
CPE Lyon - 4IRC - S7 - Année 2019/20

Architecture et Protocoles Réseaux pour l'IoT

TP 1 - Prise en main d'un micro-contrôleur

Ce TP a pour but deux objectifs :

1. Vous familiariser avec l'environnement de développement qui sera utilisé pendant tous les TP IoT.
2. Vous faire découvrir la programmation d'un micro-contrôleur.

Analyse et préliminaires

Exercice 1. Étude du système

La première étape lorsqu'on veut développer un système embarqué est de sélectionner le matériel qui sera utilisé. Cela passe par la définition des besoins, puis l'étude des solutions existantes.

Dans notre cas, le besoin est relativement simple : la découverte de la programmation sur micro-contrôleur. Pour ce qui est du choix du système, il a été fait pour vous, ce qui est bien souvent le pire des cas possible, car vous n'avez pas été consultés en amont, et devez donc vous débrouiller avec ce qui vous est imposé.

Dans le cas de ce TP, quatre choix étaient possibles.

- Utiliser des cartes C8051F02x de SiliconLabs.
- Utiliser des cartes Arduino Uno.
- Utiliser des cartes STM32 de ST Micro-electronics.
- Utiliser les modules RF Sub-1Ghz de Techno-Innov.

Question 1. Rechercher les caractéristiques des diverses cartes en question et les micro-contrôleurs utilisés par chacune d'entre elles.

Le choix c'est porté sur la quatrième solution, car le micro-contrôleur utilisé par ces cartes est beaucoup plus adapté et répandu pour le développement de systèmes embarqués que ceux des deux premières solutions, qui utilisent des micro-contrôleurs 8 bits, coûteux, et peu puissants.

Les deux autres solutions utilisent des micro-contrôleurs de même architecture, mais notre choix s'est portée vers les cartes Techno-Innov.

Exercice 2. Documentations techniques

La deuxième étape lorsqu'on veut écrire du code pour un système embarqué, c'est de trouver la documentation technique du micro-contrôleur.

L'accès aux documentations techniques est un des critères principaux dans le choix des composants pour un projet, aussi bien pour les concepteurs que pour les utilisateurs.

Si vous avez réalisé la conception du circuit, vous aurez normalement déjà eu besoin de ces documentations, dans le cas contraire, c'est le moment de les récupérer. Pour le module RF Sub-1GHz, il y a plusieurs documentations techniques nécessaires pour utiliser la totalité des fonctionnalités du module. La première et la plus importante est celle du micro-contrôleur, mais vous aurez aussi besoin de [celle du module RF](#) et celles de l'EEPROM I2C et du capteur de température I2C si vous souhaitez les utiliser dans le cas du capteur météo intégré.

Note : Pour ce qui est du bridge USB-UART, la documentation n'est pas nécessaire pour la programmation du micro-contrôleur et du module, mais elle le devient si vous voulez modifier la façon dont le composant

s'identifie lorsque vous le connectez, par exemple pour pouvoir lui donner un nom spécifique ou simplifier l'identification. Cela ne sera cependant pas abordé pendant ce TP.

Pour récupérer les documentations techniques, vous avez le choix entre le site du fabricant et celui du distributeur chez qui vous avez achetés les composants. Pour la majorité des composants on utilise le site du distributeur (possible uniquement s'il fournit les bonnes documentations), mais pour les micro-contrôleurs il est préférable de consulter le site du fabricant car il fournit aussi les "errata" (notes d'informations contenant les corrections sur la documentation) et le plus souvent des notes d'application qui indiquent comment utiliser le composant pour telle ou telle application (avec parfois des exemples de code).

Outils de programmation

La troisième étape ne concerne toujours pas le code que vous voulez écrire.

Pour écrire du code vous n'avez pas besoin de la "cible" (target en anglais) qui est le matériel sur lequel vous allez exécuter la version compilée du code, mais uniquement d'un poste de développement, couramment appelé hôte (host en anglais).

Cependant, la cible en question n'a que faire des fichiers sources qui se trouvent sur votre poste de développement, et il vous faudra un certain nombre d'outils pour passer des fichiers source (quelque soit le langage) au code exécutable par votre micro-contrôleur.

C'est aussi un point très important dans le choix d'un système embarqué ou d'un micro-contrôleur : quels sont les outils dont j'aurais besoin pour passer de mon code source à un système fonctionnant avec ?

