Modbus sans fil (partie 2)

Logiciel de la carte Modbus TCP sans fil

Josef Bernhardt et **Martin Mohr** (Allemagne)

Construit autour d'un module *NodeMCU d'Espressif*, la carte décrite dans la première partie de cette série permet d'utiliser le protocole *Modbus* sur un réseau sans fil. Cet article couvre le logiciel et la configuration de la carte Modbus.

Dans la 1ère partie [1] de cette série d'articles, nous avons abordé le matériel du projet. La carte est bâtie autour d'un module NodeMCU équipé d'un µcontrôleur *ESP8266*, avec une carte additionnelle fournissant des ports de type industriel. Après une brève introduction au principe du protocole Modbus, nous étudions ici la façon dont la carte est pilotée et le logiciel utilisé à cet effet. La commande de porte d'ascenseur de la 1ère partie sert à nouveau d'exemple.

Le protocole Modbus

Le protocole Modbus est largement utilisé dans l'automobile. Il fonctionne selon le principe maître/esclave. Le maître du bus peut commander jusqu'à 246 esclaves. Les adresses 1 à 247 peuvent être attribuées aux nœuds. L'adresse 0 est réservée aux données de diffusion ; tous les nœuds reçoivent les données envoyées à l'adresse 0. Des sommes de contrôle (CRC) assurent l'intégrité des différents paquets. Les esclaves ont des registres internes pour diverses fonctions. Le **tableau 1** donne un aperçu des fonctions Modbus.

Il existe trois versions du protocole Modbus :

- Modbus RTU : transmission binaire sur RS485 (EIA485).
- Modbus ASCII : transmission en texte clair sur RS485 (EIA485). C'est moins efficace que le RTU, mais lisible par l'homme. Un simple utilitaire de terminal peut envoyer les commandes.
- Modbus TCP : les commandes Modbus sont transmises par TCP/IP. C'est en général une transmission par Ethernet, mais ici, c'est du sans fil.

Le site web de Modbus [2] donne des informations plus détaillées et décrit les protocoles.

Préparation de la carte Modbus sans fil

Pour utiliser la carte sans fil avec le protocole Modbus, il faut d'abord charger le micrologiciel requis. Pour cela, retirez le module NodeMCU de la carte Modbus et connectez-le à un port USB de l'ordinateur. Ensuite ouvrez l'EDI Arduino, configuré comme décrit dans la 1ère partie de cet article. Téléchargez le micrologiciel qui transforme le module en client Modbus, depuis la page du projet Elektor [3].
Ouvrez le fichier téléchargé *OpenPLC_ESP8266_1_0_MUX_V1_1.ino* dans l'EDI Arduino (*Fichier -> Ouvrir*). L'EDI Arduino demande si le projet doit être déplacé vers le dossier des croquis. Répondez '*Oui*'. Si l'EDI crée un nouveau dossier, copiez le fichier déclaratif (*modbus.h*) dans le nouveau dossier contenant le fichier *.ino*, afin que le compilateur l'y trouve.
Pour « accéder sans fil » au module, remplacez les données d'accès du début du code source (voir **listage 1)** par les données d'accès du réseau sans fil utilisé.
Une fois cette modification effectuée, on peut charger le programme dans la carte Modbus (voir la 1ère partie). La carte fonctionne avec

Tableau 1. Aperçu des fonctions Modbus.

Description	Mode	Bits
Entrée/sortie individuelle « Bobine »	Lecture/écriture	1
« Entrées discrètes » individuelles	Lecture seulement	1
« Registres d'entrée » (analogique/numérique)	Lecture uniquement	16
Entrées/sorties « registres de stockage » (analogiques/numériques)	Lecture/écriture	16

DHCP, ce qui signifie que le routeur lui attribuera automatiquement une adresse IP. C'est vérifiable dans le micrologiciel du routeur, ou bien en observant la sortie du programme sur le moniteur série de l'EDI Arduino (à ouvrir avec l'icône de la loupe en haut à droite). Dans la boîte déroulante en bas à droite, réglez le débit de données de l'interface sur 115 200 bauds. Le **listage 2** montre une façon de sortir l'adresse IP sur le moniteur série. Pour ne pas répéter cette procédure à chaque mise sous tension de la carte Modbus, il faut lui attribuer une adresse IP statique. Tous les routeurs récents le permettent. Dans tous les cas, il faut noter l'adresse IP de la carte fournie par le programme afin de pouvoir y accéder *sans fil*. L'adresse IP n'étant émise que pendant le processus de démarrage, si aucune sortie n'apparaît, appuyez sur le bouton *reset* du module NodeMCU pour redémarrer.

