iioinput

Introduction to IIO and Input Drivers

Matt Porter <mporter@konsulko.com>

e-ale

© CC-BY SA4

© CC-BY SA4

The E-ALE (Embedded Apprentice Linux Engineer) is a series of seminars held at existing conferences covering topics which are fundamental to a Linux professional in the field of Embedded Linux.

This seminar will spend equal time on lecture and hands on labs at the end of each seminar which allow you to practice the material you've learned.

> This material makes the assumption that you have minimal experience with using Linux in general, and a basic understanding of general industry terms. The assumption is also made that you have access to your own computers upon which to practice this material.

More information can be found at **https://e-ale.org/**

This material is licensed under **CC-BY SA4**

Contents

 © CC-BY SA4

Chapter 1

Preliminaries

e-ale

1.1 Introductions

About Me

- CTO at Konsulko Group
- Using Linux since 1992
- Professional embedded Linux engineer since 1998
- Previously maintained embedded PPC platforms, RapidIO subsystem, and Broadcom Mobile SoCs in the kernel
- Various small contributions around the kernel

© CC-BY SA4

About Konsulko Group

- Konsulko Group is a services company founded by embedded Linux veterans
- Community and commercial embedded, Linux, and Open Source Software development
- See https://www.konsulko.com for more information
- Linux Foundation training partners
  - Use code **Konsulko.10.ATP.kr** for a 10% discount on any Linux Foundation training course.

© CC-BY SA4

1.2 Project Plan

What To Do?

- Let's build a joystick driver for Linux
- BaconBits MiniCape has a thumbwheel and a spare button we can use
- We'll make a single axis joystick with one button (a paddle-style controller)

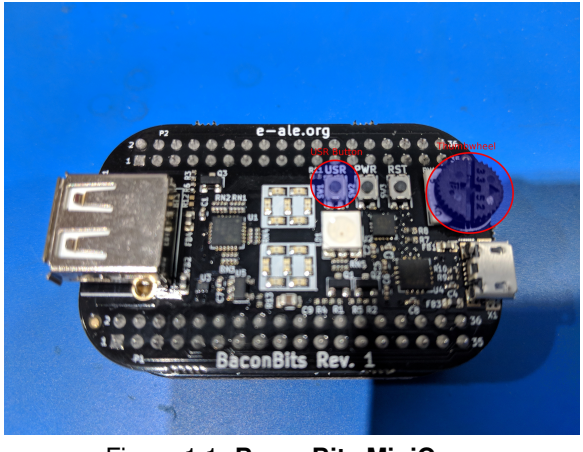


Figure 1.1: **BaconBits MiniCape**

© CC-BY SA4

Exact Steps

- Understand how the thumbwheel and button are interfaced in hardware
- Understand how to make the paddle controller usable by any application that understands Linux joystick or input APIs
- Write Device Tree description of the paddle controller
- Write a kernel driver for the paddle controller
- Test the paddle controller

