Writing a FreeBSD IR driver for small ARM boards using evdev interface

GANBOLD Tsagaankhuu

The FreeBSD Project,
ganbold@freebsd.org

Abstract
There are various input devices including keyboard, mouse and touchscreens exist these days. They need to have corresponding driver in operating system in order to work correctly. Input drivers in FreeBSD still have a lot of room for improvement, especially for new types of devices such as touchscreens and infrareds.
This paper describes the possible way of writing input driver in case of Consumer IR (CIR) controller using new Evdev interface recently committed by Oleksandr Tymoshenko. The paper also shows how to test CIR driver and demonstrates the use of it on small ARM boards.

1. Introduction

Nowadays most consumer electronics such as televisions, dvd players and set top boxes have remote control. This makes easier to control them. Those devices use Consumer IR or CIR in short and it usually refers to a wide variety of devices that uses infrared electromagnetic spectrum for wireless communications (1). In simpler way, Consumer IR interface is used for remote controls through infrared light (2). Lots of ARM SoC and development boards have CIR controllers these days.

1.1 Information about A10/A20 Consumer IR

Most Allwinner SoCs (A10, A13, A20, A64, A83T, H3 etc) and some other SoCs have Consumer IR controller. There are some A20 SoC based boards such as Cubieboard and BananaPI expose IR receiver that can be used. Many media players and set-top boxes based on A1x SoCs feature a built-in standard CIR infrared receiver for 38 kHz based IR controllers that has LIRC software support, which in turn is compatible with most known media center remotes made for computers and media players (3).

According to the documentation A20 Consumer IR has following features:

- Full physical layer implementation
- Support CIR for remote control or wireless keyboard
- Dual 16x8 bits FIFO for data buffer
- Programmable FIFO thresholds
- Support interrupt and DMA

The Consumer IR receiver samples the input signal on the programmable frequency and records these samples into RX FIFO when one Consumer IR signal is found on the air. The Consumer IR receiver uses Run-Length Code (RLC) to encode pulse width. The encoded data is buffered in a 64 levels and 8-bit width RX FIFO; the MSB bit is used to record the polarity of the receiving Consumer IR signal. The high level is represented as '1' and the low level is represented as '0'. The rest 7 bits are used for the length of RLC. The maximum length is 128. If the duration of one level (high or low level) is more than 128, another byte is used. Since there is always some noise in the air, a threshold can be set to filter the noise to reduce system loading and improve the system stability (2).

2. CIR driver

Based on publicly available documentation of A20 SoC and linux-sunxi BSP driver (9), I wrote FreeBSD Allwinner CIR driver (10). This driver checks "allwinner,sun4i-a10-ir" compatible string defined in corresponding device tree to detect the device in device _probe() function.
Then initialization takes place in device _attach() function:

- Gets clocks, sets rate for ir clock and enables them: apb and ir clocks
- Enables CIR mode: sets bits 4 and 5
- Sets clock sample, filter and idle thresholds
- Inverts Input Signal
- Clears all RX Interrupt Status
- Enables RX interrupt in case of overflow, packet end and FIFO available
- Enables IR Module
- Initializes evdev interface

In the interrupt handler, it handles Rx FIFO Data available and Packet end cases. It was also written decoding and validation of raw codes.

3. evdev interface

3.1 What is evdev?

It is a generic input event interface compatible with Linux evdev API at ioctl level. It allows using unmodified (apart from header name) input evdev drivers in Xorg, Wayland and Qt. It generalizes raw input events from device drivers and makes them available through character devices in the /dev/input/ directory. It is very flexible and allows to present easily multi-touch events from touchscreens, mouse buttons and so on, including IR (4).
evdev support and support for individual hardware drivers like ukbd(5) and ums(4) are committed by Oleksandr Tymoshenko. Initial project was started by Jakub Klama as part of GSoC 2014. Jakub's evdev implementation was later used as a base, updated and finished by Vladimir Kondratiev. evdev support source codes are in sys/dev/evdev.

When using evdev, each device provides one or more /dev/input/eventN nodes that a process can interact with (11). This usually means checking a few capability bits ("does this device have a left mouse button?") and reading events from the device. The events themselves are in the form of *struct input_event*, defined in *sys/dev/evdev/input.h* and consist of an event type (relative, absolute, key, ...) and an event code specific to the type (x axis, left button, etc.) and time and the value. *sys/dev/evdev/input-event-codes.h* provides a list of event codes.
A single hardware event generates multiple input events. Each input event contains the new value of a single data item (5).

