Writing FreeBSD IR driver for ARM boards using evdev interface

Ganbold Tsagaankhuu, Mongolian Unix User Group

AsiaBSDCon

Tokyo, 2017

About me

- FreeBSD user/sysadmin for over 18 years
- FreeBSD committer, doc – 2008, src – 2013
- Worked for Government, ISP, mobile operator
- Do remote works related to FreeBSD – system administration, consulting
- Hack ARM boards
- Football

Content

- Introduction
- Information about A10/A20 Consumer IR
- CIR driver
- evdev interface
- Using evdev in CIR driver
- Testing CIR driver
- Demonstration
- Conclusion

Introduction

- Input devices
  - keyboard
  - Mouse
  - Touchscreens etc
- Corresponding drivers
- This presentation describes writing input driver in case of Consumer IR (CIR) controller using new evdev interface

Introduction

- Remote control / Consumer IR (CIR)
  - TVs
  - DVD players
  - Set top boxes
  - Android TV sticks
  - … etc
- Refers to a wide variety of devices that uses infrared electromagnetic spectrum for wireless communications
- In simpler way, CIR is used for remote controls through infrared light
- Lots of ARM SoC and development boards have CIR controllers these days

Information about A10/A20 Consumer IR

- Most Allwinner SoCs have CIR
  - A10, A13, A20, A64, A83T, H3 etc
- A20 SoC based boards expose IR receiver that can be used
  - Cubieboard
  - BananaPI etc
- Media players and set-top boxes feature 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
- A10/A20 CIR features
  - Full physical layer implementation
  - Support CIR for remote control / wireless keyboard
  - Dual 16x8 bits FIFO for data buffer
  - Programmable FIFO thresholds
  - Interrupt and DMA support

CIR driver

- Based on
  - Public documentation of A20 SoC
  - Linux-sunxi BSP driver
  - Other drivers … but messy ...
- Detects compatible string in device tree
  - allwinner,sun4i-a10-ir

CIR driver

- Initialization in _attach()
  - Clock related initializations
    - Gets clocks -> apb, ir clocks
    - Sets rate for ir clock
    - Enables clocks -> apb, ir clocks
  - Enables CIR mode
    - Sets bits 4 and 5 in IR control register
  - Sets
    - Clock sample
    - Filter and idle thresholds

CIR driver

- Inverts Input Signal
- RX related initializations
  - Clears all RX Interrupt Status
  - Enables RX interrupt in case of
    - Overflow
    - Packet end
    - FIFO available
- Enables IR module
- Initializes evdev interface
- Interrupt handler
  - RX FIFO Data available
  - Packet end
- Decoding and validation of raw codes

evdev interface

- What is evdev?
  - Generic input event interface
  - Compatible with linux evdev API at ioctl level
  - Allows using input evdev drivers in
    - Xorg
    - Wayland
    - QT etc
  - Generalizes raw input events from device drivers
  - Makes then available through character devices in /dev/input/eventN form
  - Very flexible
  - Can easily present multi-touch events from
    - Touchscreens, mouse buttons, IR and so on

evdev interface

- evdev support
  - Started by Jakub Klama (GSoC 2014)
  - Updated and finished by Vladimir Kondratiev
  - Committed by Olexandr Tymoshenko including
    - Individual hardware drivers like
      - ukbd(5)
      - ums(4) etc
  - Source codes are in
    - sys/dev/evdev

evdev interface

- Each device provides
  - One or more /dev/input/eventN nodes that a process can interact with
    - Checks capability bits, like for instance:
      - Does this device have left mouse button?
    - Reading events from the device
- Events
  - Single hardware generates multiple input events
  - In form of struct input_event (sys/dev/evdev/input.h)
    - Event type (relative, absolute, key, etc …)
      - Groupings of codes under a logical input construct
      - Each type has a set of applicable codes to be used in generating events
    - Event code (x axis, left button etc.)
      - Defines the precise type of event
      - List of event codes are in sys/dev/evdev/input-event-code.h
    - Timestamp and the value

evdev interface

- Event types
  - 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.

evdev interface

- Event types
  - 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.

evdev interface

- Events
  - Serialized or framed by events of
    - Type EV_SYN
    - Code SYN_REPORT
      - Anything before it considered one logical hardware event
      - For instance if x, y movement -> means diagonal movement
  - So event coming from physical hardware goes to
    - Kernel's input subsystem
    - Converted to an evdev event
    - Then available on the event node
  - However events and reports can be discarded based on device
    - Capabilities
    - State
      - For instance consecutive events with same coordinates – evdev discard all except first one

