

Bras Robot éducatif- Arduino, ROS2

- [Etat de l'art Bras robot low-cost](#)
- [SO-ARM100 - Robotique éducative](#)
- [ROS2 avec SO-ARM101](#)
- [Pilotage des servomoteurs : TTL, RS232, RS485](#)
- [Transmission TTL et protocole RS485](#)
- [Introduction à Modbus](#)

Etat de l'art Bras robot low-cost

Modèles commerciaux fermés

Niryo Ned 2

- 6DOF + Pince

<https://niryo.com/fr/produit/bras-robotise-6-axes/>

<https://github.com/NiryoRobotics>

DAGU Six-servo Robot Arm



- 5DOF Manipulateur + 1DOF Pince
- 6 servos
 - 3x 13 kg.cm torque metal gear, 40.4 * 19.8 * 36 mm, 48g, 0.22s/60°

- 1x 3.2 kg.cm, 39.5 x20.0x35.5mm, 41g, 0.27s/60°
- 2x 2.3 kg.cm, 28 x14x29.8mm, 18g, 0.13/60°
- Carte de contrôle AREXX Intelligence Centre

<https://seafire.unistra.fr/d/693101e6046d4819a3af/>

<https://arexx.com/product/robot-arm/>

www.arexx.com.cn

Modèles Open Source

<https://github.com/ AntoBrandi/Robotics-and-ROS-2-Learn-by-Doing-Manipulators>

Trossen Robotics ALOHA

Stationary

https://docs.trossenrobotics.com/trossen_arm/main/specifications.html

https://docs.trossenrobotics.com/aloha_docs/2.0/specifications.html#aloha-stationary

https://docs.trossenrobotics.com/aloha_docs/2.0/operation/stationary.html

Solo

Dimensions	1019D x 1066H x 1225W mm
Leader Arms	WidowX 250 S - Aloha Version
Follower Arms	ViperX 300 S - Aloha Version
Camera	2x Intel RealSense D405
Chassis	Modular
Computer	Coming Soon
USB Hubs	Yes 1X

https://docs.trossenrobotics.com/aloha_docs/2.0/specifications.html#aloha-solo

https://docs.trossenrobotics.com/aloha_docs/2.0/operation/solo.html

Trossen Robotics (Interbotix) X-Series Arms

https://docs.trossenrobotics.com/interbotix_xsarms_docs/specifications.html

ALOHA WidowX-250 6DOF

ALOHA ViperX-300 6DOF

Waveshare RoArm

- 5DOF + pince Waveshare
- https://github.com/waveshareteam/roarm_ws
- <https://www.waveshare.com/product/roarm-m3.htm?sku=30444>

ROBOTIS OMX

Follower:

- 5DOF + pince
- https://ai.robotis.com/omx/introduction_omx.html

Leader:

ROBOTIS Open Manipulator-P

- 5DOF + pince
- Modbus-RTU
- https://emmanual.robotis.com/docs/en/platform/openmanipulator_p/overview/

ROBOTIS Open Manipulator-X

https://emmanual.robotis.com/docs/en/platform/openmanipulator_x/specification/#specification

- 4 DOF Manipulateur + 1 DOF Pince
- 6x Dynamixel XM430-W350 <https://emmanual.robotis.com/docs/en/dxl/x/xm430-w350/>
- Carte de contrôle Robotis OpenCR1.0
<https://emmanual.robotis.com/docs/en/parts/controller/opencr10/>

SO-ARM100

<https://github.com/TheRobotStudio/SO-ARM100>

- 5 DOF Manipulateur + 1 DOF Pince
- 6 servos Feetech STS3215 https://www.feetechrc.com/en/2020-05-13_56655.html
- Waveshare Serial Bus Servo Driver Board
[https://www.waveshare.com/wiki/Bus_Servo_Adapter_\(A\)](https://www.waveshare.com/wiki/Bus_Servo_Adapter_(A))
- OU
- Feetech FE-URT-1 <https://www.feetechrc.com/FE-URT1-C001.html>

https://github.com/huggingface/lerobot/blob/main/examples/10_use_so100.md

<https://medium.com/@sarohapranav/my-experiences-and-tips-for-creating-a-robotic-so100-arm-3df779a4aae7>

https://github.com/JafarAbdi/ros2_so_arm100

pince compatible SO-ARM

- Waveshare <https://www.waveshare.com/gripper-a.htm?sku=30386>

Poppy Ergo JR

6DOF mais architecture semble optimisée pour faible le coût, qui n'a plus trop de sens avec le coût des servos Feetech. Une architecture 5DOF ou "typique industrielle" de type épaule poignet semble plus intéressante.

<https://www.poppy-education.org/robots/poppy-ergo-jr/>

Cartes de contrôle

OpenCR1.0

<https://emmanual.robotis.com/docs/en/parts/controller/opencr10/>

- STM32F746ZGT6 / 32-bit ARM Cortex®-M7 with FPU (216MHz, 462DMIPS)
[Reference Manual](#), [Datasheet](#)
- Programmer : ARM Cortex 10pin JTAG/SWD connector
USB Device Firmware Upgrade (DFU)
Serial
- Digital I/O
 - 32 pins (L 14, R 18) *Arduino connectivity
 - 5Pin OLLO x 4
 - GPIO x 18 pins

- PWM x 6
- I2C x 1
- SPI x 1
- Communication Ports
 - USB x 1 (Micro-B USB connector/USB 2.0/Host/Peripheral/OTG)
 - TTL x 3 (B3B-EH-A / DYNAMIXEL)
 - RS485 x 3 (B4B-EH-A / DYNAMIXEL)
 - UART x 2 (20010WS-04)
 - CAN x 1 (20010WS-04)

Waveshare Serial Bus Servo Driver Board

[https://www.waveshare.com/wiki/Bus_Servo_Adapter_\(A\)](https://www.waveshare.com/wiki/Bus_Servo_Adapter_(A))

- Supports connecting to a host or MCU
- up to 253 ST/SC series serial bus servos
- RS485
- UART pour contrôle depuis Arduino, ESP32, STM32 (RX-RX, TX-TX)
- USB pour contrôle via Raspberry, Jetson ou PC
- 9~12.6V voltage input (the input voltage and the servo voltage must be matched)

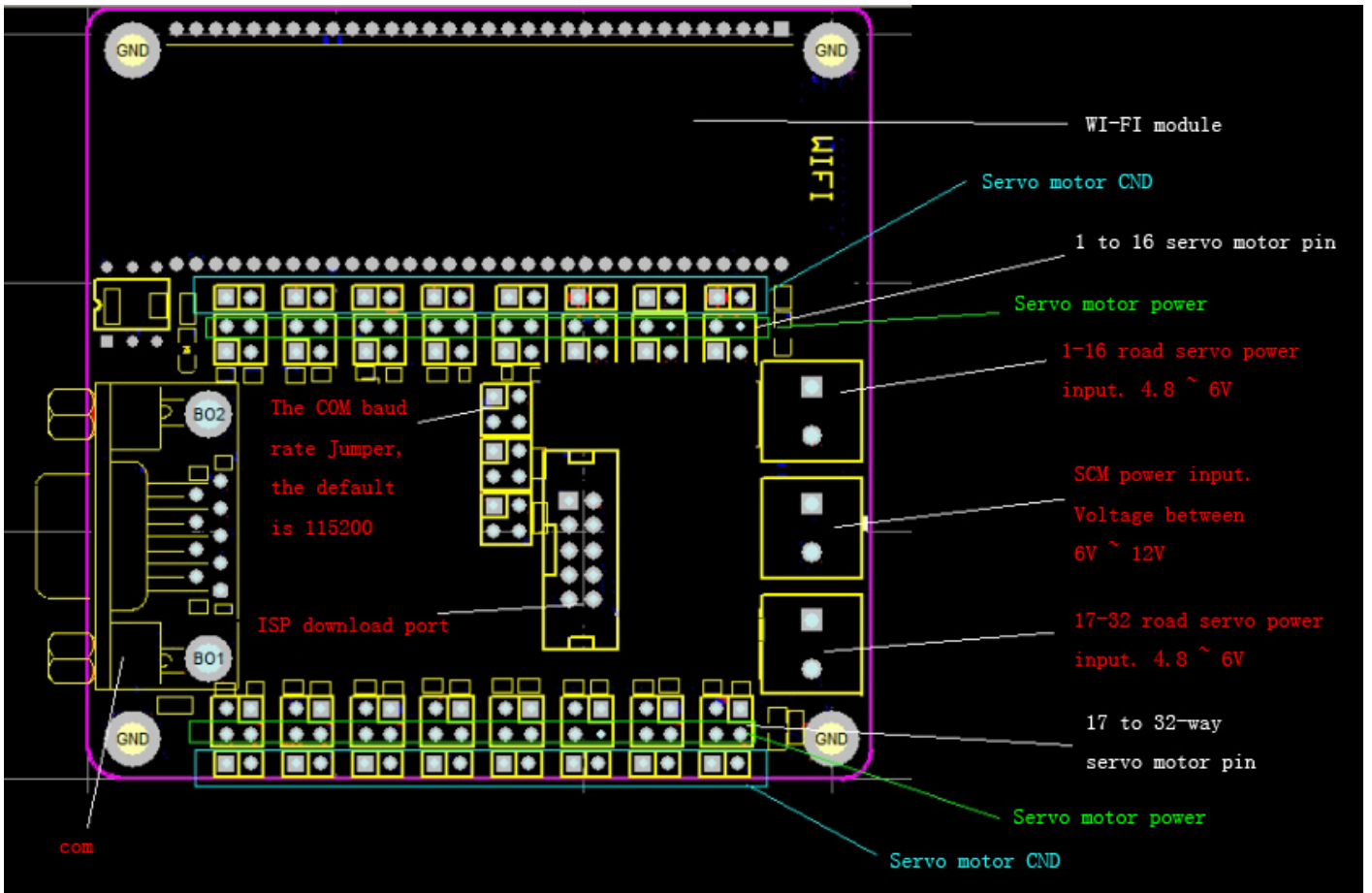
Feetech FE-URT-1

<https://www.feetechrc.com/FE-URT1-C001.html>

AREXX Intelligence Centre

<https://seafire.unistra.fr/d/693101e6046d4819a3af/>

- atmega168 MCU
- RS232
- default baud rate is 115.2k
- Wifi wireless control reserve the ISP downloaded, you can download the MCU controller program using the STK500 ISP cable



