Chapitre 1

Contexte

Le Moteino est un ordinateur de très petite taille et peu cher. Il est basé sur l'Arduino, très modulaire. Les Moteino ont une puce mémoire EEPROM, un port série, une puce réseau RFM69.

Le but de ce projet est de prendre en main l'architecture embarquée du Moteino : produire du code C++ à installer sur un Moteino, mais surtout comprendre comment manipuler correctement cette architecture. La production de code C++ demande l'utilisation d'outils particuliers. Arduino propose un IDE minimaliste qui ne convient pas aux projets industriels, nous utilisons donc platformio pour automatiser la gestion des librairies, compiler et envoyer le code sur les chipsets.

Chapitre 2

Liens utiles

le git du projet
Le site de platformio

Chapitre 3

Organisation du projet

3.1 Code source

Le répetoire src/ contient trois répertoires :

src/main contient les sources principaux à installer sur les chipsets. Chaque répertoire contient un module à installer sur un Moteino.

src/lib contient les librairies que nous développons, utilisées dans les main.

src/ext contient le librairies que nous avons importées, après d'éventuelles modifications, depuis d'autres sites.

src/test Contient les tests (pas utilisé pour l'instant)

Le répertoire .tmp/ contient entre autre les dépendances gérées par plateformio. Il peut etre supprimé sans danger.

3.2 Documentation

Le répertoire doc/ contient la documentation de ce projet, en particulier ce fichier et le code permettant de le générer.

3.3 Outils de développement

diffrents outils sont utlisés pour s'assurer du fonctionnement de ce projet.

3.3.1 Gestionnaire de version : git

Présent par défaut sous ubuntu.

Pour télécharger le git du projet à des fins de test il suffit de faire
`git clone https://github.com/glelouet/Moteino.git`

Pour modifier le projet il faut par contre utiliser la connexion git en ssh, par exemple
`git@github.com:glelouet/Moteino.git`

3.3.2 outils en ligne de commande : pio

Platformio met à disposition une commande pour compiler automatiquement le code, pio.

Une fois le code téléchargé, il suffit de faire `pio -t upload` dans un répertoire de main pour déployer les binaires sur les chipsets.

3.3.3 Interface graphique : Atom/Platformio

Utilisant pio pour le déploiement en ligne de commande, il est pratique d'utiliser l'IDE correspondant qui a une intégration native avec les fichiers de configuration. Cependant d'autres IDE peuvent surement etre utilisés.

Éditeur Atom

Atom est un éditeur de texte hautement modulaire. Télécharger atom sur le site officiel

Plug-in Platformio

Le plug-in Plaformio pour Atom permet le déploiement rapide depuis l'éditeur des binaires sur les chipsets.

Pour l'installer suivre le site officiel.

Une fois installé, il faut ouvrir le projet avec `file/open project folder` et entrer un des répertoires présents dans src/main. Pour installer ce projet il suffit de cliquer à gauche sur la flêche de légende `platformio:upload` . si l'upoad ne fonctionne pas, il faut s'assurer que le `upload_port` du fichier platformio.ini correspond bien au port USB du moteino. En cas de multiple pors USB ceux-ci peuvent varier.

Chapitre 4

Programmation du Moteino

Le moteino est mono-thread et basé sur une boucle de gestion. Un code destiné au Moteino, ou plus généralement à un Arduino, doit avoir deux fonctions :

1. une fonction d'initialisation des élements appelée `init()`.
2. une fonction utilisée périodiquement appelée `loop()`

Si en c++ l'initialisation des objets peut être exécutée dans leur constructeur, dans l'arduino le code peut être chargé alors que les puces ne sont pas toutes prêtes. La fonction init() garantit une execution correcte.

Chaque fichier source dédié à être injecté dans un moteino doit donc avoir ces deux méthodes, et en particulier les librairies doivent être initialisées dans l'appel à l'init().

