GNU LINUX MAGAZINE / FRANCE

Novembre/decembre 2005

France Métro : 6,40€ - DOM 6,95€ - BEL : 7,30€ - LUX : 7,30€ - PORT. CONT. : 7,30€ - CH : 13FS - CAN : 12$ - MAR : 65DH

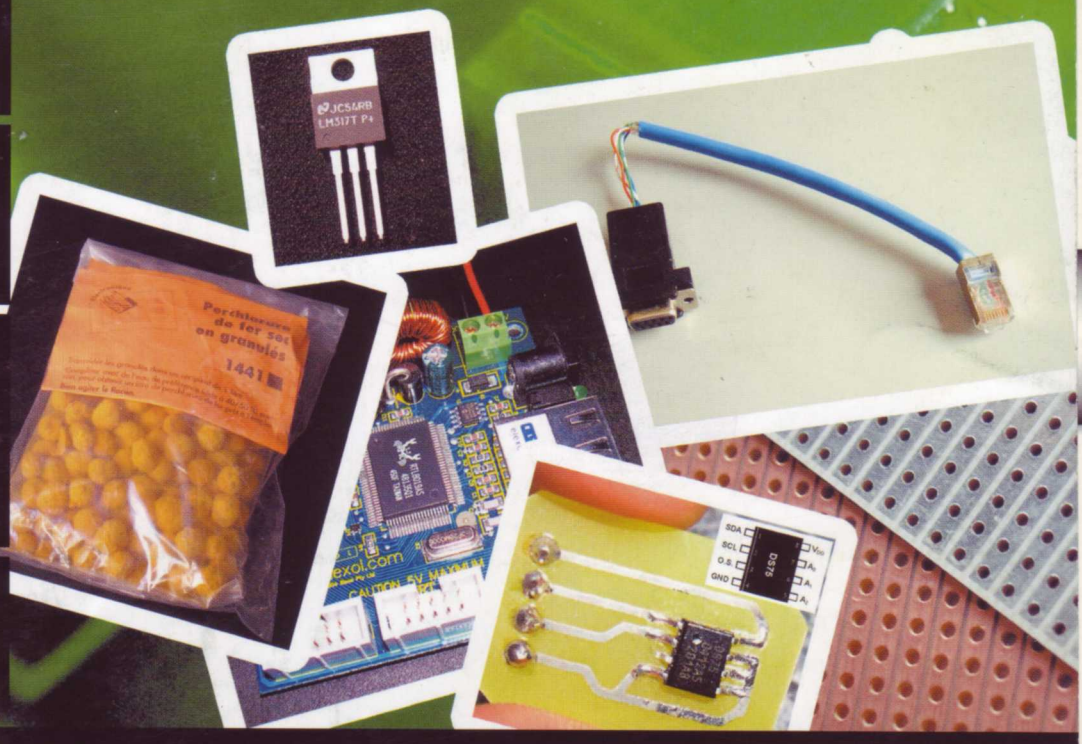
L 15066 - 23 H - F: 6,40 € - RD

Hors série N°23

- Les bases pour bien débuter en électronique
- Présentation et programmation du port parallèle
- Programmation et interfaçage d'un microcontrôleur USB Freescale 68HC908JB8
- Présentation et programmation d'un capteur de température sur bus I2C
- Configuration et utilisation d'un récepteur infra-rouge pour Linux
- Présentation et premiers pas avec les microcontrôleurs Atmel AVR
- Pilotage de relais depuis Linux

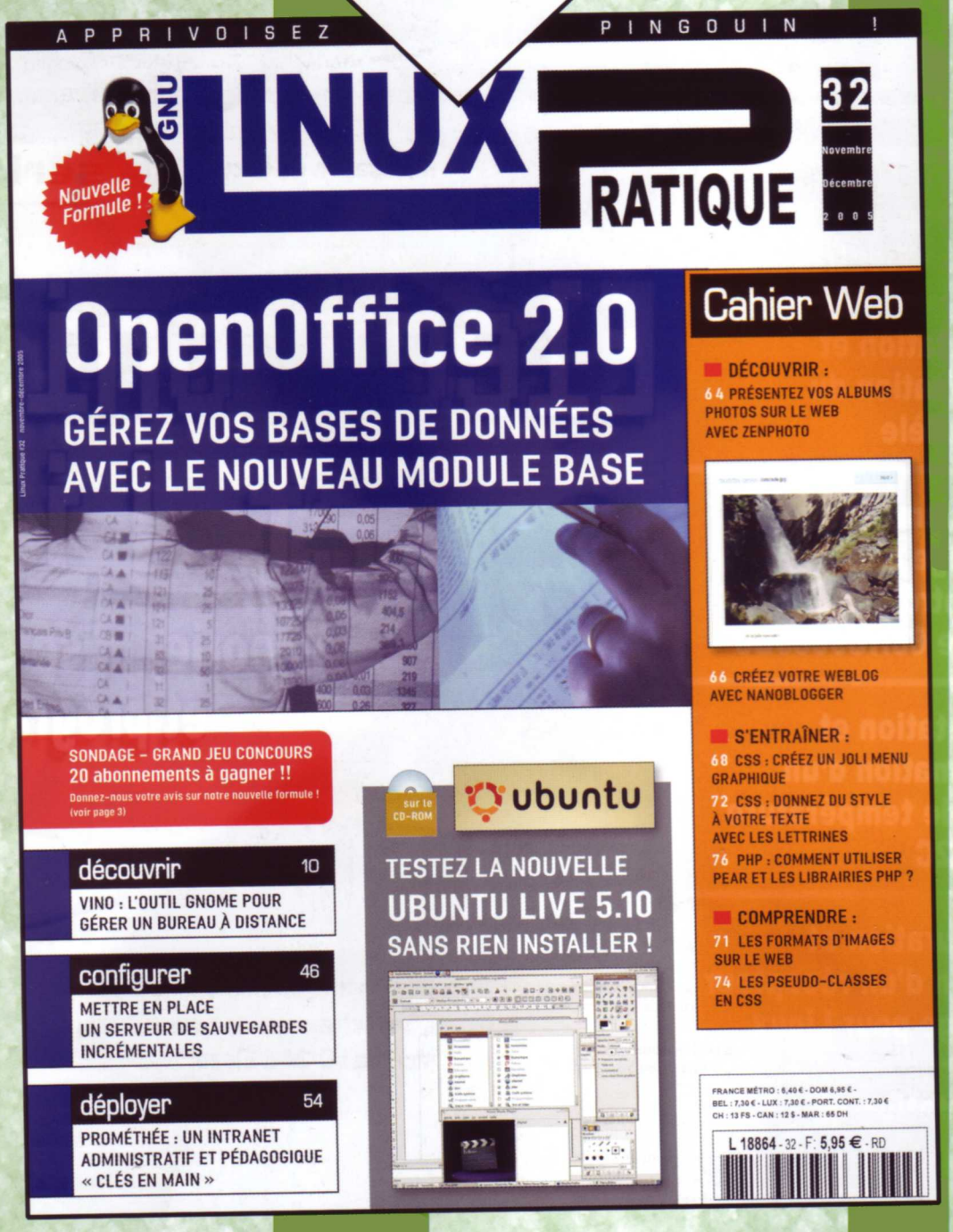
Electronique et Linux

Théorie, mise en œuvre et programmation



-- 100% ELECTRONIQUE --

Linux Pratique nouvelle formule est arrivé !



Disponible en kiosque le 10 novembre

edito

Bienvenue dans ce nouveau hors-série,

Cela fait quelques temps que l'idée me trotte dans la tête : mélanger deux passions en un seul magazine, GNU/Linux, d'une part, et l'électronique d'autre part. Après tout, les deux domaines, vous le verrez, ne sont pas si éloignés que cela, bien au contraire.

Mais le plus intéressant reste la motivation qui mène des systèmes UNIX en sous-licences libres aux expérimentations autour de l'électronique. Lorsque j'ai touché mon premier GNU/Linux fin 94/début 95, il fallait tout configurer. Plus exactement, il fallait batailler sec pour obtenir du son sur sa Compatible SoundBlaster, fourrer son nez dans la documentation d'XFree86 3.0 et jongler joyeusement entre `ifconfig` et `route` pour configurer son réseau personnel. Aujourd'hui, cette exploration forcée n'est plus nécessaire, la totalité des distributions s'installent clés en main et l'âme du pionnier reste un peu sur sa faim. Il ne reste que l'exploration du code et le développement pour retrouver l'exotisme des premières heures.

Le développement ou... quitter le milieu presque aseptisé du logiciel pour partir à la découverte de ce qui constitue encore la base de tout ordinateur : des composants, des connexions, des transistors et les lois strictes qui régissent le déplacement des électrons. C'est donc presque un chemin naturel qui conduit de GNU/Linux à l'électronique, poussé par la motivation qui se cache en chacun d'entre nous, celle de chercher la maîtrise de ce qu'il tient entre ses mains, source inépuisable de plaisir, de frustration et de défi... j'ai nommé, l'ordinateur. C'est avec cette motivation que le présent magazine a été rédigé. Je parle en mon nom et sans doute également en celui des auteurs qui ont contribué à ce hors-série. Et j'espère que c'est avec cette même motivation et ce même esprit d'explorateur que vous le lirez.

Je vous souhaite une heureuse lecture que j'espère enrichissante, que vous soyez utilisateur GNU/Linux partant à la conquête du monde « d'en dessous » ou électronicien amateur à la conquête du monde UNIX.

Denis Bodor

sommaire

Linux Magazine France
Hors Série

est édité par
Diamond Editions
B.P. 121 - 67603 Sélestat Cedex
Tél. : 03 88 58 02 08
Fax : 03 88 58 02 09
E-mail :
cial@ed-diamond.com
Service commercial :
abo@ed-diamond.com
Site :
www.ed-diamond.com

Directeur de publication :
Arnaud Metzler

PRINTED IN Germany / Imprimé en allemagne / Dépôt légal : 3e Trimestre 1998 / N° ISSN : 1291-78 34 / Commission Paritaire : 09 03 K78 976 / Périodicité : Bimestrielle / Prix de vente : 6,40 €uros

RÉDACTION
Rédacteur en chef :
Denis Bodor
Conception graphique :
Franck Toussaint
Responsable publicité :
Véronique Wilhelm
Tél. : 03 88 58 02 08
Service abonnement :
Tél. : 03 88 58 02 08

Impression :
VPM DRUCK /
www.vpm-druck.de
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

Denis Bodor :: db@ed-diamond.com :: lefinnois@lefinnois.net

SORTEZ DU SOFTWARE !

Peut-être vous demandez-vous pourquoi nous avons décidé de consacrer un hors-série complet au monde de l'électronique alors qu'il ne s'agit pas d'une spécialité de GNU/Linux. Cette brève introduction répondra à cette question et vous donnera et exposera les pré-requis pour poursuivre la lecture de ce magazine et ensuite explorer par vous-même.

Ce n'est pas une nouveauté, informatique et électronique sont deux domaines intimement liés. C'est encore plus vrai depuis quelques années avec l'arrivée en masse des microcontrôleurs.

Point de rencontre entre les deux disciplines, ces circuits embarquent les trois éléments de base d'un ordinateur : CPU pour le traitement des informations, mémoire pour le stockage et unités d'Entrées/Sorties permettant de communiquer avec le reste du monde.

Loin de la puissance de calcul de nos machines de bureau et de nos serveurs, les microcontrôleurs n'en sont pas moins des petits ordinateurs embarquant un code développé spécifiquement. La plupart d'entre eux permettent d'effacer et de répéter l'opération de chargement de code plusieurs milliers de fois.

GNU/LINUX ET L'ÉLECTRONIQUE, ÉTAT DES LIEUX

Il ne s'agit là que d'un exemple, de plus en plus de montages s'interfacent avec les ordinateurs ou les réseaux TCP/IP. En tant qu'utilisateur d'un système UNIX, qu'administrateur système ou que développeur, ce nouvel aspect de l'électronique informatisée représente un eldorado technique.

Cependant, il vous faudra acquérir de nouvelles connaissances pour l'aborder. Parallèlement, l'électronicien amateur ou professionnel devra faire le chemin inverse.

Le présent magazine tente de faciliter l'approche pour les deux mondes, d'une part le « peuple du soft » et de l'autre celui du « hard ».

En tant que lecteur de GNU/Linux Magazine, vous avez peut-être, comme moi, parcouru les kiosques et consulté la presse spécialisée en électronique. Le fossé qui sépare les deux mondes est très important sur bien des points :

- La notion d'*Open Source* et de Logiciel libre est très peu présente. Alors même que de plus en plus de montages présentés utilisent des microcontrôleurs, le partage ou même la divulgation des sources est très rare. Certains auteurs plutôt que d'ouvrir leur code en arrivent même à demander aux lecteurs d'utiliser un éditeur hexadécimal pour personnaliser, par exemple, une chaîne de caractères dans le binaire qu'ils mettent à disposition.

- Les langages communément utilisés sous UNIX sont peu présents. C, Perl et Python apparaissent rarement au bénéfice de Delphi, du Basic et, dans le meilleur des cas, de l'assembleur. Ce dernier sort un peu du lot, puisque presque naturel d'apprentissage au programmeur C et surtout indispensable avec certains composants aux capacités mémoire réduites.

- GNU/Linux et les UNIX de manière générale (*BSD, Solaris et autres) sont tout simplement ignorés dans la quasi-totalité des articles. La plate-forme la plus populaire reste Microsoft Windows dans le domaine pourtant très technique et peu grand public qu'est celui de l'électronique amateur.

Pour toutes ces raisons, l'électronique de loisir reste très difficile d'accès aux utilisateurs UNIX évoluant dans des environnements très différents.

Pourtant, GNU/Linux est sans doute le système d'exploitation le plus à même de répondre aux besoins avec la disponibilité des sources, de nombreux logiciels, une ouverture totale et des outils de développement puissants et reconnus pour leur qualité.

Voilà pourquoi depuis quelques mois, des articles traitant d'électronique sont publiés dans GNU/Linux Magazine et rencontrent un grand intérêt de votre part. Le présent hors-série tente d'être à la fois une introduction théorique et pratique.

Il y a fort à parier que certains d'entre vous sauteront tel ou tel article qui représentera, pour d'autres, l'article le plus important. Si le succès est au rendez-vous de ce numéro spécial, soyez certain qu'il donnera suite à d'autres, plus pratiques et sans l'aspect introductif nécessaire à celui-ci.

PRÉ-REQUIS

L'aspect théorique est traité dans les pages qui suivent. En cela, aucune connaissance en électronique, autre que le bon sens et la logique, n'est nécessaire pour appréhender ce magazine. Côté GNU/Linux et développement, une connaissance du système est indispensable et des familiarités avec le langage C seront parfois nécessaires.

Mais les pré-requis les plus importants sont d'ordre matériel. Contrairement au logiciel, le développement matériel repose sur des éléments physiques.

ÉQUIPEMENT

Pour faire connaissance avec l'électronique, un équipement de base est nécessaire. Celui-ci n'est pas forcément du matériel haut de gamme pour débuter, même si par la suite, l'expérience aidant, vous vous dirigerez vers des produits plus onéreux.

Fig. 1

- Le multimètre. C'est l'un des matériels indispensables. Le multimètre permet de mesurer une tension, l'intensité d'un courant, une résistance, contrôler une diode, reconnaître un type de transistor, etc. Les modèles courants se composent d'un afficheur LCD, de deux sondes (câbles) et d'une molette centrale permettant de choisir le mode de fonctionnement. Les modèles d'entrée de gamme se trouvent entre 10 et 15 euros. Les modèles plus sophistiqués, plus précis et permettant de mesurer toutes sortes de choses (température, fréquence, hygrométrie, etc.) coûtent 40 à 60 euros (voire plus). Choisissez un modèle de base incluant les 5 types de mesure classiques pour débuter. Vous pourrez ensuite, en fonction des besoins, vous tourner vers un modèle plus complet.

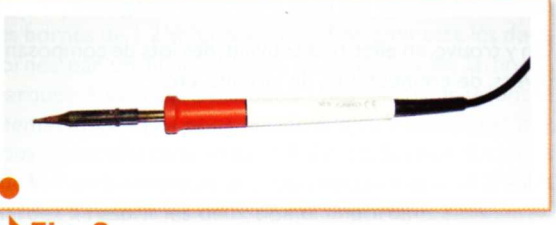


Fig. 2

- Fer à souder. Voilà un matériel à ne pas choisir au hasard. Deux solutions s'offrent à vous, soit acquérir un modèle moyen de gamme et peut-être en changer par la suite, soit prendre un fer un peu plus onéreux, mais que vous garderez des dizaines d'années en ne changeant qu'une partie occasionnellement. Dans tous les cas, évitez les modèles bas de gamme. On les reconnaît facilement, ils sont très abordables et leur finition révèle la qualité générale du produit.

Les fers à souder se distinguent par leur puissance qui leur confère température de soudure et vitesse de chauffe. En électronique, on utilise des fers de quelques dizaines de Watts.

Pour les fers milieu de gamme la relation puissance/température est claire et précisée lors de l'achat. 25 W (autour de 380°C) est une bonne solution pour débuter. Un fer de ce type vous coûtera quelques 30 euros. Des fers ou station de soudure permettant de régler la température coûtent entre 50 et 80 euros. La température est importante.

Trop élevée, elle risque d'endommager certains composants, trop faible, elle permet difficilement de souder ou dessouder de gros éléments.

Conseil tout ce qu'il y a de personnel, j'ai toujours été satisfait de mon fer JBC (non, je n'ai pas d'action chez le fabricant). Bien entendu, avec le fer, il vous faut de l'étain permettant de faire des soudures.

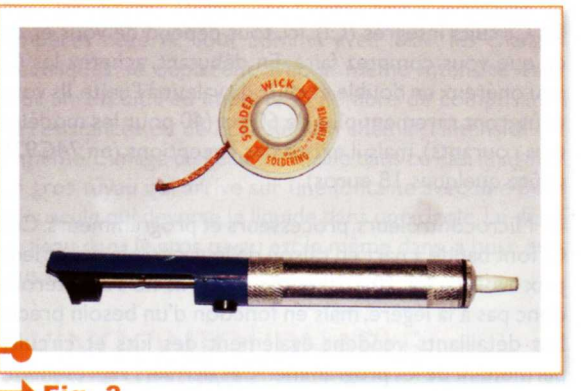


Fig. 3

- Pompe et tresse à dessouder. Indispensable si vous comptez faire de la récupération de composants et bien pratique en cas d'erreur, ces équipements vous permettent de retirer l'étain sur un circuit et de désolidariser le composant de l'ensemble. Évitons la polémique et les batailles de clochers. Certains préfèrent la tresse et d'autres la pompe. Personnellement, j'utilise les deux selon les travaux.

- Platines à essais. Cet équipement n'est pas critique, mais très intéressant pour débuter. Une platine permet d'insérer des composants et de tester un circuit complet sans soudure. Nous en reparlerons dans l'article concernant

la fabrication des circuits. C'est également une très bonne solution pour faire ses tous premiers pas en électronique et apprendre les bases.

COMPOSANTS NEUFS

Les composants sont la matière première que l'électronicien utilise pour créer des circuits. Certains de ces composants ont un coût si faible qu'ils se vendent à la dizaine voire à la centaine.

- Condensateurs, résistances et diodes. Ce sont les composants de base. On en utilise dans tous les montages ou presque et mieux vaut se constituer un petit stock. Les résistances sont vendues par valeur à la dizaine (15 centimes pour dix). Les détaillants de composants proposent souvent des boîtes contenant des résistances panachées. Pour moins de 3 euros, vous aurez un jeu de 200 résistances de toutes valeurs. Pour 15 euros, ce sont 1000 pièces qui vous seront proposées et, avec un peu de chance, dans une belle boîte réutilisable. Débuter avec un tel ensemble est une bonne idée. Vous complèterez avec d'autres achats au fur et à mesure et en fonction des valeurs que vous utiliserez le plus. Il en va de même pour les condensateurs, mais les jeux sont spécifiques à des types (céramique, chimique, etc.). Enfin, les diodes s'achèteront en fonction des besoins. Il n'est pas vraiment utile d'en disposer de toutes sortes en quantité. Dans tous les cas d'achat pour ce type de composants, n'achetez jamais exactement la quantité nécessaire, mais doublez-la (sauf pour les composants rares et chers). Leur prix minuscule justifiera la dépense inutile sur le moment.

- Circuits intégrés (CI). Ici, tout dépend de vous et de ce que vous comptez faire. En débutant, achetez les CI peu onéreux en double et ceux de valeur à l'unité. Ils vous coûteront rarement plus de 60 cts (40 pour les modèles très courants), mais il existe des exceptions (un 74C925 coûte quelques 18 euros).

- Microcontrôleurs, processeurs et programmeurs. Ces CI font bande à part en raison de leur complexité et leur prix avoisine les 10 à 20 euros pièce. Ils ne s'achèteront donc pas à la légère, mais en fonction d'un besoin précis. Les détaillants vendent également des kits et circuits permettant de les programmer. Ces derniers représentent une solution si vous ne vous sentez pas capable de créer votre propre programmeur. Pensez toutefois à vous assurer de la compatibilité avec les applications GNU/Linux qui n'est jamais clairement spécifiée.

- Câbles et connecteurs. Lorsqu'on mélange électronique et informatique, il est souvent nécessaire de disposer de connecteurs adéquats (parallèle, série, USB). Ces éléments sont vendus par les détaillants de produits et, à moins de favoriser une interface pour tous vos montages, ils seront acquis au besoin à l'unité ou en double. Les câbles et fils de connexion sont vendus au mètre, mais, dans bien des cas, la récupération sera une solution plus rentable.

RÉCUPÉRATION

Avec la masse de produits électroniques qui nous entoure, il n'est pas difficile de trouver son bonheur dans les poubelles.

Hi-fi, ordinateurs, téléphones ou encore imprimantes sont des sources de récupération pour peu que le produit ait quelques années.

Aujourd'hui, l'intégration poussée des composants ne permet plus la récupération (technologie CMS et autres).

- Câbles et connecteurs. C'est là le consommable qui sera le plus facile et le plus rentable à récupérer. Les câbles téléphoniques et réseaux sont idéaux pour tout type de travaux. Un câble paires torsadées pourra être utilisé dans l'état ou désassemblé. En effet, les câbles dit « CAT 5 » sont souvent mis au rebut en raison de leur usure. Mais une gaine ou un blindage endommagé ne sera pas un problème une fois les 4 paires extraites pour servir de câble d'alimentation ou connexion pour des signaux logiques. Bien entendu, pour des applications pointues (USB, au débit, etc.) mieux vaut prendre un câble neuf et conforme à l'utilisation qui en sera faite.

- Condensateurs, résistances et diodes. Il n'est pas rentable de récupérer ces composants très peu cher à l'achat. Le temps que vous passerez à dessouder et tester ces composants vaudra plus que leur prix neuf dans tous les cas. Seules les LED spéciales tirent leur épingle du jeu. Souvent facile à récupérer et à tester, certaines ont un coût non négligeable (blanches ou bleues).

- Circuits intégrés. Deux cas se présentent. Soit le CI est soudé et la récupération est trop délicate, soit le circuit est monté sur un support et il pourra être retiré sans le moindre problème. Lorsqu'un composant est fragile et supporte mal les hautes températures, on soude un support dans lequel il s'enfiche. Ainsi, les CI les plus chers sont aussi ceux que l'on retrouve sur des supports. Remarquez qu'aujourd'hui, le format des CI dans les produits manufacturés n'est plus DIL/DIP ou PLCC, facilement manipulable, mais SOIC, TSOP ou QFP quasiment inutilisable pour un humain.

Gardez à l'esprit que la récupération se fait sur des matériels au rebut. Ils se trouvent là parce qu'il sont défectueux ou trop vieux.

Vérifiez au multimètre ou à l'aide de circuits simples leur bon fonctionnement avant de les utiliser. Une dernière solution pour se procurer des composants et matériels consiste à utiliser les sites d'enchère en ligne.

On y trouve, en effet, très souvent, des lots de composants divers, de connecteurs, de circuits, etc.

PETITE INTRODUCTION À L'ÉLECTRONIQUE À L'USAGE DES SYSADMINS ET CODEURS

Lorsqu'on est développeur, sysadmin ou utilisateur avancé de système UNIX, on éprouve bien du mal à débuter en électronique. Mais rassurez-vous, un électronicien aurait tout autant de mal, voire plus, à appréhender un système comme GNU/Linux.

L'objectif de cet article est de servir d'introduction à l'électronique selon une optique particulière. Bien des textes, ouvrages et site Web d'initiation sont difficiles d'accès pour un utilisateur ayant déjà un bagage informatique.

Je vais tenter ici de vous apprendre les principes de base de l'électricité et de l'électronique comme j'aurai aimé qu'on me les enseigne, avec des mots d'informaticien. Ceux d'entre vous qui ont déjà tenté de se faire expliquer tel ou tel point de détail par un électronicien savent de quoi je parle. Tout comme pour nous, certains principes et mœurs UNIX semblent logiques, pour un amateur d'électronique certaines bases semblent innées. Ce n'est pas toujours le cas.

COURANT ET TENSION

L'électricité, c'est deux notions importantes. Nous avons d'une part la tension, exprimée en Volts, et, de l'autre, le courant, exprimé en Ampères.

La tension est une différence de potentiel entre deux points. Une prise murale, par exemple, présente une tension entre les deux trous de 230 Volts. Dans votre machine, entre le fil rouge d'un connecteur d'alimentation de disque dur et le fil noir, la différence de potentiel est de 5 Volts. Si nous décidons que le fil noir est la masse (ce qui est généralement d'usage) alors nous avons +5 Volts. Le + exprime le sens de déplacement de la charge électrique.

Une source de courant génère une différence de potentiel à ces bornes. Une pile par exemple est un générateur qui, de par son fonctionnement, crée une tension entre ses bornes de 1.2 Volts environ. Si l'on connecte les deux bornes par un fil, le courant se déplacera de la borne marquée + vers celle marquée - (les électrons feront le chemin inverse). Si nous décidons que la masse est le -, nous avons une tension de + 1.2 Volts. Si nous décidons que le + est la masse, nous avons une tension de -1.2 Volts. Gardez à l'esprit les deux points importants :

- La tension est une différence de potentiel entre deux points ;
- La masse est choisie arbitrairement pour l'ensemble d'un montage complet. Voilà pourquoi on connecte toujours toutes les masses ensemble. C'est la masse commune et il n'en existe qu'une seule pour un ensemble indépendant.

Les charges électriques se déplacent dans le conducteur (le fil). La grandeur de ce déplacement est nommée « intensité ». On représente souvent la notion d'intensité en utilisant l'image de l'eau dans un tuyau. L'intensité est le débit de l'eau, le nombre de litres/seconde qui y circulent. Avec les charges électriques, l'intensité est exprimée en Ampères noté, A. Tout comme avec l'eau, les charges électriques se déplacent avec la même intensité dans tout un circuit. Peu importe le nombre de composants, de résistances ou de LED sur son chemin, l'intensité est la même. L'image de l'eau est idéale dans ce cas. Imaginez un gros tuyau qui arrive sur une fontaine avec une buse minuscule qui déverse le liquide dans une rigole. Le débit de l'eau dans le gros tuyau est le même dans la buse que, plus loin, dans la rigole.

L'INCONTOURNABLE LOI DE OHM

Si vous voulez faire de l'électronique vous DEVEZ comprendre et assimiler cette loi. C'est un pilier indispensable sur lequel repose tout ce que vous apprendrez par la suite. La loi d'Ohm établie la relation entre le courant et la tension. Si nous reprenons notre image de l'eau et du tuyau, le débit est l'intensité du courant et la pression la tension.

Imaginez un tuyau souple dans lequel circule de l'eau de manière continue. Si nous écrasons le tuyau, le débit de l'eau va baisser et la pression en amont de l'écrasement augmente. En aval la pression diminue. Plus l'écrasement sera important plus le débit va baisser (dans tout le tuyau) et plus la différence de pression sera importante entre l'amont et l'aval du point d'écrasement.

Denis Bodor :: db@ed-diamond.com :: lefinnois@lefinnois.net

Il en va de même pour l'électricité. L'écrasement présente une résistance, plus celle-ci est importante plus l'intensité du courant diminue et plus la tension entre avant et après la résistance est importante. La résistance (force d'écrasement) est exprimée en Ohms et notée avec le symbole Omega.

L'anode est la borne + et la cathode la borne -. Un moyen efficace de s'en souvenir est de se dire que l'Âne est plus grand que le Coq (pour A et C). C'est tellement idiot qu'on s'en souvient.

Il existe une relation entre la tension aux bornes d'un circuit, l'intensité du courant qui le traverse et la résistance qu'il présente. Cette relation est la loi d'Ohm :

U = R.I

La tension aux bornes en Volts est égale à la résistance en Ohms fois l'intensité en Ampères. Un exemple clair et précis s'impose pour concrétiser cela. La figure 1 présente un circuit complet. Le courant traverse le circuit du début à la fin.

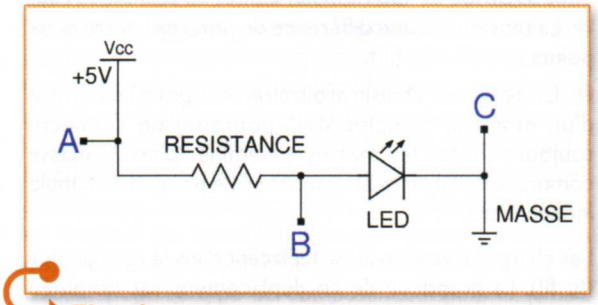


Fig. 1

Son intensité dépend de la résistance. C'est l'écrasement du tuyau qui limite le débit de l'eau. La tension aux bornes du circuit est de 5 Volts (entre A et C). Ce n'est pas nous qui décidons, nous avons cette différence de potentiel entre Vcc et la masse dès le départ, comme aux bornes d'une pile.

Nous savons qu'une LED, diode électro-luminescente, a besoin d'être traversée par un courant de 10 mA, soit 0.001 Ampère. On sait également que la tension de seuil de la LED est de 2 Volts. C'est la tension minimum aux bornes de la LED (entre B et C) pour qu'elle émette de la lumière.

L'intensité du courant et la tension de seuil proviennent de la documentation du fabricant de la LED et sont respectivement notées Vf et If.

Ces deux caractéristiques dépendent de la couleur de la LED et de l'intensité lumineuse qu'elle produit. 10 mA et 2 V sont des valeurs courantes pour des LED rouges standards.

Nous avons toutes les informations nécessaires, il ne reste plus qu'à écraser le tuyau pour régler le débit. Aux bornes de la LED doit se trouver une tension de 2 V, il nous reste donc 3 V qui seront aux bornes de la résistance (entre A et B).

Nous avons besoin d'une intensité de 10 mA. Il ne reste qu'à appliquer la loi d'Ohm :

U = R.I
*3 = R * 0.01*
3/0.02 = R
R = 300 Ohms

De telles résistances ne se trouvent pas facilement, on utilisera donc la résistance immédiatement supérieure, soit 330 Ohms. Ce qui nous donne :

U = R.I
*3 = 330 * I*
3/330 = I
I = 0.009 A = 9mA

La question qu'on se pose tout naturellement en débutant est : qu'est ce qui se passe si j'utilise une résistance de 100 Ohms ? La réponse de l'électronicien sera simple : il ne faut pas le faire, c'est mal. Dans la pratique, si on mesure, la tension aux bornes de la LED et de la résistance reste presque inchangée.

C'est le courant qui traverse le circuit qui augmente (30 mA) et l'ensemble n'est plus viable. Dans les calculs, cela reste indéfini. On dépasse une caractéristique documentée, on arrive dans l'inconnu, tout peut arriver.

Au final la réponse frustrante de l'électronicien est juste, il ne faut pas que cela arrive. C'est un peu comme jouer avec des variables non initialisées en programmation.

Dans les faits, cela génère souvent un changement d'aspect d'un ou plusieurs composants et le dégagement d'une odeur caractéristique très désagréable. Avec du code, vous risquez de perdre des données, ici vous risquez de perdre des composants à quelques centimes d'euro, voire une interface ou un circuit à plusieurs dizaines ou centaines d'euros.

Ce calcul simple une fois compris permet de connecter et d'alimenter des composants dont les caractéristiques sont documentées et doivent être respectées. Cependant, il faut également prendre en compte la puissance qui traverse la résistance qu'on choisit.

Ici, la tension de départ était 5 V, une valeur courante lorsqu'on travaille avec des circuits logiques et du matériel informatique. Le calcul suivant est également exact :

Tension entre A et C = 50 Volts
48 Volts aux bornes de la résistance = U
U = R.I
*48 = R * 0.01*
48/0.01 = R
R = 4800 = 4.8 K Ohms

Le calcul fonctionne, mais la résistance ne va pas apprécier du tout. Il faut faire intervenir la notion de puissance, exprimée en Watts. La puissance P est calculée en multipliant l'intensité par la tension, soit I fois U :

$P = U.I$

$P = 48 * 0.01 = 0.48$ Watts

Les résistances sont caractérisées par leur valeur en Ohms et la puissance qu'elles peuvent dissiper (transformer en chaleur). Or, les résistances standards sont souvent dites « quart de Watt ».

Elles peuvent dissiper 0.25 Watts maximum. Au-delà de cette valeur, elles surchauffent et sont détruites. Ici, nous sommes largement au-delà de cette limite. Pour régler le problème, il faut répartir l'écrasement du tuyau en plusieurs points. En d'autres termes, il faut plusieurs résistances. Connectées en série (l'une à la suite de l'autre), leur valeur en Ohms s'additionne, mais la tension à leurs bornes se divise.

Bien entendu, le courant qui traverse le circuit ne change pas. Ainsi, en utilisant quatre résistances, on arrive à :

$48/4 = I2 = R1 = R2 = R3 = R4$

$U = R1.I$

$I2 = R1 * 0.01$

$I2/0.01 = 1200$

$P = U.I$

$P = I2*0.01 = 0.12$ Watts

Là encore, difficile de dire ce qui se passera si on ne prend pas en considération ces données. Mais peu importe en réalité de savoir quel composant sera détruit en premier et quelles seront les conséquences de cette destruction. Il ne faut pas que cela arrive.

Tab. 1

Bague1	Bague2	Bague3
Chiffre1	Chiffre2	Multiplicateur
Noir 0	0	1 Ohm
Brun 1	1	10 Ohms
Rouge 2	2	100 Ohms
Orange 3	3	1K Ohms
Jaune 4	4	10K Ohms
Vert 5	5	100K Ohms
Bleu 6	6	1M Ohms
Violet 7	7	10M Ohms
Gris 8	8	-
Blanc 9	9	-

Les codes de couleurs des résistances

Nous venons de voir un circuit simple, mais d'autres cas de figure peuvent se présenter. La figure 2 montre deux circuits. Le circuit A est un montage en parallèle de deux

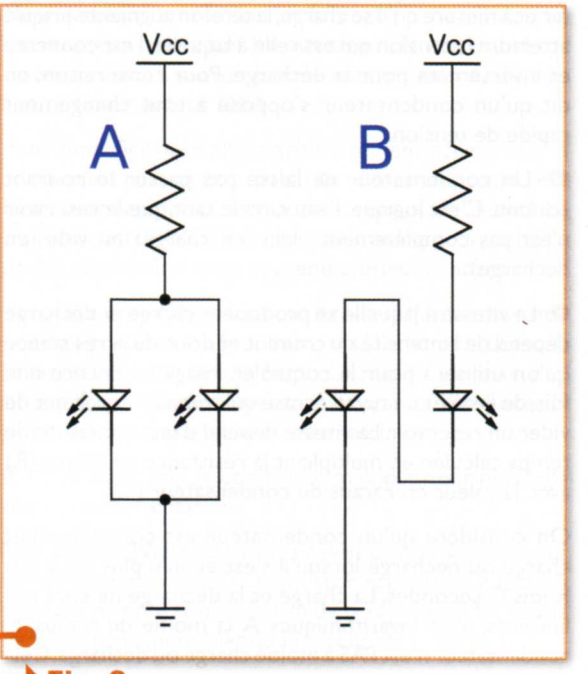


Fig. 2

LED. Chacune d'elles a besoin d'être traversée par un courant de 10 mA. La totalité du courant traversant alors le circuit est de 20 mA et il faudra calculer la valeur de la résistance sur cette base (R = 150 Ohms).

Le circuit de droite met les LED en série, le courant qui le traverse est 10 mA. N'oubliez pas, l'intensité du courant est la même en tout point du circuit. En revanche, la tension de seuil de chaque LED est de 2 Volts. La résistance sera, là encore, à calculer avec U = 5 - (2*2) et vaudra 100 Ohms.

Comprendre la loi d'Ohm est capital. Ne passez pas à la suite si cela ne vous semble pas clair. Dans le doute, si ces explications ne vous paraissent pas limpides, recherchez sur le Web. Une masse très importante de sites d'initiation en français existe.

Série E12 de résistances regroupe des valeurs standardisées. 1.0, 1.2, 1.5, 1.8, 2.2, 2.7, 3.3, 3.9, 4.7, 5.6, 6.8, 8.2 par décade. Ainsi la valeur 3.3 se retrouvera sous la forme de résistance de 33, 330, 3300 ou encore 33 K Ohms.

CONDENSATEURS

Nous venons de faire connaissance avec les LED et les résistances. Il est temps de passer à un autre composant important et courant : le condensateur. Il en existe de plusieurs types : chimique, plastique, tantale, céramique... Tous possèdent la même propriété, celle de pouvoir accumuler une certaine quantité de charge électrique et de pouvoir la restituer. Cette capacité de stockage est mesurée en Farads. Souvent, lorsque cette capacité est faible, on parle de « capa » plutôt que de condensateur.

Dans le cadre de cette introduction, il faut remarquer un certain nombre de choses à propos du condensateur :

- La tension à ses bornes dépend du courant de charge et de décharge du condensateur. Autrement dit, lorsqu'il se remplit ou se vide, la tension à ses bornes change. Au

fur et à mesure qu'il se charge, la tension augmente jusqu'à atteindre la tension qui est celle à laquelle il est connecté et inversement pour la décharge. Pour cette raison, on dit qu'un condensateur s'oppose à tout changement rapide de tension.

- Un condensateur ne laisse pas passer le courant continu. C'est logique, l'eau circule tant que le réservoir n'est pas complètement plein (en charge) ou vide (en décharge).

Les résistances en série s'additionnent tout simplement. Placées en parallèle, on calculera 1/(1/R1 + 1/R2 + 1/R3...) pour obtenir la valeur de l'ensemble.

- La vitesse à laquelle se produisent charge et décharge dépend de l'intensité du courant et donc de la résistance qu'on utilisera pour la contrôler. Imaginez, encore une fois, de l'eau et un tuyau écrasé qui remplit ou permet de vider un réservoir. La vitesse dépend d'une constante de temps calculée en multipliant la résistance en Ohms (R) avec la valeur en Farads du condensateur (C).

On considère qu'un condensateur est complètement chargé ou déchargé lorsqu'il s'est écoulé plus de 3 fois R fois C secondes. La charge et la décharge ne sont pas linéaires, mais logarithmiques. A la moitié du temps, le condensateur n'est PAS à moitié chargé ou déchargé. Cela arrive bien plus tôt, mais je vous fais grâce de la formule exacte que vous retrouverez facilement.

mili (m) = 10^-3 = 0.001
micro (µ) = 10^-6 = 0.000001
nano (n) = 10^-9 = 0.000000001
pico (p) = 10^-12 = 0.000000000001
kilo (K) = 10^3 = 1000
mega (M) = 10^6 = 1000000

Les caractéristiques des condensateurs présentent bien des intérêts. Ils permettent de nettoyer une tension en se chargeant et déchargeant lorsque celle-ci varie. Ils permettent également d'obtenir des délais découlant de leur charge et décharge. C'est pour cette raison que votre carte mère en est remplie, ils assurent la stabilité des signaux électriques.

Les types de condensateurs déterminent leur capacité et indiquent la technique de fabrication. Le tableau 2 résume cela. Ils se distinguent également par la tension maximale qu'ils tolèrent. Celle-ci, comme leur capacité, est indiquée sur le composant. Certains condensateurs sont polarisés, il faut alors prendre garde lors de la connexion.

Les condensateurs chimiques sont faciles à reconnaître. Il s'agit de cylindres métalliques couverts d'une gaine de plastique. Peut-être vous êtes-vous déjà demandé à quoi servaient les petits motifs sur la partie supérieure (croix, trait, etc.). La réponse est assez inquiétante : c'est une sécurité en cas de mauvaise utilisation. Ils explosent par le haut comme un cotillon. C'est ce qui arrive lorsqu'on dépasse la tension de service. C'est extrêmement dangereux. D'autant plus avec les céramiques qui chauffent et « claquent » en projetant des éclats. Prenez garde !

DIODES

Une diode est un composant simple. Elle permet le passage du courant dans un sens, mais pas dans l'autre. Pour comprendre le fonctionnement, c'est très simple. Reprenez le circuit en figure 1 et ajoutez une diode entre la résistance et la LED. Dans un sens, la LED s'allume et, dans l'autre, non. Les charges électriques se déplacent du + vers le -. La diode est marquée d'une bague représentant le « barrage ». Si la patte du côté de la bague est reliée à la masse, le courant passe, si elle est au +, il ne passe plus.

Une diode est un composant semi-conducteur. Je n'entrerai, là encore, pas dans les détails (notion de jonction NP et PN), mais sachez qu'une diode est faiblement résistive pour une tension donnée et bloquante (résistance infinie) avec une tension inverse. Pour les circuits nécessitant une haute précision, il faut prendre en compte cette résistivité.

Une diode se caractérise par sa vitesse de blocage, le courant maximum qui peut la traverser et surtout la tension inverse qu'elle peut supporter. En effet, en fonction du type de diode, une tension inverse trop importante peut détruire le composant. Ainsi, la diode 1N4148 supporte un courant de 120 mA (courant direct continu noté « If ») et une tension de pointe inverse répétitive de 75 V (notée « Vrrm »). Pour une autre diode courante, la 1N4001, ce sera 1 A et 100 V.

Tab. 2

Type	Matériau	Capacité	Tension de service	Intérêt
Céramique	Céramique	1pF à 1nF	50V à 100V	Faible encombrement
Plastique	polyester/mylar (MKT), polypropylène (MKP), polystyrène (MKS), polycarbonate (MKC)	1nF à 1µF	50V à 400V	Stable, fiable et précis
Tantale goutte	Tantale	1µF à 100µF	6V à 50V	Petit et avec de grosses capacités mais cher
Électrolytique Chimique	Aluminium	4,7 µF à 4700 µF	10V à 350V	Encombrant, peu précis mais facile à récupérer

Type de condensateurs

Certaines diodes sont particulières, comme la LED qui émet de la lumière lorsqu'un courant suffisant la traverse. La diode Zener quant à elle permet de réguler la tension. Elle maintient une tension constante à ses bornes indépendamment du courant, par opposition aux résistances. Dans tous les cas, une diode reste une diode et la connexion dans le mauvais sens bloquera le passage du courant. C'est pour cette raison qu'on utilise des diodes, entre autres pour protéger un circuit ou un composant des maladresses humaines.

TRANSISTORS

Les transistors représentent la pierre angulaire de l'informatique. Inventée en 1947, ses inventeurs ont reçu le prix Nobel en 1956, tant l'importance de la découverte était grande.

Ce composant est un semi-conducteur tout comme la diode, mais se compose de trois bornes ou électrodes, l'émetteur, la base et le collecteur. Il existe plusieurs types de transistors et plusieurs modes de fonctionnement. Ceux qui nous intéressent ici sont les transistors bipolaires en régime ou mode de commutation. Ce dernier point est à préciser. Un transistor bipolaire est un amplificateur de courant. On injecte du courant entre la base et l'émetteur pour créer un courant entre l'émetteur et le collecteur qui est multiplié par le gain du transistor. Il existe deux types de transistors bipolaires, les PNP et les NPN. Nous verrons cela par la suite.

Plutôt que ne nous lancer dans de grandes théories, passons directement à la pratique. Ce qui nous intéresse, nous, informaticiens, ce n'est pas d'amplifier proportionnellement un courant, mais d'utiliser un transistor comme porte. Prenons le port parallèle. Celui-ci serait incapable d'alimenter un groupe de LED monté en parallèle nécessitant quelques 100 mA.

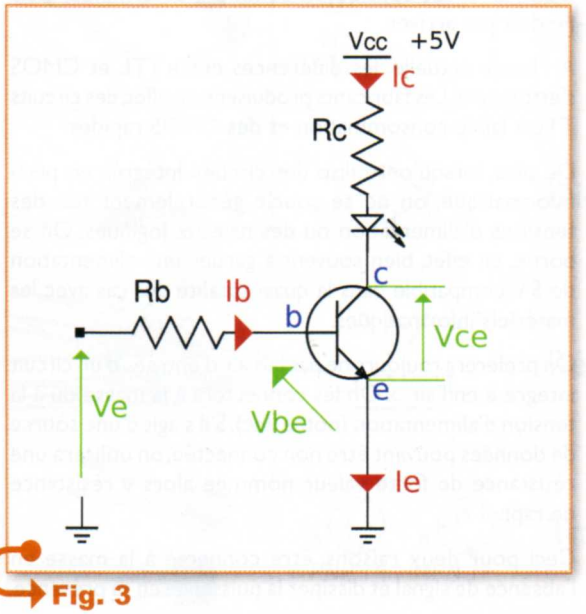


Fig. 3

Nous voulons que la sortie du port puisse piloter un interrupteur contrôlant l'alimentation. Notre transistor en commutation est soit bloqué (interrupteur OFF), soit saturé (interrupteur ON).

