Driver Linux

Création dynamique et IOCTL

Création d'un device "manuel"

Création

C	module_init	major = register_chrdev(0, name, fops);
$	insmod module.ko	créer un module driver caractère le numéro ***major*** est placé dans /proc/devices : major name
	mknod /dev/name c ***major*** 0	créer le fichier spécial dans le système de fichier

Destruction

C	module_exit	unregister_chrdev(major, name);
$	rmmod module	supprime le module driver
	rm /dev/name	efface le noeud dans le système de fichier

Création d'un device "scripté"

L'insertion d'un module driver passe par un script bash qui

- charge le module driver
- recherche le major dans /proc/devices avec une commande awk
- créer le device

Le script **insdev** suppose que le nom du module est le même que le nom du driver

insdev

```
#!/bin/sh
module=$1
shift
/sbin/insmod ./$module.ko $* || exit 1
rm -f /dev/$module
major=$(awk "\$2==\"$module\" {print \$1}" /proc/devices)
mknod /dev/$module c $major 0
chmod 666 /dev/$module
```

$ sudo insdev device params…

destruction d'un device "scripté"

L'effacement d'un module driver passe par un script bash qui

- décharge le module driver
- efface le device

rmdev suppose que le nom du driver est le même que le nom du module

rmdev

```
#!/bin/sh
module=$1

/sbin/rmmod $module  || exit 1
rm -f /dev/$module
```

```
$ sudo rmdev driver
```

Création dynamique device au chargement

Types

dev_t	Entier contenant les numéro major/minor
	int MAJOR(dev_t dev);
	int MINOR(dev_t dev);

struct cdev	Structure réprésentant un device caractère

Prototypes

```
int alloc_chrdev_region(   // enregistre des numéros de device
                        dev_t *dev, unsigned int firstminor,
                        unsigned int count, char *name)
void cdev_init          (   // crée la structure device et ses fonctions
                        struct cdev *cdev, struct file_operations *fops);
int cdev_add            (   // ajoute la structure dans le kernel
                        struct cdev *cdev, dev_t dev, unsigned int count);
devfs_mk_cdev           (   // crée le fichier
                        dev_t dev, int mode , char *name);
```

Création dynamique device au chargement

Le but de créer le noeud dans le système de fichier au chargement du module

dans module_init

```
#include <linux/cdev.h>
    dev_t lcd_dev;                  → identifiant MAJOR:MINOR
    struct cdev lcd_cdev;           → structure interne représentant un device caractère
    struct file_operations lcd_fops = {...} ;  → vecteurs d'opérations

    alloc_chrdev_region(&lcd_dev, 0, 1, "lcd"); → alloue les numéros
    cdev_init(&lcd_cdev, &lcd_fops);            → init le device (fonctions)
    cdev_add(&lcd_cdev, lcd_dev, 1);            → connecte le device
    devfs_mk_cdev(lcd_dev, S_IFCHR | S_IRUSR | S_IWUSR, "lcd");
                                                → cree le noeud ds les fichiers
```

Destruction dynamique au déchargement

L'effacement est également dynamique

dans module_exit

```
devfs_remove("lcd");
cdev_del(lcd_cdev);;
unregister_chrdev_region(lcd_dev, 1);
```

```
devfs_remove(char *name);
cdev_del(strut cdev cdev);;
unregister_chrdev_region(dev_t dev, unsigned count);
```

Allocation dynamique pour le fichier

La structure `file` dans le noyau correspond au descripteur de fichier ouvert par la fonction `open` de l'utilisateur

```
fd = open("filename", mode);
```

Les fonction `read, write,` etc prennent `fd` en paramètre.

