N°8 | SEPT. - OCT. 2015

HACKABLE MAGAZINE

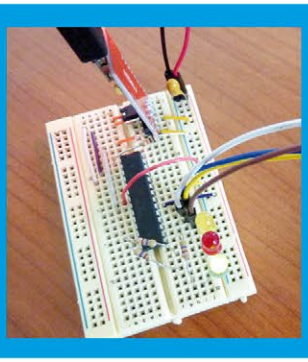
DÉMONTEZ | COMPRENEZ | ADAPTEZ | PARTAGEZ

France METRO : 7,90 € – CH : 13 CHF – BEL/PORT.CONT : 8,90 € – DOM TOM : 8,50 € – CAN : 14 $ cad – TUNISIE : 18 TND – MAR : 100 MAD

ÉCONOMIES

Construisez votre programmeur pour quelques euros et sans carte Arduino

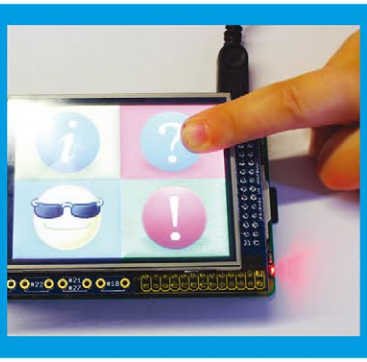
p. 14



INTERFACES

Exploitez un petit écran tactile pour Raspberry Pi avec Python et Pygame

p. 84



ARDUINO

Passez des croquis à la programmation AVR pour optimiser vos projets

p. 24

TERMINAL

Utilisez la ligne de commandes de votre Raspberry Pi depuis un navigateur

p. 92

Raspberry Pi : Prenez le contrôle de votre habitation !

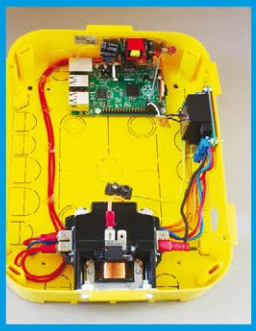
DOMOTIQUE « MAISON » !

- Créez un réseau de capteurs thermiques en Wifi
- Pilotez votre chauffage au fioul
- Surveillez votre consommation électrique

AUTOMOBILE

Chargez une voiture électrique avec du matériel ouvert et libre

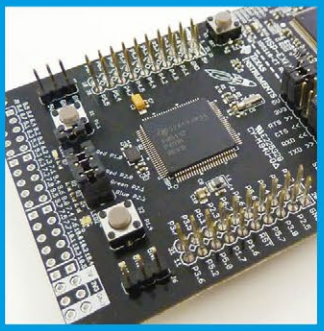
p. 76



ARDUINO & CO

Initiez-vous facilement au multitâche avec Energia/MSP432

p. 04



L 19338 - 8 - F: 7,90 € - RD

1ER ÉVÉNEMENT EUROPÉEN
LIBRE & OPEN SOURCE

OPEN FOR
INNOVATION
opensourcesummit.paris
#OSSPARIS15

PARIS OPEN SOURCE SUMMIT

18&19
NOVEMBRE

DOCK PULLMAN
Plaine Saint-Denis

PARTENAIRES INSTITUTIONNELS

SPONSORS PLATINUM

SPONSORS GOLD

SPONSORS SILVER

POUR TOUTE INFORMATION COMPLÉMENTAIRE :
Email : contact@opensourcesummit.paris – Tel : 01 41 18 60 52

un événement

coromonadalix

ÉDITO

C'est la rentrée !

Vous savez, cette étrange période où on sent que quelque chose démarre, mais aussi où on regrette le temps libre qui était le nôtre durant les congés d'été. Enfin, temps libre, c'est une façon de parler, car les esprits vifs que nous sommes vous et moi n'avons guère de temps vraiment « libre ». En lieu et place, nous avons une liste de choses à faire, alias « la fameuse *todo list* », avec son petit côté TARDIS (vous savez, elle est plus grande à l'intérieur et elle se réorganise à volonté).

Nous n'avons pas chômé cet été comme vous pourrez le constater dans les pages qui suivent. Pour autant, notre liste, qu'il s'agisse de la mienne ou celle des auteurs ayant contribué à ce numéro ne s'est pas pour autant réduite, au contraire. Chaque nouveau projet raisonnablement mené jusqu'à son objectif primaire fait apparaître une nuée de nouvelles pistes et de nouvelles idées. Et je ne parle que des projets qui sont suffisamment avancés pour faire l'objet d'un article. Car nous en avons une bonne quantité qui sont encore à l'état d'expérimentations amusantes, de codes trop « nocturnes » pour être publiés sans honte ou de notes et dessins griffonnés dans de multiples carnets à spirales (j'aime les carnets à spirales).

Mais à bien y réfléchir, l'objectif n'est pas tant de vider nos listes que de remplir la vôtre. Car le but premier des articles n'est pas réellement de vous inviter à reproduire les montages, mais plutôt de vous inciter à vous en inspirer pour produire vos propres évolutions. L'exemple du réseau de capteurs de ce numéro en est la parfaite démonstration. N'hésitez pas à nous faire part de vos réalisations, basées ou non sur les articles du magazine, via Twitter ou par e-mail, nous nous ferons un plaisir de les relayer à tous nos lecteurs. Vous aussi, montrez ce que vous faites et soyez certains que cela titillera la curiosité de bien des bidouilleurs !

Denis Bodor

Hackable Magazine

est édité par Les Éditions Diamond

LES ÉDITIONS DIAMOND
STIMULANT DE MATIÈRE GRISE

10, Place de la Cathédrale - 68000 Colmar
Tél. : 03 67 10 00 20 – Fax : 03 67 10 00 21
E-mail : lecteurs@hackable.fr
Service commercial : cial@ed-diamond.com
Sites : www.ed-diamond.com
Directeur de publication : Arnaud Metzler
Rédacteur en chef : Denis Bodor
Réalisation graphique : Kathrin Scali
Responsable publicité : Valérie Fréchard,
Tél. : 03 67 10 00 27 v.frechard@ed-diamond.com
Service abonnement : Tél. : 03 67 10 00 20

Impression : pva, Landau, Allemagne
Distribution France : (uniquement pour les dépositaires de presse)
MLP Réassort : Plate-forme de Saint-Barthélemy-d'Anjou. Tél. : 02 41 27 53 12
Plate-forme de Saint-Quentin-Fallavier.
Tél. : 04 74 82 63 04
Service des ventes : Abomarque : 09 53 15 21 77
IMPRIMÉ en Allemagne - PRINTED in Germany
Dépôt légal : À parution,
N° ISSN : 2427-4631
Commission paritaire : K92470
Périodicité : bimestriel
Prix de vente : 7,90 €

MIXTE
Papier issu de sources responsables
FSC® C015136

Suivez-nous sur Twitter
@hackablemag

À PROPOS DE HACKABLE...

HACKS, HACKERS & HACKABLE

Ce magazine ne traite pas de piratage. Un **hack** est une solution rapide et bricolée pour régler un problème, tantôt élégante, tantôt brouillonne, mais systématiquement créative. Les personnes utilisant ce type de techniques sont appelées **hackers**, quel que soit le domaine technologique. C'est un abus de langage médiatisé que de confondre « pirate informatique » et « hacker ». Le nom de ce magazine a été choisi pour refléter cette notion de **bidouillage créatif** sur la base d'un terme utilisé dans sa définition légitime, véritable et historique.

SOMMAIRE

ÉQUIPEMENT

04
LaunchPad MSP432 : en route pour le multitâche !

ARDU'N'CO

14
Construisez votre programmeur Arduino, sans utiliser d'Arduino

24
Arduino presque sans Arduino, ou comment programmer sans les roulettes

EN COUVERTURE

34
Assemblez un réseau de capteurs de température sans vous ruiner

48
Réseau de capteurs de température : le côté Pi

60
Contrôler sa chaudière à distance avec un Raspberry Pi

68
Supervisez votre consommation électrique sur Raspberry Pi

TENSION

76
Charger une voiture électrique avec du matériel libre

EMBARQUÉ & INFORMATIQUE

84
Exploitez un petit écran tactile pour Raspberry Pi avec Pygame

92
Une ligne de commandes dans un navigateur pour votre Pi 2

ABONNEMENT

13
Offres spéciales professionnels

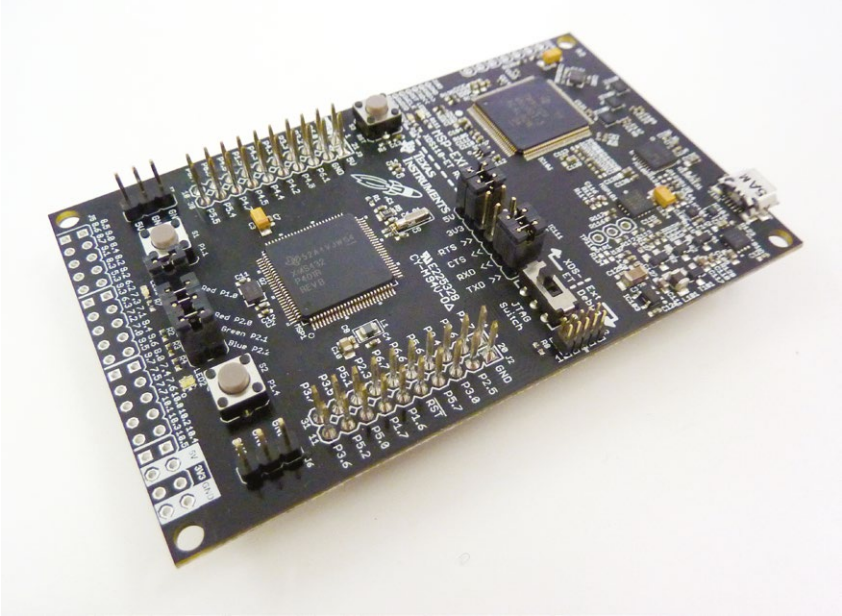
57/58
Abonnements tous supports

LAUNCHPAD MSP432 : EN ROUTE POUR LE MULTITÂCHE !

Denis Bodor

Classiquement, la programmation sur microcontrôleur est très différente de celle sur un ordinateur. La proximité avec le matériel, les outils utilisés, l'enregistrement des programmes sont autant de points où les différences sont évidentes. Le principe même de fonctionnement et d'utilisation des ressources est sans le moindre doute la caractéristique la plus notable. Pourtant ceci est en train de changer et la nouvelle carte proposée par Texas Instruments en est la parfaite démonstration…

coromonadalix



La carte LaunchPad MSP432P401R intègre tout ce qu'il faut pour procéder à ses premières expérimentations. On retrouve la logique habituelle des LaunchPad avec son emplacement deux fois deux rangées de 10 broches mâles/femelles pour les BoosterPack.

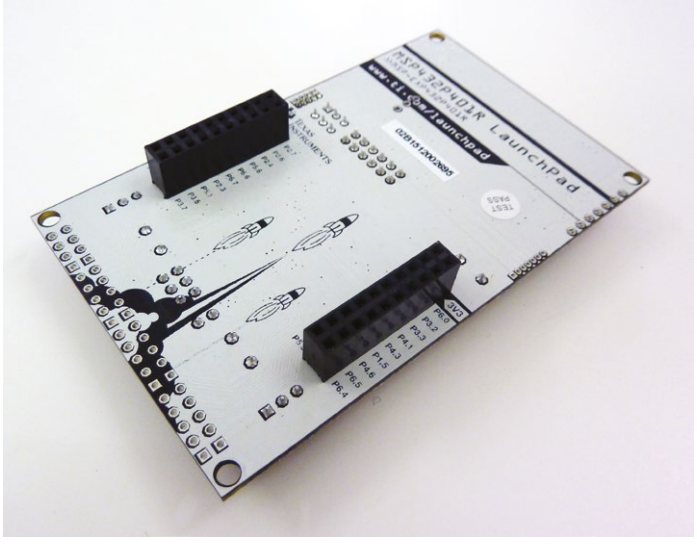
Nous avons déjà parlé des cartes de la famille Ti LaunchPad dans ce magazine (article sur la Tiva Connected dans le numéro 5). Elles forment le côté DIY/hobbyiste de la gamme proposée par le géant Texas Instruments. À l'instar de l'écosystème Arduino, les cartes LaunchPad permettent de faire connaissance avec certains microcontrôleurs de façon rapide et aisée. Cependant, quelques différences notables se doivent d'être soulignées :

- elles sont conçues et produites par Texas Instruments directement ;
- elles doivent être vues davantage comme des kits d'évaluation que comme des plateformes ludiques ;
- elles proposent deux approches différentes, l'une communautaire et l'autre professionnelle ;
- elles utilisent un environnement de développement similaire à celui d'Arduino, appelé Energia, mais son développement est communautaire et officiellement détaché de Ti (un peu comme Arduino vis-à-vis d'Atmel, mais Energia ne produit pas de carte) ;
- elles utilisent des modules additionnels appelés BoosterPacks et non shields.

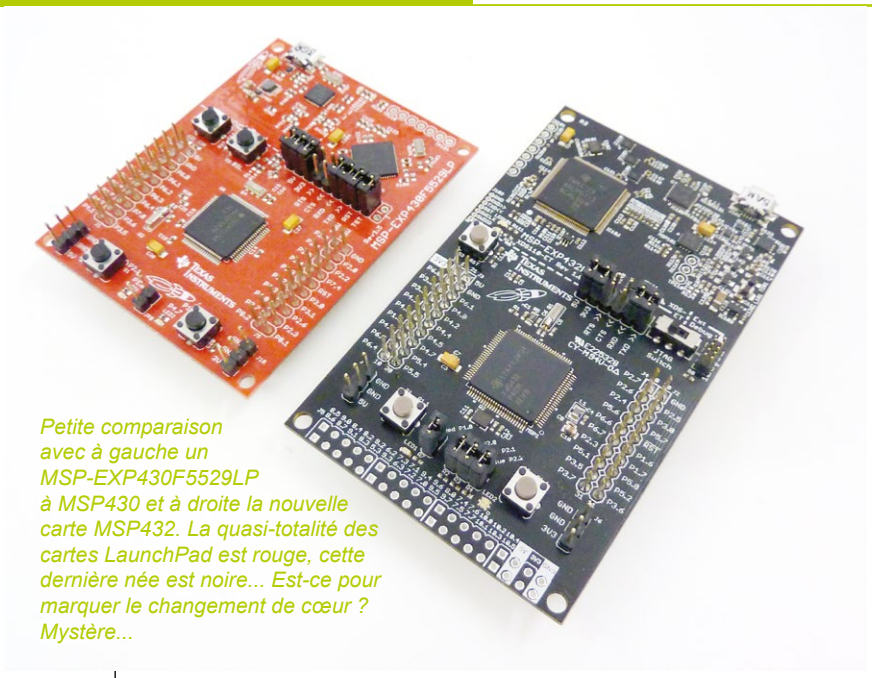
La gamme Launchpad se divise en 4 grandes familles : les MSP, les C2000, les Tiva et les Hercules. L'ensemble forme une collection de pas moins de 20 cartes différentes, proposant chacune des caractéristiques particulières, mais également un produit Ti à évaluer.

La nouveauté qui nous intéresse dans cet article a été introduite via l'apparition d'une nouvelle carte dans la famille MSP elle-même reposant sur un nouveau microcontrôleur : le MSP432. Le MSP430 dont il est le successeur est utilisé de longue date aussi bien par des constructeurs que par des amateurs éclairés. Il existe une importante communauté autour de ce composant, ou plutôt cette famille de microcontrôleurs. Le MSP430 est construit autour d'un processeur 16 bits (20 selon les modèles) conçu par Ti (l'AVR des Arduino Uno par exemple est 8 bits), mais l'arrivée du MSP432 change totalement la donne.

En effet, ce nouveau microcontrôleur troque le processeur Ti contre un ARM Cortex-M4F, 32 bits, à 48 Mhz. Rappelons qu'ARM ne fabrique rien, mais



Pas grand-chose sous la carte si ce n'est un beau dessin et le connecteur BoosterPack femelle.



Petite comparaison avec à gauche un MSP-EXP430F5529LP à MSP430 et à droite la nouvelle carte MSP432. La quasi-totalité des cartes LaunchPad est rouge, cette dernière née est noire... Est-ce pour marquer le changement de cœur ? Mystère...

conçoit des processeurs et vend des licences aux constructeurs pour qu'ils puissent les inclure dans leurs microcontrôleurs (et leurs SoC qui peuplent nos smartphones par exemple). C'est précisément ce qu'a fait Ti pour son MSP432, tout comme il l'avait fait pour la gamme Tiva.

La carte LaunchPad MSP432, MSP-EXP432P401R de son petit nom, comprend ainsi un microcontrôleur MSP432P401R à 48 Mhz incluant 256 Ko de mémoire flash pour les programmes, 64 Ko de mémoire vive, un convertisseur analogique/numérique 12 bits 24 canaux, 6 timers et une tripotée de périphériques (4 I2C, 8 SPI, 4 UART). Ce à quoi s'ajoutent un connecteur pour boosterpack (2x2x10 broches) plus un connecteur 38 broches, des boutons poussoir, deux leds dont une RVB, un programmeur intégré avec suivi de consommation (EnergyTrace+)... Le tout pour environ 16 euros (soit moins cher qu'une Arduino Uno).

1. ET LA NOUVEAUTÉ ALORS ?

« Tout le monde utilise maintenant des cœurs ARM Cortex-M pour ses microcontrôleurs », me direz-vous. Et vous avez absolument raison. L'ATSAMD21G18 d'Atmel équipant la nouvelle Arduino Zero utilise un ARM Cortex-M0+, exactement de la même manière que l'AT91SAM3X8E des Arduino Due est construit autour d'un ARM Cortex-M3.

La belle affaire donc ! Les MSP432 ne sont donc que l'évolution ARM des MSP430 accompagnée d'un changement obligatoire de compilateur dans l'environnement de développement ? Oui et non. « Oui » parce que c'est exactement le cas et « Non » parce que les nouveautés logicielles ne s'arrêtent pas là.

Le gain de puissance apporté par le processeur a été l'occasion pour les développeurs d'Energia d'ajouter un grand plus à la plateforme : le multitâche. Alors qu'avec les précédentes versions de l'environnement il n'était possible, tout comme avec l'IDE Arduino, que de développer un croquis exécutant une unique tâche, Energia 15+ (alias Energia MT, comme *MultiThreading*), permet de programmer la carte pour qu'elle exécute plusieurs croquis en même temps.

Ceci est rendu possible par l'ajout d'un système d'exploitation : TI-RTOS. L'utilité principale d'un système d'exploitation est de gérer le fonctionnement de l'ensemble et de fournir une interface vers le matériel. Contrairement à ce qu'on pourrait penser, la notion de système d'exploitation ne s'oppose pas à celle de microcontrôleur. De tels ensembles existent depuis longtemps avec des projets comme FreeRTOS, Chibios, CooCox, Contiki, TinyOS... On parle généralement de système d'exploitation à faible empreinte mémoire. La prouesse des développeurs Energia a été d'intégrer cela dans un environnement de développement simple tout en conservant son accessibilité au plus grand nombre. Il n'est donc pas plus compliqué de développer des croquis multitâches que du code standard Energia/Arduino/Wiring, du moins dans la pratique. Nous le verrons plus loin, l'exécution de plusieurs tâches concurrentes implique certaines précautions.

Ainsi, lorsque vous développez avec Energia 15+ pour le LaunchPad MSP432, vous écrivez un ou plusieurs croquis exactement comme vous le faisiez avec les précédentes versions ou avec l'IDE Arduino. Votre code est ensuite compilé en même temps que le système

coromonadalix

En dehors de la couleur dominante, l'environnement de développement Energia ressemble comme deux gouttes d'eau à celui d'Arduino. Les deux sont basés sur l'interface de Processing et les bibliothèques Wiring, mais la mécanique sous-jacente est très différente.

d'exploitation TI-RTOS et le tout est programmé dans la mémoire de la carte. Au final chaque croquis, représenté sous la forme d'onglets dans IDE est exécuté comme une tâche distincte et indépendante.

2. INSTALLATION

Une fois la carte LaunchPad MSP432P401R en votre possession, il vous faudra installer l'environnement de développement Energia disponible au téléchargement sur **http://energia.nu**. La dernière version en date est Energia 16 et, comme la précédente version 15+, elle comprend le support de Energia MT pour le MSP432, mais maintenant également pour le LaunchPad CC3200 (intégrant le Wifi avec SSL/TLS en ROM).

Comme le précise la documentation en ligne, un point important est à considérer dans l'installation de l'environnement concernant Energia MT : le répertoire d'installation ne doit pas comporter d'espace dans son nom. En effet, le système de construction/compilation est différent de celui utilisé pour les croquis simples (pour MSP430 par exemple) et intègre des éléments de TI-RTOS. Ceux-ci utilisent des scripts tiers fonctionnant de manière différente et ne tolèrent pas d'espace dans les noms de fichiers et de répertoire. Cette mention n'est faite que concernant l'installation Windows et nous n'avons pas vérifié si cela pouvait poser problème avec Mac OS X ou GNU/Linux (en fait je ne tolère pas non plus les espaces dans les noms de répertoires ou de fichiers, donc le problème ne se pose jamais). Dans tous les cas, disons-le clairement : les espaces et les caractères accentués sont le mal, point.

2.1 Windows

L'installation de l'IDE Energia est une chose, celle des pilotes permettant de prendre en charge la carte LaunchPad MSP432P401R en est une autre. Sous Windows, vous devez en effet installer un pilote **avant de connecter la carte en USB**. Le pilote en question peut être téléchargé avec le lien **http://energia.nu/files/xds110_drivers.zip**. Après désarchivage du Zip, vous trouverez un répertoire `xds110_drivers` contenant un répertoire `xds110_drivers` (oui, oui) contenant lui-même un ensemble de fichiers, dont `DPInst.exe` et `DPInst64.exe`. Le premier permet l'installation du pilote sur un Windows 32 bits et le second sur un Windows 64 bits.

Notez que pour Windows 8 et 10 vous devez désactiver la vérification de signature sur les pilotes pour procéder à l'installation. De nombreux tutoriels détaillent la procédure sur le Web et sur YouTube.

La prise en charge de la carte LaunchPad MSP432P401R nécessite l'installation de pilotes pour Windows. Ce n'est pas le cas pour GNU/Linux ou Mac OS X.

L'exécution de `DPInst.exe` ou `DPInst64.exe` provoque le lancement d'un assistant et vous devrez confirmer l'installation du pilote. Il ne sera pas nécessaire de redémarrer la machine. La connexion de la carte LaunchPad devrait alors provoquer une recherche de pilotes et vous pourrez éventuellement passer l'étape de recherche en ligne. Windows devrait alors afficher le message précisant que les pilotes ont bien été installés et les connexions suivantes de la carte ne devraient alors provoquer aucun changement.

Il ne vous restera alors plus qu'à installer l'environnement en téléchargeant le fichier Zip et en copiant son contenu (le dossier `energia-0101E0016`) dans votre répertoire personnel, par exemple. Vous y trouverez l'icône Energia permettant le lancement de l'environnement.

2.2 GNU/Linux

La prise en charge du LaunchPad MSP432P401R sous GNU/Linux ne nécessite aucune installation de pilote. La carte est directement prise en charge par le noyau. Cependant, il vous sera nécessaire d'ajouter une règle udev afin de permettre aux utilisateurs « normaux » d'accéder au matériel.

Vous n'avez pas besoin d'écrire vous-même cette configuration, car le site d'Energia vous propose un fichier complet à l'adresse **http://energia.nu/files/71-ti-permissions.rules**. Celui-ci contient les règles udev pour l'ensemble des cartes prises en charge par Energia. Il vous suffira de télécharger le fichier et de le copier dans le répertoire `/etc/udev/rules.d/` (`sudo` nécessaire) avant de redémarrer le service udev.

La connexion de la carte LaunchPad MSP432P401R en USB activera la règle correspondant aux identifiants USB du périphérique et l'environnement pourra y accéder pour la programmation.

Remarquez que le fichier de règles proposé par le site Energia n'est pas très subtil puisqu'il autorise l'accès en lecture/écriture pour absolument tous les utilisateurs du système. Si vous êtes coutumier de l'utilisation de GNU/Linux, il sera préférable d'adapter le contenu de ce fichier à vos besoins pour limiter les autorisations conférées à un seul utilisateur ou au groupe `plugdev` par exemple.

L'installation de l'IDE lui-même se limite à un désarchivage du fichier téléchargé dans un emplacement quelconque. Vous y trouverez un répertoire `energia-0101E0016` contenant le script `energia` permettant le lancement de l'environnent écrit en Java.

2.3 Mac OS X

Dans la grande tradition de simplicité digne de la firme de Cupertino, l'installation sur Mac OS X ne nécessite absolument aucune manipulation particulière, le matériel est simplement pris en charge à sa connexion (ce n'est pas le cas pour les CC3200 et les MSP430). Il vous suffira donc de télécharger l'environnement fourni sous forme d'image DMG et de glisser l'icône Energia dans votre dossier *Applications*.

3. UN EXEMPLE ! PAR PITIÉ !

Rien de tel qu'un peu de code pour découvrir une nouveauté. Nous allons, très classiquement, écrire le programme que tout débutant se doit de réaliser lorsqu'il fait connaissance avec une plateforme : la led qui clignote.

Après vous être assuré d'avoir choisi la bonne carte (« *LaunchPad w/ msp432 EMT (48MHz)* ») et le bon port dans le menu ***Tools***, créez un nouveau « projet ». Tout comme avec l'IDE Arduino. Vous voici devant un simple croquis contenant les fonctions `setup()` et `loop()` vides. Nous complétons le code comme ceci :

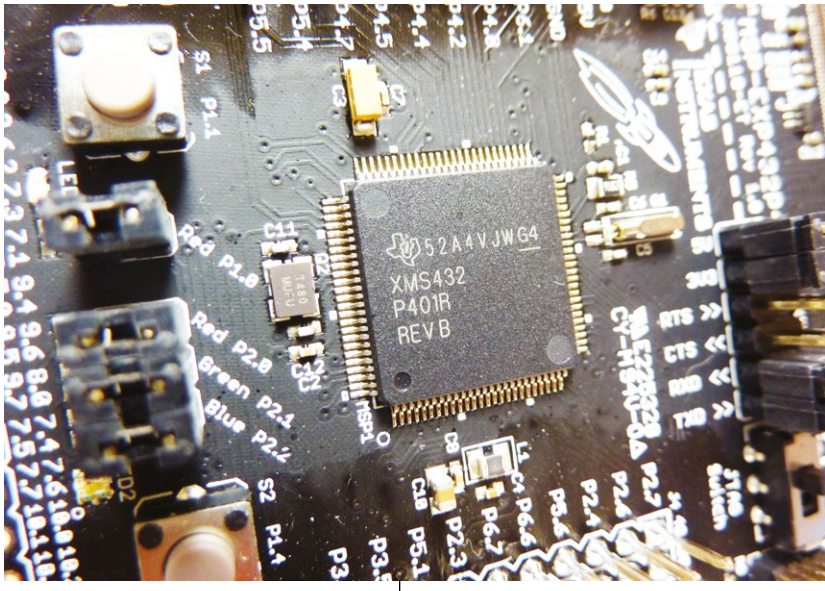
```
#define LED BLUE_LED

unsigned int compteur;
unsigned int compteurR;
unsigned int compteurV;
unsigned int compteurB;

void setup() {
  pinMode(LED, OUTPUT);
}

void loop() {
  digitalWrite(LED, HIGH);
  delay(200);
  digitalWrite(LED, LOW);
  delay(200);
  compteur++;
  compteurB++;
}
```

Vous devez normalement parfaitement reconnaître le classique **Blink** Arduino à peine agrémenté de quelques suppléments. En premier lieu, nous avons la définition de **LED** correspondant à **BLUE_LED** qui lui-même provient de l'environnement (fichier **hardware/msp432/variants/MSP_EXP432P401R/pins_energia.h**). La carte LaunchPad MSP432P401R dispose d'une led RVB et chaque couleur est reliée à un port du microcontrôleur. Nous avons respectivement **RED_LED**, **GREEN_LED** et **BLUE_LED**, ce à quoi s'ajoute **YELLOW_LED** qui est la seconde led de la carte (qui est rouge et non jaune).

Nous avons également déclaré quatre variables que nous incrémenterons dans nos codes. Notre fonction **setup()** configure le port en sortie et **loop()** l'active/désactive à intervalles réguliers (200 ms). Nous incrémentons **compteur** et **compteurB** avant de boucler indéfiniment.

Vous remarquerez qu'il n'y a rien dans ce croquis qui fait référence au multitâche et c'est là tout l'intérêt de l'intégration dans Energia. Enregistrez votre croquis et testez-le. La led RVB doit clignoter frénétiquement en bleu.

À présent, utilisez le menu déroulant accessible par l'icône à droite sur la barre d'onglet où figure le nom du croquis, et choisissez



Voici le cœur de la nouvelle carte de Texas Instruments : le microcontrôleur MSP432P401R. Au menu, ARM Cortex-M4F 32 bits à 48 Mhz, 256 Ko de flash, 64 Ko de RAM, un ADC 24 canaux 12 bits, 6 timers, 4 bus i2c, 8 bus SPI, 4 ports série et une flopée de GPIOs...

New Tab. Vous obtiendrez un nouvel onglet correspondant à un nouveau croquis faisant partie du même « projet ». Appelez celui-ci « rouge » par exemple et complétez avec le code suivant :

```
#define LED RED_LED

void setupRouge() {
  pinMode(LED, OUTPUT);
}

void loopRouge() {
  digitalWrite(LED, HIGH);
  delay(201);
  digitalWrite(LED, LOW);
  delay(201);
  compteur++;
  compteurR++;
}
```

Oui, c'est peu ou prou la même chose que le code précédent. Vous remarquerez que les deux fonctions sont ici nommées **setupRouge()** et **loopRouge()**. Le principe est le suivant : dans l'environnement Energia MT les fonctions débutant par **setup** et **loop** sont automatiquement transformées en tâches qui s'exécuteront de façon indépendante. Il n'est pas nécessaire de procéder à ces déclarations dans des

coromonadalix

onglets (fichiers) distincts, mais c'est une façon de structurer son projet de manière à simplifier sa présentation. Notez cependant que **LED** est défini par **RED_LED** dans ce code. Ce type de multi-déclaration ne pose pas de problème avec plusieurs onglets, mais c'est une mauvaise idée avec un seul. Tenez-vous-en donc à la règle « une tâche = un onglet »...

Remarquez également que la fonction **loopRouge()** incrémente **compteur** et **compteurR** qui sont déclarés dans le premier onglet. Avec Energia MT, une variable globale l'est réellement. Ceci nous permet de déclarer une variable dans une tâche et de l'utiliser dans les autres, et de ce fait d'avoir une méthode de communication inter-processus. Ce n'est, bien entendu, pas le cas pour les variables locales.

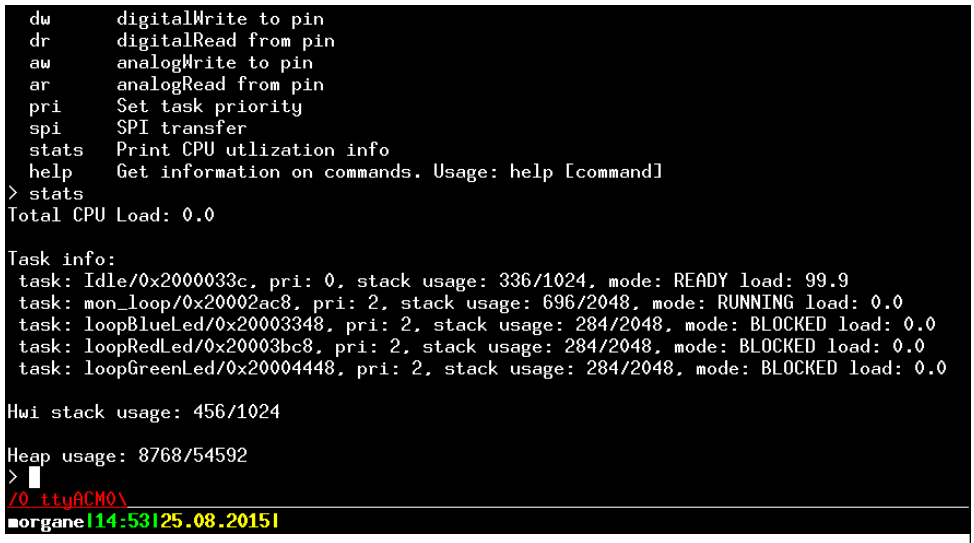
Réitérons l'opération avec un nouvel onglet que nous appelons « vert » pour l'occasion :

```
#define LED GREEN_LED

void setupVert() {
  pinMode(LED, OUTPUT);
}

void loopVert() {
  digitalWrite(LED, HIGH);
  delay(202);
  digitalWrite(LED, LOW);
  delay(202);
  compteur++;
  compteurV++;
}
```

À ce stade, nous avons trois tâches, chacune activant et désactivant une sortie correspondant à une couleur de la led RVB. Remarquez que les délais utilisés diffèrent d'une milliseconde. Si les trois tâches fonctionnent effectivement de façon indépendante, nous aurons donc des clignotements simultanés au démarrage, mais qui se désynchroniseront progressivement jusqu'à devenir visibles indépendamment.

Nous pourrions compiler et charger le code dès maintenant, mais nous allons ajouter un petit bonus. Un nouvel onglet comprendra le code d'une nouvelle tâche :

```
void setupMoniteur() {
  Serial.begin(115200);
}

void loopMoniteur() {
  Serial.print("Compteur= R:");
  Serial.print(compteurR);
  Serial.print("  V:");
  Serial.print(compteurV);
  Serial.print("  B:");
  Serial.print(compteurB);
  Serial.print("  T:");
  Serial.print(compteur);
  Serial.print("  D:");
  Serial.println(compteur-(compteurR+compteurV+compteurB));
  delay(1000);
}
```



L'un des exemples fournis, Monitor, permet d'avoir un aperçu de l'exécution des tâches via une liaison série. L'exemple, une fois complété de codes/tâches « maison » nous montre par exemple la charge du système et le partage du temps d'exécution, ici entre trois tâches utilisateurs, le moniteur lui-même et « Idle » correspondant à « au repos » ou « inoccupé ».

Ici, toutes les secondes, nous allons afficher sur le moniteur série la valeur de chaque compteur. Vous l'avez compris, nous en avons un pour chaque tâche/couleur ainsi qu'un compteur incrémenté par toutes les tâches. Méfiants, nous en profitons pour nous assurer que la somme des compteurs correspond effectivement à la valeur du compteur total.

Il ne nous reste plus qu'à compiler et charger l'ensemble dans la mémoire du LaunchPad MSP432P401R avant d'ouvrir le moniteur série (l'IDE Energia offre d'ailleurs une entrée de menu et un raccourci très pratique pour faire cela en une fois : Ctrl+M).

Dès le code chargé, la led RVB va se comporter comme attendu et nous allons voir apparaître dans la fenêtre du moniteur série les lignes :

```
Compteur= R:2   V:2   B:2   T:6   D:0
Compteur= R:4   V:4   B:5   T:13   D:0
Compteur= R:7   V:7   B:7   T:21   D:0
Compteur= R:9   V:9   B:10   T:28   D:0
[...]
Compteur= R:112   V:111   B:112   T:335   D:0
Compteur= R:114   V:114   B:115   T:343   D:0
Compteur= R:117   V:116   B:117   T:350   D:0
[...]
```

Vous venez d'écrire votre premier programme multitâche sur MSP432 ! Bravo ! Nous avons ici quatre tâches concurrentes se partageant le processeur ARM Cortex-M4F, le tout orchestré par TI-RTOS.

Le site d'Energia ne fournit pas davantage de documentation et renvoie, tout comme je vais le faire, vers les exemples livrés et accessibles via le menu ***File***, ***Examples***, ***MultiTasking***. C'est en effet là que vous trouverez des démonstrations intéressantes concernant cette fonctionnalité. Je vous recommande l'exemple *Monitor* permettant d'avoir une vue sur le système via une communication série, et je vous conseille en particulier de jouer avec, en ajoutant des tâches et en intégrant vos propres commandes.

4. SIMPLE, NON ? EN FAIT, PAS VRAIMENT…

À première vue tout cela parait être d'une simplicité enfantine. Après tout il suffit de diviser son code en plusieurs tâches et d'écrire tout cela en conséquence. Jusqu'au moment où arrive la question fatidique : que se passe-t-il si plusieurs tâches essaient d'utiliser la même ressource en même temps ? La réponse est simple : des problèmes.

La simple exécution de plusieurs tâches n'est pas un problème en soi tant que tout ce petit monde s'occupe de ses affaires. Mais pour que les tâches utilisent un périphérique, comme la liaison série par

coromonadalix

exemple, ceci ne peut se faire n'importe comment. Si trois tâches appellent chaotiquement `Serial.print()`, le résultat sera très certainement un mélange de toutes les chaînes de caractères émises. Pour régler le problème, on utilise généralement un objet appelé sémaphore où une variable globale sert de compteur et sa valeur représente le nombre de ressources disponibles à un instant donné.

D'autres mécanismes existent prenant en compte les problématiques liées au multitâche :

- gestion d'événements ;
- gestion de priorités ;
- mutex ;
- système de boîte à lettres (*mailboxes*) ;
- interruptions logicielles ;
- etc.

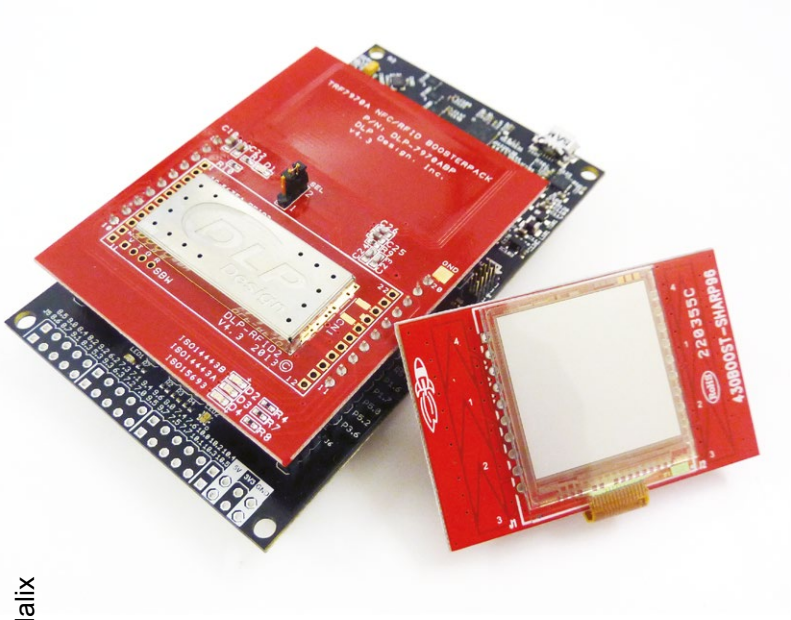
Une grande partie de ces mécanismes sont présents dans TI-RTOS et accessibles via Energia MT sous forme de bibliothèques (voir exemples), mais c'est un environnement totalement différent de celui du développement Arduino standard. Il est possible de faire des merveilles avec ce type de fonctionnalités, mais vous devrez, à un moment ou un autre, prendre le temps d'apprendre à en faire un usage judicieux.

5. VERS LE MULTITÂCHE POUR TOUS ET PARTOUT

Le multitâche n'est pas quelque chose de nouveau. Tout utilisateur d'ordinateur en profite depuis des dizaines d'années, mais c'est quelque chose qui semble être une « nouveauté récurrente ». En effet, si vous avez bonne mémoire, fut un temps nos chers téléphones mobiles (avant qu'il ne faille les charger tous les deux jours) n'étaient pas capables de proposer plusieurs fonctions, tâches, applications en même temps. La notion de multitâche est intimement liée à celle de partage de ressources et pour que des ressources puissent devoir être partagées il faut qu'elles existent en quantité suffisante pour plusieurs programmes.

Tout comme il était inutile, il y a 30 ans, de devoir partager les 64 Ko de mémoire et la puissance d'un processeur MOS 6510 à environ 1 Mhz, il était inutile jusqu'alors d'occuper la quasi-totalité de la mémoire flash d'un Atmel AVR, par exemple, pour faire fonctionner un système d'exploitation. Les choses sont différentes aujourd'hui, car les cœurs ARM Cortex-M intègrent des fonctionnalités adaptées et peuvent gérer une grande quantité de mémoire.

La carte LaunchPad MSP432P401R, grâce à Energia et TI-RTOS ouvre la voie pour une nouvelle génération d'environnement de développement et une nouvelle façon d'appréhender le développement de croquis reposant sur Wiring. Il y a fort à parier que ce qui est pour l'instant réservé aux utilisateurs d'Energia se généralise. Les plateformes et microcontrôleurs existent et les systèmes d'exploitation aussi, il ne manque que l'initiative d'intégrer tout cela dans un IDE pour en faire une réalité. Les développeurs Energia l'ont fait et je pense que d'autres suivront... **DB**



Toutes les cartes LaunchPad permettent d'accueillir des cartes filles additionnelles appelées BoosterPack. Naturellement, la LaunchPad MSP432P401R fait de même. Ici enfiché le BoosterPack NFC/RFID et sur la droite, l'afficheur à mémoire et à très faible basse consommation Sharp96.

coromonadalix

PROFESSIONNELS !

DÉCOUVREZ NOS NOUVELLES OFFRES D'ABONNEMENTS ...

