

Documentation Projet Pont IP-Serie pour boucle d'asservissement.

COUTAUD Ulysse

<2022-04-08 Fri>

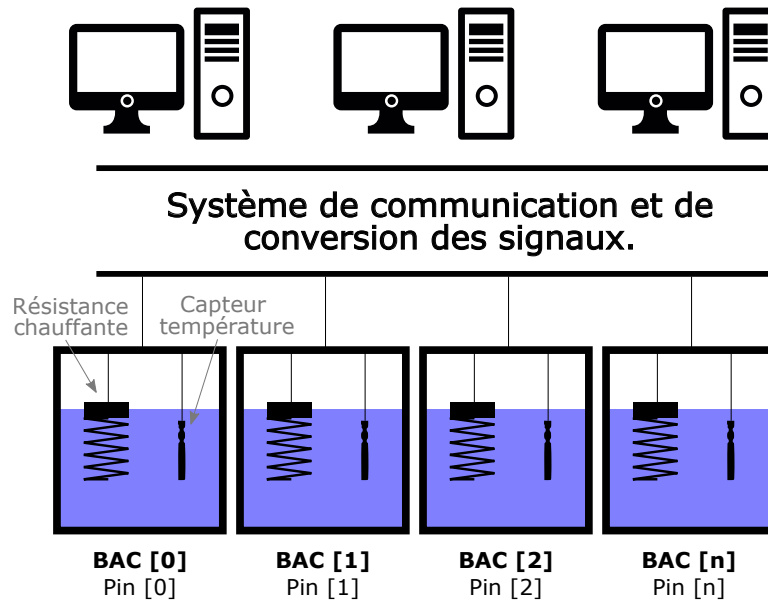
Contents

1	Intro	2
2	Installation	3
2.1	Installation de l'environnement	3
2.1.1	Télécharger le projet	3
2.1.2	Installer le projet	3
3	Architecture	6
3.1	Matérielle	6
3.2	Architecture réseau	6
3.3	Architecture logicielle	6
4	Protocole	8
4.1	Version du protocole	8
4.2	Les fonctions	8
4.2.1	0 - Erreur	8
4.2.2	1 - Lecture analogue	9
5	Guide d'utilisation	10
6	Améliorations	11

1 Intro

- Le système présenté dans ce document est l'interface de communication pour l'asservissement en température d'un parc de bac d'eau:

Clients (Windows LabView, Linux CLI).



Bacs à température régulée.

- Le système permet au *Client* :
 - D'envoyer des requêtes pour déclencher une lecture de la valeur d'un capteur, et de recevoir la valeur lue.
 - D'envoyer des requêtes pour inscrire une valeur de commande sur un actionneur.

2 Installation

2.1 Installation de l'environnement

2.1.1 Télécharger le projet

1. En ligne de commande:
 - Télécharger le dépôt git (permet d'avoir les mise à jour via simple "git pull":
 - Soit : `git clone https://github.com/coutaudu/CommandLabviewRpiArduino.git`
 - Soit : `git clone ssh://git@github.com/coutaudu/CommandLabviewRpiArduino.git`
 - Télécharger juste les sources:
 - `wget https://github.com/coutaudu/CommandLabviewRpiArduino/archive/refs/heads/main.zip`
 - Si besoin:
 - `sudo apt-get install git`
 - `sudo apt-get install wget`
2. Via un navigateur web:
 - Page <https://github.com/coutaudu/CommandLabviewRpiArduino>

2.1.2 Installer le projet

1. Les micro-contrôleurs
 - (a) Flasher un UID sur chacun des micro-contrôleurs du système
 - Il est IMPERATIF d'inscrire un UID dans chacun des micro-contrôleurs utilisés dans le système (nécessaire pour la détection puis la redirection des requêtes vers le bon micro-contrôleur).
 - Pour cela il faut utiliser la fonction `int setBoardUID(unsigned char uid);`
 - Décommenter la ligne dans la fonction `setup` en inscrivant l'UID voulu.
 - Compiler puis flasher le micro-contrôleur. L'UID a été gravé dans l'EEPROM.