Toutes les fonctions du driver reçoivent `file`

```
static int
my_open_function(struct inode *inode, struct file *file) {
    printk(KERN_DEBUG "open()\n");
    return 0;
}

static ssize_t
my_write_function(struct file *file, const char *buf, size_t count, loff_t *ppos) {
    printk(KERN_DEBUG "write()\n");
    return 0;
}
```

Allocation dynamique pour le fichier

La structure file contient un champ void *private_date

```
static int
my_open_function(struct inode *inode, struct file *file) {
    printk(KERN_DEBUG "open()\n");
    file->private_data = kmalloc(SIZE, GFP_KERNEL);
    return 0;
}

static int
my_release_function(struct inode *inode, struct file *file) {
    printk(KERN_DEBUG "close()\n");
    kfree(file->private_data);
    return 0;
}
```

appel système ioctl

```
struct file_operations fops =
{
    .owner   = THIS_MODULE,        /* pointeur sur le module courrant */
    .open    = my_open_function,
    .read    = my_read_function,
    .write   = my_write_function,
    .ioct    = my_ioct_function,
    .release= my_release_function  /* appelée par le dernier close */
};
```

IOCTL : input output control

- permet de faire des opérations qui ne peuvent pas être faites par les autres appels
- par exemple, dans le cas du LCD, positionner le curseur

appel de ioctl par l'utilisateur

Appel système coté utilisateur

err = ioctl (fd, cmd, arg...)

- fd file descriptor
- cmd unsigned long doit être **unique** dans le système
- arg liste d'argument optionnels
- err 0 ou -1 et errno en cas d'erreur

- type : doit être différent pour chaque pilote
- num : numéro d'ordre de la commande
- taille : quantité de données échangées
- sens : sens des échanges de données, par rapport au programme utilisateur

Fabrication du paramètre cmd

cmd est obtenu par des macros de sys/ioctl.h

- _IO (type, num)
- _IOW (type, num, taille)
- _IOR (type, num, taille)
- _IOWR (type, num, taille)

Le sens du transfert est du point de vue de l'application

Pour garantir l'unicité le type doit être choisi après consultation du fichier

⇒ linux/Documentation/ioctl/ioctl-number.tx

- num est un nombre séquentiel
- taille c'est la quantité de données échangées

```
#define   TIOCSETAF _IOW('t', 22, struct termios) /* drn out, fls in, set */
#define   TIOCGETD  _IOR('t', 26, int) /* get line discipline */
```

Convention de nommage de la commande

DRIVER_NAME_IOCXXXXX

- X — Type d'opération
- XXXX — Nom de la commande

Si l'argument est entier

- Tell: — donne l'argument
- Query: — la réponse est dans la valeur de retour
- sHift: — T + Q atomique

Si l'argument est un pointeur

- Set: — définir
- Get: — obtenir
- eXchange — G + S atomique

Paramètre arg

unsigned long

Absent si rien à échanger

On a toute liberté sur la signification et l'utilisation de donnée fournie au pilote

- Adresse de données fournies au pilote
- Adresse à laquelle le pilote renvoie des données
- Pointeur sur une structure
- Valeur entière

Les adresses sont dans l'espace utilisateur

Gestion de ioctl coté noyau

```
static int
lcd_ioctl( struct inode *inodep, struct file *filep,
           unsigned int cmd, unsigned long arg)
```

La commande et l'argument sont ceux de l'utilisateur

On doit vérifier la validité de la commande

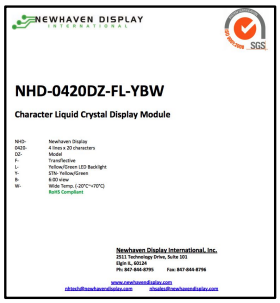
- _IOC_DIR(cmd)
- _IOC_TYPE(cmd)
- _IOC_NR(cmd)
- _IOC_SIZE(cmd)

Décodage de la commande

```
static int
peri_ioctl(struct inode *inodep,struct file *filep,unsigned int cmd, unsigned long arg)
{
  unsigned long err = 0;
  unsigned long n,z,i;
  struct lcd_data  *p = filep->private_data;

  switch(cmd)
     {
       case  LCD_IOCCLEAR :
                filep->f_pos = 0;
                p->nbcar = sizeof(data);
                for(i=0;i<sizeof(data);i++) p->mess[i] = data[i];
                return 0;

       case  LCD_IOCTCURPOS :
                ...
                return 0;

       case  LCD_IOCSDATA :
                ...
                return 0;

       default :
                return -EINVAL;     /* Invalid argument   */
     }
}
```

