Control GPIOs of Raspberry Pi as Pseudo-file in Kernel Space

(apply on 7 segments display)

1701210963 Da-Wei Lee

Why this topic?

- Combine my previous major in Electronic Engineering (EECS)
- I was used to manipulate GPIO
  either in lower-level chips (e.g. ARM M series)
  or in User space (by using driver or library)

Goal of this subject

- Create a pseudo-file in Sysfs that can be used by any user
- Create a kernel thread that run in background which will simplify the programming process

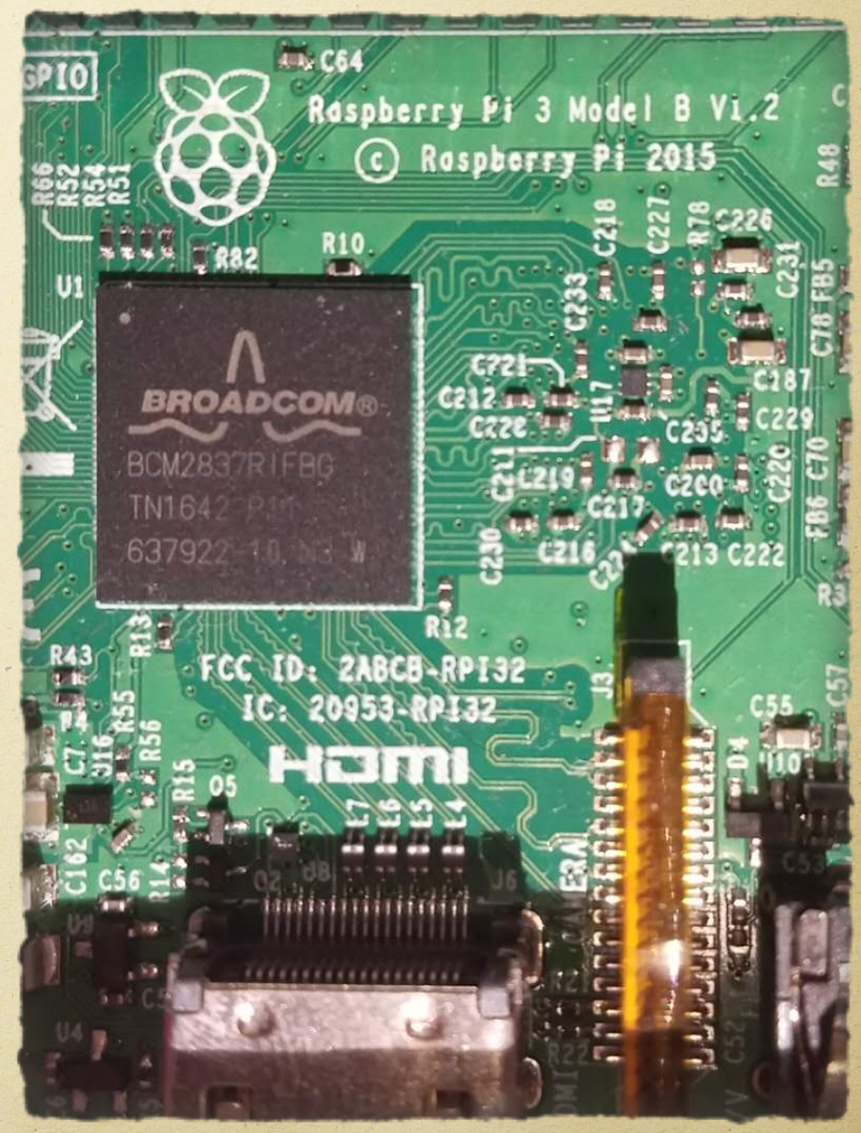
Table of Content

- Hardware background knowledge
  - Prerequisite environment, 7-segment Display
- Raspberry Pi Linux Kernel
  - GPIO, Sysfs, Thread…
- Experiment steps and Result