- dual - Power Supply
 - 6 ~ 12 V SCM power
 - 4.8 ~ 6 V, 1.2A servo motor power [servo motor power supply Road 1-16 respectively, a 17-32 road supply port])

Servomoteurs

Dynamixel XM430-W350

<https://emanual.robotis.com/docs/en/dxl/x/xm430-w350/>

- 4.1 [N.m] (at 12.0 [V], 2.3 [A])
- 46 [rev/min] (at 12.0 [V])
- 10.0 ~ 14.8 [V]
- Operating Modes
 - Current Control Mode
 - Velocity Control Mode
 - Position Control Mode (0 ~ 360 [°])
 - Extended Position Control Mode (Multi-turn)
 - Current-based Position Control Mode
 - PWM Control Mode (Voltage Control Mode)
- baud rate 9,600 [bps] ~ 4.5 [Mbps]
- TTL Half Duplex Asynchronous Serial Communication with 8bit, 1stop, No Parity
- RS485 Asynchronous Serial Communication with 8bit, 1stop, No Parity

Feetech STS3215

https://www.feetechrc.com/en/2020-05-13_56655.html

SO-ARM100 - Robotique éducative

Les bases d'un bras robot

<https://docs.phospho.ai/learn/overview>

Pilotage bras robot avec Scratch

<https://www.poppy-education.org/activites/initiation-ergo-jr-et-scratch/>

Pilotage du SO-ARM100 avec Phospho

<https://docs.phospho.ai/learn/overview>

Simulation et pilotage du SO-ARM100 avec ROS2

Attention avant d'utiliser le robot avec ROS2, il faut avoir calibré les servomoteurs, par ex. avec le script de calibration du projet LeRobot

https://github.com/JafarAbdi/ros2_so_arm100

<https://discourse.openrobotics.org/t/interactive-so-101-ik-in-ros-2-with-viser-robokin/53850>

Jumeau numérique

Pilotage de la simulation ou du vrai robot

<https://github.com/tessel-la/robo-boy>

Adapter le tuto suivant au SO-ARM100 : <https://innovation.iha.unistra.fr/books/robotique-open-source/page/programmer-un-robot-avec-moveit2-jumeau-numerique>

Contrôle des moteurs par un GUI de "jogging"

Joint Trajectory Controller

```
ros2 run rqt_joint_trajectory_controller rqt_joint_trajectory_controller
```

<https://github.com/tessel-la/robo-boy>

Contrôle de l'outil sans collisions via le plugin Moveit de RViz

Cartesian Trajectory

```
ros2 launch so_arm100_moveit_config moveit_rviz.launch.py
```

Réalisation d'un programme en Python

https://moveit.picknik.ai/main/doc/examples/motion_planning_python_api/motion_planning_python_api_tutorial.html#single-pipeline-planning-pose-goal

```
# set plan start state to current state
panda_arm.set_start_state_to_current_state()

# set pose goal with PoseStamped message
pose_goal = PoseStamped()
pose_goal.header.frame_id = "panda_link0"
pose_goal.pose.orientation.w = 1.0
pose_goal.pose.position.x = 0.28
pose_goal.pose.position.y = -0.2
pose_goal.pose.position.z = 0.5
panda_arm.set_goal_state(pose_stamped_msg=pose_goal, pose_link="panda_link8")

# plan to goal
plan_and_execute(panda, panda_arm, logger)
```

En utilisant l'environnement de développement Jupyter Notebook

https://moveit.picknik.ai/main/doc/examples/jupyter_notebook_prototyping/jupyter_notebook_prototyping_tutorial.html

Pilotage du bras robot par LeRobot (IA, VR, etc.)

Environnement Python sous Windows ou Linux

Contrôle du bras par clavier ou manette

Avec LeRobot+Phospho <https://docs.phospho.ai/basic-usage/teleop>

ou avec ROS2+MoveIt2

https://moveit.picknik.ai/main/doc/examples/jupyter_notebook_prototyping/jupyter_notebook_prototyping_tutorial.html

Contrôle du bras par Oculus Quest

Compatible LeRobot (Windows et Ubuntu) :

- alternative gratuite et open source à phospho
- <https://github.com/vladfatu/telerobot>

Depuis Windows :

- Appli Oculus Phospho <https://docs.phospho.ai/examples/teleop>
- 222€ <https://www.meta.com/en-gb/experiences/phospho-teleoperation/8873978782723478/>

Depuis Ubuntu avec ROS2 et moveit_servo :

- https://github.com/ZorAttC/franka_vr
- https://moveit.picknik.ai/main/doc/examples/realtime_servo/realtime_servo_tutorial.html
- https://github.com/rail-berkeley/oculus_reader
 - Enable Oculus Quest development mode
 - Always allow USB debugging from this computer
 - Connexion USB (ADB) ou wifi

Autre : <https://github.com/lts0429/teleoperation>

Contrôle du bras via un modèle d'IA

<https://docs.phospho.ai/basic-usage/inference>

- Créer un compte huggingface.ia
- Sign In dans phosphobot
- Dans les paramètres, ajouter la clé d'API huggingface
- Par défaut l'inférence du modèle d'IA qui permet de piloter le robot depuis l'image des caméras tournera sur un GPU sur les serveurs de huggingface ou phospho
- On peut faire tourner l'inférence du modèle d'IA sur le PC local s'il a une bonne carte graphique NVidia
- Suivre ces instructions : <https://github.com/phospho-app/phosphobot/tree/main/inference#setup-a-server>
- Démarrer le serveur d'inférence uv
- ```
uv run ACT/server.py --model_id=<CHEMIN_VERS_LE_MODELE_LOCAL>
```
- Appeler le serveur d'inférence depuis un script python : <https://docs.phospho.ai/basic-usage/inference#2-call-your-inference-server-from-a-python-script>

## Pilotage ST3215 depuis un ESP32 (embarqué, micro-ROS)

Sans utiliser la carte de contrôle Feetech/Waveshare

(5€) : [https://github.com/sepastian/ESP32\\_ST3215](https://github.com/sepastian/ESP32_ST3215)

## Pilotage bluetooth depuis un smartphone

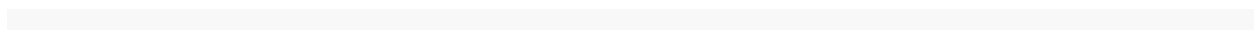
Utiliser l'ESP32 pour faire l'interface bluetooth vers une télécommande smartphone ? Avec ou sans carte de contrôle officielle (cf. ci-dessus) ?

## Micro-ROS avec bras robot

- approche pour grasping référencée capteur via un TOF-sensor

[https://micro.ros.org/docs/tutorials/demos/openmanipulator\\_demo/](https://micro.ros.org/docs/tutorials/demos/openmanipulator_demo/)

- Télécommander une pose relative de la pince via un capteur  
[https://micro.ros.org/docs/tutorials/demos/moveit2\\_demo/](https://micro.ros.org/docs/tutorials/demos/moveit2_demo/)



6-DoF Inertial Measurement Unit (LSM6DSL), composed of an accelerometer and a gyroscope, and a 3-DoF magnetometer (LIS3MDL). The fusion of the measurements fetched by these sensors outputs the pose, or relative orientation of the board with respect to a fixed reference frame.

# ROS2 avec SO-ARM101

## ROS2 et MoveIt2

Installer les paquets ROS2 du SO-ARM100 :

- Cloner le paquet dans un workspace ROS2 [https://github.com/JafarAbdi/ros2\\_so\\_arm100](https://github.com/JafarAbdi/ros2_so_arm100)
- Cloner le submodule <https://github.com/TheRobotStudio/SO-ARM100> dans `so_arm100_description/SO-ARM100` (<https://www.freecodecamp.org/news/how-to-use-git-submodules/>)
- Ou simplement :

```
mkdir -p ~/ws_so_arm100/src
cd ~/ws_so_arm100/src
git clone --recurse-submodules https://github.com/JafarAbdi/ros2_so_arm100
cd ~/ws_so_arm100
sudo rosdep init
rosdep update && rosdep install --ignore-src --from-paths src -y
colcon build --symlink-install # dans une VM ajouter --parallel-workers 1
source install/setup.bash
ros2 launch so_arm100_moveit_config demo.launch.py hardware_type:=mock_components #
hardware_type:=real for running with hardware
```

Tester la démo en simulation :

- Lancer un des scripts : [https://github.com/JafarAbdi/ros2\\_so\\_arm100?tab=readme-ov-file#usage](https://github.com/JafarAbdi/ros2_so_arm100?tab=readme-ov-file#usage)

# Pilotage des servomoteurs : TTL, RS232, RS485

## Modes de contrôle des servomoteurs

Regarder la classification des constructeurs permet de se rendre compte des différentes manières de piloter un servomoteur :

- Feetech <https://www.feetechrc.com/>
- Robotis :
  - [https://www.robotis.fr/index.php?id\\_category=7&controller=category](https://www.robotis.fr/index.php?id_category=7&controller=category)
  - <https://emanual.robotis.com/docs/en/dxl/>
  - <https://www.dynamixel.com/>
  - <https://www.dynamixel.com/whatisdxl.php>

