

GNU  
**LINUX**  
 MAGAZINE / FRANCE

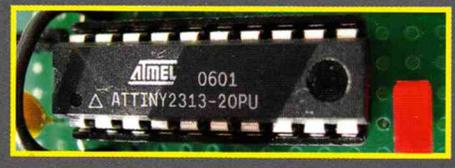


**HORS-SÉRIE**

Administration et développement sur systèmes UNIX

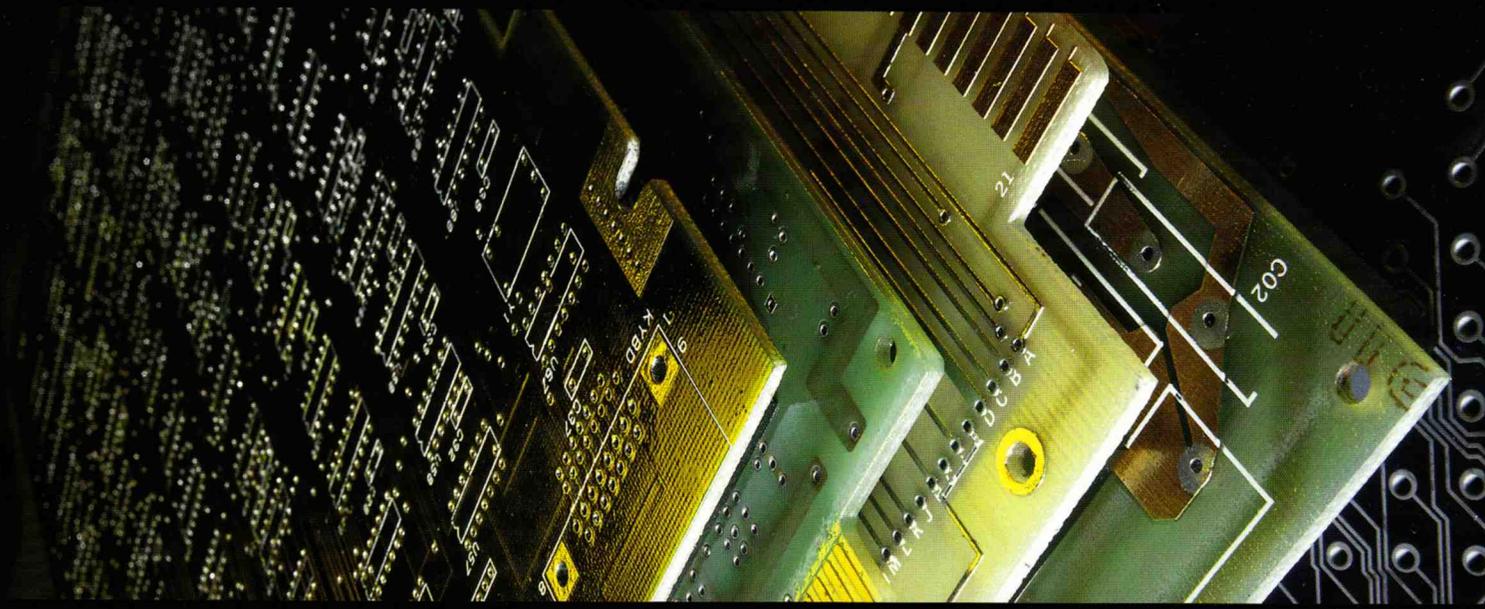
**ÉLECTRONIQUE**

► Réalisez votre périphérique série ou USB à peu de frais grâce au microcontrôleur ATtiny2313 (p. 12)



**ÉLECTRONIQUE  
 EMBARQUÉ & DOMOTIQUE**

**100 % PRATIQUE**



**PROGRAMMATION**

► Développez des applications graphiques portables grâce à Qt4 (p. 40)

**DOMOTIQUE**

► Découvrez le standard sans fil ZigBee et réalisez un capteur de température (p. 26)

**EMBARQUÉ**

► Explorez la puissance des FPGA, les réseaux de portes logiques programmables (p. 58)



L 15066 - 38 H - F: 6,50 € - RD  
 France Métro : 6,50 € - DOM : 7,00 €  
 TOM Surface : 950 XPF  
 POL. A : 1400 XPF  
 BEL/PORT.CONT : 7,50 €  
 CH : 13,8 CHF  
 CAN : 13 \$CAD  
 MAR : 75 MAD

## 100% SÉCURITÉ INFORMATIQUE

Actuellement chez votre marchand de journaux !!

38 JUILLET/AOÛT 2008

France Métro : 9 € / DOM : 8,50 € / TOM Surface : 9,50 XPF / TOM Avion : 13,00 XPF / CH : 15,50 CHF  
BEL, LUX, PORT/CANT : 9 Eur / CAN : 15 \$CAD

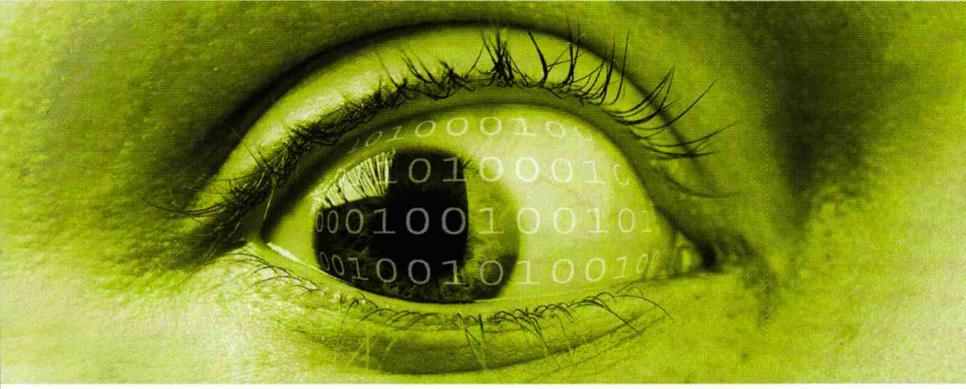
**MISC**  
Multi-System & Internet Security Cookbook

100 % SÉCURITÉ INFORMATIQUE

DOSSIER

### CODES MALICIEUX QUOI DE NEUF ?

- Le ver Storm
- Extorsion par des Ransomwares
- Menace virale en PDF
- ...



<b>VULNÉRABILITÉ</b> LA faille OpenSSL chez Debian (p. 18)	<b>SCIENCE</b> Sécurité des communications vocales (3/3) : techniques numériques (p. 77)	<b>INFOWAR</b> La doctrine chinoise vue par les États-Unis (p. 26)
---	---	---

L 19018 - 38 - F : 8,00 € - RD



### Sommaire

#### Témoignage

- Évaluation de l'antivirus Dr Web : l'antivirus qui venait du froid

#### Vulnérabilité

- La faille OpenSSL/Debian

#### Infowar

- La puissance militaire de la République Populaire de Chine : rapport 2008 du département de la Défense des États-Unis

#### DOSSIER :

##### Code malicieux : quoi de neuf ?

- Les changements climatiques et les logiciels malicieux
- Analyse du phénomène ransomware
- Les nouveaux malwares de document : analyse de la menace virale dans les documents PDF

#### Système

- Détection de malware par analyse système

#### Science

- La sécurité des communications vocales (3) : techniques numériques

## Reportage

p. 04 **RMLL 2008 : un cru exceptionnel !**

## Électronique

p. 06 **Offrez une grenouille à votre pingouin !**

p. 12 **Développez vos périphériques série**

p. 18 **PIC et USB**

p. 23 **L'Attiny13, tout petit mais costaud**

p. 24 **Time lapse : la méthode quick & dirty**

## Domotique

p. 26 **Étude et réalisation d'un capteur sans fil ZigBee**

## Code(s)

p. 40 **Communication asynchrone et interface graphique portables sous Qt**

## Embarqué

p. 52 **IPCop sur Soekris net4801**

p. 58 **Instrumentation scientifique reconfigurable**

## Abonnement

p. 71, 73, 74 **Bons d'abonnement et de commande**



Bienvenue dans ce nouveau hors-série consacré, une fois n'est pas coutume, à l'électronique, la domotique et l'embarqué avec/sous GNU/Linux. Je vais vous laisser découvrir par vous-même le contenu de ce numéro qui, je l'espère, vous permettra de vous occuper l'esprit et les mains pendant la période estivale.

J'aimerais maintenant profiter de cette page pour, comme d'autres le font déjà, éveiller votre attention sur les dérives légales concernant l'utilisation des nouvelles technologies. Je parle, bien entendu, de ce mouvement visant à sur-contrôler l'utilisation d'Internet. Ce mouvement de fond prend, et c'est bien ce qui est inquiétant, plusieurs formes et plusieurs noms : projet Olivennes, riposte graduée, Paquet Télécom, labellisation, lutte contre la cybercriminalité...

Les motivations annoncées sont légitimes. Il s'agit de protéger l'internaute contre les menaces qui le guettent et de faire appliquer les lois mises en place de longue date comme celles protégeant les droits d'auteur. Cependant, les mesures destinées à être mises en place sont, selon un grand nombre d'acteurs du Net et des TIC, non seulement démesurées, mais parfaitement inadaptées.

On ne peut s'empêcher de comparer l'ensemble des éléments de ces projets à une surveillance sans limite du réseau par des autorités soumises à aucun contrôle. Certains parlent de soviétisation d'internet, d'autres de l'émergence d'un Big Brother bien plus effrayant encore que celui décrit par George Orwell.

Mais, il y a pire encore. Nous avons sous les yeux un remake d'erreurs qui sont souvent commises dans le monde des technologies fermées : un scénario catastrophe découlant d'une malheureuse convergence d'intérêts. Qu'ont en commun un cybercriminel, un terroriste, un distributeur de médias illégalement copiés, un développeur de logiciels libres et un internaute soucieux de ses libertés individuelles et de la protection de sa vie privée ? Réponse : le fait de disposer d'outils, de technologies lui permettant de protéger ses communications et ses données.

Voilà bien le cocktail explosif qui s'annonce. Des groupes aux motivations bien différentes (voire opposées) en viennent à trouver un intérêt commun et s'unissent implicitement pour défendre leurs intérêts individuels. Rappelez-vous de la présentation de Michael Steil au 22c3 à Berlin intitulée « Les 17 erreurs que Microsoft a commises dans le système de sécurité de la Xbox ». L'une d'elle, sinon la plus grave, a été de provoquer une situation où les développeurs GNU/Linux, les développeurs de jeux Homebrew et les revendeurs de jeux pirates ont tous eu intérêt à comprendre, analyser et partager les informations sur la console et son système de sécurité.

En dehors de considérations citoyennes évidentes, ce qui se présente dans un avenir proche ressemble à cette convergence malsaine et les conséquences seront sans doute terribles pour tous. Ce mouvement sécuritaire est maintenant définitivement une bombe à retardement...

Je vous laisse réfléchir à tout cela et vous donne rendez-vous au 19 octobre pour un nouveau hors-série, mais également au 30 août prochain pour le numéro 108 de la rentrée.

Denis Bodor

GNU/Linux Magazine France Hors-série

est édité par Diamond Editions  
B.P. 20142 - 67603 Sélestat Cedex

Tél. : 03 88 58 02 08 - Fax : 03 88 58 02 09

E-mail : [lecteurs@gnulinuxmag.com](mailto:lecteurs@gnulinuxmag.com)

Service commercial : [abo@gnulinuxmag.com](mailto:abo@gnulinuxmag.com)

Sites : [www.gnulinuxmag.com](http://www.gnulinuxmag.com)  
[www.ed-diamond.com](http://www.ed-diamond.com)

Directeur de publication : Arnaud Metzler

Rédacteur en chef : Denis Bodor

Secrétaire de rédaction : Véronique Wilhelm

Relecture : Dominique Grosse

Conception graphique : Fabrice Krachenfels

Responsable publicité : Tél. : 03 88 58 02 08

Service abonnement : Tél. : 03 88 58 02 08

Impression : VPM Druck 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 :

Distri-médias :

Tél. : 05 61 72 76 24

IMPRIMÉ en Allemagne - PRINTED in Germany

Dépôt légal : À parution / N° ISSN : 1291-78 34

Commission paritaire : 09 08 K78 976

Périodicité : Bimestrielle

Prix de vente : 6,50 €

## RMLL 2008 : un cru exceptionnel !



Comme chaque année, les RMLL, Rencontres Mondiales du Logiciel Libre ou Rheumeulheulheu pour les intimes, marquent le début de l'été et sont l'occasion pour la communauté des développeurs et utilisateurs de logiciels libres de se retrouver pour travailler, promouvoir l'esprit open source et faire découvrir des projets prometteurs.

Et cette année, c'est la ville de Mont-de-Marsan qui a eu le plaisir d'accueillir cette manifestation et on ne peut s'empêcher de faire un lien entre le logiciel libre et la gastronomie, tant il est évident que, qui dit « richesse gastronomique du terroir » dit « RMLL exceptionnelles » ! Quel savoureux mélange : canard, vin du pays, code, floc de Gascogne, distributions GNU/Linux, foie gras, embarqué, cèpes, \*BSD... Il y a des choses qui se marient si bien en ce monde...

Mais, remettons l'analyse et la démonstration de cette théorie à plus tard, car il n'est pas de mariage réussi sans un orchestre capable de rythmer la fête. L'orchestre dans une manifestation comme les RMLL, c'est le comité d'organisation.

Profitions-en pour remercier le chef d'orchestre, Jean-Christophe Elineau, président du comité d'organisation et président de Landinux, ainsi que toute son équipe, sans oublier les étudiants de l'IUT informatique de Mont-de-Marsan, qui ont mis tout en œuvre pour que ces 9èmes RMLL se déroulent sans la moindre anicroche. Une organisation irréprochable, une équipe soudée et motivée et une chaleur humaine incontestée. Voilà la formule magique pour réussir des RMLL exemplaires.

Car oui, avec plus de 4000 inscrits, on peut parler de vraie réussite ! La manifestation a également accueilli des visiteurs d'autres pays, pas moins de 40 nations étaient représentées sur le site de Mont-de-Marsan, dont une délégation tunisienne (impliquée dans la promotion des



Dès l'accueil, le ton est donné. Les RMLL 2008, c'est du sérieux et on ne plaisante pas avec l'organisation.

logiciels libres en Tunisie) et le CHALA (Club des Hommes et Femmes d'Affaires du Libre en Afrique), organisateur des RALL (Rencontres Africaines du Logiciel Libre).

Monsieur Richard M. Stallman était également présent sur les lieux et a pris place au repas du Libre parmi les 400 autres participants. Pour le plus grand plaisir de tous, il s'est ainsi mêlé à la foule pour vendre (avec beaucoup de succès d'ailleurs) ses gnous en peluche et se plier au jeu du « j'ai une photo de moi avec RMS aux RMLL2008 »... Ceci avant de gratifier tous les convives du repas d'un solo de flûte. Eh oui, ce n'est pas parce qu'on est une figure, sinon la plus importante, du logiciel libre que l'on doit avoir cela comme seule passion.



Les RMLL et Armelle, le personnage faisant office de logo dessiné par André Pascual pour l'édition 2001.



L'abus d'alcool est dangereux pour la santé... même pour celle des gnous vendus par Richard.



*Le comité d'organisation de l'édition 2008 presque au complet. Les traits tirés mais le sourire, des personnes qu'on ne remerciera jamais assez pour leur travail...*



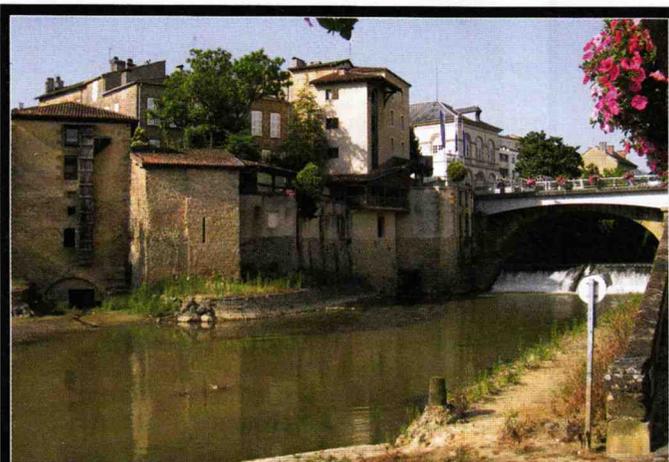
*Le village associatif regroupant les stands des associations et projets venant faire leur promotion et celle du logiciel libre, croulant sous la chaleur d'un mardi caniculaire.*

Ces 9èmes RMLL ont également eu l'honneur de recevoir Henri Emmanuelli, député PS des Landes, Président du Conseil Général des Landes et Conseiller Général du canton de Mugron, qui est quand même resté – soulignons-le – toute une après-midi sur les lieux, chose relativement exceptionnelle pour un « politique » qui, au départ, venait simplement y faire un saut pour se faire une idée. Sa venue était notamment l'occasion d'un échange constructif autour d'une table ronde, à laquelle ont pris place à ses côtés Jean-Christophe Elineau, Jacques Marsant, directeur de Landes Mutualité et Benoît Sibaud, président de l'APRIL.

Comme à l'habitude, chacun pouvait y trouver son compte parmi les thématiques proposées : Accessibilité, Collectivités locales, Éducation, Solutions d'entreprises,



*La photo qu'on se doit de faire, avec de gauche à droite Aline Gérard (LP), Denis Bodor (rédac-chef GLMF), RMS (GNU/FSF) et Fleur Brosseau (rédac-chef LP).*



*Les RMLL sont aussi l'occasion de découvrir une région, une ville, son architecture, son âme et sa gastronomie.*

Loisirs numériques, Santé, Sécurité, Web, Embarqué, etc. Certes, la plupart des visiteurs appartenait plus ou moins au monde du logiciel libre, mais le « grand public » n'était pas en reste : de nombreux ateliers de découverte étaient à leur disposition, de même que certaines conférences dont le sujet était plus ouvert que d'autres.

Quid des scoops ? Oui, il y en a eu et pas des moindres : Madame le Maire de Mont-de-Marsan a annoncé lors de discours d'ouverture que la municipalité allait migrer entièrement vers des logiciels libres. Presque une évidence finalement, lorsque l'on prend conscience des intérêts à la fois

culturels, économiques et sociaux d'une telle démarche. De quoi peut-être réveiller clairement et définitivement à la fois l'ensemble du paysage politique, mais également les décideurs de tous poils.

Que dire de plus, les RMLL sont un peu comme toute manifestation incontournable : si vous n'y étiez pas, vous avez raté quelque chose. Bien entendu, lors de cette édition 2008, fut annoncée la prochaine destination des RMLL : il s'agit de la ville de Nantes. Un autre climat, un autre joli coin de notre beau pays. Les Landais ont mis la barre haute, mais le défi est relevé pour 2009... Voilà l'occasion de ne pas commettre la même erreur deux fois. Vous voilà prévenu.

Nous avons tout simplement été charmé par ces rencontres mondiales qui furent l'occasion de rencontrer ou retrouver des personnes impliquées, ouvertes, motivées, promptes à partager, bref, exceptionnelles à plus d'un titre.

**Auteurs : Denis Bodor & Fleur Brosseau**

## Offrez une grenouille à votre



*Il est assez facile de trouver dans le commerce des « stations météo » pas trop onéreuses. Trop facile ? En tous cas, pas très amusant. Quel plaisir pour un bricoleur que de monter soi-même sa propre station... Surtout si l'on pense à la puissance d'un système informatique. Hélas, les ordinateurs ne mesurent pas encore la pression atmosphérique. Vite, arrangeons cela !*

### 1 Le capteur

Pour réaliser un baromètre, il nous faut un capteur de pression ! Voici qui frise la lapalissade, mais le choix dudit capteur n'est pas aussi facile qu'il y paraît. Évidemment, tout le monde pense immédiatement aux populaires et très répandus MPX4115 (par exemple) de Freescale. Je pense que leur précision est insuffisante pour réaliser un baromètre. De plus, leur interface est analogique. Elle oblige donc la mise en place d'une chaîne de numérisation. Pourquoi ne pas trouver un composant plus adapté ?

Laissez-moi vous présenter le **MS5534** de la société Intersema [1]. Ce capteur de pression présente, à mon avis, plusieurs avantages. D'abord, son interface est numérique... pas besoin ni de convertisseur, ni de microcontrôleur pour l'interfacer simplement avec un PC (voir la partie suivante). De plus, il est très précis, puisque son erreur dans la plage de mesure courante est inférieure à 1hPa. Enfin, il est extrêmement petit (voir photo) et économe en énergie.



Le MS5534-BM à côté d'une pièce d'un centime d'euro

Détaillons un peu le fonctionnement de ce capteur. Tous les renseignements qui suivent sont tirés de la documentation [2]. La pression est mesurée par l'intermédiaire d'un capteur piezo-résistif. La tension obtenue est numérisée

par un convertisseur analogique-numérique intégré de grande précision. Le composant peut également mesurer et numériser la température du capteur. La lecture des 2 mots de 16 bits correspondants à ces 2 numérisations se fait grâce à une interface série synchrone. Le composant est calibré en usine ce qui permet d'obtenir six paramètres stockés dans une EPROM interne au capteur sous la forme de 4 mots de 16 bits. Ces paramètres, par calcul, donnent une compensation en température de la mesure effectuée. Tous les synoptiques de mesure et de calcul de la pression et de la température sont donnés dans la documentation, ainsi que le protocole de communication avec l'interface série du capteur. À la fin de l'acquisition et du calcul, les deux entiers représentent la température en dixièmes de degrés Celsius et la pression en dixièmes de millibars.

#### Note

Depuis que j'ai acheté et expérimenté ce composant, Intersema en a sorti une nouvelle version MS5534-C au fonctionnement identique et dont les différences essentielles sont une encore plus grande précision et une meilleure résistance aux décharges électrostatiques.

Le point délicat concerne l'achat de ce capteur... Parmi les distributeurs courants, je ne l'ai trouvé que chez Sélectronic et à un prix double de celui des MPX\* ! Heureusement, Intersema peut vous en vendre directement à titre d'échantillon, le problème étant que la commande se fait pour un nombre minimum d'unités. Organisez des commandes groupées dans les GUL ou entre amis électroniciens... Grâce à cette méthode, vous pourrez acquérir le capteur à environ 40% du prix affiché chez le revendeur cité plus haut (frais de port inclus). Ainsi, le capteur vous coûtera moins cher qu'un MPX\*. :-)

# pingouin !

## 2 Une première approche sur port parallèle

Voici un premier montage simple du point de vue électronique qui permet une mise en application immédiate du capteur. Tous les fichiers nécessaires pour cette partie sont rassemblés dans l'archive compressée **MS5534pp.tar.bz2** disponible au téléchargement sur le site de l'auteur [14]. Le contenu de l'archive est réparti en répertoires faisant référence aux différentes sous-parties suivantes.

### 2.1 Hardware

L'interface numérique du MS5534 nous permet de le brancher directement sur le port parallèle de notre PC. La société Intersema fournit un schéma type dans sa note d'application 505 [3].

L'alimentation du montage se fait par l'intermédiaire de 6 lignes de données (qui devront être portées au niveau haut par le logiciel). Il est donc autonome. Chacune de ces lignes est protégée par une diode, puis un circuit de régulation de tension avec une diode Zéner fournit une tension de 3V, tension de service du capteur. Cette alimentation doit impérativement être découplée par un condensateur au tantale de 47µF au plus proche de la broche **VDD** du capteur.

Le MS5534 a également besoin d'une horloge (convertisseur analogique-numérique oblige) d'une fréquence typique de 32,768 kHz sur la broche **MCLK**. Cette horloge se doit d'être à quartz, car la conversion est très sensible à l'instabilité de phase (*jitter* en anglais).

Des portes « inverseuses » sont utilisées pour assurer la translation des niveaux de tension entre les lignes du port parallèle et celles du capteur. L'interface de communication série est synchrone et utilise donc 3 lignes : **DIN**, **DOUT** et **SCLK**. L'horloge de transmission est fournie par la ligne 2 (**DATA0**) du port parallèle à la broche **SCLK**.

L'envoi des commandes du PC au capteur se fait par la ligne 3 (**DATA1**) à la broche **DIN** et la réception des données par la ligne 12 (**PAPER**) en provenance de la broche **DOUT**.

Je n'ai pas pu trouver d'horloge intégrée chez mon détaillant d'électronique. J'ai donc réalisé un circuit avec des portes et un quartz dont le schéma est représenté à la figure 1. L'intégralité du schéma se trouve, au format PostScript (imprimable), dans le fichier **ms5534pp.ps**. J'ai également changé la référence des portes (celles proposées sur le schéma d'origine n'existant qu'en boîtier CMS).

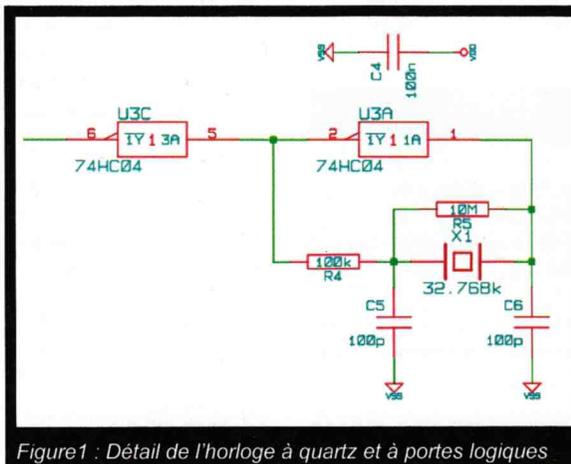


Figure 1 : Détail de l'horloge à quartz et à portes logiques

### Note

Les fichiers de CAO ont été réalisés avec **Kicad** [4].

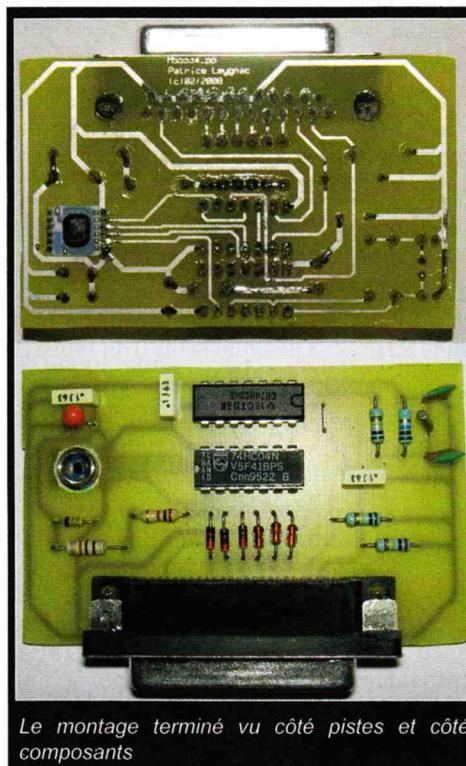
J'ai employé toute mon ingéniosité pour réaliser un circuit imprimé simple face avec le minimum de ponts (ce qui explique la répartition un peu tarabiscotée des inverseurs). Je vous propose le typon en résultant dans le fichier **ms5534pp-Copper.ps**.

Il n'y a que 2 ponts sur l'alimentation. Sa gravure peut paraître délicate au vu du peu d'écart existant entre les pastilles du capteur, mais c'est réalisable (personnellement, j'utilise la méthode à l'acide). Concernant le montage du capteur, j'ai opté pour un montage « à travers », car le trou à faire est rond et d'un diamètre de 7,5mm ; consultez la documentation du composant pour plus d'information.

Le soudage du capteur doit être fait précautionneusement. En plus de la finesse des pistes, il faut tenir compte de la sensibilité de ce genre de composant vis à vis de la chaleur et des décharges électrostatiques.

Enfin, le montage doit impérativement être mis dans un boîtier afin de protéger le capteur de la lumière, l'exactitude de la mesure en dépend.

Veillez quand même à ce que le boîtier ne soit pas étanche de manière à ce que la pression à l'intérieur suive la pression extérieure.



Le montage terminé vu côté pistes et côté composants

## 2.2 Software

Du côté commande, il faut piloter le port parallèle. Pas de problème ici, puisque nous avons déjà vu comment réaliser ceci grâce au module noyau **ppdev** [5]. La bibliothèque de pilotage du MS5534 décrite dans ce qui suit contient une classe C++. L'implémentation de cette classe s'appuie sur une autre classe, **SPC**, qui encapsule les instructions de dialogue avec **ppdev**.

### Note

La bibliothèque **SPC** est disponible sous licence **GNU/GPLv3** sur le site de l'auteur [14]. Pour l'installer, comme pour installer **MS5534pp**, vous avez besoin de **CMake**. Que ce soit pour l'une ou pour l'autre de ces bibliothèques, la construction et l'installation se font de la même manière : consultez le fichier **INSTALL**.

Expliquons l'utilisation de la classe **MS5534pp**.

Le constructeur, auquel on doit passer en paramètre le chemin complet vers le fichier spécial de périphérique attaché au port où est connecté notre montage, se charge de construire une instance de la classe **SPC** pour accéder à ce port.

La méthode **MS5534pp::open** se charge d'ouvrir ce port. Si l'ouverture réussit, elle exécute les instructions nécessaires pour alimenter le montage, provoquer un reset du capteur, récupérer les 4 mots de calibration, puis en extraire les 6 paramètres de calcul. À ce stade, nous sommes prêts à mesurer la pression.

La méthode **MS5534pp::close** assure la libération du port parallèle. Elle est systématiquement appelée par le destructeur de la classe.

La méthode **MS5534pp::acquire** demande au capteur de faire une acquisition de température et de pression, puis calcule celles-ci. Si on le souhaite, on peut opérer une correction supplémentaire dite du second ordre en utilisant la méthode **MS5534pp::order2**.

Étudions un peu l'implémentation de la classe dans le fichier **MS5534pp.cpp** :

L'horloge de la transmission série est produite en inversant à intervalles réguliers l'état de la ligne **2 (DATA0)** du port parallèle, par exemple ici (**DSCLK** désigne le bit 0 du registre **DATA**) :

```
240: port.highDataBit (DSCLK); // one duty cycle
241: // usleep (1000);
242: port.lowDataBit (DSCLK);
243: // usleep (1000);
```

Vous remarquerez sans doute que les lignes de code comportant les instructions **usleep** sont commentées. En environnement utilisateur sur PC, nous sommes loin du temps réel. Sans timing, sur mon PC (je l'avoue, un peu ancien), la fréquence ainsi générée est d'environ 1kHz. Si votre PC est suffisamment rapide pour générer une fréquence beaucoup plus élevée, vous aurez alors besoin de décommenter ces lignes et d'ajuster les valeurs des temporisations.

Les codes des commandes à envoyer au capteur sont enregistrés sous la forme de tableau d'octets : le premier élément contient le nombre de bits à transmettre pour la commande ; les éléments suivants contiennent le masque binaire permettant de positionner la ligne **3 (DATA1)** au niveau haut ou au niveau bas conformément au code inscrit dans la documentation.

La méthode privée **MS5534pp::out** envoie les séquences ainsi obtenues sur le port parallèle en respectant les chronogrammes indiqués par le constructeur : positionnement du bit sur le front descendant de **SCLK**, lecture par le capteur sur la broche **DIN** sur le front montant.

La méthode privée **MS5534pp::in** lit les bits sur le front descendant de l'horloge (ils sont positionnés par le capteur sur la broche **DOUT** sur le front montant de celle-ci). Les bits lus sont poussés progressivement dans un tampon de lecture de 16 bits.

Concernant la lecture des mots de pression et de température dans la méthode **MS5534pp::acquire**, après avoir envoyé la commande au capteur, il faut attendre que celui-ci fasse l'acquisition :

```
...
356: do // wait for acquisition
357: {
358: } while (port.isHighPaper ());
...
```

Le capteur nous informe que l'acquisition est terminée en passant sa ligne **DOUT** du niveau haut au niveau bas. Je reconnais que le code ci-dessus contient une faiblesse : il n'y a pas de **timeout** au cas où le capteur serait planté. Si vous pensez que c'est indispensable, je vous laisse le soin de l'implémenter sachant que la durée maximale de la conversion analogique-numérique est donnée à 35ms.

Pour connaître plus de détails sur l'implémentation que je propose, vous pouvez parcourir le code source. Il est abondamment commenté (en anglais). De plus, vous pouvez générer la documentation de l'API grâce à **Doxygen** en lançant la commande **make html** après configuration par **CMake**.

Le programme d'exemple **rPressure** fourni avec la bibliothèque **MS5534pp** est assez simple et illustre l'utilisation de la classe :

```
...
#include "MS5534pp.h"
...
int p, t;
MS5534pp *s;
...
s = new MS5534pp (argv[1]);
if (! s->open ())
{
    cerr << s->strerror () << endl;
    exit (EXIT_FAILURE);
}
if (! s->acquire (p, t))
{
    cerr << s->strerror () << endl;
    delete s;
    exit (EXIT_FAILURE);
}
s->order2 (p, t);
s->close ();
delete s;
cout << fixed << setprecision(1) << (float)p/10 << endl;
...
```

Le programme prend en paramètre sur la ligne de commande le chemin vers le fichier spécial de périphérique du port parallèle sur lequel est branché le capteur. Il renvoie sur la sortie standard la valeur de la pression avec une résolution de 0,1mbar, celle du capteur. Il est adapté à un relevé périodique grâce à **RDDtool** [6].

## 3 Soyons moderne !

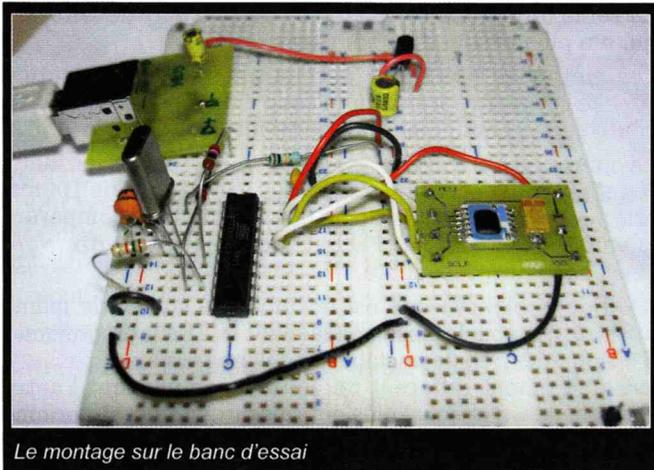
« Être moderne... bus USB ? Oui, bus USB ! Aïe, caramba ! » Eh oui, ça fait toujours ça au début. Mais vous allez voir, un bout du chemin a déjà été fait.

Donc l'objectif, vous l'aurez compris, est de réaliser un capteur de pression sur bus USB. Là, pas d'alternative, nous sommes obligés de passer par un microcontrôleur. À l'heure actuelle, l'une des solutions les plus simples est l'utilisation d'un microcontrôleur **Atmel AVR** et de l'implémentation software du protocole USB par la bibliothèque **USBtiny** [7]. Cette solution a été présentée dans ce même magazine par notre rédac-chef [8].

Comme pour la partie précédente, tous les fichiers nécessaires sont dans l'archive compressée **USBtinyMS5534.tar.bz2** téléchargeable sur le site de l'auteur [14].

### 3.1 Hardware et firmware

Le montage que je décris dans la suite s'appuie sur un microcontrôleur **AVR ATtiny2313**. Le protocole USB est supporté grâce à la bibliothèque **USBtiny**. De plus, le microcontrôleur va nous fournir l'horloge dont a besoin le convertisseur du capteur. Pour dialoguer avec le MS5534, nous allons utiliser l'interface série universelle (USI) de l'ATtiny2313.



Le montage sur le banc d'essai

Vous trouverez le schéma au format PostScript dans le fichier **hardware/USBtinyMS5534.ps**. Commençons par décrire l'alimentation. Le capteur doit être alimenté sous 3V. La norme USB pour les lignes de transmission est à 3,3V. En jetant un coup d'œil à ladite norme, nous pouvons constater que celle-ci tolère une tension minimum de 2,8V sur les lignes de transmission. La décision est prise : le microcontrôleur sera lui aussi alimenté sous 3V. Ainsi, nous n'aurons qu'une seule source pour tout le montage. Pour

Ce montage constitue une excellente présentation de notre composant. Bien que son interface ne soit pas très actuelle, il est parfaitement adapté à ceux d'entre vous qui ne sont ni motivés, ni équipés pour la programmation d'un microcontrôleur.

obtenir cette tension de 3V, nous utilisons un régulateur de tension **LP2950CZ3.0** connecté à l'alimentation du bus USB avec les capacités de découplage qui vont bien.

Le reste du circuit est fort simple. La broche **MCLK** du MS5534 est connectée à la broche **OC1B (PB4)** de l'AVR. L'horloge sera générée par le Timer1 en mode CTC. Pour la communication série, la broche **DOUT** du capteur est connectée à la broche **DI (PB5)**, la broche **DIN** à **DO (PB6)** et enfin **SCLK** à **USCK (PB7)**.

Vous trouverez le typon dans le fichier **USBtinyMS5534-Copper.ps**. Le circuit est simple face, sans aucun pont. Son format est adapté à un boîtier en plastique ABS bleu de dimensions 72 x 46 x 23 mm vendu par Électronique Diffusion.

### Remarque

Pour ceux d'entre vous qui souhaitent recompilier le *firmware*, vous devez d'abord télécharger la dernière version d'**USBtiny** et copier le répertoire **MS5534** contenu dans **firmware** dans celui contenant les sources de **USBtiny** obtenu après extraction de l'archive. Le firmware a été compilé avec la version 1.4 de la bibliothèque.

Le fichier source du firmware **main.c** est dans le sous-répertoire **firmware/MS5534**.

Avant de parcourir un peu son implémentation, je vais vous exposer le protocole de communication que j'ai choisi. Les requêtes USB sont au nombre de 4. Deux sont envoyées sur l'endpoint 0 OUT (transmission de l'hôte vers le périphérique). Ce sont les requêtes **MS5534\_RESET** et **MS5534\_ACQREQ** ; les autres sont envoyées sur l'endpoint 0 IN (transmission du périphérique vers l'hôte), **MS5534\_CALLLOAD** et **MS5534\_ACQLOAD**. Afin de ne pas mettre le bus USB en attente pendant l'acquisition, j'ai opté pour un découpage en 2 phases de celle-ci. D'abord, on envoie une requête permettant de commander au capteur de faire une acquisition de pression et de température (**MS5534\_ACQREQ**), puis, après avoir attendu le temps qu'il faut pour réaliser ces acquisitions (2 x 35ms), on envoie une seconde requête permettant de lire le résultat de la précédente (**MS5534\_ACQLOAD**). Pour que l'utilisateur puisse s'assurer que la valeur de l'acquisition a bien été mise à jour, j'ai implémenté un compteur d'acquisition : deux acquisitions successives ne doivent pas avoir un compteur identique. Par conséquent, la requête **MS5534\_ACQLOAD** renvoie 5 octets, les deux mots de 16 bits D1 et D2, plus l'octet du compteur. La requête **MS5534\_RESET** provoque l'envoi d'une commande reset au MS5534. Enfin, la requête **MS5534\_CALLAOD** est utilisée pour lire les 4 mots de calibration du capteur. Elle renvoie 8 octets.

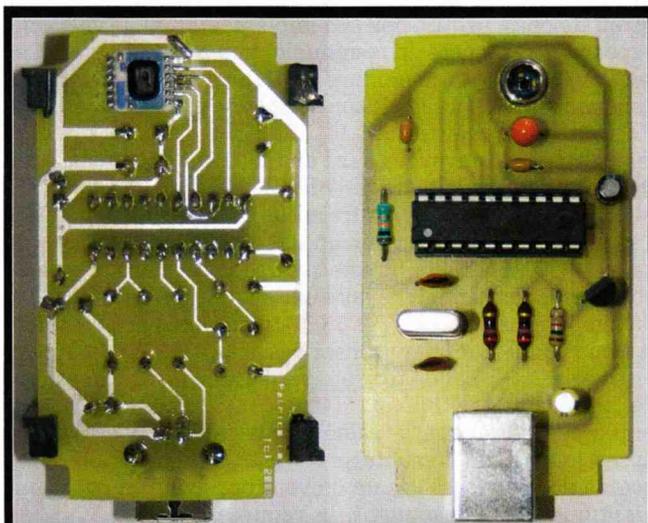
Nous allons suivre le déroulement de la fonction principale :

```

242: extern int main (void)
243: {
244:     MCLK_init ();
245:     USI_init ();
246:     usb_init ();
247:     MS5534_reset ();
248:     MS5534_readWs ();
249:     for ( ;; )
250:     {
251:         usb_poll ();
252:     }
253:     return (0);
254: }
    
```

Nous commençons par les initialisations. D'abord, concernant le Timer1 (fonction **MCLK\_init**) qui génère l'horloge **MCLK**, nous configurons celui-ci de manière à ce qu'il fournisse une horloge à une fréquence la plus proche possible de 32,768kHz. Compte tenu de l'horloge de base à 12 MHz, le mieux que nous puissions faire est 32,789 kHz... très bien. Nous configurons ensuite l'interface série universelle (**USI\_init**). Nous utilisons le mode à 3 fils (type SPI) et c'est le Timer0 qui fournira une horloge à 10 kHz pour effectuer la transmission entre le microcontrôleur et le capteur. Vient enfin l'initialisation de l'interface USB par l'appel de la fonction **usb\_init**. Pour terminer, nous nous assurons de l'état du MS5534 en provoquant un reset, puis nous récupérons les mots de calibration W1, W2, W3 et W4 qui sont stockés dans la mémoire du microcontrôleur. Après tout ceci, nous entrons dans la boucle principale qui traite les requêtes USB.

Revenons sur la transmission entre le microcontrôleur et le capteur. L'interface série du MS5534 est compatible avec la transmission SPI. Les séquences d'octets à envoyer pour les commandes sont fournies dans la note d'application 510 d'Intersema [9]. Elles tiennent compte des cycles d'horloge supplémentaires nécessaires lors des échanges de données. La lecture ou l'écriture sont initiées par les appels aux fonctions **USI\_starttr** et **USI\_startw**. La suite de la transmission est gérée par le vecteur d'interruption de l'interface USI.



Le montage terminé côté pistes et côté composants.

La programmation du microcontrôleur peut se faire grâce à **avrdude** :

```

$ avrdude -p t2313 -c usbtiny -U flash:w:main.hex:i \
-U hfuse:w:0xdf:m -U lfuse:w:0xef:m
    
```

## 3.2 Software

Passons au logiciel côté PC. Je vous propose d'une part un programme écrit en C et d'autre part, sur le même principe que dans la partie précédente, une bibliothèque en C++. Les deux utilisent la bibliothèque **libusb** [10] permettant de contrôler des périphériques USB depuis l'espace utilisateur.

Le programme en C est entièrement implémenté dans le fichier **main.c**. Son architecture n'est pas différente de celle proposée par Denis Bodor dans sa présentation d'USBtiny. Nous avons une fonction **usbOpenDevice** qui scanne le bus USB à la recherche de notre montage. J'ai juste rajouté un contrôle des chaînes renvoyées par le périphérique pour être sûr de son identité. En effet, le numéro d'identification de vendeur est celui dédié aux prototypes **0x6666**. Le montage ne doit bien sûr pas sortir de votre laboratoire.

```

106: if ((0 != strcmp (svend, USBTINY_VENDOR_NAME,
sizeof (USBTINY_VENDOR_NAME))) &&
(0 != strcmp (sprod, USBTINY_DEVICE_NAME,
sizeof (USBTINY_DEVICE_NAME))))
107: {
108:     usb_close (handle);
109:     handle = NULL;
110:     errorCode = USB_ERROR_MATCH;
111: }
    
```

Nous avons ensuite 4 fonctions. Elles adressent chacune une requête au montage :

- **MS5534\_reset**
- **MS5534\_calibration\_load**
- **MS5534\_request\_acquisition**
- **MS5534\_acquisition\_load**

La fonction principale appelle chacune de ces fonctions, puis calcule pression et température.

La bibliothèque C++ contient une classe **MS5534usb** dont l'interface est quasi identique à celle de la classe **MS5534pp**. La seule différence concerne le constructeur de la classe qui n'a plus besoin d'aucun paramètre.

Les méthodes publiques permettant la communication avec le capteur reprennent le code d'envoi de message de contrôle sur l'endpoint 0 du périphérique.

La méthode **MS5534usb::acquire** effectue les deux requêtes USB nécessaires séparées par un délai de 100ms. Elle vérifie que l'acquisition a bien été faite en comparant la valeur du compteur d'acquisition interne au firmware avec la valeur précédente.

L'API de la bibliothèque est documentée suivant la même méthode que pour la bibliothèque **MS5534pp**. Deux programmes d'exemple sont également fournis. Le programme **rPressure** est légèrement différent de celui accompagnant l'autre bibliothèque. Il effectue plusieurs mesures de suite (d'origine 20) affiche la moyenne de ces mesures. Ceci permet de s'affranchir du bruit numérique dû à la conversion analogique-numérique effectuée par le capteur.