Pour simplifier, nous allons arbitrairement décider que la sortie n'est pas même capable de piloter une seule LED. Le montage complet en situation est donné en figure 3. Le transistor est un BC547. Il s'agit d'un modèle courant NPN. Sa documentation nous apprend plusieurs choses :

Vce_max = 45 V est la tension maximale que le transistor peut supporter entre le collecteur et l'émetteur. Lorsque le transistor est saturé, le problème ne se pose pas. Lorsqu'il est bloqué, il ne faut pas que la différence de potentiel soit supérieure à cette valeur.

ß = 200 est le gain, le coefficient multiplicateur noté Hfe dans la documentation.

Vce_sat = 0.2V est la tension présente entre le collecteur et l'émetteur lorsque le transistor est saturé. Il faut en prendre compte dans le calcul de la résistance Rc qui ne sera pas forcément identique à celle de la figure 1.

Vbe_sat = 0.7V est la tension présente entre la base et l'émetteur lorsque le transistor est saturé.

Commençons par recalculer Rc en suivant la loi d'Ohm, comme nous l'avons appris, mais cette fois, en tenant compte de Vce.

Nous avons donc :

5 - 2 - 0.2 = 2.8

U = Rc.I

*2.8 = Rc * 0.01*

2.8/0.01 = 280

Rc = 280 Ohms

Choisissons Rc à 220 comme résistance standard et déduisons l'intensité du courant :

U = R.I

*2.8 = 220 * I*

2.8/220 = 0.0127A = I

12.7 mA c'est un peu plus que la normale pour une petite LED, mais cela reste acceptable. MAIS, ce que nous venons de calculer c'est Ic. Or, avec un transistor, le courant traversant la LED n'est pas Ic mais Ie. Ie est égal à Ic+Ib. Ib est le courant minimum à fournir pour que le transistor sature.

Il se calcule en fonction du gain du transistor : Ib min = Ic / ß. soit ici 0.0127/200 = 0.00006. Comme le préconisent les électroniciens, multiplions cette intensité par 1.5 pour ne laisser aucun doute quant à la saturation du transistor, ce qui nous donne 0.00009 que nous appelons « Ib sat ».

Le courant traversant la LED sera donc 0.0127 + 0.00009, soit 0.01279 ou 13 mA arrondis largement. C'est toujours acceptable pour la LED. Nous en avons fini avec Rc.

Nous connaissons déjà l'intensité du courant de saturation, Ib sat. Il devient donc simple de calculer la résistance Rb à utiliser en appliquant la loi d'Ohm. Le transistor saturé présente 0.7 V entre la base et l'émetteur (Vbe_sat), la tension aux bornes de Rb sera donc 5 - 0.7 soit 4.3 V :

U = R.I

*4.3 = R * 0.00009*

4.3/0.00009 = 47777

Rb = 47777 Ohms

On utilisera une résistance de 47K Ohms nous permettant d'avoir une intensité de 0.091 mA, mais nous pourrions également utiliser une 33K Ohms qui nous donnerait 0.13 mA.

La résistance peut être ainsi réduite car, plus elle est petite, plus le courant sera important et dépassera l'intensité minimum permettant de saturer le transistor. Il faut simplement prendre garde à ne pas dépasser le courant maximum acceptable spécifié dans la documentation.

Avec un BC547, c'est 200 mA... nous avons de la marge. Il ne faut pas oublier non plus que Ie = Ic + Ib. Ici, ce n'est pas une différence de 0.03 mA qui changera quoi que ce soit pour la LED, mais ce n'est pas toujours le cas pour tous les composants.

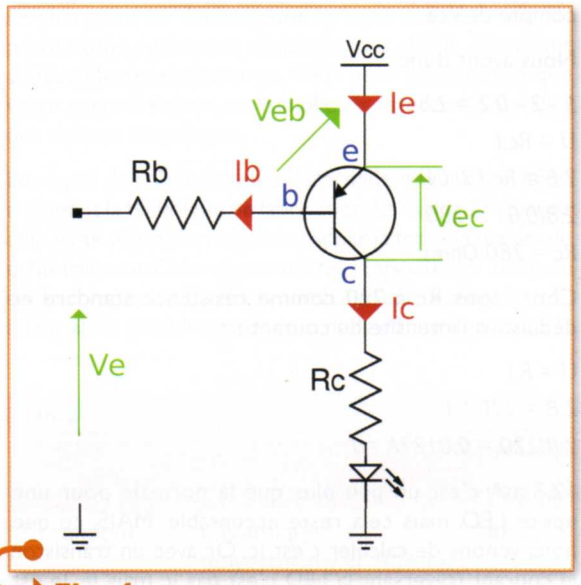


Fig. 4

Vous comprenez sans doute tout l'intérêt du transistor avec cet exemple. Avec un courant de 0.13 mA, nous pilotons un dispositif traversé par un courant cent fois plus important.

Les transistors PNP reposent sur le même principe, mais fonctionnent à l'inverse. Grossièrement, avec le même type de circuit et les calculs révisés, c'est lorsque la sortie est à la masse que le transistor est saturé et lorsqu'elle est au niveau haut qu'il est bloqué (figure 4).

CIRCUITS INTÉGRÉS

Les transistors, en raison de leurs propriétés, permettent de créer des circuits complexes. On arrive ainsi à créer toutes les portes logiques de l'algèbre booléen. En regroupant ces portes logiques et d'autres fonctionnalités en un seul composant, on crée des circuits intégrés.

Dans l'absolu, et sans tenir compte de la physique, on pourrait à l'aide des composants précédemment détaillés, créer un processeur tel que celui présent dans votre PC ou Mac, mais il vous faudra beaucoup de place.

L'intégration de tous ces composants en un seul fut une révolution en 1959. La société Texas Instruments montra qu'il était possible de graver plusieurs transistors et leurs interconnexions dans du silicium par des techniques photographiques.

Le circuit intégré (CI ou IC en anglais) était né. On trouve ainsi, sous plusieurs formes, des circuits complets pouvant être directement utilisés avec un encombrement très réduit.

Beaucoup de ces circuits intégrés sont devenus tellement courants qu'ils ne coûtent que quelques dizaines de centimes d'euros. Des broutilles face à un éventuel équivalent à l'échelle humaine. On peut regrouper les CI en trois grandes familles :

- Les TTL : Il s'agit de circuits logiques fonctionnant avec des niveaux TTL et alimentés entre 4.75 V et 5.25 V (idéalement en 5 V). Les TTL sont plus rapides mais consomment plus. Une entrée « en l'air » (non connecté) est considérée comme un 1 logique.

- Les CMOS (*Complementary Metal Oxyde Semi-conductor*) : Ils s'alimentent entre 3 V et 15 V et utilisent des niveaux logiques différents déterminés par la tension d'alimentation. Ils sont plus lents, mais consomment moins. Une entrée « en l'air » est considérée comme indéterminée. Cela ne doit pas arriver.

A l'heure actuelle, les différences entre TTL et CMOS s'estompent. Les fabricants produisent, en effet, des circuits TTL à faible consommation et des CMOS rapides.

De plus, lorsqu'on utilise des circuits intégrés en péri-informatique, on ne se soucie généralement pas des tensions d'alimentation ou des niveaux logiques. On se borne, en effet, bien souvent à garder une alimentation de 5 V compatible dans la quasi-totalité des cas avec les matériels informatiques.

On préfèrera toujours ne pas laisser d'entrées d'un circuit intégré « en l'air ». On les connectera à la masse ou à la tension d'alimentation (notée Vcc). S'il s'agit d'une source de données pouvant être non connectée, on utilisera une résistance de forte valeur nommée alors « résistance de rappel ».

Ceci pour deux raisons, être connecté à la masse en l'absence de signal et dissiper la puissance en sa présence.

Et oui, une connexion directe à la masse et au +5 est un court-circuit.

On parle alors de résistance de rappel à la masse (*pull-down* et *pull-up* pour l'inverse). On choisira la résistance non seulement en fonction de la tension, mais également de l'intensité du courant pour dissiper entre 1 kW (1000 W) et 10 kW. Par exemple, pour un circuit alimenté en 5 V et traversé par un courant de 20 mA, on utilisera des résistances entre 56 et 560 K Ohms. La figure 5 résume le principe des résistances de rappel.

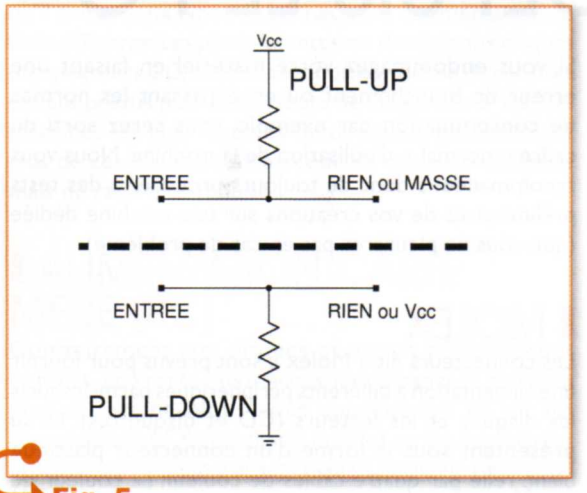


Fig. 5

CONCLUSION

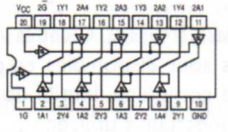
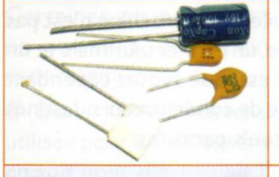
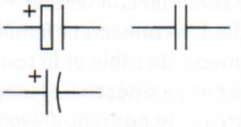
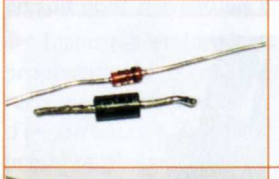
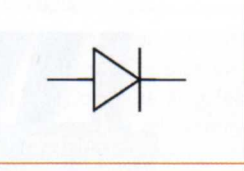
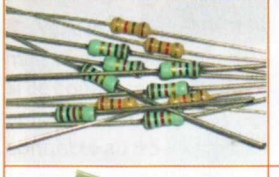
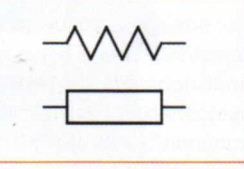
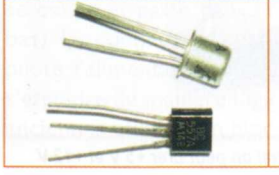
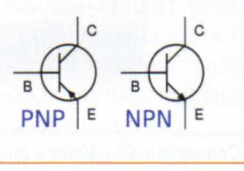
Cette brève introduction a normalement dû vous fournir les bases de l'électronique indispensables pour débuter. Cet article est relativement dense en informations.

Des livres et des sites complets existent regroupant des indications plus complètes. Difficile d'en indiquer un en particulier tant les explications peuvent être données différemment.

Faites une recherche sur le Web en précisant le point qui ne vous paraît pas clair et recoupez les informations des différents résultats.

Enfin, vérifiez chaque calcul et passez à la pratique avec des circuits simples (alimentation, résistance, LED, transistors).

Index

Photo	Schéma	Description
		Circuits intégrés
		Condensateurs
		Diodes
		Résistances
		Transistors

Denis Bodor :: db@ed-diamond.com :: lefinnois@lefinnois.net

OBTENIR DU COURANT DEPUIS LE PC

Quel que soit le montage ou le périphérique que vous souhaitez connecter à votre machine, il faut l'alimenter. Dans bien des cas, il n'est pas nécessaire de faire appel à une source d'alimentation externe, le PC lui-même peut fournir suffisamment de courant.

Un ordinateur, qu'il s'agisse d'un Mac ou d'un PC est une véritable alimentation en puissance. En effet, la plupart des éléments nécessitent une alimentation stable et régulée. Les tensions généralement disponibles et utiles sont +12 volts et +5 volts. D'autres sont disponibles un peu partout dans les machines, mais ne présentent que peu d'intérêt (à part peut-être le +3,3 volts parfois utilisé pour certains composants). Un PC, conçu dès le départ comme une architecture évolutive (ou plutôt « bricolable »), tire son alimentation d'un bloc spécialement prévu à cet effet. Se connectant sur une prise « secteur » de 230 volts alternatifs, cette alimentation fournit du courant pour chaque élément de la machine, et ce, sous diverses tensions. L'évolution du marché et la consommation des machines ne cessent de demander des alimentations de plus en plus puissantes.

Il n'est pas rare de nos jours de trouver des machines équipées d'alimentation de 400 watts. Le premier consommateur de courant est, bien entendu, le processeur, mais les adaptateurs graphiques ne sont pas en reste et affichent des consommations impressionnantes.

Il était, fut un temps, concevable de modifier l'alimentation d'une machine de manière à aérer les composants et à en retirer la ventilation afin de gagner en silence. Ceci n'est plus raisonnablement possible de nos jours en raison du nombre de watts en présence et de la dissipation thermique qui en découle. Le bloc d'alimentation est la source d'énergie de la machine. C'est de là que nous tirerons de manière plus ou moins directe le courant dont nous pouvons avoir besoin. La plupart des alimentations récentes sont protégées contre les mauvaises manipulations et les problèmes électriques en provenance de l'intérieur de la machine. Ainsi, un court-circuit, n'entraînera plus nécessairement le changement du bloc ou le remplacement d'un fusible. Dans la plupart des cas, la coupure d'alimentation (déconnexion du 230V ou arrêt par interrupteur) et un petit délai d'attente sont suffisants. Notez cependant que cela ne justifie pas pour autant un quelconque manque de rigueur. Le court-circuit reste la bête noire en la matière et doit être évité comme la peste. Dans tous les cas, le fait de connecter un périphérique ou un montage présente un risque qui n'est absolument pas couvert par la garantie constructeur.

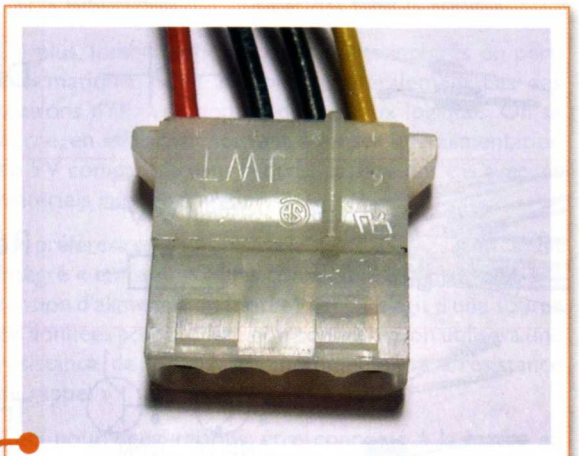
Si vous endommagez votre matériel en faisant une erreur de branchement ou en dépassant les normes de consommation par exemple, vous serez sorti du cadre « normal » d'utilisation de la machine. Nous vous recommandons donc de toujours procéder à des tests préliminaires de vos créations sur une machine dédiée (que vous ne pleurerez pas en cas de problème).

MOLEX

Les connecteurs dit « Molex » sont prévus pour fournir une alimentation à différents périphériques parmi lesquels les disques et les lecteurs (CD et disquettes). Ils se présentent sous la forme d'un connecteur plastique blanc relié par quatre câbles de couleur. La couleur de ces câbles indique la tension présente :

- Rouge (+5 V). Le courant disponible va généralement jusqu'à 30 ampères, ce qui est largement suffisant à la fois pour les périphériques « normaux » et les montages « maison ».
- Jaune (+12 V). Avec un courant allant jusqu'à 10 ampères, là aussi, nous avons largement de quoi faire.
- Noir (Masse). C'est la masse de la machine. Deux câbles sont disponibles et, par définition, il s'agit d'une masse commune. Inutile donc de vous questionner sur le connecteur à utiliser.

Faire sortir un connecteur Molex de la machine n'est pas difficile. Un connecteur femelle, un connecteur mâle et un bon mètre de câble et le tour est joué. Notez cependant que la distance nécessite l'usage de condensateurs destinés à nettoyer le courant d'éventuels parasites.



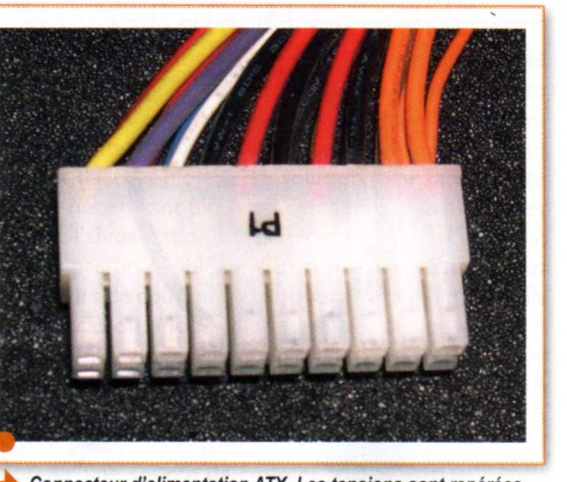
Connecteur dit « Molex » duquel on peut tirer +5 V et +12 V

Mac et PC partagent une codification commune des couleurs et des connecteurs identiques sur les machines actuelles. Notez cependant que ce n'est pas le cas avec des Mac plus anciens comme les LC-III. L'usage du voltmètre et la lecture des indications présentes sur le bloc d'alimentation restent de mise dans tous les cas. Enfin, il faut préciser qu'il existe deux types de connecteurs d'alimentation de taille différente. Les plus courants sont destinés aux disques durs, lecteurs CD, etc. Les seconds, en voie de disparition, étaient prévus pour les lecteurs de disquettes 3"1/2. Les machines les plus récentes ne disposent généralement plus de ces connecteurs au code de couleurs identique mais de taille plus réduite.

ALIMENTATION CARTE MÈRE (PC)

Contrairement aux disques et autres périphériques internes, une carte mère nécessite un panel de tensions plus important. Nous nous concentrons ici sur les alimentations ATX introduites avec les cartes mères pour processeurs Pentium II et supérieurs. L'ancienne génération (AT) se retrouve encore dans de vieilles machines. Moins « intelligentes », ces alimentations utilisaient un code de couleurs identique à l'ATX, exception faite des fonctions absentes, bien entendu.

- Noir (Masse). La masse commune de la machine.
- Rouge (+5 V). La codification est identique aux connecteurs Molex.
- Jaune (+12 V). Là encore, on retrouve la même correspondance que sur les connecteurs d'alimentation interne pour les périphériques.
- Orange (+3.3 V). Rarement utilisé dans les montages « maison », cette tension peut cependant s'avérer utile. Certaines mémoires et afficheurs LCD l'utilisent et on pourra, à loisir, venir chercher le courant sur ce connecteur.
- Bleu (-12 V). Cette tension négative est le plus souvent utilisée pour les ports série. Elle ne présente pas d'intérêt en soit pour d'éventuels bricolages.
- Blanc (-5 V). Les blocs d'alimentation récents ne proposent pas toujours ce connecteur. De plus en plus rarement utilisé (voire plus du tout), on évitera également d'en faire usage. La tension négative, en elle-même, ne présente généralement pas d'intérêt pour nous.
- Vert POWER-ON. Il ne s'agit pas d'une alimentation, mais d'une commande permettant d'activer l'alimentation. Si ce connecteur est mis à la masse, l'alimentation démarre et fournit toutes les autres tensions. Si ce connecteur est connecté au +5 V (signal TTL ON), le bloc d'alimentation se coupe et passe dans un mode STANDBY (voir plus bas). Le connecteur correspondant sur la carte mère pilote l'alimentation. Voilà pourquoi une machine peut s'éteindre de manière logicielle (contrairement aux plus anciennes utilisant un bloc d'alimentation AT).
- Violet (+5 V). Ce connecteur, dit « STANDBY », fournit du +5 volts en permanence à la carte mère. Cette dernière est ainsi en mesure de démarrer via l'appui sur le bouton POWER ou divers évènements générés extérieurement (comme le *Wake On* LAN) ou intérieurement (*Wake On Time*). La chose intéressante ici tient dans le fait d'avoir une source d'alimentation même lorsque la machine est à l'arrêt. On peut donc très facilement concevoir un montage destiné à démarrer le PC qui serait auto-alimenté par STANDBY.



Connecteur d'alimentation ATX. Les tensions sont repérées par une codification en couleurs.

- Brun (+3.3 V). Ce connecteur REMOTE SENSING est destiné à faciliter la régulation du courant directement par le bloc d'alimentation. Ceci ne présente que peu d'intérêt dans le cas de montages autour du PC.
- Gris. Le connecteur POWER-OK (également appelé « POWER-GOOD ») donne une indication sur l'état de l'alimentation. Dès la connexion du bloc au 230 V, une série de tests (POST, *Power On Self Tests*) est engagée afin de s'assurer de la fourniture du courant dans des marges qualitatives. Si les tests sont passés avec succès, ce connecteur indique à la carte mère que l'alimentation est prête à être utilisée, en fournissant une tension de +5 Volts.

USB

Les spécifications USB parlent d'unités de charge. Une unité correspond à 100 mA. En temps normal, un périphérique connecté en USB ne doit pas consommer plus d'une unité. Ce n'est qu'après négociation que le périphérique détermine s'il peut ou non consommer jusqu'à 5 unités (500 mA). Dans le cas d'une ponction d'alimentation sur le port sans mise en œuvre de circuits spécialisés, il faut donc se limiter, dans la mesure du possible, à une consommation n'excédant pas 100 mA. Au-delà, il n'est pas certain que le port USB puisse fournir suffisamment de courant. Cependant, il est généralement constaté qu'un connecteur hôte peut fournir 5 unités par port tout comme un hub alimenté.

Le brochage d'un port USB est très simple :

- 1 : +5 Volts
- 2 : Data -
- 3 : Data +
- 4 : Masse

Il suffit de réutiliser un connecteur USB type A d'un périphérique hors-service et le tour est joué. Une autre solution, plus coûteuse, consiste à équiper votre montage d'un connecteur type B et à utiliser un câble correspondant très facile à trouver dans le commerce.

Exemple de connecteur USB reconverti en adaptateur fournissant + 5V

PORT PARALLÈLE

Le port parallèle, initialement destiné à la connexion d'imprimantes, n'est pas conçu, à l'origine, pour alimenter les périphériques. Sur ce port les différentes broches utilisent des signaux TTL pour communiquer avec le matériel raccordé. Ce sont les tensions qui déterminent si une broche est à l'état haut (1 logique) ou bas (0 logique) et non le courant. Quelques milliampères sont cependant disponibles sur les broches à l'état haut. C'est habituellement suffisant pour alimenter des LED. Lorsqu'il s'agit d'un montage plus complexe ou nécessitant davantage de courant, une alimentation externe devient indispensable. N'espérez donc pas alimenter un montage, un écran LCD ou un périphérique avec le courant disponible sur une broche à l'état haut.

PORT SÉRIE

De moins en moins présent sur les PC récents, le port série utilise des tensions atypiques. Il ne s'agit absolument pas ici de signaux TTL. Les tensions utilisées sont de l'ordre de +12 volts et –12 volts. Si l'on regarde du coté des souris série, on remarque que les spécifications précisent des consommations très basses. C'est parfaitement

NOTE

TTL : Acronyme de Transistor-Transistor Logic, TTF est un ensemble de spécifications électriques précisant les valeurs de tension pour un 1 et un 0 logiques. En entrée, une interface TTL considère comme 1 une tension de 2.0 V ou supérieure et comme 0 une tension de 0.8 V ou inférieure. L'interface elle-même, lorsqu'elle envoie des signaux, considèrera 3.3 V comme minimum pour signifier un 1 logique et 0.35 V comme maximum pour signifier un 0. En dehors de ces spécifications, le bon fonctionnement d'une interface TTL n'est pas garanti. Dans tous les cas, gardez simplement à l'esprit les valeurs nominales et claires : 1 = 5 V et 0 = 0 V.

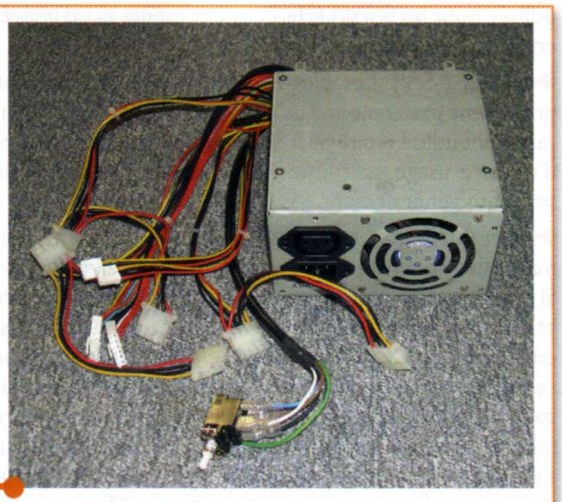
Pour en savoir plus sur les spécifications TTL, CMOS ou ETL, référez-vous à l'excellente page de Leroy Davis : http://www.interfacebus.com/voltage_threshold.html.

NOTE

Dernière précision concernant ce port, beaucoup parlent encore de « connecteur Centronic ». Cette appellation est obsolète tout comme, en principe, les termes « port parallèle ». Il convient de parler de bus d'IEEE 1284 pour désigner les ports de ce type se trouvant à l'arrière de nos PC. Par abus de langage, « port parallèle » est généralement utilisé et nous ferons de même dans le présent hors-série.

compréhensible, puisque les déplacements du périphérique sont détectés à l'aide de simples diodes infrarouges. Il est possible, via l'utilisation d'un régulateur (78L05, par exemple) d'obtenir une tension inférieure (+5 volts) avec un ampérage un peu plus important en utilisant, par exemple, la broche DTR ou RTS.

On pourra ainsi obtenir une source d'alimentation pour un petit périphérique. C'est le cas pour le récepteur



Les alimentations PC AT font des sources intéressantes de courant décliné en toute une gamme de tensions

infrarouge série du projet LIRC, par exemple. La page web de Tomi Engdahl (http://www.tkk.fi/Misc/Electronics/circuits/rspower.html) montre une manière de procéder. On remarque également que le courant ainsi obtenu ne dépasse pas 7 mA pour une tension de 5 volts, ce qui n'est pas suffisant pour une LED classique (20 mA contre 2 mA pour une LED basse consommation). Le fait que ce port soit en voie de disparition et le peu de courant qu'il peut fournir en font, tout comme le port parallèle, un bien mauvais candidat comme source d'alimentation.

CLAVIER

Dans cette catégorie, je place aussi bien les ports DIN 5 des configurations les plus anciennes que les connecteurs PS/2 qui les ont remplacés. Tout comme sur les connecteurs USB, on trouve sur les connecteurs clavier/souris une broche destinée à alimenter le périphérique. Les ports PS/2 destinés à la souris et au clavier sont électriquement équivalents et possèdent un brochage identique. Lorsqu'une différenciation est faite, c'est l'œuvre de la logique intégrée à la carte mère. J'ai eu le loisir de rencontrer quelques BIOS qui refusaient la connexion d'une souris sur le port clavier et inversement.

Un port PS/2 se compose de 6 broches sur un connecteur dit « miniDIN » :

- 1 : DATA. C'est ici que transitent les signaux depuis et vers le périphérique.
- 2 : non connecté
- 3 : Masse
- 4 : +5 Volts
- 5 : Horloge. Cette broche permet de « cadencer » l'envoi et la réception des données sur et depuis le périphérique. A chaque « top » d'horloge, un bit d'information est transmis.
- 6 : non connecté.

Un périphérique (souris ou clavier) ne doit pas, en principe, demander plus de quelques centaines de milliampères à la machine. Les ports PS/2 ne sont pas *hotplug* et sont protégés des surcharges par un fusible. Sur certains modèles de carte mère, ce fusible n'est pas réamorçable et sa destruction rend la carte définitivement inutilisable. Sur la plupart des modèles récents, il s'agit d'un poly-fusible. En cas de surcharge, celui-ci se déconnectera immédiatement comme un fusible classique. Mais, après avoir été déconnecté pendant un certain temps, le poly-fusible retournera à son état initial. Ce type de solutions est moins destructeur qu'un fusible classique (on les retrouve également dans les blocs d'alimentation ATX). Sur les vieilles machines de type Pentium et inférieur, le connecteur clavier est au format DIN 5 et utilise le brochage suivant :

- 1 : Horloge
- 2 : DATA
- 3 : non connecté
- 4 : Masse
- 5 : +5 volts

Les 5 volts présents sur ce type de ports peuvent être bien utiles d'autant que le courant disponible est relativement suffisant pour un petit montage. Nous sommes cependant très loin des valeurs affichées par un connecteur d'alimentation interne (Molex). D'autre part, le risque d'avoir une carte mère équipée d'un fusible standard a de quoi calmer les ardeurs des plus téméraires. On ne choisira donc d'utiliser ce type de sources qu'en dernier recours.

ALIMENTATION EXTERNE

Comme nous venons de le constater, bien qu'il existe une grande variété de sources de courant dans un ordinateur, peu sont directement utilisables. Seules les sources d'alimentations prévues à cet effet s'avèrent dignes d'intérêt (USB et Molex). La solution pour un montage externe nécessitant plusieurs centaines de milliampères est donc l'alimentation par un bloc spécifique. Ceci n'est souvent pas un problème. On trouve par dizaines des adaptateurs de courant de toutes sortes aussi bien neufs que d'occasion. Modem, imprimante, disque externe, webcam... sont autant de périphériques qui sont livrés avec un « bloc secteur » fournissant souvent plus de 500 mA avec des tensions allant de 4 à 30 volts. Il faudra le plus souvent utiliser un régulateur et des condensateurs afin d'obtenir une source de courant stable et propre dans la tension souhaitée.

Le bloc secteur de récupération reste la source la plus facile d'utilisation et la plus économique. La tension et l'intensité du courant fourni sont généralement directement spécifiés

Enfin, il est important de mentionner que LES MASSES DOIVENT ETRE RELIEES ! Cela peut sembler évident pour un utilisateur initié, mais il reste important de le répéter. Un bloc d'alimentation (en courant continu) fournit deux fils, + et -. La différence de potentiel entre ces deux fils est la tension indiquée sur le bloc.

On relie le - (0 volt) à la masse du montage et le + au connecteur d'alimentation. Le montage devra également voir sa masse connectée à celle du PC (ou Mac) et ce quel que soit le port de connexion (USB, parallèle, série, etc.). Ainsi un +5 volts sera équivalent, quelle que soit la référence (ordinateur, montage ou bloc d'alimentation), puisque la masse est la même pour l'ensemble.

Denis Bodor :: db@ed-diamond.com :: lefinnois@lefinnois.net

RÉALISATION DE CIRCUITS

Lorsqu'on se lance dans l'électronique pour y faire ses premiers pas, la question de la réalisation de circuits est un point critique. Cet article se propose de faire un tour d'horizon des différentes techniques existantes.

Il est vrai que lorsqu'il est question de mettre en œuvre moins d'une dizaine de composants, le bricolage reste une solution plus ou moins viable. A grand recours de fils et de soudures à la va-vite, on arrive ainsi à obtenir des montages fonctionnels, mais techniquement brouillons. A un moment ou un autre, il devient nécessaire de passer à l'étape suivante et d'investir en matériel et en rigueur. J'ai tenté ici de regrouper les techniques de réalisation de circuits dans un ordre précis. L'accent sera, bien entendu, mis sur la technique la plus utilisée et la plus « propre ». C'est également celle nécessitant le plus de matériels aussi bien en termes de matériels que de consommables.

LA PLATINE À ESSAIS

Se présentant sous la forme d'un support plastique de plus ou moins grande taille, une platine à essais est une solution d'expérimentation également appelée « boîte de montage rapide à connexions sans soudure ». La platine se compose d'un support permettant d'enficher les composants (contacts) et d'établir des connexions. A l'intérieur de la platine, certaines connexions sont déjà établies, permettant de minimiser l'utilisation de fils de connexions. La figure 1 montre une platine de faible dimension avec les connexions internes en surimpression.

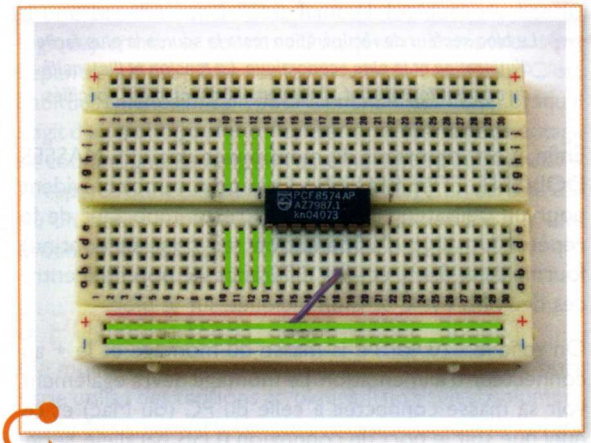


Fig. 1

La figure 2, une platine de grande dimension, permet des réalisations et des tests plus poussés de circuits. Du fait des connexions existantes dans la platine, il convient de placer les composants judicieusement en évitant les court-circuits. Ce type de matériel est très limitatif. Les circuits intégrés (mémoire, circuit logique, microcontrôleur, etc.) doivent être placés sur la partie centrale afin de pouvoir relier leurs pattes aux autres composants.

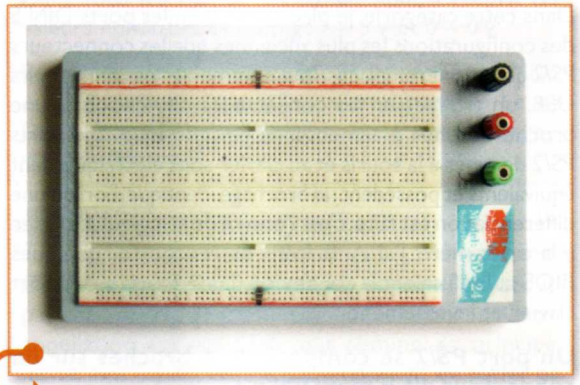


Fig. 2

Le prix des platines à essais dépend de leur taille ou, plus exactement, du nombre de contacts. Une petite platine de 840 contacts (168 x 55 mm) vous coûtera moins de 10 euros. Une platine plus complexe de 2420 contacts (237 x 175 mm) vaut environ 30 euros.

Afin d'établir les connexions, il est possible d'utiliser du fil préalablement coupé et dénudé sur les extrémités. Une solution plus élégante consiste à utiliser des connecteurs tout fait (ponts) qu'il est possible d'acquérir à faible coût (moins de 7 euros pour une boîte de 140 ponts de diverses tailles).

Il n'est pas possible raisonnablement de créer des montages en leur version définitive à l'aide de telles platines. Cependant disposer de ce matériel est toujours intéressant pour les phases de conception pratique et de tests.

LES PLAQUES PASTILLÉES

A mi-chemin entre les platines à essais et les véritables circuits imprimés, nous avons les plaques pastillées (parfois également appelées « plaques à essais »). Il s'agit de plaques d'époxy ou de bakélite recouvertes d'éléments en cuivre sur l'une des faces. Les plaques sont percées d'une matrice de trous permettant de placer les composants et de faire les soudures côté cuivre.

Deux sortes de plaques sont disponibles, celles recouvertes de pastilles (figure 3) et d'autres couvertes de bandes de cuivre (figure 4). Cette dernière catégorie permet des montages suivant la même logique que celle des platines

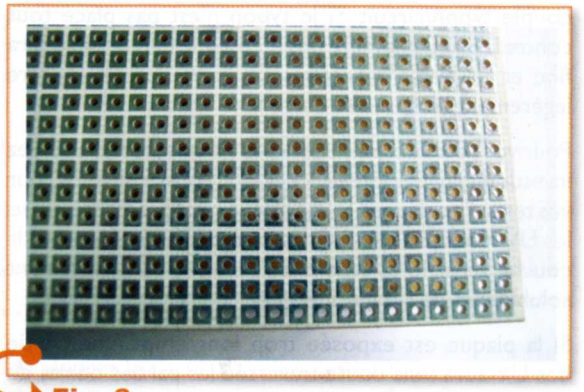


Fig. 3

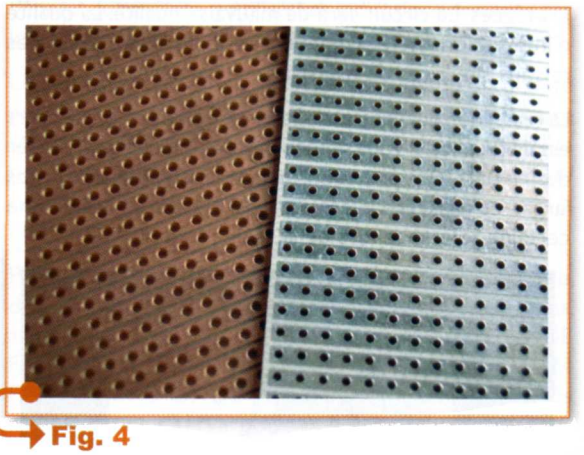


Fig. 4

à essais. On placera alors les composants de manière à profiter au mieux des connexions existantes et on coupera les connexions superflues à l'aide d'une mini-perceuse de type Dremel équipée d'une fraise.

Les plaques pastillées offrent plus de libertés. Les connexions peuvent se faire du côté composant à l'aide de fils (ponts) ou de liaisons côté soudure. Avec un peu d'habitude, on arrive ainsi à réaliser des montages propres. Les soudures sont cependant délicates, puisque les pistes côté cuivre dépendent entièrement de la précision du soudeur et de son habileté à ne pas faire de « pâtés ».

Les circuits obtenus peuvent être acceptables et viables, mais absolument pas professionnels. Si vous compter investir du temps dans le domaine électronique, cette solution deviendra vite inadaptée. Il faudra alors vous tourner vers la réalisation de vrais circuits imprimés, comme expliqué ci-après.

Notons toutefois que certaines de ces plaques peuvent parfois être un gain de temps. Je pense en particulier aux plaques d'étude mélangeant circuits/pistes spécialisées (connecteurs de cartes à puces) et pastilles.

Les plaques pastillées existent dans différents formats. Les plus courants sont 100 x 200 mm et 100 x 160 mm et se déclinent en versions « bandes » ou « pastilles ». Afin de faciliter la soudure, les éléments de cuivre peuvent être étamés.

On notera cependant qu'une norme récente nommée RoHS (Restriction sur l'usage de certaines substances dangereuses) provoque la disparition des plaques étamées. En ce qui concerne le prix, une plaque standard vous en coûtera aux alentours de 5 euros pièce.

Ce type de plaques peut être découpé à la taille du circuit à l'aide d'une scie à métaux ou d'un disque de découpe monté sur une mini-perceuse. Les plaques d'époxy sont plus faciles à travailler que celles en bakélite. Attention toutefois à la fine poussière générée par la découpe qu'il ne faut pas inhaler.

WRAPPING

Voici une technique un peu passée de mode, mais qui compte encore quelques adeptes. Il s'agit de placer les composants sur une plaque de matière plastique percée et d'établir les connexions à l'aide de fil fin. Afin de faciliter la manœuvre, on utilise un outil spécifique ayant la forme d'un stylo permettant de dérouler le fil conducteur.

On enroule tout simplement le fil de connexion autour des pattes des composants et on établit les connexions. Aucune soudure n'est nécessaire. Les faux contacts sont évités par l'utilisation de morceau de papier adhésif.

RÉALISATION DE « VRAIS » CIRCUITS

La réalisation de circuits imprimés (PCB en anglais) est une succession d'étapes. Chacune d'entre elles nécessite une grande précision, car elles ont toutes une influence directe sur le résultat final. Il est fort probable que vos premiers circuits ne seront pas utilisables, mais il ne faudra pas vous décourager. Une fois le « coup de main » pris, vous obtiendrez à coup sûr des résultats parfaits et vous pourrez vous attaquer à des circuits plus complexes.

INSOLATION

Le consommable de base pour la réalisation de circuits est la plaque présensibilisée. Il s'agit d'une plaque d'époxy ou de bakélite couverte d'une couche de cuivre et d'un film sensible à la lumière ultraviolette. Afin de protéger ces derniers, les plaques sont vendues couvertes d'un film plastique noir autocollant opaque aux UV.

Le principe est le suivant : une fois le film de protection retiré, une exposition prolongée à la lumière ultraviolette provoque une réaction dans le film sensible. Celui-ci devient alors soluble dans le révélateur. Pour obtenir un tracé de pistes, il suffit donc de n'exposer que la partie du circuit correspondant au cuivre qui doit disparaître à la gravure.

Pour créer un masque (pochoir) appelé « typon », plusieurs techniques existent :

- Les films spéciaux. Sans doute la solution la plus simple. Il s'agit d'acquérir du film transparent spécial

auprès d'un détaillant de fournitures électroniques. Il suffit ensuite d'imprimer directement le tracé du circuit à l'aide d'une imprimante laser ou à jet d'encre. Attention, certains films plastiques ne doivent pas être utilisés dans une imprimante laser. Ils peuvent fondre et endommager définitivement l'imprimante.

- Le papier calque. Très économique, car disponible partout, le papier calque permet de réaliser des typons utilisables. On notera cependant que des résultats corrects sont plus difficiles à obtenir qu'avec du papier spécialement conçu pour cette utilisation. Le papier calque standard a une fâcheuse tendance à diffuser la lumière. Les circuits sont donc souvent moins précis.

- Le papier photo. C'est une technique assez couramment utilisée par les électroniciens amateurs. Il s'agit d'imprimer le circuit sur du papier standard ou photo (couché) puis d'utiliser un produit aérosol permettant de le rendre perméable aux ultraviolets (produit type transpage). Certains sites recommandent également l'huile de cuisine afin d'obtenir des résultats similaires à bas prix.

- Bandes et pastilles autocollantes. C'est là une technique en voie de disparition consistant à utiliser des petits éléments autocollants à placer sur une feuille transparente en mylar. Inutile ici de recourir à un logiciel de conception et de routage de circuits, tout le travail est fait manuellement.

Le transfert du typon vers la partie sensible de la plaque se fera par une exposition aux ultraviolets. Pour ce faire, on utilise une insoleuse. Vendu dans le commerce en kit ou assemblé, les prix de ce type de matériel est compris entre 75 et 400 euros.

On trouve également du matériel de base permettant de construire son insoleuse pour quelques 30 euros. A ce prix, on dispose de deux tubes fluorescents UV avec les supports, un ballaste de 40 W et deux starters. Le ballaste permet d'alimenter les tubes et les starters de les allumer à l'instar des luminaires utilisant des tubes fluorescents classiques.

Un vieux scanner hors-service, une valise en plastique ou une simple caisse en bois peuvent, pour quelques dizaines d'euros, devenir des insoleuses tout à fait acceptables. Le but de cet article n'est pas de vous apprendre à fabriquer une insoleuse, mais simplement de vous présenter le principe de fabrication des circuits imprimés. Vous pourrez trouver bon nombre d'informations et de conseils sur le web, si vous souhaitez fabriquer la vôtre.

Le temps d'exposition aux UV, l'insolation, est variable. Il dépend d'une part de l'insoleuse (nombre de tubes, distance entre le circuit et les tubes, puissance, etc.) et d'autre part du typon (calques, film plastique, papier photo).

L'insolation elle-même n'est pas difficile, on dispose le typon sur la vitre au-dessus des tubes, on retire le film de protection du circuit, puis on dispose un poids sur l'ensemble pour bien plaquer le typon au circuit. Certaines insoleuses ont un couvercle permettant de presser le couple typon/circuit. Si le typon n'est pas placé tout contre la plaque, l'ombre projetée sur le film sensible sera flou et les pistes imprécises. La plaque insolée montre légèrement l'empreinte du circuit.

Pour vos premiers pas, il n'y a pas de miracle, vous devrez investir quelques morceaux de plaque présensibilisés pour vos tests d'insolations. Si une plaque n'est pas assez insolée, les UV n'ont pas eu le temps de modifier la chimie de la couche sensible et celle-ci ne sera pas aussi facilement soluble lors de l'étape suivante.

Si la plaque est exposée trop longtemps, une partie des UV aura sans doute traversée les parties noires du typon et les pistes, lors de la gravure, ne seront pas assez protégées. Le circuit sera de mauvaise qualité. La qualité d'impression est également très importante pour des raisons similaires de transparence partielle.

La réalisation du typon peut être influencée par les premiers essais. En effet, si vous rencontrez des difficultés à réaliser des circuits avec précision, utilisez des typons aux pistes larges et espacées. Ceci vous permettra d'obtenir une certaine tolérance lors de la réalisation.

Fig. 5

II RÉVÉLATION

Le révélateur est un produit souvent vendu en poudre qu'il faut diluer avec de l'eau (figure 5). Il s'agit d'un produit hautement corrosif à manipuler avec la plus grande prudence (figure 6). Les fabricants de produits révélateurs recommandent l'utilisation de la solution sous quelques heures mais, d'expérience, le produit reste actif très longtemps, une fois dilué, même s'il est sans doute moins efficace.

Révéler un circuit insolé est, pour peu que l'on prenne les précautions d'usage, un jeu d'enfant. Il suffit, en effet,

Fig. 6

de plonger le circuit dans la solution à une température de 20° ou plus durant une minute. Bien entendu, cela dépend du révélateur utilisé. Les indications données par le fabricant prennent le pas sur le présent article.

Une question qu'on se pose souvent lorsqu'on débute porte sur la manière de porter le produit à la température souhaitée. 20° n'est pas très élevé, mais il s'agit d'un minimum. D'autre part, cette température dépend du révélateur utilisé. La solution la plus simple consiste à utiliser un bain-marie.

Placez un récipient contenant le révélateur dans un autre, de plus grande taille, contenant de l'eau chaude. Précisons, au cas où l'idée vous aurait traversé l'esprit, qu'IL NE FAUT PAS METTRE CE TYPE DE PRODUIT DANS UN FOUR MICRO-ONDE ! De plus, les ustensiles utilisés ne devront plus servir pour un usage culinaire, même si vous êtes un demi-dieu de la vaisselle :)

Lorsque vous placez le circuit dans le révélateur (attention aux doigts !), vous devez percevoir immédiatement une réaction. Le révélateur dissout une partie du film sensible. Vous pouvez légèrement bouger le circuit dans le bain ou faire faire un petit mouvement au récipient. Ne touchez pas le circuit, vous risquez d'endommager le film et de faire des trous dans les pistes.