Cela va donc du contrôle PWM jusqu'aux bus et protocoles industriels :

- Servos de modélisme asservis en position "servo 180°" ou en vitesse "servo 360°" via signal PWM
  - Feetech "PWM series servo"  
<https://www.feetechrc.com/pwm%20series%20servo.html>
  - <https://arduino.blaisepascal.fr/conversion-numeriqueanalogique-pwm/>
  - <https://arduino.blaisepascal.fr/communication-2/>
  - <https://arduino.blaisepascal.fr/premiers-pas/faire-tourner-les-servos-2/>
  - <https://arduino.blaisepascal.fr/servo-suiveur/>
  - <https://arduino.blaisepascal.fr/les-servomoteurs/>
  - <https://arduino.blaisepascal.fr/controle-dun-servomoteur/>
- Servos pédagogiques Dynamixel "série X" ou Feetech "Smart Serial Bus Servo"
  - TTL, ex. Feetech STS3235
  - RS485, ex. Feetech SMS..
- Servos professionnels Dynamixel "série P" ou Feetech "Modbus RTU Series Servo", par ex.
  - Modbus RTU <https://celka.fr/ocw/plc-control/modbus/intro-modbus/intro/>
  - Modbus TCP <https://celka.fr/ocw/plc-control/modbus/modbus-tcp/modbus-tcp/>

# Introduction au contrôle PLC

<https://celka.fr/ocw/plc-control/modbus/intro-modbus/intro/>

## Protocoles de communication

Dynamixel :

- Dynamixel Protocol 2.0 <https://emanual.robotis.com/docs/en/dxl/protocol2/>
- Modbus RTU pour les Dynamixel Pro (PH, RH, PM)  
<https://emanual.robotis.com/docs/en/dxl/p/ph42-020-s300-r/#protocol-type13>

Feetech :

- Modbus RTU pour les modèles : <https://www.feetechrc.com/modbus-rtu%20series%20servo.html>
  - Exemple servo 24V 24kg <https://www.feetechrc.com/24v-24kgcm-modbus-rtu%E8%88%B5%E6%9C%BA.html>

<https://esp32io.com/tutorials/esp32-rs485>

# Transmission TTL et protocole RS485

## Transmission série

### Avantages

- Câble plus fin, plus souple, moins coûteux.
- Connecteur simplifié, meilleur marché, plus vite monté.
- Plus de problème de synchronisation de signaux
  - On ne transmet qu'un seul signal. Seules les horloges doivent être de fréquence très voisine, ce qui n'est pas difficile en électronique.
- Isolation diaphonique.
  - Plus de risque d'interférence entre signaux, il n'y a qu'un seul signal.
- Utilisable sur des longueurs nettement plus importantes (km).

### Inconvénients

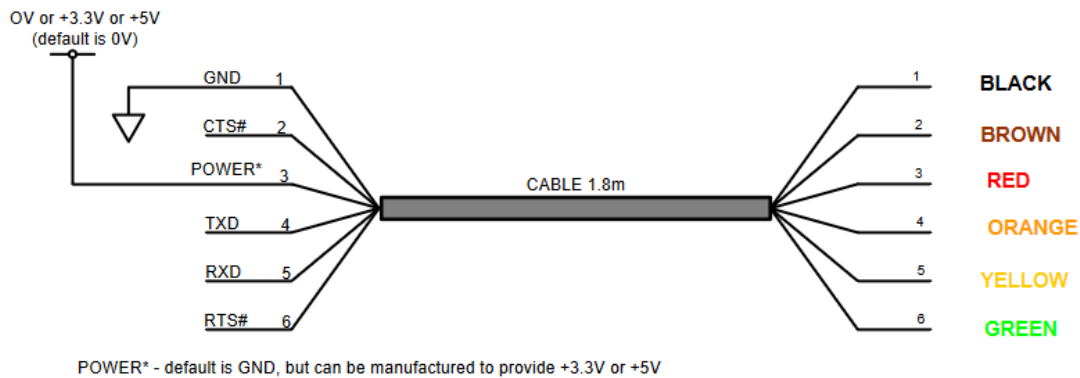
- Débit
  - A une même fréquence, on transporte un seul bit à la fois.
- Electronique plus compliquée du côté émetteur et encore plus compliquée côté récepteur (synchronisation d'horloge).
  - UART : Universal Asynchronous Receiver Transmitter.
  - L'UART peut être désynchronisé, l'information reçue est alors invalide.