evdev interface

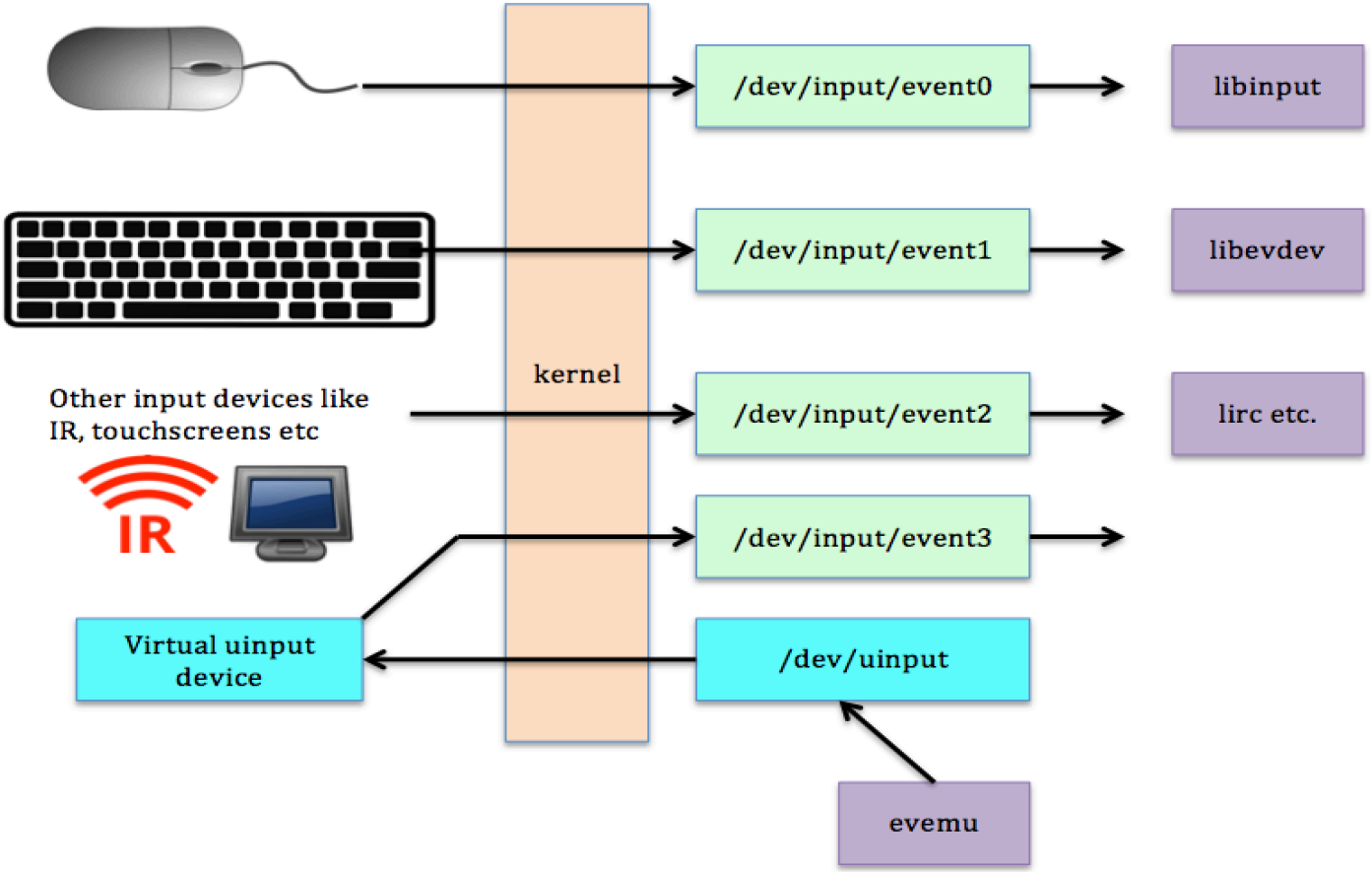
- uinput - kernel device driver that provides
  - /dev/uinput node
    - Process can interact with it writing commands to it etc
    - Kernel then creates virtual input device
      - /dev/input/eventN
    - Event written to /dev/uinput node will appear in
      - /dev/input/eventN
  - uinput device looks like physical device to a process
  - Common application that uses uinput
    - evemu tool

evdev interface

- To setup uinput device
  - evdev type/code combinations
  - uinput specific ioctls
  - Uses same event form
    - struct input_event
- libevdev can be used to initialize uinput devices
  - Has some uinput related functions

