HD44780 LCD Linux Character Device Driver

Embedded Systems

Lecturer: Assoc. Prof. Hui-Kai Su

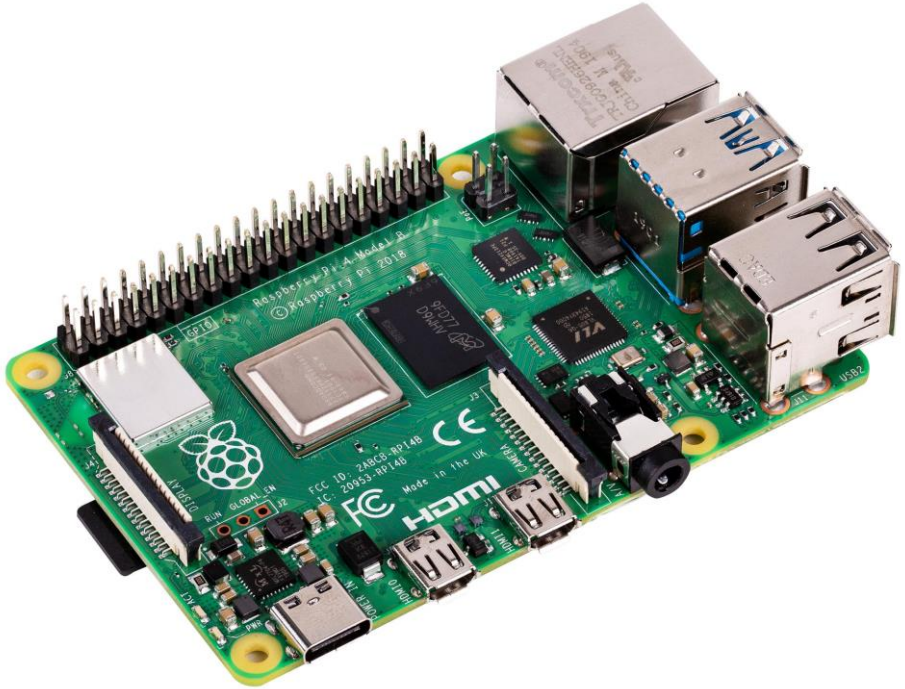
Group 18

范阮皇海 - 11065154

Outline

1. Introduction
2. Specification
3. Background
4. Implementation
5. Experiments
6. An Example Application

1. Introduction



2. Specification

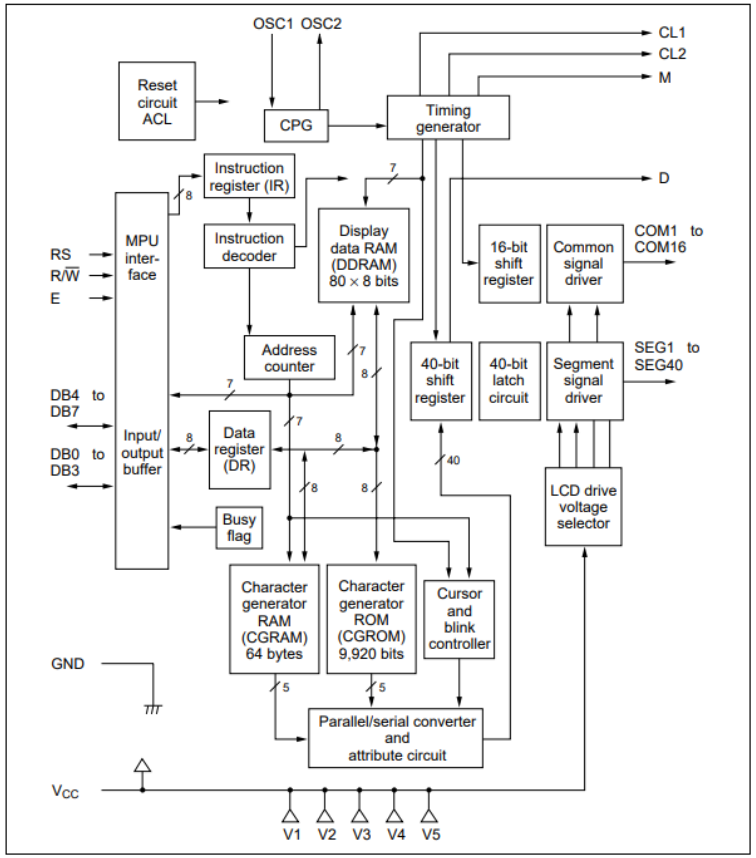
- I2C interface
- LCD Linux driver module
- Handle I/O stream
  - Send data to LCD in user program
  - echo "message" > /dev/lcdi2c

3. Background

Inside HD44780 Display Module

Display position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
DDRAM address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

HD44780U Block Diagram



Pin Functions

Signal	No. of Lines	I/O	Device Interfaced with	Function
RS	1	I	MPU	Selects registers. 0: Instruction register (for write) Busy flag: address counter (for read) 1: Data register (for write and read)
R/$\overline{W}$	1	I	MPU	Selects read or write. 0: Write 1: Read
E	1	I	MPU	Starts data read/write.
DB4 to DB7	4	I/O	MPU	Four high order bidirectional tristate data bus pins. Used for data transfer and receive between the MPU and the HD44780U. DB7 can be used as a busy flag.
DB0 to DB3	4	I/O	MPU	Four low order bidirectional tristate data bus pins. Used for data transfer and receive between the MPU and the HD44780U. These pins are not used during 4-bit operation.
CL1	1	O	Extension driver	Clock to latch serial data D sent to the extension driver
CL2	1	O	Extension driver	Clock to shift serial data D
M	1	O	Extension driver	Switch signal for converting the liquid crystal drive waveform to AC
D	1	O	Extension driver	Character pattern data corresponding to each segment signal
COM1 to COM16	16	O	LCD	Common signals that are not used are changed to non-selection waveforms. COM9 to COM16 are non-selection waveforms at 1/8 duty factor and COM12 to COM16 are non-selection waveforms at 1/11 duty factor.
SEG1 to SEG40	40	O	LCD	Segment signals
V1 to V5	5	—	Power supply	Power supply for LCD drive V_{CC} –V5 = 11 V (max)
V_{CC}, GND	2	—	Power supply	V_{CC}: 2.7V to 5.5V, GND: 0V
OSC1, OSC2	2	—	Oscillation resistor clock	When crystal oscillation is performed, a resistor must be connected externally. When the pin input is an external clock, it must be input to OSC1.

Table 4 Correspondence between Character Codes and Character Patterns (ROM Code: A00)

Lower 4 Bits \ Upper 4 Bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000	CG RAM (1)			0	@	P	`	p				ー	タ	ミ	α	p
xxxx0001	(2)		!	1	A	Q	a	q			。	ア	チ	ム	ä	q
xxxx0010	(3)		"	2	B	R	b	r			「	イ	ツ	メ	ß	θ
xxxx0011	(4)		#	3	C	S	c	s			」	ウ	テ	モ	ε	∞
xxxx0100	(5)		$	4	D	T	d	t			、	エ	ト	ヤ	μ	Ω
xxxx0101	(6)		%	5	E	U	e	u			・	オ	ナ	ユ	σ	Ü
xxxx0110	(7)		&	6	F	V	f	v			ヲ	カ	ニ	ヨ	ρ	Σ
xxxx0111	(8)		'	7	G	W	g	w			ァ	キ	ヌ	ラ	g	π
xxxx1000	(1)		(	8	H	X	h	x			ィ	ク	ネ	リ	√	X
xxxx1001	(2)		)	9	I	Y	i	y			ゥ	ケ	ノ	ル	⁻¹	y
xxxx1010	(3)		*	:	J	Z	j	z			ェ	コ	ハ	レ	j	千
xxxx1011	(4)		+	;	K	[	k	{			ォ	サ	ヒ	ロ	×	万
xxxx1100	(5)		,	<	L	¥	l	|			ャ	シ	フ	ワ	¢	円
xxxx1101	(6)		−	=	M	]	m	}			ュ	ス	ヘ	ン	£	÷
xxxx1110	(7)		.	>	N	^	n	→			ョ	セ	ホ	゛	ñ	
xxxx1111	(8)		/	?	O	_	o	←			ッ	ソ	マ	゜	ö	■

