Connexion d'un afficheur sur port parallèle

Projet Aurore, 22 juillet 2004

1 Afficheur à base de HD44780 [1]

1.1 Aspects matériels

Rappel des conventions de notation : un symbole suivi d'un # ou surligné signifie que le signal associé est actif *bas*. Cela veut dire que la tension sur cette broche doit être nulle pour activer la fonctionnalité. Par exemple, une broche nommée R/W# se lit "lecture (Read) si le signal est haut" et "écriture (Write) si le signal est bas". De la même façon, $\overline{\text{RST}}$ se lit "Reset si le niveau est bas".

Ces afficheurs relativement courants sont basés sur le contrôleur Hitachi HD44780. Ils sont en général caractérisés par une ligne de 14 ou 16 broches dans un coin de l'afficheur. Les fonctions de ces broches sont les suivantes : [2, 3, 4, 5]

1	2	3	4	5	6	7 ... 14
Masse	+5 V	polarisation	RS	R/W#	E	D0 ... D7

La fonction de ces broches est :

- Masse : la masse !
- +5 V : l'alim
- polarisation : une tension qui est soit entre la masse et l'alim, soit entre la masse et -5 V (ça dépend des afficheurs)
- RS : transmission de données (1) ou d'instructions (0)
- R/W# : écriture (1) ou lecture (0)
- E : activation de l'afficheur

Il peut être pratique de connecter l'alimentation du LCD à l'alimentation du PC (entre les fils noir et rouge, +5 V) au moyen de fiches banane 2 mm. Le port parallèle est incapable de fournir par lui même le courant nécessaire à l'alimentation de l'afficheur.

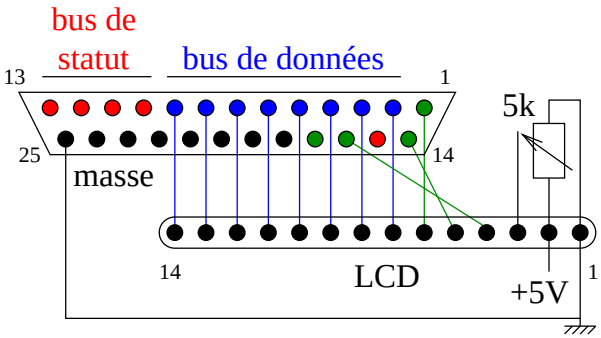
Deux modes de communication : 8 bits (rapide mais nécessite au moins 11 signaux au total, 8 de données et 3 de contrôle) ou 4 bits (pratique pour un microcontroleur : on ne mobilise qu'un port de 8 bits, soit 4 bits de données et 3 bits de contrôle).

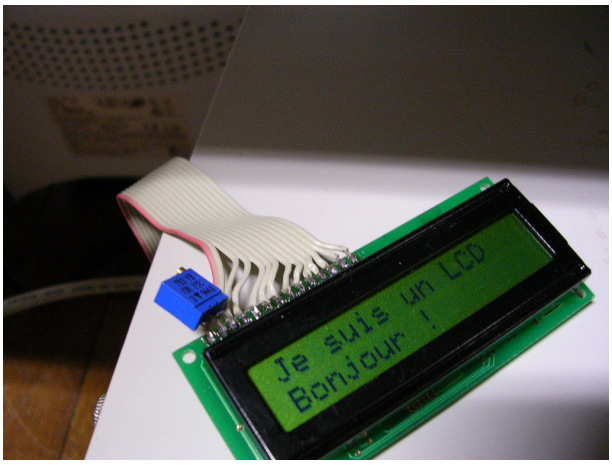
Nous utilisons ici le mode 8 bits puisque nous avons largement assez de broches disponibles sur le port parallèle (8 bits de données en sortie du port parallèle vers l'afficheur), 4 bits de port de contrôle (en sortie, open collector donc nécessitant à priori une résistance de pull-up de l'ordre de 5 kΩ à +5 V) et 5 bits de status dont nous ne nous servirons pas ici [6].