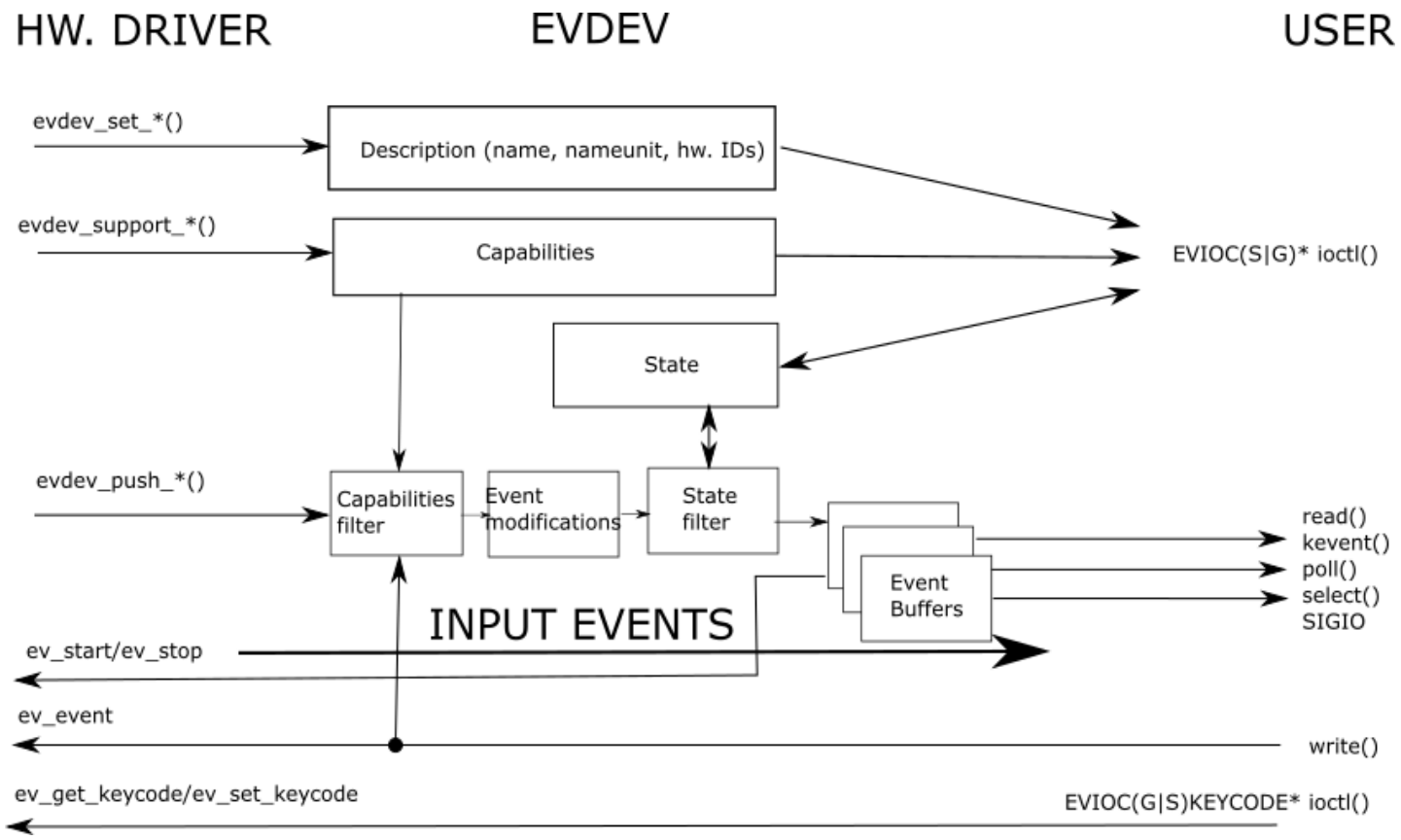
evdev interface

- evdev, uinput, devices



evdev interface

- Data flows with evdev



Using evdev in CIR driver

```
…
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);
...
```

Using evdev in CIR driver

- 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
...
```

Using evdev in CIR driver

- 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:

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

Testing CIR driver

- evdev-dump
  - Simple utility for dumping input event device streams.
  - Oleksandr Tymoshenko adapted it to FreeBSD at:
    - https://github.com/gonzoua/evdev-dump/tree/freebsd

Testing CIR driver

- evdev-dump sample output

```
# ./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
…
```

Testing CIR driver

- ir-keytable
  - Belongs to multimedia/v4l-utils port
  - 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/event0

Testing CIR driver

- ir-keytable sample output

```
# 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).
…
```

Demonstration

- Devices used
  - Cubieboard2 (BananaPI M1 can be used with some changes)
  - Dfrobot's simple remote controller (others can be used)
- LIRC
  - irrecord
  - irexec

Demonstration

- 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

Demonstration

- 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
```

Demonstration

- Copy NEC.conf.conf to /usr/local/etc/lircd.conf.
- /etc/rc.conf needs LIRC related changes:

  lircd_enable="YES"

  lircd_flags="-H devinput"

  lircd_device="/dev/input/event0”
- After starting lirc, irw tool can be used to check

  root@allwinner:/ # irw

  00040004f30cbf00 00 KEY_0 myremote

  00040004ef10bf00 00 KEY_1 myremote

  00040004ee11bf00 00 KEY_2 myremote

  00040004ed12bf00 00 KEY_3 myremote

Demonstration

- Create irexec config like following in /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
```

Conclusion

- Using evdev interface in driver
  - Easy and straightforward
  - Sample drivers exist
    - ukbd, ums, TI ADC/touchscreen etc
- Testing is easy
  - evdev-dump
  - ir-keytable (v4l-utils)
- Can control other things via IR like using
  - LIRC to control GPIO, leds etc
    - irrecord/irexec
- FreeBSD Allwinner CIR driver
  - Can be a base of other SoC CIR controllers
  - May need some changes to support newer Allwinner SoCs

Thank you for your attention

Questions?

ganbold@freebsd.org