3. Background

Interfacing the HD44780

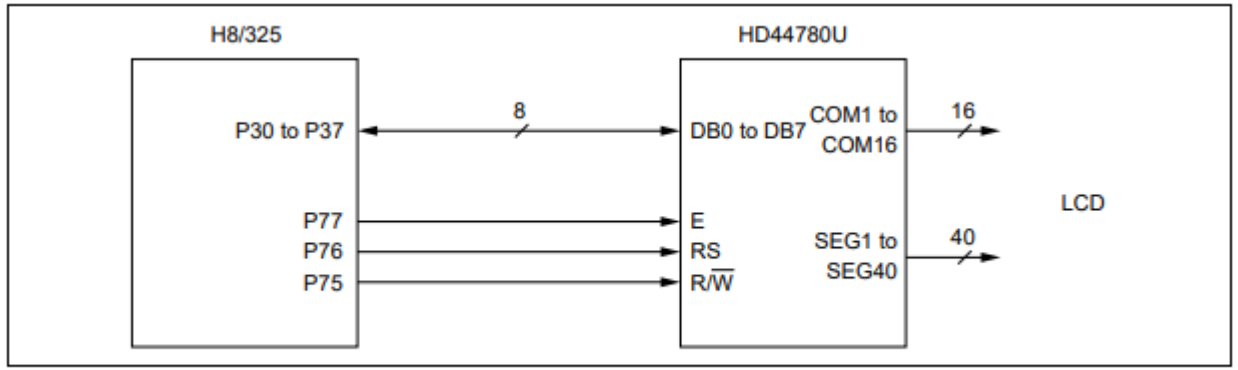


Figure 16 H8/325 Interface (Single-Chip Mode)

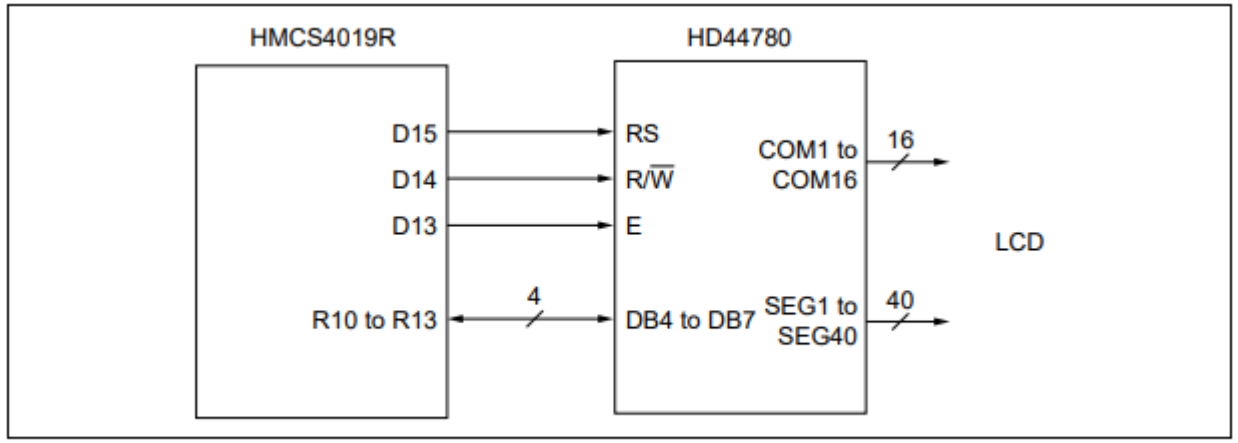


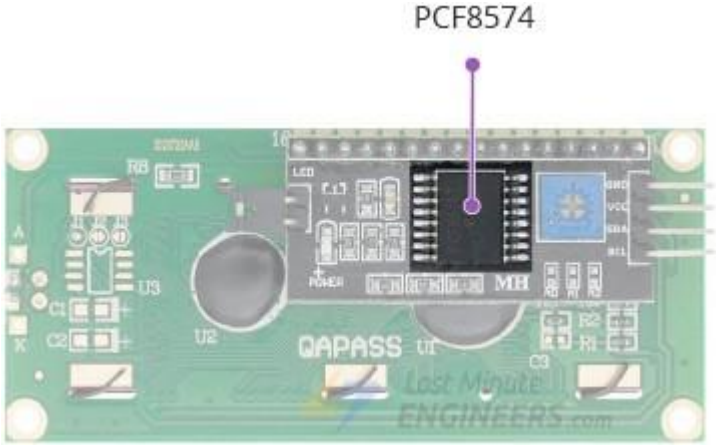
Figure 18 Example of Interface to HMCS4019R

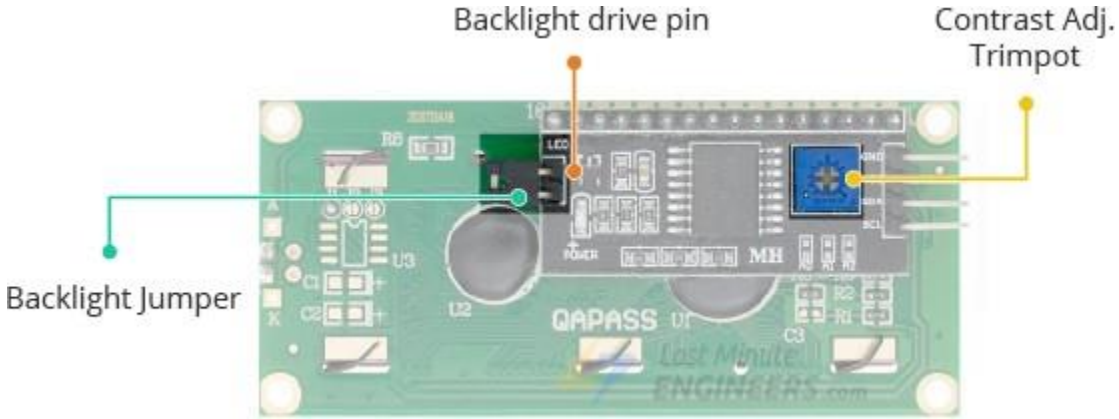
Table 6 Instructions

Instruction	Code RS	R/$\overline{W}$	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description	Execution Time (max) (when f_{cp} or f_{osc} is 270 kHz)
Clear display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DDRAM address 0 in address counter.	
Return home	0	0	0	0	0	0	0	0	1	—	Sets DDRAM address 0 in address counter. Also returns display from being shifted to original position. DDRAM contents remain unchanged.	1.52 ms
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies display shift. These operations are performed during data write and read.	37 µs
Display on/off control	0	0	0	0	0	0	1	D	C	B	Sets entire display (D) on/off, cursor on/off (C), and blinking of cursor position character (B).	37 µs
Cursor or display shift	0	0	0	0	0	1	S/C	R/L	—	—	Moves cursor and shifts display without changing DDRAM contents.	37 µs
Function set	0	0	0	0	1	DL	N	F	—	—	Sets interface data length (DL), number of display lines (N), and character font (F).	37 µs
Set CGRAM address	0	0	0	1	ACG	ACG	ACG	ACG	ACG	ACG	Sets CGRAM address. CGRAM data is sent and received after this setting.	37 µs
Set DDRAM address	0	0	1	ADD	ADD	ADD	ADD	ADD	ADD	ADD	Sets DDRAM address. DDRAM data is sent and received after this setting.	37 µs
Read busy flag & address	0	1	BF	AC	AC	AC	AC	AC	AC	AC	Reads busy flag (BF) indicating internal operation is being performed and reads address counter contents.	0 µs