© CC-BY SA4

Chapter 2

Hardware

e-ale

2.1 BaconBits Hardware

Component Placement

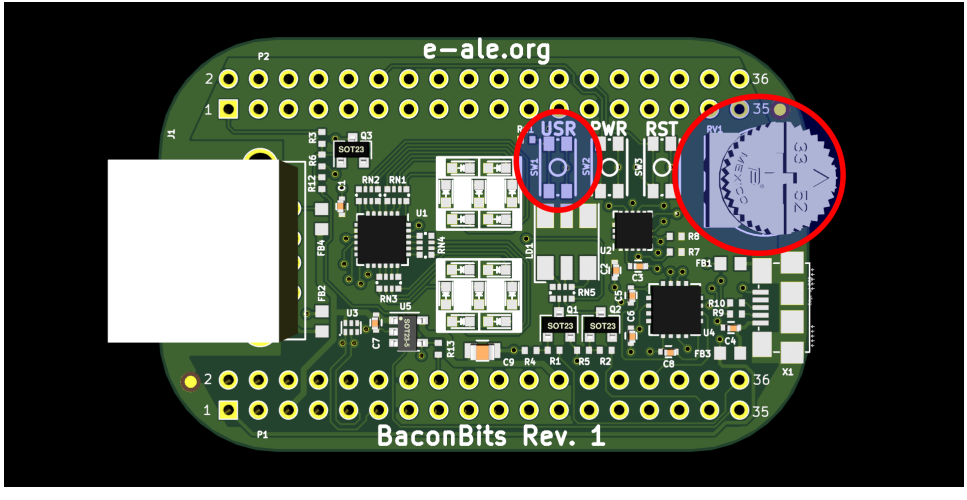


Figure 2.1: **BaconBits Component Identification**

- **RV1** is the thumbwheel device
- **SW1** is the user button

© CC-BY SA4

BaconBits Schematic Overview

- https://github.com/e-ale/BaconBitsCapeHW/blob/master/baconbits.pdf

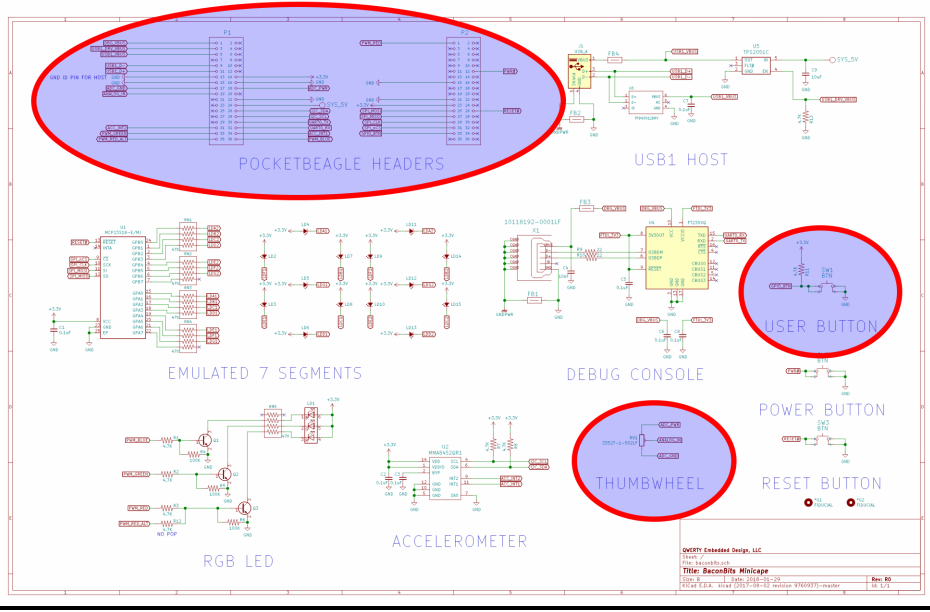


Figure 2.2: **BaconBits Schematic**

© CC-BY SA4

BaconBits Thumbwheel

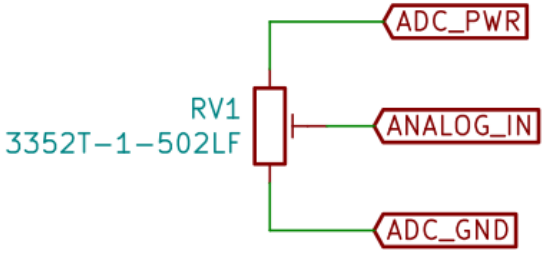


Figure 2.3: **BaconBits Thumbwheel**

- Signals:
  - **ADC_GND**
  - **ADC_PWR**
  - **ANALOG_IN**

© CC-BY SA4

BaconBits P1 Connector

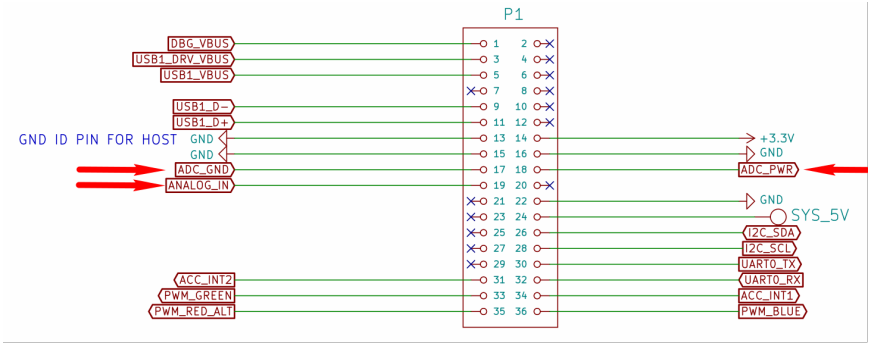


Figure 2.4: **BaconBits P1 Connector**

- Pins:
  - **ADC_GND : P1-17**
  - **ADC_PWR : P1-18**
  - **ANALOG_IN : P1-19**

© CC-BY SA4

BaconBits User Button

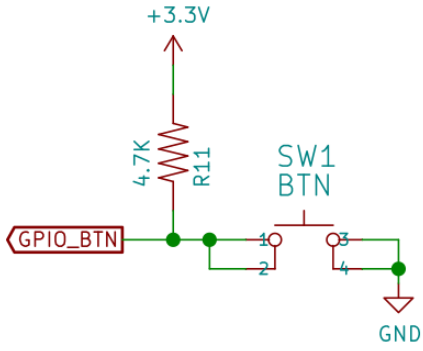


Figure 2.5: **BaconBits User Button**

- Signals:
  - **GPIO_BTN**
- Note that **GPIO_BTN** has a pull-up resistor indicating that it is active low.

