

# Labo 08 - Intégration de matériel industriel Modbus/TCP

## 1. Présentation

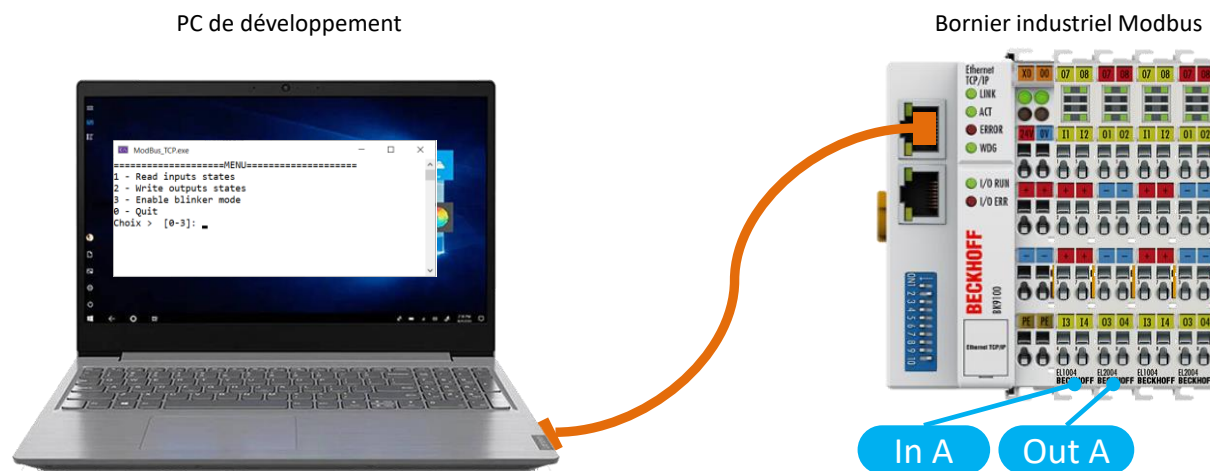
Ce laboratoire a pour but de développer en C ou C++ une implémentation du protocole Modbus/Tcp client à l'aide de l'API socket, permettant d'utiliser des systèmes entrées/sorties industriels disponibles sur ce protocole, depuis une application PC.

### Objectifs pédagogiques

- Apprendre à établir une communication TCP avec l'API socket en C.
- Implémenter en C le protocole Modbus côté client.
- Développer une fonction d'automatisation en C.

## 2. Matériel

Le schéma ci-dessous présente l'architecture finale du système à réaliser :



Le système est constitué d'un PC de développement sur lequel une application console, à développer en langage C ou C++, doit gérer la communication par Modbus/TCP avec un bornier industriel Beckhoff et remplir les fonctions d'automatisation décrites ci-après.

Le bornier industriel comporte 2 modules d'entrées tout-ou-rien de type KL1104 ou KL1404, ainsi que 2 modules de sorties tout-ou-rien de type KL2134.

## 2.1. Bornier industriel BK9100

Le bornier industriel et un coupleur Ethernet BK9100 de Beckhoff avec switch deux ports (RJ45) intégré, compatible Modbus TCP.

Il s'agit de configurer la carte réseau du module pour sélectionner une configuration manuelle de l'adresse (DIP SWITCH 9 et 10 à OFF). Dans ce cas de configuration, une partie de l'adresse IP (les trois 1<sup>ers</sup> octets) ainsi que le masque de sous réseau sont paramétrés via l'écriture de différents registres (à l'aide d'un programme et câble spécial qui ne seront pas utilisés au labo). Le dernier octet de l'adresse IP est quant à lui configuré via les DIP SWITCH 1-8, le switch 1 correspondant au poids faible et le switch 8 au poids fort.

BK9000, BK9100, BC9000, BC9100



Switch no.	1	2	3	4	5	6	7	8	
Weight	1	2	4	8	16	32	64	128	
In this example	ON	OFF	OFF	ON	OFF	OFF	ON	ON	
Value	1	0	0	8	0	0	64	128	Total=201

## 2.2. Configuration du bornier industriel

Le bornier industriel intègre un serveur Modbus TCP. Ce serveur s'active automatiquement à la mise sous tension du module. Le BK9100 possède une seule carte réseau, déjà configurée. Sa configuration est la suivante :

Adresse IP : 192.168.1. $m$  avec  $2 \leq m = \text{DIP SWITCH 1-8} \leq 254$

Masque de sous réseau : 255.255.255.0

Port de communication TCP : 502

### 3. Travaux à réaliser

#### 3.1. Désactivation du Watchdog interne du bornier Modbus

Afin d'éviter que le bornier BK9100 se mette en défaut, il est conseillé pour ce travail de laboratoire de désactiver son watchdog interne en écrivant la valeur 0 dans le registre 0x1120 à l'aide de la fonction Modbus n° 6 « Write Single Register » décrite plus loin dans ce document.

