LKM hd44780 lcd

0.1

Generated by Doxygen 1.8.8

Wed Jun 15 2016 09:49:36

Contents

Chapter 1

lcd-kernel-driver

lcd kernel driver for 16x2 HD44780

helpful link to compile a kernel module: https://github.com/diederikdehaas/cknow.wiki/blob/master/rpi/compile-kernel-module-on-raspberrypi.md

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

File Documentation

3.1 /home/pi/SysProg/lcd-driver/hd44780.c File Reference

loadable kernel module character device driver for support a simple 2x16 lcd display. within the lcd display is a commonly used HD44780 controller implemented.

```
#include <linux/module.h>
#include <linux/moduleparam.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/fs.h>
#include <linux/cdev.h>
#include <linux/types.h>
#include <linux/device.h>
#include <linux/gpio.h>
#include <linux/delay.h>
#include <linux/ctype.h>
#include <asm/uaccess.h>
#include <asm/errno.h>
#include <linux/wait.h>
#include <linux/hrtimer.h>
#include <linux/ktime.h>
#include <linux/interrupt.h>
#include "ioctl_header.h"
```

Include dependency graph for hd44780.c:

Functions

- MODULE_AUTHOR ("Daniel Obermaier <mailto:dan.obermaier@gmail.com>, Victor Nagy <mailto:victor.↩ nagy@hotmail.de>")

  < *core header for loading LKMs into the kernel*
- MODULE_DESCRIPTION ("driver for LCD Display with HD44780 controller")
- MODULE_LICENSE ("GPL")
- MODULE_VERSION ("0.1")
- static void write_nibble (int regist, int value)

  writes 4-bit values to gpio

- static void write_lcd (int regist, int value)

  uses write_nibble to shift and reverse logic values
- static int gpio_request_output (int nr)

  checks if gpio can be written successfully
- static int exit_display (void)

  frees memory on every initialized gpio pin if module gets unloaded from kernel
- static void __exit mod_exit (void)

  lkm exit function
- static int __init init_display (void)

  initializes display as soon as the module is loaded into the kernel
- static int dev_open (struct inode *inode, struct file *fp)

  executed from user via userspace interface program and increment device counter
- static int dev_release (struct inode *inode, struct file *fp)

  executed from user via userspace interface program and decrement device counter
- static ssize_t dev_write (struct file *instance, const char __user *user, size_t cnt, loff_t *offset)

  function is called when device is being written from user space
- module_param (major, int, S_IRUGO|S_IWUSR)
- module_param (count, int, S_IRUGO|S_IWUSR)
- static int __init mod_init (void)

  kernel module initialization function
- module_init (mod_init)

  a module must use the module_init() and module_exit() macros from linux/init.h
- module_exit (mod_exit)

Variables

- static int major = 0

  major number as an lkm identifier
- static int minor = 0
- static int count = 1
- dev_t dev = 0

  variable for lcd device
- static int dev_cnt = 0

  used to prevent multiple access
- static struct cdev * driver_object
- static struct class * hd44780_class
- static struct device * hd44780_dev
- static char textbuffer [1024]
- static struct file_operations fops

  Devices are represented as file structures in the kernel.

3.1.1 Detailed Description

loadable kernel module character device driver for support a simple 2x16 lcd display. within the lcd display is a commonly used HD44780 controller implemented.

Author

Daniel Obermaier Victor Nagy Markus Fischer

Date

01. June 2016

Version

0.1

See also

https://www.sparkfun.com/datasheets/LCD/HD44780.pdf datasheet for this hd44780 controller

3.1.2 Function Documentation

3.1.2.1 static int dev_open (struct inode * *inode*, struct file * *fp*) [static]

executed from user via userspace interface program and increment device counter

will be called if device will be opened

Parameters

inode	to device file
pointer	to device file

```
297                                                     {
298     printk(KERN_INFO "hd44780: device opened from user\n");
299
300     dev_cnt++;    //increment counter
301
302     return 0;
303 }
```

3.1.2.2 static int dev_release (struct inode * *inode*, struct file * *fp*) [static]

executed from user via userspace interface program and decrement device counter

will be called on closing the device

Parameters

inode	to device file
pointer	to device file

```
312                                                     {
313     printk(KERN_INFO "hd44780: device closed from user\n");
314
315     dev_cnt--;    //decrement counter
316
317     return 0;
318 }
```