## 3.3 Mesures et météo

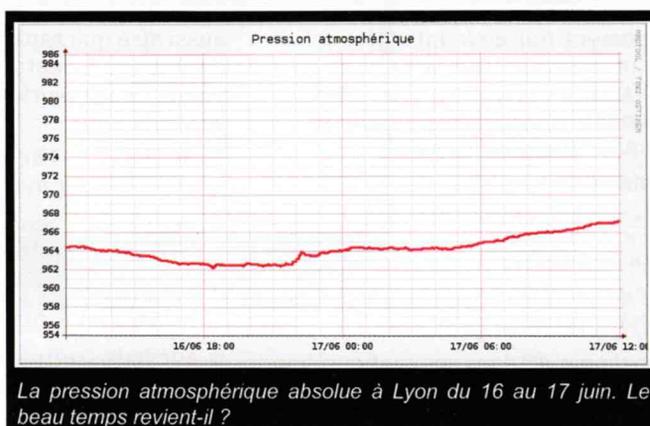
Si vous le permettez, nous allons terminer par un peu de physique (un tout petit peu).

Pour commencer, afin d'aider ceux d'entre vous qui ne sont pas familiers avec la pression, je vais faire les présentations.

La pression est une force exercée par unité de surface. Son unité internationale est le Pascal (Pa) en hommage au physicien, mathématicien et philosophe clermontois Blaise Pascal qui fut le premier à montrer l'existence même de la pression atmosphérique, ainsi que celle du vide [11]. 1 Pascal équivaut à la pression exercée par une force de 1 Newton sur une surface de 1 mètre carré (1 Newton, c'est le poids d'une masse d'environ 100 grammes).

La pression atmosphérique normale au niveau de la mer vaut 101325Pa ou bien, dans l'unité utilisée en météorologie, 1013hPa (hectoPascal, 1hPa = 100Pa). J'ai cité une autre unité : le bar. 1 bar vaut 100000 Pascal. Donc la pression atmosphérique vaut aussi 1013mbar (millibars). Cette pression atmosphérique, conformément à la définition donnée plus haut, correspond au poids de la colonne d'atmosphère au-dessus d'un mètre carré. Ça équivaut au poids d'une colonne d'eau de 10,3 mètres de haut ou encore à celui d'une colonne de mercure de 76 cm. Évidemment, la pression atmosphérique diminue lorsqu'on monte en altitude (la hauteur de la colonne d'air diminuant) [12]. La pression que nous mesurons avec notre capteur est la pression absolue. La pression qu'indiquent les météorologues sur leurs cartes est la pression relative au niveau de la mer. Si vous désirez enregistrer cette dernière, une bonne solution est de « calibrer » le logiciel en ajoutant le décalage observé par rapport aux cartes de météo [13].

Voici un exemple de relevé de pression absolue obtenu grâce à **RRDtool** [6].



La pression atmosphérique absolue à Lyon du 16 au 17 juin. Le beau temps revient-il ?

Terminons par quelques notions de météorologie. J'ai bien dit des « notions »... La météorologie est une science bien trop complexe pour prétendre la résumer en quelques lignes. Usuellement, on entend dire que si la pression relative est haute (supérieure à 1013hPa) le temps sera beau et si la pression est basse (inférieure à 1013hPa) le temps sera pluvieux. Ceci est une idée reçue. La pression actuelle indique le temps actuel.

Si l'on veut « prédire » le temps, il faut prendre en compte l'évolution de la pression. En effet, la répartition des zones de haute pression (anticyclones) et des zones de basse pression (dépressions) conditionne le déplacement des masses d'air et, par conséquent, indique les points de rencontre des masses d'air chaud humide et d'air froid sec donc les points où il va pleuvoir. L'évolution de la pression nous montre donc le déplacement des masses d'air, celles-ci allant des hautes pressions vers les basses pressions, le mouvement tourbillonnant observé sur les images satellites étant dû à la rotation de la Terre (forces de Coriolis). Si la pression augmente, l'anticyclone se rapproche. Les masses d'air se déplacent en « fuyant » le centre de l'anticyclone, il n'y aura pas de rencontre de masses d'air sous celui-ci, donc pas de pluie. Si la pression diminue, c'est la dépression qui se rapproche et donc la zone où se font les fameuses rencontres génératrices de pluie.

Pour être plus complet, il faut prendre en compte la vitesse de l'évolution. Si la pression varie de manière brutale, cela signifie qu'un épisode transitoire et peut-être violent est en approche. Par exemple, lorsqu'un orage arrive la pression diminue rapidement, plus de 2,5hPa par heure.

Voilà, j'espère que tout le monde a survécu et que j'ai été suffisamment clair.

## 4

## Conclusion

Arrivé au terme de cet article, nous avons réalisé deux montages pratiques (hors catégorie exemples) d'interfaçage avec le PC, l'un sur port parallèle, l'autre sur bus USB. Concernant le bus USB, il y a encore du travail pour qui veut s'en donner la peine : utilisation d'autres solutions hardware, mise au point d'un driver en espace noyau... Que de découvertes en perspective !

Enfin, notre ordinateur est muni d'un baromètre. Des solutions de mesures de température ont déjà été présentées dans d'autres articles.

Il nous manque encore un certain nombre d'équipements pour pouvoir transformer le PC en station de relevé météorologique, mais la voie est ouverte. Il ne reste plus qu'à faire preuve d'imagination et à se mettre au travail.

Auteur : Patrice Leygnac

## Liens et références

- [1] Site internet de la société Intersema : <http://www.intersema.com>
- [2] Datasheet du MS5534-B : <http://www.intersema.com/site/technical/files/ms5534b.pdf>
- [3] Note d'application 505 d'Intersema : <http://www.intersema.com/site/technical/files/an505.pdf>
- [4] Laboratoire du LIS à Grenoble, Kicad : [http://www.lis.inpg.fr/realise\\_au\\_lis/kicad/](http://www.lis.inpg.fr/realise_au_lis/kicad/)
- [5] BODOR (Denis) « Programmation du port parallèle », in *GNU/Linux Magazine*, hors-série numéro 23, page 24.
- [6] RRDtool, <http://oss.oetiker.ch/rrdtool/>
- [7] USBtiny, <http://www.xs4all.nl/~dicks/avr/usbtiny/index.html>
- [8] BODOR (Denis), « Création d'un périphérique USB avec supports GNU/Linux et Windows », in *GNU/Linux Magazine*, numéro 100, page 56.
- [9] Note d'application 510 d'Intersema : <http://www.intersema.com/site/technical/files/an510.pdf>
- [10] libusb, <http://libusb.wiki.sourceforge.net/>
- [11] Wikipédia « Blaise Pascal » : [http://fr.wikipedia.org/wiki/Blaise\\_Pascal](http://fr.wikipedia.org/wiki/Blaise_Pascal)
- [12] Wikipédia « Formule du nivellement barométrique », [http://fr.wikipedia.org/wiki/Formule\\_du\\_nivellement\\_barométrique](http://fr.wikipedia.org/wiki/Formule_du_nivellement_barométrique)
- [13] Météo France rubrique « Ma météo » : <http://www.meteofrance.com/FR/index.jsp>
- [14] Site de l'auteur, Arvernux : <http://www.arvernux.fr>

## Développez vos périphériques



*Le port USB est maintenant devenu le port (et le bus) par excellence, qu'il s'agisse de machines de type PC ou Mac. Malheureusement, le développement de périphériques simples pour ce type de bus est encore relativement coûteux et nécessite souvent des connaissances avancées sur le sujet. Lorsqu'on cherche plus de simplicité, la solution du port série reste une excellente alternative, d'autant plus lorsqu'on sait qu'un adaptateur USB/série permettra de faire évoluer sans mal votre périphérique.*

Le développement qui nous intéresse ici se basera sur un microcontrôleur de type AVR d'Atmel, l'ATtiny2313 en particulier. Le choix de ce composant est guidé par plusieurs caractéristiques intéressantes :

- 2 Ko de mémoire Flash ;
- 128 octets d'EEPROM ;
- 128 octets de SRAM ;
- Interface série *full duplex* (USART USI, *Universal Serial Interface*) ;
- 18 E/S (maximum, 13 dans une configuration série classique).

La quantité de mémoire Flash et de RAM permet le développement en C avec AVR-GCC, la version AVR du très classique compilateur GNU. Celui-ci couplé à la bibliothèque standard GNU permettra de rendre le développement

aussi aisé que rapide. Si vous êtes utilisateur Debian (ou Ubuntu), vous aurez besoin, pour passer de la théorie à la pratique, des paquets suivants :

- **gcc-avr** : *The GNU C compiler (cross compiler for avr)* ;
- **avr-libc** : *Standard C library for Atmel AVR development* ;
- **binutils-avr** : *Binary utilities supporting Atmel's AVR targets* ;
- **avrdude** : *software for programming Atmel AVR microcontrollers.*

Vous disposerez alors d'une suite de développement complète pour AVR que vous pourrez compléter avec **gdb-avr**, le débogueur GNU et **simulavr**, l'émulateur AVR qui malheureusement ne supporte pas encore l'ATtiny2313.

### 1 Communication série

Notre projet ne reposera pas sur le développement d'une application cliente. Si le sujet vous intéresse, je vous recommande la lecture des hors-série 23 (programmation du port série) et 27 (partie cliente pour l'afficheur uLCD). Dans le cas présent, nous nous limiterons à l'utilisation d'un outil comme Minicom ou GNU Screen pour communiquer avec notre périphérique.

Un point important à prendre en considération concernant ce qu'il est courant d'appeler le port série (norme RS-232 aussi appelée « EIA RS-232C » ou « V.24 ») concerne les tensions utilisées. Ces tensions sont normalisées et ne sont pas compatibles avec les tensions utilisées par un microcontrôleur comme l'AVR ou autre. Nous avons, en effet, en présence :

- un périphérique disposant d'une interface série où 0 logique = 0 volt et 1 logique = 5.0 volts (ou Vcc généralement) ;

- une interface RS-232 où 0 logique = entre +3 et +12 volts, et 1 logique = entre -3 et -12 volts.

Dans la pratique, sur les PC modernes, on trouve plus souvent des tensions entre -5 volts et +5 volts, ce qui reste dans les normes tout en évitant d'utiliser des convertisseurs de tension supplémentaires. Ceci ne pose généralement pas de problème lors d'une utilisation classique du port. Seuls les montages utilisant les plus fortes tensions à d'autres fins deviendront problématiques. C'est le cas, par exemple, du capteur infrarouge proposé par le projet Lirc ou du programmeur JDM pour les Microchip PIC 16F628/16F84a et compatibles.

L'interfaçage entre le port série et un microcontrôleur peut reposer sur des circuits spécialisés comme le MAX232 (conversion RS-232/TTL) ou le MAX3232 (RS-232 vers 3.3 volts). Cependant, la mise en œuvre d'un

MAX232 ou compatible nécessite l'utilisation de composants supplémentaires (condensateurs). Une solution plus économique est possible via l'utilisation de transistors NPN et de résistances. Celle-ci nous permettra, de plus, de diviser le montage en deux afin d'avoir, d'un côté un convertisseur RS-232/TTL polyvalent et de l'autre notre périphérique série.

La figure 1 détaille le montage permettant l'interface. Il utilise de simples transistors NPN en régime de commutation (ici des 2n2222, mais on peut également utiliser des BC337). Le principe de fonctionnement est

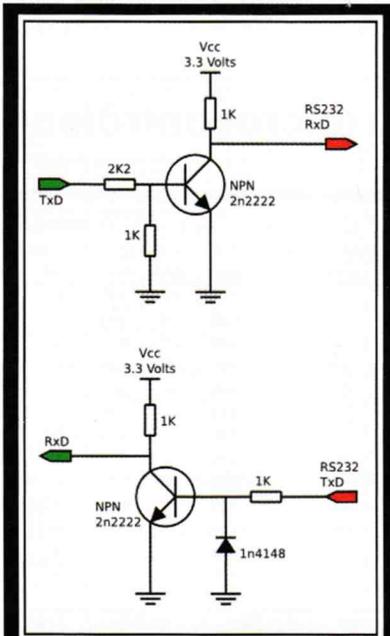


Figure 1 : Notre convertisseur RS-232 polyvalent

relativement simple. Dans le sens RS-232 vers périphérique, lorsque le transistor est non saturé, le périphérique voit Vcc (+5 volts) en entrée. La diode 1N4148 permet d'éviter de mettre une tension négative à la base du transistor (ce qui le détruirait à plus ou moins court terme) lorsque qu'un 1 logique se présente. Avec un 0 logique (tension positive), le 2n2222 est saturé et le collecteur présente une tension nulle par rapport à la masse.

Dans le sens contraire, périphérique vers RS232, le montage est encore plus simple. Lorsque RxD du périphérique est à la masse, le transistor n'est pas saturé et le collecteur présente une tension correspondant à Vcc par rapport à la masse. Lorsque le périphérique présente un 1 logique (+5 volts), le 2n2222 est saturé et le collecteur ne présente plus de tension par rapport à la masse. Notez que ce montage ne respecte pas les normes, puisque l'interface RS-232 attend une tension négative. Cependant, dans la pratique, une tension nulle est vue comme un 1 logique. C'est une liberté par rapport à la norme qui est, semble-t-il, déjà prise par les fabricants d'interfaces RS-232.

La polyvalence du montage nous permet de l'utiliser pour interfacer non seulement notre périphérique série, mais également n'importe quel autre périphérique ou système embarqué disposant d'un connecteur console en 5 ou 3.3 volts. Ceci est également applicable à des périphériques comme des routeurs ou même des machines « exotiques » comme une Silicon Graphics Indy.

## 2 Programmation de l'AVR

Nous avons déjà parlé de l'aspect logiciel du développement pour Atmel AVR. Penchons-nous maintenant sur le côté matériel. Une fois le code compilé et une image de la mémoire Flash du microcontrôleur obtenue avec les outils GNU, nous devons la copier dans la mémoire Flash du composant.

Le microcontrôleur ATtiny2313 se programme In Situ (ICSP). Cela signifie que le composant, dans le montage final, peut être programmé sans intervention mécanique (extraction du composant et insertion dans un programmeur). Pour nous, cela veut surtout dire que la seule chose à faire pour programmer le composant est de le mettre en situation comme dans un montage terminé.

La figure 2 donne le détail des connexions entre le microcontrôleur et le port parallèle d'un PC. Il vous faudra obligatoirement un vrai port parallèle et non un convertisseur parallèle/USB (qui en réalité est un adaptateur USB/imprimante).

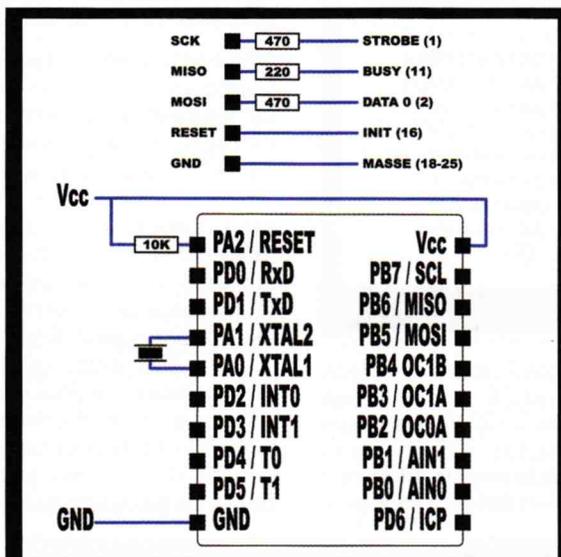


Figure 2 : Notre montage de base pour expérimentation. Notez, sur le schéma, la présence de la connectique permettant la programmation in situ via le port parallèle. Ce type d'adaptateur de programmation est connu sous le nom « dapa » dans bien des utilitaires comme AVRdude. Lors de la réalisation du circuit, pensez à prévoir des connecteurs sur la masse, et les broches RST, CLK, MISO et MOSI. Vous pourrez ainsi reprogrammer le microcontrôleur sans manipulation mécanique.

Habituellement, ce schéma est donné en deux parties. D'une part, le composant en situation avec les broches pour la programmation (MISO, MOSI, SCK, /RESET et Masse) et, de l'autre, l'adaptateur parallèle avec la correspondance entre les broches en question et les lignes du port parallèle.

C'est dans cette partie que se placent les résistances de protection. Celles-ci ne sont pas absolument nécessaires, mais mieux vaut être prudent. Quelques résistances ne sont pas cher payées pour un peu de sécurité.

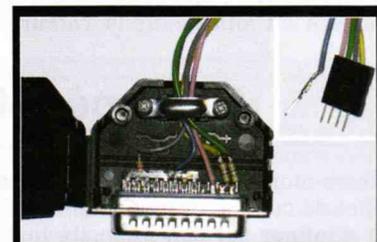


Figure 3 : L'adaptateur parallèle pour la programmation des AVR In Situ (ICSP)

La photo en figure 3 montre un adaptateur terminé. Les caches en plastique des connecteurs DB25 laissent suffisamment de place pour y loger les résistances ce qui nous permet d'obtenir quelque chose de compact et relativement propre (une fois le cache fermé).

Notez la présence de la résistance entre la ligne /RESET de l'ATtiny2313 et l'alimentation. Il s'agit d'une résistance de

rappel qui permet de mettre /RESET au +5V en l'absence de signal et ainsi ne PAS provoquer de reset. La documentation Atmel recommande l'utilisation d'une résistance supérieure ou égale à 4700 ohms.

Nous verrons plus loin la programmation de l'ATtiny2313 avec AVRdude.

## 3 Utilisation du port série du microcontrôleur

L'ATtiny2313 dispose de deux ports de communication sériels. L'USI ou *Universal Serial Interface* est une interface relativement basique. L'USART pour *Universal Synchronous and Asynchronous serial Receiver and Transmitter* propose une liste de fonctionnalités plus complète :

- Opérations en full duplex ;
- Opérations synchrones (ligne Clock) ou asynchrones ;
- Horloge haute vitesse ;
- Support de trames 5 à 9 bits de données avec 1 ou 2 bits de stop ;
- Vérification de parité matérielle.

L'utilisation de l'USART peut se faire avec l'horloge interne de l'ATtiny2313. Rappelons au passage, qu'en effet, la mise en œuvre du microcontrôleur ne nécessite pas d'autre composant que la résistance de rappel dont nous avons parlé plus haut. Cependant, afin d'obtenir un taux d'erreur nul, il devient impératif d'utiliser un quartz comme source pour l'horloge de l'AVR.

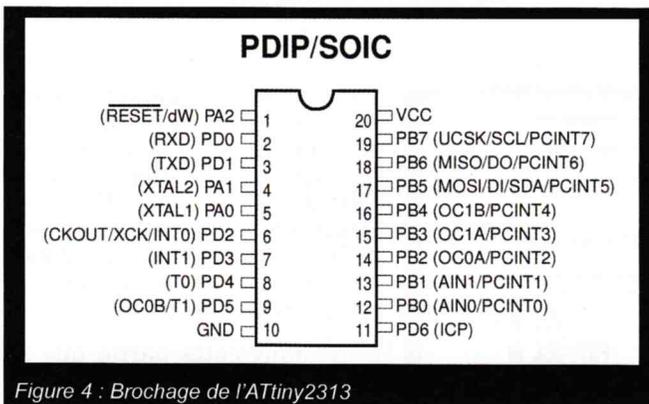


Figure 4 : Brochage de l'ATtiny2313

La figure 4 montre l'usage de chaque broche du microcontrôleur. L'alimentation se fait ici en +5 volts sur 20 avec la masse en 10. On connectera le quartz sur les broches 4 et 5 qui seront également reliées à la masse via des condensateurs céramique de 22pf. La configuration de la source d'horloge se fera en jouant sur la valeur des « fuses » (voir plus

## 4 Notre périphérique

Notre montage final sera relativement simple. Il s'agit en effet de connecter une simple LED sur le microcontrôleur et d'influer sur son intensité lumineuse en envoyant des ordres via le port série. Il s'agit là d'un simple montage didactique. L'étendue des possibilités est énorme. On peut imaginer, par exemple :

- un contrôle de relais ou d'opto-triac à des fins domotiques ;
- l'affichage sur un écran LCD texte ou graphique ;
- le contrôle de capteurs divers (température, pression, mouvement, etc.).

Baud Rate (bps)	$f_{osc} = 8.0000 \text{ MHz}$				$f_{osc} = 11.0592 \text{ MHz}$				$f_{osc} = 14.7456 \text{ MHz}$			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	207	0.2%	416	-0.1%	287	0.0%	575	0.0%	383	0.0%	767	0.0%
4800	103	0.2%	207	0.2%	143	0.0%	287	0.0%	191	0.0%	383	0.0%
9600	51	0.2%	103	0.2%	71	0.0%	143	0.0%	95	0.0%	191	0.0%
14.4k	34	-0.8%	68	0.6%	47	0.0%	95	0.0%	63	0.0%	127	0.0%
19.2k	25	0.2%	51	0.2%	35	0.0%	71	0.0%	47	0.0%	95	0.0%
28.8k	16	2.1%	34	-0.8%	23	0.0%	47	0.0%	31	0.0%	63	0.0%
38.4k	12	0.2%	25	0.2%	17	0.0%	35	0.0%	23	0.0%	47	0.0%
57.6k	8	-3.5%	16	2.1%	11	0.0%	23	0.0%	15	0.0%	31	0.0%
76.8k	6	-7.0%	12	0.2%	8	0.0%	17	0.0%	11	0.0%	23	0.0%
115.2k	3	8.5%	8	-3.5%	5	0.0%	11	0.0%	7	0.0%	15	0.0%
230.4k	1	8.5%	3	8.5%	2	0.0%	5	0.0%	3	0.0%	7	0.0%
250k	1	0.0%	3	0.0%	2	-7.8%	5	-7.8%	3	-7.8%	6	5.3%
0.5M	0	0.0%	1	0.0%	-	-	2	-7.8%	1	-7.8%	3	-7.8%
1M	-	-	0	0.0%	-	-	-	-	0	-7.8%	1	-7.8%
Max. (1)	0.5 Mbps		1 Mbps		691.2 kbps		1.3824 Mbps		921.6 kbps		1.8432 Mbps	

Figure 5 : Tableau issu de la datasheet. On détermine la valeur du registre de configuration en fonction du quartz utilisé et de la vitesse souhaitée.

loin dans l'article). La connexion entre l'ordinateur et le microcontrôleur se fera par les lignes 2 et 3 via notre adaptateur RS-232/TTL. À des fins de démonstration, nous utiliserons une LED connectée via une résistance de 330 ou 470 ohms sur la broche 15 (bit 3 du port B également utilisable en PWM sous la désignation OC1A).

Pour configurer la vitesse de transmission utilisée par notre périphérique, il faut se rendre à la page 138 de la *datasheet*. Là, figurent des tableaux indiquant la valeur correspondante du registre **UBRR**. Nous mettrons en œuvre un quartz de 14,7456 MHz. La valeur d'**UBRR** à utiliser pour un débit de 9600 bps est donc 95. En fonction du ou des quartz à votre disposition, vous déterminerez la valeur à utiliser grâce aux tableaux. Notez, encore une fois, qu'en limitant la vitesse de communication et en jonglant habilement du fer à souder pour correctement isoler la liaison (maximiser le rapport signal/bruit), vous pouvez utiliser l'horloge interne. Une autre solution consiste à utiliser des quartz d'une fréquence plus courante (1, 2, 8, 16 Mhz) et tolérer un certain taux d'erreur qui peut rester acceptable : 0,2 % en 9600 bps pour un quartz de 4 Mhz par exemple. Tout dépend du matériel à votre disposition et de l'utilisation finale du périphérique.

Le périphérique pourra tout aussi bien être connecté à un PC ou un Mac via le port série ou à un adaptateur USB, mais également à un système embarqué. Dans ce cas, il ne sera plus nécessaire, dans la plupart des cas, d'utiliser le convertisseur RS-232/TTL. On imagine alors de nouvelles possibilités puisqu'il devient, par exemple, possible d'imaginer un routeur Wifi capable de servir de centrale domotique.

La gestion de l'intensité lumineuse se fera en PWM ou, en bon français, en modulation de largeur d'impulsions. La question à laquelle répond la PWM est : comment jouer sur l'intensité lumineuse d'une LED ? Que ceux qui s'approprient à répondre qu'il suffit de jouer sur l'intensité du courant ou de la tension se retiennent. Une LED n'est pas une ampoule à filament. Au-delà d'une certaine intensité, la LED est détruite, en dessous de l'intensité conseillée, son bon fonctionnement n'est plus garanti (elle ne s'allume tout simplement pas ou la tension à ses bornes est hors normes).

La solution consiste donc à la faire clignoter de manière à utiliser la persistance rétinienne. Mais là encore, on peut se tromper. Ce n'est pas la fréquence de clignotement qui importe, mais un rapport de phase. On définit ainsi une fréquence constante et on module le rapport entre les périodes d'allumage et d'extinction. C'est la PWM, *Pulse Width Modulation* ou modulation d'impulsions en largeur. C'est l'alternance de longues phases en état allumé qui donne l'impression que la LED est plus lumineuse et non la vitesse d'alternance entre les états.

L'ATtiny2313, comme d'autres modèles de microcontrôleurs AVR, dispose en interne de fonctionnalités permettant d'utiliser la PWM sans avoir à l'implémenter. L'utilisation de la PWM est directement liée avec la configuration de certains registres du microcontrôleur. C'est la valeur donnée au comparateur OCR1A qui déterminera le rapport de phase. Nous l'avons vu dans le hors-série 23, cette configuration est déjà aisée en assembleur. Elle l'est bien plus encore en C.

Notre code source du *firmware* du périphérique débute, tout naturellement avec l'inclusion des fichiers d'en-tête classique :

```
#include <stdint.h>
#include <stdlib.h>
#include <stdio.h>
#include <avr/eeprom.h>
#include <avr/interrupt.h>
#include <avr/io.h>
#include <avr/pgmspace.h>
#include <avr/sleep.h>
#include <avr/wdt.h>
#include <util/delay.h>
```

Poursuivons avec la définition de quelques macros :

```
#define PWMDDR      DDRB
#define PWMOUT      PB3
#define HAVE_ADC    0
#define USART_RXC_vect  USART_RX_vect
#define MCUCSR      MCUSR
```

Enfin, nous définissons quelques variables globales :

```
int16_t pwm=0;
volatile char rxbuff;
```

La fonction `main` commence tout naturellement avec l'initialisation de l'USART avec l'appel à la fonction `InitUART(95)` :

```
void InitUART (unsigned char baudrate) {
    UBRR1 = baudrate;
    UCSRB = (1 << RXEN) | (1 << TXEN);
    UCSRC = (1 << UCSZ1) | (1 << UCSZ0);
}
```

Nous trouvons dans l'ordre :

- la configuration de la vitesse en fonction de la valeur présente dans le tableau dont nous avons parlé précédemment ;
- l'activation de l'USART en émission et réception ;
- la configuration du format de données : ici, 8 bits de données et un bit de stop.

Après l'initialisation du port série, nous passons à l'initialisation des registres permettant l'utilisation de la PWM :

```
TCCR1A = _BV(WGM10) | _BV(WGM11) | _BV(COM1A1) | _BV(COM1A0);
TCCR1B = _BV(CS10);
OCR1A = 0;
PWMDRR |= _BV(PWMOUT);
```

Notez l'initialisation préalable du registre `OCR1A` à zéro et l'activation de la ligne 3 du port B en sortie. Je ne vais pas détailler ici la fonction de chaque bit de chacun des registres. Tout est, en effet, parfaitement détaillé dans la datasheet.

Afin de simplifier les opérations et puisque nous disposons de beaucoup de mémoire flash du fait de la simplicité de notre code, nous allons définir un certain nombre de fonctions utilitaires. La première concerne l'utilisation de la PWM :

```
static void set_pwm(int16_t new) {
    if(new==pwm)
        return;
    if (new < 0)
        new = 0;
    else if (new > 1000)
        new = 1000;
    if(new>pwm) {
        while(pwm!=new) {
            pwm++;
            OCR1A = pwm;
            _delay_ms(10);
        }
    }
    if(new<pwm) {
        while(pwm!=new) {
            pwm--;
            OCR1A = pwm;
            _delay_ms(10);
        }
    }
    pwm=new;
}
```

Nous pourrions faire plus simple, mais nous concédons un peu de l'économie d'espace à l'aspect fini du montage. La transition d'une valeur de PWM à une autre se fera de manière fluide et en douceur. Nous ne définissons donc pas directement une nouvelle valeur pour `pwm`, mais incrémentons ou décrémentons lentement pour passer de l'ancienne valeur (`pwm`) à la nouvelle (`new`).

Les autres fonctions utilitaires concernent la gestion de la communication sérielle. La première, et la plus simple, traite de la réception d'un octet :

```
unsigned char ReceiveByte (void) {
    while (!(UCSRA & (1 << RXC)));
    return UDR;
}
```

Nous utilisons ici la méthode dite de « *polling* », à savoir, la surveillance continue d'un bit d'un registre permettant l'attente de données en entrée. Notez qu'il est également possible d'utiliser la méthode des interruptions. Le choix de l'une ou de l'autre méthode dépend de l'objet du montage. Ici, nous n'avons pas besoin d'interrompre le fonctionnement normal pour traiter l'arrivée de données. Notre microcontrôleur ne fait, de toutes façons, rien d'intéressant en dehors de ces événements.

L'envoi de données est un peu plus complexe. En effet, en réception, nous ne souhaitons traiter qu'un seul et unique octet. En envoi, nous voulons pouvoir, finalement, utiliser une fonction simple ressemblant à **printf**. Nous commençons donc par le bas avec la fonction de base la plus simple : celle permettant d'envoyer un caractère :

```
static void putchar(char c) {
    loop_until_bit_is_set(UCSRA, UDRE);
    UDR = c;
}
```

Tout comme avec la fonction précédente, nous utilisons la technique de polling, mais cette fois non pas pour nous assurer de l'arrivée de données, mais pour vérifier que le tampon d'envoi est vide. Ce n'est qu'à cette condition que nous placerons dans le registre **UDR** notre caractère. Construite sur **putchr()**, nous avons ensuite une fonction de plus haut niveau qui, elle, sera chargée d'envoyer une chaîne de caractères :

```
static void printstr_p(const char *s) {
    char c;
    for (c = pgm_read_byte(s); c; ++s,
         c = pgm_read_byte(s)) {
        if (c == '\n')
            putchar('\r');
        putchar(c);
    }
}
```

Cette fonction prend en argument un pointeur sur une chaîne placée dans la mémoire flash (la mémoire du programme lui-même). Nous récupérons cette chaîne avec **pgm\_read\_byte** octet par octet en utilisant la taille retournée par **pgm\_read\_byte**. Vous comprendrez plus loin pourquoi nous procédons ainsi. Il n'est pas utile de consommer le peu de SRAM dont nous disposons (128 octets contre 2048 de flash). Ce n'est pas critique ici, mais c'est une bonne habitude à prendre.

Enfin, nous arrivons à la fonction de plus haut niveau chargé de communiquer avec l'utilisateur ou l'application de l'autre côté du port série :

```
void TransmitByte (unsigned char data) {
    printstr_p(PSTR("\nRX [");
```

```
    putchar(data);
    printstr_p(PSTR("]\n"));
}
```

Notre fonction prend en argument un simple caractère. En réalité, nous allons retourner à l'utilisateur la valeur qu'il aura envoyée, formatée et décorée de quelques caractères supplémentaires. Pour ce faire, nous utilisons les fonctions **printstr\_p** et **putchr**. Notez l'utilisation de la macro **PSTR** permettant de stocker des données dans la mémoire du programme (voir **avr/pgmspace.h**). C'est le pointeur obtenu en retour qui est passé en argument de **printstr\_p**.

Toujours dans **main**, nous pouvons maintenant nous occuper de l'application elle-même, à commencer par le traditionnel message de mise sous tension :

```
printstr_p(PSTR("=== GO GO GO ===\n"));
```

Rien de bien grandiose ici, vous en conviendrez avec moi. Passons donc directement à la boucle principale :

```
while (1) {
    rxbuff=ReceiveByte();
    if(rxbuff) {
        TransmitByte(rxbuff);
        switch (rxbuff) {
            case '0':      set_pwm(0);           break;
            case '1':      set_pwm(111);         break;
            case '2':      set_pwm(222);         break;
            case '3':      set_pwm(333);         break;
            case '4':      set_pwm(444);         break;
            case '5':      set_pwm(555);         break;
            case '6':      set_pwm(666);         break;
            case '7':      set_pwm(777);         break;
            case '8':      set_pwm(888);         break;
            case '9':      set_pwm(999);         break;
        }
    }
}
```

La boucle infinie **while** consiste en la récupération d'un caractère via **ReceiveByte** et en sa retransmission via **TransmitByte**. Nous analysons ensuite la donnée dans une condition **switch/case** et l'utilisons pour influencer sur la PWM avec **set\_pwm**. Notre code est terminé.

## 5

## Makefile, compilation et programmation

Notre code est stocké dans le fichier **main.c**. Il ne nous reste qu'à écrire un **Makefile** pour faciliter la compilation et le chargement du binaire dans le microcontrôleur. Je reprends ici, une idée de Dick Streefland pour ses développements sur AVR. L'astuce consiste à décomposer le **Makefile** en une partie réutilisable et une partie configurable. Voici la première partie stockée dans un fichier **common.mk** :

```
CC = avr-gcc
CFLAGS = -Os -g -Wall -I.
ASFLAGS = -Os -g -Wall -I.
LDFLAGS = -g
MODULES = $(OBJECTS)
all: main.hex
clean:
    rm -f main.elf *.o tags *.sch~ gschem.log *.hex
lobber:clean
    rm -f main.hex
```

```
main.elf: $(MODULES)
$(LINK.o) -o $@ $(MODULES)
main.hex: main.elf
avr-objcopy -j .text -j .data -O ihex main.elf main.hex
main.o:
```

On retrouve tous les éléments et cibles classiques d'un **Makefile**, ainsi que les arguments et options génériques pour le compilateur, l'assembleur et l'éditeur de liens.

Le contenu de ce fichier n'est pas censé changer, mais sera directement utilisé pour tous vos développements à venir. Le fichier **Makefile** lui-même sera configuré pour l'ATtiny2313 :

```
TARGET_ARCH = -mmcu=attiny2313
OBJECTS = main.o
STACK = 32
```

```
FLASH = 2048
SRAM = 128
include common.mk
dprog:
  avrdude -p t2313 -c dapa -i 5 -U flash:w:main.hex
```

On retrouve le fichier objet à produire et à traiter, puis la taille de la pile, de la mémoire flash et de la SRAM. La cible **dprog** permet la programmation du composant.

Dans l'article présentant le microcontrôleur ATmega16 publié dans le hors-série 23, j'utilisais UISP. Depuis, je me suis tourné vers AVRdude aussi bien pour des raisons propres à son ergonomie générale que pour le nombre de composants supportés (dont le très sympathique ATtiny13). AVRdude est très intéressant d'un point de vue de l'évolutivité. Les montages permettant la programmation des composants ne sont pas pris en compte dans le code de l'outil, mais dans le fichier de configuration (`/etc/avrdude.conf`). Comprenez bien que je ne parle pas de la configuration du montage, mais bel et bien de la description de la manière de l'utiliser. Il en va de même pour les différents AVR supportés. Ainsi, si vous décidez, par exemple, de brocher différemment votre adaptateur parallèle, vous pourrez l'utiliser tel quel, sans même avoir à recompiler l'utilitaire. Un vrai bonheur.

La syntaxe de la commande **avrdude** est suffisamment claire. L'option **-t** permet de spécifier le composant sachant qu'AVRdude tentera de vérifier la signature de ce dernier et ne manquera pas de vous informer d'un éventuel problème. **-i** permet de spécifier un intervalle en microsecondes (valeur approximative bien sûr) entre chaque changement d'état. Cette option est très utile pour les adaptateurs sujets aux mauvais fonctionnements dus aux parasites (long câble parallèle, problème d'isolation, etc.). L'impact est négligeable sur le temps de programmation du microcontrôleur et on s'assure ainsi d'éviter de très désagréables surprises comme un fonctionnement aléatoire.

Nous utilisons l'ATtiny2313 avec un oscillateur à quartz comme source d'horloge. Il convient donc de configurer le composant de manière à ce qu'il l'utilise, ce qui n'est pas le cas par défaut (oscillateur interne). Pour ce faire, nous jouons sur les fuses, une série de bits stockés dans une zone particulière du composant. Ces fuses sont répartis en deux groupes, nous avons les hfuses (*high fuses*) permettant, par exemple de configurer le *watchdog*, la détection *brown out* (chute de tension d'alimentation) ou l'effacement de l'EEPROM lors de l'effacement de la mémoire flash. Puis, nous avons les lfuses (*low fuses*) concernant principalement le choix de la source d'horloge. Les 6 premiers bits permettent

de configurer cette source. **0b111111** permet de choisir un oscillateur quartz/céramique de 8 Mhz et supérieur. Les deux bits restant déterminent la division de l'horloge par 8 et le fait d'utiliser ou non la broche 6 pour ressortir ce signal.

La lecture des fuses se fait très simplement avec AVRdude en utilisant :

```
% avrdude -p t2313 -c dapa -i 8 \
-U hfuse:r::-b -U lfuse:r::-b
avrdude: reading hfuse memory:
Reading | ##### | 100% 0.00s
avrdude: writing output file "<stdout>"
0b11011111
avrdude: reading lfuse memory:
Reading | ##### | 100% 0.00s
avrdude: writing output file "<stdout>"
0b01100100
```

L'option **-U** permet des opérations sur des zones mémoires spécifiques. Dans le cas présent, **hfuse** et **lfuses**. L'opération est une lecture (**r** pour *read*), le fichier de sortie est STDOUT (-) et le format binaire (**b**). Les valeurs affichées ici sont celles par défaut du composant utilisant l'oscillateur interne.

Dans notre cas, nous voulons configurer l'oscillateur à quartz de plus de 8 Mhz. Nous devons donc écrire dans la zone mémoire **0b11111111**, soit **0xFF** en notation hexadécimale. Nous utilisons ainsi :

```
% avrdude -p t2313 -c dapa -i 8 -U lfuse:w:0xFF:m
avrdude: reading input file "0xFF"
avrdude: writing lfuse (1 bytes):
Writing | ##### | 100% 0.00s
avrdude: 1 bytes of lfuse written
avrdude: verifying lfuse memory against 0xFF:
avrdude: load data lfuse data from input file 0xFF:
avrdude: input file 0xFF contains 1 bytes
avrdude: reading on-chip lfuse data:
Reading | ##### | 100% 0.00s
avrdude: verifying ..
avrdude: 1 bytes of lfuse verified
```

Notre montage est terminé et il peut être utilisé. Connectez les lignes RX et TX via l'adaptateur RS-232/TTL et utiliser, par exemple, **screen /dev/ttyS0 9600** pour utiliser le premier port série du système. Coupez l'alimentation de l'AVR et reconnectez-le. Vous devez voir apparaître sur l'écran la chaîne **=== GO GO GO ===**. Utilisez ensuite les touches numériques pour envoyer des valeurs entre 0 et 9 au périphérique. La LED devrait alors varier d'intensité lumineuse tout en douceur.

## 6

## Conclusion

Le montage présenté ici est très simple. À vous d'aller plus loin et d'imaginer votre périphérique et son utilisation. Comme précisé en début d'article, la solution série est l'une des rares qui puisse être utilisée simplement et sur un grand nombre de systèmes en raison de l'utilisation d'adaptateurs USB. Autre avantage, vous n'êtes virtuellement pas limité en nombre de ports USB/série, ce qui ouvre également des perspectives très intéressantes. À vous de jouer à présent...

Auteur : Denis Bodor



Rédacteur en chef de GLMF. Utilisateur GNU/Linux depuis 1994. Randonneur du jardin magique.

## PIC et USB



L'objectif de cet article est de réaliser une carte de développement USB à l'aide d'un PIC 18F2550.

Le firmware fourni par Microchip n'étant pas sous licence GPL, nous nous sommes largement inspiré du programme PUF (Pic Usb Framework) du projet Vasco. Ce firmware fourni sous licence LGPL (pour le bootloader) permet de réaliser facilement un ensemble cohérent. Le compilateur SDCC a été utilisé pour écrire ce projet.

L'esprit de cette description est un peu différent de celui du projet, mais il permettra de se familiariser avec l'énorme travail déjà réalisé par les créateurs et nous l'espérons de le diffuser plus largement.

### 1 Le PIC 18F2550

Le 18F2550 est un microcontrôleur Microchip intégrant un module USB permettant de travailler en mode *full speed* ou *low speed*. Dans cette description, nous utiliserons le *transceiver* USB intégré pour travailler en mode *full speed*. Le 18F2550 est un chip 28 broches équipé de tous les périphériques standards sur les PIC (UART, I2C, 23 entrées/sorties).

La carte décrite dans cet article est une carte de développement destinée à être placée sur une platine d'essai type LABDEC permettant un test rapide des applications écrites. Elle a été inspirée par les précédentes réalisations que nous avons réalisées sur les microcontrôleurs de la série 18F (cf. [www.pictec.org](http://www.pictec.org)).

### 2 Le compilateur SDCC

SDCC est actuellement le seul compilateur *open source* pour PIC. Proche de C18 de Microchip. Sa mise en œuvre, ainsi que les

outils associés (GPUTILS) a été décrite dans *GNU/Linux Magazine* N°90.

### 3 Le projet VASCO

Le projet Vasco hébergé sur GFORGE (<http://gforge.enseeiht.fr/>) vise à réaliser un robot entièrement avec des outils *open source*. Le pilotage sera assuré par un périphérique de stockage réseau Linksys NSLU2 équipé

d'un *firmware* OpenWRT. Les cartes à PIC utilisant le *bootloader* décrit ici permettront la commande des capteurs et des actionneurs de l'ensemble.

### 4 Les outils utilisés

Outre SDCC et GPUTILS, nous avons utilisé KICAD pour réaliser les schémas de notre carte de test et le circuit imprimé. Pour le

reste, un simple éditeur de texte sera suffisant pour l'écriture des programmes.

## 5 Le schéma de la carte de test

La carte de test est on ne peut plus simple. Le 18F2550 a été équipé d'un oscillateur 20 Mhz. Un circuit de *reset* externe permet de réinitialiser le microcontrôleur par le poussoir SW1. L'alimentation 5V est prise sur le bus USB et une LED présence tension (Led1) signale l'activité de la carte.

Le *switch* SW2 permet de basculer du bootloader au programme de l'utilisateur. Ce passage est indiqué par la LED Led2.

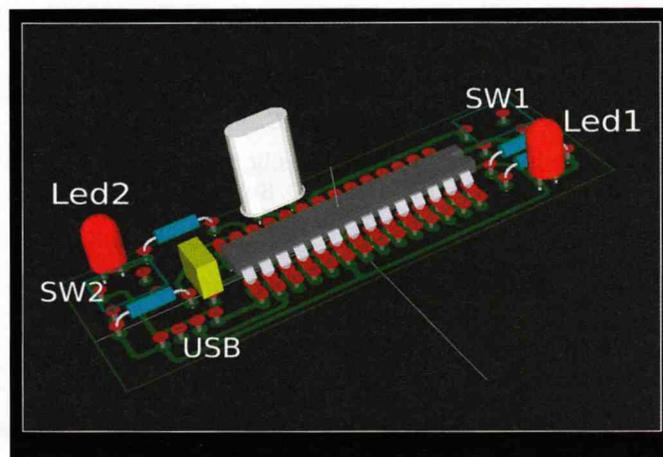
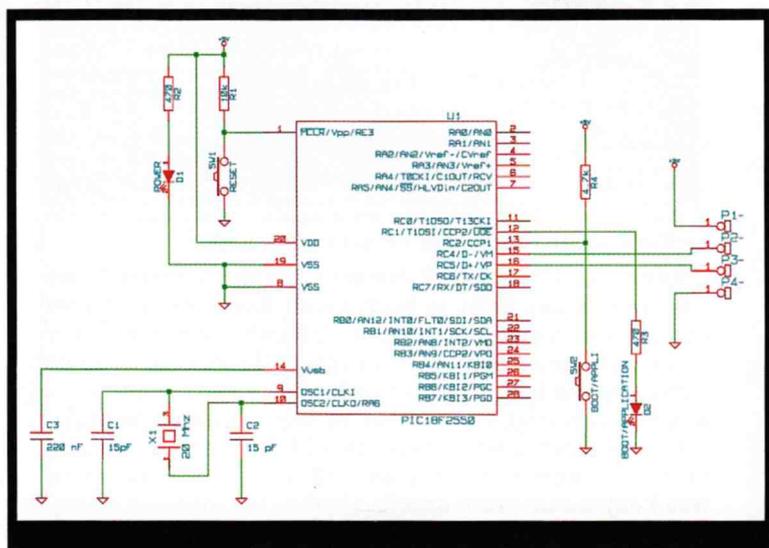
Le circuit imprimé a été réalisé en simple face et le câble USB a été soudé directement sur le circuit.

