Département de Génie Electrotechnique et Automatique
Université de 8 Mai 1945 Guelma

Licence 3ème année Automatique
Module : Bus de communication et réseaux locaux industriels

TP5: Utilisation du protocole ModBus

1. Introduction

Dans ce TP, nous abordons un autre bus de communication industrielle évolué et un nouveau protocole de communication. En particulier, nous utilisons le bus RS485 avec le protocole ModBus pour interagir avec un compteur de consommation électrique supportant le même protocole. Le but de ce TP consiste à initier l'étudiant pour faire un montage de bonne gestion de consommation électrique (consommation intelligente).

Comme le compteur de consommation fonctionne en 220v, il est hautement recommander de prendre toutes les précautions afin d'éviter tout dégât (un choc électrique). En particulier, on doit porter des gants d'électricité et on veuille à respecter les directives de raccordement.

2. Principe de fonctionnement du protocole ModBus

Le protocole ModBus est un protocole libre créé par la société Modicon et est basé sur une structure hiérarchisée entre le maître et les esclaves [1]. La communication via ModBus est une communication half-duplex utilisant deux ou quatre fils qui permettent de transmettre 9600-19200 bits/s, où on distingue deux scénarios de communication. Le premier scénario consiste à transmettre un message par le maître et attendre la réponse de l'esclave, tandis que le deuxième consiste à diffuser un message à tous les esclaves sans attendre une réponse. Il est important de noter que les esclaves ne peuvent pas communiquer entre eux, et chaque esclave demande une permission auprès du maître pour envoyer la réponse.

START	ADDRESS	FUNCTION	DATA	CRC CHECK	END
T1–T2–T3–T4*	8 BITS	8 BITS	*n* x 8 BITS	16 BITS	T1–T2–T3–T4*

*For T1–T2–T3–T4, 3.5 character times at no communication.

Modbus Frame Structure-RTU Mode

Figure 5.1. Structure d'un message du protocole ModBus [2]

Le message échangé contient quatre champs tels que le numéro de l'esclave codé sur 1 octet (0-64), l'instruction à exécuter codée sur 1 octet (19 instructions possibles), les données

composées de plusieurs caractères et le contrôle d'erreur CRC [1]codé sur 2 octets. On note que l'adresse 0x00 est réservée au mode de diffusion, ainsi la taille maximale des données est 256 octets.

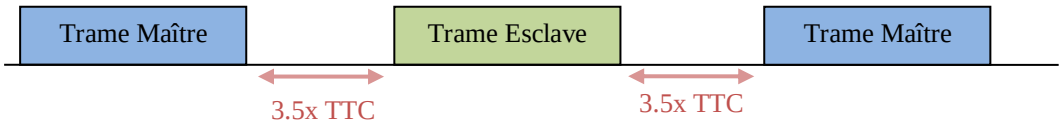

Figure 5.2. Echange de données entre le maître et l'esclave. TTC signifie le temps de transmission d'un caractère

Chaque message est délimité par un silence de minimum 3.5 fois le temps de transmission d'un caractère (TTC). Au cours de la transmission, s'il y a un délai de 1.5 TTC entre la transmission de deux caractères successifs, le système détecte une erreur de transmission. Un caractère en ModBus est une suite de 11 bits dont 1 bit de début, 8 bits de données, 1 bit de parité et 1 bit d'arrêt.

Le protocole ModBus est basé sur la transmission sérielle et fonctionne à base de RS232, RS422, RS485 et Ethernet TCP/IP, où chaque support de transmission a des avantages et des inconvénients. Comme nous nous intéressons au RS485, il offre la possibilité de communiquer avec 32 périphériques sur une distance de 1200m maximum, ainsi il est tolérant aux parasites.