3. Background

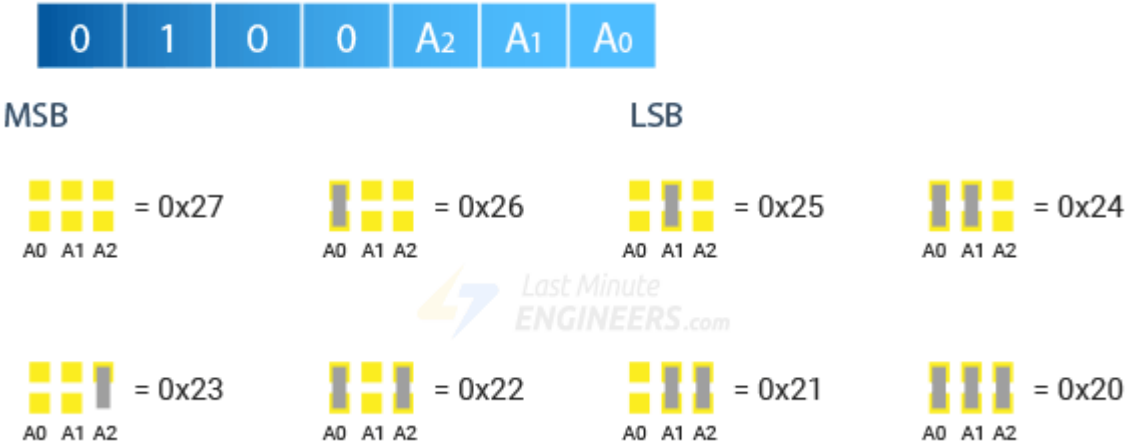
I2C LCD Adapter



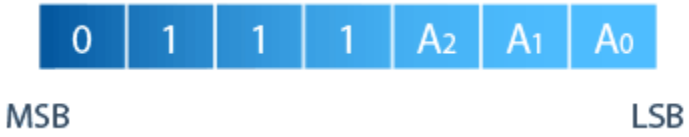


I2C Address of LCD

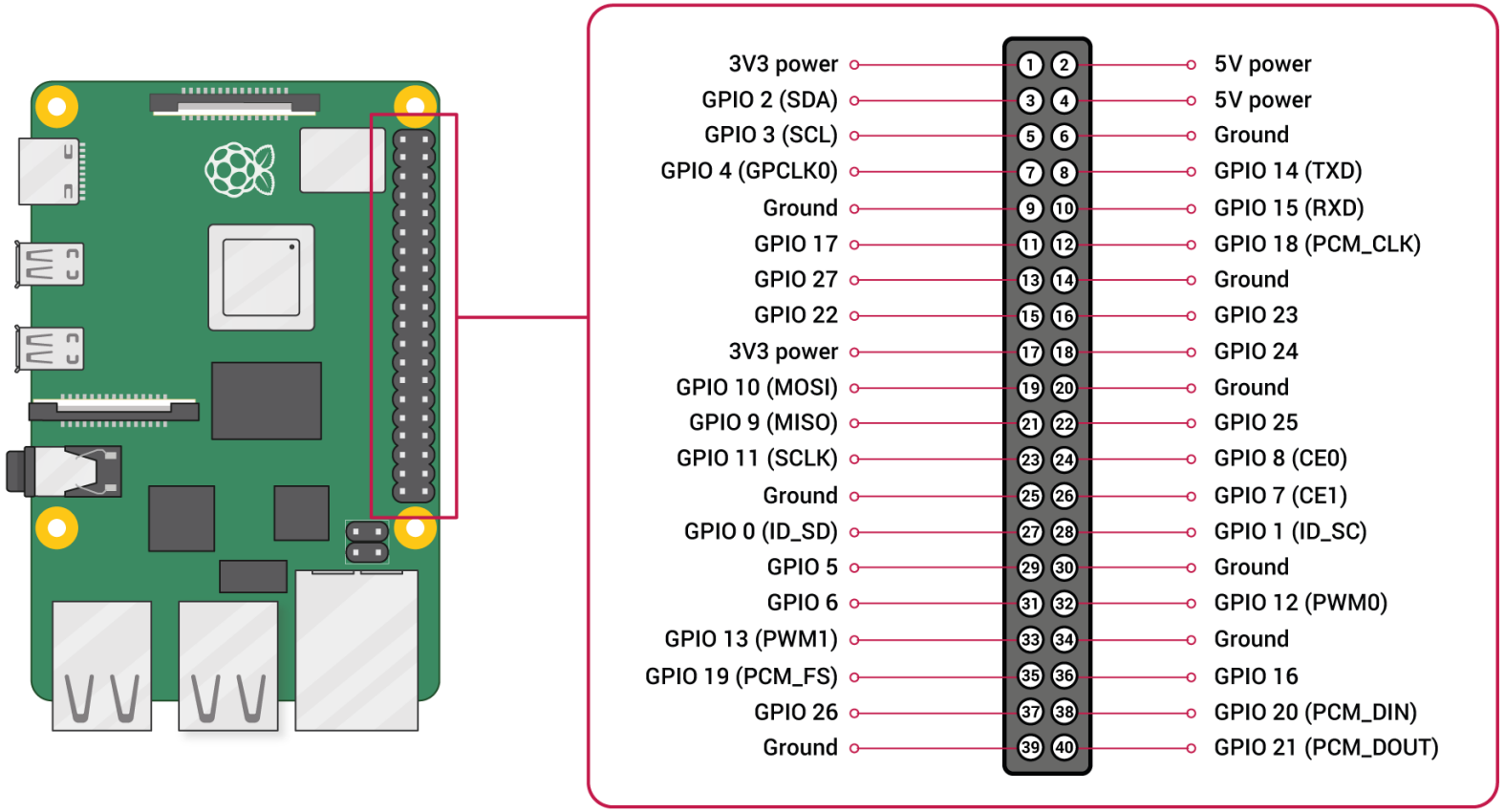
Texas Instruments' PCF8574

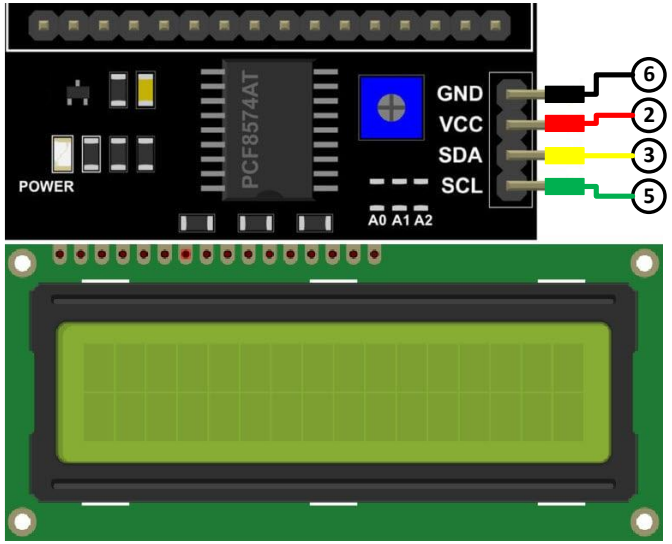


NXP's PCF8574



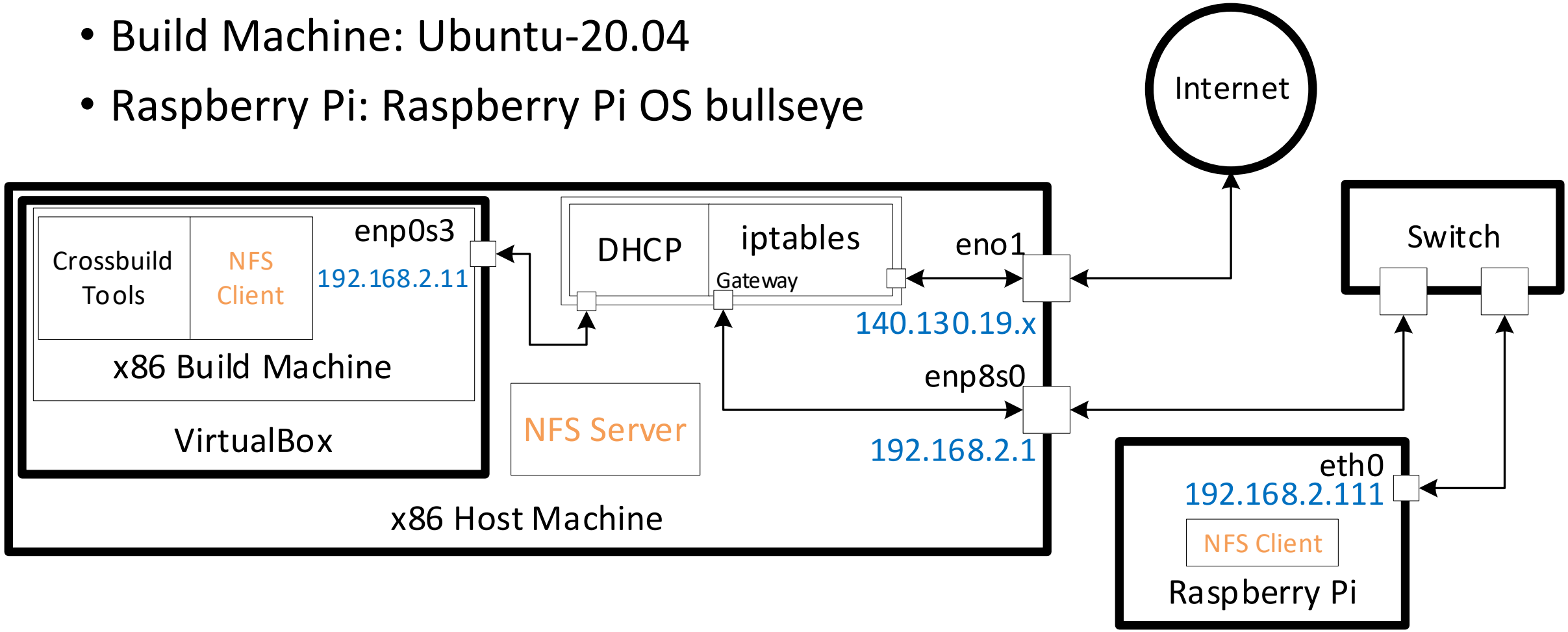
4. Implementation – Hardware Connection





4. Implementation – Dev Environment

- Host Machine: Ubuntu (any version)
- Build Machine: Ubuntu-20.04
- Raspberry Pi: Raspberry Pi OS bullseye