Pour cela, il faut tout d'abord établir une communication TCP avec le bornier industriel en établissant une connexion sur le port 502, puis envoyer les octets correspondant au télégramme de la fonction Modbus n° 6.

Cette fonction peut être exécutée automatiquement au démarrage du programme.

#### 3.2. Menu pour accéder aux fonctions

Le menu principal suivant doit être mis en place pour donner l'accès aux différentes fonctionnalités :

```
=====MENU=====
1 - Read inputs states
2 - Write outputs states
3 - Enable blinker mode
0 - Quit
Choix > [0-3]:
```

#### 3.3. Lecture des entrées

Lors du choix du menu 1, il s'agit de gérer l'échange avec le bornier pour la fonction Modbus n°2, permettant de lire 8 entrées tout-ou-rien, et d'afficher l'état des 8 entrées sur la console.

#### 3.4. Ecriture des sorties

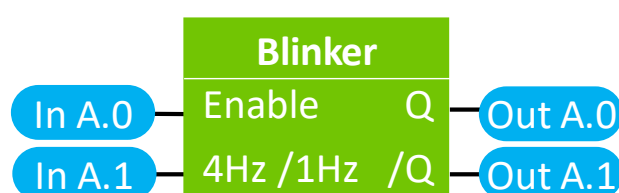
Lors du choix du menu 2, il s'agit de procéder à la saisie de la valeur désirée pour les 8 sorties tout-ou-rien et de les appliquer sur le bornier en gérant un échange de télégrammes pour la fonction Modbus n° 15.

### 3.5. Clignoteur avec le bornier industriel Modbus/TCP

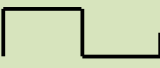
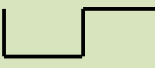


Pour cette dernière étape, il s'agit de compléter le programme gérant la communication avec le bornier industriel Modbus/TCP, et réalisant la fonctionnalité suivante :

- On utilise 2 entrées tout ou rien InA.0 et InA.1
- On utilise 2 sorties tout ou rien OutA.0 et OutA.1

Le comportement à réaliser entre ces entrées et sorties est décrit par le bloc fonctionnel suivant :



La table de vérité ci-dessous résume le comportement attendu :

Enable	4Hz /1Hz	Q	/Q
0	0	0	0
0	1	0	0
1	0		
1	1		

Le comportement de ce processus est le suivant :

- Activation du clignotement de la sortie *Q* lorsque l'entrée *Enable* est active. La fréquence de base du clignotement est de 1Hz.
- Modification de la fréquence de clignotement à 4Hz lorsque l'entrée *4Hz / 1Hz* est active.
- Lorsque le clignotement est actif, la sortie */Q* prend l'état inverse de la sortie *Q*, afin de réaliser un clignotement en inversion de phase.
- Lorsque l'entrée *Enable* est désactivée, les sorties *Q* et */Q* sont à zéro.

Remarque : cette gestion du clignotement s'exécute en boucle continue. Il est souhaitable de prévoir une possibilité pour l'utilisateur de quitter cette boucle, par exemple à la pression d'une touche du clavier. Il est possible de détecter la pression d'une touche à l'aide de la fonction `int _kbhit()` de la bibliothèque `<conio.h>`. Cette fonction non bloquante retourne 1 si une touche a été pressée et 0 dans les autres cas.

### 3.6. Type de données abstrait TON

Pour faciliter le travail de programmation, un type de données abstrait écrit en C et appelé TON est fourni avec un programme exemple. Il permet de réaliser les chronométrages dans une logique similaire à ce qui est possible avec le bloc fonctionnel TON du langage IEC1131.

```
typedef struct
{
    unsigned long start_tick;
    unsigned long duration;
} TON;

/// <summary>
/// Initializes a TON data structure representing a Timer.
/// </summary>
/// <param name="ton">The timer data structure to initialize</param>
void TON_Initialize(TON* ton);

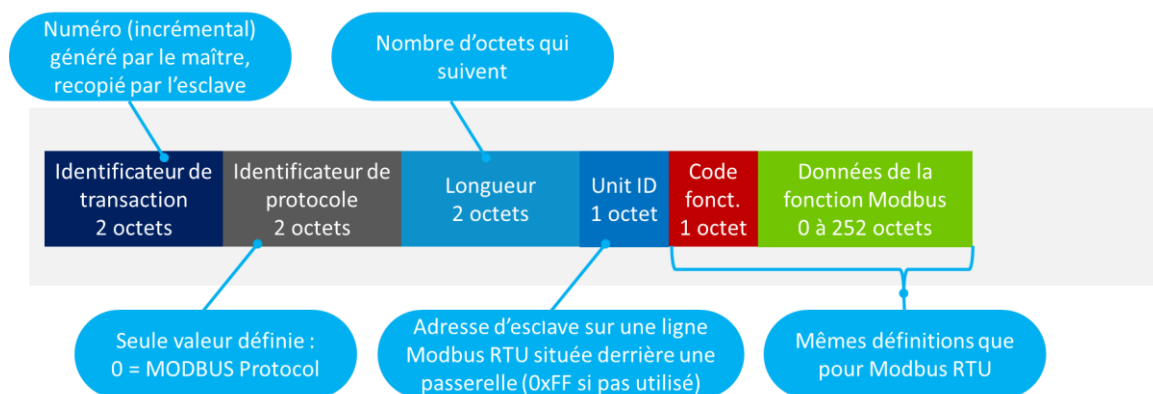
/// <summary>
/// Starts a timer with the corresponding duration.
/// </summary>
/// <param name="ton">The timer to start</param>
/// <param name="duration">Timer duration in ms</param>
void TON_Start(TON* ton, int duration);

/// <summary>
/// Determines if a timer has elapsed.
/// </summary>
/// <param name="ton">The timer</param>
/// <returns>1 if the timer is elapsed, 0 otherwise.</returns>
char TON_IsElapsed(TON* ton);
```