Il semble que les couleurs des câbles soient standardisées. Rouge +5Vcc, noir GND, jaune D-, vert D+.

Les fichiers sources des schémas, programmes et circuits imprimés de cet article peuvent être téléchargés à l'adresse suivante :

<http://www.hackinglab.org/linuxmag/>

Suivez ensuite l'arborescence pour récupérer les fichiers.



## 6. Pic USB framework

Nous allons maintenant nous intéresser au contenu du framework USB. L'ensemble des fichiers du projet peuvent être téléchargés à l'adresse suivante :

[http://gforge.enseeiht.fr/frs/?group\\_id=10](http://gforge.enseeiht.fr/frs/?group_id=10)

La dernière version à l'heure où nous écrivons ces lignes est la 1.1. Il faut, dans un premier temps, extraire le contenu du fichier **puf-1.1.tar.bz2**.

Voici l'arborescence de ce fichier :

PUF 1.1

- application
- archives
- bootloader
- include
- lib
- scripts
- udev

La particularité de PUF est d'intégrer l'ensemble des outils qui ont servi à son développement. Ainsi, dans le dossier **tools**, on trouve les outils `sdcc` et `gputils`, ainsi que les bibliothèques associées (pour les PIC uniquement). Les *makefiles* de chaque dossier ont été créés en respectant cette arborescence :

### ■ application

Le dossier application contient un exemple de mise en œuvre d'une application avec le bootloader PUF. Nous utiliserons cet exemple pour faire un premier programme après avoir installé le bootloader sur le 18F2550.

### ■ archives

Il s'agit des paquetages des outils ayant servi au développement de PUF.

### ■ bootloader

Les sources du bootloader écrit pour le 18F4550. Nous reviendrons sur les modifications à apporter pour utiliser le 18F2550.

## ■ include

Les fichiers communs au bootloader et aux applications développées autour du bootloader.

## ■ lib

La bibliothèque **libpuf.lib** est utilisée par les applications.

## ■ scripts

Une série de scripts pour le débogage.

## ■ udev

Ce dossier contient un fichier de règles permettant d'accéder au périphérique USB PUF sans être administrateur.

## 7

## Mise en œuvre du bootloader

Le bootloader a été écrit pour un 18F4550 dans une configuration un peu différente de celle que nous avons dessinée pour notre carte de test. Il s'agit de réaliser les adaptations nécessaires pour qu'il puisse fonctionner correctement avec notre hardware.

Pour cela, nous devons effectuer les modifications suivantes :

- adapter les bits de configuration au PIC 18F2550 ;
- adapter le bootloader aux LED de signalisation et aux boutons de commande ;
- adapter le makefile pour pouvoir recompiler le bootloader pour le 18F2550.

Dans le dossier bootloader, nous allons commencer par adapter la configuration du PIC.

Ouvrez le fichier **config.c**. Les lignes 27 à 76 décrivent la configuration du PIC 18F4550. Supprimez ces lignes et les remplacez-les par les lignes suivantes :

```
code char __at 0x300000 conf1 = 0x24;
code char __at 0x300001 conf2 = 0x0e;
code char __at 0x300002 conf3 = 0x3f;
code char __at 0x300003 conf4 = 0x1e;
code char __at 0x300005 conf5 = 0x81;
code char __at 0x300006 conf6 = 0x81;
code char __at 0x300008 conf7 = 0x0f;
code char __at 0x300009 conf8 = 0xc0;
code char __at 0x30000A conf9 = 0x0f;
code char __at 0x30000B conf10 = 0xa0;
code char __at 0x30000C conf11 = 0x0f;
code char __at 0x30000D conf12 = 0x40;
```

Le fichier **boot\_main.c** doit maintenant être adapté aux différentes modifications que nous avons apportées sur notre carte de test. Ouvrez le fichier et modifiez **init\_boot** et **main** suivant les indications ci-dessous :

```
void init_boot(void)
{
    static ulong count;
    TRISC = 0xFD;
    count = 0x80000;
    while(count)
    {
        count--;
    }
    PORTC = 0x00;

    // use boot descriptors
    device_descriptor = &boot_device_descriptor;
    configuration_descriptor = boot_configuration_descriptor;
    string_descriptor = boot_string_descriptor;
    // By default always use the bootloader vectors
    // because the device is started in configuration 0
    ep_init = boot_ep_init;
    ep_in = boot_ep_in;
    ep_out = boot_ep_out;
    ep_setup = boot_ep_setup;
}
```

```
void main(void)
{
    init_boot();
    init_usb();

    while(1)
    {
        usb_sleep();
        dispatch_usb_event();
        if((application_data.invalid == 0) && !PORTCbits.RC2)
        {
            PORTC=0x02;
            application_data.main();
            while (1);
        }
    }
}
```

Il est temps ici de détailler les modifications qui ont été apportées par rapport au projet PUF initial.

L'objectif de PUF est d'exploiter dans l'application les routines USB contenues dans le bootloader. Après de multiples essais, nous n'avons pu utiliser PUF dans ce cadre. L'idée a donc été de conserver la structure de PUF pour ne se servir que de la partie bootloader, mais sans avoir accès aux routines USB. Dans le cas où une utilisation de l'USB est souhaitable, il est toujours possible de recopier la partie nécessaire dans le programme utilisateur. Ce cas de figure fera l'objet d'un autre article.

Une dernière modification est à réaliser sur le fichier **usb\_descriptors.c**. Nous allons remplacer le **vendor\_id** et le **product\_id** par celui de Microchip. Ceci n'est nécessaire que pour permettre le fonctionnement de notre carte de test sous Windows.

```
33 EP0_BUFFER_SIZE, // Max packet size for EP0
34 0x04D8, // Vendor ID
35 0x000b, // Product ID
36 0x0001, // Device release number
in BCD format
```

Il est important de noter que, en production industrielle, le fabricant d'une carte USB devra obtenir un numéro de vendeur. Microchip fournit sur demande un **product\_id** vous permettant d'utiliser le **vendor\_id** de Microchip. En échange, vous vous engagez à acheter votre propre **vendor\_id** si votre production dépasse 10000 cartes...

Nous n'avons donc conservé de PUF que la partie bootloader et adapté le hardware à notre utilisation. Le fonctionnement devient le suivant :

- À la mise sous tension ou après un reset, le montage est en mode bootloader, avec la possibilité de transférer un programme dans la zone « application » à l'adresse 2000H.
- Le programme peut être transféré dans le PIC à l'aide de l'utilitaire Docker. Nous verrons son utilisation dans la partie « Application » de cet article.

- Une fois le programme transféré, le switch SW2 permet d'exécuter le programme de la zone application. La LED L2 est allumée.

PUF ayant été écrit pour un 18F4550, nous allons maintenant adapter le makefile pour un 18F2550. Pour cela, éditez le makefile situé dans le dossier **bootloader** et remplacez 18F4550 par 18F2550.

Une fois cette modification effectuée, il suffit de faire une copie du fichier **18f4550.lkr** et de la renommer en **18f2550.lkr**.

```
PIC_TYPE = PIC18F2550
sdcc_PIC_TYPE = 18f2550
lkr_PIC_TYPE = 18f2550
```

Compilez ensuite PUF en ouvrant une fenêtre terminal et en tapant, dans le dossier **bootloader**, **make**.

Le fichier **bootloader.hex** a été créé. Programmez ce fichier dans le PIC 18f2550 de la carte application en utilisant votre programmeur de PIC favori.

## 8 Test du bootloader

Raccordez votre carte de test sur un port USB libre du PC hôte, puis tapez la commande **lsusb**. Vous devez voir apparaître la carte de test avec l'identification vendeur Microchip.

```
Bus 003 Device 006: ID 04d8:000b Microchip Technology, Inc.
Bus 003 Device 005: ID 059b:027a Iomega Corp.
Bus 003 Device 003: ID 043d:00f7 Lexmark International, Inc.
Bus 003 Device 002: ID 05e3:0608 Genesys Logic, Inc.
Bus 003 Device 001: ID 0000:0000
Bus 002 Device 003: ID 045e:0083 Microsoft Corp. Basic
Optical Mouse
Bus 002 Device 001: ID 0000:0000
Bus 001 Device 001: ID 0000:0000
```

Dans notre cas, la carte apparaît sur le bus 003 en **device 006**. Nous avons volontairement mis l'identification vendeur de Microchip pour permettre à la carte d'être vue sous Linux, Mac OS X ou windows.

Cette particularité permet sous Windows d'utiliser le *driver* USB de Microchip. Malgré nos multiples tests, nous n'avons pu faire fonctionner sous Windows le driver HID générique.

L'étape suivante est celle du programme utilisateur. Pour cela, nous devons disposer de deux choses :

- Un programme de test compilé ;
- Un *downloader*.

Le downloader est un programme permettant de lire un fichier **hex** (celui du programme que vous aurez écrit) et de le transférer dans la zone application de notre carte de test.

Le downloader écrit dans le cadre du projet PUF est un programme en ligne de commande qui reçoit en paramètres le numéro d'identification vendeur de la carte de test et le nom du fichier à transférer.

Ce programme est disponible sur le site du projet PUF et s'appelle **docker**.

Le downloader se présente sous la forme d'un fichier compressé **docker-1.1.tar.bz2**.

Décompressez ce fichier dans un dossier, puis exécutez **./configure**.

Tapez ensuite **make**.

Dans le dossier **src (/docker-1.1/src)**, vous allez trouver l'exécutable **docker**.

En mode terminal, allez dans le dossier **src**, puis tapez :

```
sudo ./docker -v04d8 read test.hex
```

Le résultat sera le suivant :

```
Processing device 006
reading section [0, 1fff]
reading section [2000, 7fff]
reading section [300000, 30000d]
```

Nous avons fourni au downloader le numéro de vendeur du device USB (**04d8**), l'instruction **read** et le nom du fichier **hex** dans lequel il fallait placer le résultat de la lecture de la mémoire de notre PIC.

Le résultat est un fichier **test.hex** qui pourra être lu avec n'importe quel éditeur de texte.

## 9 Application

Le temps est venu de construire une application pour notre bootloader. Pour cela, nous allons reprendre la structure du dossier **application** situé dans le répertoire **puf-1.1**. Faites une copie du dossier application et appelez-la **myapplication**.

Allez ensuite dans le dossier **myapplication** et éditez le fichier **main.c**.

À partir de la ligne 30, supprimez les lignes originales et recopiez celles-ci :

```
unsigned long i;
void application_main(void)
{
    TRISB=0;
```

```
    while (1)
    {
        PORTB=0x01;
        for (i=0;i<100000;i++);
        PORTB=0x00;
        for (i=0;i<100000;i++);
    }
    /* Interrupt vectors */
    #pragma code high_priority_isr 0x2020
    void high_priority_isr(void) interrupt
    {
    }
    #pragma code low_priority_isr 0x4000
    void low_priority_isr(void) interrupt
    {
    }
```

Il s'agit là du traditionnel premier programme que l'on écrit pour un microcontrôleur : le programme qui permet de faire clignoter une LED !!

La LED est connectée sur le port B.0 à travers une résistance de 470 ohms. Il faut bien garder à l'esprit que bien que déjà « équipée » du bootloader, notre carte de test est censée être vide. On positionne donc le registre TRISB pour définir le port B en sortie, puis on fait deux boucles pour alternativement positionner le port à 1 ou à 0.

On voit que la structure du programme application est composé :

- d'un sous-programme `application_main()` qui est en fait l'équivalent du `main()` d'un programme C standard ;
- des routines d'interruption *remappées* en 2020H et 4000H du fait de l'utilisation de la mémoire basse du PIC par le bootloader.

Pour compiler ce programme, nous allons devoir modifier le makefile de ce dossier de la même manière que pour le bootloader. Remplacez les lignes de ce makefile contenant PIC18F4550 par 18F2550.

Faites ensuite une copie du fichier `18f4550.lkr` et renommez-la en `18f2550.lkr`.

Dans un terminal, tapez ensuite `make`.

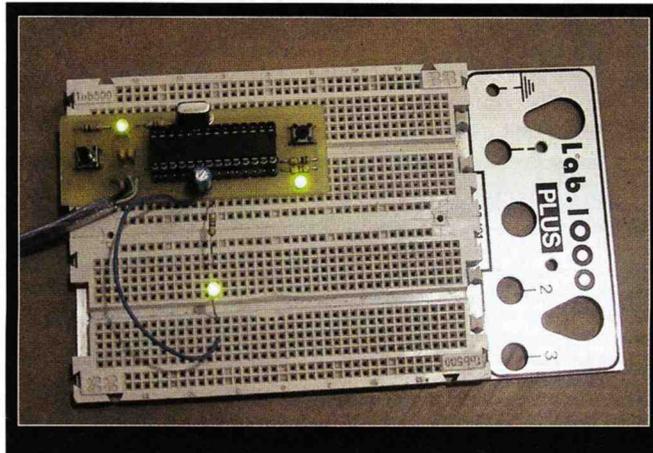
Vous disposez maintenant d'un fichier `application.hex` généré par `sdcc` et `gputils`.

Il suffit de le transférer avec `docker` dans la mémoire du PIC :

```
sudo ./docker -v04d8 write /home/mandon/PIC/puf-1.1/myapplication/application.hex
```

```
Processing device 006
erasing section [2000, 7fff]
writing section [2000, 7fff]
```

Appuyez sur le switch SW2 pour faire clignoter votre LED !!



## 10 Modifier les règles

L'accès aux périphériques USB non standards ne peut être fait qu'en tant qu'administrateur. Nous avons donc dû faire précéder nos commandes de `sudo` pour accéder à l'écriture ou à la lecture de notre carte de test.

Pour remédier à ce problème et permettre l'accès à l'ensemble des utilisateurs, il suffit d'ajouter une règle dans le répertoire `/etc/udev/rules.d`.

Modifiez le fichier `vasco.rules` situé dans le répertoire `/udev/` du projet PUF en adaptant le `vendor_id` et le `product_id` du périphérique USB :

```
# udev rules file for vasco
# $Id: vasco.rules,v 1.3 2006/11/11 16:18:19 gaufille Exp $
#
BUS!="usb", ACTION!="add", GOTO="vasco_rules_end"
# All Vasco devices
# Comment out if you are looking for product id rules
# and uncomment the next rule
SYSFS{idVendor}=="04d8", MODE="660"
# Vasco device with product ID #1
#SYSFS{idVendor}=="04d8", SYSFS{idProduct}=="000b", MODE="660",
LABEL="vasco_rules_end"
```

Votre carte de test sera maintenant accessible par tous les utilisateurs.

## 11 Conclusion

La documentation sur ce sujet, bien qu'abondante, ne nous avait pas permis de « faire démarrer » un PIC USB dans de bonnes conditions. PUF est un travail remarquable. Même si l'esprit dans lequel nous faisons fonctionner le bootloader est un mode dégradé de l'original, il permet toutefois de réaliser bon nombre d'applications à base de PIC sur des machines qui sont, pour la plupart, dépourvues de ports série. Il s'agit maintenant d'utiliser à fond l'USB en écrivant les lignes qui manquent pour l'USB. Ce sera l'objet d'un prochain article.

Auteur : JP Mandon

### Références et liens

- [www.pitect.org](http://www.pitect.org)
- [www.hackinglab.org](http://www.hackinglab.org)
- <http://vasco.gforge.enseiht.fr/>
- [www.microchip.com](http://www.microchip.com)

# L'ATtiny13, tout petit mais costaud

Lorsqu'on cherche à développer en C pour des microcontrôleurs, on en arrive souvent à utiliser des modèles surdimensionnés. En effet, l'utilisation du C impose des caractéristiques qui ne sont souvent disponibles que sur les « gros » microcontrôleurs. Mais, il y a des exceptions.

L'ATtiny13 d'Atmel est l'une de ces exceptions. En effet, il s'agit d'un microcontrôleur 8 bits disponible, entre autres, au format PDIP-8. Oui, vous avez bien lu, il s'agit d'un composant de 8 broches disposant des caractéristiques suivantes :

- oscillateur interne calibré 20 Mhz ;
- 1 Ko de mémoire Flash ;
- 64 octets d'EEPROM ;
- 64 octets de SRAM ;
- *watchdog* ;
- 6 lignes d'E/S (5 utilisables par défaut) ;
- programmation In Situ (ICSP) ;
- détection de chute de tension (*Brown out*) ;
- un compteur/timer 8 bits et deux canaux PWM ;
- un convertisseur analogique/numérique 4 voies 10 bits ;
- plusieurs sources d'interruption (internes et externes).

La présence des 64 bits de SRAM est la condition indispensable pour le développement C. Par opposition à l'ATtiny15, ce modèle peut donc être utilisé avec les outils et le compilateur GNU. La mise en œuvre elle-même et la programmation est simplissime. Une alimentation via la broche 8 (Vcc) et une résistance de rappel de quelques 10 kilohms entre Vcc et la broche 1 (*reset*) et le tour est joué. Il ne vous reste plus qu'à connecter l'adaptateur parallèle/ICSP (voir l'article sur le développement de périphériques série dans le présent magazine) et vous voilà prêt.

En guise de démonstration, nous allons développer une application très simple : utiliser l'ATtiny13 pour faire clignoter de manière aléatoire quelques LED : une utilisation parfaitement inutile si ce n'est pour son aspect purement décoratif. Comme nous souhaitons faire les choses correctement, voici le cahier des charges :

- une LED connectée sur la broche 5 qui variera d'intensité aléatoirement ;
- deux LED respectivement connectées sur 2 et 3 (ligne 3 et 4 du port B) clignotant alternativement en fonction de la valeur aléatoire utilisée pour régler l'intensité de la première LED.

Lorsqu'on parle de fonctionnement aléatoire, on entend, bien sûr, pseudo-aléatoire (voir l'article de Yann Guidon dans GLMF 81). Il existe plusieurs solutions permettant de développer des générateurs de nombres pseudo-aléatoires et la plupart d'entre elles sont portables sur un petit microcontrôleur. Cependant, pourquoi perdre du temps lorsque nous avons à notre disposition une bibliothèque C standard pour notre composant ?

Sans plus attendre, voici le code de notre application :

```
#define F_CPU 2000000UL
#include <stdint.h>
#include <stdlib.h>
#include <stdio.h>
#include <avr/EEPROM.h>
#include <avr/interrupt.h>
#include <avr/io.h>
#include <avr/pgmspace.h>
#include <avr/sleep.h>
#include <avr/wdt.h>
#include <util/delay.h>

#define PWMDDR    DDRB
#define PWMOUT    PB0
#define MCUCSR    MCUSR
#define DELAY     5

uint8_t mcucsr __attribute__((section(".noinit")));

int main (void) {
    uint16_t j;

    /* configuration timer/PWM */
    TCCR0A = _BV(WGM00) | _BV(WGM01) | _BV(COM0A1) | _BV(COM0A0);
    TCCR0B = _BV(CS00); OCR0A = 0;

    /* configuration des sorties */
    PWMDDR |= _BV(PWMOUT);
    DDRB |= _BV(PB4); DDRB |= _BV(PB3);

    /* initialisation PRNG */
    srand(42);

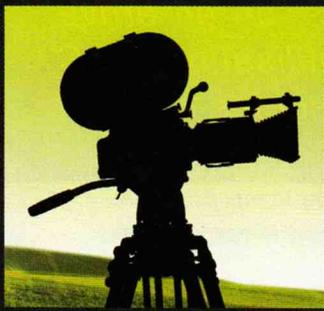
    while (1) {
        /* valeur aléatoire sur PWM */
        j = (rand()/128); // 128
        OCR0A = (uint8_t) j;
        /* alternance par seuil */
        if(j > RAND_MAX/256 ) {
            PORTB |= _BV(PB3); PORTB &= ~_BV(PB4);
        } else {
            PORTB |= _BV(PB4); PORTB &= ~_BV(PB3);
        }
        _delay_ms(100);
    }
}
```

Ce code sera compilé avec un **Makefile** identique à celui de l'article de la page 12 avec les modifications qui s'imposent concernant les caractéristiques du microcontrôleur :

```
TARGET_ARCH    = -mmcu=attiny13
STACK          = 32
FLASH         = 1024
SRAM          = 64
```

La programmation se fera également tout comme pour l'ATtiny2313 avec AVRdude auquel on précisera le modèle d'AVR (**-p t13**). Comme vous pouvez le voir, la simplicité n'a d'égal ici que la souplesse de développement. Concluons en précisant qu'un ATtiny13 se trouve, dans le commerce, pour moins de 2 euros. L'initiation au développement C pour microcontrôleur est véritablement à la portée de tous.

## Time lapse : la méthode quick



*Le time lapse est une technique qui consiste à filmer une séquence avec un framerate (nombre d'images par seconde) inférieur au framerate de lecture. L'effet ainsi obtenu sera une vidéo accélérée d'un évènement qui se déroule bien trop lentement pour être intéressant à regarder à sa vitesse réelle.*

Prenons un exemple concret comme la pousse d'une plante. Une vidéo d'une durée de 30 secondes, à vitesse réelle, serait très rapidement lassante, puisque, dans ce laps de temps, on ne peut observer aucun changement. La technique du *time lapse* consiste donc à capturer, par exemple, une image toutes les heures. Ainsi, en une journée, nous aurons 24 images. Partant du principe que la norme vidéo est de 24 images par seconde, en une journée nous totalisons donc 1 seconde de vidéo. Le calcul est vite fait, pour une vidéo d'une durée de 30 secondes, il nous faudra 30 jours de prise de vue à raison d'une photo par heure, ce qui nous fera un total de 720 photos.

Certains appareils photo numériques disposent d'une fonction « time lapse » native, mais ce n'est pas très répandu à ce jour et il y a de fortes chances que, tout comme le mien, votre appareil ne dispose pas de cette fonction. Ayant eu un besoin ponctuel de cette fonctionnalité, je me suis lancé dans la modification d'un Casio QV-R61 dont je n'avais plus l'utilité.

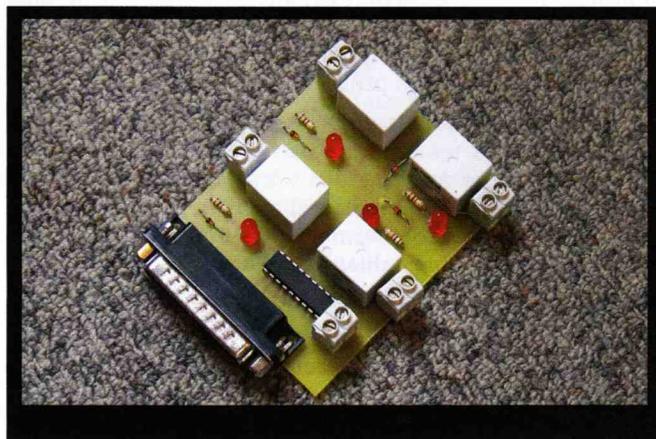
Les étapes pour prendre une photo sont assez simples. Il suffit d'allumer l'appareil, de faire la mise au point, de prendre la photo, puis simplement éteindre l'appareil. Ces étapes se font à l'aide de boutons pression sur l'appareil et, pour automatiser cette séquence, il me suffisait de déporter ces boutons sur des relais commandés par un PC via une carte relais sur port parallèle. Après démontage de l'appareil, le repérage s'est fait

à l'aide d'un multimètre. Cette partie peut être assez longue et demande beaucoup de patience et de précision. Une fois le repérage effectué, il faut maintenant souder les fils sur les soudures des boutons pression. Là encore, la précision et la minutie sont de rigueur et je vous déconseille cette étape après avoir avalé un demi-litre de café :)

Une fois les soudures effectuées, un petit point de colle sur chaque fil est conseillé pour limiter les risques de les arracher. Après s'être assuré que les contacts se font correctement, l'appareil peut être remonté et les fils reliés à la carte à relais.

Je ne m'attarderai pas sur la conception de la carte à relais, mais sachez que l'on en trouve sur Ebay pour une dizaine d'euros.

Les relais 1, 2, 3 et 4 sont reliés respectivement aux sorties du port parallèle D0, D1, D2, D3 et les fils d'allumage reliés au relais 1, les fils de mise au point reliés au relais 2, les fils de prise de vue reliés au relais 3.



## & dirty

La méthode se voulant *quick & dirty*, le contrôle des relais se fera en écrivant directement sur le port parallèle :

```
#include <stdio.h>
#include <unistd.h>
#include <asm/io.h>

#define ADRESSEPORT 0x378

int main(void)
{
    if(ioperm(ADRESSEPORT, 2, 1))
    {
        perror("Erreur de droits");
        exit(1);
    }

    /* On simule l'appui sur le bouton ON/OFF
     * de l'appareil pendant 1 seconde, on
     * relache puis on patiente 4 secondes
     * le temps que l'appareil photo soit
     * initialisé */

    outb(1, ADRESSEPORT);
    sleep(1);

    outb(0, ADRESSEPORT);
    sleep(4);

    /* On simule l'appui sur le bouton de
     * mise au point pendant 2 secondes */

    outb(2, ADRESSEPORT);
    sleep(2);

    /* On simule l'appui sur le bouton de
     * prise de vue tout en maintenant celui
     * de mise au point */

    outb(6, ADRESSEPORT);
    sleep(1);

    /* La photo est prise, on relache tout
     * et on patiente 5 secondes le temps que
     * la photo soit enregistrée sur la carte
     * mémoire */

    outb(0, ADRESSEPORT);
    sleep(5);

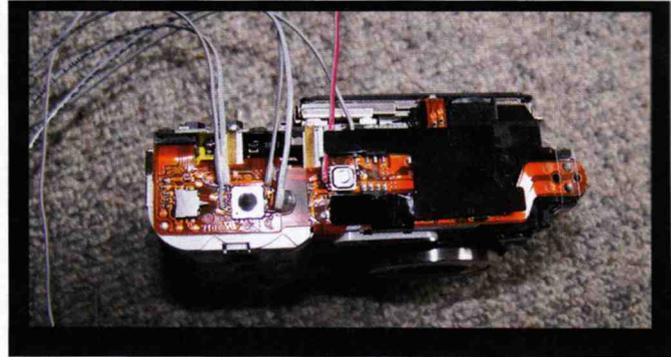
    /* On simule l'appui sur le bouton ON/OFF
     * de l'appareil pendant 1 seconde.
     * L'appareil photo s'éteint */

    outb(1, ADRESSEPORT);
    sleep(1);

    outb(0, ADRESSEPORT);

    if(ioperm(ADRESSEPORT, 2, 0))
    {
        perror("Erreur de fermeture");
        exit(1);
    }
}
```

Les commentaires dans le code source détaillent la séquence, mais, comme vous pouvez le constater, ce n'est rien de bien complexe.



Après s'être assuré que la séquence fonctionne correctement, vous pouvez automatiser celle-ci via le *cron*, à la fréquence que vous souhaitez (dans notre exemple, cette tâche sera exécutée toutes les minutes).

Après avoir laissé tourner un certain temps votre installation, il est temps de vider la carte mémoire.

Une fois toutes les photos récupérées, vous pouvez, si vous le souhaitez, les recadrer, puis il va maintenant falloir les convertir en vidéo. Pour cela, j'ai utilisé Mencoder, mais ffmpeg ou tout autre logiciel pourra remplir cette tâche :

```
mencoder "mf://*.JPG" -fps 24 -ovc lavc -lavcopts
vcodec=mjpeg -o video-timelapse.avi
```

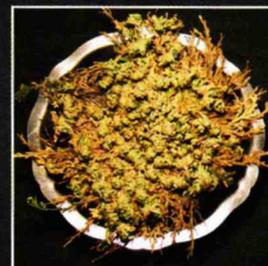
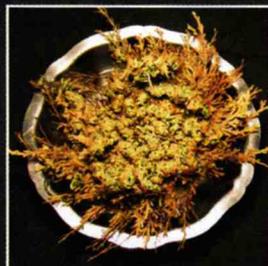
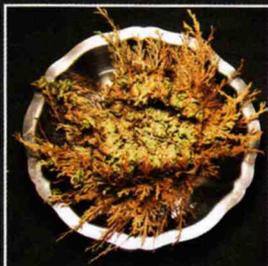
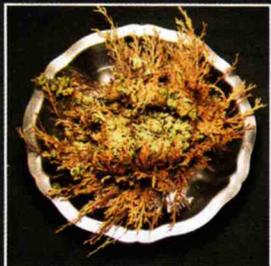
Et voilà ! Vous disposez maintenant de votre vidéo timelapse.

Pour avoir une idée du résultat obtenu, vous pouvez vous rendre à cette adresse : <http://www.dailymotion.com/video/x5ubqlq>.

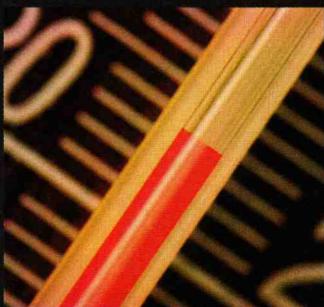
Une petite précision tout de même quant à la carte à relais. Comme vous pouvez le constater, je n'utilise que 3 relais sur les 4 disponibles. Le 4ème pourra par exemple servir à commuter un éclairage lors de la prise de photo, afin d'avoir la même intensité lumineuse pour chaque photo et ainsi éviter les scintillements dans la vidéo. On peut aussi imaginer pouvoir commuter la liaison USB entre l'appareil photo et le PC, afin d'automatiser la récupération de photos « à chaud ». Il serait ainsi possible de prendre des clichés pendant une très longue période tout en ayant une carte mémoire de faible capacité.

Il ne vous reste maintenant plus qu'à vous équiper d'un vieil appareil photo, de votre fer à souder, et de patienter jusqu'au prochain coucher de soleil !)

Auteur : Rodolphe Rudler



# Étude et réalisation d'un



*Cet article aborde la réalisation d'un capteur sans fil ZigBee permettant de récupérer la température du capteur distant. Le contexte de cette réalisation sera précisé. Au-delà de l'aspect didactique de la réalisation, ce travail s'inscrit dans le cadre du projet HomeSIP qui sera brièvement présenté. Toutes les étapes de réalisation du capteur sans fil seront décrites : de la conception de la carte jusqu'au logiciel embarqué. Enfin, l'intégration de ce capteur sans fil ZigBee dans le projet HomeSIP sera expliquée.*

## 1

### Contexte de la réalisation : le projet HomeSIP

HomeSIP est l'acronyme de « *Home Automation with SIP* ». Il s'agit d'un projet domotique basé sur la mise en œuvre du protocole SIP (Session Initiation Protocol) utilisé généralement pour la téléphonie sur Internet VoIP (Voice Over IP). Le projet HomeSIP a été présenté dans le numéro hors-série 25 de *GNU/Linux Magazine*.

L'idée est donc d'utiliser SIP pour collecter des informations provenant de différents capteurs et de piloter aussi différents équipements (actionneurs).

Le projet HomeSIP consiste donc :

- En une infrastructure matérielle composée de capteurs et d'actionneurs connectés à des systèmes embarqués qui sont communicants et qui possèdent tous une connectivité IP.
- À mettre en place les logiciels adéquats dans les systèmes embarqués sous Linux.
- À développer des langages dédiés de type DSL (*Domain Specific Language*) pour développer de nouveaux services autour de la plateforme.

Le projet HomeSIP est par nature un projet de domotique, la domotique étant en fait une composante du M2M.

Le M2M (*Machine to Machine*) est une infrastructure basée autour d'un réseau qui autorise des communications directement entre équipements ou via un serveur central sans aucune intervention humaine. Cela permet, par exemple, la capture automatique de données et leur traitement par les équipements après transmission.

Ce marché émergent est, en fait, l'évolution naturelle de la mise en place de la connectivité réseau et, en particulier, la connectivité IP dans les équipements électroniques.

Dans le cadre du projet HomeSIP, différents capteurs ont été mis en œuvre : détection de mouvement, RFID, température, X10... Un capteur de température sans fil a été spécifiquement réalisé pour être ensuite intégré de façon opérationnelle dans le projet. La suite de cet article va donc présenter cette réalisation...

## 2

### Le protocole ZigBee : une introduction

#### 2.1

#### Introduction

Qu'est-ce que ZigBee ?

ZigBee est un standard de communication sans fil à bas coût pour échanger des données issues d'équipements sans fil simples et faible consommation dans le milieu industriel.

Avec la convergence de l'informatique, de l'électronique et des télécommunications, il est ainsi possible de mettre en place des réseaux de capteurs sans fil dans un contexte domotique ou de contrôle industriel.

Il est clair qu'il existe un marché que ZigBee permet de combler : surveillance de locaux pour la détection de départ de feu, surveillance

# capteur sans fil ZigBee

de bâtiments contre les intrusions, Gestion Technique de Bâtiment (GTB), aide à la personne, assistance aux personnes...

Ce standard est promu par l'alliance ZigBee qui regroupe un ensemble d'industriels travaillant sur l'élaboration de spécifications afin de pouvoir développer des applications sans fil bon marché, faible consommation et sécurisées.

Comme tout standard, ZigBee permet à l'utilisateur final de cette technologie d'être constructeur indépendant ; ce qui lui permet d'acheter ses modules compatibles ZigBee auprès de n'importe quel fournisseur.

La norme ZigBee s'appuie sur la norme IEEE 202.15.4. Elle est librement téléchargeable sur le site de l'Alliance ZigBee : <http://www.zigbee.org/>.

Nous en détaillerons les grandes lignes et le lecteur pourra se référer à la norme pour plus de détails techniques...

## 2.2

### Architecture protocolaire de ZigBee

L'architecture protocolaire est donnée par la figure 1.

ZigBee s'appuie sur la norme IEEE 802.15.4 pour les niveaux physique et MAC (*Medium Access*).

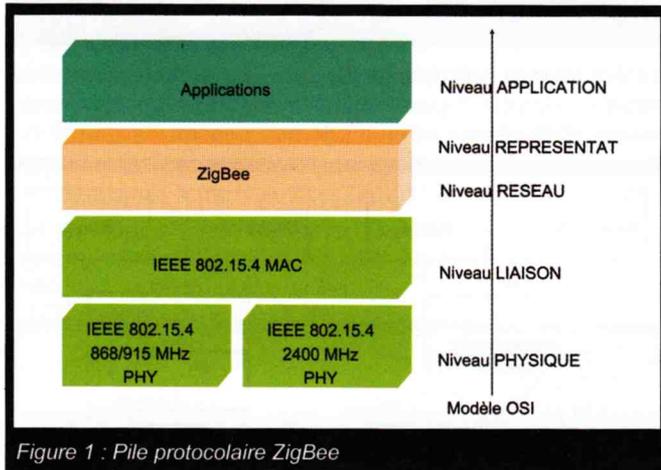


Figure 1 : Pile protocolaire ZigBee

La norme IEEE 802.15.4 a les caractéristiques générales suivantes :

- 3 bandes de fréquence de fonctionnement : 868 MHz (1 canal), 915 MHz (10 canaux) et 2,4 GHz (16 canaux).
- Débit de 20 kb/s, 40 kb/s et 250 kb/s suivant la fréquence.
- Méthode d'accès au support de type CSMA-CA (*Carrier Sense Multiple Access - Collision Avoidance*).
- Protocole fiable avec acquittement.
- Faible consommation (alimentation sur pile de type AA).

La norme ZigBee apporte en plus les éléments suivants :

- Au niveau réseau de la couche ZigBee, topologie de type point à point (direct), en étoile, en *cluster* ou maillé (*mesh*).
- Au niveau représentation de la couche ZigBee, la sécurité avec l'emploi optionnel d'un chiffrement AES 128 des données.
- Au niveau application, la définition de profils d'utilisation : exemple profil domotique HA (*Home Automation*).

## 2.3

### Couche physique IEEE 802.15.4

La norme IEEE 802.15.4 supporte les 3 bandes ISM (*Industrial, Scientific, Medical*) de 868 MHz, 915 MHz et 2,4 GHz.

Bande	Usage	Débit	Nombre de canaux	
2.4 GHz	ISM	Mondial	250kb/s	16
915 MHz	ISM	Amérique	40kb/s	10
868 MHz	ISM	Europe	20kb/s	1

Figure 2 : Bandes de fréquence pour ZigBee

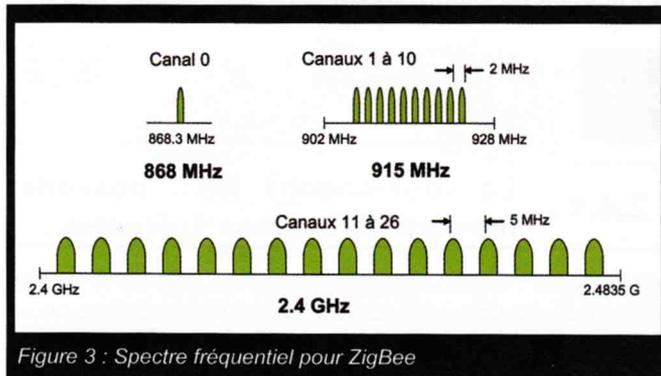


Figure 3 : Spectre fréquentiel pour ZigBee

Pour la bande 2,4 GHz, on a les caractéristiques générales suivantes :

- Débit binaire de 250 kb/s.
- 4 bits par symbole soit un débit symbole de 62.5 kbaud.
- Modulation orthogonale O-QPSK avec 16 symboles.
- Correspondance à chaque symbole d'une séquence d'étalement « 32 chips » pour un étalement DSSS, soit 2 Mchip/s.

Pour la bande 915 MHz, on a les caractéristiques générales suivantes :

- Débit binaire de 40 kb/s.
- 1 bit par symbole, soit un débit symbole de 40 kbaud.
- Modulation BPSK.

- Correspondance à chaque symbole d'une séquence « 15 chips » pour un étalement DSSS, soit 600 kchip/s.

Pour la bande 868 MHz, on a les caractéristiques générales suivantes :

- Débit binaire de 20 kb/s.
- 1 bit par symbole, soit un débit symbole de 20 kBaud.
- Modulation BPSK.
- Correspondance à chaque symbole d'une séquence « 15 chips » pour un étalement DSSS, soit 300 kchip/s.

La structure du « paquet » de données émis/reçu est donnée Figure 4.

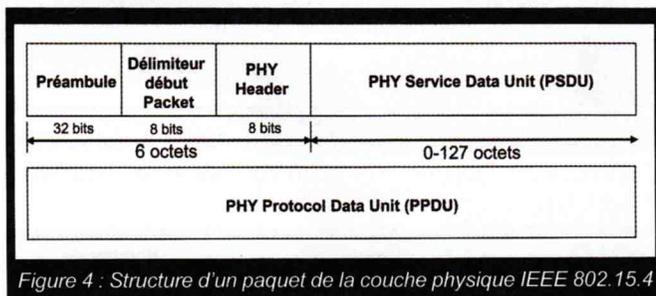


Figure 4 : Structure d'un paquet de la couche physique IEEE 802.15.4

Le niveau physique gère les fonctionnalités suivantes :

- Activation/désactivation de l'interface radio.
- Détection de l'énergie dans le canal. Choix du canal radio.
- Qualité du lien radio.
- Évaluation du canal pour la mise en œuvre du protocole d'accès CSMA.
- Émission/réception des paquets dans le canal radio.

## 2.4

## Sous-couche MAC IEEE 802.15.4

### 2.4.1

### La sous-couche MAC possède les caractéristiques suivantes :

- Mise en œuvre de trames avec un adressage IEEE 64 bits ou un adressage court sur 16 bits, soit 65536 équipements adressables au plus dans ce dernier cas.
- Structure de trame simple.
- Mode de transmission fiable.
- Gestion de l'association/déassociation d'équipements.
- 3 niveaux de sécurité possibles sur les communications : aucun, liste ACL, chiffrement AES 128 de la charge (payload) de la trame.
- Mode de transfert de type *half duplex*.

Deux mécanismes d'accès au réseau sont possibles :

- Réseau en mode non *beacon* (balise) : on utilise des communications avec une politique d'accès au médium de type CSMA-CA. Les trames de données reçues correctement sont acquittées.
- Réseau en mode *beacon* : un coordinateur du réseau (voir après) émet périodiquement des trames beacon (de 15 ms jusqu'à 252 s) délimitant ainsi une supertrame.

Le format de la supertrame est fixé par le coordinateur. Elle se compose de 16 intervalles de temps. Cela permet ainsi de synchroniser tous les équipements (nœuds) du réseau. Un équipement pourra émettre ses données dans un intervalle de temps donné pendant la durée de l'intervalle de temps. L'accès aux premiers intervalles de temps est non garanti donc soumis à contention (*Contention Access Period*), alors que sur les derniers, il est garanti (*Contention Free Period*), ce qui permet d'assurer un certain déterminisme et une certaine qualité de service.

4 trames sont définies :

- Trame de données (*Data Frame*) : transfert de données.
- Trame d'acquiescement (*Acknowledgment Frame*) : confirmation de données bien reçues.
- Trame beacon (*Beacon Frame*) : émise par le coordinateur de réseau en mode beacon.
- Trame de commande MAC (*MAC Command Frame*) : pour le contrôle des nœuds.

La structure de la trame de données est donnée Figure 5.

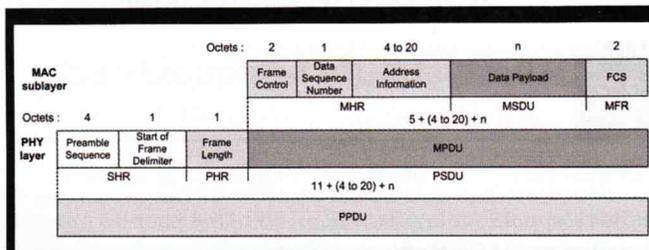


Figure 5 : Structure d'une trame de la sous-couche MAC IEEE 802.15.4

Un échange d'une trame de données à l'initiative d'un nœud est donnée Figure 6 dans le cas d'un réseau en mode beacon d'un réseau et en mode non beacon.

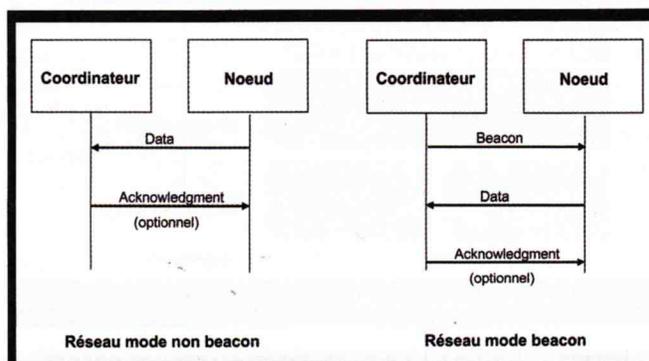


Figure 6 : Échange d'une trame de données

Enfin, la norme IEEE 802.15.4 définit 3 types d'équipements (nœuds du réseau) :

- Le coordinateur du réseau.
- L'équipement à fonctionnalités complètes FFD (*Full Function Device*).
- L'équipement à fonctionnalités réduites RFD (*Reduced Function Device*).

L'équipement FFD peut être soit un coordinateur, soit un routeur, soit un équipement terminal (capteur).

L'équipement RFD est un équipement simplifié comme un équipement terminal (*End Device*) muni de capteurs.

Pour communiquer au sein d'un même réseau, au moins un équipement FFD et des équipements RFD utilisent de concert le même canal radio parmi ceux définis dans la norme.

Un équipement FFD peut dialoguer avec des équipements RFD ou FFD, mais un équipement RFD ne peut dialoguer qu'avec un équipement FFD.

## 2.4.2 Couche ZigBee

Le niveau ZigBee précise les algorithmes de routage pouvant être mis en œuvre dans le réseau.

Il autorise aussi le chiffrement des données par AES-128.

Le routage mis en œuvre peut être soit direct, soit indirect.

Dans le cas du routage direct, l'équipement source connaît l'adresse de l'équipement destinataire. L'adresse est celle définie au niveau de la trame MAC.

Dans le cas d'un routage indirect, l'équipement source ne connaît pas l'adresse de l'équipement destinataire. Dans ce cas, un équipement routeur ou coordinateur fera la mise en relation avec l'équipement destinataire en utilisant sa table de routage.

L'algorithme de routage préconisé dans la norme ZigBee pour les réseaux maillés est l'algorithme AODV (*Ad hoc On-Demand Vector Routing*). C'est un algorithme de routage réactif : une route est établie uniquement sur demande.

Si l'on regarde maintenant la topologie réseau et sa taille, le réseau ZigBee est un réseau PAN (*Personal Area Network*). C'est un réseau de quelques dizaines de mètres permettant un échange de données avec des équipements électroniques.

On peut mettre dans la catégorie PAN les normes suivantes : USB, Bluetooth, Infrarouge IR et ZigBee et éventuellement Wifi...

Les topologies mises en œuvre avec ZigBee dépendent de la complexité de l'application utilisée. La figure 7 présente quelques topologies possibles.

