IoTize your Arduino

On répète qu’il ne faut que quelques minutes pour ‘Iotizer’ une application existante à base de Cortex-M et quelques heures pour toute autre architecture. Mais dans le cas des applications Arduino, l’intégration est facilitée par la notion de ‘bibliothèques’.

Cette note d’application est un guide montrant comment profiter d’un module de connectivité TapNLink sur une application basée sur Arduino.

# Les différents types de cartes ‘Arduino’

Arduino est une famille avec laquelle il faut composer. Les différents modèles de cartes sont basés non seulement sur des microcontrôleurs différents mais sur des architectures de cœurs différentes. En pratique, TapNLink supporte tout microcontrôleur et toute architecture. Les modes d’utilisation et les fonctionnalités varieront cependant selon la carte, et plus exactement selon le processeur et la tension d’alimentation de ses entrées/sorties. Prenons par exemple trois cartes très populaires avec lesquelles nous avons testé TapNLink :

|  |  |  |
| --- | --- | --- |
| Model | Processor | IOs Voltage |
| Uno | AVR (**ATmega328)** | 5V |
| Due | Cortex-M (ATSAM3X8E) | 3V |
| Mega | AVR (ATmega2560) | 5V |

Il nous a fallu faire quelques adaptations :

* Pour les cartes à base de Cortex-M, on peut utiliser indifféremment une connexion SWD ou S3P. SWD est le protocole de debug du cœur. Il apporte :
  + Une utilisation immédiate, sans même à avoir à modifier le firmware…
  + Des fonctionnalités avancées comme la reprogrammation depuis le smartphone.

S3P apporte juste plus de sécurité, ce qui n’est pas fondamentalement notre souci dans un premier temps. Donc pour une carte à base de Cortex-M on préfèrera souvent débuter avec l’accès SWD.

* Pour les cartes à base d’AVR, il nous a fallu ajouter une bibliothèque ‘S3P’ et recompiler le firmware.

Pour l’ensemble de ces cartes, on pourrait encore utiliser une liaison série (UART) classique et fonctionner en modbus. Des sources sont disponibles en tant qu’exemple, mais ils n’ont pas à ce jour été adaptés pour l’environnement Arduino. Il faut donc choisir comme protocole target :

* Soit S3P indexé,
* Soit SWD (pour les Cortex-M seulement).

Enfin, pour les cartes dont la tension d’alimentation (surtout celle des IOs) est 5V, il a fallu adapter les niveaux de tensions (par l’ajout d’une simple résistance en série pour nos tests).

Cette note d’application traitera donc ces différents cas.

# Connexion filaire entre Arduino et TapNLink

Que l’on soit en SWD ou en S3P, et quelle que soit la carte Arduino, 4 fils sont nécessaires pour porter les signaux suivants :

|  |  |  |
| --- | --- | --- |
| Type | Name | Description |
| **Power** | Vcc | **MUST BE 3V or 3.3V** |
| Gnd | Ground |
| **Digital** | Clock | Must be an interrupt (or could be SWD-CLK for Cortex-M devices) |
| IO |  |

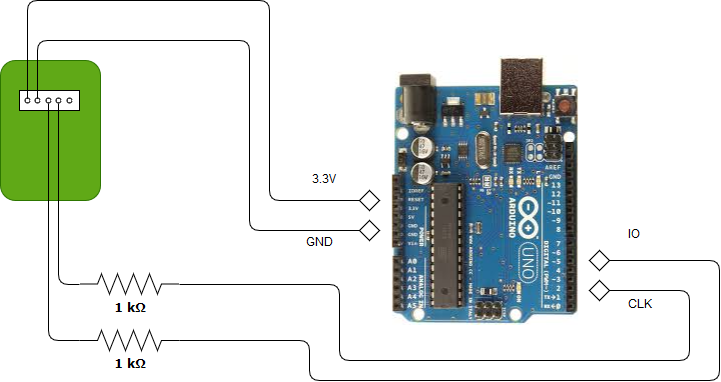
Donc deux pour fils pour l’alimentation (GND et Vcc3.3) et deux signaux logiques : une clock et un signal de données auxquels on peut optionnellement ajouter le signal de reset si l’on souhaite que TapNLink soit capable de resetter à la carte applicative.