Event types:
===========
Event types are groupings of codes under a logical input construct. Each type has a set of applicable codes to be used in generating events.

* EV_SYN:
 - Used as markers to separate events. Events may be separated in time or in space, such as with the multi touch protocol.

* EV_KEY:
 - Used to describe state changes of keyboards, buttons, or other key-like devices.

* EV_REL:
 - Used to describe relative axis value changes, e.g. moving the mouse 5 units to the left.

* EV_ABS:
 - Used to describe absolute axis value changes, e.g. describing the coordinates of a touch on a touchscreen.

* EV_MSC:
 - Used to describe miscellaneous input data that do not fit into other types.

* EV_SW:
 - Used to describe binary state input switches.

* EV_LED:
 - Used to turn LEDs on devices on and off.

* EV_SND:
 - Used to output sound to devices.

* EV_REP:
 - Used for auto repeating devices.

* EV_FF:
 - Used to send force feedback commands to an input device.

* EV_PWR:
 - A special type for power button and switch input.

* EV_FF_STATUS:
 - Used to receive force feedback device status.

Event codes:
===========
Event codes define the precise type of event.

In evdev, events are serialized or framed by events of type EV_SYN and code SYN_REPORT. Anything before a SYN_REPORT should be considered one logical hardware event. For instance, if you receive x and y movement within the same SYN_REPORT frame, the device has moved diagonally. So basically an event coming from the physical hardware goes into the kernel's input subsystem and is converted to an evdev event that is then available on the event node. However events and even reports can be discarded by evdev based on device capabilities and state. For instance, if several consecutive touch reports would be done with the same absolute coordinates, evdev would discard all the reports except first one.
uinput is a kernel device driver that provides the */dev/uinput* node. A process can open this node, write custom commands to it and the kernel then creates a virtual input device. That device, like all others, presents an */dev/input/eventN* node. An event written to the */dev/uinput* node will re-appear in that */dev/input/eventN* node and a device created through uinput looks just pretty much like a physical device to a process. It is possible to detect uinput-created virtual devices, but usually a process doesn't need to care, so all the common userspace (libinput, Xorg) doesn't bother. The evemu tool is one of the most commonly used applications using uinput.
However there is one thing that may cause confusion: first, to set up a uinput device, one has to use the familiar evdev type/code combinations (followed-by a couple of uinput-specific ioctls). Events written to uinput also use the *struct input_event* form, so looking at uinput code one can easily mistake it for evdev code. Nevertheless, the two serve a completely different purpose. As with evdev, libevdev can be used to initialize uinput devices. libevdev has a couple of uinput-related functions (12) that make things easier.
The Figure 1 shows how things work together using evdev and uinput. The physical devices such as mouse, keyboard, IR or touchscreen send their events through the event nodes and libinput, libevdev, lirc etc reads those events. evemu talks to uinput and creates a virtual device which then also sends events through its event node.
Figure 2 shows more detailed information about data flows from hardware to user and vice versa when using evdev. USER in picture means some userspace program or library that do open() on /dev/input/eventN character device directly or via proxy libraries like libevdev, libinput etc.