Hardware Background Knowledge and Environment

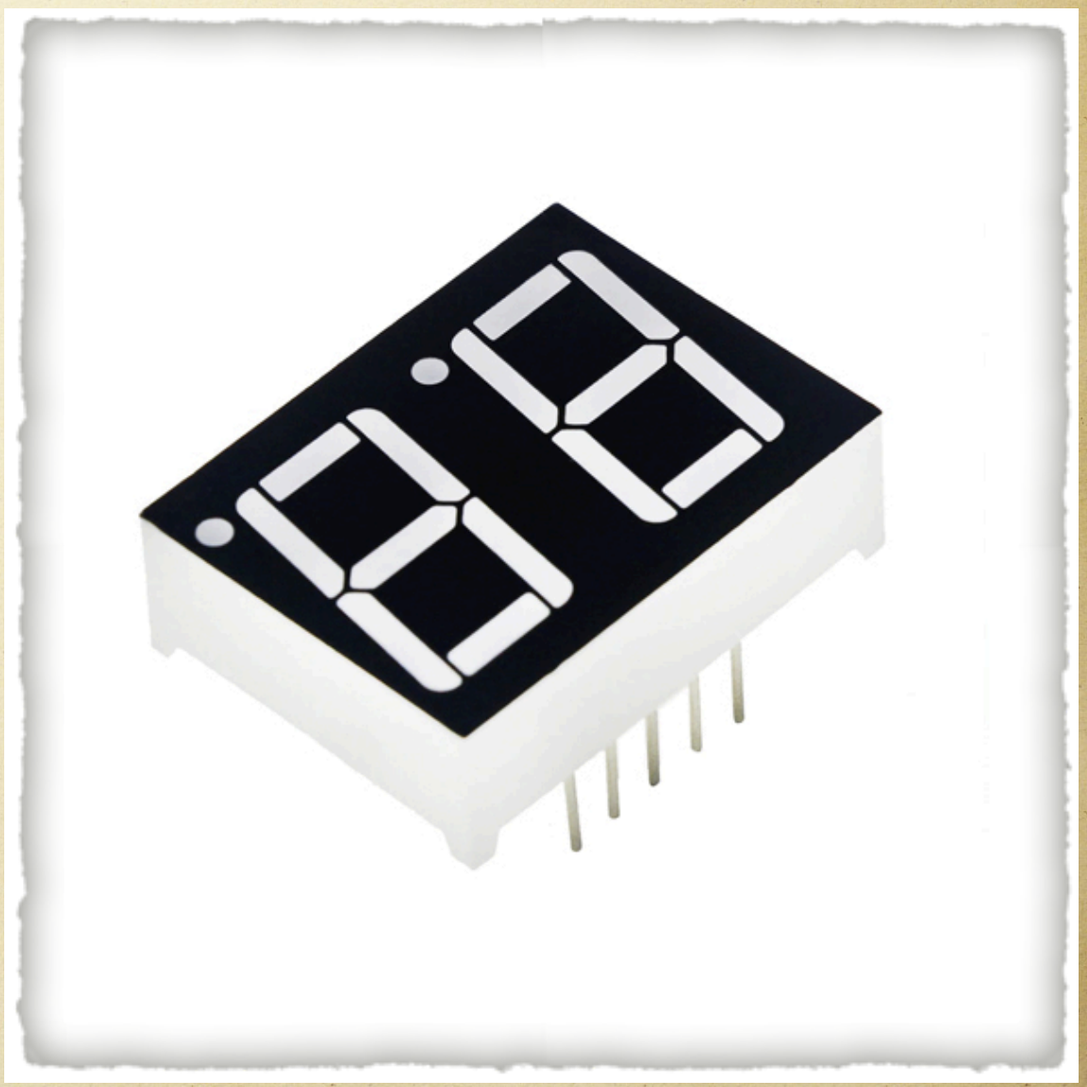
Experiment Environment

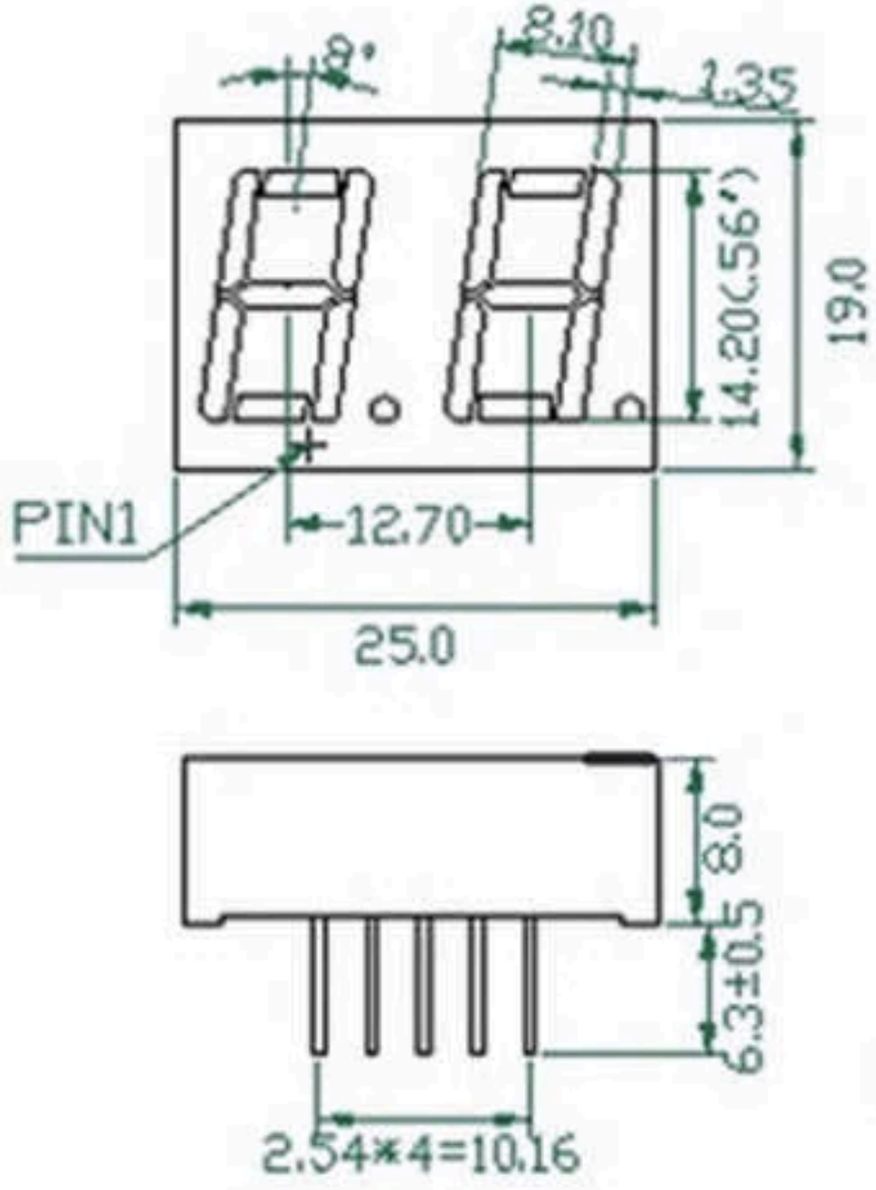
- Raspberry Pi 3 Model B Rev 1.2
- Chip: BCM2837
- Kernel version: 4.19.42-v7+
- Operating system: Raspbian GNU/Linux 9.9 (stretch)