**ATTENTION :**  Avec une carte 5V, c’est bien **l’alimentation 3.3V** qu’il convient d’apporter le module TapNLink. **DO NOT CONNECT 5V**, it will destroy your Tap!

## Connexion avec une carte 5V (UNO, MEGA, ..)

A nouveau, bien choisir son alimentation : il faut connecter **la source 3.3V** de la carte et non l’alimentation générale 5V. Mais il convient aussi d’adapter les signaux logiques car les pattes du TapNLink ne supportent pas non plus une tension de 5V. Pour cela, le plus simple consiste à intercaler une résistance 1ko (mille ohm) en série entre les deux cartes. La chute de tension sera suffisante pour limiter le courant et ne pas endommager TapNLink lorsqu’un signal haut est sorti depuis la carte Arduino. Elle sera aussi suffisamment faible pour ne pas ralentir les fronts de montée, et assurer des niveaux de tension convenable de part et d’autre.

Le schéma suivant résume ces branchements :



Dans le cas particulier de l’Arduino Uno, le signal IO peut être connecté sur n’importe quelle entrée sortie digitale alors que le signal CLK doit être connecté sur une patte d’interruption externe. Deux pattes seulement sont possibles : 2 ou 3.

Dans l’exemple fourni avec la bibliothèque, CLK est relié à la patte 3 alors que IO est relié à la patte 5.

## Connexion avec une carte à base de Cortex-M (Due)

Si le protocole SWD est utilisé, le plus simple est de le connecter directement au port de debug du processeur :

# Configuration de l’IDE-Arduino

## Fichier de sortie (ELF)

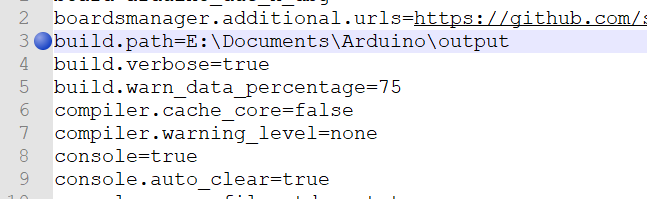
Pour IoTize Studio, nous avons besoin pour chaque projet du fichier exécutable (associé au suffixe ‘.ELF’) généré par l’IDE Arduino. Or l’IDE considère ce fichier comme un fichier de travail qu’il place par défaut dans un répertoire temporaire, peu accessible. On peut cependant modifier cet emplacement temporaire et le remplacer par un endroit plus accessible.

Pour cela, il faut ouvrir le fichier ‘Preferences.txt’ Redirection par ‘préférences’ :

Menu Fichier | Preferences | ‘More preferences can be edited…’

Avant d’éditer ce fichier, il faut fermer Arduino IDE (qui l’écrase à chaque fermeture) car l’on perdrait sinon les modifications.

Ajouter une ligne ‘build.path=…’ en spécifiant après ‘=’ {sketchbook\_directory\output}. Dans mon cas, mon répertoire ‘sketchbook’ est E:\Documents\Arduino :



Sauver ce fichier, et les fichiers générés par le compilateur GCC de l’IDE Arduino seront désormais placés dans ce répertoire, non loin de nos projets (sketch) Arduino.

## Installation de la bibliothèque TAP

## Modification d’un projet existant

Structure des répertoires

# Configuration du module

## Installation d’IoTize Studio

## Les choix de configuration

## Sélection et affichage de variables

# Test

## Monitoring local

## Publication de l’Apps mobile

## Depuis un mobile…

# Pour aller plus loin…