4. Implementation – Bringup Test

Enable I2C interface

$ sudo raspi-config

$ pip install smbus2 rpi-lcd

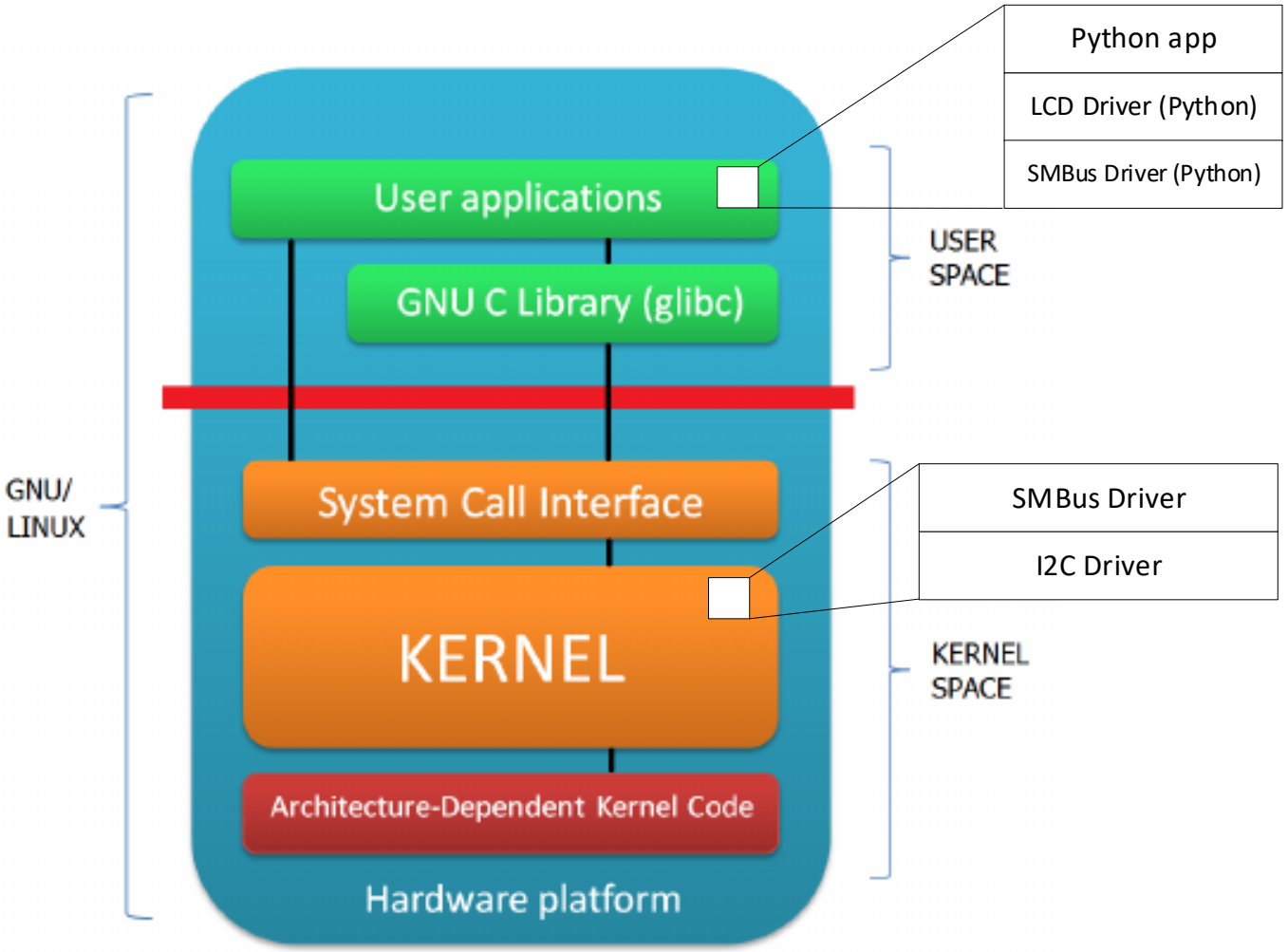
$ python

```
from rpi_lcd import LCD
lcd = LCD()
lcd.text('Hello World!', 1)
```