PDF COLLECTIFS

		PROFESSIONNELS					
		1 - 5 lecteurs		6 - 10 lecteurs		11 - 25 lecteurs	
OFFRE	ABONNEMENT	Réf	Tarif TTC	Réf	Tarif TTC	Réf	Tarif TTC
PROHK2	6ⁿ° HK	☐ PRO HK2/5	156,-	☐ PRO HK2/10	312,-	☐ PRO HK2/25	624,-

Prix TTC en Euros / France Métropolitaine

PROFESSIONNELS : N'HÉSITEZ PAS À NOUS CONTACTER POUR UN DEVIS PERSONNALISÉ PAR E-MAIL : abopro@ed-diamond.com OU PAR TÉLÉPHONE : 03 67 10 00 20

ACCÈS COLLECTIFS BASE DOCU

		PROFESSIONNELS					
		1 - 5 connexion(s)		6 - 10 connexions		11 - 25 connexions	
OFFRE	ABONNEMENT	Réf	Tarif TTC	Réf	Tarif TTC	Réf	Tarif TTC
PROOS+3	OS	☐ PRO OS+3/5	90,-	☐ PRO OS+3/10	180,-	☐ PRO OS+3/25	360,-
PROH+3	GLMF + HS + LP + HS + MISC + HS + OS	☐ PRO H+3/5	447,-	☐ PRO H+3/10	894,-	☐ PRO H+3/25	1788,-

Prix TTC en Euros / France Métropolitaine

OS = Open Silicium
LP = Linux Pratique
HS = Hors-Série
LM = GNU/Linux Magazine France

coromonadalix

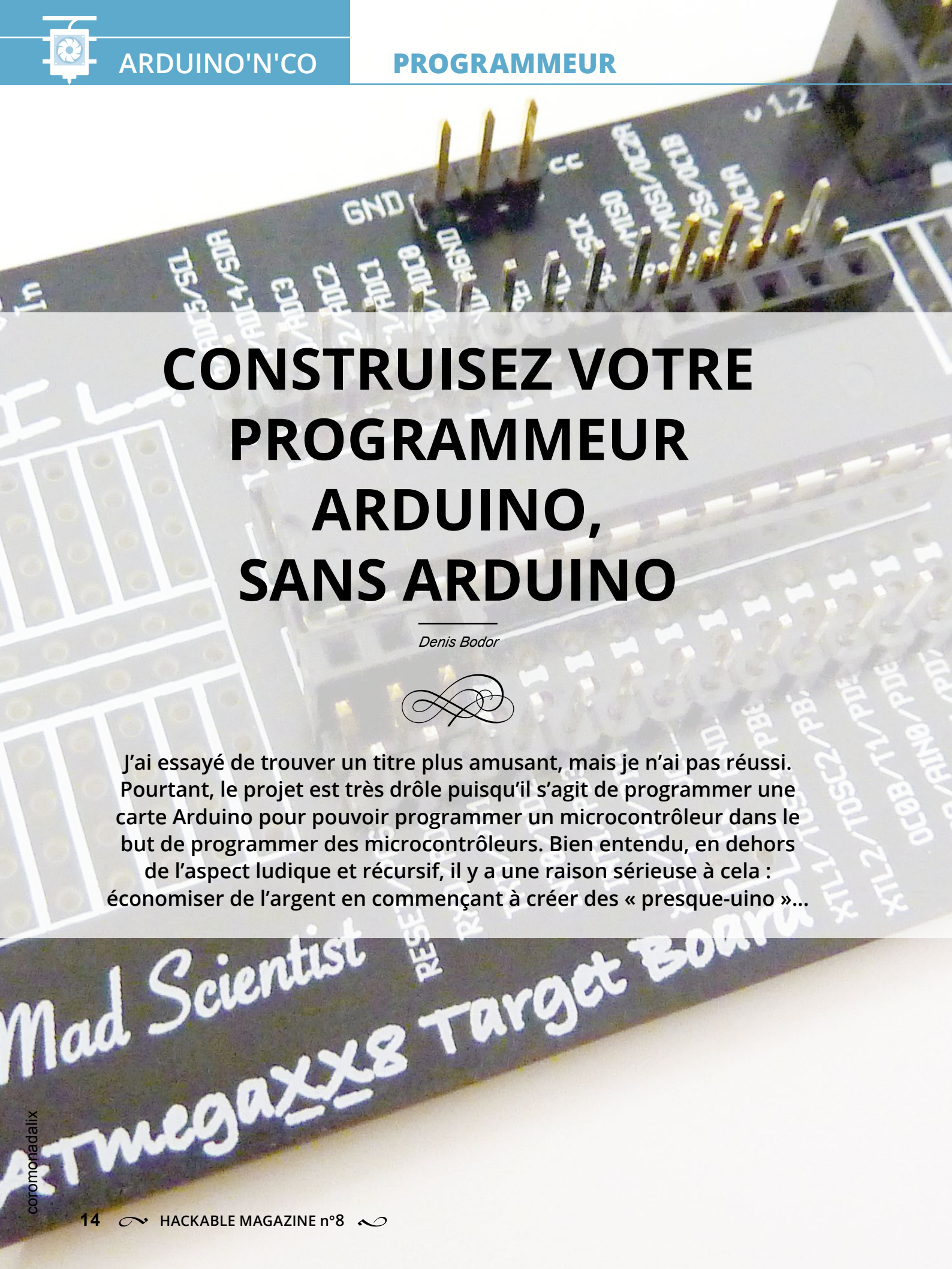
...EN VOUS CONNECTANT À L'ESPACE DÉDIÉ AUX PROFESSIONNELS SUR :

www.ed-diamond.com

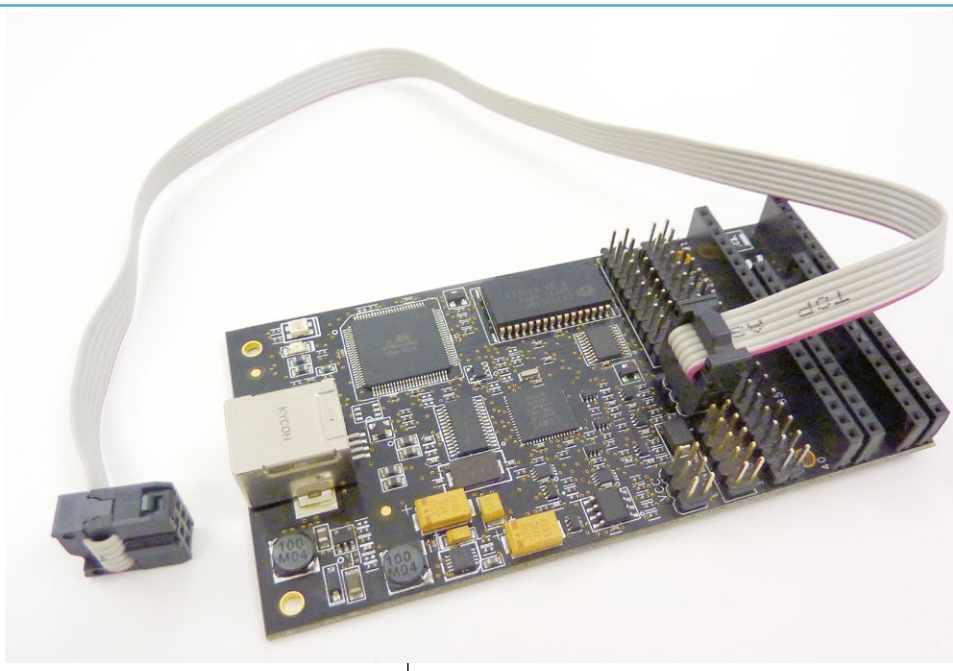
CONSTRUISEZ VOTRE PROGRAMMEUR ARDUINO, SANS ARDUINO

Denis Bodor

J'ai essayé de trouver un titre plus amusant, mais je n'ai pas réussi. Pourtant, le projet est très drôle puisqu'il s'agit de programmer une carte Arduino pour pouvoir programmer un microcontrôleur dans le but de programmer des microcontrôleurs. Bien entendu, en dehors de l'aspect ludique et récursif, il y a une raison sérieuse à cela : économiser de l'argent en commençant à créer des « presque-uino »...



coromonadalix



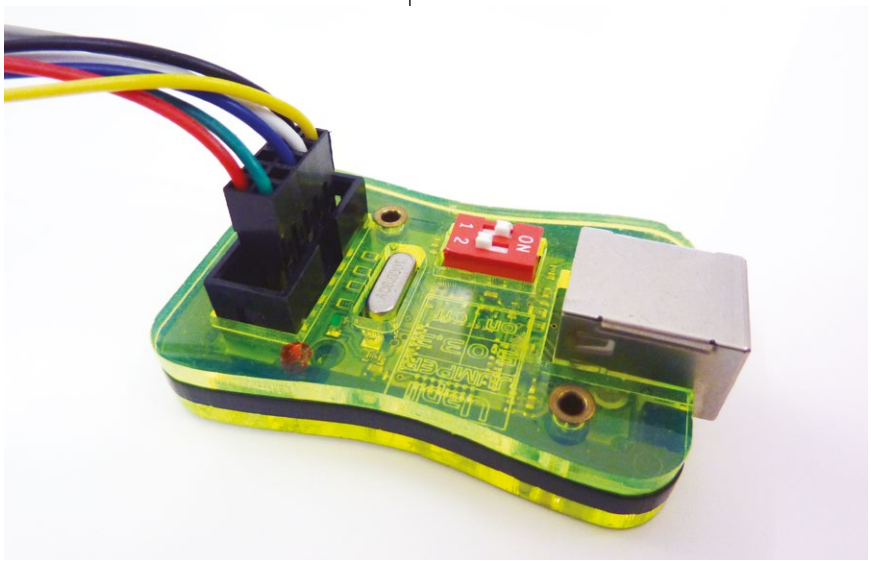
L'AVR Dragon d'Atmel est un programmeur de microcontrôleur AVR 8 et 32 bits. Il permettra la programmation en ISP, PDI (Xmega) et JTAG. Mais le plus important reste sa capacité à utiliser le HVSP pour « sauver » un AVR mal configuré.

Nous l'avons vu il y a quelques numéros de cela (*Hackable n°1*), il est depuis longtemps possible d'utiliser une carte Arduino Uno comme un programmeur de microcontrôleurs Atmel AVR. Ce faisant, il est possible de composer sur platine à essai un montage à base du même microcontrôleur que celui d'une Arduino Uno (ETmega328p), mais en une déclinaison épurée. On se passe alors de toute la partie alimentation, du circuit USB/série (FTDI ou ATmega16u2) et du *bootloader* qui permet de programmer le composant via une liaison série. Le résultat est extrêmement concis et économique en plus de démarrer très rapidement, et se résume en un microcontrôleur ATmega328p, par exemple, quelques connecteurs et une source d'alimentation en 5 volts stable.

Le seul problème dans l'histoire c'est qu'on est forcément obligé de monopoliser une carte Arduino Uno dès lors qu'on souhaite programmer un microcontrôleur AVR. Même si l'usage est temporaire, encore faut-il avoir une Uno libre sous la main. À 20 euros la carte, ce n'est pas hors de prix, mais c'est tout de même embêtant (je ne sais pas pour vous, mais je n'ai jamais une carte Arduino libre).

Pourquoi alors ne pas pousser un cran plus loin ? Le principe consiste à programmer un croquis spécifique dans l'Arduino pour qu'il se comporte comme un programmeur stk500v1 qui peut alors être utilisé par l'environnement Arduino. Le croquis est d'ailleurs livré directement avec l'environnement, sous le nom ArduinoISP. On peut donc, en principe charger ce croquis dans l'Arduino Uno pour programmer un ATmega328p nu sur platine à essai. Le croquis utilisé pour cet ATmega328p sera également ArduinoISP, transformant le montage « nu » en programmeur indépendant qu'il faudra, bien entendu, compléter d'un adaptateur USB/série et de quelques leds (c'est toujours bien les leds). On libérera ainsi définitivement l'Arduino Uno.

Un programmeur ISP plus ou moins générique de fabrication allemande tantôt désigné par DX-ISP. Celui-ci date d'une époque où personne n'avait encore prononcé le nom Arduino, mais est toujours très utile. C'est mon premier « vrai » programmeur ISP qui a remplacé un montage maison utilisant le, maintenant disparu, port parallèle du PC (LPT).



coromonadalix

QUEL ACTE STUPIDE ?

Ceci vous est totalement invisible lorsque vous utilisez une carte Arduino, mais le microcontrôleur qui s'y trouve possède une configuration. Celle-ci prend la forme de « fusibles » (*fuses*) qui sont en réalité une série de bits stockés dans trois registres particuliers (le terme « fusible » n'est pas adapté, car l'écriture n'est pas irréversible. C'est un héritage du passé où les bits étaient effectivement définitivement inscrits). Ces bits permettent par exemple de régler l'utilisation d'un bootloader appelé « séquence d'initialisation » dans l'IDE Arduino (programme de démarrage, lancé juste avant votre croquis en mémoire flash), la vitesse de démarrage ou encore, la source d'horloge du composant.

Cette dernière configuration est souvent celle qui peut conduire aux problèmes. Le microcontrôleur AVR d'une carte Arduino utilise un oscillateur à quartz à 16 Mhz pour fonctionner et il est donc configuré dans ce sens. Mais un AVR peut également fonctionner avec d'autres sources comme un résonateur céramique, un signal externe ou encore des quartz de fréquence différentes (32,768 Khz par exemple). Il peut également ne pas utiliser du tout de composant extérieur et reposer entièrement sur une horloge interne, moins précise, mais permettant d'avoir un circuit plus simple.

Une fois le microcontrôleur configuré pour une source, celui-ci fonctionne totalement sur cette base, et ceci vaut pour le mode « programmation » du composant. En clair, si vous configurez votre microcontrôleur pour une source externe à quartz de haute fréquence alors que ce composant n'est pas présent, la configuration sera effectivement inscrite, mais les tentatives de programmation ultérieures ne fonctionneront pas. Du moins pas temps que le quartz en question (et ces deux condensateurs) n'est pas installé. Vous ne pourrez donc plus revenir en arrière, car le microcontrôleur ne vous répondra simplement plus.

La configuration des fusibles d'un microcontrôleur AVR peut donc déboucher sur une situation bloquante, et la source d'horloge n'est pas le seul paramètre qui peut conduire à ce type de chose. En l'absence de circuit correspondant à la configuration, il ne reste plus qu'une seule solution : le HVSP pour *High Voltage Serial Programming*. Une technique de programmation permettant de mettre le microcontrôleur dans un mode particulier pour reprendre la main et reconfigurer les fusibles correctement.

La plupart des programmeurs existants ne disposent pas de ce mode de programmation particulier nécessitant, entre autres, l'application d'une tension de 12 volts sur la broche « reset ». Il faut alors soit passer par un montage maison, soit utiliser un programmeur comme l'Atmel AVR Dragon.

Conclusion : si vous commencez à jouer avec les fusibles des microcontrôleurs, mieux vaut avoir un programmeur qui supporte le HVSP.

Dans cette réalisation, nous allons économiser un ATmega328p en le remplaçant par un Atmega168 moins coûteux et surtout traînant inutilement depuis des années dans un de mes tiroirs. L'idée ici n'est pas réellement d'économiser au maximum, mais de simplement éviter d'utiliser une carte Arduino pour cet usage unique. Si vous recherchez une manière la plus économique possible de programmer un microcontrôleur Atmel AVR, il existe des dizaines de produits « programmeur USB ISP » compatibles STK500 ou USBasp pour quelques euros. À l'opposé, si le prix vous importe peu ou que vous souhaitez le programmeur le plus polyvalent et le plus puissant possible, vous avez le Atmel-ICE de chez Atmel capable de programmer énormément de composants Atmel (MegaAVR, TinyAVR, Xmega, SAM, etc.), mais il vous en coûtera environ 60 euros. À ce prix, vous pourrez vous sortir de presque n'importe quelle situation découlant d'un acte stupide. Le programmeur Atmel AVR Dragon est également un très bon choix pour cela. Si vous êtes entre les deux solutions, cet article est fait pour vous.

Terminons cette introduction en précisant que les opérations qui vont suivre reposent sur les fonctionnalités offertes par la nouvelle version 1.6.4 de l'environnement Arduino et en particulier l'ajout de cartes décrit par ailleurs dans ce numéro. Nous utiliserons, en effet, le support « Barebones ATmega Chips » de C. Rodrigues, puis le support « attiny » de D. Mellis pour le premier essai.

coromonadalix

1. PHASE 1 : TRANSFORMER SA CARTE ARDUINO UNO EN PROGRAMMEUR ISP

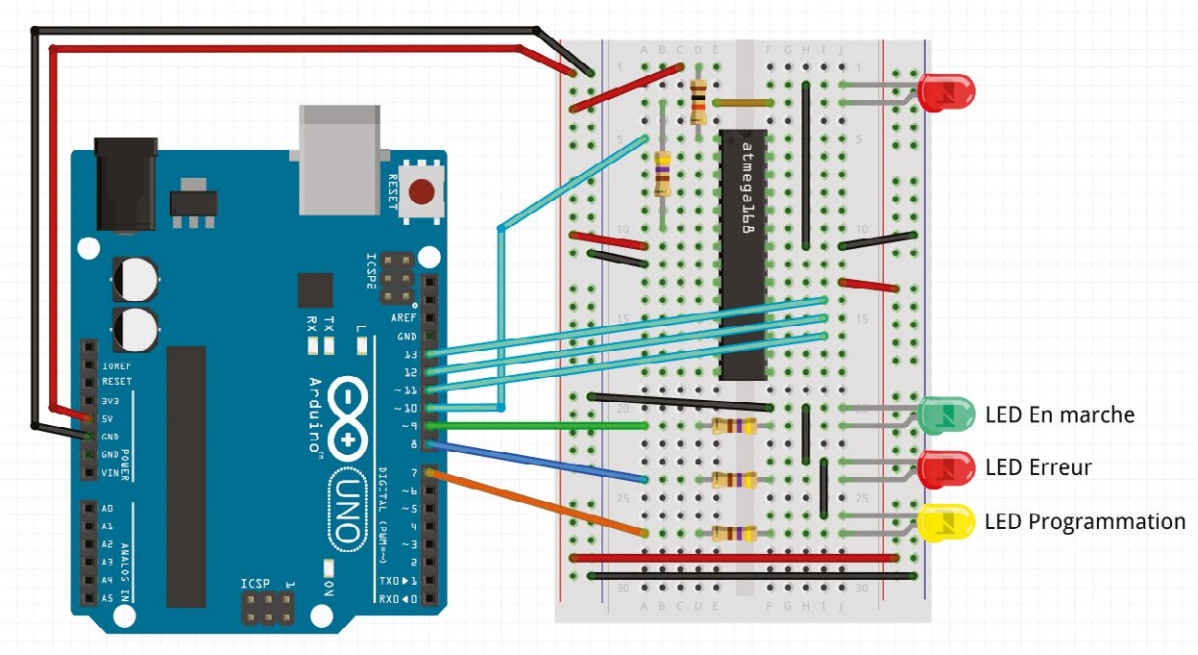
La première étape consiste ici à programmer l'Arduino Uno avec un croquis permettant de faire en sorte qu'il se comporte comme un programmeur STK500. Ceci est relativement simple puisqu'il suffit d'ouvrir le croquis ArduinoISP depuis la liste d'exemples fournis avec l'environnement Arduino, de le compiler pour l'Arduino UNO et de le charger (téléverser).

Dès lors, notre Uno se comporte comme un programmeur. Pour l'utiliser et programmer un ATmega168 par exemple, il faut connecter l'ensemble correctement :

Le montage utilise tout d'abord les sorties 7, 8 et 9 pour y connecter des leds permettant d'avoir une idée de l'état de fonctionnement du programmeur. Ceci n'est pas une obligation, mais s'avère toujours utile lorsque quelque chose ne fonctionne pas. La sortie 7 est le témoin de programmation (led jaune), la 8 permet de signifier une erreur (led rouge) et la 9 pulse régulièrement indiquant que le croquis ArduinoISP est en marche et prêt à être utilisé (led verte).

Vient ensuite la connexion au microcontrôleur à programmer. Ici, j'utilise un ATmega168, mais un ATmega328p fera parfaitement l'affaire. Nous devons tout d'abord nous pencher sur l'alimentation. L'ATmega168 dispose de deux connecteurs pour la masse, sur les broches 8 et 22. La tension d'alimentation, +5 volts, sera appliquée sur la broche 7 VCC ainsi que sur la broche 20 AVCC. Cette dernière est la tension pour le convertisseur analogique/numérique du microcontrôleur. La documentation Atmel est très claire sur ce point et précise qu'il faut que cette broche soit reliée à la tension d'alimentation même si le convertisseur n'est pas utilisé (et avec un filtre passe bas, si le convertisseur est utilisé).

Enfin la broche 1, reset, permettant de réinitialiser le microcontrôleur, est reliée à la tension l'alimentation (VCC) via une résistance de rappel de 10 Kohms. Bien que l'ATmega168 intègre une telle résistance, il est recommandé, dans un environnement riche en perturbation électromagnétique, d'utiliser ce type de montage. La résistance de rappel force la broche reset à +5V, sauf si on la connecte à la masse. Dans ce cas précis, ceci déclenche une réinitialisation.



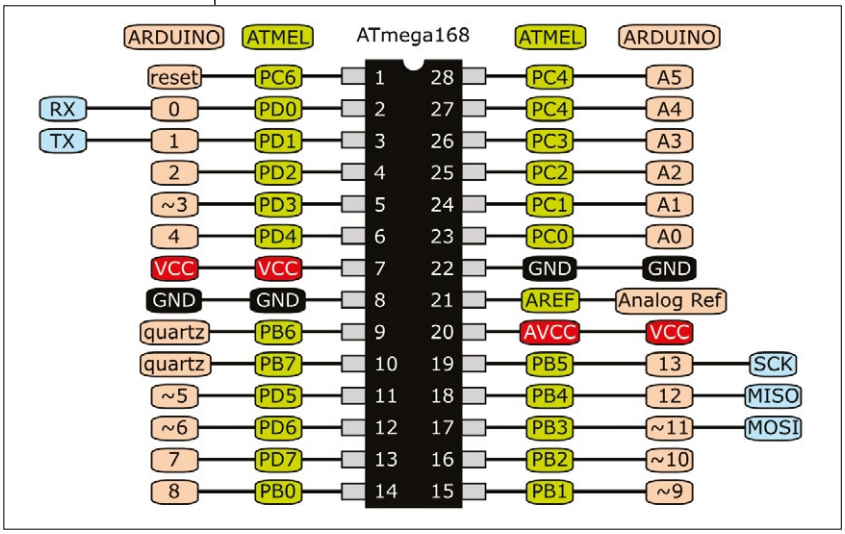
coromonadalix

Penchons-nous maintenant sur la connexion entre la Uno et l'ATmega168. La famille de microcontrôleur AVR d'Atmel utilise ce qu'on appelle de l'ISP pour *In-System Programming* ou programmation in situ, car il n'est pas nécessaire de retirer le composant de son circuit et de le placer dans un programmeur pour y inscrire un programme. Beaucoup de microcontrôleurs utilisent aujourd'hui cette technique, mais fut un temps c'était réellement une nouveauté.

Les broches utilisées pour la programmation sont les mêmes que celle du bus SPI du microcontrôleur et sont utilisées de façon similaire. À la différence d'une utilisation d'une carte Arduino pour piloter un composant SPI, ici les rôles sont en quelque sorte inversés. Côté microcontrôleur « cible », ATmega168 donc, le composant est esclave et le programmeur est maître. On connecte donc les broches ainsi :

- Uno 11 - ATmega168 broche 17, MOSI *Master Out, Slave In* ;
- Uno 12 - ATmega168 broche 18, MISO *Master In, Slave Out* ;
- Uno 13 - ATmega168 broche 19, SCK signal d'horloge.

Ce n'est pas tout, la programmation ne peut avoir lieu que dans certaines conditions, contrôlées par l'état de la ligne reset de la cible. On connecte donc également la sortie 10 de l'Arduino Uno à la broche 1, reset, de l'ATmega168.



Correspondance entre la nomenclature des broches d'un ATmega168 dans la documentation Atmel et le brochage Arduino.

Enfin, afin de tester le fonctionnement de la programmation avant de passer à l'étape suivante, on relie la broche 6 de l'ATmega168 à l'anode d'une led via une résistance de 470 ohms, et la cathode de la led à la masse. Cette broche correspond au bit 4 du port D du microcontrôleur, qui en nomenclature Arduino correspond à la broche digitale 4.

Il ne reste plus qu'à tester si tout fonctionne. Pour cela, nous devons tout d'abord installer le support *Barebones ATmega Chips* de C. Rodrigues. Dans les préférences de l'environnement Arduino 1.6.4, ajoutez le dépôt **https://raw.githubusercontent.com/carlosefr/atmega/master/package_carlosefr_atmega_index.json** dans ***Additionnal Boards Manager URLs***. Enregistrez les modifications en confirmant avec le bouton ***OK***, puis allez faire un tour dans le menu ***Outils*** et lancez le ***Boards Manager*** dans le sous-menu ***Type de carte***. Dans la fenêtre, vous devez trouver une entrée « *Barebones ATmega Chips* ». Cliquez dessus, puis sur le bouton ***Install***.

Vous devez maintenant trouver dans le menu ***Outils***, ***Type de carte*** une mention « ATmega168 » et « Atmega328p ». Sélectionnez le type de microcontrôleur que vous avez utilisé comme cible sur la platine à essai. Après cette sélection, le menu présentera une entrée ***Clock*** permettant de choisir le type d'horloge utilisé par l'ATmega168. Ici, nous n'avons

coromonadalix

utilisé aucun quartz et nous nous contenterons donc de choisir entre « 1 Mhz (internal) » et « 8 Mhz (internal) » correspondant aux deux options utilisables avec l'oscillateur RC interne. **Si vous n'avez pas de quartz avec les condensateurs adaptés, ne sélectionnez surtout pas « 16 Mhz (external) »** (en fait la sélection n'est pas un problème, mais si vous appliquez les changements, l'aventure s'arrête là, cf. boîte sur l'acte stupide).

Par défaut, avec un ATmega168 tout neuf sorti d'usine, les fusibles du composant sont configurés sur l'horloge interne à 1 Mhz. Nous avons l'opportunité de faire fonctionner le microcontrôleur 8 fois plus rapidement, ne nous en privons pas. Sélectionnez donc « 8 Mhz (internal) ».

Vous devez également changer le programmeur utilisé. Par défaut, pour une carte Arduino c'est AVRISPmkII qui est utilisé (bootloader), mais nous ne voulons pas programmer l'ATmega328p de la Uno, mais l'ATmega168 via la Uno. Rendez-vous donc dans le menu ***Outils***, ***Programmateur*** et choisissez ***Arduino as ISP***. Dès lors, l'environnement va dialoguer avec le croquis dans la carte Uno pour accéder au microcontrôleur sur la platine à essais (note toute personnelle : remarquez le terme « programmateur » utilisé dans l'IDE. « téléverser » et « vérifier » à la place de « uploader » et « compiler » passe encore, mais programmateur... Non. Pour moi les « programmateurs » sont uniquement sur les lave-linges).

Passez maintenant par le menu ***Outil***, ***Graver la séquence d'initialisation***. Avec comme type de carte « ATmega168 » et « Atmega328p », ceci ne programme pas réellement de bootloader, mais configure simplement des fusibles. L'opération doit déboucher sur un message « Gravure de la séquence d'initialisation terminée ».

Si vous obtenez une erreur, ceci peut être pour plusieurs raisons :

- vous n'avez pas sélectionné le bon microcontrôleur ;
- les branchements ne sont pas faits correctement ;
- avez-vous utilisé un microcontrôleur provenant d'une carte Arduino ? Si oui, il est configuré pour utiliser un Quartz et vous devez soit en installer un, soit placer le microcontrôleur sur une carte Arduino pour reconfigurer les fusibles comme sur la platine à essai. Dans les deux cas, c'est une mauvaise idée, il n'y a aucun intérêt à économiser sur l'achat d'un microcontrôleur AVR à 4 € en rendant une carte Uno à 20 € inutile.

Si l'opération réussit, ceci signifie deux choses : vous pouvez effectivement programmer votre ATmega168 et celui-ci fonctionne à présent à 8 Mhz.

Vous pouvez maintenant charger le croquis de démonstration Blink présent dans les exemples, changer la sortie utilisée en **4** et programmer votre ATmega168. Si tout se passe correctement, le croquis doit être compilé puis chargé dans le microcontrôleur. Celui-ci, très rapidement, doit faire clignoter la led 1 seconde allumée et 1 seconde éteinte. Cette vérification est importante pour vérifier la cohérence avec la configuration de l'horloge interne. En résumé :

- le choix de l'horloge impacte le microcontrôleur au moment de l'inscription des fusibles qui déterminent sa configuration ;
- le choix de l'horloge impacte le croquis pour les calculs des délais.

Il faut donc vous souvenir quel microcontrôleur est configuré avec quelle fréquence interne, car dans le cas contraire un croquis pour 8 Mhz fonctionnera 8 fois moins vite sur un microcontrôleur à 1 Mhz. Inversement, un croquis à 1 Mhz aura des délais divisés par 8 sur un microcontrôleur configuré à 8 Mhz. C'est peut-être secondaire pour un simple clignotement de led, mais **capital** pour la communication série, le pilotage de leds WS2812b ou encore pour le contrôle de servomoteurs. Astuce : la gommette de couleur est votre amie.

Enfin, précisons que l'utilisation de l'horloge interne à 1 Mhz ou 8 Mhz n'est pas qu'une simple question de vitesse, mais aussi et surtout de précision. La communication série à haute vitesse aura un taux d'erreur clairement spécifié dans la documentation Atmel. C'est également le cas pour les oscillateurs à quartz à 16 ou 20 Mhz. À 16 Mhz par exemple, une liaison série à 115200 bps possède un taux d'erreur de 2,1%. Pour faire tomber le taux d'erreur à 0% pour les débits standards

(2400, 9600, 19200, 57600, etc.), il faut utiliser un oscillateur à 14,7456 MHz ou 18,4320 MHz. Pour une utilisation courante, cependant, ce n'est pas un gros problème. Ici nous utilisons un ATmega168 sur oscillateur interne à 8 Mhz communiquant en série pour programmer d'autres microcontrôleurs et cela fonctionne très bien.

Dernière étape, comme Blink fonctionne parfaitement, nous n'avons qu'à ouvrir le croquis ArduinoISP et le programmer dans l'ATmega168 de la même manière. Nous avons donc un Arduino Uno chargé avec ArduinoISP qui programme ArduinoISP dans un ATmega168... (une machine qui programme une machine, mon dieu, Skynet n'est pas loin !).

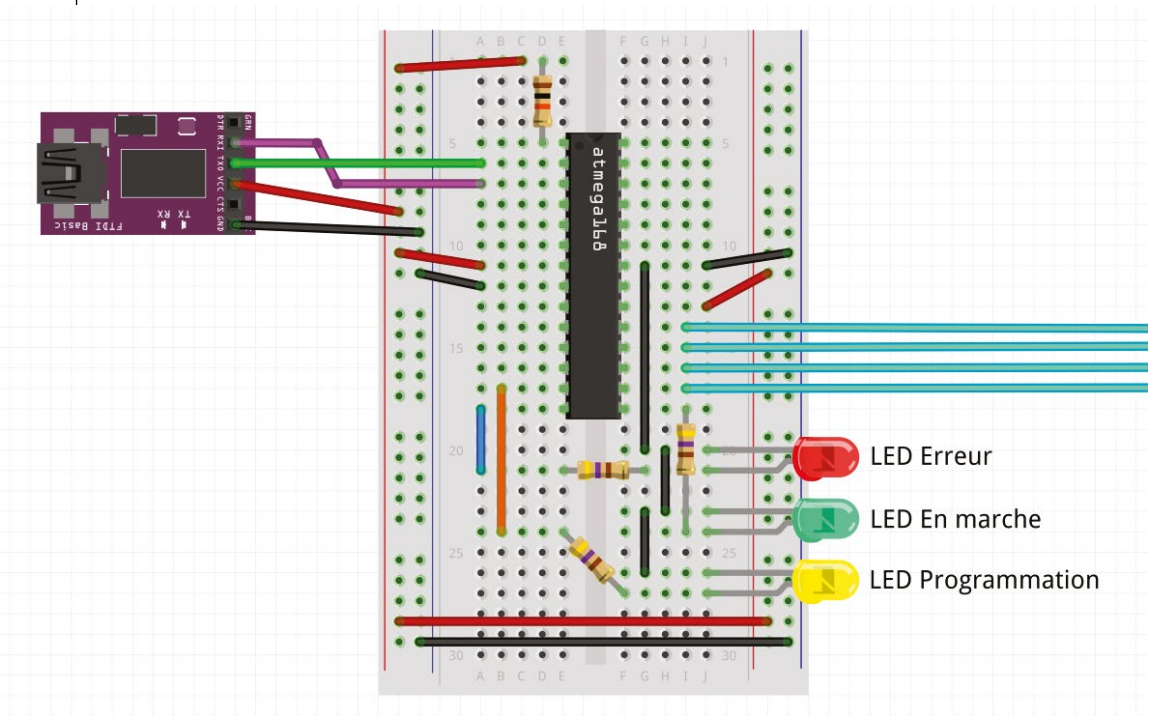
2. PHASE 2 : ASSEMBLER LE PROGRAMMEUR ATMEGA168

Notre ATmega168 est maintenant programmé avec ArduinoISP et est en mesure de se comporter exactement comme la carte Arduino Uno que nous avons utilisée. Il lui manque cependant un certain nombre de choses et nous devons donc revoir notre montage :

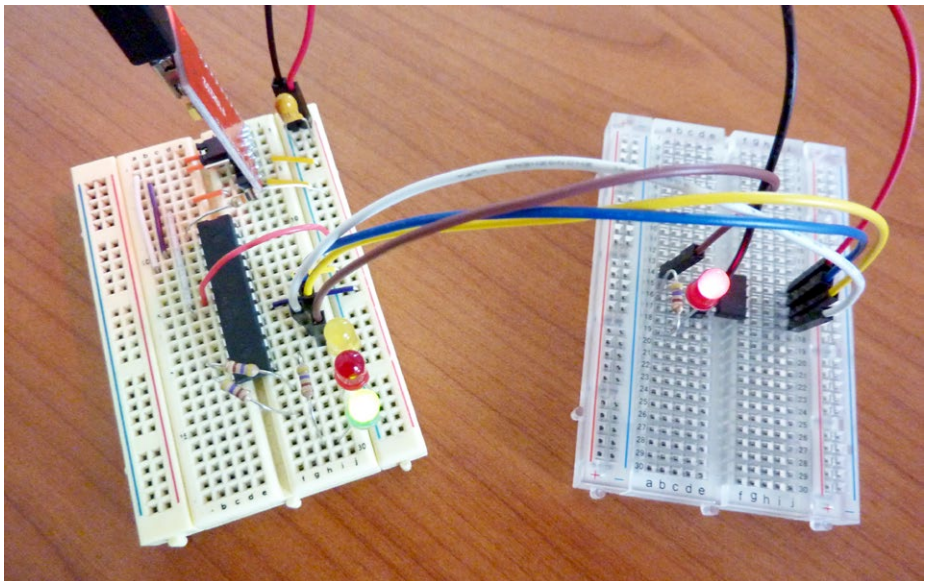
En premier lieu, nous devons transférer les leds de l'Uno à la platine à essai. C'est une simple affaire de correspondance entre les broches Arduino et celles du microcontrôleur :

- led jaune : broche 13 alias PD7 (port D bit 7) est 7 sur l'Arduino ;
- led rouge : broche 14 alias PB0 (port B bit 0) est 8 sur l'Arduino ;
- led verte : broche 15 alias PB1 (port B bit 1) est 9 sur l'Arduino.

Concernant les broches MISO, MOSI et SCK nous n'avons rien à faire si ce n'est les déconnecter côté Arduino Uno. Nous les brancherons sur le microcontrôleur cible qui devra être programmé



coromonadalix



À droite, le montage en programmateur ISP à base d'ATmega168 et à gauche le microcontrôleur ATtiny85 cible. Certes, utiliser un ATmega168 pour une tâche aussi simple est un peu démesuré, mais rien n'empêche d'étoffer le croquis ArduinoISP pour supporter quelques boutons, un afficheur LCD et une carte SD contenant le code à charger dans la cible.

(un ATtiny85 pour l'exemple). La broche reset en revanche qui est 10 sur l'Arduino devient la broche 16 (PB2) sur l'ATmega168.

Enfin, il nous faut un moyen pour le PC ou le Mac de communiquer avec le montage. Pour cela, nous utilisons simplement un convertisseur USB/série. Celui-ci fournira également l'alimentation à la platine et nous connectons donc Vcc et la masse (GND). La broche TX (envoi) de l'adaptateur se connecte sur le RX de l'Atmega168, broche 2. Inversement le RX (réception) de l'adaptateur va sur le TX de l'ATmega168, broche 3.

Le premier test qu'on peut maintenant appliquer consiste à tout simplement connecter l'adaptateur USB au PC ou au Mac. Ceci devrait alimenter l'ATmega168 et donc provoquer une pulsation sur la led verte. C'est le cas ? Parfait, le croquis ArduinoISP fonctionne sur l'ATmega168, nous n'avons plus besoin de la carte Uno pour programmer des AVR.

Remarquez que comme l'ATmega168 ne possède pas de bootloader, il démarre bien plus rapidement que l'Arduino Uno. Rappelons que le bootloader ou séquence d'initialisation, démarre normalement systématiquement avant votre croquis de manière à permettre une programmation (téléversement). L'environnement Arduino réinitialise le microcontrôleur et profite de ce laps de temps pour prendre la main et charger le code. En l'absence de bootloader, quelques millisecondes après la mise sous tension ou un reset, c'est directement votre croquis qui est exécuté.

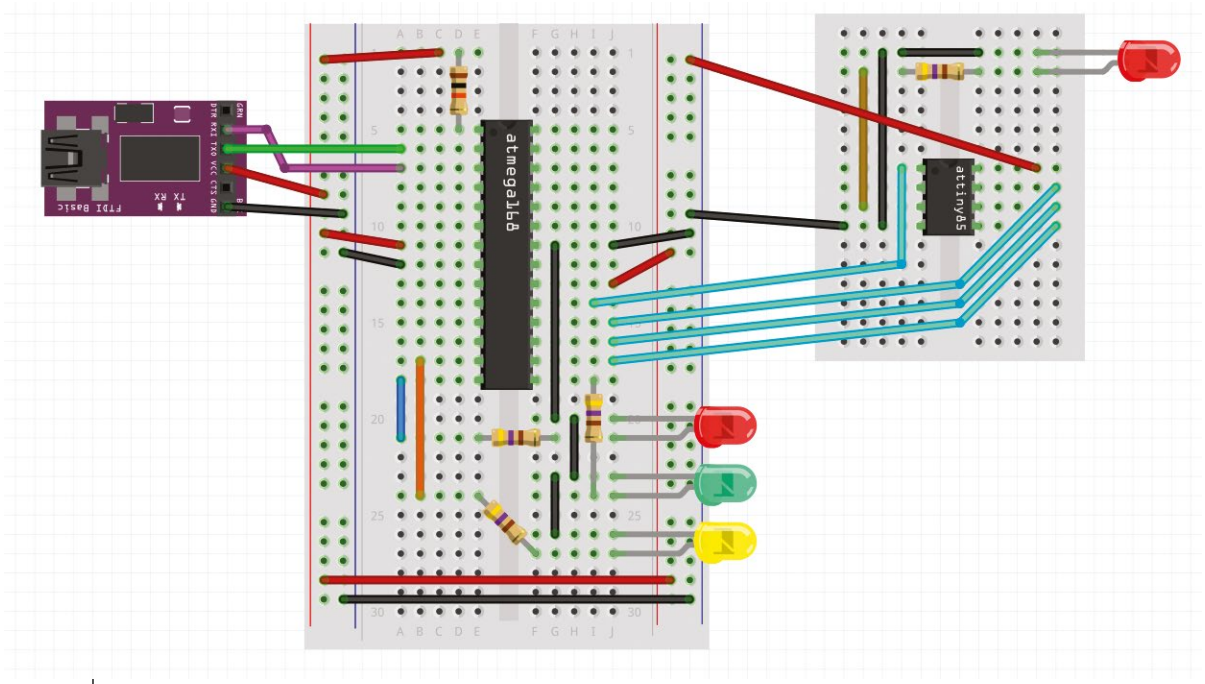
3. PHASE 3 : UTILISER LE MONTAGE ATMEGA168 POUR PROGRAMMER UN ATTINY85

Pour vérifier le fonctionnement de notre programmeur tout neuf, tentons de programmer un autre microcontrôleur. Pour diversifier un peu, utilisons un ATtiny85. Ceci rejoint les explications données dans le premier numéro de Hackable où nous avons parlé de ce microcontrôleur minuscule. Mais depuis la publication de l'article, les choses ont sensiblement changé.

Le support pour les ATtiny85, 45, 85 et 44 ne nécessite plus d'installation séparée de l'environnement. Tout comme avec le support ATmega168 et ATmega328p, à présent ceci passe par le *Boards Manager*. Vous devez donc, dans les préférences de l'environnement, ajouter une URL : **https://raw.githubusercontent.com/damellis/attiny/ide-1.6.x-boards-manager/package_damellis_attiny_index.json** en plus de celle déjà présente, en les séparant par une virgule.

Dès lors, le *Boards Manager* vous permet d'installer le support « attiny » de D. Mellis, exactement comme nous l'avons fait précédemment. Le menu ***Outil***, ***Type de carte*** permet maintenant de

coromonadalix



choisir « ATtiny ». Un sous-menu permet de choisir le modèle exact ainsi que la source d'horloge, tout comme pour le montage précédent.

Il ne nous reste plus qu'à assembler le tout : voir montage ci-dessus.

On retrouve ici la connexion à l'alimentation (VCC) et à la masse, respectivement sur les broches 4 et 8 de l'ATtiny85. L'ordre des broches MOSI, MISO et SCK est identique entre l'Atmega168 et l'ATtiny85, seule la position change avec les broches 5, 6 et 7. Le reset est sur la broche 1 comme pour presque tous les Atmel AVR. Enfin, pour test, nous n'oublions pas de connecter une résistance et une led à la broche 3 de ATtiny85, correspondant à PB4 dans la nomenclature Atmel et à la sortie 4 dans celle d'Arduino. Vous avez remarqué, on a utilisé la même broche dans tous les montages. Mais attention, le vrai nom de la broche est PB4 pour l'ATtiny85, mais PD4 pour l'ATmega168 !

Il ne reste plus qu'à tester :

- ouvrez le croquis Blink ;
- changez la sortie utilisée en **4** ;
- choisissez le type de carte ATtiny ;
- sélectionnez le modèle ATtiny85 ;
- assurez-vous d'utiliser « Arduino as ISP » comme programmeur ;
- réglez l'horloge sur 8 Mhz interne ;
- gravez la séquence d'initialisation ;
- chargez le croquis dans la cible (téléversez).

Normalement si tout est parfaitement branché et configuré, la led connectée à l'ATtiny85 clignote, 1 seconde allumée, 2 secondes éteinte. Bravo, vous avez un montage vous permettant de faire office de programmeur ISP et n'avez plus besoin d'utiliser une carte Arduino Uno pour cela. Il ne vous reste plus qu'à vous libérer de la platine à essai en soudant le tout bien proprement sur une plaque pastillée (ou en créant un vrai circuit imprimé).

4. BILAN

Purement financièrement, les choses sont claires, mais peuvent être vues de deux façons distinctes.

D'un côté, nous avons de quoi maintenant remplacer les cartes Arduino de tous nos projets par de simples microcontrôleurs AVR. Selon le projet, ceci revient à

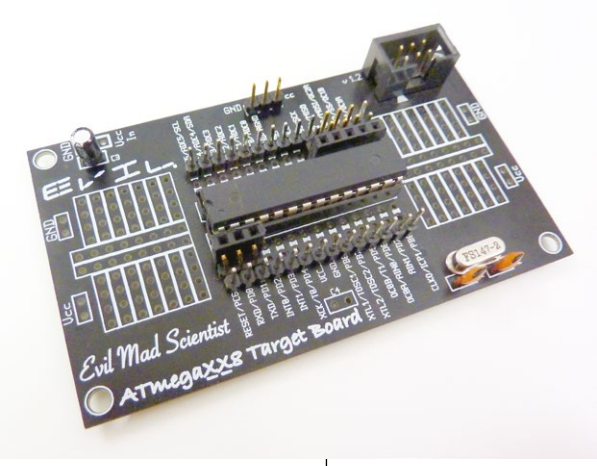
coromonadalix

troquer une carte à 20 euros contre un microcontrôleur presque 10 fois moins cher (selon la source). Nos amis de Hong Kong vous proposeront des ATmega328p pour moins de 2 euros sur eBay, port offert. Bien entendu, il faudra ajouter à cela de quoi alimenter le montage (câble, condensateur, régulateur, etc.) et éventuellement une plaque pastillée pour faire tenir le tout ensemble (vous ne pouvez pas éternellement utiliser des platines à essais dans des boites de surimi). Si le projet implique une communication avec un PC ou un Mac, un adaptateur USB/série sera nécessaire. Là encore, la solution super-économique vient d'Asie. Il semblerait que le convertisseur CH340G fasse actuellement un carton à Shenzen et offre une solution ultime pour remplacer les puces FTDI, Prolific, etc. On trouve ainsi des adaptateurs complets (5v/3,3V) à moins de 1,70€ pièce (toujours du eBay) port offert... et avec un câble 5 connecteurs ! Ils sont trop forts ces Chinois !

D'un autre côté, en parlant de chinois, vous trouvez un peu partout des programmeurs USBasp/USBisp, utilisant un ATmega8 ou un ATtiny2313 pour 4 ou 5€. Ceux-ci se passent de convertisseur USB/série et dialoguent directement en USB avec le PC ou le Mac. Certes, ceci nécessite l'installation d'un pilote, mais l'économie est prodigieuse, même face à la solution ATmega168 proposée ici.