Une fois le temps recommandé par le fabricant écoulé, sortez avec délicatesse le circuit du bain et rincez-le. Ne faites pas couler d'eau directement sur le circuit, le film est fragilisé par le révélateur. Plongez plutôt le circuit dans de l'eau en l'agitant un peu. Laissez sécher quelques instant, mais n'essuyez pas le circuit avec un tissu, un mouchoir ou de l'essuie-tout.

La figure 7 montre une plaque révélée. On distingue clairement le circuit. Toute la partie du film sensible exposée aux UV a été dissoute par le bain dans le révélateur. Le circuit est prêt pour la gravure.

Il s'agit ici d'un tout petit circuit, les pistes ont une épaisseur de moins d'un millimètre. On remarquera d'ailleurs de légères imperfections dues à une imprimante trop vieille (Canon BJ-200) et un typon sur papier spécial, mais économique (type calque).

Certains sites conseillent l'utilisation de lessive de soude diluée en guise de révélateur. D'autres précisent que des

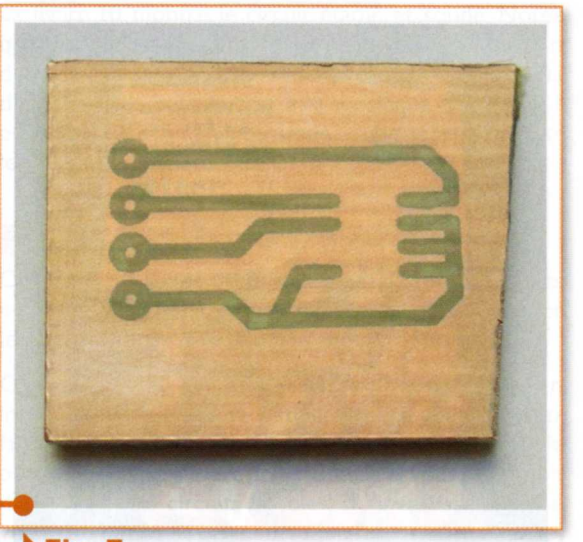


Fig. 7

produits ménagers destinés à déboucher les éviers font également l'affaire une fois dilués.

Suivez ces recommandations à vos risques et périls en n'oubliant pas que ces produits ne sont pas destinés à une telle utilisation. Pour vos premiers pas, je vous conseille d'utiliser les produits vendus par les détaillants de matériels électroniques. Vous pourrez ensuite vous tourner vers des solutions alternatives plus économiques, si vous le souhaitez.

GRAVURE

La plaque est maintenant prête à devenir un vrai circuit imprimé. Une partie du cuivre de la plaque est à nu. Celle-ci doit disparaître. On utilise pour cela une réaction chimique, le perchlorure de fer, qui ronge le cuivre.

Ce produit est vendu en granulés solubles à l'eau (figure 8) ou en bidon de produit liquide plus actif, mais plus cher et devant respecter des conditions de livraison spécifiques (transporteur). La solution de perchlorure de fer s'utilise à une température de 50° environ (suivre les indications du fabricant).

Pour graver les circuits, on utilise normalement du matériel spécifique vendu sous la désignation de machine à graver. Il s'agit d'un conteneur en plastique couplé à un système de chauffage et un autre permettant d'agiter la solution durant la réaction chimique. Il s'agit souvent de diffuseurs à bulles similaires à ceux des aquariums.

Les machines à graver sont vendues à des prix allant de 50 euros à plus de 1000 euros. Les versions les plus économiques sont très basiques, mais remplissent leur office. Prenez garde lors de l'achat du matériel, certaines machines à graver coûteuses utilisent de la mousse ou du persulfate de sodium et non du perchlorure de fer en solution.

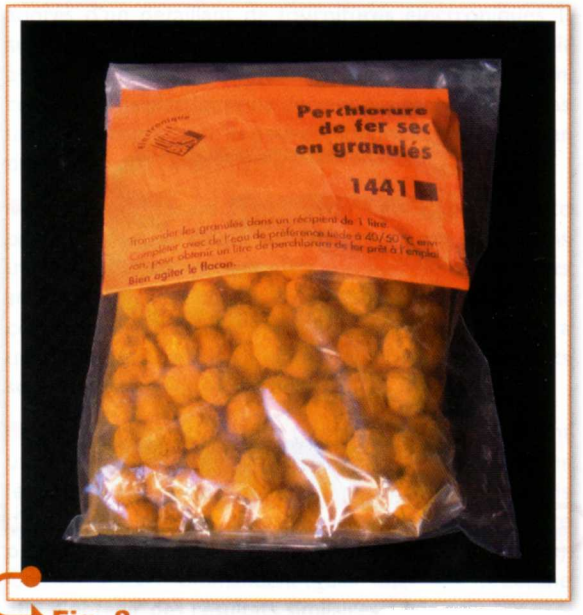


Fig. 8

Une technique plus économique consiste à fabriquer sa machine à graver soi-même. Un bac en plastique, un système permettant d'agiter la solution et une résistance de chauffage thermostatée et vous aurez de quoi graver vos circuits.

Une technique plus économique encore consiste à utiliser le système de bain-marie expliqué plus haut. Le perchlorure de fer porté à la bonne température de cette manière est parfaitement utilisable.

Il vous suffira alors d'agiter manuellement le circuit dans la solution. Un morceau de fil rigide attaché au circuit fera l'affaire, il suffira de « touiller » le liquide.

Cela reste une technique vraiment de bricolage, mais cela fonctionne parfaitement si vous ne souhaitez pas dépenser beaucoup d'argent et que vous avez du temps à perdre. Dernière précision : PAS DE PERCHLORURE DANS LE MICRO-ONDE !

Le temps de gravure du circuit dépend de la concentration de perchlorure, de la température et de l'épaisseur de la couche de cuivre. Dès les premières secondes, le cuivre prend une coloration brune orangé et devient terne. Il faut surveiller la gravure avec attention, et ce, même avec une machine à graver haut de gamme.

Le cuivre doit être entièrement rongé, mais les pistes doivent rester nettes et précises. Si vous laissez agir trop peu de temps, du cuivre restera entre les pistes les plus serrées. Si vous attendez trop longtemps, le perchlorure commencera à ronger le cuivre en bordure des pistes en passant sous le film sensible aux UV. On obtient alors trous, coupures des pistes et un magnifique effet de dentelles...

Une fois la gravure terminée (figure 9), rincez le circuit à l'eau. Ici, pas de précaution particulière, le cuivre ne risque pas de partir sous l'action de l'eau ou du frottement. Il ne reste plus qu'à retirer le film sensible qui protégeait le cuivre jusqu'à présent. Pour cela, utilisez simplement un tissu ou un mouchoir en papier, imbibé d'un solvant (acétone, *white spirit* ou dissolvant pour vernis à ongle). Votre circuit est prêt pour la suite.

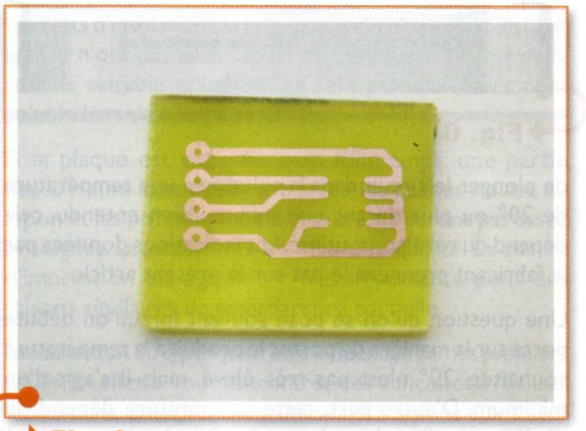


Fig. 9

Le perchlorure de fer est un produit corrosif. Bien qu'il n'irrite pas vraiment la peau, il reste à manipuler avec prudence et vous devrez penser à protéger vos yeux d'éventuelles projections. Le problème majeur du perchlorure de fer reste les tâches et son action sur les métaux. Prenez vos dispositions pour le manipuler en toute sécurité.

La nature corrosive du perchlorure de fer diminue à force d'utilisation. Il devient saturé et finit par devenir inactif sur les circuits. Évitez donc de mélanger une solution utilisée avec votre stock de produit « frais ». Lorsque votre bain de perchlorure est saturé, ne le jetez pas !

Il s'agit d'un produit très polluant. Stockez-le dans un récipient judicieusement étiqueté et apportez-le dans un centre de retraitement ou une déchetterie, où il pourra être pris en charge.

Il ne s'agit pas seulement d'écologie, se débarrasser du perchlorure de fer en le versant dans votre évier finira par provoquer la corrosion des canalisations. Si vous avez un doute, plongez tout simplement un clou, ou tout autre élément, quelques jours dans la solution et vous aurez une idée de l'effet.

II ÉTAMAGE, PERÇAGE ET SOUDURE

Le circuit est maintenant terminé. Il ne reste plus que les petites retouches d'usage. La première consiste à retailler le circuit à la scie à métaux ou avec un disque monté sur une mini-perceuse. Le perçage des connecteurs est une étape qui demande de la précision.

Il serait dommage après toutes ces étapes de tout gâcher en laissant filer le foret de la perceuse. Vous aurez besoin ici d'une perceuse de qualité permettant de monter dans les tours afin de percer vite et bien.

Investir dans un modèle de marque, comme Dremel ou Maxicraft, n'est souvent pas inutile. Ce type de matériel est suffisamment polyvalent pour satisfaire tout bricoleur.

Pour faciliter l'étape de soudure des composants, il est possible d'étamer le circuit. Comprenez par là qu'on applique une couche d'étain sur le circuit de cuivre. Pour ce faire, on distingue deux écoles. La première consiste tout simplement à utiliser le fer à souder et couvrir d'étain le circuit.

On retirera ensuite le surplus d'étain à l'aide d'une tresse à dessouder. Il s'agit de fil de cuivre tressé qu'on applique sur le circuit chauffé. L'étain fond et est absorbé dans la tresse par capillarité ne laissant qu'une fine couche.

L'autre technique est l'étamage à froid. On plonge le circuit dans un produit chimique permettant de déposer une fine couche d'étain sur le cuivre.

Ce type de produit coûte un peu plus de 10 euros pour un bidon d'un demi-litre. Il s'agit d'un produit extrêmement toxique, nocif, irritant et corrosif composé, entre autres, de cyanure de sodium et de nitrate d'argent. Il convient donc de le manipuler avec la plus grande prudence et de le stocker dans un lieu sûr.

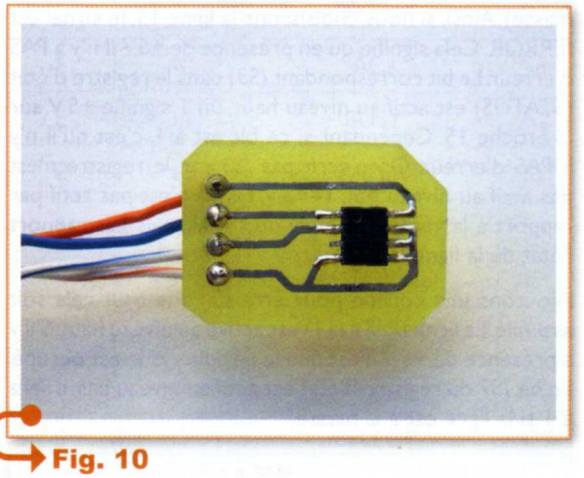


Fig. 10

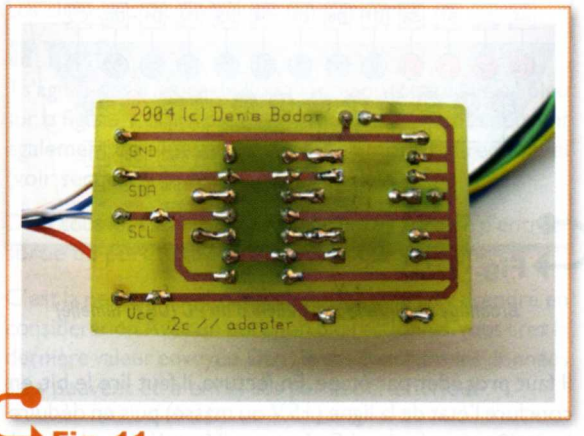


Fig. 11

La figure 10 montre le circuit étamé et soudé. Notez qu'ici le composant est soudé côté circuit en raison de son format. Encore une fois, précisons qu'il s'agit là d'un circuit de petit format très spécifique. La figure 11 présente un circuit plus classique et plus accessible pour les premières expériences.

CONCLUSION

Comme vous venez de le voir, la réalisation de circuits imprimés n'est pas un travail difficile. Il suffit de trouver les bons réglages et les bonnes temporisations.

Certes, réaliser ainsi de vrais circuits nécessite un investissement qui ne se justifiera que si vous comptez faire un certain nombre de réalisations. L'utilisation de plaques pastillées s'avère peu coûteuse et vous permettra de réaliser des circuits simples, si l'investissement ne vous tente pas.

Ajoutons que cet article ne présente que la technique classique de réalisation de circuits. Il en existe d'autres comme le transfert direct du papier vers la plaque. Il s'agit d'un papier spécifique sur lequel on imprime avec une imprimante laser.

Le toner déposé par l'imprimante est ensuite plaqué contre le cuivre et chauffé au fer à repasser. Le toner fond et colle sur le cuivre. On baigne ensuite le tout dans l'eau chaude et le papier se dissout, ne laissant sur le cuivre que le toner. On passe ensuite à l'étape de gravure au perchlorure de fer.

Dans le registre des solutions alternatives, nous avons également l'utilisation d'un mélange d'eau oxygénée et d'acide chlorhydrique. Ceci permet une gravure ultra-rapide, mais les produits sont très dangereux.

Enfin, pour les plus économes, nous avons une source d'énergie qui plane au-dessus de nos têtes tous les jours : le soleil. Il est possible ainsi, théoriquement, d'insoler une plaque en la plaçant sous les rayons du soleil.

Ces mêmes rayons qui serviront ensuite à chauffer le perchlorure de fer pour la gravure. Je n'ai pas testé ces solutions personnellement, mais certains sites web la présentent comme fonctionnelle, bien que totalement aléatoire.

Dans tous les cas, quitte à me répéter, vos premiers circuits n'auront sans doute pas le niveau de qualité souhaité (bien que je vous souhaite tout le contraire). Vous allez gâcher des plaques sensibles, des produits chimiques, du temps et des efforts dans vos premières tentatives. Ceci est parfaitement normal, voire pédagogique. Ne jetez pas l'éponge dès le départ.

De plus, en cas de changement, il est également probable que des « ratés » soient à prévoir. Support du typon, plaque sensible, révélateur, insoleuse... sont autant de sources de problèmes en cas de changement et vous devrez alors revoir vos réglages.

Denis Bodor :: db@ed-diamond.com :: lefinnois@lefinnois.net

PROGRAMMATION DU PORT PARALLÈLE

Un port est présent sur la quasi-totalité des PC, et ce, quel que soit son âge. Le port parallèle, initialement créé pour connecter une imprimante, est aussi une des interfaces les plus faciles d'accès. Ceci, autant d'un point de vue matériel que logiciel.

Le port parallèle n'est pas une nouveauté. Déjà présent en 1981 sur l'IBM PC, celui-ci était prévu pour piloter les toutes nouvelles imprimantes matricielles et compenser la lenteur du port série de l'époque. Comme son nom l'indique, le port parallèle permet la transmission de 8 bits de données en une seule fois de manière parallèle sur huit lignes spécifiques.

En 1991, avec la montée en puissance des ordinateurs et des configurations, le port parallèle IBM commença à montrer ses limites. Une alliance nommée NPA (*Network Printing Alliance*) se forma, composée de grands fabricants comme Lexmark, IBM, Texas Instruments, etc. Ce groupe travailla à la définition d'un cahier des charges permettant un contrôle parfait des imprimantes et des travaux d'impression. Il s'avéra rapidement qu'un port parallèle bidirectionnel devenait indispensable. La NPA proposa les résultats de ses travaux à l'IEEE (*Institute of Electrical and Electronic Engineer*). En 1994, ce dernier publia le standard 1284 « *Standard Signaling Method for a Bi-directional Parallel Peripheral Interface for Personal Computers* ».

LE PORT

Un port parallèle IEEE 1284 se compose de 17 lignes (broches) de signaux et de 8 lignes connectées à la masse pour former les 25 broches d'un connecteur DB25 femelle à l'arrière d'un PC. A chaque ligne correspond un bit dans un registre donné. Le premier registre est accessible à l'adresse d'entrée/sortie propre à chaque port parallèle. Un certain nombre d'adresses sont standardisées comme 0x378 pour le premier port, mais des cartes additionnelles peuvent utiliser des adresses totalement différentes. Le premier registre est celui contrôlant les lignes de donnée, le deuxième, 8 bits plus loin concerne les lignes d'état et enfin, le troisième, les lignes de contrôle.

Il faut clairement distinguer les signaux des registres. C'est une source de confusion importante lorsqu'on travaille avec ce port. Certains signaux sont actifs au niveau haut, ils sont vrais lorsque le signal est au +5 V. D'autres sont actifs au niveau bas, vrais lorsqu'ils sont à la masse. De la même manière, les bits des registres peuvent être à 1 logique en présence de +5 V et d'autres à 1 logique si la ligne est à la masse.

Pour faire la distinction entre des signaux de l'un ou l'autre type, on utilise une barre placée au-dessus du non du signal ou un slash (barre de fraction). Ainsi, /ERROR est un signal actif au niveau bas et BUSY est actif au niveau haut. De manière identique, les registres peuvent être actifs au niveau haut comme S5 (registre d'état, bit 5) ou actif au niveau bas comme /S7 (si le bit est à 1, c'est que la ligne est à la masse).

Là où cela devient épineux, c'est dans le fait que cette notion d'activation au niveau haut ou bas est relative à un état (vrai/faux) par rapport à une tension (+5 V/masse). Ainsi, si nous choisissons la ligne 15, le signal est /ERROR. Cela signifie qu'en présence de +5V il n'y a PAS d'erreur. Le bit correspondant (S3) dans le registre d'état (STATUS) est actif au niveau haut, un 1 signifie +5 V sur la broche 15. Cependant si ce bit est à 1, c'est qu'il n'y a PAS d'erreur. On n'écrit pas /S3 car le registre n'est pas actif au niveau bas, 1=+5 V. Le bit n'est pas actif par rapport à la véracité (l'état) du signal, mais par rapport l'état de la ligne (tension).

Ajoutons une couche pour être sûr que tout cela soit assimilé. La ligne BUSY (11) est active au niveau haut. S'il y a présence de +5 V c'est que le périphérique est occupé. Le bit /S7 du registre d'état est actif au niveau bas, il sera à 1 si la ligne est à la masse.

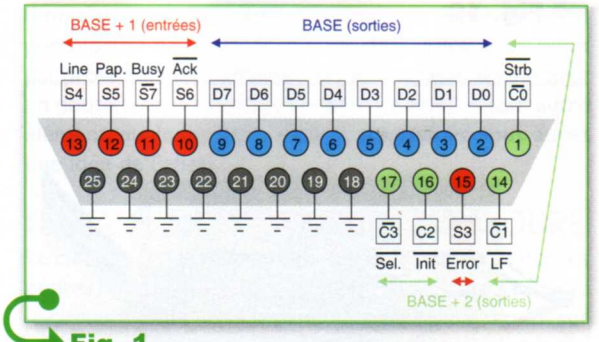


Fig. 1

Brochage de l'interface parallèle d'un PC (DB25 femelle)

Il faut procéder par étape. En lecture, il faut lire le bit, en conclure l'état de la ligne (+5 V ou masse) puis en déduire la signification du signal. En écriture, il faut décider du signal,

déduire l'état à appliquer à la ligne et choisir la valeur du bit dans le registre. Notre travail ici est simplifié puisque nous ne cherchons pas à développer un périphérique de type imprimante. La signification des signaux n'importe pas vraiment, nous voulons simplement utiliser les entrées et les sorties.

SIGAUX DU PORT PARALLÈLE

Sens	Broche	Signal	Bit
S	1	/STROBE	/C0
S	2	DATA 0	D0
S	3	DATA 1	D1
S	4	DATA 2	D2
S	5	DATA 3	D3
S	6	DATA 4	D4
S	7	DATA 5	D5
S	8	DATA 6	D6
S	9	DATA 7	D7
E	10	/ACK	S6
E	11	BUSY	/S7
E	12	PAPER	S5
E	13	ONLINE	S4
S	14	/AUTOFEED	/C1
E	15	/ERROR	S3
S	16	/INIT	C2
S	17	/SELECT	/C3

Les signaux sont divisés en trois groupes.

DATA

Il s'agit là des 8 lignes de données du port parallèle en bleu sur la figure 1. Normalement en sorties, ces lignes peuvent également être lues si le port est en mode bidirectionnel (voir section "Modes du port parallèle").

Dans tous les cas, il est possible de lire à l'adresse d'entrée/sortie du port (0x378 par défaut pour le premier).

C'est la pertinence des informations qu'il faut prendre en considération. Avec un port non bidirectionnel, vous lirez la dernière valeur envoyée. Dans le cas contraire, les données lues peuvent être envoyées par le périphérique.

STATUS

Accessibles à l'adresse du port plus 1, ces lignes sont en lecture seule et permettent d'obtenir des informations de la part du périphérique :

- /ACK (bit 6 – broche 10) : le périphérique met cette ligne à l'état bas brièvement (environ 5 µs) pour indiquer à l'ordinateur qu'il a bien reçu le caractère transmis précédemment. C'est un signal d'accusé de réception.
- BUSY (bit 7 – broche 11) : actif au niveau haut cette ligne indique un état spécifique du périphèrique, celui-ci ne peut plus recevoir de données et les traiter. L'ordinateur doit attendre que le périphérique soit à nouveau disponible.
- PE (bit 5 – broche 12) : « *paper error* », « *paper out* » ou « *paper empty* » selon les documentations. Cette ligne indique que l'imprimante n'a plus de papier.
- ON LINE ou SELECT (bit 4 – broche 13) : cette ligne indique à l'ordinateur que le périphérique est en ligne (« *on line* »)
- /ERROR (bit 3 – broche 15) : indique à l'ordinateur que le périphérique est en erreur. Ce signal est actif au niveau bas et le registre au niveau haut. En clair, si à la lecture vous avez un 1 logique, c'est qu'il n'y a PAS d'erreur.

CONTROL

En vert sur le schéma en figure 1, les lignes de contrôle sont au nombre de 4 et permettent le pilotage de l'interface et la négociation avec l'imprimante. Il s'agit de sorties :

- /STROBE : ce signal indique à l'imprimante que des données sont présentées sur les lignes de données et qu'elle doit les prendre en compte. Ce signal, comme les suivants, est actif au niveau bas (donc préfixé d'un slash). Ceci signifie que lorsqu'on veut signaler à l'imprimante la présence de données, on passe brièvement (moins de 0.5 µS) le signal à la masse avant de revenir au +5 V.
- /AUTOFEED : ce signal demande à l'imprimante de faire automatiquement un saut de ligne. A l'époque des imprimantes matricielles, certaines configurations d'imprimante ne faisaient pas de « retour, à la ligne » (CR/LF), mais un simple retour de la tête d'impression en début de ligne (CR). Il fallait donc logiciellement avancer le papier d'une ligne (LF) de manière automatique si les données ne contenaient qu'un retour chariot.
- /SELECT IN : ce signal permet une sélection d'imprimante. Tout comme avec certains composants que nous verrons plus loin dans ce hors-série, cette ligne active au niveau bas permet de signaler au périphérique qu'il est sélectionné et que les ordres émis le concernent.
- /INIT : via ce signal, l'ordinateur peut demander une réinitialisation de l'imprimante. Là encore le signal est actif au niveau bas, ce qui signifie qu'il faut mettre la ligne à la masse pour réinitialiser l'imprimante.

MODES DU PORT PARALLÈLE

Pour l'heure, nous n'avons parlé que d'un seul mode du port parallèle : SPP pour *Standard Parallel Port*. Dans ce mode également appelé « *Printer Mode* », les lignes de données du port ne peuvent être utilisées qu'en mode unidirectionnel. Le mode SPP peut également, selon l'interface, être utilisé de manière bidirectionnelle. On utilise le bit 5 du registre des lignes CONTROL pour inverser le mode de fonctionnement des huit lignes de données. En plaçant ce bit à 1, on peut lire les lignes DATA comme valeurs d'entrées et non plus comme dernières valeurs envoyées (*latchs*). Si le port n'est pas en mesure de travailler de manière bidirectionnelle, l'écriture du bit 5 n'aura aucun effet.

SPP/EPP1.7/EPP1.9, parfois noté simplement EPP dans l'écran de configuration du BIOS est le mode le plus intéressant pour une machine moderne. EPP, *Enhanced Parallel Port*, est le résultat d'un rapprochement des sociétés Intel et Xircom & Zenith Data Systems. Le mode EPP présente l'intérêt de pouvoir continuer à travailler comme avec le mode SPP. Les registres SPP sont utilisables comme expliqué plus haut (base, base+1 et base+2) en tenant compte du fait que le bit 5 du registre CONTROLE permet d'inverser le sens écriture/lecture pour les lignes de données.

Deux adresses supplémentaires, base+3 et base+4, donnent respectivement accès aux registres des ports d'adresses EPP et de données EPP. Un port parallèle en EPP peut donc être librement utilisé selon les besoins. La différence entre les versions 1.7 et 1.9 du mode EPP réside principalement dans l'absence du bit de *timeout*. Dans tous les cas, on préfèrera le standard le plus récent possible à moins d'avoir une bonne raison de choisir 1.7 (compatibilité avec un périphérique).

Enfin, nous avons le mode ECP pour *Extended Capabilities Port* développé par Hewlett Packard et Microsoft. Ce mode est également compatible SPP et utilise des registres de configuration supplémentaires comme EPP : Base+0x400 pour les données en FIFO en mode FIFO ou configuration. A en mode configuration, Base+0x401 pour le registre de configuration B en mode configuration et Base+0x402 pour le registre ECR (*Extended Control Register*) dans tous les modes. Le mode ECP présente quelques caractéristiques majeures dont l'utilisation d'un canal DMA et un tampon FIFO pour les données.

Cependant, ces éléments ne sont intéressants qu'en cas de débit important (périphérique bloc sur port parallèle) et sont largement compensés par une certaine difficulté d'utilisation. On notera que la grande majorité des BIOS permettent de configurer le port parallèle intégré à la carte mère en mode multiple. Ainsi, vous pouvez choisir ECP/EPP1.9 pour cumuler tous les avantages : port bidirectionnel SPP, registres du mode EPP base+3 et base+4, registre et mode de configuration ECP/ECR.

SUPPORT LINUX

Le noyau Linux prend en charge parfaitement les ports parallèles de tout type et dans tous les modes. Depuis la version 2.4 du noyau, le support de cette interface est devenu très hiérarchisé.

Nous avons d'un coté le support spécifique à chaque plate-forme dont le plus utilisé est `partport_pc`. Au-dessus de ce pilote se place le support `parport` permettant d'unifier les méthodes d'accès au port sous la forme d'une API unique. Nous y reviendrons par la suite, mais la réorganisation du support `parport` du noyau 2.4 a également été l'occasion d'introduire `ppdev`. Tout comme le support `lp` (*line printer*) permet le dialogue avec une imprimante depuis l'espace utilisateur, `ppdev` permet le contrôle de chaque ligne du port.

Notez la nette différente entre `lp` et `ppdev`. Le second offre un accès au port parallèle IEEE 1284 alors que le premier fournit une méthode pour imprimer. `echo "coucou" > /dev/lp0` imprimera la chaîne de caractères sur le périphérique. Rien d'équivalent n'est possible avec `/dev/parport0` avec `ppdev`. Ceci pose problème, non pas avec les ports parallèles intégrés aux cartes mères ou vendus sous forme de cartes additionnelles PCI ou ISA, mais avec les adaptateurs USB.

Ceux-ci sont supportés par le pilote `usblp` (noyau 2.6) ou `printer` (noyau 2.4). Ceux-ci ne fournissent pas une couche de bas niveau pour `parport` comme c'est le cas pour `parport_pc`, `parport_cs` ou `parport_gsc`, mais un support pour `/dev/lp`. De ce fait, l'utilisation de `ppdev` n'est pas possible car les adaptateurs USB concernés sont normalement dédiés aux connexions avec les imprimantes ou directement intégrés à celles-ci (imprimantes USB).

La question de savoir s'il fallait ou non intégrer le support `usblp` comme un élément de `parport` a déjà fait l'objet de discussions sur la LKML et la réponse était toujours la même : les adaptateurs USB/imprimante sont pris en charge comme des canaux pour l'impression, non comme des convertisseurs USB/Parallèle (comme `uss720`) ou un nouveau port parallèle. Dans tous les cas, vous pourrez constater que le support de votre port parallèle est correctement pris en charge par le noyau en inspectant les journaux d'activité (`/var/log/kern.log`) :

```
parport: PnPBIOS parport detected.
parport0: PC-style at 0x378 (0x778), irq 7, dma 3
  [PCSPP,TRISTATE,COMPAT,EPP,ECP,DMA]
```

Le noyau 2.6 est quelque peu plus bavard que le 2.4, puisqu'il informe des différents modes supportés par l'interface, mais aussi des IRQ et DMA utilisés (selon le mode) et des adresses utiles. Vous retrouverez une partie de ces informations dans `/proc` dans les fichiers `dma`, `interrupts` et `ioports`.

ÉTAT DU PORT AU DÉMARRAGE DE LA MACHINE

Aucune documentation officielle ni standard ne définit clairement l'état des lignes du port parallèle au démarrage d'un PC. Pire encore, durant le processus de démarrage entre l'initialisation par le BIOS, le chargement puis l'exécution du noyau et le lancement des différents services, les différentes lignes du port parallèle changent d'état.

Pour cette raison, le périphérique doit réagir ou plutôt ne pas réagir en conséquence. Il était de coutume, fut un temps, de n'allumer l'imprimante qu'après le démarrage complet du système. On retrouve encore ce type de recommandation dans certaines documentations constructeur. J'ai cependant constaté sur bon nombre de machines que la ligne /STROBE restait, de la mise sous tension à l'utilisation par un outil logiciel, à l'état haut (+5 V). Ceci paraît parfaitement logique étant donné la fonction du signal. Toutefois, il ne s'agit là que d'une constatation empirique. Prenez garde, lors de la conception d'un nouveau montage connecté à ce port de ne pas le laisser agir/réagir avant une étape où vous avez entièrement le contrôle. Bien entendu, lorsqu'il s'agit du module LCD ou d'afficheurs lumineux, ceci n'est en rien critique. Il n'en va pas de même avec une carte à relais ou un triac pilotant des équipements importants (éclairage, moteur, etc.).

PROGRAMMATION : LES MAUVAISES SOLUTIONS

L'accès et la programmation de l'interface parallèle sous Linux est un vaste domaine qui pourrait couvrir la totalité de ce magazine. Nous ne traiterons ici que les cas les plus courants. Il existe plusieurs manières de piloter les lignes du port parallèle. Nous allons les voir de la plus mauvaise à la plus convenable.

LA MÉTHODE SAUVAGE

Des lecteurs ayant déjà développé des applications sous MS/DOS sauront qu'il n'est pas difficile de piloter directement le port parallèle sans l'assistance du système. En mode SPP, les adresses du port sont connues. Habituellement 0x378 est l'adresse d'entrée/sortie de base du premier port parallèle (LPT1). Avec les anciennes versions du noyau Linux, accéder directement au port via cette adresse était la seule manière de le contrôler. Aujourd'hui, il s'agit tout simplement de la technique la plus « sale ». On utilisera les fonctions de bas niveau `outb` et `inb` pour écrire et lire directement dans les registres :

```
#include<unistd.h>
#include<stdio.h>
#include <asm/io.h>
#define BASEPORT 0x378
int main(int argc, char **argv)
{
                if(ioperm(BASEPORT,3,1)) {
                perror("ioperm");
                exit(1);
                }
                outb(11,BASEPORT);
                printf("lu : %X\n", inb(BASEPORT+1));
                return(0);
}
```

Je vous ai prévenu, c'est moche. On utilise la fonction `ioperm` pour positionner les autorisations d'entrée/sortie sur les ports à commencer par 0x378 et ce sur 3 octets. L'utilisation de cette fonction nécessite les privilèges super-utilisateur. Le binaire doit donc être exécuté par ce dernier ou être SUID root avec tous les risques que cela comporte. Cette méthode d'accès n'est présentée ici qu'à titre indicatif. Ne l'utilisez pas. Vous court-circuitez tous les mécanismes de protection et de gestion du noyau et risquez de perturber gravement le fonctionnement du système en cas d'erreur. Il existe des méthodes plus élégantes, plus pratiques et surtout plus « propres ».

/DEV/PORT

Sous GNU/Linux, il existe une méthode alternative permettant d'accéder aux ports d'entrée/sortie. Il suffit, en effet, d'utiliser `/dev/port`. Ce pseudo fichier est un *mappage* mémoire des adresses d'E/S. Pour écrire dans le registre en 0x378, il suffit de positionner le pointeur dans le fichier à cette adresse et d'écrire ou lire l'octet. Les problèmes posés sont globalement les mêmes que précédemment à la différence que des permissions spécifiques sur `/dev/port` règleront le problème de l'identité de l'utilisateur exécutant le binaire. On notera que l'utilisation de ce pseudo fichier permet de développement avec un langage comme Python ou Perl :

```
#!/usr/bin/perl -w
use Fcntl;

$BASE=888;

sysopen(PORT, "/dev/port", O_WRONLY)
  or die "Impossible d'ouvrir le fichier : $!\n";
binmode(PORT);

sysseek(PORT, $BASE,0)
  or die "Probleme seek : $!\n";

syswrite(PORT, chr(1) )
  or die "Probleme syswrite : $!\n";

Close PORT
  or die "Probleme close : $!\n";
```

Comme précédemment, ces exemples ne sont qu'indicatifs. Il existe une méthode bien moins dangereuse et applicable aussi bien à du code C que Perl.

PPDEV : PROGRAMMATION PROPRE DU PORT

Voici la méthode « normale » pour une application devant accéder aux lignes du port parallèle. L'accès depuis l'espace utilisateur se fait selon deux techniques en fonction des besoins. Si votre application doit envoyer des données à une imprimante, cela se fera via le protocole « imprimante » et `/dev/lp0`. Pour chaque octet envoyé, les lignes de données sont initialisées, `/STROBE` signale la mise à disposition et l'imprimante accuse réception avec la ligne `/ACK`. Une application de ce type pourrait tenir en un simple `echo coucou > /dev/lp0`. Ceci n'est pas l'objet du présent article. Le pilote `ppdev` permet d'agir avec la même automatisation et sécurité avec davantage de finesse quant au contrôle de chaque ligne du port. Pour ce faire, on utilisera l'appel `ioctl` sur l'entrée correspondant au port dans `/dev`. La syntaxe d'`ioctl` utilise trois arguments : un descripteur de fichier, une commande et, éventuellement, un pointeur vers les données à utiliser. Avant toute opération sur le port via `ppdev`, nous devons ouvrir le pseudo-fichier correspondant en lecture/écriture selon une technique tout à fait classique :

```
#include <sys/ioctl.h>
#include <string.h>
#include <fcntl.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/errno.h>
#include <linux/ppdev.h>
#include <linux/parport.h>

int errno, fd;

if ((fd = open("/dev/parport0", O_RDWR)) < 0) {
  fprintf(stderr,"Open Error : %s (%d)\n",
          strerror(errno),errno);
  exit(EXIT_FAILURE);
}
```

Dès le descripteur de fichier obtenu, il est nécessaire de demander l'accès au port. Celui-ci peut être refusé par le système car une autre application ou un module noyau en a obtenu l'accès exclusif. On a généralement le problème lorsque `lp` est chargé. Il conviendra alors de retirer le module. Notez que l'accès au port n'est pas nécessaire pour toutes les opérations. Certaines commandes ne concernent que l'accès à `ppdev` et non au port lui-même. Ces dernières n'ont pas besoin de réclamer l'accès.

```
if (ioctl(fd, PPCLAIM) < 0) {
  fprintf(stderr,"PPCLAIM ioctl Error : %s (%d)\n",
          strerror(errno),errno);
  exit(EXIT_FAILURE);
}
```

Si cette commande réussit, on peut manipuler les lignes à loisir puis, au terme de la procédure, on rend la main et on referme de fichier :

```
if (ioctl(fd, PPRELEASE) < 0) {
  fprintf(stderr,"PPRELEASE ioctl Error : %s (%d)\n",
          strerror(errno),errno);
  exit(EXIT_FAILURE);
}

if(close(fd) < 0) {
  fprintf(stderr,"Close Error : %s (%d)\n",
          strerror(errno),errno);
  exit(EXIT_FAILURE);
}
```

La commande `PPEXCL` utilisée avant `PPCLAIM` demandera un accès exclusif au port. Tout autre accès sera refusé jusqu'au moment où nous utiliserons `PPRELEASE`. L'accès exclusif ne sera obtenu au moment de l'utilisation de `PPCLAIM`.

ÉCRITURE DES DONNÉES

Une fois l'accès au port obtenu, l'action la plus courante est d'écrire sur les lignes de données. On utilise la commande `PPWDATA` avec, en argument, un pointeur vers les données à écrire :

```
unsigned char valout=0x01;

if (ioctl (fd, PPWDATA, &valout) < 0) {
  fprintf(stderr,"PPWDATA ioctl Error : %s (%d)\n",
          strerror(errno),errno);
  exit(EXIT_FAILURE);
}
```

Le port parallèle n'est pas en mesure de fournir suffisamment de courant pour toutes les applications. Vous pouvez parfaitement connecter une LED sur chaque ligne de données via une résistance de 470 Ohms et contrôler leur allumage via ce type de code. Cela n'ira pas plus loin. Les lignes du port parallèle sont conçues pour véhiculer des signaux et vous pourrez y connecter toutes sortes de circuits logiques. Mais s'il vous faut piloter des composants nécessitant beaucoup de courant, l'utilisation de transistors en saturation ou d'optocoupleurs est indispensable.

LECTURE DES DONNÉES

Cela a été dit plus haut, les lignes de données peuvent fonctionner de manière bidirectionnelle. Techniquement, c'est le bit 5 du registre des lignes CONTROL qui permet de passer les lignes de données en entrée. Avec `ppdev`, il n'est pas permis de définir ce bit depuis votre code. Il faut utiliser la commande `PPDATADIR`. Celle-ci n'influencera que le comportement des commandes `PPWDATA` que nous venons de voir et de `PPRDATA`. Il faut absolument éviter que le périphérique contrôle les lignes alors que le port est en sortie. Il y a un risque majeur d'endommager le port et/ou le périphérique. En effet, si les lignes de données sont en sortie et à 0 (à la masse) et que le périphérique en passe une à 1 (+5 V) un court-circuit se produit et une partie des circuits logiques du port parallèle peut être détruit.

ATTENTION

Tenir compte du fonctionnement interne du port parallèle est ici capital. Lorsque les lignes de données du port sont en entrée, le périphérique peut contrôler l'état des lignes.

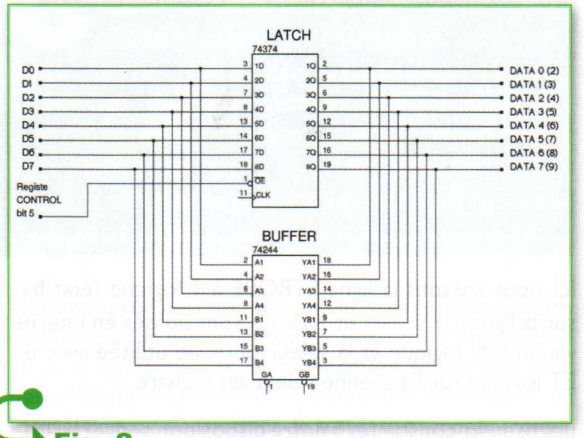


Fig. 2

Exemple théorique du fonctionnement interne du mode bidirectionnel des lignes de données du port parallèle. Le bit 5 du registre CONTROL désactive le latch et permet au périphérique de contrôler les lignes dont nous lisons l'état dans le buffer. Si le périphérique impose un +5 V sur une ligne alors que le latch est activé un risque de court-circuit existe.

Il est important de négocier avec votre périphérique ou votre montage le moment où celui-ci pourra émettre des données et donc contrôler les lignes. Vous pouvez utiliser les lignes CONTROL et STATUS qui sont unidirectionnelles pour cette négociation. La quasi-totalité des ports parallèles disposent, en entrée sur les lignes de données, de résistances de *pull-up* (rappel au +5 V). Il s'agit de résistances qui ramènent la ligne au +5 V de manière à obtenir un 1 logique en l'absence de connexion.

```
unsigned char valin;
int dirin=1;
int dirout=0;

if (ioctl (fd, PPDATADIR, &dirin)  < 0) {
  fprintf(stderr,"PPDATADIR ioctl Error : %s (%d)\n",
          strerror(errno),errno);
  exit(EXIT_FAILURE);
}

if (ioctl (fd, PPRDATA, &valin) < 0) {
  fprintf(stderr,"PPRDATA ioctl Error : %s (%d)\n",
          strerror(errno),errno);
  exit(EXIT_FAILURE);
}
```

Si on applique un signal haut (+5 V), rien ne change et si on met la ligne à la masse, on obtient un 0 logique. Dans tous les cas, l'état d'une ligne n'est donc jamais indéfini. Toutes les documentations consultées sont d'accord sur un point : aucun fabricant ne semble suivre une ligne directrice unique et il devient nécessaire de tester avant toute connexion et configuration. Placez les lignes en entrée et lisez avec PPRDATA. Écrivez une valeur quelconque et relisez. La valeur lue ne doit pas être celle précédemment écrite si les lignes sont effectivement configurées en entrée.

A ce moment, vous pouvez partir du principe que le PC ne contrôle plus les lignes et vous pouvez relier l'une d'elle à la masse avant de lire à nouveau avec PPRDATA. Là, vous devez voir changer le bit correspondant dans le registre. Si votre groupe de ligne ne dispose pas en interne de résistances de pull-up, vous pouvez les ajouter à l'extérieur (voir la partie concernant les lignes STATUS ci-après).

LECTURE DE L'ÉTAT

Pour lire le registre d'état et donc l'état des lignes 10, 11, 12, 13 et 15, il suffit d'utiliser la commande PPRSTATUS :

Plutôt que d'analyser les bits de l'octet nous-mêmes, nous pouvons utiliser les pseudo-constantes définies dans linux/parport.h : PARPORT_STATUS_ERROR, PARPORT_STATUS_SELECT, PARPORT_STATUS_PAPEROUT, PARPORT_STATUS_ACK et PARPORT_STATUS_BUSY.

```
unsigned char status;

if (ioctl (fd, PPRSTATUS, &status) < 0) {
        fprintf(stderr,"PPRSTATUS ioctl Error : %s (%d)\n",
                strerror(errno),errno);
        exit(EXIT_FAILURE);
} else {
        if (status & PARPORT_STATUS_ERROR) {
                printf("Pas d'erreur (1)\n");
        } else {
                printf("Erreur (0)\n");
        }
}
```

Normalement, tout comme avec les lignes de données, des résistances de pull-up internes au port permettent un rappel au +5 V. Cependant, comme il n'est pas garanti que ce fait soit établi sur toutes les machines, on utilisera un montage comme celui présenté en figure 3.

Les résistances de 4.7 K Ohms forcent l'état haut en l'absence de pression sur les boutons poussoirs. Lorsqu'on établit la connexion, on force la ligne à la masse et la résistance limite le courant selon la formule U=R.I, soit 5=4700*I, soit 5/4700=1mA... négligeable. Bien entendu, des résistances plus importantes (10 K Ohms) feront également l'affaire. Le schéma présenté est complètement théorique. Il présente des boutons poussoirs ayant ce qu'il est courant d'appeler « la maladie des rebonds ».

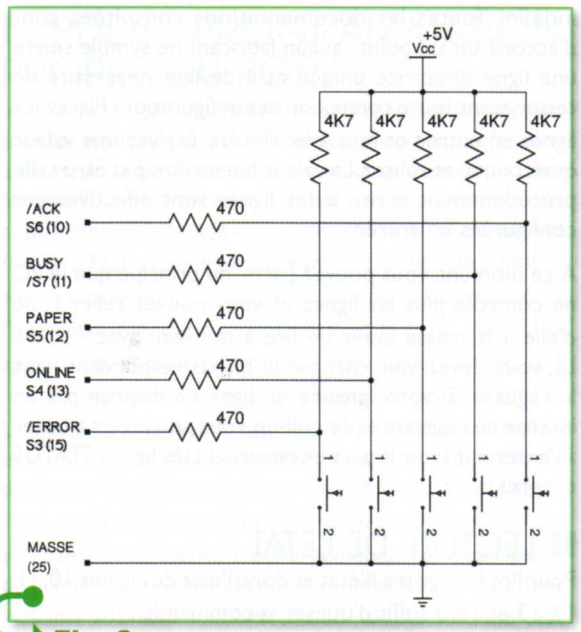


Fig. 3

Utilisation de résistances de pull-up et de protection pour les lignes STATUS.

A l'échelle du PC, le contact du bouton n'est pas franc, mais entrecoupé à cause de raisons mécaniques évidentes. Un autre article présente une solution à ce type de problème.

D'autre part, la liaison avec un circuit logique (TTL/CMOS) ne pose pas de problème et les résistances de pull-up deviennent inutiles.

ÉCRITURE DES LIGNES DE CONTRÔLE

Si les huit lignes de sortie ne sont pas suffisantes pour votre application, vous pouvez également « piocher » dans les quatre lignes du groupe CONTROL.

