Systemprogrammierung

22.10.

- Tastatur, Bildschirm angeschlossen. ✓
- changed pwd ✓
- changed hostname (richardpi) ✓
- Ethernet connection ✓ Ethernet HW LABOR.

→ Einstellungen Ethernet → WLAN?

+ WLAN Modul:

▶ under /etc/network/interfaces:

add

```
allow-hotplug wlan0
iface wlan0 inet manual
```

▶ add wlan-connections:

under /etc/wpa-supplicant/wpa-supplicant.conf

add

```
network={
        ssid="Theater"
        psk=xxxxxx
}
```

◆ Found CMake projekt to build kernel module?

Install kernel-headers:

```
sudo apt update
sudo apt install linux-headers-
    -$(uname -r)
```

→ Troubleshooting

→ Too complex, MakeFile is easier to use.

TODO: build with MakeFile.
TODO: read about kernel? Research.

29.10.2020

No Display. (Smart Driving LABor)
Get access per SSH (for Home.)
?Add WLAN-config to /etc/wpa-supp....
microSSD → PC terminal.

```
sudo nano
/etc/wpa-supp....
```

Get to the display ? — VNC
Get to the SSH ? — IP Adress in Network.

Local Machine: `ifconfig`

Just guess the IP-Address with trying out:

```
(sudo) ssh pi@ip-adress
```

ip-address: 192.168.1.125
(HTWK SmaLA)

add to .bashrc as alias

```
alias raspiSD='ssh pi@192.168.1.125
```

! TODO: same for HOME.

VNC: client Programm: Remmina.

Raspi: Enable VNC server? on boot?

sudo raspi-config
Interfacing Options: VNC enable
btw enable other
Interfaces: I2C, SP

Kernel Modules

Nützliche Infos:

sudo insmod module.ko

sudo rmmod module

dmesg: kernel display msgs.

uname -r: kernel version

lsmod: list kernel modules(loaded)

Resources: "Einführung in die Linux Treiber-
entwicklung": Interstaatli
v1.3 Hochschule
für Technik

Building a kernel module

MakeFile

```
obj-m += test.o

all: make -C /lib/modules/$(shell uname -r)/build
     M=$(PWD) modules

clean: make -C /lib/modules/$(shell uname -r)/build.
     M=$(PWD) clean.
```

Gelernt:

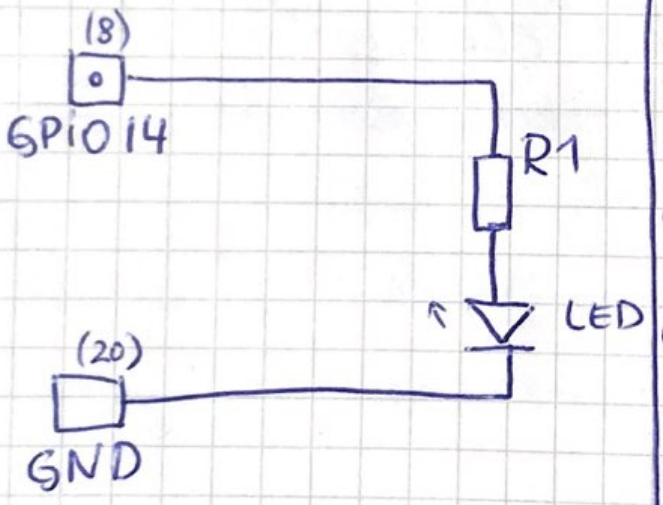
5.11.2020

TODO

- Setup WLAN-Access for Home ✓
- TODO Read Hardware stuff from Book
- TODO small Projekt (GPIO access)

> Seite 365: Blinken einer LED

Schaltung:



Code led.py

```
import RPi.GPIO as GPIO
import time
LEDGPIOPin = 14
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(LEDGPIOPin,
           GPIO.OUT)

DELAYON = 1.0
DELAYOFF=1.0
:
```

Resources: "Die elektronische Welt mit Raspberry Pi entdecken"
Erik Bartman.

```
:

while True:
    GPIO.output(LED_GPIO_Pin,
    GPIO.HIGH)
    time.sleep(DELAY_ON)
    GPIO.output(LED_GPIO_Pin,
    GPIO.LOW)
    time.sleep(DELAY_OFF)
```

- RPi.GPIO → Lib
- BCM mode → GPIO Pins numbers
- setup → festlegen der Richtung
- output → Signal schicken.