Tout dépend donc de l'objectif que vous poursuivez. Personnellement, ayant quelques années de vécu électronique avec des AVR, j'ai accumulé une tripotée de programmeurs ISP, de l'AVR Dragon original aux montages maison en passant par un programmeur allemand vert fluo et le Pololu USB AVR Programmer. Chaque solution a ses qualités et influe sur des préférences personnelles.

Une solution en boîtier est plus facilement transportable. L'AVR Dragon permet de programmer plus de composants et de se sortir de faux pas avec les fusibles (raison initiale de l'achat) et les montages maison sont réutilisables pour d'autres usages. De plus, il n'est pas rare de travailler sur deux projets ou deux montages d'un même projet en même temps. Connecter/déconnecter le programmeur devient vite pénible et c'est ce qui déclenche généralement un achat. Idem pour la programmation et la communication avec le montage : le programmeur Pololu apparaît au PC sous forme de deux ports séries, un pour le programmateur et un pour tester le montage. C'est très pratique.

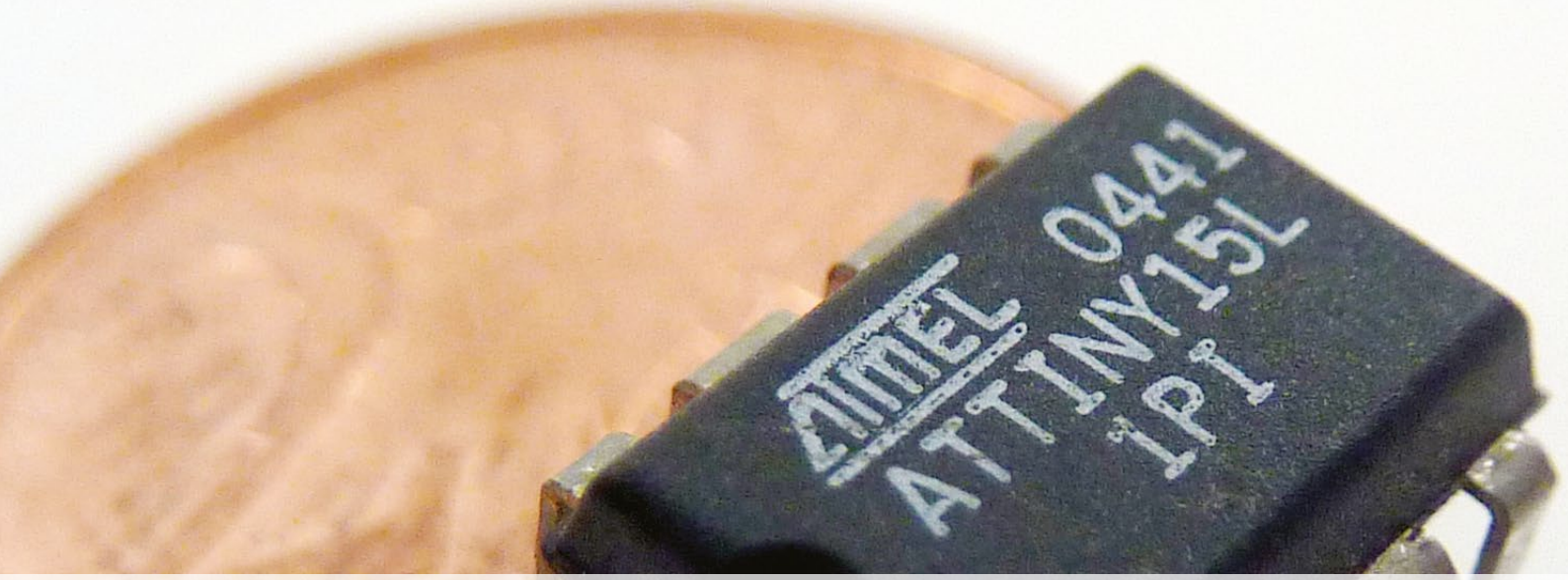


Si vous voulez vous débarrasser de la platine à essai sans pour autant utiliser une plaque pastillée ou un circuit imprimé maison, Evil Mad Scientist vend à $3 des petites cartes double face permettant de rapidement assembler un montage à base d'ATmega48/88/168/328. Une déclinaison est également disponible au même prix pour l'ATmega2313/4313 avec en plus un emplacement pour connecteur jack d'alimentation, manquant cruellement sur la version ATmega48/88/168/328.

Il existe une autre solution dont nous n'avons pas parlé ici : installer le bootloader Arduino dans un ATmega328p sur platine à essai. Ceci permet, à l'aide d'un convertisseur USB/série, de programmer le microcontrôleur comme cela se fait sur une carte Arduino. Cette technique se limite cependant à un certain nombre de modèles d'AVR et implique le même temps d'attente au démarrage qu'avec une carte Arduino. Personnellement, je trouve que l'intérêt est assez limité.

Au final, créer son programmeur est moins cher que d'utiliser une carte Arduino, mais plus cher que d'opter pour un programmeur ISP chinois. Le plaisir, l'expérience acquise et la satisfaction personnelle (quand ça marche), ça, vous ne le trouverez pas sur un site d'enchères en ligne, car ça n'a pas de prix. **DB**

coromonadalix



ARDUINO PRESQUE SANS ARDUINO, OU COMMENT PROGRAMMER SANS LES ROULETTES

Denis Bodor

Arduino, ou Genuino, a chamboulé le monde de l'électronique hobbyiste… et pas seulement. Le projet, les cartes, l'environnement et les bibliothèques ont massivement démocratisé la programmation de ces fantastiques petites puces que sont les microcontrôleurs. Pour autant, cette activité n'a rien de récente auprès des amateurs d'électronique, elle s'est simplement ouverte au plus grand nombre avec Arduino. Mais saviez-vous qu'il est tout à fait possible de programmer le microcontrôleur d'une carte Arduino de manière « native » ?

coromonadalix

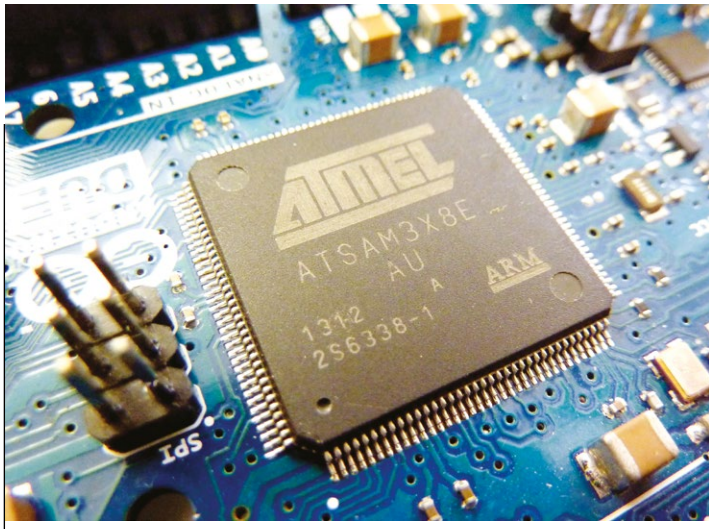
Avant toutes choses, plantons le décor et rappelons ce qu'est Arduino. Il s'agit avant tout d'une carte comprenant un microcontrôleur Atmel AVR, un circuit d'alimentation, un convertisseur USB/série pour la communication avec le PC/Mac et une poignée de composants (résistances, quartz, condensateurs) et connecteurs pour faire fonctionner le tout.

À cela s'ajoute un environnement permettant d'écrire des programmes appelés croquis (*sketch*) ainsi qu'un ensemble de petits bouts de code facilitant l'utilisation des fonctionnalités de la carte : les bibliothèques. Cet ensemble logiciel a pour objectif de simplifier l'utilisation des cartes Arduino en mettant à disposition de l'utilisateur des éléments de programmation rapidement et instinctivement compréhensibles. Ces éléments sont hérités de Wiring, lui-même s'inspirant grandement de Processing (philosophie, langage et environnement).

La partie d'Arduino s'installant sur PC ou Mac et vous permettant de simplement débuter en programmation, est composée d'un environnement de développement (IDE) et de ce qu'on appelle une couche d'abstraction. Son but est de masquer les parties qui peuvent paraître complexes dans la prise en main du microcontrôleur équipant la carte.

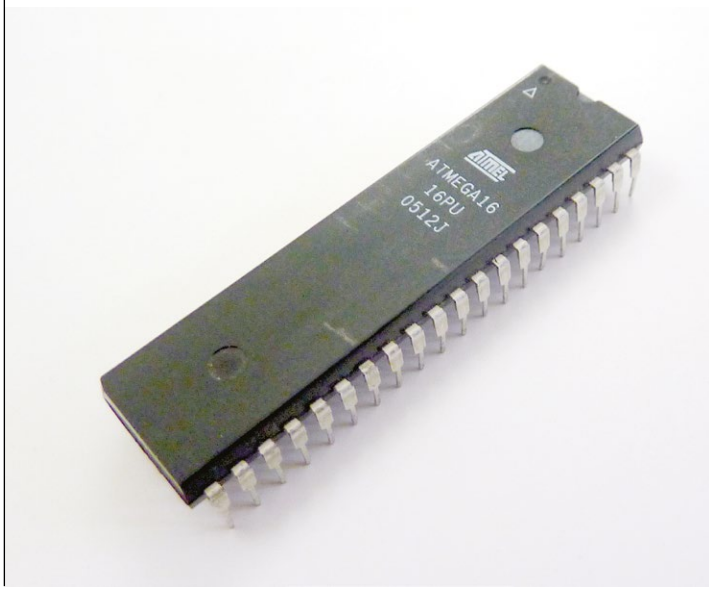
En effet, il y a tout un monde entre le croquis que vous allez écrire et le code qui va effectivement trouver sa place dans la mémoire de la carte. On peut facilement illustrer cela en imaginant différents niveaux de profondeur dans un océan. Avec les facilités offertes par Arduino, vous vous trouvez entre la surface et les quelques mètres de profondeur (parfait pour les palmes et un tuba). Tout en bas, se trouve le matériel lui-même (scaphandre de plongée) et juste au-dessus, le code en langage machine (binaire) exécuté directement par le matériel ou plus exactement l'ALU (unité arithmétique et logique, ou *Arithmetic Logic Unit* en anglais) dans le microcontrôleur.

Une partie des niveaux les plus bas représente le territoire presque exclusif des outils de développement (compilateur, éditeur de liens, etc.). D'autres, plus en haut, sont ceux des programmeurs les plus exigeants ou ayant besoin d'un maximum de contrôle (langage assembleur). Enfin, juste sous le niveau Arduino se trouve le domaine de la programmation C/C++.



Les cartes Arduino Due n'utilisent pas de microcontrôleurs AVR, mais un ATSAM3 totalement différent. Au plus bas niveau, cette différence se fait grandement ressentir, mais en restant dans les facilités offertes par l'environnement Arduino la transition est aisée. Pour la plupart des croquis, aucune modification n'est nécessaire. C'est là l'un des nombreux avantages du « langage Arduino », mais cela se fait au détriment de la pleine maîtrise de la plateforme.

Avant l'arrivée d'Arduino, les microcontrôleurs Atmel AVR étaient utilisés de longue date, aussi bien par les professionnels que les hobbyistes. Les choses étaient simplement moins accessibles et demandaient davantage de travail pour obtenir un premier résultat. Ci-dessous un ATmega16 de 2005, un « gros » microcontrôleur pour l'époque, mais pas très différent des ATmega328p des Arduino.



Le langage est assez similaire à celui utilisé pour les croquis Arduino, mais les programmes utilisent directement les fonctionnalités offertes par le microcontrôleur ainsi que quelques facilités fournies par les briques qui forment les fondations de l'environnement Arduino (l'ARV-Libc en particulier). Notez que je fais ici délibérément abstraction de la carte Arduino Due, reposant sur un microcontrôleur, certes de chez Atmel, mais totalement différent des AVR de la plupart des cartes Arduino (Uno, Leonardo, etc.). La même logique peut s'appliquer, mais la mise en pratique est différente.

L'idée derrière cet article est de tout simplement lever un peu le voile sur ce qui se cache en coulisse, derrière les croquis que vous manipulez ou écrivez. Ceci pour quelques raisons parmi lesquelles :

- comprendre la nature des facilités offertes par l'environnement Arduino afin de juger au mieux des bénéfices qui sinon risqueraient de vous paraître naturels, communs, voire exigibles ;
- entrevoir une évolution possible de votre façon de programmer et donc en apprendre davantage sur la maîtrise du matériel ;
- découvrir des mécanismes qui sont relativement communs dans le monde des microcontrôleurs et pourront être utilisés pour accéder à d'autres architectures, plateformes ou processeurs ;
- explorer une voie permettant l'optimisation de vos programmes, aussi bien en termes de gain en vitesse ou de concision du code (économie de mémoire), mais aussi d'économie d'énergie en reposant sur des mécanismes qui ne sont pas accessibles via l'environnement Arduino ;
- faire tout simplement connaissance avec la programmation sur microcontrôleur dans sa déclinaison la plus « nature » possible.

Ce que cet article n'est pas en revanche, est une introduction ou un cours de programmation en C avec l'AVR-Libc. Mon objectif ici est de simplement vous désigner une porte et l'entrouvrir pour vous montrer ce qui se trouve de l'autre côté. Prendre la décision de la franchir et de vous lancer dans ce domaine est vôtre et implique inéluctablement la lecture totale ou partielle de la documentation (*datasheet*) Atmel du microcontrôleur utilisé. Celle-ci décrit l'ensemble du composant et son utilisation au niveau le plus bas, elle se compose généralement de 600 pages ou plus (en anglais). C'est long, c'est dur, mais tout y est décrit.

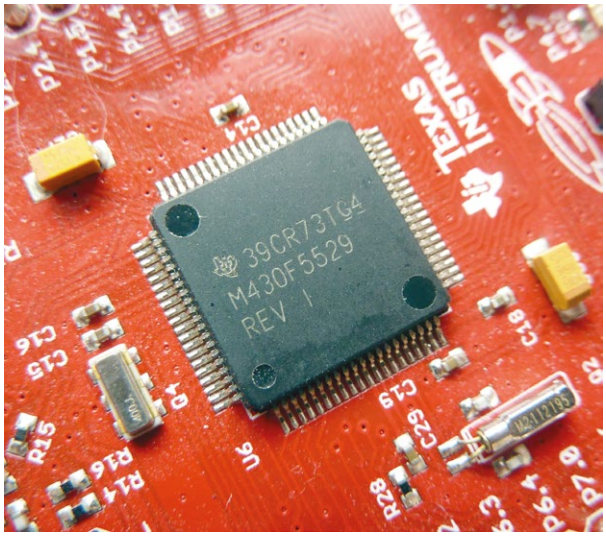
Cet article n'est pas non plus une façon de vous faire peur, et encore moins le présage d'un changement du contenu du magazine. Vous allez le constater, sans les facilités livrées avec Arduino (les bibliothèques internes) il y a généralement plus de code à écrire et plus de connaissances à avoir sur le fonctionnement du microcontrôleur. C'est là quelque chose qui s'apprend, de la même façon que vous avez appris ou apprenez à écrire des croquis. Si vous vous lancez dans cette voie, au fil du temps vous allez développer vos propres facilités et vos petites routines réutilisables qui vous correspondent. Vous vous composerez votre propre environnement facilitant l'écriture de programmes et tout cela vous paraîtra naturel.

C'est comme le vélo. Cela semble difficile, voire impossible au début lorsque papa dévisse les petites roulettes. Mais après quelque temps et quelques chutes, on finit par trouver son équilibre et peu après, on va beaucoup, beaucoup plus vite qu'avec les roulettes...

1. LE TRAVAIL DE L'ENVIRONNEMENT ARDUINO

L'environnement Arduino, en dehors de l'écosystème de cartes et de *shields*, se compose de plusieurs parties. Nous avons tout d'abord l'interface graphique intégrant un éditeur pour saisir le code et des menus/boutons pour déclencher les

coromonadalix



Les Atmel AVR ne sont pas les seuls à pouvoir être programmés en C. Ce langage est, avec l'assembleur, le dénominateur commun à presque tous ces composants qui repose également sur les mêmes mécanismes (ports, registres, timers, interruptions, etc.). L'expérience acquise en programmant un microcontrôleur est souvent transposable à 90% sur un autre. Ici, un MSP430 d'une carte TI Launchpad. Cette plateforme possède aussi son environnement à la « Arduino », c'est Energia.

opérations utiles. Cette interface est écrite en Java afin de facilement pouvoir être déclinée pour plusieurs systèmes (Windows, Mac OS, GNU/Linux).

L'interface sert d'interprète entre vous et les outils qui permettent de traiter le code, d'en faire quelque chose d'intelligible pour le microcontrôleur et pour le charger dans sa mémoire. Ces outils existaient bien avant l'arrivée d'Arduino et se nomment AVR GCC, AVR-Libc ou encore AVRdude. Ils sont utilisables individuellement pour faire exactement la même chose que ce que vous propose l'environnement Arduino. Généralement, ils sont tout de même contrôlés de concert par un outil unique. Celui-ci n'est cependant pas graphique et ne se limite pas au code de l'Arduino, mais concerne la construction de n'importe quel logiciel : il s'agit de GNU Make. La technique donc, là encore, n'a rien de nouveau. Arduino est simplement plus intuitif, pour un utilisateur coutumier des interfaces graphiques.

Pour le chargement du programme dans le microcontrôleur, Arduino ajoute également son grain de sel, mais là aussi, en reposant sur quelque chose qui existait bien avant son arrivée. Le mécanisme de bootloader ou « séquence d'initialisation » dans le jargon Arduino, permet de se passer de la nécessité d'avoir un programmeur ISP. Grâce au bootloader, les cartes Arduino et plus exactement le microcontrôleur lui-même sont capables de se programmer. Il démarre dans un mode particulier afin de recevoir le code à utiliser et l'enregistre dans sa mémoire avant de redémarrer pour l'exécuter, ceci sans avoir à utiliser de circuit, de matériel ou de montage supplémentaire.

Enfin, en termes de code et de programmation, Arduino facilite la compréhension du fonctionnement global et/ou impose ses mécanismes. Ces choix et ces ajouts font qu'il est courant de parler du « langage Arduino » alors qu'il s'agit simplement de C/C++. Il est cependant important de faire la distinction, car les habitudes que vous prenez en programmant vos cartes Arduino ne sont pas directement transposables dans le monde de la programmation C/C++. Un excellent exemple concerne **`setup()`** et **`loop()`**. De telles fonctions n'existent pas en programmation C/C++ standard où une et une seule fonction est appelée au démarrage du programme : **`main()`**. L'exécution de cette fonction, dans le cas d'un microcontrôleur, n'est pas censée se terminer car, dans une telle éventualité, ceci signifierait que pour la relancer il faudrait réinitialiser le microcontrôleur. En programmant en C/C++ donc, il faut que le programmeur pense à inclure une boucle sans fin dans son programme (**`for`**, **`while`**, etc.), sinon il ne s'exécutera qu'une fois.

Arduino dissimule cela en obligeant le programmeur à écrire **`setup()`** et **`loop()`** et, en coulisse, compose pourtant effectivement bien **`main()`**, ainsi :

```
int main(void)
{
  setup();

  for (;;) {
    loop();
  }

  return 0;
}
```

Vous le voyez, Arduino simplifie les choses en les faisant à votre place afin de limiter au maximum les erreurs et ne pas imposer un apprentissage préalable pouvant retarder la réalisation d'un premier projet (pédagogiquement, le premier résultat viable est très important). Bien entendu, c'est comme en cuisine, une préparation où il suffit d'ajouter un œuf vous permet d'avoir un cake plus facilement et rapidement, mais vous prive d'une partie de la maîtrise de la recette complète. Ce qu'on gagne d'un côté on le perd toujours par ailleurs...

2. LE MEILLEUR DES DEUX MONDES

Nous avons décrit ici deux univers existant à des « profondeurs d'abstraction » différentes vous permettant de choisir le niveau de maîtrise et de facilité que vous souhaitez utiliser. Mais ces deux mondes sont frontaliers, voire se juxtaposent dans la réalité.

Ainsi il vous est parfaitement possible de développer vos croquis Arduino, en « langage Arduino », avec des outils classiques tels qu'un bon éditeur (Vim), le compilateur AVR GCC, la bibliothèque AVR-Libc, AVRdude et le bootloader Arduino, le tout contrôlé par GNU Make et un **Makefile**. Il est même possible d'utiliser des IDE bien plus complets comme Eclipse pour vos croquis.

Inversement, rien ne vous empêche de programmer un microcontrôleur Atmel AVR sans carte Arduino ou encore, et c'est l'objet du présent article, d'utiliser l'interface graphique de l'environnement Arduino pour programmer en C/C++ sans les facilités/simplifications telles que **setup()** et **loop()**. Il vous suffit d'écrire votre croquis, qui est alors un code source en C ou C++, sans utiliser ces facilités et les bibliothèques Arduino. L'environnement comme les outils sous-jacents se débrouillent parfaitement avec cela et aucune configuration particulière n'est nécessaire, pas même une case à cocher dans les préférences.

3. UN SIMPLE CROQUIS ARDUINO

Entrons dans le vif du sujet avec une comparaison entre un croquis Arduino et son équivalent en programmation « classique » C. Notre croquis se veut délibérément simple : nous faisons clignoter la led intégrée à la carte Arduino Uno tout en envoyant sur le moniteur série le traditionnel message « Coucou monde » :

coromonadalix

4. LA MÊME CHOSE VERSION C/AVR

Voici à présent la même chose, mais sans les fonctions et facilités ajoutées par Arduino. Ce programme est donc assez similaire à ce que votre carte va effectivement exécuter après traitement par une succession d'étapes de traduction du croquis vers le programme en C/C++, qui est ensuite compilé et chargé dans la mémoire flash du microcontrôleur :


```
// vitesse du port série
#define BAUD_RATE 115200

// initialisation du port série
void serial_init() {
  // calcul valeur du registre
  #if BAUD_RATE < 57600
    // basse vitesse
    uint16_t UBRR0_value = ((F_CPU / (8L * BAUD_RATE)) - 1)/2 ;
    UCSR0A &= ~(1 << U2X0); // doubleur vitesse off
  #else
    // haute vitesse
    uint16_t UBRR0_value = ((F_CPU / (4L * BAUD_RATE)) - 1)/2;
    UCSR0A |= (1 << U2X0);  // doubleur vitesse on
  #endif

  // écriture des registres en deux fois (haut/bas)
  UBRR0H = UBRR0_value >> 8;
  UBRR0L = UBRR0_value;

  // activer l'émission
  UCSR0B |= 1<<TXEN0;

  // le reste est en valeur par défaut
  // 8 bit, pas de parité, 1 bit de stop
}

// fonction d'envoi d'un caractère
static void putchr(char c) {
  // on attend que ce soit disponible/utilisable
  loop_until_bit_is_set(UCSR0A, UDRE0);
  // écriture registre = envoi de l'octet
  UDR0 = c;
}

// fonction d'envoi d'une chaîne de caractères
// utilisant putchr()
static void printstr(const char *s) {
```

```
  // on boucle tant qu'il y a des caractères
  while (*s) {
    // et on change LF en CR si nécessaire
    if (*s == '\n')
      putchr('\r');
    putchr(*s++);
  }
}

// fonction principale
int main(void) {
  // appel de notre fonction pour configurer le port
  serial_init();

  // broche PB5 en sortie
  DDRB |= _BV(PB5);

  // pause en millisecondes (fournie par AVR Libc)
  _delay_ms(500);

  // boucle sans fin
  for(;;) {
    // port à 1 (+5V)
    PORTB |= _BV(PB5);
    // pause
    _delay_ms(200);
    // port à 0 (0V)
    PORTB &= ~_BV(PB5);

    // envoi du message sur le port série
    printstr("Coucou monde\n");

    // pause
    _delay_ms(500);
  }

  // on n'arrive jamais ici
  return 0;
}
```

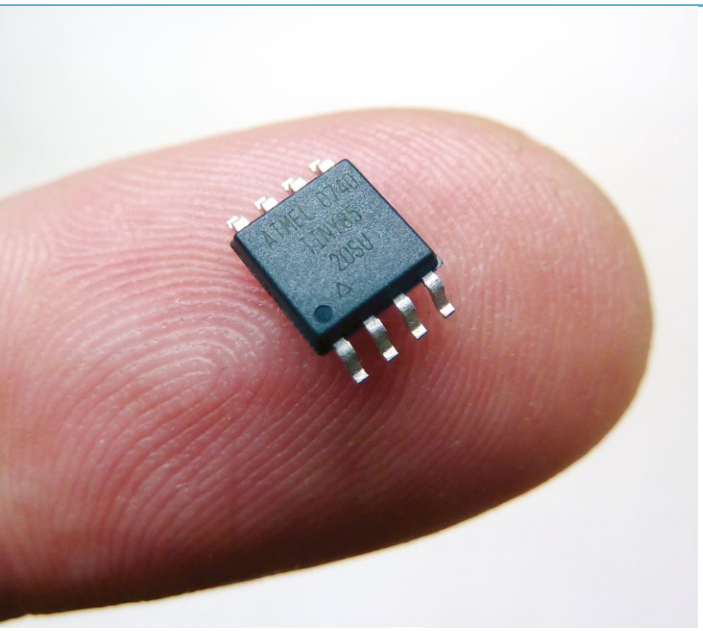
Arduino

Ouch ! Le moins qu'on puisse dire c'est que c'est un peu plus... touffu ! N'oubliez pas, avec ce type de programmation, vous êtes aux commandes directes du microcontrôleur. Ceci signifie aussi que vous devez tout faire.

Les commentaires dans le code devraient être suffisants pour en comprendre la logique. Cependant, encore faut-il savoir de quoi on parle et quels sont les mécanismes en œuvre.

En premier lieu, nous avons l'architecture même du code. En ce qui concerne l'utilisation de la communication série, par exemple, nous divisons le travail en briques élémentaires. Nous avons d'une part **`serial_init()`** permettant la configuration (un peu l'équivalent du **`Serial.begin()`**, puis une fonction

coromonadalix



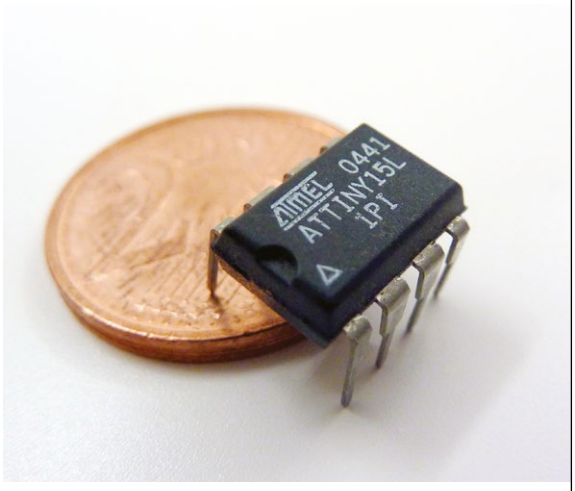
L'ATtiny85 est un microcontrôleur utilisable avec l'environnement Arduino. Généralement utilisé au format PDIP parfaitement adapté pour une platine à essais, le voici en SOIC. Cette petite chose de 5 mm de côté possède 8 Ko de flash pour les programmes et 512 octets de mémoire (SRAM). Mieux vaut faire attention à sa façon de programmer pour ne pas se retrouver coincé faute de place.

pour envoyer un caractère et, elle-même est utilisée pour envoyer une chaîne de caractères. Ce type de répartition en tâches minimales est tout à fait caractéristique d'une programmation en C.

Un autre élément qui est totalement absent de l'écriture de croquis Arduino (mais tantôt présent dans la création de bibliothèques) est la notion de registre. Un registre est un emplacement mémoire interne au processeur. Ce n'est pas la mémoire vive ou SRAM, bien que selon le type de microcontrôleur et son architecture les deux notions peuvent se rejoindre.

Un registre est une case mémoire ayant une fonction et une utilité très particulière. Dans notre programme, par exemple, nous utilisons **DDRB** et **PORTB**. Le premier permet de stocker une valeur sur 8 bits où chaque bit détermine si une broche est en entrée (**0**) ou en sortie (**1**). La notion de port n'existe pas avec Arduino, les broches de la carte sont simplement numérotées, mais pour le microcontrôleur il y a un port B avec des broches désignées par PB0 à PB7. PB5 correspond à la sixième broche du port B, qui est la broche 13 d'une Arduino Uno.

Pour changer l'état d'une broche sur un port, un autre registre est utilisé, comme ici avec **PORTB**. Ici, les 8 bits correspondent également aux 8 broches du port, mais cette fois un **0** correspond à une mise à 0 volt (la masse) et un **1** à la sortie à l'état haut, soit +5 volts.



L'ATiny15 ressemble très fortement à l'ATiny45 ou 85, mais celui-ci ne dispose pas des mêmes ressources internes. Ce composant se programme généralement en assembleur, car il ne possède pas de pile en SRAM nécessaire pour le compilateur C. D'ailleurs, il n'a pas de SRAM du tout...

Les registres contiennent des valeurs sur 8 bits et donc entre 0 et 255. Les registres peuvent être lus et écrits, mais en entier. Pour changer un bit, il faut donc lire le registre, changer un bit et le réécrire. On met en œuvre des opérations logiques pour faire cela en un minimum d'étapes. Nous avons ici un magnifique exemple puisque nous voulons changer le bit 5 de **PORTB** d'abord à **1** puis à **0**.

Prenons la première opération :

```
PORTB |= _BV(PB5);
```

_BV(PB5) est une macro. Le préprocesseur, va remplacer cela en **(1 << (PB5))** puis encore simplifier en **32**, car 32 c'est **00100000** en binaire (le bit 5, le sixième en partant de droite, à **1** et tout le reste à **0**), et **00100000** c'est **00000001** décalé 5 fois vers la gauche (**<<**).

coromonadalix

Le |= correspond à « OU EGAL », soit « on prend ce qu'il y a dans PORTB et on fait une opération logique OU entre tous les bits et notre 32, pour ensuite mettre le résultat dans PORTB ». Le principe est le même avec le passage de la broche à 0, mais avec :

```
PORTB &= ~_BV(PB5);
```

Ne revenons pas sur la macro, la partie importante est ~ et &=. Le premier correspond à l'opération d'inversion, notre 00100000 devient 11011111. C'est cette valeur que nous utilisons pour notre opération logique. Mais cette fois ce n'est pas un « OU EGAL », mais un « ET EGAL », soit l'actuel contenu du registre ET notre valeur qui est le résultat du bit 5 à un, mais inversé. Ce qui donne :

```
  11100111    valeur actuelle
& 11011111    notre 32 inversée
  --------
  11000111    le résultat
```

Le bit 5 est bel et bien passé de 1 à 0 et les autres bits n'ont pas changé. Nous venons de passer la broche 5 du port B à 0 sans toucher aux autres.

Vous l'avez compris, bien que nous ne sommes pas encore au niveau du langage machine ou de l'assembleur, nous nous rapprochons drastiquement du matériel, des bits, des registres, etc. À ce niveau, mieux vaut potasser l'algèbre de Boole, les opérations logiques et les tables de vérité.

J'arrêterai là les explications autour de ce code. Le but, je l'ai dit plus haut, n'est pas de vous enseigner le développement en C sur microcontrôleur AVR, mais plutôt de vous donner une idée de ce en quoi cela consiste. Je pense que la démonstration du niveau d'intimité entre le développeur (potentiellement vous) et le matériel, est relativement explicite.

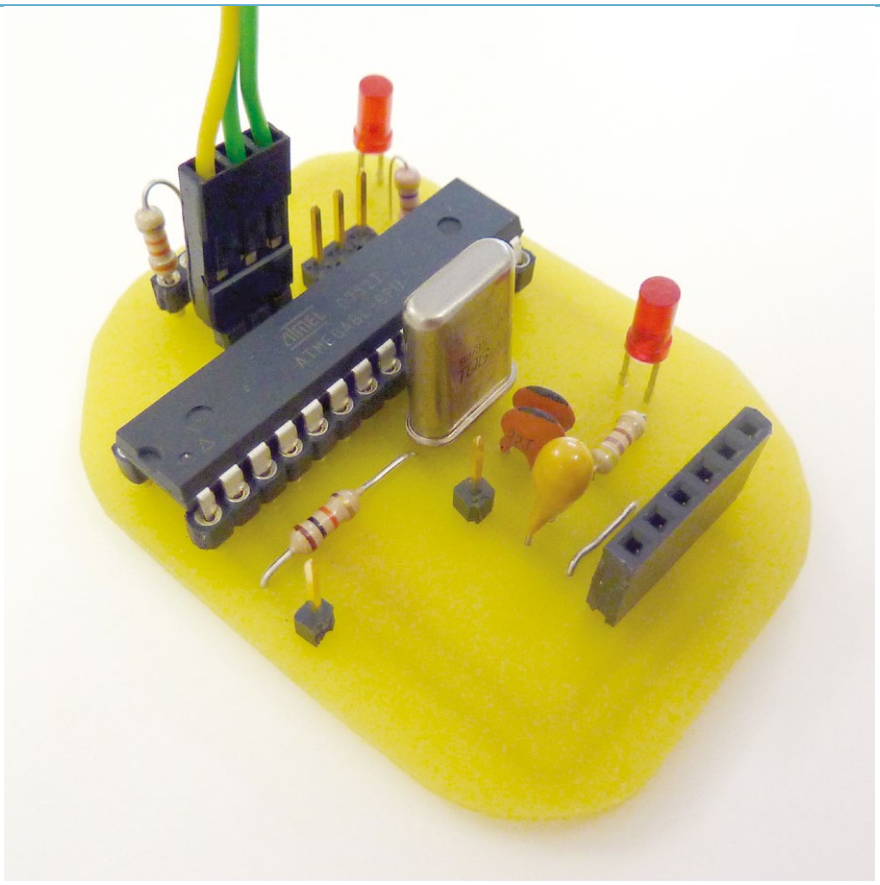
5. LES PLUS, LES MOINS...

À ce stade de la lecture de l'article, si vous avez tenu jusqu'ici, il est probable que vous soyez prêt à aduler Massimo Banzi ainsi que toute l'équipe de développement du projet Arduino pour leur travail de démocratisation de la programmation sur microcontrôleur. Et je suppose qu'une question vous taraude l'esprit par la même occasion : « mais pourquoi diable écrirais-je autre chose que des croquis Arduino si ce n'est en raison d'une intense pulsion masochiste ? ».

Il existe plein de raisons, mais la plus évidente est simplement donnée par l'environnement Arduino lui-même sous la forme du message qui apparaît juste après la « vérification » (compilation) du code. Pour le croquis Arduino nous avons : « *Le croquis utilise 2 410 octets (7%) de l'espace de stockage de programmes[...] Les variables globales utilisent 196 octets (9%) de mémoire dynamique[...]* ».

Et pour le code C équivalent : « *Le croquis utilise 310 octets (0%) de l'espace de stockage de programmes[...] Les variables globales utilisent 14 octets (0%) de mémoire dynamique[...]* ».

Nous avons ici presque 8 fois moins de mémoire flash utilisée et 14 fois moins de mémoire vive (SRAM). C'est un argument plutôt convaincant même s'il est vrai qu'un microcontrôleur ATmega328P comme celui d'une Arduino Uno dispose de 32 Ko de flash et 2 Ko de SRAM. L'argument aura d'autant plus de poids si vous jetez votre dévolu sur un ATtiny45, par exemple (4 Ko de flash, 256 octets de SRAM). Et disons-le tout à fait clairement au risque de contrarier certains : passer d'un microcontrôleur avec 32 Ko de flash à un autre avec 256 Ko simplement parce que son code occupe 40 Ko est un réflexe de mauvais programmeur !



Un « vieux » montage éthylomètre autour d'un ATmega8 avec un quartz comme oscillateur externe afin de garantir le minimum d'erreur pour la communication série (via un adaptateur USB/série non présent). Les premières cartes Arduino utilisaient également des ATmega8 qui comportent les mêmes périphériques que le 168 et le 328. La seule différence est la quantité de flash et de SRAM disponible.

Un bon programmeur ne change pas de processeur lorsque son programme est lent, il optimise d'abord son code.

D'où viennent de telles différences ? D'une accumulation de petites choses. Le `Serial.begin()` d'Arduino configure le port, mais il le fait en émission comme en réception. Dans notre code en C, nous avons économisé quelques instructions en nous passant de la réception. Le compilateur GCC fait tout ce qu'il peut pour économiser de la mémoire, mais il ne peut pas faire de miracles. Tout ce qui est ajouté pour faciliter la vie au programmeur prend de la place. Créez un nouveau croquis qui ne fait rien, compilez-le et vous constaterez qu'il occupera 450 octets de flash (plus que notre exemple en C) et 9 octets de SRAM. Faites de même avec un équivalent en C :

```
int main() {
  for(;;);
  return 0;
}
```

Et il occupera 134 octets de flash et 0 (oui, zéro) de SRAM.

Tout dépend donc de vos besoins, de votre intérêt pour le domaine, du temps que vous comptez y investir, de la nature de vos projets ou encore de votre curiosité concernant le fonctionnement des microcontrôleurs. Une chose est sûre cependant, si vous avez le temps et l'envie, ceci ne peut qu'améliorer vos talents de programmeur et vous aider à développer de meilleurs programmes.

L'avantage que représente l'utilisation de l'IDE Arduino pour s'essayer à la programmation C « pure » réside principalement dans le fait de ne pas avoir à installer d'outils supplémentaires (comme Atmel Studio) qui peuvent paraître complexes au premier abord, ou encore à devoir assimiler les mécanismes de la composition d'un environnement (ligne de commandes, syntaxe de GNU Make, etc.). On retire donc ses petites roulettes, mais on garde tout de même le joli vélo qu'on connaît bien et avec lequel on est à l'aise... **DB**

coromonadalix

ASSEMBLEZ UN RÉSEAU DE CAPTEURS DE TEMPÉRATURE SANS VOUS RUINER

Denis Bodor

La domotique n'est pas compliquée, mais le domaine est vaste, très vaste. La définition elle-même du terme couvre tout et n'importe quoi plus ou moins en rapport avec l'habitat et l'électronique. Nous allons ici attaquer le sujet par un côté à la fois simple, mais souvent délicat à mettre en œuvre : la mesure de températures et la centralisation de ces informations, le tout avec deux mots en tête : économie et évolutivité.

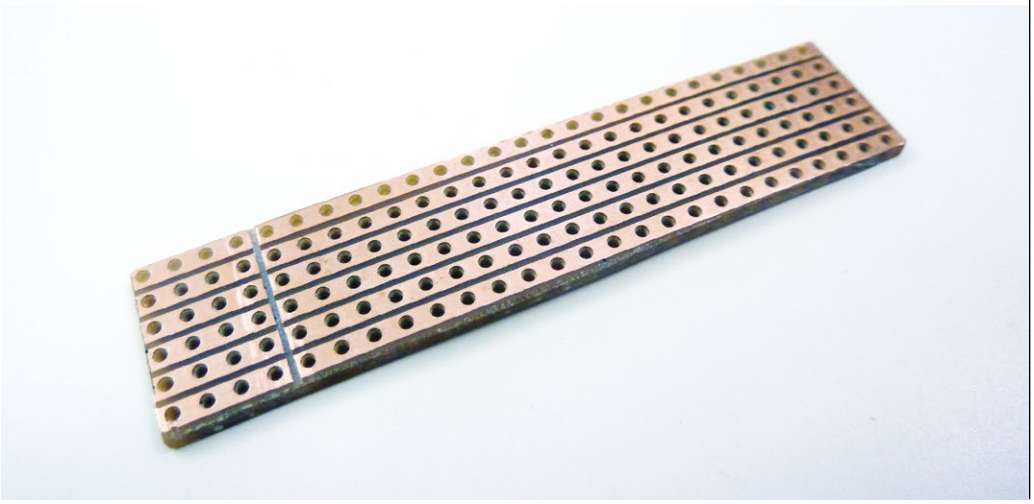
coromonadalix

Comme annoncé nombre de fois dans une série (TV et de livres) à la mode, dont l'intrigue consiste principalement à deviner comment et dans quel ordre vont mourir vos personnages préférés (celle du prince Oberyn m'a fâché définitivement avec la série) : *Winter Is Coming* (l'hiver arrive). Et vous savez ce qui me met hors de moi en hiver ? Ma chaudière, ou plus exactement la façon dont est déclenchée la mise en route du chauffage.

1. LA PROBLÉMATIQUE

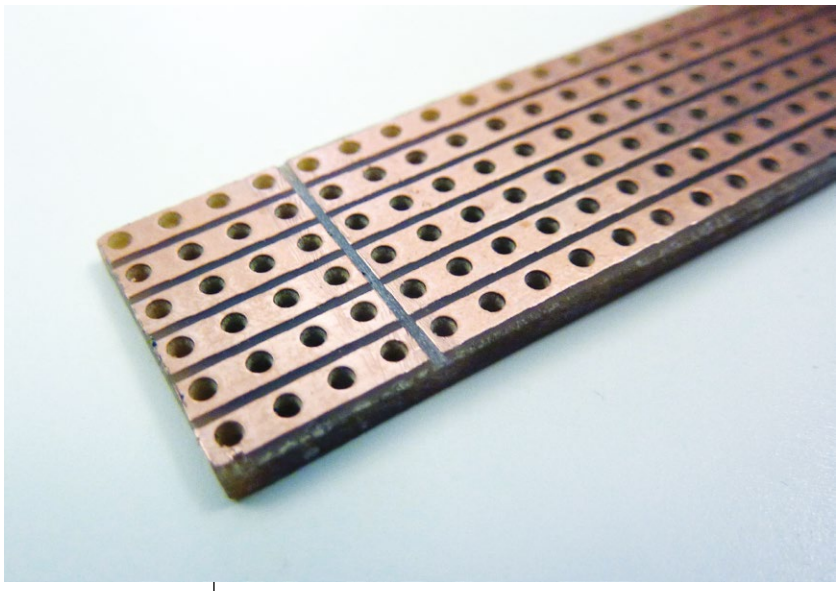
Comme dans bien des installations reposant sur une chaudière au gaz, l'ensemble est contrôlé par un programmateur disposant d'un unique capteur de température (intégré). L'endroit où l'on place ce programmateur détermine donc la mise en fonction de l'ensemble des radiateurs de l'appartement (par ailleurs équipés de têtes thermostatiques). Seulement voilà, j'ai semble-t-il (d'après mon installateur sanitaire), un « profil de chauffage particulier ». Comprenez par là que je ne chauffe jamais mes chambres (une couette, ça fonctionne très bien), ni la cuisine (on peut faire un gratin si besoin) et mon salon/bureau, où je passe le plus clair de mon temps, ne dépasse pas 22°C (et en plus c'est côté sud). C'est donc tout naturellement à cet endroit que j'ai placé le programmateur, c'est ma « pièce à vivre ». Le problème, parce qu'il y a forcément un problème, c'est la salle de bain exposée plein nord, son radiateur et son sèche-serviettes dépendant implicitement de la température du salon.

Idéalement, en bon programmeur, la solution serait une gestion dynamique de la mise en route du chauffage en fonction de critères particuliers du type : si la température de la salle de bain descend sous 18°C, alors chauffer même si le salon est à 22°C... à moins que le salon ne dépasse 26°C côté bureau ou que... vous avez compris ! Ajoutez à cela une possibilité d'une gestion de la ventilation (type VMC) prenant en compte la température extérieure et on se rapproche du système optimal. Avec un immeuble typiquement alsacien (grès et granite) de 1906 (murs de 60 cm), l'inertie thermique est non négligeable et il n'est pas rare qu'en été il fasse plus frais que dehors le soir et en hiver plus froid dedans le jour. Faire circuler l'air dans le bon sens



Le support de base de nos capteurs est une veroboard/stripboard permettant la soudure des composants avec un minimum de connexions à faire à la main. Toute l'astuce consiste à penser le circuit en tenant compte des pistes, un peu comme avec une platine à essais.

coromonadalix



Gros plan sur la coupure des pistes à l'endroit où va se placer le module ESP8266. Un peu de dextérité, de patience et une bonne scie fine et le résultat est propre et très acceptable.