3.1.2.3 static ssize_t dev_write (struct file * *instance*, const char __user * *user*, size_t *cnt*, loff_t * *offset*) [static]

function is called when device is being written from user space

Parameters

pointer	to a file instance
buffer	contains the string to write onto the device
size	of array that is being passed in the const char buffer
offset	if required

Returns

number of characters left

```
330                                                                                                 {
331
332     unsigned long not_copied;
333     unsigned long to_copy;
334     int i;
335
336     char msg_from_user[128] = { 0 };
337
338     to_copy = min(cnt, sizeof(textbuffer));
339     not_copied = copy_from_user(textbuffer, user, to_copy);
340
341     write_lcd(0, 0x80);         //set cursor to begin
342
343     for(i = 0; i < to_copy && textbuffer[i]; i++){
344         if(isprint(textbuffer[i])){          //checks, whether the passed character is printable
345             write_lcd(1, textbuffer[i]);
346         }
347         if( i == 15){
348             write_lcd(0, 0xc0);             //jump to second row of display
349         }
350     }
351
352     if(copy_from_user(msg_from_user, user, cnt)) {
353         printk("failed copy from user");
354     }
355 return to_copy-not_copied;              //returns how many characters are left -> returns 0 if everything
       is copied properly
356 }
```

3.1.2.4 static int exit_display (void) [static]

frees memory on every initialized gpio pin if module gets unloaded from kernel

called from kernel callbacks

```
279                             {
280     printk("exit display called\n");
281     gpio_free(25);
282     gpio_free(24);
283     gpio_free(23);
284     gpio_free(18);
285     gpio_free(8);
286     gpio_free(7);
287     return 0;
288 }
```

3.1.2.5 static int gpio_request_output (int *nr*) [static]

checks if gpio can be written successfully

Parameters

pin	number of gpio to be requested

Returns

request successful or not as integer

```
196                                     {
197
198     char gpio_name[12];
199     int err = 0;
200
201     snprintf( gpio_name, sizeof(gpio_name), "rpi-gpio-%d", nr);
202     err = gpio_request(nr, gpio_name);
203
204     if(err){
205         printk("gpio request for %s failed with %d\n", gpio_name, err);
206         return err;
207     }
208     err = gpio_direction_output(nr, 0);
209     if(err){
210         printk("gpio direction output failed %d\n", err);
211         gpio_free(nr);
212         return err;
213     }
214     return err;
215 }
```

3.1.2.6 static int init_display (void) [static]

initializes display as soon as the module is loaded into the kernel

Returns

returns 0 if initializing is successful, otherwise <0 checks if every gpio can be requested successfully and write control bits to the lcd. using sleep and delays to be sure writing is working problerly

```
224                                 {
225
226 printk("initialize display\n");
227
228 if(gpio_request_output(7) == -1){
229      return -EIO;
230 }
231 if(gpio_request_output(8) == -1){
232     goto free7;
233 }
234 if(gpio_request_output(18) == -1){
235     goto free8;
236 }
237 if(gpio_request_output(23) == -1){
238     goto free18;
239 }
240 if(gpio_request_output(24) == -1){
241     goto free23;
242 }
243 if(gpio_request_output(25) == -1){
244     goto free24;
245 }
246
247 msleep(15);
248 write_nibble(0, 0x3);
249 msleep(5);
250 write_nibble(0, 0x3);
251 udelay(100);
252 write_nibble(0, 0x3);
253 msleep(5);
254 write_nibble(0, 0x2);
255 msleep(5);
256 write_lcd(0, 0x28);    //Command: 4-Bit Mode, 2 lines
257 msleep(2);
258 write_lcd(0, 0x01);
259 msleep(2);
260
261 write_lcd(0, 0x0c);     //display on, cursor off, blink off
262 write_lcd(0, 0xc0);
263     return 0;
264
265 free24: gpio_free(24);
266 free23: gpio_free(23);
267 free18: gpio_free(18);
268 free8: gpio_free(8);
269 free7: gpio_free(7);
```

```
270     return -EIO;
271 }
```

3.1.2.7 static void __exit mod_exit (void) [static]

lkm exit function

similiar to initialization function, it is static. __exit macro notifies if code is used for a built-in driver (not a lkm) that this function is not required.

```
423                                 {
424     dev_info(hd44780_dev, "mod_exit called\n");
425     exit_display();
426     device_destroy(hd44780_class, dev);
427     class_destroy(hd44780_class);
428     cdev_del(driver_object);
429     unregister_chrdev_region(dev, 1);
430     return;
431 }
```

3.1.2.8 static int __init mod_init (void) [static]

kernel module initialization function

the static keyword restricts the visibility of the function within this C file __init macro means that for a built-in driver (not a kernel module!) is only used at initialization time and that it can be discarded and its memory freed after.