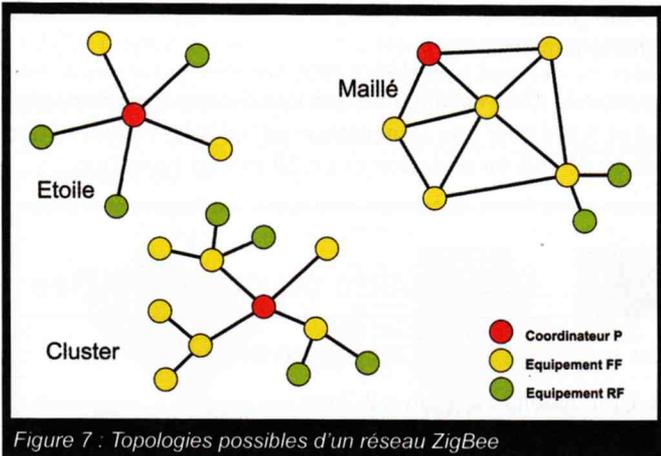


Figure 7 : Topologies possibles d'un réseau ZigBee

Des topologies plus complexes peuvent être mises en œuvre en utilisant des routeurs ZigBee. La figure 8 donne un exemple d'une telle topologie.

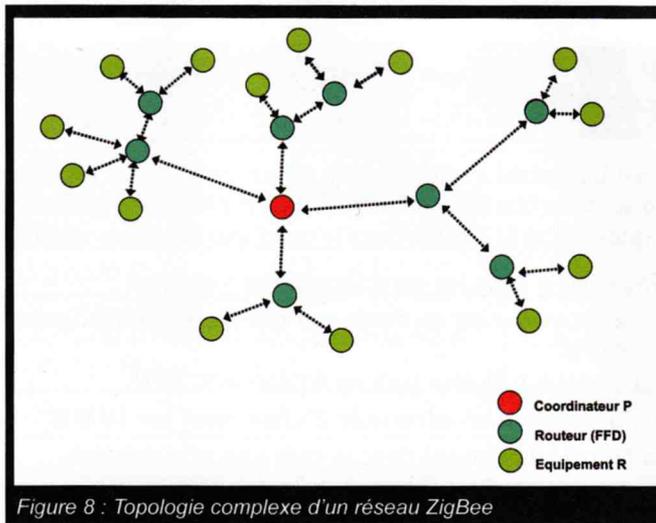


Figure 8 : Topologie complexe d'un réseau ZigBee

## 2.4.3 Bilan

ZigBee a toute sa place dans les réseaux de capteurs sans fil. C'est un concurrent sérieux à Bluetooth.

La figure 9 reprecise ZigBee dans le panorama des réseaux sans fil.

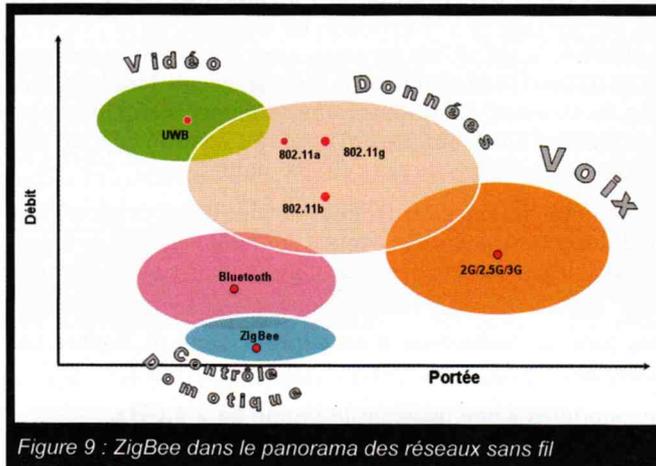


Figure 9 : ZigBee dans le panorama des réseaux sans fil

ZigBee a donc bien un marché ciblé dans la domotique dans le cadre des réseaux de capteurs sans fil. C'est d'ailleurs un choix judicieux pour ce type de réseau sans fil par rapport à Bluetooth comme le montre le tableau 1.

Caractéristiques	ZigBee	Bluetooth
Portée	10 - 100 m	10 m
Rôle	jusqu'à 400 m	100 m
Débit	20, 40, 250 kbps	1 Mbps
Latence		
Énumération nouvel équipement	30 ms	20 s
Réveil équipement	15 ms	3 s
Autonomie	En années	En jours
Sécurité	128 bits AES	64 bits, 128 bits
Fréquences	868 MHz, 915 MHz, 2.4 GHz ISM	2.4 GHz ISM
Complexité	Simple	Complexe
Topologie réseau	Ad hoc, étoile, maillé	Ad hoc
Équipements par réseau	jusqu'à 65536	8
Extensibilité	Ouï/Très grande	Non/Très faible
Flexibilité	Très grande	Moyenne
Fiabilité	Très grande	Moyenne

Tableau 1 : Comparaison de ZigBee et de Bluetooth

## 2.5

## ZigBee dans le cadre de la réalisation du capteur sans fil

Il est important maintenant de situer ZigBee dans le cadre de la réalisation du capteur sans fil. Il convient de réaliser ce capteur sans fil ZigBee dans le cas d'une topologie simple.

Nous avons donc les caractéristiques suivantes :

- La topologie est en étoile. Le réseau est en mode non beacon.
- Le capteur ZigBee sera un équipement RFD.
- On utilisera un adressage ZigBee court sur 16 bits.

La figure 10 présente donc le cadre de la réalisation.

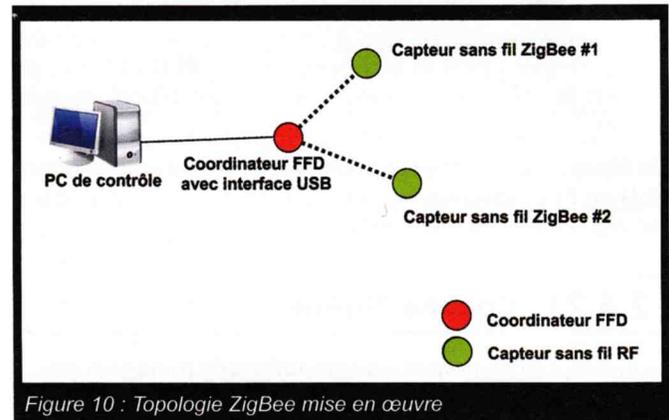


Figure 10 : Topologie ZigBee mise en œuvre

## 3

## Interface ZigBee : le module XBee

Réaliser un *transceiver* RF ZigBee est hors de portée d'un électronicien amateur. Il faut donc se tourner vers une solution intégrée.

Les électroniciens doivent se souvenir des circuits FTDI qui permettent une conversion de données série TTL vers l'interface USB et réciproquement. Le module intégré USBMODx (USBMOD3) basé sur un circuit FTDI permet très facilement de rajouter une connectivité USB à un équipement électronique qui apparaît ensuite comme une liaison série USB (`/dev/ttyUSBx` sous Linux).

La société Maxstream (rachetée par Digi) propose des modules intégrés ZigBee prêts à l'emploi : le module XBee. Ce module permet grossièrement une conversion de données série TTL vers le réseau ZigBee.

Dès lors, la réalisation d'un capteur sans fil ZigBee est possible...

Les modules XBee utilisent la bande de 2.4 GHz.

Pouvant être utilisés sans aucun paramétrage, ils ont différents modes de fonctionnement programmables par la liaison série en utilisant des commandes AT.

Ce point est important pour la programmabilité, car il suffira d'écrire des programmes simples capables de générer ces commandes AT.

Enfin, les modules XBee peuvent être achetés en France chez Lextronic. Il existe différents kits de développement pour démarrer des expérimentations. Ces kits contiennent des platines support RS.232 ou USB pour enficher les modules XBee.

Il faut noter néanmoins que les broches du module XBee sont au pas de 2 mm donc différents du pas traditionnel de 0.1 inch (2,54 mm). Cela obligeait, il y a 2 ans, à « jongler » au niveau du PCB. Il existe maintenant une petite platine d'interface pour le module XBee qui permet de se remettre au pas de 2,54 mm. Cette platine intègre aussi un convertisseur de tension 3,3 V pour alimenter correctement le module XBee, ce qui simplifie le design...

## 3.1

## Caractéristiques du module XBee

Les modules XBee présentent une puissance de sortie RF de 10 mW pour la version standard et 100 mW pour la version PRO.

Concernant la portée intérieure, elle peut atteindre au maximum 30 mètres suivant la nature des obstacles. Pour la portée extérieure, elle est au maximum de 100 mètres en champ libre.

Le débit est de 250 kb/s sur l'interface radio. Le débit à l'interface série est au maximum de 230.4 kb/s. Enfin, le récepteur possède une sensibilité de -92 dBm.

Il est possible de choisir son canal radio, ainsi que l'adresse ZigBee du module en adressage court parmi les 65536 possibles... La topologie réseau supportée est celle offerte par ZigBee.

Le module XBee est alimenté par une tension comprise entre 2.8 et 3.4 V. Pour une alimentation de 3.3 V, la consommation est de 45 mA en émission et de 50 mA en réception.

## 3.2

## Description du module XBee

Les modules XBee ont les caractéristiques suivantes :

- Dimensions : 2.4 cm x 2.7 cm.
- Poids : 3g.
- Température de fonctionnement : -40 °C à 85° C

Le tableau 2 présente le brochage du module XBee.

Broche	Nom	Direction	Description
1	VCC	-	Alimentation
2	DOUT	Out	Sortie UART
3	DIN/CONFIG	In	Entrée UART
4	DO8	Out	Sortie digitale 8
5	RESET	-	Reset (au moins 200 ns)
6	PWM0/RSSI	Out	Sortie PWM0/Indication puissance Rx
7	PWM1	Out	Sortie PWM1
8	Réservé	-	-
9	DTR*/SLEEP_RQ/DI8	In	Contrôle Sleep/Entrée digitale 8
10	GND	-	Ground
11	AD4/DIO4	Inout	Entrée analogique 4 ou E/S digitale 4
12	CTS/DIO7	Inout	Clear To Send/ E/S digitale 7
13	ON/SLEEP	Out	Indicateur état
14	VREF	-	Tension de référence pour conversion
15	Associate/AD5/DIO5	Inout	Indication association/Entrée analogique 5 ou E/S digitale 5
16	RTS/AD6/DIO6	Inout	Reday To Send/Entrée analogique 6 ou E/S digitale 6
17	AD3/DIO3	Inout	Entrée analogique 3 ou E/S digitale 3
18	AD2/DIO2	Inout	Entrée analogique 2 ou E/S digitale 2
19	AD1/DIO1	Inout	Entrée analogique 1 ou E/S digitale 1
20	AD0/DIO0	Inout	Entrée analogique 0 ou E/S digitale 0

Tableau 2 : Brochage du module XBee

Sur les 20 broches du module XBee, seules 9 broches sont utiles pour notre application (en vert dans le tableau 2).

En effet, au minimum, le module XBee nécessite de câbler les broches d'alimentation (VCC et GND) et les signaux DIN et DOUT pour respectivement les données entrantes et sortantes de l'interface UART. Les autres broches utilisées servent à configurer le module...

Le module XBee sera enfiché sur la platine d'interface dont le brochage est donné Figure 11.

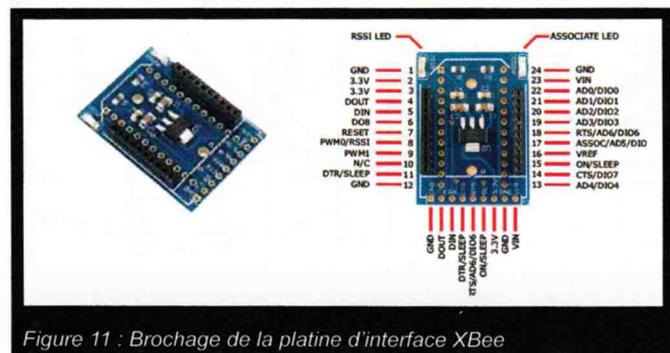


Figure 11 : Brochage de la platine d'interface XBee

Il est à noter qu'il existe différentes versions de modules XBee :

- Version avec antenne chip intégrée (XBee1).
- Version avec connecteur U.FL (MMCX) pour antenne externe (XBee2).
- Version avec antenne filaire intégrée (XBee3).



Figure 12 : Les différentes versions de modules XBee

### 3.3 Communications avec le module XBee

Le module XBee peut être connecté à n'importe processeur possédant une interface série UART comme indiqué Figure 13.

Le processeur envoie ses données (caractères) sur le port DI en mode série asynchrone. Chaque caractère est composé d'un bit de *start* (niveau logique 0), suivi de 8 bits de données avec le bit de poids faible en premier et enfin un bit de stop.

La figure 14 présente le mode de fonctionnement interne du module XBee.

Il existe un *buffer* à l'émission (*DI buffer*) et un buffer en réception (*DO buffer*). Un contrôle de flux matériel peut être mis en place en utilisant le traditionnel couple CTS/RTS pour gérer le flux à l'émission et à la réception des données suivant le taux de remplissage des buffers (comme dans un circuit UART classique).

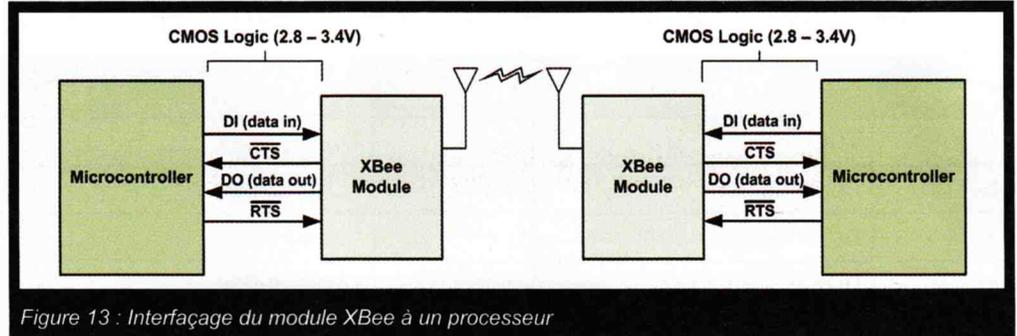


Figure 13 : Interfaçage du module XBee à un processeur

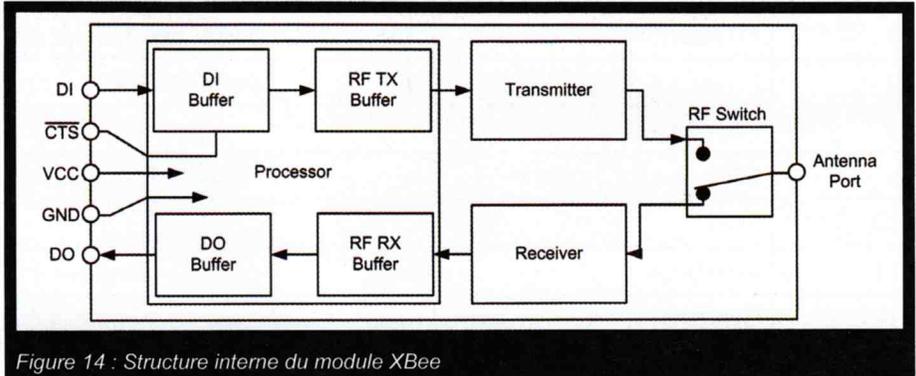


Figure 14 : Structure interne du module XBee

## 4 Capteur sans fil ZigBee : réalisation matérielle

### 4.1

### Cahier des charges – Choix des composants

Nous désirons réaliser un capteur sans fil ZigBee le plus simple possible, programmable et permettant d'interfacer simplement tout type de capteurs, de préférence à interface numérique.

Concernant l'interface numérique, les bus SPI, I2C ou 1 fil seront préférés, car il existe une multitude de circuits électroniques compatibles avec ces normes de communication série.

À partir de ce cahier des charges, nous avons choisi les composants principaux suivant :

- Microcontrôleur PIC Microchip. Il faut une interface série UART. La version moyenne gamme 16F877 fera l'affaire.
- Module XBee avec sa platine d'interface pour l'interface ZigBee.
- Un capteur de température pour l'exemple. Nous avons choisi le célèbre circuit DS1620 avec son interface 1 fil.

### 4.2

### Réalisation de la carte

Le schéma de principe est donné Figure 15.

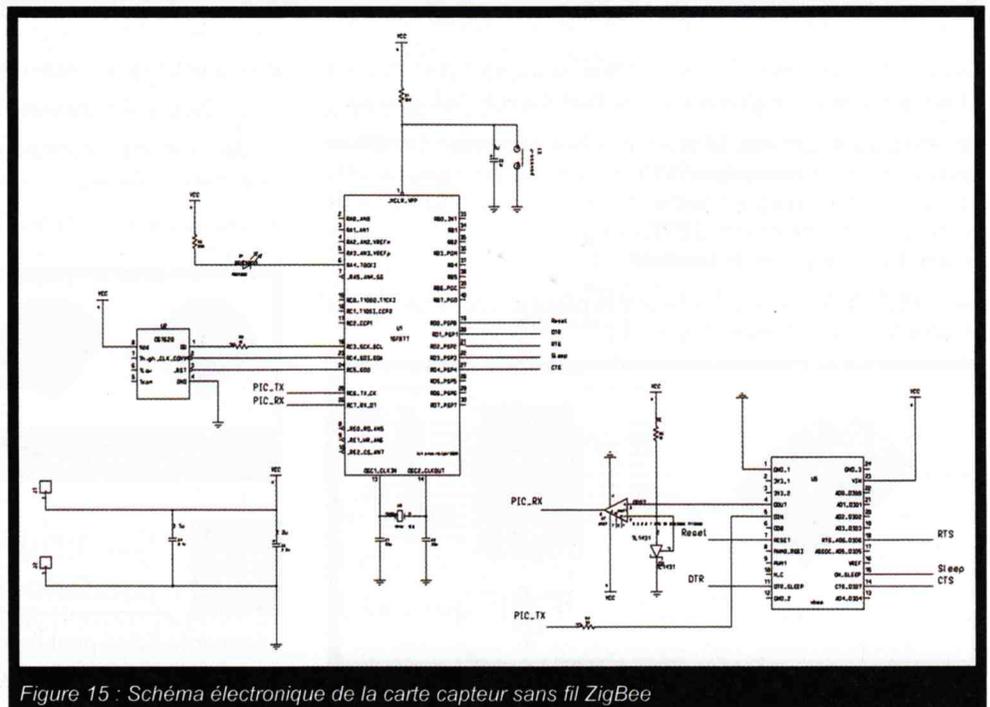


Figure 15 : Schéma électronique de la carte capteur sans fil ZigBee

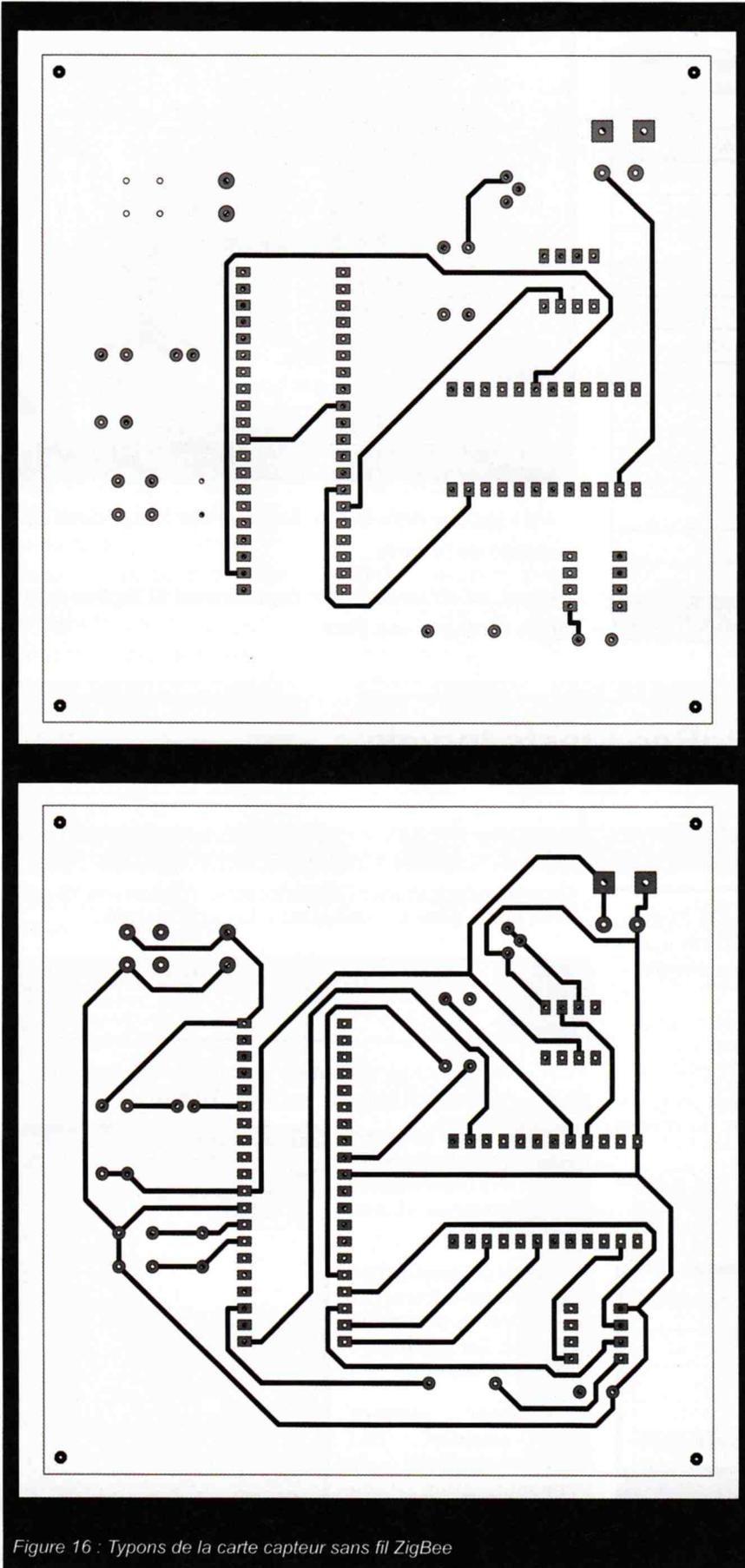


Figure 16 : Typons de la carte capteur sans fil ZigBee

Le capteur de température DS1620 utilise les broches **RC3, RC4** et **RC5** du microcontrôleur PIC.

Une LED servant de *heartbeat* utilise la **broche RA4** du microcontrôleur **PIC**.

Le module XBee utilise les broches RC6 (Tx de l'UART) et RC7 (Rx de l'UART) du microcontrôleur PIC.

Il est à noter que pour l'adaptation de niveaux entre l'interface UART du microcontrôleur et le module ZigBee, on a utilisé une simple résistance pour l'émission vers le module XBee (5V vers 3,3V) et un amplificateur opérationnel *Rail-to-Rail* monté en comparateur pour la réception depuis le module XBee (3,3V vers 5V).

Le circuit imprimé réalisé est double face. Les typons à l'échelle 1 sont disponibles au libre téléchargement à l'adresse donnée en annexe.

Le schéma d'implantation des composants est donné Figure 17 :

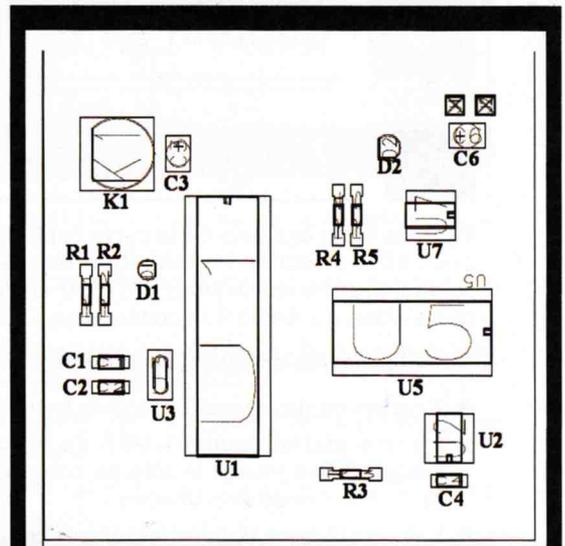


Figure 17 : Implantation des composants sur la carte capteur sans fil ZigBee.

Le circuit s'avère relativement simple même s'il s'agit de double couche. Différentes techniques de réalisation sont ainsi envisageables allant de la classique insolation à l'utilisation du transfert par toner.

Enfin, il reste la possibilité de faire réaliser le circuit par un professionnel. Différentes boutiques en ligne offrent ce genre de prestation pour des budgets tout à fait raisonnables. Ne vous restera, ensuite, que l'implantation et la soudure des composants, une étape bien plus à la portée du développeur standard.

La liste exhaustive des composants est la suivante :

Nom	Valeur	Référence
16F877	-	U1
DS1620	-	U2
Module XBee et son support	-	U5
Quartz	4 MHz	U3
AOP OPA337	-	U7
LED	-	D1
Bouton poussoir ITT	-	K1
Capacité tantale	2,2 µF	C3, C6
Capacité	100 nF	C4
Capacité	33 pF	C1, C2
Zéner TL1431C	-	D2
Résistance	10 kΩ	R1
Résistance	680 Ω	R2
Résistance	1 kΩ	R3, R5
Résistance	33 Ω	R4

Tableau 3 : Liste des composants de la carte capteur sans fil ZigBee

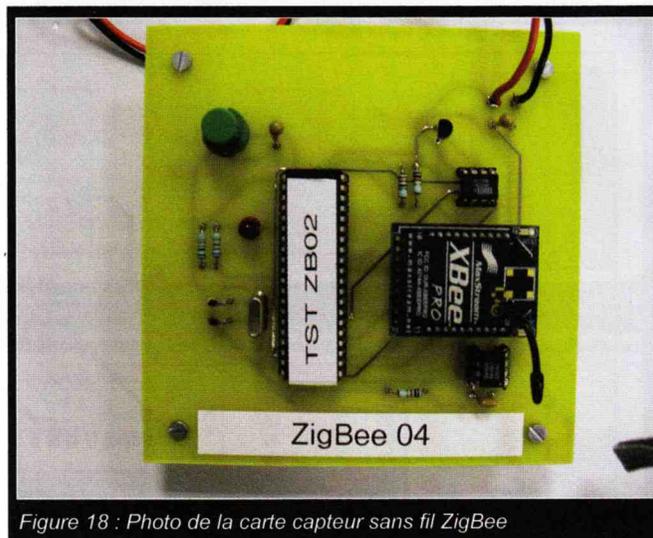


Figure 18 : Photo de la carte capteur sans fil ZigBee

Il n'y a aucune difficulté au soudage des composants et au montage de la carte.

Au final, on obtient la carte capteur sans fil ZigBee dont la figure 18 donne une photo.

## 5

## Capteur sans fil ZigBee : tests logiciels

### 5.1

### Configuration pour les tests

Pour les tests logiciels de la carte capteur sans fil ZigBee, nous allons décrire l'application de tests qui récupère la température du capteur DS1620 et l'envoie ensuite à destination du nœud de coordination.

Du point de vue matériel, nous avons :

- 2 cartes capteur sans fil ZigBee notées ZB01 et ZB02.
- 1 carte platine support USB de Maxstream avec un module XBee jouant le rôle de coordinateur du réseau ZigBee en mode non beacon.
- 1 PC *dual boot* Windows/Linux relié à la carte platine précédente.

On a donc la configuration de la figure 19.

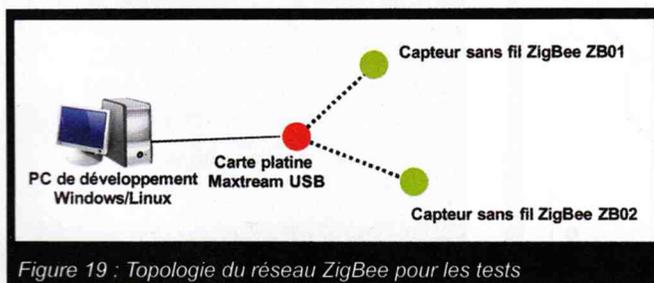


Figure 19 : Topologie du réseau ZigBee pour les tests

La carte platine support USB est transparente : elle possède une interface USB vue comme une liaison série

### 5.2

### Programmation des modules XBee

USB `/dev/ttyUSBx`. Tout ce qu'elle reçoit du réseau ZigBee est envoyé sur l'USB et réciproquement.

On pourra donc utiliser l'Hyperterminal Windows ou Minicom sous Linux pour se connecter à la carte platine...

Cette carte platine sert aussi à programmer les modules XBee via un outil Maxstream sous Windows.

C'est toujours un grand classique. Les outils offerts sont généralement sous Windows et non sous Linux.

L'outil de programmation sous Windows fourni par Maxstream s'appelle X-CTU. La figure 20 présente cet outil.

Il faut préciser en premier lieu les paramètres de communication de l'interface UART du module XBee (9600, 8, N, 1 par défaut).

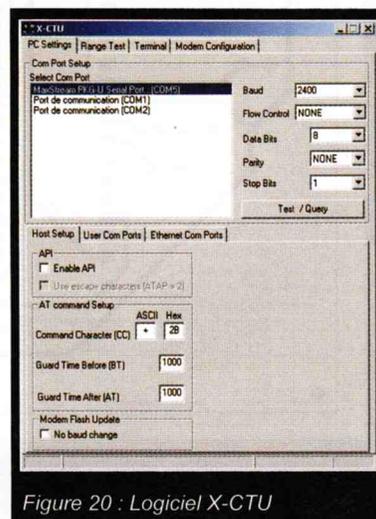


Figure 20 : Logiciel X-CTU

Il est possible de mettre à jour le *firmware* du module (version 10A5 ici), mais aussi de changer la valeur par défaut des paramètres de configuration du module. X-CTU va ensuite générer les commandes AT qui vont bien pour reprogrammer le module.

Les paramètres importants du module XBee sont les suivants :

- CH : canal radio.
- ID : identificateur PAN du réseau ZigBee.
- DH : adresse destination, 16 bits de poids fort.
- DL : adresse destination, 16 bits de poids faible.
- MY : adresse source, 16 bits de poids faible.
- CE : validation de la fonctionnalité coordinateur.
- BD : vitesse de l'interface UART du module.

Il convient d'avoir la même valeur pour l'ensemble des modules pour CH et ID.

Pour travailler en adresse courte sur 16 bits, il faut que DH=0 pour l'ensemble des modules.

Le module coordinateur a CE=1.

Dans ce cas, pour adresser le module y (adresse source MY<sub>y</sub>) depuis le module x, il faut que DL<sub>x</sub> = MY<sub>y</sub> pour le module x.

Pour les tests, on a donc les valeurs suivantes :

Module	CH	ID	DH	DL	MY	CE	BD
Carte platine support USB	0x0C	1	0	0	0	1	1 (2400 b/s)
ZB01	0x0C	1	0	0	0	0	1
ZB02	0x0C	1	0	0	0	0	1

Tableau 4 : Valeurs à programmer dans les modules XBee

La figure 21 montre l'étape de programmation des modules XBee.

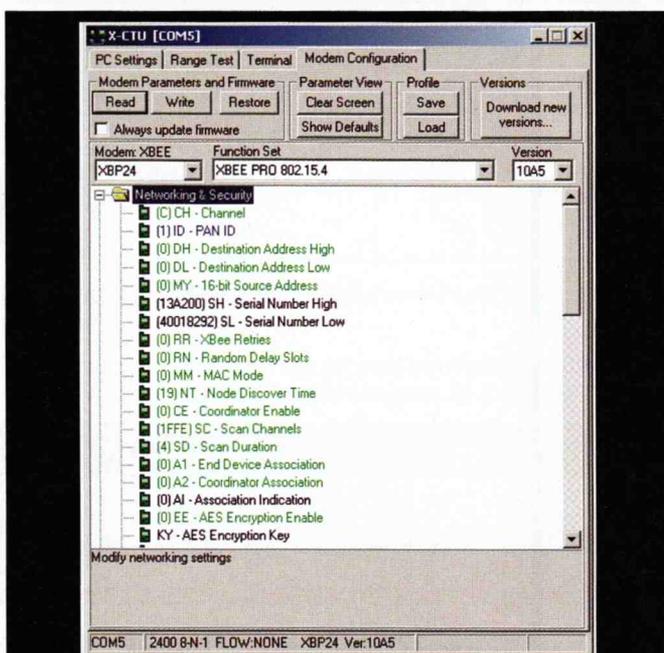


Figure 21 : Programmation des modules XBee avec X-CTU

## 5.3

### Développement du programme pour le microcontrôleur PIC

On utilise les outils classiques de développement sous Windows :

- Compilateur C PICC Hi-Tech pour PIC. Il existe une version freeware limitée PCC-Lite sur le site de Hi-Tech.
- IDE MPLAB de Microchip.
- Programmeur de PIC.

Il convient d'adopter une programmation modulaire pour créer des bibliothèques de fonctions afin de piloter les différents périphériques de la carte.

La bibliothèque **lib1620** permet d'initialiser, de dialoguer et de récupérer la température du capteur DS1620. Le protocole utilisé est le protocole 1 fil (*One Wire*).

On retrouve les fonctions suivantes :

- `void ds1620_init (void)` : initialisation du circuit DS1620.
- `void ds1620_write8 (unsigned char)` : écriture de 8 bits vers le circuit DS1620.

- `void ds1620_start (void)` : lancement acquisition de la température.
- `short ds1620_readtemp (void)` : lecture de la température.
- `unsigned char ds1620_read8 (void)` : lecture de 8 bits en provenance du circuit DS1620.
- `short ds1620_read9 (void)` : lecture de 9 bits en provenance du circuit DS1620.

La température est sur 9 bits en complément à 2 suivant le codage du tableau 5.

TEMP	DIGITAL OUTPUT (Binary)
+125°C	011111010
+25°C	000110010
+½°C	000000001
+0°C	000000000
-½°C	111111111
-25°C	111001110
-55°C	110010010

Tableau 5 : Codage de la température par le circuit DS1620

La bibliothèque **libuart** permet d'initialiser et d'utiliser la liaison série du microcontrôleur PIC.

On retrouve les fonctions suivantes :

- `void InitUART(void)` : initialisation de la liaison série du PIC.
- `void EmitUART(char)` : émission d'un caractère.
- `char RecUART(void)` : réception d'un caractère.

La bibliothèque `libxbee` permet d'initialiser et d'utiliser le module XBee. Elle fait appel à la bibliothèque `libuart`.

On retrouve les fonctions suivantes :

- `void InitXbee(void)` : initialisation du module XBee.
- `void SendByte(char)` : émission d'un octet vers le module XBee.
- `char ReadByte(void)` : réception d'un octet du module XBee.

À partir de ces bibliothèques, il ne reste plus qu'à écrire le programme principal dont le canevas est donné ci-après :

```
void main(void)
{
    short temperature;
    char counter, slope;

    InitPort();
    InitUART();
    InitXbee();
    ds1620_init();
    ds1620_start();

    Delays(2);

    while(1)
    {
        PORTA=0x00;

        lecture température;
        calcul température;
        émission température;

        PORTA=0xFF;
        Delays(5);
    }
}
```

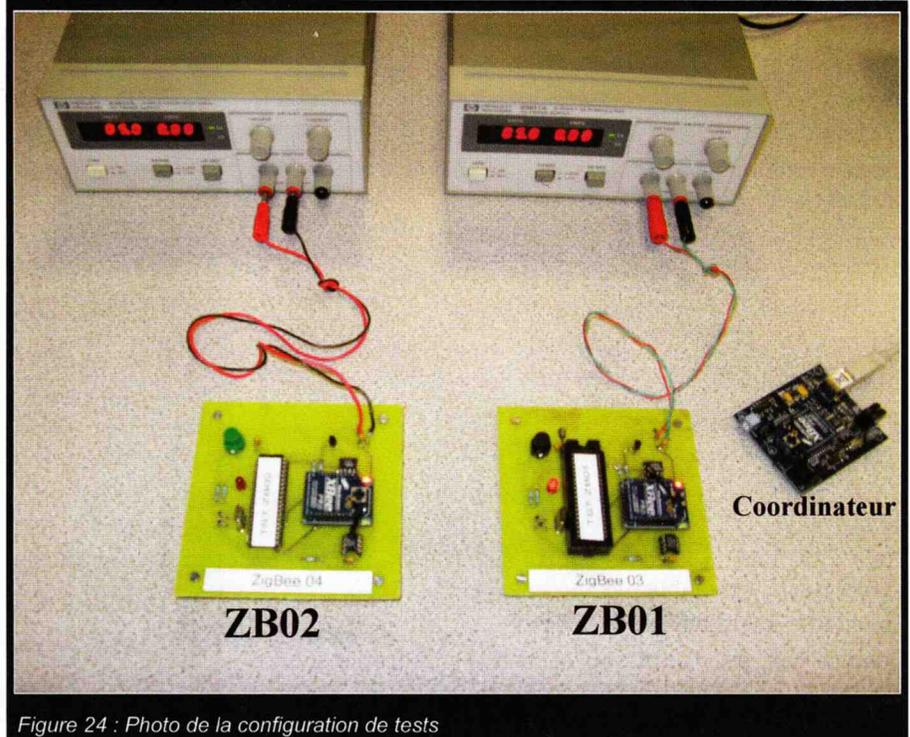


Figure 24 : Photo de la configuration de tests

Le programme de tests est ensuite compilé. On programme ensuite les 2 microcontrôleurs PIC des cartes ZB01 et ZB02...

## 5.4 Tests

On utilise le programme X-CTU pour les tests sous Windows. En effet, il possède une fonctionnalité Hyperterminal. On obtient les résultats de la figure 22.

On voit bien sur la figure 22 que l'on récupère la température des cartes capteur sans fil ZigBee ZB01 et ZB02.

Sous Linux, la carte platine support USB est vue comme un port série USB `/dev/ttyUSB0`.

On peut alors utiliser l'outil Minicom.

On voit bien sur la figure 23 que l'on récupère aussi la température des cartes capteur sans fil ZigBee ZB01 et ZB02.

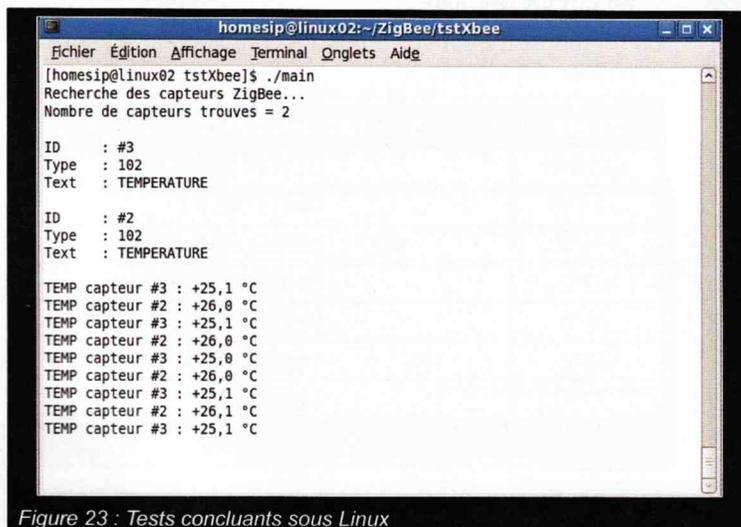


Figure 23 : Tests concluants sous Linux

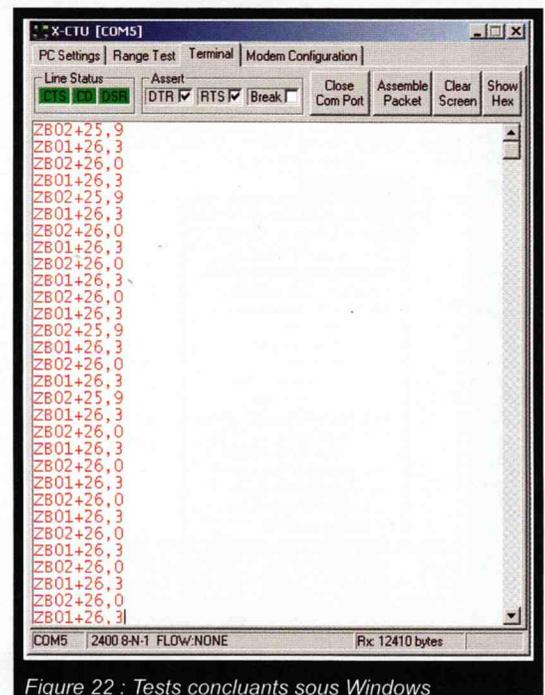


Figure 22 : Tests concluants sous Windows

## 6 Capteur sans fil ZigBee : intégration dans le projet HomeSIP

Dans le cadre du projet HomeSIP, les capteurs sans fil ZigBee ont été intégrés. Un protocole de dialogue entre les capteurs sans fil et le nœud de coordination a été élaboré pour supporter différents types de capteurs sans fil (température, pression, humidité...).

soit, dans notre cas, DL=2 pour interroger ZB01 et DL=3 pour interroger ZB02.

Pour les capteurs sans fil ZigBee ZB01 et ZB02, DL est fixé à 1 pour adresser la carte platine support USB ZigBee.

### 6.1 Protocole de dialogue

Un protocole de dialogue plus élaboré a été mis en œuvre pour pouvoir supporter différents types de capteurs ZigBee dont celui que l'on vient de concevoir.

Le protocole employé est le suivant :

Sens carte platine support USB vers capteur sans fil ZigBee :

Commande	ID	Nombre d'octets	Message	Tag de fin
Demande type capteur	'1'	1	Space	'z'
Demande lecture capteur	'2'	1	Space	'z'
Demande changement paramètre XBee	'3'	4	Commande AT	'z'

Tableau 6 : Protocole carte platine support USB vers capteur

Sens capteur sans fil ZigBee vers carte platine support USB :

Réponse	ID	Nombre d'octets	Message	Tag de fin
Réponse type capteur	'4'	1-5	Texte	'z'
Lecture capteur	'5'	5	Valeur capteur	'z'
Changement paramètre XBee	'6'	2	0/1 - 0/1	'z'

Tableau 7 : Protocole capteur vers carte platine support USB

Il est ainsi possible de connaître le type de capteur sans fil ZigBee, de récupérer la valeur mesurée par le capteur (ici une température) et même de reprogrammer le module XBee du capteur sans fil ZigBee.

On retrouve les fonctions suivantes :

- *unsigned char SendMsg (msg \*)* : envoi d'une réponse suivant le format du protocole.
- *unsigned char ReceiveMsg (msg \*)* : réception d'une commande suivant le format du protocole.
- *unsigned char TraiteMsg (msg \*trame)* : traitement de la commande reçue.

À partir de ces bibliothèques, il ne reste plus qu'à écrire le programme principal dont le canevas est donné ci-après :

### 6.2 Programmation des modules XBee

Les 3 modules XBee sont reprogrammés dans le cadre du contexte du projet HomeSIP.

Nous avons fixé l'adresse (en adressage court) des différents modules Xbee, donc des capteurs sans fil ZigBee.

Module	CH	ID	DH	DL	MY	CE	BD
Carte platine support USB	0x0C	1	0	x	1	1	3 (9600 b/s)
ZB01	0x0C	1	0	1	2	0	1
ZB02	0x0C	1	0	1	3	0	1

À chaque fois que la carte platine support USB veut interroger un capteur sans fil, il réajustera la valeur DL de son module XBee à la valeur MY du capteur à adresser,

```
#include "lib1620.c"
#include "libuart.c"
#include "libxbee.c"
```

```

#include "calcul.c"
#include "protocol.c"

__CONFIG (0x3F72);

void InitPort(void)
{
    TRISA = 0x00;
    TRISC = 0x80;
}

void main(void)
{
    msg MessageBuffer;
    InitPort();
    InitUART();
    InitXbee();

    ds1620_init();

    while(1)
    {
        while ((PIR1 & 0x20) == 0x00)
            ;
        MessageBuffer.id = ReadByte();

        if (('1' <= MessageBuffer.id) && (MessageBuffer.id <= '3'))
        {
            ReceiveMsg (&MessageBuffer);
            TraiteMsg (&MessageBuffer);
            SendMsg (&MessageBuffer);
        }
    }
}

```

Le programme de tests est ensuite compilé. On programme ensuite les 2 microcontrôleurs PIC des cartes ZB01 et ZB02...

## 6.4 Programme de tests sous Linux

Sous Linux, la carte platine support USB est vue comme un port série USB `/dev/ttyUSB0`.

La bibliothèque **libuart** permet d'initialiser et de dialoguer avec la platine support USB sous Linux.

On retrouve les fonctions suivantes :

- `void UartInit()` : initialisation du port série USB.
- `void UartClose()` : fermeture du port série USB.
- `void SendByte (char)` : émission d'un octet.
- `char ReceiveByte (void)` : lecture d'un octet.
- `void SendString(const char *)` : envoi d'une chaîne de caractères.

La bibliothèque **libxbee** permet d'initialiser et d'utiliser le module XBee de la platine support USB. Elle fait appel à la bibliothèque **libuart**.

On retrouve les fonctions suivantes :

- `unsigned char SendMsg (msg *)` : envoi d'un message formaté suivant le protocole défini.
- `unsigned char ReceiveMsg (msg *)` : réception d'un message suivant le protocole défini.
- `unsigned char Xbee_Scan (char *, Liste_capteurs *)` : balayage du réseau ZigBee pour découvrir les capteurs sans fil ZigBee.