12.11.2020

Geräteklassen: char devices (serial API) | block devices (Datasyste

- Bytestreams
  System sieht sie als Dateien
- Diff: Verwaltung der Daten im Kernel

network devices (austausch von Daten zwischen Systemen)

- Angeschlossene HW
- keine Dateien in der Repräsen

Gerätedateien: char devices | block devices | socket devices

/dev : devices.

>ls -la

d... directory
c... char dev
b... block dev
s... socket dev
-... normal
l... link
p... fifo

Major Number: Treiber
Minor Number: Funktionalität für das entspr. Gerät

> 1 Treiber für mehrere Geräte

dev_t struct for Minor & MAJor numbers.

MAJOR (dev-t dev)
MiNOR (dev-t dev)
MKDEV (int major, int minor)

procfs : Infos über Kernelsubsystem
process : Speichernutzung
file : Peripherie
system : Steuerung (teilweise) Kernel Modul-verhalten

Ubuntu: : Laden Kernelmodule.
/dev/kmem

sysfs : Infos über KernelObjekte
> Übergabe in der Userspace
> Konfigs aus Userspace
> binäre Schnittstellen

/udev
- Verwaltung /dev
- in Userspace
- Abstrahierte Schnittstelle
- Ein Gerät hinzugefügt
  ⇓
  Kernel-Event ausgelöst
  ⇕
  /udev informiert.

Resources: pcwelt.de: Linux Ordner Struktur.

```
int hello_start(void)
// register device (char device's) numbers:
result = alloc_chrdev_region(&this_dev_t,
                             0
                             count
                             char_name)

if(result<0)
{ printk(KERN_INFO "Cannot alloc char dev
  numbers. Aborting!\n");
  return result;
}

// register char devices

  pThis_cdev = cdev_alloc();
  pThis_cdev->owner = THIS_MODULE,
// pThis_cdev->ops = ?
// add char device
  result = cdev_add(pThis_cdev, this_dev_t,
                    count);
  if (result<0)
  { printk(KERN_ERR "Couldn't add

  }return 0;
failed_add_cdev_: clear_cdev_chdev_region
  return result;
```

! use alloc_chrdev_region

~~register_chrdev_region~~ anstatt

Allozierung von freien Gerätenummern.
Kompatibel mit allen Systemen
mit jeder Menge verschiedenen Kernel Modulen

> List associated chrdev numbers:

cat /proc/devices

char devices
block devices

cat /proc/modules = lsmod

04.12.2020

TODO:
- Book kernel
- Create chrdev file for Kernel-User-Communik.
- fops file Operations ?

▶ Nice to have (Initialization)

```
static int __init on_init(void){....}
module_init(on_init);
```

Macros

on_exit function : ... __exit ...

```
static int __initdata var=0;
```

- no effect for loadable modules
- freeing muory of init function after loading

Erstellen Device Node:

```
struct class *my_class = class_create(
                              THIS_MODULE,
                              "My class");
device_create(my_class, NULL, dev,
              NULL, "MyNameOfTheDevice");
```

On exit not forget:

```
device_destroy(my_class, dev);
class_destroy(my_class);
cdev_del(&this_cdev);
```

List/see char device files under

```
ls /dev
```

see class under

```
ls /sys/kernel
```

17.12.2020

```
static struct file_operations this_fops={
    .owner    = THIS_MODULE
    .read     = read_func;
    .write    = write_func;
//  .unlocked_ioctl
    .open     = open_func;
    .release  = release_func;
//  .poll
```

KOMPILIERT!

Params:

```
read_func(struct file *, char *, size_t, loff_t *)
write_func(struct file*, const char *, size_t, loff_t *)
open_func(struct inode *inode, struct file *filp);
release_func(....)
```

file pointer private data / buffer / size of buffer / file position / offset

also another file representation. / file pointer

Printk(KERN_INFO "Smith");

kern_levels.h

KERN_INFO
KERN_ERR
KERN_ALERT
KERN_DEBUG
KERN_WARNING

LOG LEVELS.

.....

Compiling

-c // not indep. exec, but an obj file linked (at runtime of insmod) into the kernel

-O2 // Optimizing flag on (inline funcs?)

