte **Arduino Uno** qui utilise la bibliothèque **`ModbusMaster`** pour établir une communication Modbus RTU avec un esclave identifié par l'ID 1. La transmission s'effectue à travers une liaison série logicielle (**`SoftwareSerial`**), et la direction du flux série est contrôlée par une broche dédiée (RS485_CTRL_PIN).

À chaque itération (toutes les 2 secondes), une valeur entière est incrémentée puis envoyée à l'esclave à l'adresse du **registre de maintien 0 (holding register 0)** à l'aide de la commande :

```
node.writeSingleRegister(0, sendData);
```

Le maître attend ensuite un court délai, puis tente de lire la même adresse pour vérifier que la valeur a bien été enregistrée. Le retour d'état (succès ou échec) est affiché sur le moniteur série. En cas d'erreur, un message explicite est produit grâce à une fonction dédiée printModbusError() qui interprète les codes d'erreurs Modbus tels que : *timeout*, *ID invalide*, ou *CRC incorrect*.

Le système de contrôle DE/RE est géré par deux fonctions : preTransmission() et postTransmission() qui activent ou désactivent la transmission sur la ligne RS-485.

2. Code de l'esclave

L'esclave est également une carte **Arduino Uno**, qui utilise la bibliothèque `ModbusRTUSlave` développée par CMB27. Il est configuré pour répondre en tant qu'esclave Modbus avec l'ID 1, et gère un tableau de **10 registres de maintien (holdingRegisters[10])**, initialisés à zéro.

Le programme utilise également `SoftwareSerial` pour établir une liaison série sur des broches dédiées, et configure le module RS-485 avec une broche de contrôle pour la direction de communication. La fonction `modbus.poll()` est appelée en continu dans la boucle principale (`loop()`) pour écouter et répondre aux requêtes du maître.

Un affichage périodique (chaque seconde) permet de visualiser sur le moniteur série la valeur courante de tous les registres. Cela offre une vérification en temps réel de la bonne réception des données envoyées par le maître.

Extrait de configuration des registres :

```
modbus.configureHoldingRegisters(holdingRegisters, NUM_REGISTERS);
```

Cela signifie que l'esclave autorise la lecture et l'écriture sur une plage de 10 registres consécutifs à partir de l'adresse 0.

Cette structure logicielle permet une communication fluide, contrôlée et surveillée en continu entre les deux cartes. Les deux programmes assurent la synchronisation des données et la gestion des erreurs, ce qui reflète les pratiques des systèmes industriels utilisant Modbus RTU.

Résultats et observations

Après avoir correctement câblé le circuit et téléversé les programmes sur les deux cartes Arduino, la communication Modbus RTU a été testée dans des conditions réelles. Le système a permis une **transmission fiable de données du maître vers l'esclave** via le bus RS-485, avec une fréquence d'envoi configurée à **une fois toutes les deux secondes**.

Du côté **maître**, le moniteur série affichait à chaque cycle la valeur envoyée à l'esclave, ainsi qu'un message de confirmation en cas de transmission réussie. Lorsqu'une lecture était effectuée après l'écriture, la valeur retournée par l'esclave correspondait exactement à la valeur envoyée, ce qui confirmait la **cohérence des données** échangées. En cas d'anomalie (par exemple si l'esclave n'était pas alimenté ou mal câblé), le maître affichait un message d'erreur clair tel que Response timeout ou Invalid CRC, grâce à la fonction printModbusError().

Côté **esclave**, le moniteur série affichait chaque seconde l'état actuel de tous les **registres de maintien**. À chaque transmission réussie, la valeur du **registre 0** était mise à jour, conformément à la commande envoyée par le maître. Ce comportement confirme que le système esclave recevait et traitait correctement les requêtes Modbus. Le tableau de registres restait stable, et aucune déformation ou saut de valeur n'a été observé pendant l'exécution du programme.

Ces résultats démontrent la **stabilité de la communication**, ainsi que le bon fonctionnement de la configuration RS-485, tant sur le plan matériel que logiciel. Aucun blocage ou erreur critique n'a été détecté pendant les tests prolongés.

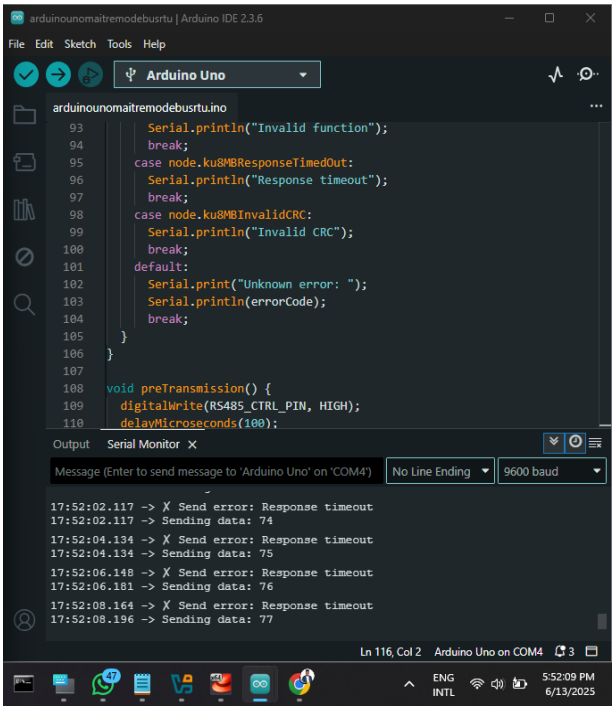

Figure 5: Capture d'écran du moniteur série côté maître (transmission réussie)