- `unsigned char ChangerId (char *, unsigned char)` : changement du champ DL de la platine support USB pour dialoguer avec un capteur sans fil particulier.
- `int Xbee_Lecture (char *, unsigned char, char *)` : lecture de la valeur mesurée par le capteur.

Le fichier **xbee.c** précise les fonctions pour envoyer des commandes AT au module XBee de la platine support USB.

On retrouve les fonctions suivantes :

- `unsigned char EnterConfiguration (void)` : entrée dans le mode de configuration du module XBee.
- `unsigned char EnterCommand (char *, char *)` : envoi d'une commande AT.
- `unsigned char TestOK (void)` : test du code de retour de la commande AT.

À partir de ces bibliothèques, il ne reste plus qu'à écrire le programme principal dont le canevas est donné ci-après :

```

#include "../libXbee/libxbee.h"
#include <stdio.h>

int nbr, i;
char buff[6];
Liste_capteurs Xbee;

int main(int argc, char *argv[])
{
    buff[5]=0;
    printf("Recherche des capteurs ZigBee... \n");
    Xbee_Scan(XBEE_DEV, &Xbee);
    nbr = Xbee.nbr;
    printf("Nombre de capteurs trouves = %d\n\n", nbr);

    for(i=0; i< nbr; i++) {
        printf("ID      : #c\n", Xbee.id[i][7]);
        printf("Type   : %d\n", Xbee.id[i][0]);
        printf("Text   : %s\n", Xbee.type_capteur[i]);
        printf("\n");
    }

    while(1) {
        for(i=0; i< nbr; i++) {
            Xbee_Lecture(XBEE_DEV, Xbee.id[i][7], buff);
            printf("TEMP capteur #c : %s A°C\n", Xbee.id[i][7], buff);
            fflush(stdout);
        }
    }
    exit(0);
}

```

On obtient alors le résultat escompté donné par la figure 25.

```

homesip@linux02:~/ZigBee/tstXbee
[homesip@linux02 tstXbee]$ ./main
Recherche des capteurs ZigBee...
Nombre de capteurs trouves = 2

ID      : #3
Type   : 102
Text   : TEMPERATURE

ID      : #2
Type   : 102
Text   : TEMPERATURE

TEMP capteur #3 : +25,1 °C
TEMP capteur #2 : +26,0 °C
TEMP capteur #3 : +25,1 °C
TEMP capteur #2 : +26,0 °C
TEMP capteur #3 : +25,0 °C
TEMP capteur #2 : +26,0 °C
TEMP capteur #3 : +25,1 °C
TEMP capteur #2 : +26,1 °C
TEMP capteur #3 : +25,1 °C

```

Figure 25 : Tests concluants sous Linux de l'intégration dans le projet HomeSIP

## 7 Conclusion

Nous avons pu voir l'étude et la réalisation d'un capteur sans fil ZigBee, du matériel au logiciel embarqué. De plus, son intégration dans le projet HomeSIP a été abordé.

ZigBee joue et jouera un rôle de plus en plus important dans la domotique et dans les réseaux de capteurs sans fil, et de plus en plus avec la convergence des disciplines comme l'électronique, l'informatique et les réseaux.

Il faut enfin noter la chance de disposer de modules ZigBee faciles à intégrer dans des réalisations amateurs. Nous avons pu voir l'usage des modules XBee, les tous premiers disponibles il y a 2 ans. D'autres existent maintenant...

À vos fers à souder !

L'ensemble des fichiers nécessaires pour réaliser la carte et des fichiers sources est disponible à l'adresse : <http://uuu.enseirb.fr/~kadionik/glmf/hs38/>.

Auteur : Patrice Kadionik

Patrice Kadionik, Maître de Conférences à l'ENSEIRB-IMS

### Références

- Projet HomeSIP : <http://www.enseirb.fr/cosynux/HomeSIP/>
- KADIONIK (P.), « Le projet HomeSIP : la domotique avec le protocole SIP », *GNU/Linux Magazine*, hors-série 25, avril-mai 2006, p. 34-43.
- Alliance ZigBee : <http://www.zigbee.org/>
- Norme IEEE 802.15.4 : <http://standards.ieee.org/getieee802/download/802.15.4-2003.pdf>
- Norme IEEE 802.15.4 sur Wikipédia : <http://fr.wikipedia.org/wiki/802.15.4>
- ZigBee sur Wikipédia : <http://fr.wikipedia.org/wiki/Zigbee>
- Circuit FTDI : <http://www.ftdichip.com/>
- Module Xbee de Maxstream : <http://www.maxstream.net/>
- Manuel du module Xbee : [http://ftp1.digi.com/support/documentation/manual\\_xb\\_oem-rf-modules\\_802.15.4\\_v1.xAx.pdf](http://ftp1.digi.com/support/documentation/manual_xb_oem-rf-modules_802.15.4_v1.xAx.pdf)
- Module Xbee chez Lextronic : <http://www.lextronic.fr/P1319-module-oem-xbee.html>
- Kit d'évaluation Xbee STK1 : <http://www.lextronic.fr/produit.php?id=1351>
- Platine d'interface Xbee : <http://www.lextronic.fr/P1901-platine-dinterface-pour-modules-xbee.html>
- Compilateur croisé C pour PIC PICC-Lite : <http://microchip.htsoft.com/products/compilers/PICClite.php>

■ Fichiers pour réaliser la carte capteur sans fil ZigBee, ainsi que les fichiers sources : <http://uuu.enseirb.fr/~kadionik/glmf/hs38/>

Toutes les marques produits et logos cités appartiennent à leurs compagnies respectives.

### Remerciements

L'auteur tient à remercier les étudiants de l'ENSEIRB qui ont travaillé sur le projet HomeSIP et sur la réalisation des capteurs sans fil ZigBee dans le cadre de leur projet avancé d'option Systèmes Embarqués : Cédric Beausse, Mohamed Bouguerra, Hong Yu Guan, El Ayachi Moktad, Willy Aubry, Mikel Azkarate-Askasua, Fabien Marteau, Hanen Sabeur, Guillaume Gardet, Benjamin Pussacq, Hervé Spitz et Damien Lozach.

# Pragmatec

## Module ARM9 Linux 2.6

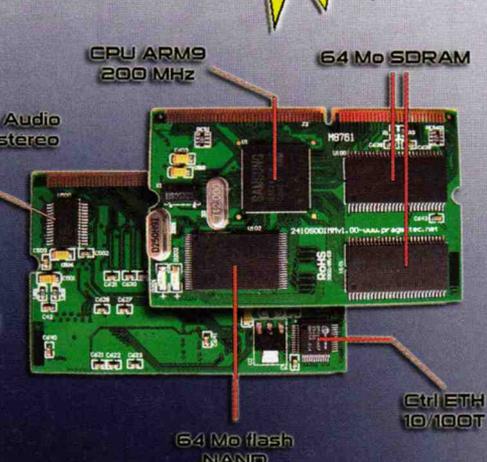


SODIMM DDR144  
ARM9 - 200MHz

**95 € HT**

**Notre toute nouvelle carte SODIMM ARM9 :**

- \* ARM9 S3C2410
- \* 200 MHz
- \* 64 Mo de SDRam
- \* 64 Mo de NAND
- \* USB host (x2)
- \* USB device
- \* Ports RS232 (x3)
- \* Ctrl Eth 10/100T
- \* Ctrl Audio 16bits
- \* Format DDR144
- \* BIOS
- \* Linux 2.6.23
- \* Module pré-programmé
- \* Customisation par 100





[www.pragmatec.net/catalog](http://www.pragmatec.net/catalog)

# Communication asynchrone et interface g



*Malgré la popularité de GNU/Linux dans la communauté des développeurs, de nombreux utilisateurs s'obstinent à utiliser d'autres systèmes d'exploitations. Notre objectif dans cette présentation est de proposer une solution pour générer des applications utilisables pour un maximum d'utilisateurs, donc pour un maximum de plateformes. De plus, la mode aujourd'hui est aux applications graphiques.*

## 1 Introduction

Alors que des applications en mode texte [1] réaliseraient parfaitement la même fonction, la majorité des utilisateurs est rebutée à l'idée de lancer une fenêtre de commande DOS sous Windows ou un terminal sous Unix. Bien que nous regrettions cette attitude et la perte de performances associée, nous allons ici nous efforcer de répondre aux exigences de l'utilisateur en présentant les méthodes de développement d'applications graphiques portables. Nous allons de plus nous imposer la contrainte de communiquer avec des systèmes embarqués tels que ceux présentés dans les numéros précédents de ce magazine [2, 3, 4]. Ceci implique la capacité à utiliser

des ports de communication tels que le port série (RS232), afin de communiquer avec des microcontrôleurs.

Nous avons exploré diverses voies avant de converger sur le choix de la programmation sous environnement Qt : Perl et son environnement graphique Perl/Tk, GTK, Java. Qt4 est la seule solution que nous ayons testée qui réponde à nos critères de portabilité (pas de modification des codes source pour compiler une application GNU/Linux ou Windows) et de performances. Mentionnons la disponibilité d'une classe pour l'accès au port série en Java - **rxtx** disponible à <http://rxtx.org/> - tout à fait satisfaisante, mais qui ne sera pas développée ici.

## 2 Communication par interfaces asynchrones

Malgré la mort annoncée du port série (RS232), la communication asynchrone reste un périphérique communément utilisé sur les applications embarquées. Sa simplicité et sa fiabilité lui garantissent encore de beaux jours, malgré la volonté des industriels de l'informatique grand public d'imposer l'USB.

L'interface entre la communication asynchrone (côté microcontrôleur) et l'USB (ordinateur personnel) est notamment fournie par un composant très simple d'emploi, le FT232R de FTDI [5]. Ce composant s'interface d'un côté aux broches de communication issues de l'UART (*Universal Asynchronous Receiver Transmitter*, implémentation matérielle du

protocole RS232) de tout microcontrôleur, et, de l'autre côté, fournit une interface USB qui se programme, grâce aux *drivers* (<http://www.ftdichip.com/FTDrivers.htm>) fournis par le fabricant FTDI, comme un port série. Sous GNU/Linux, le module noyau **ftdi\_sio** associe **/dev/ttyUSBx** (x=0, 1...) à une interface de programmation classique de port série, telle que nous en avons l'habitude avec **/dev/ttySx**.

Si nous désirons que nos applications embarquées communiquent avec un maximum d'ordinateurs personnels, il est fondamental de savoir communiquer avec un port série de façon portable, indépendante du système d'exploitation.

## graphique portables sous Qt

## 3 L'accès au port série en C

Commençons par poser le problème en présentant les méthodes « classiques » d'accès au port série en C sous Linux, et en Visual C(++) pour Microsoft Windows. Bien qu'il soit techniquement possible de jouer sur les **#define** pour définir deux environnements portables, cette méthode est fastidieuse (bien que fonctionnelle). Elle se couple efficacement avec tout environnement portable d'interface graphique et mérite en ce sens d'être mentionnée.

```
#ifdef linux // All examples have been derived
from miniterm.c
#include <arpa/inet.h>
struct termios oldtio,newtio;
#define BAUDRATE B38400
#define RSDEVICE "/dev/ttyS0"

extern struct termios oldtio,newtio;

int init_RS232()
{int fd;
fd=open(RSDEVICE, O_RDWR | O_NOCTTY );
if (fd <0) {perror(RSDEVICE); exit(-1); }
tcgetattr(fd,&oldtio); /* save current serial port settings */
bzero(&newtio, sizeof(newtio)); /* clear struct for new
port settings */
newtio.c_cflag = BAUDRATE | CS8 | CLOCAL | CREAD; /*
no CRTSCTS */
newtio.c_iflag = IGNPAR; // | ICRNL | IXON;
newtio.c_oflag = IGNPAR; //ONOCR|ONLRET|OLCUC;
newtio.c_cc[VTIME] = 0; /* inter-character timer unused */
newtio.c_cc[VMIN] = 1; /* blocking read until 1
character arrives */
tcflush(fd, TCIFLUSH);tcsetattr(fd,TCSANOW,&newtio);
return(fd);
}

void free_rs232(int fd) /* restore the old port settings */
{tcsetattr(fd,TCSANOW,&oldtio);close(fd);}

#else // cas NON-Linux = MS-Windows

// hton[l/s] : use lib ws2_32.lib in
// C:\Program Files\Microsoft Visual Studio\VC98\Lib
// #include <winsock2.h.>

#define RSdelay 10

HANDLE portcom ;
DWORD dwBytesRecd,dwBytesWritten;
```

```
int init_RS232()
{DCB dcb;COMMTIMEOUTS cmm;
portcom=CreateFile("COM1", GENERIC_READ | GENERIC_WRITE,1,NULL,
OPEN_EXISTING,0,NULL ); // 0=NO overlapped IO
dcb.DCBLength = sizeof( DCB ) ;
GetCommState(portcom, &dcb ) ;
dcb.BaudRate=38400;
dcb.ByteSize=8;
dcb.Parity=NOPARITY;
dcb.StopBits=ONESTOPBIT;
SetCommState( portcom, &dcb ) ;
GetCommTimeouts(portcom, &cmm ) ;
cmm.ReadIntervalTimeout=RSdelay;
cmm.ReadTotalTimeoutConstant=RSdelay;
cmm.ReadTotalTimeoutMultiplier=RSdelay;
cmm.WriteTotalTimeoutConstant=RSdelay;
cmm.WriteTotalTimeoutMultiplier=RSdelay;
SetCommTimeouts(portcom, &cmm) ;
return(1);
}

void write(int dummy,char *cmd,int len) // POSIX
write for windows
{WriteFile(portcom,cmd,len,&dwBytesWritten,0);}

int read(int dummy,unsigned char *cmd,int len) // POSIX
read for windows
{do {ReadFile(portcom,cmd,len,&dwBytesRecd,0);}
while (dwBytesRecd==0);
return(dwBytesRecd);
}
#endif
```

Code 1 : Exemple de code fonctionnel sous GNU/Linux ou MS Windows selon la disponibilité de la variable d'environnement Linux : en cas d'utilisation de Windows, nous tentons d'émuler grossièrement les fonctions read() et write() POSIX.

Notre objectif dans ce document est d'éviter le genre de gestion des cas particuliers que nous présentons dans un code couramment utilisé dans nos applications C gérant une communication RS232 (code 1). En effet, ce genre de cas particulier se multiplie en fonction des plateformes que nous désirons supporter, jusqu'à rendre le code illisible et difficile à maintenir. Par ailleurs, le C de base manque d'un support portable d'interface graphique : parfait pour les logiciels en console (**stdio**), l'ajout d'une interface utilisateur devient rapidement fastidieuse.

## 4 L'accès au port série sous Qt

Afin de suppléer à ces déficiences du C, nous proposons de développer une classe Qt – environnement de développement proposé par Trolltech ([trolltech.com](http://trolltech.com)), libre si l'utilisateur propose ses applications sous licence GPL – pour gérer le port série de façon portable, et donc d'exploiter les possibilités d'interface graphique portable fournies par cet environnement de développement. Nous allons dans un premier temps présenter l'installation de l'environnement de développement Qt4, et, en particulier, la capacité à cross-compiler et tester une application à destination de Windows

sur plateforme GNU/Linux. Cette documentation n'a pas pour vocation de présenter le développement d'interfaces graphiques sous Qt pour interagir avec l'utilisateur, mais uniquement d'aborder les points de l'affichage des informations récupérées lors de la communication asynchrone avec un système embarqué. Le lecteur pourra se référer à la série d'articles sur Qt3 parue entre 2002 et 2004 dans ces pages pour une présentation plus exhaustive de cet environnement de développement [6].

## 4.1 Qt4 sous Debian GNU/Linux

L'installation se déroule trivialement par `apt-get install libqt4-dev` qui se chargera d'aller chercher les dépendances requises.

La compilation s'obtient ensuite :

- En développant son code Qt en C++ dans son éditeur de texte favori.
- En effectuant une première fois `qmake -project` pour générer un fichier `.pro` contenant les sources et les dépendances nécessaires à la compilation.
- Une première fois `qmake` pour générer le `Makefile`.
- Finalement, le classique `make` pour chaque remise à jour de l'exécutable lorsque les sources ont été modifiées.

```

/*****
 * http://doc.trolltech.com/4.3/tutorial-t1.html
 *****/

#include <QApplication>
#include <QPushButton>

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QPushButton hello("Hello world!");
    hello.resize(200, 30);
    hello.show();
    return app.exec();
}

```

Nous validons la chaîne de compilation en exécutant le programme résultant (Fig. 1).



Figure 1 : Résultat de la compilation de l'exemple Hello World fourni par TrollTech pour valider la chaîne de compilation, ici sous GNU/Linux.

Nous sommes donc capables de générer une interface graphique sous GNU/Linux. Après cette introduction triviale, nous allons dans un premier temps considérer la cross-compilation d'applications pour Windows. Le lecteur qui ne s'intéresse pas à ce système, mais désire tout de suite consulter la communication asynchrone sous Qt4, peut se reporter directement à la section 4.3.

## 4.2 Cross-compilation d'une application Qt4 pour Windows sous Linux

L'installation d'un environnement de compilation pour Windows sous Linux nécessite un compilateur croisé capable de traduire un code source en fichier exécutable pour une autre architecture, en l'occurrence un programme pour Windows [7, 8] :

- Dans cet exemple, `mingw32` sera notre compilateur.
- Il nous faudra avoir à disposition les bibliothèques Qt pour Windows, le plus simple étant de les télécharger sur le site de Trolltech.
- Finalement, le dernier outil nécessaire sera `wine` pour pouvoir tester notre application.

Sous Debian, nous installons les paquets appropriés pour la cross-compilation et le test des binaires que nous générerons :

```
su -c 'apt-get install mingw32 wine'
```

Une fois cet environnement fonctionnel, nous ajoutons Qt pour Windows que nous obtenons à partir du site de Trolltech [9] :

```
wget http://ftp.ntua.gr/pub/X11/Qt/qt/source/qt-win-opensource-4.3.4-mingw.exe
```

L'installation de Qt avec wine (figure 2) s'obtient par la commande

```
wine qt-win-opensource-4.3.4-mingw.exe
```

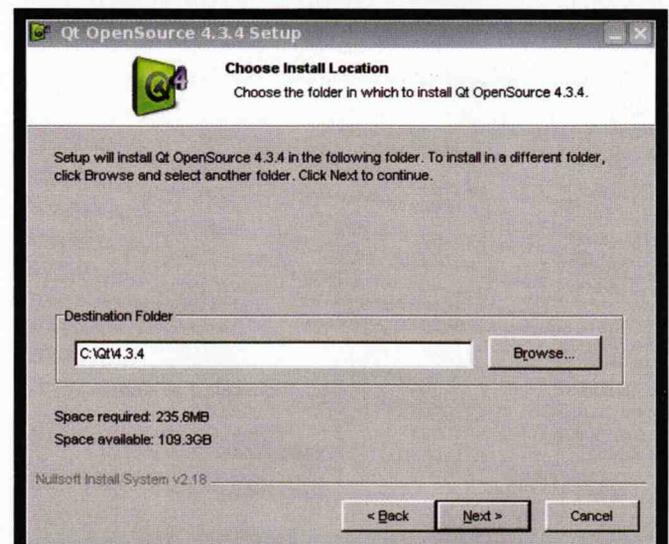


Figure 2 : Installation de Qt pour Windows avec wine : cette fenêtre s'obtient par `wine qt-win-opensource-4.3.4-mingw.exe`.

Une fois l'installation de Qt achevée correctement, il faut modifier la base de registres de votre environnement Windows avec `wine` pour y ajouter Qt dans le `PATH` : par défaut l'installation s'effectue dans `C:/Qt/4.3.4/`. Si la clef `HKEY_CURRENT_USER/Environment/PATH` n'existe pas, il faut la créer pour y ajouter : `C:/Qt/4.3.4/bin` comme sur la fig. 3

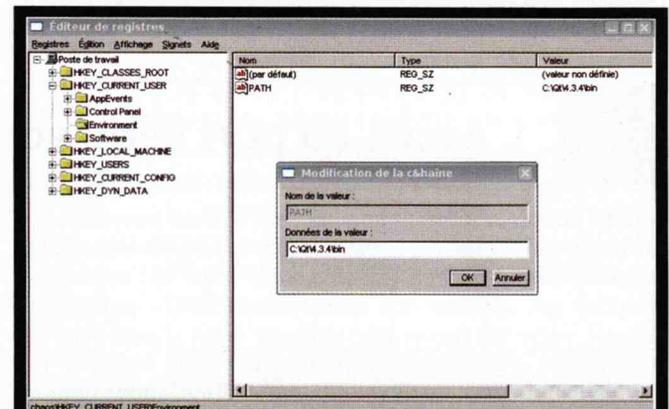


Figure 3 : Ajout de Qt dans le `PATH` de wine, obtenu par l'exécution de la commande `wine regedit`

Pour achever la configuration de notre environnement Windows, il nous reste à placer la bibliothèque **mingwm10.dll** dans le répertoire système approprié :

```
gunzip /usr/share/doc/mingw32-runtime/mingwm10.dll.gz
mv mingwm10.dll ~/.wine/drive_c/windows/system32/
```

Nous devons modifier les règles de compilation définies dans le fichier **mkspec**, afin de passer les bonnes options dans nos futurs **Makefile**. Afin de respecter les règles pour ce nouvel environnement, les principales différences avec le fichier original sont décrites ci-dessous :

```
cp -ar /usr/share/qt4/mkspecs/win32-g++ /usr/share/qt4/mkspecs/win32-cross
su -c 'cat > /usr/share/qt4/mkspecs/win32-cross/qmake.conf'
#
# qmake configuration for win32-cross
# Written for MinGW
#

MAKEFILE_GENERATOR = MINGW
QMAKE_CC = i586-mingw32msvc-gcc
QMAKE_CXX = i586-mingw32msvc-g++
QMAKE_INCDIR = /usr/i586-mingw32msvc/include/
QMAKE_INCDIR_QT = /home/julien/.wine/drive_c/Qt/4.3.4/include
QMAKE_LIBDIR_QT = /home/julien/.wine/drive_c/Qt/4.3.4/lib
QMAKE_LINK = i586-mingw32msvc-g++
MINGW_IN_SHELL = $$ (MINGW_IN_SHELL)
#isEqual(MINGW_IN_SHELL, 1) {
    QMAKE_DIR_SEP = /
    QMAKE_COPY = cp
    QMAKE_COPY_DIR = cp -r
    QMAKE_MOVE = mv
    QMAKE_DEL_FILE = rm -f
    QMAKE_MKDIR = mkdir -p
    QMAKE_DEL_DIR = rm -rf
#} else {
    ...
#}

QMAKE_MOC = $$[QT_INSTALL_BINS]$$[DIR_SEPARATOR]moc-qt4
QMAKE_UIC = $$[QT_INSTALL_BINS]$$[DIR_SEPARATOR]uic-qt4
QMAKE_IDC = $$[QT_INSTALL_BINS]$$[DIR_SEPARATOR]idc

QMAKE_LIB = i586-mingw32-ar -ru
QMAKE_RC = i586-mingw32msvc-windres
QMAKE_ZIP = zip -r -9

QMAKE_STRIP = i586-mingw32msvc-strip
```

Une fois le **mkspec** créé, nous pouvons commencer les premiers tests en reprenant le programme de test vu précédemment (section 4.1) afin de vérifier, avec un exemple simple, que notre environnement de compilation fonctionne bien.

Comme précédemment, dans le cas de la compilation sous GNU/Linux, pour la première fois, il faut lancer **qmake -project** pour générer un fichier **.pro** qui servira à **qmake** pour faire un Makefile.

L'étape suivante génère un Makefile avec les options adéquates pour compiler un programme exécutable pour Windows.

**qmake -spec win32-cross** suivi de **make** pour créer l'exécutable.

Si la compilation s'est bien terminée, nous pouvons tester notre programme avec la commande suivante comme sur la figure 4.

```
julien@chaos:~/docs/lm/src/Hello-World$ wine debug/Hello-World.exe
```

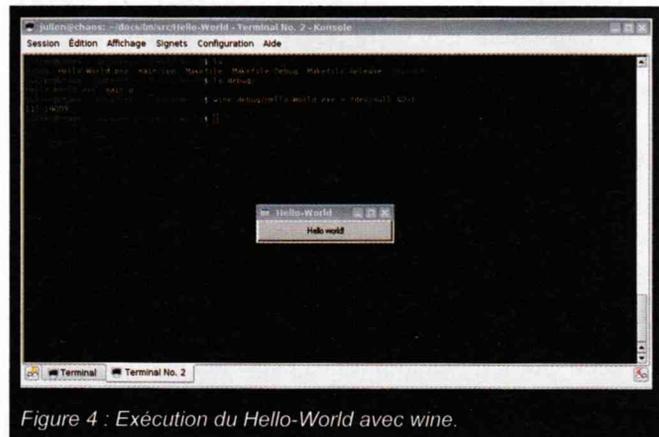


Figure 4 : Exécution du Hello-World avec wine.

## 4.3

### Gestion du port RS232 sous Qt4

Nous allons, afin d'exploiter la capacité de Qt4 à fournir une interface unifiée de communication avec les ports série, utiliser la bibliothèque **qextserialport** disponible à <http://qextserialport.sourceforge.net/>. Pour cela, il nous faut ajouter l'accès à cet objet additionnel dans nos Makefiles. Nous avons vu que le fichier **.pro** servait à **qmake** pour générer les Makefiles automatiquement. Partant d'un **.pro** existant (dans cet exemple, un programme nommé **PhotoAutoCapture**, nous ajoutons les références à **qextserialport** dans les champs appropriés (Code 2).

```
TEMPLATE = app
TARGET = PAC
CONFIG += qt warn_off thread release

HEADERS += \
    Sources/PhotoAutoCapture.h \
    Sources/qextserialbase.h \
    Sources/qserialportwidget.h \
    Sources/qextserialport.h

SOURCES += \
    Sources/main.cpp \
    Sources/PhotoAutoCapture.cpp \
    Sources/qextserialport.cpp \
    Sources/qserialportwidget.cpp \
    Sources/qextserialbase.cpp

linux-g++ {
    HEADERS += Sources/Posix_QextSerialPort.h
    SOURCES += Sources/Posix_QextSerialPort.cpp
    DEFINES += _TTY_POSIX_
    LIBS += -lm
}

win32-cross {
    HEADERS += Sources/win_qextserialport.h
    SOURCES += Sources/Win_QextSerialPort.cpp
    DEFINES += _TTY_WIN_
}
```

Code 2 : Fichier de configuration **.pro** modifié pour appeler la classe **qextserialport** d'accès au port série. Nous y voyons apparaître les deux configurations qui nous intéressent pour la génération d'exécutables pour GNU/Linux et Windows.

Dans le code source, l'accès au port série se fait de la façon suivante (Code 3) :

1. Initialisation du port série (**QextSerialPort \*port = new QextSerialPort();**).

2. Affectation du port `port->setPortName("COM1")`. Nous constatons que cette phase est encore dépendante du système d'exploitation, puisque le nom du port varie selon les OS.

3. Une ouverture du port en lecture seule (`port->open(QIODevice::ReadOnly)`) suivie de sa configuration.

4. La lecture de données provenant du port par `port->read()`.

5. Si, en plus de recevoir des informations du système embarqué, nous désirons lui envoyer des commandes, la méthode `port->write(buf, n)` envoie `n` octets du tableau de caractères `buf`.

Le code Qt4 implémentant ces concepts est présenté ci-dessous :

```
#include "qextserialport.h"
QextSerialPort *port = new QextSerialPort();

/* Ouverture du port */
QString portName;
#ifdef Q_OS_WIN
    portName="COM1";
#else
    portName="/dev/ttyUSB0";
#endif

// L'ordre est important
port->setPortName(portName);
port->open( QIODevice::ReadOnly );
port->setFlowControl(FLOW_OFF);
port->setParity(PAR_NONE);
port->setDataBits(DATA_8);
port->setStopBits(STOP_1);
port->setBaudRate(BAUD57600);

if( port->isOpen( ) ) /* reception de donnees */
{
    int numBytes = port->bytesAvailable();
    if(numBytes > 0) {
        if(numBytes > 512) numBytes = 512;
        char buff[512];
        int i = port->read(buff, numBytes);
        buff[numBytes] = '\0';
    }
}

if( port->isOpen( ) ) /* emission de donnees */
{
    char tmp = 'P';
    int i = port->write(&tmp,1);
}

port->close(); /* fermeture du port */
```

Code 3 : Exemple d'initialisation du port série. Ce code met en pratique la réception de 512 octets, puis l'envoi du caractère P avant de quitter l'application. Le `#ifdef Q_OS_WIN` permet de s'affranchir de l'OS et ainsi d'être portable.

Un exemple concret d'utilisation de ces codes (Fig. 5) est une application graphique de restitution des trames GPS enregistrées par un microcontrôleur, disponible dans notre dépôt subversion par `svn co https://rcs.sequanux.org/gpsqt/tags/stable` (login et mot de passe : anonyme). Ce programme a pour vocation de reproduire – pour une application très précise de récupération et de traitement de trames GPS – les fonctionnalités de `minicom` sous GNU/Linux ou `Hypterterminal` sous Windows, en éliminant pour l'utilisateur les étapes de configuration du port série, puisque nous fournissons des valeurs par défaut « les plus probables ». L'exploitation de ce programme pour une application précise nous a conduit à ajouter des fonctions dédiées à l'exploitation des trames GPS (filtrage, conversion de formats) que nous chercherons à réutiliser dans un maximum de circonstances (section 5).

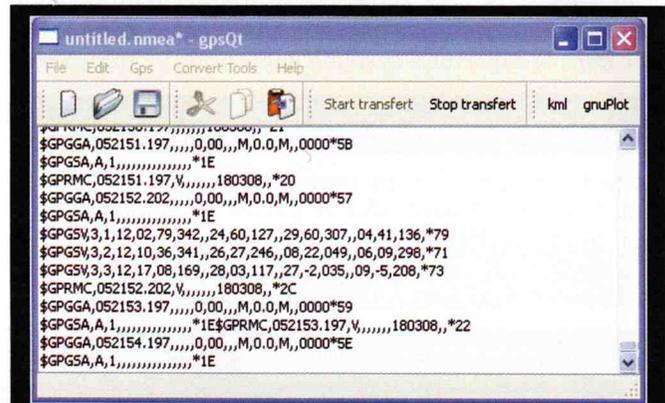


Figure 5 : Exemple d'application chargée uniquement de recevoir des informations du système présenté dans [2] : après une identification du port série le plus probable (section 4.4), une boucle infinie attend les trames GPS qui avaient été stockées en vue d'être restituées à l'utilisateur à l'issue de son trajet.

## 4.4 Détection du port série

Un problème récurrent sous Windows est le changement continu du nom du port série virtuel associé à un système embarqué muni d'un composant FTDI. En effet, alors que sous GNU/Linux le premier périphérique muni d'une telle interface est systématiquement associé à `/dev/ttyUSB0`, le second à `/dev/ttyUSB1...`, sous Windows, nous retrouvons rapidement à manipuler des COM15 ou COM20 lorsque 20 circuits différents munis de 20 composants FT232RL ont été utilisés. Afin d'éviter de rechercher pour chaque nouveau système connecté au PC sous Windows, nous proposons de balayer tous les ports disponibles entre COM3 et COM100, à la recherche de ceux qui sont occupés par un périphérique, et de sélectionner par défaut celui dont le nombre est le plus élevé. Cette stratégie a bien fonctionné, sauf sur un portable équipé d'une interface Bluetooth qui apparaissait comme un port série de nombre très élevé.

```
/* Parcours COM101 à COM2 pour trouver le port de + haut
niveau ouvrable
QStringList gpsQt::testPort()
{
    QString portSerieTmp;
    QStringList portSerieTmpList;
    for (int i = 101 ; i > 2 ; i-- )
    {
        QString str;
        str.setNum(i);
        if ( i < 10 )
            portSerieTmp = "COM"+str;
        else
            portSerieTmp = "\\.\COM"+str;

        port->setPortName(portSerieTmp);
        if ( port->open(QIODevice::ReadOnly) )
        {
            port->close();
            portSerieTmpList<<portSerieTmp;
        }
    }
    return portSerieTmpList;
}

...
#ifdef Q_OS_WIN
    listport = testPort();
#else
    portSerie = "/dev/ttyUSB0"
#endif
```

Notez la syntaxe différente pour les ports COM inférieurs ou supérieurs à 10. Cette syntaxe différente est la cause du dysfonctionnement d'un certain nombre de logiciels sous Windows incapables de gérer les ports supérieurs à 10, car ne tenant pas compte de cette nouvelle syntaxe (par exemple des programmeurs de microcontrôleurs, initialement écrits pour fonctionner sur port série « classique » – généralement COM1 à COM4 – qui ne fonctionnent plus lors du remplacement du MAX232 par un FT232R).

L'application doit ici encore tenir compte explicitement des cas Windows et GNU/Linux, d'où la condition associée à `#ifdef Q_OS_WIN`.

## 4.5

## Acquisition d'un flux continu de données

Afin de récupérer des données à partir d'un port série de façon continue, il existe plusieurs techniques, dont les *threads*, qui lisent en permanence ce qui arrive sur le port série, mais, en parallèle à l'application, et les *timers* qui déclenchent à intervalle régulier la lecture du *buffer*. Ces deux méthodes permettent de recevoir les informations en permanence et de les traiter en même temps.

La classe suivante est un exemple de thread utilisant la classe `QThread` de Qt :

```
//fichier receiverthread.h
#ifdef ReceiverThread_H
#define ReceiverThread_H

#include <QThread>
#include "qextserialport.h"

class ReceiverThread : public QThread
{
public:
    ReceiverThread(QextSerialPort &inPort);
    void run();
private:
    QextSerialPort port;
};

#endif
//fichier receiverthread.cpp
#include <QThread>
#include <iostream>
#include "receiverthread.h"

ReceiverThread::ReceiverThread(QextSerialPort &inPort)
:port(0)
{
    port = inPort;
}

void ReceiverThread::run()
{
    int numBytes = 0;
    int l = 512;
    while (1){
        numBytes = port.bytesAvailable();
        if(numBytes > l) {
            if(numBytes > l) numBytes = l;
            char buff[l];
            qint64 lineLength = port.readLine(buff, sizeof(buff));
            std::cout<<buff<<std::endl; //affichage des donnees
        }
    }
    exec();
}
```

Dans le code de l'application, une fois le port correctement ouvert, il suffit de définir le thread et de le démarrer :

```
#include "receiverthread.h"
[...]
ReceiverThread *receiver = new ReceiverThread(port);
receiver->start();
```

## 4.6

## Application à la communication avec un microcontrôleur

Les systèmes embarqués qui vont nous intéresser désormais doivent être des dispositifs de faible consommation, fonctionnant plusieurs années sur piles, notamment à des fins d'observation de l'environnement. Outre le choix d'un microcontrôleur dédié à ce genre de tâches (MSP430 [4]), la stratégie de base pour augmenter l'autonomie reste toujours de réduire au maximum les ressources afin de ne mettre sous tension que les périphériques utiles.

Par ailleurs, l'utilisateur de tels dispositifs requiert généralement une information datée, soit avec un échantillonnage à intervalle de temps régulier ou une mise en marche à une heure donnée chaque jour. Notre stratégie est donc d'implémenter une horloge temps-réel logicielle dans le MSP430, qui n'est donc réveillé de son sommeil profond (mode LPM3) qu'une fois par seconde.

Le bus USB amène une alimentation de 5 V : en observant l'état de la broche associée sur le connecteur USB du circuit embarqué, nous serons en mesure de savoir si un câble de communication est connecté ou non, et agir en conséquence. Afin de protéger la logique 3,3 V, nous connectons la broche d'alimentation 5 V (broche 1 de l'embase ou fil rouge dans un câble USB) à une broche de port général (GPIO – dans les exemples qui vont suivre, P3.4) du microcontrôleur via une résistance de quelques dizaines de kilohms. Afin de définir l'état de la broche en l'absence de câble, une résistance de tirage un peu plus élevée est placée entre cette même broche et la masse. Si le MSP430 se rend compte qu'un câble USB ou série est connecté, il tentera de communiquer (recevoir des ordres du PC et retransmettre les informations acquises), sinon il se contente de vaquer à ses tâches (vérifier si une condition sur l'horloge est vérifiée afin de lancer une acquisition de données ou de réveiller un autre périphérique plus gourmand en énergie dont le microcontrôleur autorise l'alimentation) avant de se rendormir (Fig. 6).

## 4.7

## Lecture de températures sur MSP430

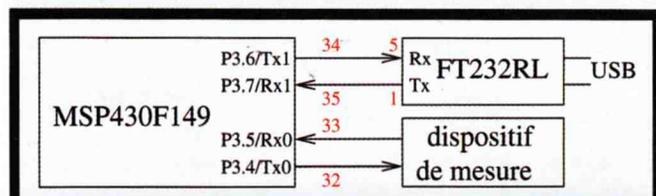


Figure 6 : Circuit de base utilisé lors des mesures présentées par la suite : un microcontrôleur MSP430, nécessitant comme seul composant additionnel un quartz à 32 kHz, communique avec un PC par liaison USB via un convertisseur RS232-USB FT232RL. Le second port série du MSP430 est éventuellement connecté à un autre instrument de mesure pour acquérir des informations additionnelles à la température du microcontrôleur obtenue par mesure de la dérive en tension de la sonde interne. Le FT232RL peut être remplacé par un module XbeePro (<http://www.lextronic.fr/produit.php?id=1319>) si une liaison sans fil est nécessaire. Les chiffres en rouge indiquent les numéros des broches utilisées.

```

...
#define      tmp      R9
#define      tmp2     R8
#define      tmp3     R7
...
; ADC
mov.w      #SHT0_6+REFON+ADC12ON,&ADC12CTL0 ; V_REF=1.5 V
mov.w      #SHP,&ADC12CTL1
mov.b      #INCH_10+SREF_1,&ADC12MCTL0 ; p.17-11: single
conversion
bis.w      #ENC,&ADC12CTL0          ; MUST be .w jmfriedt
...
temperature:
push      tmp
push      tmp2
push      tmp3
mov.w      #0,tmp2
mov.b      #12,tmp3 ; moyenne sur 12 mesures :
12->16 bits
somme:    call #SetupADC12
add.w      tmp,tmp2
dec.b      tmp3
jnz      somme
mov      tmp2,tmp ; envoi de la somme sur 16 bits
swpb      tmp
call      #byte_asc
swpb      tmp
call      #byte_asc
mov.b      #' ',tmp
call      #rs_tx1
rlc.w      tmp2 ; on fait l'hypothese que bit 15 est 1
swpb      tmp2 ; => val = (val+256)*1.5/512 sur 9 bits
mov.b      tmp2,tmp
call      #byte_asc
mov.b      #'\\n',tmp
call      #rs_tx1
mov.b      #'\\r',tmp
call      #rs_tx1
fintemp:
pop      tmp3
pop      tmp2
pop      tmp
ret

```

```

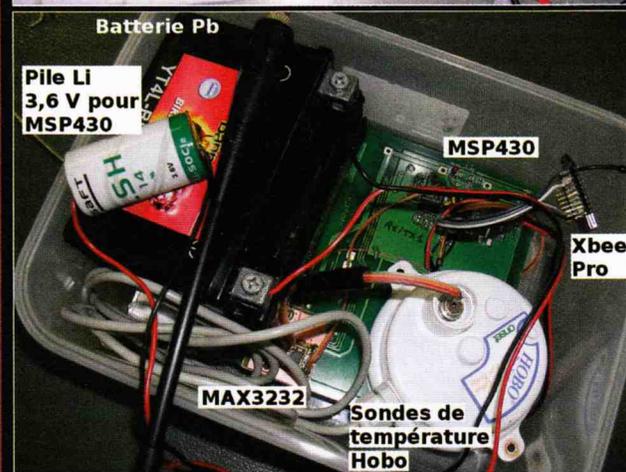
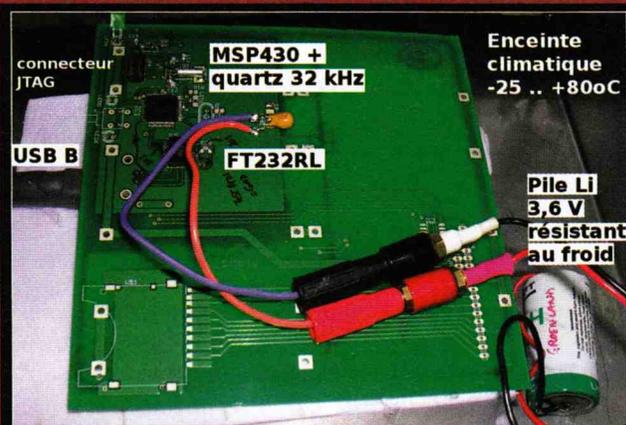
...
SetupADC12: ; book C. Nagy p.90 Vtemp=0.00355(T_C)+0.986
bis.w      #ADC12SC,&ADC12CTL0 ; start conversion
adc:      bit.b #1,&ADC12CTL1 ; wait while ADC busy==1
jnz      adc
mov.w      &ADC12MEM0,tmp ; conversion result
ret

```

À titre d'exemple d'instrument que nous avons interrogé périodiquement par le port série afin d'en communiquer par liaison sans fil le résultat de mesures, mentionnons la station météorologique Hobo ([http://www.onsetcomp.com/solutions/data\\_logger\\_kits](http://www.onsetcomp.com/solutions/data_logger_kits)) communément utilisée lors des mesures de température de l'environnement.

Ce dispositif est muni d'un port série (RS232) pour la communication, qu'il s'agisse de la configuration des intervalles de temps entre deux acquisitions ou de la récupération des données. À l'allumage, ce dispositif communique au débit de 1200 bauds. Bien que le logiciel propriétaire fourni avec ces instruments passe rapidement à une communication haut-débit à 38400 bauds, nous avons choisi de maintenir toutes nos transactions au débit initial, compatible avec les performances en mode faible consommation (seul un résonateur basse tension à 32768 Hz alimente le microcontrôleur) du MSP430. La transaction visant à récupérer les deux mesures de température des sondes internes et externes du Hobo se font comme suit :

1. Initialiser la transaction par les commandes 'D' suivi de 'E'.
2. Pour chaque nouvelle mesure, émettre l'ordre 'C' et attendre la réception de 6 caractères. Ces 6 caractères contiennent, dans le modèle de Hobo à notre disposition, deux mesures de températures codées sur 16 bits chacune.
3. Cette quantité lue est transmise au PC qui se charge d'appliquer les coefficients de calibration obtenus au préalable de la mise en place de l'expérience.

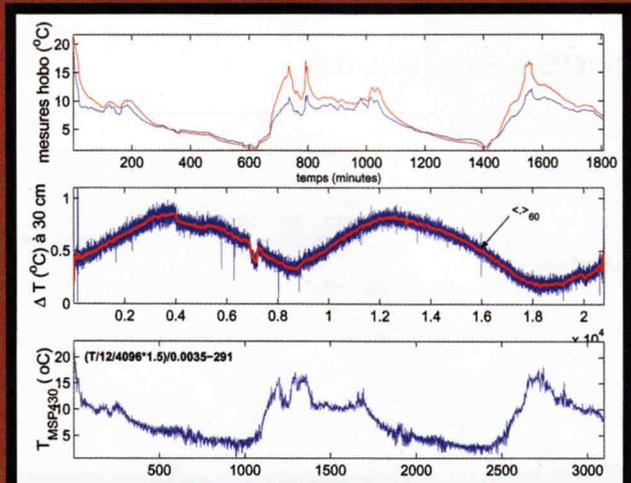


En haut, le montage en enceinte climatique lors de la phase de calibration de la sonde interne de température au MSP430. La même mesure a été faite avec un Hobo connecté au port série du MSP430. En bas, le montage placé en extérieur pour la mesure pendant 1 semaine, en totale autonomie, de la température : chaque minute (déterminée par une horloge temps réelle programmée dans le MSP430), le système se réveille, interroge le Hobo et transmet par liaison Zigbee les températures lues à un PC chargé de la réception, de l'interprétation et de l'affichage des données, tel que nous le décrivons dans ce document. Ce montage a par ailleurs la particularité de récupérer par RS232 les informations d'un montage capable d'interroger un capteur enterré sous quelques dizaines de cm sous terre, alimenté par une batterie au plomb.

Notez le remplacement de la liaison USB utilisée en enceinte climatique par un module Xbee Pro acheté chez Lextronic <http://www.lextronic.fr/produit.php?id=1319> : en plus de connecter directement les broches d'émission et de réception du MSP430 sur les broches 3 et 2 respectivement du Xbee Pro, nous connectons une broche de contrôle de la mise sous tension.

La consommation de ce module est en effet très élevée (plus de 50 mA en mode écoute et plus de 100 mA en mode émission - 1000 fois plus important que la consommation du MSP430 seul) : afin de conserver une autonomie acceptable du montage, nous exploitons la broche de mise en veille SLEEP# (13) du Xbee Pro afin de ne l'alimenter que lors de la communication (une fraction de seconde chaque minute).