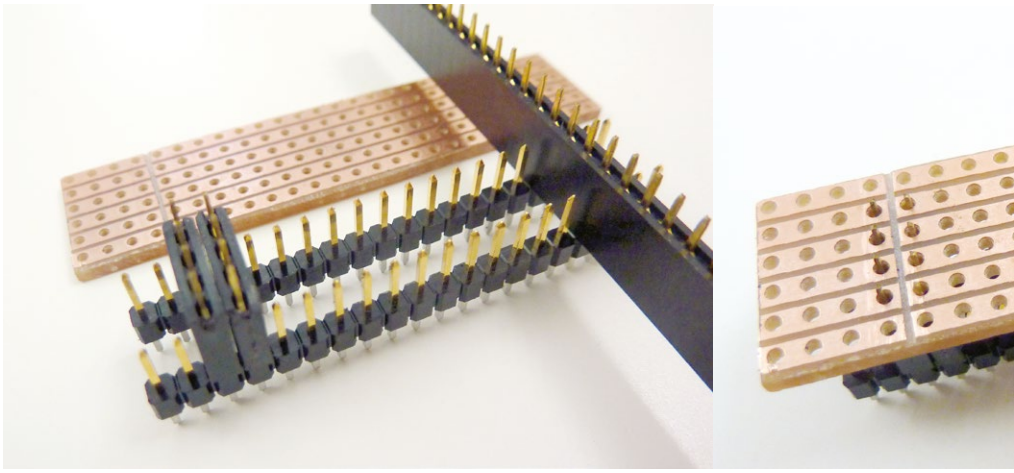
peut s'avérer intelligent, économique et sans doute contribuer à sauver les pandas, les ours blancs et le grand hamster d'Alsace.

La clé du début de solution repose donc sur une mesure judicieuse des températures, non pas dans une seule pièce, mais dans tous les recoins possibles, ou presque. Ceci ne changera pas la performance énergétique d'une habitation de plus de 100 ans royalement isolée à l'aide de planchers en bois et de scories, mais ceci sera, à terme, entièrement paramétrable et surtout amusant (et un peu techno-bling-bling). Notez que ce qui est décrit ici n'est en aucune façon la solution complète finale. Ceci nécessitera bien plus de travail, de recherche et d'expérimentation, en particulier concernant le contrôle de la chaudière via son interface propriétaire selon un protocole peu documenté. Nous nous bornerons ici à la façon de relever des températures et de collecter ces informations.

2. VOUS AVEZ DIT RÉSEAU ?

En plein boum de l'Internet des objets (*IoT* pour les intimes), les solutions domotiques ne manquent pas. Si l'on s'en tient à celles correspondant aux besoins, l'offre se réduit. D'autant plus si l'on ajoute un critère financier strict et le fait d'avoir réellement la main sur le système (souvenez-vous, « si ça ne se démonte pas, ça ne vous appartient pas »). Exit donc des solutions clés en main et penchons-nous sur ce qui existe en petits morceaux assemblables à l'huile de coude.

Nous parlons d'un réseau de capteurs et l'idée même de « tirer des câbles » partout en perçant les murs plein de grès des Vosges



Souder des connecteurs bien droits est toujours un moment pénible. L'astuce consiste ici à utiliser ces mêmes connecteurs mâles et femelles pour créer un petit support. Il suffit ensuite de placer le circuit dessus et souder pour obtenir un résultat bien propre et aligné.

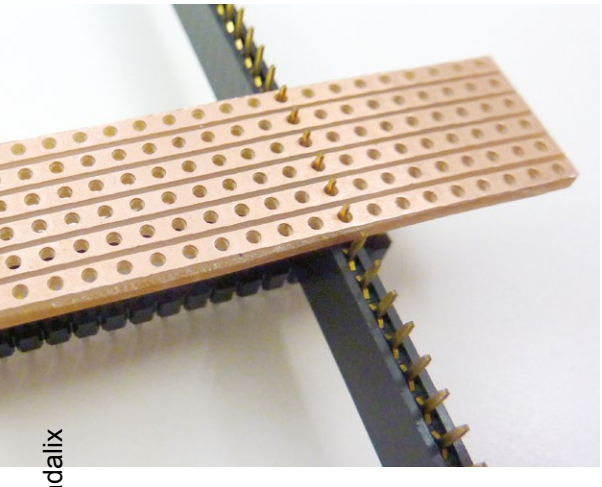
coromonadalix

est clairement rebutante (une fois pour passer le réseau Ethernet est une leçon suffisante). Il sera donc question de liaisons sans fil.

Zigbee et consorts sont hors de prix, le Bluetooth est peu adapté, les solutions nRF24L01 demandent plus d'expérimentation en ce qui me concerne. Reste... le Wifi. Et quel heureux hasard, dans le précédent numéro du magazine, nous avons justement découvert un module très économique, programmable comme une carte Arduino et disponible à profusion dans mes tiroirs : l'ESP8266.

L'architecture générale préalable sera donc la suivante :

- Une myriade de capteurs de températures interfaçables avec les ESP8266. Le choix est simple : Maxim/Dallas DS18b20, un composant très courant, interfacé en 1-Wire ne nécessitant qu'une seule ligne GPIO et donc parfait pour le plus économique et modeste des modules ESP (dit « ESP-01 »).



coromonadalix

- Une solution pour assembler chaque capteur ESP8266 de manière à ce qu'ils ne nécessitent finalement qu'une prise murale.
- Une centralisation des informations sur une Raspberry Pi B+ équipée d'un adaptateur Wifi et d'une connexion Ethernet locale. Il n'est pas question d'avoir une dizaine de capteurs Wifi sur mon point d'accès standard. En premier lieu parce qu'il n'est pas en fonction de manière constante, mais aussi et surtout, car la Pi va générer un trafic important, ce qui forme un point d'attaque idéal et enfin, car l'ensemble ne sera pas directement relié à Internet, et encore moins dépendant d'une solution *cloud* quelconque (faire le paon avec des beaux graphes c'est bien amusant, mais la température qu'il fait sous mon lit ne regarde que moi).

3. CHIFFRONS

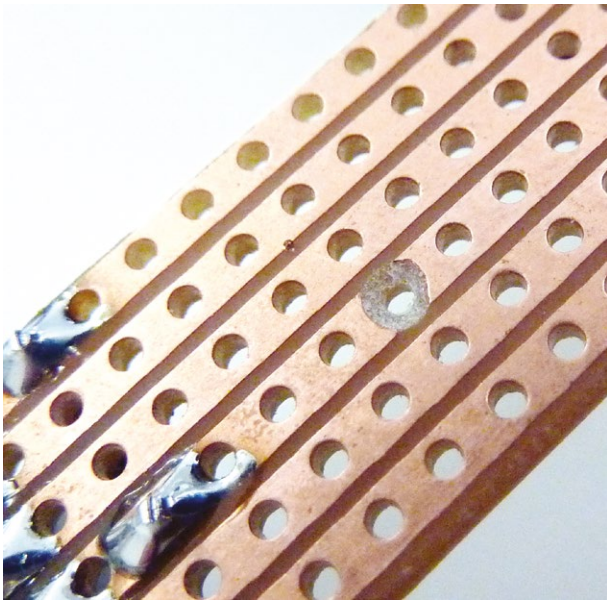
Il est temps de sortir la calculette et de savoir combien cette lubie va nous coûter. Nous partirons ici sur deux budgets : un pour un ensemble de 10 capteurs Wifi et l'autre pour le contrôleur central.

Côté capteurs, nous avons besoin de :

- 10 capteurs de températures BS18B20. Là il est important de préciser deux choses capitales : les bonnes offres sont des pièges à pigeons et l'électronique ça chauffe ! Si vous êtes amateur d'eBay pour vos fournitures, méfiez-vous des trop bonnes affaires. 10 DS18B20 en boîtier TO-92 pour 8€ ça n'existe pas, ce sont des contrefaçons. Après une triste mésaventure de ce type, je me suis tourné vers un lot de capteurs étanches équipés de câbles d'un mètre au prix de 15€ (vendeur excellbay - Chine). Ce format règle également l'autre problème, celui de la proximité du module ESP8266 et de son alimentation générant de la chaleur et faussant les mesures.
- 10 modules ESP8266 ESP-01. Nous en avons parlé dans le précédent numéro, il s'agit de modules permettant de fournir une connectivité Wifi à un montage Arduino par exemple. Mais l'architecture même du module permet sa reprogrammation et son utilisation comme une carte Arduino via l'installation d'une nouvelle plateforme dans votre environnement de développement (voir également l'article du numéro précédent sur la gestion de plateformes depuis la version 1.6.4 de

l'IDE Arduino). Mes 10 modules m'ont coûté 26 € (vendeur czb6721960 - Chine) et il s'agit de la déclinaison la plus modeste qui soit (2 broches utilisables, dont une contrôlant le passage en mode programmation).

- 10 modules de régulation de tension 3,3 volts. L'ESP8266 fonctionne en 3,3V et n'est pas tolérant au 5V. Il est donc nécessaire de trouver une source d'alimentation adéquate. La solution pour laquelle j'ai opté consiste à utiliser une source et un régulateur LDO (*Low-DropOut*) AMS1117 capable de fournir la bonne tension à partir de 5V. Il est possible d'acheter les régulateurs seuls, mais la magie chinoise nous évite d'avoir à compléter le montage avec des condensateurs en proposant des modules complets à 1€ pièce (vendeur ssearIver2009 - Chine).
- Pour alimenter le régulateur, rien de tel qu'un adaptateur secteur vers USB normalement prévu pour les gens qui égarent leur bloc secteur de smartphone par exemple. C'est là sans contexte le point faible de l'ensemble en termes de sécurité puisque j'ai opté pour un lot de 10 adaptateurs à 4€ + 4€ de port (vendeur omgift - Chine). Je n'ai absolument aucun doute sur la faible qualité de ces produits (voire leur dangerosité), la certaine présence de composants recyclés et l'absence complète de sécurité... mais 8€ les 10 port inclus est intéressant pour un coup d'essai (oui, je les changerais plus tard).
- 20 résistances 2 Kohms et 10 résistances 4,7 Kohms pour un montant surévalué d'environ 2€. Les premières sont les résistances de rappel pour les broches Reset et CH_PD de l'ESP8266, et les secondes sont destinées à la connexion des capteurs DS18B20 avec une fonction identique.
- Quelques barrettes sécables femelles (également appelées « connecteurs femelles FH ») au pas de 2,54 mm. Il en existe de toutes tailles et de tous formats, mais j'ai une préférence pour les 1×40 tantôt vendues en lots mâle/femelle. Un vendeur eBay, cuckooshop (Hong Kong - non testé), propose par exemple 20 paires de ce type pour moins de 10€.
- Enfin, pour assembler le tout, nous avons besoin d'un circuit. Hors de question d'utiliser des platines à essais (surtout avec trois chats curieux et joueurs), nous allons monter tout cela sur un circuit imprimé. La solution idéale serait la conception complète d'un circuit



(et ce sera peut-être fait plus tard), mais dans un premier temps, nous nous tournerons vers la plaque pastillée et plus exactement une *veroboard/stripboard* judicieusement découpée. La différence avec une simple plaque pastillée réside dans la présence de pistes parallèles, un peu dans l'esprit de la platine à essais. L'astuce consiste à souder les composants et couper les pistes de manière adéquate pour former le montage. Un autre avantage intéressant de cette solution (valable également pour le circuit imprimé) est le fait qu'elle s'adaptera presque parfaitement à un connecteur USB type A. Chaque circuit habilement (ou pas) découpé et généreusement étamé fera office de connecteur USB mâle. J'ai utilisé une plaque dont la provenance est depuis longtemps oubliée, mais des vendeurs proposent des plaques de 100×160 mm pour quelques 14€ + 4€ de (vendeur g00fie – UK).

Sectionner une piste est bien plus aisé en « gâchant » un trou au passage. Il suffit d'un petit coup de foret ou de lame pour couper la piste. On s'assurera cependant systématiquement du résultat visuellement, par transparence et avec un multimètre avant une mise en fonction du montage.

Notez que ne sont pas inclus dans cette liste les éléments permettant la programmation de l'ESP8266 et en particulier l'adaptateur USB/série 3,3V, la platine à essais et les connecteurs/câbles. Tout ceci fait normalement partie de la trousse à outils standard du bidouilleur. Le coût complet pour 10 capteurs se monte ainsi à environ 80 €. Il est certainement possible de réduire ce montant avec un peu de récupération/recyclage ou quelques concessions. Une source 5V ou un régulateur peut alimenter le montage et chaque ESP8266 peut parfaitement piloter plusieurs capteurs DS18B20. D'autre part, il est peut-être possible, selon la pièce, de trouver une autre source d'alimentation comme une box Internet ou un disque dur réseau (NAS).

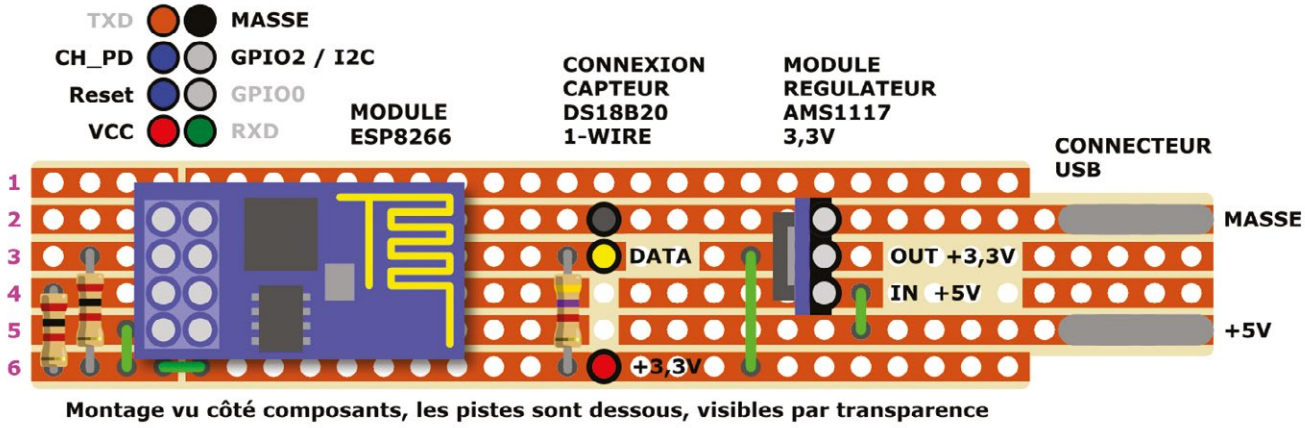
Côté centralisation à présent, nous aurons besoin de :

- Une Raspberry Pi modèle B+ à environ 25€ chez n'importe quel revendeur qui se respecte. Le modèle B conviendra tout aussi bien, tout comme la Raspberry Pi 2, mais j'écarte les modèles A et A+ pour deux raisons : je préfère avoir un hub pour la connexion de l'adaptateur Wifi et l'Ethernet sera l'interface de pilotage/contrôle de la centrale (aucun service ne sera à l'écoute sur l'interface Wifi pour des raisons de sécurité).
- L'alimentation de la Pi sous la forme d'un adaptateur USB standard. Celui d'un « vieux » smartphone fera l'affaire et au besoin on pourra toujours en trouver d'une qualité acceptable pour quelques 10€.
- Un afficheur LCD TFT 2.2" 240×320 à base de contrôleur ILI9340C. C'est le modèle dont nous avons parlé dans le numéro 4 du magazine, disponible pour environ 6€ (vendeur czb6721960 - Chine). Un écran LCD comme ceux proposés par AdaFruit, parfaitement compatible avec le pilote fbtft, fera tout aussi bien l'affaire. Notez que ceci n'est pas un impératif dans le sens où la Raspberry Pi peut parfaitement collecter les informations sans interface utilisateur, utiliser un écran HDMI ou simplement retourner des informations via un serveur Web ou un accès en ligne de commandes (SSH). Mais il faut bien avouer qu'un mini-écran affichant des informations colorées est la petite touche qui satisfait l'ego de façon optimale...
- Un adaptateur USB Wifi compatible GNU/Linux et disposant de pilotes pas trop pénibles (et de préférence intégrés par défaut au système). L'un des adaptateurs Linksys WUSB54GC v1 à puce Ralink RT73 traînant dans un tiroir depuis des années a ici fait parfaitement l'affaire. L'achat remonte à une sombre et lointaine époque où il était difficile de trouver un matériel acceptable (et accepté) pour GNU/Linux et où certains utilisaient même des pilotes Windows avec une horreur appelée NDISwrapper. On retrouve facilement le WUSB54GC v1 pour quelques 10€ sur eBay, les PC portables étant aujourd'hui systématiquement équipés d'interfaces Wifi.

Notre centrale de collecte (et, à terme sans doute, notre centrale domotique) nous coûtera donc un peu plus de 50€ tout compris. Ceci monte le budget total de l'opération à environ 130€. Ceci peut sembler beaucoup, mais on parle ici d'un ensemble de capteurs et d'une base très évolutive. C'est toujours moins cher que le Nest de Google incompatible avec ma chaudière Vaillant et n'étant finalement qu'un unique capteur et un algorithme un peu futé derrière une belle interface.

coromonadalix

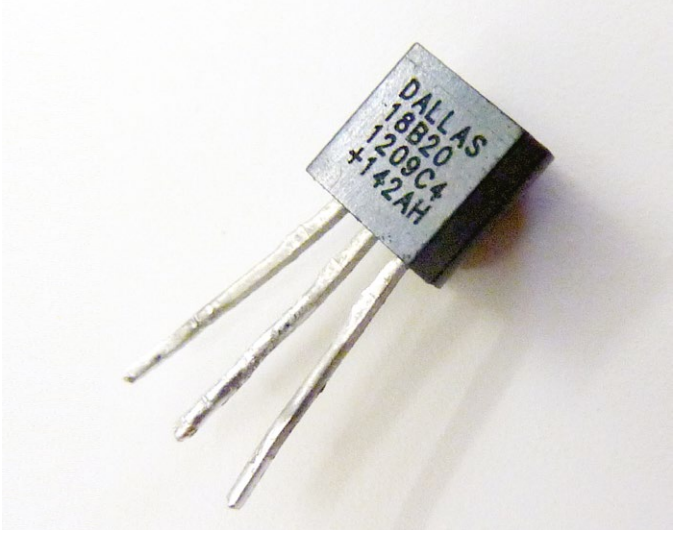
4. ASSEMBLAGE D'UN CAPTEUR WIFI



Montage vu côté composants, les pistes sont dessous, visibles par transparence

Un capteur de température Maxim/ Dallas DS18B82 en boîtier TO-92. Après quelques essais et une poignée de composants contrefaits, cette solution a été écartée au bénéfice de capteurs similaires, mais étanches et équipés d'un câble d'un mètre permettant de faire une mesure loin du montage générant de la chaleur.

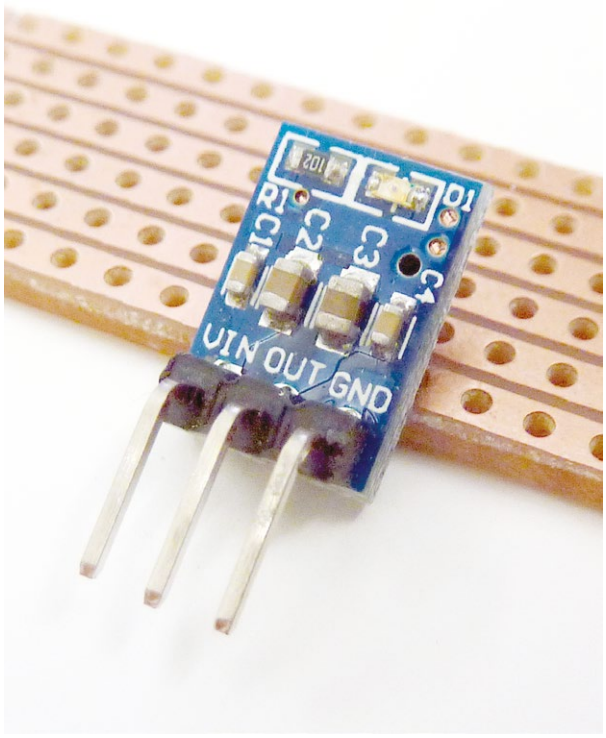
Le montage du circuit d'un capteur est relativement aisé. Il s'agit de simplement alimenter le module ESP8266 via le régulateur tout en ajoutant les résistances de rappel de 2 Kohms sur les broches Reset et CH_PD ainsi que celle de 4,7 Kohms sur GPIO2. On soude ensuite les trois connecteurs du capteur DS18B20 à la masse (noir), à 3,3V (rouge) et à GPIO2 (jaune).



L'ensemble prend place sur un morceau de *stripboard* de 6 pistes parallèles d'une longueur suffisante pour accueillir le module ESP8266 et le régulateur tout en se ménageant une extrémité pour faire office de connecteur USB mâle. Le schéma présenté ici montre le montage **côté composants** avec les 6 pistes vues en transparence. La disposition des composants nécessite un peu de jonglage mental de façon à avoir un minimum de connexion à faire tout en minimisant le nombre de coupures de pistes.

Remarquez à ce propos la séparation centrale à l'endroit où se trouve le connecteur du module ESP8266. Un simple petit coup de scie fait parfaitement l'affaire si l'on se montre délicat. Toute la partie gauche du circuit est donc déconnectée de la partie droite à l'exception de la piste 6 correspondant à l'alimentation 3,3V. Il est bien plus facile de couper 6 pistes entre deux trous pour ensuite souder une patte de composant que de chercher à couper 4 pistes au cutter.

Pour économiser de la place et du circuit, vous pouvez également souder directement les résistances de 2 Kohms sur le module ESP8266. Personnellement, n'étant jamais certain que tout reste figé dans le temps, j'ai une nette préférence pour les connecteurs. Ainsi, le régulateur est directement soudé sur le



Les modules de régulation de tension reposant sur un LDO AMS1117 sont équipés de condensateur de découpage et d'un indicateur à led. Autant de composants dont nous pourrons donc nous passer sur le reste du circuit.

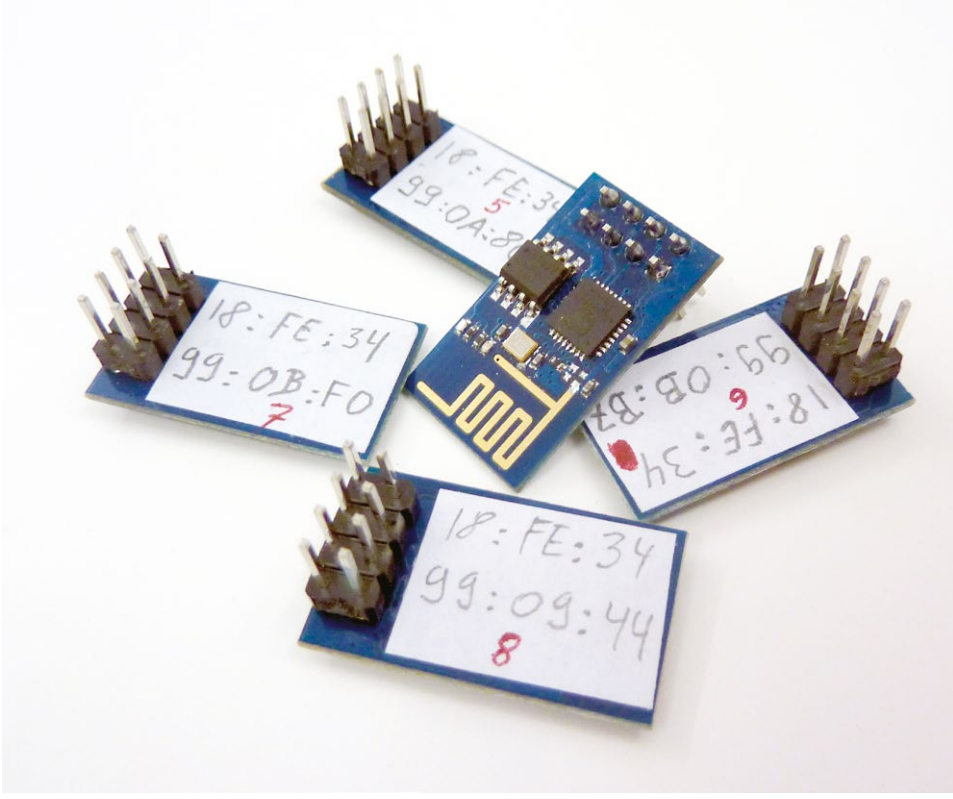
circuit, mais le module ESP8266 est placé sur un support et laissé intact (ce qui permet également de rapidement le retirer pour mettre à jour son programme). L'ESP8266 est équipé d'un connecteur mâle 2×4 et l'astuce consiste à simplement utiliser deux morceaux de barrette femelle de 1×40.

Ma technique consiste à retirer un connecteur métallique à la pince pour libérer un connecteur, à couper au cutter à cet endroit puis à limer les bords. On perd un connecteur dans l'opération, mais c'est rapide, économique et propre. L'astuce pour la soudure au circuit consiste à utiliser deux barrettes mâles enfichées perpendiculairement. On assure ainsi la symétrie et la stabilité en formant un pseudo support, et il ne reste plus qu'à souder...

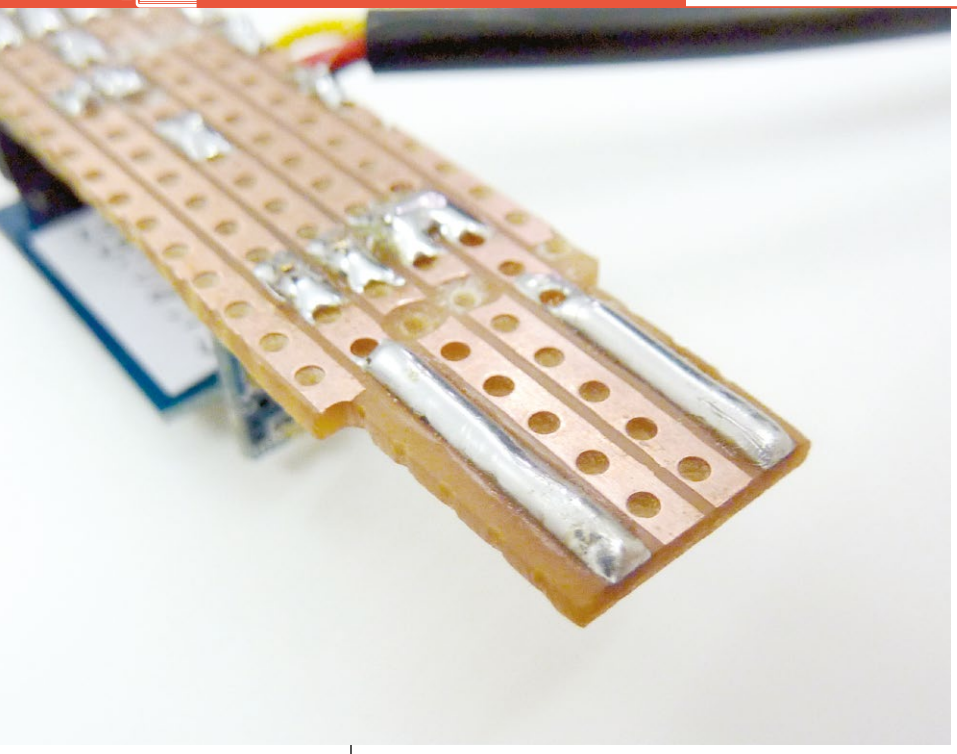
Ma technique pour couper les pistes consiste généralement à procéder à la manipulation sur un des trous, en utilisant soit un foret du bon diamètre pour retirer le cuivre, soit la lame d'un couteau suisse, d'un cutter ou d'un scalpel. Cette seconde option nécessite plus de dextérité, mais évite de perdre 30 mn à fouiller pour trouver le bon foret...

Enfin, le dernier point concernant l'assemblage concerne le connecteur USB. L'idée est de déposer suffisamment d'étain au bout des pistes 2 et 5 de manière

De l'ordre et de la méthode ! Lorsqu'on travaille avec un lot de modules, c'est toujours une bonne idée de prendre le temps d'étiqueter tout cela afin de trouver rapidement la source en cas de problème matériel ou de configuration.



coromonadalix



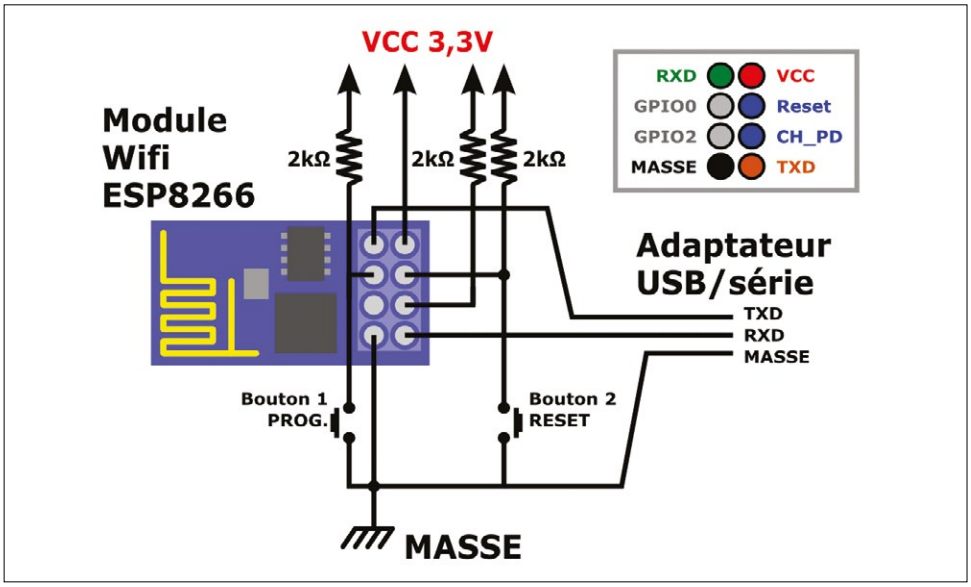
Les spécifications USB précisent un écartement des contacts qui correspond précisément au pas utilisé pour notre circuit (2,54 mm). Pour transformer un bout de circuit en connecteur type A mâle, il nous suffit d'ajouter un peu d'étain et limer les bords superflus de façon à bien aligner le tout.

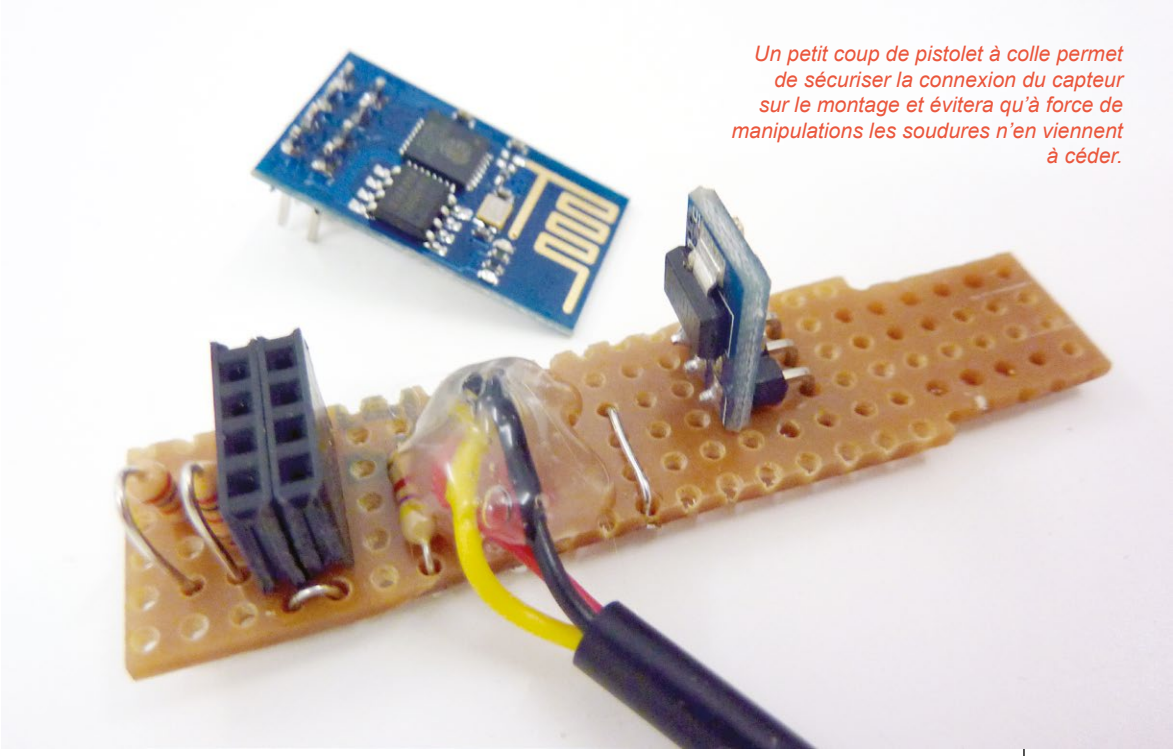
à gagner en épaisseur pour que le connecteur femelle l'accepte et le maintienne en place fermement. L'étain doit être assez chaud et en quantité suffisante pour qu'au moment où l'on retire la panne du fer à souder l'ensemble forme une surface plane, mais surélevée d'un bon demi-millimètre. Une autre solution peut consister à épaissir le circuit, côté composants cette fois, avec un morceau de plastique collé. Il suffira finalement de réduire la largeur de la partie enfichable à la lime jusqu'à la lisière des trous des pistes 1 et 6.

Terminons cette partie avec un dernier conseil : vérifiez avant de brancher ! Et quand je dis « vérifiez », je dis :

- inspectez les pistes, côté cuivre et par transparence ;
- testez les connexions (ou l'absence de connexion) avec un multimètre ;
- branchez le circuit à l'adaptateur USB sans module ESP8266 et mesurez les tensions ;
- branchez le circuit complet.

La programmation d'un module ESP8266 a été traitée dans le précédent numéro. Le schéma de connexion, repris de l'article en question, est simple et consiste à alimenter le module en 3,3V tout en connectant les lignes RX et TX à un adaptateur USB/série relié à un PC ou un Mac. Côté logiciel, vous devrez avoir une version la plus récente possible de l'environnement Arduino et





Un petit coup de pistolet à colle permet de sécuriser la connexion du capteur sur le montage et évitera qu'à force de manipulations les soudures n'en viennent à céder.

intégrer le support pour l'ESP8266 via le *Boards Manager* (**http://arduino.esp8266.com/package_esp8266com_index.json**). Il suffira alors de tenir la ligne GPIO0 à la masse lors de la mise sous tension ou en procédant à un reset (ligne RESET à la masse un bref instant). Le module passe alors en mode programmation et le croquis peut être chargé dans sa mémoire comme avec une carte Arduino.

Le croquis est très similaire à l'exemple du numéro précédent sauf, bien entendu, pour les points qui concernent l'utilisation du capteur DS18B20. Nous ajoutons également des informations utilisées uniquement lors du premier démarrage du module suite à sa programmation. Nous utilisons ainsi la liaison série, accessible via le moniteur série de l'IDE Arduino, pour nous assurer que la température est bien mesurée, mais également pour afficher une information très importante : l'adresse matérielle de l'interface Wifi.

Chaque interface réseau (Wifi ou Ethernet) est identifiée au plus bas niveau par une série de 6 valeurs 8 bits : l'adresse MAC. Celle-ci est en principe unique à chaque carte, module, interface, etc., et nous permettra, par la suite, d'associer une adresse IP par capteur Wifi de manière cohérente et systématique. C'est dans la configuration du service DHCP sur la Raspberry Pi que nous distribuerons les adresses IP de la sorte et non de façon dynamique (premier venu, premier servi). C'est un peu plus lourd en termes de configuration, mais nous saurons qui est qui. Une autre solution aurait pu être de prévoir une autre URL, en plus de celle retournant la température, de façon à ce que chaque module décline son identité.

Notez enfin que la majeure partie du croquis est inutile une fois les capteurs en place (tout ce qui touche à la communication série) et il pourra être utile d'alléger le code par la suite pour reprogrammer les ESP8266 dans une version finale.

Fichier Édition Croquis Outils Aide

```
#include <ESP8266WiFi.h>
#include <OneWire.h>
#include <DallasTemperature.h>

const char* ssid = "domoAP";
const char* password = "mot2passe";

// broche data du DS18B20
#define ONE_WIRE_BUS 2
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature DS18B20(&oneWire);

// Crée le serveur Web en spécifiant le port TCP/IP
// 80 est le port par défaut pour HTTP
WiFiServer server(80);

// Démarrage
void setup() {
  uint8_t mac[WL_MAC_ADDR_LENGTH];
  float temp;

  // Communication série 115200
  Serial.begin(115200);
  // petit pause
  delay(10);

  // Deux sauts de ligne pour faire le ménage, car
  // le module au démarrage envoie des caractères sur le port série
  Serial.println();
  Serial.println();

  // On mesure et affiche la température pour
  // vérifier le fonctionnement sur le moniteur série
  // juste après la programmation
  do {
    DS18B20.requestTemperatures();
    temp = DS18B20.getTempCByIndex(0);
  } while (temp == 85.0 || temp == (-127.0));
  Serial.print("Temperature: ");
  Serial.println(temp);

  // On récupère et on affiche l'adresse matérielle (MAC)
  // de l'interface. Ceci nous permettra de la noter sur une
  // étiquette sur le module pour l'utiliser dans la
  // configuration du serveur DHCP sur la Raspberry Pi
```

```
  if(WiFi.macAddress(mac) != 0) {
    for(int i=0; i<WL_MAC_ADDR_LENGTH; i++) {
      if(mac[i]<16) Serial.print ("0");
      Serial.print(mac[i],HEX);
      Serial.print((i < WL_MAC_ADDR_LENGTH-1) ? ":" : "\n");
    }
  }

  Serial.print("Connexion a : ");
  Serial.println(ssid);
  // Connexion au point d'accès
  WiFi.begin(ssid, password);

  // On boucle en attendant une connexion
  // Si l'état est WL_CONNECTED la connexion est acceptée
  // et on a obtenu une adresse IP
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("WiFi connected");

  // Démarrage du serveur Web
  server.begin();
  Serial.println("Server started");

  // On affiche notre adresse IP
  Serial.println(WiFi.localIP());
}

// Boucle principale
void loop() {
  float temp;
  char strbuffer[10];

  // Est-ce qu'un client Web est connecté ?
  WiFiClient client = server.available();
  if (!client) {
    // Non, on abandonne ici et on repart dans un tour
    return;
  }

  // Un client est connecté
  Serial.println("new client");
  // On attend qu'il envoie des données
  while(!client.available()){
    delay(1);
  }

  // On récupère la ligne qu'il envoie jusqu'au premier retour chariot (CR)
  String req = client.readStringUntil('\r');
```

coromonadalix

```
  // On affiche la ligne obtenue pour information
  Serial.println(req);
  // La ligne est récupérée, on purge
  client.flush();

  // Test de la requête
  int val;
  // Est-ce que le chemin dans la requête est "/gettemp" ?
  if (req.indexOf("/gettemp") != -1) {
    // Oui
    // On mesure la température
    do {
      DS18B20.requestTemperatures();
      temp = DS18B20.getTempCByIndex(0);
    } while (temp == 85.0 || temp == (-127.0));

    Serial.print("Temperature: ");
    Serial.println(temp);
  // C'est un autre chemin qui est demandé et il ne correspond à rien
  } else {
    // On signale que cette demande est invalide
    Serial.println("invalid request");
    // et on déconnecte le client du serveur Web
    client.stop();
    // et on repart dans un tour de boucle
    return;
  }

  // Peut-être le client a-t-il envoyé d'autres données,
  // mais nous purgeons la mémoire avant d'envoyer notre réponse
  client.flush();

  // Composition de la réponse à envoyer au client
  String s = "HTTP/1.1 200 OK\r\nContent-Type: text/plain\r\n\r\n";
  dtostrf(temp, 4, 3, strbuffer);
  /*
  for(int i=0;i<sizeof(strbuffer);i++){
    s += strbuffer[i];
  }
  */
  s += strbuffer;
  s += "\n";
  client.print(s);

  // Petite pause
  delay(1);
  // et on le déconnecte du serveur Web
  Serial.println("Client disonnected");
}
```

Arduino

coromonadalix

Un des 10 montages terminé et complet. L'ESP8266 est détachable en cas de mise à jour de notre croquis, contrairement au régulateur de tension et au capteur de température. Il ne reste plus qu'à en faire 9 autres et à les placer un peu partout dans l'appartement.

PRÉ-CONCLUSION

Sur la base de ces manipulations, il ne reste plus qu'à réitérer autant de fois que l'on désire de capteurs. Cet exemple se base sur la relève de températures, mais la même logique peut être appliquée à bien des mesures. Il suffira pour cela de faire évoluer le croquis afin de piloter un nouveau composant et structurer correctement les échanges Wifi/HTTP. Il est également possible d'envisager l'utilisation de la broche GPIO0 de l'ESP8266 afin par exemple, d'y brancher une led permettant de signifier la connexion au réseau. Le pilotage des GPIOs de l'ESP8266 est identique à ce qui se fait sur d'autres plateformes type Arduino et a fait l'objet d'explications dans le précédent numéro.

Remarquez également que le choix du protocole HTTP ici est surtout une solution de facilité tant côté module que Raspberry Pi. D'autres protocoles plus spécifiques comme MQTT (*Message Queuing Telemetry Transport*) sont spécialement dédiés au domaine des capteurs et pourraient être utilisés. Ce sera peut-être plus tard une piste pour faire évoluer l'ensemble de la solution (et l'occasion de détailler cela dans un nouvel article).

Pour l'heure, il est temps de finir et tester la « flotte » de capteurs avant de passer à l'aspect collecte par une carte Raspberry Pi... **DB**

coromonadalix

RÉSEAU DE CAPTEURS DE TEMPÉRATURE : LE CÔTÉ PI

Denis Bodor

L'article précédent a traité en détail le concept proposé ainsi que la réalisation des capteurs. Il est maintenant temps de configurer le système central chargé de collecter les données et de servir de point d'accès.

coromonadalix

Nous utiliserons ici une Raspberry Pi B+, mais n'importe quel SBC (*Single-Board Computer* ou nano-ordinateur) pourvu d'un port USB pour l'adaptateur Wifi, d'un connecteur Ethernet et d'une solution pour connecter un mini-écran LCD fera l'affaire. Le duo *Beagleboard Black* + écran *4D system* 4 ou 7 pouces, pour ne citer que cette alternative, sera par exemple un excellent choix.

La distribution utilisée sur la Raspberry Pi est naturellement Raspbian avec une carte microSD fraîchement initialisée et un système prêt à l'emploi. Nous partirons ici du principe que le système est configuré, à jour et vierge de toutes utilisations précédentes.

1. CONFIGURATION WIFI, RÉSEAU ET SYSTÈME CÔTÉ RASPBERRY PI

La configuration d'une carte Raspberry Pi en guise de point d'accès Wifi a déjà fait l'objet d'un article dans le numéro 6 du magazine, nous ne nous attarderons donc pas sur le sujet plus que nécessaire à la reproduction de l'installation. Un point d'accès se compose d'un service se chargeant d'accepter et d'authentifier les connexions Wifi, et d'un service distribuant les adresses IP pour former un réseau : respectivement HostAP et un serveur DHCP.