Les interruptions sont gérées par le chipset intégré et donc transparentes pour l'utilisateur.

4.1 Principe de code

Les librairies sont des éléments de code qui peuvent être partagées. Elles représentent généralement un élément physique du moteino, ou un protocole. Un fichier source à injecter dans un moteino peut utiliser des librairies, il est responsable de les initialiser durant son init() et de les appeler durant son loop().

Étant donné que le Moteino est mono-thread, il ne faut surtout pas avoir d'appel à sleep() dans le code des librairies, que ce soit dans le main ou dans les librairies. En effet un appel à sleep() bloque tout le système, les protocole réseaux auront donc des timeout. Il vaut mieux à la place enregistrer le temps de la prochaine "sortie de veille" et ne rien faire dans les fonctions loop() tant que ce temps n'est pas atteint.

La création d'un binaire pour Arduino demande donc la création d'un code contenant une fonction init() et une fonction loop(), la fonction init() étant responsable d'initialiser les éléments du chipset, la fonction loop() responsable d'appeler périodiquement (ou d'utiliser) ces éléments.

4.2 Les librairies présentes

Deux types principaux de librairies sont présentes dans le répertoire src/lib. Les librairies de drivers codent l'accès en dur aux élements du moteino. Les librairies de gestion codent les protocoles logiciels. Dans certains cas le driver est fourni dans le même répertoire que la librairie de gestion.

4.2.1 Moteino

Ce driver propose l'accès aux chipset généralement intégrés au Moteino.

Le port série est accessible pour transfert de données via/depuis un autre PC.

Le Moteino spécifie la structure de ses paramètre et stocke ceux-ci dans l'EEPROM, il les charge si possible à l'initialisation. En particulier les paramètres réseau y sont stockés, ce qui permet de conserver le réseau lors du redémarrage des moteinos.

Le Moteino étant équipée d'une LED, des fonctions basiques de manipulation de celle-ci sont mises à disposition : allumer/éteindre, clignoter à une fréquence donnée ou un nombre de fois spécifique.

4.2.2 Lib RFM69

La puce RFM69 est intégrée à la plupart des moteino. La librairie rfm69 est en dépendance dans platformio pour apporter les drivers nécessaires à l'utilisation de cette puce.

La librairie RFM69Manager a pour rôle d'assurer l'utilisation de cet élément. Elle gère la découverte et l'association à un réseau ad-hoc.

Puce RFM69

La puce par défaut permet de spécifier un numéro de réseau (0-255) et un identifiant propre sur ce réseau, ainsi qu'une clé de cryptage. Le réseau ad-hoc est défini par l'ensemble des éléments partageant ce couple (num, clé).

Une fois le réseau spécifié, la puce peut envoyer sur ce réseau, que ce soit à un élément du réseau en spécifiant son identifiant, ou en broadcast à tous les éléments du réseau. Dans un cas comme dans l'autre, tous les éléments physiques du réseau reçoivent les paquets, mais un filtrage dans la puce ignore les paquets reçus qui ne sont pas destinés à celle-ci. Les éléments du réseau peuvent être mis en mode "promiscuous" pour accéder à tous les paquets du réseau.

Un mode d'envoi "synchrone" ré envoie le même paquet plusieurs fois tant que la puce n'a pas reçu un paquet spécifique d'acquittement.

À l'extinction de la puce, les données de configuration sont perdues : elles doivent donc etre stockées, dans le code ou dans la ROM, ou utiliser un protocole de découverte. La libirairie RFM69Mananger se charge de cette dernière opération.

Mode découverte

En mode de découverte réseau, le manager se connecte à un réseau spécifique, désactive l'encryption, se met en mode promiscuous, puis envoie régulièrement une trame de demande d'information réseau en broadcast.

Une fois une réponse reçue, il traduit cette réponse en spécification réseau et se connecte à ce réseau. Cette spécification est enregistrée dans la mémoire du moteino, ausi au redémarrage le manager repassera directement en mode connexion au réseau.