Il s'agit là d'une particularité du protocole Zigbee qui le distingue de son concurrent le plus proche, Bluetooth : là où le second protocole met plusieurs secondes à initialiser la communication, avec une consommation électrique associée rédhitoire pour un fonctionnement sur pile avec une bonne autonomie, Zigbee ne met que quelques millisecondes à établir la communication lors de son réveil. Nous n'exploitons ici le Zigbee que pour une liaison point à point, sans utiliser sa capacité à créer un réseau en étoile.



En haut : mesure toutes les minutes de la température lue par les deux sondes du Hobo : ces deux valeurs sur 16 bits sont converties en degrés Celsius suite à une étape de calibration. Milieu : mesure de la température à 30 cm sous terre aux mêmes instants. En bas : transmission de la mesure issue de la sonde de température interne au MSP430F149, translatée afin de correspondre au mieux aux valeurs du Hobo. Ces mesures ont été acquises et transmises par zigbee pendant 2 jours (vieillessement accéléré).

## 4.8 Réception des ordres sur MSP430

Afin de détecter la présence d'un câble USB et ne gérer la communication que dans ce cas, nous connectons la broche alimentée par la tension +5 V sur le câble USB (broche 1) à la broche P3.4 du MSP430 via une résistance de quelques kilohms. La valeur de cette broche est testée à chaque réveil de l'horloge temps réel logicielle programmée dans le MSP430 : la latence entre la connexion d'un câble USB et sa détection est au maximum de 1s. Lorsque le câble est détecté, les broches P3.6 et P3.7 (puisque nous utilisons le second port série) passent de l'état GPIO en entrée vers le mode communication UART. Une fois le câble

retiré, ces broches repassent en mode GPIO en entrée. En effet, le FT232RL n'étant pas alimenté en l'absence de câble USB (Fig. 11 du paragraphe 7.1 de [5] : *Bus powered configuration*), nous avons constaté un courant de fuite de plusieurs mA entre le MSP430 et le FT232RL en l'absence de ces précautions. Il est probable que la broche d'émission (Tx) du MSP430 essaie d'alimenter le FT232RL en l'absence de câble série, alimentation rendue impossible si la broche est en mode « entrée ».

La fonction **comm**, présentée ci-dessous, est appelée chaque seconde lors de l'activation de l'horloge temps réel par l'interruption timer et implémente les fonctions que nous venons de décrire : scruter le port auquel est connectée la ligne d'alimentation du bus USB (dans notre cas, la broche P3.4), et, si la liaison USB est détectée (niveau haut sur le port), alors le microcontrôleur passe les broches associées à la communication en mode UART.

En présence du câble USB, nous passons dans la fonction **recoit** qui teste les caractères reçus : chaque caractère est associé à diverses fonctions dans le code (caractères '?', 'P'...). Notez qu'en l'absence de communication, le microcontrôleur ne bloque pas en attente d'un ordre si le câble est débranché, car la fonction **rs\_rx** incorpore elle-même le test de présence du câble en scrutant la valeur de P3.4. La sortie de cette fonction est donc conditionnée soit par la réception d'un caractère sur le port série, soit par la déconnexion du câble.

```

...
mov.b #0x40,&P3DIR ; P3.6 = output direction
mov.b #0x00,&P3OUT ; P3.6 = output val
...
comm:
; communication par port serie ?
bit.b #0x10,&P3IN ; teste alim du port serie = P3.4
mov.b #0x00,&P3SEL ; P3.6,7 = GPIO select
jnz recoit
ret

recoit:
mov.b #0xC0,&P3SEL ; P3.6,7 = USART select
call #rs_rx
cmp.b #'?',tmp
jne s1
call #dumpall ; renvoie les parametres
jmp comm

s1:
cmp.b #'P',tmp ; programmation parametres
jne s2
call #getall
jmp comm

s2:
cmp.b #'S',tmp ; on ne fait rien !
jne s3

s3:
cmp.b #'T',tmp
jne s4
call #rendtemp

s4:
cmp.b #'U',tmp
jne s5
call #testxmit

s5:
jmp comm ; on n'a pas vu de char connu

rs_rx:
bit.b #URXIFG1,&IFG2 ; lecture port1
jnz fin_rsrx ; char recu
bit #0x10,&P3IN; teste alim du port serie = P3.4
jnz rs_rx
mov #0x20,tmp
ret ; si pas recu de char et pas de cable rs232

fin_rsrx:
; mov.b &RXBUF0,tmp ; lecture port0
mov.b &RXBUF1,tmp ; lecture port1
ret
    
```

## 5

## Utilisation des classes Qt en mode console

Nous avons exploité Qt pour développer un logiciel d'acquisition et de traitement des trames NMEA issues d'un récepteur GPS (Fig. 5). Or, il est fastidieux de graphiquement sélectionner chaque fichier à traiter quand de nombreuses traces GPS ont déjà été acquises : la ligne de commande présente là tout son intérêt dans sa capacité d'appliquer un filtre sur un grand nombre de fichiers dont le nom est fourni en argument (`for i in *; do ./mon_filtre.qft $i > $i.out;done`).

Dans le cadre du développement d'interfaces graphiques avec les bibliothèques Qt, nous développons des classes plus ou moins complexes pour des conversions de données ou de validation de codes correcteurs. Ces classes C++/Qt

peuvent également être utilisées pour une application en ligne de commande : de cette manière nous bénéficions des mêmes performances en ligne de commande et dans notre interface graphique. Par exemple, dans le gestionnaire de trames GPS, il faut cliquer sur *Open a nmea file*, puis sur *convert2kml* et enfin choisir l'emplacement et le nom de son fichier destination pour convertir ses trames. Cette action utilise deux classes distinctes, l'une pour vérifier l'intégrité des données et l'autre pour filtrer et convertir les données. En ligne de commande, le programme utilise les mêmes classes, mais il accepte en argument (`argv[1]`) un nom de fichier, et lance la conversion automatiquement en envoyant le résultat sur la sortie standard.

## 6

## Affichage graphique de données acquises

Afin de compléter l'interface utilisateur graphique permettant de contrôler et recevoir des informations d'un système embarqué, nous désirons afficher en temps réel des courbes présentant l'évolution temporelle des quantités mesurées. Pour ce faire, nous exploitons la bibliothèque `qwt` disponible à <http://qwt.sourceforge.net/>, développée pour simplifier l'inclusion de graphiques dans les applications Qt. Les exemples qui suivent sont basés sur la version 5.0(.2) de Qwt et compilés avec Qt4.3.4.

Un graphique est défini dans une classe nommée `FreqPlot`, dans laquelle sont définies toutes les propriétés et l'apparence du schéma (échelle, titre, axes, fréquence de rafraîchissement de l'affichage...). Cette classe fera appel à une variable statique contenant les données lues sur le port série par le thread afin de compléter la courbe.

```
/* dans le programme */
#include "freqplot.h"
[...]
/* Initialisation d'une nouvelle courbe */
FreqPlot *plot = new FreqPlot();
plot->setTitle("Temperature");
```

Cet objet fait appel à la classe `FreqPlot` qui exploite les objets et les structures fournies par la bibliothèque Qwt :

```
/** classe FreqPlot **/
FreqPlot::FreqPlot(int sensor, int type):
[...]
/* ajout d'une légende : */
QwtLegend *legend = new QwtLegend;
legend->setItemMode(QwtLegend::CheckableItem);
insertLegend(legend, QwtPlot::RightLegend);

/* ajout d'un quadrillage */
QwtPlotGrid *grid = new QwtPlotGrid();
grid->enableX(true);
grid->enableY(true);
grid->setXDiv(*(this->axisScaleDiv(QwtPlot::xBottom)));
grid->setYDiv(*(this->axisScaleDiv(QwtPlot::yLeft)));
grid->attach(this);

/* on n'a pas ajouté le zoom */
[...]
```

```
/* borne min et max en abscisse et ordonnées */
#define HISTORY 600 //seconds
setAxisScale(QwtPlot::yLeft, -50,200);
setAxisScaleDraw(QwtPlot::xBottom, new TimeScaleDraw(My_
Time()));
setAxisScale(QwtPlot::xBottom, 0, HISTORY);

/* initialisation de la courbe (des données)*/
FreqCurve *curve;
curve = new FreqCurve("Temperature");
curve->setColor(Qt::blue);
curve->setStyle(QwtPlotCurve::Lines);
curve->attach(this);
showCurve(curve, true);

/* initialisation du timer qui rafraichira le graphique
*/
refreshTimer = new QTimer();
connect(refreshTimer, SIGNAL(timeout()),
SLOT(timerEvent()));
refreshTimer->start(1000); // 1/s
}

/* SLOT connecté au Timer */
void FreqPlot::timerEvent()
{
//remplissage de curve avec les nouvelles données
[...]
replot();
}
```

Afin de nous affranchir des problèmes d'endianness, nous avons choisi de transférer la valeur 16 bits en ASCII. La lecture du côté Qt se fait en découpant la chaîne de caractères acquise par le port série :

```
bool ok = true;
double celcius, d;
char buff[20];
QStringList l;
qint64 lineLength = port.readLine(buff, sizeof(buff));
QString paquet = buff;
l = paquet.split(' ');
d = l[0].toInt(&ok,16);
if ( ok )
{celcius = (((float) d/12 )/4096 * 1.5 - 0.986)/0.00355;}
```

# Abonnez-vous et Économisez 20%\*

Plus de

\* Sur le prix de vente unitaire France Métropolitaine

Voir page 73



Rendez-vous sur le site <http://www.ed-diamond.com>  
pour découvrir et commander tous nos magazines.

La valeur convertie en température est alors empilée dans la liste `QList<double> temperature` de points à afficher :

```
temperature<<celcius;
```

Le transfert de l'octet de poids le plus fort en premier, afin de respecter la convention de lecture classiquement utilisée en occident de gauche à droite, est sélectionné du côté du MSP430 :

```
swpb R9
call #byte_asc
swpb R9
call #byte_asc
mov.b #' ',tmp
call #rs_tx

byte_asc: ; parametre d'entree : R9
push.w R9
push.w R9
rrc R9 ; quartet fort
rrc R9
rrc R9p
rrc R9
and.w #0x0f,R9
add.w #0x30,R9
cmp.w #0x3a,R9
jlo byte1
add.w #7,R9
byte1: call #rs_tx
pop.w R9 ; quartet faible
and.w #0x0f,R9
add.w #0x30,R9
cmp.w #0x3a,R9
jlo byte2
add.w #7,R9
byte2: call #rs_tx
pop.w R9
ret

rs_tx: bit.b #UTXIFG1,&IFG2
jz rs_tx
mov.b R9,&TXBUF1
ret
```

R9 est un registre sur 16 bits, tandis que la fonction `byte_asc` transfère uniquement l'octet de poids faible du registre qui sert au passage de paramètre.

Le fait de placer un `swpb` avant le premier appel à `byte_asc` permet de transférer l'octet de poids fort en premier.

Du côté de la réception, les chaînes de caractères contenant les valeurs en hexadécimal, séparées d'espaces, sont découpées par la méthode `split(' ');` des chaînes de caractères Qt.

En résumé, nous assemblons toutes ces fonctions dans le code final fonctionnel suivant, dont le résultat est présenté sur la fig. 7.

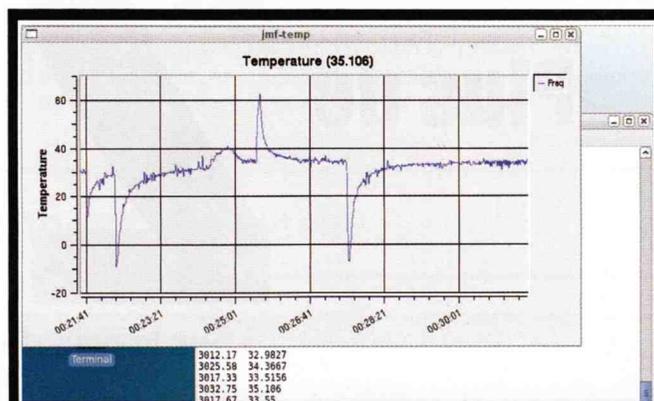


Figure 7 : Affichage de la température (GNU/Linux et X11) issue du MSP430, transférée sous forme de 2 valeurs ASCII (avant et après moyenne des 12 acquisitions). Le capteur a été artificiellement refroidi deux fois et réchauffé une fois.

## 7

## Mise en œuvre sur un système embarqué : la carte Armadeus

Un atout de Qt apparu en cours de développement est la disponibilité d'une version ne nécessitant pas de serveur graphique (X11) et moins de ressources, et donc compatible avec des systèmes embarqués disposant d'un afficheur. Embedded Qt – connu aussi sous son ancien nom QTopia – fonctionne sur la couche `framebuffer` ou sur `svgalib`. La stratégie de cross-compilation d'une application à destination d'ARM (le processeur disponible sur la carte Armadeus) est décrit succinctement à <http://www.armadeus.com/wiki/index.php?title=Qt/Embedded>. Chez TrollTech, <http://doc.trolltech.com/qtopia2.1/html/environment-setup-build.html> propose une autre description de la procédure de cross-compilation.

Nous n'avons pas nous-mêmes mis en place cet environnement de compilation : afin de démontrer la portabilité sur système embarqué de code développé sur PC (GNU/Linux, Intel), nous nous sommes contentés d'envoyer les codes sources de l'application présentée plus haut à F. Burkhart (Armadeus Systems) qui a pu les compiler en l'état et nous communiquer les résultats d'exécution de l'application (Fig. 8). Pour cet exemple, la broche d'émission des données Tx du MSP430 est directement connectée à ttySMX1 (le second port série) de l'APF9328 pour fournir la température lue sur la sonde interne au microcontrôleur.

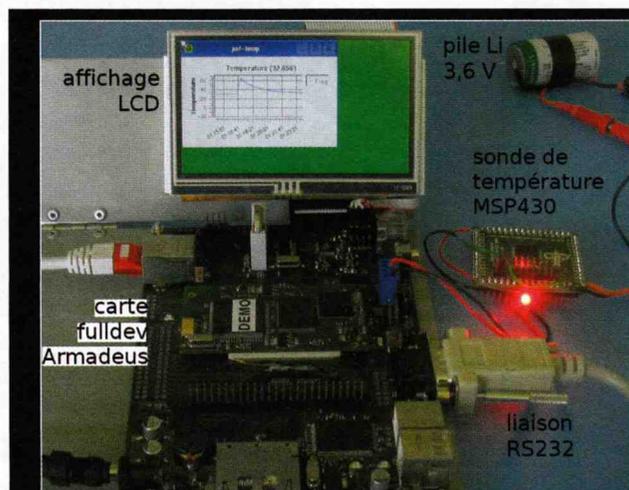


Figure 8 : Photographie de la carte de développement DevFull de Armadeus, équipée d'un circuit APF9128 sur lequel tourne une application développée sous Qt et compilée ici par Embedded Qt (QTopia) pour fonctionner sur le framebuffer. Il s'agit de la même application de mesure de température que présentée précédemment (section 6) (photographie fournie par F. Burkhart, Armadeus Systems).

Nous constatons que l'application se comporte de façon strictement identique sur la carte Armadeus que sur PC fonctionnant sous GNU/Linux ou Windows. La seule modification nécessaire est l'ajout, pour cette configuration, du nom du port de communication, défini dans la série des `#ifdef _TTY_WIN_` de `qextserialbase.cpp` et `posix_qextserialport.cpp` pour fournir un nom approprié :

```
#elif _TTY_ARMADEUS
    portSerie="/dev/ttySMX1";
...
```

(Ce port est en logique 3,3 V sur la DevLight et en  $\pm 12$  V sur la DevFull). Par ailleurs, le fichier de configuration `.pro` est complété pour y définir les variables associées à l'environnement de l'Armadeus :

```
contains( CONFIG , armadeus) {
    message("armadeus")
    TARGET = ./bin/$(OUTPUT_DIR)/temp
    HEADERS += src/Posix_QextSerialPort.h
    SOURCES += src/Posix_QextSerialPort.cpp
    DEFINES += _TTY_POSIX
    DEFINES += _TTY_ARMADEUS
    LIBS += -lm Lib/$(OUTPUT_DIR)/libqwt.a
    OBJECTS_DIR = bin/$(OUTPUT_DIR)/objs/
    MOC_DIR = bin/$(OUTPUT_DIR)/mocs/
    UI_DIR= bin/$(OUTPUT_DIR)/uis/
}
```

Nous aurons pris soin pour la cross-compilation d'installer la *toolchain* appropriée, `QMAKE_CC= arm-linux-gcc`, `QMAKE_CXX= arm-linux-g++...` et d'y faire appel, et de faire appel aux bibliothèques pour applications embarquées : `-lqte -lqte-mt -lqpe -lqtokia`.

## 8 Conclusion

Nous avons présenté un environnement de développement basé sur Qt4 pour réaliser des interfaces graphiques capables de communiquer avec des systèmes embarqués. Nous avons démontré la capacité de cet environnement à générer des binaires pour un système d'exploitation propriétaire (Windows) depuis GNU/Linux afin de fournir une interface commune pour ces deux plateformes. Nous avons focalisé notre travail sur la communication par protocole asynchrone (RS232) avec des systèmes embarqués faible consommation, éventuellement via le convertisseur FTDI FT232 lorsqu'un port série RS232 n'est pas disponible sur le PC.

Les interfaces graphiques proposées incluent l'interaction avec l'utilisateur (envoi de commandes et lecture de paramètres) et l'affichage de courbes acquises par le système embarqué. La portabilité de la solution proposée ne se limite pas aux interfaces graphiques, puisque les bibliothèques de traitement des données ont aussi été exploitées en ligne de commande afin d'automatiser le traitement d'un grand nombre de fichiers déjà disponibles. Enfin, nous avons étendu ces fonctionnalités à l'affichage sur une carte de développement aux ressources modestes comparées à un PC (carte Armadeus), sans serveur X11, afin d'illustrer l'utilisation de nos codes avec Embedded Qt.

### Auteurs : J.-M Friedt, Julien Garcia



J.-M Friedt est ingénieur dans la société Sensor, hébergé par l'institut FEMTO-ST de Besançon, et membre de l'association Projet Aurore pour la diffusion de la culture scientifique et technique ([projetaurore.asso.univ-fcomte.fr/](http://projetaurore.asso.univ-fcomte.fr/)). Il n'aime pas les interfaces graphiques.

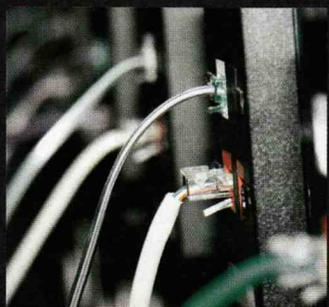


Julien Garcia est technicien, développeur dans le département Temps-Fréquence de l'institut FEMTO-ST de Besançon. Il est membre de l'association Sequanux ([www.sequanux.org](http://www.sequanux.org)) pour la diffusion des logiciels libres en Franche Comté.

## Références

- [1] STEPHENSON (N.), *In the beginning... was the command line*, Harper Perennial, 1999.
- [2] FRIEDT (J.-M.), CARRY (É.), « Acquisition et dissémination de trames GPS à des fins de cartographie libre », *GNU/Linux Magazine France*, hors-série 27, octobre 2006.
- [3] FRIEDT (J.-M.), « Géolocalisation de photographies numériques », *GNU/Linux Magazine France* 96, juillet/août 2007.
- [4] FRIEDT (J.-M.), MASSE (A.), BASSIGNOT (F.), « Les microcontrôleurs MSP430 pour les applications faibles consommations – asservissement d'un oscillateur sur le GPS », *GNU/Linux Magazine France* 98, octobre 2007.
- [5] <http://www.ftdichip.com/Products/FT232R.htm> et la datasheet [http://www.ftdichip.com/Documents/DataSheets/DS\\_FT232R.pdf](http://www.ftdichip.com/Documents/DataSheets/DS_FT232R.pdf)
- [6] BAILLY (Y.), « À la découverte de Qt », *GNU/Linux Magazine France*, février 2002, p. 56-58, suivie d'une série de 20 articles jusqu'à février 2004.
- [7] BODOR (D.), « Création d'un périphérique USB avec supports GNU/Linux et Windows », *GNU/Linux Magazine France* 100, déc. 2007, p. 56-63.
- [8] <http://doc.qtfr.org/post/2007/04/10/Cross-Compilation-Native-dapplication-Qt-depuis-Linux>
- [9] <http://doc.qtfr.org/post/2007/04/10/Cross-Compilation-Native-dapplication-Qt-depuis-Linux>

# IPCop sur Soekris net4801



*Beaucoup d'entre nous ont recyclé de vieux PC en routeurs/firewalls domestiques, voire d'entreprise. Les problèmes remontés sont la consommation électrique, le bruit et l'encombrement. Sans compter qu'en cas de besoin réel d'un routeur, on n'a pas forcément de « vieux PC » sous la main.*

## 1 Les présentations

### 1.1 Hardware

On peut alors se tourner vers d'onnéreuses solutions dédiées ou faire des choix plus exotiques, moins chers et bien plus excitants. Parmi les possibilités qui s'offrent à nous, on trouve les cartes de PC Engines, dont la célèbre WRAP. Cette dernière est à présent arrivée en fin de vie et est remplacée par un modèle plus puissant baptisé Alix. Elles sont d'ores et déjà disponibles.

D'autres solutions techniquement envoûtantes sont celles proposées par la société californienne Soekris, principalement connue pour ces cartes **net4501** et **net4801** à base de processeurs Geode. La dernière née de la famille est la net5501, version musclée de la net4801. C'est néanmoins cette dernière que j'ai choisi de présenter, car elle représente à mes yeux la juste puissance pour un usage domestique ou dans les petites entreprises.

La net4801 offre 3 interfaces réseau, un port USB, un slot Compact Flash, un connecteur miniPCI, un port IDE, deux ports série et une poignée d'entrées/sorties programmables. Il existe des cartes filles pouvant augmenter le nombre d'interfaces réseau (jusqu'à 7 au total).

### 1.2 Software

Le net4801 ne sait pas *booter* sur une clé USB et c'est là son principal défaut à mes yeux. Lui adjoindre un disque dur risquerait d'augmenter le bruit et la chauffe. On décide donc d'utiliser la carte CF comme support de stockage principal. Cette dernière est vue comme un disque dur et il est possible de booter dessus.

Pour celui qui cherche une solution rapidement, stable, et sans vouloir jouer plus avant avec le Soekris, il est possible de transférer, grâce à **dd** par exemple, des images téléchargées sur internet dans une CF et d'avoir directement un routeur Soekris fonctionnel. Citons par exemple l'excellent **monowall** qui utilise une base FreeBSD ou EmbCOP, maintenu à jour régulièrement et proposant de nombreuses images en fonction de la taille de votre carte CF. Pour les explorateurs, on dispose surtout de la possibilité de booter via le réseau. Ça tombe très bien ! Depuis la version 1.4.11, **IPCop** est installable grâce à **PXElinux** et un serveur web/FTP (ou une clé USB) utilisé en stockage secondaire.

Voilà un couple bien assorti... à quelques menus détails près... comme nous allons le voir dans la suite de l'article présenté sous la forme d'une suite d'expériences plutôt que d'un how-to magistral.

## 2 Préliminaires avec le net4801

Dans un premier temps, assurez-vous qu'aucune carte Compact Flash n'est insérée dans le net4801. Cette partie ne viendra que plus tard, il ne faut pas être impatient.

### 2.1 Liaison série

Il faut commencer par installer un environnement permettant la communication avec le net4801.

Le plus simple consiste en l'utilisation de son port série.

La communication se fait via un câble série croisé (Null Modem) en utilisant une vitesse de **19200bauds, 8 bits de données, aucun de parité et 1 bit de stop** (habituellement noté **19200 8N1**). Disposant d'un ordinateur « moderne », je n'ai bien sûr pas de vrai port série. Un adaptateur USB à base de **pl2303**

en fournit un pour un coup modeste et est parfaitement reconnu par Linux. Il est pris en charge par les modules **usbserial** et **pl2303** et crée un port série **/dev/ttyUSB0**. Théoriquement, tout cela est automatique lors d'un branchement « à chaud ».

Consultez la sortie de la commande **dmesg** pour détecter les problèmes éventuels ou l'utilisation de noms de périphérique non standards.

Choisissez le logiciel qui vous convient pour le dialogue sur le port série. J'avais, pour ma part, choisi **minicom** dans le cadre de ce projet. Toutefois, l'affichage des boîtes d'installation n'était pas agréable. J'ai donc choisi de remplacer minicom par **gtkterm** pour la partie « Installation ».

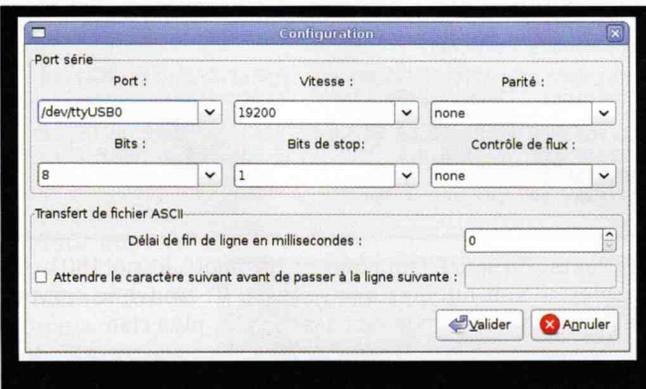
Vous pouvez lancer minicom avec l'option **-s** pour le configurer. L'interface **ncurses** est simple, je ne détaillerai donc pas la configuration de minicom ici. À l'issue de la configuration, vous devriez avoir un fichier **.minirc.dfl** ressemblant fortement à celui-ci :

```
pu port           /dev/ttyUSB0
pu baudrate      19200
pu bits          8
pu parity        N
pu stopbits      1
pu rtscts        No
```

Lancez alors **minicom -o**. L'option **-o** évite que minicom tente d'initialiser un hypothétique modem à l'autre bout du cordon série.

Pour quitter minicom, utilisez la combinaison de touches **[Ctrl]+[A]** puis **[Q]** et valider la réponse « oui » à la question « Quitter sans remettre à zéro ? ».

Pour **gtkterm**, il suffit de configurer le port grâce à l'entrée **Port** du menu **Configuration** avant de brancher le Soekris.



## 2.2 Le BIOS monitor

Il est à présent temps de connecter un câble Null Modem à l'ordinateur et au net4801 et de brancher ce dernier sur le secteur. Si tout s'est déroulé correctement, vous devriez voir apparaître les messages du BIOS du net4801 (comBIOS). Tapez **[Ctrl]+[P]** pendant qu'il en est encore temps pour accéder au « moniteur » de comBIOS.

```
> show
ConSpeed = 19200
ConLock = Enabled
ConMute = Disabled
BIOSentry = Enabled
PCIROMS = Enabled
PXEBoot = Enabled
FLASH = Primary
BootDelay = 2
FastBoot = Disabled
BootPartition = Disabled
BootDrive = 80 81 F0 FF
ShowPCI = Enabled
Reset = Hard
```

La configuration actuelle est affichée via la commande **show**. Elle permet de vérifier l'état de la configuration, ce qui peut être utile, notamment dans le cas d'un achat de matériel d'occasion, pour vérifier la possibilité de booter via PXE par exemple (**PXEBoot**). Le cas échéant, on peut modifier les paramètres grâce à la commande **set**. Pour en savoir plus, vous pouvez consulter la documentation en ligne sur le site de Soekris et/ou utiliser la commande **? ou help**.

Notez également les valeurs suivant le paramètre **BootDrive**. En cas de boot depuis le moniteur, on passe à la commande **boot** une des valeurs qui suivent, **80** correspondant à la carte CompactFlash et **F0** à un boot réseau sur la première interface réseau.

Vous pouvez pour l'instant éteindre le net4801 et quitter minicom.

## 2.3

### Mise en place du nécessaire pour PXE

#### 2.3.1 Serveur DHCP

Pour booter via PXE, vous devez disposer d'un serveur **dhcp**. L'installation en est évidente sur la plupart des distributions courantes. À titre d'exemple, sur une Ubuntu, il faut installer le paquet **dhcp**.

```
$ sudo aptitude install dhcp
```

On configure notre interface réseau. On lui affecte l'IP 192.168.0.254 par exemple.

```
$ sudo ifconfig eth0 192.168.0.254
```

On peut alors déclarer un **subnet** pour le serveur **dhcp** dans le fichier **/etc/dhcpd.conf**.

```
subnet 192.168.0.0 netmask 255.255.255.0 {
    range 192.168.0.1 192.168.0.100;
    option broadcast-address 192.168.0.255;
    option routers 192.168.0.254;
    filename "/pxelinux.0";
    max-lease-time 60;
    default-lease-time 60;
}
```

On se déclare prêt à servir une centaine d'adresses IP aux clients potentiels. Contrairement à ce qu'on voit écrit

ici ou là, les lignes **allow bootp** et **allow booting** ne sont en aucun cas nécessaires. Quant à la directive **next-server**, elle n'est nécessaire que si on utilise deux serveurs différents pour distribuer les adresses IP et les fichiers de boot PXE.

On pourrait utiliser une déclaration **host** plutôt que **subnet**, mais le serveur DHCP étant dans ce cas dédié au Soekris, c'est inutile.

Reste à redémarrer le serveur dhcp grâce à une commande semblable à celle ci-dessous.

```
$ sudo /etc/init.d/dhcp restart
Stopping DHCP server: dhcpd.
Starting DHCP server: dhcpd.
```

Les plus bourrins pourront par exemple utiliser :

```
$ sudo killall dhcpd
$ sudo dhcpd
```

Attention toutefois, si vous choisissez de démarrer le serveur dhcpd à la main, vous devrez spécifier l'interface réseau concernée en paramètre (**dhcpd eth0**) ou insérer une déclaration de **subnet** vide pour toutes les interfaces actives dans le fichier de configuration :

```
subnet 192.168.1.0 netmask 255.255.255.0 {
}
```

### 2.3.2 Serveur TFTP

Avant tout, une précision nécessaire. Les protocoles TFTP et FTP sont deux protocoles différents. On ne peut donc pas se connecter à un serveur FTP avec un client TFTP et vice et versa. On voit beaucoup de confusion sur le sujet, ne vous laissez pas avoir.

L'installation n'est pas plus compliquée que dans le cas du serveur dhcp. Il s'agit d'installer le paquet **tftpd-hpa** :

```
$ sudo aptitude install tftpd-hpa
```

Sous Ubuntu, la configuration du serveur TFTP consiste à renseigner les valeurs suivantes dans le fichier **/etc/default/tftpd-hpa** (ce sont théoriquement les valeurs par défaut) :

```
RUN_DAEMON="no"
OPTIONS="-l -s /var/lib/tftpboot"
```

Vous n'avez pas à lancer le serveur TFTP, il fonctionne avec **inetd**, à moins que vous ne passiez la variable **RUN\_DAEMON** à « yes ». Si tel est le cas, il vous faudra le démarrer automatiquement via la commande suivante :

```
$ sudo /etc/init.d/tftpd-hpa restart
```

Il nous faut récupérer et mettre en place les fichiers nécessaires pour l'installation d'IPCop, à savoir ceux contenus dans l'archive **ipcop-X.Y.Z-install-pxe.i386.tgz**.

```
$ sudo -s
...
# cd /var/lib/tftpboot
# wget http://heanet.dl.sourceforge.net/sourceforge/ipcop/ipcop-1.4.18-install-pxe.i386.tgz
...
# tar xvzf ipcop-1.4.18-install-pxe.i386.tgz
1.4.18/
1.4.18/instrout.gz
1.4.18/f3.txt
1.4.18/README-ipcop-pxe
1.4.18/memtest
1.4.18/vmlinuz
1.4.18/f2.txt
1.4.18/f1.txt
pxelinux.0
pxelinux.cfg/
pxelinux.cfg/ipcop-pxe-1.4.18.model
# mv pxelinux.cfg/ipcop-pxe-1.4.18.model pxelinux.cfg/default
```

On décompresse cette archive dans un répertoire **/var/lib/tftpboot**. L'arborescence est prête à être utilisée à une exception près. On doit renommer le fichier de configuration. PXELinux permet d'utiliser un nommage de fichier de configuration permettant d'en utiliser de différents en fonction des adresses IP ou des adresses MAC. Je vous renvoie à la page de présentation de PXELinux pour les détails de nommage. Dans notre cas, fort simple, pour que tous les périphériques utilisent le même fichier, on doit le renommer en **default**.

On édite ensuite ce fichier pour le simplifier au maximum et en obtenir une version simple et épurée.

```
DEFAULT 1.4.18/vmlinuz
APPEND ide=nodma initrd=1.4.18/instrout.gz root=/dev/ram0 rw
```

## 2.4 Premier test

```
NSC DP83815/DP83816 Fast Ethernet UNDI, v1.03

Copyright (C) 2002, 2003 National Semiconductor Corporation
All rights reserved.

Pre-boot eXecution Environment PXE-2.0 (build 082)
Copyright (C) 1997-2000 Intel Corporation

CLIENT MAC ADDR: 00 00 24 C8 BF 88
CLIENT IP: 192.168.0.1 MASK: 255.255.255.0 DHCP IP: 192.168.0.254
GATEWAY IP: 192.168.0.254
...
```

Contents, on lance minicom et on boote le net4801. Ce dernier se voit affecter une adresse IP, trouve le serveur TFTP, tout semble bien se passer, puis, plus rien ...

### Note

L'affichage des messages pendant la phase TFTP se fait sur 15 caractères. Je n'ai pas trouvé la cause de ce bug. L'affichage est ensuite correct.

Il nous manque effectivement une phase de configuration ! Le **kernel** doit envoyer ses messages sur le port série pour que l'on puisse suivre les phases de boot. On édite alors le fichier **default** pour adapter les options comme ci-après :

```
APPEND console=ttyS0,19200n81 ide=nodma initrd=1.4.18/
instroot.gz root=/dev/ram0 rw
```

On reconnaît le port série et sa vitesse suivie des paramètres du port. Notez au passage que le port série s'en trouve configuré pour la suite.

## 2.5 Deuxième test

Cette fois ci, on obtient les messages du kernel, mais on est à nouveau aveugles une fois qu'il passe la main à **init**. Nous allons donc examiner le contenu d'**instroot.gz**. On se crée un répertoire permettant de monter l'image *zippée* dans l'archive **instroot.gz**.

```
# cd /var/lib/tftpbboot/1.4.18/
# mkdir loopmount
# gunzip instroot.gz
# mount -o loop instroot loopmount/
```

On va modifier les programmes lancés par **/etc/inittab**. On commente la ligne lançant **login** sur **tty3**. On remplace les **tty1**, habituellement premier terminal virtuel par **ttyS0**, notre console série.

**tty2** est utilisé pour afficher une trace de ce que fait le programme **install**. On en redirige la sortie vers **/dev/null**.

Le programme **iowrap** est un utilitaire de redirection de port, ne vous en souciez pas. Le vrai programme utile lancé est **install**, via **ash**, une *shell*.

## Note

Vous devez éditer **loopmount/etc/inittab** et non **/etc/inittab** sur votre machine !

Au final, le fichier **inittab** sera le suivant :

```
# System initialization.
::sysinit:/etc/rc

# Run gettys in standard runlevels
ttyS0::respawn:/bin/iowrap /dev/ttyS0 /bin/ash --login
-c "/bin/install /dev/null"
#tty3::respawn:/bin/iowrap /dev/tty3 /bin/ash --login

# Stuff to do before rebooting
::ctrlaltdel:/etc/halt
::shutdown:/etc/halt
```

Enfin, il nous faut éditer le fichier **loopmount/etc/rc** pour adapter le lancement de **syslogd**. Comme on ne veut pas être gêné par les messages système pendant l'installation, on choisit de rediriger les *logs* vers **/dev/null** où l'espace disque est largement suffisant pour les accueillir.

```
echo "Starting syslogd"
syslogd -L -0 /dev/null
```

On peut ensuite démonter l'image **instroot**, la re-zipper et rebooter le **soekris**.

```
# umount loopmount
# gzip instroot
```

Un troisième test et...

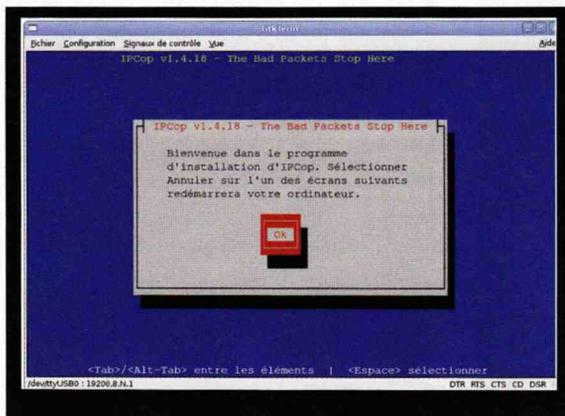
Victoire ! On peut passer à la phase suivante !

## 3 Installation

Il est temps si ce n'était pas déjà fait de mettre une carte Compact Flash dans le réceptacle prévu à cet effet sur la carte Soekris.

On est alors confronté à un problème. Le Soekris tente de booter sur la carte CF. Il nous faut entrer dans le moniteur du BIOS avant le boot grâce à la combinaison de touches [Ctrl]+[P], puis demander expressément un boot réseau grâce à la commande **boot F0**.

Il reste ensuite à choisir le type d'installation à effectuer. Il y a le choix entre une installation depuis la clé USB ou depuis un serveur HTTP ou FTP. Si on dispose d'un serveur HTTP sur la machine utilisée, on peut choisir ce moyen, sinon, il suffit d'utiliser une clé.



### 3.1

## Cas d'une installation depuis un serveur HTTP

Il faut télécharger les fichiers d'installation et les placer dans l'arborescence du serveur afin qu'ils soient accessibles via une adresse pratique (il faudra la saisir pendant l'installation). On monte ici l'iso du CD d'installation dans un répertoire **/var/www/ipcop**.

```
# cd /var/www
# wget http://kent.dl.sourceforge.net/sourceforge/
ipcop/ipcop-1.4.18-install-cd.i386.iso
# mkdir ipcop
# mount -o loop ipcop-1.4.18-install-cd.i386.iso ipcop
```

Lorsque la question « Sélectionner le support pour l'installation » apparaît, choisissez **HTTP/FTP**.

Lors de la configuration du réseau, il est possible de laisser l'installateur rechercher la carte réseau. Il s'agit d'une **National Semiconductor DP8381x**. On peut ensuite lui affecter l'adresse IP 192.168.0.253 par exemple.

Lorsque vient la question de l'adresse HTTP où aller chercher les fichiers d'installation, il faut saisir **http://192.168.0.254/ipcop**, puis valider. Les

diodes **Net** et **Disk** devraient alors s'affoler gentiment pour signifier qu'il « se passe des trucs ».

## 3.2

## Cas d'une installation depuis une clé USB

Il est nécessaire pour ce type d'installation de préparer la clé. Heureusement, le projet IPCop a mis à disposition des images toutes faites.

Celle que l'on choisit est l'image **usb-hdd**. Elle doit être brutalement envoyée sur la clé, grâce à l'ami **dd** par exemple.

Le périphérique correspondant à la clé doit être présent, mais non monté. Chez moi, il s'agit de **/dev/sdf**, ce qui est cocasse pour un périphérique amovible, vous en conviendrez.

```
# wget http://heanet.dl.sourceforge.net/sourceforge/
ipcop/ipcop-1.4.18-install-usb-hdd.i386.img.gz
# gunzip icop-1.4.18-install-usb-hdd.i386.img.gz
# dd if=ipcop-1.4.18-install-usb-hdd.i386.img of=/dev/sdf bs=1M
# eject /dev/sdf
```

Reste à mettre la clé dans le port USB du Soekris et à le rebooter pour pouvoir installer IPCop.

## 3.3

## Poursuivons

Une erreur apparaît à la fin du processus. On n'en tient pas compte pour l'instant.

Le redémarrage suivant apporte son lot de déception. En effet, on n'y voit rien. On a à peine le temps de voir que Grub va se lancer, puis rien.

## 4

## Configuration

Ce premier boot est relativement long, notamment à cause de la génération des paires de clés pour SSH. Toujours est-il que vous devriez tout de même rapidement avoir une invite de login :

```
IPCop v1.4.18 - The Bad Packets Stop Here
(none) login:
```

Euh... C'est quoi déjà le mot de passe *root* ? Théoriquement, arrivé là, on se rappelle que d'habitude, il y a plus de questions lors de l'installation d'IPCop.

Un rapide coup d'œil dans les sources d'**install** confirme qu'il aurait dû y avoir lancement d'un programme supplémentaire nommé **setup** avant le redémarrage. Malheureusement, ce dernier est lancé pour loguer dans **/dev/tty2**, ce qui ne risque pas d'arriver sur le Soekris...

```
system("/bin/chroot /harddisk /usr/local/sbin/setup /
dev/tty2 INSTALL")
```

Un parcours des sources de **setup** achève de confirmer le problème, (éloignez les enfants, c'est un poil brutal) :

```
if (!(flog = fopen(mylog, "w+")))
{
printf("Couldn't open log terminal\n");
```

Forts des expériences précédentes, on décide d'indiquer à Grub qu'il doit envoyer ses messages sur le port série. On fait immédiatement de même pour le kernel.

Enfin, on va éditer l'**inittab** du système installé pour qu'il puisse accepter une connexion sur la console série.

On commence par épurer le fichier de configuration de Grub :

```
serial --speed=19200
terminal serial
timeout 1
title IPCop
root (hd0,0)
kernel /vmlinuz console=ttyS0,19200n81 ide=nodma
root=/dev/hda4 panic=10 acpi=off ro
```

Les deux premières lignes servent à configurer l'utilisation du port série. Suit la seule entrée du fichier au sein de laquelle on retrouve dans les options le paramétrage du kernel lui-même pour l'affichage des messages de démarrage via le port série.

Reste à permettre le *login* via le port série. Cela se paramètre dans le fichier **/etc/inittab**. Il faut ajouter une entrée et commenter les entrées existantes. Théoriquement, **mingetty** ne prend pas en charge les ports série. Toutefois, comme ce dernier a été configuré auparavant (via Grub et le kernel), cela fonctionne :

```
S0:2345:respawn:/sbin/mingetty ttyS0
#1:2345:respawn:/sbin/mingetty tty1
#2:2345:respawn:/sbin/mingetty tty2
#3:2345:respawn:/sbin/mingetty tty3
#4:2345:respawn:/sbin/mingetty tty4
#5:2345:respawn:/sbin/mingetty tty5
#6:2345:respawn:/sbin/mingetty tty6
```

Naturellement, il ne faut pas oublier d'ajouter **ttys0** dans le fichier **/etc/securetty**.

Il reste donc à rebooter le Soekris pour pouvoir terminer l'installation.

```
return 1;
}
```

Donc, setup a bien été lancé, mais s'est arrêté immédiatement. Nous laissant avec un superbe routeur non configuré dont on n'a pas le mot de passe root.

Plusieurs choix s'offrent à nous :

- Hacker les sources d'install et setup et faire profiter le monde de nos améliorations potentielles.
- Décompresser l'archive **.tar.gz** servant à l'installation, y supprimer **dev/tty2** pour le remplacer par un lien vers **dev/null**.
- Se débrouiller avec ce qu'on a.

À l'occasion, participer au projet IPCop pourrait être sympathique, mais on va plutôt faire dans le bourrin et se contenter de ce qu'on a et s'arranger pour reprendre manuellement là où on en était.

Comme on n'a pas de mot de passe pour le root, on va repartir en *single*. Pour cela, on doit surveiller le boot dans **GtkTerm** (ou minicom, kermit, ...) et rapidement presser la touche [e] lors de l'apparition du menu de Grub. On se trouve alors présenté aux commandes exécutées. On

doit éditer la seconde, celle qui commence par le mot-clé **kernel**. Pressez [e] à nouveau, saisissez le mot-clé **single** et validez le changement grâce à la touche [Entrée]. Si vous sortez du mode d'édition en utilisant la touche [Echap], cette modification ne sera pas prise en compte. Lancez le boot en pressant [b]. Quelques secondes plus tard, vous devriez avoir un shell dans lequel il est possible de modifier le mot de passe de l'ami root.

```
bash-2.05b# passwd
Changing password for root
Enter the new password (minimum of 5, maximum of 127 characters)
Please use a combination of upper and lower case letters and numbers.
New password:
Re-enter new password:
Password changed.
```

Ça y est, on est chez nous. Pour avoir l'esprit tranquille, on peut redémarrer avant de finir l'installation.

Loguez-vous en tant que root lorsque vous y êtes invité et exécutez la suite de l'installation :

```
# setup
```

Vous pouvez alors (par exemple) passer en clavier français, sélectionner le fuseau horaire Europe/Paris, sélectionner un nom d'hôte (ipcop) et un domaine (athome).

Les personnes équipées d'une liaison RNIS peuvent la configurer sous l'entrée **ISDN Configuration**.