© CC-BY SA4

BaconBits P2 Connector

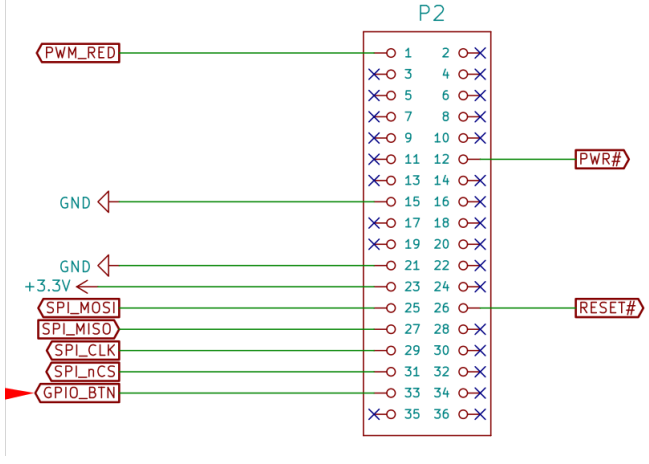


Figure 2.6: **BaconBits P2 Connector**

- Pins:
  - **GPIO_BTN : P2-33**

© CC-BY SA4

2.2 PocketBeagle Hardware

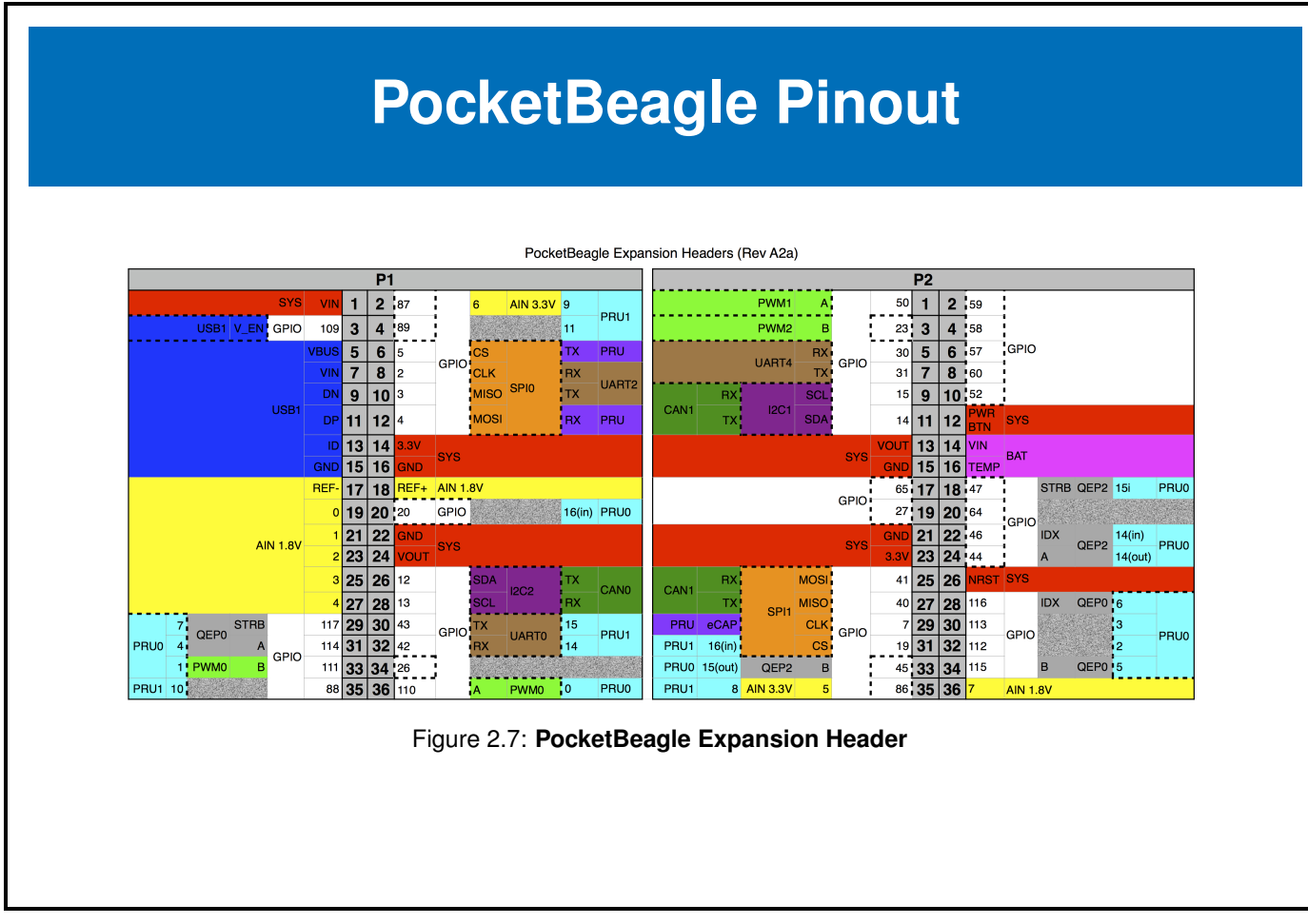


Figure 2.7: **PocketBeagle Expansion Header**

© CC-BY SA4

PocketBeagle Schematic Headers

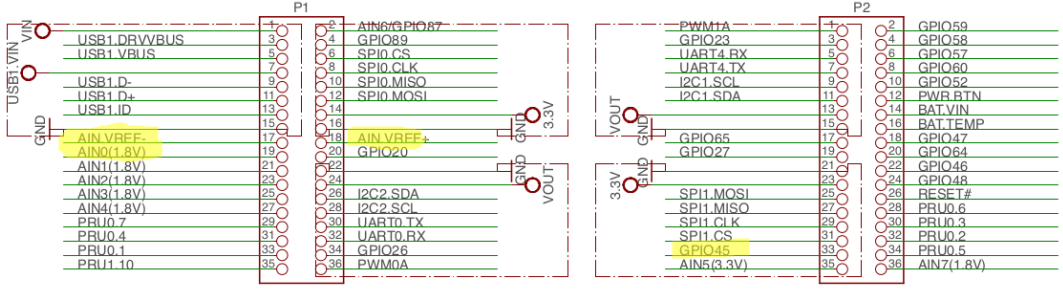