La commande `PPWCONTROL` permettra l'écriture dans le registre correspondant. Cependant, comme ces lignes sont souvent utilisées individuellement, l'API `ppdev` offre une commande plus intéressante : `PPFCONTROL`.

Celle-ci permet d'éviter de lire le registre avec `PPRCONTROL`, faire le changement dans la valeur obtenue et réécrire le tout. `PPFCONTROL` le fait en un seul mouvement, il suffit de lui passer en argument un pointeur sur une structure de type `ppdev_frob_struct` :

```
struct ppdev_frob_struct {
  unsigned char mask;
  unsigned char val;
}
```

Si nous souhaitons changer l'état de la ligne STROBE, il nous suffit alors de faire ceci :

```
struct ppdev_frob_struct frob;

frob.mask = PARPORT_CONTROL_STROBE;
frob.val = PARPORT_CONTROL_STROBE;

if (ioctl(fd, PPFCONTROL, &frob) < 0) {
  fprintf(stderr,"PPFCONTROL ioctl Error : %s (%d)\n",
          strerror(errno),errno);
  exit(EXIT_FAILURE);
}
```

Ici, nous passons la ligne STROBE à 1 logique (état bas sur la ligne). `frob.mask` et `frob.val` sont utilisés en interne via un OU logique et la valeur obtenue utilisée avec un ET logique sur l'ancienne valeur du registre.

Les pseudo-constantes à notre disposition sont : `PARPORT_CONTROL_STROBE`, `PARPORT_CONTROL_AUTOFD`, `PARPORT_CONTROL_INIT` et `PARPORT_CONTROL_SELECT`.

CONCLUSION

Nous venons de voir le principe de fonctionnement de base du port parallèle et la méthode d'usage pour l'accès depuis un code utilisateur sous GNU/Linux.

Il reste un certain nombre de choses à explorer parmi lesquelles le développement d'un code noyau pilotant un montage sur ce port et la gestion des interruptions dans un code utilisateur.

Ces deux sujets seront traités par ailleurs dans le présent hors-série dans le cadre d'une mise en pratique.

Si vous voulez en apprendre davantage sur le port parallèle, son fonctionnement et sa programmation, référez-vous aux liens ci-dessous.

LIENS

- *Interfacing the Standard Parallel Port* : http://www.beyondlogic.org/spp/parallel.htm
- *The Linux 2.4 Parallel Port Subsystem* : http://people.redhat.com/twaugh/parport/html/parportguide.html
- *Parallel port interfacing made easy* : http://www.epanorama.net/circuits/parallel_output.html

PROGRAMMATION ET INTERFAÇAGE D'UN MICROCONTRÔLEUR PAR USB SOUS LINUX : LE 68HC908JB8

Classiquement, les microcontrôleurs communiquent avec les ordinateurs personnels par le port série (RS232). Or, la tendance actuelle de l'évolution du matériel informatique force aux constats suivants : le port série (RS232) est voué à disparaître, et le port série est trop lent pour certaines applications.

Nous nous sommes par conséquent fixés comme objectifs de maîtriser un microcontrôleur possédant un port USB [1, 2, 3], d'effectuer le développement totalement avec des outils libres sous GNU/Linux et de développer un logiciel de base permettant de tester le microcontrôleur sous ce même système d'exploitation.

LE MICROCONTRÔLEUR FREESCALE 68HC908JB8

Nous avons sélectionné, parmi les rares microcontrôleurs possédant un port USB, le Freescale 68HC908JB8 selon les critères suivants :

- Disponible chez les principaux revendeurs européens de matériel électronique[1] ;
- Basé sur un noyau connu (le 6808) et donc avec des outils de développement *open source* disponibles (assembleur as6808 de l'ensemble d'outils asxxxx[2] et asl) ;
- Et finalement ne nécessitant qu'un nombre minimum de composants passifs pour sa mise en œuvre (mémoire flash ne nécessitant pas de programmeur dédié).

Le 68HC908JB8 possède une mémoire FLASH (8 KB), RAM (256 B), un port USB 1.1, est disponible en packages CMS SOIC28 ou DIL20 et est cadencé par un résonateur à 6 MHz. Cependant, un inconvénient majeur est l'absence de convertisseur analogique/numérique qui limite ses applications en termes d'instrumentation.

Nous avons utilisé un circuit de test (Fig. 1), en partie inspiré de http://elmicro.com/hc08web/usb08/images/usb08_schema.gif, comprenant le strict minimum de composants passifs : une paire de diodes pour utiliser une broche comme port RS232 bidirectionnel, mise en œuvre du port USB et les résistances pour la mise au niveau des signaux de RESET et interruption matérielle IRQ#. Ce montage nous permet déjà d'effectuer quelques tests préliminaires de programmation pour nous familiariser avec le protocole de communication, l'accès à la mémoire flash et les assembleurs disponibles sous Linux.

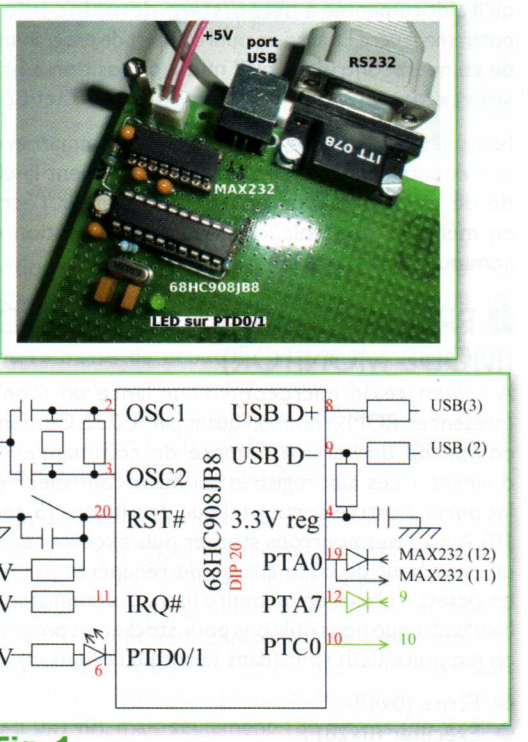


Fig. 1

Gauche : schéma de principe pour faire fonctionner un 68HC908. On y retrouve les éléments fondamentaux que sont le résonateur à 6 MHz, les résistances définissant certains signaux de contrôle (reset RST#, interruption matérielle IRQ#), la gestion de l'USB sur les broches USB D+ et D- (la résistance entre D- et 3,3 V indique à l'hôte auquel se connecte le microcontrôleur que la communication USB est lente). Finalement quelques composants accessoires tels que les diodes connectées à PTA0 pour émuler un port de communication série et une LED sur PTD0/1 qui nous sera utile comme indicateur visuel de l'état du système. Notez que sur la version CMS du composant, PTD0 et PTD1 sont séparés (broches 6 et 7) alors que sur la version DIL utilisée ici ces deux signaux sont connectés en interne (broche 6). Droite : montage prototype sur plaque pré-percée illustrant qu'aucun matériel de réalisation de circuits imprimés spécifique n'est nécessaire pour exploiter ce microcontrôleur.

[1] Références 3480252, 3480264 et 4133020 chez Farnell ; référence 445-6744 chez Radiospares pour environ 6 euros/pièce.

[2] http://shop-pdp.kent.edu/ashtml/asxget.htm ou alternativement installé sous Debian avec le package sdcc – http://sdcc.sourceforge.net/ – qui fournit aussi un compilateur C que nous n'avons pas testé. Dans ce dernier cas, l'assembleur s'appelle as-hc08 au lieu de as6808.

Denis Bodor :: db@ed-diamond.com :: lefinnois@lefinnois.net

I LA PROGRAMMATION DU 68HC908

Nous avons initialement (début 2001) recherché les outils disponibles en open source pour programmer le 68HC908 pour n'identifier alors qu'un outil, spgmr08[3]. Cet outil, très complet, ne nous donnait pas l'opportunité d'appréhender en détail le fonctionnement du microcontrôleur et nous avons décidé d'écrire quelques petits utilitaires simples pour mettre en évidence les méthodes d'initialisation du microcontrôleur : il s'agit dans ce document de cette première partie, qui sera suivie dans un second temps par une exploitation sous GNU/Linux du code fourni en exemple par Freescale [4] pour la communication par bus USB.

Plus récemment, Benoît Demaine s'est intéressé à ce microcontrôleur dans le cadre d'une étude universitaire qu'il a documentée à http://stwp.demaine.info/ et a notamment écrit lui aussi un outil complet de programmation de ce microcontrôleur qu'il met à disposition à http://sourceforge.net/projects/monitor-68hc08/.

Nous allons ici décrire le mode de programmation de ce microcontrôleur afin de maîtriser totalement la chaîne de développement qui va se conclure par l'écriture en mémoire non volatile d'un code de gestion de la communication par USB.

II PREMIERS PAS : LE BOOTLOADER (MODE MONITOR)

À l'allumage, le microcontrôleur lance un moniteur (présent en ROM), communiquant par RS232. Ce moniteur comprend un certain nombre de commandes nous donnant accès aux registres du microcontrôleur (dont les ports d'entrée/sortie), tel que décrit dans [5, section 10]. Ainsi, nous pourrons stocker puis exécuter en RAM un petit bout de code chargé de récupérer par RS232 les octets à placer en mémoire flash. Les commandes du *bootloader* que nous utilisons pour stocker un programme en mémoire flash sont (dans l'ordre d'utilisation) :

- Écrire (0x49) ;
- Exécuter (0x28) ;
- Lire et définir l'adresse du pointeur de programme (0x0C).

Une action préliminaire à l'envoi d'ordres au bootloader est de s'identifier par la transmission d'un code de sécurité [5, p. 175]. Ce code de sécurité est la séquence des octets stockée dans les vecteurs d'interruption situés en 0xFFF6-0xFFFD. Sur un microcontrôleur neuf, ces valeurs sont toutes égales à 0xFF donc il nous suffit d'initialiser la communication en transmettant une séquence de 8 octets égaux à 0xFF. Ultérieurement, lorsque ces vecteurs d'interruption auront été modifiés (ce sera sûrement le cas puisqu'ils comprennent un *timer*, une interruption matérielle et l'interface de communication USB). Il faudra prendre soin d'envoyer la bonne séquence d'octets (les nouvelles adresses des vecteurs d'interruption), faute de quoi le mode monitor ne sera pas activé.

Dans la suite de ce document, les programmes qui seront décrits sont mis à la disposition du lecteur sur notre site web à http://projets.sequanux.org/membres/sim/usb/ et http://jmfriedt.free.fr. La fonction qui initialise la communication avec le bootloader pour lancer le mode monitor se trouve ainsi dans le programme `hc08.c - fonction init_hc08mon()`.

Une façon simple de tester si le matériel fonctionne, sans même passer par une phase de programmation/assemblage en 6808, est de changer la valeur stockée dans le registre associé à un port matériel dont nous visualisons par exemple l'état par une diode électroluminescente (LED). La diode étant connectée au port D, son allumage/extinction est commandé par la valeur stockée dans le registre d'adresse 0x0003 (PTD) après avoir défini ce port comme étant une sortie, en stockant la valeur 0xFF dans le registre d'adresse 0x0007 (DDRD). Faire clignoter une diode depuis le mode moniteur se résume donc à l'envoi des commandes suivantes depuis un programme de communication par le port série du PC :

```
lo=0x07; hi=0; writ_hc08(fd,hi,lo,0xFF);
while(1) {
    lo=0x03; hi=0; writ_hc08(fd,hi,lo,0xFF);
    sleep(1);
    lo=0x03; hi=0; writ_hc08(fd,hi,lo,0x00);
    sleep(1);
}
```

où nous avons implémenté les fonctions de base de communication entre le 68HC908 et le PC – à savoir ici une écriture en mémoire du HC908 `writ_hc08()` – par l'envoi sur le port RS232 (ouvert en lecture/écriture avec un descripteur de fichier `fd`) de la séquence suivante :

- Le PC envoie le code de commande 0x49 (`WRITE`).
- Le PC écoute la réponse du HC908 qui lui confirme que la commande a été comprise en répondant 0x49.
- Le PC envoie l'octet de poids fort de l'adresse du registre à modifier et écoute la réponse du HC908 qui confirme la réception de cette donnée. La séquence est alors la même pour l'octet de poids faible.
- Finalement, l'octet à écrire (la donnée) est transféré par le PC puis confirmé par le HC908.

Ce programme est disponible dans notre archive sous le nom `hc08test.c`.

De la même façon, nous pouvons stocker une valeur en RAM et la relire :

```
lo=0x57; hi=0; read_hc08(fd,hi,lo);
lo=0x57; hi=0; writ_hc08(fd,hi,lo,0xaa);
lo=0x57; hi=0; read_hc08(fd,hi,lo);
```

[3] http://sourceforge.net/projects/spgmr08/

ATTENTION

Le 68HC908 ne possède pas d'UART pour gérer la communication RS232. Une émulation logicielle de ce protocole est effectuée, basée sur une *unique* broche (PTA0) connectée aux broches d'émission et de réception du port série du PC par deux diodes protégeant chaque signal des fluctuations de niveau de l'autre signal (voir Fig. 1). La conséquence est que tout signal émis par le PC (signal de transmission de l'UART) est immédiatement vu par le PC sur sa ligne de réception. L'UART place en mémoire cette donnée avant de recevoir la réponse du 68HC908 acquittant la réception de l'ordre. En conséquent, il faut toujours lire 2 *octets* pour vérifier la bonne transmission d'une donnée du PC vers le 68HC908, le premier octet étant directement transmis de la voie d'émission vers la voie de réception (sans intérêt) et le second étant la réponse du 68HC908 (intéressant). La procédure pour envoyer la commande 0x49 est donc (en supposant que le port série a été ouvert et associé à un descripteur de fichier `fd` et que `buf` est un tableau de caractères :

```
buf[0]=0x49;write(fd,buf,1);read(fd,buf,2);printf("%d
",buf[1]&0xff);
```

Ici la fonction `read_hc08` fonctionne de la même façon que vu précédemment à l'exception du dernier octet – la donnée lue en mémoire du HC908 et transmise au PC – qui ne nécessite une lecture que d'un octet unique puisqu'il ne s'agit pas d'une réponse à un ordre issu du PC.

LA COMMUNICATION RS232 ET LA MÉMOIRE FLASH

Un premier exemple simple (programme en Table 1) consiste à faire clignoter cette même LED connectée au port PTD0/1 au moyen d'un programme exécuté depuis la mémoire volatile (RAM). Ce premier exemple très simple permet de se familiariser avec la syntaxe de l'assembleur 6808 et l'architecture de ce cœur de microprocesseur (un unique accumulateur A sur 8 bits, un index d'adresse H:X qui peut éventuellement fournir un second registre de travail sur 8 bits, un pointeur de pile et un pointeur de position d'exécution du programme [6, p. 24]) puis avec la séquence de compilation et de transfert du programme en RAM.

Ayant assemblé ce programme (`as6808 -o programme.asm`) et extrait les *opcodes* (Table 2) du fichier ainsi généré (`grep T programme.rel | cut -c9-80 > programme.out`) nous sommes prêts à transmettre le programme en RAM (`hc08ram programme.out`) et à l'exécuter. Cette opération nécessite une dernière exploration du mode de fonctionnement du processeur. Le bootloader nous fournit une commande RUN (commande 0x28) dont le fonctionnement n'est pas expliqué dans le manuel du microcontrôleur [5] mais dans le programme `mongp32.c`, fonction `mon_runpc()` de `spgmr08`. Il faut demander au bootloader quelle est la position actuelle du pointeur de pile (*stack pointer* SP), et placer en SP+4 l'octet de poids fort de l'adresse de début de programme et en SP+5 l'octet de poids faible, avant d'appeler la fonction 0x28. Dans le cas d'un programme exécuté depuis la RAM, nous plaçons donc 0x00 en SP+4 et 0x40 en SP+5 lors de l'exécution de la commande 0x0C.

Tab. 1 :

```
start:  ldhx #0x0140        ; TXS : (SP)<-(H:X)-1 => STACK=0x013f
        txs                 ; reset stack pointer
        ; mov  #0x01,CONFIG1 ; disable COP watchdog, CONFIG1=0x001f
        clra                ; clear accumulator
        mov  #0x0f,0x0007   ; DDRD: port D as output

loop:   eor  #0x0f          ; toggle diode
        sta  0x0003         ; store accumulator on PortD
        bsr  delay
        bra  loop

delay:  psha
        pshx
        clrx                ; 256*0,9375ms=240ms

loopx:  clra                ; 9*256=2304 (0,9375ms @ 2,4576MHz)

loopa:  nsa                 ; [3]
        nsa                 ; [3]
        dbnza loopa         ; [3]
        dbnzx loopx         ;
        pulx
        pula
        rts
```

Premier programme permettant de faire clignoter une diode connectée au port PTD0/1 du 68HC908JB8

Enfin, un dernier point important à considérer lors de l'exécution d'un programme depuis la RAM est que le bootloader du 68HC908JB8 définit sa propre pile en 0x00FF. Ainsi, tout programme en RAM qui se trouve à cet emplacement sera écrasé lors de l'exécution de commandes du bootloader : nous n'avons donc pas 256 octets de RAM disponibles pour nos tests depuis la RAM (0x40 à 0x140), mais seulement 186 octets (de 0x40 à 0xFA, si on se laisse un peu de marge). Nous comprenons ainsi la nécessité de placer le programme de gestion de la communication USB (l'exemple Freescale prend 1,8 KB) en mémoire flash.

Tab. 2 :

```
45 01 40 94 4F 6E 0F 07 A8 0F C7 00 03
AD 02 20 F7 87 89 5F 4F 62 62 4B FC 5B
F9 88 86 81
```

Opcodes issus de l'assemblage du programme présenté en Table 1 : il s'agit là de la séquence d'octets prête à être transférée en RAM pour exécution par hc08ram. Il peut être instructif de se familiariser avec le processus d'assemblage en l'effectuant à la main [6, pp. 79-87] sur cet exemple court.

Ceci nous indique donc la procédure à suivre pour mettre en pratique la procédure de programmation de la mémoire flash : nous devrons dans un premier temps remplir la RAM du code permettant de programmer la mémoire flash (en effet nous ne disposons que de 256 octets de RAM – insuffisant pour contenir tout le code de gestion de la communication USB – et il nous faut absolument accéder aux 8 KB de mémoire flash pour utiliser efficacement ce microcontrôleur), puis exécuter ce code afin de lire le programme plus volumineux sur le port série et le stocker en flash. Nous avons mentionné auparavant que l'absence d'UART rend la lecture sur le port série plus complexe que sur la majorité des microcontrôleurs : il nous faut donc émuler un tel périphérique de façon logicielle dans le programme de réception des octets à stocker en mémoire flash, tel que présenté dans le programme de la Table 3.

ATTENTION

La RAM du 68HC908 commence en 0x40, les 64 premiers octets étant réservés pour les registres de gestion du matériel. Tout programme destiné à être exécuté depuis la RAM devra donc être compilé pour commencer à cette adresse. De même, étant donné que nous avons 256 octets de RAM, la pile est placée à la fin de cette RAM (placer une valeur sur la pile *décrémente* le pointeur de pile), donc nous chargeons la pile en 0x140 par la séquence `ldhx #0x0140` et `txs` qui commencera tout nouveau programme (syntaxe `asxxxx`).

Une fois équipé de cette procédure de communication RS232 (remplacer ce code 3 chargé de l'écriture par une lecture est une opération simple), nous pouvons nous atteler au problème de la récupération des octets sur le port série tels que transmis par le PC et les stocker en mémoire flash. La mémoire flash commence en 0xDC00 donc :

- Tout programme destiné à être exécuté depuis la flash doit être recompilé pour commencer à cette adresse (redéfinition des sauts en adresses absolues JMP).
- Nous commencerons à écrire à cet emplacement les octets reçus du PC lors du remplissage de la flash.

Plutôt qu'utiliser les procédures fournies en ROM par le 68HC908 pour accéder à la flash (lecture/écriture), relativement contraignantes en termes de gestion de la RAM partagée avec notre programme, nous avons décidé d'implémenter quelques routines de base pour écrire en flash tel que décrit dans [5, section 4]. L'ensemble de ces routines, fastidieuses à décrire ici, sont disponibles dans l'archive mise à disposition sur **http://projets.sequanux.org/membres/sim/usb/** sous les noms `flash_write.asm` pour recevoir un programme du PC et le stocker en flash, `flash_read.asm` pour lire le contenu de la flash et transmettre au PC et finalement `flash_erase.asm` pour effacer le contenu de la flash, opération nécessaire avant toute nouvelle écriture. Ainsi la procédure pour stocker un programme en mémoire flash avec nos outils est :

- `hc08flash flash_erase.out` pour effacer la flash.
- `hc08flash flash_write.out programme.out` pour écrire en flash la séquence d'opcodes `programme.out`.
- `hc08flash` (sans argument) pour exécuter le contenu de la flash.

Dans le cas particulier de l'utilisation des interruptions, nous avons écrit un programme additionnel qui est volontairement conservé distinct des autres procédures de gestion de la flash du fait du danger associé à son utilisation. En effet, les vecteurs d'interruption se trouvent en mémoire non volatile dans la plage 0xFFF0-0xFFFF, donc dans une zone disjointe de la flash. Lors de l'initialisation (mise sous tension) du 68HC908, le bootloader teste le contenu du vecteur d'interruption de Reset en 0xFFFE-0xFFFF et ne s'exécute que si ces deux octets sont à 0xFFFF. Ainsi, avec notre montage électronique qui ne permet pas d'appliquer une tension de 8 V sur la broche IRQ# (condition de désactivation de ce test initial), le 68HC908 devient inutilisable si une valeur est écrite dans ces deux registres. Nous avons donc écrit un petit programme autonome, `flash_irqs.asm`, qui prend en entrée une liste de 16 valeurs qui sont les 16 octets définissant les vecteurs d'interruption, mais ne stocke en mémoire que les 14 premières valeurs afin de s'assurer de ne pas modifier le vecteur de reset. Ce programme s'utilise donc à la fin de la séquence de programmation vue plus haut par hc08flash `flash_irqs.out programme.irqs` avec `programme.irqs` contenant les 16 octets à placer dans les vecteurs d'interruption.

LA COMMUNICATION USB SUR LE 68HC908

Un circuit de développement est vendu par la société MCT Elektronikladen (**http://hc08web.de/usb08/**) et un exemple d'application communiquant par le port USB est distribué gratuitement sur leur site web. C'est de cette application que nous allons partir pour comprendre le fonctionnement du bus USB et développer un driver GNU/Linux communiquant avec le microcontrôleur. Nous nous affranchissons ainsi d'une description détaillée du programme exécuté sur le microcontrôleur en utilisant un code prêt à l'emploi. Ce programme, écrit en C, est proposé sous forme de code source et fichier S19 après compilation par divers compilateurs commerciaux sous Windows. Nos tentatives de porter ce code à sdcc n'ont pas été fructueuses : nous nous contenterons donc d'exploiter les fichiers compilés (S19) et n'utiliserons le code C que pour comprendre le sens du programme après désassemblage – ce code désassemblé est mis à disposition dans notre archive[4] et pourra servir de base aux applications spécifiques dont chaque lecteur pourra avoir besoin. Alternativement, une application totalement

[4] http://projets.sequanux.org/membres/sim/usb/

open source basée sur le 68HC908JB8 est USB-IR-Boy tel que présenté à http://usbirboy.sourceforge.net/ sur lequel nous ne nous attarderons pas ici puisqu'il est le sujet d'un autre article dans ce même numéro.

Un point fondamental que nous préciserons ultérieurement est que le bus USB est très sensible aux délais. Or il apparaît que le code C compilé avec le compilateur commercial ICC (http://www.imagecraft.com/ sous Windows) tel que proposé par Motorola ne respecte pas ces conditions et ne fonctionne par correctement avec les drivers ohci et usb-uhci de GNU/Linux. Il faut donc prendre soin de flasher le code compilé avec le compilateur C Cosmic (http://www.cosmic-software.com/ sous Windows) tel que fourni par Motorola. Nous convertissons donc le fichier au format S19 fourni dans l'archive http://hc08web.de/usb08/files/usb08_fwv102_cosmic.zip pour générer la suite d'opcodes transférée en mémoire flash selon la méthode présentée plus haut.

Notre approche a donc été la suivante : partir du code d'exemple fourni par MCT Elektronikladen, vérifier son bon fonctionnement, puis désassembler ce code afin de pouvoir modifier l'action associée à la transmission de trames USB sans toucher au gestionnaire de communication lui-même, pour finalement re-assembler le code résultant. Le passage d'une suite d'opcodes (le fichier .S19 fourni en exemple) au programme assembleur étant une opération bijective, nous évitons tout problème de version de compilateur de langage évolué (ici le C) en travaillant systématiquement en assembleur : le programme assemblé après nos modifications aura la même latence dans la communication USB que le code original (tant que nous ne touchons pas aux routines de gestion des interruptions associées à la communication USB). Ainsi, sans même savoir ce qu'est un driver et sans maîtriser le protocole USB, nous pouvons vérifier si le client USB fonctionne après programmation du microcontrôleur et déjà valider l'aspect matériel. Avec le code de http://www.elektronikladen.de/en_usb08.html qui occupe 30% de la mémoire flash, le client USB est reconnu sous GNU/Linux par le message suivant :

```
$ tail -f /var/log/syslog
cheyenne1 kernel: Manufacturer: MCT Elektronikladen
cheyenne1 kernel: Product: USB08 Evaluation Board
cheyenne1 kernel: usb.c: unhandled interfaces on device
cheyenne1 kernel: usb.c: USB device 31 (vend/prod 0xc70/0x0)
is
not claimed by any active driver.
Nous identifions là deux champs qui nous seront utiles : les
identificateurs vendeur (0x0c70) et produit (0x0000).
```

COUCHE MATÉRIELLE D'USB

Un bus USB porte l'alimentation sous 5 V permettant d'alimenter le microcontrôleur, ainsi qu'un signal asynchrone bidirectionnel sous une tension de 3,3 V.

Tab. 3 :

```
start:  ldhx    #0x0140     ; TXS : (SP)<-(H:X)-1 => STACK=0x013f
        txs                 ; reset stack pointer
        ; mov   #0x01,CONFIG1 ; disable COP watchdog, CONFIG1=0x001f

        mov     #0x00,0x0003 ; PTD0/1 lo => LED lit
        mov     #0x03,0x0007 ; DDRD: PTD0/1 as output

        mov     #0x01,0x0000 ; PTA: PTA0 hi
        mov     #0x01,0x0004 ; DDRA: PTA0 as output
        ldx     #0h00

loop:   incx                ; increment counter
        txa
        bsr send            ; send value of counter to serial port
        bra loop

send:   pshx
        ldx     #0x08       ;   snd through PTA0 the content of Acc (@9600)
        mov     #0x00,0x0000 ;  PTA: PTA0 lo : START bit

looprs: bsr     delay       ; X
        bsr     delay       ; X
        rora                ; 1 rotate right Acc through carry
        bcc     bit0        ; 3 branch if carry is clear (is A&1=0)
        mov     #0x01,0x0000 ; 4 PTA0=hi
        bra     bit1        ; 3

bit0:   mov     #0x00,0x0000 ; 4 PTA0=lo

bit1:   dbnzx   looprs      ; 3   --> sum=11 or 14

fin:    bsr     delay
        bsr     delay
        mov     #0x01,0x0000 ; PTA: PTA0 hi : STOP bit
        bsr     delay
        bsr     delay
        pulx
        rts

delay:  pshx                ; 2+4 for bsr   (12+2*((X*9)+15))*.3333=833
        ldx     #0h0f       ; 3                                  104

loopx:  nsa                 ; 3    => Xinit=0x88 for 1200
        nsa                 ; 3           =0x0f for 9600
        dbnzx   loopx       ; 3
        pulx                ; 2
        rts                 ; 4
```

Exemple de programme transmettant à la vitesse de 9600 bauds les valeurs d'un compteur sur le port série. La vitesse de transmission est définie par la valeur du délai dans la fonction delay qui est utilisée pour définir de façon logicielle la durée de chaque bit transmis afin d'être compatible avec le protocole asynchrone : une valeur de 0x0F dans le registre X permet une transmission à 9600 bauds tandis qu'il faut utiliser une valeur de 0x88 pour une transmission à 1200 bauds.

Une fois le client USB connecté, une adresse lui est dynamiquement assignée et servira aux transactions ultérieures : dans l'exemple précédent, l'identifiant 31 nous avait été assigné.

Les données transitant entre le contrôleur USB et les clients sont encapsulées dans des paquets dont le destinataire est défini par l'identifiant vu précédemment,

tandis qu'un certain nombre d'*endpoints* (voir section 6.3) jouent le même rôle que les ports sur un réseau TCP/IP ou UDP sur Ethernet. Nous n'aurons pas à nous préoccuper de la structure détaillée des paquets puisque le microcontrôleur gère leur assemblage de façon transparente pour l'utilisateur.

De façon générale, la communication sur bus USB se fait par paquets de 8 (lent), 16, 32 ou 64 octets (rapide) [7, p. 41]. Un certain nombre de modes de communication ont été définis pour partager la bande passante du bus USB en fonction des besoins (et donc des applications) des divers périphériques USB. Nous les citons ici pour comparer les capacités du 68HC908JB8 (mode Interrupt) aux autres modes disponibles et ainsi mieux cerner son champ d'applications [1] :

- **Interrupt** (débit constant, réessaie jusqu`à succès, paquets <64 B) ;
- Isochronous (débit constant, latence bornée, pas de corrections) ;
- Bulk (prend toute la bande passante, garantit le transfert mais pas la latence).

LES DRIVERS ET LA COMMUNICATION AVEC LE MATÉRIEL SOUS LINUX

GÉNÉRALITÉS SUR LES DRIVERS

Nous allons présenter dans ce document les points importants de la réalisation d'un pilote USB pour les noyaux 2.4.x du système d'exploitation GNU/Linux [1, 8]. Dans notre archive, un port de ce module est également disponible pour les versions 2.6 du noyau. Même si ce pilote est dédié à la communication avec le microcontrôleur 68HC908JB8, il se veut aussi générique que possible. Cela signifie qu'il doit pouvoir être facilement adapté à un autre microcontrôleur implémentant l'USB en mode interrupt. Nous ne nous attarderons pas ici sur la définition d'un driver et l'intérêt de programmer un module pour la communication avec les périphériques (portabilité, simplicité de compilation, fournir des méthodes standards à l'utilisateur) [9]. Nous supposons connu le fait que tous les pseudo-fichiers interfaçant les processus utilisateurs avec un driver se trouvent dans le répertoire dédié `/dev` et sont définis par deux identificateurs, le nombre majeur (classe de périphérique) et le nombre mineur (indice dans cette classe de périphériques utilisant tous le même pilote). Un driver doit fournir un certain nombre de méthodes pour permettre aux programmes utilisateurs d'accéder à un périphérique matériel : `write()`, `read()` et `ioctl()`. La première permet d'écrire des données sur le périphérique, la seconde d'y lire des données et la troisième de le configurer. Deux autres méthodes standards d'initialisation sont `open()` et `close()`. Il nous faut tout d'abord définir la structure de donnée présente dans tout module qui contient les pointeurs vers les fonctions permettant d'interagir avec le système de fichiers.

```
static struct file_operations hc08_fops = {
        owner:          THIS_MODULE,
        read:           hc08_read,
        write:          hc08_write,
        ioctl:          hc08_ioctl,
        open:           hc08_open,
        release:        hc08_release,
};
```

Cette structure va permettre d'enregistrer auprès du noyau notre périphérique de type caractère (*char device* – nous communiquerons avec notre microcontrôleur octet par octet et non par blocs) :

```
if ((hc08_major = register_chrdev (hc08_major, "hc08", &hc08_fops)) < 0)
```

CAS PARTICULIER DES DRIVERS USB

Une des particularités d'un pilote USB est qu'il n'interagit pas directement avec son matériel. Comme on peut le remarquer dans le schéma de la figure 2, drivers et périphériques sont séparés par des couches modulaires. Au niveau le plus bas, directement en contact avec le périphérique, on trouve une couche représentant le pilote du contrôleur USB maître (*Host Controller*). Le choix de ce pilote (`usb-uhci` ou `usb-ohci`) va être déterminé par le type de contrôleur. Pour un contrôleur UHCI (*Universal Host Controller Interface d'Intel*), le module usb-uhci sera utilisé tandis que pour un contrôleur OHCI (*Open Host Controller Interface de Compaq*, Microsoft et National Semiconductor), le module `usb-ohci` sera préféré.

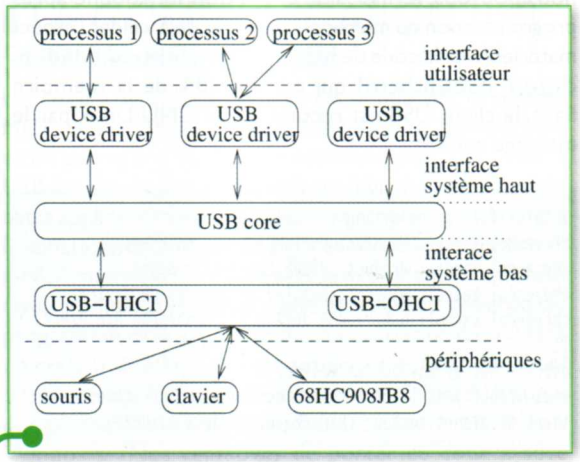


Fig. 2

Implémentation par couches de l'USB sous GNU/Linux [1].

Entre le Host Controller et les drivers USB, on trouve le module `usbcore`. Son rôle est de fournir aux pilotes de périphériques une interface standard indépendante

du type de contrôleur utilisé. Il va également se charger d'arbitrer la transmission de données sur le bus commun à tous les périphériques USB. Ce module usbcore va faciliter le développement des pilotes USB en mettant à leur disposition un certain nombre de méthodes. La communication avec le périphérique (initialisation, transfert de données,...) passera donc par des fonctions exportées par le module usbcore.

Pour pouvoir bénéficier de toutes ces ressources, notre module va devoir s'enregistrer auprès des couches précédemment décrites. Pour cela, une structure de type struct usb_driver est renseignée puis enregistrée auprès de usbcore à l'aide de la fonction usb_register.

```
static struct usb_driver hc08_driver = {
        name:           "hc08",
        probe:          hc08_probe,
        disconnect:     hc08_disconnect,
        fops:           &hc08_fops,
        minor:          USB_HC08_MINOR_BASE,
        id_table:       hc08_table,
};

result = usb_register(&hc08_driver);
```

Mises à part les méthodes probe et disconnect, dont nous expliquerons le rôle plus tard (section 6.3), on peut remarquer la présence d'un tableau de type id_table. Il contient les identifiants vendeur et produit correspondant à notre périphérique, le microcontrôleur 68HC908JB8.

```
#define USB_HC08_VENDOR_ID      0x0c70
#define USB_HC08_PRODUCT_ID     0x0000

static struct usb_device_id hc08_table [] = {
        { USB_DEVICE(USB_HC08_VENDOR_ID, USB_HC08_PRODUCT_ID)
},
        { }
};
```

Fournir cette table au gestionnaire de périphériques USB lui permet de créer une liste reliant les identifiants de périphériques avec les drivers associés. À chaque fois qu'un matériel est connecté au bus USB, le noyau va essayer de déterminer s'il dispose d'un pilote susceptible de le prendre en charge. Pour cela, il va comparer l'identifiant du périphérique nouvellement connecté à la liste des identifiants déclarés par les pilotes USB. Si une correspondance est établie avec un pilote, alors la méthode probe de ce dernier est appelée.

II LA MÉTHODE PROBE

La méthode probe d'un pilote USB va tenter de déterminer avec précision si le périphérique arrivant dépend bien du pilote qu'elle représente. En effet, une authentification reposant uniquement sur les identifiants vendeur et produit est insuffisante. Notre travail va donc être de vérifier que le périphérique proposé par le gestionnaire USB est bien le microcontrôleur 68HC908JB8. Si à l'issue de cette série de tests la fonction probe retourne un pointeur non nul, alors le noyau attribuera la gestion du périphérique à notre pilote. C'est grâce à cette méthode probe que Linux va être capable de gérer le branchement à chaud d'un matériel USB. Chaque périphérique connecté au bus USB est décrit par un objet ou une hiérarchie de structures (Fig. 3). La structure struct usb_device est le point d'entrée de cette arborescence qui contient toutes les informations relatives au périphérique USB. Lorsqu'elle est appelée, la méthode probe reçoit une telle structure en paramètre d'entrée. Cette dernière a été préalablement initialisée par le module usbcore. Nous allons donc parcourir et examiner cette arborescence afin d'établir ou non une correspondance avec le microcontrôleur 68HC908JB8.

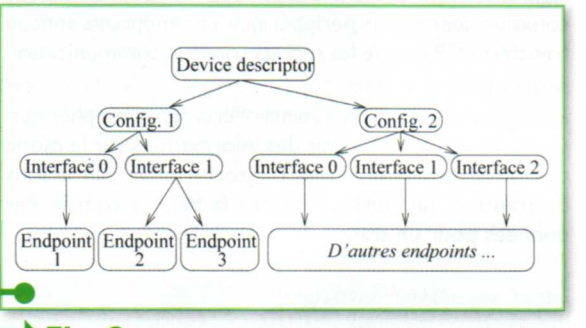


Fig. 3

Représentation sous forme d'arbre d'un périphérique USB sous GNU/Linux.

Device descriptor :

Ce descripteur se présente sous la forme d'une structure struct usb_device_descriptor. Il regroupe toutes les informations d'ordre global sur le périphérique. On y trouvera entre autres les identifiants vendeur et produit. L'examen de ces deux champs est fondamental pour un pilote dans sa décision de s'attribuer ou non la gestion d'un périphérique. Cette structure contient également le nombre de configurations possibles pour le périphérique.

Configuration descriptor :

Chaque configuration du périphérique est décrite dans une structure struct usb_config_descriptor. Ce système de configuration sert à séparer les différents modes de fonctionnement du périphérique. Par exemple, dans le cas d'une imprimante-scanner, on peut imaginer une configuration pour le mode imprimante et une autre pour le mode scanner. Dans le cas particulier du 68HC908JB8, une seule configuration est définie.

```
struct usb_config_descriptor {
 [...]
        __u8  bNumInterfaces    __attribute__ ((packed));
        struct usb_interface *interface;
 [...]
```

Interface descriptor :

Au sein d'une même configuration, un périphérique peut disposer de plusieurs interfaces. On peut comparer une interface à un réglage particulier du périphérique dans un mode de fonctionnement. Par exemple, dans le cas d'une caméra vidéo, on peut imaginer une interface pour chacun de ses modes d'acquisition. Encore une fois, le 68HC908JB8, en raison de sa simplicité ne définit qu'une seule interface.

```
struct usb_interface_descriptor {
   __u8  bNumEndpoints     __attribute__ ((packed));
   struct usb_endpoint_descriptor *endpoint;
   [...]
```

Endpoint descriptor :

Ces objets permettent de représenter, de manière logicielle, les ports d'entrées et de sorties utilisables pour communiquer avec le périphérique. Les endpoints sont au transfert USB ce que les *sockets* sont à la communication réseau. Une structure `struct usb_endpoint_descriptor` est associée à chacun des endpoints offerts par le périphérique. Cette structure va fournir des informations sur le mode de transfert (bulk, interrupt ou isochronous), sur le sens du transfert (in, out) ou encore la taille maximale des données pour un transfert...

```
struct usb_endpoint_descriptor {
  __u8  bEndpointAddress  __attribute__ ((packed));
  __u8  bmAttributes      __attribute__ ((packed));
  __u16 wMaxPacketSize    __attribute__ ((packed));
  __u8  bInterval         __attribute__ ((packed));
  [...]
```

Dans le cas du 68HC908JB8, les endpoints peuvent être définis comme des points d'accès présentant une correspondance directe avec un ensemble de registres du côté du microcontrôleur.

Le 68HC908JB8 fournit 3 endpoints :

- Un port de contrôle (endpoint 0) : bidirectionnel, *buffer* 8 B I/O.
- L'endpoint 0 est toujours le port lié au contrôle du port USB : c'est par lui que le maître attribue une adresse à un périphérique esclave lorsqu'il se connecte au bus USB.
- Un port en émission en mode interrupt (endpoint 1) : buffer 8 B O.
- Un port bidirectionnel en mode interrupt (endpoint 2) : buffer 8 B I/O.

Une lecture du fichier d'en-tête `/usr/src/linux-2.4.22/include/linux/usb.h` fournira au lecteur curieux des informations plus détaillées sur le contenu de ces différentes structures. Fort de toutes ces précieuses informations, voyons comment la fonction `hc08_probe` identifie ou non le microcontrôleur 68HC908JB8.

```
static void * hc08_probe (struct usb_device *udev, unsigned int ifnum,
                          const struct usb_device_id *id)
{
        struct usb_hc08 *dev = NULL;
        struct usb_interface *interface;
        struct usb_interface_descriptor *iface_desc;
        struct usb_endpoint_descriptor *endpoint;
        int i, endpoint_match = 0;

        [...]
```

On vérifie que les identifiants vendeur et produit correspondent bien à ceux du 68HC908JB8. Ce test peut être qualifié de paranoïaque puisque cette condition doit être vérifiée pour que `hc08_probe` soit appelée.

```
        /* See if the device offered to us matches
        what we can accept */
        if ((udev->descriptor.idVendor != USB_HC08_VENDOR_ID) ||
            (udev->descriptor.idProduct != USB_HC08_PRODUCT_ID))
        {
                info ("unable to probe new device");
                return NULL;
        }

        [...]
```

Le microcontrôleur 68HC908JB8 n'étant pas un périphérique complexe, il ne dispose que d'une seule configuration et que d'une unique interface. Nous pouvons donc avancer dans l'arborescence directement jusqu'à l'interface active. Pour un périphérique plus sophistiqué, des tests complémentaires, sur sa classe par exemple, peuvent s'avérer utiles.

```
        interface = &udev->actconfig->interface[ifnum];

        [...]
```

Nous allons donc passer en revue les différents endpoints et voir s'ils sont conformes à nos attentes. En fait, nous devons identifier 2 endpoints en mode interrupt, un étant configuré pour l'envoi de données (OUT) et l'autre pour la réception (IN). Un endpoint supplémentaire, de contrôle, doit également apparaître.

```
        /* setup the endpoint information */
        /* check out the endpoints */
        iface_desc = &interface->altsetting[0];
        for (i = 0; i < iface_desc->bNumEndpoints; ++i)
        {
                endpoint = &iface_desc->endpoint[i];

                if ((endpoint->bmAttributes & USB_ENDPOINT_
XFERTYPE_MASK)
                                == USB_ENDPOINT_XFER_CONTROL)
                        dbg ("%s - control endpoint found",
                                        __FUNCTION__);
```

```
            [...]

            if (!(endpoint->bEndpointAddress & USB_DIR_IN) &&
               ((endpoint->bmAttributes & USB_ENDPOINT_
XFERTYPE_MASK)
                == USB_ENDPOINT_XFER_INT))
            {
```

Une fois un endpoint détecté, une structure de données `struct usb_hc08`, définie et utilisée en interne par notre driver, est initialisée avec des valeurs propres à l'endpoint (adresse, taille du buffer de transfert, etc.). Un URB (USB *Request Block*) est également alloué. Sa fonction et son utilité seront présentées au chapitre suivant. Un compteur `endpoint_match`, correspondant au nombre d'endpoints identifiés est également incrémenté.

```
        /* we found an interrupt out endpoint */
        dev->write_urb = usb_alloc_urb(0);
        if (!dev->write_urb)
        {
        err("No free urbs available");
        goto error;
        }

        buffer_size = endpoint->wMaxPacketSize;
        dev->int_out_size = buffer_size;
        dev->int_out_endpointAddr = endpoint->bEndpointAddress;
        /* as we want to write data packets one by one
        * the interval used to poll urb transfers is
        * set to 0
        * that means ... only one transfer */
        dev->int_out_interval = 0;
        dev->int_out_buffer = kmalloc (buffer_size, GFP_KERNEL);
        if (!dev->int_out_buffer)
        {
        err("Couldn't allocate int_out_buffer");
        goto error;
    }

    endpoint_match ++;
  }

}
```

Si le compte d'endpoints est bon, un pointeur non NULL est retourné au gestionnaire de l'USB. Ce dernier confiera la gestion du périphérique à notre pilote. À l'inverse, si la méthode `probe` détecte une anomalie, le pointeur NULL sera retourné, dégageant notre pilote de toutes obligations envers le matériel.

```
      /* if no endpoint is matching, just exit */
      if (endpoint_match < 2)
      {
            info ("only %d endpoints found instead of 2",
                        endpoint_match);
            goto error;
      }

      [...]
```

```
error:
        hc08_delete (dev);
        dev = NULL;

exit:
        up (&minor_table_mutex);
        return dev;
}
```

LA COMMUNICATION SUR BUS USB

Afin d'illustrer la communication entre le pilote et le 68HC908JB8, nous présenterons la méthode `write` permettant de transmettre des données au microcontrôleur.

```
static ssize_t hc08_write (struct file *file, const char
*buffer, size_t count, loff_t *ppos)
{
        struct usb_hc08 *dev;
        ssize_t bytes_written = 0;
        int retval = 0;

        dev = (struct usb_hc08 *)file->private_data;
        [...]
```

Dans un premier temps, nous récupérons les données que le processus appelant souhaite transmettre. La recopie des données depuis l'espace utilisateur vers le noyau est réalisée à l'aide de la fonction `copy_from_user`.

```
      [...]
      /* we can only write as much as 1 urb will hold */
      bytes_written = (count > dev->int_out_size) ?
                        dev->int_out_size : count;

      /* copy the data from userspace into our urb */
      if (copy_from_user(dev->int_out_buffer, buffer, bytes_written))
      {
            retval = -EFAULT;
            goto exit;
      }
      [...]
```

Le noyau Linux utilise une structure appelée URB (*USB Request Block*) afin de définir les transactions sur le bus USB. Elle contient toutes les informations nécessaires à la transmission physique des données. Elle est décrite dans le fichier `/usr/src/linux/Documentation/usb/URB.txt` fourni avec les sources du noyau Linux. L'URB est défini sous la forme d'une structure de type `struct urb`. Étant donné la complexité de cette dernière, le module `usbcore` met à disposition des pilotes des fonctions afin d'en faciliter l'initialisation. Nous utiliserons la fonction `usb_fill_int_urb` afin de construire un URB destiné à opérer un transfert en mode interrupt avec notre microcontrôleur.

```
      [...]
      /* set up our write urb */
      usb_fill_int_urb (dev->write_urb,
                        dev->udev,
                        usb_sndintpipe (dev->udev, dev-
>int_out_endpointAddr),
```

```
                        dev->int_out_buffer,
                        dev->int_out_size,
                        hc08_write_int_callback,
                        dev, dev->int_out_interval);
        [...]
```

Détaillons les paramètres de cette fonction :

- `dev->write_urb` : représente la `struct urb` en construction.
- `dev->udev` : désigne l'arborescence représentant le périphérique USB.
- `usb_sndintpipe (dev->udev, dev->int_out_endpointAddr)` : cette fonction permet d'associer un *pipe* ou canal de communication à l'endpoint 1 du microcontrôleur 68HC908JB8.
- `dev->int_out_buffer` : buffer contenant les données à transmettre.
- `dev->int_out_size` : taille en octets du transfert à effectuer.
- `dev->int_out_interval` : cette valeur précise l'intervalle de temps entre deux transferts de données. N'oublions pas que le mode de transmission interrupt est périodique. Dans notre application, répéter un même transfert périodiquement ne nous intéresse pas. C'est pourquoi `dev->int_out_interval` est initialisé a 0. Cela signifie que nous souhaitons procéder à un unique transfert.
- `hc08_write_int_callback` : ce paramètre est en fait un pointeur de fonction. Cette dernière sera appelée après la fin du transfert afin de le compléter. Son rôle est de vérifier que tout s'est déroulé normalement puis de réveiller tous les processus en attente de l'achèvement du transfert. Il est utile de préciser que la fonction `hc08_write_int_callback` est appelée dans un contexte d'interruption. Elle doit donc être définie avec les mêmes précautions qu'un gestionnaire d'interruption classique.