Nous commençons par configurer le réseau pour la carte elle-même via `/etc/network/interfaces` :

```
# interface loopback
auto lo
iface lo inet loopback

# Ethernet
# adresse obtenue via DHCP
auto eth0
iface eth0 inet dhcp

# Wifi
# adresse statique
# le serveur c'est nous !
auto wlan0
iface wlan0 inet static
  address 172.16.16.1
  netmask 255.255.255.0
```

On se penche ensuite sur la configuration du paquet `hostapd` immédiatement après son installation, via `/etc/hostapd/hostapd.conf` :

```
# interface à utiliser
interface=wlan0
# pilote
driver=nl80211
# nom de l'AP
ssid=domoAP
# mode Wifi
hw_mode=g
# canal à utiliser
channel=3
# on accepte toutes les MAC
macaddr_acl=0
# on accepte que OSA
auth_algs=1
# on ne cache pas le SSID
ignore_broadcast_ssid=0
# chiffrement WPA2
wpa=2
# la phrase secrète
wpa_passphrase=mot2passe
# gestion PSK
wpa_key_mgmt=WPA-PSK
# chiffrement TKIP pour WPA
wpa_pairwise=TKIP
# chiffrement CCMP pour WPA2
rsn_pairwise=CCMP
```

Puis on précise le fichier de configuration à utiliser dans `/etc/default/hostapd` :

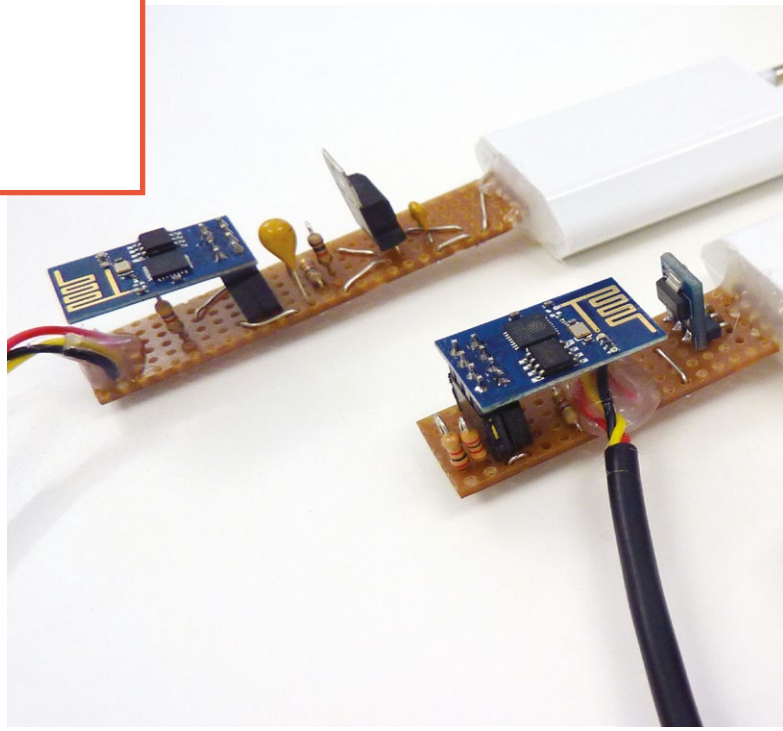
```
DAEMON_CONF="/etc/hostapd/hostapd.conf"
```

coromonadalix

Et finalement on installe le paquet **isc-dhcp-server** et on édite le fichier **/etc/dhcp/dhcpd.conf** :

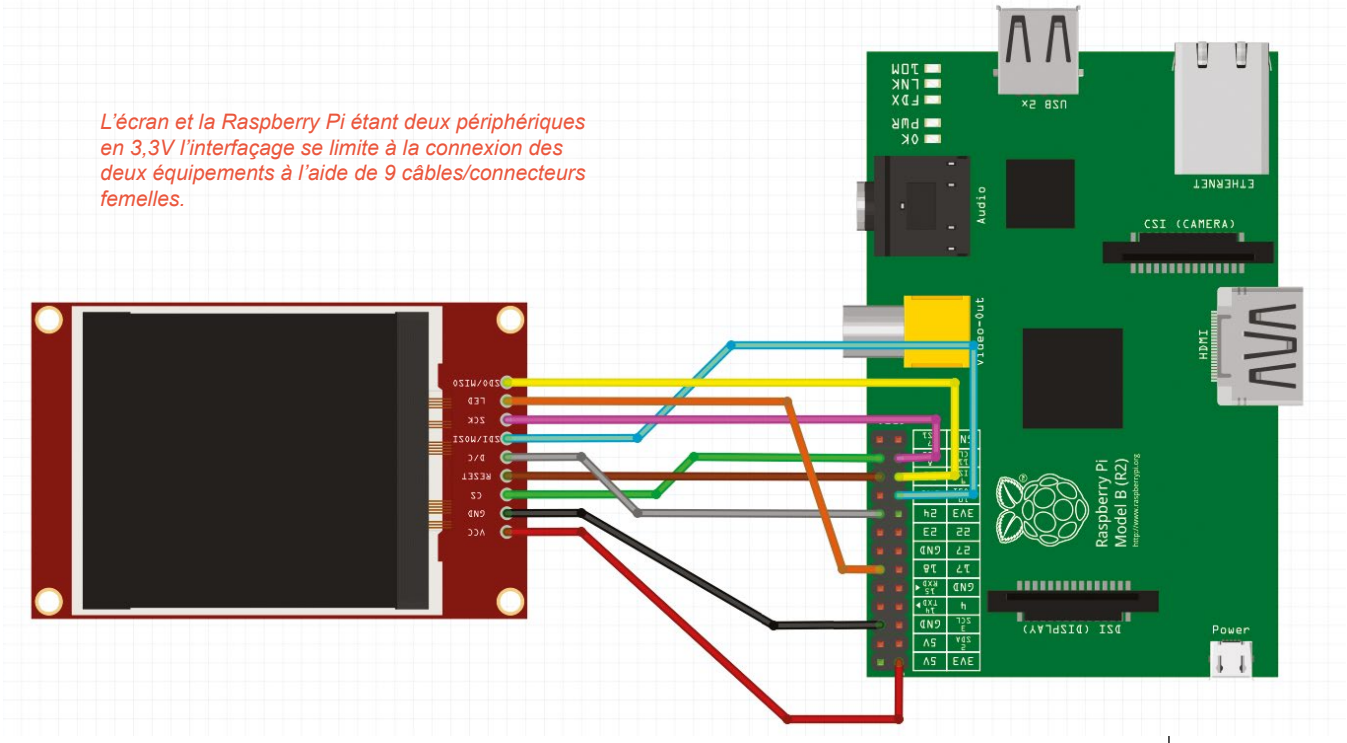
```
# pas de gestion DDNS
ddns-update-style none;
# adresses données pour 600s
default-lease-time 240;
# maximum de temps accepté
max-lease-time 600;
# c'est moi le chef !
authoritative;
# les messages vont dans le journal sys
log-facility local7;

subnet 172.16.16.0 netmask 255.255.255.0 {
  # l'adresse de diffusion
  option broadcast-address 172.16.16.255;
  # adresse du routeur (moi-même)
  option routers 172.16.16.1;
  # mon nom de domaine (réseau local)
  option domain-name "local";
  # les serveurs DNS à utiliser
  option domain-name-servers 8.8.8.8, 8.8.4.4;
}

host esp01 {
  hardware ethernet 18:fe:34:99:0a:1d;
  fixed-address 172.16.16.101;
}

[...]

host esp08 {
  hardware ethernet 18:fe:34:99:09:44;
  fixed-address 172.16.16.108;
}
```

Petite précision ici puisque notre fichier est légèrement différent de celui du point d'accès de la configuration présentée au numéro 6. La ligne contenant le mot-clé **range** a disparu de la portée de **subnet {}**. Nous ne spécifions pas de plage d'adresses à distribuer, car nous les attribuons au cas par cas. Ceci implique de remplir le reste du fichier avec des **host {}** en précisant un nom, une adresse MAC (**hardware ethernet**) et l'adresse IP (**fixed-address**) correspondante à chaque client/module/capteur. Il vous faudra donc autant de blocs **host {}** que de modules susceptibles de se connecter au point d'accès et vous comprenez pourquoi c'était une bonne idée que de relever et noter les adresses MAC des ESP8266 sur des étiquettes...



La fabrication des capteurs ne s'est pas déroulée sans quelques variations et évolutions. En arrière-plan se trouve le premier capteur assemblé avec empressement. Remarquez l'utilisation d'un régulateur ST LD33V accompagné d'un condensateur de filtrage, ainsi que l'orientation du module ESP8266 et surtout le nombre de connexions de pistes côté composants. Au centre se trouve le montage correspondant au modèle décrit dans l'article, passablement optimisé, et enfin devant on peut apercevoir la dernière « génération », plus compacte avec le module régulateur placé sous l'ESP8266.

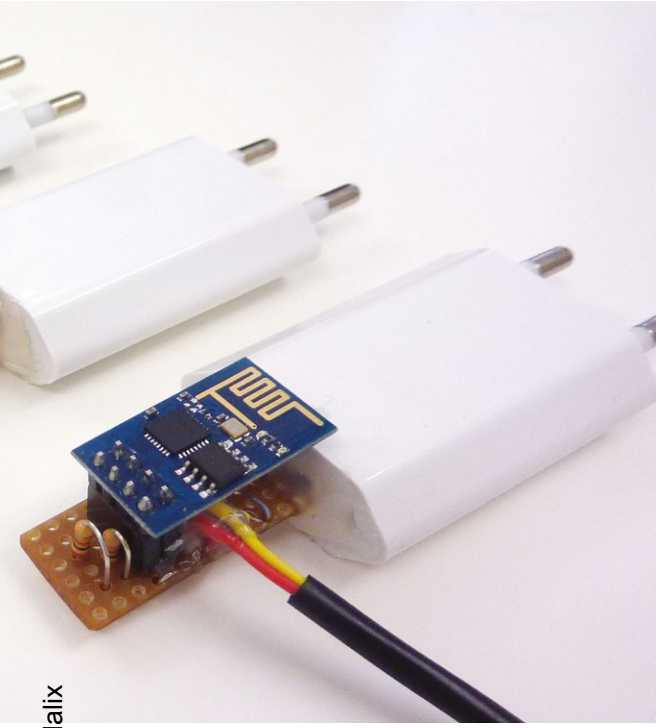
coromonadalix



L'écran et la Raspberry Pi étant deux périphériques en 3,3V l'interfaçage se limite à la connexion des deux équipements à l'aide de 9 câbles/connecteurs femelles.

On n'oubliera pas enfin, de spécifier quelle interface doit être utilisée par le serveur DHCP via **/etc/default/isc-dhcp-server** :

```
INTERFACES="wlan0"
```



Un redémarrage des différents services, ou tout simplement de tout le système, et nous pourrons alors procéder à un premier test. Il suffit pour cela d'alimenter un premier capteur et, si tout est correctement configuré, celui-ci va se connecter au point d'accès *domoAP*, s'identifier, s'authentifier et finalement obtenir son adresse IP. La commande **curl** nous permet alors de faire une requête HTTP et d'obtenir en retour la température mesurée :

```
$ curl http://172.16.16.101/gettemp
28.562
```

En cas de problème, je recommande dans un premier temps la suppression du paquet **ifplugd** (et éventuellement **avahi-daemon**) interférant avec les configurations réseau manuelles, ainsi que l'installation du paquet **haveged** permettant de maintenir un volume important de nombres pseudo-aléatoires dans le système, nécessaires au fonctionnement correct de **hostapd** (en particulier avec un système exécutant peu de programmes et gérant de nombreux clients Wifi).

Une fois le réseau fonctionnel et testé avec chaque capteur Wifi, on pourra se pencher sur la prise en charge de l'écran LCD TFT connecté en SPI sur la carte. Le schéma de connexion est identique à celui vu dans le numéro 4 du magazine.

Pour prendre en charge ce type de matériel, une simple commande suffit :

coromonadalix

```
pi@raspberrypi ~ $ sudo REPO_URI=https://github.com/notro/rpi-firmware rpi-update
*** Raspberry Pi firmware updater by Hexxeh, enhanced by AndrewS and Dom
*** Performing self-update
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 10186  100 10186    0     0  35302      0 --:--:-- --:--:-- --:--:-- 43905
*** Relaunching after update
*** Raspberry Pi firmware updater by Hexxeh, enhanced by AndrewS and Dom
*** We're running for the first time
*** Backing up files (this will take a few minutes)
*** Backing up firmware
*** Backing up modules 3.18.11-v7+
*** Downloading specific firmware revision (this will take a few minutes)
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   167    0   167    0     0    202      0 --:--:-- --:--:-- --:--:--   242
100 47.9M  100 47.9M    0     0   514k      0  0:01:35  0:01:35 --:--:--  312k
*** Updating firmware
*** Updating kernel modules
*** depmod 4.0.7+
*** depmod 4.0.7-v7+
*** Updating VideoCore libraries
*** Using HardFP libraries
*** Updating SDK
*** Running ldconfig
*** Storing current firmware revision
*** Deleting downloaded files
*** Syncing changes to disk
*** If no errors appeared, your firmware was successfully updated to
b2f5782d2a61cdfa1967942d34eae48900266ae1
*** A reboot is needed to activate the new firmware
```

La commande utilisée, **rpi-update**, est la même que pour une mise à jour classique du noyau Linux, mais on précise ici une source (dépôt ou *repository*) différente. Après quelques instants, un noyau Linux 4.0.7 sera installé et un redémarrage de la Pi sera alors nécessaire. Ceci fait, on éditera le fichier **/etc/modules** pour y ajouter deux lignes :

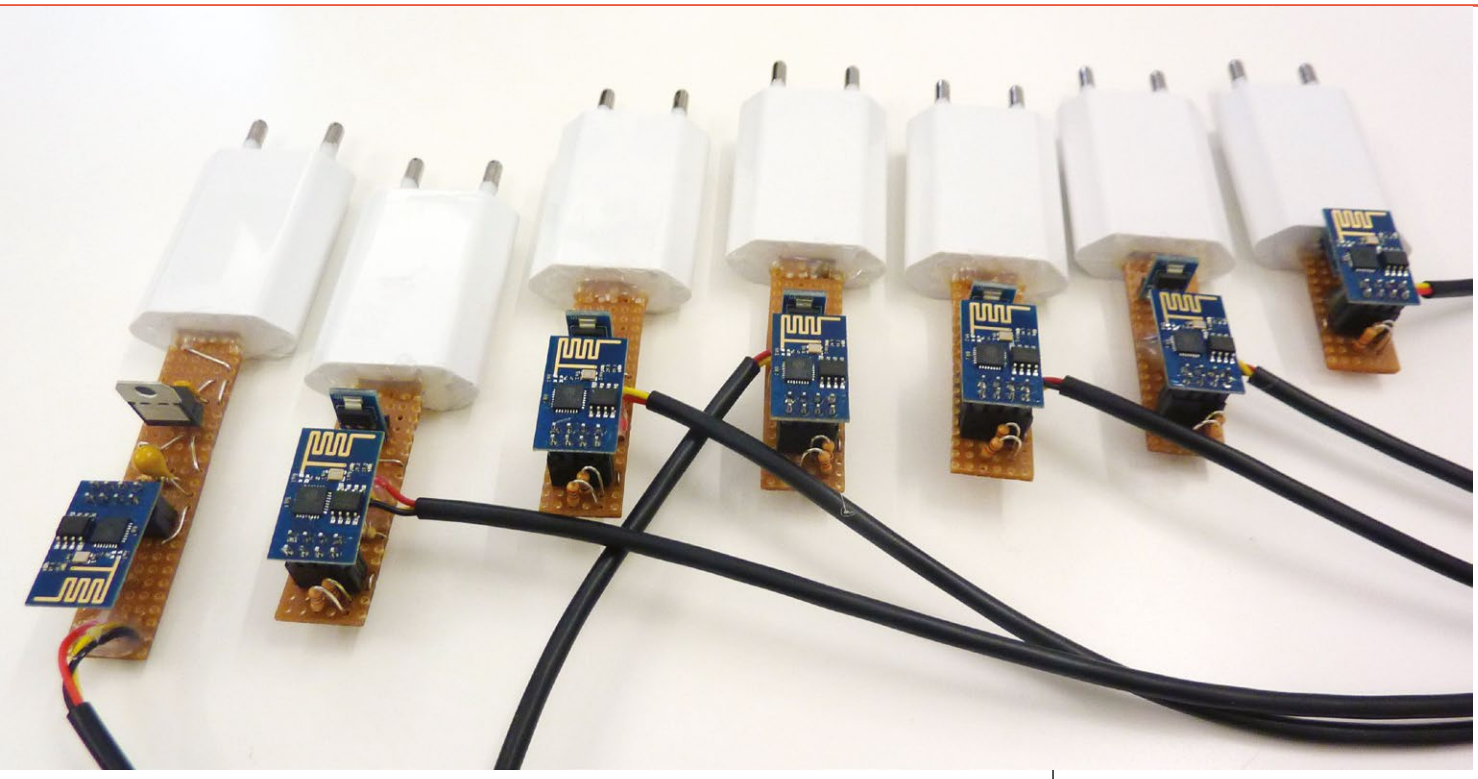
```
fbtft dma
fbtft_device name=adafruit22a verbose=0 speed=48000000 rotate=90
```

Ceci permettra de prendre en charge automatiquement l'écran SPI à chaque démarrage du système. Notez la présence de **rotate=90** en fin de ligne afin de « tourner » l'affichage et obtenir une orientation similaire à celle d'un écran classique (paysage en 320×240 pixels). Vous pouvez également ajouter **fbcon=map:10 fbcon=font:VGA8x8** à la fin de la ligne présente dans **/boot/cmdline.txt** afin d'avoir les messages de démarrage sur le mini-écran ainsi qu'un accès à la ligne de commandes par ce biais (à condition d'ajouter un clavier USB).

À ce stade, un démarrage de la Raspberry Pi devrait se faire sans problème avec activation de l'écran et du point d'accès. Les capteurs Wifi se chargeront seuls de se reconnecter après ces démarrages successifs et ne devraient pas nécessiter de reset.

1.1 Un peu de Python ?

Nous l'avons vu précédemment, pour consulter la température relevée par l'un des capteurs Wifi, la simple commande **curl** est suffisante. Nous avons conçu le croquis enregistré dans les ESP8266 de manière à ce que la consultation de l'URL **http://adresse-ip/gettemp** nous retourne



La « flotte » actuelle de capteurs prêts à l'emploi. Un est encore en cours d'assemblage, mais les autres modules ESP8266 se trouvent respectivement sur le circuit de pilotage de relais du numéro précédent et sur une platine à essais pour expérimentation. Notez qu'une dernière touche a été portée en collant chaque circuit à l'adaptateur USB au pistolet à colle. Malgré l'étamage du connecteur pour en augmenter l'épaisseur, l'ensemble n'était pas assez solide pour éviter les problèmes de contact. La mise en boîtier sera l'étape suivante, dès lors qu'une solution économique se fera jour...

du simple texte et non une page HTML. Avec **curl**, il serait alors possible d'écrire un simple script shell pour procéder à l'affichage de toutes les températures.

Cependant, avec une pointe de Python, nous pouvons aller plus loin. Voici donc le script Python que nous enregistrerons, par exemple, sous le nom **gettemp.py** avant de changer son attribut d'exécution avec la commande **chmod +x gettemp.py**. Dès lors, le script pourra être appelé simplement en se plaçant dans le répertoire où il se trouve et en utilisant, par exemple **./gettemp.py 1** pour le premier capteur :

```
#!/usr/bin/env python
# -*- coding: UTF-8 -*-

from types import *
import os
import sys
import time
import requests

# vérification des arguments
if len(sys.argv) != 2:
  print "arg = number"
  sys.exit(1)

# vérification de la plage
if not 1 <= int(sys.argv[1]) <=10:
  print "range [1..9]"
  sys.exit(1)

# tentative de requête HTTP
try:
```

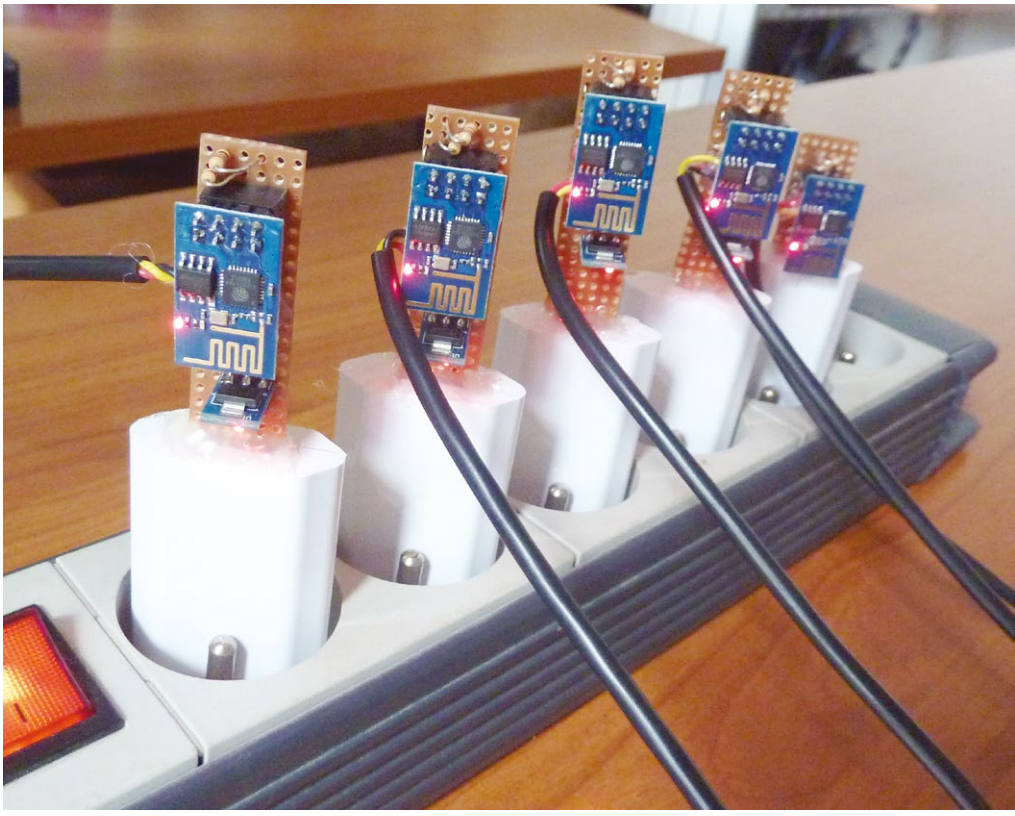
coromonadalix

```
  r = requests.get("http://172.16.16.10"+sys.argv[1]+"/gettemp", timeout=2)
except requests.exceptions.Timeout:
  # le capteur ne répond pas
  print time.strftime("%d/%m/%Y  %H:%M:%S")+"  capteur "+sys.argv[1]+" : No response"
except:
  # une autre erreur est survenue
  print time.strftime("%d/%m/%Y  %H:%M:%S")+"  capteur "+sys.argv[1]+" : Error"
else:
  # sinon, on affiche les informations à commencer par la date et heure,
  # puis le numéro du capteur, le niveau de signal et la température
  print time.strftime("%d/%m/%Y  %H:%M:%S")\
  +"  capteur "+sys.argv[1]\
  +" ("\
  +os.popen("iw dev wlan0 station get `arp -a 172.16.16.10"\
  +sys.argv[1]\
  +" | awk '{print $4}'' | grep signal: | cut -f 3").read().rstrip()+") : "\
  +r.content.rstrip()+"°C"
```

Le script prend en argument un nombre qui est utilisé pour former l'adresse IP du capteur à consulter. 1, par exemple, s'ajoute simplement à **"172.16.16.10"** pour former l'adresse 172.16.16.101. Nous connaissons la correspondance capteur/numéro puisque c'est nous qui la déterminons en fonction de l'adresse MAC via le serveur DHCP. 1 correspond donc **toujours** à 172.16.16.101, 2 à 172.16.16.102, etc.

Nous traitons un seul type d'erreur de manière spécifique : le temps d'attente dépassé pour la réponse. Nous fixons celui-ci à 2 secondes au moment de la requête en invoquant la méthode **`requests.get()`**. Si le capteur ne répond pas en deux secondes, nous avons un *timeout*. Si une autre erreur survient, quelle qu'elle soit, nous affichons juste **`"Error"`** (ceci n'est pas censé se produire).

Les informations de date et d'heure sont facilement accessibles depuis un script Python, il suffit d'utiliser **`time.strftime()`** et formater la sortie comme il nous chante. La température



Pour procéder à des tests en masse, rien de tel que la bonne vieille multiprise. L'établissement quasi simultané de 5 connexions Wifi en une fois sur la Raspberry Pi ne pose aucun problème.

coromonadalix

relevée, qui n'est qu'une chaîne de caractères obtenue en réponse de la requête HTTP, est récupérée dans `r.content` (`r` étant l'objet créé par la requête). On utilise `rstrip()` afin de se débarrasser d'éventuels espaces et surtout du saut de ligne.

Obtenir le niveau de signal est un peu plus délicat. Il n'existe pas, à ma connaissance, de modules ou bibliothèques Python, disponibles pour la Raspberry Pi, permettant d'extraire ces informations du système. L'astuce consiste donc à lancer une commande du shell depuis notre script et en particulier la commande `iw`. Nous pouvons par exemple utiliser :

```
$ iw dev wlan0 station dump
[...]
Station 18:fe:34:99:0a:1d (on wlan0)
        inactive time:  42600 ms
        rx bytes:       21368
        rx packets:     110
        tx bytes:       20333
        tx packets:     101
        tx retries:     0
        tx failed:      0
        signal:         -40 dBm
        signal avg:     -39 dBm
        tx bitrate:     1.0 MBit/s
        authorized:     yes
        authenticated:  yes
        preamble:       long
        WMM/WME:        no
        MFP:            no
        TDLS peer:      no
[...]
```

Cette commande permet de lister tous les clients Wifi connectés à notre point d'accès ainsi que les informations les concernant. On retrouve ainsi celui qui nous intéresse sur les différentes occurrences des lignes `signal:`. Le problème est que cette commande nous retourne justement les informations de **tous** les clients et non d'un seul en particulier. Il est possible d'utiliser `iw dev wlan0 station get` en complétant la ligne avec l'adresse MAC de l'interface du client qui nous intéresse et dans ce cas nous obtiendrons que ses informations. Il nous faut donc un moyen de récupérer l'adresse MAC à partir de l'adresse IP, elle-même déduite du numéro du capteur. Pour cela, nous avons la commande `arp` :

```
$ arp -a 172.16.16.101
? (172.16.16.101) at
18:fe:34:99:0a:1d [ether] on wlan0
```

Mais l'information se trouve noyée dans la ligne que retourne la commande. Il faut utiliser une redirection et l'outil `awk` pour extraire la donnée qui nous intéresse :

```
$ arp -a 172.16.16.101 | awk '{print $4}'
18:fe:34:99:0a:1d
```

Cette ligne de commandes complète retourne l'adresse MAC à partir de l'IP d'un client, nous pouvons donc l'utiliser dans la ligne invoquant `iw` ainsi : `iw dev wlan0 station get 'arp -a 172.16.16.101 | awk '{print $4}''`. L'utilisation de l'apostrophe inverse (AltGr+è) est une fonctionnalité du shell permettant d'utiliser une commande et d'insérer le résultat dans la ligne actuelle. Ce qui se trouve entre ces symboles sera donc l'adresse MAC à passer en argument.

Mais ce n'est pas fini, nous obtenons toujours plus d'informations qu'il nous en faut. Pour faire le ménage, nous redirigeons la sortie de la commande vers `grep` en spécifiant que nous ne voulons que la ligne comprenant la chaîne `"signal:"` :

```
$ iw dev wlan0 station get \
'arp -a 172.16.16.101 | awk '{print $4}'' \
| grep signal:
        signal:         -40 dBm
```

Nous y sommes presque. Il ne nous reste plus qu'à nous débarrasser des éléments « parasites » en utilisant cette fois `cut` pour ne conserver que le troisième « champ » puisque les « mots » sont séparés par des tabulations. Ce qui nous donne :

```
$ iw dev wlan0 station get \
'arp -a 172.16.16.101 | awk '{print $4}'' \
| grep signal: | cut -f 3
-40 dBm
```

En intégrant tout cela dans notre script Python grâce à `os.popen()` et à grands coups de `+` (concaténation), nous créons un affichage regroupant tout ce qui nous intéresse. Nous pouvons alors utiliser notre script avec :

coromonadalix

```
$ ./gettemp.py 1
14/08/2015  15:20:42  capteur 1 (-41 dBm) : 27.751°C
```

En ajoutant une pointe de shell pour boucler sur les six premiers capteurs (4 restent à assembler à ce moment précis), le résultat est tout de suite plus impressionnant :

```
pi@raspberrypi ~/PY $ for i in {1..6}; do ./gettemp.py
$i; done
14/08/2015  15:21:58  capteur 1 (-42 dBm) : 27.937°C
14/08/2015  15:21:59  capteur 2 (-64 dBm) : 28.750°C
14/08/2015  15:22:00  capteur 3 (-36 dBm) : 28.187°C
14/08/2015  15:22:01  capteur 4 (-56 dBm) : 26.562°C
14/08/2015  15:22:02  capteur 5 (-80 dBm) : 27.000°C
14/08/2015  15:22:03  capteur 6 (-82 dBm) : 27.312°C
```

Notez que ce script n'est ni propre ni parfait et encore moins sûr. Le fait d'utiliser une valeur passée en argument sans la vérifier n'est pas une bonne idée. Nous nous en tenons ici au minimum pour assurer le fonctionnement, mais ceci demandera un peu plus de travail à l'avenir. Une autre amélioration consistera à utiliser la bibliothèque PyGame (voir l'article de Sébastien dans ce numéro) pour créer un affichage graphique et non textuel sur le mini écran LCD. Dans ce cas, la boucle parcourant l'ensemble des capteurs installés se fera dans le script Python modifié (et non via un **for** du shell), qui fonctionnera alors de manière continue. L'interrogation de tous les capteurs dans un seul script Python nous permettra aussi de comparer facilement les mesures et de procéder à des opérations mathématiques, comme faire des moyennes par exemple ou afficher des écarts par rapport à une consigne par capteur stockée dans un fichier de configuration.

Enfin, parmi les améliorations possibles, nous avons des petites choses plus cosmétiques que réellement nécessaires comme :

- utiliser les autres GPIOs de la Raspberry Pi pour indiquer de façon voyante la connexion d'un capteur et son état, avec des leds par exemple ;
- connecter un DS18B20 directement sur la Pi pour ajouter une source de mesure (le bus 1-wire comme le composant lui-même sont parfaitement pris en charge par Linux) ;
- remplacer le minuscule écran TFT économique par quelque chose de plus conséquent et complet, comme un écran tactile 4" ou 7" ;
- ajouter une interface avec quelques boutons pour éviter de connecter un clavier USB et leur attribuer quelques fonctions (mode été, forçage chauffage, etc.).

2. ET LA SUITE ?

À ce stade, nous disposons d'un système complet capable de collecter régulièrement les températures de tous les capteurs Wifi accessibles. C'est la base indispensable de la solution. L'étape suivante dans le processus nécessitera bien du travail d'investigation afin de prendre le contrôle de la chaudière. Je pourrai utiliser des moyens détournés pour arriver à mes fins. L'une des plus horribles et brouillonnes serait de tout simplement tromper le capteur de température dans le programmateur de la chaudière placé au salon. Je n'ai pas pris le temps de démonter l'équipement, mais il y a fort à parier qu'une simple thermistance (CTN) se charge de la mesure. Une autre solution pourrait être de trouver un moyen d'interfacer la Raspberry Pi en lieu et place du programmeur. Le protocole utilisé étant propriétaire et peu documenté, ceci risque de prendre du temps.

Mais dans le même ordre d'idée, il existe une autre solution : se connecter à la chaudière directement et obtenir bien plus que le contrôle de la mise en fonction. Ma chaudière à condensation Vaillant, comme bien des modèles récents d'autres marques (mais sans l'amusant logo à tête de lapin), dispose d'une prise spécifique (un RJ-12, le même que le RJ-11 des téléphones analogiques fixes, mais avec 6 connecteurs et non 4). Celui-ci sert normalement à connecter un outil de diagnostic appelé vrDIALOG ainsi que, semble-t-il, un appareil appelé vrnetDIALOG permettant

DÉCOUVREZ NOS NOUVELLES OFFRES D'ABONNEMENTS !

PRO OU PARTICULIER = CONNECTEZ-VOUS SUR :

www.ed-diamond.com

LES COUPLAGES PAR SUPPORT :

VERSION PAPIER

Retrouvez votre magazine favori en papier dans votre boîte à lettres !

VERSION PDF

Envie de lire votre magazine sur votre tablette ou votre ordinateur ?

Sélectionnez votre offre dans la grille au verso et renvoyez ce document complet à l'adresse ci-dessous !

Voici mes coordonnées postales :	
Société :	
Nom :	
Prénom :	
Adresse :	
Code Postal :	
Ville :	
Pays :	
Téléphone :	
E-mail :	

coromonadalix

❑ Je souhaite recevoir les offres promotionnelles et newsletters des Éditions Diamond.

❑ Je souhaite recevoir les offres promotionnelles des partenaires des Éditions Diamond.

En envoyant ce bon de commande, je reconnais avoir pris connaissance des conditions générales de vente des Éditions Diamond à l'adresse internet suivante : **boutique.ed-diamond.com/content/3-conditions-generales-de-ventes** et reconnais que ces conditions de vente me sont opposables.

HACKABLE
MAGAZINE

Édité par Les Éditions Diamond
Service des Abonnements
B.P. 20142 - 67603 Sélestat Cedex
Tél. : + 33 (0) 3 67 10 00 20
Fax : + 33 (0) 3 67 10 00 21

Vos remarques :

.....

.....

VOICI TOUTES LES OFFRES COUPLÉES AVEC HACKABLE !
POUR LE PARTICULIER ET LE PROFESSIONNEL ...

Prix TTC en Euros / France Métropolitaine

CHOISISSEZ VOTRE OFFRE !

Prix en Euros / France Métropolitaine

Offre	SUPPORT ABONNEMENT	PAPIER Réf	PAPIER Tarif TTC	PAPIER + PDF (PDF 1 lecteur) Réf	PAPIER + PDF Tarif TTC	PAPIER + BASE DOCUMENTAIRE (1 connexion BD) Réf	PAPIER + BASE DOCUMENTAIRE Tarif TTC	PAPIER + PDF + BASE DOCUMENTAIRE (PDF 1 lecteur + 1 connexion BD) Réf	PAPIER + PDF + BASE DOCUMENTAIRE Tarif TTC
HK	6n° HK*	☐ HK1	39,-	☐ HK12	58,-				
LES COUPLAGES « EMBARQUÉ »									
D	6n° HK* + 4n° OS	☐ D1	65,-	☐ D12	98,-	☐ D13	85,-*	☐ D123	118,-*
E	6n° HK* + 4n° OS + 6n° MISC	☐ E1	105,-	☐ E12	158,-	☐ E13	179,-*	☐ E123	232,-*
E+	6n° HK* + 4n° OS + 6n° MISC + 2n° HS	☐ E+1	119,-	☐ E+12	179,-	☐ E+13	193,-*	☐ E+123	253,-*
F	6n° HK* + 4n° OS + 11n° GLMF	☐ F1	125,-	☐ F12	188,-	☐ F13	229,-*	☐ F123	292,-*
F+	6n° HK* + 4n° OS + 11n° GLMF + 6n° HS	☐ F+1	183,-	☐ F+12	275,-	☐ F+13	287,-*	☐ F+123	379,-*
G	6n° HK* + 4n° OS + 6n° LP	☐ G1	100,-	☐ G12	150,-	☐ G13	164,- *	☐ G123	214,-*
G+	6n° HK* + 4n° OS + 6n° LP + 3n° HS	☐ G+1	129,-	☐ G+12	194,-	☐ G+13	193,-*	☐ G+123	258,-*
LES COUPLAGES « GÉNÉRAUX »									
H	6n° HK* + 4n° OS + 6n° LP + 6n° MISC + 11n° GLMF	☐ H1	200,-	☐ H12	300,-	☐ H13	402,-*	☐ H123	499,-*
H+	6n° HK* + 4n° OS + 6n° LP + 3n° HS + 6n° MISC + 2n° HS + 11n° GLMF + 6n° HS	☐ H+1	301,-	☐ H+12	452,-	☐ H+13	493,-*	☐ H+123	639,-*

Les abréviations des offres sont les suivantes : LM = GNU/Linux Magazine France | HS = Hors-Série | LP = Linux Pratique | OS = Open Silicium | HC = Hackable

* HK : Attention : La base Documentaire de Hackable n'est pas incluse dans l'offre.

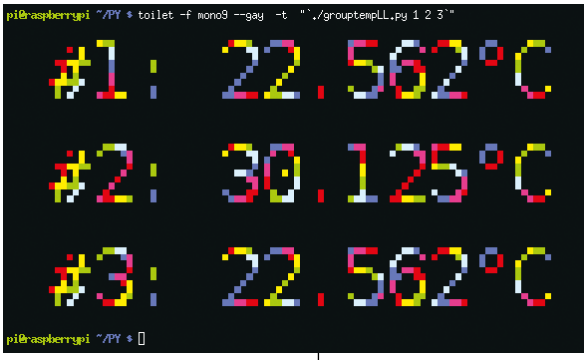
N'hésitez pas à consulter les détails des offres ci-dessus sur : www.ed-diamond.com !

de connecter la chaudière... au web (?!). Quoi qu'il en soit, le protocole utilisé est eBUS (*energy bus*) vastement déployé parmi les constructeurs allemands pour bien des équipements. Niveau matériel, il s'agit simplement d'un bus série TTL à 2400 bps sur lequel repose une couche de communication standardisée.

Là où le bât blesse en revanche, c'est dans la couche de plus haut niveau (application) définissant la nature et la syntaxe des messages qui transitent. Vaillant, comme d'autres, utilise un format de données propriétaire sur cette interface eBUS. Tout le jeu (dangereux) consiste donc à trouver les bonnes informations et à les vérifier (souvent à partir de sites en russe ou en allemand). C'est un travail de longue haleine mais, à terme, on peut prendre le contrôle complet de la chaudière, obtenir plein d'informations intéressantes et même revoir sa configuration (mais là on joue littéralement avec le feu).

L'article d'Axelle et Ludovic dans le présent numéro détaille le pilotage d'une chaudière fioul plus ancienne et bien plus facile à contrôler. Si votre équipement a quelques années, il est fort probable que la solution qu'ils ont mise en œuvre pour leur installation puisse parfaitement s'appliquer à la vôtre.

L'autre objectif sur ma liste demandera moins de travail puisqu'il s'agira simplement de piloter des VMC (Ventilation Mécanique Contrôlée) entre l'intérieur et l'extérieur et, pourquoi pas, entre les pièces. Dans ce genre de scénario, on se laisse aisément emporter par l'idée consistant à transférer l'air chaud d'une pièce vers une autre plutôt que de mettre en route le chauffage. Ceci demandera quelques expérimentations et mesures, mais l'option existe (reste à savoir si elle est efficace).



Si vous souhaitez afficher du texte de manière très voyante sans pour autant basculer en mode graphique, vous pouvez utiliser les outils figlet et toilet (paquets éponymes). Les deux permettent de choisir une police de caractères et le second offre même une option d'affichage en arc-en-ciel. Il suffit de rendre le script moins bavard et d'utiliser l'apostrophe inverse pour passer la sortie de votre script à figlet ou toilet.

3. AUTRES PERSPECTIVES

La solution présentée ici s'adaptera sans doute à bien des réalisations. Pour rester dans le domaine de la mesure de température par exemple, on peut envisager d'autres applications comme la température d'un composte, d'un bassin ou d'un jardin (détection du risque de gel). Il est également envisageable de mesurer d'autres choses comme des pressions, des positions, un ensoleillement, un taux d'hygrométrie. La simple utilisation d'un convertisseur analogique/numérique (ADC) interfacé en 1-wire (comme le DS18B20) offre bien des perspectives. D'autres capteurs en i2c ou SPI nécessiteront cependant d'opter pour un module ESP8266 plus complet, avec plus de ports (comme le NodeMCU ou celui vendu par Olimex).

Inversement, il est également possible d'utiliser son réseau de capteurs non pas pour collecter des données, mais pour provoquer des actions, voire combiner les deux options pour obtenir une interaction complète.

Enfin, n'oublions pas de préciser que la Raspberry Pi offre à elle seule une foule de possibilités tant pour présenter les données (écran, serveur Web, passerelle pour le *cloud*, etc.) que pour agir en guise de « cerveau de la maison ». Reproduire ainsi l'intelligence du Google Nest consistant à analyser les demandes répétées de l'utilisateur pour en déduire des règles de fonctionnement ne devrait pas être hors de portée. Avec un peu de ténacité, un bel écran tactile et quelques leds type WS2812b, il devrait même être possible de faire encore plus flashy que l'original... **DB**

coromonadalix

CONTRÔLER SA CHAUDIÈRE À DISTANCE AVEC UN RASPBERRY PI

Axelle et Ludovic Apvrille

Que ce soit par souci d'économie d'énergie (et de facture…) ou par confort, les envies de contrôler sa chaudière ne manquent pas. Électriciens, installateurs ou revendeurs de domotique pourront très certainement vous proposer une solution adéquate, seulement elle risque d'être assez onéreuse (plus de 150 euros) [1]. Alors, pourquoi ne pas le faire soi-même ?

coromonadalix

C'est ce que nous avons fait ! Nous avons relié notre propre chaudière fioul, une Viessmann Vitola, à un Raspberry Pi connecté à Internet, et pilotons donc notre chauffage à distance. Voici comment faire...

1. AVERTISSEMENT

Nous vous expliquons comment nous avons fait, mais bien sûr si vous voulez faire pareil, bon sens et précautions d'usage sont indispensables. Si vous ne connaissez pas bien votre chaudière, contactez votre chauffagiste et/ou électricien pour lui demander les implications sur votre équipement.

2. MATÉRIEL

Nous supposons que vous disposez d'un Raspberry Pi en état de marche, c'est-à-dire avec son alimentation, sa carte SD et une distribution installée dessus. Dans notre cas, nous avons utilisé un Raspberry Pi modèle B Rev 2.0 512 MB (Egoman). Ce modèle est maintenant remplacé par un modèle B+ ou même un Pi 2, voire un A+, qui conviendront tout aussi bien, et que l'on peut acquérir pour environ 25 euros (le A+).

Sur la carte SD, nous avons personnellement installé une Raspbian, mais une autre distribution Linux devrait faire l'affaire pourvu que vous puissiez piloter les ports GPIO dont nous parlerons plus bas.

Enfin, c'est optionnel (mais quasi obligatoire à notre avis !), l'on vous conseille l'achat d'un boîtier protégeant le Raspberry Pi. Sachant qu'on en trouve à 5 euros, ce serait dommage de faire l'impasse dessus et d'abîmer la carte... Si c'est pour votre salon, il en existe de plus beaux, plus design... plus chers :)

Il n'y a que très peu de matériel additionnel à acheter :

- des fils à *breadboard* mâle/mâle (**https://hackspark.fr/fr/catalogsearch/result/?q=CON00002**) - 7 euros ;
- module de relais 4 canaux, 5V. En fait, nous n'avons besoin que de 3 relais, il y en aura donc un inutilisé (**https://hackspark.fr/fr/catalogsearch/result/?q=RELITD4CH5V**) - 9.60 euros. Idéalement, il faudrait que les relais soient opto-couplés en cas de retour de charge de la chaudière vers le Raspberry Pi ;
- un interrupteur 250V électrique classique - environ 8 euros s'il est de marque ;
- des fils électriques pour câbler - le prix - faible - variant selon la distance entre le Rapsberry Pi et la chaudière, bien sûr ;
- une sonde de température dans la maison, dont la valeur peut-être lue par le Raspberry Pi.

3. CIRCUIT ÉLECTRIQUE

Dans la pratique, notre chaudière (au sous-sol) sert à deux choses distinctes :

1. avoir de l'eau chaude sanitaire ;
2. chauffer la maison (radiateurs en fonte).