Returns

returns 0 if successful

```
367                                 {
368 int error = 0;
369
370 dev = MKDEV(major, minor);
371
372 if(register_chrdev_region(MKDEV(major, 0),count,"hd44780") < 0){
373     printk("devicenumber(255, 0) in use!\n");
374     return -EIO;
375 }
376 else{
377     error = alloc_chrdev_region(&dev, 0, count, "hd44780");
378     major = MAJOR(dev);
379 }
380
381 driver_object = cdev_alloc();  /* registered object reserved*/
382
383 if(driver_object == NULL){
384     goto free_device_number;
385 }
386
387 driver_object->owner = THIS_MODULE;
388 driver_object->ops = &fops;
389
390 if(cdev_add(driver_object, dev, 1)){
391     goto free_cdev;
392 }
393
394 hd44780_class = class_create(THIS_MODULE, "hd44780");
395
396 if(IS_ERR(hd44780_class)){
397     pr_err("hd44780: no udev support!\n");
398     goto free_cdev;
399 }
400
401 hd44780_dev = device_create(hd44780_class, NULL, dev, NULL, "%s", "hd44780");
402 dev_info(hd44780_dev, "mod_init called\n");
403
404 if(init_display() == 0){
405     return 0;
406 }
407
408 free_cdev:
409     kobject_put(&driver_object->kobj);
```

```
410 free_device_number:
411     unregister_chrdev_region(dev, 1);
412     printk("mod_init failed\n");
413     return -EIO;
414 }
```

3.1.2.9 MODULE_AUTHOR ("Daniel Obermaier <mailto:dan.obermaier@gmail.com> , Victor Nagy< mailto:victor.nagy @hotmail.de >")

< core header for loading LKMs into the kernel

< for prink priority macros < for entry/exit macros to mark up functions e.g. __init __exit header for the linux file system support < header to support the kernel module driver < required for copy_to_user() function

3.1.2.10 MODULE_DESCRIPTION ("driver for LCD Display with HD44780 controller")

3.1.2.11 module_exit (mod_exit)

3.1.2.12 module_init (mod_init)

a module must use the module_init() and module_exit() macros from linux/init.h

which identify the initialization function at insertion time and the cleanup function (as listed above)

3.1.2.13 MODULE_LICENSE ("GPL")

3.1.2.14 module_param (major , int , S_IRUGO| *S_IWUSR*)

module parameters -> allow arguments to be passed to modules

3.1.2.15 module_param (count , int , S_IRUGO| *S_IWUSR*)

3.1.2.16 MODULE_VERSION ("0.1")

3.1.2.17 static void write_lcd (int *regist*, int *value*) [static]

uses write_nibble to shift and reverse logic values

Parameters

control	character
value	to write

```
185                                         {
186     write_nibble(regist, value >> 4); //HIGH-Nibble logic
187     write_nibble(regist, value & 0xf); //LOW-Nibble logic
188 }
```

3.1.2.18 static void write_nibble (int *regist*, int *value*) [static]

writes 4-bit values to gpio

Parameters

control	character
value	to write

```
163                                                 {
164
165     gpio_set_value(7, regist);
166
167     gpio_set_value(25, value & 0x1); //DATABIT 4
168     gpio_set_value(24, value & 0x2); //DATABIT 5
169     gpio_set_value(23, value & 0x4); //DATABIT 6
170     gpio_set_value(18, value & 0x8); //DATABIT 7
171
172     gpio_set_value(8, 1); //enabled to write values
173
174     udelay(40);
175
176     gpio_set_value(8, 0); //disabled to write values
177 }
```

3.1.3 Variable Documentation

3.1.3.1 int count = 1 [static]

3.1.3.2 dev_t dev = 0

variable for lcd device

3.1.3.3 int dev_cnt = 0 [static]

used to prevent multiple access

3.1.3.4 struct cdev* driver_object [static]