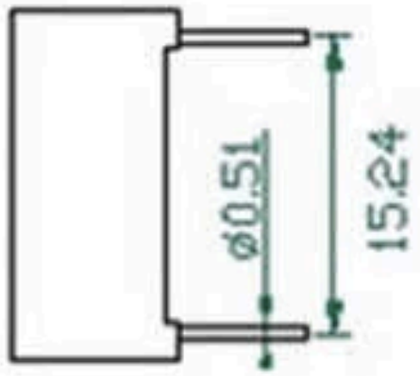


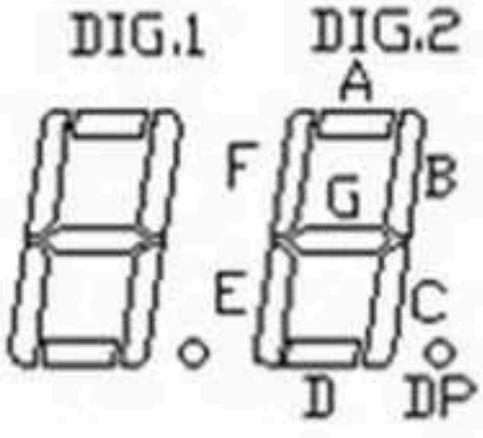
7-segment Display

- Common Cathode Double Digits
- High voltage for LEDs to display
- Low voltage for the digit to display



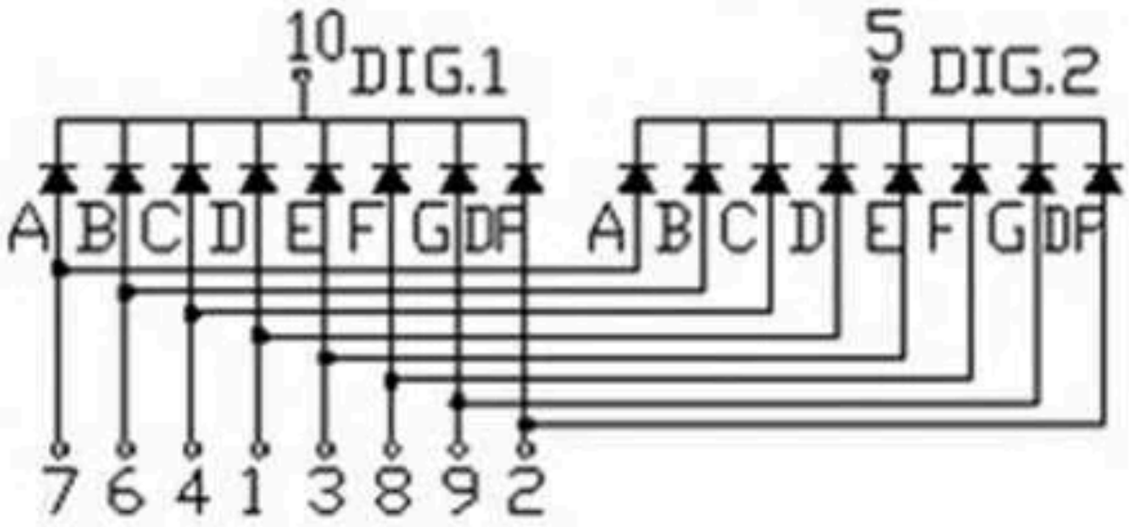






Size:mm

Common cathode



Truth Table of LEDs

	A	B	C	D	E	F	G
0	1	1	1	1	1	1	0
1	0	1	1	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1
7	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1

Specifications

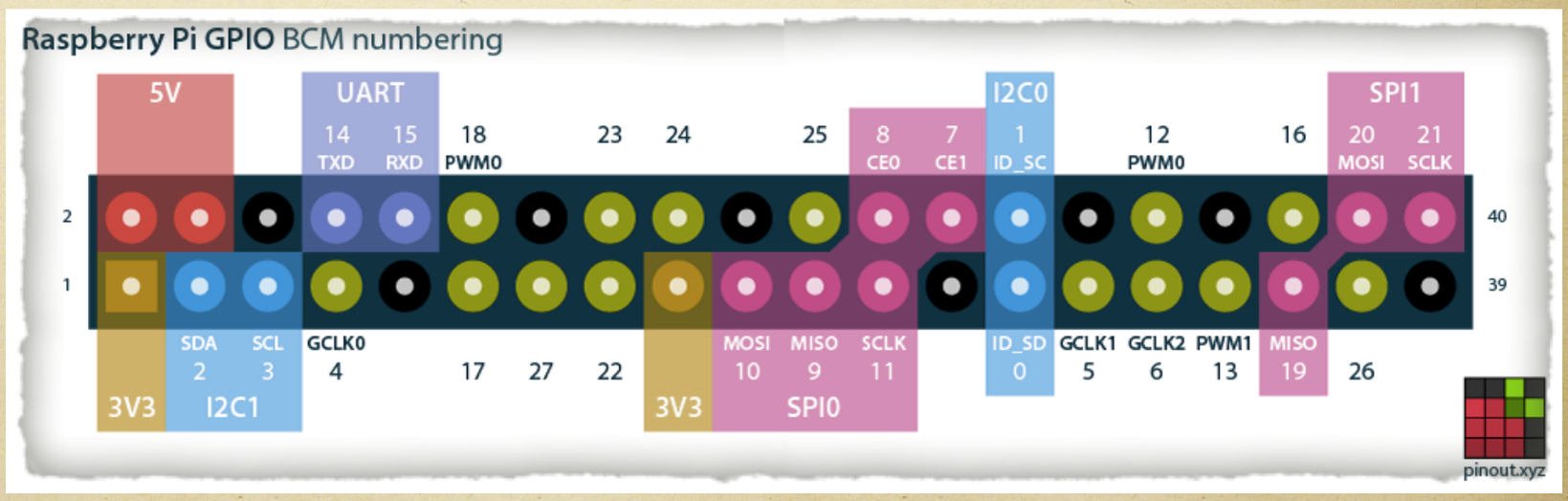
Raspberry Pi 3	
Processor Operating Voltage	3.3V
Maximum current through each I/O	16mA
Maximum total current drawn from all I/O pins	54mA