Le port parallèle se trouve généralement à l'adresse 0x378 (vérifier au moyen de `/proc/ioports` : les adresses possibles sont 0x378, 0x278 ou 0x3BC). Cela signifie que les 8 bits de données sont accessibles à l'octet d'adresse base=0x378, le 5 bits de contrôle dans l'octet base+2 et les bits de statut (inutiles ici) en base+1. Le bus de données se situe sur les broches 2 à 9, les broches 18 à 25 sont connectées à la masse, les bus de contrôle et de status se partageant les broches restantes sans ordre logique.

Nous reprenons ici la connexion entre le port parallèle et l'écran dit `winamp` : les broches 2 à 9 du bus de données du port parallèle sont connectées aux broches 7-14 de l'afficheur tandis que le bus de contrôle est branché de la façon suivante :

LCD	port parallèle
4	16
5	14
6	1





Sous linux, l'accès direct en ports d'entrée-sortie nécessite au préalable de mobiliser ce port pour notre programme au moyen de la commande `ioperm()`. Après, les ports ainsi réservés sont accessibles au moyen de `outb()` (out byte) avec, sous linux, un ordre des argument inversé par rapport aux autres systèmes d'exploitation !

1.2 Initialisation de l'afficheur

Avant toute programmation il faut vérifier que la polarisation a été ajustée de façon à voir apparaître à l'allumage la première ligne de caractères.

Les commandes (RS=1) suivants sont envoyées à l'afficheur pour l'initialiser :

- 0x01 pour réinitialiser l'écran
- 0x38 pour commander l'écran de se mettre en mode 8 bits de données, 2 lignes d'affichage, police 5×7 pixels
- 0x0C pour mettre en marche l'afficheur, éteindre le curseur et ne pas le faire clignoter
- 0x06 pour dire à l'afficheur d'incrémenter la position du curseur après affichage d'un caractère

1.3 Affichage et positionnement d'un caractère

L'affichage d'un caractère s'effectue en abaissant la ligne RS (envoi de données) et en activant E avec sur le bus de données le code ASCII du caractère à afficher (`man ascii`).

Le placement d'un caractère à un emplacement particulier se fait au moyen de la commande 0x80, RS étant à l'état haut, masqué par l'abscisse voulue (les abscisses de la première ligne commencent en 0x80, la seconde ligne en 0xC0).

1.4 Exemple de programme (Linux 2.4.22, gcc 3.3.3)

```
#include <stdio.h>            // CTRL PORT bit  3   2   1   0
#include <stdlib.h>           //               C3- C2+ C1- C0-
#include <unistd.h>           //      LCD           RS RW#  E
#include <asm/io.h>

#define base 0x378

void dela()
{usleep(100);
}

void funct(char d)                  // funct: RS=0, R/W#=0, E=1, E=0, R/W#=0
{outb(d,base);                                          // outb(val,port)
 outb(0x02,base+2); // RS=0, R/W#=0, E=1 ecrit avec LCD actif
 dela();
 outb(0x03,base+2); // RS=0, R/W#=0, E=0 desactive le LCD
```