Pilotage du LCD en mode USR

Présentation du LCD


NEWHAVEN DISPLAY INTERNATIONAL

SGS

NHD-0420DZ-FL-YBW

Character Liquid Crystal Display Module

NHD- Newhaven Display
0420- 4 lines x 20 characters
DZ- Version
FL- Transflective
Y- Yellow/Green
B- 6:00 view
W- Wide Temp (-20°C~+70°C)
RoHS Compliant

Newhaven Display International, Inc.
2511 Technology Drive, Suite 101
Elgin, IL, 60124
Ph: 847-844-8795 Fax: 847-844-8796

www.newhavendisplay.com

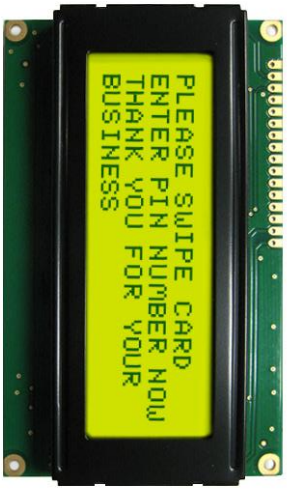
http://www.newhavendisplay.com/specs/NHD-0420DZ-FL-YBW.pdf

→ http://www.newhavendisplay.com/app_notes/ST7066U.pdf
http://en.wikipedia.org/wiki/Hitachi_HD44780_LCD_controller

Dans une documentation, on trouve

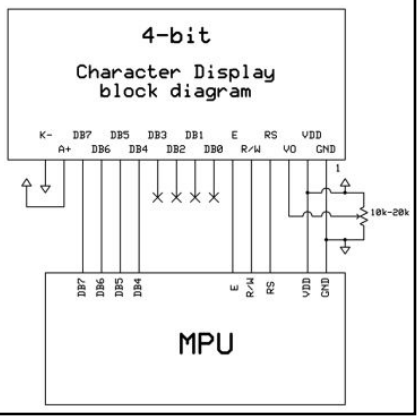
- La connectique
- Un schéma interne du ou des circuits de contrôle
- Les instructions
- Les chronogrammes pour l'envoi des commandes ou la lecture des données internes
- Les séquences spécifiques
  - Démarrage, arrêt, etc
- Des morceaux de codes d'usage

Connectique du LCD



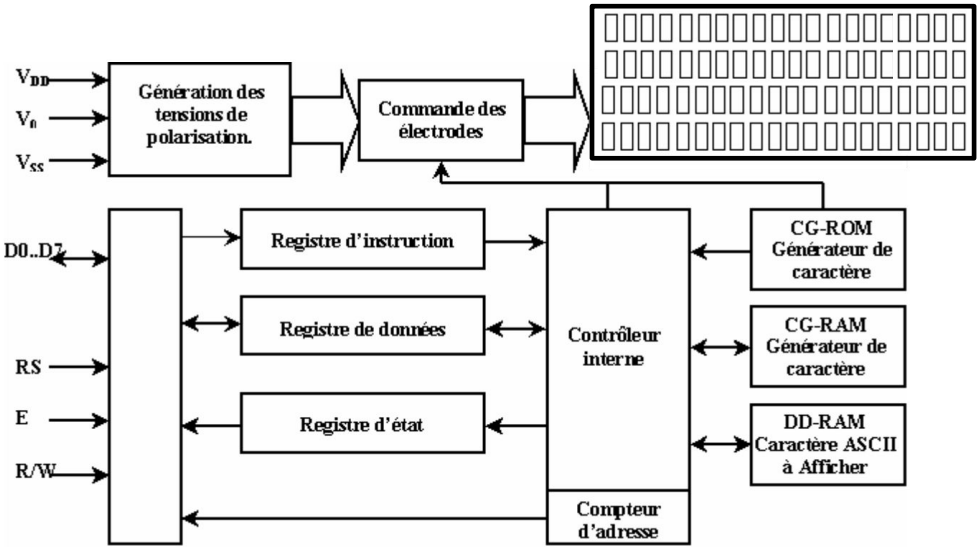
Pin No.	Symbol	External Connection	Function Description
1	Vss	Power Supply	Ground
2	VDD	Power Supply	Supply voltage for logic (+5.0V)
3	V0	Power Supply	Power supply for contrast (approx. 0.5V)
4	RS	MPU	Register select signal. RS=0: Command, RS=1: Data
5	R/W	MPU	Read/Write select signal, R/W=1: Read R/W:=0: Write
6	E	MPU	Operation enable signal. Falling edge triggered.
7-10	DB0-DB3	MPU	Four low order bi-directional three-state data bus lines. These four are not used during 4-bit operation.
11-14	DB4-DB7	MPU	Four high order bi-directional three-state data bus lines.
15	LED+	Power Supply	Power supply for LED Backlight (+5.0V via on-board resistor)
16	LED-	Power Supply	Ground for backlight