3.1.3.5 struct file_operations fops [static]

Initial value:

```
= {
    .owner = THIS_MODULE,
    .open = dev_open,
    .write = dev_write,
    .release = dev_release
}
```

Devices are represented as file structures in the kernel.

file_operation struct from /linux/fs.h lists the various callback functions which can be associated with file operations

3.1.3.6 struct class* hd44780_class [static]

3.1.3.7 struct device* hd44780_dev [static]

3.1.3.8 int major = 0 [static]

major number as an lkm identifier

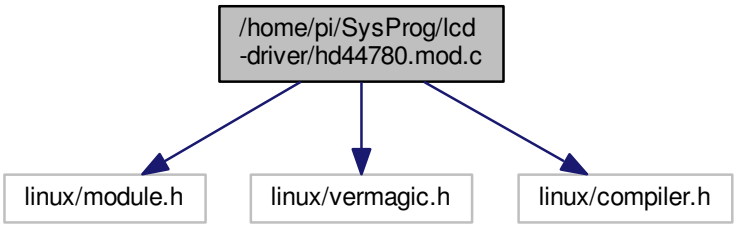
3.1.3.9 int minor = 0 [static]

3.1.3.10 char textbuffer[1024] [static]

3.2 /home/pi/SysProg/lcd-driver/hd44780.mod.c File Reference

```
#include <linux/module.h>
#include <linux/vermagic.h>
#include <linux/compiler.h>
```

Include dependency graph for hd44780.mod.c:



Functions

- MODULE_INFO (vermagic, VERMAGIC_STRING)
- __visible struct module
  __this_module __attribute__ ((section(".gnu.linkonce.this_module")))
- static const struct
  modversion_info ____versions[]
  __used __attribute__ ((section("__versions")))
- static const char
  __module_depends[] __used __attribute__ ((section(".modinfo")))
- MODULE_INFO (srcversion,"9490EAC9968BAC7D1AF5989")

3.2.1 Function Documentation

3.2.1.1 __visible struct module __this_module __attribute__ ((section(".gnu.linkonce.this_module")))

3.2.1.2 static const struct modversion_info ____versions [] __used __attribute__ ((section("__versions"))) [static]

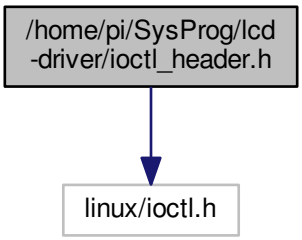
3.2.1.3 static const char __module_depends [] __used __attribute__ ((section(".modinfo"))) [static]

3.2.1.4 MODULE_INFO (vermagic , VERMAGIC_STRING)

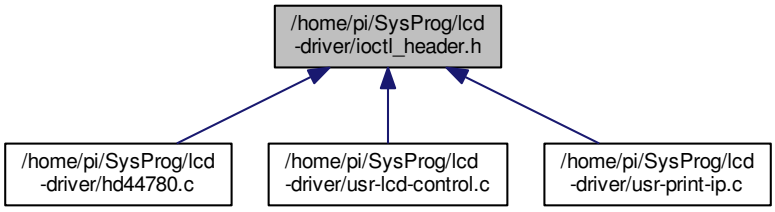
3.2.1.5 MODULE_INFO (srcversion , "9490EAC9968BAC7D1AF5989")

3.3 /home/pi/SysProg/lcd-driver/ioctl_header.h File Reference

`#include <linux/ioctl.h>`
Include dependency graph for ioctl_header.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define MAGIC_NUM 'k' /∗unique arbitrary number for driver∗/
- #define IOCTL _IO(MAGIC_NUM, 0); /∗int argument∗/
- #define IOCTL_WRITE _IOW(MAGIC_NUM, 2, int) /∗returns sizeof(int) bytes to the user ∗/
- #define IOCTL_READ _IOR(MAGIC_NUM, 3, int) /∗...∗/

3.3.1 Macro Definition Documentation

3.3.1.1 #define IOCTL _IO(MAGIC_NUM, 0); /∗int argument∗/

3.3.1.2 #define IOCTL_READ _IOR(MAGIC_NUM, 3, int) /∗...∗/

3.3.1.3 #define IOCTL_WRITE _IOW(MAGIC_NUM, 2, int) /∗returns sizeof(int) bytes to the user ∗/