Choisissez ensuite les mots de passe pour les utilisateurs **admin** et **backup**.

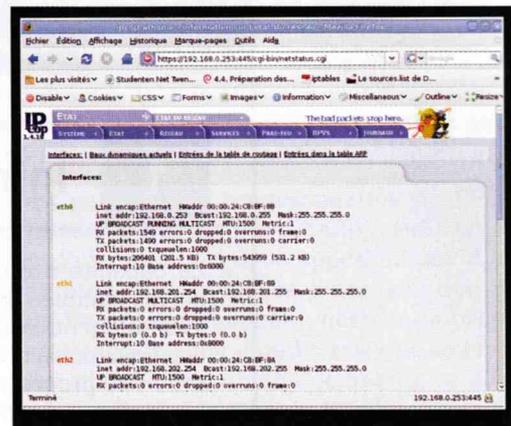
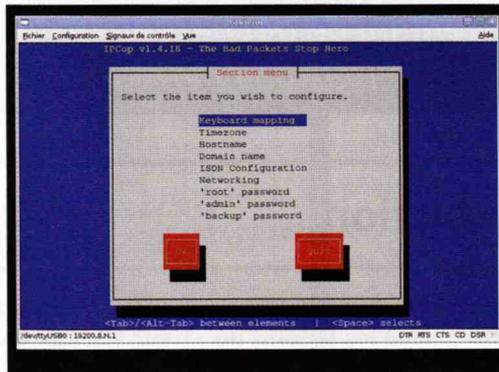
Enfin, attaquez-vous à la partie primordiale de la configuration, le réseau. Ici, je n'ai nul conseil à vous donner. Le Soekris se configure comme un PC comportant 3 interfaces réseau. Choisissez donc votre topologie réseau et affectez des *drivers* aux cartes réseau (Pour rappel : National Semiconductor

DP8381x pour tout le monde). Il ne vous reste plus alors qu'à affecter des adresses IP à toutes ces interfaces.

Le paramétrage de l'interface **GREEN** n'est pas pris en compte, vraisemblablement à cause de la fin violente de la précédente phase. On doit donc ajouter des variables au fichier **/var/ipcop/ethernet/settings** :

```
GREEN_DEV=eth0
GREEN_DRIVER=natsemi
GREEN_DRIVER_OPTIONS=
GREEN_DISPLAYDRIVER=natsemi
```

Suite à cela, votre IPCop devrait être fonctionnel. Si tel est le cas, je vous invite chaudement à garder une image de votre carte compact flash. Elle vous permettra d'installer des batteries de Soekris en moins de temps qu'il n'en faut pour le dire (faisons abstraction du temps pris par **dd** pour remplir une CF :)).



Un redémarrage (ou un *reload* réseau) plus tard, vous pouvez vous connecter à votre Soekris via son adresse IP sur **GREEN/eth0** sur le port **81** ou sur le port **445**, <http://192.168.0.253:81/> ou <https://192.168.0.253:445/> par exemple.

Ici, grâce au mot de passe admin, vous pourrez configurer votre IPCop, juste après avoir été visiter le menu **System/GUI Settings** pour repasser l'interface en français.

## 5 Conclusion

Cette installation pleine de rebondissements a permis d'installer un IPCop sur un Soekris, mais ça a surtout été l'occasion de découvrir ou de revoir les notions qui reviennent souvent en matière de développement embarqué ou d'intégration de matériel « exotique ».

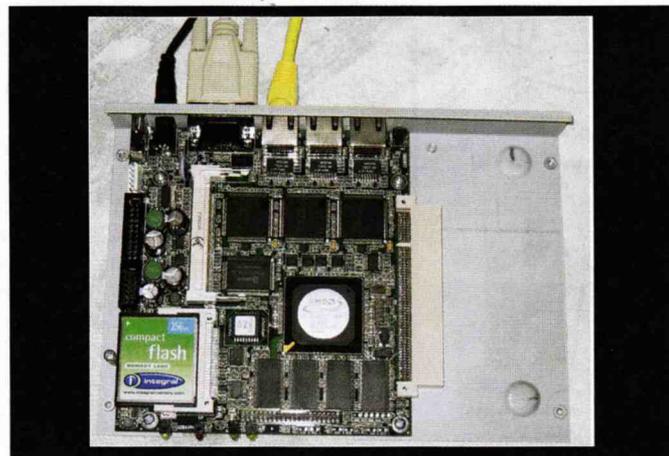
Vous pouvez ensuite modifier cette base de travail, supprimer les logs, refaire un partitionnement plus adapté, installer des *add-on* supplémentaires ou tenter l'aventure sur un autre matériel, une carte Alix par exemple.

Auteur : Xavier Garreau

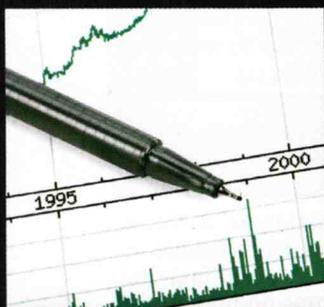
### Liens et références

- Soekris Linux : <http://www.soekris.com/>
- Cortex Systems : <http://www.cortexsystems.dk/>
- IPCop : <http://ipcop.org/>
- MonoWall : <http://mOn0.ch/wall/>
- EmbCop : [http://en.embcop.org/?page\\_id=21](http://en.embcop.org/?page_id=21)

- PC Engines : <http://www.pcengines.ch/>
- PXELinux : <http://syslinux.zytor.com/pxe.php>



# Instrumentation scientifique



*Nous proposons d'explorer pas à pas l'utilisation d'une carte comprenant un FPGA et un processeur pour le développement de systèmes de mesures reconfigurables. Nous allons appliquer les compétences acquises à la réalisation et caractérisation d'un compteur de fréquence pour la mesure d'oscillateurs. Une illustration plus complexe des performances du couple FPGA+processeur sera l'acquisition d'images et affichage sur écran LCD en tentant de décharger au maximum le processeur par l'exploitation du FPGA comme module autonome d'acquisition.*

## 1 Introduction

La capacité à reconfigurer une matrice de portes logiques (FPGA<sup>1</sup>) selon les besoins de l'utilisateur ouvre de vastes possibilités dans de nombreux domaines, qu'il s'agisse d'informatique avec des processeurs reconfigurables selon les tâches exécutées [1] ou de matériel qui s'adapte aux demandes de l'utilisateur [2] (configuration du FPGA comme coprocesseur vidéo [3], traitement du signal ou pour des opérations de calcul flottant par exemple [4, 5]). Nous allons ici nous intéresser à l'utilisation de FPGA à des fins de prototypage d'instruments de mesure scientifique. L'intérêt du FPGA consiste alors dans le coût réduit du matériel nécessaire lors de la phase de développement

(notamment en comparaison avec la réalisation sur silicium d'un composant avec les mêmes fonctionnalités) et à la compatibilité des outils de développement avec la fabrication de circuits dédiés sur silicium (ASIC<sup>2</sup>).

Nous allons pour ces tests utiliser la plateforme de développement proposée par l'association Armadeus Project [6] (nommée Armadeus ultérieurement), qui associe la souplesse du FPGA à un processeur ARM9 fonctionnant sous GNU/Linux, et grâce auquel nous pourrons donc rapidement contrôler notre instrument, récupérer et traiter des informations, pour ensuite les communiquer à l'utilisateur.

## 2 Outils de développement sur Armadeus

La carte DevLight proposée par Armadeus inclut les divers connecteurs (difficiles à trouver par ailleurs) pour exploiter pleinement la carte mère dénommée APF9328 que nous utiliserons au cours des nos développements. Cette dernière inclut un processeur, basé sur une architecture ARM9, cadencé à 200 MHz (comportant lui-même un grand nombre de périphériques embarqués, les seuls dont nous ferons usage étant le contrôleur Ethernet, l'UART pour la communication RS232, et l'interface de communication I<sup>2</sup>C), et un FPGA Xilinx Spartan3 de 200 kportes.

L'utilisation de l'architecture Xilinx a son importance quant à la disponibilité d'outils gratuits

(à défaut d'être libres) de développements, de consommation et de vitesse de fonctionnement. En effet, un des inconvénients majeurs des FPGA pour les applications embarquées est leur consommation élevée.

### 2.1

### Installation des outils de développement pour ARM9

Une *toolchain* est l'ensemble des outils nécessaires à la compilation de binaires à destination d'une architecture :

## reconfigurable

- La chaîne compilateur-éditeur de liens tenant compte de l'agencement de la mémoire sur la plateforme cible : le trio gcc, binutils et newlib est généralement nécessaire pour compiler une toolchain fonctionnelle,
- Éventuellement un débogueur si la plateforme cible le supporte. Sous GNU/Linux, il s'agira généralement du couple client et serveur de **gdb**.
- Les bibliothèques associées afin de fournir les outils de base (**libc** dans le cas de **gcc**), ainsi que diverses bibliothèques annexes répondant aux besoins du développeur.

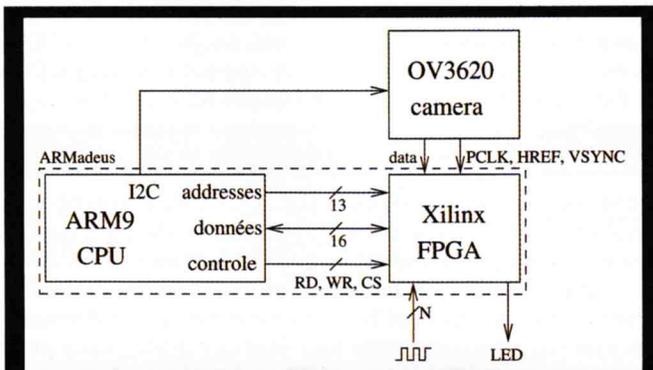


Figure 1 : La carte APF9328 d'Armadeus sera le cœur des expériences présentées dans ce document. Elle intègre un processeur basé sur un processeur ARM9 et un FPGA puissant reliés par des bus de communication. Nous ajoutons des périphériques afin d'illustrer diverses utilisations du FPGA comme instrument reconfigurable : une diode électroluminescente (LED), un capteur d'image CMOS pour l'imagerie et des signaux radiofréquences pour le développement de compteurs rapides.

Nous désirons développer un certain nombre de programmes pour communiquer entre un PC et la carte Armadeus ou entre l'ARM et le FPGA. Il nous faut donc les outils pour compiler des applications à destination d'un ARM9 sur un PC (probablement basé sur une architecture Intel). Parmi les excellentes pages mises à disposition par l'association Armadeus [6], celle de l'installation de la toolchain est particulièrement claire : <http://www.armadeus.com/wiki/index.php?title=LinuxInstall>.

Notez que le développement d'une toolchain à destination d'un nouveau circuit embarqué est une tâche fastidieuse, car, au-delà de la simple installation du *cross*-compilateur, il faut renseigner ce dernier sur les plages d'adresses accessibles par le processeur et dans lesquelles le compilateur a le droit de placer le code, la pile, le tas, etc. Une connaissance précise du matériel est donc nécessaire pour accomplir ces opérations. Une description de ce type d'activités est par exemple faite à [http://uldp.home.at/uclinux\\_doc\\_4.html](http://uldp.home.at/uclinux_doc_4.html) : le script de *linkage* (généralement avec une extension **.ld** ou **.x**) est donné comme argument à **gcc** par l'option **-T fichier.ld**.

## 2.2

Outils de développement  
Xilinx pour GNU/Linux

Comme tout logiciel propriétaire distribué sous GNU/Linux, les outils de synthèse pour le FPGA sont distribués sous la forme d'une archive compressée. On se contentera de les désarchiver au bon endroit, puis de rendre les binaires associés exécutables. Là encore, toute la procédure est décrite sur le wiki d'Armadeus : [http://www.armadeus.com/wiki/index.php?title=ISE\\_WebPack\\_installation\\_on\\_Linux](http://www.armadeus.com/wiki/index.php?title=ISE_WebPack_installation_on_Linux). On peut s'attendre, comme ce fut le cas pour MapleV ou Mathematica il y a quelques années (passage de **libc5** à **glibc**), à des problèmes de compatibilités de bibliothèques dynamiques dans le futur, mais, pour le moment, les binaires sont fonctionnels en l'état sur une distribution Debian/etch.

Notre choix s'est porté sur la programmation du FPGA en VHDL pour diverses raisons : pérennité, réutilisation et maintenance d'une solution développée autour d'un langage de programmation (opposée à une solution de configuration par interface graphique), gestion du projet multi-utilisateur avec utilisation d'outils tels que **diff** et **patch**, utilisation d'un langage bien documenté et dont la synthèse ne dépend a priori pas de l'architecture du composant cible, avec notamment la capacité d'utiliser le code ainsi validé sur un composant dédié sur silicium (ASIC). Une archive contenant tous les codes présentés dans ce document est disponible à <http://jmfriedt.free.fr>.

Bien qu'il soit techniquement possible d'installer une plateforme de développement et de simulation VHDL complètement libre sous licence GNU, il est toujours nécessaire de recourir à l'ISE (*Integrated Software Environment* propriétaire Xilinx lors de la synthèse du code en fin de conception du projet ([http://www.armadeus.com/wiki/index.php?title=How\\_to\\_make\\_a\\_VHDL\\_design\\_in\\_Ubuntu/Debian](http://www.armadeus.com/wiki/index.php?title=How_to_make_a_VHDL_design_in_Ubuntu/Debian)). Nous utiliserons donc cet outil au cours de l'ensemble des développements présentés dans ce document (Fig. 2).

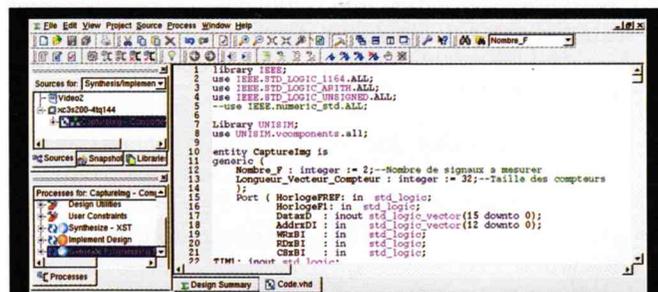


Figure 2 : Capture d'écran du logiciel fourni par Xilinx, fonctionnel sous GNU/Linux, pour synthétiser le code VHDL à destination du FPGA Spartan3. Cette interface est exclusivement utilisée pour la phase de synthèse du code VHDL en fichier binaire servant à la configuration du FPGA.

## 3

## Premiers pas sur FPGA

## 3.1 Contrôler une diode

Comme pour tout nouveau circuit embarqué que nous découvrons, notre premier objectif est de contrôler une diode électroluminescente (LED) connectée à une broche du FPGA. Il s'agit aussi d'une opportunité pour se familiariser avec les points d'accès aux broches du FPGA reportés sur la carte de développement DevLight, et d'apprendre à transférer un fichier de configuration d'un FPGA depuis la version de GNU/Linux tournant sur l'ARM9.

Nous allons donc commencer par nous familiariser avec un des atouts de la carte Armadeus, à savoir la liaison directe des bus de données et d'adresses entre l'ARM et le FPGA (Fig. 1). Cette communication parallèle à haut débit va être utilisée ici pour piloter l'alimentation d'une LED, connectée en sortie du FPGA. Comme sur tout bus, de nombreux signaux transitent continuellement à destination des autres périphériques partageant ce support de communication : RAM, mémoire non-volatile (flash), contrôleur Ethernet ... Afin de ne pas perturber la communication entre le processeur et ces périphériques, notre configuration sur le FPGA doit prendre soin de :

- Ne pas mobiliser le bus de données en configurant les broches associées en sortie (D0 à D15 du processeur sont reliées aux broches du banc 7 du FPGA tel que décrit dans [http://www.armadeus.com/\\_downloads/apf9328/hardware/apf\\_schema.pdf](http://www.armadeus.com/_downloads/apf9328/hardware/apf_schema.pdf)), induisant un conflit avec les autres périphériques communiquant sur ce bus (broche en haute impédance ou entrée par défaut)
- Ne pas mobiliser le bus d'adresses qui, en l'absence de DMA, impose sa valeur au FPGA qui n'a jamais à y écrire (broches en entrée).
- Identifier lorsque le FPGA est appelé par un décodage d'adresse (lecture de la valeur sur le bus d'adresses et comparaison avec une plage assignée à notre composant programmable, évitant tout conflit avec les périphériques installés par ailleurs sur la carte) associé à la gestion des signaux de contrôle issus du processeur (lecture ou écriture).
- Conserver la valeur lue sur le bus de données une fois le cycle de communication achevé : une bascule (ou un *latch*) déclenchée par le décodeur d'adresse mémorise la donnée fournie sur le bus d'adresses pour traitement ultérieur.

On a donc affaire à 3 ensembles de signaux : le bus d'adresse qui identifie l'interlocuteur du processeur dans la discussion, le bus de données identifiant le contenu de la discussion, et le bus de contrôle identifiant la nature de la transaction (notamment écriture ou lecture pour les cas qui vont nous intéresser ici).

## 3.1.1 Code VHDL

Le plus simple pour démarrer avec la synthèse de code VHDL consiste à reprendre le code fourni en exemple dans le projet Armadeus (**BRAMTEST.vhdl**). On aura ainsi une trame dont on conservera la structure générale.

Le composant définissant la configuration du FPGA est inclus entre :

```
entity ComposantVHDL is ... end ComposantVHDL;
```

La description du composant inclut avant tout la définition des ports de communication en entrée ou en sortie (E/S) ainsi que leurs tailles : après le mot-clé **Port**(, nous ajoutons la définition d'un signal sur un unique bit en sortie

```
LED: out std_logic;
```

à notre entité modélisée en VHDL. On y fera correspondre un état d'entrée (issu de la valeur du bus de données) et des états logiques en sortie : dans notre exemple, il s'agira d'un état binaire : LED allumée et LED éteinte. Un autre état possible du port est la haute impédance dans laquelle les broches n'imposent par leur état sur le bus, sans pour autant en traiter le contenu.

Vient ensuite la description de l'architecture du composant :

```
architecture Version01 of ComposantVHDL is ...
end Version01;
```

Cela regroupe les variables système (**signal**), les en-têtes d'E/S des composants VHDL génériques qui vont être intégrés au composant final (**component**), les constantes (numérique et de connexion), et les fonctions événementielles (**process**).

La variable système qui nous intéresse est :

```
signal Etat_LED:std_logic;
```

Il s'agit d'une variable binaire, qui fera correspondre son état à celui de la sortie de la broche LED par :

```
LED <= Etat_LED;
```

Ceci représente une constante de connexion. Ce routage est en dehors de tout process. Il sera donc opérationnel pendant toute la durée d'exécution du code dans le FPGA.

Nous adaptons ensuite le process chargé du décodage d'adresse et de la gestion des données par le process :

```
process (RDxBI, WRxBI, ClkxCI, CSxBI, AddrxDI, DataxD,
...
end process;
```

Ce process scrute les événements sur les signaux de contrôle **CSxBI** (*chip select*) et **WRxBI** (sélection du mode écriture), ainsi que le bus d'adresses **AddrxDI**

```
if(WRxBI = '0' and CSxBI = '0') --ARM ecrit dans FPGA
... case AddrxDI is
```

afin de voir si l'ARM veut communiquer avec le FPGA au moyen du bus de données **DataxD**. En cas de validation du décodeur d'adresse

```
if WRxBI = '0' and CSxBI = '0' then--Linux ecrit
  case AddrxDI is
    when "0001011110100" => --AddrxDI=0x02f4
      Etat_LED <= DataxD(0);
    when "0001011110000" => --AddrxDI=0x02f0
  ...
```

l'état du bit de poids faible du bus de données (**Etat\_LED<=DataxD(0);**) est recopié sur la broche contrôlant la LED.

Nous avons omis dans cet exemple de synchroniser les opérations sur un signal additionnel **ClkxCi='1'** afin de revenir au cas le plus général des processeurs qui valident la communication sur un décodage d'adresse, un bit de direction (ici **WRxBI**) et éventuellement un chip select lorsqu'il existe (**CSxBI**). Ce signal doit être correctement géré en cas de communication de valeurs 32 bits, puisque la transaction de 2 mots de 16 bits se fait sur une seule transition de chip select, mais deux transitions de l'horloge [8, **FPGA timing diagrams**].

Afin d'interfacer ce composant avec le reste du monde (ici le processeur ARM), il faut assigner chaque signal à des broches et configurer la direction de communication de chacune de ces broches. Ces opération se définissent dans le fichier d'extension **.ucf**.

Ce fichier fait correspondre le numéro d'une broche à sa désignation. Pour connaître le numéro de la broche, et sa position sur la carte DevLight, il faut recourir à la *datasheet* du Spartan 3 [9] et à la documentation d'Armadeus ([http://www.armadeus.com/\\_downloads/apf9328/hardware/apf\\_schema.pdf](http://www.armadeus.com/_downloads/apf9328/hardware/apf_schema.pdf)). Dans notre cas :

```
NET "LED" LOC = "P112" | IOSTANDARD = LVCMOS33;
```

configure la broche (dénommée **L01P\_1**) de contrôle de la LED.

Il faudra prendre soin, lors de la phase de communication du FPGA vers le processeur, de toujours avoir une condition par défaut qui place le bus de données en haute impédance et n'en impose le niveau que lorsque la logique d'écriture est validée.

```
if RDxBI = '0' and CSxBI = '0' then -- Linux lit(FPGA)
  -- phase de lecture ...
else if WRxBI = '0' and CSxBI = '0' then -- Linux ecrit(FPGA)
  -- phase d'écriture ...
else DataxD <= (others => 'Z'); -- etat de haute impendance
```

En l'absence de cette précaution, nous créons un conflit sur le bus de données et fournissons un *opcode* invalide au processeur.

Si nous suivons correctement les conseils sur la page FPGA du wiki Armadeus (bien activer la génération d'un **.bin** dans les propriétés du module tel que décrit à [8, **Bitstream generation**], nous obtenons le fichier **.bin** à la fin de la synthèse du code. Ce fichier doit être accessible au

système embarqué, soit par envoi par **scp** (cela nécessite une recompilation du *buildroot*), soit via un système de fichier NFS monté depuis l'Armadeus.

Après avoir chargé le module noyau (**fpga-loader.ko**) permettant le transfert du binaire dans le FPGA, l'implantation du composant s'obtient par :

```
dd if=ComposantVHDL.bin of=/dev/fpga/fpga-loader
```

## 3.1.2 Communication

Le projet Armadeus fournit un exemple de communication entre le processeur et le FPGA par cartographie (mappage) mémoire : **fpga-regs**. Cet exemple illustre la programmation sous GNU/Linux pour placer les valeurs appropriées sur les bus d'adresse et de données, utile pour déboguer rapidement les diverses versions de vos *firmwares*.

Par exemple, dans le cas présent, l'exécution de

```
fpga-regs 02f4 1
```

permet d'allumer la diode connectée par l'anode au FPGA et par la cathode à la masse, tandis que

```
fpga-regs 02f4 0
```

met le bit de poids faible du bus de données à 0 et donc éteint la LED.

Depuis l'espace utilisateur, l'exploitation de cette cartographie de la mémoire pour communiquer avec le FPGA s'obtient en C par :

```
int MEM = open("/dev/mem", O_RDWR | O_SYNC);
mappage_FPGA = mmap(0, MAP_SIZE=4096, PROT_READ | PROT_WRITE, \
MAP_SHARED, MEM, FPGA_BASE_ADDR=0x12000000);
*(u16*) (mappage_FPGA + AddrxDI) = DataxD; // écriture
DataxD = *(u16*) (mappage_FPGA + AddrxDI); // lecture
```

La même action s'obtient depuis un module noyau par :

```
mappage_FPGA = (void *) ioremap(FPGA_BASE_ADDR, 4096);
*(u16*) (mappage_FPGA + AddrxDI) = ; // écriture
DataxD = *(u16*) (mappage_FPGA + AddrxDI); // lecture
```

Le lecteur averti aura remarqué que, mise à part l'initialisation du mappage de la mémoire, les accès sont identiques, que l'on soit en espace utilisateur ou noyau. On peut donc avoir un en-tête commun entre les deux codes résultants :

```
#define FPGA_BASE_ADDR 0x12000000
#define MAP_SIZE 4096
#define MAP_MASK ( MAP_SIZE - 1 )
void * mappage_FPGA;
void ecriture_fpga(unsigned short AddrxDI, unsigned short DataxD){
*(unsigned short*)(mappage_FPGA + AddrxDI) = DataxD;}
unsigned short lecture_fpga(unsigned short AddrxDI){
return *(unsigned int*)(mappage_FPGA + AddrxDI);}
```

On remarquera que la lecture nécessite une conversion de type (*cast*) vers entier non signé **unsigned int**. Ceci est uniquement valable en espace utilisateur, et la raison nous en est inconnue. Si la conversion de type est un mot de 16 bits (**unsigned short**), on ne récupère alors que les 8 bits de poids faible.

Il est intéressant d'estimer les temps de communication entre l'ARM et le FPGA, en fonction de ces 3 modes (communication depuis le *shell*, depuis un programme en C en espace utilisateur ou depuis l'espace noyau). Pour ce faire, une séquence commune va être réalisée, utilisant la lecture (RDxBI = '0' and CSxBI = '0') d'un registre (compteur) qui sera incrémenté par les impulsions qu'il reçoit de l'horloge ClkxCI (96 MHz) fournie sur la broche 55 (L32P\_4) du FPGA, lorsque notre LED sera allumé. Ce compteur peut par ailleurs être remis à zéro (cf. compteur de fréquence, section 4).

La séquence de tests suivante permettra d'établir ces différents temps :

Action	Script sh	Appel C
RAZ<='1'	fpgaregs 02f0 1	ecriture_fpga(0x02f0,1);
RAZ<='0'	fpgaregs 02f0 0	ecriture_fpga(0x02f0,0);
LED<='1'	fpgaregs 02f4 1	ecriture_fpga(0x02f4,1);
LED<='0'	fpgaregs 02f4 0	ecriture_fpga(0x02f4,0);
Compteur(15 downto 0)	fpgaregs 0670	lecture_fpga(0x0670);
Compteur(31 downto 16)	fpgaregs 0674	lecture_fpga(0x0674);

Voici les résultats du script **sh**, suivi du C et du module noyau :

```
# ./Get.sh
read 0x0011 at 0x12000674
read 0x0497 at 0x12000670
```

Nous obtenons dans tous les cas de mesures d'abord le mot de poids fort suivi du mot de poids faible.

Le calcul du nombre de cycles détectés par le compteur est obtenu par exemple sous GNU/Octave par **0x0011\***

**65536+0x0497=1115287**. Ceci correspond au nombre d'oscillations de l'horloge de référence à 96 MHz fournie au FPGA par le signal ClkxCI (L'utilisation de ce signal comme référence de temps n'est pas judicieuse, car il est issu d'une multiplication d'un oscillateur de référence à 32 kHz et présente des fluctuations de plus d'un kilohertz autour de sa valeur nominale. Cette précision suffit cependant pour cette démonstration).

Ces mêmes mesures sont faites pour le programme C compilé pour une exécution depuis l'espace utilisateur et le module noyau, et donnent respectivement :

```
# ./inter
Compteur=0
Compteur=56

# dmesg
...
Compteur=0
Compteur=27
```

Le temps d'éclairement de la LED est mesuré à l'oscilloscope à titre de comparaison avec la valeur du compteur :

Méthode	Compteur	Compteur/96MHz	Temps oscillo
sh	1115287	11,618 ms	11,62 ms
espace utilisateur	56	583 ns	585 ns
module noyau	26	270 ns	269 ns

Nous observons un ordre prévisible des performances, avec un interpréteur shell excessivement lent comparé à l'exécution en espace utilisateur d'un programme C, avec les performances optimales lors de l'accès au FPGA depuis le module noyau.

## 4

## Application au compteur de fréquences

### 4.1

### Principe du compteur réciproque

Un compteur conventionnel de fréquence  $f_i$  rapporte une valeur numérique  $C_i$  du nombre d'événements observables (transition d'un événement oscillant au-delà d'un seuil) et discernables (nécessite des tensions suffisantes pour faire basculer les transistors, éléments de base du calcul numérique et composant le FPGA) pendant un laps de temps  $t_g$  d'intégration défini, nommé temps de porte (*gate time*).

On détermine alors la fréquence du signal oscillant par le rapport du registre ( $C_i \pm 1$ ) contenant la valeur du compteur et de  $t_g$ . Ce  $\Delta(t)$  n'est pas constant dans le temps, et limite la précision d'un tel compteur (Fig. 3).

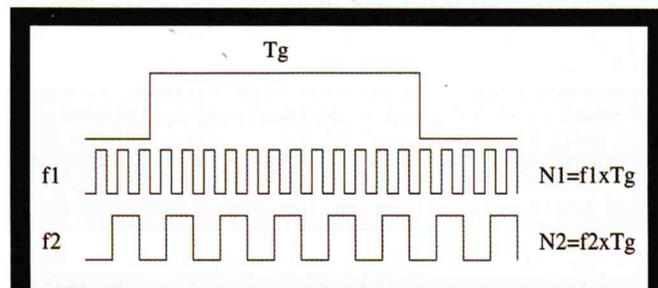


Figure 3 : Principe de fonctionnement du compteur réciproque

Le compteur réciproque permet de nous affranchir en grande partie de  $\Delta(t)$ , en mesurant le nombre d'oscillations  $C_{ref}$  d'une seconde fréquence  $f_{ref}$  dite de référence. On obtient

alors la fréquence inconnue par :  $f_{ref} \times C_i / C_{ref}$  L'incertitude relative ( $\Delta f/f$ ) sur la valeur de la fréquence mesurée est alors de

$$\frac{\Delta f}{f} = \frac{1}{t_g} \left( \frac{1}{f} + \frac{1}{f_{ref}} \right)$$

en supposant l'incertitude sur les compteurs de  $\Delta C_i=1$ .

## 4.2

### Synthèse du compteur de fréquence en VHDL

Le compteur utilise un temps de porte (gate time) défini de façon logicielle [10, 11, 12]. Les N compteurs sont déclenchés sur le front montant du signal de porte et sont arrêtés par le front descendant. Ces N mesures sont alors transmises au processeur pour traitement (quotients pour revenir à une mesure relative à l'oscillateur de référence, s'affranchissant donc de l'incertitude sur le temps de porte).

Le code VHDL nécessite en général un fonctionnement synchrone. Mais, pour le comptage des oscillations (+1 à chaque front montant => **rising\_edge**), un fonctionnement asynchrone peut convenir.

```
signal Compteur : std_logic_vector (31 downto 0);
process ( Signal, GateTIME, RAZ )
begin
if RAZ = '1' then -- Remise @ 0
Compteur <= "00000000000000000000000000000000";
elsif rising_edge ( Signal ) and GateTIME = '1' then
Compteur <= Compteur + 1;
end if ;
end process ;
```

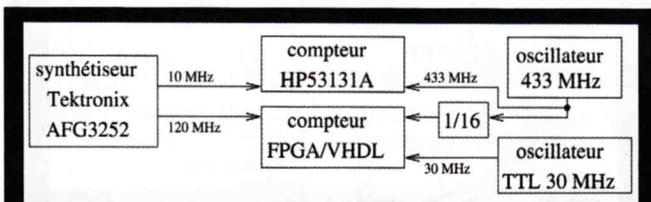


Figure 4 : Schéma du montage visant à mesurer le bruit de mesure du compteur de fréquence. Un compteur de bonne qualité (Agilent 53131A) sert de référence par rapport à laquelle nous comparerons notre compteur réciproque. Le signal de référence de ces deux compteurs est synthétisé par un Tektronix AFG3252 (deux voies), d'une part pour former le 10 MHz de référence du compteur Agilent, et d'autre par 120 MHz pour notre compteur réciproque. Le signal de mesure qui nous intéresse, un oscillateur à 433 MHz sensible à la température, est divisé par 16 afin d'entrer dans la gamme de fréquences de fonctionnement du FPGA. Un oscillateur 30 MHz sert de référence secondaire et illustre la souplesse de la configuration du FPGA, puisque nous pouvons rapidement étendre le compteur 2 voies à 3 voies.

La fréquence maximum de fonctionnement donnée lors de la synthèse par l'ISE de Xilinx est estimée à 194 MHz, mais des fréquences allant jusqu'à 250 MHz ont pu être mesurées.

Le signal qui nous intéresse, issu d'un oscillateur 433 MHz, sera donc divisé (Zarlink SP8402) et mis en forme avant d'arriver au FPGA.

## 4.3

### Exploitation et performances du compteur

La réalisation d'un instrument n'est qu'une première étape, préliminaire à sa caractérisation et sa qualification dans les conditions d'utilisation qui vont nous intéresser.

Nous allons donc dépasser les aspects purement informatiques de ces développements pour nous pencher sur la métrologie de la mesure de la fréquence et ainsi estimer les performances du compteur réciproque que nous avons réalisé.

Le lecteur qui n'est pas concerné par l'application concrète de mesure de fréquence peut passer à la section suivante, bien qu'il nous semble utile d'insister sur le fait que la réalisation d'un instrument dépasse l'aspect purement informatique et électronique des développements.

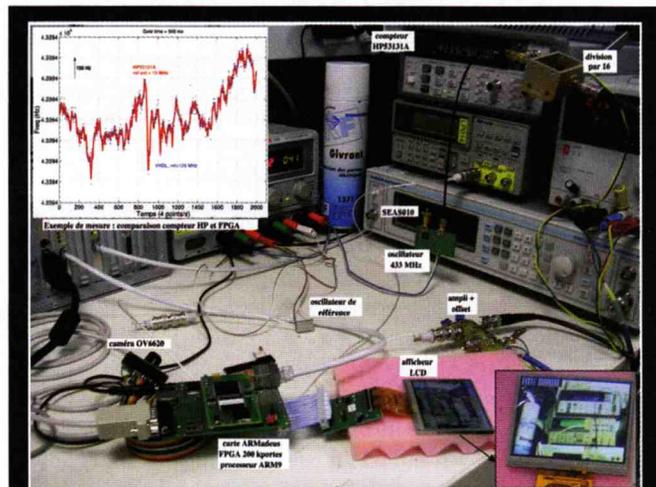


Figure 5 : Prototype d'un instrument capable de compter deux fréquences autour de 433 MHz (divisée par 16 ≈ 27 MHz) relativement à une référence autour de 100 MHz, simultanément avec une capture d'image en 56 ms et un affichage des résultats de ces mesures sur écran LCD. La mesure de fréquence présentée en haut à gauche a été obtenue avec un temps de porte de 1 s (bleu), et est comparée avec la mesure d'un compteur Agilent 53131A (courbe rouge) : les deux courbes coïncident visuellement, mais une analyse quantitative est fournie dans le texte.

Nous commençons par une utilisation démontrant le bon fonctionnement du montage (Fig. 5) avec une mesure de température observée comme variation de fréquence de l'oscillateur 433 MHz. Des refroidissements et échauffements successifs sont clairement visibles sur la figure 6.

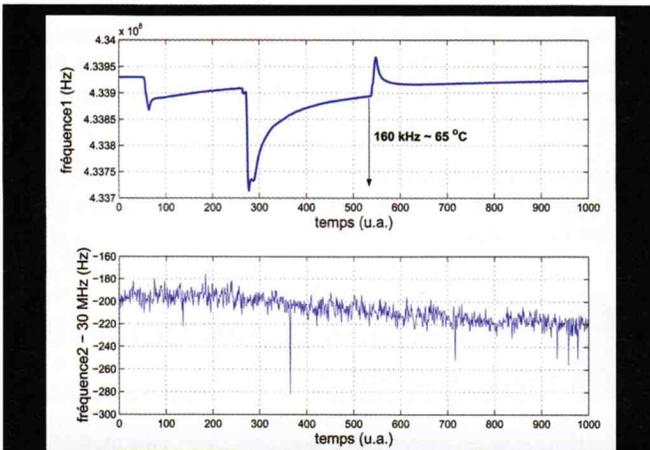


Figure 6 : Mesure de température (courbe du haut) observée sous forme de dérive de fréquence d'un oscillateur 433 MHz spécialement conçu à cet effet. Le résonateur est refroidi à deux reprises avec une bombe givrante (dates 60 et 280) et chauffé à la date 550. La mesure est référencée sur la sortie 120 MHz du synthétiseur de fréquence Tektronix AFG3252. Un oscillateur TTL de 30 MHz en entrée d'un second compteur (courbe du bas) présente une stabilité de l'ordre de la dizaine de hertz relativement à cette référence.

Une première analyse purement visuelle (Fig. 7 et courbe insérée dans Fig. 5) permet de comparer les performances de notre circuit avec celles d'un compteur de fréquence d'excellente qualité, le Agilent 53131A qui nous servira de référence au cours de cette discussion [13]. Il apparaît par exemple clairement que la tendance de la mesure est correcte, avec des fluctuations à long terme de la fréquence correctement observées, mais qu'avec un temps de porte de 100 ms notre mesure est plus bruitée que celle du compteur Agilent. Les bruit de mesure semblent comparables entre les deux instruments pour un temps de porte de 1 s, qui sera désormais le paramètre sélectionné.

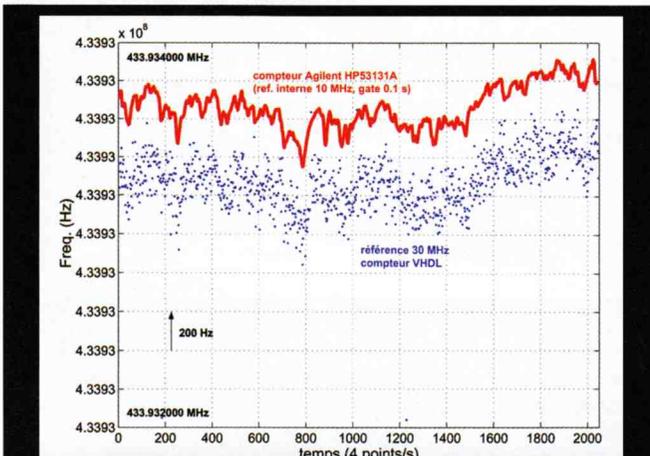


Figure 7 : Exemple de mesure d'un oscillateur 433 MHz en prenant pour référence un oscillateur TTL à 30 MHz (courbe bleue). Comparaison avec la mesure simultanément obtenue avec un compteur Agilent 53131A programmé avec un temps de porte équivalent à celui utilisé par notre montage. Temps de porte de 100 ms (courbe rouge). Un offset a été volontairement ajouté pour mieux distinguer les deux courbes.

Au-delà de cet aspect purement visuel, des outils ont été développés pour quantifier les performances d'oscillateurs. L'outil le plus couramment utilisé est la variance d'Allan

$\sigma_y(\tau)$  [14], qui calcule l'écart type des mesures moyennées sur un intervalle de temps  $\tau$  :

$$\sigma_y(\tau) = \sqrt{\frac{1}{2(M-1)} \sum_{i=1}^{M-1} (y_{i+1} - y_i)^2}$$

pour  $M$  mesures de la série temporelle  $y$  régulièrement distribuées dans des segments de  $\tau$  secondes.

$$y_n = \left\langle \frac{\delta f}{f} \right\rangle_n$$

pour une variation relative de fréquence  $\delta f$  autour de  $f$ .

Cette quantité  $\sigma_y(\tau)$  estime donc les fluctuations de fréquence de l'oscillateur en fonction du temps d'intégration du compteur : le bruit d'un oscillateur n'est pas une quantité intrinsèque, mais dépend de la durée sur laquelle il est interrogé. À court terme, un oscillateur présente des fluctuations associées au bruit de l'électronique d'entretien de l'oscillation, auquel s'ajoute éventuellement un bruit de mesure (lié au compteur et non à l'oscillateur). À long terme, un oscillateur dérive du fait des fluctuations lentes de température (effets thermiques sur le résonateur ou l'amplificateur) ou de tension d'alimentation. Afin de comparer les divers oscillateurs que nous utilisons, nous considérerons toujours la variation relative de fréquence  $\Delta f/f$ , quantité sans unité qui permet donc de comparer les performances de systèmes fonctionnant à des fréquences  $f$  différentes.

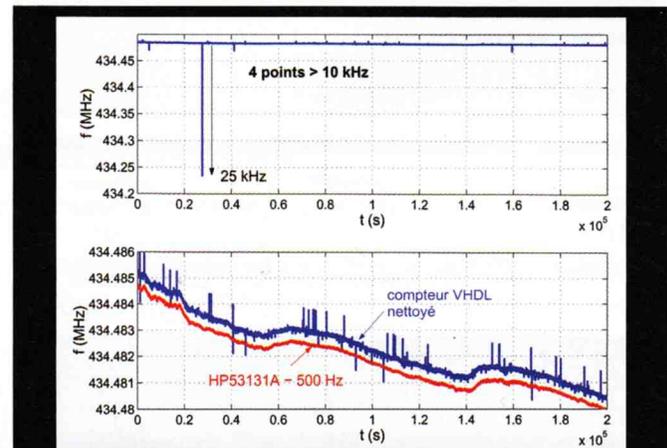


Figure 8 : Évolution temporelle au cours d'un week end de la fréquence d'un oscillateur à 433 MHz comprenant un résonateur sensible à la température. En haut, mesures brutes comprenant 4 points présentant un écart de fréquence de plus de 10 kHz par rapport à leur voisin. En bas, en bleu, la mesure obtenue par compteur dans le FPGA, nettoyée des points présentant un écart de plus de 1 kHz avec leur voisin (ce nettoyage a affecté moins de 30 points sur les 200000 acquis), et, en rouge, la mesure obtenue simultanément sur compteur Agilent 53131A, translaturée de 500 Hz par souci de clarté.

Une acquisition longue (3 jours) avec une fréquence de 1 s permet de mettre en évidence diverses propriétés de l'oscillateur. L'évolution temporelle du signal (Fig. 8) montre d'une part une dérive long terme du signal – probablement associée à une dérive thermique ou de la tension d'alimentation de l'oscillateur – observée à la fois par notre compteur et le compteur Agilent. Par ailleurs, notre compteur présente quelques mesures erronées, aisément éliminées par post-traitement numérique, mais sur lesquelles nous reviendrons plus tard.

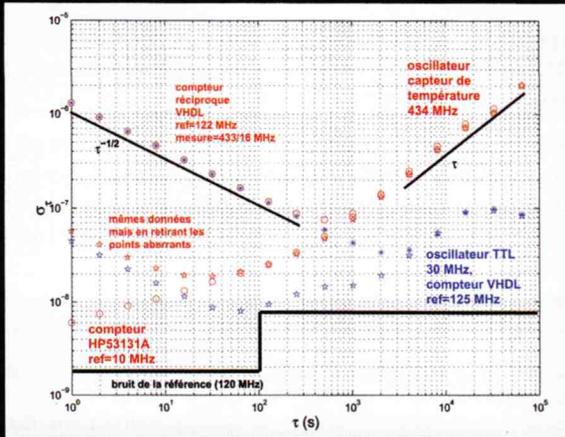


Figure 9 : Analyse quantitative de la stabilité de l'oscillateur par calcul de la variance d'Allan. Le temps de porte est de l'ordre de 1 s. La pente de la variance d'Allan en  $\tau^{-1/2}$  correspond à une fluctuation en bruit blanc de fréquence. Le pente de la variance d'Allan en  $\tau$  correspond à la dérive thermique de l'oscillateur. Les ronds rouges et les « \* » bleus sont obtenus par traitement des données brutes, incluant quelques points aberrants présentant un écart de plus de 1 kHz par rapport à leurs voisins : la variance d'Allan à court terme est dominée par ces points aberrants. Les étoiles rouges et bleues sont obtenues par post-traitement numérique des signaux (retrait des points aberrants, affectant moins de 1 point sur 5000 de la séquence acquise), et se rapprochent du signal « idéal » obtenu par compteur 53131A. Ce compteur professionnel élimine par moyenne glissante le bruit blanc de fréquence de pente  $\tau^{-1/2}$ , d'où l'écart entre nos mesures et celles de ce compteur à  $\tau < 40$  s.

La variance d'Allan calculée sur cet ensemble de points est illustrée par les ronds rouges sur la Fig. 9 pour le compteur synthétisé dans le FPGA et le compteur Agilent. Nous constatons que les deux calculs fournissent le même résultat à long terme ( $\tau > 700$  s), mais les deux courbes diffèrent pour des temps d'intégration plus courts.

Il est surprenant de constater que notre compteur, configuré pour mesurer simultanément un oscillateur 433 MHz et un oscillateur TTL 30 MHz, présente le même bruit de fréquence pour ces deux oscillateurs pour  $\tau < 700$  s. La source de référence à 120 MHz a alors été qualifiée indépendamment et a démontré une bonne stabilité avec des fluctuations relatives de fréquences inférieures à  $10^{-8}$  (trait noir en bas du graphique 9). Nous avons identifié la cause de cette variance élevée à court terme en éliminant les points aberrants de mesure aisément visibles sur la Fig. 8 (haut). Seuls 4 points aberrants avec un saut de fréquence de plus de 10 kHz par rapport à leurs voisins sont la cause du seuil élevé de la variance d'Allan à court terme. En éliminant les points donc, l'écart avec leur voisin est de plus de 1 kHz, le calcul de la variance d'Allan sur l'oscillateur 433 MHz et l'oscillateur TTL 30 MHz donne les courbes en étoile sur la figure 9.