Voici comment notre pilote la définit :

```
static void hc08_write_int_callback (struct urb *urb)
{
        struct usb_hc08 *dev = (struct usb_hc08 *)urb->context;

        dbg("%s - minor %d", __FUNCTION__, dev->minor);

        if ((urb->status == -ENOENT) ||
            (urb->status == -ECONNRESET))
        {
    dbg ("%s - write urb cancelled assynchronously or by user ", __FUNCTION__);
        }
        else
        {
                dbg ("%s - write int status : %d", __FUNCTION__, urb->status);
        }

        /* send a signal to the sleeping process,
         * waiting for write again */

        wake_up_interruptible (&dev->int_out_wait);

        return;
}
```

Après avoir initialisé l'URB, il ne reste plus qu'à le soumettre au module `usbcore` afin de procéder au transfert. Pour cela, la fonction `usb_submit_urb`, encore une fois exportée par `usbcore`, est utilisée.

```
        [...]
        retval = usb_submit_urb (dev->write_urb);
        [...]
```

Une fois le transfert initié, le processus utilisateur appelant est mis en sommeil. La fonction de complétion `hc08_write_int_callback` le réveillera une fois le travail achevé. La communication avec le 68HC908JB8 est donc bloquante. Cela signifie qu'un programme communiquant avec le microcontrôleur sortira des appels système `read` et `write` uniquement après le transfert effectif des données.

```
while (dev->write_urb->status == -EINPROGRESS)
{
        retval = wait_event_interruptible (dev->int_out_wait,
                dev->write_urb->status);
        if (retval == -ERESTARTSYS)
        {
                dbg ("%s - error wait_event_interruptible ()",
                    __FUNCTION__);
                goto exit;
        }
}
```

En raison de ses nombreuses similitudes avec `hc08_write`, la méthode `hc08_read`, permettant de lire les données transmises par le microcontrôleur, n'est pas présentée ici. Après avoir défini au niveau du driver les méthodes de communication avec le 68HC908JB8, il ne reste plus qu'à les utiliser au niveau du programme utilisateur par l'intermédiaire des appels système `read` et `write`. La communication avec le microcontrôleur se déroulera donc classiquement comme avec tout autre périphérique.

LA COMMUNICATION AVEC LE PROGRAMME UTILISATEUR

Une fois le driver fonctionnel, le développement d'une application pour communiquer par USB avec le 68HC908JB8 est une simple formalité. Voici l'extrait principal du programme utilisateur permettant de faire clignoter la diode du circuit de test (PTD0/1) et de lire les valeurs renvoyées par le microcontrôleur :

```
/* read on the hc08 device */
while (num)
{
        /* trying to write... in hope of a reading */
        if ((trans=write (desc, data_write, SIZE)) < SIZE)
        {
                fprintf (stderr, "only %d bytes written
instead of %d\n", trans, SIZE);
                perror ("write ()");
        }
        memset (data_read, 0x00, SIZE);
        if ((trans=read (desc, data_read, SIZE)) < SIZE)
        {
```

```
                fprintf (stderr, "only %d bytes read instead
of %d\n", trans, SIZE);
                perror ("read ()");
        }
        printf ("buffer : ");
        for (j=0; j<SIZE; j++)
        {
                printf ("%.2x ", data_read[j] & 0xff);
        }
        printf ("\n");
        if (*data_write)
        {
                memset (data_write, 0x00, SIZE);
        }
        else
        {
                memset (data_write, 0xff, SIZE);
        }
        usleep (200000);
        num --;
}
```

Le lecteur attentif aura remarqué une alternance entre les lectures et les écritures sur le microcontrôleur 68HC908JB8. Cette bizarrerie est à attribuer à une limitation du *firmware* fourni par MCT Elektronikladen. En effet, le tampon utilisé en interne est le même pour les lectures et les écritures. Trois écritures remplissent le tampon et les suivantes sont bloquées. La seule méthode pour permettre la poursuite des transactions est de vider le tampon. Cette opération est réalisée par la lecture de données provenant du 68HC908JB8. L'inverse est également vrai : 3 lectures vident le tampon et les lectures suivantes sont bloquées. Cette fois, des écritures sont nécessaires pour recharger le tampon. En conclusion, pour pouvoir lire, il faut écrire et inversement.

CONCLUSION ET PERSPECTIVES

Nous avons présenté ici toutes les étapes de développement au moyen d'outils open source d'un périphérique USB basé sur le microcontrôleur Freescale 68HC908JB8. Nous avons présenté le protocole de programmation et les bases de l'assembleur 6808 au moyen de quelques exemples simples, pour finalement aboutir au stockage en mémoire non volatile d'un programme d'exemple utilisant le port USB. Nous avons ensuite abordé les méthodes de programmation de modules GNU/Linux pour communiquer sur le bus USB et avons insisté sur les diverses couches d'abstractions implémentées. Ces notions ont été appliquées au cas particulier de la communication avec le 68HC908JB8 : un programme a été présenté pour commander l'allumage ou l'extinction d'une diode via le port USB. Nous nous sommes imposés de garder le code USB fixe afin de limiter les risques d'erreur et avons focalisé nos efforts sur le développement d'un driver GNU/Linux communiquant avec un microcontrôleur 68HC908JB8 exécutant ce code, avec les contraintes liées à la réutilisation d'un code existant (dans ce cas, une subtilité du code distribué est que toute lecture nécessite une écriture et réciproquement).

Nous avons donc atteint nos objectifs de :

- Nous familiariser avec USB ;
- Développer l'ensemble des outils pour le 68HC908 sous Linux ;
- Développer un module Linux pour l'EVB USB du 68HC908.

Le travail à réaliser à partir de là est le développement de notre propre code exécuté sur le 68HC908 répondant plus spécifiquement à une application donnée et la modification du driver de façon conjuguée pour s'adapter au nouveau protocole de communication qui sera alors implémenté. Le module présenté ici s'adapte facilement à toutes sortes d'applications.

REMERCIEMENTS

Nous remercions Benoît Demaine pour les discussions stimulantes au cours de cette étude. L'association de diffusion des Logiciels libres sur la Franche-Comté – Sequanux (www.sequanux.org) est remerciée pour son support logistique.

Jean-Michel Friedt (friedtj@free.fr),
Simon Guinot (simon@sequanux.org)
Association Projet Aurore

LIENS

▶ [1] « *Programming Guide for Linux USB Device Drivers* », disponible à http://wwwbode.cs.tum.edu/Par/arch/usb/usbdoc/

▶ [2] M. Zerkus, J. Lusher & J. Ward, « USB *Primer* », *Circuit Cellar* 106 (mai 1999), pp. 58-69, puis J. Lyle, « USB *Primer* », *Circuit Cellar* **107** (juin 1999), pp. 68-73.

▶ [3] « Développez vos pilotes de périphériques USB », *GNU/Linux Magazine* Hors-Série 17, novembre-décembre 2003.

▶ [4] « *USB08 Universal Serial Bus Evaluation Board Using the MC68HC908JB8* », disponible à http://www.freescale.com/files/microcontrollers/doc/ref_manual/DRM002.pdf

▶ [5] « MC68HC908JB8, MC68HC08JB8, MC68HC08JT8 Technical Data (rev. 2.3) » (09/2005) à www.freescale.com/files/microcontrollers/doc/data_sheet/MC68HC908JB8.pdf

▶ [6] « CPU08 Central Processor Unit Reference Manual (CPU08RM/AD Rev. 3, 2/2001) » à http://www.freescale.com/files/microcontrollers/doc/ref_manual/CPU08RM.pdf

▶ [7] « *Universal Serial Bus Specifications* (rev. 1.1) » à http://www.usb.org/developers/docs/

▶ [8] G. Kroah-Hartman, `/usr/src/linux/drivers/usb/usb-skeleton.c`, dans les sources de Linux

▶ [9] A. Rubini & J. Corbet, « *Linux Device Drivers, 2nd ed.* » , O'Reilly Ed. (2001), disponible à http://www.xml.com/ldd/chapter/book/

Pierre Ficheux :: pficheux@com1.fr

PORTS SÉRIE SOUS LINUX

Le but de cet article est la compréhension du fonctionnement des ports série sous LINUX. L'article abordera également la configuration des ports série depuis le shell sh ainsi qu'en langage C.

RAPPELS SUR LE PORT SÉRIE

PRINCIPE DU PORT SÉRIE

Le port série est une interface *hardware* permettant de transmettre des informations binaires de manière séquentielle (1 par 1). Par opposition, un port parallèle comme le port Centronics des imprimantes transmet les bits de manière simultanée (8 à la fois).

Le mode de transmission sur une ligne série (appelée aussi RS-232) est décrit dans le schéma ci-dessous :

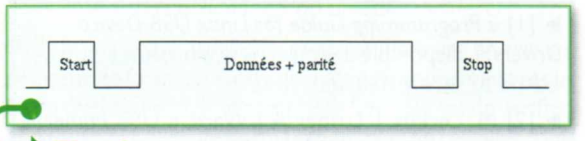


Fig. 1

L'arrivée des données est indiquée par un bit de START. Les données sont transmises sous forme d'une suite de 5 à 8 bits contenant éventuellement un bit de PARITE utilisé pour la détection des erreurs de transmission. La fin de la transmission est indiquée par un bit de STOP. La fréquence de transmission des bits indique la vitesse de la ligne : 9600 bits/s, 19200 bits/s, etc.

CONNECTIQUE ET SIGNAUX

Concrètement, un port série est matérialisé par un connecteur contenant 25 points (DB25) ou 9 points (DB9). Ce dernier est le plus répandu sur les machines modernes lorsqu'elles ont encore la chance de disposer d'un tel port. Au besoin, on pourra acquérir une carte PCI/Série pour quelques euros. Un port série à bas débit n'a besoin que de trois lignes :

Tab. 1

1	FGND (*Frame Ground*)	Masse mécanique
2	TD (*Transmit Data*)	Transmission des données
3	RD (*Receive Data*)	Réception des données

Ce type de connexion minimale sera utilisable pour connecter un équipement à basse vitesse, comme un terminal asynchrone (en croisant le TD et le RD). Malheureusement, ces signaux ne sont pas suffisants pour gérer correctement des périphériques plus évolués comme des MODEMS. On doit alors utiliser les signaux de contrôle suivant :

Tab. 2

20	DTR (*Data Terminal Ready*)	Présence du DTE
6	DSR (*Data Set Ready*)	Présence du DCE
8	DCD (*Data Carrier Detect*)	Connexion DCE établie

Dans la terminologie des lignes série, l'équipement de communication est appelé DCE (*Data Communication Equipment*) et le calculateur est appelé DTE (*Data Terminal Equipment*). Les signaux DTR et DCD sont les plus utilisés dans la pratique :

- Le signal DTR indique au MODEM que le calculateur est actif. La chute de ce signal provoquera la déconnexion du MODEM, ce qui est bien pratique si l'on veut éviter de rincer excessivement France Télécom ;-))
- Le signal DCD indique au calculateur que le MODEM est connecté au MODEM distant. La chute de ce signal (indiquant la déconnexion) sera détectable par le programme applicatif afin de déclencher la procédure adéquate.

En plus des signaux de contrôle indispensables, on peut également connecter les signaux RTS et CTS (*hardware handshake*, contrôle de flux hard). Le DTE peut affirmer RTS (*Request To Send*) pour demander au DCE s'il peut envoyer des données. En réponse, le DCE affirme CTS (*Clear To Send*) pour indiquer qu'il est prêt. Si le DCE n'est pas prêt, il fait tomber CTS, idem pour de DTE avec RTS.

Dans le cas d'une prise 9 points cotés DTE (cas des PC en particulier), cela donne :

Tab. 3

3	TD	→	TD	2
2	RD	←	RD	3
7	RTS	→	RTS	4
8	CTS	←	CTS	5
6	DSR	←	DSR	6
5	GND	↔	GND	7
1	DCD	←	DCD	8
4	DTR	→	DTR	20

DÉTECTION DU HARDWARE RS-232 PAR LINUX

Il existe de nombreux circuits permettant de gérer des ports RS-232. Les plus anciens se souviendront avec nostalgie de l'ACIA 6850, périphérique bien connu de l'antique processeur 6800 de chez MOTOROLA. Les PC d'aujourd'hui utilisent des circuits appelés UART comme le 16550A. Le noyau Linux contient naturellement des pilotes pour ce type de périphérique. La détection des ports série par le noyau est indiqué au démarrage de Linux par les lignes :

```
Serial: 8250/16550 driver $Revision: 1.90
  $ 48 ports, IRQ sharing enabled
ttyS0 at I/O 0x3f8 (irq = 4) is a 16550A
ttyS1 at I/O 0x2f8 (irq = 3) is a 16550A
```

Pour cela il faut avoir validé l'option suivante lors de la compilation du noyau :

```
Standard/generic serial support
  (CONFIG_SERIAL) [Y/m/n/?]
```

Si le système dispose d'une carte série additionnelle, on verra apparaître à la suite les messages :

```
tty02 at I/O 0x03e8 (irq = 4) is a 16550A
```

Dans cet exemple, on voit que la deuxième carte est bien détectée à une adresse différente (3e8), mais que le système lui a attribué la même interruption (IRQ) que le premier port série. Pour assurer le bon fonctionnement, il sera nécessaire de modifier le niveau d'interruption en accord avec la configuration de la carte. Pour cela, on utilisera la commande `setserial` appelée par exemple dans le script `rc.local`:

```
/bin/setserial /dev/cua2 irq 5
```

On voit que les ports série sont utilisables à travers les fichiers spéciaux `/dev/ttyS0`, `/dev/ttyS1` et ainsi de suite.

Jusqu'au noyau 2.0, les utilisateurs de GNU/Linux avaient l'habitude de distinguer les *devices* série sortants (*call-out*) `cuax` des devices entrants (*call-in*) `ttySx`. Les actuelles distributions utilisant un noyau bien supérieur indiquent que les devices des ports série sont maintenant `ttySx` (plus de `cuax`) et que les applications doivent utiliser ces devices. Il existe cependant une différence notoire décrite dans le document Serial-HOWTO.

La seule différence est dans la manière dont le périphérique est ouvert. Le périphérique d'entrée `/dev/ttySx` est ouvert en mode bloquant, jusqu'à ce que CD soit positionné (i. e., quelqu'un se connecte). Cette différence peut provoquer des disfonctionnements de certaines applications si l'on utilise `ttySx` à la place de `cuax` alors que l'application n'a pas été prévue pour cela (en l'occurrence des blocages sur l'ouverture du device). Quelques infos complémentaires sont disponibles au paragraphe 4.2.

Pour tester la communication sur un port série, le mieux est d'utiliser un outil d'émulation de terminal comme `kermit` ou `minicom`. Ce dernier est livré en standard avec les distributions classiques.

UTILISATION À TRAVERS LE SHELL

LA COMMANDE STTY

Cette commande permet de visualiser et modifier les paramètres d'un terminal. Cette dénomination est à prendre au sens large et l'utilisation de `stty` ne se limite pas aux lignes série. Au niveau Linux, le terminal est défini comme un canal de communication associé à une discipline de ligne (line discipline) qui détermine le comportement de ce canal.

Au niveau de l'émulateur de terminal xterm, on peut utiliser simplement la commande en faisant :

```
pierre@mmxpf % stty -a
speed 9600 baud; rows 24; columns 80; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = ;
eol2 = ; start = ^Q; stop = ^S; susp = ^Z; rprnt = ^R; werase = ^W;
lnext = ^V; flush = ^O; min = 1; time = 0;
-parenb -parodd cs8 -hupcl -cstopb cread -clocal -crtscts
-ignbrk -brkint -ignpar -parmrk -inpck -istrip -inlcr -igncr icrnl ixon -ixoff
-iuclc -ixany -imaxbel
opost -olcuc -ocrnl onlcr -onocr -onlret -ofill -ofdel nl0 cr0 tab0 bs0 vt0 ff0
isig icanon iexten echo echoe echok -echonl -noflsh -xcase -tostop -echoprt
echoctl echoke
```

L'option `-a` permet d'afficher l'ensemble des paramètres du terminal. L'exemple ci-dessous nous indique quelles valeurs sont affectées aux caractères de contrôle de ce terminal (effacement, interruption, etc.). Il est bien entendu possible de modifier ces caractères de contrôle en utilisant `stty`, par exemple :

```
stty erase "^H"
```

Affecte [CTRL-H] à l'effacement de caractère. De même, on peut modifier le paramétrage de la ligne : vitesse, nombre de bits de données, parité, traitement des signaux de contrôle. Dans notre exemple, la ligne n'utilise pas de parité (`-parenb -parodd`), utilise 8 bits de données (`cs8`) et ne traite pas de contrôle de flux hardware (`-crtscts`) ce qui n'est pas très étonnant pour un terminal attaché à un xterm ! La liste complète des paramètres gérés par `stty` est bien entendu disponible en faisant `man stty`.

Pour revenir au sujet qui nous intéresse aujourd'hui, on peut bien entendu utiliser `stty` sur un terminal série. Par la magie de la redirection des entrées/sorties sous Linux,

on peut par exemple afficher les paramètres de la ligne utilisée par le modem en faisant :

```
stty -a < /dev/modem
```

Le device /dev/modem étant en général un lien symbolique sur un device série réel :

```
pierre@mmxpf % ls -l /dev/modem
lrwxrwxrwx  /dev/modem -> cua0
```

Le paramétrage d'une ligne série par un shell-script utilisant stty est de la même veine :

```
#!/bin/sh
(
stty 115200
stty cs8
stty -parenb
stty -parodd
stty -clocal
stty crtscts
stty -echo
# Suite du script
...
) < /dev/cua0 > /dev/cua0
```

Dans cet exemple, on effectue le paramétrage suivant :

Tab. 4

115200	Vitesse à 115200 bps
cs8	8 bits de données
-parenb	Pas de parité paire
-parodd	ni impaire d'ailleurs...
-clocal	contrôle de flux non local traitement des signaux modem (DCD, DTR)
crtscts	activation du contrôle de flux hardware
-echo	pas d'écho local des caractères

LA COMMANDE CHAT

Le mot *chat* signifie en anglais « bavarder, discuter ». La commande chat permet d'exécuter une séquence de type expect/send (j'envoie/j'attends) sur un terminal donné. La commande est en général associée au package du démon pppd. Elle sera donc installée par défaut (ou du moins disponible) sur la plupart des distributions courantes.

Une utilisation pratique de chat pourra être par exemple l'initialisation d'un périphérique connecté à une ligne série. Comme beaucoup de commandes sous Linux, chat utilise par défaut l'entrée et la sortie standard ce qui permet de tester le chat-script plus facilement.

Prenons l'exemple d'un script destiné à initialiser un modem, puis appeler un numéro de téléphone et attendre la connexion. On pourra facilement mettre au point un tel script directement dans un xterm en tapant la commande :

```
chat -v "" ATM0\&C1\&D2\&K3 OK ATDT3611 CONNECT
```

Ce qui signifie :

Tab. 5

""	On n'attend aucune chaîne au départ
ATM0\&C1\&D2\&K3	On envoie une chaîne d'init...
OK	Et on attend OK
ATDT3611	On compose le numéro...
CONNECT	Et on attend CONNECT

L'option -v indique de tracer le dialogue en utilisant le syslog. En général, la trace sortira sur le fichier /var/log/message (suivant la configuration du syslog). En simulant la réponse du modem à la main (en tapant OK puis CONNECT dans le même terminal), on obtient la trace du dialogue dans le fichier de log :

```
Sep 26 21:31:01 mmxpf chat[463]: send (ATM0&C1&D2&K3^M)
Sep 26 21:31:02 mmxpf chat[463]: expect (OK)
Sep 26 21:31:03 mmxpf chat[463]:  -- got it
Sep 26 21:31:03 mmxpf chat[463]: send (ATDT3611^M)
Sep 26 21:31:03 mmxpf chat[463]: expect (CONNECT)
Sep 26 21:31:05 mmxpf chat[463]:  -- got it
```

Lorsque le script est au point, il suffit de rediriger l'entrée et la sortie sur le device du modem :

```
chat -v "" ATM0\&C1\&D2\&K3 OK ATDT3611 CONNECT < /dev/cua0 >
/dev/cua0
```

Forts de notre expérience de la commande stty, nous pourrons donc combiner l'initialisation de la ligne série avec le chat-script à envoyer au modem :

```
#!/bin/sh
(
# Init ligne série
stty 115200
stty cs7
stty parenb
stty -clocal
stty crtscts
stty -echo
# Init modem et appel
chat -v "" ATM0\&C1\&D2\&K3 OK ATDT3611 CONNECT
) < /dev/cua0 > /dev/cua0
```

Le man chat vous donnera beaucoup plus d'informations concernant les possibilités de ce petit programme comme par exemple la définition du chat-script dans un fichier séparé.

PROGRAMMATION DES PORTS SÉRIE EN C

INTERFACE POSIX (TERMIOS)

La programmation des ports série sous Linux utilise l'interface POSIX définie dans le fichier termios.h. Le but de ce paragraphe n'est pas de fournir une description

complète de termios. Vous devrez pour cela vous référer à la page de manuel termios, aux divers documents consacrés au sujet. Le premier exemple ci-dessous permet de forcer le raccrochage (*hangup*) d'un modem en faisant chuter le signal DTR. Dans la définition de l'interface POSIX des terminaux, cela s'effectue simplement en positionnant la vitesse à 0 (valeur B0) :

```
#include <stdio.h>
#include <stdlib.h>
#include <termios.h>
#include <fcntl.h>

main (int ac, char **av)
{
  int fd;
  struct termios tty, old;
  /* Ouverture du device */
  if ((fd = open (av[1], O_RDWR)) < 0) {
    perror (av[1]);
    exit (1);
  }
  /* Lecture des paramètres */
  tcgetattr(fd, &tty);
  tcgetattr(fd, &old);

  /* On passe la vitesse à 0 ==> hangup */
  cfsetospeed(&tty, B0);
  cfsetispeed(&tty, B0);

  /* On applique le nouveau paramètrage pendant 1s */
  tcsetattr(fd, TCSANOW, &tty);
  sleep(1);

  /* On revient à l'ancien et on quitte */
  tcsetattr(fd, TCSANOW, &old);

  close (fd);
}
```

On pourra faire exactement la même chose en shell en faisant :

```
stty 0 < /dev/cua0 > /dev/cua0
```

Par défaut, une ligne série sous Linux est ouverte en mode canonique. Le mode canonique consiste à traiter les entrées d'un terminal comme une ligne terminée par un séparateur (le plus souvent LF : *line-feed* correspondant à la touche RETURN). Cela signifie que le programme applicatif ne pourra disposer de la saisie que lorsque le séparateur est reçu. Dans le mode canonique, un certain nombre de caractères de contrôles sont disponibles (effacement, etc. voir la commande stty).

Ce comportement est adapté au traitement d'un terminal réel avec un dialogue opérateur, mais il n'est pas utilisable dans le cas du dialogue avec un équipement de type modem par exemple. Le deuxième exemple permet donc de positionner et d'utiliser une ligne série en mode direct (raw), et donc d'inhiber le mode canonique :

```
/* Fixe un device en mode RAW */
void raw_mode (fd, old_term)
int fd;
struct termios *old_term;
{
    struct termios term;
    tcgetattr(fd, &term);
    /* Sauve l'ancienne config dans le paramètre */
    tcgetattr(fd, old_term);
    /* mode RAW, pas de mode canonique, pas d'echo */
    term.c_iflag = IGNBRK;
    term.c_lflag = 0;
    term.c_oflag = 0;
    /* Controle de flux hardware RTS/CTS)
    term.c_cflag |= (CREAD | CRTSCTS);
    /* 1 caractère suffit */
    term.c_cc[VMIN] = 1;
    /* Donnée disponible immédiatement */
    term.c_cc[VTIME] = 0;
    /* Inhibe le controle de flux XON/XOFF */
    term.c_iflag &= ~(IXON|IXOFF|IXANY);
    /* 8 bits de données, pas de parité */
    term.c_cflag &= ~(PARENB | CSIZE);
    term.c_cflag |= CS8;
    /* Gestion des signaux modem */
    term.c_cflag &= ~CLOCAL;
    tcsetattr(fd, TCSANOW, &term);
}
```

AUTRE TYPE DE CONFIGURATION

Un descripteur de fichier sur un port série sera ouvert par une ligne du type

```
fd = open (device_name, O_RDWR)
```

Par défaut, cette ligne est ouverte en mode bloquant, ce qui peut parfois poser des problèmes dans le cas de l'utilisation du device /dev/ttySx (voir paragraphe 2). On peut alors ouvrir le device en mode non bloquant par l'appel :

```
fd = open (device_name, O_RDWR | O_NDELAY)
```

CONCLUSION

Malgré son grand âge, le port série est largement utilisé dans de nombreuses applications industrielles (acquisition de données, pilotage d'équipement). La maîtrise des techniques d'exploitation de cette interface est souvent indispensable pour la réalisation d'un système industriel complexe. En dépit de son apparente complexité, l'interface POSIX utilisée sous LINUX est d'une grande efficacité et laisse peu de place aux comportements imprévus.

LIENS

- Le Serial-HOWTO sur : http://www.freenix.fr/unix/linux/HOWTO-vo/Serial-HOWTO.html
- Le Serial-Programming-HOWTO sur : http://www.freenix.fr/unix/linux/HOWTO-vo/Serial-Programming-HOWTO.html
- Le document *Serial Programming Guide for POSIX Compliant Operating Systems* sur : http://dns-gate.easysw.com/~mike/serial
- L'ouvrage *Programmation Linux 2.0* (Card/Dumas/Mével) aux éditions EYROLLES
- L'utilitaire Kermit sur : http://www.columbia.edu/kermit/ck60.html
- L'utilitaire Minicom sur : http://www.pp.clinet.fi/~walker/minicom.html

Denis Bodor :: db@ed-diamond.com :: lefinnois@lefinnois.net

UN CAPTEUR DE TEMPÉRATURE SUR BUS I2C

Le but de cet article est la compréhension du fonctionnement des ports série sous LINUX. L'article abordera également la configuration des ports série depuis le shell sh ainsi qu'en langage C.

Le bus i2c est parfaitement pris en charge par le noyau Linux. En effet, le projet Lm_sensors (http://secure.netroedge.com/~lm78/) développant la prise en charge et la gestion des capteurs présents dans un PC a, par effet de bord, largement contribué à faciliter l'accès à ce bus.

i2c est l'acronyme de *Inter-Integrated Circuit*. Il s'agit d'un bus développé au début des années 80 par Philips. L'objectif poursuivi était de minimiser et standardiser les liaisons entre les circuits intégrés numériques de ses produits comme les magnétoscopes, les TV, la Hi-fi, etc. Depuis, le bus a « fait son trou » et est maintenant présent un peu partout. On trouve également au détail une gamme complète de composants i2c ou compatibles (la désignation « i2c » est une marque déposée par Philips). Le bus i2c est normalisé et également utilisé sous la forme de SMBus ou du DDS permettant la reconnaissance automatique du moniteur par la carte graphique et le système d'exploitation.

Physiquement, le bus i2c se compose de quatre lignes :

- Une alimentation en +5 V permettant, dans la plupart des cas, d'alimenter directement les composants présents sur le bus comme les mémoires EEPROM ou les *latchs* ;
- Une masse (référence) ;
- Une ligne d'horloge permettant la synchronisation du bus et la transmission des données nommée « SCL » ;
- Une ligne de donnée nommée « SDA ».

Les interfaces électriques sont de type collecteur ouvert et on utilise une résistance de *pull-up* au +5 V. Cela signifie que, lorsque la sortie d'une interface est au niveau logique 0, elle n'est connectée à rien. Cela permet ici la présence de plusieurs « maîtres » sur le bus et évite les conflits entre composants.

i2c est un bus. Cela signifie que les périphériques i2c y sont connectés en parallèle (figure 1). Il est donc possible d'avoir, sur un même bus, plusieurs composants avec lesquels dialoguer. Sur le bus, le maître est le donneur d'ordre et l'esclave celui qui y répond. L'esclave ne fait que répondre. S'il se met à « parler » de lui-même et donner des ordres, il devient maître. i2c permet la présence de plusieurs maîtres sur un seul et même bus.

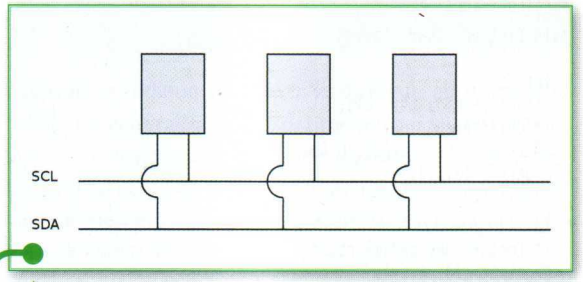


Fig. 1

PROTOCOLE

Avant de voir le plus simple, qu'est l'utilisation via le support Linux, voyons l'aspect plus délicat qu'est le protocole lui-même. Si vous souhaitez simplement contrôler des composants i2c depuis GNU/Linux, vous n'aurez pas forcément besoin de le maîtriser. Il en va tout autrement si vous souhaitez faire communiquer un microcontrôleur en i2c avec des composants compatibles ou avec votre PC sous GNU/Linux. Dans tous les cas, cependant, une certaine compréhension du protocole est toujours utile.

Le protocole de communication repose sur la transmission d'une succession de bits cadencée par la ligne SCL. La figure 2 présente une trame i2c complète. Avant d'envoyer des données sur le bus, un composant doit s'assurer de sa disponibilité. Pour ce faire, il vérifie que les lignes SDA et SCL sont au repos à l'état haut. Si c'est le cas, il définit la condition de départ avec un changement d'état de SDA sans influer sur SCL (S sur la figure 2). Ensuite, il peut transmettre l'adresse du composant i2c auquel il s'adresse sur 7 bits (en bleu).

Les composants i2c ont tous une adresse fixe propre à leur modèle qui peut être déclinée en utilisant des broches spécifiques. La partie fixe de l'adresse est constituée des bits 3 à 6 et la partie variable de 0 à 2. On peut ainsi avoir sur un même bus 8 composants identiques avec des adresses différentes. Après l'adresse suit un bit de lecture (SDA haut) ou d'écriture (SDA bas). Le composant esclave pointé par l'adresse envoyée sur le bus par le maître accuse réception en mettant SDA à l'état bas alors que le maître le maintient à l'état haut. Ceci n'est possible que grâce à l'architecture même du bus i2c et à ses résistances de pull-up.

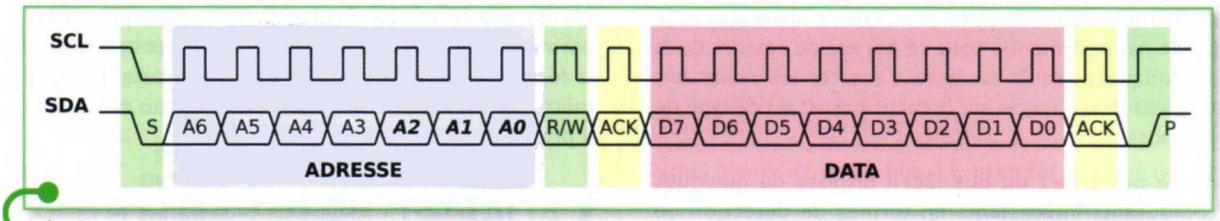


Fig. 2

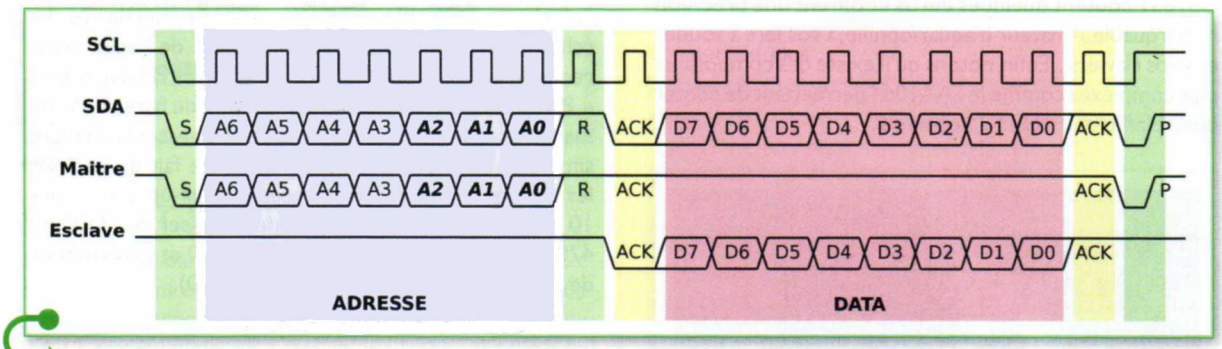


Fig. 3

Lors d'une lecture commandée par le maître, l'esclave se met alors à transmettre des données sur le bus. La figure 3 montre les données sur la ligne SDA et la séparation maître/esclave de l'émission de celles-ci. Un bit d'acquittement (accusé réception) imposé par le maître suit chaque groupe de 8 bits transmis par l'esclave.

Le bit d'acquittement est très important. Il permet au receveur (maître ou esclave) de signifier qu'il a bien reçu une information (adresse ou donnée). Un bit d'acquittement est émis par le receveur en mettant SDA au niveau bas lors de l'impulsion d'horloge (toujours contrôlée par le maître) alors que l'émetteur libère SDA (niveau haut). Dans le cas précis d'un maître recevant des données d'un esclave, le dernier octet reçu n'est pas acquitté. L'esclave libère alors le bus pour que le maître puisse générer une condition d'arrêt. La figure 4 présente un diagramme provenant de la documentation du capteur de température DS75 de Maxim Dallas. On voit clairement la réception des deux octets de données concernant la température. Le premier octet est correctement acquitté par le maître et le second non.

La condition d'arrêt marque la libération du bus par le maître. Alors que SDA et SCL sont tous deux au niveau bas, SDA est passé au niveau haut, puis il est fait de même pour SCL. Maître et esclave retournent alors à un état de veille en attendant une nouvelle condition de départ, les deux lignes étant au repos (niveau haut). L'écriture de données par le maître n'est pas très différente si ce n'est pour le bit d'acquittement. Dans ce cas, c'est le maître qui libère le SDA (état haut) et l'esclave qui accuse réception en mettant la ligne à l'état bas. En fin de transmission, le maître génère la condition d'arrêt.

COMPOSANTS ET BUS I2C

Je l'ai dit précédemment, il existe une vaste gamme de composants directement compatibles i2c. Il en existe pour tout usage, comme la mémoire statique PCF8571P (128 x 8 bits) ou EEPROM PCF8582C2P (256 x 8 bits). Mais les composants les plus intéressants à mon goût restent les PCF8574P et PCF8574AP.

Identiques en dehors de leurs 4 bits d'adresse fixe, ils permettent tout simplement de disposer, par composant, de 8 lignes de données parallèles. Pour quelques 4 euros par pièce, vous pouvez ainsi vous créer un système d'entrées/sorties de 16 fois 8 bits soit 128 lignes adressables individuellement. Certes, le bus i2c n'offre pas les performances d'une carte industrielle, mais vous pourrez ainsi, sur un seul

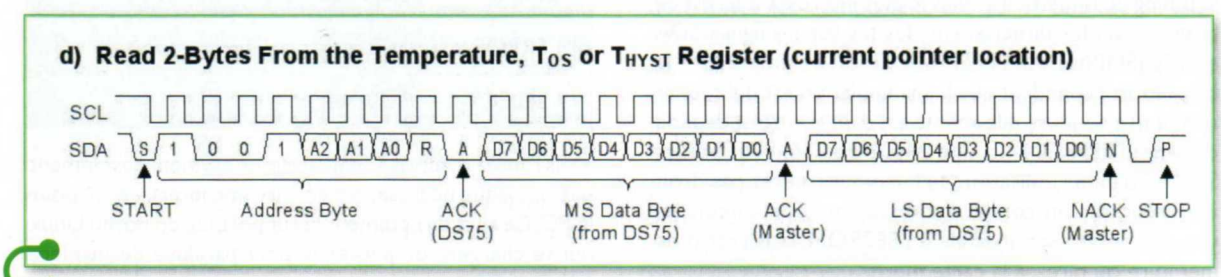


Fig. 4

et même bus, développer tout type d'applications. Je pense naturellement à tout ce qui est du ressort de la domotique, puisqu'il est facile d'adapter les indications données dans l'article sur la carte à relais de ce type de composants.

Autre point fort du bus i2c, il dispose de quelques composants intéressants en termes de détection de température. Des capteurs comme le DS75, LM78 ou DS1631 coûtent quelques euros et offrent une précision remarquable. Amateur d'aquariophilie, à vos fers à souder et à vos claviers... Enfin, notons qu'il existe des composants plus complexes comme le SAA1064 permettant de piloter quatre afficheurs LED 7 segments.

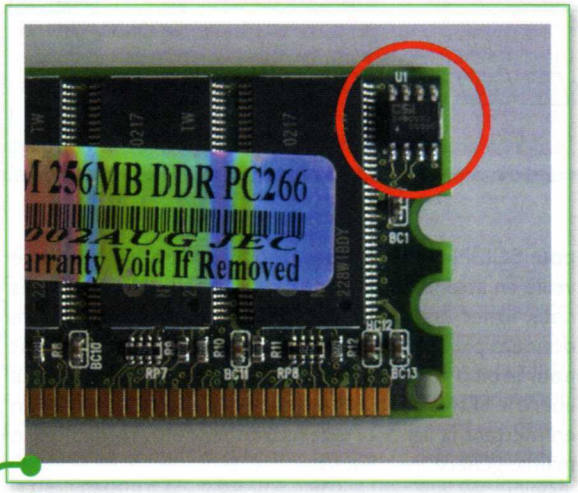


Fig. 5

Un bus i2c est sans doute déjà présent dans votre PC. Si vous avez de la chance, vous disposerez d'un connecteur SMBus de quatre broches directement sur la carte mère. Le SMBus est une déclinaison de l'i2c, mais ne supporte qu'une vitesse de 100kHz, alors que les dernières spécifications i2c précisent également des vitesses de 400kHz et 3.4MHz. Cependant, à notre échelle de mise en œuvre, on peut considérer que SMBus et i2c sont équivalents.

Si vous n'avez pas de chance, le SMBus sera difficilement accessible sur la carte mère. La méthode la plus simple (la moins dangereuse pour l'équipement) est d'accéder au bus via une EEPROM placée sur une barrette mémoire (figure 5). Cette EEPROM est programmée par le BIOS pour stocker différentes informations comme par exemple le timing de la mémoire. Mais le plus intéressant ici est de pouvoir souder directement des fils sur les lignes SDA et SCL (SMBus DATA et SMBus Clock) et ainsi utiliser le bus. Notez cependant que la manipulation est délicate et risquée. La soudure elle-même est dangereuse et dépend du type d'EEPROM (ici une 24c02 de 256x8 en boîtier SO8). Quant à l'utilisation du bus, vous n'aurez pas droit à l'erreur, car un conflit d'adresse, un court-circuit ou une surtension serait fatale à l'EEPROM, la barrette de mémoire ou pire... à la carte mère.

Notez que la société Analog Devices fabrique un kit permettant une connexion identique. L'adaptateur prend la forme d'une barrette DIMM se plaçant sur la carte mère. La technique est toujours la même mais en version « propre ».

ADAPTATEUR PARALLÈLE

Si vous ne souhaitez pas prendre de risques, la meilleure solution est d'utiliser un adaptateur parallèle/i2c. Le schéma est donné en figure 6. Il s'agit de l'adaptateur reconnu par le support i2c de Linux sous la désignation « Philips adapter ». L'adaptateur permet de faire du PC un maître ou un esclave sur le bus. Le circuit est relativement simple, le sextuple inverseur 74LS05 ne fait qu'inverser les signaux et les six résistances de pull-up seront des 10 K Ohms. Vous pouvez également utiliser des 3300 ou 4700 Ohms. Notez la présence de la LED et sa résistance de 330 Ohms (à recalculer selon la LED).

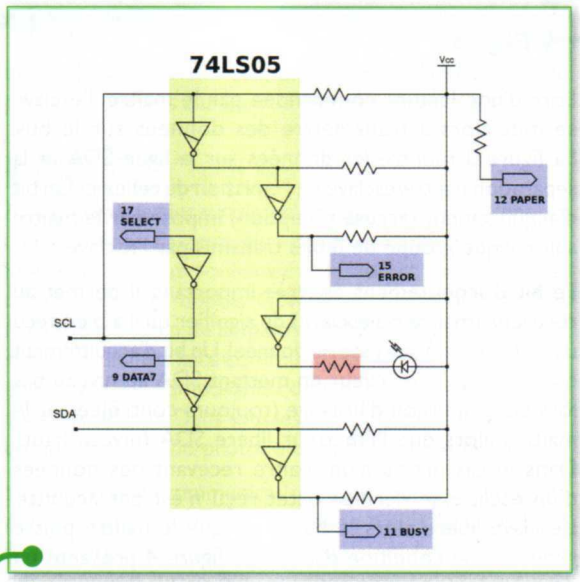


Fig. 6

Nous avons ainsi la correspondance suivante :

Tab. I

11 BUSY	SDA in
9 DATA7	SDA out
15 ERROR	SCL in
17 SELECT	SCL out

Vous l'aurez compris, l'adaptateur ne contient absolument pas la logique nécessaire à en faire une interface i2c pour le PC. Ce sera directement le support i2c du noyau Linux qui se chargera de piloter le port parallèle de manière adéquate.

La figure 7 présente un typon pour l'adaptateur. Vous pouvez également le télécharger au format PDF à l'URL http://www.lefinnois.net/pcb_i2c.pdf.

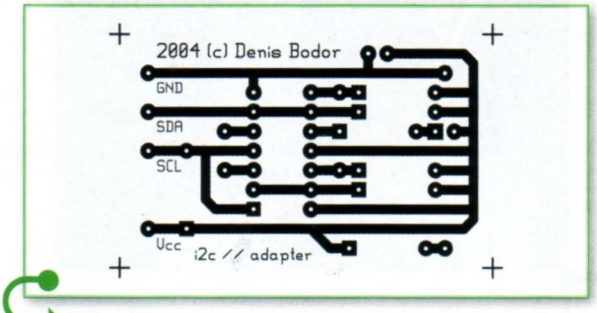


Fig. 7

I2C ET LINUX

Déjà présent dans la série 2.4 du noyau, le support i2c s'est largement étoffé avec la série 2.6. En jetant un coup d'œil dans l'arborescence des modules d'un noyau de 2.6.12, par exemple, on y trouve une large collection de modules. Le sous-répertoire `busses` contient les éléments permettant la prise en charge de la plupart des bus présents sur les cartes mère. La machine de test ayant servi à cet article (un AMD Athlon XP 2400+ sur une carte mère ASRock) dispose d'un bus i2c interne supporté par le module `i2c-sis96x`.

L'adaptateur parallèle décrit précédemment est supporté par le module `i2c-parport`. On notera également l'existence de `i2c-parport-light` ne reposant pas sur le support `parport`, mais utilisant directement `outb`/`inb` sur l'adresse E/S du port. Le module développé par Jean Delvare supporte plusieurs types d'adaptateurs qui se connectent tous au port parallèle se distinguant par l'utilisation différente des lignes du port. Celui décrit ici est le type 0 correspondant au « Philips adapter ». Un simple `modinfo i2c-parport` vous listera les adaptateurs supportés.