- 4 lignes de 20 caractères
- 2 modes : 4 bits et 8 bits
- Fonts programmables
- Mémoire interne en R/W



http://www.newhavendisplay.com/specs/NHD-0420DZ-FL-YBW.pdf

Contrôleur interne du LCD



- Le contrôleur du LCD est un ST7066U compatible avec le HD44780
  http://www.newhavendisplay.com/app_notes/ST7066U.pdf
  http://en.wikipedia.org/wiki/Hitachi_HD44780_LCD_controller

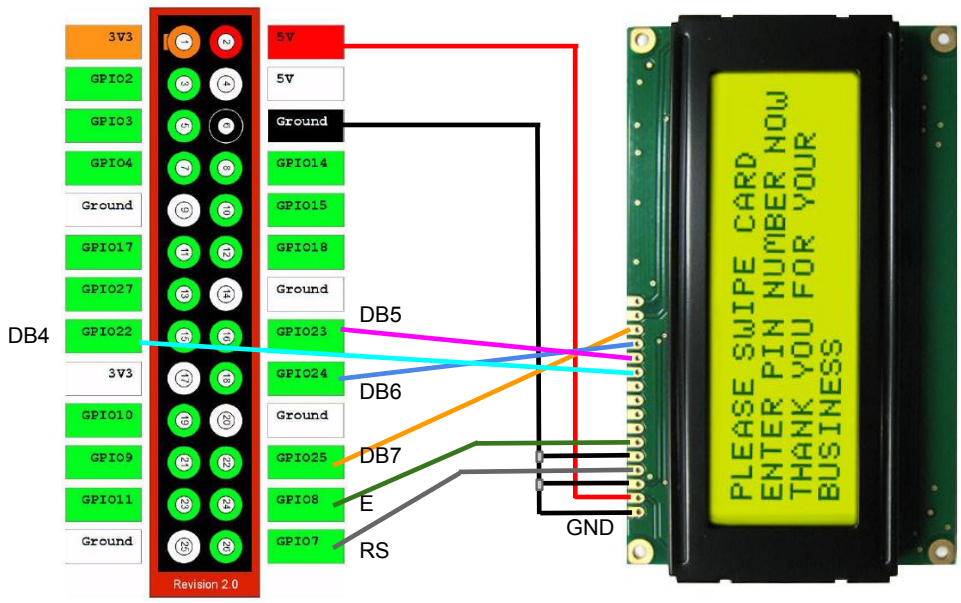
Jeu d'instructions

Instruction	Code										Description	Execution time (max) (when f_{cp} = 270 kHz)
	RS	R/W	B7	B6	B5	B4	B3	B2	B1	B0		
Clear display	0	0	0	0	0	0	0	0	0	1	Clears display and returns cursor to the home position (address 0).	1.52 ms
Cursor home	0	0	0	0	0	0	0	0	1	*	Returns cursor to home position. Also returns display being shifted to the original position. DDRAM content remains unchanged.	1.52 ms
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction (I/D); specifies to shift the display (S). These operations are performed during data read/write.	37 µs
Display on/off control	0	0	0	0	0	0	1	D	C	B	Sets on/off of all display (D), cursor on/off (C), and blink of cursor position character (B).	37 µs
Cursor/display shift	0	0	0	0	0	1	S/C	R/L	*	*	Sets cursor-move or display-shift (S/C), shift direction (R/L). DDRAM content remains unchanged.	37 µs
Function set	0	0	0	0	1	DL	N	F	*	*	Sets interface data length (DL), number of display line (N), and character font (F).	37 µs
Set CGRAM address	0	0	0	1	CGRAM address						Sets the CGRAM address. CGRAM data are sent and received after this setting.	37 µs