Listage 1. Configuration sans fil de la carte Modbus.

```
/*********NETWORK CONFIGURATION*********/

const char *ssid = "<YOUR_SSID>";
const char *password = "<YOUR_PASSWORD>";

/*****************************************/
```

Listage 2. Sortie de l'adresse IP de la carte Modbus sans fil.

```
Connecting to Vodafone-3980
.......
WiFi connected
Server started
My IP: 192.168.0.85
```

Premier test

Pour vérifier si la carte Modbus sans fil fonctionne correctement, l'outil *EasyModbus* convient. Téléchargez-le depuis SourceForge [4]. EasyModbus fournit un serveur, un client et une bibliothèque. Pour ce test, seul le client est nécessaire. Il peut être téléchargé directement [5]. Pour utiliser cet outil, il faut installer une version de Java sur le PC. Si Java n'est pas encore installé, le logiciel gratuit OpenJDK [6] convient. Il est plébiscité par la communauté des développeurs et des créateurs et c'est un composant standard de toutes les distributions *Linux*. Dans Ubuntu, par exemple, la commande `sudo apt install openjdk-11-jdk` l'installe.

Pour lancer le client EasyModbus, entrez la commande `java -jar EasyModbusJavaClient.jar`. La **figure 1** montre la sortie du client EasyModbus (à gauche sous Linux ; à droite sous Windows). Ici sélectionnez *Modbus TCP* et entrez l'adresse IP de la carte. Laissez l'*adresse de départ* sur *1* et changez le *nombre de valeurs* : *4*. Cliquez ensuite sur *Read Discrete Inputs - FC 2* pour lire et afficher les quatre entrées numériques de la carte. Appliquez un signal à au moins une des entrées afin de voir si tout fonctionne comme prévu.

Le client EasyModbus ne peut que lire des données envoyées par la carte Modbus ; il ne peut pas en écrire sur le « bus ». Pour tester l'écriture, utilisez le programme d'exemple décrit ci-dessous. Les données effectivement transmises peuvent être vues en bas de la

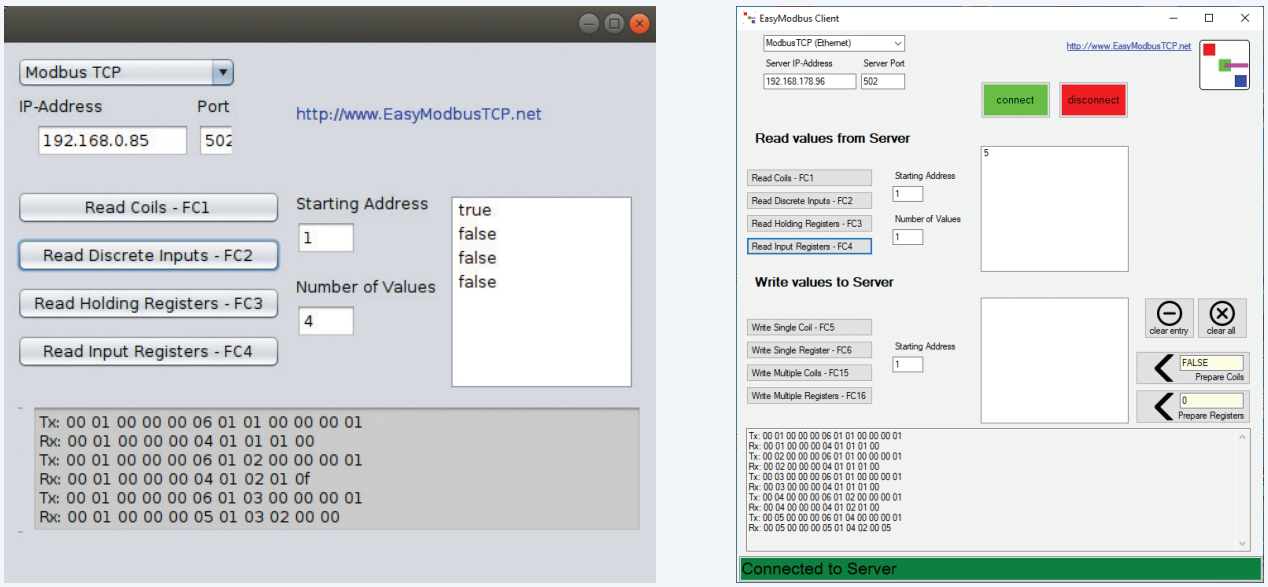


Figure 1. Le client EasyModbus en train de lire les données de la carte. La version Linux est à gauche, et la version Windows à droite.