Tableau 5.1. Liste des codes de fonctions utilisées par le protocole ModBus [1]

Code	Nature des fonctions MODBUS
H'01'	Lecture de n bits de sortie consécutifs
H'02'	Lecture de n bits de sortie consécutifs
H'03'	Lecture de n mots de sortie consécutifs
H'04'	Lecture de n mots consécutifs d'entrée
H'05'	Ecriture de 1 bit de sortie
H'06'	Ecriture de 1 mot de sortie
H'07'	Lecture du statut d'exception
H'08'	Accès aux compteurs de diagnostic
H'09'	Téléchargement, télé déchargement et mode de marche
H'0A'	Demande de CR de fonctionnement
H'0B'	Lecture du compteur d'événements
H'0C'	Lecture des événements de connexion
H'0D'	Téléchargement, télé déchargement et mode de marche
H'0E'	Demande de CR de fonctionnement
H'0F'	Ecriture de n bits de sortie
H'10'	Ecriture de n mots de sortie
H'11'	Lecture d'identification
H'12'	Téléchargement, télé déchargement et mode de marche
H'13'	Reset de l'esclave après erreur non recouverte

[1] CRC est l'abréviation de *Cyclic Redundancy Control*

Il est possible de communiquer avec 248 périphériques, mais en utilisant des répéteurs sur le même réseau. Vu que la transmission par défaut est basée sur deux fils, on peut utiliser quatre fils pour passer en mode full-duplex qui permet d'obtenir un débit de transmission plus rapide.

3. Matériel utilisé dans ce TP

Afin de réaliser ce TP et pratiquer le protocole SPI, on utilisera le matériel suivant :

- Une carte Arduino UNO ou Arduino Mega ;
- Un compteur de consommation électrique SDM230-MODBUS de la marque EASTRON;
- Un module RS485 pour Arduino ;
- Des fils de prototypage male-femelle (*male-male en cas d'utiliser une plaque d'essai*) ;
- Un câble électrique de section 1.5mm ou 2.5mm pour raccorder le compteur à la prise ;
- Un équipement quelconque fonctionnant en 220v comme un ampoule.

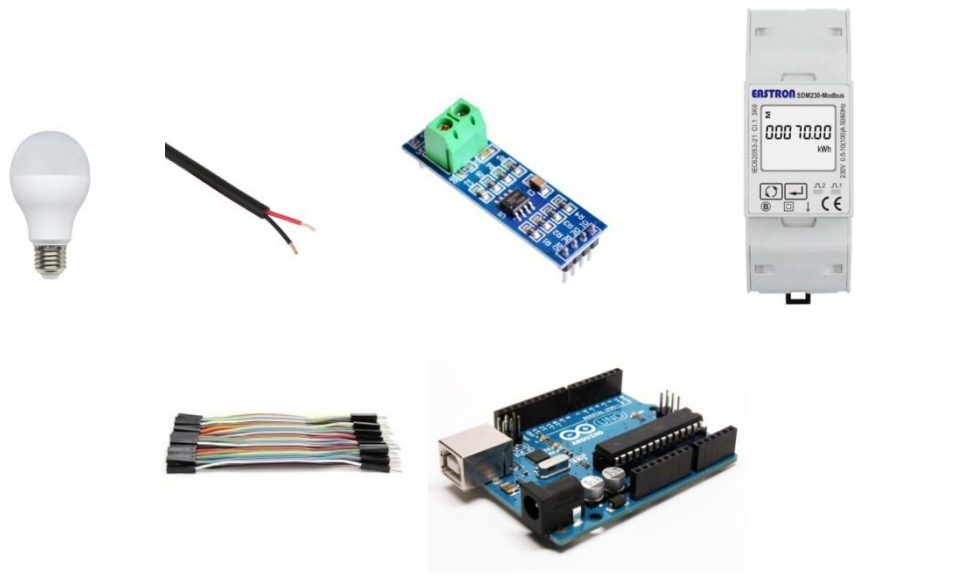

Figure 5.3. Matériel utilisé pour faire communiquer deux Arduino via le bus CAN

4. Interfaçage d'un compteur d'énergie monophasé via le protocole ModBus

Dans ce TP, nous interfaçons un compteur de consommation électrique avec un Arduino via le protocole ModBus, et nous essayons de manipuler le compteur à distance. Donc, on a besoin d'une carte Arduino, un module MAX485 qui convertit de RS485 ModBus vers un

signal TTL[2] et vice versa, des fils de prototypage et un compteur d'énergie dotée d'une interface ModBus comme le modèle SDM230 de la marque EASTRON. Donc, dans ce TP, l'Arduino sert comme maître et le compteur est utilisé comme esclave.