Set DDRAM address	0	0	1	DDRAM address							Sets the DDRAM address. DDRAM data are sent and received after this setting.	37 µs
Read busy flag & address counter	0	1	BF	CGRAM/DDRAM address							Reads busy flag (BF) indicating internal operation being performed and reads CGRAM or DDRAM address counter contents (depending on previous instruction).	0 µs
Write CGRAM or DDRAM	1	0	Write Data								Write data to CGRAM or DDRAM.	37 µs
Read from CG/DDRAM	1	1	Read Data								Read data from CGRAM or DDRAM.	37 µs

Instruction bit names —

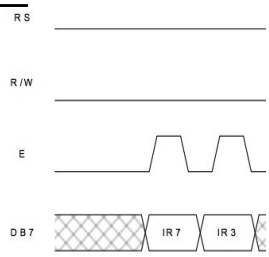
I/D - 0 = decrement cursor position, 1 = increment cursor position; **S** - 0 = no display shift, 1 = display shift; **D** - 0 = display off, 1 = display on; **C** - 0 = cursor off, 1 = cursor on; **B** - 0 = cursor blink off, 1 = cursor blink on ; **S/C** - 0 = move cursor, 1 = shift display; **R/L** - 0 = shift left, 1 = shift right; **DL** - 0 = 4-bit interface, 1 = 8-bit interface; **N** - 0 = 1/8 or 1/11 duty (1 line), 1 = 1/16 duty (2 lines); **F** - 0 = 5×8 dots, 1 = 5×10 dots; **BF** - 0 = can accept instruction, 1 = internal operation in progress.

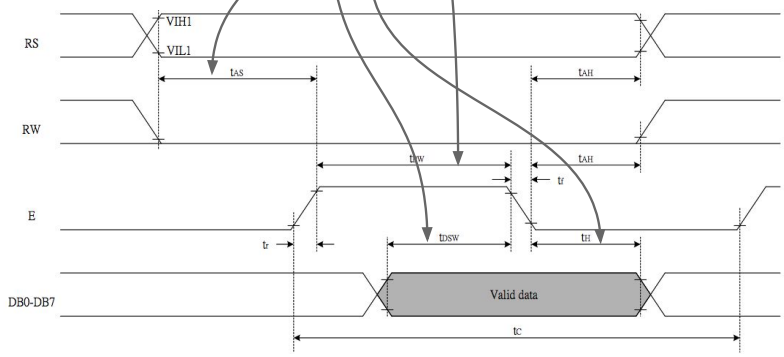
Connexion avec la Raspberry Pi



Séquence d'écriture

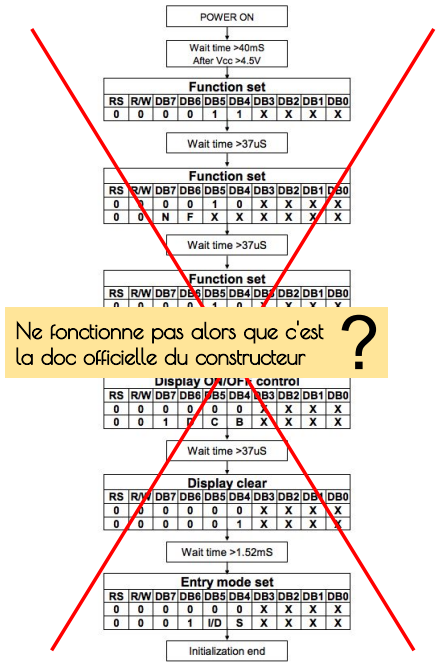
Item	Symbol	Min	Typ	Max	Unit
Enable cycle time	t_{cycE}	1000	—	—	ns
Enable pulse width (high level)	PW_{EH}	450	—	—	
Enable rise/fall time	t_{Er}, t_{Ef}	—	—	25	
Address set-up time (RS, R/$\overline{W}$ to E)	t_{AS}	60	—	—	
Address hold time	t_{AH}	20	—	—	
Data set-up time	t_{DSW}	195	—	—	
Data hold time	t_H	10	—	—	