## Transmission série synchrone

## Transmission série asynchrone

Exemple : port série RS232 du PC

## Exemple d'un adaptateur FTDI USB-RS232

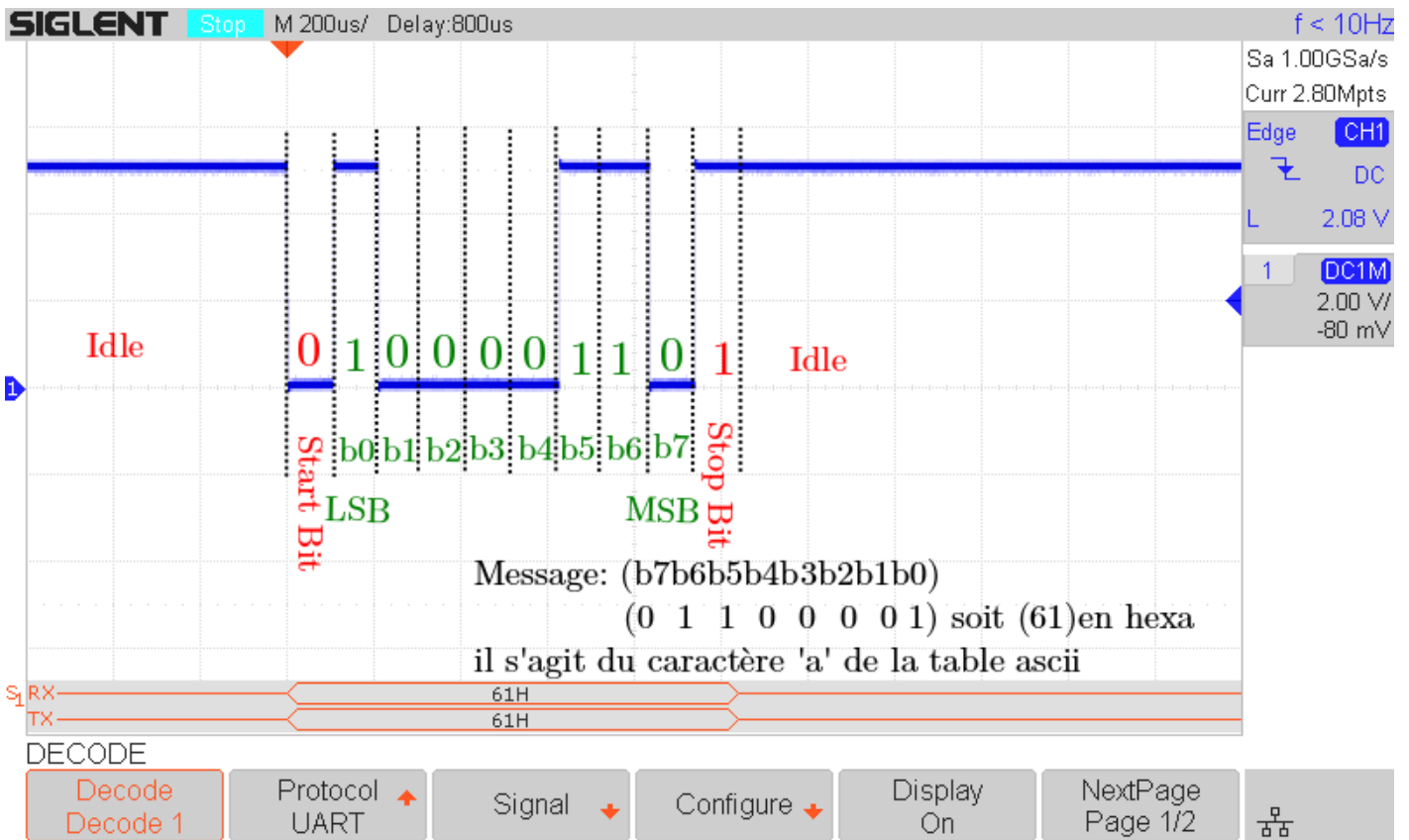


# Transmission série asynchrone TTL

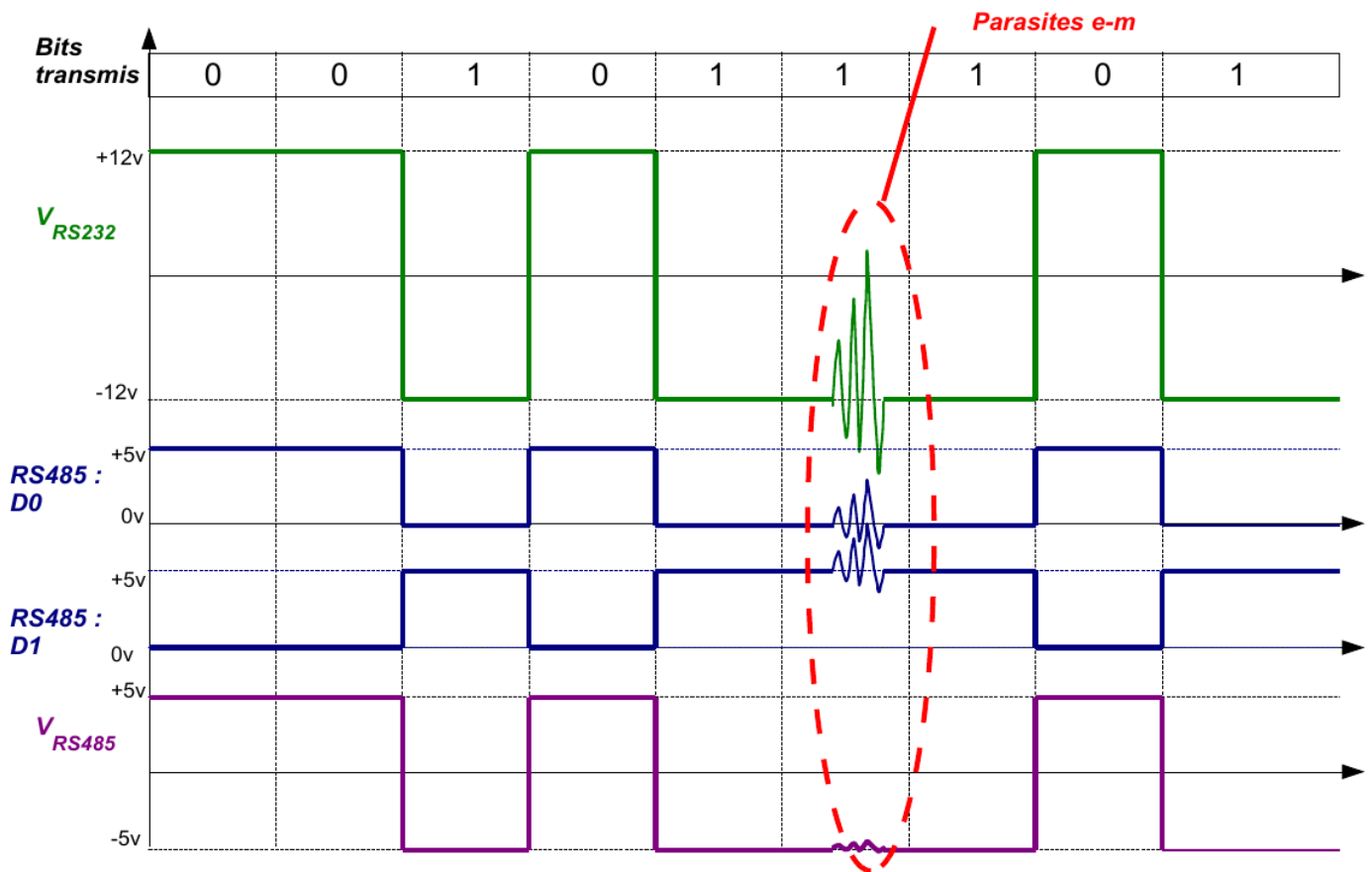
Exemple de trame série (TTL)

'1' logique = +5V

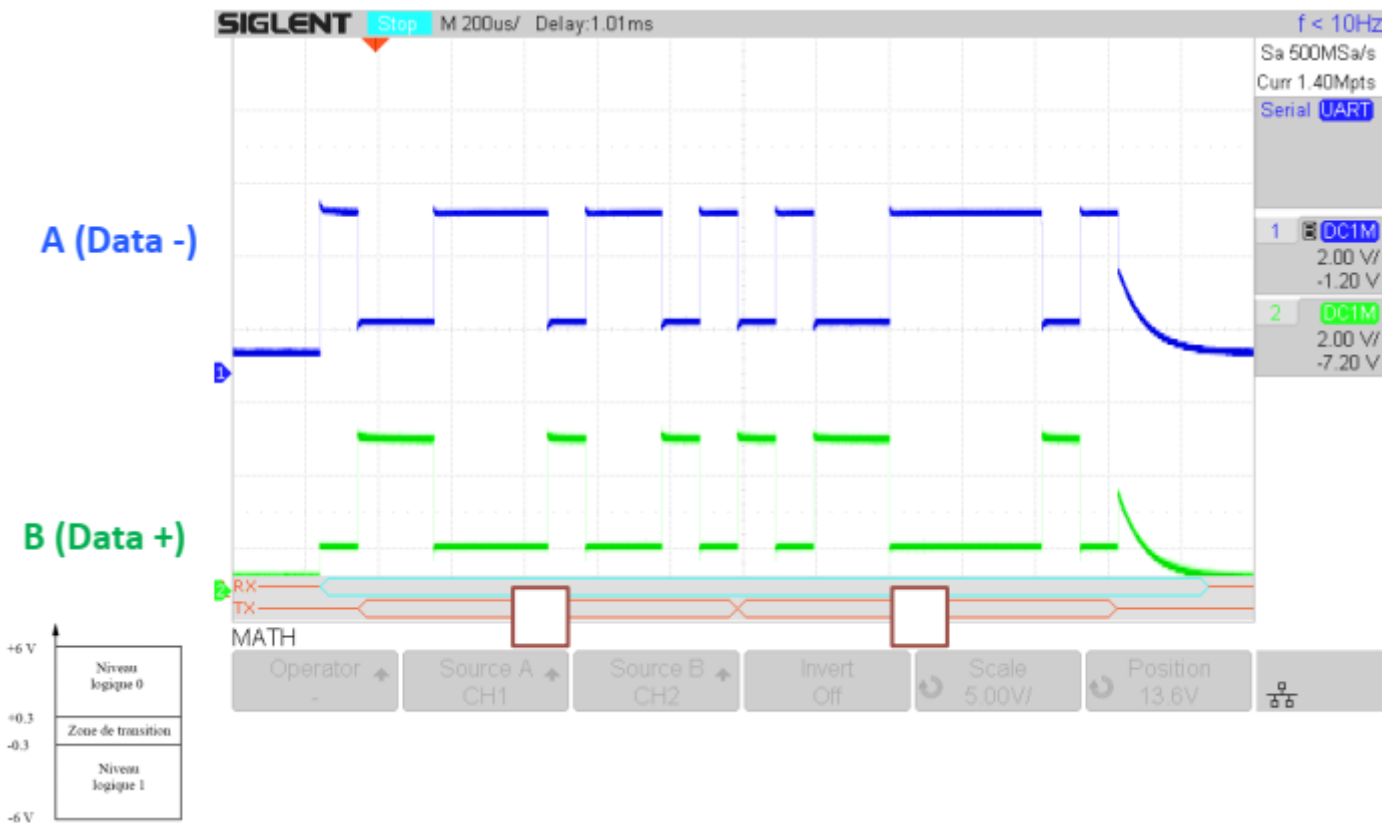
'0' logique = 0V



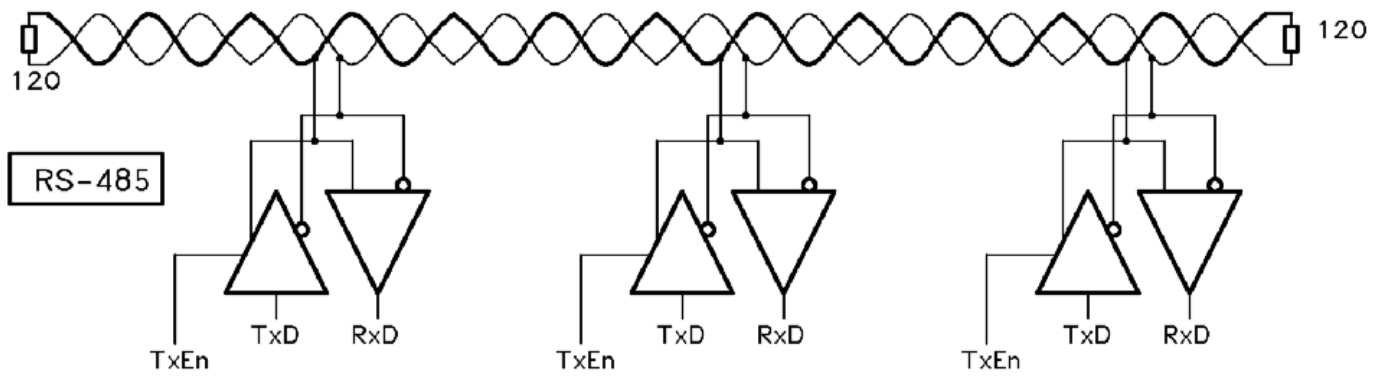
# Transmission série asynchrone RS485 vs RS232