La première étape à faire consiste à relier l'Arduino avec le module MAX485, et pour ce faire on alimente le module (VCC) avec 5v de l'Arduino et on branche le GND du module avec celui de l'Arduino. Ensuite, on branche le port A (ligne non inversée) dans le TX+/RX+ du compteur et le port B (ligne inversée) dans le TX-/RX- du compteur.

On branche les quatre ports qui restent tels que R0 (receiver output), RE (receiver enable), DE (driver enable), DI (driver input) dans quatre ports numériques de l'Arduino (e.g. 0, 3, 3 et 1, respectivement). On remarque qu'on a utilisé les ports du bus UART avec lequel l'Arduino communique avec le ***Serial Monitor*** de l'ordinateur. Dans le cas où on interagit avec ce dernier, il est préférable d'utiliser la bibliothèque ***SerialSoftware*** pour émuler le bus UART dans des ports différents (e.g. ports 4 et 5).

Figure 5.4. Compteur de consommation électrique SDM230-MODBUS de la marque EASTRON

Chacun des quatre ports correspond à un usage. Par exemple, le port R0 est destiné à lire les données entrantes, DI pour envoyer des données sortantes, RE pour activer le mode de réception et DE pour activer le mode d'envoi. On note qu'on a utilisé le même port numérique (numéro 3) pour RE et DE, car lorsque RE est mis à LOW le mode de réception est activé, tandis que lorsque DE est mis à HIGH le mode d'envoi est activé.

[2] TTL est l'abréviation de *Transistor-to-Transistor Logic*

Figure 5.5. Module MAX485 ModBus qui convertit du RS485 vers TTL et vice-versa

Pour interfacer le module ***MAX485*** afin de recevoir et envoyer des données, on a deux possibilités, soit on utilise la bibliothèque ***ModBusRtu***[3] qui doit être téléchargée, ou bien on envoie les paquets octet par octet en utilisant l'objet ***Serial***. En référant à la liste des codes des fonctions utilisées dans le protocole ModBus (Tableau 7.1), le code 0x04 est dédié pour lire une suite de caractères consécutifs d'entrée. Donc, on envoie un paquet contenant l'adresse de l'esclave (0x01 par défaut) et le code de la fonction de lecture.

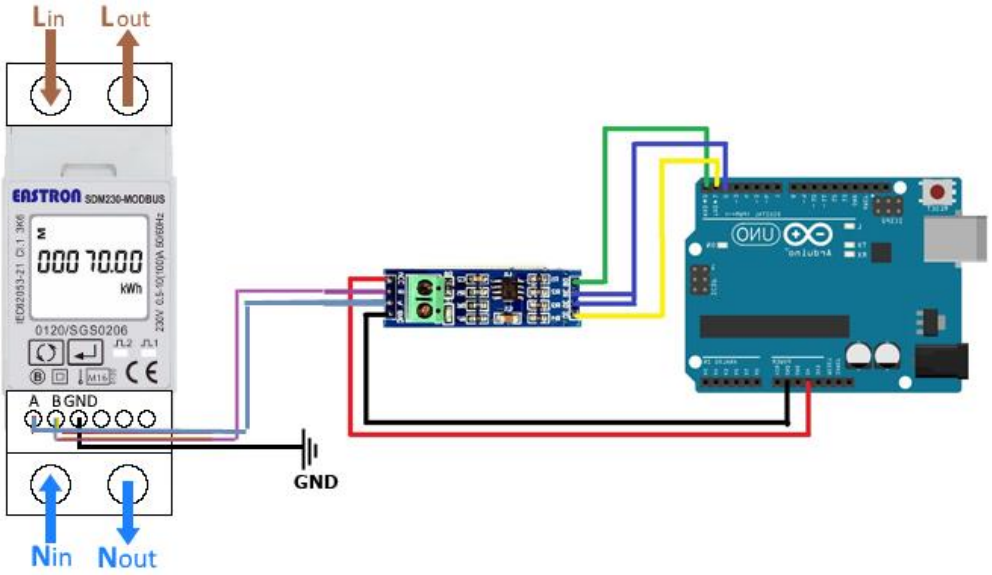

Figure 5.6. Schéma de branchement du module MAX485 et le compteur SDM230 sur l'Arduino