Nous constatons que le résultat à long terme ( $\tau > 700$  s) n'a pas été affecté, mais maintenant les deux mesures sont bien distinctes et suivent le résultat obtenu avec compteur Agilent jusqu'à  $\tau = 30$  s. La correction n'a porté que sur 16 points parmi les 200000 acquis, mais cette extrême sensibilité de la variance d'Allan à très peu de point aberrants est un problème connu qui mériterait d'être corrigé lors de la phase d'acquisition et de traitement des valeurs des compteurs.

Nous attribuons les points erronés à du bruit lors des mesures, puisque le montage n'a pas été blindé et aucun soin particulier sur le filtrage des alimentations n'a été amené. Il serait souhaitable de réaliser un circuit convenablement agencé afin de blinder les divers signaux radiofréquences et les isoler du bruit numérique amené par la caméra connectée au même circuit, sujet de la section qui suit.

La synthèse d'un grand nombre de compteurs actifs simultanément ( $N >$  supérieur à quelques unités) n'a probablement pas de grand intérêt pratique compte tenu du nombre d'oscillateurs que nous sommes susceptibles d'interroger. Cependant, les perspectives de traitement plus complexes des données des compteurs en implémentant des fenêtres pondérées et des moyennes glissantes [15] – opération nécessitant plus d'un registre par compteur – sont ouvertes, puisque nous n'occupons aujourd'hui qu'une fraction réduite ( $\approx 10\%$ ) de la surface du FPGA.

## 5 Interfaçage d'une caméra CMOS

Ayant acquis la maîtrise de l'incrément d'un compteur sur un front de signal radiofréquence et du transfert vers le microprocesseur de la valeur accumulée sur le FPGA, nous désirons maintenant aborder un problème plus complexe. Un capteur optique CMOS est un composant numérique, qui fournit à chaque front d'un signal d'horloge de quelques dizaines de MHz la nouvelle valeur d'un pixel. Au lieu de simplement incrémenter un compteur, nous allons stocker dans une mémoire tampon les valeurs de ces pixels afin de restituer une image au processeur. L'objectif d'utiliser un

FPGA comme coprocesseur dédié à cette tâche est de limiter le temps d'occupation du processeur pour l'acquisition d'images par une sollicitation uniquement lorsque la RAM synthétisée dans le FPGA est pleine.

### 5.1 Le capteur OV6620

Le capteur OV6620 a été acquis auprès de Lextronic [16], monté sur un circuit imprimé et équipé d'une optique digne

d'une webcam, sous la nomenclature CA88. D'une résolution de 100 kpixels, l'OV6620 n'est évidemment pas concurrent des appareils photographiques numériques récents, mais suffisant pour se familiariser avec les concepts de base de l'acquisition d'images et une restitution sur écran LCD de résolution comparable.

Notez qu'il faut absolument éviter d'acquérir la version 3 Mpixels (OV3620) de ce capteur qui, sous une apparence de signaux de contrôle et de méthodes d'acquisition d'images similaires, ne fonctionne pas avec la configuration par défaut fournie par le constructeur, sachant que ledit constructeur refuse de fournir les valeurs des registres de configuration permettant l'utilisation du capteur sans le paiement d'un forfait dépassant largement les moyens du développeur amateur.

À titre indicatif, nous fournissons pour le lecteur désireux d'exploiter un capteur CMOS Omnivision OV3620 (3 Mpixels, adresse I2C 0x60) la séquence des registres (première colonne) et des valeurs à y placer (seconde colonne). Noter que la majorité de ces registres ne sont pas documentés dans la datasheet et que leur redéfinition à plusieurs reprises lors de l'initialisation fait penser à une série d'opcodes plutôt que des valeurs de registres de configuration

12 80	36 4C	36 4C	0F 42	43 00	34 58	32 36
12 80	39 F3	39 F3	14 C6	45 80	12 00	03 40
E5 3E	13 0D	13 C5	15 10	48 C0	17 10	02 86
E4 26	24 58	24 58	33 09	49 19	18 90	2D 00
E1 67	25 48	25 48	34 50	4B 80	19 01	00 00
F8 00	12 80	12 80	36 00	4D C4	1A C1	01 90
23 00	12 80	12 80	37 04	35 4C	32 36	12 00
F8 01	E5 3E	12 80	38 52	3D 00	03 40	10 43
FF B0	F8 01	12 80	3A 00	3E 00	11 02	13 C7
12 80	12 80	13 C7	3C 1F	3B 18	12 20	15 12
0E 05	0E 05	09 00	44 00	33 19	17 10	
12 04	12 04	0C 08	40 00	34 5A	18 90	
3C 0C	3C 0C	0D A1	41 00	3B 08	19 01	
33 37	33 37	0E 70	42 00	33 09	1A C1	

Nous avons déjà décrit ce capteur dans le cadre de son interfaçage avec un processeur de la série Coldfire. Cette fois, au lieu de ralentir la vitesse d'acquisition des images afin de permettre au processeur de capturer chaque valeur de pixel sur son bus de données, nous allons nous servir du FPGA pour synthétiser une RAM qui générera une interruption pour prévenir le processeur qu'elle est pleine.

De cette façon, nous sommes capables d'utiliser le capteur CMOS avec une capture d'image (registre 0x11 programmé par I2C à 0x02) en 56 ms, fréquence optimale de fonctionnement de la caméra utilisée afin d'éviter d'obtenir une image floue lorsque le capteur est en mouvement, tout en étant

compatible avec la vitesse du FPGA et sa communication avec le processeur. Cette méthode limite le temps d'occupation du processeur, puisque celui-ci n'est sollicité que chaque fois que le tampon du FPGA est plein.

## 5.2

### Configuration de la caméra : utilisation du bus I<sup>2</sup>C

Le support I2C est compilé en statique dans le noyau fourni par défaut avec la carte Armadeus. La communication se fait avec le *device* `/dev/i2c-0`.

La configuration de la caméra ne nécessite que des phases d'écriture afin de configurer les registres de la caméra qui définissent le protocole de communication et la vitesse de rafraîchissement des images (et donc la vitesse de transfert des pixels).

```
int write_byte(int fd, unsigned char reg, unsigned char value)
{
    unsigned char buf[2] = {reg,value}; // initialise a data
    // address and data
    // create an I2C write message
    struct i2c_msg msg = { CA88=0x60, 0, sizeof(buf), buf };
    // create a I2C IOCTL request
    struct i2c_rdwr_ioctl_data rdwr = { &msg, 1 };

    printf("write ref %02X value %02X\n",reg, value);
    if ( ioctl( fd, I2C_RDWR, &rdwr ) < 0 ){
        printf("Write error\n");return -1;}
    return 0;}

```

Voici les registres modifiés, et leurs valeurs respectives [17] :

```
//Mode RGB, Auto (Gain Control & White Balance)
write_byte (i2c, 0x12, 0xa4);
write_byte (i2c, 0x11, 0x02); // 1 img / 56 ms
write_byte (i2c, 0x12, 0x2c);
write_byte (i2c, 0x13, 0x21); // Format 8 bit

```

L'architecture de la mesure est différente du cas du compteur : dans ce premier cas, le processeur imposait la cadence en définissant le temps de porte et en lisant les résultats de la mesure après le temps d'acquisition.

Cette fois, la caméra nous impose la cadence en fournissant les signaux PCLK (nouveau pixel sur le bus de données), HREF (nouvelle ligne) et VSYNC (nouvelle image).

Nous devons donc synthétiser dans le FPGA la logique associée à ces trois signaux : attendre une nouvelle image (condition sur VSYNC), et tant que les pixels de chaque ligne sont valides (HSYNC haut), nous allons lire les données fournies par le capteur CMOS. Ces données sont accumulées dans une RAM servant de tampon afin de libérer la charge du processeur.

Le signal VSYNC informe sur le début de l'image (coin haut à gauche). Ensuite, chaque pixel est envoyé séquentiellement,

et est valide si Href et Pclk sont à l'état haut. La solution retenue consiste à incrémenter une adresse de registre à chaque front descendant de Pclk :

```
process(Vsync, Href, Pclk, Nb_Img) is
begin
  if Vsync='1' then
    ADDRA<="0000000000";
  else if Href='1' and falling_edge(Pclk) then --and
    Nb_Img=1 then
    ADDRA<=ADDRA+4;
  end if;
end if;
end process;
```

ADDRA est incrémenté par pas de 4, du fait que nous n'avons pas maîtrisé l'implémentation de la RAM et la communication entre FPGA et processeur ARM9. Cette valeur a été validée de façon expérimentale, et nous l'avons conservée.

Il serait cependant important d'en comprendre la raison fondamentale, et notamment l'organisation de la valeur 32 bits reçue lors de l'interrogation des registres synthétisés dans le FPGA ([http://www.armadeus.com/wiki/index.php?title=FPGA\\_register](http://www.armadeus.com/wiki/index.php?title=FPGA_register)).

Ces points sont en cours de validation par les membres de l'association Armadeus et une version corrigée pour des lectures 16 ou 32 bits de **fpgaregs** et la RAM associée dans le FPGA **bBRAMTest** devrait être disponible à la date de parution de cet article.

## 5.3 Acquisition des images

L'adresse ainsi obtenue est utilisée pour le contrôle du port d'entrée-sortie 8 bits d'une RAM qui comporte un second bus de 16 bits.

```
-- RAMB16_S9_S18: Virtex-II/II-Pro, Spartan-3/3E 2k/1k x
8/16 + 1/2 Parity bits Parity bits Dual-Port RAM
-- Xilinx HDL Language Template, version 9.2i
```

Cette RAM est un des nombreux composants VHDL génériques fournis par Xilinx. Leur implémentation est relativement simple à mettre en œuvre.

Ici, le premier port (8 bits) est destiné à recevoir les valeurs des pixels envoyés par la caméra, et n'est donc considéré que comme une entrée.

Le second bus de 16 bits convient parfaitement au bus de données existant entre l'ARM et le FPGA, et il n'est donc utilisé qu'en sortie pour communiquer au monde extérieur les informations que la RAM contient.

L'inconvénient majeur de cette solution est que la RAM synthétisable ne permet pas de stocker une image entière. Il faut donc faire transiter en plusieurs petits paquets les informations contenues dans le FPGA au noyau Linux.

Pour ce faire, la ligne **TIM1**, associée à une interruption matérielle, reliant le FPGA à l'ARM va être utilisée.

Elle permet de générer une interruption lors d'une transition haut/bas, afin de signaler un évènement au noyau.

Un module doit s'être déclaré comme capable de gérer cet évènement : la fonction d'interruption **f\_interruption()** est initialisée à l'insertion du module par **insmod** :

```
request_irq(IRQ_GPIOA(1), f_interruption, SA_INTERRUPT,
"TIM1", NULL);
set_irq_type( IRQ_GPIOA(1), IRQF_TRIGGER_FALLING );
```

Cette interruption est générée lorsque notre codeur d'adresse **ADDRA** dépasse une valeur que l'on ajuste expérimentalement afin que le temps de lecture corresponde exactement au temps qu'il faut à la caméra pour finir de remplir la mémoire.

```
process(ADDRA, Vsync, Nb_Img) is
begin
  if Nb_Img=1 and (ADDRA>"10111110000" or (Vsync='1'
and ADDRA>"0")) then
    TIM1<='0';
  else
    TIM1<='1';
  end if;
end process;
```

Le noyau Linux ayant été prévenu que la RAM va bientôt être pleine, la lecture est amorcée.

Une solution basée sur l'utilisation de 2 RAM individuelles – l'une en cours de remplissage pendant que l'autre est vidée – a échoué faute de temps (et de compétences dans le domaine), bien que des résultats préliminaires montrent que cette solution est réalisable (et souhaitable).

La lecture des données dans la RAM se fait par la fonction suivante :

```
unsigned short lecture_fpga_ram(unsigned short addr){
  static unsigned short resultat;
  static unsigned long resl;
  resl= *(unsigned long*)(FPGA_Addr+addr);
  resultat = ((resl&0xff0000)>>8)+(resl&0x0000ff);
  return resultat;
}
```

Cette fonction traduit du côté Linux l'architecture de la RAM dans le FPGA : la donnée lue (sur 32 bits) est retouchée pour former un mot de 16 bits représentatif de la valeur du pixel stocké en RAM.

L'adresse **addr** est incrémentée de 4 entre deux appels à cette fonction afin de lire deux mots successifs en RAM :

```
static int f_interruption(int irq, void *dev_id, struct
pt_regs *regs)
{static int i;
  static unsigned short ushortsize;
  for(i=0;i<256;i++){
```

```

    ushortsize=lecture_fpga_ram(i*4);
    img1[Indice_Tableau]=ushortsize; // unsigned short
*img1; recupere 2 pixels
    Indice_Tableau++;
}
return(Indice_Tableau);
}

```

À la fin des interruptions (plus précisément à la fin d'un temps pré-défini `usleep()` dans notre implémentation actuelle), le programme utilisateur récupère le tableau `img1` par l'intermédiaire d'un appel à la fonction `read()`, associée dans le noyau à la fonction `read_cmos()`.

Il serait probablement souhaitable d'utiliser un signal du type `SIGUSR` afin de réveiller le programme en espace utilisateur à la fin d'une acquisition d'image au lieu d'une attente d'une durée prédéfinie.

La fonction `read()` du C étant définie comme bloquante, la façon « normale » de gérer le retour d'information serait de ne renvoyer l'image que si le nombre prévu d'octets a été effectivement lu depuis la caméra.

Cela nécessite d'implémenter quelques sécurités additionnelles pour valider l'image acquise (décompte du nombre de lignes et de pixels par exemple).

Cette méthode n'a pas été proposée ici, car elle suppose que les images acquises sont parfaites, cas rarement vérifié en phase de débogage de la couche matérielle.

```

static ssize_t read_cmos(struct file *file, char *buf,
size_t count, loff_t *ppos)
{copy_to_user(buf, (unsigned char *)img1 , count);}

```

Du point de vue de l'espace utilisateur, la requête pour une nouvelle image est initialisée par un `ioctl()` qui demande au module noyau de déclencher une capture sur le FPGA.

Suite à une attente de plus de 56 ms, l'espace utilisateur effectue un appel à `read()` pour récupérer le nombre attendu de pixels. Le résultat d'une acquisition, après traitement pour convertir le format de Bayer en image couleur RGB (section 5.4) est présenté sur la figure 10.

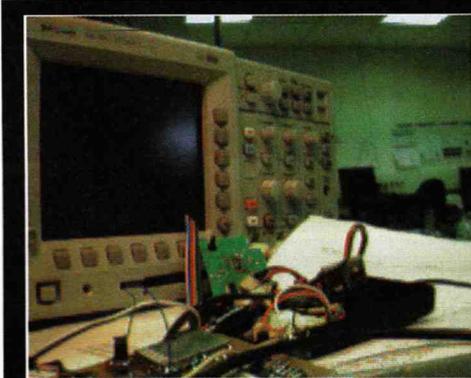


Figure 10 : Exemple d'image capturée en pleine vitesse (pas de ralentissement de l'horloge du capteur CMOS OV6620) par le FPGA, transmise au processeur pour traitement en vue d'une conversion en image couleur et affichage sur écran LCD, puis diffusion par réseau Ethernet.

## 5.4 Affichage sur écran LCD

L'affichage sur *framebuffer* est particulièrement simple, puisqu'il s'agit d'une zone mémoire dont le contenu est directement transféré sur l'écran (l'équivalent moderne du vénérable mode 13 des cartes VGA [19]). Ainsi, pour un écran de  $long \times large$  pixels, la zone mémoire  $(x+y \times large) \times N$  contient l'information de couleur du pixel de coordonnées  $(x,y)$  si  $N$  octet(s) sont utilisés pour chaque pixel. Avant d'afficher des pixels sur le LCD, il faut l'initialiser (cf. [18]) :

```

int fbfd = 0;
struct fb_var_screeninfo vinfo;
struct fb_fix_screeninfo finfo;
long int screensize = 0;
char *fbp = 0;
init_lcd(){
fbfd = open("/dev/fb0", O_RDWR);
ioctl(fbfd, FBIOGET_FSCREENINFO, &finfo);
ioctl(fbfd, FBIOGET_VSCREENINFO, &vinfo);
// Figure out the size of the screen in bytes
screensize = vinfo.xres * vinfo.yres * vinfo.bits_per_
pixel / 8;
// Map the device to memory
fbp = (char *)mmap(0, screensize, PROT_READ | PROT_
WRITE, MAP_SHARED, fbfd, 0);
}

```

Il est alors possible de spécifier la couleur R,G,B du pixel  $(x,y)$  par la fonction `pix()` :

```

void pix(int x, int y, int r, int g, int b){
long int location = 0;
location = (x+vinfo.xoffset) * (vinfo.bits_per_pixel/8) +
(y+vinfo.yoffset) * finfo.line_length;
unsigned short int t = ((r&0xf8)>>3)<<11 | ((g&0xfc)>>2)
<< 5 | ((b&0xf8)>>3);
*((unsigned short int*)(fbp + location)) = t;
}

```

Dans notre cas, les valeurs des pixels envoyées par la caméra suivent la séquence suivante : `rgbgrgbgrgbgrgbgrgb...` Sachant que la caméra possède un filtrage de Bayer, quatre pixels réels ( $rg_1bg_2$ ), correspondent à deux pixels image ( $rg_1b, rg_2b$ ).

Afficher notre image conformément aux spécifications de ce filtre devient (Fig. 11) :

```

for(j=0;j<240;j++){//Hauteur du LCD
for(i=0;i<160;i++){//Demi-Largeur du LCD
pix(2*i, j, Mon_Image[j*704+4*i+2], Mon_
Image[j*704+4*i+1], Mon_Image[j*704+4*i]);
pix(2*i+1, j, Mon_Image[j*704+4*i+2], Mon_
Image[j*704+4*i+3], Mon_Image[j*704+4*i]);
}
}

```

Figure 11 : Affichage d'une image sur l'écran LCD fourni avec la carte d'évaluation DevLight : l'image est acquise via le FPGA sur un capteur optique OV6620, transmise en espace noyau Linux pour répondre à la requête d'un programme en espace utilisateur. Ces données sont converties du format Bayer en RGB pour être affichées via le framebuffer /dev/fb0.



## 6 Conclusion et perspectives

Nous avons présenté l'utilisation d'une plateforme comprenant un processeur généraliste sur lequel est exécuté un système complet GNU/Linux, couplé à une matrice de portes logiques reconfigurables (FPGA) pour la réalisation d'un compteur réciproque fonctionnant dans la gamme radiofréquence (<200 MHz), et pour la capture à près de 20 images/seconde d'images issues d'un capteur optique CMOS. Nous avons pris soin de qualifier ces montages et d'en optimiser les performances en tentant de placer les opérations nécessitant une réaction rapide dans la logique reprogrammable, afin de limiter la surcharge de travail du processeur sur lequel tourne le système d'exploitation multitâche.

Bien que ce travail réponde à nos attentes et fournisse un résultat exploitable pour l'interrogation de capteurs basés sur la mesure de fréquence d'oscillateurs dérivant avec la grandeur physique mesurée, un certain nombre de points restent à éclaircir ou pourraient être améliorés :

- Le compteur de fréquence présente des mesures grossièrement erronées que nous pouvons filtrer numériquement après acquisition, dont la source reste à identifier.
- Des points d'ombre subsistent sur la communication entre une RAM implémentée dans le FPGA et l'ARM9 : nous avons dû lire des valeurs sur 32 bits (deux cycles d'accès mémoire) quand seuls 16 bits nous intéressaient. Nous n'avons pas été capables de générer 32 bits pertinents ou de faire des requêtes en mémoire sur 16 bits (1 cycle).

- La lecture du tampon en RAM synthétisée dans le FPGA est déclenchée par interruption depuis le FPGA afin que le temps de lecture des données par le processeur soit exactement égal au temps de remplissage des derniers mots de la RAM par l'image. Cette solution, qui serait acceptable sur un système d'exploitation temps réel avec des latences bornées, n'est pas acceptable sous GNU/Linux. Il serait donc judicieux de faire fonctionner une solution de deux tampons appelés alternativement, l'un étant rempli par l'image issue du capteur CMOS pendant que les données de l'autre sont transférées au processeur.
- Une image erronée peut être transmise au processeur faute de validation sur le nombre de lignes ou de pixels lus. Par ailleurs, nous n'avons pas exploité une fonctionnalité de notre capteur CMOS de rafraîchir plus rapidement un sous-ensemble de l'image (ROI, *Region of Interest*) qui pourrait présenter un intérêt, notamment dans le couplage des capteurs radiofréquence avec une mesure optique pour la mesure de vibration de poutres : ces améliorations pourront être implémentées en cas de besoin.
- Au cours de nos différents développements, nous avons tenté d'utiliser les avantages du code VHDL pour rendre générique (fonction VHDL **generate**) nos modules de comptage. Cependant, l'ISE de Xilinx génère une erreur : **"INTERNAL\_ERROR:Xst:cmain.c:3111 :1.8.6.1"**. Le site web Xilinx signale qu'il s'agit d'une erreur connue en cours de correction pour les versions ultérieures (> 9.2i).

## 7 Remerciements

Ce travail a nécessité des interactions diverses avec des interlocuteurs expérimentés dans leur domaine d'expertise.

Nous remercions Julien Boibessot (Armadeus) pour son assistance et le temps qu'il a passé à refaire fonctionner notre carte Armadeus après quelques manipulations malheureuses

(ne jamais essayer de flasher un fichier binaire destiné au FPGA à la place du *bootloader*), ainsi que l'ensemble des membres du *chan #armadeus* sur IRC ([irc.rezosup.org](http://irc.rezosup.org)).

Sébastien Euphrasie (FEMTO-ST) nous a assisté sur le développement en VHDL. Les mesures radiofréquences et l'analyse des données issues du compteur n'auraient pas été possibles sans l'aide de nos collègues de l'équipe Temps-Fréquence de l'institut FEMTO-ST : Pierre-Yves Bourgeois, Yann Kersalé, Gilles Martin et Enrico Rubiola ont contribué par leur expérience au succès de ce travail.

## Auteurs : T. Réternaz, J.-M Friedt



T. Réternaz complète son Master 3I (Image, Informatique, Ingénierie) entre Besançon et Dijon. Il est membre des associations *Projet Aurore* et *Sequanux* ([www.sequanux.org](http://www.sequanux.org)) pour la diffusion de la culture scientifique et du logiciel libre en Franche-Comté. Cette photo a été acquise avec la caméra présentée dans cet article.



J.-M Friedt est ingénieur dans la société *Sensor* ([www.sensor.com](http://www.sensor.com)), hébergé par l'institut FEMTO-ST de Besançon, et membre de l'association *Projet Aurore* (<http://projetaurore.assos.univ-fcomte.fr/>).

## Notes

1 *Field-Programmable Gate Array* est un composant électronique, reconfigurable par logiciel, contenant typiquement quelques centaines de milliers de fonctions logiques. Ces fonctions sont agencées selon une description fonctionnelle du composant dans des langages évolués tels que VHDL ou Verilog.

2 *Application Specific Integrated Circuit* est un composant électronique développé spécifiquement pour exécuter une tâche très spécifique, généralement en vue d'une réduction de coût par rapport à une solution utilisant un composant générique (microcontrôleur par exemple) programmé.

## Références

- [1] Un exemple au hasard : [http://www.imec.be/ovinter/static\\_research/reconfigurable.shtml](http://www.imec.be/ovinter/static_research/reconfigurable.shtml)
- [2] CRISTALDI (L.), FERRERO (A.) & PIURI (V.), « *Programmable instruments, virtual instruments, and distributed measurement systems: what is really useful, innovative and technically sound ?* », *IEEE Instrumentation & Measurement Magazine*, sept. 1999, p. 20-27, met l'accent sur l'utilisation de processeurs dédiés au traitement du signal (DSP) plutôt que sur les FPGA, et les interfaces utilisateur.

- [3] BEN ATITALLAH (A.) & KADIONIK (P.), « *Linux et les systèmes embarqués multimédias* », *GNU/Linux Magazine France*, 97, sept. 2007, p. 75-81.
- [4] DANECEK (J.), DRÁPAL (F.), PLUHÁČEK (A.), SALCIC (Z.), SERVÍT (M.), « *DOP – a simple processor for custom computing machines* », *J. of Microcomputer Applications*, 17, 1994, p. 239-253.
- [5] SALCIC (Z.), « *PROTOS – a microcontroller/FPGA-based prototyping system for embedded applications* », *Microprocessors and microsystems*, 21, 1997, p. 249-256.
- [6] BOIBESSOT (J.), « *Linux Embarqué pour tous !* », *GNU/Linux Magazine France*, 92, mars 2007, p. 70-79, et le site web, <http://www.armadeus.org>.
- [7] <http://www.arm.com/products/CPUs/families/ARM9Family.html>
- [8] <http://www.armadeus.com/wiki/index.php?title=FPGA>
- [9] [http://www.xilinx.com/support/documentation/data\\_sheets/ds099.pdf](http://www.xilinx.com/support/documentation/data_sheets/ds099.pdf)
- [10] KALISZ (J.), SZPLET (R.), PASIERBINSKI (J.) & PONIECKI (A.), « *Field-Programmable-Gate-Array-Based Time-to-Digital Converter with 200-ps Resolution* », *IEEE Transactions on instrumentation and measurement*, 46 (1), 1997, p. 51-55.
- [11] SZYMANOWSKIA (R.) & KALISZ (J.), « *Field programmable gate array time counter with two-stage interpolation* », *Rev. Sci. Instrum.*, 76, 2005, 045104.
- [12] KALISZ (J.), « *Review of methods for time interval measurements with picosecond resolution*, *Metrologia* », 41, 2004, p.17-32.
- [13] DAWKINS (S. T.), MCFERRAN (J.J.) & LUITEN (N.), « *Considerations on the Measurement of the Stability of Oscillators with Frequency Counters* », Comptes rendus de la conférence IFCS-EFTF 2007, p. 759-764.
- [14] RUBIOLA (E.), *The Leeson Effect*, p. 15, disponible à <http://www.femto-st.fr/~rubiola/slides/leeson-effect-slides2006.pdf>
- [15] RUBIOLA (E.), « *On the measurement of frequency and its sample variance with high-resolution counters* », *Rev. Sci. Instrum.*, 77, 2005, 054703.
- [16] <http://lextronic.fr/P1729-camera-numerique-ca88.html>
- [17] FRIEDT (J.-M.), GUINOT (S.), « *Introduction au Coldfire 5282* », *GNU/Linux Magazine France*, 75 septembre 2005.
- [18] <http://thunder.prohosting.com/~bricks/linux/program/framebuffer.html>
- [19] LAMOTHE (A.), *Teach Yourself Game Programming In 21 Days*, SAMS Publishing, 1994 ou <http://www.brackeen.com/vga/index.html>

# Avez-vous l'âme du collectionneur ?

**Boostez  
votre  
collection !**

Vous recherchez un magazine en particulier ? Allez sur [www.ed-diamond.com](http://www.ed-diamond.com) pour voir le sommaire détaillé de chaque magazine et ensuite... Boostez votre collection avec les « Power packs x5 », soit 5 GNU/Linux Magazine pour 15€ et les « Power packs x10 », soit 10 GNU/Linux Magazine pour 25€ à choisir dans la liste ci-dessous :

## LES 4 FAÇONS DE COMMANDER !

### Par courrier

En nous renvoyant ce bon de commande.

### Par le Web

Sur notre site : [www.ed-diamond.com](http://www.ed-diamond.com).

### Par téléphone

(paiement C.B.) entre 9h-12h & 15h-18h au **03 88 58 02 08**.

### Par fax

Au **03 88 58 02 09** C.B. et/ou bon de commande administratif.

## Numéros GNU/Linux Magazine épuisés

N°01, N°02, N°03, N°04, N°05, N°08, N°20, N°21, N°23, N°31, N°33 et N°90

Choisissez vos numéros dans le tableau ci-dessous\*

\* Seuls les numéros ci-dessous sont disponibles pour une commande de Power Packs par x5 et x10

N°06	GNOME - The Gimp	GNU Linux	N°74	VFS : Système de fichiers virtuel	
N°07	Dopez Linux	N°45	Cohabitation : UnDNS Bind dans un réseau Windows 2000	N°75	Tuning de code
N°09	Prêt pour le jeu !	N°46	Debian : Utilisez Samba avec le support ACL	N°76	Algorithmes évolutionnistes
N°10	The HURD : 100% GNU	N°47	GNUstep : le petit frère de Mac OS X ?	N°77	Systèmes de fichiers chiffrés
N°11	Exclusif : l'avenir de G.N.O.M.E	N°48	Caudium, votre prochain serveur Web !	N°78	Bluetooth, Spécifications, protocoles et configuration
N°12	NT et Linux : Guerre ou complément ?	N°49	Après MySQL & PostgreSQL SAP DB : La base de données libre & puissante	N°79	Sécurité du Noyau avec PAX
N°13	Cryptage : la clé de la sécurité	N°50	Créer un album Photo avec PHP ... et sans MySQL	N°80	Run in memory
N°14	XFree 4.0 : le futur à notre portée	N°51	Boostez votre site Web avec XML grâce à XSLT, CSS & XPath	N°81	Comment fonctionner les générateurs de nombres pseudo-aléatoires
N°15	Passer à la vitesse supérieure	N°52	Linux Temps réel où en est-on aujourd'hui ?	N°82	eCos, une solution libre pour systèmes embarqués
N°16	OpenSource : Est-ce suffisant ?	N°53	Linux sur PDA : Linux dans votre poche !	N°83	Greylist Éliminez le SPAM à la racine
N°17	Linux : Système embarqué	N°54	Maîtrisez LVM	N°84	Déploiement de hotspots Wifi sécurisés
N°18	Spécial interview : l'avenir de Linux	N°55	Intelligence Artificielle : Principes & programmation de jeux de stratégie classique	N°85	Firewall : Netfilter & NuFW
N°19	Dossier spécial : Postgre SQL 7.0	N°56	Développez vos applications Mozilla avec XPF & XPCOM	N°86	Serveur SMTP: Routage des mails avec Postfix
N°22	Le multi-threading : Une manière moderne de programmer le Multithé	N°57	Maîtrisez la gestion... Slots & Signaux ... des événements en C++	N°87	Le point sur Mono .NET Java et les Brevets
N°24	Palm et Linux	N°58	Djbdns enfin une alternative viable à BIND !	N°88	Sécurité: Smartcards & Tokens
N°25	Kernel 2.4.0	N°59	Zope, Créez un CD "Live" Zope en 10 minutes !	N°89	Utilisation avancée de XEN
N°26	<Dossier> XML </Dossier>	N°60	JBoss serveur d'applications J2EE OpenSource	N°91	Ajax Avancé
N°27	Les systèmes de fichiers journalisés	N°61	Découvrez MySQL 5 et les procédures stockées	N°92	Paravirtualisation XEN & SLO Répartition de charge
N°28	Scripting : la force d'Unix	N°62	Créez votre OS, principe et implémentation	N°93	Développez vos extensions Firefox
N°29	L.F.S. Linux From Scratch	N°63	Les threads : kernel 2.6 et 2.4	N°94	PABX Vidéo avec Asterisk
N°30	Le chiffrement des données	N°64	Adamato	N°95	Administration et configuration centralisées avec Cengine
N°32	Changez de coquille	N°65	Théorie et pratique : Supervision avec Nagios	N°96	Géolocalisation de photos numériques
N°34	XSL - FO : leX Killer ?	N°66	Créez votre Distribution Live	N°97	Haute-Disponibilité & Equilibrage de charge avec LVS/IPVS, heartbeat et Apache
N°35	QoS et iptable : optimisation et contrôle du trafic IP	N°67	C# .NET		
N°36	Linux embarqué : Le projet mGlinux	N°68	Le crash disque vous guette		
N°37	L'impression sous Linux	N°69	La réponse de Sun à Linux ? SOLARIS 10		
N°38	Le desktop Shell : Enlightenment	N°70	Découvrez et comprenez la technologie GRID		
N°39	Sécurité : Patchez votre noyau !	N°71	Présentation et installation du Hurd		
N°40	MySQL : la base de donnée OpenSource	N°72	Services Web... C/C++ et gSOAP		
N°41	Steganographie ou l'art de la dissimulation de données	N°73	Compression théorie algorithmes et programmation		
N°42	Développez vos pilotes de périphérique				
N°43	Administrez facilement votre réseau SNMP				
N°44	Comprenez Netbios pour Maîtriser l'interopérabilité windows				

# BON DE COMMANDE POWER PACKS À REMPLIR ET À RETOURNER À (OU PHOTOCOPIE)

## Diamond éditions - GNU/Linux Magazine - BP 20142 - 67603 SÉLESTAT Cedex

		OUI, je désire acquérir un POWER PACK X5		1 <sup>er</sup> 1PP* X5	2 <sup>ème</sup> 2PP* X5	3 <sup>ème</sup> 3PP* X5
Cochez ici POWER PACKS X5	1, GNU/Linux Magazine N°					
	2, GNU/Linux Magazine N°					
	3, GNU/Linux Magazine N°					
	4, GNU/Linux Magazine N°					
	5, GNU/Linux Magazine N°					
Total par série de POWER PACKS X5 :		15 €	30 €	45 €		
		OUI, je désire acquérir un POWER PACK X10		1 <sup>er</sup> 1PP* X10	2 <sup>ème</sup> 2PP* X10	3 <sup>ème</sup> 3PP* X10
Cochez ici POWER PACKS X10	1, GNU/Linux Magazine N°					
	2, GNU/Linux Magazine N°					
	3, GNU/Linux Magazine N°					
	4, GNU/Linux Magazine N°					
	5, GNU/Linux Magazine N°					
	6, GNU/Linux Magazine N°					
	7, GNU/Linux Magazine N°					
	8, GNU/Linux Magazine N°					
	9, GNU/Linux Magazine N°					
	10, GNU/Linux Magazine N°					
Total par série de POWER PACKS X10 :		25 €	50 €	75 €		
Les hors-séries et numéros spéciaux sont exclus des POWER PACKS. Montant TOTAL 15€ + 3,81€ de frais de port. Le TOTAL s'élève à 18,81€ pour l'achat d'un POWER Pack x5.		TOTAL :				
SEULEMENT EN FRANCE MÉTROPOLITAINE !		Frais de port :			+3,81€	
*PP= POWER PACK		TOTAL :				

**1 Voici mes coordonnées postales :**

Nom : \_\_\_\_\_

Prénom : \_\_\_\_\_

Adresse : \_\_\_\_\_

Code Postal : \_\_\_\_\_

Ville : \_\_\_\_\_

**2 Je joins mon règlement :**

Je règle par chèque bancaire ou postal à l'ordre de Diamond Editions

Paiement par carte bancaire :

N° Carte : \_\_\_\_\_

Expire le : \_\_\_\_\_

Cryptogramme Visuel : \_\_\_\_\_

Voir image ci-dessous

Date et signature obligatoire : \_\_\_\_\_

200



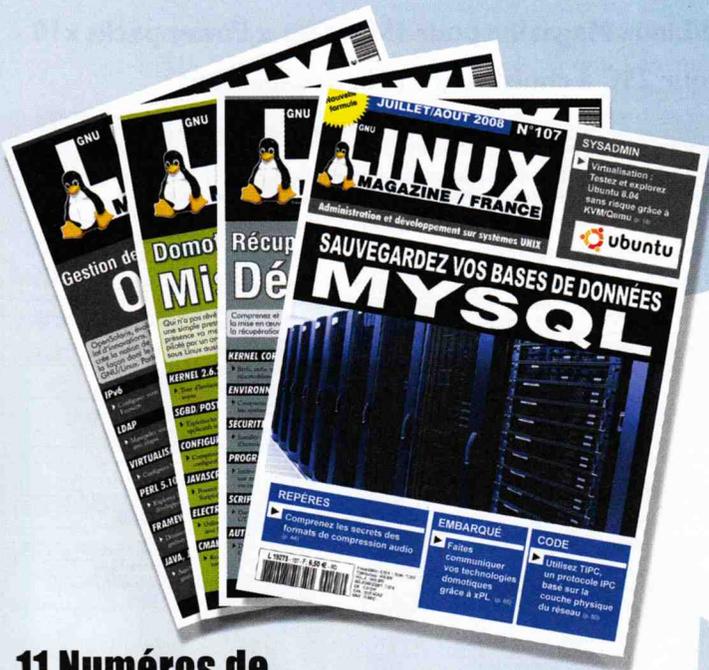
# Abonnez-vous !

## Économisez

Plus de

# 20%\*

\* Sur le prix de vente unitaire France Métropolitaine



11 Numéros de  
GNU/Linux Magazine

# 11

Numéros de  
GNU/Linux Magazine  
pour  
le prix de 9\*

\* Gain pour un abonnement France Métropolitaine, par rapport au prix unitaire France Métropolitaine

à **55€**  
(Offre France Métro)

Soit

Votre GNU/Linux Magazine

à **5,00€**  
(Tarif au numéro dans le cadre  
d'un abonnement France Métro)

Vous lisez d'autres magazines des Éditions Diamond ?  
Des offres de couplage sont disponibles dans ce magazine.

### Les 3 bonnes raisons de vous abonner !

- ▶ Ne manquez plus aucun numéro.
- ▶ Recevez GNU/Linux Magazine chaque mois chez vous ou dans votre entreprise.
- ▶ Économisez 16,50 €/an ! (soit plus de 2 magazines offerts !)

Pour les tarifs hors France Métropolitaine, consultez notre site : [www.ed-diamond.com](http://www.ed-diamond.com).

# Offres d'abonnement

(Nos tarifs s'entendent TTC et en euros)

	F	D	T	E1	E2	EUC	A	RM
	France Métro	DOM	TOM	Europe 1	Europe 2	Etats-unis Canada	Afrique	Reste du Monde
1 Abonnement Linux Magazine	55 €	59 €	67 €	69 €	66 €	70 €	68 €	77 €
2 Linux Magazine + Hors-série	83 €	89 €	101 €	104 €	100 €	105 €	103 €	116 €
3 Linux Magazine + MISC	84 €	90 €	102 €	105 €	101 €	107 €	104 €	117 €
4 Linux Magazine + Linux Pratique	78 €	85 €	96 €	99 €	95 €	101 €	98 €	111 €
5 Linux Magazine + Hors-série + Linux Pratique	110 €	119 €	134 €	138 €	133 €	140 €	137 €	154 €
6 Linux Magazine + Hors-série + MISC	116 €	124 €	140 €	144 €	139 €	146 €	143 €	160 €
7 Linux Magazine + Hors-série + MISC + Linux Pratique	143 €	154 €	173 €	178 €	172 €	181 €	177 €	198 €
8 Linux Pratique Essentiel + Linux Pratique	57 €	62 €	69 €	71 €	69 €	73 €	71 €	79 €

• Europe 1 : Allemagne, Belgique, Danemark, Italie, Luxembourg, Norvège, Pays-Bas, Portugal, Suède  
• Europe 2 : Autriche, Espagne, Finlande, Grande Bretagne, Grèce, Islande, Suisse, Irlande

• Zone Reste du Monde : Autre Amérique, Asie, Océanie  
• Zone Afrique : Europe de l'Est, Proche et Moyen-Orient

Toutes les offres d'abonnement : en exemple les tarifs ci-dessous correspondant à la zone France Métro (F)  
(Vous pouvez également vous abonner sur : [www.ed-diamond.com](http://www.ed-diamond.com))

**offre 1**

Linux Magazine (11 n°s)

par ABO : **55€**

Economie : 16,50 €

en kiosque : **71,50€**

**offre 2**

Linux Magazine (11 n°s) + Linux Magazine hors-série (6 n°s)

en kiosque : **110,50€**

par ABO : **83€**

Economie : 27,50 €

**offre 3**

Linux Magazine (11 n°s) + Misc (6 n°s)

en kiosque : **119,50€**

par ABO : **84€**

Economie : 35,50 €

**offre 4**

Linux Magazine (11 n°s) + Linux Pratique (6 n°s)

en kiosque : **107,20€**

par ABO : **78€**

Economie : 29,20 €

**offre 5**

Linux Magazine (11 n°s) + Linux Magazine hors-série (6 n°s) + Linux Pratique (6 n°s)

en kiosque : **146,20€**

par ABO : **110€**

Economie : 36,20 €

**offre 6**

Linux Magazine (11 n°s) + Linux Magazine hors-série (6 n°s) + Misc (6 n°s)

en kiosque : **158,50€**

par ABO : **116€**

Economie : 42,50 €

**offre 7**

Linux Magazine (11 n°s) + Linux Magazine hors-série (6 n°s) + Linux Pratique (6 n°s) + Misc (6 n°s)

en kiosque : **194,20€**

par ABO : **143€**

Economie : 51,20 €

**offre 8**

Linux Pratique Essentiel (6 n°s) + Linux Pratique (6 n°s)

en kiosque : **74,70€**

par ABO : **57€**

Economie : 17,70 €

Bon d'abonnement à découper et à renvoyer à l'adresse ci-dessous :

Je fais mon choix de la 1ère offre :

Je fais mon choix de la 2ème offre :

Je sélectionne le N° (1 à 8) de l'offre choisie :	
Je sélectionne ma zone géographique (F à RM) :	
J'indique la somme due :	€

Je sélectionne le N° (1 à 8) de l'offre choisie :	
Je sélectionne ma zone géographique (F à RM) :	
J'indique la somme due :	€
Total	€

Exemple : je souhaite m'abonner à l'offre Linux Magazine + Hors-série + MISC (offre 6) et je vis en Belgique (E1), ma référence est donc 6E1 et le montant de l'abonnement est de 144 euros.

Je choisis de régler par :

- Chèque bancaire ou postal à l'ordre de Diamond Editions
- Carte bancaire n° \_\_\_\_\_
- Expire le : \_\_\_\_\_
- Cryptogramme visuel : \_\_\_\_\_



Date et signature obligatoires

**Diamond Editions**  
Service des Abonnements  
B.P. 20142 - 67603 Sélestat Cedex



# GNU/Linux Magazine N° 107

## DÉCOUVREZ LA NOUVELLE FORMULE !

Actuellement chez votre marchand de journaux...

Nouvelle formule

JUILLET/AOÛT 2008 N°107

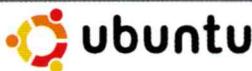
GNU **LINUX** MAGAZINE / FRANCE



Administration et développement sur systèmes UNIX

### SYSADMIN

▶ Virtualisation : Testez et explorez Ubuntu 8.04 sans risque grâce à KVM/Qemu (p. 18)



## SAUVEGARDEZ VOS BASES DE DONNÉES MYSQL



### REPÈRES

▶ Comprenez les secrets des formats de compression audio (p. 44)

### EMBARQUÉ

▶ Faites communiquer vos technologies domotiques grâce à xPL (p. 86)

### CODE

▶ Utilisez TIPC, un protocole IPC basé sur la couche physique du réseau (p. 80)

L 19275 - 107 - F: 6,50 € - RD



France Métro : 6,50 € - DOM : 7,00 €  
TOM Surface : 9,50 XPF  
POL. A : 1400 XPF  
BELUPORT CONT : 7,50 €  
CH : 13,8 CHF  
CAN : 10,25 \$CAD  
MAR : 75 MAD

## Sommaire

### News

- Debian/OpenSSL : une faille sans précédent dans le monde du logiciel libre

### Kernel

- Kernel Corner

### SysAdmin

- Virtualisation : testez et explorez Ubuntu 8.04 sans risque grâce à KVM/Qemu
- Gestion mémoire avec PostgreSQL
- Sauvegardez vos données MySQL : Les bases de données sont des éléments critiques dans le fonctionnement d'un serveur. Apprendre à procéder à des sauvegardes de manière optimale est donc incontournable. Explications en détail avec MySQL et LVM2.

### NetAdmin

- Visiophonie, web conférence – Présentation d'une solution : OpenMeetings

### Repères

- Les secrets des formats de compression audio

### Code(s)

- ncurses facile avec CDK
- Créez de vrais outils avec Gtkdialog
- Livre du mois : Le guide de survie, Expressions régulières
- TIPC : Transparent Inter Process Communication

### Embarqué

- Faites communiquer vos technologies domotiques grâce à xPL

### Hack(s)

- Un login graphique automatique sans GDM

Administration et développement sur systèmes UNIX

# www.UnixGarden.com

Récoltez l'actu **UNIX** et cultivez vos connaissances de l'**Open Source** !



**Administration système**

**Utilitaires**

**Comprendre**

**Embarqué**

**Graphisme**

**Environnement de bureau**

**Bureautique**

**Audio-vidéo**

**News**

**Administration réseau**

**Distribution**

**Programmation**

**Sécurité**

**Agenda-Interview**

**Matériel**

**Web**

**Jeux**

**Réfléchir**

**UnixGarden**