7-segment Display	
Forward Voltage	1.84V (measured with 10mA)
Maximum Continuous Forward Current	20mA

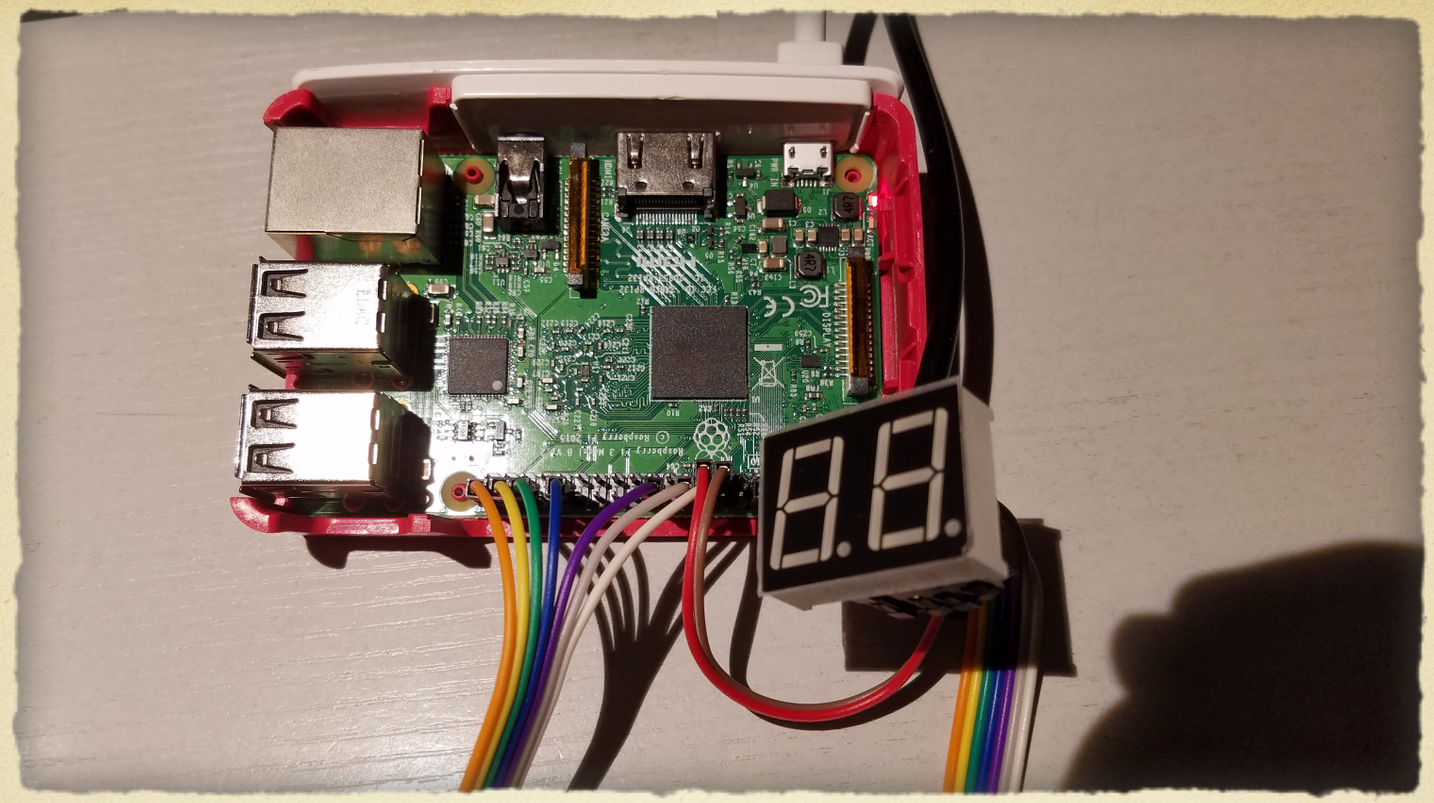
- Resistor: 3.3V / 330Ω = 10mA < 16mA

Raspberry Pi GPIO Pinout

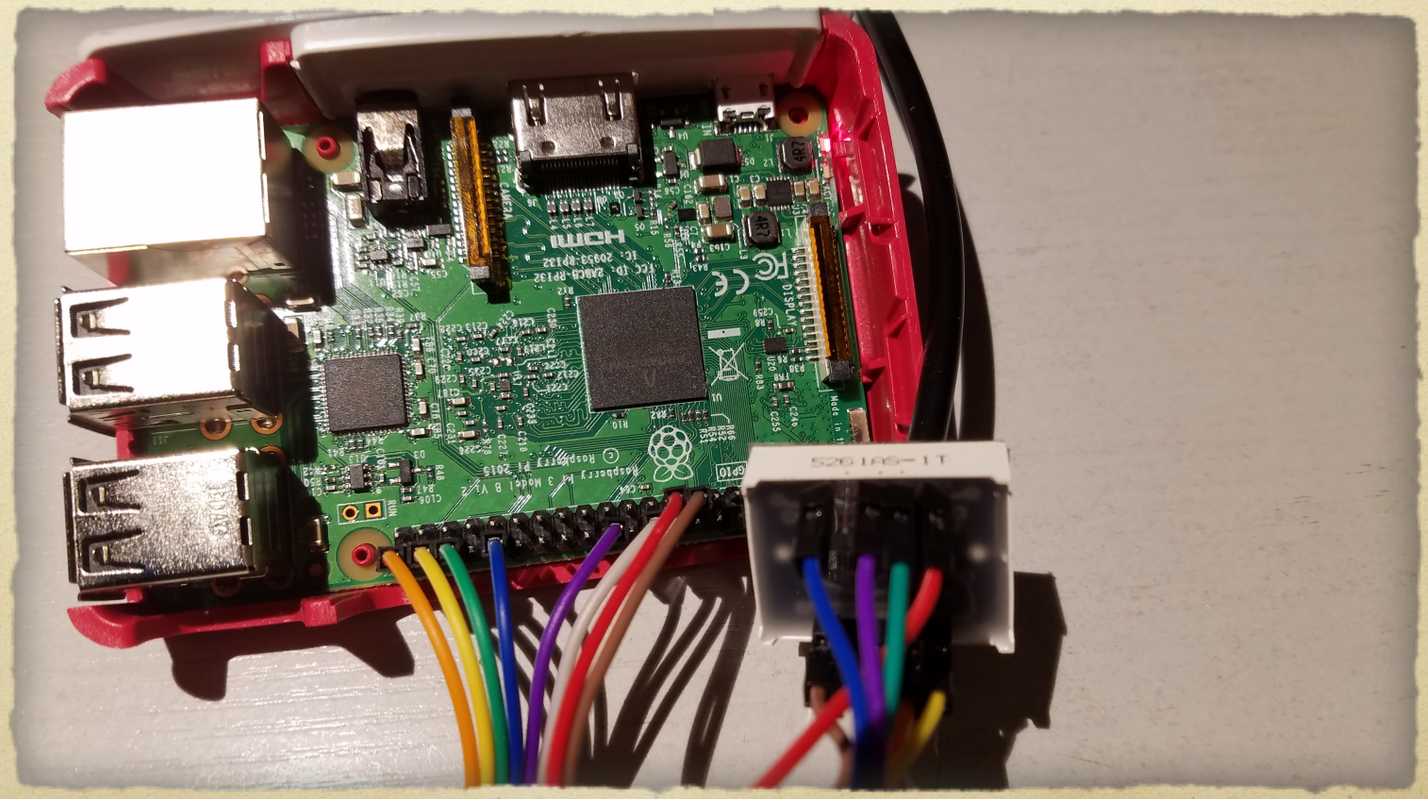
- LEDs (A to G) : 21, 20, 16, 12, 25, 24, 23
- Digit Pin (D0, D1): 22, 27



Front

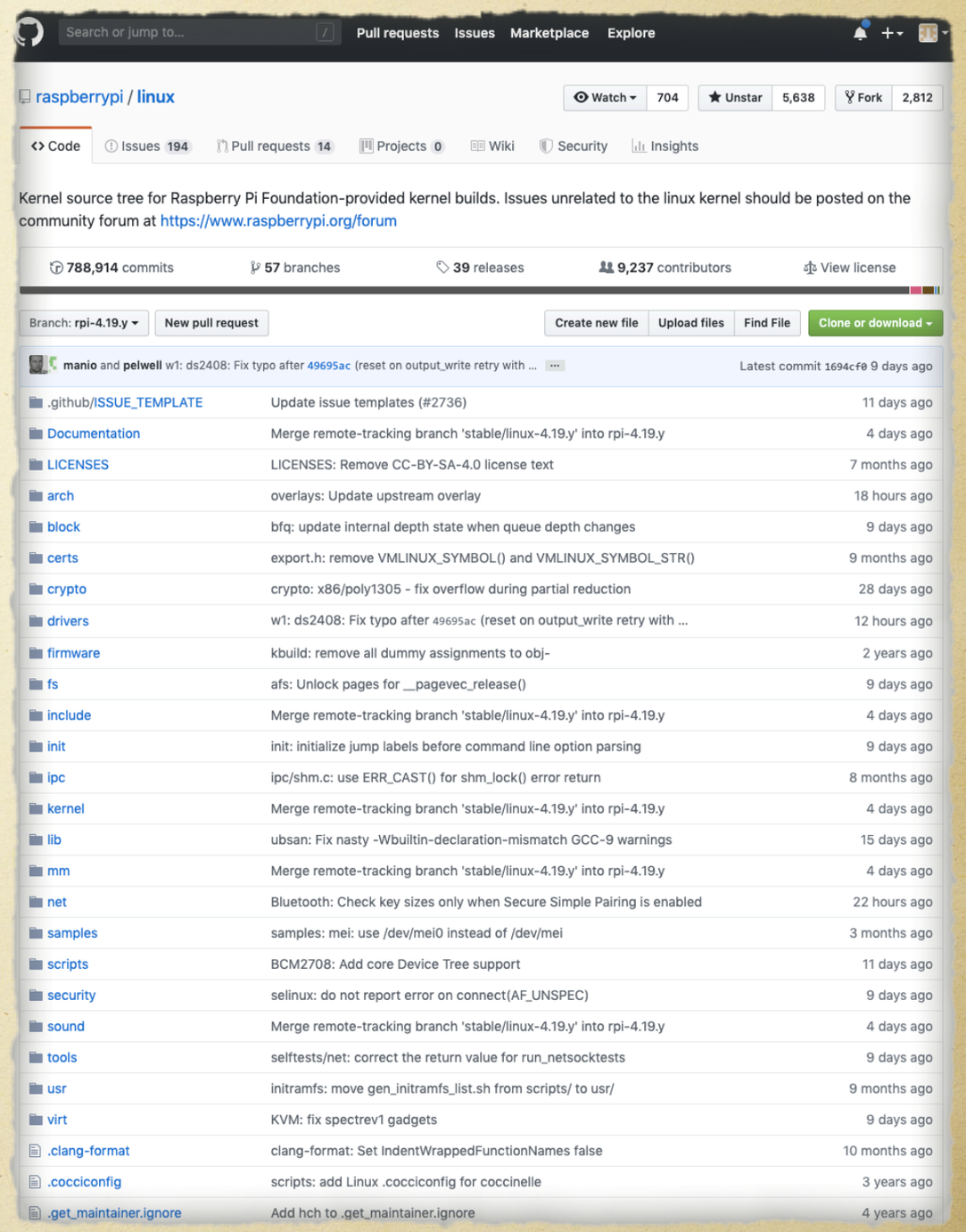


Back



Raspberry Pi Linux Kernel

4.19.42-v7+



GPIO

1. Request GPIO
2. Set direction
   (in my case,
   output mode)
3. Set value
   (high or low voltage)
4. Free GPIO

```c
/* <linux/gpio.h> */

static inline int gpio_request(unsigned gpio, const char *label)
{
    return -ENOSYS;
}

static inline void gpio_free(unsigned gpio)
{
    might_sleep();

    /* GPIO can never have been requested */
    WARN_ON(1);
}

static inline int gpio_direction_output(unsigned gpio, int value)
{
    return -ENOSYS;
}

static inline void gpio_set_value(unsigned gpio, int value)
{
    /* GPIO can never have been requested or set as output */
    WARN_ON(1);
}
```