Adresse d'esclave	Code fonction	Adresse de début (Hi)	Adresse de début (Lo)	Nombre de points (Hi)	Nombre de points (Lo)	CRC (Hi)	CRC (Lo)

Figure 5.7. Structure d'un paquet d'une requête

Afin de transmettre le paquet de commande de lecture, on active le mode d'écriture avec l'état HIGH dans DE et systématiquement on bloque la lecture (RE = HIGH). Puis, en se référant au manuel[4] du constructeur du module, on prépare le paquet contenant 8 octets dont la structure illustrée par la Figure 5.7. Il faut savoir toutefois que le module EASTRON SDM230 accepte seulement quelques codes de fonctions tels que 0x03, 0x04, 0x08 et 0x0F. Par conséquent, le module répond avec 9 octets dont la structure illustrée par la Figure 5.8, ainsi pour recevoir le paquet on doit mettre LOW dans RE.

Adresse d'esclave	Code fonction	Nombre d'octets	Premier registre (Hi)	Premier registre (Lo)	Second registre (Hi)	Second registre (Lo)	CRC (Hi)	CRC (Lo)

Figure 5.8. Structure d'un paquet de réponse à une requête

[3] *https://github.com/smarmengol/Modbus-Master-Slave-for-Arduino/blob/master/ModbusRtu.h*
[4] Manuel d'utilisation de ModBus avec SDM230. *https://www.desouttertools.com/resource-centre/file/download/126505189*

A titre d'exemple, pour lire le voltage d'entrée (alternatif) à partir du module SDM230, on met l'adresse 0x01 comme adresse d'esclave, 0x04 comme code de fonction pour lire plusieurs caractères consécutifs d'entrée (Tableau 5.1), 0x00 0x00 comme adresse de début d'après le manuel (lire le voltage), 0x00 0x02 comme nombre de registres à lire (2 registres à lire) et enfin la valeur de CRC (e.g. 0x71, 0xCB).

0x01	0x04	0x00	0x00	0x00	0x02	0x71	0xCB

Figure 5.9. Exemple de requête de lecture du voltage d'entrée via le protocole ModBus

```
1.  #define RE_DE 4
2.
3.  int Buffer[9];
4.  byte VoltageReadCommand[] ={0x01 , 0x04 , 0x00 , 0x00 , 0x00 , 0x02 , 0x71 , 0xCB};
5.
6.  void setup()
7.  {
8.      Serial.begin(2400);
9.
10.     pinMode(RE_DE , OUTPUT);
11.     digitalWrite(RE_DE , HIGH);
12. }
13.
14. void loop()
15. {
16.     digitalWrite(RE_DE , HIGH);
17.
18.     for (int i=0; i < 8; i++) Serial.write( VoltageReadCommand[i] );
19.     delay(50);
20.
21.     digitalWrite(RE_DE , LOW);
22.
23.     while(Serial.available() < 9);
24.     for (int i=0; i < 9; i++) Buffer[i] = Serial.read();
25.
26.     Serial.flush();
27.     Serial.println();
28.
29.     Serial.print(Buffer[3] , HEX); Serial.print(" ");
30.     Serial.print(Buffer[4] , HEX); Serial.print(" ");
31.     Serial.print(Buffer[5] , HEX); Serial.print(" ");
32.     Serial.print(Buffer[6] , HEX); Serial.print(" ");
33.     Serial.println();
34.
35.     float x;
36.     ((byte*) &x)[3]= Buffer[3];
37.     ((byte*) &x)[2]= Buffer[4];
38.     ((byte*) &x)[1]= Buffer[5];
39.     ((byte* )&x)[0]= Buffer[6];
40.
41.     Serial.print(x , 2);
42.     Serial.println("Volts");
43.     delay(500);
44. }
```

Figure 5.10. Code source de lecture du voltage d'entrée à partir du module SDM230

Après la réception du paquet de réponse, on extrait les quatre octets représentant le voltage et on les converti en valeur flottante (lignes 35-39 Figure 5.10), car le courant alternatif n'est pas stable et contient la virgule.

Références

[1] Liaison série Modbus RS485. Cours réseau de terrain, Université de Grenoble

[2] Format du message ModBus. *https://www.rfwireless-world.com/Terminology/Modbus-message-frame.html* [Consulté en 10/2019]