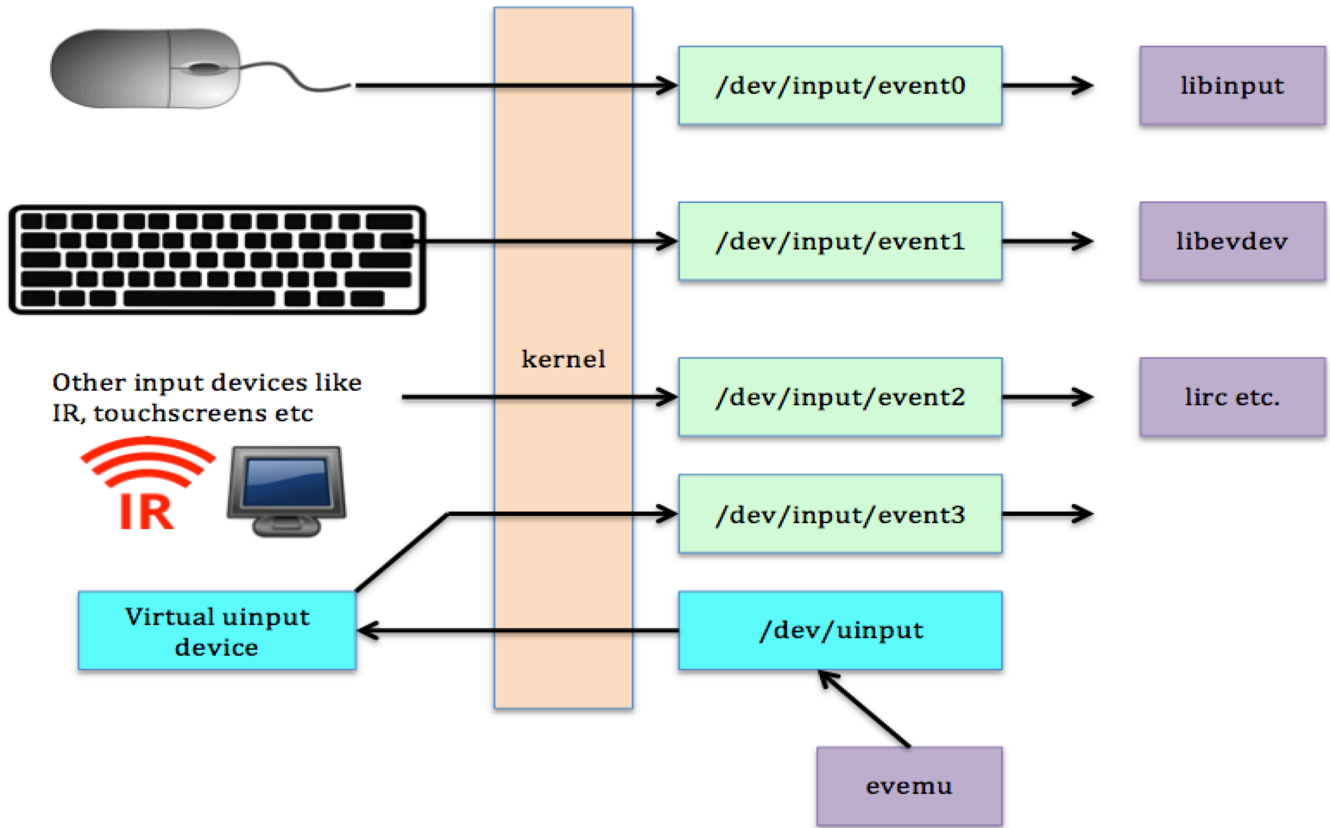

Figure 1: evdev, uinput, devices

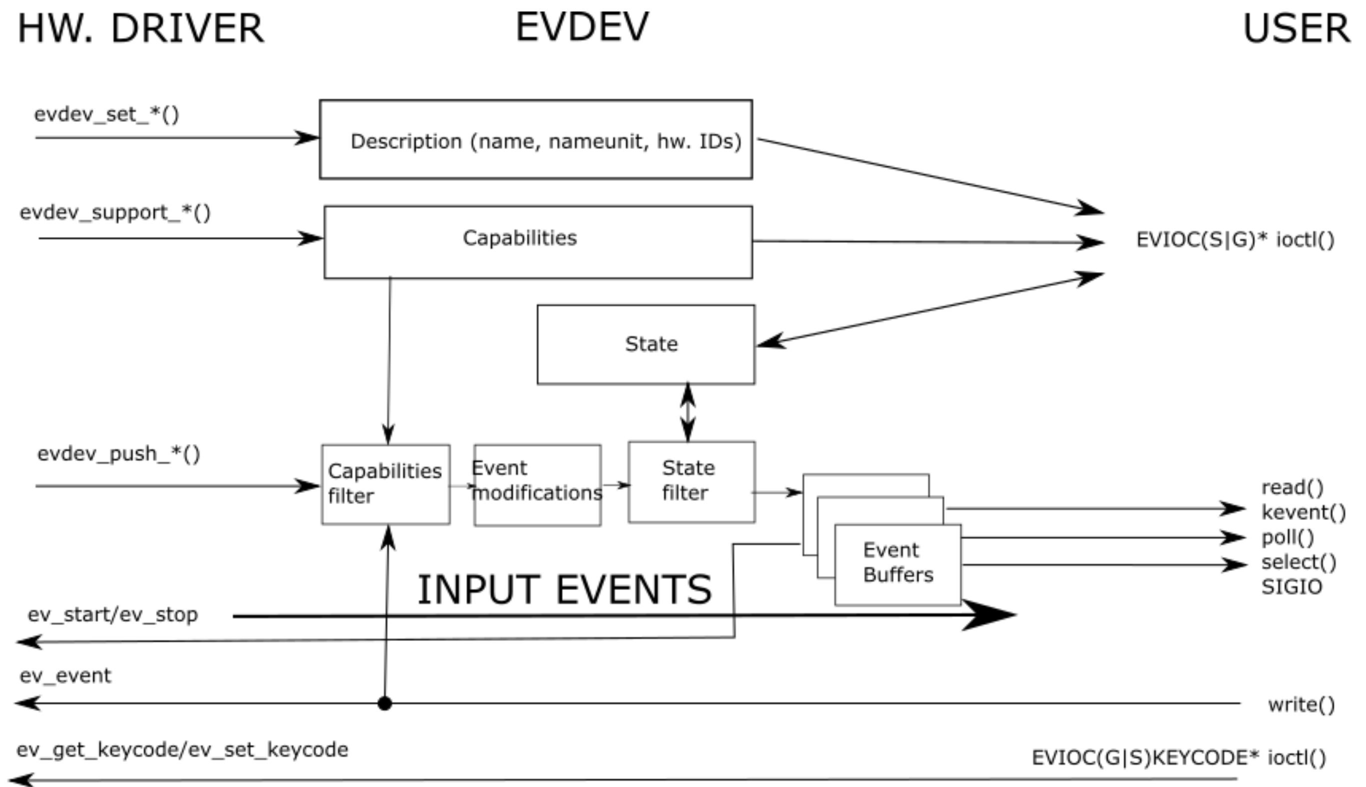

Figure 2: Data flows with evdev

3.2 Using evdev in CIR driver

It is initialized in device _attach() function in following way (10):

```
…
sc->sc_evdev = evdev_alloc();
evdev_set_name(sc->sc_evdev,
        device_get_desc(sc->dev));
evdev_set_phys(sc->sc_evdev,
        device_get_nameunit(sc->dev));
evdev_set_id(sc->sc_evdev, BUS_HOST, 0, 0, 0);
/* Support EV_SYN */
evdev_support_event(sc->sc_evdev, EV_SYN);
/* Support EV_MSC */
evdev_support_event(sc->sc_evdev, EV_MSC);
/* Support MSC_SCAN code */
evdev_support_msc(sc->sc_evdev, MSC_SCAN);
err = evdev_register(sc->sc_evdev);
...
```

Following kernel options need to be added to kernel config file in order to support evdev interface:

```
...
# EVDEV support
# input event device support
device    evdev
# evdev support in legacy drivers
options   EVDEV_SUPPORT
# install /dev/uinput cdev
device    uinput
device    aw_cir
...

# For debugging:
options   EVDEV_DEBUG  # enable event debug msgs
options   UINPUT_DEBUG # enable uinput debug msgs
```

evdev_push_event() and evdev_sync() evdev functions need to be used to send IR codes (10):

```
...
evdev_push_event(sc->sc_evdev,EV_MSC, MSC_SCAN,
        ir_code);
evdev_sync(sc->sc_evdev);
...
```

4. Testing CIR driver

Using evdev-dump:
evdev-dump is a very simple utility for dumping input event device streams.
Oleksandr Tymoshenko adapted it to FreeBSD at: https://github.com/gonzoua/evdev-dump/tree/freebsd
There is a similar tool called evtest. Compared to evdev-dump it can also list device names and dump device capabilities, but can't dump more than one device at once. It is found in many distributions as a separate package with the same name (6).