Plus précisément, chauffer la maison revient à chauffer de l'eau dans un ballon (« eau de chauffage »), puis, à l'aide d'une pompe, à envoyer cette eau chaude vers les radiateurs en fonte de la maison. La pompe est régulée par un thermostat mécanique manuel, situé dans notre salon. Dans la pratique, dès que la température du salon descend en dessous d'un certain seuil, cela démarre la pompe. De l'eau chaude circule alors dans les radiateurs, les pièces chauffent jusqu'à une température agréable. Puis lorsque le thermostat dépasse la température seuil, il coupe la pompe, et ainsi de suite.

Notons que l'eau chaude sanitaire et l'eau de chauffage sont complètement indépendantes, et il y a un ballon de chauffage pour chaque. La chaudière permet d'ailleurs deux modes de fonctionnement :

- mode (i) : chauffer l'eau chaude sanitaire uniquement ;
- mode (ii) : chauffer à la fois l'eau sanitaire et l'eau de chauffage.

coromonadalix

Ces modes de fonctionnement se sélectionnent directement depuis la chaudière elle-même. On ne les pilote pas à distance. En revanche, à distance, nous souhaitons piloter :

1. L'arrêt ou la marche de la chaudière. Lorsque la chaudière est arrêtée, il n'y a rien : ni eau chaude sanitaire, ni eau de chauffage. Lorsqu'elle est en route, il y a toujours au moins de l'eau chaude sanitaire.
2. L'arrêt ou le mode de déclenchement du chauffage. Dans la pratique, nous pouvons choisir à distance entre 3 options :
   - (a) Le chauffage est coupé.
   - (b) Le chauffage se déclenche grâce au thermostat manuel du salon. Nous laissons le thermostat à une certaine température, et sommes ainsi assurés que la maison restera chauffée à cette température. Évidemment, il n'est pas possible de bouger le thermostat à distance, c'est pourquoi il y a un 3ème cas...
   - (c) Le chauffage est régulé par un programme informatique fait maison tournant sur le Rapsberry Pi, et implémentant des algorithmes de notre choix (par exemple, en fonction de la température extérieure).

Nous avons fait cela sans toucher à la chaudière elle-même, c'est-à-dire sans intervenir sur son câblage ou ses cartes électroniques internes. En effet, l'intervention sur le câblage interne permettrait plus de possibilités, notamment le contrôle des sondes de température des deux ballons d'eau.

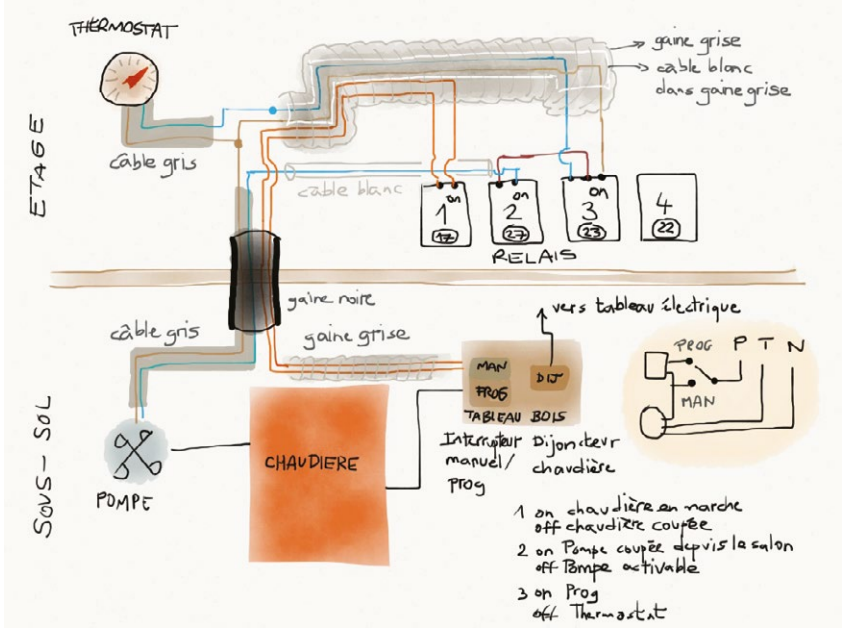


Figure 1 : Schéma général du câblage de la chaudière.

Un schéma du circuit électrique que nous avons effectué est illustré à la Figure 1. La chaudière (et sa pompe), ainsi qu'un interrupteur et disjoncteur, se trouvent au sous-sol de notre habitation. Au rez-de-chaussée, dans le salon, on trouve le thermostat manuel, les relais, notre Raspberry Pi et notre connexion à Internet.

L'interrupteur 230V au sous-sol sert à basculer la chaudière en mode manuel (sans aucune possibilité de pilotage externe... le mode utilisé avant) ou en mode programme (contrôle à distance par le Raspberry Pi). Quelle que soit votre dextérité informatique, nous ne pouvons qu'insister sur la pose de cet interrupteur. Que se passera-t-il si le Raspberry Pi tombe en panne ? Si la carte à relais grille ? Il ne faut surtout pas se retrouver dans une situation où on ne peut plus arrêter ou démarrer la chaudière. Vous ne serez que trop heureux d'avoir cet interrupteur

coromonadalix

Notre superbe chaudière, avec sa poussière et ses interrupteurs. L'interrupteur noir (à droite) est un disjoncteur à l'ancienne (rassurez-vous, on a de vrais disjoncteurs sur un vrai tableau électrique ailleurs !). L'interrupteur gris (au centre) permet de basculer la chaudière entre mode manuel et mode piloté à distance. Sage précaution...

en cas d'imprévu ou d'urgence, ou alors lorsque vous prêtez la maison à des amis : prévoyez-le et prenez le temps de le poser.

À l'étage, les 3 relais pilotent :

1. la marche/arrêt de la chaudière ;
2. l'activation/désactivation de la pompe ;
3. le déclenchement de la pompe via thermostat ou via programmation.

Le 4ème relais n'est pas utilisé.

Le 2ème relais n'a pas d'utilité si l'on déclenche la pompe à l'aide du thermostat, car le thermostat a intrinsèquement la capacité d'activer ou de couper la pompe.

En revanche, ce relais est utile pour le déclenchement via programme : le programme activera ou déclenchera la pompe en contrôlant ce relais.

Les relais ont tous deux états possibles :

- NC - *Normally Close* (valeur par défaut au démarrage du Raspberry Pi) : nous appelons par la suite cet état "0".
- NO - *Normally Open* : nous appelons cet état par la suite "1".

Si le Raspberry Pi venait à planter/redémarrer, l'on souhaite se retrouver dans un état sans risque où la chaudière est coupée, et si on l'active, par défaut, nous souhaitons que la régulation de la température se fasse par le thermostat manuel intérieur et que la pompe soit coupée. Oui, c'est étrange, on n'a pas totalement confiance en nos propres programmes...

Donc, la valeur par défaut (0) du relais 1 (arrêt/marche chaudière) doit correspondre à une chaudière coupée.

Le 0 du relais 2 doit être pompe coupée.

Et enfin, le 0 du relais 3 doit correspondre à la régulation par le thermostat.

Cela donne un schéma logique comme à la figure 2.

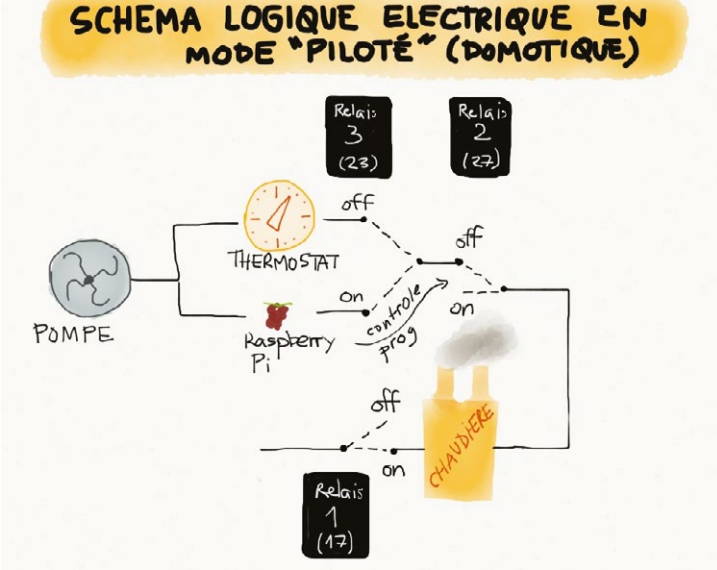


Figure 2 : Schéma logique du câblage des GPIOs du Raspberry Pi.

4. CONFIGURATION DU RASPBERRY PI

Nous allons commander les relais à l'aide des GPIO du Raspberry Pi. En anglais, GPIO veut dire *General Purpose Input/Output*, c'est-à-dire entrée/sortie à usage général. Nous disposons d'un Raspberry Pi modèle B « revision 2.0 ». En cherchant sur Internet, l'on trouve très facilement le schéma des broches de ce modèle (attention à bien prendre un schéma pour le bon modèle) **[2]**. Les GPIO des

coromonadalix

À gauche, le Raspberry Pi avec les GPIOs. À droite, une boite artisanale contenant les relais.

Raspberry Pi sont numériques uniquement, c'est-à-dire, qu'ils ont un état haut ou bas (1 ou 0).

Le relais 1 est connecté dans notre cas sur le GPIO 17, le 2 sur le 27 et le 3 sur le 23, sans aucune raison particulière (il fallait bien faire un choix). Chaque relais possède une diode qui est allumée s'il est dans l'état 1, et éteinte dans l'état 0. Les relais sont alimentés en 5V via la broche 5V du Raspberry Pi. Enfin, la masse du Raspberry Pi est aussi reliée aux relais.

Du point de vue configuration logicielle du Raspberry Pi, l'utilisation des GPIOs nécessite :

1. De réserver le GPIO. En anglais on parle de l'*exporter*. Par exemple, pour réserver l'utilisation du GPIO 17, on fait (dans un terminal) :

```
$ echo "17" > /sys/class/gpio/export
```

2. Ensuite, de régler sa direction. Cela consiste à dire si la broche fonctionne en entrée (*in*) ou en sortie (*out*). Dans notre cas, on souhaite commander les relais à partir du Raspberry Pi, c'est donc une sortie :

```
$ echo out > /sys/class/gpio/gpio17/direction
```

5. PROGRAMME POUR PILOTER LA CHAUDIÈRE

Le programme pour piloter la chaudière est un simple script Bash.

Peu importe le langage utilisé, l'essentiel est de conserver un programme simple, facile à relire et débogger.

Notre programme comprend plusieurs commandes :

- **`chaudiere_on`** / **`chaudiere_off`** pour démarrer/arrêter la chaudière ;
- **`chaudiere_status`** : pour connaître l'état actuel de la chaudière (marche/arrêt) ;
- **`chauffage_manuel`** : pour réguler le chauffage à l'aide du thermostat manuel. À noter que pour que le chauffage fonctionne, la chaudière doit être allumée !
- **`chauffage_prog`** : pareil, mais le chauffage est régulé par un programme ;
- **`chauffage_off`** : pour couper le chauffage ;
- **`chauffage_status`** : pour connaître l'état actuel du chauffage (thermostat, programmation, arrêté).

Chacune de ces commandes contrôle directement le GPIO qui lui correspond.

Par exemple, le démarrage de la chaudière fait simplement :

```
echo "1" > /sys/class/gpio/gpio17/value
```

La commande **`chauffage_manuel`** fait :

```
echo "0" > /sys/class/gpio/gpio23/value
echo "0" > /sys/class/gpio/gpio27/value
```

Les commandes **`chaudiere_off`** et **`chauffage_prog`** ne sont pas plus compliquées à implémenter.

coromonadalix

Les commandes de *status* consistent à lire l'état des GPIO. Par exemple, `chaudiere_status` lit la valeur du GPIO 17 et la traduit en `On` ou `Off` :

```
function status_chaudiere() {
    local status='cat /sys/class/
gpio/gpio17/value'
    if [ $status = "0" ]; then
 chaudiere="Off"
    else
 chaudiere="On"
    fi
}
```

Pour débogger plus facilement, chaque commande est loggée dans le log système du Raspberry Pi (`/var/log/syslog`).

Pour cela, on réutilise un script très bien fait de `/lib/lsb/init-functions` :

```
# import de init-functions
. /lib/lsb/init-functions
# nécessaire pour qu'init-
functions logge correctement
NAME="CHAUDIERE"
...
# exemple de log
log_daemon_msg "Arret du
chauffage" "$NAME"
```

6. SITE WEB

Les scripts Bash, c'est pour les geeks :) mais comment frimer auprès d'amis non informaticiens ? Avec un site web !

Nous sommes très loin d'être des experts en site web, nous avons opté pour un site web, simple, qui s'affiche sans difficulté quel que soit le navigateur, y compris depuis notre téléphone sous Firefox OS ;) Si ce n'est pas déjà fait, courez donc installer Apache et PHP sur votre Raspberry Pi : il existe de nombreux tutoriels là dessus **[3]**.

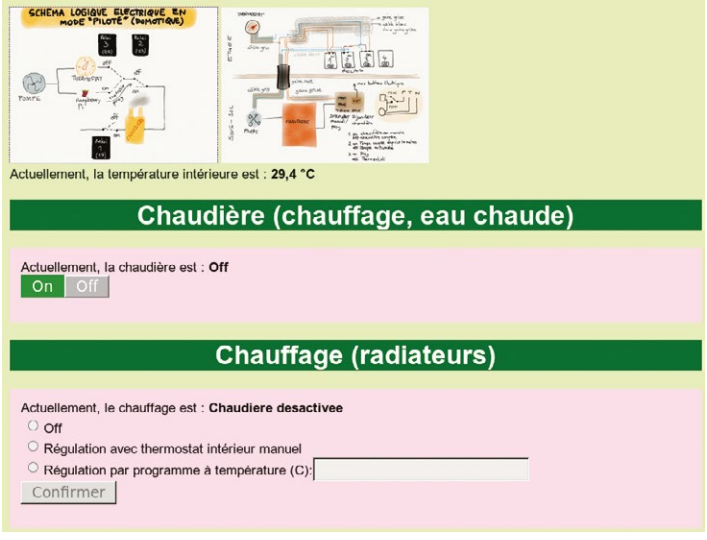
En outre, n'oubliez pas de sécuriser ce site web notamment avec un mécanisme d'authentification (`.htaccess` par exemple), ou carrément en ne le rendant accessible que depuis vos machines (réseau virtuel).

Le site web affiche l'état actuel de la chaudière et du chauffage. À l'aide d'un bouton, on peut allumer/éteindre la chaudière. Puis, si la chaudière est allumée, on a le choix entre réguler le chauffage par thermostat, par programme ou l'éteindre complètement.

Le site web affiche l'état actuel de la chaudière en appelant notre script Bash depuis le PHP :

```
   echo "Actuellement, la chaudi&egrave;re
est : <b>";
   $output = array();
   exec('/usr/bin/chaudiere.sh chaudiere_
status 2>&1', $output);
   echo $output[0];
```

Si vous n'êtes pas familiers des sites web, il y a un petit détail à régler. Par défaut, le serveur web tourne sous sa propre identité, par exemple `www-data`. Or, le pilotage de GPIO requiert par défaut d'être root, donc pour lancer le script `chaudiere.sh`, il faudrait être `root`, pas `www-data`. La première solution qui vient à l'esprit est de faire un `sudo` (éventuellement en restreignant les droits dans `/etc/sudoers`). Mieux, il suffit d'ajouter `www-data` dans le groupe `gpio`, ce qui lui donne le droit de manipuler GPIOs.



Interface web de commande de notre chaudière. Oui, c'est l'été, il fait 29.4° à l'intérieur, alors elle est coupée !

Le bouton et le radio bouton sont gérés par un formulaire HTML. Il suffit ensuite de lancer la bonne commande à notre script Bash, par exemple : **chaudiere.sh chaudiere_on**.

```
if (! empty($_POST["button_chaudiere_on"])) {
    exec('/usr/bin/chaudiere.sh chaudiere_on 2>&1', $output);
}
```

Enfin, on grise/active le bouton On/Off avec un petit peu de JavaScript :

```
// $output[0] contient la réponse du script à la requête de demande d'état
if ($output[0] == "On") {
   echo "<button name='button_chaudiere_on' class='btndisabled'
disabled='True'>On</button>";
   echo "<button name='button_chaudiere_off' class='btnred' value='Off'
onClick='window.location.reload()'>Off</button>";
} else {
  echo "<button value='On' name='button_chaudiere_on' class='btngreen'
onClick='window.location.reload()'>On</button>";
   echo "<button value='Off' name='button_chaudiere_off' class='btndisabled'
disabled='True'>Off</button>";
 }
```

De même, on sélectionne le bon radio bouton :

```
if ($output[0] == 'Off') {
    echo "<input type='radio' name='chauffage' value='off' checked>Off<br>";
  } else {
    echo "<input type='radio' name='chauffage' value='off' >Off<br>";
  }
  if ($output[0] == 'Manuel') {
    echo "<input type='radio' name='chauffage' value='manuel'
checked>R&eacute;gulation avec thermostat int&eacute;rieur manuel<br>";
  } else {
    echo "<input type='radio' name='chauffage' value='manuel'>R&eacute;
gulation avec thermostat int&eacute;rieur manuel<br>";
  }
  ...
```

7. PROGRAMMATION HORAIRE

Le contrôle par site web ou par script Bash est pratique pour contrôler l'état et gérer les cas particuliers, mais la plupart du temps la mise en route de la chaudière peut être programmée automatiquement, par exemple mise en marche de 7h à 22h.

C'est là qu'on mesure l'avantage d'avoir un simple script pour contrôler la chaudière : il suffit pour la programmer de mettre la commande dans la *crontab*, et le tour est réglé.

Par exemple, la commande ci-dessous indique qu'il faut couper la chaudière tous les soirs à 22h30.

```
$ sudo crontab -l
30 22 * * * /usr/bin/chaudiere.sh chaudiere_off
```

coromonadalix

8. DÉCLENCHEMENT DE LA POMPE PAR THERMOSTAT PROGRAMMÉ

Si l'on ne désire pas utiliser le thermostat mécanique placé dans la maison, par exemple, pour pouvoir choisir une température à distance, il est possible de réaliser un programme qui contrôle la pompe de circulation d'eau, par exemple en fonction de la température relevée par une sonde (notamment celle d'une station météo).

La manière de lire la température dépend de la station météo et du logiciel qui la gère. Pour d'autres besoins, nous lisons sans problème les températures intérieures et extérieures d'une station météo Oregon Scientific WMR 200 gérée par le logiciel weewx (**http://weewx.com/**). La lecture consiste en une simple requête SQL :

```
select inTemp from archive where dateTime=XXX
```

où **XXX** correspond à l'*epoch* du dernier relevé de température.

Bref, c'est parfaitement faisable, et pas très compliqué. Nous l'avions prévu (cf. les captures d'écran du site web !) ; mais dans la pratique, on ne l'a pas (encore ?) implémenté, car le déclenchement via le thermostat mécanique nous convient...

CONCLUSION

Voilà, vous en savez normalement suffisamment pour avoir chaud l'hiver prochain, tout en frimant avec votre propre site web sur votre smartphone !

Il y a bien sûr de nombreuses optimisations possibles. Par exemple, dans notre cas, dès que la chaudière est en marche, elle chauffe l'eau de chauffage (celle qui circule dans les radiateurs). Or, si le chauffage est coupé, c'est parfaitement inutile. Notre chaudière permet de couper le chauffage de cette eau, mais nous ne l'avons pas pris en compte dans notre système de contrôle (il nécessite de modifier le câblage interne de la chaudière).

Les différents algorithmes de déclenchement/arrêt de la pompe sont également un autre domaine où votre imagination peut s'exprimer... **ALA**

RÉFÉRENCES

[1] https://www.aruco.com/2015/04/thermostats-connectes/

[2] http://www.raspberrypi-spy.co.uk/wp-content/uploads/2012/09/Raspberry-Pi-GPIO-Layout-Revision-2.png

[3] http://raspbian-france.fr/installer-serveur-web-raspberry/

coromonadalix

SUPERVISEZ VOTRE CONSOMMATION ÉLECTRIQUE SUR RASPBERRY PI

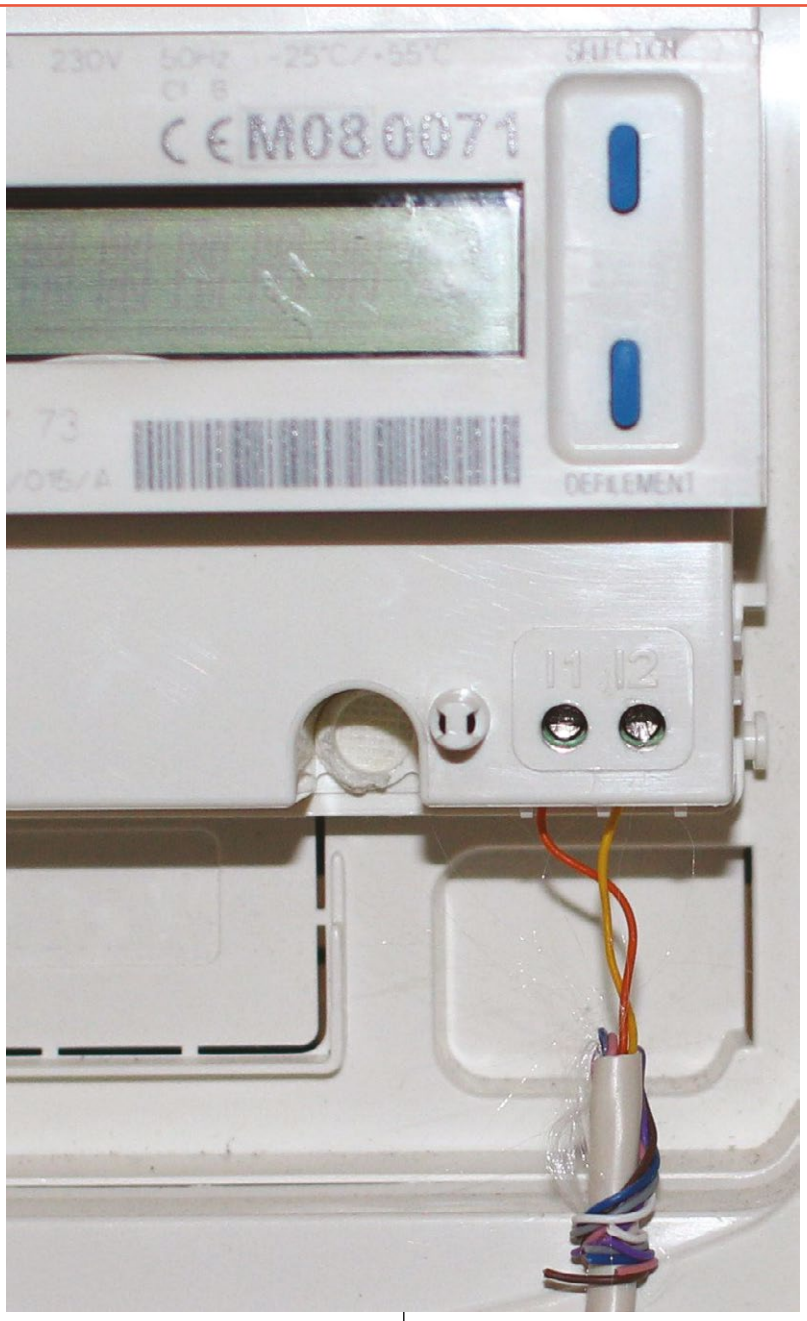
Sébastien Maccagnoni-Munch

Les compteurs de courant électroniques, installés dans les foyers français par ERDF, comportent une sortie appelée « téléinformation », qui permet de recevoir le détail des informations sur un autre appareil. Habituellement utilisée par des boîtiers appelés « gestionnaires d'énergie », cette sortie peut être connectée sur un ordinateur…

coromonadalix

1. LA TÉLÉINFORMATION

En France, deux bornes présentes sur les compteurs électroniques (les compteurs blancs, les plus récents...) sont dédiées à la sortie « téléinformation », sur laquelle sont envoyés l'ensemble des informations concernant votre abonnement et votre consommation d'électricité. Ces bornes (nommées *I1* et *I2*) sont situées au bas à droite du compteur. Bien sûr, cette sortie n'existe pas sur les compteurs électromécaniques (à roue), qui n'embarquent pas d'électronique. Par ailleurs, elle est reprise sur le nouveau compteur « intelligent » Linky, actuellement en expérimentation, qui sera déployé à grande échelle dans les années qui viennent. Enfin, il faut que cette sortie ait été activée, ce qui n'est pas systématique ; quand ce n'est pas le cas, il faut faire une demande au fournisseur d'énergie afin qu'il planifie le déplacement d'un agent ERDF...



La sortie de téléinformation d'un compteur.

1.1 Format des données

Cette téléinformation est envoyée sous forme de signaux modulés à 50 kHz : la présence de modulation correspond à un 0 logique, l'absence de tension à un 1. Les informations remontées sont envoyées en continu, de manière cyclique, sous forme de caractères ASCII, à la vitesse de 1200 bauds. Chaque groupe d'informations (trame) commence par un caractère STX (*start of text*, code ASCII 2) et termine par un caractère ETX (*end of text*, code ASCII 3). Enfin, au sein de cette trame, chaque donnée est formatée de la manière suivante :

LF (*line feed*, ASCII 0x0A)	Étiquette (4 à 8 caractères)	Espace (ASCII 0x20)	Donnée (1 à 12 caractères)	Espace (ASCII 0x20)	Caractère de contrôle	CR (*carriage return*, ASCII 0x0D)

1.2 Étiquettes et données

De nombreuses données sont envoyées par votre compteur, liées à votre abonnement ou à votre consommation. Par ailleurs, les données envoyées diffèrent selon votre type d'abonnement (nombre de phases, tarification, etc.). Voici une liste non exhaustive de couples étiquette/données envoyés par le compteur :

Étiquette	Description	Unité	Nombre de caractères
OPTARIF	Tarif souscrit (valeur BASE, HC.., EJP. ou BBRx, x étant variable selon la configuration pour la sortie auxiliaire/déclenchement heure creuse)	/	4
ISOUSC	Intensité souscrite	A	2
BASE	Index de consommation en abonnement « base »	Wh	9
HCHC	Index des heures creuses en abonnement HP/HC	Wh	9
HCHP	Index des heures pleines en abonnement HP/HC	Wh	9
BBRHxJy	Index des six périodes tarifaires, pour un abonnement avec option « tempo » (x prend pour valeur « C » pour les heures creuses, « P » pour les pleines ; y prend pour valeur « B » pour les jours bleus, « W » pour les blancs et « R » pour les rouges)	Wh	9
PTEC	Période tarifaire en cours en option tempo (valeur BLEU, BLAN ou ROUG)	/	4
DEMAIN	Période tarifaire du lendemain en option tempo (valeur BLEU, BLAN ou ROUG)	/	4
IINST	Intensité instantanée	A	3
IMAX	Intensité maximale appelée, depuis la mise en service	A	3

La liste complète des étiquettes peut être consultée dans les spécifications détaillées de la sortie de téléinformation, disponible sur le site d'ERDF : **http://www.erdf.fr/sites/default/files/ERDF-NOI-CPT_02E.pdf**.

1.3 Caractère de contrôle

Le caractère contrôle présent après la valeur, quant à lui, correspond au calcul suivant :

- on fait la somme des valeurs ASCII de tous les caractères allant du début de l'étiquette à la fin de la donnée, espace incluse ;
- on ne conserve que les 6 bits de poids faible (ce qui correspond à un ET logique entre la somme ci-dessus et la valeur 3F en hexadécimal) ;
- on ajoute 20 en hexadécimal.

Le résultat sera toujours un caractère compris entre 20 (le caractère) et 5F (le caractère de soulignement ou *underscore*).

Par exemple, pour obtenir le caractère de contrôle pour l'étiquette « ISOUSC » avec la valeur 45, on peut exécuter la commande suivante :

```
$ python -c "print chr((sum(bytearray('ISOUSC 45')) & 0x3f) + 0x20)"
?
```

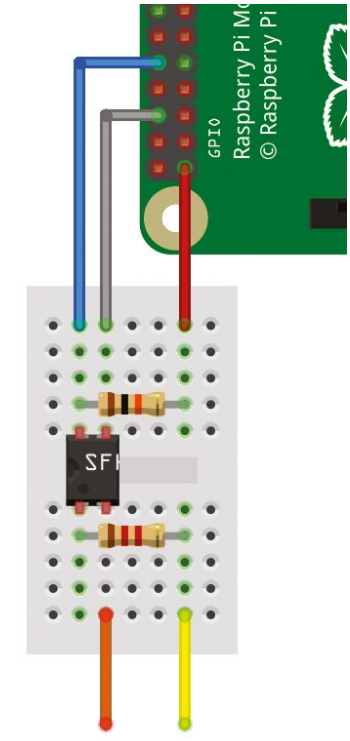
coromonadalix

On remarque, un peu plus loin dans cet article, que c'est bien le caractère « ? » qui est envoyé par le compteur pour cette valeur.

2. CONNEXION AU RASPBERRY PI

Pour lire ces données, nous allons brancher cette sortie de téléinformation sur l'entrée série d'un Raspberry Pi (nous avons également testé et validé cela avec un Beaglebone Black). Mais pour cela, il faut transformer ce signal modulé en signal logique : le Raspberry Pi n'est pas capable d'interpréter cette modulation de lui-même.

Pour cela, on va utiliser un optocoupleur SFH620A, accompagné de deux résistances...



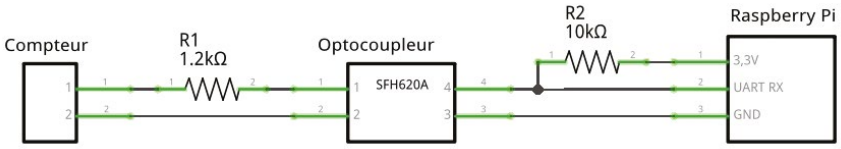
Bornes I1 et I2 du compteur

Le montage est très simple :

- d'un côté, les sorties de la téléinformation sont en « entrée » de l'optocoupleur, avec une résistance de 1,2 kOhm sur la patte 1 (il n'y a pas de polarité, on peut brancher les bornes de la téléinformation dans n'importe quel sens) - entre la sortie du compteur et le montage, on peut utiliser un câble à paires croisées de type câble téléphonique par exemple ;
- de l'autre côté, l'optocoupleur est directement branché sur le Raspberry Pi, avec une résistance de tirage (pull-up), qui est ici de 10 kOhm.

Ces trois composants peuvent être achetés ensemble chez certains vendeurs en ligne, sous le nom « kit téléinfo ».

Notons que 10 kOhm est une résistance trop élevée pour le pull-up sur un Beaglebone Black : cela fonctionne lorsque l'on descend à 3,3 kOhm, en mettant trois de ces résistances en parallèle. Nous n'avons pas testé d'autre matériel, à vous de vous assurer que cette résistance est bien dimensionnée...



Côté Raspberry Pi, on utilise trois broches :

- l'alimentation 3,3v (broche 1) ;
- la masse (broche 6) ;
- l'entrée du port série (UART RX, broche 10).

On ne branche pas la sortie du port série : ce canal de communication est unidirectionnel.

La configuration du port série nécessaire pour la téléinformation est la suivante :

- 1200 bits par seconde ;
- caractères sur 7 bits ;
- un bit de parité, paire ;
- un bit d'arrêt, 1 logique.

Ceci peut être résumé par la notation « 1200 7E1 ». On utilise donc 10 bits pour transmettre un caractère (7 bits de données, un bit de départ, un bit de parité et un bit d'arrêt) ; cela donne 120 caractères par seconde.

2.1 Lecture

Commençons par créer un petit script très simple en Python pour lire les données reçues sur le port série...

```
#!/usr/bin/env python

import serial, sys

s = serial.Serial(port='/dev/ttyAMA0', baudrate=1200, bytesize=serial.SEVENBITS,
                  parity=serial.PARITY_EVEN, stopbits=serial.STOPBITS_ONE)
while True:
  data = s.read(1)
  if data == '\2':   sys.stdout.write('\n===== DEBUT =====')
  elif data == '\3': sys.stdout.write('\n===== FIN =====')
  else:              sys.stdout.write(data)
```

On initialise d'abord le port série avec les paramètres adéquats, puis on lance une boucle infinie dans laquelle on lit les données caractère par caractère. Si on rencontre un STX, on indique qu'il s'agit d'un début de trame ; si on rencontre un ETX, on indique qu'il s'agit d'une fin de trame ; sinon on affiche le caractère rencontré. Pour sortir de la boucle, on ne prévoit rien : le programme s'arrêtera avec un simple *Ctrl-C*, rien de plus classique.

```
# ./teleinfo.py
C Y D
MOTDETAT 000000 B
===== FIN =====
===== DEBUT =====
ADCO 020828428827 J
OPTARIF BBR( S
ISOUSC 45 ?
BBRHCJB 026426186 @
BBRHPJB 041644955 P
^C
```

On constate qu'on reçoit bien les données dans le format voulu, le début de la réception n'étant pas nécessairement au début d'une trame, ni même au début d'une « ligne ».

2.2 Extraction

Étant donné qu'on travaille en Python, l'idéal est de stocker l'ensemble de ces valeurs dans un dictionnaire ! Créons alors un script qui lit les données reçues, qui valide les lignes, qui stocke les données et qui envoie des trames complètes à une fonction tierce...

```
#!/usr/bin/env python

import datetime
import serial

def got_frame(f):
  print f

if __name__ == '__main__':
  s = serial.Serial(port='/dev/ttyAMA0', baudrate=1200, bytesize=serial.SEVENBITS,
                    parity=serial.PARITY_EVEN, stopbits=serial.STOPBITS_ONE)
  char = None
  while char != '\x03': char = s.read(1) # Attendre une fin de trame
  while True:
    char = s.read(1)
    if char == '\x03': # Fin de trame, on envoie la trame en traitement
      got_frame(frame)
    elif char == '\x02': # Debut de trame, on reinitialise la trame
      frame = {}
    elif char == '\n': # Debut de ligne, on reinitialise la ligne
      line = ''
    elif char == '\r': # Fin de ligne, on traite les donnees
      try: tag, data, checksum = line.split()
      except ValueError: continue # Ligne invalide, on laisse tomber
      checksum = ord(checksum)
      calculated = (sum(bytearray(tag+' '+data)) & 0x3f) + 0x20
      if checksum == calculated: # Ligne prise en compte uniquement si valide
        frame[tag] = data # Tout est valide, on ajoute la donnee dans le dico
    else:
      line = line + char
```

Ce script reçoit les données du port série, caractère par caractère :

- quand il rencontre un début de trame, il crée un nouveau dictionnaire avec sa date de début ;
- quand il rencontre une fin de trame, il exécute la fonction `got_frame` avec le dictionnaire comme argument ;
- quand il rencontre un début de ligne, il crée une nouvelle ligne vide ;
- quand il rencontre une fin de ligne, il valide la ligne courante puis ajoute les données dans le dictionnaire ;
- quand il rencontre autre chose, il complète la ligne courante.

La fonction `got_frame`, quant à elle, ne fait pour l'instant qu'une impression à l'écran du dictionnaire reçu...

Chacun pourra alors modifier cette fonction afin d'effectuer les opérations souhaitées : suivi et statistiques de consommation, puissance nécessaire selon la journée, adéquation (ou non) de l'abonnement choisi (avec comparatif à d'autres tarifs), envoi des données à une autre machine...

coromonadalix

3. FAIRE UN GRAPHIQUE

Pour avoir un début de résultat intéressant, on va générer un fichier d'historique des intensités instantanées, exploitable par **gnuplot**. Pour cela, la fonction **got_frame** va être remplacée par le code ci-dessous :

```
def got_frame(f):
  date = ('{d.year:04d}-{d.month:02d}-{d.day:02d}-{d.hour:02d}:'
          '{d.minute:02d}:{d.second:02d}'.format(d=f['date']))
  with open('intensite.data', 'a') as outfile:
    outfile.write('{} {} {}\n'.format(date, int(f['IINST']),
int(f['ISOUSC'])))
```

On obtient alors un fichier dont le contenu ressemble aux lignes suivantes :

```
2015-05-21-07:12:46 17 45
2015-05-21-07:12:48 16 45
2015-05-21-07:12:51 16 45
2015-05-21-07:12:53 14 45
2015-05-21-07:12:56 12 45
```

Ce fichier peut alors être transféré à tout moment vers un autre ordinateur, disons un poste de travail sous Linux. Pour exploiter ce fichier, on va alors créer sur ce poste de travail un fichier de définition de graphe pour **gnuplot**, que nous appellerons **intensite.plot** :

```
set terminal png size 800,400
set xdata time
set timefmt "%Y-%m-%d-%H:%M:%S"
set output "intensite.png"
set title "Consommation électrique"
set xlabel "Date/heure"
set ylabel "Intensité"
set key left box
set yrange [0:50]
plot "intensite.data" using 1:2 title "Intensité" with line,
"intensite.data" using 1:3 title "Maximum" with line
```

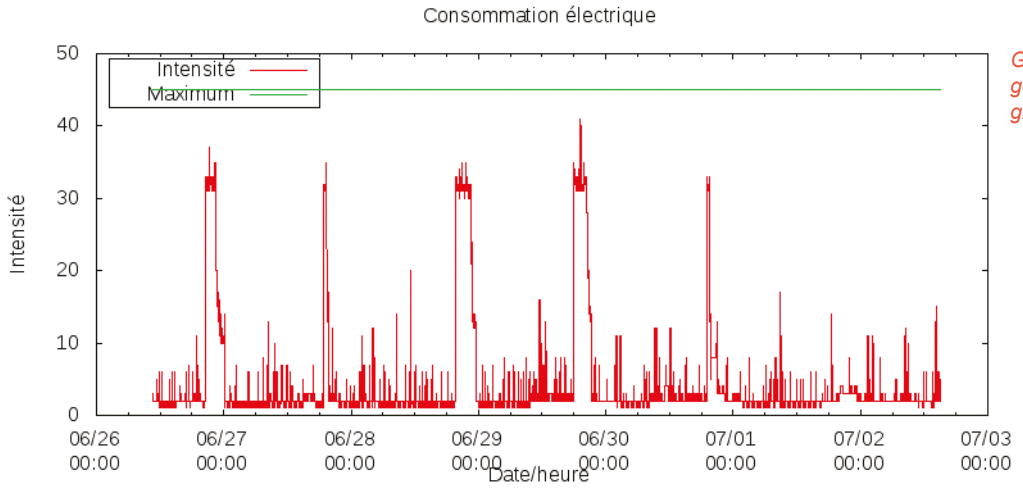
Il n'y a alors plus qu'à transférer le fichier **intensite.data** sur le poste de travail en question et à exécuter **gnuplot** :

```
$ gnuplot intensite.plot
```

... et l'image **intensite.png** (ci-contre) sera générée.

Bien sûr, ceci n'est qu'une approche très simplifiée et limitée de l'exploitation de ces données : idéalement, on aurait un ensemble de fonctions bien plus complexe, qui pourrait générer des graphiques à la volée, basés sur les différents indicateurs remontés, notamment les index de consommation (**BASE**, **HCHC**, **HCHP** ou d'autres, selon l'abonnement en cours). L'utilisation que l'on en fera dépend simplement de notre objectif...

coromonadalix



Graphique généré par gnuplot.

3.1 Daemon

Dans la mesure où ce programme tourne en continu, il peut être intéressant d'en faire un *daemon* : un processus qui se détache du terminal courant et qui tourne en tâche de fond. Pour cela, de nombreux bouts de code et modules Python existent sur Internet. On choisira ici d'aller au plus simple, en ajoutant quelques lignes (très simples et sans gestion d'erreur) après le test portant sur `__name__` et avant l'initialisation de l'interface série :

```
[...]
if __name__ == '__main__':
  import os
  pid = os.fork()
  if pid == 0:
    pid = os.fork()
    if pid != 0:
      os._exit(0)
  else:
    os._exit(0)
  os.close(0)
  os.close(1)
  os.close(2)
  s = serial.Serial(port='/dev/ttyAMA0', [...]
```

À partir de ce moment-là, le programme rendra la main et tournera en tâche de fond dès qu'on l'exécutera.

4. OUTILS EXISTANTS

Ce n'est bien sûr pas la première fois que quelqu'un lit les données de téléinformation sur son ordinateur ! Différents logiciels ont déjà été faits pour cela, plus ou moins aboutis, plus ou moins flexibles. Avant de réinventer la roue, peut-être voudrez-vous vous intéresser à ces outils...

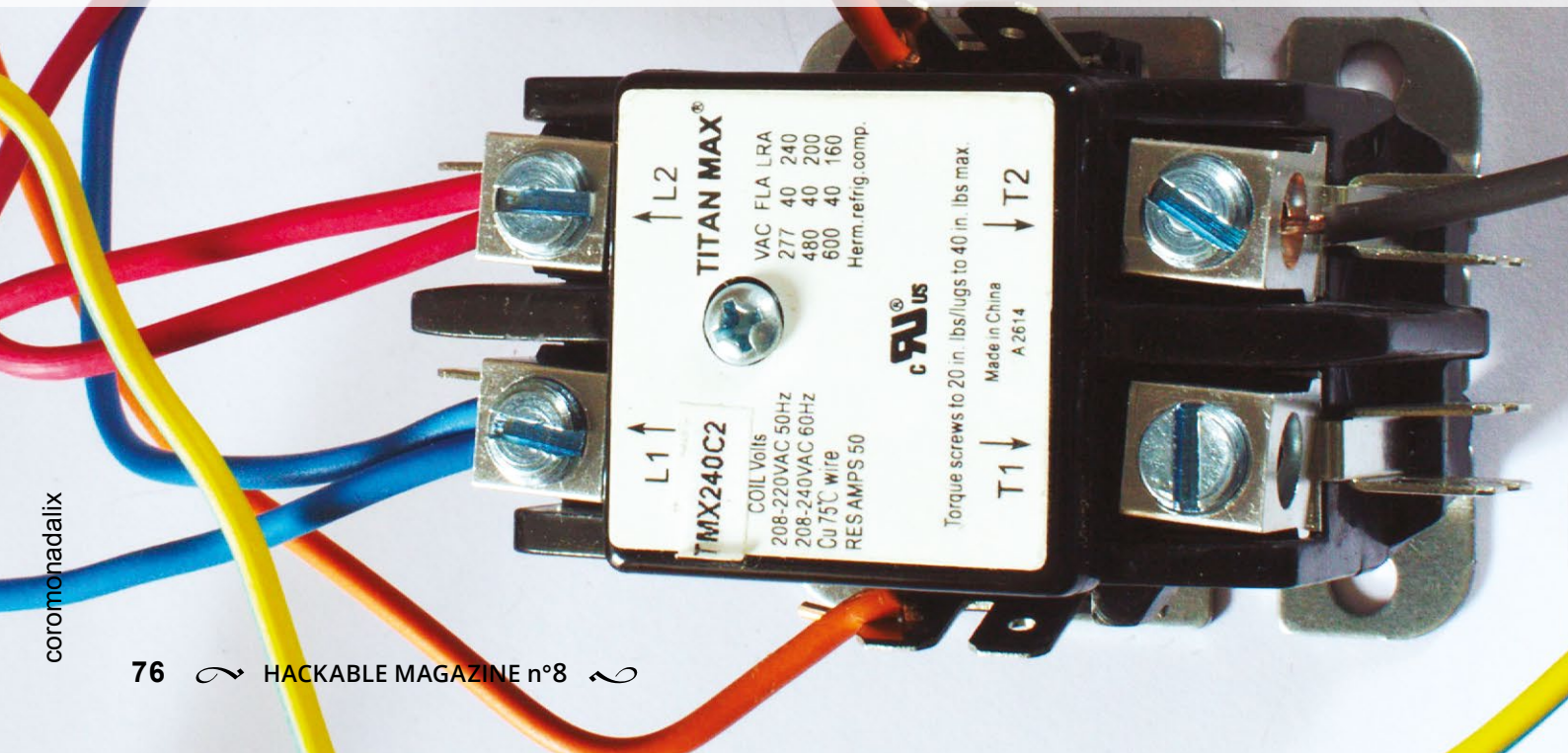
On pourra remarquer en particulier *teleinfuse*, qui permet d'accéder à ces données sous forme de pseudo-fichiers : **https://github.com/neomilium/teleinfuse**.