Figure 2.8: **PocketBeagle Schematic Headers**

© CC-BY SA4

PocketBeagle Schematic Analog

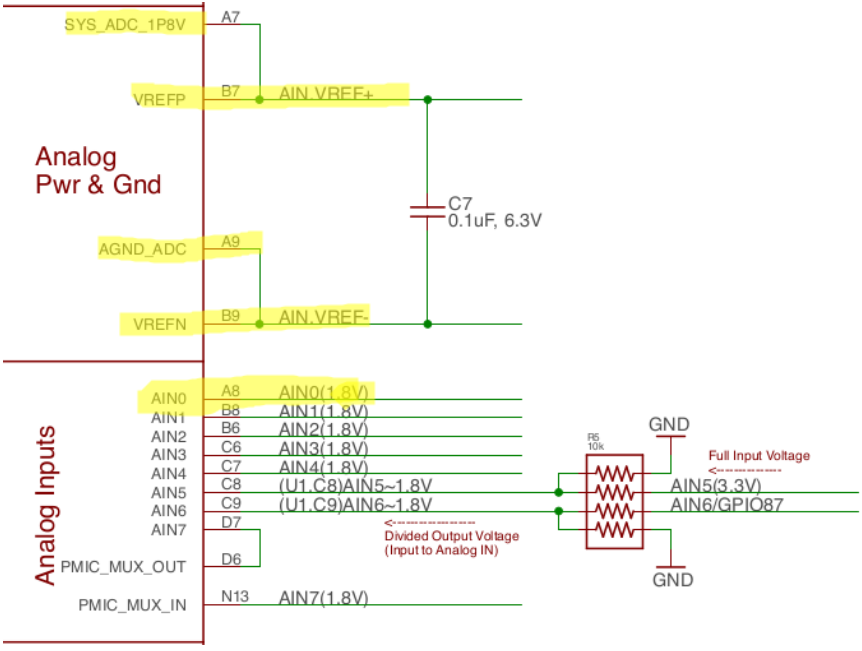


Figure 2.9: **PocketBeagle Schematic Analog**

© CC-BY SA4

PocketBeagle Schematic GPIO

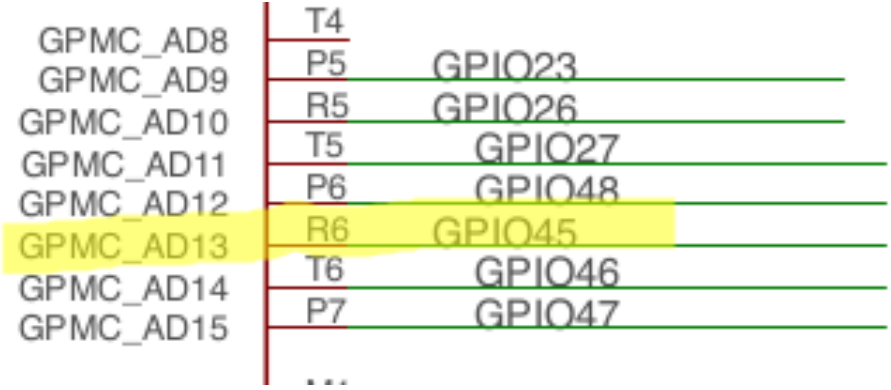


Figure 2.10: **PocketBeagle Schematic GPIO**

© CC-BY SA4

2.3 OSD335x and AM335x Hardware

OSD335x Pin Map

- https://octavosystems.com/docs/osd335x-c-sip-datasheet/