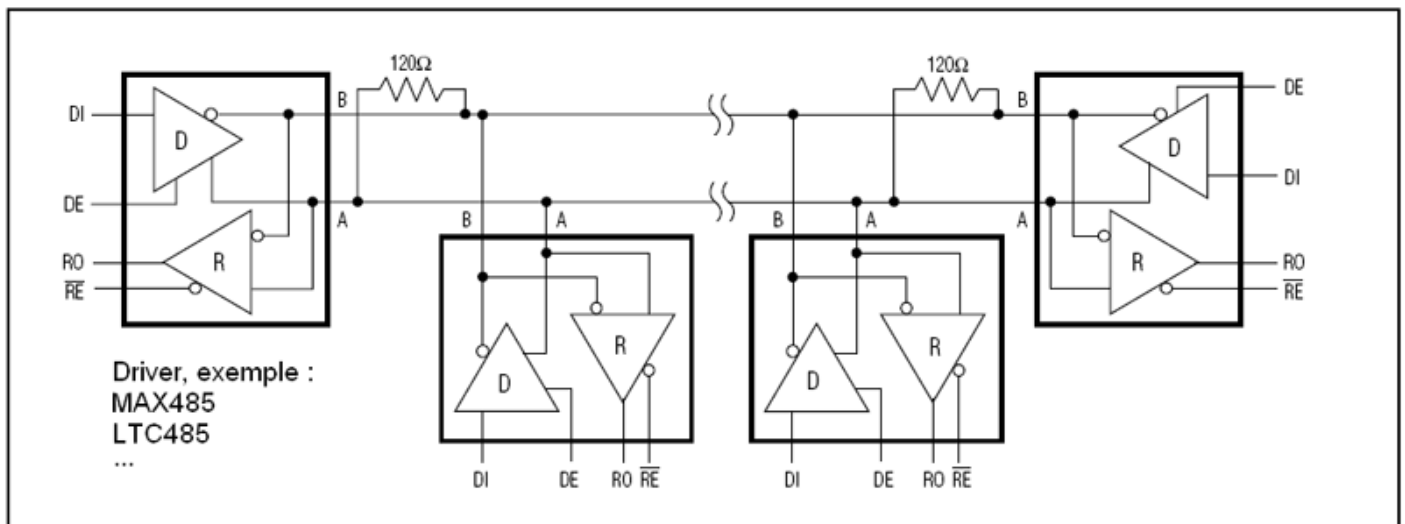
## Transmission série asynchrone RS485

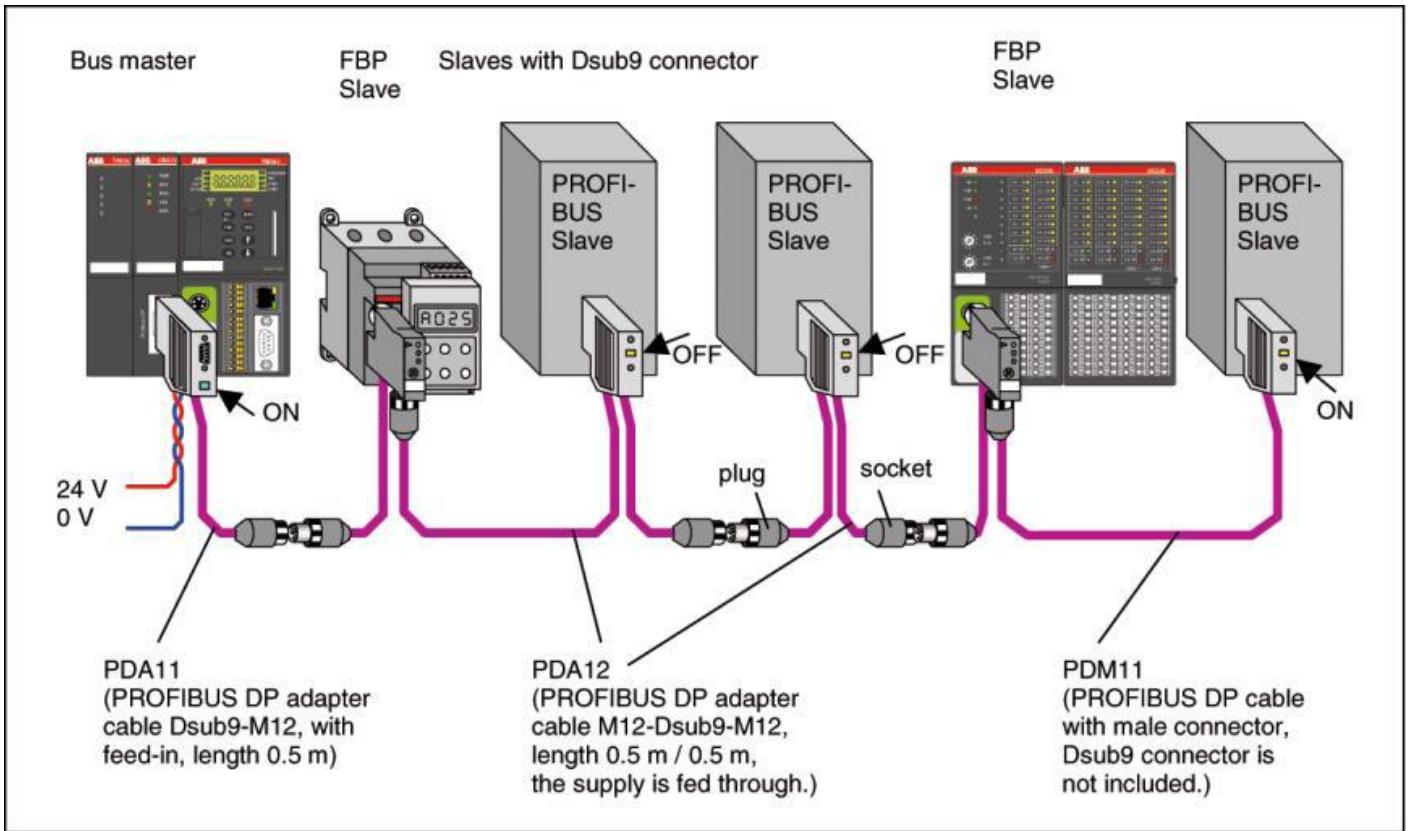


# Liaisons multipoints



Penser aux résistances de Terminaison de  $120\ \Omega$  au début et à la fin de la liaison RS485.





# Half Duplex

## Définition

- Liaison bidirectionnelle.
- 1 canal de transmission est partagé :
  - Il est utilisé dans un sens et dans l'autre.
  - Une règle doit définir comment gérer l'accès au média.
  - Moins cher, plus facile, mais plus lent.

## Exemple

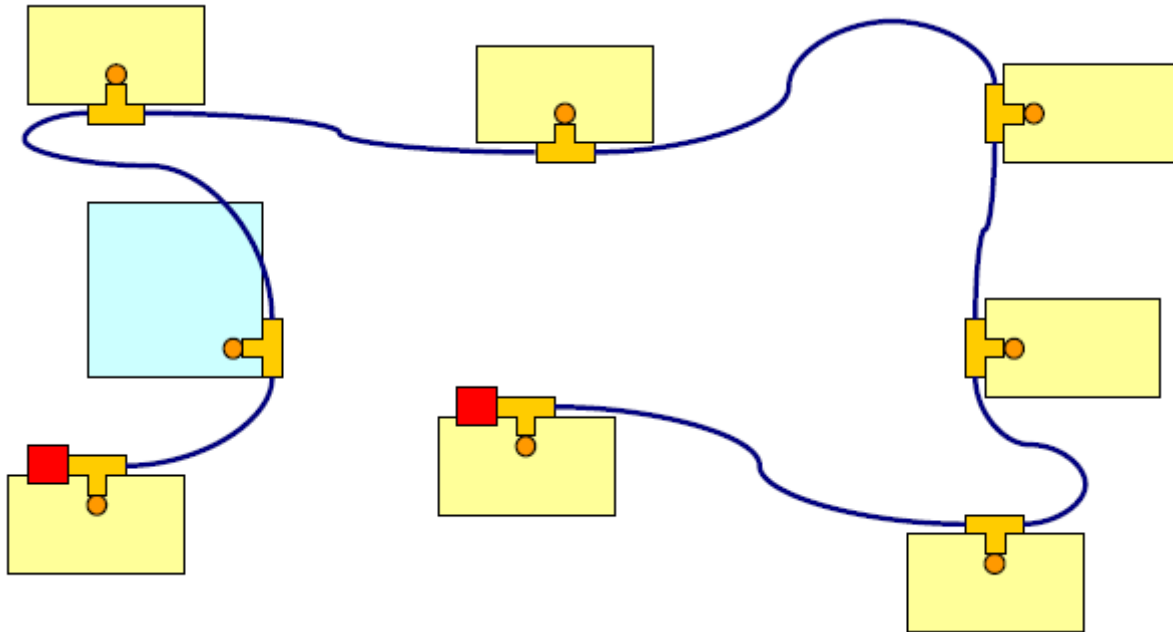
- De nombreux bus de terrain, RS485,



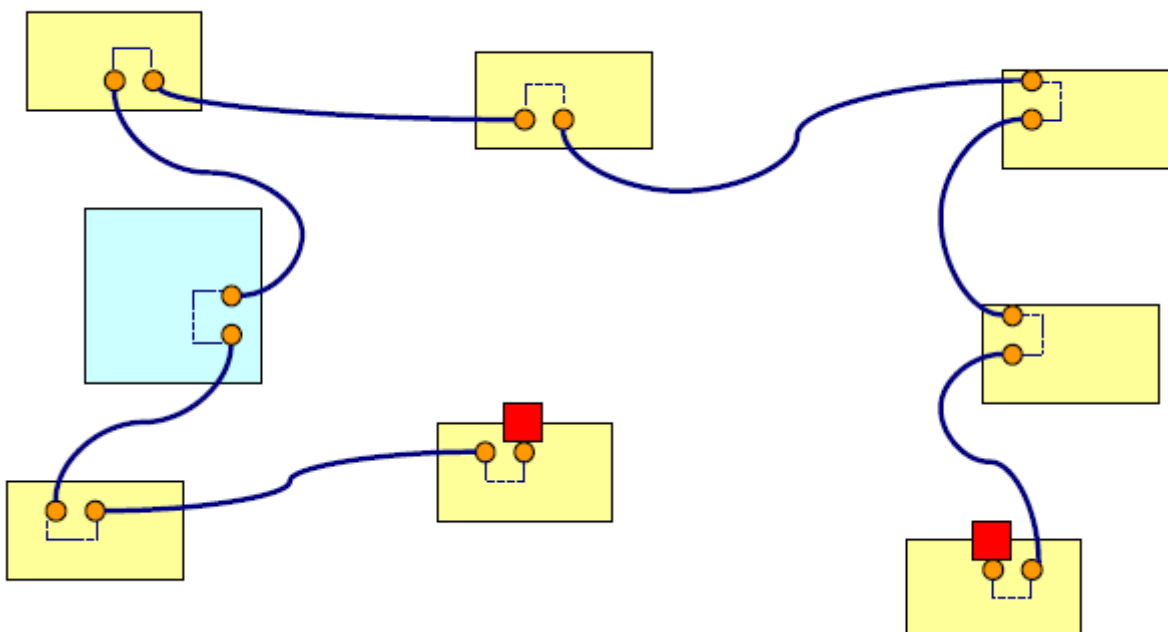
# Topologie : Bus

## Principe

- Connexions de toutes les stations sur un même câble
  - Toujours half duplex.
- 2 topologies selon les possibilités techniques
  - connexion en T "par prise vampire"



- chaînage



## Avantages

- Simplicité d'adjonction de stations.
- Fonctionne même en cas de panne d'une station.
- Transmission en diffusion ( broadcasting , multicasting
- Longueur de câble réduite.