Dans le cas des micro-contrôleurs de la gamme LPC de NXP il existe de nombreuses solutions pour passer de l'un à l'autre, mais un des points appréciés c'est qu'il est possible de le faire avec un minimum de matériel et de logiciel : une liaison série "TTL 3.3V", une chaîne de compilation croisée et un utilitaire pour "uploader" le code binaire (et bien évidemment votre éditeur de texte préféré).

Environnement de travail

Pour la suite de TPs micro-contrôleurs, vous allez utiliser la VM VirtualBox nommée **debian_aiot** :

- **Attention** : Pour démarrer cette VM vous devez avoir tout le temps votre micro-contrôleur inséré dans votre machine.
- Dossier sous Linux : **sync/VMs/debian** ; si non disponible, la récupérer sur le serveur **/softwares/sync/VMs/debian**
- OS : Debian 64 bits
- RAM : 2 Go au minimum
- Augmenter la taille de la mémoire vidéo
- Login (root) : **tp** ; mot de passe : **tp**
- Activer le presse-papier partagé (bidirectionnel)

Matériel : un port USB (et le PC qui va avec)

Pour ce qui est de la liaison série "TTL 3.3V", il existe plein d'adaptateurs USB-UART fonctionnant en 3.3V, et pour le cas du module RF Sub-1GHz cet adaptateur est intégré dans le module, un port USB suffit donc.

L'accès à cette liaison série se fera via un fichier spécial dont le nom devrait ressembler à **"/dev/ttyUSB0"**, le "USB0" pouvant différer selon votre système ("USB1", "USB2", ... ou même "ACM0" avec d'autres adaptateurs).

Attention : La machine virtuelle que vous allez utiliser à une liaison du port **/dev/ttyUSB0** de votre machine hôte vers le port **/dev/ttyS3** de la distribution Linux virtuelle.

Compilation : GNU

Pour ce qui est de la chaîne de compilation croisée, les micro-contrôleurs LPC de NXP utilisent des cœurs ARM Cortex-M*, très bien supportés par gcc, que vous devriez connaître, et parfaitement adapté à la cross-compilation.

Dit comme ça, c'est simple ... mais en fait, pas tant que ça. Cette problématique pourrait faire l'objet d'un (petit ?) TP, voici juste quelques pistes et supposons que vous saurez trouver les informations sur le net.

Lorsque vous n'avez pas déjà une chaîne de cross-compilation installée pour ARM vous avez globalement trois solutions : Debian/EmDebian (celle que vous allez utiliser), Launchpad, et Crosstools-ng.

Le projet EmDebian fournit des paquets debian pour différentes chaînes de cross-compilation (ARM parmi tant d'autres), qui sont désormais intégrées à Debian. Cela devrait simplifier les problèmes de dépôts et de dépendances dont souffrait les dépôts EmDebian, même si lors de l'écriture de l'article dont est tiré ce TP seule la version "arm-none-eabi" (qui nous suffit, nous n'avons pas besoin de libC) était intégrée dans la distribution Debian/SID sans problèmes de dépendances. (On a bien dit "devrait"). Pour être tranquille pendant les TP, cette chaîne de cross-compilation est déjà installée sur la VM fournie.

Le site de Launchpad fourni aussi des versions binaires de la chaîne de cross-compilation GCC ARM.

La solution CrossTools-ng : recompilation de sa propre chaîne de compilation croisée : la solution du dernier recours, si votre distribution ne fournit pas de solution packagée et que les versions binaires non signées ne vous conviennent pas.

Question 1. Une chaîne de cross-compilation pour micro-contrôleurs ARM est déjà installée sur la VM fournie. Quelle est cette chaîne de cross-compilation ? Comment l'utilise-t-on ? Utilisez l'option -v de gcc (celui de la chaîne de cross-compilation) pour identifier la version et vérifier qu'il s'exécute correctement.

Pour permettre l'étude de l'exemple suivant et permettre le développement "linux embarqué" par la suite, nous avons aussi installé deux autres chaînes de compilation pour ARM : gcc-arm-linux-gnueabi et gcc-arm-linux-gnueabihf.

Question 2. Quelles sont les différences entre ces trois chaînes de compilation croisée ?

Programmation : lpctools (ou autre)

Enfin, pour ce qui est de l'utilitaire permettant de charger (uploader) le code binaire sur le micro-contrôleur (flasher le micro-contrôleur), nous allons utiliser un utilitaire développé par Nathael Pajani, un ancien élève CPE qui a pris quelques heures de son temps pour coder un utilitaire permettant de programmer les micro-contrôleurs LPC, et placé le tout sous licence GPL v3. `lpctools` est désormais disponible sur le [dépôt GIT](http://git.techno-innov.fr/?p=lpctools) de Techno-Innov (<http://git.techno-innov.fr/?p=lpctools>), et est intégré depuis peu à la distribution Debian GNU/Linux.