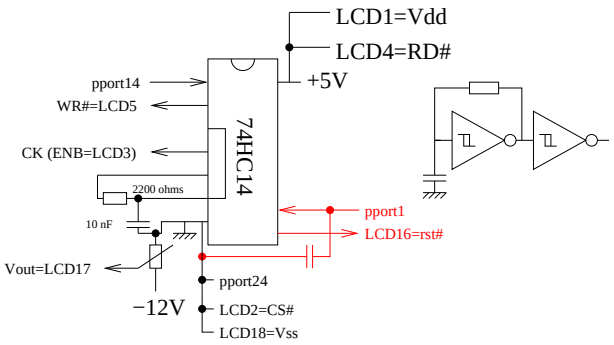
```
 dela();
}

void gotoxy(char x,char y) // A FINIR: 0x80+x si y=1, 0xC0+x si y=2
{if (x<16)                          // 0x80=home line 1, 0x0C=home line 2
    if (y==1) funct(0x80+x); else funct(0xC0+x);
 else funct(0x80);
}

void chara(char d)                // funct: RS=0, R/W#=0, E=1, E=0, R/W#=0
{outb(d,base);                                          // outb(val,port)
 outb(0x06,base+2);dela(); // RS=1, R/W#=0, E=1     ecrit la char
 outb(0x07,base+2);dela(); // RS=1, R/W#=0, E=0     desactive le LCD
}

void stra(char *buf)
{int i=0;
 while (buf[i]!=0) {chara(buf[i]);i++;}
}

int main()
{unsigned char i;char buf[255];
 ioperm(0x378,3,1);
 outb(0x03,base+2);      // set ctrl port: RS[C2]=0, R/W#[C1#]=0, E[C0#]=0
 dela();
 funct(0x01);            // funct 0x01 : clear display
 funct(0x38);            // funct 0x38 : 8 bits, 2 lines, 5x7 font [8]
 funct(0x0C);            // funct 0x0C : display ON, cursor OFF, blink curs. OFF
 funct(0x06);            // funct 0x06 : inc cursor after display
 sprintf(buf,"Hello World !");gotoxy(0,2);stra(buf);
 sprintf(buf,"Test de LCD");gotoxy(2,1);stra(buf);
 for (i=32;i<=254;i++) {printf("%d\n",i);gotoxy(0,1);chara(i);usleep(100);}
 printf("Enter ASCII val to be displayed:\n");
 while (1) {scanf("%d",&i);gotoxy(0,1);chara(i);usleep(100);}
}
```

Ce montage fonctionne avec `lcdmod` (version 1.0.0) après avoir configuré `config.h` avec l'option `WIRING_STUART`. Il est alors possible d'afficher un texte sur le LCD au moyen de `/dev/lcd`.

2 Écran LCD Epson

2.1 Aspects matériel et logiciel

Il s'agit d'écrans qui en plus de prendre des signaux très similaires à ceux utilisés pas ceux à base de HD44780 (RS s'appelle maintenant A0, et R/W# est séparé en RD# et WR#), nécessitent un oscillateur externe entre 500 et 2000 kHz. Nous avons dans un premier temps utilisé un oscillateur TTL prêt à l'emploi à 1.8 MHz [7, 8], pour ensuite le remplacer par un oscillateur RC autour d'un trigger de schmitt 74HC14 (ou 74HCT14 selon disponibilité) tel que preésenté ci-dessous. La constante de temps de cet oscillateur est de l'ordre de $R \times C$, et un second trigger de schmitt sert de buffer à l'oscillateur pour former des créneaux avec des front compatibles avec les besoins du LCD. Noter que la connexion des signaux de controle du LCD Epson, décrite ci dessous et dans [8], diffère considérablement de celle utilisée pour le LCD Hitachi.



LCD6(A0)=pport16
LCD7−LCD14=pport2−pport9
LCD15=NC

Un problème avec ces afficheurs et qu'au lieu de travailler sur les fronts ils travaillent sur les niveaux, et que par conséquent un signal WR# trop long résulte dans l'affichage de plusieurs caractères consécutifs (communication asynchrone). Le délai est très court (25 temps d'horloge, soit de l'ordre de 12.5 μs à 50 μs selon la fréquence de quartz utilisée) et doit être ajusté pour chaque nouveau PC si le délai est implémenté sous forme de boucle vide. Une méthode plus propre serait la lecture du statut (D7 : occupé ou libre) pour définir quand le signal d'écriture est remis à un niveau haut. Nous avons été obligé de compenser les signaux inversés issus du port de controle en les connectant au LCD via une porte inverseuse : nous utilisons pour cela des portes vides du 74HC14 qui, en plus de servir de trigger de schmitt, peut aussi servir de porte inverseuse. Ceci est dû à une gestion apparemment étrange du port de controle par linux qui force le niveau de repos du port de controle au bout d'un certain temps, quelle que soit la valeur envoyée sur le port de controle. Le condensateur entre la broche 17 du port parallèle et la masse a été ajoutée suite à l'observation de pics de tension sur cette ligne qui provoquaient des resets intempestifs.