-W // Compiler warn ON

[-isystem /lib/modules/'uname -r'/build
[-I (headers?) include

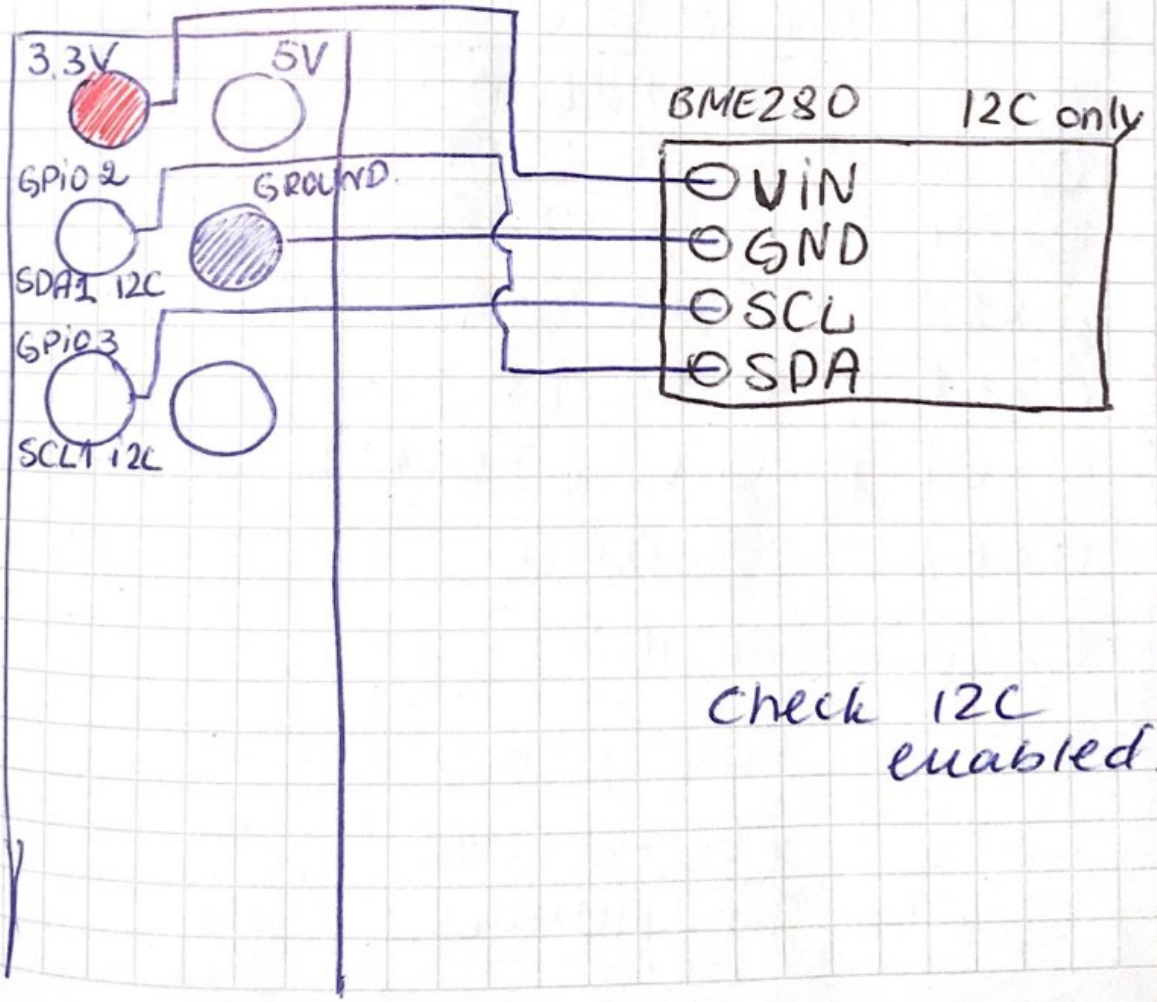
16.01.2021

PROJECT

TODO :
- SPECIFICATION
- BME280 I2C ?
- TEST IN USER SPACE IF WORKS ✓
- SMTH ELSE : ?

Schaltung

RASPBERY PI PINS, PORTS:



Check I2C enabled.

➢ Found Python Script von Matt Hawkins

➢ smbus used (RPi.smbus) Lib

? smbus read_i2c_block_data(...)
alternatives in C (for kernel?)

➢ it works!

➢ Useful addresses

0x76 – I2C Address (BME) I2C 1.
0XF2 – REG_CONTROL_HUMIDITY
0XF4 – REG_CONTROL
0x88 – Cal 1 (24 Byte long)
0xA1 – Cal 2 (1 Byte)
0xE1 – Cal 3 (7 Bytes)
0XF7 – REG_DATA
0xD0 – REG_ID
0xF5 – REG_CONFIG
0xFD – REG_HUM_MSB
0xFE – RE_HUM_LSB

oversample_time :

oversample_temp = oversample_pres =
= oversample_Humidity = 2.