La première chose à faire est de charger le module permettant l'émulation de l'interface. C'est ce module qui remplacera la logique interne d'une véritable interface. Nous chargerons ce support avec une option nous permettant d'avoir un minimum d'informations intéressantes sur ce qui se passe :

```
% modprobe i2c_algo_bit i2c_debug=3 bit_test=1
% modprobe i2c_dev
```

Nous chargeons également le support `/dev` nous permettant d'accéder aux bus depuis l'espace utilisateur. Enfin, nous pouvons charger le support de l'adaptateur parallèle :

```
% modprobe i2c-parport type=0
% modprobe i2c-sis96x
```

Dès à présent, nous pouvons utiliser les outils du projet Lm_sensors pour obtenir des informations sur le ou les bus i2c en présence :

```
% i2cdetect
[...]
Installed I2C busses:
i2c-1  unknown  SiS96x SMBus adapter at 0x0c00
i2c-0  unknown  Parallel port adapter
```

Nous avons ici deux bus. Le premier (0) est notre adaptateur parallèle, le second est celui intégré à la carte mère. Notez que la machine de test dispose de trois ports parallèles, mais seul un bus est détecté, celui où se trouve l'adaptateur. Le journal d'activité du noyau nous en apprend plus :

```
kernel: i2c-algo-bit.o: (0) scl=1, sda=0
kernel: i2c-algo-bit.o: Parallel port adapter seems to be busy.
kernel: i2c-parport: Unable to register with I2C
kernel: i2c-algo-bit.o: (0) scl=1, sda=0
kernel: i2c-algo-bit.o: Parallel port adapter seems to be busy.
kernel: i2c-parport: Unable to register with I2C
kernel: i2c-algo-bit.o: (0) scl=1, sda=1
kernel: i2c-algo-bit.o: (1) scl=1, sda=0
kernel: i2c-algo-bit.o: (2) scl=1, sda=1
kernel: i2c-algo-bit.o: (3) scl=0, sda=1
kernel: i2c-algo-bit.o: (4) scl=1, sda=1
kernel: i2c-algo-bit.o: Parallel port adapter passed test.
```

Le module `i2c-algo-bit` procède à des tests sur les lignes SDA et SCL en mode lecture/écriture afin de s'assurer de la présence du circuit. Ainsi, nous pouvons utiliser librement les autres ports à notre disposition. Ce comportement est obtenu grâce au paramètre `bit_test=1` du module `i2c-algo-bit`. `i2cdetect` nous permet également de scanner un bus. Il suffit de lui passer en argument le numéro du bus :

```
% i2cdetect 0
WARNING! This program can confuse your I2C bus,
cause data loss and worse!
I will probe file /dev/i2c-0.
I will probe address range 0x03-0x77.
Continue? [Y/n]
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          XX XX XX XX XX XX XX XX XX XX XX XX
10: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX
20: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX
30: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX
40: XX XX XX XX XX XX XX XX 48 XX XX XX XX XX XX XX
50: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX
60: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX
70: XX XX XX XX XX XX XX XX
```

Notez le message d'avertissement. Cette commande comporte en effet un risque, puisque le seul moyen de trouver les composants i2c sur le bus est de leur adresser un message et d'attendre un acquittement de leur part. Ce type de manipulations, selon le modèle de composant peut soulever certains problèmes. On évitera donc d'utiliser la fonction de scan sur le bus interne d'un PC. Perturber le fonctionnement des EEPROM des mémoires serait vraiment une très mauvaise chose. Les adresses « vides » sont marquées `XX` et là où se trouve un composant son adresse est affichée. Ici, nous avons un composant à l'adresse 0x48.

Il s'agit tout simplement de l'objet de cet article, une sonde de température DS75 (figure 8). Inutile de présenter le diagramme de connexion puisqu'il suffit de connecter toutes les pattes. Les trois broches d'adresse sont reliées à la masse et le connecteur OS (la sortie du thermostat) reste non connecté.

Bien qu'il s'agisse d'un bus, notre montage relie directement la sonde à l'adaptateur (figure 9). Nous pourrons, au besoin greffer un autre composant au bus.

Notez que la longueur des fils entre l'adaptateur et la sonde peut être source de problème. Il faudra, en effet, avec de simples fils ne pas dépasser 50 cm. Au-delà, le recyclage d'un câble réseau donne de bons résultats.

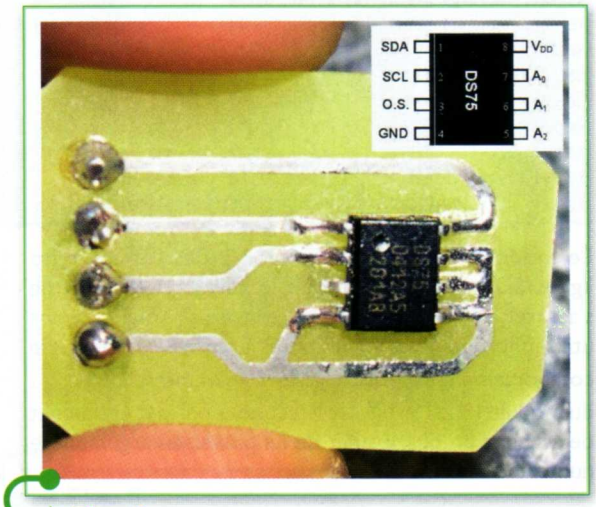


Fig. 8

PROGRAMMATION I2C

Le module i2c-dev permet l'accès aux bus depuis l'espace utilisateur. Il existe deux méthodes d'accès documentées dans le support i2c du noyau (Documentation/i2c/dev-interface).

La première consiste à utiliser les fonctions i2c_smbus_read_word_data et i2c_smbus_write_word_data décrites dans linux/i2c.h.

Il n'est normalement pas recommandé d'inclure les fichiers d'en-tête du noyau dans une application, mais c'est le seul moyen à notre disposition, puisque la Libc GNU ne connaît pas le bus i2c.

L'autre solution consiste à utiliser les opérations de lecture/écriture sur le périphérique /dev ainsi que les ioctl. La première étape consiste à ouvrir le fichier :

```
if ((file = open(filename,O_RDWR)) < 0) {
  fprintf(stderr,"Open Error : %s (%d)\n",
          strerror(errno),errno);
  exit(EXIT_FAILURE);
}
```

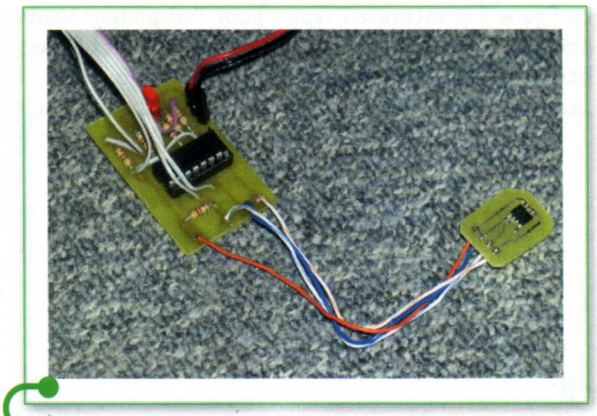


Fig. 9

On spécifie ensuite l'adresse du composant i2c esclave auquel on souhaite accéder :

```
#define I2C_SLAVE 0x0703

if (ioctl(file,I2C_SLAVE,0x48) < 0) {
  fprintf(stderr,"I2C_SLAVE Error : %s (%d)\n",
          strerror(errno),errno);
  exit(EXIT_FAILURE);
}
```

Il ne nous reste plus qu'à nous adresser au composant. La documentation du DS75 est claire, pour lire une température sur le composant, il suffit de l'adresser en lecture.

Cependant, nous préférons configurer le DS75 avant lecture. Celui-ci dispose de plusieurs résolutions entre 9 et 12 bits. Nous choisissons, bien sûr, la plus grande résolution. Pour ce faire, nous écrivons la valeur 0x60 (voir doc page 9) dans le registre 0x01 du DS75 :

```
char buf[10];
buf[0]=0x01;
buf[1]=0x60;

if (write(file,buf,2) != 2) {
  fprintf(stderr,"Write Error : %s (%d)\n",
          strerror(errno),errno);
  exit(EXIT_FAILURE);
}
```

Nous écrivons tout simplement les deux valeurs dans le fichier. Cette écriture provoque dans le DS75 le pointage du registre de configuration.

Si nous lisons le composant maintenant, nous obtiendrons la valeur du registre de configuration. Il nous faut donc, avant lecture de la température, replacer le pointeur sur le registre nous informant de la température détectée, puis lire ce registre :

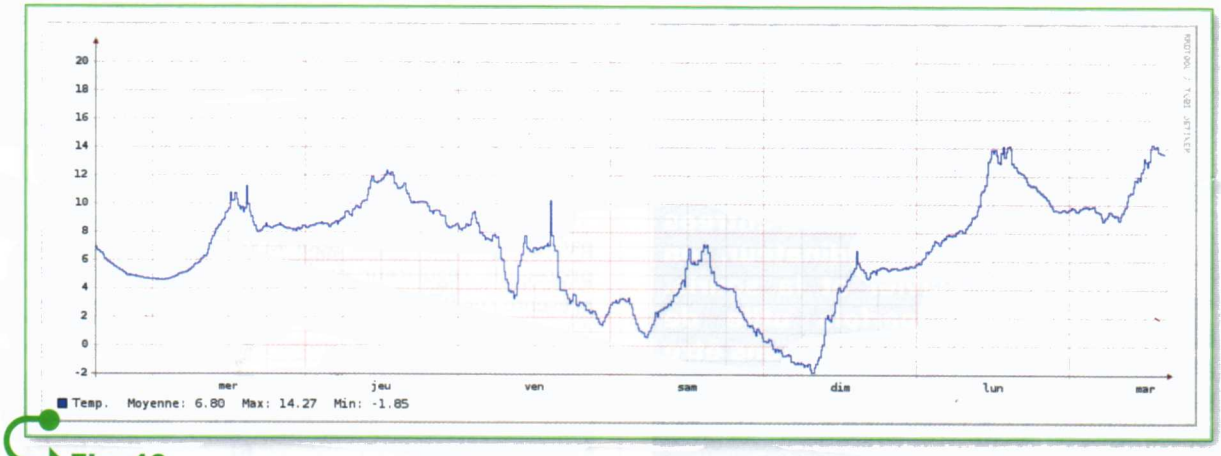


Fig. 10

```
buf[0]=0x00;

if (write(file,buf,1) != 1) {
  fprintf(stderr,"Write Error : %s (%d)\n",
          strerror(errno),errno);
  exit(EXIT_FAILURE);
}

if (read(file,buf,2) != 2) {
  fprintf(stderr,"Read Error : %s (%d)\n",
          strerror(errno),errno);
  exit(EXIT_FAILURE);
}
```

Nous obtenons dans `buf` les deux octets qui nous intéressent, le MSB dans `buf[0]` et LSB dans `buf[1]`. Nous pouvons alors décoder la température grâce à la documentation du DS75 :

Le bit 15 (bit 7 de `buf[0]`) nous indique une température négative. Les autres bits nous donnent l'exposant de la température, sa valeur entière en degrés. Nous décalons `buf[1]` de 4 bits vers la droite pour faciliter le calcul de la mantisse. Nous obtenons au final la température en degrés sous la forme d'un type `float`. Voici un code identique utilisant les fonctions de l'API i2c du noyau :

```
if (i2c_smbus_write_byte_data(file,0x01,0x60)<0)
res = i2c_smbus_read_word_data(file,0x00);
```

UTILISATION DE LA SONDE

Le code de lecture du DS75, une fois finalisé et utilisant `getopt` pour prendre en argument le fichier `/dev` et l'adresse du composant, nous permettra la relève des températures à intervalle régulier.

Tab. 2

bit	15	13	14	12	11	10	9	8
Valeur	+/-	2^6	2^5	2^4	2^3	2^2	2^1	2^0
bit	7	6	5	4	3	2	1	0
Valeur	2^-1	2^-2	2^-3	2^-4	0	0	0	0

```
int mantisse, exposant;
float deg = 0;

exposant = buf[0] & 0x7f;
mantisse = buf[1] >> 4;

if(mantisse & 1) deg+=0.0625;
if(mantisse & 2) deg+=0.125;
if(mantisse & 4) deg+=0.25;
if(mantisse & 8) deg+=0.50;
deg+=exposant;
if(buf[0] & 0x80) deg-=128;

printf("ioctl:\t%3.2f\n",deg);
```

Il nous suffira alors d'utiliser les outils RRDtool nous permettant de construire une base de données « *Round Robin* » (les plus récentes données écrasant les plus anciennes).

Ainsi, toutes les 5 minutes, via cron, nous pourrons relever la température et l'intégrer dans la base. Nous pourrons ensuite générer un graphique (figure 10) ou utiliser les données statistiques à notre guise.

Bien entendu, un résultat similaire peut être obtenu avec MRTG, mais RRDtool est plus souple d'utilisation tant en termes de stockage des données que de représentation graphique.

Denis Bodor :: db@ed-diamond.com :: lefinnois@lefinnois.net

RÉCEPTEUR INFRA-ROUGE POUR LINUX

Parmi les périphériques d'entrées classiques d'un ordinateur, on compte le clavier et la souris. Il est cependant parfois utile de pouvoir contrôler la machine sans être nécessairement à proximité. C'est précisément ce que vous permettra le simple montage suivant.

Bien que les ports séries soient en voie de disparition sur les machines modernes, la construction d'un récepteur IR (Infra-Rouge) connecté de cette manière reste la solution la plus simple.

Au besoin, il est toujours possible d'acquérir pour quelques euros une carte série sur bus PCI à intégrer dans la machine. On notera que les convertisseurs USB/série ne conviennent pas pour une telle utilisation puisque le logiciel nécessite d'avoir un contrôle particulier du port.

Une solution USB est proposée en fin d'article, mais nécessite la mise en œuvre et la programmation d'un microcontrôleur Freescale (ex-Motorola).

LE MONTAGE

Le montage est d'une extrême simplicité et repose entièrement sur un récepteur IR TSOP1738 ou similaire. Ce récepteur capte les signaux IR et les transforme en signaux TTL (0-5 volts) qui seront « remontés » à l'ordinateur après ajustement des tensions.

Le TSOP1738 est un composant très facile à trouver dans le commerce et très économique (moins de 3 euros). On trouve également le TSOP1736 qui pourra convenir pour notre montage. La seule différence entre les deux composants est la fréquence d'échantillonnage passant de 38 kHz (idéal) à 36 kHz. Cette différence de valeur influera sur la qualité de réception, mais n'empêchera pas l'ensemble de fonctionner.

Le port série utilisant des tensions qui n'ont rien à voir avec le TTL, nous ajouterons un régulateur de tension 78L05, une résistance de rappel, un condensateur et une diode de protection. L'ensemble pourra tenir sur un circuit de taille très réduite ne nécessitant pas d'alimentation externe.

Le schéma figure 1 montre l'agencement et l'utilisation des composants. L'alimentation du montage est fournie par la broche RTS (*Ready To Send*) et convertie en +5 V par le régulateur 78L05 pour le Vcc du TSOP. La diode protège le régulateur de la tension négative (-10 V) présente sur RTS. Le condensateur permet de nettoyer l'alimentation du TSOP.

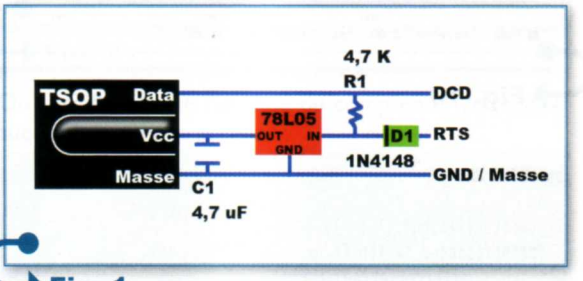


Fig. 1

C'est une habitude à prendre lorsqu'on alimente un montage ou un composant avec un régulateur. La résistance permet au port série de détecter le signal sur la broche DCD (*Data Carrier Detect*) puisque le TSOP utilise des signaux TTL normalement invisibles pour ce type de ports.

La figure 2 montre, à gauche, un assemblage sur un circuit imprimé spécifique et à droite le même montage sur plaque pastillée.

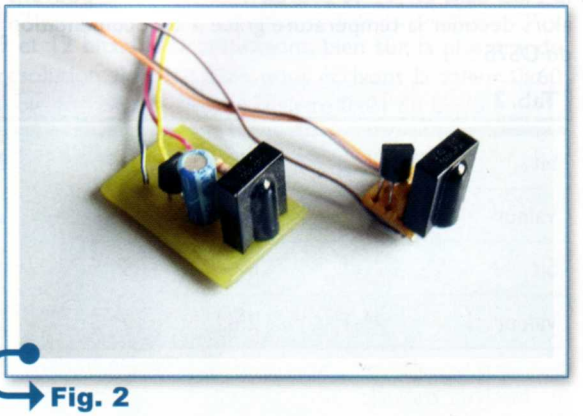


Fig. 2

LE LOGICIEL

Le support logiciel LIRC se compose de trois éléments : un module noyau, un serveur `lircd` et des applications clientes. Les tests ici réalisés portent sur un noyau 2.6.12 sur une distribution Debian stable.

Avec cette distribution, il est nécessaire de compiler et d'empaqueter les modules noyau pour LIRC. Ce ne sera pas nécessairement le cas si vous utilisez une autre distribution comme Mandriva, Fedora ou SUSE.

Il est d'ailleurs fort probable que le support LIRC existe soit de base dans la distribution, soit via l'installation d'un paquet RPM binaire.

Le récepteur présenté ici utilise le port série et celui-ci ne devra être utilisé par aucun autre élément logiciel. C'est une source courante de problème.

Lors de l'installation des sources ou du module, il vous sera sans doute demandé de spécifier le type de récepteur et l'adresse du port série à utiliser. Le tableau ci-contre donne les correspondances entre les désignations `/dev`, les dénominations DOS, les adresses et les interruptions.

NOTE

Le support du noyau 2.6 ne semble pas encore être parfait pour l'instant. Il faut, selon le problème, forcer le chargement avec l'option -f de modprobe et ainsi passer outre les messages d'erreur FATAL: Error inserting lirc_serial...Invalid module format. Les logs noyau sont plus parlants : lirc_serial: version magic '2.6.12-1-k7 K7 gcc-3.3' should be '2.6.12-1-k7 K7 gcc-4.0'. Dans tous les cas, inspectez les listes de discussions en rapport avec LIRC et/ou votre distribution le cas échéant.

Pour réserver l'usage du port série aux modules LIRC, vous devez utiliser `setserial` pour retirer totalement la prise en charge du port :

```
setserial /dev/ttyS0 uart none
```

Selon l'architecture de la distribution et sa configuration de démarrage, il est possible, voire nécessaire, de définir cela de manière permanente en modifiant le fichier `/etc/serial.conf`. Le chargement correct du module est détectable en inspectant simplement les messages du noyau (`dmesg` ou log) :

```
lirc_dev: IR Remote Control driver registered, at major 61
lirc_serial: auto-detected active low receiver
lirc_dev: lirc_register_plugin:sample_rate: 0
```

Si tout a fonctionné correctement, vous pouvez procéder aux premiers tests avec l'utilitaire `mode2` ou `xmode2`. Vous devez voir apparaître les impulsions IR envoyées par votre télécommande et réceptionnées par le montage.

L'étape suivante est la configuration du démon `lircd`. Il faut pour cela référencer votre ou vos télécommandes et les différents signaux qu'elles envoient.

Heureusement pour nous, un utilitaire spécifique a été écrit : `irrecord`. Il vous suffit de le lancer avec en argument un nom de fichier (qui sera également le nom de la télécommande dans la configuration). Rien de bien complexe ici. Il suffit de suivre les indications données à l'écran pour calibrer la réception et capturer toutes les séquences IR.

LES PORTS

ttyS0	COM1	0x3f8	4
ttyS1	COM2	0x2f8	3
ttyS2	COM3	0x3e8	4
ttyS3	COM4	0x2e8	3

Au terme de la procédure, copiez simplement le contenu de ce fichier dans votre `/etc/lirc/lircd.conf`. Le démon `lircd` peut maintenant être lancé.

Dans la plupart des cas, un script init aura été installé en même temps que les paquets LIRC et un `/etc/init.d/lirc start` suffira. Là encore, des spécificités de votre distribution devront être prises en compte pour, par exemple, lancer automatique le service au démarrage du système.

Le démon LIRC est livré avec tout un jeu d'utilitaires permettant de générer des actions en fonction des codes provenant des boutons de la télécommande. Vous pouvez ainsi injecter des évènements clavier, lancer des applications, basculer dans un mode spécifique, etc.

Le lancement du démon permettra également aux applications compatibles LIRC (xmms, MPlayer, etc.) de prendre en charge vos télécommandes.

Bien entendu, une étape de configuration sera nécessaire pour faire correspondre les boutons avec les actions dans l'application cible.

Nous ne traiterons pas cet aspect ici, car c'est un vaste domaine qui a déjà été couvert en partie par un article dans *GNU/Linux Magazine*. De plus, les documentations du projet LIRC sont largement suffisantes.

Sachez cependant qu'en configurant l'ensemble, vous pourrez tout aussi bien contrôler le système à distance avec les applications supportant LIRC qu'avec celles parfaitement standards. La combinaison de modes permet d'optimiser les quelques boutons d'une seule télécommande pour tous les usages.

USB

Attention ! Les informations qui suivent sont données à titre informatif et dans le but de vous dirigez vers le site officiel.

Nous n'avons pas eu le loisir de procéder aux tests et les indications du site sont quelque peu éparpillées entre les schémas proposés. Pour en savoir plus sur la programmation USB, reportez-vous à l'article correspondant dans le présent magazine.

Le projet USB-IR-Boy d'Aapo Tamminen et Ilkka Urtamo fait usage d'un microcontrôleur Freescale MC68HC908. Il s'agit de l'un des rares microcontrôleurs à mémoire Flash peu coûteux et capable de communiquer via USB.

Vous l'aurez compris, le montage est sensiblement plus complexe que le précédent et surtout, il faut programmer le microcontrôleur livré vierge.

Il vous faudra donc, tout d'abord, construire le programmeur pour le MC68HC908, appelé « prommer » sur le site officiel (http://usbirboy.sourceforge.net/).

Utilisant la programmation in situ, le microcontrôleur est mis en situation comme à l'intérieur du montage final à la différence près qu'il est connecté au port série du PC (et oui, un port série sera tout de même nécessaire).

Le schéma en figure 3 montre le MC68HC908 et le circuit d'interface MAX232. Le schéma nécessite un minimum de connaissances et de logique d'électronicien. Vreg, par exemple, est une tension de +3,3 V produite de manière interne par le microcontrôleur. Il faut connecter les Vregs ensembles. Le MAX232 est un classique traité par ailleurs dans ce hors-série.

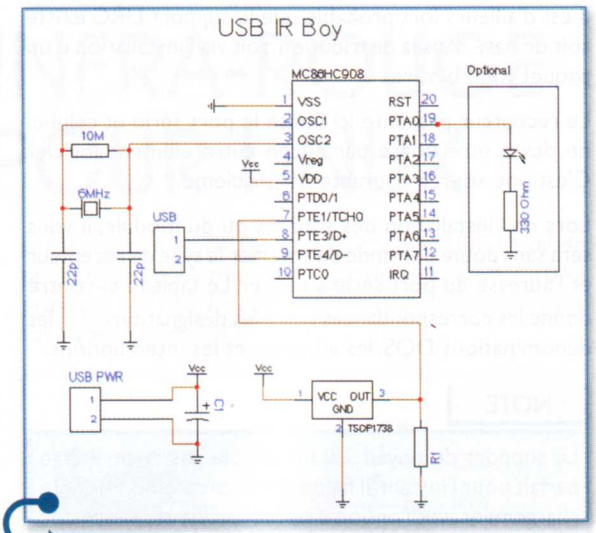


Fig. 4

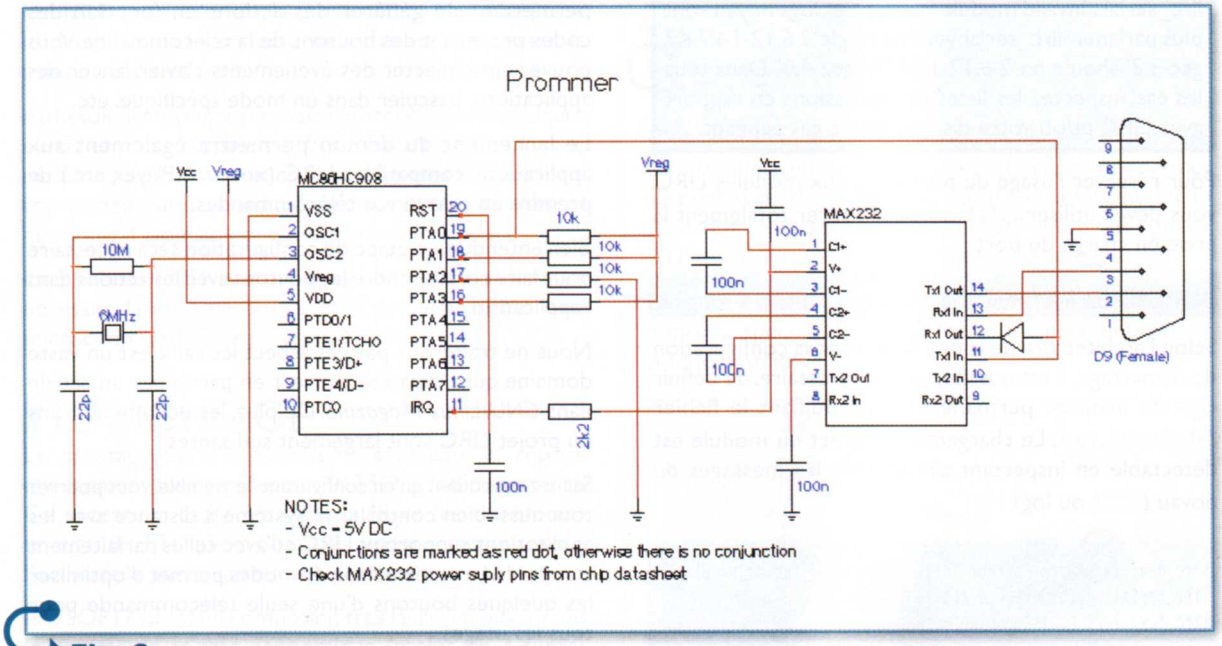


Fig. 3

Le code devant être chargé dans le MC68HC908 est écrit en C et sera compilé avec SDCC (*Small Device C Compiler*) normalement disponible sous forme de paquet dans votre distribution.

Vous pouvez également télécharger le code compilé depuis le site officiel d'USB-IR-Boy. La programmation du microcontrôleur se fera par l'utilitaire Spgmr08, également référencé sur le site du projet.

Une fois le MC68HC908 programmé, il pourra être intégré au montage dont le schéma est présenté en figure 4. Un schéma vous permettant de réaliser un récepteur en utilisant une plaque à bande est également disponible en ligne.

L'aspect logiciel comprend un module `usbirboy` ne nécessitant pas d'autre support noyau provenant du projet LIRC. La compilation du module est, à l'instar du montage, plus complexe qu'avec le récepteur sur port série. Il vous sera nécessaire de modifier le fichier `Makefile` de manière à faire correspondre les chemins d'installation des fichiers. D'autre part, si vous n'utilisez pas le support `devfs`, vous devrez créer manuellement l'entrée dans `/dev` avec la commande `mknod /dev/usbirboy c 180 240`.

Vous devrez également lancer le démon LIRC en lui spécifiant manuellement le périphérique à utiliser (`-d /dev/usbirboy`) ou modifier les fichiers d'init en conséquence. En suivant toutes les indications du site, vous devrez être en mesure d'obtenir un montage fonctionnel comme celui présenté en figure 5.

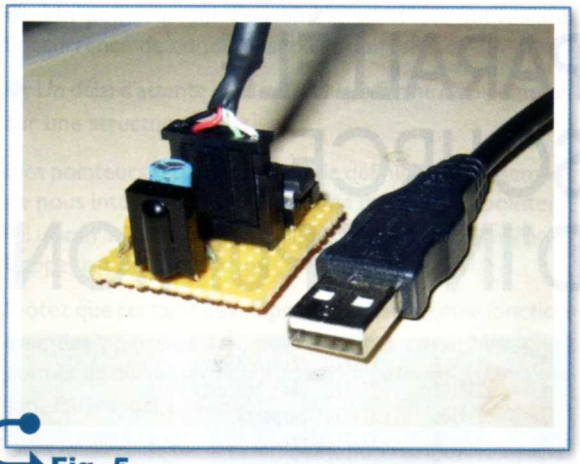


Fig. 5

Comme vous le voyez, sans être embryonnaire, le support USB pour LIRC en est à ses débuts. Le projet est en constante évolution.

Il y a quelques mois encore, les sources du code pour le microcontrôleur n'étaient pas disponibles ou uniquement compilables par Metrowerks CodeWarrior.

Espérons que l'intégration dans le projet LIRC sera plus affirmée dans les prochains temps et, pourquoi pas, que cette solution finira par remplacer celle du port série en tant que récepteur par défaut.

NOTE

Ce sujet a fait l'objet d'un article différent dans GNU/Linux Magazine France 72. Le sujet se devait d'être traité dans ce hors-série, mais de manière plus succincte pour limiter la redondance.

Denis Bodor :: db@ed-diamond.com :: lefinnois@lefinnois.net

LE PORT PARALLÈLE SOURCE D'INTERRUPTION

Lorsqu'on regarde les spécifications de matériel pour système embarqué, on cherche systématiquement le nombre de sources d'interruption. Le PC, pour sa part, dispose d'un port capable de gérer cela très facilement grâce à une couche logiciel fort sympathique.

Comment se tenir au courant de ce qui se passe ? Oubliez télévision, journal et autre, je vous parle d'architecture système. Les réponses sont au nombre de deux. Vous pouvez inspecter régulièrement l'objet de votre attention afin d'y déceler un changement éventuel.

C'est là une méthode très courante lorsqu'on programme des microcontrôleurs. N'étant pas multitâches, ces composants n'ont pas à se soucier du fonctionnement d'autres processus et peuvent, à loisir, consommer 100% des ressources dans une boucle d'attente.

Un 16F628, par exemple, sera en mesure d'inspecter les lignes d'entrée quelques millions de fois par seconde. Il n'en va pas de même avec un système UNIX. De nombreuses tâches fonctionnent en mémoire et une boucle infinie dans un programme disposant de certains privilèges poserait un réel problème.

En utilisant les fonctions `sleep()` ou `usleep()`, on laisse une chance au système de réguler la charge et on obtient un code viable.

Lorsqu'on est en attente d'un évènement ou d'une information provenant d'une ressource matérielle, il est parfois possible de profiter du mécanisme d'interruption. Grossièrement, un périphérique interrompt le processeur pour lui signaler un évènement.

Le système d'exploitation est au fait de cette interruption et l'évènement peut remonter jusque dans l'espace utilisateur comme c'est le cas avec le noyau Linux et le support `ppdev`.

Le nombre d'interruptions matérielles est limité sur une architecture PC. Un coup d'œil au fichier `/proc/interrupts` et vous saurez quel périphérique utilise quelle interruption pour requérir l'attention du processeur :

```
           CPU0
  0:  887988708      XT-PIC  timer
  1:     280990      XT-PIC  i8042
  2:          0      XT-PIC  cascade
  3:          0      XT-PIC  ohci_hcd:usb1
  5:     330181      XT-PIC  ohci_hcd:usb2
  7:     994386      XT-PIC  parport0
  8:          4      XT-PIC  rtc
  9:          0      XT-PIC  acpi
 10:   16016510      XT-PIC  es1371, eth0
 11:   53288330      XT-PIC  radeon@pci:0000:01:00.0
 12:    1757406      XT-PIC  i8042
 14:    1443358      XT-PIC  ide0
 15:     228990      XT-PIC  ide1
NMI:          0
LOC:  888016688
ERR:         57
MIS:          0
```

Ici, on constate que l'interruption matérielle 7 est utilisée par `parport0`. Notez toutefois que la machine de test ici utilisée dispose de trois ports parallèles, mais que seul celui intégré à la carte mère utilise une interruption. La carte PCI additionnelle basée sur un contrôleur Timedia Technology Ltd est accessible via `parport1` et `parport2`, mais ne permet pas les manipulations qui vont suivre.

GESTION DES INTERRUPTIONS

`ppdev` permet à une application de l'espace utilisateur d'attendre une interruption sur un port parallèle. Ceci est très intéressant dans le sens où nous n'avons pas besoin de développer un module noyau à cet effet, chose relativement rare lorsque l'on touche à du matériel.

Cette prise en charge est accessible via la fonction de multiplexage d'entrées/sorties synchrones `select`. Celle-ci attend un changement d'état sur les descripteur de fichier (simplement celui obtenu lors de l'ouverture du pseudo fichier `/dev/parport0` ici).

`select` prend en arguments plusieurs éléments :

- Un entier spécifiant la valeur la plus importante des descripteurs surveillés ;
- Un pointeur vers un ensemble `readfds` permettant de surveiller la disponibilité de caractères en lecture ;
- Un pointeur sur un ensemble `writefds` permettant de connaître la possibilité d'écriture immédiate via le descripteur ;

- Un pointeur sur l'ensemble `exceptfds` surveillant l'occurrence de conditions exceptionnelles ;
- Un délai d'attente maximale sous la forme d'un pointeur sur une structure `timeval`.

Des pointeurs `NULL` permettent de définir qu'un ensemble ne nous intéresse pas. De la même manière, un pointeur `NULL` pour le délai d'attente signifie une attente indéfinie. L'appel est donc bloquant.

Notez que certains développeurs utilisent cette fonction avec des pointeurs `NULL` pour les trois ensembles. Ceci permet de définir un délai d'attente plus fin que la seconde sans faire appel à `usleep()`.

Dans le cadre de cet article, l'utilisation réputée complexe de `select` est assez aisée. Nous n'utilisons, en effet, qu'un seul descripteur de fichier et ce, uniquement en lecture. De plus, un certain nombre de macros facilitent la tâche du développeur pour créer les ensembles :

```
fd_set rfds;

FD_ZERO (&rfds);
FD_SET (fd, &rfds);

if (select (fd+1, &rfds, NULL, NULL, NULL) <= 0) {
  fprintf(stderr,"Select Error : %s (%d)\n",
          strerror(errno),errno);
} else {
  printf("Interruption !\n");
}
```

`FD_ZERO` initialise à zéro un ensemble. `FD_SET` permet d'ajouter un descripteur dans l'ensemble. Suit l'appel à `select` où nous n'avons qu'à utiliser notre seul descripteur plus 1 et spécifier l'ensemble `readfds`.

Les deux autres ensembles ne nous intéressent pas, pas plus que le délai d'attente. `select` retourne une valeur négative en cas de problème, 0 en cas de dépassement du délai d'attente ou une autre valeur positive indiquant le nombre de descripteurs dans les ensembles. Dans ce cas précis, nous savons que le port parallèle a provoqué une interruption.

Le code dans son ensemble est ensuite fort simple (voir article sur le port parallèle) :

- On ouvre le fichier `/dev/parport0`.
- On obtient l'accès au port.
- On lance notre `select`.
- On lit la ligne ACK.
- Enfin, avant de rendre la main, on utilise l'`ioctl PPCLRIRQ` pour remettre le compteur d'interruptions à zéro et relever le compteur :

```
int irqc;

if (ioctl (fd, PPCLRIRQ, &irqc) < 0) {
  fprintf(stderr,"PPCLRIRQ ioctl Error : %s (%d)\n",
          strerror(errno),errno);
  exit(EXIT_FAILURE);
} else {
  fprintf (stderr, "%d int ratée(s)\n", irqc-1);
}
```

Nous affichons le nombre d'interruptions moins une, celle que nous venons de détecter. Cela peut paraître surprenant, mais vous allez comprendre. Pour un premier test, nous connectons simplement un bouton poussoir entre la masse et la ligne `ACK` (figure 1, la partie bleue est optionnelle) et lançons le binaire obtenu après compilation :

```
% ./irq
[ PRESSION SUR LE BOUTON ]
Interruption !
ACK : 1
3 int ratée(s)
```

Si nous ajoutons une pause en sortie de la condition `select` avec la fonction `sleep(1)`, c'est encore pire :

```
% ./irq
[ PRESSION SUR LE BOUTON ]
Interruption !
ACK : 1
46 int ratée(s)
```

Pourtant, nous n'avons appliqué qu'une seule pression au bouton poussoir. Que s'est-il passé ? Vous venez de faire connaissance avec l'un des problèmes majeurs lorsque la mécanique rencontre l'électronique.

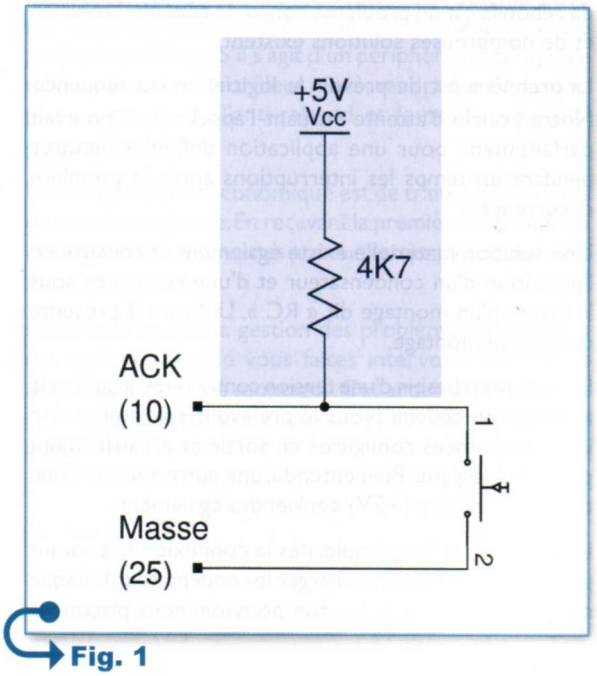


Fig. 1

MAUDIS REBONDS !

Le bouton poussoir utilisé pour les essais, tout comme un relais ou un ILS (interrupteur magnétique) est un élément mécanique. Il s'agit de contacteurs au-dessus desquels repose une lamelle métallique parfois maintenue par un ressort.

En appuyant sur le bouton, la lamelle met en contact les contacteurs et la connexion est établie. Mais la connexion n'est jamais franche à l'échelle électronique. La lamelle rebondie sur les contacteurs générant non pas un contact, mais une série de connexions/déconnexions.

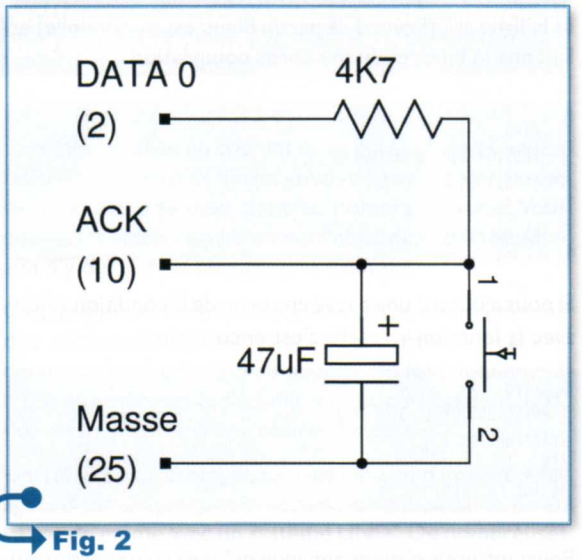


Fig. 2

Voilà pourquoi, en ajoutant un délai d'une seconde entre le traitement de l'interruption et la relève du compteur, on arrive à rater autant d'évènements. Ce phénomène de rebonds est un problème connu de tout électronicien et de nombreuses solutions existent.

La première est de prévoir le logiciel en conséquence. Notre boucle d'attente incluant l'appel `select` pourrait parfaitement, pour une application définitive, ignorer pendant un temps les interruptions après la première occurrence.

Une solution matérielle existe également et consiste en l'utilisation d'un condensateur et d'une résistance sous la forme d'un montage dit « RC ». La figure 2 présente ce type de montage.

Ici nous avons besoin d'une tension contrairement au simple montage précédent. Nous le prélevons sur la première ligne de données configurée en sortie et préalablement placée à 1 logique. Bien entendu, une autre source d'une tension identique (+5 V) conviendra également.

Le principe est fort simple, dès la connexion le courant provenant de DATA 0 va charger le condensateur. Lorsque nous appuyons sur le bouton poussoir, nous plaçons la ligne ACK au niveau bas et vidons le condensateur de manière instantanée.

En relâchant le bouton poussoir, ACK ne revient pas immédiatement au niveau haut mais progressivement, en trois fois R fois C (R en Ohm et C en Farad) secondes (ici trois fois 4700 Ohms fois 0.000047 Farads, soit 0.66 secondes). Les rebonds seront compensés automatiquement puisqu'une fois ACK à la masse, d'éventuelles déconnexions et reconnexions des contacteurs ne suffiront pas à la charge du condensateur et donc au retour au 1 logique.

Cette solution, bien que fonctionnelle, n'est pas satisfaisante. Toutefois, elle présente un aspect pédagogique certain. Il s'avère, sur la machine de test, que l'interruption n'est pas générée par le passage du niveau haut au niveau bas sur la ligne ACK, mais l'inverse.

En d'autres termes, il ne se passe rien lorsqu'on appuie sur le bouton, mais l'interruption est déclenchée lorsqu'on le relâche. Or, cette transition est progressive et problématique. On dit que le signal est détecté sur le front montant.

Les niveaux TTL utilisés par le port définissent un signal au niveau haut comme valide dès +2 V et le calcul du temps correspond à la charge complète du condensateur. Le signal passe donc à un biveau TTL valide avant le temps calculé.

De plus, nous n'avons pas connaissance de ce qui se trouve à l'intérieur du port parallèle. En effet, une résistance de *pull-up* semble bien présente, mais nous ne connaissons pas sa valeur et celle-ci influe également sur le montage.

Au final, le montage fonctionne très bien mais... à l'envers. Un bouton poussoir établissant la connexion par défaut et déconnectant lors de la pression règle une partie du problème, mais pas le temps d'attente avec l'interruption.

La figure 3 propose une autre solution. On utilise une résistance de faible valeur entre la ligne ACK et la masse. Suffisamment faible pour mettre la ligne à l'état bas.

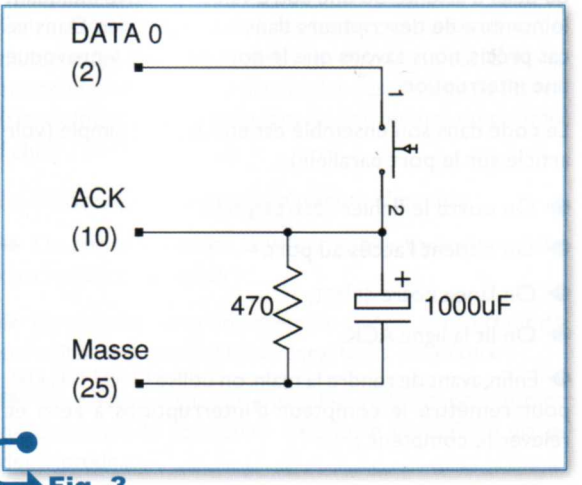


Fig. 3

Ici, les essais ont montré que 470 Ohms étaient suffisants mais, là encore, tout dépend de la résistance de pull-up interne du port et du courant (infime) en présence. Le bouton poussoir mettra la ligne à l'état haut et chargera le condensateur. Ici, l'interruption est immédiatement obtenue, nous passons de l'état bas à haut.

En raison de la faible valeur de la résistance, nous devons utiliser un condensateur de forte capacité pour revenir à l'état bas progressivement et ainsi annuler l'effet des rebonds.

Là encore, cela fonctionne mais n'est pas idéal. Les amateurs d'électronique feraient les gros yeux en voyant un condensateur de 1000 µF utilisé pour un simple anti-rebond.

De plus, ce que nous avons construit ici est également un diviseur de tension en raison de la présence de la résistance de pull-up. Rien de suffisant pour perturber le fonctionnement TTL des signaux, mais tout de même gênant.

La solution tient dans l'utilisation de la première solution et d'un *trigger* de Schmitt (circuit 4093 par exemple). Je ne rentrerai pas dans le détail du fonctionnement du composant, des sites Internet spécialisés en électronique le font très bien.

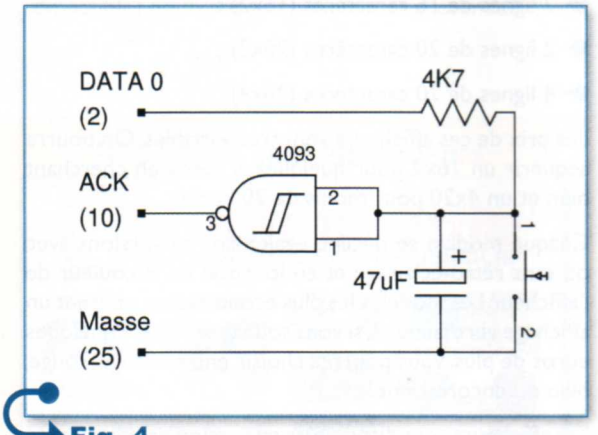


Fig. 4

Le trigger permet grossièrement, lorsqu'on lui applique un signal variable, de passer à l'état haut à une tension donnée, mais de revenir à un état bas en dessous de cette tension.

Ainsi on obtient une marge de fonctionnement très intéressante. En appliquant en entrée un signal sinusoïdal, on obtient, en sortie, un signal carré. Comble du raffinement, la sortie d'un trigger de Schmitt est généralement inversée.

La figure 4 montre le schéma en figure 2 avec l'une des quatre portes NAND (NON ET) à trigger d'un circuit 4093. Nous relions les deux entrées ensemble et connectons la sortie au signal `ACK`. Que se passe-t-il à présent ?