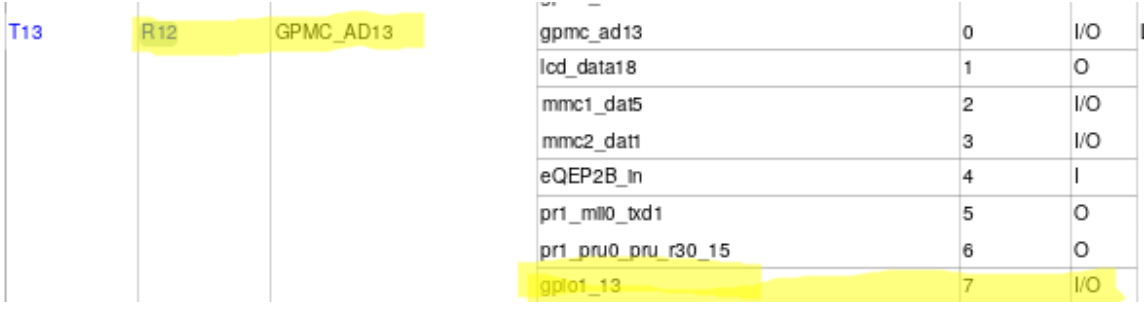
GPMC_AD8	GPMC Address and Data	A17	U10
GPMC_AD9	GPMC Address and Data	C16	T10
GPMC_AD10	GPMC Address and Data	B16	T11
GPMC_AD11	GPMC Address and Data	A16	U12
GPMC_AD12	GPMC Address and Data	C15	T12
GPMC_AD13	GPMC Address and Data	B15	R12
GPMC_AD14	GPMC Address and Data	A15	V13
GPMC_AD15	GPMC Address and Data	C14	U13