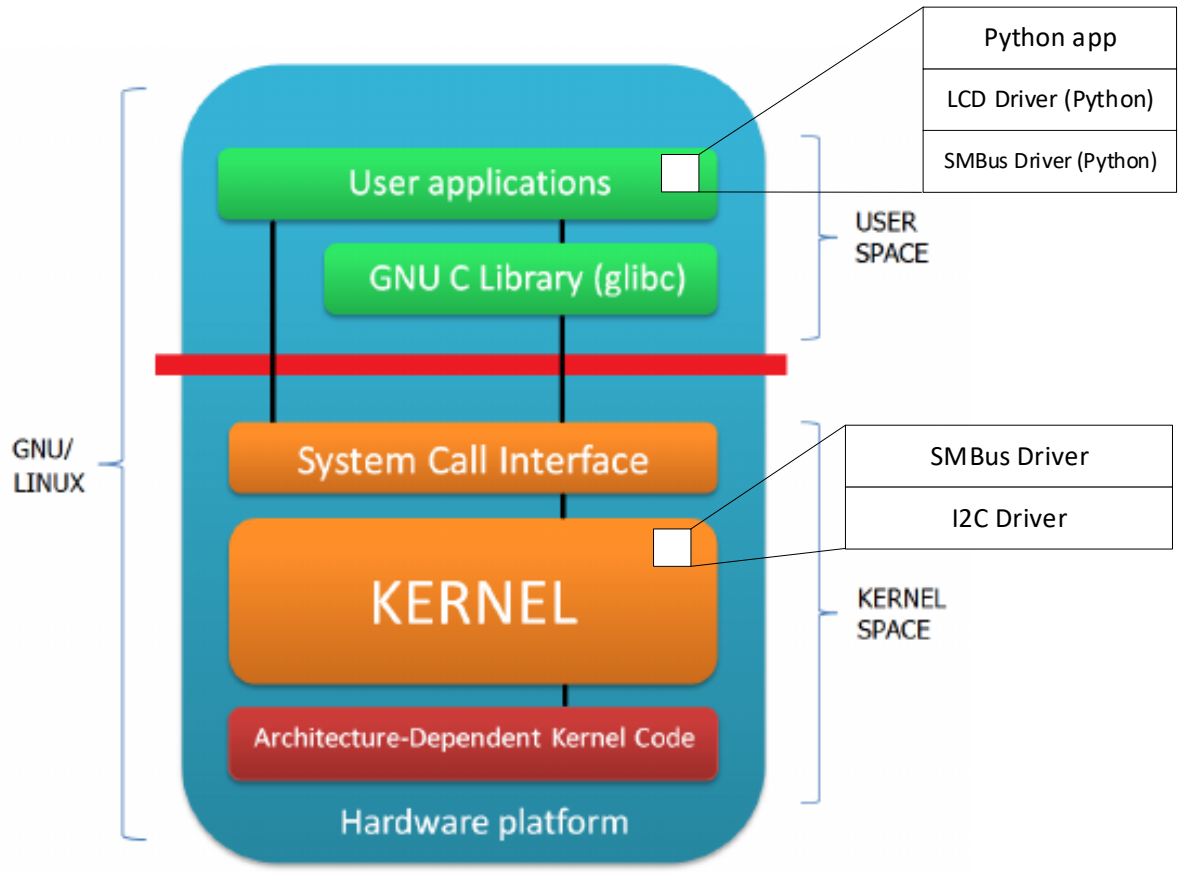
Troubleshoot: $ sudo rpi-update

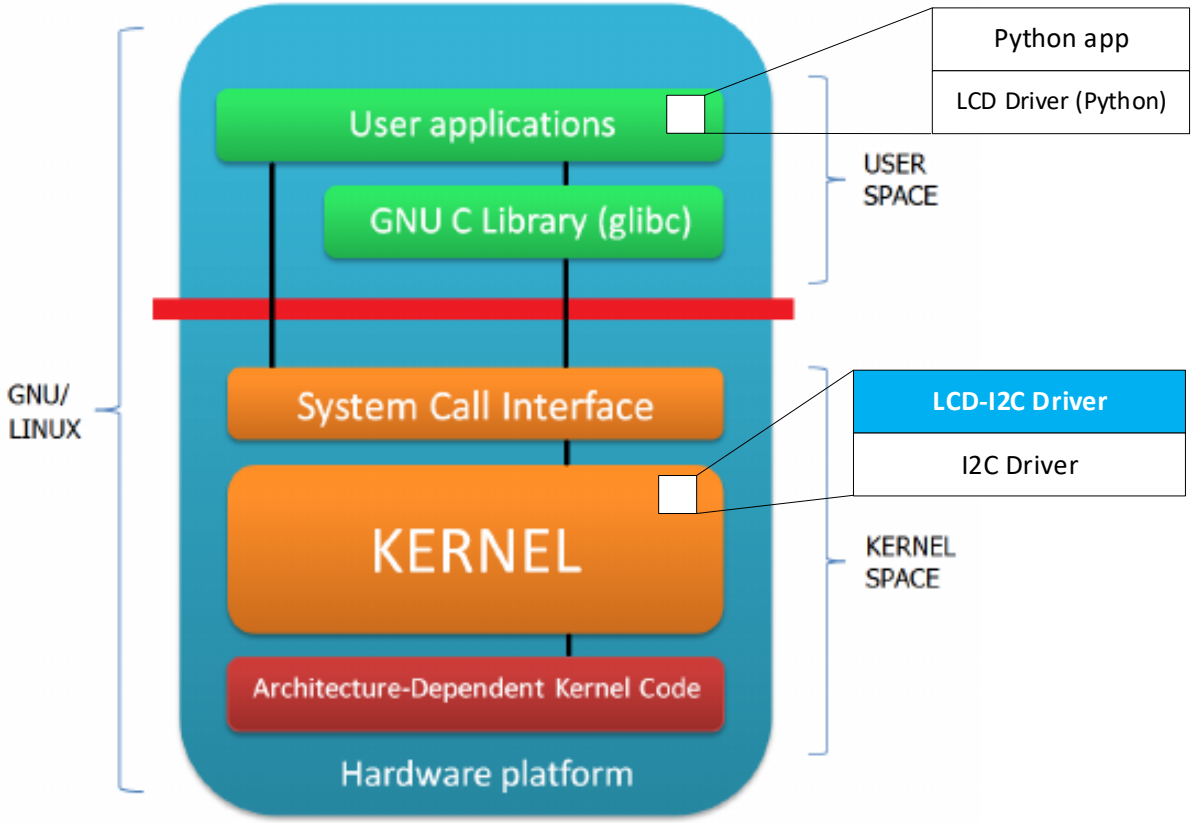
4. Implementation – Bringup Test (cont.)

