CPE Lyon - Formation Continue - Année 2018/19

Fondamentaux des Sciences du Numérique pour les Chimistes

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.

Exercice 2. Documentations techniques

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

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 module GPIO-Démo 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 fabriquant 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 fourni les bonnes documentations), mais pour les micro-contrôleurs il est préférable de consulter le site du fabriquant car il fourni 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 point apprécié 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).

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 les clés USB fournies. 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 chaines de compilation pour ARM : gcc-arm-linux-gnueabi et gcc-arm-linux-gnueabih.

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), Nathael n'a pas trouvé d'outil libre permettant de réaliser cette étape au moment où il a étudié la possibilité d'utiliser les micro-contrôleurs de NXP, mais le protocole série permettant de réaliser cette opération étant documenté dans la documentation technique des micro-contrôleurs, il ne devait donc pas y avoir de problème de ce côté là.

La seule solution "gratuite" qu'il avait trouvé à l'époque ne fonctionnait que sur un seul système (pas le sien), ne fournissait pas ses sources, et interdisait une utilisation commerciale sans payer une licence, hors il voulait justement en faire une utilisation commerciale.

Il a donc 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 de Techno-Innov, 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 permetant 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).

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 :

```
RF: ret:6, st: 8.
CC1101 RF link init done.
```