3.3.1.4 #define MAGIC_NUM 'k' /∗unique arbitrary number for driver∗/

IOCTL creates a kernelmessage IOCTL_WRITE writes a variable to the character driver and also prints a kernel message IOCTL_READ reads a variable from the driver

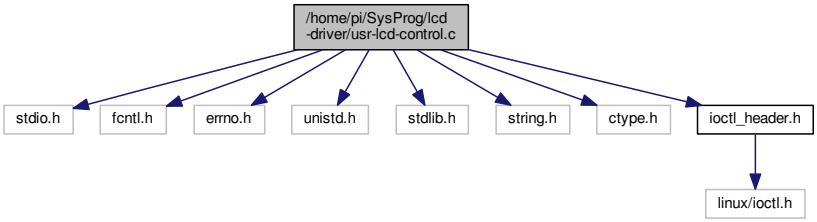
3.4 /home/pi/SysProg/lcd-driver/README.md File Reference

3.5 /home/pi/SysProg/lcd-driver/usr-lcd-control.c File Reference

a linux user space program that communicates with the hd44780 linux kernel module (LKM). to work with the device the /dev/hd44780 must be called

```
#include <stdio.h>
#include <fcntl.h>
#include <errno.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "ioctl_header.h"
```

Include dependency graph for usr-lcd-control.c:



Functions

- int main (int argc, char **argv)

3.5.1 Detailed Description

a linux user space program that communicates with the hd44780 linux kernel module (LKM). to work with the device the /dev/hd44780 must be called

Author

Daniel Obermaier

Date

02.06.2016

Version

0.1

3.5.2 Function Documentation

3.5.2.1 int main (int *argc,* char ** *argv*)

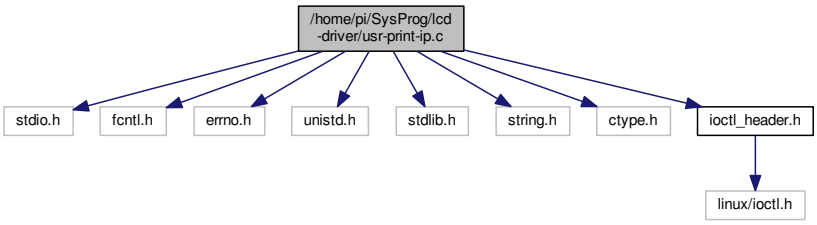
```
22                                  {
```

```
23
24 static const char* devname = "/dev/hd44780";
25
26 int ret = 0;
27 char buff[128] = "";
28 int dev = 0;
29
30 if(argc != 2){ /*argc should be 2 for correct execution*/
31     printf("usage:  <filename> <string>\n");
32 }
33 else{
34     int dev = open(devname, O_WRONLY);  //open device file -> WRITE ONLY
35
36     ret = write(dev, "                                ", 32);   //clear display
37     ret = write(dev, "input from website", 32);     //just a test
38     if(dev == -1){
39         perror("can't open device file\n");
40         return -EIO;
41     }
42
43     ret = write(dev, argv[1], 128);     //write string to device
44
45     if(ret < 0){
46         perror("cant write to devicefile\n");
47         return EIO;
48     }
49 }
50
51 close(dev);     //close device afterwards
52
53 return 0;
54 }
```

3.6 /home/pi/SysProg/lcd-driver/usr-print-ip.c File Reference

```
#include <stdio.h>
#include <fcntl.h>
#include <errno.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "ioctl_header.h"
```

Include dependency graph for usr-print-ip.c:



Functions

- int main (void)

3.6.1 Function Documentation

3.6.1.1 int main (void)

```
22              {
23
24 /*static const char* devname = "/dev/hd44780";
25
26 int ret = 0;
27 char buff[128] = "";
28 int dev = 0;*/
29
30 //  int dev = open(devname, O_WRONLY);  //open device file -> WRITE ONLY
31
32 //  ret = write(dev, "                                ", 32);   //clear display
33 /*  if(dev == -1){
34         perror("can't open device file\n");
35         return -EIO;
36     }
37
38     ret = write(dev, argv[1], 128);     //write string to device
39
40     if(ret < 0){
41         perror("cant write to devicefile\n");
42         return EIO;
43     }*/
44
45 //close(dev);       //close device afterwards
46
47 return 0;
48 }
```