- Recommander la ligne, compiler et flasher pour ne pas re-graver l'UID par la suite.
- (b) Flasher le firmware sur chacun des micro-contrôleurs du système
 - Le firmware doit être flashé sur chacun des micros contrôleurs.
 - Testé avec Arduino IDE 1.8.19.
 - Carte Arduino AVR Board > Arduino UNO.
 - Utilise librairie "EEPROM.h" (built-in).
 - Les micro-contrôleurs sont branchés en USB.
 - Une fois branché, le Serveur doit les détecter: `/dev/ttyACM0` et `/dev/ttyACM0`.
 - Les paramètres de communication série sont:
 - * 115200 bauds
 - * 8 bits de données
 - * 0 bits de parité
 - * 1 bit de stop
 - * binaire nu (toutes les valeurs de contrôles soft, telles que XonXoff sont désactivées).

2. Le serveur

- Installer l'environnement: `rpi@raspberrypi:~/CommandLabViewRpiArduino`
`$ sudo ./install-required-software.sh`
- Compiler le code du serveur: `rpi@raspberrypi:~/CommandLabViewRpiArduino/Serveur`
`$ make`
- Le Serveur peut être lancé soit comme un exécutable "classique" (`./SerialServeur`), soit être installé et exécuter comme un service.
- Le fichier de configuration `SerialServeur.conf` contient les paramètres du service. À adapter si besoin de modifier des paramètres, notamment le nom d'utilisateur par défaut (`rpi`).
- Installer le service: via les commandes `make`:
 - `make install_service` pour installer le service.
 - `make run_service` pour lancer manuellement le service:
 - `stop_service` pour stopper le service.
 - `make auto_run_service` pour lancer automatiquement le service à la mise sous tension.
 - `make disable_service` pour désactiver le démarrage automatique.

3. Les clients

(a) Client CLI

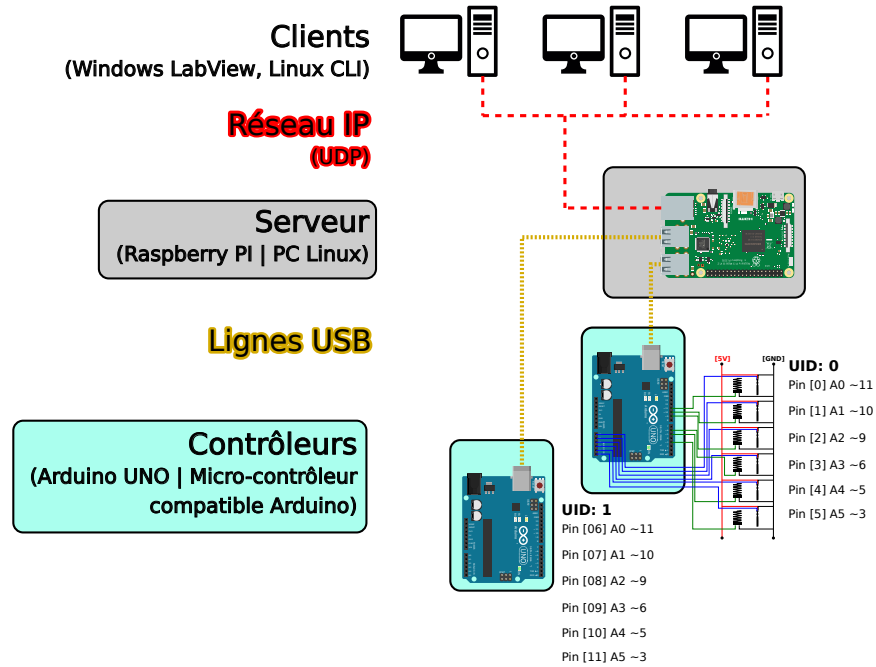
- Configurer IP et port destination (du serveur donc) dans le code source.
- Compiler via la commande *make*.
- Executer *./Client*

(b) Clients Labview

- Installer Labview 21.
- Ouvrir le projet.
- Choisir le programme d'exemple souhaité.
- Executer.