Following is the sample output when running evdev-dump:

```
# ./evdev-dump /dev/input/event0
/dev/input/event0  1480942005.263216 EV_MSC
MSC_SCAN 0xF30CBF00
/dev/input/event0  1480942005.263216 EV_SYN
SYN_REPORT 0x00000001
/dev/input/event0  1480942007.546713 EV_MSC
MSC_SCAN 0xF20DBF00
/dev/input/event0  1480942007.546713 EV_SYN
SYN_REPORT 0x00000001
/dev/input/event0  1480942008.464845 EV_MSC
MSC_SCAN 0xF10EBF00
/dev/input/event0  1480942008.464845 EV_SYN
SYN_REPORT 0x00000001
/dev/input/event0  1480942009.860569 EV_MSC
MSC_SCAN 0xEF10BF00
/dev/input/event0  1480942009.860569 EV_SYN
SYN_REPORT 0x00000001
/dev/input/event0  1480942010.675686 EV_MSC
MSC_SCAN 0xEE11BF00
/dev/input/event0  1480942010.675686 EV_SYN
SYN_REPORT 0x00000001
/dev/input/event0  1480942019.853649 EV_MSC
MSC_SCAN 0xE619BF00
/dev/input/event0  1480942019.853649 EV_SYN
SYN_REPORT 0x00000001
/dev/input/event0  1480942023.780611 EV_MSC
MSC_SCAN 0xFA05BF00
/dev/input/event0  1480942023.780611 EV_SYN
SYN_REPORT 0x00000001
/dev/input/event0  1480942024.492457 EV_MSC
MSC_SCAN 0xF708BF00
/dev/input/event0  1480942024.492457 EV_SYN
SYN_REPORT 0x00000001
/dev/input/event0  1480942025.091073 EV_MSC
MSC_SCAN 0xF906BF00
/dev/input/event0  1480942025.091073 EV_SYN
SYN_REPORT 0x00000001
```

Using ir-keytable:
ir-keytable is a tool that lists the Remote Controller devices, allows to get/set IR keycode/scancode tables, tests events generated by IR, and to adjust other Remote Controller options.

```
# ir-keytable -t -d /dev/input/eventX
```

(replace <X> by where your devinput device node is; ir-keytable(1) belongs to the multimedia/v4l-utils port (7).)

```
# ir-keytable -t -d /dev/input/event0
Testing events. Please, press CTRL-C to abort.
97472.1480942116: event type (null)(0xc21a2): scancode
= 0xf30cbf00
97440.1480942116: event type (null)(0xc21a2).
97472.1480942117: event type (null)(0xf1596): scancode
= 0xef10bf00
97440.1480942117: event type (null)(0xf1596).
97472.1480942118: event type (null)(0xb3ad3): scancode
= 0xee11bf00
97440.1480942118: event type (null)(0xb3ad3).
```

```
97472.1480942119: event type (null)(0x59255): scancode = 0xed12bf00
97440.1480942119: event type (null)(0x59255).
97472.1480942120: event type (null)(0x37720): scancode = 0xeb14bf00
97440.1480942120: event type (null)(0x37720).
97472.1480942120: event type (null)(0xa9a4f): scancode = 0xea15bf00
97440.1480942120: event type (null)(0xa9a4f).
97472.1480942121: event type (null)(0x2d69e): scancode = 0xe916bf00
97440.1480942121: event type (null)(0x2d69e).
97472.1480942122: event type (null)(0xd1772): scancode = 0xf609bf00
97440.1480942122: event type (null)(0xd1772).
97472.1480942123: event type (null)(0xc11f7): scancode = 0xf50abf00
97440.1480942123: event type (null)(0xc11f7).
97472.1480942124: event type (null)(0xa918f): scancode = 0xf609bf00
97440.1480942124: event type (null)(0xa918f).
97472.1480942125: event type (null)(0x6df63): scancode = 0xf20dbf00
97440.1480942125: event type (null)(0x6df63).
97472.1480942126: event type (null)(0xda875): scancode = 0xfa05bf00
97440.1480942126: event type (null)(0xda875).
```

5. Demonstration

Following devices were used for this demo:
- Cubieboard2 (BananaPI M1 can be used too with proper configuration changes)
- Dfrobot's simple remote controller

Here I will show some demonstration using LIRC and one of above devices to turn on and off some LEDs on the board. First we need to install lirc from either packages or ports in FreeBSD. Then we will use LIRC/irrecord to record IR key codes and use these key codes for lirc config (8). Last but not least we will use irexec to turn on and off some LEDs on the board.