FOUND RESSOURCES :

- bootlin Elixir Cross Referencer (linux headers)
- DerekMolloy.ie (kernel modules)
- Matt Hawkins (BME Python program)

23.01.2021

DONE: kabeln abgeholt
DONE: BASIC KERNEL MODULE (DUMMY)
DONE: LED GPIO, I2C BME280
USER SPACE

TODO: KERNEL SPACE – GPIO ACCESS.

Program 1: Button + LED + Interrupt Handler

- linux/gpio.h : - linux/interrupt.h

Associating an Interrupt request (IRQ) with GPIO

with:

```
static inline int gpio_to_irq(
    unsigned int gpio)
```

Interrupt: - High priority Condition

ISR Interrupt Service Routine

- Stops all processes, saves a state and handle an interrupt.
- The last state will be loaded → continuing.

Mapping GPIO → IRQ-Number

- request_irq(irqNumber,
  handle function → handler_function,
  → IRQF_TRIGGER_RISING,
  Trigger FLAG → "Name_of_Interrupt",
  → NULL);
  dev-id (for shared interrupts)

- free_irq( )

FLAGS:
- RISING
- FALLING
- HIGH
- LOW
- NONE.
- ⋮

Interrupt to see:

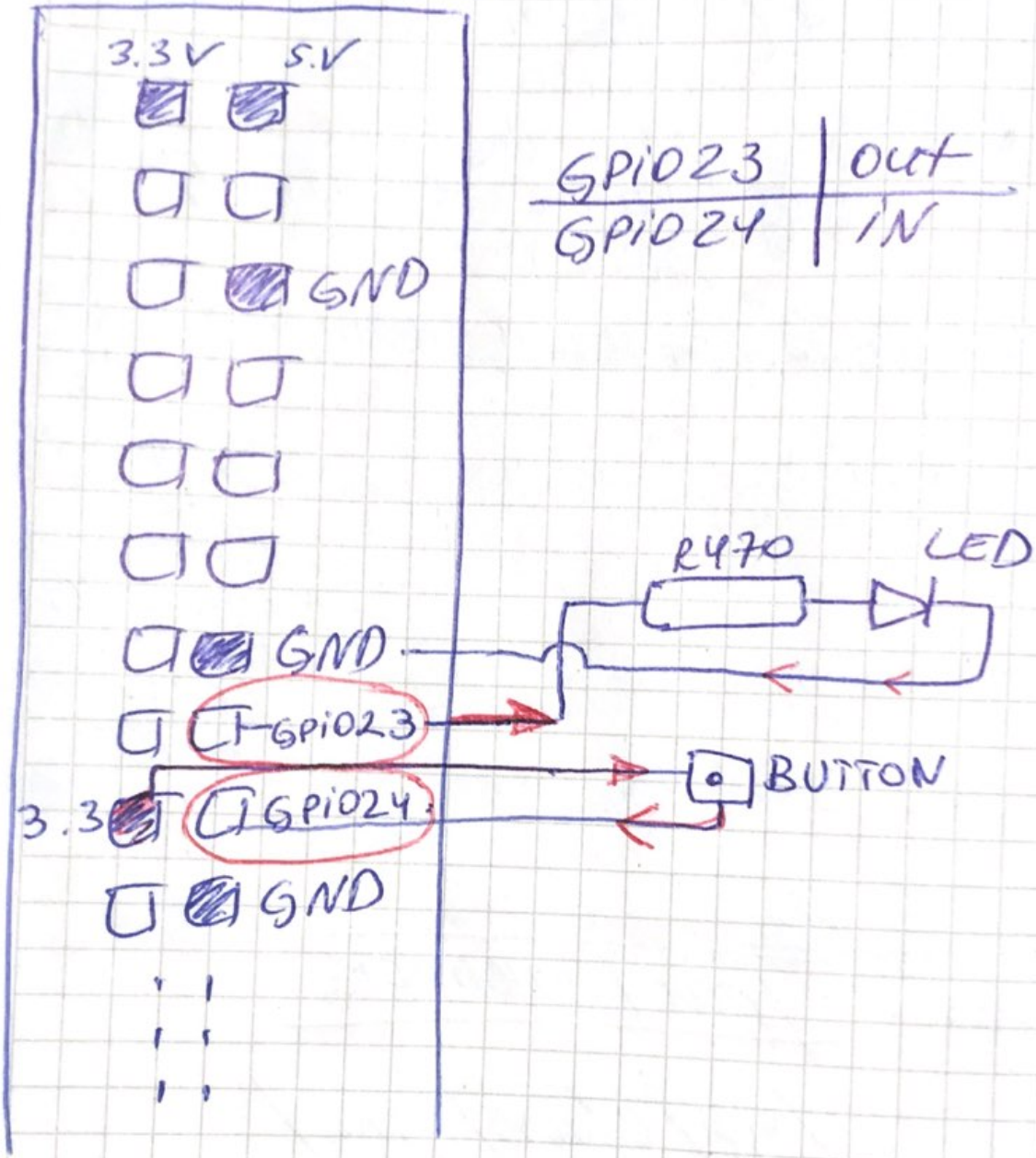
under /proc/interrupts

ls -l /sys/class/gpio/

Corresponding gpio is exported

BUTTON + LED

Schaltung:

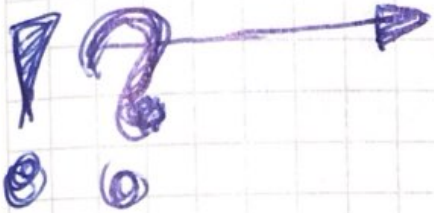


GPIO23	OUT
GPIO24	IN

→ Added Interrupt handler assoc. to pin GPIO24.

RISING-FLAG.

→ OUTPUT in KERNEL (INFO: INTERRUPT HANDLER Function called →



→ function is triggered multiple times / button press.

In the Interrupt Handler function

⇒ change state of LED
set LED to ON/OFF.

gpio_set_value(gpioLED, ~~0~~ 1)

return (irq_handler_t) IRQ_HADLED;

24.01.2021

Kernel-Space - User-Space - Interaction.