#### Inconvénients

- 1 seule station peut émettre à la fois.
- Les résistances de terminaison sont externes (à câbler).
- Liaison en chaîne : échange d'appareil impossible sans arrêt du système.
- Liaison en T : coûts de connexion plus importants.

#### Exemples

- Connexion en prise vampire : ASi
- Profibus, Modbus

Source : Cours IUT Haguenau - Département GEII - Automatismes Spé. 4 - Réseaux Locaux Industriels - Philippe Celka, le 28.02.2022

Philippe Celka Copyright © 2025 CC Attribution-Non Commercial-Share Alike 4.0 International

# Introduction à Modbus

## Protocole Modbus

Logo Modbus or type unknown

## Introduction

Modbus est un protocole de communication **non propriétaire** créé par Modicon en 1979. Les spécifications du protocole sont données librement sur le site de la [Modbus Organization](#). Ce consortium a été créé par Schneider suite au rachat de Modicon en 1997 pour promouvoir Modbus auprès des fabricants et utilisateurs.

Modbus est très populaire dans les environnements industriels car c'est un protocole simple, facile à intégrer, efficace, fiable, **ouvert** et royalty-free ! Vous pouvez très facilement intégrer Modbus dans vos projets à base d'ESP32, Raspberry, STM32 ...

Le protocole Modbus était à l'origine un protocole sur bus série (Modbus RTU). Il a évolué pour s'intégrer aux technologies TCP/IP quand Ethernet est monté en puissance. On le retrouve dans les domaines de:

- gestion technique des bâtiments
- systèmes de management de l'énergie
- processus complexes d'automatisation industrielle

C'est une couche applicative (niveau 7 OSI) qui se base sur les liaisons séries ou sur les trames Ethernet et les couches TCP/IP.

### Stack de communication Modbus :

Modbus Stack or type unknown

On distingue les différents modes de communication :

- **Modbus TCP** : communication TCP/IP basée sur le modèle client/serveur
- **Modbus RTU** : transmission **série** asynchrone via **RS-485**, RS-232 ou RS-422.
- Modbus ASCII : similaire au protocole RTU, format sur 7 bits (utilisation très rare)

Nous débuterons l'analyse du protocole suivant la chronologie avec l'étude du **Modbus RTU (Remote Terminal Unit)** sur liaison série.

# Modbus RTU

## Principe du protocole Master / Slave utilisé en Modbus Serial

La terminologie Master / Slave est remise en cause ces dernières années dans la communauté des développeurs et l'on évite de l'utiliser sur de nouveaux projets. Comme ces termes sont utilisés dans les spécifications officielles "Modbus Serial", je continuerai des les employer sur cet exemple par cohérence avec les documentations.

### Principe de fonctionnement :

- Seul un Master (au même moment) est connecté au bus, et un ou plusieurs (247 maxi) Slaves sont également connectés sur le bus.
- Une communication Modbus est toujours initiée par le Master. Les Slaves ne vont jamais transmettre de données sans requête du Master.
- Les Slaves ne peuvent pas communiquer entre eux.

Le Master peut initier une transaction avec le Slave suivant deux modes :

- **mode unicast** le Master s'adresse à un Slave individuel. Après réception de la requête et traitement de celle-ci, le Slave renvoie la réponse au Master. Dans ce mode, une transaction Modbus consiste en deux messages: la requête du Master (request) et la réponse du Slave (reply). Chaque Slave doit posséder une adresse unique (de 1 à 247) de manière à ce qu'il puisse être interrogé indépendamment des autres Slaves.

Modbus RTU unicast

Le fait d'interroger les Slaves les uns à la suite des autres consiste à effectuer du "Polling".

- **mode broadcast** le Master envoie **un message à l'ensemble des Slaves**. Les messages de broadcast sont forcément de type écriture. L'adresse 0 est réservée pour identifier un échange de type broadcast.

## Description du protocole

Le protocole Modbus définit un Protocol Data Unit (**PDU**) indépendant des couches de communication. Il s'agit de la structure du message de base :

Modbus PDU

Function Code représente le type d'ordre (lire, écrire) et les datas sont les paramètres de l'ordre (lire 4 registres mémoire depuis l'adresse 0x3214 par exemple).

Empaqueter le protocole Modbus sur un bus série ou Ethernet nécessite des champs additionnels au PDU.

## Modbus RTU PDU

Sur une liaison Modbus série, l'Address field contient uniquement l'adresse du Slave.

Le champ CRC contient un code de contrôle d'intégrité de message pour détecter les erreurs de transmission.

## Les règles d'adressage Modbus

Les adresses des appareils (devices) Modbus sont codés sur 1 octet (8 bits). Il y a donc 256 adresses possibles.

| 0                 | From 1 to 247              | From 248 to 255 |
|-------------------|----------------------------|-----------------|
| Broadcast address | Slave individual addresses | Reserved        |

- L'adresse 0 est réservée comme adresse de broadcast.
- Le Master Modbus n'a pas d'adresse spécifique. Seuls les Slaves doivent posséder une adresse qui doit être unique sur le bus série.
- Le fait que les spécifications Modbus indiquent qu'il est possible d'affecter des adresses comprises entre 1 et 247 ne veut pas forcément dire que tous les fabricants permettent cet interval (certains fabricants limitent les adresses de 1 à 100).

## Les types de données

Il y a deux types de données en Modbus, le bit et le Word (16 bits).

|                | Type d'objet   | Accès      | Exemple                                                            |
|----------------|----------------|------------|--------------------------------------------------------------------|
| Discrete Input | bit            | Read-Only  | Entrée TOR, fin de course, contact auxiliaire de disjoncteurs, ... |
| Coil           | bit            | Read-Write | Sortie TOR, bit interne, RAZ d'un compteur d'énergie, ...          |
| Input Register | Word (16 bits) | Read-Only  | Entrée analogique, lecture d'un capteur, ...                       |

|                  | Type d'objet   | Accès      | Exemple                                                                                                                                                                                 |
|------------------|----------------|------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Holding Register | Word (16 bits) | Read-Write | Sortie analogique, variable de programme (ex : temporisation, opérande d'un calcul,...) Valeur de paramétrage d'un équipement (ex: consigne de vitesse d'un variateur de fréquence,...) |

- Input et Input Register correspondent à des entrées. Ce sont des variables que l'on peut uniquement accéder en lecture (Read-Only).
- Coil (bobine) et Holding Register correspondent à des sorties que l'on peut forcer (write) mais également lire (read).

Un registre est codé sur 16 bits. Holding Register correspond ainsi à 16 Coil en mode Read-Write tandis que Input Register correspond à 16 entrées que l'on peut seulement accéder en lecture (Read-only).

### Rappel :

- 1 Word = 2 bytes = 16 bits
- 1 Register est codé sur un Word soit 16 bits

## Les fonctions Modbus

Les "Function Code" correspondent aux types d'ordres, lire ou écrire par exemple, ainsi que le type d'accès (accès au niveau bit ou au niveau registre de 16 bits). Les fonctions sont identifiées par un code sur 8 bits qui peut être représenté en décimal ou en hexa.

### Bit access

| Code | Hex  | Nom de fonction      | Commentaire                     |
|------|------|----------------------|---------------------------------|
| 02   | 0x02 | Read Discrete Inputs | Physical Discrete Inputs        |
| 01   | 0x01 | Read Coils           | Internal Bits or Physical coils |
| 05   | 0x05 | Write Single Coil    | Internal Bits or Physical coils |
| 15   | 0x0F | Write Multiple Coils | Internal Bits or Physical coils |

### 16-bit access (register)

| Code | Hex  | Nom de fonction     | Commentaire              |
|------|------|---------------------|--------------------------|
| 04   | 0x04 | Read Input Register | Physical Input Registers |

| Code | Hex  | Nom de fonction          | Commentaire                                     |
|------|------|--------------------------|-------------------------------------------------|
| 03   | 0x03 | Read Holding Registers   | Internal Registers or Physical Output Registers |
| 06   | 0x06 | Write Single Register    | Internal Registers or Physical Output Registers |
| 16   | 0x10 | Write Multiple Registers | Internal Registers or Physical Output Registers |

Les tableaux ci-dessus ne sont pas exhaustif, il y a également des Function Code pour réaliser du diagnostic. Il faut savoir que les fabricants de matériel Modbus n'intègre pas forcément toutes les fonctions possibles. Les fonctions Modbus disponibles sont données dans la documentation technique du constructeur.

### Description d'une trame Modbus série

Une communication Modbus série est définie par

- vitesse en bit/s ( 9600, 19200, 115200, autre )
- 1 bit de start
- 8 bits de données (LSB envoyé en premier)
- 1 bit de parité
- 1 ou 2 bit de stop

Classiquement, en Modbus RTU, c'est la parité paire (**Even**) qui est utilisée. Si l'on choisit de ne pas implémenter le contrôle de parité (**None**) il faut placer 2 bits de stop.