Listage 3. Exemple de programme tor.py pour piloter la porte.

```
from pyModbusTCP.client import ModbusClient
client = ModbusClient(host="192.168.0.85",
   port=502, auto_open=True, debug=False)
while(True):
    inputs=client.read_discrete_inputs(0,4)
    end_switch_top  = inputs[0]
    end_switch_bottom = inputs[1]
    push_button_down     = inputs[2]
    push_button_up       = inputs[3]
    motor_up        = 0
    motor_down      = 1
    # request end_switch
    if(end_switch_top):
        client.write_single_coil(motor_up,False)
        print("gate open")
    if(end_switch_bottom):
        client.write_single_coil(motor_down,False)
        print("gate closed")
    # request push button
    if (push_button_up and not end_switch_top):
        client.write_single_coil(motor_up,True)
    if (push_button_down and not
            end-switch_bottom):
        client.write_single_coil(motor_down,True)
```

fenêtre du client EasyModbus. Cette fonction est très utile pour le débogage. Si le premier test est réussi, passez à la programmation de la carte en Python.

Installation de la bibliothèque Modbus

Pour rendre le test plus réaliste, nous utilisons ici la maquette de porte d'ascenseur de la 1ère partie [7]. Nous avons écrit un programme pour le PC qui scrute les positions des boutons et capteurs de fin de course par Modbus pour transmettre aussi les actions idoines aux moteurs par Modbus.

Notre programme Python a besoin d'une bibliothèque [8] afin d'accéder au Modbus. Cela n'a rien d'inhabituel ; il existe des bibliothèques Modbus pour presque tous les langages de programmation. Au besoin, on peut créer soi-même les séquences de bits des commandes et les envoyer sur le réseau, mais ce n'est pas notre propos ici.

D'abord, assurez-vous que l'installateur de paquets Python *pip* est présent, ce qui doit toujours être le cas avec des installations Python assez récentes. Pour le vérifier, utilisez `$ python -m pip -version` sous Linux ou `C:\> py -m pip -version` sous Windows.

Si l'installeur *pip* n'est pas présent, installez-le via Python avec `get-pip.py` [9] (ici n'utilisez pas le gestionnaire de paquets Linux) :

```
wget https://bootstrap.pypa.io/get-pip.py
python get-pip.py
get-pip.py
```

Ensuite, installez la bibliothèque Modbus :

```
$ python -m pip install -U pymodbusTCP
C:\> py -m pip install -U pymodbusTCP
```

Une documentation complète sur la bibliothèque et de nombreux exemples de code sont disponibles [10].

Exemple de programme

Examinons en détail le programme d'exemple du **listage 3**. La 1ère ligne importe la bibliothèque Modbus. La suivante crée un nouvel objet de connexion afin de communiquer avec un appareil Modbus spécifique. À cet effet, on peut passer plusieurs paramètres. Dans notre cas, nous passons l'adresse IP de l'appareil cible comme `hôte`. *502* est le port par défaut pour la communication Modbus TCP. Le paramètre `auto_open` détermine si la connexion doit être établie automatiquement ou manuellement. Dans notre exemple, nous avons choisi `True` pour des raisons de commodité. Pour avoir un contrôle total de la configuration de la connexion, choisissez `False`. Avec la configuration manuelle de la connexion, il y a plus d'options de contrôle et de gestion des erreurs. Si le paramètre `debug` est défini sur `True`, le logiciel affiche sur la console toutes les données transmises. C'est très utile pour le débogage.

La boucle `while`, permet l'exécution répétée du programme décrit dans la 1ère partie. Ainsi le programme tourne cycliquement, comme dans tout contrôleur industriel. La partie principale du programme est le code interne de la boucle qui lit d'abord les entrées et les convertit ensuite en variables pertinentes. Des variables pour les deux sorties numériques sont aussi définies. Cela facilite la lecture des lignes de programme suivantes.

On voit que le moteur s'arrête lorsque les fins de course sont atteintes. Un message de type texte est envoyé à la console pour que la position en cours de la porte s'affiche sur l'ordinateur. Enfin, le moteur entraîne la porte dans le sens indiqué par les boutons Haut et Bas. Le ET logique entre les capteurs de fin de course empêche d'activer les relais si la porte se trouve déjà en fin de course.