Understand the bringup test



4. Implementation – Character Driver Module





4. Implementation – Character Driver Module

Raspberry Pi Linux kernel source code

+ Header files

+ Compiled modules

haipnh@raspberrypi: ~

```
haipnh@raspberrypi:~ $ uname -r
5.15.84-v8+
```

Search or jump to... /

Pull r

raspberrypi / linux Public

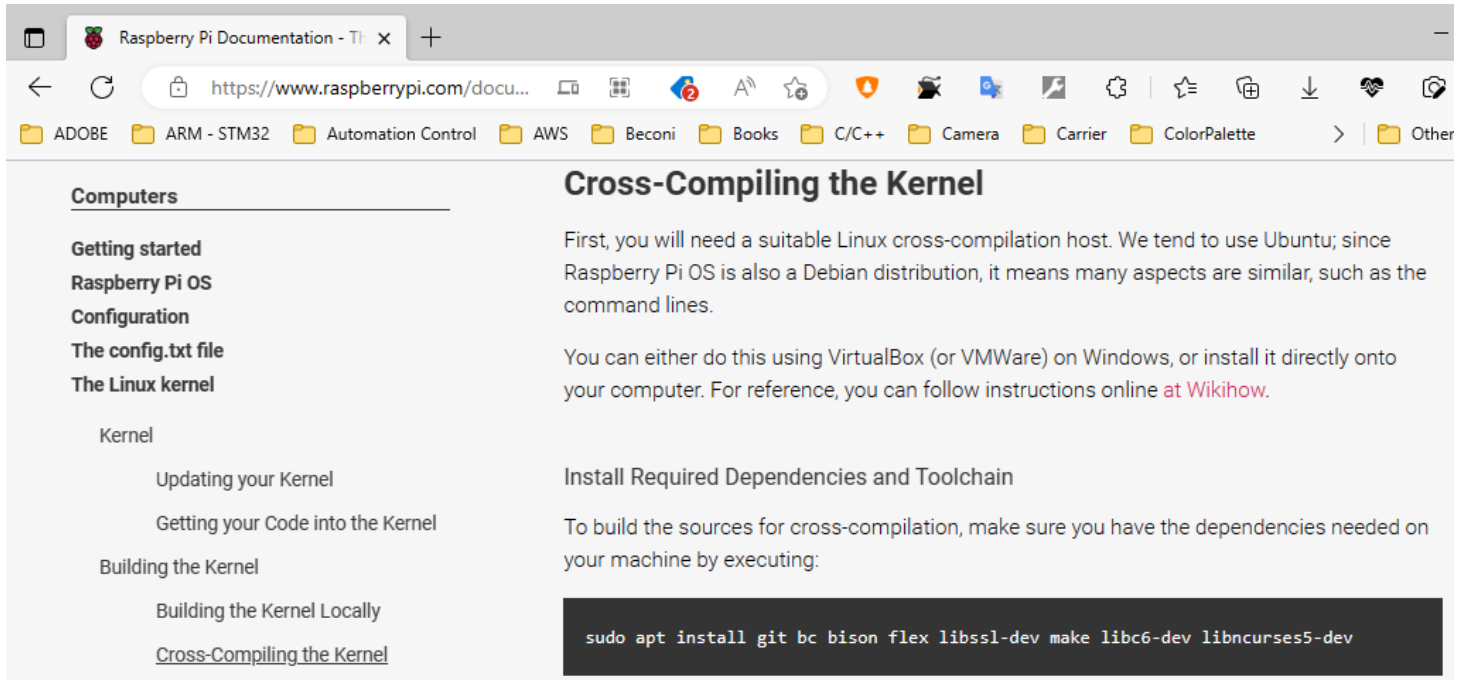
Code Issues 674 Pull requests 56

rpi-5.15.y 78 branches 90 tags

Makefile Linux 5.15.84

4. Implementation – Character Driver Module

Cross-compile the source code

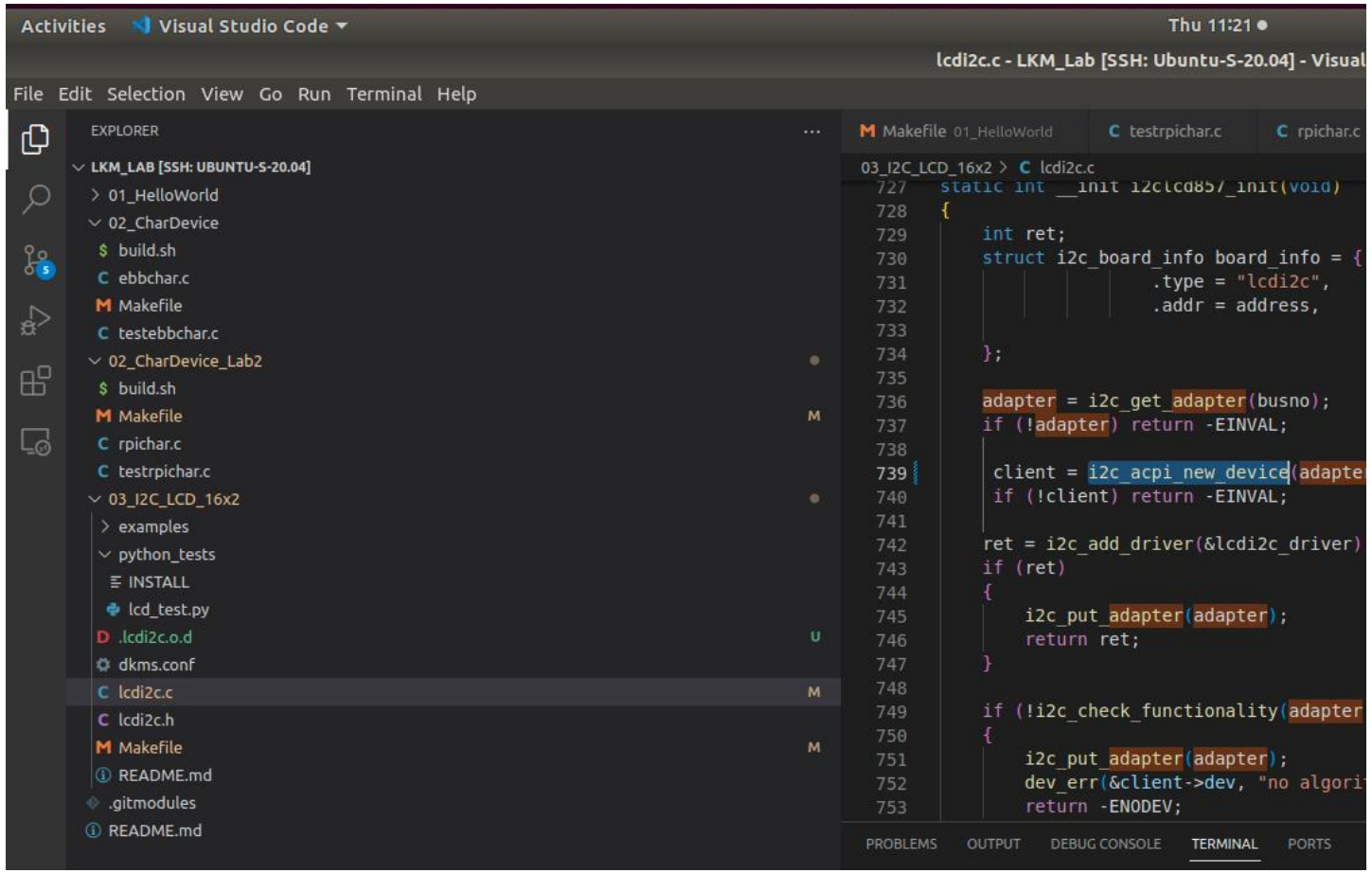


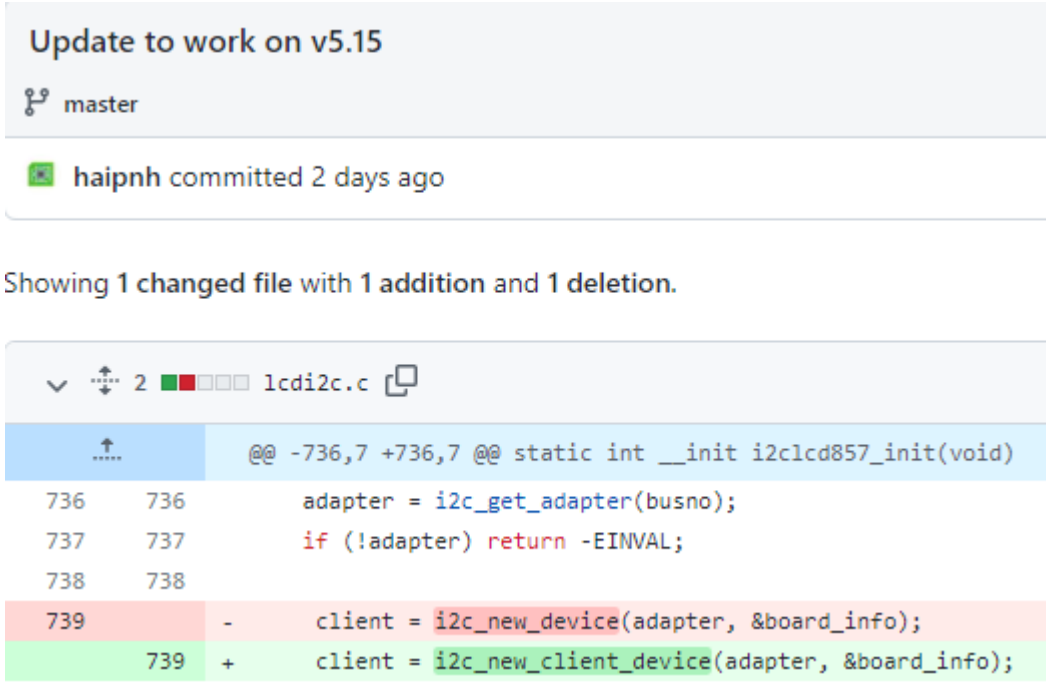

```
haipnh@ubuntu-s-2004:~$ cd /mnt/nfs_server/linux/5.15.84
haipnh@ubuntu-s-2004:/mnt/nfs_server/linux/5.15.84$ make ARCH=arm64
 CROSS_COMPILE=aarch64-linux-gnu- modules
  CALL    scripts/checksyscalls.sh
  CALL    scripts/atomic/check-atomics.sh
  CHK     include/generated/compile.h
```

4. Implementation – Character Driver Module

https://github.com/lucidm/lcdi2c

➔ **Folked:** https://github.com/haipnh/lcdi2c





4. Implementation – Character Driver Module

Probing and Removing

```c
static int lcdi2c_probe(struct i2c_client *client, const struct i2c_device_id *id)
{

    data = (LcdData_t *) devm_kzalloc(&client->dev, sizeof(LcdData_t),
                                      GFP_KERNEL);
    if (!data)
        return -ENOMEM;

    i2c_set_clientdata(client, data);
    sema_init(&data->sem, 1);

    data->row = 0;
    data->column = 0;
    data->handle = client;
    data->backlight = 1;
    data->cursor = cursor;
    data->blink = blink;
    data->deviceopencnt = 0;
    data->major = major;

    lcdinit(data, topo);
    lcdprint(data, "HD44780\nDriver");

    dev_info(&client->dev, "%ux%u LCD using bus 0x%X, at address 0x%X",
             data->organization.columns,
             data->organization.rows, busno, address);

    return 0;
```

```c
static int lcdi2c_remove(struct i2c_client *client)
{
    LcdData_t *data = i2c_get_clientdata(client);

        dev_info(&client->dev, "going to be removed");
        if (data)
            lcdfinalize(data);

    return 0;
}

```

```c
static const struct i2c_device_id lcdi2c_id[] = {
    { "lcdi2c", 0 },
        { },
};
MODULE_DEVICE_TABLE(i2c, lcdi2c_id);

static struct i2c_driver lcdi2c_driver = {
    .driver = {
                .owner  = THIS_MODULE,
        .name   = "lcdi2c",
    },
        .probe          = lcdi2c_probe,
        .remove         = lcdi2c_remove,
    .id_table   = lcdi2c_id,
};
```

4. Implementation – Character Driver Module

Supported /dev/lcdi2c device interfaces

```c
#include "lcdi2c.h"

#define CRIT_BEG(d, error) if(down_interruptible(&d->sem)) return -error
#define CRIT_END(d) up(&d->sem)

static uint busno = 1;      //I2C Bus number
static uint address = DEFAULT_CHIP_ADDRESS; //Device address
static uint topo = LCD_DEFAULT_ORGANIZATION;
static uint cursor = 1;
static uint blink = 1;
static IOCTLDescription_t ioctls[] = {
  { .ioctlcode = LCD_IOCTL_GETCHAR, .name = "GETCHAR", },
  { .ioctlcode = LCD_IOCTL_SETCHAR, .name = "SETCHAR", },
  { .ioctlcode = LCD_IOCTL_GETPOSITION, .name = "GETPOSITION" },
  { .ioctlcode = LCD_IOCTL_SETPOSITION, .name = "SETPOSITION" },
  { .ioctlcode = LCD_IOCTL_RESET, .name = "RESET" },
  { .ioctlcode = LCD_IOCTL_HOME, .name = "HOME" },
  { .ioctlcode = LCD_IOCTL_GETBACKLIGHT, .name = "GETBACKLIGHT" },
  { .ioctlcode = LCD_IOCTL_SETBACKLIGHT, .name = "SETBACKLIGHT" },
  { .ioctlcode = LCD_IOCTL_GETCURSOR, .name = "GETCURSOR" },
  { .ioctlcode = LCD_IOCTL_SETCURSOR, .name = "SETCURSOR" },
  { .ioctlcode = LCD_IOCTL_GETBLINK,  .name = "GETBLINK" },
  { .ioctlcode = LCD_IOCTL_SETBLINK,  .name = "SETBLINK" },
  { .ioctlcode = LCD_IOCTL_SCROLLHZ,  .name = "SCROLLHZ" },
  { .ioctlcode = LCD_IOCTL_GETCUSTOMCHAR, .name = "GETCUSTOMCHAR" },
  { .ioctlcode = LCD_IOCTL_SETCUSTOMCHAR, .name = "SETCUSTOMCHAR" },
  { .ioctlcode = LCD_IOCTL_CLEAR, .name = "CLEAR" },
};
```

```c
static long lcdi2c_ioctl(struct file *file,
            unsigned int ioctl_num,
            unsigned long arg)
{

  char *buffer = (char*)arg, ccb[10];
  u8 memaddr, i, ch;
  long status = SUCCESS;

  CRIT_BEG(data, EAGAIN);

  switch (ioctl_num)
  {
    case LCD_IOCTL_SETCHAR:
      get_user(ch, buffer);
      memaddr = (1 + data->column + (data->row * data->organization.columns)) % LCD_BUFFER_SIZE;
      lcdwrite(data, ch);
      data->column = (memaddr % data->organization.columns);
      data->row = (memaddr / data->organization.columns);
      lcdsetcursor(data, data->column, data->row);
      break;
    case LCD_IOCTL_GETCHAR:
      memaddr = (data->column + (data->row * data->organization.columns)) % LCD_BUFFER_SIZE;
      ch = data->buffer[memaddr];
      put_user(ch, buffer);
      break;
    case LCD_IOCTL_GETPOSITION:
      printk(KERN_INFO "GETPOSITION called\n");
      put_user(data->column, buffer);
      put_user(data->row, buffer+1);
      break;
    case LCD_IOCTL_SETPOSITION:
      get_user(data->column, buffer);
      get_user(data->row, buffer+1);
      lcdsetcursor(data, data->column, data->row);
      break;
    case LCD_IOCTL_RESET:
      get_user(ch, buffer);
      if (ch == '1')
```