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. Utilisez la commande suivante pour cloner le dépôt contenant le code :

```
git clone http://gitclone.techno-innov.fr/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 lpctools 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 dégrées.

Exercice 9. Feu de circulation

FIGURE 1 – 1 led rouge, 1 led orange, 1 led verte et ses résistances

Pour cet exercice vous allez récupérer 3 LEDs (rouge, orange, verte), 1 résistance, et 1 breadboard, voir la Figure 1.

D'abord vous allez faire un montage des LEDs et la résistance dans votre breadboard. Pour ceux qui n'ont pas encore travaillez avec un breadboard, vous pouvez consulter la guide sparkfun sur `https://learn.sparkfun.com/tutorials/how-to-use-a-breadboard`.

Le montage des LEDs et résistance vous pouvez le faire à votre préférence, une suggestion de montage est montrée dans la figure 2

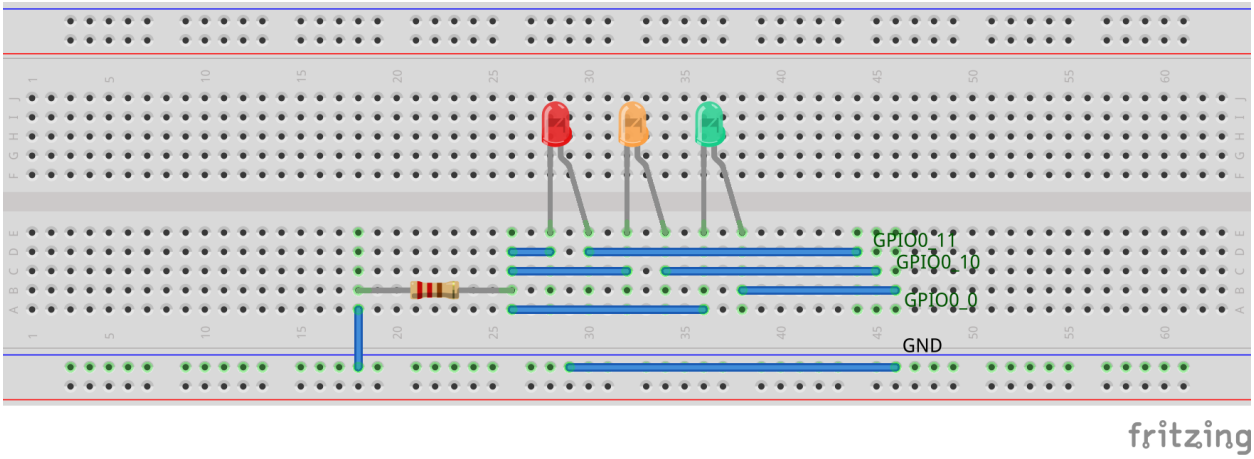

FIGURE 2 – Exemple montage feu

Du point de vue de notre micro-contrôleur, allumer ou éteindre une Led revient à changer l'état de la sortie correspondante. Pendant le TP 1, vous avez utilisez les sorties GPIO connectés en internet aux LEDs du module (`LPC_GPIO_0_28` et `LPC_GPIO_0_29`)

Pour cet exercice, vous allez programmer le comportement d'un feu de circulation. En utilisant des temporisateurs comme la fonction `msleep(t)` en C, le but de cet exercice sera de simuler le changement de l'état d'allumages des LEDs. Pensez bien à garder l'ordre d'allumage, vous pouvez utiliser la structure `switch/case`.

Exercice 10. LED RGB Neopixel

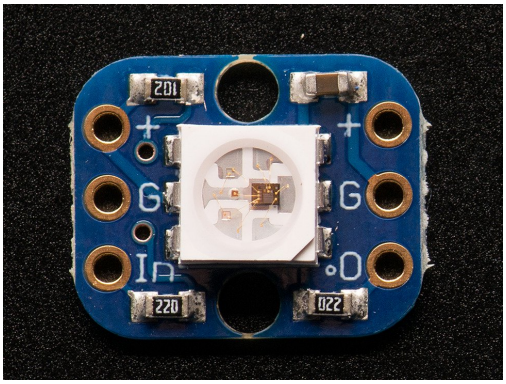

FIGURE 3 – 1 Led RGB

Pour cet exercice vous allez récupérer le matériel demandé dans la Figure 3.

A partir des exemples présents dans le code du module (dispo sur git), créez le support d'une led RGB de type Neopixel dans votre nouvelle application, ce type de LED permet de connecter plusieurs LEDs en chaîne. La LED doît être connectée de cette façon :

— Pin `+` à la sortie `+3.3V` du micro-contrôleur
— Pin `G` à la sortie `GND` du micro-contrôleur
— Pin `I` à une sortie `GPIO`, dans le code donné ça sera le `GPIO0_19`.

Pour connecter plusieurs LEDs en chaîne vous devez connecter le pin `O` de la première LED au pin `I` de la LED suivante et les pins `+` et `G` entre les deux LEDs.

Le code pour gérer ce type de LED est donné dans l'exemple `/apps/base/ledstrip/`, prennez le temps de bien comprendre le code et le tester avec une LED. Modifiez ce code pour itérer entre les couleurs bleu, blanc, rouge à une duréé de 250ms. Puis si vous avez le temps, demandez à vos encadrants une deuxième/troisième LED selon la disponibilité de matériel et tester avec plusieurs LEDs.

Exercice 11. Capteurs météo

Pour cet exercice vous allez récupérer le matériel demandé dans la Figure 4. Cette puce regroupe plusieurs capteurs environnementaux que vous allez devoir interroger pour obtenir leurs valeurs. Ces valeurs seront renvoyés par le port série et lues avec `minicom`.

Pour ce faire, vous disposez du code contrôleur sur le git suivant : https://github.com/CPELyon/modules-techno-innov. Attention, ce git et différent du git techno-innov!!!. Le code est dans le sous-dossier `/apps/rf_sub1G/sensors/`.

Le module *sensors* a une interface I^2C pour pouvoir accéder à ses données, la distribution des pins est présenté dans la figure 4.

Exercice 12. Écran mono-couleur

Pour cet exercice vous allez récupérer le matériel demandé dans la Figure 5.

Dans cette exercice le but est de pouvoir afficher sur l'écran la valeur de la température obtenue à partir du capteur intégré au module RF Sub-1GHz ainsi que les noms des intégrants de votre binôme.

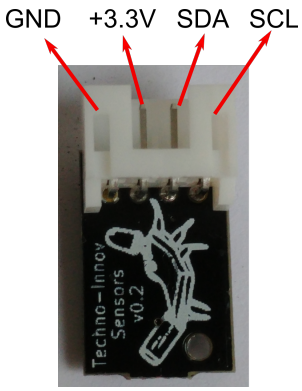

FIGURE 4 – Capteur météo *sensors*

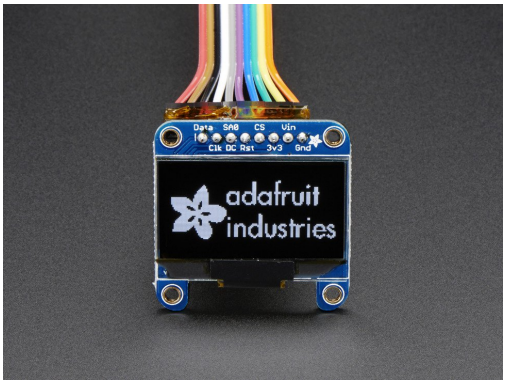

FIGURE 5 – 1 écran mono-couleur

Avant d'afficher directement les informations, assurez vous de comprendre comment afficher des informations sur cet écran. Pour ce faire, vous disposez du code contrôleur d'un écran OLED Adafruit sur le git : https://github.com/CPELyon/modules-techno-innov. Le code est dans le sous-dossier `/apps/rf_sub1G/oled/` est vous devez l'adapter selon les connecteurs de votre micro-contrôleur.

Dans le code donné, on utilise l'interface I^2C pour pouvoir envoyer les informations à afficher, on positionne les données en indiquant la ligne à utiliser, la distribution des pins est présenté dans la figure 6.

Pour finir, affichez aussi la température obtenue à partir du capteur *sensors* pour la comparer à celle intégrée au micro-contrôleur.

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/
- Neopixel : https://learn.adafruit.com/adafruit-neopixel-uberguide/the-magic-of-neopixels
- Écran mono-couleur : https://www.adafruit.com/product/938

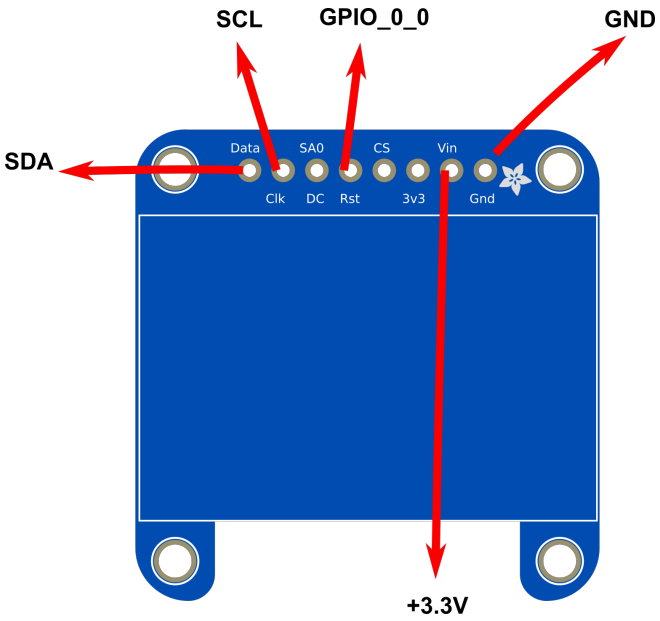

FIGURE 6 – Disposition des pins du module *OLED*