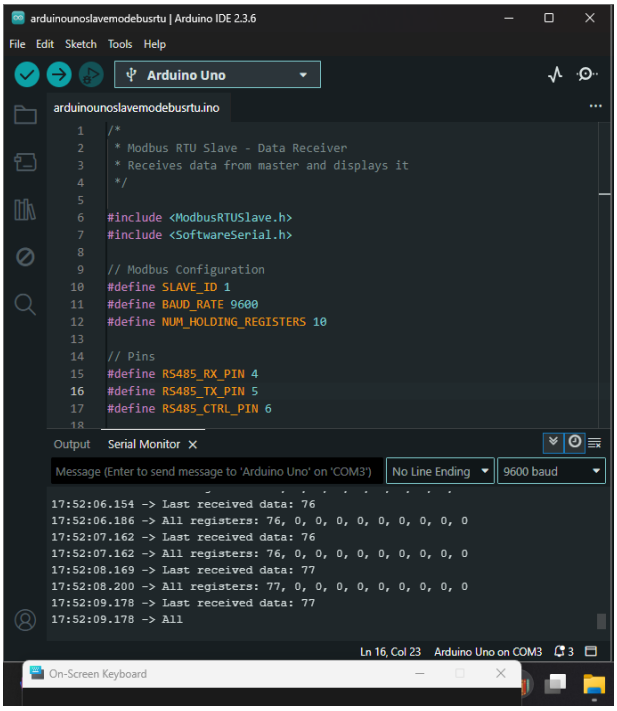

Figure 4: Capture d'écran du moniteur série côté esclave (valeurs des registres affichées en temps réel)

Ces deux figures montrent le système en fonctionnement : la **valeur envoyée et confirmée par le maître**, et la **réception correcte affichée par l'esclave**. Cela confirme que la synchronisation entre les deux dispositifs est bien respectée.

Problèmes rencontrés et solutions

Au cours de la réalisation de ce TP, plusieurs problèmes techniques ont été rencontrés, principalement liés à la configuration logicielle des bibliothèques et à l'utilisation des ports de communication série. Ces difficultés sont courantes dans les projets embarqués et ont nécessité des ajustements spécifiques pour assurer le bon fonctionnement du système.

1. Problème de compatibilité des bibliothèques

Lors de l'installation des bibliothèques nécessaires (`ModbusMaster` pour le maître et `ModbusRTUSlave` pour l'esclave), une incompatibilité a été constatée. En effet, certaines versions des bibliothèques ne fonctionnaient pas correctement avec les cartes Arduino utilisées. Ce problème était dû à l'installation automatique sans vérification de la **version compatible**.

Solution :
Il a été nécessaire de consulter la documentation officielle des bibliothèques et de vérifier manuellement la version requise. Pour `ModbusRTUSlave`, la version stable développée par

CMB27 a été utilisée. L'installation correcte a été faite via **Outils > Gérer les bibliothèques**, en recherchant spécifiquement le nom de la bibliothèque et en sélectionnant une version stable et testée.

2. Problème de communication via les broches RX/TX standards (0 et 1)

Un autre problème a été rencontré en utilisant les broches **0 (RX)** et **1 (TX)** de la carte Arduino Uno pour établir la communication avec le module RS-485. Ces broches sont également utilisées par le **port série USB**, ce qui a provoqué un conflit avec le moniteur série. En conséquence, **aucune donnée n'était visible dans le moniteur série**, rendant impossible le suivi du fonctionnement du programme.

Solution :
Pour contourner ce conflit, une **liaison série logicielle** a été mise en place à l'aide de la bibliothèque `SoftwareSerial`, en utilisant les broches numériques **4 (RX)** et **5 (TX)**. Cette solution a permis de séparer la communication Modbus de la liaison USB/Moniteur série, assurant une transmission fiable des données tout en conservant la possibilité d'afficher des messages de débogage.

3. Incompatibilité de SoftwareSerial avec Arduino Mega

Enfin, lors des essais avec une carte **Arduino Mega**, il a été constaté que la bibliothèque `SoftwareSerial` ne fonctionnait pas correctement. Les broches choisies pour RX et TX ne réagissaient pas, et aucune communication série Modbus n'était détectée.

Explication :
La carte Arduino Mega dispose de **plusieurs ports série matériels (Serial1, Serial2, Serial3)**. Or, la bibliothèque `SoftwareSerial` n'est pas recommandée sur Mega, car ses broches numériques ne supportent pas toutes les interruptions nécessaires au fonctionnement de la communication série logicielle.

Conclusion

Ce travail pratique a permis de mettre en œuvre une communication **Modbus RTU** entre deux cartes **Arduino Uno**, en suivant une architecture **maître-esclave** via le protocole **RS-485**, largement utilisé dans les systèmes industriels pour sa robustesse et sa fiabilité.

Grâce à l'utilisation des bibliothèques ModbusMaster et ModbusRTUSlave, nous avons pu configurer efficacement les deux cartes : le maître envoyant périodiquement une donnée, et l'esclave la recevant et l'enregistrant dans un registre de maintien. L'affichage dans le moniteur série des deux côtés a permis une **vérification en temps réel de la transmission**, confirmant la synchronisation correcte entre les dispositifs.

Ce TP a également été l'occasion d'approfondir plusieurs notions clés :

- La structure des **registres Modbus** et leur adressage,
- La **gestion de la direction de communication** sur un bus RS-485 en mode demi-duplex,
- La configuration des **liaisons série matérielles et logicielles** (SoftwareSerial),
- Et la résolution de problèmes techniques fréquents (conflits RX/TX, compatibilité des bibliothèques).

En somme, cette expérimentation constitue une introduction pratique aux **communications industrielles série**, tout en posant les bases pour des applications futures plus complexes, telles que l'intégration de capteurs, l'interfaçage avec des automates programmables industriels (API/PLC) ou encore la mise en réseau de plusieurs esclaves.