4. Implementation – Character Driver Module

Supported /sys device interfaces

```c
static ssize_t lcdi2c_clear(struct device* dev, struct device_attribute* attr,
                const char* buf, size_t count)
{
    CRIT_BEG(data, ERESTARTSYS);

    if (count > 0 && buf[0] == '1')
        lcdclear(data);

    CRIT_END(data);
    return count;
}
```

```c
DEVICE_ATTR(meta, S_IRUSR | S_IRGRP, lcdi2c_meta_show, NULL);
DEVICE_ATTR(cursor, S_IWUSR | S_IWGRP | S_IRUSR | S_IRGRP | S_IROTH,
        lcdi2c_cursor_show, lcdi2c_cursor);
DEVICE_ATTR(blink, S_IWUSR | S_IWGRP | S_IRUSR | S_IRGRP | S_IROTH,
        lcdi2c_blink_show, lcdi2c_blink);
DEVICE_ATTR(home, S_IWUSR | S_IWGRP, NULL, lcdi2c_home);
DEVICE_ATTR(clear, S_IWUSR | S_IWGRP, NULL, lcdi2c_clear);
```

4. Implementation – Character Driver Module

__init() and __exit()

```c
static int __init i2clcd857_init(void)
{
    int ret;
    struct i2c_board_info board_info = {
                .type = "lcdi2c",
                .addr = address,

    };

    adapter = i2c_get_adapter(busno);
    if (!adapter) return -EINVAL;

     client = i2c_new_client_device(adapter, &board_info);
     if (!client) return -EINVAL;

    ret = i2c_add_driver(&lcdi2c_driver);
    if (ret)
    {
        i2c_put_adapter(adapter);
        return ret;
    }

    if (!i2c_check_functionality(adapter, I2C_FUNC_I2C))
    {
        i2c_put_adapter(adapter);
        dev_err(&client->dev, "no algorithms associated to i2c bus\n");
        return -ENODEV;
    }
```

```c
static void __exit i2clcd857_exit(void)
{

    unregister_chrdev(major, DEVICE_NAME);

    sysfs_remove_group(&lcdi2c_device->kobj, &i2clcd_device_attr_group);
    device_destroy(lcdi2c_class, MKDEV(major, 0));
    class_unregister(lcdi2c_class);
    class_destroy(lcdi2c_class);
    unregister_chrdev(major, DEVICE_NAME);

     if (client)
         i2c_unregister_device(client);

    i2c_del_driver(&lcdi2c_driver);
}
```

4. Implementation – Character Driver Module

Compile the driver module

Makefile

```
TOOLS := /usr/bin
PREFIX := aarch64-linux-gnu-
KDIR := /mnt/nfs_server/linux/5.15.84
PWD := $(shell pwd)

obj-m :=  lcdi2c.o

all:
        $(MAKE) -C $(KDIR) \
                M=$(PWD) \
                ARCH=arm64 CROSS_COMPILE=$(TOOLS)/$(PREFIX) \
                modules

clean:
        $(MAKE) -C $(KDIR) M=$(PWD) clean
```

```
haipnh@ubuntu-s-2004:~$ cd /mnt/nfs_server/LKM_Lab/03_I2C_LCD_16x2
haipnh@ubuntu-s-2004:/mnt/nfs_server/LKM_Lab/03_I2C_LCD_16x2$ ls -l
total 72
drwxrwxr-x 2 haipnh haipnh  4096 Dec 23 12:09 bringup_test
-rw-rw-r-- 1 haipnh haipnh   144 Dec 21 09:55 dkms.conf
drwxrwxr-x 5 haipnh haipnh  4096 Dec 23 12:09 examples
-rw-rw-r-- 1 haipnh haipnh 33174 Dec 23 12:09 lcdi2c.c
-rw-rw-r-- 1 haipnh haipnh  7344 Dec 21 09:55 lcdi2c.h
-rw-rw-r-- 1 haipnh haipnh   268 Dec 23 12:39 Makefile
-rw-rw-r-- 1 haipnh haipnh 11221 Dec 21 09:55 README.md
haipnh@ubuntu-s-2004:/mnt/nfs_server/LKM_Lab/03_I2C_LCD_16x2$ make all
make -C /mnt/nfs_server/linux/5.15.84 \
        M=/mnt/nfs_server/LKM_Lab/03_I2C_LCD_16x2 \
        ARCH=arm64 CROSS_COMPILE=/usr/bin/aarch64-linux-gnu- \
        modules
make[1]: Entering directory '/mnt/nfs_server/linux/5.15.84'
  CC [M]  /mnt/nfs_server/LKM_Lab/03_I2C_LCD_16x2/lcdi2c.o
  MODPOST /mnt/nfs_server/LKM_Lab/03_I2C_LCD_16x2/Module.symvers
  CC [M]  /mnt/nfs_server/LKM_Lab/03_I2C_LCD_16x2/lcdi2c.mod.o
  LD [M]  /mnt/nfs_server/LKM_Lab/03_I2C_LCD_16x2/lcdi2c.ko
make[1]: Leaving directory '/mnt/nfs_server/linux/5.15.84'
haipnh@ubuntu-s-2004:/mnt/nfs_server/LKM_Lab/03_I2C_LCD_16x2$ ls
bringup_test  examples  lcdi2c.h   lcdi2c.mod    lcdi2c.mod.o  Makefile       Module.symvers
dkms.conf     lcdi2c.c  lcdi2c.ko  lcdi2c.mod.c  lcdi2c.o      modules.order  README.md
```

4. Implementation – Character Driver Module

haipnh@raspberrypi: /mnt/nfs_server/LKM_Lab/03_I2C_LCD_16x2

```
haipnh@raspberrypi:/mnt/nfs_server/LKM_Lab/03_I2C_LCD_16x2 $ modinfo lcdi2c.ko
filename:       /mnt/nfs_server/LKM_Lab/03_I2C_LCD_16x2/lcdi2c.ko
version:        0.1.0
description:    Driver for HD44780 LCD with PCF8574 I2C extension.
author:         Jarek Zok <jarekzok@gmail.com>
license:        GPL
srcversion:     B0919DE439C0B38BE80C278
alias:          i2c:lcdi2c
depends:
name:           lcdi2c
vermagic:       5.15.84-v8 SMP preempt mod_unload modversions aarch64
parm:           busno: I2C Bus number, default 1 (uint)
parm:           address: LCD I2C Address, default 0x27 (uint)
parm:           pinout: I2C module pinout configuration, eight numbers
                representing following LCD modulepins in order: RS,RW,E,D4,D5,D6,D7,
                default 0,1,2,3,4,5,6,7 (array of uint)
parm:           cursor: Show cursor at start 1 - Yes, 0 - No, default 1 (uint)
parm:           blink: Blink cursor 1 - Yes, 0 - No, defualt 1 (uint)
parm:           major: Device major number, default 0 (int)
parm:           topo: Display organization, following values are currently supported:
                0 - 40x2
                1 - 20x4
                2 - 20x2
                3 - 16x4
                4 - 16x2
                5 - 16x1 Type 1
                6 - 16x1 Type 2
                7 - 8x2
                Default set to 16x2 (uint)
```

```
haipnh@raspberrypi:/mnt/nfs_server/LKM_Lab/03_I2C_LCD_16x2 $ sudo insmod lcdi2c.ko
```

5. Experiments

System calls

```
haipnh@raspberrypi:/mnt/nfs_server/LKM_Lab/03_I2C_LCD_16x2 $ sudo -s
root@raspberrypi:/mnt/nfs_server/LKM_Lab/03_I2C_LCD_16x2# ls /sys/class/alphalcd/lcdi2c
blink       character  cursor      data  home  position  reset     subsystem
brightness  clear      customchar  dev   meta  power     scrollhz  uevent
root@raspberrypi:/mnt/nfs_server/LKM_Lab/03_I2C_LCD_16x2# echo 1 > /sys/class/alphalcd/lcdi2c/clear
root@raspberrypi:/mnt/nfs_server/LKM_Lab/03_I2C_LCD_16x2# echo "System calls!" > /dev/lcdi2c
```