Thread and Delay

```c
/* <linux/kthread.h> */

#define kthread_run(threadfn, data, namefmt, ...)                       \
({                                                                      \
    struct task_struct *__k                                             \
        = kthread_create(threadfn, data, namefmt, ## __VA_ARGS__); \
    if (!IS_ERR(__k))                                                   \
        wake_up_process(__k);                                           \
    __k;                                                                \
})

int kthread_stop(struct task_struct *k);

bool kthread_should_stop(void);

/* linux/sched.h */

struct task_struct

extern int sched_setscheduler(struct task_struct *, int, const struct sched_param *);

/* self define */

struct sched_param
{
    int sched_priority;
};

/* <linux/delay.h> */

void usleep_range(unsigned long min, unsigned long max);
```

Sysfs

- Default file mode 0444 (read only)
- Default allow 0644 (otherwise compile will fail)

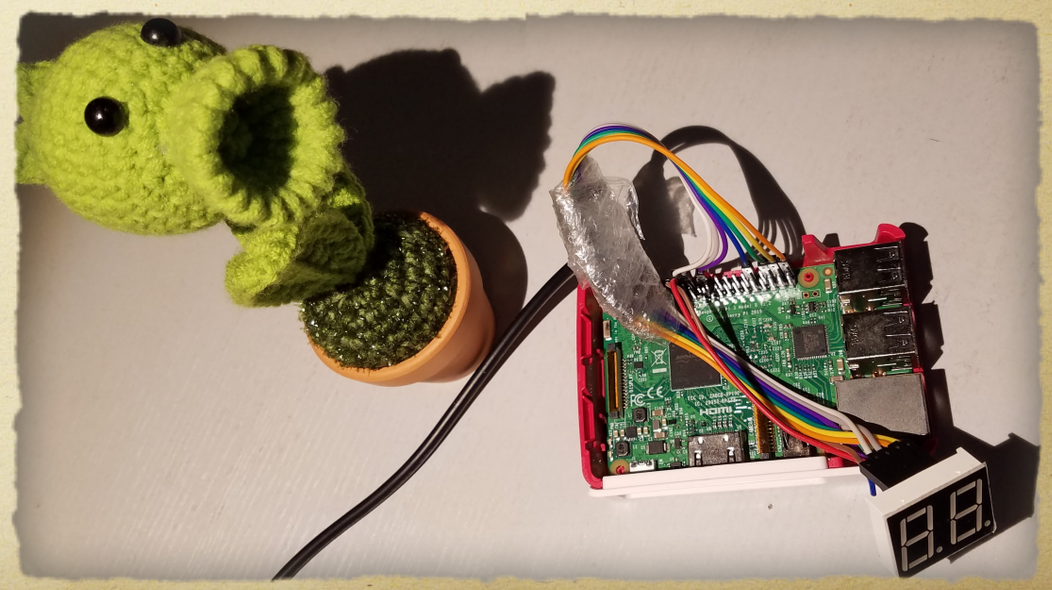
```c
/* linux/kobject.h  (inlcude in <linux/module.h>) */

extern struct kobject * __must_check kobject_create_and_add(const char *name,
                                struct kobject *parent);

extern void kobject_put(struct kobject *kobj);

struct kobj_attribute {
    struct attribute attr;
    ssize_t (*show)(struct kobject *kobj, struct kobj_attribute *attr,
            char *buf);
    ssize_t (*store)(struct kobject *kobj, struct kobj_attribute *attr,
             const char *buf, size_t count);
};

/* linux/sysfs.h */

#define __ATTR(_name, _mode, _show, _store) {                \
    .attr = {.name = __stringify(_name),                     \
         .mode = VERIFY_OCTAL_PERMISSIONS(_mode) },          \
    .show   = _show,                                         \
    .store  = _store,                                        \
}

/* linux/stat.h and uapi/linux/stat.h */

#define S_IRUGO (S_IRUSR|S_IRGRP|S_IROTH)
#define S_IWUSR 00200
#define S_IWGRP 00020
#define S_IWOTH 00002
```

Experiment Steps

Setup the hardware

- Assembly the hardware
- Update the kernel version (optional)
- Make sure the kernel headers are ready