A la connexion, nous avons un niveau haut en entrée de la porte NAND et donc un niveau bas en sortie. Si nous appuyons sur le bouton poussoir, le niveau passe subitement à 0 en entrée et instantanément à 1 en sortie. L'interruption est déclenchée.

En relâchant le bouton, le condensateur et la résistance font doucement revenir la tension en entrée de la porte NAND au niveau haut.

C'est là que le trigger agit en « mettant au carré » le signal et en basculant `ACK` au niveau bas de manière abrupte, mais au bout d'un temps seulement, et en effaçant d'éventuelles perturbations.

Le troisième essai est donc le bon. Mais le montage doit être alimenté par une source externe de courant. Les 4093 sont vendus pour quelques dizaines de centimes d'euros sous la forme de circuit en boîtier DIP 14.

n seul 4093 intègre 4 portes NAND à trigger de Schmitt et doit être alimenté. D'autres circuits logiques peuvent convenir comme le 74LS14 (6 inverseurs à trigger de Schmitt), le 40106 (idem) ou le 74HC132 (similaire au 4093).

CONCLUSION

Cela fait beaucoup de chose pour un simple bouton, mais il faut bien comprendre que le port parallèle n'est pas conçu à l'origine pour ce type d'utilisation. Il attend et émet des signaux logiques loin des aléas mécaniques que nous venons de voir.

L'imprimante connectée au port parallèle intègre toute une logique interne et, par exemple, le contacteur chargé de détecter la présence du papier n'est jamais directement relié à la ligne du même nom adéquat.

Bien entendu, la manière de régler le problème posé par la partie mécanique dépend entièrement de votre projet de montage. S'il s'agit d'un périphérique complexe utilisant un microcontrôleur, mieux vaudra que ce dernier prenne en charge l'anti-rebond et fournisse un signal propre sur `ACK`.

La solution la plus économique est de traiter les rebonds de manière logicielle. En recevant la première interruption et en ignorant les suivantes pour un temps puis en appelant `select` à nouveau.

Dans tous les cas, la gestion des problèmes de rebonds est capitale lorsque vous faites intervenir une partie mécanique dans vos montages. Cet article vous a donné l'occasion d'entrevoir quelques solutions directement applicables.

Denis Bodor :: db@ed-diamond.com :: lefinnois@lefinnois.net

CONNECTEZ UN AFFICHEUR LCD

Il n'est pas rare de voir aujourd'hui bon nombre de PC équipés d'afficheurs en tout genre. La plupart du temps, ceux-ci ne sont pas pilotés par le système et n'indiquent que température et vitesse des ventilateurs. Le montage suivant est totalement différent.

Qu'il s'agisse d'un PC de bureau ou d'un serveur, l'affichage d'informations sur un écran déporté en façade est une excellente solution pour être informé de l'état du système. Un projet a vu le jour il y a fort longtemps sous le nom de « LCDproc » et continue à être maintenu aujourd'hui. Les mises à jour se font rares car le projet a atteint une grande maturité. LCDproc est un projet de développement d'un démon fonctionnant sous système UNIX (GNU/Linux, BSD, etc.) permettant de piloter un afficheur à cristaux liquides alphanumériques. Il se constitue du démon lui-même pilotant différents modèles d'afficheurs via des pilotes/greffons spécifiques. Un utilitaire client permet d'afficher quelques informations de base concernant le système comme la charge CPU, la mémoire disponible, etc. Dernière fonctionnalité intéressante, il est capable de fonctionner localement, mais également via le réseau.

Ainsi, la machine où est connectée l'afficheur n'est pas nécessairement celle qui affiche les informations. Il est même possible de connecter plusieurs clients à un seul serveur d'affichage et ainsi partager la ressource.

MATÉRIEL

Le montage ne présente pas de difficulté particulière. Nous opterons ici pour la solution la plus simple et la moins coûteuse consistant à utiliser un afficheur LCD basé sur le contrôleur HD44780 ou compatible connecté au port parallèle du PC.

Notez cependant qu'il existe d'autres solutions prises en charge par les pilotes LCDproc, comme la connexion USB ou série. Bien qu'il soit fort probable que vous souhaitiez utiliser un véritable montage, sachez qu'il est également possible d'utiliser une émulation basée sur ncurses pour un affichage en mode texte dans un terminal ou une console. Tout le montage se base sur l'afficheur lui-même qui embarque toute la logique d'affichage. La figure 1 montre la connexion d'un afficheur avec le port parallèle. Il existe plusieurs types d'afficheurs LCD compatibles HD44780 se différenciant par leur nombre de lignes et de caractères par ligne :

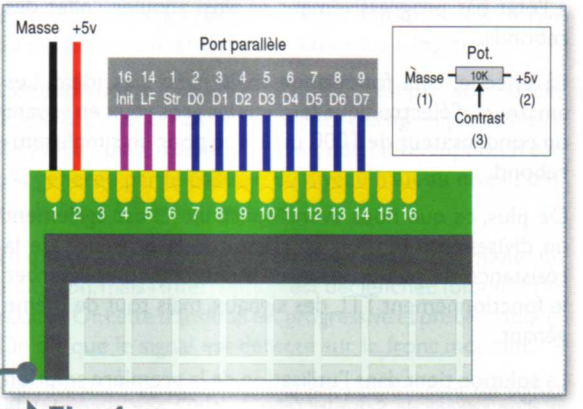


Fig. 1

- 1 ligne de 16 caractères (16x1) ;
- 1 ligne de 20 caractères (20x1) ;
- 2 lignes de 16 caractères (16x2) ;
- 2 lignes de 20 caractères (20x2) ;
- 4 lignes de 20 caractères (20x4).

Les prix de ces afficheurs sont très variables. On pourra acquérir un 16x2 pour quelques 6 euros en cherchant bien et un 4x20 pour moins de 20 euros.

Chaque modèle se décline également en versions avec ou sans rétro-éclairage et en fonction de la couleur de l'afficheur. Les modèles les plus économiques utilisent un affichage vert/jaune et, si vous souhaitez mettre quelques euros de plus, vous pourrez choisir entre orange, rouge, bleu ou encore blanc.

Les afficheurs sont disponibles en version positive (texte noir sur fond de couleur) ou négative (texte de couleur sur fond noir).

Une gamme qui devrait satisfaire n'importe quel amateur de JackyPC exigeant. Comme le montrent les schémas en figure 1, l'afficheur est entièrement piloté par les huit lignes de données du port parallèle et trois des cinq lignes de contrôle. Le module d'affichage est normalement alimenté en +5 V, mais vérifiez la documentation avant toute connexion.

Le réglage du contraste se fait via un potentiomètre de 10 K Ohms comme spécifié sur le schéma. Vous pouvez le constater, la réalisation est très simple. Veillez simplement à bien réguler l'alimentation en ajoutant, au besoin, quelques condensateurs et à ne pas dépasser une trop grande distance pour la connexion au port parallèle.

LOGICIEL

Quelle que soit la distribution utilisée, les outils LCDproc sont normalement présents sous forme de paquets dont l'installation sera très simple. Il vous suffira de renseigner quelques points dans le fichier `/etc/LCDd.conf` concernant le type d'afficheur et le mode de connexion. Ici :

```
Driver=HD44780

[HD44780]
Port=0x378
ConnectionType=winamp
Keypad=no
Backlight=no
Size=20x4
DelayBus=true
```

Le schéma de connexion fourni est celui référencé sous la désignation `winamp`. Il en existe d'autres que vous pourrez consulter sur le site officiel (http://lcdproc.omnipotent.net/) à la section « Hardware ».

Une fois le fichier correctement renseigné, il ne vous restera plus qu'à lancer le démon via le script d'init fourni avec le paquet. Certaines distributions nécessiteront des ajustements spécifiques pour lancer le service dès le démarrage du système. Le démon lancé, tous les clients compatibles pourront s'y connecter et piloter l'afficheur.

Un premier test se résumera par le lancement de l'application cliente par défaut : `lcdproc`. Un certain nombre d'informations s'afficheront alors sur le module.

NOTE

Avec la distribution Debian, le binaire client est `/usr/share/doc/lcdproc/examples/lcdproc.gz`. Il faut le décompresser et l'exécuter localement ou le placer dans `/usr/local/bin`.

CONTRÔLE DE L'AFFICHEUR

Comme dit plus haut, le démon fonctionne en réseau. Par défaut, celui-ci écoute sur l'interface du localhost et le port 13666. Le protocole permettant de piloter l'afficheur prend la forme d'un échange de données textuelles. Ainsi, il est possible d'afficher ce qui nous chante à l'aide d'un simple client Telnet.

Le dialogue avec le démon est relativement simple et suit la procédure suivante :

- Dire « bonjour » ;
- Définir un ou plusieurs nouveaux écrans et leurs *widgets* ;
- Couper la connexion pour arrêter l'affichage des données.

EN KIOSQUE LE 4 NOVEMBRE 2005

Voici un exemple (les > et < ne sont là que pour indiquer la provenance des messages) :

```
% telnet 127.0.0.1 13666
> hello
< connect LCDproc 0.4.5 protocol 0.3
    lcd wid 20 hgt 4 cellwid 5 cellhgt 8
> screen_add monecran
< success
< listen monecran
> widget_add monecran letitre title
< success
> widget_set monecran letitre "Voila un beau titre"
< success
> widget_add monecran lecoucou string
< success
> widget_set monecran lecoucou 1 2 "tralala le texte"
< success
> widget_add monecran barre hbar
< success
> widget_set monecran barre 1 4 74
< success
> screen_set monecran -priority 128
< success
```

Voici la syntaxe des commandes qui viennent d'être utilisées :

- `screen_add` nom_de_l_écran
- `widget_add` nom_de_l_écran nom_du_widget type
- `widget_set` nom_de_l_écran nom_du_widget données

Et la syntaxe des données pour les widgets utilisés :

- `title` : «chaîne de titre»
- `string` : x y «chaîne à afficher»
- `hbar` : x y taille_en_pixels

Les informations utilisées par les widgets comme le placement x et y ainsi que la taille dépendent des informations retournées après la directive `hello` :

- `wid 20` : Afficheur de 20 caractères (cellules) par ligne
- `hgt 4` : Afficheur de 4 lignes
- `cellwid 5` : Une cellule fait 5 pixels de large
- `cellhgt 8` : Une cellule fait 8 pixels de haut

Référez-vous à la page de manuel de `LCDd` pour connaître les directives et les widgets utilisables ainsi que la syntaxe complète. La communication via Telnet permet de faire des essais, mais l'écriture du client final se fera de préférence en Perl. Un exemple complet est disponible sous le nom `lcdmetar.pl` et fait appel au module `IO::Socket` relativement simple d'usage. N'oubliez pas, le client doit rester connecté pour que l'écran reste affiché. Il faut donc prévoir que votre application boucle et soit à la fois en attente des informations à afficher et en liaison avec le démon LCDproc. L'usage de `cron` reste possible pour les mises à jour des écrans, mais s'avère souvent être trop contraignant.

PROCMETER 3

En plus du binaire client `lcdproc`, il existe un utilitaire capable d'afficher un certain nombre d'informations sur l'écran LCD. ProcMeter, c'est son nom, est généralement

disponible pour toutes les distributions. Son aspect en tant qu'application graphique fait qu'on lui préfère souvent Gkrellm. Cependant son architecture modulaire est sa compatibilité avec LCDproc en font un outil de choix ici. ProcMeter utilise une syntaxe très particulière sous la forme d'une énumération d'informations à afficher : « module.info-format ». En appelant un des binaires, comme `procmeter3-lcd` avec l'option `-h`, vous verrez apparaître l'ensemble des formations disponibles :

```
[...]
Memory
------
Mem_Free   (GTB) : The amount of memory that is free,
                   completely unused, wasted.
Mem_Used   (GTB) : The amount of memory that is used,
                   excluding cache and buffers.
[...]
```

Ici, pour afficher la quantité de mémoire libre, on utilisera `Memory.Mem_Free-t` par exemple. Les lettres définissant le format sont `t` pour texte, `g` pour un graphe et `b` pour une barre. La sortie fournie par l'option `-h` permet, pour chaque information de chaque module, de connaître les formats utilisables (ici, les trois). Pour spécifier les informations à afficher, trois solutions s'offrent à vous. En argument sur la ligne de commande, dans le fichier `/etc/procmeterrc` ou dans le fichier personnel `~/.procmeterrc` (ou autre spécifié par l'option `-rc=`).

CONCLUSION

Le support des afficheurs LCD pour GNU/Linux est à la fois très avancé et très stable. On regrettera l'absence d'équivalence pour les afficheurs graphiques qui, comme les modèles alphanumériques, sont de moins en moins chers. Enfin, notez qu'il existe également un support noyau pour les afficheurs HD44780 et compatibles. Rendez-vous sur **http://lcd-mod.sourceforge.net/** pour plus d'informations.

NOTE

Ce sujet a fait l'objet d'un article très différent dans GNU/Linux Magazine France 60. Le sujet se devait d'être traité dans ce hors-série, mais d'une autre manière pour éviter toute redondance.

Denis Bodor :: db@ed-diamond.com :: lefinnois@lefinnois.net

PILOTEZ DES RELAIS DEPUIS LINUX

Lorsqu'il s'agit de contrôler des équipements haute tension depuis un PC, il existe plusieurs solutions. La plus courante fait appel aux relais, des éléments électromécaniques garantissant une isolation indispensable.

L'article d'introduction à l'électronique présente une utilisation des transistors permettant le contrôle de tensions et de courants importants depuis des lignes TTL comme celles du port parallèle. Lorsqu'il s'agit de faire de même avec un courant alternatif plus important et des tensions mortelles, le transistor montre ses limites.

Les prises qui remplissent nos bureaux, nos appartements et nos maisons fournissent une tension alternative de 230 Volts et ce avec parfois de forts ampérages. Pour piloter de telles sources de courant, il faut utiliser un composant électromécanique appelé "relais". D'autres solutions existent comme le Mosfet, thyristor, triac, etc. Mais ce n'est pas le sujet de cet article.

Il existe des centaines sinon des milliers de relais différents. Quelques modèles sont présentés en figure 1. Cependant, un relais, quel qu'il soit, repose toujours sur le même principe. Un bobinage alimenté en courant constitue un électroaimant. Alimenté, celui-ci attire un système mécanique mettant en contact des connecteurs et en détachant d'autres. Il s'agit, ni plus ni moins, d'un interrupteur contrôlé par un électroaimant.

Cela peut paraître archaïque, mais le relais dispose d'arguments de poids. Parmi les plus intéressants, nous avons celui de l'isolation électrique. Il est quasi impossible que le courant piloté puisse détruire l'électronique de contrôle. Voilà pourquoi les relais ont encore de beaux jours devant eux dans le monde de la domotique et du contrôle d'équipement 230 Volts. Je ne parle même pas du mélodieux cliquetis qu'ils génèrent...

Un relais se caractérise en fonction trois principaux éléments :

- Le pouvoir de coupure : C'est la tension et l'intensité du courant maximales que le relais peut couper. Au-delà, un arc électrique peut se former entre les contacteurs. L'isolation n'est plus garantie, ni la coupure du circuit. La figure 2 montre un relais miniature spécifiant une tension de coupure alternative de 125 V et une intensité maximale de 1A. La figure 3 présente un relais affichant clairement des tensions de coupure différentes pour le courant alternatif (VAC) et continu (VDC).

- Le U bobine : C'est la tension à appliquer aux bornes du relais pour qu'il fonctionne. A cette tension, le bobinage interne devient magnétique et commute le ou les contacteurs. Il ne faut, bien sûr, pas dépasser cette tension au risque de faire fondre le bobinage et de détruire le composant. Il existe plusieurs types de relais, les plus intéressants pour nous possèdent un U bobine de 5 V ou 12 V (Le relais en figure 3 précise clairement « Coil » alors que celui en figure 4 indique uniquement « 12VDC », mais la confusion n'est pas vraiment possible.

- Le R bobine : Un bobinage présente une résistance qui nous permet de calculer le courant consommé par celui-ci. Cette valeur contrairement aux deux précédentes est

Figure 1

Figure 2

Figure 3

Figure 4

rarement spécifiée sur le composant. Elle est donnée à l'achat par le catalogue du détaillant ou dans la documentation constructeur, mais peut également être, tout simplement mesurée au multimètre.

RÈGLES DE BASE ET CALCULS

Vous ne pouvez pas, à l'aide de signaux TTL, piloter directement un relais, celui-ci consomme bien trop de courant. On utilisera un transistor pour piloter l'alimentation du bobinage. Bien entendu, qui dit « transistor en commutation » dit « calculs ». Je vous conseille de lire l'article d'introduction à l'électronique si ce n'est déjà fait. Allons-y ! Le montage est donné en figure 5 et le transistor est un *NPN BC547B (ß = 200, Vce_sat = 0.2V, Vbe_sat = 0.7V)*. Ici, inutile d'ajouter une résistance sur le collecteur du transistor, la bobine du relais fait office de résistance. La première chose à calculer est le courant Ic. La loi d'Ohm est là pour cela. Notre U est notre Vcc moins la tension entre le collecteur et l'émetteur avec le transistor en saturation. R est R bobine :

Ic = (Vcc - Vce_sat)/R
Ic = (12 - 0.2)/R
Ic = 0.045

Calculons maintenant le courant à fournir à la base du transistor pour qu'il sature :

Ib_min = Ic/ß
Ib_min = 0.045 / 200
Ib_min = 0.000225

*Ib_sat = Ib_min * 1.5*
*Ib_sat = 0.000225 * 1.5*
Ib_sat = 0.0003375

Enfin, nous avons notre Ib de saturation du transistor, il ne nous reste plus qu'à calculer la résistance à utiliser sur la base du transistor pour pouvoir piloter l'ensemble depuis, par exemple, un port parallèle :

R = (Ve - Vbe_sat)/Ib_sat
R = (5 - 0.7) / 0.0003375
R = 12740 Ohms

La résistance immédiatement inférieure dans la série E12 est de 12 K Ohms. Nos calculs sont terminés. Notez la présence indispensable d'une diode (1N4004 par exemple). La bobine d'un relais possède un certain nombre de propriétés dont l'inductance. Une fois sous tension, le courant ne s'établit pas immédiatement dans la bobine. Plus dangereux encore, la coupure du courant entraîne une surtension importante qui pourrait détruire le transistor (loi de Lenz). Pour protéger le transistor de cette surtension, la solution consiste à disposer une diode en parallèle sur la bobine. Il arrive que cette diode soit intégrée au relais. Sa connexion devra alors se faire en respectent la polarité. Dans la plupart des cas, le relais ne dispose pas de cette protection et celui-ci pourra être utilisé sans polarité particulière.

CONCLUSION

Un relais neuf avec une tension de coupure de 250VAC entre 1A et 6A vous coûtera environ 5 euros pièce. Il est possible de récupérer ce type de composants dans des rebuts électriques (minuterie d'éclairage, etc.) ou tout simplement, à moindre coût sur des sites d'enchères en ligne. Dans tous les cas, il vous faudra refaire les calculs donnés en exemple dans cet article en fonction du modèle de relais et des transistors utilisés.

Enfin, je conclurais en précisant que la création de montages utilisant des tensions élevées présente un réel danger. Les 230 Volts alternatifs normalisés en France sont mortels. Les risques d'incendies sont également à prendre sérieusement en considération. N'hésitez pas à demander de l'aide dans les groupes USENET comme **fr.sci.electronique**. Les débutants sont généralement très bien accueillis s'ils respectent la netiquette et font preuve de politesse et de courtoisie.

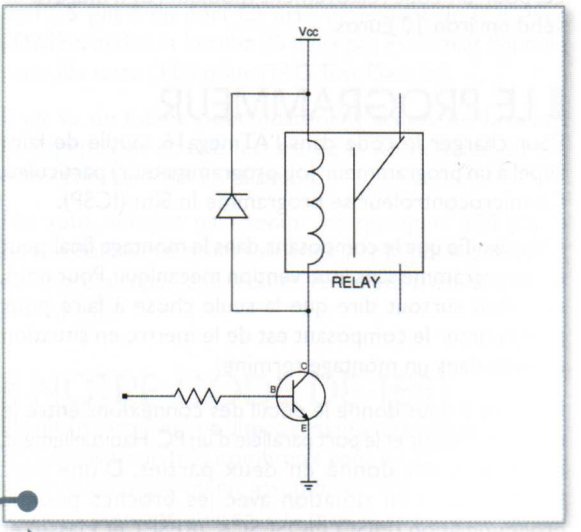


Figure 5

Denis Bodor :: db@ed-diamond.com :: lefinnois@lefinnois.net

DÉCOUVERTE DU MICROCONTRÔLEUR AVR

Un précédent article paru dans GNU/Linux Magazine vous présentait les microcontrôleurs Microchip PIC. Une autre famille de ces composants est très populaire et bénéficie d'un très bon support sous GNU/Linux : les AVR de chez Atmel.

La famille de microcontrôleurs AVR est très vaste à l'instar de la gamme PIC de Microchip et s'étend depuis le minuscule ATtiny15L et le monstrueux ATmega256. Pour cette introduction, nous avons choisi l'ATmega16. Il est parfait pour faire ses premiers pas :

- Il dispose d'une mémoire importante de 16Ko de Flash, 512 d'EEPROM et 1Ko de SRAM. Ces caractéristiques lui permettent d'accueillir un code compilé par... GCC. En comparaison, l'ATtiny15L peut également être utilisé sous GNU/Linux mais impérativement en assembleur.

- Il existe en boîtier PDIP 40 et est donc facile d'utilisation et de mise en œuvre, même sur une simple plaque pastillée ou une platine à essais. De plus, comme d'autres AVR il ne nécessite pas de composant supplémentaire comme un quartz ou autre.

- Il est très riche en fonctionnalités (timer, convertisseurs A/D, port série, fonctions sommeil, Watchdog, 32 lignes E/S programmables, etc.). Avec un seul composant, vous pouvez explorer toutes les fonctionnalités des AVR, et ce, pour un coût relativement modique. L'ATmega16 se vend environ 10 Euros.

LE PROGRAMMEUR

Pour charger le code dans l'ATmega16, inutile de faire appel à un programmeur (ou programmateur) particulier. Ce microcontrôleur se programme In Situ (ICSP).

Cela signifie que le composant, dans le montage final, peut être programmé sans intervention mécanique. 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 1 vous donne le détail des connexions entre le microcontrôleur et le port parallèle d'un PC. 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

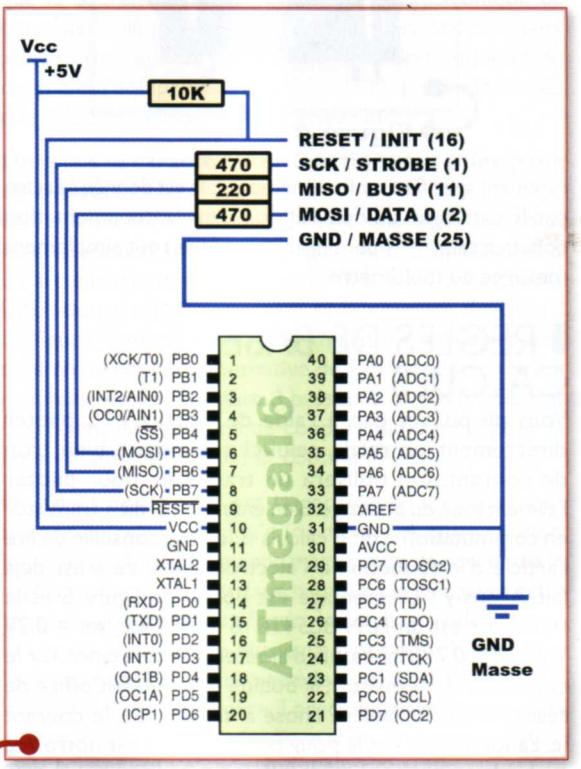


Fig. 1

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 avec des composants à 10 Euros pièce.

La photo en figure 2 montre un tel adaptateur terminé. Les caches en plastique des connecteurs DB25 laissent suffisamment de place pour y loger les résistances.

Notez la présence de la résistance entre la ligne /RESET de l'ATmega16 et l'alimentation. Il s'agit d'une résistance de rappel (voir articles dans le présent numéro sur le port parallèle en source d'interruption et l'introduction à l'électronique).

Celle-ci permet de mettre /RESET au +5V en l'absence de signal et ainsi ne PAS provoquer de reset.

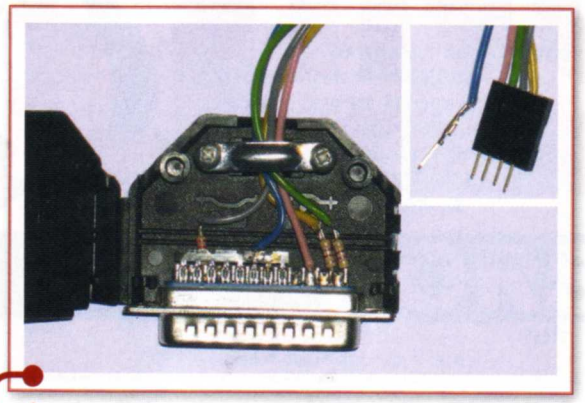


Fig. 2

LOGICIELS

Pour Développer du code AVR et ATmega16 en particulier, rien de plus simple, nous pouvons utiliser le C si peu familier aux électroniciens, mais idéal pour nous. La chaîne de compilation GNU permet de développer pour l'AVR et il suffira souvent d'installer les paquets `gcc-avr`, `binutils-avr` et `avr-libc`.

Il existe plusieurs outils permettant de programmer le code dans la mémoire Flash du microcontrôleur. Nous avons choisi UISP parce qu'il supporte énormément d'interfaces de programmation In Situ et qu'il est recommandé par Guido Socher, auteur de plusieurs articles sur les AVR pour LinuxFocus.

UISP, développé par Uros Platise et Marek Michalkiewicz, supportera aussi bien le montage donné ici (le DAPA pour "*Direct AVR Parallel Access*") qu'avec les kits Atmel STK200, 300 et 500 ou encore l'avr910 sur port série.

UISP permet également de programmer les deux registres FUSE permettant de configurer le microcontrôleur. Chaque bit ou groupe de bits de ce registre divisé en deux (l'octet haut et l'octet bas) définissent une activation ou désactivation d'une fonctionnalité ou le choix d'une configuration.

Le tableau en annexe présente la signification de chaque bit et leur valeur par défaut. A la livraison, l'ATmega16 est livré en configuration par défaut suite aux tests d'usine.

La première manipulation avec UISP nous permettra également de vérifier le bon fonctionnement du composant et de l'adaptateur parallèle. Un seul bit nous intéresse et il se trouve dans le registre de configuration, c'est le bit 6 de l'octet haut, JTAGEN.

Celui-ci programmé par défaut (0 = programmé et 1 = non programmé) active les fonctionnalités de tests et empêche l'utilisation complète du port d'E/S C. Commençons par lire les registres :

```
% uisp -dlpt=/dev/parport0 \
  -dprog=dapa --rd_fuses
Atmel AVR ATmega16 is found.
Fuse Low Byte      = 0xe1
Fuse High Byte     = 0x99
Fuse Extended Byte = 0xff
Calibration Byte   = 0xb9  --  Read Only
Lock Bits          = 0xff
    BLB12 -> 1
    BLB11 -> 1
    BLB02 -> 1
    BLB01 -> 1
      LB2 -> 1
      LB1 -> 1
```

Bonne nouvelle, notre adaptateur fonctionne et UISP a reconnu l'ATmega16. Les options utilisées sont `-dlpt` pour spécifier le port parallèle via `ppdev` et `-dprog` pour spécifier le type d'adaptateur que nous possédons. `--rd_fuses` permet de lire les registres de configuration.

Nous trouvons respectivement 0xe1 (11100001) et 0x99 (10011001), la configuration par défaut telle que spécifiée dans la documentation. Nous devons changer 0x99 en 0xd9 (11011001) pour désactiver le bit JTAGEN :

```
% uisp -dlpt=/dev/parport0 -dprog=dapa \
  --wr_fuse_h=0xd9
Atmel AVR ATmega16 is found.
Fuse High Byte set to 0xd9
```

Dès lors, nous pouvons utiliser le port C comme n'importe lequel des ports disponibles sur l'ATmega16. Revenons un instant sur la raison de ce changement (en dehors de la vérification du fonctionnement de l'adaptateur et d'UISP). Vous l'avez sans doute remarqué sur le schéma en figure 1, les broches du microcontrôleur ont plusieurs désignations.

Si nous prenons la broche 35, nous remarquons qu'il s'agit de PC5 (bit 5 du port C) mais également de TDI. Avec JTAGEN activé, la broche 35 n'est pas PC5, mais l'entrée pour les tests (TDI pour JTAG Test Data In).

Il en va de même pour quasiment toutes les broches du composant et une lecture de la documentation est nécessaire pour leur utilisation.

Ne vous effrayez pas devant les quelques 350 pages qu'elle comporte, il suffit de chercher le point précis qui nous préoccupe. Une lecture complète n'est ni utile ni souhaitable.

NOTRE CODE DE TEST

Un microcontrôleur ne dispose pas des ressources mémoire d'un ordinateur. Très nombreux sont les composants ne se programmant qu'en assembleur pour cette raison. Ici, nous pouvons utiliser le C mais le code reste simple et concis.

A l'instar du classique «Hello World!» cher aux premiers programmes de tout langage, il existe un premier programme classique pour les microcontrôleurs : le clignotement d'une LED. C'est précisément l'objet du code suivant :

```
#include <avr/io.h>

void mondelai(unsigned short i)
{
  while(i>0) {
          i--;
  }
}

int main(void)
{
  DDRC|= _BV(PC5);

  while (1) {
          PORTC &= ~_BV(PC5);
          mondelai(65000);
          PORTC|= _BV(PC5);
          mondelai(5000);
  }
}
```

Détaillons tout cela. Le fichier `avr/io.h` est en fait une simple astuce permettant d'inclure les macros spécifiques au modèle de microcontrôleur. Dans le cas de l'ATmega16, il s'agit de `avr/iom16.h`. La fonction `mondelai` est une simple boucle de retard afin que nous ayons le temps de voir clignoter la LED. Notez que nous devons utiliser un type `unsigned short`.

`DDRC` est le registre de direction pour le port C. Chaque ligne de chaque port peut être définie en entrée (0) ou en sortie (1). La macro `_BV` correspond tout simplement à `(1 << (bit))`. C'est une facilité permettant d'activer un bit au choix dans un octet. Ainsi, nous mettons à 1 le bit 5 du registre `DDRC`. La ligne est en sortie.

Vient ensuite la boucle principale. `PORTC` correspond à l'adresse du port C. Là encore, nous utilisons la macro `_BV` pour jouer sur le bit 5. Nous procédons à un ET logique après avoir inversé l'octet obtenu et passons ainsi le bit à 0 pour mettre la ligne à la masse.

Nous appelons notre fonction de retardement puis nous procédons à l'opération inverse (avec un OU).

Nous enregistrons ce source dans un fichier `premierAVR.c` puis construisons notre `Makefile` :

Les points importants ici sont :

```
CC=avr-gcc
OBJCOPY=avr-objcopy
PARPORT=/dev/parport2

CFLAGS=-g -mmcu=atmega16 -Wall \
 -Wstrict-prototypes -Os -mcall-prologues

all: premierAVR.hex

premierAVR.hex : premierAVR.out
          $(OBJCOPY) -R .eeprom -O ihex \
          premierAVR.out premierAVR.hex

premierAVR.out : premierAVR.o
          $(CC) $(CFLAGS) -o premierAVR.out \
          -Wl,-Map,premierAVR.map premierAVR.o

premierAVR.o : premierAVR.c
          $(CC) $(CFLAGS) -c premierAVR.c

clean:
          rm -f *.o *.map *.out *.hex
```

- L'option `-mmcu=atmega16` permet de spécifier le type de microcontrôleur que nous utilisons. D'elle découle les fichiers d'en-tête utilisés et ainsi des macros propres au modèle d'AVR.
- `avr-objcopy` permet de convertir le fichier objet dans un format intelligible par UISP

Après exécution de la commande `make`, nous obtenons plusieurs fichiers dont celui qui sera utilisé par UISP (`premierAVR.hex`).

C'est une représentation hexadécimale du code qui sera chargé dans l'ATmega16. Le fichier `premierAVR.map` est un résumé de l'occupation mémoire du microcontrôleur.

Nous ajoutons quelques lignes à notre `Makefile` afin d'automatiser le chargement du code :

```
prog_erase:
          uisp -dlpt=$(PARPORT) --erase -dprog=dapa

prog_burn:
          uisp -dlpt=$(PARPORT) --upload \
          if=premierAVR.hex -dprog=dapa -dno-poll -v

prog_verif:
          uisp -dlpt=$(PARPORT) --verify \
          if=premierAVR.hex -dprog=dapa -dno-poll -v

prog: prog_erase prog_burn prog_verif

cprog: premierAVR.hex prog_erase prog_burn prog_verif
```

Les options utilisées sont respectivement `--erase`, `--upload` et `--verify` pour l'effacement de la mémoire flash, le chargement/programmation du composant et la vérification de l'opération d'écriture.

Notez la présence de l'option `-dno-poll` permettant de désactiver le *polling* sur le port et avoir un accès direct à ce dernier.

Il ne nous reste plus qu'à mettre le microcontrôleur en situation en l'alimentant et en connectant sur la broche 35 une résistance de 470 Ohms et une LED dont la cathode sera reliée à la masse commune (figure 3).

Apprivoisez votre pingouin !

GNU LINUX PRATIQUE

HORS SÉRIE 2

64 pages

SPÉCIALES EDUCATION

Collège, lycée, enseignement supérieur

En kiosque le 9 décembre

LIVE CD

basé sur Ubuntu

OpenOffice.org

Incluant toutes les applications phares !

PCB et gEDA (électronique), Maxima et SciGraphica (maths), Celestia (astronomie), Chemtool (chimie), etc. Sans oublier la suite bureautique OpenOffice.org, Firefox et Gimp.

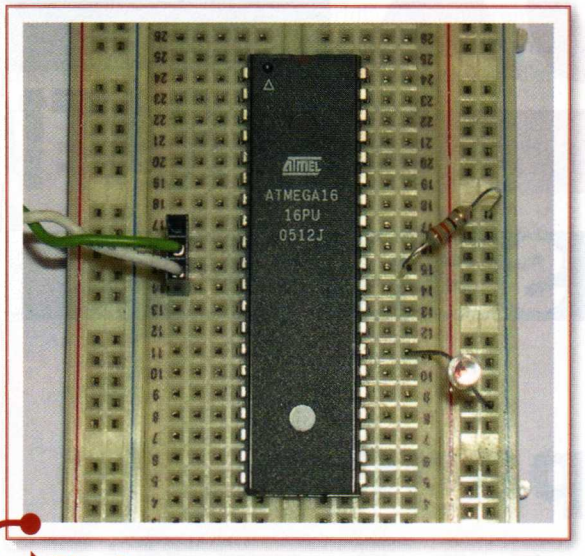


Fig. 3

Enfin, connectons l'adaptateur pour la programmation In Situ et exécutons joyeusement un `make prog` :

```
[...]
Erasing device ...
Reinitializing device
[...]
Uploading: flash
#######
(total 200 bytes transferred in 0.15s (1301 bytes/s)
[...]
Verifying: flash
#######
(total 200 bytes transferred in 0.13s (1584 bytes/s)
```

Immédiatement après la programmation l'ATmega16 exécute le code dans sa mémoire flash et la LED clignote comme prévu. Si vous rencontrez des problèmes lors de la programmation, vérifiez que le port utilisé est bien le bon, que vous disposez des permissions adéquates et que les connexions sont correctes. Enfin, si le problème persiste, vous pouvez jouer sur les options de paramétrage des délais comme `-dt_sck`. N'oubliez pas, enfin, que par défaut Linux n'est pas en temps réel et que la charge système influe directement sur la gestion des délais.

UN PEU PLUS LOIN

Notre premier code est non seulement simpliste mais relativement cavalier. En effet, la libc pour les microcontrôleurs AVR offre quelques facilités de développement. Ainsi, lorsqu'on souhaite utiliser des délais d'attente, nous pouvons utiliser les fonctions prévues dans `avr/delay.h` comme `_delay_us` ou `_delay_ms`. Malheureusement, dans le cas présent c'est insuffisant. La première fonction prend en paramètre un maximum de 768 microsecondes et la seconde une valeur identique divisée par la fréquence du microcontrôleur (1Mhz par défaut).

Comme spécifié dans `delay.h`, pour des délais plus importants, on utilise normalement l'un des timers inclus dans l'AVR. Un timer est une horloge/compteur interne. Il se configure en lui associant une source d'horloge et éventuellement un diviseur.

Le microcontrôleur fonctionnant par défaut à 1Mhz, le diviseur permet d'espacer les tick d'horloge. Pour utiliser un timer en C, il ne nous faut que quelques lignes. Commençons par inclure des fichiers d'en-tête appropriés, puis configurons le timer :

```
#include <avr/interrupt.h>
#include <avr/signal.h>
[...]
TCCR1B = _BV(CS10) | _BV(CS11)  | _BV(WGM12);
```

`TCCR1B` est le registre B permettant la configuration du Timer/Counter 1. A l'aide de OU logiques, nous activons certains bits qui nous intéressent. `CS10`, `CS11` et `CS12` nous permettent de choisir la source d'horloge et le diviseur à utiliser.

Ici 011 définit un diviseur de 64 (voir documentation page 112). Le bit `WGM12` précise que le timer (TCNT1) est remis à zéro lorsqu'il y a correspondance avec le registre de comparaison.

Dès lors, l'ATmega16 en fonctionnement incrémentera le timer à une fréquence de 1Mhz/64 soit environ 15Hz. Il faut maintenant surveiller ce timer afin de l'utiliser. Pour cela, inutile d'écrire le moindre code, le microcontrôleur possède un registre de comparaison qu'il suffit d'initialiser avec la valeur de timer qui nous intéresse :

```
OCR1A = (F_CPU/64);
```

`F_CPU` sera défini avec la fréquence à laquelle fonctionne l'AVR. Nous configurons ici le registre de comparaison à la valeur correspondante à celle du diviseur pour obtenir une fréquence de 1Hz au final. `OCR1A` et le timer sont comparés à chaque cycle.

A présent, lorsqu'il y a correspondance, le timer est remis à zéro, mais nous en attendons plus. Nous voulons qu'une interruption se déclenche.

Pour ce faire, il faut configurer le registre `TIMSK` (*Timer Interrupt Mask Register*) et en particulier le bit 7 nommé OCIE1 (*Timer/Counter1 Output Compare Match Interrupt Enable*) :

```
TIMSK  = _BV(OCIE1A);
```

Maintenant, lorsque qu'il y a correspondance entre le timer et OCR1A une interruption survient et nous pouvons l'utiliser. Il nous suffit de définir la macro de callback SIGNAL par ailleurs :

```
SIGNAL (SIG_OUTPUT_COMPARE1A)
{
        PORTC|= _BV(PC5);
        _delay_us(768);
        PORTC &= ~_BV(PC5);
}
```

Le signal concerné est SIG_OUTPUT_COMPARE1A, celui-là même généré à la correspondance de OCR1A. Nous réutilisons les macros d'activation de la broche 35.

Notez que cette fois, nous utilisons la fonction _delay_us avec sa valeur maximale en lieu et place de notre boucle d'attente.

Cela donnera une brève impulsion à la LED, suffisante pour être visible. Pour utiliser _delay_us nous devons définir F_CPU avec la fréquence du microcontrôleur avant d'inclure avr/delay.h.

La totalité du code devient donc :

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/signal.h>

#define F_CPU 1000000UL
#include <avr/delay.h>

SIGNAL (SIG_OUTPUT_COMPARE1A)
{
  PORTC|= _BV(PC5);
  _delay_us(768);
  PORTC &= ~_BV(PC5);
}

int main(void)
{
  DDRC|= _BV(PC5);
  PORTC &= ~_BV(PC5);

  TCCR1B = _BV(CS10) | _BV(CS11) | _BV(WGM12);
  OCR1A = (F_CPU/64);
  TIMSK  = _BV(OCIE1A);

  sei();
  for (;;) {}
}
```

Notez l'appel à la fonction sei() permettant le traitement des signaux qui devient transparent pour le développeur. A présent, notre code simpliste utilise presque pleinement les ressources et facilités mises à notre disposition à la fois par le microcontrôleur et par la libc AVR.

ENCORE TROP DE CODE

Et oui, j'ai dis que nous utilisions « presque pleinement » les ressources de l'AVR. Pour le type d'utilisation que nous avons ici, il n'est pas forcément nécessaire de développer son propre code pour faire clignoter la LED.

Il est possible de directement relier la sortie du timer avec une ligne du port D. L'une des lignes utilisables est OC1A/PD5 (19). Pour faire l'association, il suffit d'activer le bit COM1A0 du registre TCCR1A :

```
TCCR1A = _BV(COM1A0);
```

Dès lors, la correspondance entre le timer et OCR1A provoque une bascule sur la ligne OC1A/PD5 à la fréquence correspondante. Notez que le délai d'allumage et d'extinction de la LED sont égaux. Nous n'avons plus besoin de SIGNAL ni de la fonction de gestion seo(). Notre code se résume ainsi à cinq lignes :

```
DDRD|= _BV(PD5);
TCCR1B = _BV(CS10) | _BV(CS11) | _BV(WGM12);
TCCR1A = _BV(COM1A0);
OCR1A = (F_CPU/64);
for (;;) {}
```

Je comptais conclure l'article sur ce dernier exemple, mais hypnotisé par la LED clignotant frénétiquement sur mon bureau, je ne peux m'empêcher de vous parler de la PWM.

PWM

La question de base qui mène ici à la PWM (*Pulse Width Modulation*) est : comment jouer sur l'intensité lumineuse de la LED. Que ceux qui s'apprêtaient à répondre qu'il suffit de jouer sur l'intensité du courant se détrompent.

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.

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 ou modulation d'impulsions en largeur en bon français.

L'ATmega16, comme d'autres modèles de microcontrôleurs Atmel, dispose en interne de fonctionnalités permettant d'utiliser la PWM sans avoir à la coder dans un programme. Tout comme avec le code précédent, la PWM est directement liée avec la configuration du timer et du registre TCCR1A.

C'est la valeur donnée au comparateur OCR1A qui déterminera le rapport de phase.

Afin de bien voir et donc comprendre le fonctionnement de la PWM, configurons le timer nous permettant d'avoir une fréquence PWM d'environ 4KHz (avec un diviseur de 256) :

```
TCCR1B = _BV(CS12) | _BV(WGM12);
```

Reste ensuite à configurer TCCR1A de manière à obtenir la PWM sur la sortie OC1A/PD5 :

```
TCCR1A = _BV(COM1A1) | _BV(WGM10) | _BV(WGM12);
```

En activant les bits WGM10 et WGM12, nous configurons un mode dit "*Fast PWM*". Le bit COM1A1 définit le mode de comparaison et l'effet sur OC1A/PD5 (et OC1B/PD4).

Le timer est remis à zéro lorsqu'il atteint la valeur maximale (TOP) définie par la résolution PWM. Avec WGM10 et WGM12 cette résolution est de 8 bits et la valeur maximale de 255.

En activant COM1A1 on définit les règles suivantes : on active OC1A/PD5 lorsque le timer atteint la valeur maximale (TOP) et on désactive OC1A/PD5 lorsqu'il y a correspondance entre le timer et OCR1A.

Si OCR1A est égal à TOP (255) alors notre sortie est toujours active. Il n'y a plus de phase où la ligne est ramenée à la masse. C'est un cas particulier géré par l'AVR, il n'y a pas de changement du doute, même d'une microseconde. Si nous définissons OCR1A à 127 voici ce qui arrivera :

- Le timer au bout de 256 tick atteindra la valeur maximale TOP
- OC1A/PD5 va alors être active
- Le timer bouclera
- Le timer arrivera à 127, il y a correspondance avec OCR1A
- OC1A/PD5 est ramené à la masse, notre LED s'éteint
- le timer continue et arrive à nouveau à TOP
- OC1A/PD5 allume notre LED
- etc.

Dans ce cas de figure, les périodes allumées/éteintes de la LED sont égales, elle clignote. Avec un diviseur de 256 (CS12) et une fréquence de 1Mhz, cela devient parfaitement visible.

Si nous descendons encore la valeur de OCR1A, les périodes où la LED est allumée deviennent plus courtes que lorsqu'elle est éteinte. Elle ne scintille plus, elle clignote franchement. Faites l'essai avec le code suivant :

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/signal.h>

int main(void)
{
  DDRD|= _BV(PD5);
  TCCR1B = _BV(CS12) | _BV(WGM12);
  TCCR1A = _BV(COM1A1) | _BV(WGM10) | _BV(WGM12);
  OCR1A=127;
  for (;;) {}
}
```

Compilez et chargez le code dans le microcontrôleur, puis changez la valeur de OCR1AL entre 0 et 255. On voit alors clairement ce qu'est la PWM.

Maintenant, remplacez _BV(CS12) par _BV(CS11) pour un diviseur de 8 au lieu de 256. La fréquence est telle que nous ne voyons plus le clignotement, l'intensité lumineuse de la LED semble différente. Plus loin encore, faisons varier OCR1A en plaçant dans la portée de la boucle for le code suivant :

```
for (i=15; i<255; i++)
  {
  OCR1A = i;
  _delay_us(768);
  }
for(i=255; i>15; i--)
  {
  OCR1A = i;
  _delay_us(768);
  }
```

La LED n'est jamais complètement éteinte car OCR1AL n'est jamais à 0 et la LED "*pulse*". Nous pouvons ensuite jouer sur la vitesse de pulsation en définissant des délais plus longs que les 768 microsecondes.

La PWM permet d'influer sur le fonctionnement de composants pour lesquels il n'est pas possible de moduler l'intensité du courant ou la tension. Ceci s'applique aux LED comme ici, mais également à des moteurs. La PWM permet également la régulation de tensions. Il suffit d'utiliser un condensateur et on arrive ainsi à obtenir toute une gamme de tensions.