5. Experiments

/dev/lcdi2c : File I/O Manipulation with ioctl in C

man7.org > Linux > man-pages Linux/UNIX system programming training

ioctl(2) — Linux manual page

NAME | SYNOPSIS | DESCRIPTION | RETURN VALUE | ERRORS | CONFORMING TO | NOTES | SEE ALSO | COLOPHON

Search online pages

```
IOCTL(2)                Linux Programmer's Manual                IOCTL(2)
```

NAME top

```
       ioctl - control device
```

SYNOPSIS top

```
       #include <sys/ioctl.h>

       int ioctl(int fd, unsigned long request, ...);
```

5. Experiments

/dev/lcdi2c : File I/O Manipulation with ioctl in Python

1

```python
import os, fcntl

dev_filename = '/dev/lcdi2c'
dev_meta_filename = '/sys/class/alphalcd/lcdi2c/meta'

ioctls = dict()

f = os.open(dev_meta_filename, os.O_RDONLY)

if f:
  print("Opened {}".format(dev_filename))
  meta = os.read(f, 512).decode('ascii').rstrip().split("\n")
  os.close(f)
  try:
    iioc = meta.index("IOCTLS:") + 1
  except ValueError as e:
    print("No IOCTLS section in meta file")
    raise e
  ioctls = dict(k.split("=") for k in [s.lstrip() for s in meta[iioc:]])
else:
    print("Unable to open meta file for driver.")
    exit(1)
```

2

```
""" Supported IOCTLs

dict_keys(['GETCHAR', 'SETCHAR', 'GETPOSITION', 'SETPOSITION', 'RESET',
'HOME', 'GETBACKLIGHT', 'SETBACKLIGHT', 'GETCURSOR', 'SETCURSOR', 'GETBLINK',
'SETBLINK', 'SCROLLHZ', 'GETCUSTOMCHAR', 'SETCUSTOMCHAR', 'CLEAR'])

"""
```

5. Experiments

/dev/lcdi2c : File I/O Manipulation with ioctl in Python (cont.)

```
""" Supported IOCTLs

dict_keys(['GETCHAR', 'SETCHAR', 'GETPOSITION', 'SETPOSITION', 'RESET',
'HOME', 'GETBACKLIGHT', 'SETBACKLIGHT', 'GETCURSOR', 'SETCURSOR', 'GETBLINK',
'SETBLINK', 'SCROLLHZ', 'GETCUSTOMCHAR', 'SETCUSTOMCHAR', 'CLEAR'])
```

```
write_ioctl('CLEAR', '1')
write_ioctl('SETBLINK', '1')
write_ioctl('SETBLINK', '1')
write_ioctl('SETPOSITION', [0,0])
```

```
def write_ioctl(cmd: str, value):
    if cmd not in ioctls:
        print("[ERROR] Not supported IOCTL: {}".format(cmd))
        return 1
    cmd_hex = ioctls[cmd]
    s = array.array('B')
    if isinstance(value, str):
        value = bytes(value, encoding='ascii')
    s.extend(value)
    f = open(dev_filename, 'rb+')
    fcntl.ioctl(f, int(cmd_hex, base=16), s)
    f.close()
    return s

def write(input: str):
    output = bytes(input, encoding='ascii')
    f = open(dev_filename, 'rb+')
    f.write(output)
    f.close()
```

```
def set_cursor(x, y):
    return write_ioctl('SETPOSITION', [x, y])
```

```
while True:
    event = keyboard.read_event()

    valid_char_events = [ 'a', 'b', 'c', 'd', 'e', 'f', 'g',
                          'h', 'i', 'j', 'k', 'l', 'm', 'n',
                          'o', 'p', 'q', 'r', 's', 't', 'u',
                          'v', 'w', 'x', 'y', 'z',
                          '0', '1', '2', '3', '4',
                          '5', '6', '7', '8', '9']

    if event.event_type == keyboard.KEY_DOWN:
        if event.name in valid_char_events:
            char = event.name
            x, y = get_cursor()
            if x == cols-1: y += 1
            write(char)
            set_cursor(x+1, y)
```

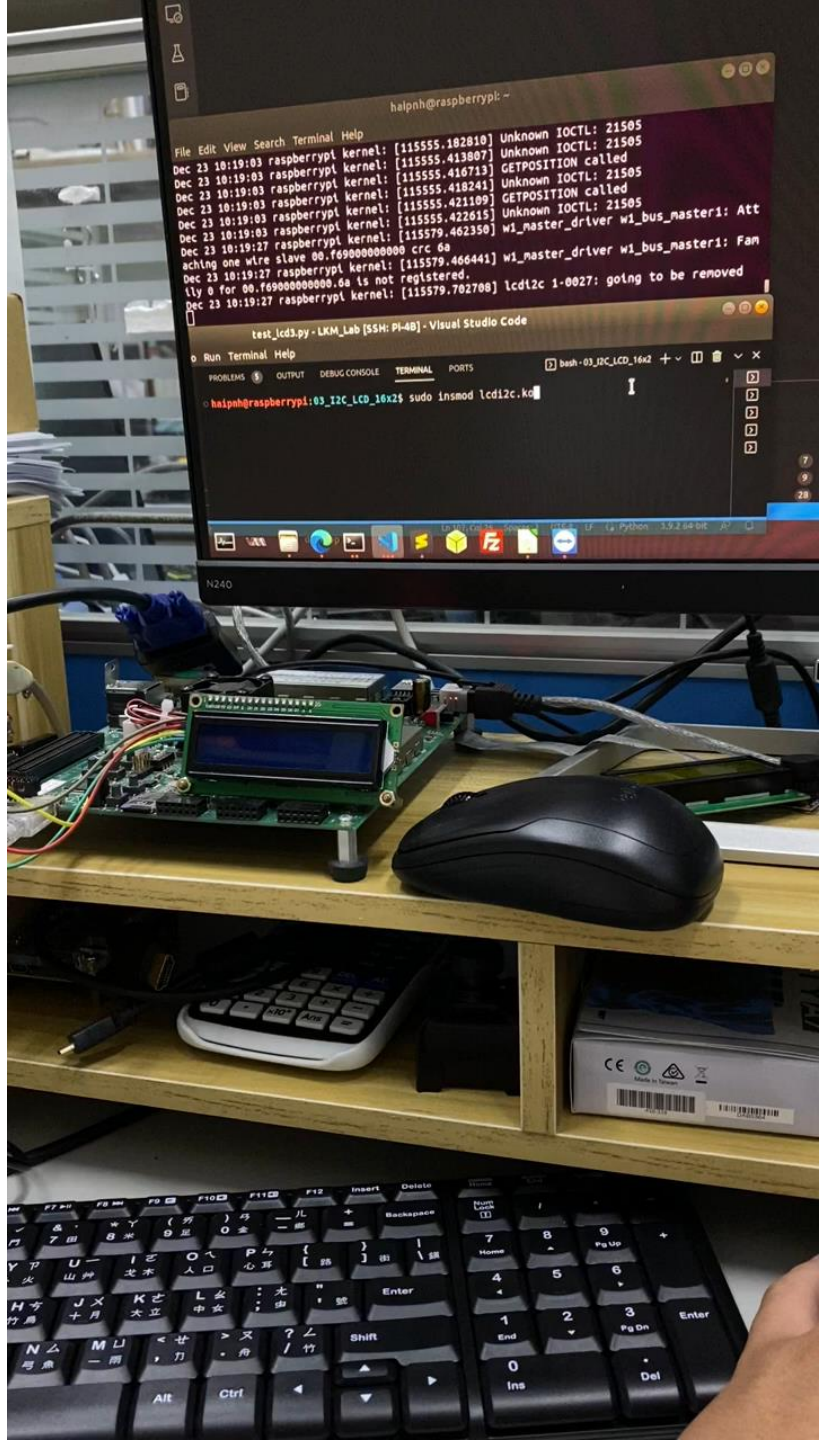
6. An Example Application

LCD On-screen Text Editor

Project source code:

https://github.com/haipnh/lcdi2c/tree/master

Demo video: https://youtu.be/GQAF7vmpGsA



References

1. https://www.raspberrypi.com/documentation/computers/linux_kernel.html
2. https://github.com/raspberrypi/linux
3. http://derekmolloy.ie/writing-a-linux-kernel-module-part-2-a-character-device
4. https://github.com/bogdal/rpi-lcd
5. https://www.kernel.org/doc/html/v5.15/i2c/index.html

Thank you for Listening