Entre-temps d'autres projets permettant de programmer des micro-contrôleurs LPC sont apparus, mais on ne les a pas encore testés (nxpprog - licence MIT, mxli - GPLv3, pyLPCTools - GPLv2). Un autre paquet Debian fournit les outils permettant de programmer les micro-contrôleurs LPC de NXP : `lpc21isp` (qui dispose d'ailleurs d'une interface graphique : GLPC, qui elle n'est pas dans les dépôts). Voir liens en bas de page.

La VM `debian_aiot` a déjà l'utilitaire `lpctools` installé, toujours pour éviter les surprises pendant le TP. Ce paquet fournit deux utilitaires (`lpcisp` et `lpcprog`) et une page man pour l'utilitaire `lpcprog` qui est celui que nous utiliserons.

Remarque. Comme vous n'avez pas les permissions nécessaires pour installer des paquets dans les machines CPE, vous pouvez aussi télécharger les sources de `lpctools` et les compiler pour l'utiliser depuis votre espace de travail.

Test de l'environnement de développement

On va commencer par tester l'environnement de développement avant de passer à la suite du TP.

Exercice 3. Test du module

Commencez par vérifier que votre module fonctionne (rien de plus frustrant que d'essayer de programmer un micro-contrôleur qui ne réponds pas).

Les modules ont été programmés avec un firmware qui affiche le nom du module et quelques informations *sur la liaison série du module*.

Dans la VM, l'utilitaire `minicom` est déjà installé, et il permet de récupérer ces messages. (La configuration de `minicom` est expliquée [sur la page concernant le système de développement](http://www.nathael.org/wiki_cpe/index.php/Dev/Host#Minicom) http://www.nathael.org/wiki_cpe/index.php/Dev/Host#Minicom).

Remarque. Le port USB de votre machine normalement apparaît sous le nom `/dev/ttyUSB0`, pour pouvoir l'utiliser dans ce TP, il est lié à la VM `debian_aiot` directement, il apparaît sous le nom `/dev/ttyS3` dans votre VM.

Depuis votre VM testé la connexion avec `minicom` :

```
minicom -ow -D /dev/ttyS3
```

Vous devriez voir les messages suivants s'afficher :

```
Module RF-SUB 1GHz.  
CPE Lyon.
```

Remarque. Pour sortir de `minicom` utiliser le raccourci `Ctrl+A`, puis `Z` et puis `Q`

Exercice 4. Code source et compilation

Le code source correspondant au support du module RF Sub-1GHz est disponible sur le GIT techno-Innov ou sur le compte github de CPE. Utilisez la commande suivante pour cloner le dépôt contenant le code :

```
git clone https://github.com/CPELyon/modules-techno-innov modules  
cd modules/
```

Le code est alors présent dans le dossier "modules". Ce dossier contient le code de gestion du micro-contrôleur (cœur (core) et modules (drivers)), plusieurs drivers pour des périphériques externes (extdrv), et des applications de démo (apps) pour différentes cartes utilisant le même micro-contrôleur.

Les applications de démo pour notre module sont dans le dossier "apps/base".

Certaines applications présentes dans ce dossier nécessitent du matériel supplémentaire, mais l'application `i2c_temp` utilise uniquement le capteur de température et l'interface USB-to-UART présents sur le module.

Pour tester cette application, il faut la compiler et envoyer le binaire sur le module. Pour compiler, le plus simple est d'utiliser le Makefile présent dans le dossier de l'application :

```
cd apps/base/i2c_temp/  
make
```

Mettre le code sur le micro-contrôleur

Ensuite, utilisez `lpcprog` pour envoyer le binaire sur le module (dans la flash du micro-contrôleur). Pour cela le module doit être branché sur un des ports USB de votre poste de développement (relativement évident), mais aussi être en mode "programmation" (ISP).

Exercice 5. Mode ISP

Pour mettre le micro-contrôleur en mode programmation (ISP : "In System Programming" en anglais), le chapitre 20 de la documentation du micro-contrôleur (LPC122x Flash ISP/IAP) indique qu'il faut maintenir la pin 12 du port 0 (broche 27) à l'état bas (0V) pendant au moins 3ms après avoir relâché le signal "Reset"

(pin 13 du port 0, broche 28). La suite du chapitre décrit le protocole de programmation, uniquement utile si vous voulez créer votre propre utilitaire pour programmer le micro-contrôleur, ou si vous voulez réaliser des opérations très spécifiques.