  $ sudo apt-get install raspberrypi-kernel-headers



7-segment Display

```c
/* seven_seg.c */

// GPIO pinout
const short SegmentPin[TotalLED] = {21, 20, 16, 12, 25, 24, 23, 18}; // a, b, c, d, e, f, g,
decimal point
const short DigitPin[TotalDigits] = {22, 27, 17, 4};                 // digit 0~3

// Number Table
const short NumTable[11][PinUsed] = {{1, 1, 1, 1, 1, 1, 0},  // 0
                                     {0, 1, 1, 0, 0, 0, 0},  // 1
                                     {1, 1, 0, 1, 1, 0, 1},  // 2
                                     {1, 1, 1, 1, 0, 0, 1},  // 3
                                     {0, 1, 1, 0, 0, 1, 1},  // 4
                                     {1, 0, 1, 1, 0, 1, 1},  // 5
                                     {1, 0, 1, 1, 1, 1, 1},  // 6
                                     {1, 1, 1, 0, 0, 0, 0},  // 7
                                     {1, 1, 1, 1, 1, 1, 1},  // 8
                                     {1, 1, 1, 1, 0, 1, 1},  // 9
                                     {0, 0, 0, 0, 0, 0, 0}}; // Empty

short NumToShow[TotalDigits] = {10, 10, 10, 10}; // unit, ten, hundred, thousand
```

```c
/* seven_seg.h */

#ifndef SEVEN_SEG_H
#define SEVEN_SEG_H

// Global define
#define TotalLED      8
#define PinUsed       7
#define TotalDigits   4
#define DigitUsed     2
#define DISP_DELAY_US 10000 // for visual persistence

// GPIO pinout
extern const short SegmentPin[TotalLED];  // a, b, c, d, e, f, g, decimal point
extern const short DigitPin[TotalDigits]; // digit 0~3

// Number Table
extern const short NumTable[10 + 1][PinUsed]; // from 0 to 9 + a None

// Basic Function
extern void init_7seg_gpio(void); // Initialize the GPIOs used by 7-seg display
extern void free_7seg_gpio(void); // Free the GPIOs used by 7-seg display
extern void clearDisplay(void);   // Set the digits to "not display"

// Higher Level Function
extern void setNumber(int number);                // Set the number to display
extern void showAllDigits(unsigned int duration); // Show a number for a duration (usec)

#endif // SEVEN_SEG_H
```

(continue) init and free

- init
  - request
  - set direction
- free

```c
/* seven_seg.c */

#define GPIO_REQUEST_LABEL "rpi_7seg"

// Function
void init_7seg_gpio(void)
{
    int i;
    for (i = 0; i < PinUsed; i++)
    {
        gpio_request(SegmentPin[i], GPIO_REQUEST_LABEL);
        gpio_direction_output(SegmentPin[i], 0);
    }
    for (i = 0; i < DigitUsed; i++)
    {
        gpio_request(DigitPin[i], GPIO_REQUEST_LABEL);
        gpio_direction_output(DigitPin[i], 1);
    }
}

void free_7seg_gpio(void)
{
    int i;
    for (i = 0; i < PinUsed; i++)
    {
        gpio_free(SegmentPin[i]);
    }
    for (i = 0; i < DigitUsed; i++)
    {
        gpio_free(DigitPin[i]);
    }
}
```

(continue) set and show

```c
/* seven_seg.c */

void setDigit(int digit)
{
    int i;
    for (i = 0; i < PinUsed; i++)
    {
        gpio_set_value(SegmentPin[i], NumTable[digit][i]);
    }
}

void showDigit(int pos)
{
    int i;
    for (i = 0; i < DigitUsed; i++)
    {
        gpio_set_value(DigitPin[i], (pos != i));
    }
}

void clearDisplay(void)
{
    int pos;
    for (pos = 0; pos < DigitUsed; pos++)
    {
        NumToShow[pos] = 10;
    }
}
```

(continue) display

```c
/* seven_seg.c */

void setNumber(int number)
{
    int pos;
    if (number < 0)
    {
        // If receive negative number, then clear the display.
        clearDisplay();
        return;
    }
    for (pos = 0; pos < DigitUsed; pos++)
    {
        // If the number position greater than 10 is 0 then don't display it!
        if (number == 0 && pos > 0)
            NumToShow[pos] = 10;
        else
        {
            NumToShow[pos] = number % 10;
            number /= 10;
        }
    }
}

void showAllDigits(unsigned int duration)
{
    int t, pos;
    for (t = 0; t < duration; t += DISP_DELAY_US)
    {
        for (pos = 0; pos < DigitUsed; pos++)
        {
            showDigit(pos);
            setDigit(NumToShow[pos]);
            usleep_range(DISP_DELAY_US, DISP_DELAY_US);
        }
    }
}
```

Kernel Module - GPIO

- Call the function that we've made in seven_seg.h

```c
/* rpi_7seg_module.c GPIO */

#include "seven_seg.h"

void rpi_7seg_gpio_init(void)
{
    printk(KERN_INFO "RPi7Seg: starting gpio...");
    init_7seg_gpio();
    printk(KERN_INFO "RPi7Seg: starting gpio done.");
}

void rpi_7seg_gpio_exit(void)
{
    printk(KERN_INFO "RPi7Seg: stopping gpio...");
    free_7seg_gpio();
    printk(KERN_INFO "RPi7Seg: stopping gpio done.");
}
```

Kernel Module - Sysfs

- In order to let the compile pass, we need to redefine the VERIFY_OCTAL_PERMISSIONS macro

```c
/* rpi_7seg_module.c Sysfs */

#define PSEUDO_FILENAME display

static struct kobject *g_kobject;

static ssize_t set_7seg(struct kobject *kobj, struct kobj_attribute *attr, const char *buff,
size_t count)
{
    int number;
    sscanf(buff, "%d", &number);
    printk(KERN_INFO "RPi7Seg: received number: %d\n", number);
    setNumber(number);

    return count;
}

/* warning! need write-all permission so overriding check */
#undef VERIFY_OCTAL_PERMISSIONS
#define VERIFY_OCTAL_PERMISSIONS(perms) (perms)
#define PERMISSION (S_IWUSR | S_IRUGO | S_IWGRP | S_IWOTH) // 0666
static struct kobj_attribute rpi_7seg_attribute
                            = __ATTR(PSEUDO_FILENAME, PERMISSION, NULL, set_7seg);

void rpi_7seg_sysfs_init(void)
{
    printk(KERN_INFO "RPi7Seg: starting sysfs...");
    g_kobject = kobject_create_and_add("rpi_7seg", NULL);
    if (sysfs_create_file(g_kobject, &rpi_7seg_attribute.attr))
    {
        pr_debug("failed to create rpi_7seg sysfs!\n");
    }
    printk(KERN_INFO "RPi7Seg: starting sysfs done.");
}

void rpi_7seg_sysfs_exit(void)
{
    printk(KERN_INFO "RPi7Seg: stopping sysfs...");
    kobject_put(g_kobject);
    printk(KERN_INFO "RPi7Seg: stopping sysfs done.");
}
```