Using (sysfs) : export kernel :
- Data
- Structures
- Attributes
- Linkage

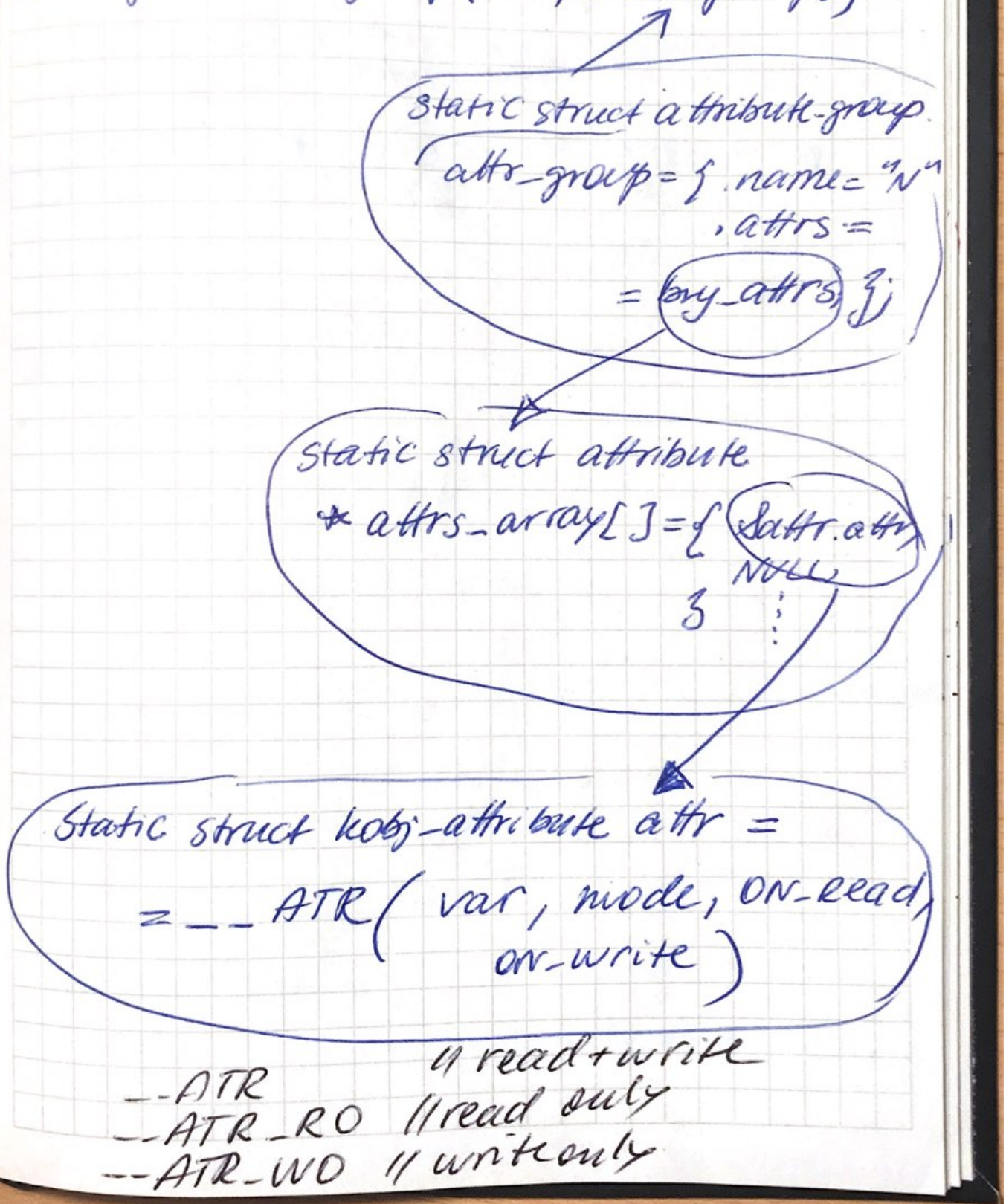
kobject Interface

kobject
- name
- ref count
- type
- sysfs represent.
- parent object

ktype
- type of the object that kobject is embedded within.

kset Group of kobjects (different types)
→ sysfs directory contains a collection of sub-directories (kobjects)

// add kobject
rko = kobject_create_and_add("...", k_kobj);
// bind a group off params
rc = sysfs_create_group(rko, &attr_group);



→ Tutorial kompiliert

→ under /sys/kernel/test_group/gpio24
found ~~parameter~~ files
attributes

→ writing to ledON → light ON/OFF
1/0

25.01.2021

TODO: BME280 over kernel

Python Script anpassen

smbus I2C : <linux/i2c.h>
<linux/i2c-dev.h>

i2c_smbus_write_byte_data(
(i2c_client),
address,
byte_count,
byff_to_write)

i2c_client = i2c_new_device(i2c_get_adapter(
1|2
(i2c_bus_number),
&(i2c_board_info));

struct i2c_board_info i2c_board_info =
{ I2C_BOARD_INFO("board", ADDR));

ADDR : 0x76.

ON OPEN:

- konfiguration HUMIDITY.
  write oversample (2) to 0xF2
- konfiguration. temp + pres + mode

write (73) (2<<5 | 2<<2 | 1) to 0xF4

ON READ :

		Address	Bytes
read	calibration 1	0x88	(24)
	calibration 2	0xA1	(1)
	calibration 3	0xE1	(7)
	data	0xF7	(8)

i2c_smbus_read_i2c_block_data(

- i2c_client,
- adress,
- bytescount,
- buffer_to_read)