3 Architecture

3.1 Matérielle



3.2 Architecture réseau

- Les clients communiquent avec le serveur en UDP/IP.
- Le serveur communique avec les micro-contrôleurs en USB.
- Le protocole applicatif [TODO NOM DE PROTOCOLE] est un développement spécifique au projet. Le protocole est décrit en section 4.
- Le serveur fait :
 - Pont UDP<->USB.
 - Routage N°Pin<->UID micro-contrôleur.

3.3 Architecture logicielle

- Le code du projet est découpé en 3 modules:
 - **Arduino** qui contient le firmware des micro-contrôleurs.

- **Serveur** qui contient le code pour la communication réseau USB/UDP (Raspberry PI/ Linux).
- **Client** qui contient des exemples d'application client utilisant le système:
 - * CLI: Programme minimal codé en C avec interface en ligne de commande pour envoyer et recevoir des requetes.
 - * GUI: Panel de fonction LabView faisant office de bibliothèque d'utilisation du système de communication avec des exemples d'applications graphiques.
- Le fichier d'entête Arduino/shared.h contient les définitions des types du protocole de communication. Il est donc partagé entre le code du firmware des micro-controlleurs et le code du Serveur.
- Le code doit être sujet à une passe de nettoyage et de structuration. Je ne met donc pas de détails sur la structure du code dans son implémentation actuelle qui doit être considérée comme un *Proof Of Concept*.

4 Protocole

- Le protocole est basé sur le principe de *requête/réponse*.
 - 1 *requête* implique 1 *réponse* de même code fonction.
 - Les clients ou le serveur pilotent la communication: ils envoient les requêtes.
 - Les micro-contrôleurs sont esclaves de la communication: ils reçoivent les requêtes et y répondent pas des réponses.
- Les requêtes et les réponses ont le même format.

Numéro de version	Code fonction	Argument [0]	Argument [1]
1 octet	1 octet	1 octet	1 octet

4.1 Version du protocole

- Version 0 = erreur.
- Version 1 = version actuelle.
- Version 2-255 = RFU (Reserved for Future Use). Le numéro de version est vérifié lors des communications entre contrôleurs et serveur doit être cohérents. Le numéro de version doit être incrémenter en cas d'ajout ou de modification des fonctions implémentées.

4.2 Les fonctions

- 0 = Erreur.
- 1 = Lecture analogue.
- 2 = Ecriture digitale PWM.
- 3 = Lire UID carte.
- 4-255 = RFU.

4.2.1 0 - Erreur

- Le code fonction 0 indique une erreur.
 - Par exemple: lecture sur un numéro de pin inconnu, réponse à un code fonction inconnu.

- Une requête de code fonction *erreur* doit recevoir une réponse de code fonction erreur.

4.2.2 1 - Lecture analogue

- Le code fonction 1 indique une lecture analogue.
- Requête:
 - Argument[0] contient le numéro de pin cible (voir schémas en section 3.1 et 1).
 - Argument[1] n'est pas utilisé.
- Réponse:
 - Argument contient la valeur analogue lue par le micro-contrôleur (valeur entre 0 et 1024 au format unsigned int sur 2 octets dans le cas *Arduino Uno*).

5 Guide d'utilisation

6 Améliorations

- ☐ Restructuration et nettoyage du code.
- ☐ Log rotatifs.
- ☐ Passage des arguments du serveur en ligne de commande.
- ☐ Passage des arguments du serveur en fichier etc/xml/json/csv.
- ☐ Installer serveur smb sur serveur pour accès au fichier de configuration
- ☐ Mesures des temps d'exécution.
- ☐ Documentation.
- ☐ WebViewer