N'oubliez pas, le condensateur s'oppose à tout changement brusque de tension. Les convertisseurs digitaux/analogiques fonctionnent également de cette manière avec la PWM.

CONCLUSION

Le microcontrôleur ATmega16, comme le reste de la famille AVR d'Atmel, est un composant très riche. Nous n'avons ici fait qu'effleurer le sujet, même si cela paraît déjà beaucoup.

Mais son plus grand intérêt réside dans le fait de pouvoir le programmer en C grâce à des outils libres et une bibliothèque C vraiment complète.

Si vous souhaitez investir du temps et un peu d'argent dans le monde des microcontrôleurs, on peut affirmer sans trop de risque que les AVR sont un excellent point de départ. Vous pourrez toujours, par la suite, vous tourner vers l'assembleur puis aborder d'autres microcontrôleurs comme les PIC ou ceux produits par Freescale.

Personnellement, depuis que j'ai fait connaissance avec les AVR, je n'ai plus vraiment envie de toucher aux PIC.

OCTET BAS (FUSE LOW BYTE)

Bit	Nom	Description	Par défaut	Commentaire
7	BODLVL	Brown-Out Detector Level	1	Règle de niveau de détection Brown-Out
6	BODEN	Brown-Out Detector Enable	1 (BOD désact.)	La détection Brown-Out permet de surveiller le niveau d'alimentation du composant et provoquer un reset le cas échéant
5	SUT1	Start-Up	1	Règlage du délai d'activation de l'horloge. Les 65 ms par défaut conviennent pour l'horloge interne
4	SUT0	Start-Up	0	-
3	CKSEL3	Clock Select.	0	Règlage du type d'horloge.
2	CKSEL2	Clock Select.	0	Par défaut l'horloge interne est à 1 Mhz (0001)
1	CKSEL1	Clock Select.	0	On peut également choisir 2 Mhz (0010)
0	CKSEL0	Clock Select.	1	Ou 4 Mhz (0011) ou encore 8 Mhz (0100)

OCTET HAUT (FUSE HIGH BYTE)

Bit	Nom	Description	Par défaut	Commentaire
7	OCDEN	OCD enable	1 (OCD désact.)	Activation du système de Debugage sur le composant
6	JTAGEN	JTAG enable	0 (JTAG activ.)	Permet d'autoriser l'utilisation du port d'accès pour test (TAP)
5	SPIEN	SPI serial programming	0 (SPI activ.)	Ne peut être changé en programmation série, logique
4	CKOPT	Oscillator Options	1	Change les options d'horloge, variable en fonction de CKSEL
3	EESAVE	EEPROM preserved	1 (non préservée)	Conserve l'EEPROM en cas d'effacement du code
2	BOOTSZ1	Boot Size	0	-
1	BOOTSZ0	Boot Size	0	-
0	BOOTRST	Reset Vector	1	L'ATmega16 supporte plusieurs méthode de programmation. Ceci permet d'utiliser un bootloader

http://en.wikibooks.org/wiki/Atmel_AVR

Denis Bodor :: db@ed-diamond.com :: lefinnois@lefinnois.net

CONNECTEZ-VOUS AUX CONSOLES !

Derrière ce titre se cache un problème et une solution typique pour tout bidouilleur qui se respecte. Comment se connecter à un périphérique communiquant en mode série qui n'utilise pas des tensions similaires à celles d'un PC ? La solution tient en une famille de composants spécifiques.

Routeurs, calculatrices scientifiques, modems ADSL, microcontrôleurs, systèmes embarqués divers... autant de périphériques et de matériels disposant d'un port de communication série ou d'un port dénommé « console » souvent incompatibles avec les ports série d'un PC.

Non seulement, ces derniers sont en voie de disparition, mais cela est d'autant plus grave qu'ils constituent une interface idéale pour piloter, flasher et configurer de nombreux systèmes.

Si nous prenons l'exemple d'un routeur ou d'un modem, il n'est pas rare d'y trouver un port spécifique utilisant un connecteur RJ45, RJ11 ou DB9.

Parfois, un simple adaptateur est suffisant pour le connecter à un PC et ainsi rattraper une configuration maladroite ayant bloqué son fonctionnement ou la prise de contrôle via le port ethernet.

Le plus souvent, la connexion directe n'est pas possible et nécessite une mise en forme des signaux. La solution la plus évidente est alors l'achat d'un câble de connexion dédié vendu à un prix dépassant de loin sa valeur matérielle.

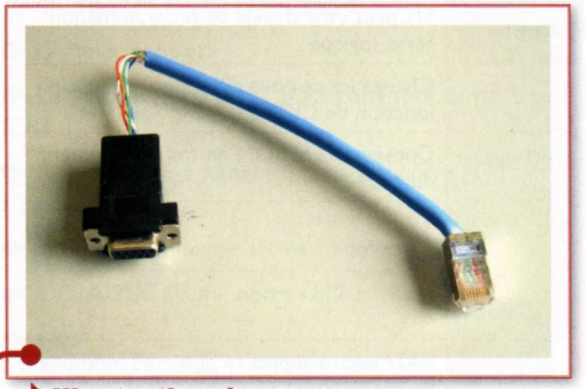


Illustration 1

Exemple de câble de connexion RJ45/DB9 fait maison pour une connexion RS232 entre un routeur et le PC.

Illustration 2

Vue arrière d'un routeur Efficient Networks 5711 et de son port console. Rien ne l'indique clairement ici, mais la documentation du produit indique qu'il s'agit bien de RS232.

RAPPEL SUR LE PORT SÉRIE

La désignation « port série » fait référence à un port de communication spécifique fonctionnant, on s'en doute, de manière sérielle. Les bits d'information sont envoyés via une seule ligne un après l'autre (idem pour la réception). Cette communication série est à voir en opposition à une communication parallèle. C'est le cas pour le port du même nom où un octet est envoyé sur huit lignes représentant chacune la valeur d'un bit. La communication série n'est pas spécifique au PC, mais largement utilisée pour bon nombre de liaisons. La grande spécificité du port série des PC réside dans les tensions utilisées. Un tel port respecte le standard RS232 (plus exactement EIA232). Celui-ci spécifie l'assignation des broches pour les connecteurs DB9 et DB25, les délais, la fonction de chaque signal et surtout les tensions utilisées.

C'est d'ailleurs là que tout le problème d'interconnexion se pose. Un port série EIA232 utilise aussi bien des tensions positives que négatives. Une tension entre –3 V et –25 V est un 1 logique, une tension entre +3 V et +25 V est un 0 logique. Les tensions entre –3 V et +3 V sont dites « de transition » et ne signifient rien. Nous sommes loin des signaux TTL où 0 V est un 0 logique et +5 V un 1 logique. Vous comprendrez qu'une simple interconnexion entre un port EIA232 et le port « console » non EIA232 d'un routeur risque de conduire à la destruction partielle de l'interface d'un côté ou de l'autre... ou des deux.

BROCHE

Broche	Nom	Nom complet	Direction	Fonction
3	TxD	*Transmit Data*	➡	Envoi des données depuis le PC
2	RxD	*Receive Data*	⬅	Réception des données sur le PC
7	RTS	*Request To Send*	➡	Contrôle de flux RTS/CTS
8	CTS	*Clear To Send*	⬅	Contrôle de flux RTS/CTS
6	DSR	*Data Set Ready*	⬅	Prêt à communiquer
4	DTR	*Data Terminal Ready*	➡	Prêt à communiquer
1	DCD	*Data Carrier Detect*	⬅	Indicateur de connexion
9	RI	*Ring Indicator*	⬅	Détection de sonnerie
5	SG	*Signal Ground*		Masse

RS232 VERS TTL

Bon nombre de périphériques utilisent des signaux dit « TTL ». C'est un standard largement utilisé aussi bien par les composants du même nom que par bon nombre de périphériques. La mise en forme des signaux de l'une à l'autre interface se fait simplement via un composant dédié largement répandu : le MAX232 (figure 1). Ce composant originellement fabriqué par Maxim Dallas est un convertisseur RS232/TTL double. Comprenez par là qu'il intègre deux unités de conversion indépendantes.

La plupart du temps, seule l'une d'entre elles est utilisée. Le MAX232 est alimenté en +5 V et des condensateurs doivent être ajoutés pour élever cette tension à +10 V ou –10 V pour la partie communicant avec le PC (figure 2). La documentation du composant indique une valeur de 1 µF pour le MAX232. D'autres composants de la famille utilisent des valeurs différentes. On notera cependant, que le standard RS232/EIA232 permet une certaine tolérance et on pourra donc utiliser des condensateurs de 2.2 µF ou 4.7 µF (parfois même 22 µF sur certains schémas présents sur le web).

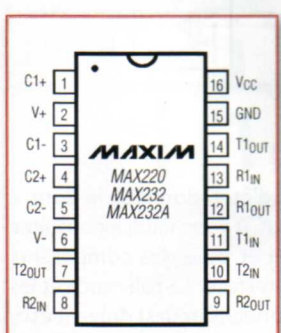


Fig. 1

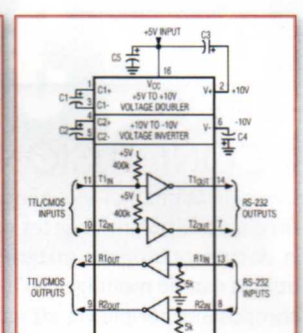


Fig. 2

Les condensateurs polarisés doivent être de bonne qualité. On optera de préférence pour des tantales gouttes, plus chers mais bien moins encombrants que les condensateurs électrolytiques (chimiques). La mise en œuvre est simplissime (figure 3) et le schéma parle de lui-même. Il suffit, en effet, de correctement repérer les pattes du composant et de connecter les condensateurs. Ce type de montages est directement utilisable avec certains routeurs, calculatrices (Casio, TI) et téléphones GSM/GPRS.

Pour alimenter le montage, il n'est pas forcément nécessaire d'utiliser une source externe. Dans la plupart des cas, celui-ci peut tirer suffisamment de courant du port série lui-même. Pour cela, on connectera tout simplement une diode sur chacune des broches DTR et RTS (afin de bloquer les tensions négatives) et on les reliera ensemble en entrée d'un régulateur 78L05 (figure 4). Le condensateur correspond au C5 de la figure 2 et sera de même valeur que tous les autres. On obtient ainsi un ensemble autonome. En utilisant des composants de taille réduite (CMS) et en faisant preuve de patience, on arrivera à faire tenir tout cela dans le cache plastique d'un connecteur DB9. (*Voir tableau broche*)

Pour faciliter la prise en charge avec un terminal de communication (comme Minicom), on pourra connecter les broches 7 et 8 (RTS et CTS) pour simuler un contrôle de flux matériel ainsi que les broches 4 et 6 (DTR et DSR) pour simuler la disponibilité systématique des deux équipements : le PC (DTE, *Data Terminal Equipment*) et le périphérique (DCE, *Data Communications Equipment*).

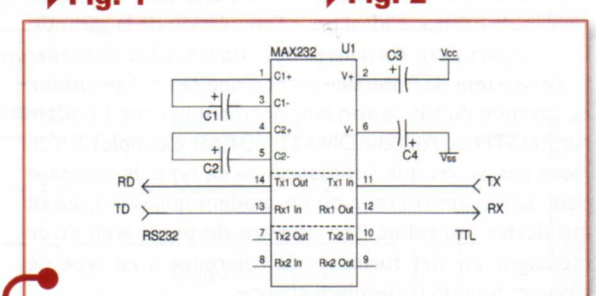


Fig. 3

BROCHE		
Vcc	**C1**	**C1, C3 et C4**
3.3V	0.1 µF (100nF)	0.1 µF (100nF)
5V	0.047 µF (47nF)	0.33 µF (330 nF)
de 3V à 5.5V	0.1 µF (100nF)	0.47 µF (470nF)

RS232 VERS NON-TTL

De plus en plus d'équipements n'utilisent plus en sortie « console » de signaux TTL, mais LV-TTL (LV pour *Low Voltage*). Nous n'avons donc plus affaire à du +5 V mais à du +3.3 V. C'est le cas, par exemple, du connecteur interne des modems *SpeedTouch Home* d'Alcatel. Pour ce type d'interfaçage, on utilisera un MAX3232. La principale différence avec le MAX232 est le fait qu'il soit capable de travailler avec différentes tensions allant de +3 V à +5.5 V. La tension d'alimentation du composant (Vcc) déterminera la tension du côté du périphérique. Il conviendra également de choisir les condensateurs en fonction de cette tension selon le tableau ci-contre. Pour alimenter le composant en +3.3 V, plusieurs solutions existent. On peut directement utiliser une source utilisant cette tension (connecteur d'alimentation ATX de la carte mère) ou utiliser une source externe ou encore le port série lui-même comme avec un MAX232.

Le régulateur 78L05 ne fera bien sûr pas l'affaire. On pourra le remplacer par un 78L033 fournissant exactement la tension recherchée. Autre solution, utiliser un LM317 (moins coûteux) que l'on utilisera avec des résistances permettant d'ajuster la tension selon la formule 1.25*(1+R2/R1). La figure 5 montre l'utilisation du régulateur LM317 pour obtenir une tension d'environ +3.3 V (1.25*(1+(2200/(1000+330)))=3.31766). La figure 6 montre le brochage du MAX3232 ainsi que le schéma logique. On utilisera celui-ci de la même manière que le MAX232 (figure 3).

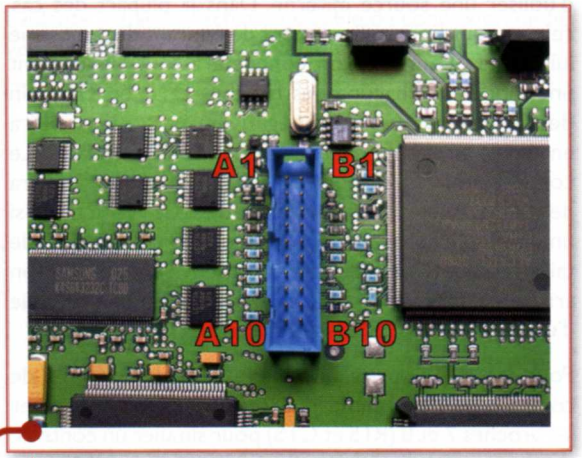


Illustration 3

Vue interne d'un modem SpeedTouch Home d'Alcatel. Les broches A10 et B10 seront connectées respectivement à la broche 9 et 10 du MAX3232. B6 correspond à la masse.

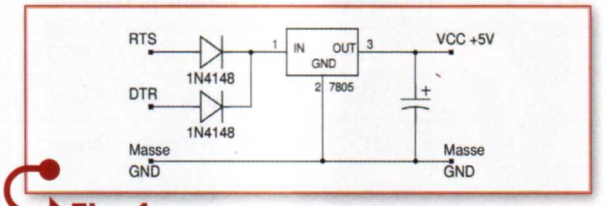


Fig. 4

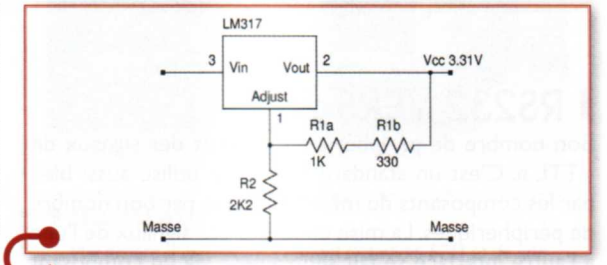


Fig. 5

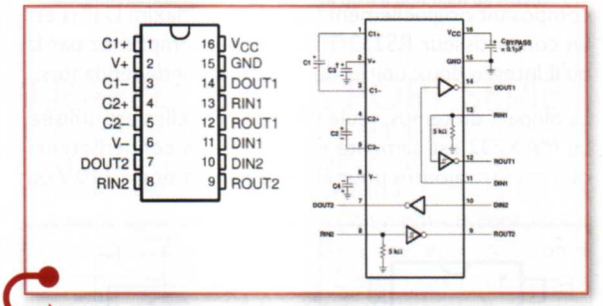


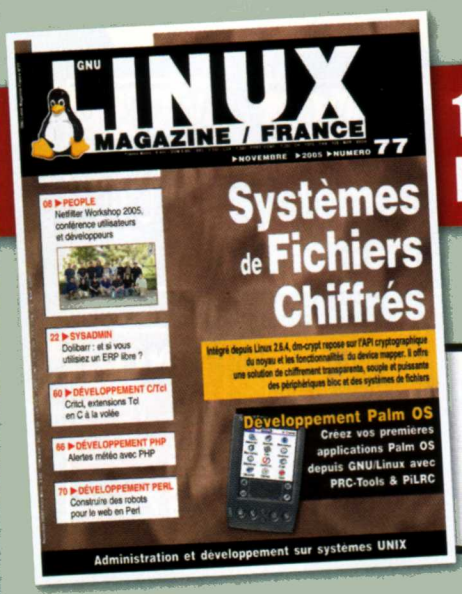
Fig. 5

CONCLUSION

Les indications qui viennent d'être données le sont à titre indicatif. Dans tous les cas, il vous faudra consulter la documentation du matériel et celle des composants utilisés dans le montage de l'interface. La tolérance et les composants périphériques (condensateurs) doivent être choisis avec attention. Enfin, il est évident que l'utilisation d'un câble fait maison n'est pas recommandée par les fabricants. Dans la plupart des cas, une telle utilisation servira de prétexte idéal pour l'annulation de la garantie. D'autre part, pour certains périphériques, le fait d'accéder au connecteur de l'interface est déjà une cause d'annulation de garantie du fait de son emplacement interne (modem Alcatel STH ou routeur Dlink DI-604, par exemple). Il n'en reste pas moins que l'utilisation de ce type de montage peut sauver un routeur ou un modem qui, sinon, aurait été destiné au rebut. Bon nombre de pages web et de messages sur des forums font référence à ce type de résurrection de la dernière chance.

Abonnez - vous !

11 Numéros de Linux Magazine



1 an de bonne lecture, bien UNIX, bien technique... Bref...

LES 3 BONNES RAISONS DE VOUS ABONNER !

- Ne manquez plus aucun numéro
- Recevez Linux Magazine chaque mois chez vous, ou dans votre entreprise
- Economisez 17,40 €/an ! (soit presque 3 magazines offerts !)

- Des offres de couplage sont disponibles
- Retrouvez les Tarifs étrangers hors France Métro sur www.ed-diamond.com

BON D'ABONNEMENT À REMPLIR ET À RETOURNER À (OU PHOTOCOPIE)

*DIAMOND Éditions - LINUX MAGAZINE - 6 Rue de la scheer - 67603 SÉLESTAT Cedex

11 Numéros de Linux Magazine

53€ Offre France Métro

Soit

Votre Linux Magazine à 4,82€

(Tarif au numéro dans le cadre d'un abonnement France Métro)

Pour les tarifs étrangers, consultez notre site : www.ed-diamond.com

LES 4 FAÇONS DE VOUS ABONNER !

- Par courrier postal en nous renvoyant le bon ci-dessous.
- Par le Web, sur notre site : www.ed-diamond.com.
- Par téléphone (paiement C.B.) entre 9h-12h & 14h -17h au 03 88 58 02 08.
- Par Fax au 03 88 58 02 09 C.B. et/ou bon de commande administratif

❍ Oui, je souhaite m'abonner à Linux Magazine pour 11 numéros

1 Voici mes coordonnées postales

Nom :

Prénom :

Adresse :

Code Postal :

Ville :

2 Je joins mon réglement de 53€ :

☐ Je règle par chèque bancaire ou postal à l'ordre de Diamond Editions*

☐ Paiement par carte bancaire :

N° Carte :

Expire le :

Date et signature obligatoire : 200

☐ Pour avoir un suivi par e-mail de vos abonnements, merci de nous indiquer votre adresse e-mail** :

**En application des articles 27 et 34 de la loi dite «Informatique et libertés» n° 78-17 du 6 janvier 1978, vous disposez d'un droit d'accès et de rectification aux données vous concernant.

Offres collectionneurs

Les anciens numéros !

LES 4 FAÇONS DE COMMANDER !

- Par courrier postal en nous renvoyant le bon ci-dessous.
- Par le Web, sur notre site : www.ed-diamond.com.
- Par téléphone (paiement C.B.) entre 9h-12h & 14h -17h au 03 88 58 02 08.
- Par Fax au 03 88 58 02 09 C.B. et/ou bon de commande administratif

TOUJOURS DISPONIBLES !

BON DE COMMANDE À REMPLIR ET À RETOURNER À (OU PHOTOCOPIE)

*DIAMOND Editions - LINUX MAGAZINE 6 Rue de la Scheer - 67603 SELESTAT Cedex

Bon de commande Linux Magazine			
Référence	**Prix / N°s**	**Qté.**	**Total**
Linux Magazine 66	6,40 €		
Linux Magazine 67	6,40 €		
Linux Magazine 68	6,40 €		
Linux Magazine 69	6,40 €		
Linux Magazine 70	6,40 €		
Linux Magazine 71	6,40 €		
Linux Magazine 72	6,40 €		
Linux Magazine 73	6,40 €		
Linux Magazine 74	6,40 €		
Linux Magazine 75	6,40 €		
Bon de commande Linux Magazine Hors Série			
LM HS 08 Introduction à la crypto	5,95 €		
LM HS 09 Installer son serveur Web à la maison	5,95 €		
LM HS 10 Installation de votre serveur internet	5,95 €		
LM HS 11 GIMP par la pratique	5,95 €		
LM HS 12 Firewall votre meilleur ennemi Acte 1	5,95 €		
LM HS 13 Firewall votre meilleur ennemi Acte 2	5,95 €		
LM HS 14 Maitrisez Blender	5,95 €		
LM HS 15 GIMP et la Photo	5,95 €		
LM HS 16 Kernel (1)	5,95 €		
LM HS 17 Kernel (2)	5,95 €		
LM HS 18 Haute Disponibilité	5,95 €		
LM HS 19 The Gimp 2.0	5,95 €		
LM HS 20 PHP 5	5,95 €		
LM HS 21 Recyclez vos PC	6,40 €		
		TOTAL	
		Frais de port France	+ 3,81 €
		Frais de port Etranger	+ 5,34 €
		TOTAL	

1 Voici mes coordonnées postales

Nom :

Prénom :

Adresse :

Code Postal :

Ville :

2 Je joins mon réglement :

☐ Je règle par chèque bancaire ou postal à l'ordre de Diamond Editions*

☐ Paiement par carte bancaire :

N° Carte :

Expire le :

Date et signature obligatoire : 200

MODULE I/O 24 D'ELEXOL

Développer ses propres montages est souvent intéressant mais, lorsque la complexité du cahier des charges augmente, on bascule facilement dans des modules préfabriqués et adaptables. C'est le cas du module présenté ici.

Comment disposer de plusieurs entrées ou sorties numériques accessibles au travers d'un réseau Ethernet ? Plusieurs solutions répondent pleinement à cette question. L'usage d'un PC ou d'un système embarqué peut être une solution. Cependant, la première est souvent un gouffre en termes de consommation électrique et la seconde relativement onéreuse. Dans les deux cas, la solution est disproportionnée.

Une autre solution consiste à utiliser un microcontrôleur auquel on ajoute un module Ethernet comme le Sollae Systems EZL-50. Le microcontrôleur reçoit et envoie les données de manière sérielle et le module s'occupe de la conversion, à la charge de l'utilisateur de développer le code pour le microcontrôleur choisi.

Enfin, une dernière solution consiste à utiliser un module dédié. Le fabricant Elexol, par exemple, déjà connu pour ses modules USB/série ou USB/parallèle propose au détail un produit sous la désignation « Ether I/O 24 ». Celui-ci dispose de 24 entrées/sorties réparties en trois ports. Il se pilote via UDP et peut être configuré de manière à obtenir une adresse via DHCP ou une adresse fixe stockée en EEPROM.

Les caractéristiques du module sont les suivantes :

- E/S haute vitesse. Le module offre des temps de latence très réduits permettant quelques 250000 lectures et/ou 500000 écritures à la seconde.
- Détection de changement d'état en entrée. Le module est capable de contacter l'hôte maître sans être interrogé. Ce type de fonctionnalités permettra, par exemple, de connecter en Ethernet deux modules sans intervention d'un PC serveur.
- Etat de mise sous tension programmable. Il est possible de programmer l'état des ports du module lors de sa mise en route. Inutile ensuite de recourir à un hôte pour le reconfigurer.
- Isolation électrique. Les E/S sont électriquement isolées entre elles afin de protéger à la fois le montage auquel est raccordé le module et le réseau.

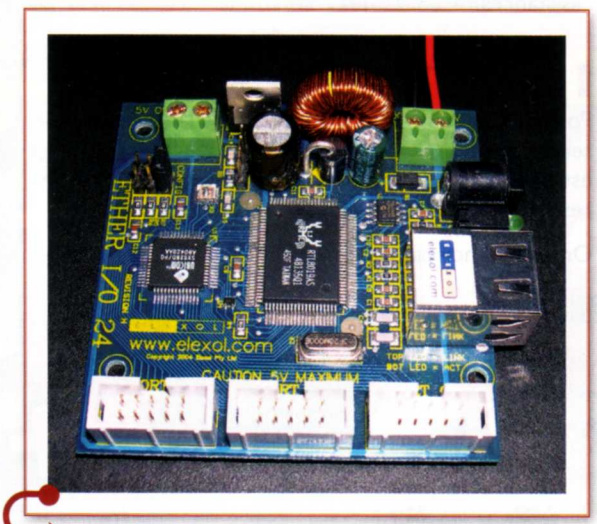


Illustration 1

- Contrôle des résistances de *pull-up*. Configurées en entrée, les lignes du module peuvent utiliser ou non des résistances de pull-up internet. La configuration se fait par logiciel via UDP comme le reste du contrôle du module.

PROTOCOLE

Dans sa configuration par défaut, le module tentera d'obtenir une adresse IP via DHCP. L'installation d'un serveur DHCP est donc indispensable. On attribuera son adresse au module de préférence en fonction de son adresse MAC. Dès lors, communiquer avec le module Elexol est un jeu d'enfant. Les commandes permettant le contrôle sont alphanumériques et envoyées via UDP.

On commencera par définir la direction du ou des ports utilisés en envoyant 0x21 (caractère « ! ») suivi d'une valeur identifiant le port A, B ou C (0x41, 0x42 ou 0x43) puis du masque de direction sur 8 bits, 0 pour une sortie et 1 pour une entrée. Ensuite, il suffit d'écrire le numéro du port 0x41, 0x42 ou 0x43 pour « A », « B » et « C » puis l'octet à écrire, pour activer ou désactiver une sortie. Si nous souhaitons ne changer qu'une seule ligne d'un port, il nous faudra, soit stocker dans notre programme l'état du port, soit le lire puis réécrire l'octet modifié.

La lecture n'est pas plus difficile. Il suffit en effet d'envoyer au module le nom du port en minuscule avec 0x61, 0x62 ou 0x63, respectivement « a », « b » ou « c ». On attend ensuite la réponse du module sur deux octets.

Denis Bodor :: db@ed-diamond.com :: lefinnois@lefinnois.net

Le premier est le nom du port en majuscule (0x41, 0x42 ou 0x43) et le second la valeur lue.

Si les résistances de pull-up sont activées via l'envoi de 0x40, du nom du port et d'un masque d'activation (1 non active et 0 active), la lecture d'un port sans aucune connexion donnera 0xFF. Toutes les lignes qui ne sont pas reliées à la masse sont ramenées au +5 V par les résistances.

LE CODE

Pour changer du reste du hors-série, le langage utilisé ici sera Perl et non C. Il faut également avouer que ce langage est particulièrement bien adapté aux communications réseau.

On commence par ouvrir un *socket* :

```
#!/usr/bin/perl

use IO::Socket::INET;
use Fcntl;

MySocket=new IO::Socket::INET->new(
  PeerPort=>2424,
  Proto=>'udp',
  PeerAddr=>'192.168.0.51')
  or die "Pas de Socket ! \n";
```

Nous pouvons ensuite dialoguer avec le module en passant, tout d'abord, la totalité du port B en sortie :

```
$msg=chr(0x21).chr(0x42).chr(0x00);
$MySocket->send($msg)
          or die "Pas de Send ! \n";
print("port B output\n");
```

Nous pouvons ensuite envoyer une donnée sur le port :

```
$msg=chr(0x42).chr(0x01);
$MySocket->send($msg)
          or die "Pas de Send ! \n";
print("port B 0xFF\n");
```

Pour lire le port A, nous procédons de même :

```
$msg=chr(0x21).chr(0x41).chr(0xFF);
$MySocket->send($msg)
          or die "Pas de Send ! \n";
print("port A input\n");
$msg=chr(0x61);
$MySocket->send($msg)
          or die "Pas de Send ! \n";
print("lecture port A\n");
$MySocket->recv($text,2)
          or die "Pas de Recv ! \n";
@bytes = split(//, $text);
printf "%c %02x\n", ord($bytes[0]),ord($bytes[1]);
```

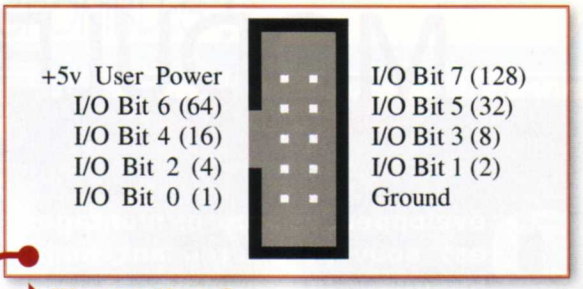


Illustration 2

Si nous souhaitons activer les résistances de pull-up sur le port A, il nous suffit d'ajouter le code suivant :

```
$msg=chr(0x40).chr(0x41).chr(0xFF);
$MySocket->send($msg)
          or die "Pas de Send ! \n";
print("pull-up port A\n");
```

UN MODULE POUR LE MODULE

En recherchant des exemples concrets d'utilisation du module Elexol, j'ai découvert qu'un développeur Perl avait créé un module spécifique au matériel en question. Chris Luke a écrit `Net::Elexol::EtherIO24` qui pourra être téléchargé sur http://www.flirble.org/chrisy/elexol/. Que les utilisateurs Debian se réjouissent, car il leur suffira d'utiliser les commandes suivantes :

```
% cd /tmp
% dh-make-perl --build \
  --cpan Net::Elexol::EtherIO24
% sudo dpkg -i \
  libnet-elexol-etherio24-perl_0.01-1_all.deb
```

Le code permettant d'accéder au module Elexol s'en trouve grandement simplifié :

```
#!/usr/bin/perl
use Net::Elexol::EtherIO24;
my $addr = "192.168.0.51";
my $eio = Net::Elexol::EtherIO24->new(target_addr=>$addr)
  or die "Pas de Socket ! \n";
$eio->set_line_dir(8,0);
$eio->set_line(8,1);
```

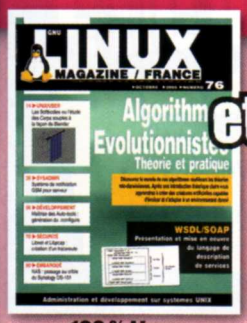
CONCLUSION

Le module Ether I/O 24 est relativement coûteux au regard des fonctionnalités offertes. Il vous en coûtera quelques 120 euros TTC, montant auquel il faut ajouter un système d'alimentation (le module accepte de 8 à 35 Volts pour 1.1 Watts) et toute la connectique nécessaire pour les ports d'E/S.

Il faut cependant prendre en compte qu'à ce prix on achète également une solution de facilité. Le temps et l'énergie ainsi gagnés pourront être utilisés pour l'électronique gravitant autour du module. C'est un choix personnel qu'il faudra faire en fonction des besoins du moment.

Offres de couplage

Lisez-vous RÉGULIÈREMENT :



100 % Linux

et / ou

100 % Sécurité

et / ou

100 % PRATIQUE

et / ou

Apprivoisez votre pingouin !

Si OUI, alors ces offres d'abonnement à tarif préférentiel vous sont destinées...

(1) Pour 11 Nos Linux Magazine + 6 Nos Linux Mag HS - (2) Pour 11 Nos Linux Magazine + 6 Nos MISC - (3) Pour 11 Nos Linux Magazine + 6 Nos MISC + 6 Nos Linux Mag. HS - (4) Pour 11 Nos Linux Magazine + 6 Nos MISC + 6 Nos Linux Mag. HS + 6 Nos Linux Pratique

OFFRE DE COUPLAGE À REMPLIR ET À RETOURNER À (OU PHOTOCOPIE)

*DIAMOND Éditions - LINUX MAGAZINE - 6 Rue de la Scheer - 67603 SÉLESTAT Cedex

☐ **OUI, je m'abonne et désire prof ter des offres spéciales de couplage**

Référence de l'offre :	Prix	Qté.	Total
☐ 11 Nos Linux Mag. + 6 Nos Linux Mag HS	79 €		
☐ 11 Nos Linux Mag. + 6 Nos MISC	83 €		
☐ 11 Nos Linux Mag. + 6 Nos MISC + 6 Nos Linux Mag HS	105 €		
☐ 11 Nos Linux Mag. + 6 Nos MISC + 6 Nos Linux Mag HS + 6 Nos Linux Pratique	129 €		
OFFRES VALABLES UNIQUEMENTS EN FRANCE MÉTRO.		TOTAL	

Pour les tarifs étrangers, consultez notre site : www.ed-diamond.com

LES 4 FAÇONS DE VOUS ABONNER !

- Par courrier postal en nous renvoyant le bon ci-dessous.
- Par le Web, sur notre site : www.ed-diamond.com.
- Par téléphone (paiement C.B.) entre 9h-12h & 14h -17h au 03 88 58 02 08.
- Par Fax au 03 88 58 02 09 C.B. et/ou bon de commande administratif

1 Voici mes coordonnées postales

Nom :

Prénom :

Adresse :

Code Postal :

Ville :

2 Je joins mon réglement :

☐ Je règle par chèque bancaire ou postal à l'ordre de Diamond Editions*

☐ Paiement par carte bancaire :

N° Carte :

Expire le :

Date et signature obligatoire : 200

Boostez votre collection !

Avez-vous L'ÂME du COLLECTIONNEUR ?

Vous recherchez un magazine en particulier? Allez sur www.ed-diamond.com pour voir le sommaire détaillé de chaque magazine et ensuite... Boostez votre collection avec les "POWER PACKS x5", soit 5 Linux Magazine pour 15€ et les "POWER PACKS x10", soit 10 Linux Magazine pour 25€ à choisir dans la liste ci-dessous :

LES 4 FAÇONS DE COMMANDER !

- Par courrier postal en nous renvoyant le bon ci-dessous.
- Par le Web, sur notre site : www.ed-diamond.com.
- Par téléphone (paiement C.B.) entre 9h-12h & 14h -17h au 03 88 58 02 08.
- Par Fax au 03 88 58 02 09 C.B. et/ou bon de commande administratif

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
N°07	Dopez Linux
N°08	Le futur résolument objet
N°09	Prêt pour le jeu !
N°10	The HURD : 100% GNU
N°11	Exclusif : l'avenir de G.N.O.M.E
N°12	NT et Linux : Guerre ou complément ?
N°13	Cryptage : la clé de la sécurité
N°14	XFree 4.0 : le futur à notre portée
N°15	Passez à la vitesse supérieure
N°16	OpenSources : Est-ce suffisant?
N°17	Linux : Système embarqué
N°18	Spécial interview : l'avenir de Linux
N°19	Dossier spécial : Postgre SQL 7.0
N°21	Le protocole Internet du 21e siècle : IPv6
N°22	Le multi-hreading : Une manière moderne de programmer le Multitâche
N°23	Débugger sous Linux
N°24	Palm et Linux
N°25	Kernel 2.4.0
N°26	<Dossier> XML </Dossier>
N°27	Les systèmes de fichiers journalisés
N°28	Scripting : la force d'Unix
N°29	L.F.S. Linux From Scratch
N°30	Le chiffrement des données
N°31	VPN et tunneling
N°32	Changez de coquille
N°34	XSL - FO : TeX Killer ?
N°35	QoS et iproute : optimisation et contrôle du trafic IP
N°36	Linux embarqué : Le projet mGlinux
N°37	L'impression sous Linux
N°38	Le desktop Shell : Enlightenment
N°39	Sécurité : Patchez votre noyau !
N°40	MySQL : la base de donnée OpenSource
N°41	Steganographieou l'art de la dissimulation de données
N°42	Développez vos pilotes de périphérique
N°43	Administrez facolement votre réseau SNMP
N°44	Comprenez NetBios pour Maîtriser l'interopérabilité windows GNU Linux
N°45	Cohabitation : UnDNS Bind dans un reseau Windows 2000
N°46	Debian : Utilisez Samba avec le support ACL
N°47	GNUstep : le petit frère de Mac OS X ?
N°48	Caudium, votre prochain serveur Web !
N°49	Après MySQL & PostgreSQL SAP DB : La base de données libre & puisante
N°50	Créez un album Photo avec PHP ...et sans MySQL
N°51	Boostez votre site Web avec XML grâce à XSLT, CSS & XPath
N°52	Linux Temps réel où en est-on aujourd'hui ?
N°53	Linux sur PDA : Linux dans votre poche !
N°54	Maîtrisez LVM : Présentation, installation & cohabitation
N°55	Intelligence Artificielle : Principes & programmation de jeux de stratégie classique
N°56	Développez vos applications Mozilla avec XPFE & XPCOM
N°57	Maîtrisez la gestion... Slots & Signaux ... des événements en C++
N°58	Djbdns enfin une alternative viable à BIND !
N°59	Zopix, Créez un CD "Live" Zope en 10 minutes !
N°60	JBoss serveur d'applications J2EE OpenSource
N°61	Découvrez MySQL 5 et les procédures stockées
N°62	Créez votre OS, principe et implémentation

NUMÉROS LINUX MAGAZINE ÉPUISÉS

N°01, N°02, N°03, N°04, N°05, N°20, N°33.

BON DE COMMANDE POWER PACKS À REMPLIR ET À RETOURNER À (OU PHOTOCOPIE)

**DIAMOND Editions - LINUX MAGAZINE - 6 Rue de la Scheer - 67603 SELESTAT Cedex

Cochez ici ▶ POWER PACKS X5

OUI, je désire acquérir un POWER PACK X5	1er 1PP* X5	2ème 2PP* X5	3ème 3PP* X5
1, Linux Magazine N°			
2, Linux Magazine N°			
3, Linux Magazine N°			
4, Linux Magazine N°			
5, Linux Magazine N°			
Total par série de POWER PACKS X5 :	15 €	30 €	45 €

Cochez ici ▶ POWER PACKS X10

OUI, je désire acquérir un POWER PACK X10	1er 1PP* X10	2ème 2PP* X10	3ème 3PP* X10
1, Linux Magazine N°			
2, Linux Magazine N°			
3, Linux Magazine N°			
4, Linux Magazine N°			
5, Linux Magazine N°			
6, Linux Magazine N°			
7, Linux Magazine N°			
8, Linux Magazine N°			
9, Linux Magazine N°			
10, 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.

SEULEMENT EN FRANCE MÉTROPOLITAINE !

TOTAL :	
Frais de port :	+3,81€
TOTAL :	

1 Voici mes coordonnées postales

Nom :

Prénom :

Adresse :

Code Postal :

Ville :

2 Je joins mon réglement :

☐ Je règle par chèque bancaire ou postal à l'ordre de Diamond Editions**

☐ Paiement par carte bancaire :

N° Carte :

Expire le :

Date et signature obligatoire : 200

EN KIOSQUE LE

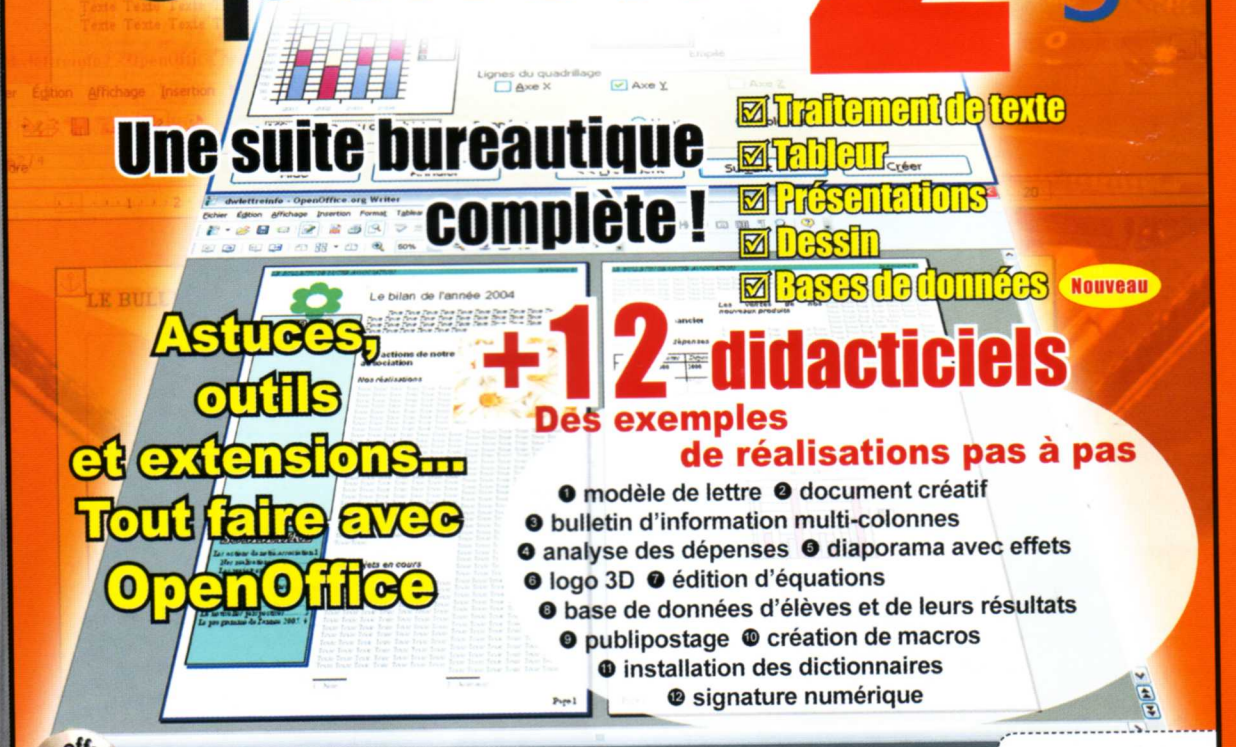
Presqu'OFFERT !

#40 décembre 2005-janvier 2006

France Métro : 7 Eur - BEL : 8 Eur - CAN : 11 SC

Nouvelle Version en français
pour Windows,
Linux, Mac OS X

OpenOffice.org 2



Une suite bureautique complète !

- ☑ Traitement de texte
- ☑ Tableur
- ☑ Présentations
- ☑ Dessin
- ☑ Bases de données

Nouveau

Astuces, outils et extensions... Tout faire avec OpenOffice

+12 didacticiels

Des exemples de réalisations pas à pas

❶ modèle de lettre ❷ document créatif
❸ bulletin d'information multi-colonnes
❹ analyse des dépenses ❺ diaporama avec effets
❻ logo 3D ❼ édition d'équations
❽ base de données d'élèves et de leurs résultats
❾ publipostage ❿ création de macros
⓫ installation des dictionnaires
⓬ signature numérique

CD offert !

Inclus OpenOffice 2 pour Windows/Linux /Solaris/FreeBSD/Macintosh

EN COLLABORATION AVEC LINUX PRATIQUE

25 NOVEMBRE

NOMS DE DOMAINE • EMAILS • HÉBERGEMENT • CRÉATION DE SITE • E-COMMERCE • RÉFÉRENCEMENT

A la naissance de vos projets, il y a le serveur dédié AMEN...

amen
IN WEB WE TRUST

baby

le serveur dédié le moins cher du marché

à 39 € HT/MOIS*
soit 46,64 € TTC/MOIS

- Linux
- Redhat/Fedora/Debian
- AMD Sempron 2200 ou Duron 1600
- IDE 80 Go/RAM 256 Mo
- Interface d'administration
- 1 adresse IP
- Trafic 1 To

livré en 2H**

> AUCUN FRAIS DE DOSSIER
> AUCUN FRAIS DE MISE EN SERVICE
> HAUTE DISPONIBILITE : 99,9 % !
> SUPPORT TECHNIQUE PAR MAIL ET PAR TELEPHONE
> SATISFAIT OU REMBOURSE***

Serveurs Dédiés AMEN en version Linux

SMALL	MEDIUM	LARGE
Athlon XP 2400 ou Sempron 2400	Duron ou Sempron 2800	Athlon XP 3000
IDE 80 Go / RAM 512 Mo	IDE 120 Go / RAM 768 Mo	RAID 2x160 Go / RAM 2 Go
2 adresses IP	4 adresses IP	4 adresses IP
Amen Data Backup 2 Go	Amen Data Backup 4 Go	Amen Data Backup 6 Go
Amen Reboot instantané	Amen Reboot instantané	Amen Reboot instantané
Interface Plesk 7	Interface Plesk 7	Interface Plesk 7
Trafic illimité[1]	Trafic illimité[1]	Trafic illimité[1]
~~69 €~~ 59 € HT/MOIS* 70,56 € TTC/MOIS	~~99 €~~ 89 € HT/MOIS* 106,44 € TTC/MOIS	~~169 €~~ 139 € HT/MOIS* 166,24 € TTC/MOIS

Nos 10 engagements*** : • satisfait ou remboursé • votre serveur en 2 H • surveillance réseau 24x7 • mise en service offerte • haute disponibilité 99,9 % • monitoring proactif 24x7 • interface d'administration en ligne • réseau redondant • bande passante garantie • aucun frais caché

www.amen.fr
.eu registrar

0892 55 66 77
0,34 € TTC/mn depuis la France 9H-19H