Figure 2.11: **OSD335x Pin Map**

© CC-BY SA4

AM335x ZCZ Pins

- http://www.ti.com/lit/ds/symlink/am3352.pdf



T13	R12	GPMC_AD13	gpmc_ad13	0	I/O	I
			lcd_data18	1	O	
			mmc1_dat5	2	I/O	
			mmc2_dat1	3	I/O	
			eQEP2B_in	4	I	
			pr1_mii0_txd1	5	O	
			pr1_pru0_pru_r30_15	6	O	
			gpio1_13	7	I/O	

Figure 2.12: **AM335x ZCZ Pins**

© CC-BY SA4

2.4 Summary

Hardware Investigation Results

- Thumbwheel:
  - Connected to analog input 0 (**AIN0**)
- User Button:
  - Connected to **GPMC_AD13** which can be muxed as **GPIO1_13**
  - Active low

© CC-BY SA4

Chapter 3

Kernel Subsystems

e-ale

3.1 Input Subsystem

Overview

- The Linux Input subsystem is a framework to support all types of input devices
- https://www.kernel.org/doc/html/v4.18/input/input.html
- Consists of the core **input module**, **device drivers**, and **event handlers**

© CC-BY SA4

Device Drivers

- **Device drivers** interface with hardware and provide events to the **input module**
- Examples are:
  - **gpio_keys**
  - **hid-generic**
  - **usbmouse**

© CC-BY SA4

Event Handlers

- **Event handlers** interface with the **input module** and pass events to other kernel subsystems or userspace
  - **evdev** passes generic input events to userspace. Devices are in **/dev/input**:

```
crw-r--r--   1 root     root      13,  64 Apr  1 10:49 event0
crw-r--r--   1 root     root      13,  65 Apr  1 10:50 event1
crw-r--r--   1 root     root      13,  66 Apr  1 10:50 event2
crw-r--r--   1 root     root      13,  67 Apr  1 10:50 event3
```

  - **joydev** passes joystick events to userspace. Devices are in **/dev/input**:

```
crw-r--r--   1 root     root      13,   0 Apr  1 10:50 js0
crw-r--r--   1 root     root      13,   1 Apr  1 10:50 js1
crw-r--r--   1 root     root      13,   2 Apr  1 10:50 js2
crw-r--r--   1 root     root      13,   3 Apr  1 10:50 js3
```


© CC-BY SA4

evdev

- **evdev** nodes support blocking/non-blocking **read** and **select**
- Reading an **evdev** node returns a **struct input_event**:

```
struct input_event
{
        struct timeval time;
        unsigned short type;
        unsigned short code;
        unsigned int value;
};
```


© CC-BY SA4

evtest

- **evtest** can be used to test evdev events at the command line
- Example:

```
$ evtest /dev/input/event0
Input driver version is 1.0.1
Input device ID: bus 0x3 vendor 0x46d product 0x1028 version 0x111
Input device name: "Logitech M570"
Supported events:
Event type 0 (EV_SYN)
Event type 1 (EV_KEY)
Event code 272 (BTN_LEFT)
...
Event type 4 (EV_MSC)
Event code 4 (MSC_SCAN)
Properties:
Testing ... (interrupt to exit)
Event: time 1540159516.312712, type 4 (EV_MSC), code 4 (MSC_SCAN), value 90001
Event: time 1540159516.312712, type 1 (EV_KEY), code 272 (BTN_LEFT), value 1
Event: time 1540159516.312712, ------------- SYN_REPORT ------------
```