Mode connexion au réseau

Une fois les paramètres d'un réseau connu, la puce doit s'attribuer un numéro de réseau, si celui-ci n'est pas spécifié dans le code/la mémoire. Pour cela, le manager se met en mode promiscuous puis itère sur chacune des adresses possible. Pour chaque adresse, il lui envoie un message avec acquittement. S'il ne reçoit pas de réponse, soit cette adresse est libre, soit il y a une autre puce qui tente de l'acquérir.

Pour éviter les risques de collision lors de la recherche d'adresse IP, après avoir déterminé une adresse intéressante le manager attend entre 0 et 1s puis renvoie un message avec acquittement. En cas de non-réponse, il s'attribue cette adresse ; Sinon il passe à l'adresse suivante.

Une fois l'identifiant sur le réseau déterminé, le manager connecte la puce à cet identifiant et passe en mode transmission

Mode transmission

Dans ce mode, le manager écoute les trames sur le réseau qui lui sont destinées et est disponible pour l'envoi de trames, que ce soit en mode synchrone ou asynchrone, avec ou sans broadcast. Deux cas particulier sont traités.

Dans le cas où il reçoit un message de découverte d'IP réseau, le manager renvoie un acquittement. Cela permet à un nouvel arrivant sur le réseau de savoir que cette adresse est déjà réservée.

Dans le cas où un message provenant de lui même est découvert, cela signifie qu'il y a eu une collision lors de l'attribution de l'identifiant sur le réseau. Le manager repasse alors en mode connexion au réseau, sauf si ce réseau est inscrit dans les paramètres de la mémoire.

Mode appareillage

Ce mode permet à un Moteino de connecter des moteinos sur son réseau. Dans ce mode le manager n'a plus visibilité sur son réseau, afin de répondre aux messages des moteinos en mode découverte. Il attend donc un certain temps (par défaut 1min) , et envoie les informations de connexion au réseau aux éléments le demandant. Une fois le délai d'appareillage atteint, il repasse en mode connexion.

4.2.3 Lib ButtonCommand

La librairie Button écoute sur un pin la tension (0 ou 1) présente à chaque loop(). Elle enregistre les variations sous la forme d'une série de durées d'appui. Lorsqu'un relachement suffisamment long (3s) est fait, cette série est mise à disposition pour un autre module.

La librairie ButtonCommand quant à elle traduit les séquences d'appuis en actions à effectuer sur le Moteino, en fonction du nombre et de la durée des appuis.

2 appuis démarrage du mode pairing

3 appuis extinction du mode pairing

un appui court moins de .5s, demande aux éléments sur le même réseau de clignoter

un appui moyen entre .5 et 3s, tente de se connecter à un moteino en mode "pairing"

un appui long entre 3 et 10s acquiert un réseau aléatoire et se met en mode "pairing"

4.2.4 Lib SerialShell

Cette librairie lit à chaque loop() les données du port serial et les interprete sur le moteino :

blink A fait clignoter la LED avec un délai de A ms

burn A envoie régulièrement un message sur le réseau durant A ms

dbg=A met le niveau de débugage du moteino à A (inutilisé)

flash A fait clignoter la LED en continu avec un délai de A ms

load recharge les paramètres du moteino depuis la mémoire

pairon met le RFM69Manager en mode appareillage

pairoff quitte le mode appareillage du RFM69Manager

pbp=A modifie la période d'envoie de message sur le réseau à A ms

rdip=A met l'adresse IP de la RFM69 à A

rdnet=A met le réseau de le RFM69 à A

rdran demande au Moteino de choisir un réseau aléatoire, une clé de cryptage aléatoire, puis de se connecter sur ce réseau.

rdsip demande au manager de se connecter au réseau actuel

rdsnet demande au manager de découvrir un réseau

write enregistre les paramètres actuels du moteino dans la mémoire