Une trame Modbus RTU est composée *a minima* de 4 octets et au maximum de 256 octets. Chaque octet (byte) qui compose une trame Modbus est codé de la manière suivante :

Modbus RTU frame unknown

### Une trame Modbus RTU

Une trame Modbus RTU est ainsi composée :

- 1 byte pour Slave Address
- 1 byte pour Function Code
- 0 à 252 byte pour Data
- 2 bytes pour le CRC

Modbus RTU Frame unknown

La taille maximale d'une trame Modbus RTU est de 256 bytes.

Le CRC est calculé avec l'algo CRC-16-MODBUS.

## Acquisition d'une trame Modbus de type request

### Scope Frame Modbus RTU

Le décodage de trame Modbus intégré donne au format hexa la trame suivante :

```
01 03 00 01 00 02 95 CB
```

On en déduit :

- Slave Address : 01
- Function Code : 03 -> Read Holding Register
- Data : 00 01 00 02
- CRC : 95 CB

Pour Data, suivant les caractéristiques de la fonction 03 Read Holding Register, les deux premiers bytes 00 01 corresponde à l'adresse de registre de départ et les deux suivants 00 02 correspondent au nombre de registres que l'on souhaite lire à partir du registre de départ.

En résumé: la trame Modbus RTU suivante effectue la requête suivante -> Au Slave 01, donne la valeur des 00 02 premiers registres à partir de l'adresse mémoire 00 01.

# Branchement Modbus RTU en configuration 2 Wire

Le branchement Modbus RTU classique est le "2 Wire" en conformité avec le standard RS-485. Sur un "2W-Bus", seul un driver à la fois a la possibilité de transmettre un message.

- LT : Line Terminator, c'est les résistance de terminaison (polarisation) du Bus. Elles font classiquement 120Ω120Ω ou 150Ω150Ω
- Les résistances de terminaison sont placées au début du bus et à la fin du bus.
- Balanced Pair : Paire de fils torsadés qui constituent le support de transmission.

### Modbus Two 2 Wire

On parle de topologie 2 fils (2-Wire), mais on se rend compte sur le schéma, que finalement, 3 fils sont utilisés avec la masse (Common).

| Modbus Name | RS-485 Name | Autre Nom   | Description                                             |
|-------------|-------------|-------------|---------------------------------------------------------|
| D1          | B           | D+ ou Data+ | Transceiver Terminal 1 (V1>V0 for binary 1 [OFF] state) |
| D0          | A           | D- ou Data- | Transceiver Terminal 0 (V0>V1 for binary 0 [ON] state)  |

| Modbus Name | RS-485 Name | Autre Nom | Description        |
|-------------|-------------|-----------|--------------------|
| Common      | C           | 0v ou GND | Commun, Masse (0V) |

En RS-485, à 9600 bit/s sur une paire torsadée en AWG26, on arrive à une longueur de bus maximale de 1000 m!

Les résistances de polarisation (R<sub>Pull-Up</sub> et R<sub>Pull-Down</sub>) permettent de limiter le bruit sur le bus quand il n'y a pas de communication. Les valeurs de ces résistances sont comprises entre 450Ω et 650Ω.

**Remarques :** Il existe également des configurations de branchement en 4 fils (4-Wire) mais c'est rare.

## Connectique Modbus RTU

En Modbus RTU RS-485, trois types de connecteurs sont souvent utilisés :

- bornier à visser (ou borne automatique)
- connecteur DB9
- connecteur RJ45

### Bornier à visser :

Sur le Wago Controller 100, la connexion se fait par un bornier automatique et utilise les abréviations D+ (D1 ou B) et D- (D0 ou A). L'abréviation GND est utilisée pour le commun (0V) et SH (Shield) pour une connexion au blindage.

Modbus Wago Controller 100

### Connecteur DB9 :

L'automate PFC200 de chez Wago utilise une connectique DB9 qui permet de réaliser des liaisons RS-485 ou RS232. Pour le Modbus RTU, c'est la RS-485 qui est classiquement utilisée.

| PFC200 WAGO                     | Connecteur DB9               |
|---------------------------------|------------------------------|
| Modbus Wago PFC200 type unknown | Modbus Wago DB9 type unknown |

La documentation constructeur donne les informations suivantes pour la connectique DB9 du PFC200 en mode RS485.

| Contact | Signal RS-485 | Description  |
|---------|---------------|--------------|
| 1       | NC            | Not assigned |

| Contact | Signal RS-485 | Description             |
|---------|---------------|-------------------------|
| 2       | NC            | Not assigned            |
| 3       | A (Tx/Rx+)    | Transmitt/receive Data+ |
| 4       | NC            | Not assigned            |
| 5       | FB_GND        | Ground                  |
| 6       | FB_5V         | Power Supply            |
| 7       | NC            | Not assigned            |
| 8       | B (Tx/Rx-)    | Transmitt/receive Data- |
| 9       | NC            | Not assigned            |
| Housing | Shield        | Shield                  |

On se rend compte que Wago ne respecte pas la norme Modbus dans ce produit ! Ils appellent A -> Data + et B -> Data - qui correspond à la dénomination Profibus de Siemens ! Si votre communication ne fonctionne pas, il suffit parfois d'inverser les fils A-B car le fabricant a mélangé la norme.

## Connecteur RJ45

Les fabricants utilisent aussi parfois un connecteur RJ45 pour les liaison RS-485 ! L'erreur est de croire que l'on peut connecter ce type d'appareils sur un switch ou sur le port RJ45 de votre PC. **NE LE FAITES PAS !**

Bien qu'il s'agisse d'un connecteur RJ45, il s'agit d'une liaison série qui est transportée et il faut donc l'associer à une interface série et non au port RJ45 de votre PC ou de votre switch ! Les fabricants adoptent parfois la connectique RJ45 car les câbles sont peu chers avec un branchement qui est facile et rapide.

### Wago Current Sensor Modbus RTU

La documentation Wago donne l'association des broches du connecteur RJ45 :

| Pin | Function  |
|-----|-----------|
| 1   | Ub        |
| 2   | Ub        |
| 3   | n.c.      |
| 4   | A (Data+) |
| 5   | B (Data-) |
| 6   | n.c.      |

| Pin | Function |
|-----|----------|
| 7   | GND      |
| 8   | GND      |

Toujours la même erreur chez Wago. Ils appellent A -> Data+ et B -> Data- qui correspond à la dénomination Profibus de Siemens.

## Connexion RJ45 et DB9 selon spécifications Modbus

Modbus DB9 RJ45

| Pin on RJ45 | Pin on DB9 | Level of requirement | Modbus | RS-485 | Description                                                         |
|-------------|------------|----------------------|--------|--------|---------------------------------------------------------------------|
| 3           | 3          | optional             | PMC    | -      | Port Mode Control                                                   |
| 4           | 5          | <b>required</b>      | D1     | B      | Transceiver terminal 1, V1 Voltage (V1>V0 for binary 1 [OFF] state) |
| 5           | 9          | <b>required</b>      | D0     | A      | Transceiver terminal 0, V0 Voltage (V0>V1 for binary 0 [ON] state)  |
| 7           | 2          | recommended          | VP     | -      | Positive 5..24 Vdc Power Supply                                     |
| 8           | 1          | <b>required</b>      | Common | C      | Signal and Power Supply Common                                      |

On se rend compte que Wago n'a pas suivi les recommandations de câblage fixées par la Modbus Organization, de nombreux fabricants font de même. Quand il s'agit d'appareillages d'un même constructeur, cela ne pose pas de soucis, par contre, il faut parfois inverser les signaux A et B quand on mélange les appareillages de fabricants différents sur un même bus Modbus RTU. En Modbus TCP, comme c'est sur du câble Ethernet, on n'a pas ce problème.

# Exemple : Modbus RTU avec un capteur de Température et

# Humidité

Dans cet exemple, je vais connecter un capteur de température et d'humidité PKTH100B-CZ1 qui communique en Modbus RTU avec mon ordinateur portable.

Pour que le PC portable puisse communiquer en RS-485, je lui ajoute un convertisseur FTDI USB-RS485, ainsi qu'un Oscilloscope pour visualiser les trames Modbus-RTU (côté didactique)

| Capteur PKTH100B-CZ1          | FTDI USB-RS485              | Oscilloscope                    |
|-------------------------------|-----------------------------|---------------------------------|
| PKTH100-1 Sensor type unknown | FTDI USB-RS485 type unknown | Siglent SDS1202X-E type unknown |

L'analyse de la documentation du câble FTDI USB-RS485 nous donne les informations suivantes :

## Les câble USB-RS485

FTDI USB-RS485 colors  
Image not found or type unknown

| Signal              | Couleur de fil |
|---------------------|----------------|
| GND                 | Noir           |
| (A) Data -          | Jaune          |
| +5V                 | Rouge          |
| R de 120Ω120Ω pin 1 | Brun           |
| (B) Data +          | Orange         |
| R de 120Ω120Ω pin 2 | Vert           |

## Le capteur

PKTH100-1 Sensor type unknown  
Image not found or type unknown

| Terminals number | 1      | 2      | 3      | 4      |
|------------------|--------|--------|--------|--------|
| Identifying      | GND    | VCC    | B      | A      |
| Description      | Power- | Power+ | RS485- | RS485+ |

On remarque que sur la documentation du capteur, le signal A est nommé RS485+ tandis que sur la document du convertisseur USB-RS485, le signal A est nommé Data - ...

On va rester pragmatique et brancher le fil A (jaune) sur le bornier A (4) du capteur et le fil B (Orange) du convertisseur vers la borne B (3) du capteur. Si jamais cela ne fonctionne pas, il suffira d'inverser ;)

Les masses devant être communes, on branchera le fil GND (noir) du convertisseur à la borne GND (1) du capteur.