write simple Test.py script to read the data from /dev/bme280

Problem: endlose schleife.

Ursache: On_read-function (fops) soll genau so viele bytes zurückgeben (byte size), wie zu Userspace kopiert wurden.

Lösung: return bytes_count;

copy_to_user(buffer, data_to_copy, data_size);

26.01.2021 Zusammenfügen alle Tests zu einem Kernel Modul.

Ziel: BUTTON → change attribute (read only to user)

Interrupt → UPDATE PERIOD → for UserSpace

LED : leuchtet in der Zeit der Ausführung read-function.

TODO: + GUI : Wheather Widget

+ Diode leuchtet bei Lesen von BME280.
→ zu kurze Leuchtzeit.
→ + wait 15ms
schedule_timeout(15);

+ neuer Attribute hinzugefügt
Period

+ Interrupt umgeschrieben

Interrupt_handler mehrmals aufgerufen pro button press.

Lösung: timeout
messen der Zeit zwischen aufrufen von Interrupt_handler.

TIME TO IGNORE default: 0,25s

28.01.2021 Python GUI

Frameworks / Libraries: Tkinter — OR
kivy — installation problems on Pi.

⊕ Created simple App, that updates the Wheather each $period, kern. attribute, set by kernel (Interrupt) Button pressed.

☆ TODO: cast kommas

⊕ Added Desktop link to start the python script.

30.01.2021

- Refactor project
- Documentation (short)

  - delete/replace "magic numbers"
  - check on_exit function to deallocall stuff correctly
  - Delete LOGGING (kernel)
  - naming conversions check.

- Upload