Le site personnel suivant est particulièrement fourni à ce sujet : **http://vesta.homelinux.free.fr/wiki/demodulateur_teleinformation_edf.html**. SMM

CHARGER UNE VOITURE ÉLECTRIQUE AVEC DU MATÉRIEL LIBRE

Sébastien Maccagnoni-Munch

Différentes méthodes existent pour charger une voiture électrique : cela va de la simple prise, jusqu'aux « wallbox » très onéreuses de certains constructeurs de matériel électrique... mais on peut aussi faire cela avec un matériel libre : OpenEVSE.



coromonadalix

Le « flexichargeur » de Renault, permettant de charger jusqu'à 14A.

Bien sûr, le sujet de la recharge d'une voiture électrique ne va pas directement concerner grand monde. Mais ces véhicules vont se faire de plus en plus présents et en connaître quelques rouages peut être intéressant, ne serait-ce que par curiosité intellectuelle !

Par ailleurs, attention, on va travailler sur du 230V : c'est dangereux, ça peut être mortel, vous êtes des grandes filles et des grands garçons, nous ne pourrons être tenus pour responsables en cas de décès du chat, incendie de la cuisine, tremblement de terre ou explosion nucléaire suite à des manipulations avec ce type de matériel !

1. LA RECHARGE D'UNE VOITURE ÉLECTRIQUE

Une voiture électrique est alimentée par une très grosse batterie électrique (de nos jours, il s'agit surtout de batteries lithium-ion, comme dans nos téléphones portables par exemple), qui peut contenir plusieurs dizaines de kilowatts-heures. Nous n'allons pas lister tous les modèles disponibles sur le marché, notons simplement que la capacité de la batterie d'une voiture électrique peut aller de 8 kWh à 85 kWh, selon les modèles. Renault ZOE étant la voiture électrique la plus vendue en France, nous allons baser nos calculs ci-dessous sur sa batterie de 22 kWh, qui offre 100 à 150 km d'autonomie.

Ajoutons à cela que le rendement d'une recharge de voiture n'est pas parfait : le moteur fonctionne en courant continu, les voitures électriques fonctionnent généralement sous 400V. Il faut donc un transformateur pour que notre « 230V AC » devienne du « 400V DC », cela ne se fait pas sans perte. Il est difficile de donner un rendement moyen global, il se dit que ça peut aller de 50 % (donc 50 % de perte) avec une intensité de 8A à 90 % (donc 10 % de perte) avec une intensité de 32A. Disons que l'on consomme environ 28 kWh « à la prise ». Enfin, comme pour toute recharge de batterie, celle-ci ralentit sur la fin : il est coutume de dire que les 80 premiers pour cent prennent autant de temps à charger que les 20 derniers.

Une prise électrique classique est donnée pour une puissance maximale de 16A, mais elle n'est calibrée pour soutenir que 10A pendant de longues durées : à 230V, cela correspond à 2,3 kW : la charge de 0 à 100 % prend alors en théorie plus de 12 heures (en pratique, la charge sur une telle prise se fait à 8A par sécurité, cela peut alors dépasser 15h). Pour charger plus vite, il faut des dispositifs spécifiques : cela va des prises classiques renforcées (permettant une charge à 14A – 3,2 kW – pour environ 9 heures en théorie, 10 à 11 heures en pratique) aux bornes dites « rapides » qui permettent de gagner 80 % en une demi-heure.

Le standard international qui régit les recharges de voitures électriques (IEC 62196) définit quatre « modes » de charge :

- le mode 1 correspond au branchement direct de la voiture sur une prise de courant, sans contrôle spécifique ; dans la plupart des pays, cette connexion est interdite ;
- le mode 2 (appelé « charge lente ») correspond à l'utilisation d'un câble qui comprend un boîtier de contrôle, comme le « flexichargeur » de Renault ;
- le mode 3 (« charge accélérée ») correspond à l'utilisation d'une borne de charge (wallbox) qui gère la communication avec la voiture, on peut alors monter à des puissances de charge importantes, en courant alternatif ;
- le mode 4 (« charge rapide ») correspond à l'utilisation d'une station de charge (comme dans les parkings des supermarchés), qui délivre directement du courant continu à la batterie, le transformateur est alors dans la station.

Dans le cas de la ZOE, on peut faire une « charge rapide » en courant alternatif (jusqu'à 43 kW pour la ZOE) : elle n'accepte pas par contre de courant continu, *exit* le mode 4 pour la petite Française.

coromonadalix

	Mode 1	Mode 2	Mode 3	Mode 3	Mode 4
Principe	Branchement direct	Câble avec boîtier	Borne privée	Borne publique	Borne publique
Prise	Prise classique	Prise classique, éventuellement renforcée	Prise spécifique ou câble intégré à la borne	Câble intégré à la borne	Câble intégré à la borne
Puissance habituellement rencontrée	1,8 kW	1,8 kW à 3,2 kW	3,7 kW à 22 kW	22 kW à 43 kW	50 kW
Temps de charge	Plus de 12 heures	8 à 12 heures	2 à 8 heures	1 à 2 heures	1 à 2 heures

Le tableau ci-dessus permet de résumer les différents moyens de charger une voiture électrique...

Enfin, on ne peut pas clore ce chapitre sans mentionner les « superchargers » déployés dans le monde par Tesla, utilisables uniquement par les Tesla Model S, dont la puissance peut dépasser 120 kW et qui permettent de gagner 400 km d'autonomie en moins d'une heure...

1.1 Les wallbox

On s'intéresse ici aux boîtiers de charge muraux, permettant une recharge en mode 3, fabriqués par des sociétés très connues dans le monde de l'électricité : Schneider, Hager, Legrand... Il y a aussi de nombreux fabricants plus petits, parfois spécialisés uniquement dans la construction de ces bornes.

Ces boîtiers, qui doivent théoriquement être installés par des électriciens qualifiés, coûtent entre 600 et 2000 € selon leurs caractéristiques : puissance allant de 3,7 kW monophasé à 22 kW triphasé, fonctionnement intérieur ou extérieur, communicants ou non, etc. Par ailleurs, la plupart de ces appareils sont fournis sans câble intégré, mais avec une prise spécifique : il faut alors acheter le câble adapté s'il n'est pas livré avec la voiture.

Une wallbox fabriquée par Hager.

Je ne sais pas ce que vous en pensez, mais moi ça m'a tout de suite titillé, ce sujet... et si je pouvais fabriquer ma wallbox, que je pourrais contrôler à partir de mon ordinateur, faite à ma façon, que je maîtriserais de A à Z ? C'est en partant de cette réflexion que je suis rapidement tombé sur le projet OpenEVSE.

1.2 Le protocole de communication

La communication entre la borne et la voiture est en réalité réduite à sa plus simple expression : un « signal pilote ». Ce signal est défini par le protocole américain SAE J1772, il a été ajouté aux normes IEC 61851 et IEC 62196.

Il s'agit d'un signal carré de 1 kHz, oscillant entre 12v et -12v. Il est généré par la borne de charge et transmis par le biais du fil « pilot » qui va jusqu'à la voiture.

Lorsque la voiture est branchée, une résistance est placée entre ce fil et la terre. C'est la valeur de cette résistance qui permet à la voiture de « communiquer » avec la borne de charge ; la borne de charge peut alors « surveiller » la différence de tension entre le fil pilote et la terre afin de « recevoir » ce message et de se mettre dans un état particulier :

- absence de retour : aucune voiture n'est branchée, état 1 ;
- 2,7 kOhms (9v) : la voiture est branchée, état 2 ;
- 880 Ohms (6v) : la voiture est en charge, état 3, le courant est envoyé à la voiture ;
- 240 Ohms (3v) : une ventilation est nécessaire avant de pouvoir continuer, la charge s'arrête, état 4.

La borne de charge, quant à elle, peut communiquer à la voiture la puissance maximale que cette dernière peut demander. Cela se fait en modulant le rapport cyclique (durée à 12v et durée à -12v), autrement dit la borne effectue une modulation de la largeur d'impulsion. Par exemple, un rapport de 10 % correspond à une puissance maximale de 6A, 20 % correspondent à 12A, 50 % à 30A, cela va jusqu'à 96 % pour 80A.

2. OPENEVSE

OpenEVSE (« *Electric Vehicle Supply Equipment* »), c'est un petit circuit intégré permettant de contrôler la charge d'une voiture électrique ; c'est la partie « intelligente » d'une wallbox ; c'est un mignon petit appareil programmable ; c'est surtout du matériel libre (code source sous licence GPL v3, schémas sous licence Creative Commons 3.0 BY-SA).

Le projet OpenEVSE s'appuie sur un ATMEL AVR, processeur 8 bits fonctionnant à 16 MHz, celui-là même utilisé par l'Arduino Uno. Sur celui-ci tourne un micrologiciel spécifique, qui permet de communiquer avec la voiture électrique par le signal pilote susdécrit ; le croquis Arduino pour cette platine est disponible sur Internet, on peut entièrement personnaliser sa borne. Rien d'extrêmement compliqué en réalité, on l'a vu plus haut, mais on a là un pied dans le domaine de l'électronique et un pied dans l'électricité, où se côtoient courant fort et courant faible, avec des puissances qui peuvent monter jusqu'à 22000 watts... D'accord pour fabriquer ma wallbox communicante, mais pas au point de bricoler en ne partant de rien !

2.1 Les éléments complémentaires

Maintenant que les bases sont posées, on peut s'attaquer au montage de l'OpenEVSE, avec lequel on a pour objectif de faire une charge en 32A. Mais il ne se suffit pas à lui-même, ne rêvons pas !

Il nous faut tout d'abord la connectique pour dialoguer avec la voiture : soit une prise dans un format adapté, soit un câble embarqué, au bout duquel on retrouve une fiche dans un format adapté. Les formats les plus souvent rencontrés sont appelés « type 1 », « type 2 » et « type 3 ». Pour ce montage, je choisis un câble équipé d'une fiche type 2, qui est le format utilisé par Renault ZOE ; je prends d'ailleurs un câble prêt à être utilisé en triphasé, même si pour l'instant le montage se fait en monophasé...

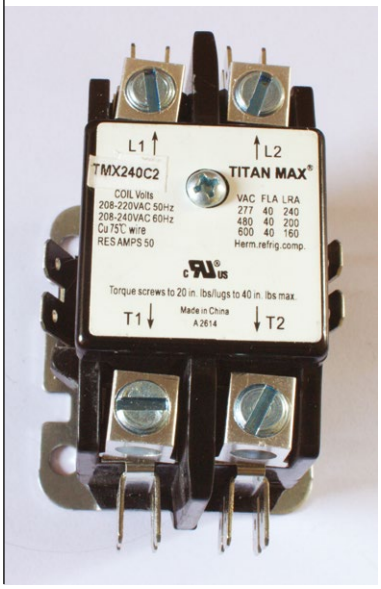
Il faut également un relais ou un contacteur, afin de ne pas laisser passer le courant dans le câble en continu : le circuit électronique n'activera le passage du courant que si une voiture est branchée (état 3). OpenEVSE v3 est capable de piloter deux relais 12VDC/230VAC ou un contacteur 230VAC/230VAC : il est évident que ce n'est pas avec un petit circuit électronique qu'on pourra faire passer de la puissance. Lorsque l'on utilise les relais, OpenEVSE permet d'en mettre deux pour éviter que le courant reste en continu

MATÉRIEL LIBRE

Un petit rappel ne faisant pas de mal, précisons ce concept de matériel libre : c'est du matériel pour lequel les plans de fabrication et les éventuels micrologiciels (firmwares) sont disponibles, permettant à tous de reconstruire le matériel de son côté, à condition bien sûr d'avoir les outils nécessaires à cette opération...

Un OpenEVSE v3.

Un contacteur 240V 40A.



coromonadalix

dans le câble dans le cas d'un « collage » du relais. Je souhaite effectuer une charge jusqu'à 32A, il est alors obligatoire d'utiliser un contacteur, car les relais que l'on peut trouver dans le commerce n'acceptent pas plus de 20A.

Un OpenEVSE et ces deux éléments permettent déjà de construire une borne de recharge, mais on peut également ajouter d'autres éléments :

- OpenEVSE accepte un écran LCD afin d'afficher les informations sur la charge en cours ;
- lorsque l'on met un écran LCD, on ajoute aussi généralement un bouton poussoir, pour pouvoir contrôler et configurer l'appareil ;
- pour sécuriser l'ensemble, il est fortement souhaitable de mettre en place un disjoncteur et un interrupteur différentiel adapté, c'est la base de la sécurité dans l'électricité domestique.

Pour le contrôle de l'OpenEVSE, notre objectif est d'ajouter un Raspberry Pi : c'est lui qui va nous permettre de rendre cette borne de charge pseudo-intelligente et communicante. C'est pourquoi on n'utilise pas d'écran LCD connecté à l'OpenEVSE, ni de bouton poussoir.

En ce qui concerne la sécurité, on choisit d'être très prudent en connectant ce montage sur un disjoncteur de 32A et un interrupteur différentiel Hpi 30mA 40A. À terme, on pourra y ajouter un déclencheur électronique, que la borne pourra activer à distance pour « tout couper » si elle détecte un problème.

Pour le câblage électrique, étant donnée la forte puissance demandée, il faut au minimum des fils ayant une section de 6mm² : phase, neutre, terre, comme pour n'importe quel circuit électrique.

Nous avons aussi commandé trois autres éléments sur la boutique d'OpenEVSE, pour une utilisation ultérieure :

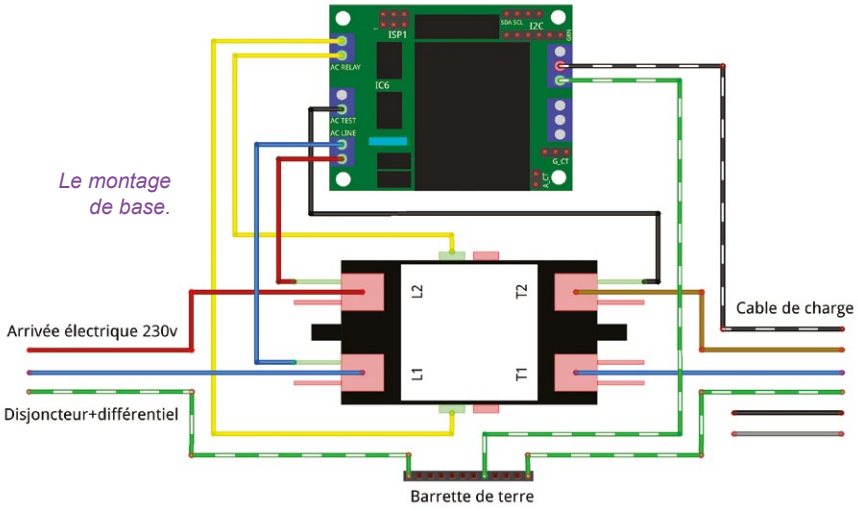
- le programmateur, permettant de charger le micrologiciel, pour mises à jour ou pour y apporter des modifications (compatible Arduino, il sera utilisable dans d'autres contextes) – il se connecte au port USB d'un PC ;
- le kit de mesure de courant, qui nous permettra de savoir en temps réel combien la voiture consomme ;
- un presse-étoupe, dont le but est d'étanchéifier le passage de câble : en effet, l'originalité de ce montage particulier passera également par le fait que le boîtier sera à l'intérieur (donc pas nécessairement étanche), mais avec un câble traversant le mur de la maison.

Enfin, on notera que l'OpenEVSE peut tout à fait piloter un contacteur triphasé, pour offrir une recharge encore plus rapide ; mais attention, seules les voitures acceptant du triphasé en profiteront...

2.2 Connexions minimales

Pour obtenir une borne de charge fonctionnelle, les connexions nécessaires sont quasi exclusivement sur la partie 230v :

- l'arrivée électrique, directement du tableau et protégée par un disjoncteur et un différentiel adaptés, alimente l'OpenEVSE (sur les bornes « AC LINE ») et le contacteur ;
- l'OpenEVSE contrôle la bobine du contacteur par ses bornes « AC RELAY » ;



Le montage de base.

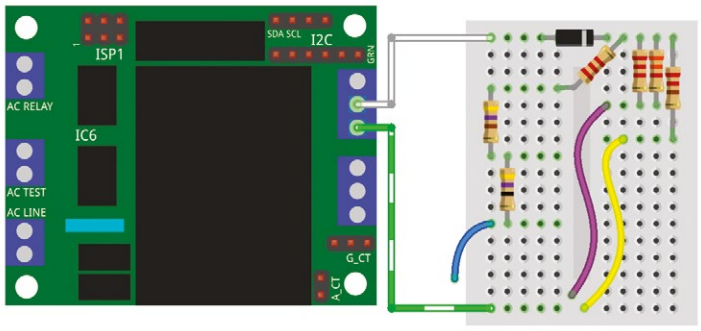
- la sortie « phase » du contacteur revient sur une entrée « AC TEST » de l'OpenEVSE ;
- la première phase (marron) et le neutre du câble de charge sont connectés sur le contacteur – les deux autres phases (noir et gris) ne sont pas connectées, on les garde en attente dans l'éventualité de passer le montage en triphasé ;
- le fil pilote du câble de charge (noir/blanc) est connecté à la borne « pilot » de l'OpenEVSE – c'est la seule connexion qui n'est pas en 230v ;
- enfin, les terres sont interconnectées : celle de l'alimentation électrique, celle du câble de charge et celle de l'OpenEVSE.

Concernant les sections de câbles, il est important de faire un dimensionnement adapté :

- de l'alimentation (tableau électrique) au relais, il est nécessaire d'utiliser des fils d'une section d'au moins 6mm² pour « tenir » 32A ;
- pour alimenter l'OpenEVSE et pour contrôler le relais, une section faible est suffisante (par exemple 0,75mm²) : le besoin de puissance est minime, 100mA maximum.

2.3 Tester l'OpenEVSE

Pour tester l'OpenEVSE, avant même d'envisager un quelconque moyen d'avoir un affichage ou d'entrer des commandes, on peut simuler la présence d'une voiture. Pour cela, il suffit de mettre en série une diode (une classique 1N4148) et un



Simulation d'une voiture branchée.

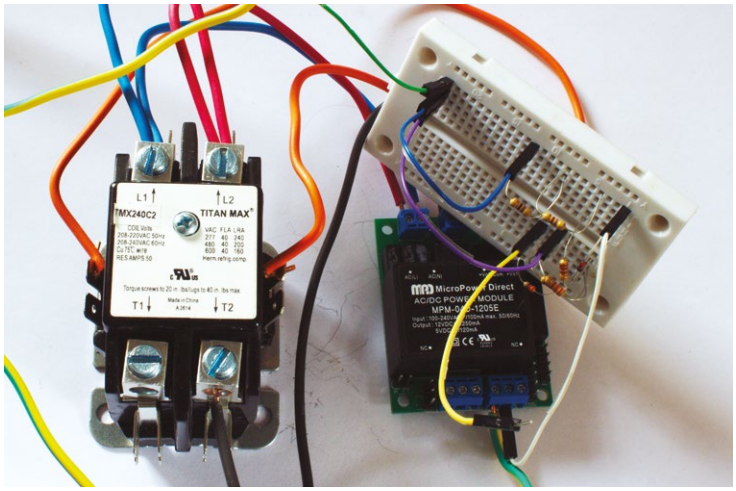
ensemble de résistances, dont nous changerons la valeur conformément aux besoins du signal pilote tel qu'il a été décrit plus haut.

On a donc besoin de différentes résistances. N'ayant pas les valeurs exactes à disposition, je vais « jouer » avec les résistances standards que l'on trouve dans les kits d'électronique afin d'obtenir des approximations satisfaisantes :

- voiture branchée, entre le fil blanc et le fil bleu 2700 Ohms : 2200 + 470 + 47 = 2717 Ohms ;
- voiture en charge, entre le fil blanc et le fil violet, 880 Ohms : ajout de 2200 et 3300 en parallèle = 1320 Ohms, pour obtenir 888,4 Ohms ;
- ventilation, entre le fil blanc et le fil jaune, 240 Ohms : ajout de 330 Ohms en parallèle aux deux précédents, pour obtenir 240,6 Ohms.

Par conséquent et selon le schéma ci-contre, la simulation pourra se faire en branchant sur la dernière ligne de la platine d'essai (connectée à la terre par un fil vert) un ou plusieurs de ces fils :

- voiture branchée : le fil bleu ;
- voiture en charge : les fils bleu et violet ;
- ventilation : les fils bleu, violet et jaune.



Simulation d'une voiture en charge.

En pratique, voilà ce qu'il se passe :

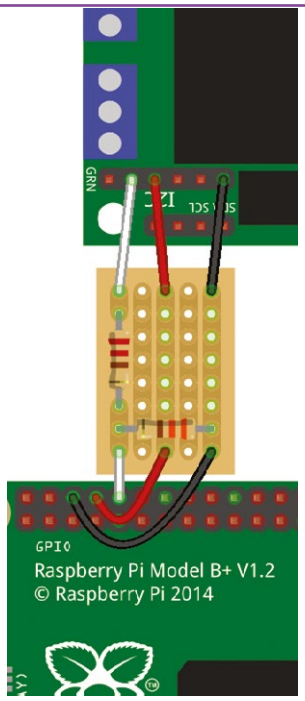
- on branche le fil bleu, on n'entend rien ;
- on ajoute le fil violet, on entend le contacteur se fermer ;
- on ajoute le fil jaune, on entend le contacteur s'ouvrir.

2.4 Communiquer avec OpenEVSE

Maintenant que l'on a validé (à l'oreille) le fonctionnement de la borne de charge, il est temps d'essayer de communiquer avec elle. Depuis la version 3 de son firmware, sortie en 2014, OpenEVSE offre une API à laquelle on peut accéder au travers du connecteur FTDI, appelée *RAPI*.

On va alors brancher ce connecteur à un Raspberry Pi, sur son port série (UART, GPIO 14 et 15). Mais attention ! Comme tout Arduino, l'OpenEVSE communique sous 5 volts, alors que les GPIO du Raspberry Pi attendent du 3,3v. Le montage idéal se baserait sur un convertisseur de niveau logique (*level shifter*), mais on va ici se contenter d'un diviseur de tension, utilisant deux résistances ; celui-ci réduit la tension de 5v à environ 3v, ce qui est conforme aux spécifications (sur un circuit en 3,3v, le niveau haut commence généralement vers 2 ou 2,3v) :

- fils noirs : les masses sont interconnectées ;
- fils rouges : la sortie 3,3v du Raspberry Pi est envoyée directement en entrée de l'OpenEVSE (cela rentre dans les spécifications, la bascule entre niveau haut et niveau bas se fait généralement aux alentours de 1,5v) ;
- fils blancs : la sortie 5v de l'OpenEVSE est connectée sur un diviseur de tension (OpenEVSE en *Vin*, Raspberry Pi en *Vout*) composé de deux résistances : une résistance de 220 Ohm entre Vin et Vout et une résistance de 330 Ohm entre Vout et la masse.



Connexion du port FTDI de l'OpenEVSE sur les broches UART du Raspberry Pi.

Il faut également penser à désactiver la console sur le port série du Raspberry Pi, en commentant la ligne idoine du fichier **/etc/inittab** puis en redémarrant :

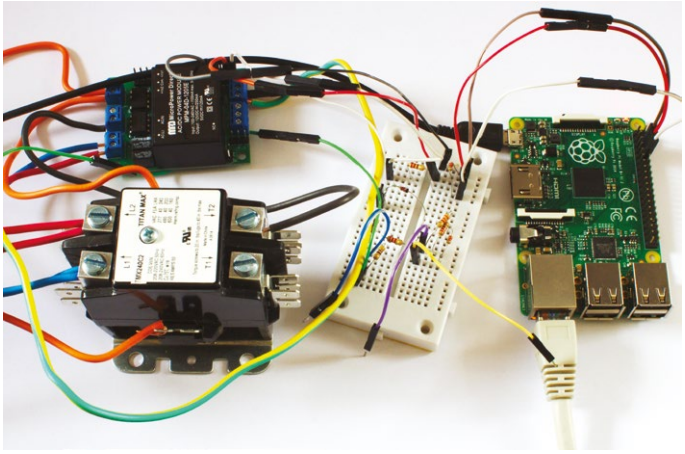
```
#T0:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100
```

On lance alors un terminal sur ce port série :

```
$ miniterm.py /dev/ttyAMA0 115200
```

Enfin, on démarre l'OpenEVSE ; s'affiche alors la chaîne **$ST 01**, indiquant que la borne passe dans l'état d'attente d'un véhicule (état 1).

```
$ miniterm.py /dev/ttyAMA0 115200
--- Miniterm on /dev/ttyAMA0: 115200,8,N,1 ---
--- Quit: Ctrl+]  |  Menu: Ctrl+T | Help: Ctrl+T
followed by Ctrl+H ---
$ST 01
```



L'ensemble OpenEVSE + simulation de la voiture + Raspberry Pi.

coromonadalix

Lorsque l'on branche une voiture, la borne passe à l'état 2, on voit alors la chaîne `$ST 02` apparaître et ainsi de suite quand la charge démarre, etc.

On peut alors dialoguer avec la platine OpenEVSE avec un ensemble de commandes, soit pour obtenir des informations soit pour configurer le comportement de la borne. Tout ceci est documenté dans le fichier `rapi_proc.h` du croquis.

Voici par exemple une suite d'opérations effectuées avec ce protocole :

```
$SE 1
$OK
$GS
$OK 1 8615
$GE
$OK 32 0021
$SC 16
$OK
$GE
$OK 16 0021
```

Ici, on a effectué les opérations suivantes :

- activation du retour (*Echo*) des commandes tapées (par défaut, ce que l'on tape n'est pas affiché) ;
- demande de l'état (*Status*) de la borne : elle est dans l'état 1 (aucune voiture branchée) et la dernière charge a duré 8615 secondes (2h23) ;
- demande des paramètres (*sEttings*) : la charge se fera à 32 ampères et les drapeaux (*flags*) de configuration sont à « 0021 » ;
- configuration de l'intensité (*current Capacity*) à 16 ampères ;
- demande des paramètres (*sEttings*) : la charge se fera à 16 ampères et les drapeaux (*flags*) de configuration sont à « 0021 ».

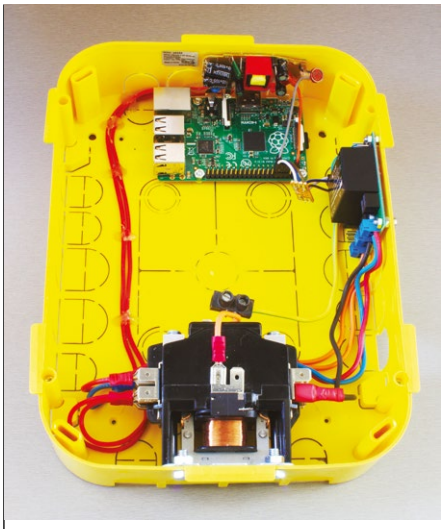
3. MONTAGE DÉFINITIF

Tous ces éléments peuvent maintenant être insérés dans un boîtier adapté, connectés au réseau électrique et au câble de charge !

En réalité, cette borne de charge sera plus complexe que cela : être maître de son matériel, cela implique que l'on peut ajouter toutes les fonctionnalités que l'on veut. Ici, ce boîtier contient en réalité l'OpenEVSE, le Raspberry Pi et son alimentation, un ampèremètre (bobine), une carte « mains libres » de la voiture (pour déverrouiller automatiquement la trappe de charge), un écran tactile... et tout cela est bien sûr contrôlé par un programme sur mesure, présent dans le Raspberry Pi et capable de communiquer avec d'autres ordinateurs grâce à un câble réseau ! **SMM**

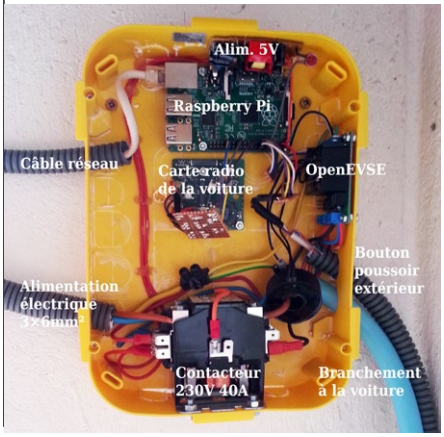
RÉFÉRENCES

- Site officiel du projet OpenEVSE : **https://code.google.com/p/open-evse/**
- Source (croquis) : **https://github.com/lincomatic/open_evse**



Le boîtier du chargeur, tel que décrit dans cet article et prêt à être posé !

Le boîtier du chargeur en place, incluant les éléments décris dans le dernier paragraphe et avant branchement de l'écran tactile, intégré à la façade.



coromonadalix

EXPLOITEZ UN PETIT ÉCRAN TACTILE POUR RASPBERRY PI AVEC PYGAME

Sébastien Maccagnoni-Munch

Un ordinateur de petite taille comme un Raspberry Pi, c'est très sympathique pour nos montages complexes. Mais s'il faut lui coller un écran 17 pouces et un clavier, ou au contraire tout faire par le réseau, c'est moins pratique. Parfois on a simplement besoin d'un petit écran pour afficher quelques informations et poser une question ou deux à l'utilisateur... Pour cela, un petit écran tactile de quelques centimètres est parfaitement adapté !

coromonadalix

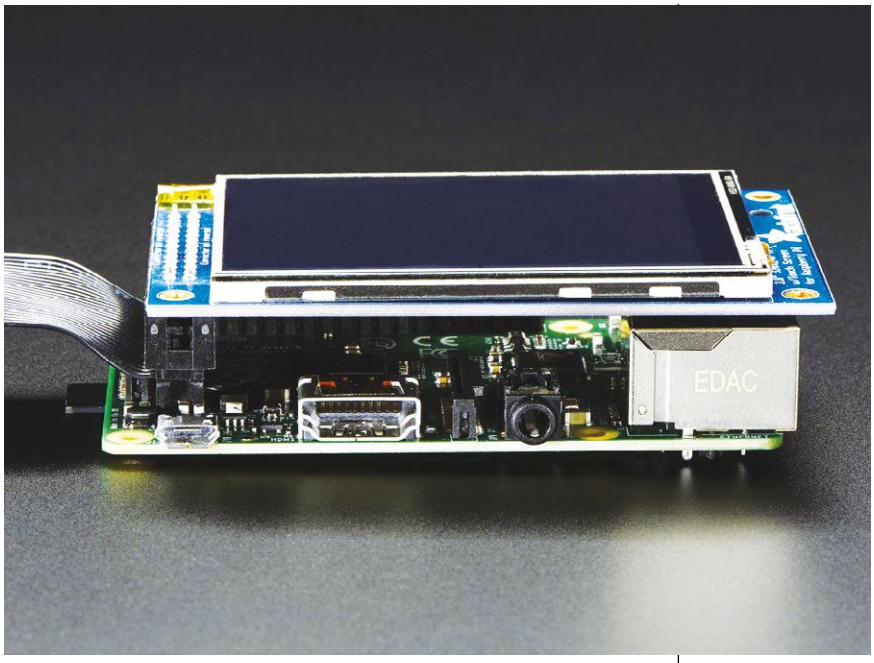
1. LE MATÉRIEL

On s'intéressera ici à un écran proposé par Adafruit. Cette sympathique société propose une série d'écrans, de diagonale allant de 2,8 à 3,5 pouces avec une définition de 320×240 ou 480×320 pixels, associés à une dalle tactile résistive (détection quand on appuie sur l'écran) ou capacitive (détection quand on effleure l'écran), qu'ils ont appelés PiTFT. Il y a également un PiTFT de 2,2", mais celui-ci n'est pas tactile.

Nous nous intéresserons ici à la version la plus courante, d'une diagonale de 2,8", une résolution de 320×240 pixels, tactile résistif, qui coûte une trentaine d'euros. Bonne nouvelle pour ceux qui n'aiment pas trop souder : cet écran vient préassemblé !

Comme son nom l'indique, cet écran est optimisé pour une utilisation avec un Raspberry Pi. Il se branche directement sur les pins GPIO, qui sont intégralement déportés sur sa carte (voir la première photographie), à partir de laquelle on peut utiliser une nappe classique pour accéder aux différents ports. Notons toutefois que, lorsque l'on utilise un Raspberry Pi A+, B+ ou 2, la carte comportant l'écran est décalée de quelques millimètres, la faute aux connecteurs GPIO qui ont été déplacés pour laisser la place à un trou de fixation.

Enfin, on remarquera que sous l'écran se trouvent quatre emplacements pour des boutons poussoirs, associés aux GPIO 18, 22, 23 et 21 (pour la révision 1) ou 27 (pour la révision 2). En l'occurrence, nous ne les utiliserons pas.



Un PiTFT 2,8" sur un Raspberry Pi B+ avec une nappe branchée (crédit : Adafruit).

Il existe également le PiTFT 2,8" Plus, plus adapté aux dimensions des nouvelles versions de la carte, au même prix...

On a bien sûr également besoin d'un Raspberry Pi (n'importe quel modèle devrait fonctionner), d'une carte mémoire SD ou micro-SD pour stocker le système d'exploitation, d'une alimentation adaptée au Raspberry Pi (l'écran consommant environ 100mA au maximum, il n'y a pas de besoin excessif) et d'un accès au réseau : bien sûr l'accès au réseau n'est pas nécessaire pour faire fonctionner l'écran, mais cela simplifiera grandement l'installation de ses pilotes !

Avant de commencer, les lecteurs assidus de *Hackable* se diront certainement que tout ça leur rappelle quelque chose... En effet dans le numéro 3 (novembre-décembre 2014) nous avions découvert un petit écran LCD à connecter à un Raspberry Pi : les écrans PiTFT sont très proches de ce matériel et, en réalité, le pilote que nous allons utiliser est `fbtft` (plus particulièrement le module `fb_ili9340`), que nous avions découvert pour cet autre écran. C'est pourquoi nous n'allons pas nous attarder sur les spécificités du pilote de l'écran, mais nous allons approfondir son utilisation avec une bibliothèque logicielle adaptée.

2. RENDRE CET ÉCRAN FONCTIONNEL

Ce qui est agréable avec Adafruit quand on est pressé de voir son jouet fonctionner, c'est que les instructions sont simplifiées au maximum. En effet, deux solutions s'offrent à nous pour exploiter ce matériel rapidement : une image de Raspbian préconfigurée est fournie par Adafruit, qu'il suffit alors de placer sur la carte mémoire avant de l'utiliser pour démarrer la platine. Allons tout de même un peu plus loin : on va utiliser une image officielle de Raspbian, sur laquelle on installera le pilote avec la procédure adaptée (simplifiée, elle aussi)...

Allons-y, Raspbian Wheezy est sur une carte SD, celle-ci est insérée dans le Raspberry Pi, l'écran est connecté aux broches GPIO, on alimente l'ensemble pour démarrer le système... l'écran s'allume alors, tout de blanc vêtu : sans pilote, il ne sait en effet pas quoi afficher, on voit toutefois qu'il est correctement alimenté, ce qui est bon signe.

Deux commandes suffisent ensuite pour mettre en place les pilotes :

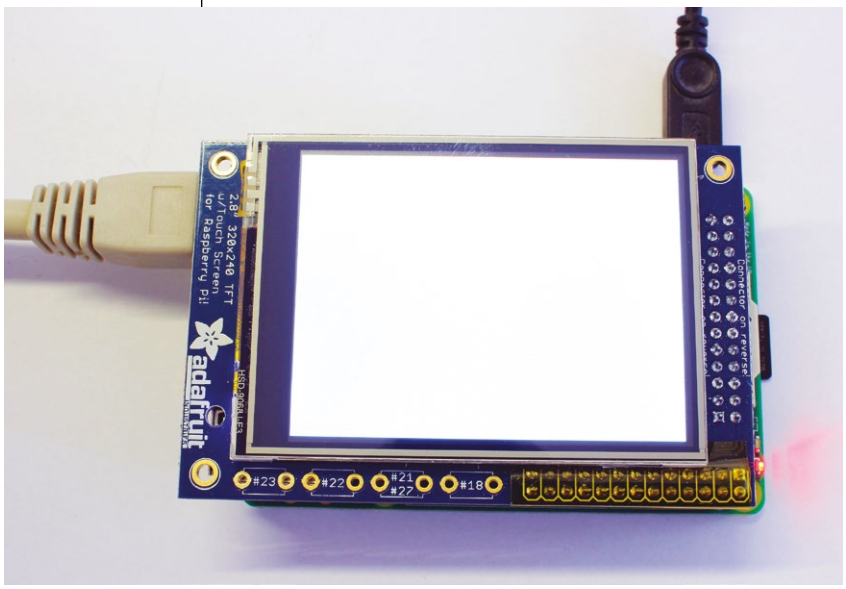
```
$ curl -SLs https://apt.adafruit.com/add | sudo bash
$ sudo apt-get install -y adafruit-pitft-helper
```

La seconde commande peut prendre beaucoup de temps et s'arrêter de longues minutes à la ligne « `Unpacking replacement raspberrypi-bootloader ...` » : c'est normal !

On active ensuite le support de ce modèle d'écran avec la commande suivante :

```
$ sudo adafruit-pitft-helper -t 28r
```

... cette commande met en place le support de l'écran 2,8" avec dalle tactile résistive, à vous de changer l'argument `-t` si vous avez acheté un autre modèle d'écran PiTFT...



Premier démarrage : l'écran n'affiche rien, il n'est pas piloté.

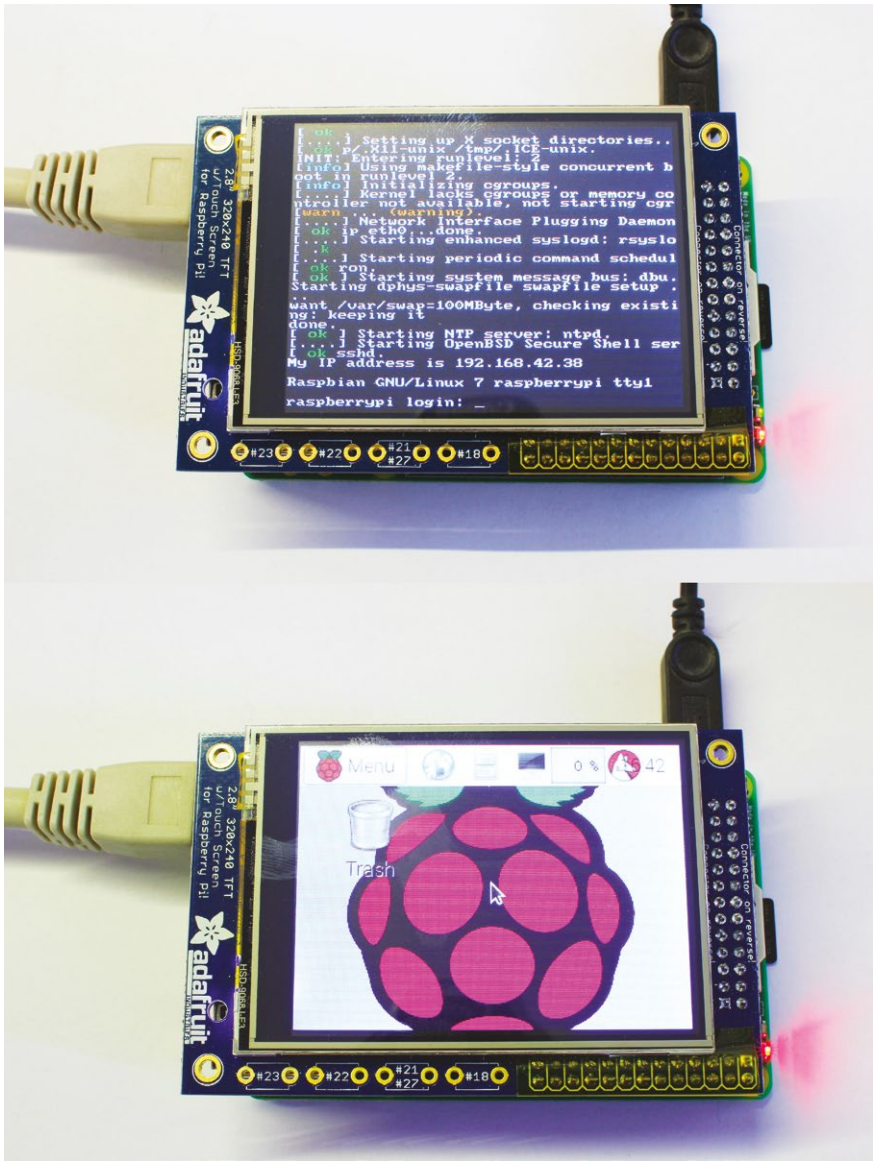
Deux questions sont alors posées :

- *Would you like the console to appear on the PiTFT display? [y/n]* : répondre « y » résultera en l'affichage de la console textuelle habituelle sur cet écran au lieu de la sortie HDMI ;
- *Would you like GPIO #23 to act as a on/off button? [y/n]* : répondre « y » fera agir le premier bouton sous l'écran comme un bouton d'allumage ou d'extinction du Raspberry Pi.

Et voilà, c'est prêt, il ne reste plus qu'à redémarrer !

```
$ sudo reboot
```

coromonadalix



En haut : une fois activé, l'écran affiche la console habituelle. En bas : avec startx, on accède à l'interface graphique... On voit des traces de doigt sur l'écran, c'est bien un écran tactile (si si, en haut à gauche, regardez bien) ! ;-)

Enfin, on peut tout à fait démarrer le serveur X pour afficher une interface graphique habituelle :

```
$ FRAMEBUFFER=/dev/fb1 startx
```

L'écran tactile est d'ores et déjà fonctionnel !

Mais on ne s'intéresse pas à l'utilisation de cet écran comme d'un écran principal, on ne veut pas y afficher la console ou le serveur X : cet écran sera géré de manière complètement indépendante, comme une interface homme/machine secondaire. On va alors relancer la commande `sudo adafruit-pitft-helper -t 28r` pour répondre « n » aux deux questions : au prochain redémarrage, l'écran reste alors complètement noir, mais il est toujours disponible comme périphérique de type framebuffer sur `/dev/fb1`.

QUE FAIT ADAFRUIT-PITFT-HELPER ?

La commande `adafruit-pitft-helper` met en place différents éléments permettant d'utiliser cet écran. Parmi les éléments concernés, on retrouve :

- **les modules du noyau à charger et leurs options ;**
- **quelques lignes de configuration dans le fichier `/boot/config.txt` pour le pilote de l'écran en lui-même, avec des lignes `dtparam` et `dtoverlay`, pour les paramètres du pilote ;**
- **les arguments `fbcon` dans le fichier `/boot/cmdline.txt`, ceux-là mêmes qu'on a utilisés dans le numéro 3 de *Hackable* ;**
- **la configuration du serveur X (dans le répertoire `/etc/X11/`) ;**
- **le calibrage par défaut de l'écran tactile (dans `/etc/pointercal`).**

coromonadalix

3. AFFICHAGE ET RETOUR UTILISATEUR AVEC PYGAME

Pygame est une bibliothèque de développement d'interfaces en Python, orientée vers les jeux, mais pouvant parfaitement être utilisée pour n'importe quel autre usage. Cette bibliothèque est proposée dans le paquet **python-pygame** ; il est normalement préinstallé dans l'image officielle de Raspbian, cependant si ce n'est pas le cas sur votre système, la commande pour l'installer est la suivante :

```
$ sudo apt-get install python-pygame
```

Les six images que l'on a créées...

L'objectif que l'on se donne ici est le suivant : diviser l'écran en quatre zones, quatre boutons sur lesquels l'utilisateur pourra appuyer pour déclencher des actions. Pour ces boutons, on crée plusieurs images :

- quatre images pour les icônes à afficher ;
- une image majoritairement transparente (dégradé sombre aux bords bas et droit) pour un effet « bouton » ;
- une seconde image (dégradé inversé) pour un effet « bouton enfoncé ».