Pour lancer le programme, entrez la ligne de commande `python tor.py`. Comme d'habitude, *Ctrl-C* arrête le programme.

AdvancedHMI

Avec divers *frameworks* pour PC, il est possible de réaliser une télécommande pratique avec retour visuel, adaptable à vos besoins. Le programme *AdvancedHMI* est un logiciel libre permettant de créer des interfaces homme-machine (*IHM*) qui communiquent avec un automate ou d'autres dispositifs d'E/S. Ce logiciel diffère des autres progiciels standard dans la mesure où il permet de produire des fichiers exécutables au lieu de simples configurations interprétées par un moteur d'exécution. Il en résulte des applications très rapides et efficaces.

AdvancedHMI [11] est basé sur le *framework* .NET de Microsoft. Les applications sont générées dans *Microsoft Visual Studio Community 2019*, qui est disponible gratuitement. Cela permet de créer des IHM rudimentaires par glisser-déposer sans avoir à écrire le moindre code. Un grand nombre de développeurs utilise le *framework* .NET qui dispose de nombreux réseaux d'assistance. AdvancedHMI assure une assistance technique bien meilleure que tous les autres progiciels IHM standard réunis.

L'auteur a utilisé cette plateforme pour créer une interface utilisateur permettant de contrôler la carte industrielle. Elle indique l'état des entrées et fournit des boutons utilisables pour commuter les sorties (**fig. 2**). Les programmes .NET, tels que cette interface utilisateur pour contrôler la carte industrielle, peuvent également fonctionner sous Linux avec *Mono*. Elle a été testée avec succès sur un *Raspberry Pi*.

Prenez les commandes…

La carte Modbus sans fil permet de piloter facilement des appareils conformes au protocole Modbus à partir d'un PC ou d'un smartphone. Ce projet a une bonne valeur didactique pour ceux qui souhaitent se familiariser avec le protocole Modbus. Cependant, la conception matérielle est si robuste et tolérante aux erreurs qu'elle peut aussi s'utiliser dans les projets ESP8266 personnels pour commander et scruter des appareils industriels p. ex. des électrovannes, moteurs et autres types de capteurs ou actionneurs.

200507-B-04

Figure 2. Le logiciel AdvancedHMI produit des applications d'interface homme-machine.

Des questions, des commentaires ?

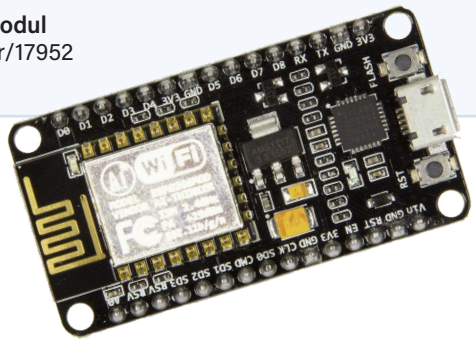
Envoyez un courriel à l'auteur (josef@bernhardt.de) ou contactez Elektor (redaction@elektor.fr).

Contributeurs

Idée, conception et texte : **Josef Bernhardt** et **Martin Mohr**
Rédaction : **Rolf Gerstendorf**
Mise en page : **Giel Dols**
Traduction : **Yves Georges**

PRODUIT

› **NodeMCU-Modul**
www.elektor.fr/17952



LIENS

[1] Modbus sans fil (partie 1), Elektor, 09-10/2021 : http://www.elektormagazine.fr/200507-04
[2] Site web Modbus : https://modbus.org/
[3] Page du projet Elektor : https://www.elektormagazine.fr/200507-B-04
[4] Site web EasyModbus : https://bit.ly/2QDIDJz
[5] Téléchargement du client EasyModbus : https://bit.ly/3d2zzFp
[6] Téléchargement d'OpenJDK : https://bit.ly/2OVK4m6
[7] Maquette de porte d'ascenseur sur YouTube : https://youtu.be/VHIBQswdA0E
[8] Bibliothèque Modbus pour Python : https://pypi.org/project/pyModbusTCP/#description
[9] get-pip.py : https://bootstrap.pypa.io/get-pip.py
[10] Exemples de bibliothèque Modbus : https://pymodbustcp.readthedocs.io/en/latest/
[11] AdvancedHMI : https://www.advancedhmi.com/index.php?main_page=page&id=14&chapter=0
[12] Plus d'infos par l'auteur : https://bit.ly/3ch1PFl