Puisque il y a des boutons poussoir reliant ces signaux à la masse, avec des résistance de "pull-up", cette opération est très simple : il faut appuyer sur les deux boutons en même temps, puis relacher le bouton reset avant de relacher le bouton ISP. Lorsque tout c'est bien passé, la led bicolore doit avoir deux petits points lumineux à peine visibles dans l'obscurité (un rouge et un vert).

Exercice 6. Dialogue avec le micro-contrôleur

La suite se passe sur le PC. Le paquet `lpc tools` contient deux binaires : `lpcisp` et `lpcprog`. Le premier donne accès aux opérations élémentaires du protocole de programmation, et ne nous sera pas utile. Le second permet de programmer le micro-contrôleur en utilisant une unique commande, bien que d'autres commandes permettent d'effectuer quelques opérations intéressantes.

Nous allons d'ailleurs commencer par une de ces autres opération : demander les identifiants de notre micro-contrôleur avec la commande `"id"`. La syntaxe des commandes `lpcprog` est simple, il suffit de lui passer un nom de device, que l'on passe comme argument de l'option `"-d"`, une commande (argument de l'option `"-c"`), et si besoin le nom du fichier à utiliser. Dans l'exemple suivant le module RF Sub-1GHz est identifié sur le poste de développement comme périphérique `"ttyS3"`, nous utiliserons donc le device `"/dev/ttyS3"`.

```
$ lpcprog -d /dev/ttyS3 -c id
Part ID 0x3640c02b found on line 26
Part ID is 0x3640c02b
UID: 0x2c0cf5f5 - 0x4b32430e - 0x02333834 - 0x4d7c501a
Boot code version is 6.1
```

La première ligne nous indique que `lpcprog` a reconnu le micro-contrôleur et qu'il sera donc possible de le programmer. (Si ce n'est pas le cas, vous devrez compléter le fichier de description des micro-contrôleurs). Ensuite, `lpcprog` nous donne les informations propres au micro-contrôleur, à savoir son identifiant unique et la version du "boot code" qui se trouve en rom sur le micro-contrôleur.

Si vous avez un affichage similaire, tout va bien. Sinon, vérifiez la connexion et que le micro-contrôleur est bien en mode ISP.

La programmation se fait ensuite très simplement avec la commande `"flash"`, en ajoutant le nom du fichier binaire à envoyer.

```
$ lpcprog -d /dev/ttyS3 -c flash i2c_temp.bin
Part ID 0x3640c02b found on line 26
Flash now all blank.
Checksum check OK
Flash size : 32768, trying to flash 8 blocks of 1024 bytes : 8192
Writing started, 8 blocks of 8192 bytes ...
```

`lpcprog` se charge d'effectuer toutes les opérations nécessaires pour mettre en place notre binaire, à savoir d'effacer la flash, de générer la somme de contrôle de l'entête et de la placer au bon endroit dans le binaire, et d'envoyer le tout au micro-contrôleur pour mise en flash.

Exercice 7. Vérification

Le code que l'on vient de mettre sur le micro-contrôleur passe son temps à faire des relevés de température et à les envoyer sur la première ligne série du micro-contrôleur.

Pour tester, il faut sortir le micro-contrôleur du mode de programmation (appuyer sur le bouton reset), et lire les messages qui arrivent sur la ligne série, avec `minicom`, exactement comme vous l'avez fait pour vérifier que la communication avec le module fonctionnait.

Vous devez voir défiler des lignes similaires aux lignes suivantes :

```
Temp read: 29,5 - raw: 0x1d80.
Temp read: 29,5 - raw: 0x1d80.
```

Exercice 8. Interaction avec les LEDs embarquées

Regardez le code du module pour la lecture de la température (fichier `main.c`) et modifiez-le pour changer la couleur de la LED, lorsque la température dépasse un seuil de 28 degrés. Le code que vous allez modifier est adapté à un autre module Techno-innov et donc vous devez adapter les numéros de GPIO selon ceux décrits dans la documentation technique du Sub 1GHz :

```
const struct pio status_led_green = LPC_GPIO_1_4;  
const struct pio status_led_red = LPC_GPIO_1_5;
```

Liens outils

Lien vers d'autres outils pour programmer les LPC :

- mxli : <http://www.windscooting.com/softy/mxli.html>
- lpc21isp : <http://sourceforge.net/projects/lpc21isp/>
- nxpprog : <http://sourceforge.net/projects/nxpprog/>
- GLPC (GUI pour lpc21isp) : <http://sourceforge.net/projects/glpc/>