Pour alimenter le capteur, j'utilise une alimentation de laboratoire de 24Vdc. Pareil, je brancherai le +24Vdc de l'alimentation à la borne VCC (2) du capteur et le 0V de l'alimentation à la borne GND (1) du capteur.

Pour les résistances de terminaison de  $120\Omega$ , je fais le choix de ne pas les placer dans un premier temps car la longueur de bus est très faible.

# La manipulation

## Manipulation Modbus Capteur Temperature

La documentation (en chinois) indique les paramètres suivants :

- Vitesse de transmission : 9600 bit/s
- 8 bits de données
- Parity : None
- 1 Stop bit (non respect de la norme)
- Slave Address (factory) : 1

Le document indique également que la requête à envoyer est une fonction de type `03` Read Holding Register à l'adresse de Slave `1` et que l'on lit à partir du registre mémoire `0` un nombre de 2 registres.

La trame à envoyer avec le CRC est la suivante : `01 03 00 00 00 02 C4 0B`

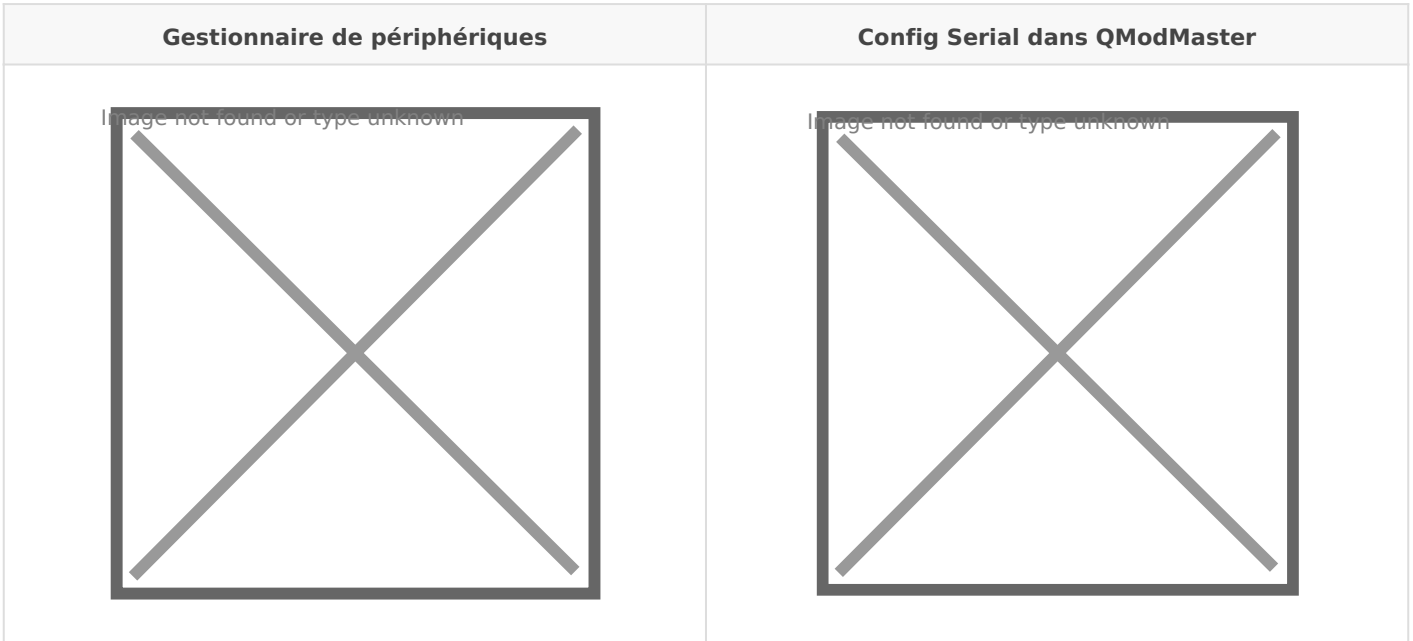
J'utilise le logiciel QModMaster pour générer facilement la trame et bien sûr, cela ne fonctionne pas :(

On va essayer d'inverser les fils A et B -> boum, ça fonctionne...bref

## Les différentes étapes de la configuration de qModMaster

On le numéro du port Com utilisé par le convertisseur USB-RS485 avec le gestionnaire de périphériques Windows. On remarque que dans mon cas, c'est le COM5 qui lui a été attribué. Cela nous permet de paramétrer la liaison série RTU dans QModMaster avec le bon numéro de Com et

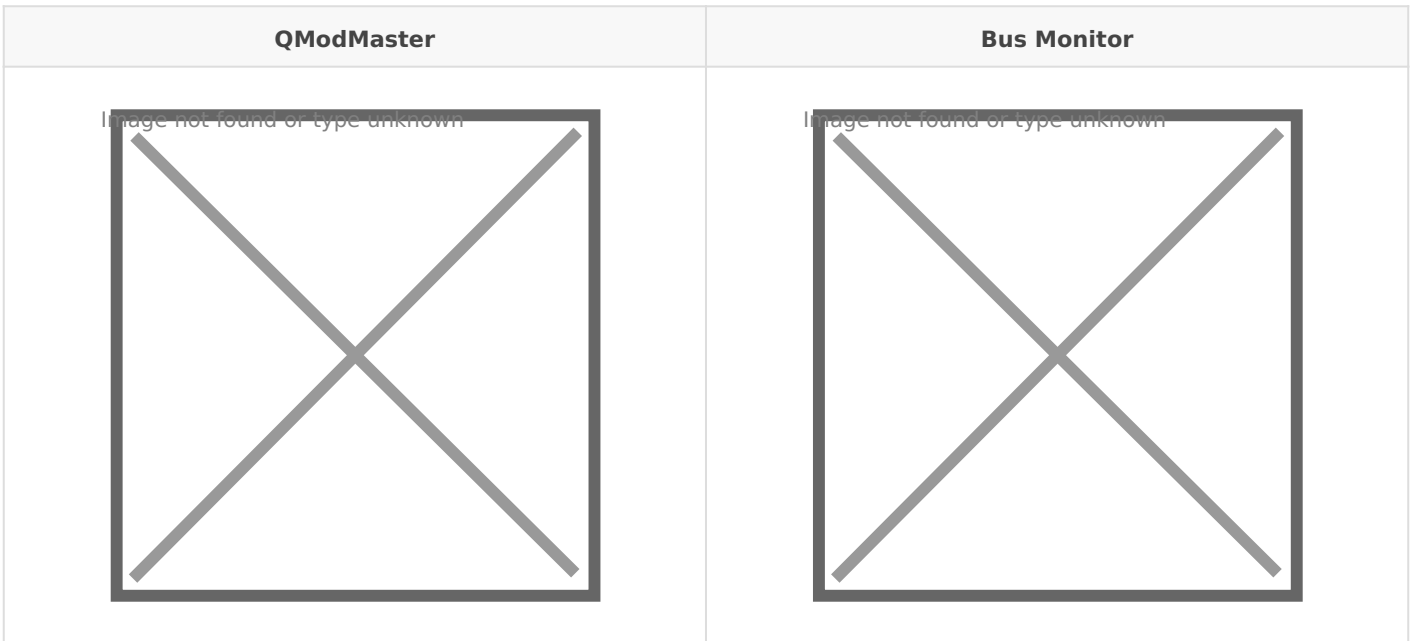
l'on saisie également les paramètres de liaison du capteur de température (9600 bit/s 8bits de données 1 bit de stop et parity None)



Dans QModMaster, je choisis le Mode RTU, le Slave Address à 1, le Function Code à `0x03` pour Read Holding Register, le Start Address à 0 et Number of Coils (Registers) à 2.

Dans le Bus Monitor, on remarque que la trame de request vaut : `01 03 00 00 00 02 C4 0B` -> ce qui était demandé par la doc, donc on est OK !

La trame de réponse (reply) du capteurs vaut : `01 03 04 01 28 02 22 FA BE`



Le décodage du résultat est donné directement par QModMaster:

- Le premier registre vaut : `296` en décimal

- Le second registre vaut : `546` en décimal

La documentation du capteur indique que la valeur du premier registre correspond à la température multipliée par 10. On en déduit qu'il fait 29,6°C en cette journée d'août -> c'est OK

L'humidité multipliée par 10 est dans le second registre. On en déduit que l'humidité relative Hr=54.6% ce qui est conforme.

## Méthode de décodage à partir de la trame de réponse

La trame de réponse (reply) du capteurs vaut : `01 03 04 01 28 02 22 FA BE` . On peut décoder le contenu de la manière suivante :

- `01` : correspond à l'adresse du capteur qui donne la réponse
- `03` : indique qu'il répond à une requête de type 03 Read Holding Register
- `04` : c'est la valeur de la fonction 03 + 1 pour dire que tout c'est bien passé !
- `01 28` : c'est la valeur en hexa du contenu du premier registre avec `01` l'octet de poids fort et `28` l'octet de poids faible. Converti en décimal, on obtient `296`
- `02 22` : correspond à la valeur en hexa du second registre. Converti en décimal, on obtient `546` .
- `FA BE` : correspond au CRC de la trame de réponse.

## Capture des trames Modbus RTU avec l'oscilloscope

On peut observer la trame de request générée par QModMaster qui vaut `01 03 00 00 00 02 C4 0B`

Modbus Oscilloscope Frame

Et la trame de réponse du capteur qui vaut `01 03 04 01 28 02 22 FA BE`

Modbus Oscilloscope Frame

Source <https://celka.fr/ocw/plc-control/modbus/intro-modbus/intro/>

Philippe Celka Copyright © 2025 CC Attribution-Non Commercial-Share Alike 4.0 International