There are some generic config files exist to test LIRC easily:

```
# Fetch NEC remote config
fetch http://lirc.sourceforge.net/remotes/generic/NEC.conf
```

There is need to run irrecord to record keys:

```
irrecord -H devinput -d /dev/input/event0 NEC.conf
```

Start trying with 2 keys for test. It will be stored as NEC.conf.conf.
You can check key names allowed with following command:

```
irrecord -l
```

If the key codes are like with zeros in the resulting file:

```
begin codes
   KEY_0         0x040004F30CBF00 0x00000000000000
   KEY_1         0x040004EF10BF00 0x00000000000000
end codes
```

Then edit this resulting NEC.conf.conf and manually remove the second code 0x00000000000000.
As a result it would become like:

```
begin codes
   KEY_0                 0x040004F30CBF00
   KEY_1                 0x040004EF10BF00
end codes
```

Finally, copy NEC.conf.conf to /usr/local/etc/lircd.conf.
In order to start LIRC daemon at boot, put following to /etc/rc.conf:

```
lircd_enable="YES"
lircd_flags="-H devinput"
lircd_device="/dev/input/event0"
```

Now start lircd:

```
service lircd restart
```

Then you can run irw to test what codes it receive when pressing keys in remote, something like:

```
root@allwinner:/ # irw
00040004f30cbf00 00 KEY_0 myremote
00040004ef10bf00 00 KEY_1 myremote
00040004ee11bf00 00 KEY_2 myremote
00040004ed12bf00 00 KEY_3 myremote
```

Now we can do more funny things.
We can use LIRC and CIR in A20 device to control gpio, for instance to turn on/off user green led.
You can use irexec to turn on/off user leds on boards.

Create irexec config like:

```
root@allwinner:/ # more /usr/local/etc/lirc/lircrc
begin
   button = KEY_0
   prog = irexec
   config = echo 0 > /dev/led/cubieboard2:green:usr
end
begin
   button = KEY_1
   prog = irexec
   config = echo 1 > /dev/led/cubieboard2:green:usr
end
begin
   button = KEY_2
   prog = irexec
   config = echo 0 > /dev/led/cubieboard2:blue:usr
end
```

```
begin
   button = KEY_3
   prog = irexec
   config = echo 1 > /dev/led/cubieboard2:blue:usr
end
```

Then run irexec and use remote controller to test turning on and off leds.

6. Conclusion

Using evdev interface in device drivers was easy and straightforward. There are already some drivers such as ukbd, ums, TI ADC/touchscreen and Allwinner CIR drivers are using Evdev interface.
By using evdev-dump and ir-keytable from v4l-utils port we can test evdev interface and the input driver's correctness. Once CIR driver was working correctly it was easy to use LIRC and irrecord and irexec to record IR key codes and use them in LIRC to control gpio pins or turn on/off LEDs of small ARM boards such as Cubieboard2 and BananaPI M1. That also makes these devices attractive, for instance, it is possible to use FreeBSD on these boards for home automation.
FreeBSD Allwinner CIR driver for A10/A20 can be a base for other SoC CIR controllers especially Allwinner based SoC such as H3, A64 etc. It may need some changes in order to work properly on those SoC based boards.

Acknowledgement

I would like to thank Vladimir Kondratiev for his help on using evdev interface in Allwinner CIR driver and valuable suggestions on this document, AsiaBSDCon program committee and organizers for accepting my talk and giving comments for improvement.

References

1. Consumer IR. https://en.wikipedia.org/wiki/Consumer_IR
2. A20 user manual. http://dl.linux-sunxi.org/A20/A20%20user%20manual%20v1.3%2020141010.pdf
3. LIRC. http://linux-sunxi.org/LIRC
4. SummerOfCode2014evdev_Touchscreens. https://wiki.freebsd.org/SummerOfCode2014/evdev_Touchscreens
5. Event codes. https://www.kernel.org/doc/Documentation/input/event-codes.txt
6. evdev-dump. https://github.com/gonzoua/evdev-dump/tree/freebsd
7. ir-keytable thread. https://forums.freebsd.org/threads/36620/
8. Issues related using LIRC in Cubieboard in linux. https://github.com/cubieplayer/Cubian/issues/75
9. Allwinner CIR driver. https://github.com/allwinner-zh/linux-3.4-sunxi/blob/master/drivers/input/keyboard/sunxi-ir-rx.c
10. FreeBSD Allwinner CIR driver. https://svnweb.freebsd.org/base/head/sys/arm/allwinner/aw_cir.c?revision=308638&view=markup
11. The difference between uinput and evdev. http://who-t.blogspot.com/2016/05/the-difference-between-uinput-and-evdev.html
12. libevdev uinput related functions https://www.freedesktop.org/software/libevdev/doc/latest/group__uinput.html