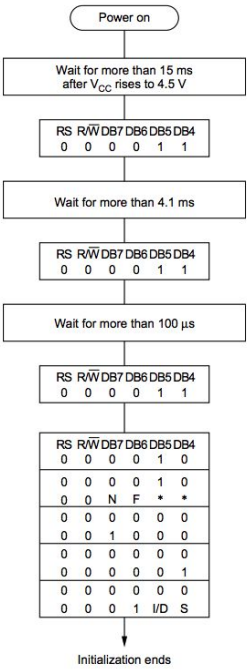


https://www.sparkfun.com/datasheets/LCD/HD44780.pdf

Initialisation



Ne fonctionne pas alors que c'est la doc officielle du constructeur ?



http://www.newhavendisplay.com/app_notes/ST7066U.pdf

https://www.sparkfun.com/datasheets/LCD/HD44780.pdf

Séquence d'écriture en C

```
void command(char i)
{
      P1 = i;      //put data on output Port
      D_I =0;      //D/I=LOW : send instruction
      R_W =0;      //R/W=LOW : Write
      Nybble();    //Send lower 4 bits
      i = i<<4;    //Shift over by 4 bits
      P1 = i;      //put data on output Port
      Nybble();    //Send upper 4 bits
}
```

```
void write(char i)
{
      P1 = i;      //put data on output Port
      D_I =1;      //D/I=HIGH : send data
      R_W =0;      //R/W=LOW : Write
      Nybble();    //Clock lower 4 bits
      i = i<<4;    //Shift over by 4 bits
      P1 = i;      //put data on output Port
      Nybble();    //Clock upper 4 bits
}
```

P1 : 8 bits de data
seuls les 4 bits msb comptent
D_I : RS

```
//enable pulse width  >= 300ns
//Clock enable: falling edge
void Nybble()
{
      E = 1;
      Delay(1);
      E = 0;
}
```

```
void init()
{
      P1 = 0;
      P3 = 0;
      Delay(100);        //Wait >15 msec after power is applied
      P1 = 0x30;         //put 0x30 on the output port
      Delay(30);         //must wait 5ms, busy flag not available
      Nybble();          //command 0x30 = Wake up
      Delay(10);         //must wait 160us, busy flag not available
      Nybble();          //command 0x30 = Wake up #2
      Delay(10);         //must wait 160us, busy flag not available
      Nybble();          //command 0x30 = Wake up #3
      Delay(10);         //can check busy flag now instead of delay
      P1= 0x20;          //put 0x20 on the output port
      Nybble();          //Function set: 4-bit interface
      command(0x28);     //Function set: 4-bit/2-line
      command(0x10);     //Set cursor
      command(0x0F);     //Display ON; Blinking cursor
      command(0x06);     //Entry Mode set
}
```

http://www.newhavendisplay.com/specs/NHD-0420DZ-FL-YBW.pdf

Commandes

Il vous faudra vérifier les commandes de gestion de l'écran.

constant DDRAM[4]={0,0x40,0x14,0x54};

- CLR_DISP 0x01
- DISP_ON 0x0C
- DISP_OFF 0x08
- CUR_HOME 0x02
- CUR_OFF 0x0C
- CUR_ON_UNDER 0x0E
- CUR_LEFT 0x10
- CUR_RIGHT 0x14
- CUR_SET(l,c) 0x80+DDRAM[(l)%4]+(c)%0x14

Organisation de la DDRAM (Display Data RAM)

0x00 ..... Ligne 1 ..... 0x13 0x14 ..... Ligne 3 ..... 0x27

0x40 ..... Ligne 2 ..... 0x53 0x54 ..... Ligne 4 ..... 0x67