Les commandes au LCD sont sensiblement les mêmes que pour le Hitachi, l'envoi et le positionnement de caractères est strictement identique au protocole Hitachi donc tout logiciel utilisant ces derniers doit être facilement adaptable aux Epsons. Une différence majeure est l'absence de la CGRAM qui interdit donc avec ce LCD de définir de nouveaux caractères. Bien que les caractères ASCII soient tous présents, le jeu de caractère du LCD Epson est un peu réduit par rapport à celui du Hitachi puisqu'il ne commence qu'avec la valeur 32 et présente des caractères différents pour les valeurs supérieurs à 128.

Un signal propre de Reset est ici fondamental : contraitement aux Hitachis qui s'initialisent à la mise sous tension, le LCD Epson nécessite un front propre pour son initialisation faute de quoi les caractères ne seront pas correctement alignés à l'affichage.

2.2 Exemple de programme (Linux 2.4.22, gcc 3.3.3)

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

//E   =C0-=pin1   (n'existe pas -- pourrait etre RST#)
//R/W#=C1-=pin14  (W#)
//RS=  C2+=pin 16 (A0)

#define base 0x378

#define tmax 400

void dela()
{  int i;for (i=0;i<tmax;i++) ; // usleep(1)
}
```

```
void ldela()
{usleep(1);}

void funct(char d)            // funct: RS=0, W#=1 -> W#=0
{outb(d,base);                // outb(val,port)
 outb(0x00+1,base+2); dela();   // RS=0, R/W#=0
 outb(0x02+1,base+2);ldela();
}

void gotoxy(char x,char y)    // A FINIR: 0x80+x si y=1, 0xC0+x si y=2
{if (x<16)                    // 0x80=home line 1, 0x0C=home line 2
    if (y==1) funct(0x80+x); else funct(0xC0+x);
 else funct(0x80);
}

void chara(char d)            // funct: RS=1, R/W#=0, E=1, E=0, R/W#=0
{outb(d,base);                // outb(val,port)
 outb(0x06+1,base+2);dela();    // RS=1, R/W#=1 -- delay MUST be short
 outb(0x04+1,base+2);dela();    // RS=1, R/W#=0
 outb(0x06+1,base+2);ldela();   // RS=1, R/W#=1
}

void stra(char *buf)
{int i=0;
 while (buf[i]!=0) {chara(buf[i]);i++;}
}

int main()
{unsigned char i;char buf[255];
 ioperm(0x378,3,1);
 outb(0x02,base+2);           // set ctrl port: RS[C2]=0, R/W#[C1#]=0, E[C0#]=0
 ldela();
 funct(0x10);                 // funct 0x10 : system RESET
 funct(0x01);                 // funct 0x01 : clear display data
 funct(0x04);                 // funct 0x04 : inc cursor pos
 funct(0x08);                 // funct 0x08 : set cursor font 08=ul+09=blinking
 funct(0x0f);                 // funct 0x0f : set cursor 0x0e=off/0x0f=on
 funct(0x0d);                 // funct 0x0d : whole display ON
 funct(0x0b);                 // funct 0x0b : blinking cursor 0x0a=off/0x0b=on
 // funct(0x06);              // funct 0x06 : inc cursor after display
 sprintf(buf,"Hello World !");gotoxy(0,2);stra(buf);
 sprintf(buf,"Test de LCD");gotoxy(2,1);stra(buf);
 for (i=32;i<=254;i++) {printf("%d\n",i);gotoxy(0,1);chara(i);usleep(100);
     usleep(100);
 }
 printf("Enter ASCII val to be displayed:\n");
 while (1) {scanf("%d",&i);gotoxy(0,1);chara(i);usleep(100);}
}
```

Références

[1] GNU Linux Magazine, Avril 2004, num. 60, pp.48-51

[2] http://www.doc.ic.ac.uk/~ih/doc/lcd/

[3] http://home.iae.nl/users/pouweha/lcd/lcd.shtml

[4] http://www.repairfaq.org/filipg/LINK/F_LCD_HD44780.html

[5] http://www.myrolypoly.com/lcd_project/lcd_project.html

[6] http://www.repairfaq.org/filipg/LINK/PORTS/F_PARALLEL1.html
[7] http://www.ele.tut.fi/~vvieri/ele/lcd.htm
[8] http://home.nikocity.de/woe/lcd/ea_x_series.pdf