Par ailleurs, afin de nous affranchir des problèmes de droits, nous lancerons exceptionnellement notre script Python en tant que root, avec la commande **sudo** si on l'exécute manuellement.

Le programme que nous allons mettre en œuvre est assez facilement compréhensible pour quelqu'un qui a déjà écrit du Python...

```
#!/usr/bin/env python
# -*- coding: utf8 -*-

import os, pygame

os.putenv('SDL_VIDEODRIVER', 'fbcon') # Affichage Framebuffer
os.putenv('SDL_FBDEV', '/dev/fb1')    # Périphérique de PiTFT
os.putenv('SDL_MOUSEDEV', '/dev/input/touchscreen')
os.putenv('SDL_MOUSEDRV', 'TSLIB')
pygame.init()
screen = pygame.display.set_mode((320, 240)) # Déf de l'écran
pygame.mouse.set_visible(False) # On n'affiche pas le curseur

button_up = pygame.image.load('button_up.png')
button_down = pygame.image.load('button_down.png')
class Button:
  def __init__(self, function, filename, x, y):
    baseimage = pygame.image.load(filename).convert()
    self.upimage = baseimage.copy()
    self.upimage.blit(button_up, (0,0))
```

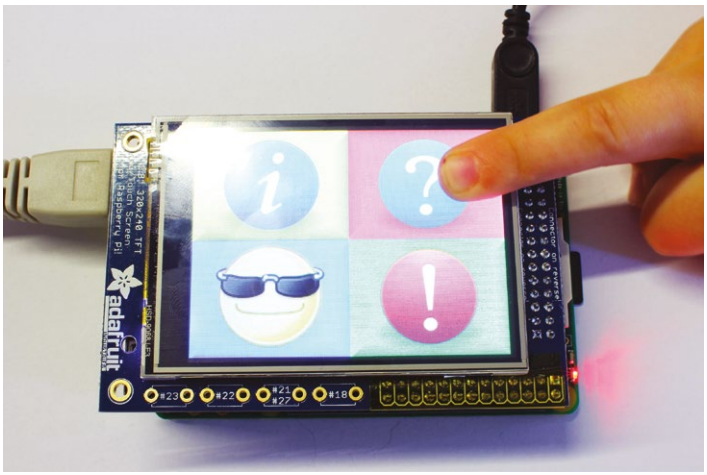
coromonadalix

```
    self.downimage = baseimage.copy()
    self.downimage.blit(button_down, (0,0))
    self.execute = function
    self.position = (x, y)
    self.up()
  def up(self): screen.blit(self.upimage, self.position)
  def down(self): screen.blit(self.downimage, self.position)

def info(): print 'Info'
def question(): print 'Question'
def cool(): print 'Cool'
def warning(): print 'Warning'

buttons = {}
buttons['NW'] = Button(info, 'info.png', 0, 0)
buttons['NE'] = Button(question, 'question.png', 160, 0)
buttons['SW'] = Button(cool, 'cool.png', 0, 120)
buttons['SE'] = Button(warning, 'warning.png', 160, 120)
pygame.display.flip()

is_down = None
while 1:
  for event in pygame.event.get():
    if event.type in (pygame.MOUSEBUTTONDOWN,
                      pygame.MOUSEMOTION, pygame.MOUSEBUTTONUP):
      button = [None, None]
      x, y = event.pos
      button[0] = 'N' if y<120 else 'S'
      button[1] = 'W' if x<160 else 'E'
      button = ''.join(button)

      if is_down: buttons[is_down].up()

      if event.type == pygame.MOUSEBUTTONUP:
        buttons[button].up()
        is_down = None
        buttons[button].execute()
      else:
        buttons[button].down()
        is_down = button

      pygame.display.flip()
  pygame.time.wait(10)
```

Décortiquons-le :

Tout d'abord, on définit les spécificités pour la bibliothèque SDL, utilisée par Pygame pour l'affichage : pilote **`fbcon`**, périphérique **`/dev/fb1`** (car **`/dev/fb0`** est la sortie HDMI), périphérique de l'écran tactile et pilote à utiliser pour l'écran tactile. On initialise alors pygame avec ces paramètres, pour ensuite lui indiquer la résolution de l'écran et lui demander de ne pas afficher le curseur de la souris.

Sont alors précisées les fonctions qui seront appelées par les différents boutons. Pour l'instant ces fonctions ne font que de simples « print », c'est ici que l'on pourra définir les opérations à exécuter quand on appuie sur un bouton !



L'écran tactile, quatre boutons affichés, avec un doigt innocent appuyant sur l'un des boutons : admirons l'effet « bouton », de toute beauté !

Le bloc suivant concerne les boutons : on crée une classe **Button** qui sera utilisée pour chacun des boutons : lors de leur création, l'image fournie sera agrémentée de l'effet « bouton sorti » ou « bouton appuyé », par les images semi-transparentes **button_up.png** et **button_down.png**. Les objets de type **Button** contiendront également les positions des boutons.

On crée ensuite un dictionnaire appelé **buttons**, qui contient ces objets **Button**, associés à une abréviation de leur position « géographique » : *SW* pour *sud-ouest*, par exemple. Une fois les objets créés, la fonction **pygame.display.flip()** permet de mettre à jour l'affichage.

Enfin, la boucle globale attend des événements liés à la souris. Pour chaque événement, le bouton correspondant est trouvé (position « géographique », l'écran étant divisé en quatre). Si un bouton était déjà enfoncé (c'est le cas quand on glisse le doigt sur l'écran), alors celui-ci est « relevé ». Ensuite, l'affichage dépend de l'opération : si on « relâche » alors le bouton sera affiché comme relâché et la fonction correspondante sera exécutée, sinon le bouton est affiché enfoncé et son identifiant « géographique » est stocké dans la variable **is_down**, pour traitement à la prochaine itération de la boucle.

Enfin, l'affichage est mis à jour s'il y a eu un événement provenant de l'écran tactile, puis dans tous les cas on attend 10 millisecondes avant la nouvelle itération.

4. LANCEMENT AUTOMATIQUE

Enfin, on peut vouloir lancer cette interface automatiquement au démarrage de la machine. On créera alors un script de démarrage **/etc/init.d/buttons**, dont le contenu sera assez classique...

```
#!/bin/sh

PATH=/sbin:/bin:/usr/sbin:/usr/bin
CHDIR=/home/pi/iface
DAEMON=$CHDIR/buttons.py
PIDFILE=/var/run/buttons.pid

. /lib/lsb/init-functions

case $1 in
 start)
  log_daemon_msg "Starting buttons" "buttons"
   start-stop-daemon --start --quiet --oknodo
--background --pidfile $PIDFILE --make-pidfile
--chdir $CHDIR --startas $DAEMON
  log_end_msg $?
    ;;
 stop)
  log_daemon_msg "Stopping buttons" "buttons"
    start-stop-daemon --stop --signal KILL
--quiet --oknodo --pidfile $PIDFILE
  log_end_msg $?
  rm -f $PIDFILE
    ;;
 restart|reload)
  $0 stop && sleep 2 && $0 start
    ;;
 *)
  echo "Usage: $0 {start|stop|restart}"
  exit 2
  ;;
esac
```

Enfin, ce script est activé au démarrage :

```
$ sudo update-rc.d buttons defaults
```

5. EXTINCTION AUTOMATIQUE

Le rétroéclairage de cet écran peut être contrôlé grâce à une broche GPIO supplémentaire, desservi par le contrôleur de l'écran tactile. Cette broche est référencée par le noyau sous le numéro 508.

La série de commandes suivantes, exécutée en tant que root, permet d'activer le contrôle du rétroéclairage, de l'éteindre puis de l'allumer :

```
echo 508 > /sys/class/gpio/export
echo out > /sys/class/gpio/gpio508/direction
echo 0 > /sys/class/gpio/gpio508/value
echo 1 > /sys/class/gpio/gpio508/value
```

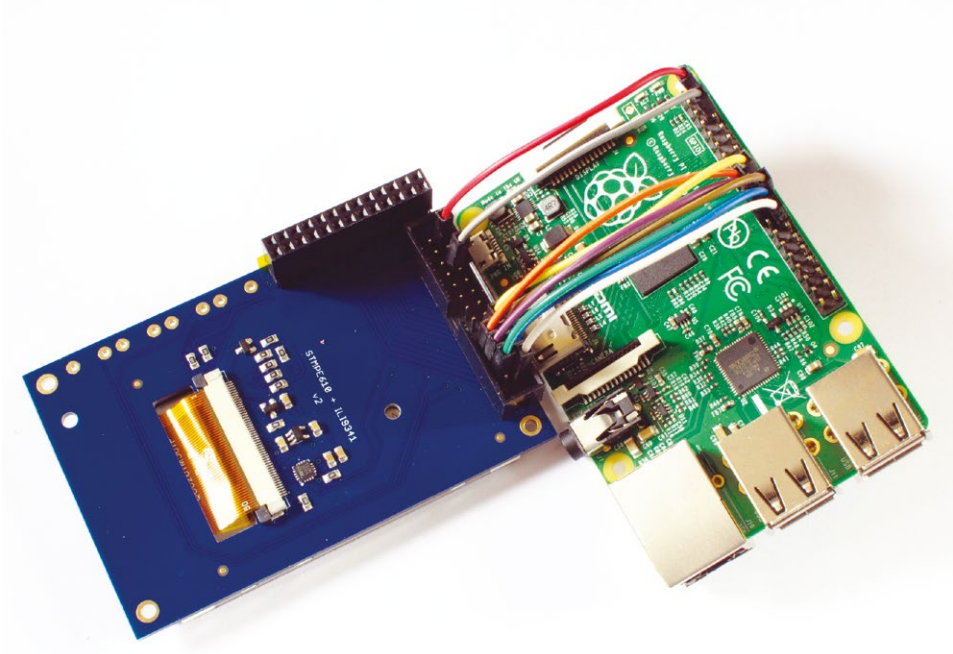
Ces opérations peuvent bien sûr être exécutées par le script Python, libre à vous d'ajouter une telle fonctionnalité au programme, par exemple en éteignant l'écran après 2 minutes d'inactivité...

Notez que seul le rétroéclairage s'éteint : on peut toujours contrôler l'affichage (même si on ne voit rien) et l'écran tactile reste pleinement fonctionnel.

6. CONNEXION PAR FILS

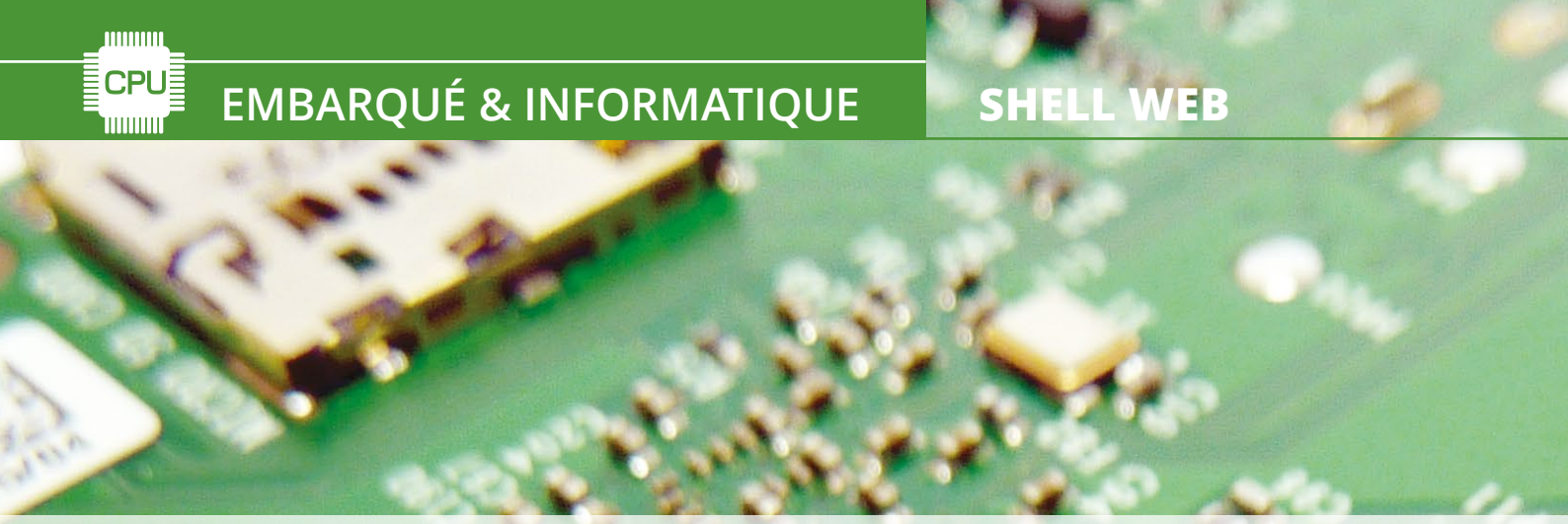
Dans le montage cible, on peut vouloir placer l'écran à une légère distance du Raspberry Pi, sans le connecter directement sur les GPIO. Pour cela, on peut utiliser les broches déportées à l'arrière de l'écran.

L'écran a besoin, au minimum, de l'alimentation 5V, de la masse, des pins SPI et des GPIO 24 et 25 ; cela correspond aux broches numérotées 2, 6, 18, 19, 21, 22, 23, 24 et 26. Il suffit de retourner l'écran puis de connecter les broches ensemble : les deux ensembles sont bien sûr identiques, vu que celles de l'écran servent à déporter les broches du Raspberry Pi ! **SMM**



L'écran et le Raspberry Pi peuvent être connectés par des fils, en utilisant uniquement les broches nécessaires.

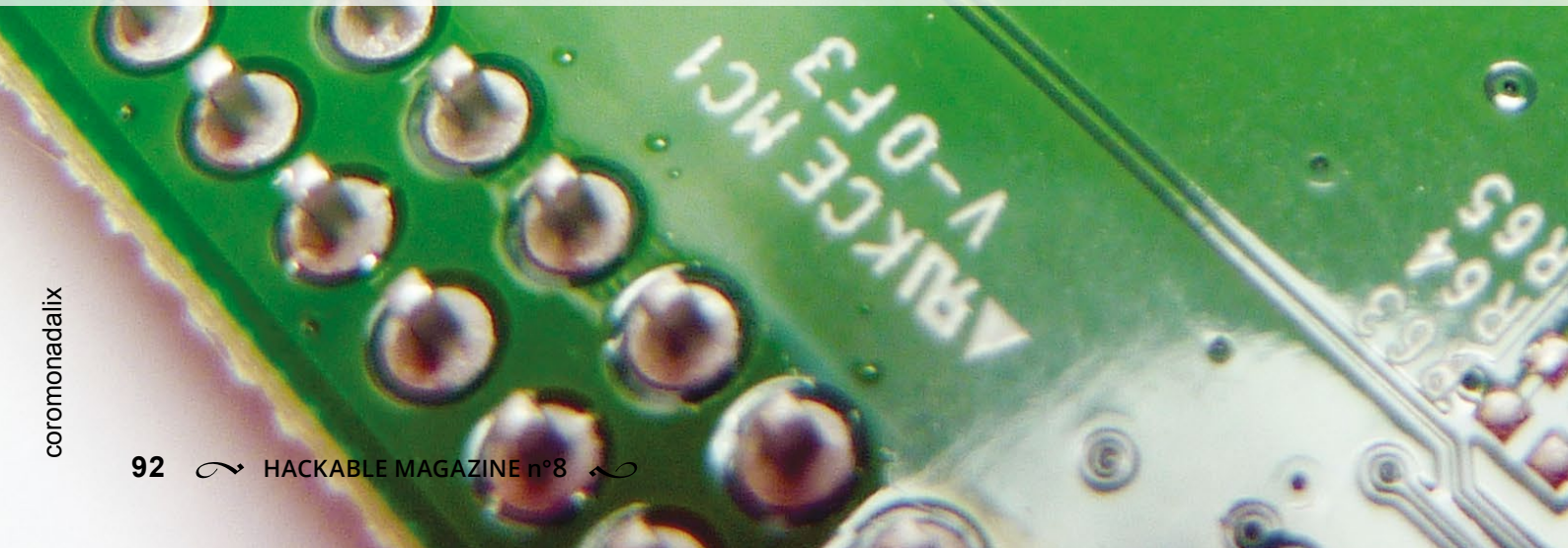
coromonadalix



UNE LIGNE DE COMMANDES DANS UN NAVIGATEUR POUR VOTRE PI 2

Denis Bodor

Gérer et manipuler le système d'une carte Raspberry Pi peut se faire de bien des façons. Écran+clavier, console série, connexion distante avec SSH… mais dans tous les cas, ceci nécessite généralement, d'une façon ou d'une autre, l'utilisation d'un logiciel sur la machine « cliente » qui se connecte à la Pi. Que diriez-vous d'éliminer cette obligation et d'accéder à votre chère framboise depuis un simple navigateur ?



coromonadalix

Cet article vous permettra d'accéder à la ligne de commandes (le shell) de la Raspberry Pi, via le réseau, avec un simple navigateur web. Mais ce n'est pas tout. En effet, nous allons utiliser ici un logiciel qui n'existe pas sous une forme directement installable pour la carte, du moins pour la Raspberry Pi 2. Rappelons au passage que la principale différence entre le modèle 2 et les précédentes déclinaisons (A, B, A+ et B+) réside dans la présence d'un processeur, ou plus exactement d'un SoC (*System on a Chip*), avec une architecture différente (ARM Cortex-A7 contre ARM11 pour les autres Raspberry Pi). Le passage d'un BCM2835 à un BCM2836, implique un changement d'architecture de ARMv6 à ARMv7, ce qui se traduit dans le système Raspbian par une transition de la plateforme *armel* à *armhf* et donc des logiciels en versions sensiblement différentes.

À l'heure où est écrit cet article, le logiciel qui nous intéresse n'existe, directement installable, qu'en version *armel* sous la forme d'un fichier **shellinabox_2.10-1_armel.deb** (notez la mention **armel** dans le nom du fichier). En tentant d'installer ce fichier sur la Raspbian d'une Raspberry Pi 2, le système n'hésitera pas à vous rappeler à l'ordre :

```
$ sudo dpkg -i ./shellinabox_2.10-1_armel.deb
dpkg: erreur de traitement de ./shellinabox_2.10-1_armel.deb (--install) :
l'architecture du paquet (armel) ne correspond pas à celle du système (armhf)
Des erreurs ont été rencontrées pendant l'exécution :
./shellinabox_2.10-1_armel.deb
```

dpkg est la commande permettant de gérer l'installation et la désinstallation de logiciels au niveau le plus bas. Pour les logiciels (ou paquets) directement mis à disposition par Raspbian, vous utiliserez plutôt **apt-get** qui se charge comme un grand d'installer les logiciels et bibliothèques nécessaires au fonctionnement de celui qui vous intéresse (on parle alors de dépendances). **dpkg** est utilisé par **apt-get** afin que vous n'ayez pas à gérer tout cela vous-même à la main. Mais pour installer un paquet non disponible au travers de Raspbian, vous n'avez d'autre choix que d'utiliser **dpkg**.

1. PETIT RAPPEL SUR LE FONCTIONNEMENT DE RASPBIAN ET DEBIAN

Au commencement était GNU/Linux et toute une tripotée d'applications, d'outils, d'utilitaires, de logiciels et de bibliothèques. Il était de coutume alors d'assembler manuellement son système en récupérant les sources des logiciels et en les compilant pour agrémenter son installation. Cette méthode, bien que parfaitement fonctionnelle et offrant un contrôle total sur les éléments installés n'est absolument pas pratique sur le long terme. En effet, au fil des mises à jour, de plus en plus de fichiers sont installés sur le système, certains encore utiles et d'autres non. De plus, l'installation d'une application suppose la copie de nombreux fichiers à différents endroits du système et la désinstaller tourne rapidement à une chasse aux fichiers disséminés dans toute l'arborescence.

Est alors arrivée la notion de distributions dans lesquelles sont intégrés des systèmes de gestion permettant de garder une trace de tout ce qui est installé, des relations entre les différents éléments (dépendances) et facilitant les mises à jour en remplaçant ce qui devient obsolète par de nouvelles versions.

Bien entendu, cette gestion supplémentaire a nécessité la création et l'utilisation d'outils spécifiques ainsi que la confection de fichiers spécialement prévus dans ce sens pour contenir les applications, logiciels et bibliothèques. Des systèmes de gestion de paquets et des paquets ont alors vu le jour, spécifiques à chaque distribution et tantôt partagés entre plusieurs d'entre elles. Mieux encore, pour

coromonadalix

faciliter la création de paquets, des outils et des documentations ont également été créés.

Les paquets permettant d'installer facilement des éléments sont généralement composés de binaires. Comprenez par là qu'ils contiennent, par exemple, un logiciel compilé et donc directement utilisable par le système, par opposition aux sources du logiciel (version intelligible par un humain/programmeur). Ceci signifie donc aussi qu'il faut une version de chaque paquet pour chaque type de machine. Un paquet pour votre distribution Debian sur PC ne pourra pas être installé tel quel sur votre Raspberry Pi.

Les avantages sont évidents et les inconvénients tout autant : si un logiciel n'a pas été spécialement préparé et qu'il n'existe donc pas sous la forme d'un paquet, son installation s'avère plus délicate. Plusieurs cas de figure peuvent se présenter :

- Il n'y a pas du tout de paquet pour ce logiciel ou cette bibliothèque : la solution optimale consiste à créer ce paquet et le partager avec les autres utilisateurs afin qu'ils ne rencontrent pas le même problème que vous. Ceci nécessite cependant un certain nombre de compétences techniques et toute une procédure pour que le fruit de votre travail rejoigne la longue liste de paquets existants et officiellement mis à disposition. Bien sûr, il vous est également possible d'installer l'élément à la main, mais ceci se fera au risque de perturber le fonctionnement de l'ensemble, le système de gestion de paquets n'étant alors pas du tout au courant de l'installation.
- Il existe un paquet, mais pas pour votre plateforme (pour PC oui, mais pas pour la Pi) : il faut alors se rabattre sur le paquet source ou, en d'autres termes, les sources du logiciel, mais agrémentées de fichiers supplémentaires permettant de construire facilement un nouveau paquet pour votre système.
- Il existe un paquet, mais pas pour votre distribution : c'est un peu plus délicat, car même s'il existe des outils pour convertir des paquets d'une distribution à une autre (Fedora vers Debian par exemple), le processus n'est jamais parfait et n'équivaudra pas à la création d'un véritable paquet. Il s'agit tout au plus de jongler avec les fichiers pour vaguement satisfaire des besoins ponctuels.

La distribution Debian, qui est à l'origine de Raspbian, mais également d'Ubuntu, utilise des outils comme **`dpkg`**, **`apt-get`**, **`aptitude`** ou encore **`apt-cache`** pour gérer les paquets au format DEB, dont les noms se terminent avec le nom de l'architecture à laquelle le paquet est destiné et l'extension **`.deb`**.

Voici précisément qui nous ramène à l'erreur que nous évoquions précédemment. Les développeurs du logiciel qui nous intéresse proposent les fichiers suivants (liste non exhaustive) :

- **`shellinabox_2.10-1_amd64.deb`** : un paquet pour architecture PC Intel/AMD en 64 bits ;
- **`shellinabox_2.10-1_i386.deb`** : un paquet pour architecture PC Intel/AMD en 32 bits ;
- **`shellinabox_2.10-1_armel.deb`** : un paquet pour architecture ARM 32 bits répondant aux spécifications avant la version 7 des architectures ARM (ARMv7). Ce paquet conviendra parfaitement pour une Raspberry Pi à base de Broadcom BCM2835 (A, B, A+, B+), mais pas pour une Raspberry Pi 2 avec un BCM2836,
- **`shellinabox-2.10.tar.gz`** : une archive contenant les sources de la version 2.10,
- **`shellinabox-2.14.tar.gz`** : une archive pour la version 2.14 (la plus récente à cette date).

Ce qu'il nous faut pour notre adorable Pi 2 manque à l'appel : pas de **`*_armhf.deb`** (pas plus que **`*_arm64.deb`** correspondant à l'architecture ARM 64 bits encore relativement rare (comme le monstrueux SoC Nvidia Tegra K1 équipant la non moins monstrueuse et coûteuse carte Jetson TK1)).

coromonadalix

2. À PROPOS DE SHELL IN A BOX

L'outil permettant de nous fournir un accès à la ligne de commandes sans avoir à utiliser de logiciel ou de matériel spécifique se nomme *Shell In A Box* et est disponible en logiciel libre sous licence GNU GPL v2. Une fois lancé sur une machine (une Raspberry Pi ou tout autre système GNU/Linux) il démarre automatiquement un serveur web proposant une unique page utilisant JavaScript et CSS/HTML. Cette page dynamique affiche une console permettant à l'utilisateur d'un quelconque navigateur de dialoguer avec le système distant, sans rien installer, même pas un greffon ou une extension.

Shell In A Box est généralement utilisé pour accéder à la ligne de commandes du système (le shell), mais pourra également être utilisé pour n'importe quel type d'interface en mode texte. Par défaut, c'est **`/bin/login`** qui est lancé par *Shell In A Box* proposant à l'utilisateur de saisir un nom et un mot de passe comme il le ferait sur une console normale (écran+clavier ou série avec adaptateur USB).

Nous utiliserons ici *Shell In A Box* plus ou moins avec sa configuration par défaut, mais sachez qu'il est possible de le personnaliser à l'extrême aussi bien d'un point de vue esthétique qu'en termes de fonctionnalités. La documentation est relativement claire (mais en anglais) et décrit les différents paramètres qu'il est possible d'utiliser.

3. CONSTRUIRE UN NOUVEAU PAQUET POUR UNE INSTALLATION PROPRE

Construire un paquet n'a rien de véritablement complexe... si les éléments adéquats sont présents. Il faut, d'une part, que les sources possèdent les éléments permettant cette manipulation. Deux cas de figure se présentent : soit les éléments sont ajoutés par les développeurs du logiciel qui nous intéresse, soit il sont disponibles séparément. Dans le cas de *Shell In A Box*, le fichier contenant les sources, **`shellinabox-2.14.tar.gz`**, dispose d'un répertoire **`debian/`** avec tout le nécessaire. L'autre cas de figure prend la forme d'une archive avec les sources du logiciel telles que développées par les programmeurs (on parle souvent de sources *upstream*), d'un fichier décrivant le paquet et d'un *patch* contenant les modifications à apporter aux sources *upstream* pour en faire un paquet source pour la distribution (lui-même permettant de produire un paquet binaire installable).

L'autre élément indispensable pour construire un paquet se compose des outils permettant l'opération, comme le minimum vital pour compiler une application écrite en C/C++ (paquet **`build-essential`**) et construire un paquet (**`dpkg-dev`** et souvent **`debhelper`**), mais également de tout ce que les sources du logiciel en question ont besoin pour être compilé. De la même manière qu'il existe des dépendances pour les paquets binaires, nous avons donc des dépendances de construction à respecter.

Entrons sans plus attendre dans le vif du sujet en tentant de construire notre paquet *Shell In A Box* pour notre Raspberry Pi 2. Nous commençons par récupérer les sources du logiciel directement sur le site du projet (**http://code.google.com/p/shellinabox/**) grâce à la commande **`wget`** :

```
$ wget http://shellinabox.googlecode.com/files/
shellinabox-2.14.tar.gz
```

Il s'agit d'une archive Tar compressée avec Gzip, nous désarchivons donc cela avec :

```
$ tar xfzv shellinabox-2.14.tar.gz
[...]
shellinabox-2.14/shellinabox/vt100.jspp
shellinabox-2.14/shellinabox/white-on-black.css
shellinabox-2.14/stresstest.sh
shellinabox-2.14/update
```

coromonadalix

Nous obtenons un répertoire **shellinabox-2.14** plein de fichiers. Nous nous plaçons dans ce répertoire et tentons une construction :

```
$ cd shellinabox-2.14

$ dpkg-buildpackage -b
dpkg-buildpackage: paquet source shellinabox
dpkg-buildpackage: version source 2.14-1
dpkg-buildpackage: source changé par Marc Singer <elf@debian.org>
dpkg-buildpackage: architecture hôte armhf
dpkg-source --before-build shellinabox-2.14
dpkg-checkbuilddeps : dépendances de construction non trouvées :
  debhelper (>= 8.0.0) autotools-dev libssl-dev libpam0g-dev
dpkg-buildpackage: avertissement: dépendances de construction
  et conflits non satisfaits ; échec.
[...]
```

dpkg-buildpackage comme son nom le laisse entendre est la commande nous permettant de construire un paquet binaire à partir d'un paquet source. Cet outil utilise le contenu du sous-répertoire **debian/** présent dans le répertoire courant. On trouve ici tout un tas d'informations sur la façon de compiler le logiciel et d'en faire un paquet, mais également une liste de ce dont il a besoin pour cela. Naturellement, notre système ne contient pas tout le nécessaire et **dpkg-buildpackage** nous rappelle gentiment à l'ordre en précisant les éléments absents.

Pour régler le problème, il suffit d'installer tout cela, non sans avoir faire une petite mise à jour préalable de la liste des paquets :

```
$ sudo apt-get update
$ sudo apt-get install debhelper autotools-dev libssl-dev libpam0g-dev
```

Si vous prenez l'habitude de construire des paquets, il ne vous sera pas systématiquement nécessaire d'installer des paquets supplémentaires. **debhelper** par exemple, mais aussi **autotools-dev**, sont utilisés par de nombreux logiciels. Au fil des manipulations, vous constituerez naturellement un minimum vital satisfaisant la plupart des constructions. Remarquez le **-dev** présent dans les noms des deux autres paquets que nous installons. Il s'agit de paquets de développement généralement associés aux bibliothèques. Ainsi, les applications utilisant les protocoles SSL/TLS ont besoin des éléments du paquet **libssl** pour fonctionner. Pour compiler/construire une application qui utilise SSL/TLS, le paquet **libssl** seul n'est pas suffisant, il faut disposer des éléments permettant de lier l'application à la bibliothèque. Comme ceci est moins courant que la simple utilisation, ces éléments sont placés dans des paquets distincts (avec **-dev** à la fin de leur nom) pour ne pas encombrer le système.

Nous pouvons maintenant tenter une nouvelle construction :

```
$ dpkg-buildpackage -b
dpkg-buildpackage: paquet source shellinabox
dpkg-buildpackage: version source 2.14-1
dpkg-buildpackage: source changé par Marc Singer <elf@debian.org>
dpkg-buildpackage: architecture hôte armhf
[...]
dpkg-source --after-build shellinabox-2.14
dpkg-buildpackage: envoi d'un binaire seulement
  (aucune inclusion de code source)
```

Nous n'avons plus aucun message d'erreur, signe que l'opération s'est bien déroulée. Dans le répertoire parent, nous retrouvons un fichier **shellinabox_2.14-1_armhf.deb**. C'est le paquet binaire que nous souhaitions obtenir (remarquez le **armhf** dans le nom). Il ne nous reste plus qu'à l'installer :

```
$ cd ..
$ sudo dpkg -i shellinabox_2.14-1_armhf.deb
Sélection du paquet shellinabox précédemment désélectionné.
(Lecture de la base de données... 80112 fichiers et répertoires déjà installés.)
Dépaquetage de shellinabox (à partir de shellinabox_2.14-1_armhf.deb) ...
Paramétrage de shellinabox (2.14-1) ...
Traitement des actions différées (" triggers ") pour " man-db "...
```

Et voilà ! Nous avons maintenant, dans notre système, la version 2.14 de *Shell In A Box* correctement et proprement installée. Le système de gestion de paquets se charge de tenir tout cela en ordre et nous pourrons, le moment venu, désinstaller cette application à l'aide d'un simple **sudo apt-get remove** sans le moindre problème. Mieux encore, si *Shell In A Box* intègre plus tard les dépôts officiels, la mise à jour désinstallera automatiquement notre paquet pour le remplacer par le nouveau.

À ce stade, vous pouvez désinstaller les paquets nécessaires à la construction si vous le souhaitez (**debhelper**, **autotools-dev**, **libssl-dev** et **libpam0g-dev**). Il ne sont pas utiles au fonctionnement de *Shell In A Box* et vous économiserez un peu de place (sauf bien entendu, si vous comptez réitérer ces manipulations avec d'autres paquets).

4. UTILISATION DE SHELL IN A BOX

Par défaut après son installation, *Shell In A Box* est automatiquement lancé avec un certain nombre de paramètres par défaut, dont le port sur lequel il attend les connexions : 4200. Il faut donc, en principe spécifier dans le navigateur une URL comme **https://ip.de.la.carte:4200**.

Par défaut également, *Shell In A Box* tentera de sécuriser la communication du mieux qu'il peut en utilisant le protocole HTTPS, autrement dit HTTP sur SSL/TLS (avec le petit cadenas dans la barre d'URL). Sans entrer trop dans le détail, le fonctionnement de SSL/TLS repose sur la notion d'autorité de certification qui garantit l'identité du serveur. Lorsque le petit cadenas est vert dans un navigateur, cela signifie que le serveur auquel vous vous connectez chiffre la communication et utilise un certificat qui lui a été confié après vérification (et paiement) par une autorité de certification reconnue par votre navigateur.

Dans le cas de *Shell In A Box*, vous n'avez pas fait ces démarches administratives pour que votre Raspberry Pi 2 puisse vous assurer de son identité. *Shell In A Box* a tout simplement généré lui-même un certificat et c'est pour cette raison que votre navigateur vous signale un potentiel problème de sécurité. Attention, ceci ne signifie pas que la communication n'est pas protégée (elle est effectivement toujours chiffrée), mais simplement que rien ne prouve l'identité de votre interlocuteur. Le certificat généré est placé dans un fichier **certificate.pem** dans le répertoire **/var/lib/shellinabox**.

Pour personnaliser le comportement de *Shell In A Box*, nous devons nous tourner vers le fichier **/etc/default/shellinabox** que vous pouvez éditer, par exemple avec la commande **sudo nano**. La première ligne qui nous intéresse est :

coromonadalix

```
SHELLINABOX_PORT=4200
```

que nous changeons en :

```
SHELLINABOX_PORT=443
```

Ceci demande à *Shell In A Box* d'attendre les connexions sur le port 443 dédié à HTTPS. Il nous suffira alors de pointer notre navigateur sur **https://ip.de.la.carte** (plus besoin du ":4200"). L'autre ligne intéressante est :

```
SHELLINABOX_ARGS="--no-beep"
```

C'est ici que nous pourrons ajouter les options à utiliser au lancement de *Shell In A Box*. Si, par exemple, vous souhaitez désactiver HTTPS et utiliser HTTP pour ne plus avoir le message d'avertissement (attention, la communication n'est alors plus chiffrée), vous modifierez la ligne en :

```
SHELLINABOX_ARGS="--no-beep -t"
```

Il sera également souhaitable de changer la ligne précédente de façon à utiliser le port 80 et non 443. Dès lors l'URL pour accéder à la ligne de commandes deviendra **http://ip.de.la.carte**.

Dans tous les cas, une modification du fichier de configuration nécessite le redémarrage de *Shell In A Box* côté Raspberry Pi avec :

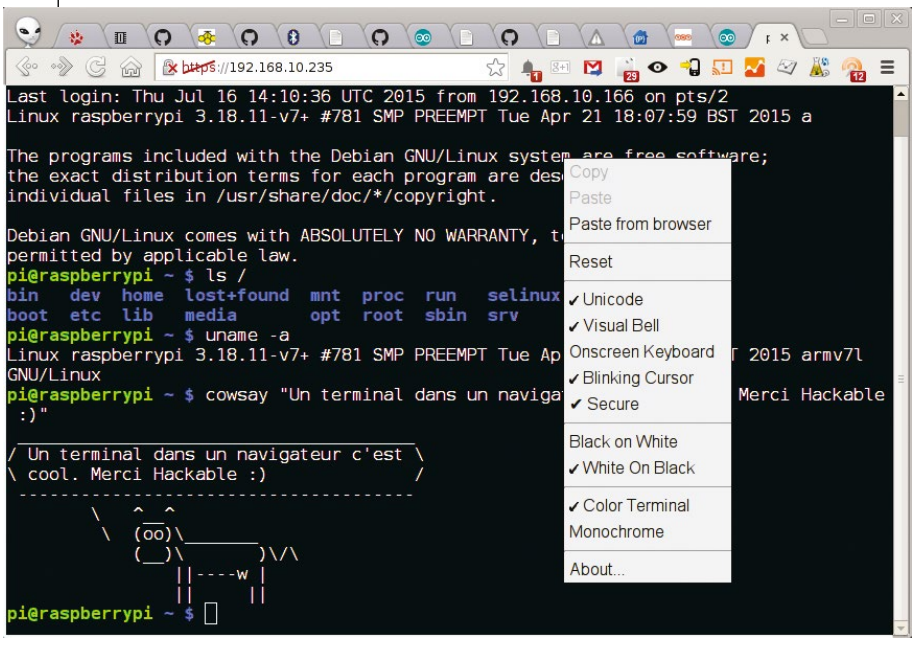
```
$ sudo service shellinabox restart
```

ou

```
$ sudo /etc/init.d/shellinabox restart
```

Dès que le service est en fonction, ainsi qu'au démarrage de la Raspberry Pi, vous pourrez accéder à la carte depuis n'importe quel navigateur web supportant JavaScript et CSS (presque tous donc). À la connexion, tout comme avec un terminal classique, un nom d'utilisateur et le mot de passe correspondant vous seront demandés. L'accès au shell que vous obtenez alors est magnifiquement similaire à ce que vous pouvez obtenir avec SSH ou une console série. Nous avons testé bon nombre d'applications et outils en mode texte (Vim, Midnight Commander, GNU Screen, etc.) et tous fonctionnent à merveille. *Shell In A Box* est définitivement une solution bien pratique pour accéder à distance à la Pi...

Le seul point noir au tableau est l'interférence entre certaines touches de raccourcis utilisables en ligne de commandes qui sont alors interprétées par le navigateur. Si comme moi vous avez des réflexes de vieil utilisateur du shell, ceci s'avère vite énervant. Typiquement CTRL+W, qui permet d'effacer un mot en ligne de commandes, ferme l'onglet du navigateur... frustrant ! **DB**



Voici le résultat de nos quelques manipulations : une ligne de commandes dans un onglet d'un navigateur (ici Chrome). Notez qu'un menu contextuel (clic droit) nous permet d'ajuster quelques paramètres et offre même la possibilité d'utiliser un clavier visuel (parfait pour un écran tactile).

coromonadalix

NE MANQUEZ PAS OPEN SILICIUM N°15 !

FOCUS :

INITIEZ-VOUS AU RÉSEAU TEMPS RÉEL

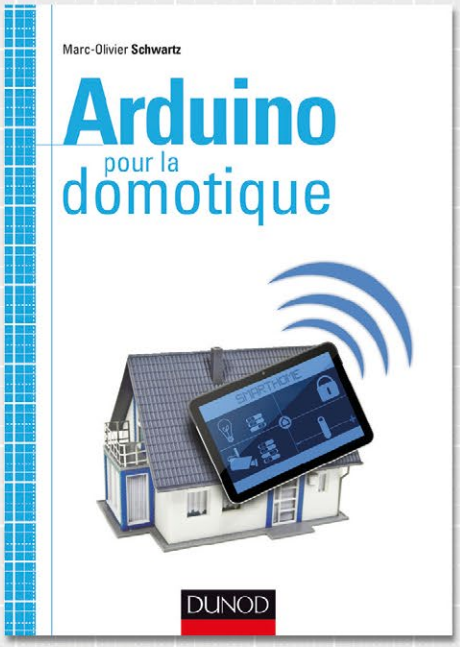
AVEC XENOMAI ET RTNET

TOUJOURS DISPONIBLE SUR :

www.ed-diamond.com

TOUS MAKERS !

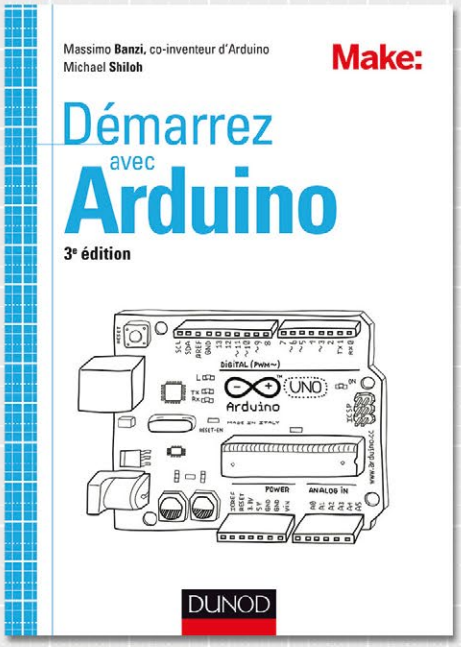
RÉVÉLEZ VOTRE POTENTIEL PAR LA CRÉATION



9782100727117, 256 p., 27,50 €

MARC-OLIVIER SCHWARTZ

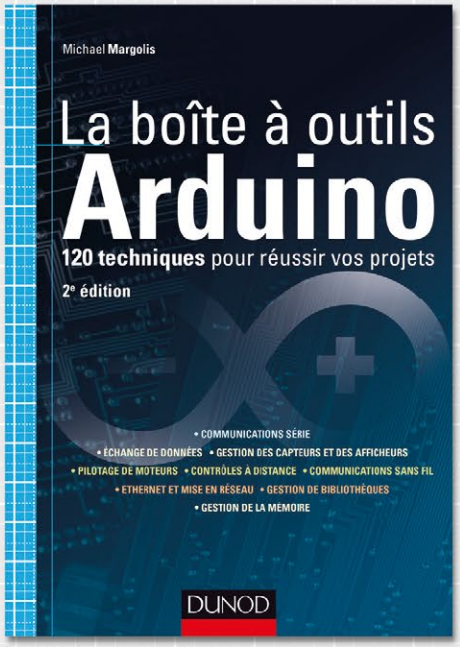
Le guide pas-à-pas de projets concrets avec des exemples de code, des schémas et des photos pour réaliser vous-même des applications domotiques.



9782100727391, 192 p., 19,90 €

MASSIMO BANZI, MICHAEL SHILOH

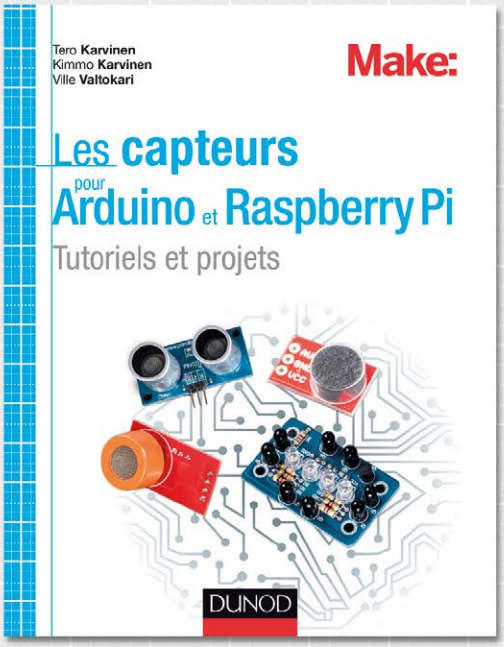
Une présentation accessible d'Arduino et les bases en électronique et programmation pour sa mise en œuvre immédiate.



9782100727124, 480 p., 35 €

MICHAEL MARGOLIS

120 solutions pour réaliser vous-même des applications concrètes avec une carte programmable Arduino.



9782100717934, 304 p., 29,90 €

TERO KARVINEN ET AL.

Bien utiliser les capteurs pour concevoir des montages avec Arduino ou Raspberry Pi qui interagissent avec leur environnement.

TOUT LE CATALOGUE SUR **WWW.DUNOD.COM**

DUNOD
ÉDITEUR DE SAVOIRS