Kernel Module - Thread

- Create a thread
- Set its priority to 45
- Put in an infinity loop
- Stop when kthread_should_stop() is true

```c
/* rpi_7seg_module.c Thread */

#include <linux/kthread.h>

#define THREAD_PRIORITY 45
#define THREAD_NAME "rpi_7seg"

struct task_struct *g_task;

struct sched_param
{
    int sched_priority;
};

int rpi_7seg_thread(void *data)
{
    struct task_struct *TSK;
    struct sched_param PARAM = {.sched_priority = MAX_RT_PRIO - 50};
    TSK = current;

    PARAM.sched_priority = THREAD_PRIORITY;
    sched_setscheduler(TSK, SCHED_FIFO, &PARAM);

    while (1)
    {
        // Display the minimal time in case of the number change
        showAllDigits(DISP_DELAY_US * DigitUsed);
        if (kthread_should_stop())
            break;
    }
    return 0;
}

void rpi_7seg_thread_init(void)
{
    printk(KERN_INFO "RPi7Seg: starting thread...");
    g_task = kthread_run(rpi_7seg_thread, NULL, THREAD_NAME);
    printk(KERN_INFO "RPi7Seg: starting thread done.");
}

void rpi_7seg_thread_exit(void)
{
    printk(KERN_INFO "RPi7Seg: stopping thread...");
    kthread_stop(g_task);
    printk(KERN_INFO "RPi7Seg: stopping thread done.");
}
```

Kernel Module

```c
/* rpi_7seg_module.c Module */

#include <linux/init.h>
#include <linux/module.h>  // module_init, module_exit, kobject

static int __init rpi_7seg_init(void)
{
    printk(KERN_INFO "RPi7Seg: staring...");
    rpi_7seg_gpio_init();
    rpi_7seg_thread_init();
    rpi_7seg_sysfs_init();
    printk(KERN_INFO "RPi7Seg: staring done.");
    return 0;
}

static void __exit rpi_7seg_exit(void)
{
    printk(KERN_INFO "RPi7Seg: stopping...");
    rpi_7seg_sysfs_exit();
    rpi_7seg_thread_exit();
    rpi_7seg_gpio_exit();
    printk(KERN_INFO "RPi7Seg: stopping done.");
}

module_init(rpi_7seg_init);
module_exit(rpi_7seg_exit);
```

Compile and Result

Makefile

- Directory Structure
  - seven_seg.c
  - seven_seg.h
  - rpi_7seg_module.c
  - Makefile

```
/* Makefile */

ifeq ($(KERNELRELEASE),)

    KERNELDIR ?= /lib/modules/$(shell uname -r)/build
    PWD := $(shell pwd)

modules:
    make -C $(KERNELDIR) M=$(PWD) modules

clean:
    rm -rf *.o *~ core .depend .*.cmd *.ko *.mod.c .tmp_versions

.PHONY: modules clean

else

    # called from kernel build system: just declare what our modules are
    obj-m := rpi_7seg.o

    rpi_7seg-objs := rpi_7seg_module.o seven_seg.o

endif
```

The outcome will be “rpi_7seg.ko”

Installation Script

```
/* smart_install.sh */

#!/bin/sh

BLUE="\033[34m"
END="\033[0m"

echo $BLUE "Compiling rpi_7seg..." $END
make
echo $BLUE "Check if the rpi_7seg module is existing..." $END
lsmod | grep rpi_7seg
if [ "$?" = "0" ]
then
    echo $BLUE "Found the rpi_7seg, remove the old one..." $END
    sudo rmmod rpi_7seg
fi
echo $BLUE "Installing the rpi_7seg module..." $END
sudo insmod rpi_7seg.ko

echo $BLUE "The install info of dmesg (/var/log/syslog)" $END
dmesg --color=always | tail -n 7

echo $BLUE "Checking the permission / file mode of the \"/sys/rpi_7seg/display\"" $END
ls -lh /sys/rpi_7seg/display

echo $BLUE "The rpi_7seg is successfully installed at \"/sys/module/rpi_7seg\"!" $END
echo $BLUE "And you should be able to manipulate the 7-segment display using the pseudo-file
\"/sys/rpi_7seg/display\"!" $END
```

After Install

- The module should be installed at
  /sys/module/rpi_7seg
- The pseudo-file should shown at
  /sys/rpi_7seg/display
- And then use stdin to assign the value to it
  echo 87 > /sys/rpi_7seg/display # show 87
  echo -1 > /sys/rpi_7seg/display # clear display

Dice example

```
/* dice.py */

#!/usr/bin/python3
import random
import sys
import time

MAX_ROUND = 50

if len(sys.argv) < 2:
    print("Usage: dice.py max_num (this will show a random number between 1~max_num)")
elif int(sys.argv[1]) < 0:
    print("Please input positive number!")
else:
    max_num = int(sys.argv[1])

    with open('/sys/rpi_7seg/display', 'w') as rpi:
        rpi.write(str(-1))  # first clean the display

    for i in range(MAX_ROUND):
        number = random.randrange(1, 1 + max_num)
        with open('/sys/rpi_7seg/display', 'w') as rpi:
            rpi.write(str(number))
        time.sleep(i * 0.003)  # make the dice effect
```

Conclusion

- I was used to manipulate the GPIO using the direct register access, but having some problem with writing memory
  (success in user space by using /dev/mem)
- Should be aware of not trying to do the user space way (e.g. tend to use syscall)

Project Source Code

- Github: daviddwlee84/**LinuxKernel**/Subject/Final