## 4. Rappels concernant Modbus TCP

### 4.1. Format des télégrammes Modbus TCP

Le schéma ci-dessous présente la structure d'un télégramme Modbus TCP.



### 4.2. Codes fonctions Modbus

Le tableau ci-dessous présente les fonctions Modbus définies par la norme. Pour la suite, nous utiliserons les fonctions 6, 15 et 2.

	Décimal	Hexa	Fonction	Description
Fonctions de Bits	1	01	Read Coil Status	Lecture de l'état de une ou plusieurs sorties
	2	02	Read Input Status	Lecture de l'état de une ou plusieurs entrées
	5	05	Write Single Coil	Ecriture d'une sortie unique
	15	0F	Write Multiple Coil	Ecriture de une ou plusieurs sorties
Fonctions de Words (16 bits)	3	03	Read Holding Register	Lecture d'un registre de mémoire
	4	04	Read Input Register	Lecture d'un registre d'entrée
	6	06	Write Single Register	Ecriture d'un registre de sortie unique
	16	10	Write Multiple Register	Ecriture de un ou plusieurs registres de sortie
	8	08	Diagnosis	6 différentes sous fonctions pour gérer les erreurs

### 4.3. Données d'un télégramme Modbus 6 « Write single register »

La fonction Modbus n° 6 permet d'écrire un registre 16 bits.

Données	Taille	Nom du paramètre	Signification
Requête	2 octets	Register address	Numéro du registre à écrire.
	2 octets	Register value	Valeur à écrire dans le registre.
Réponse	2 octets	Register address	Numéro du registre à écrire. (écho)
	2 octets	Register value	Valeur à écrire dans le registre. (écho)

#### 4.4. Données d'un télégramme Modbus 15 "Write Multiple Coils"

La fonction Modbus n°15 permet d'écrire plusieurs sorties digitales.

Données	Taille	Nom du paramètre	Signification
Requête	2 octets	Start address	Numéro du bit de la première sortie à écrire.
	2 octets	Length	Nombre de sorties à écrire.
	1 octet	Byte count	Nombre d'octets envoyés contenant les valeurs de sorties.
	n octets	Data	Valeur des sorties, LSB du 1 <sup>er</sup> octet=1 <sup>ère</sup> sortie
Réponse	2 octets	Start address	Echo, même valeur que la requête
	2 octets	Length	Echo, même valeur que la requête

#### 4.5. Données d'un télégramme Modbus 2 "Read Input Status"

La fonction Modbus n°2 permet de lire plusieurs entrées digitales.

Données	Taille	Nom du paramètre	Signification
Requête	2 octets	Start address	Numéro du 1 <sup>er</sup> bit d'entrée à lire
	2 octets	Length	Nombre de bits d'entrée à lire
Réponse	1 octet	Byte count	Taille des données de la réponse
	n octets	Data	Valeur des entrées, LSB du 1 <sup>er</sup> octet=1 <sup>ère</sup> entrée

## **5. Travaux à rendre**

### **5.1. Liste des livrables**

- Une copie papier du listing du programme embarqué et du programme sur PLC.
- L'ensemble des fichiers des projets permettant de reconstruire les programmes, à placer à la fin dans le répertoire réseau du laboratoire BusTer.

### **5.2. Délai**

Le délai pour rendre la version imprimée et les fichiers est fixé à une semaine après la séance en laboratoire.

Un test d'acceptation de votre implémentation aura lieu lors de la dernière séance de laboratoire.

### **5.3. Critères de qualité**

Les éléments suivants sont pris en compte dans l'évaluation de ce travail :

- Couverture fonctionnelle du cahier des charges.
- Respect précis des exigences.
- Performance.
- Lisibilité et maintenabilité du code source.