© CC-BY SA4

Input device driver API

- Input devices described by **struct input_dev**:

```
struct input_dev {
        const char *name;
        ...
        unsigned long evbit[BITS_TO_LONGS(EV_CNT)];
        unsigned long keybit[BITS_TO_LONGS(KEY_CNT)];
        ...
        int (*event)(struct input_dev *dev, unsigned int type, unsigned int code, int value);
        ...
};
```

- A **struct input_dev** is allocated using **devm_input_allocate_device** before event types and codes are configured and handlers are filled in.
- **input_register_device** and **input_unregister_device** are used to register and unregister the device, respectively

© CC-BY SA4

Input polled device driver API

- Simple devices that can to be polled on a timer basis can be implemented using the simpler **struct input_polled_dev**:

```
struct input_polled_dev {
        ...
        void (*poll)(struct input_polled_dev *dev);
        unsigned int poll_interval; /* msec */
        ...
        struct input_dev *input;
};
```

- Allocate a **struct input_polled_dev** with **devm_input_allocate_polled_device**
- It is necessary only to fill in the **poll** handler, **poll_interval**, and fill in the **input** device configuration (**name, id, evkey, keybit** fields and use **input_set_abs_params** if absolute events are supported)
- Register the polled device with **input_register_polled_device**
- In the **poll** handler, read the hardware and use **input_report_*** to queue events to be reported and **input_sync** to flush the queued events

© CC-BY SA4

3.2 IIO Subsystem

Overview

- The Linux IIO subsystem is a framework to support sensors and any type of device with ADCs or DACs
- https://www.kernel.org/doc/html/v4.18/driver-api/iio/intro.html
- The IIO core supports a **struct iio_dev** to represent an IIO device which may contain any number of channels
- IIO provides a userspace interface via sysfs with a hierachy under **/sys/bus/iio/devices/iio:deviceX**
- IIO also provides an interface for in-kernel users of IIO devices

© CC-BY SA4

Consumer API

- Used by other kernel drivers to build functionality on top of an IIO hardware device driver
- https://elixir.bootlin.com/linux/latest/source/include/linux/iio/consumer.h
- Get an IIO device channel

```
struct iio_channel *devm_iio_channel_get_all(struct device *dev);
```

- Get a channel type from a device channel

```
int iio_get_channel_type(struct iio_channel *channel,
                         enum iio_chan_type *type);
```

- Read a raw (unprocessed) ADC value from a device channel

```
int iio_read_channel_raw(struct iio_channel *chan,
                         int *val);
```


© CC-BY SA4

3.3 GPIO Subsystem

Overview

- The Linux GPIO subsystem is a framework to support control of General Purpose Input/Output pins
- https://www.kernel.org/doc/Documentation/gpio/gpio.txt
- No longer using the legacy GPIO APIs.
  - https://www.kernel.org/doc/Documentation/gpio/gpio-legacy.txt
- Prefer the descriptor-based consumer GPIO APIs.
  - https://www.kernel.org/doc/Documentation/gpio/consumer.txt

© CC-BY SA4

Consumer

- Get a GPIO descriptor

```
struct gpio_desc *gpiod_get(struct device *dev, const char *con_id,
                            enum gpiod_flags flags)
```

  - **con_id** is typically the prefix of a **Device Tree gpio(s)** property. e.g. a **power-gpio** property would require **power** for **con_id**
    - https://www.kernel.org/doc/Documentation/gpio/board.txt
  - **flags** are optional and can include direction and/or initial value for a GPIO. e.g. **GPIOD_IN** for an input

- Get a GPIO value (0 for low, nonzero for high)

```
int gpiod_get_value(const struct gpio_desc *desc);
```


© CC-BY SA4

Chapter 4

Device Tree

e-ale

4.1 Device Tree

What is Needed?

- Mux the **GPMC_AD13** pin as **GPIO1_13**
- Create a paddle device with a compatible string
- Link to the GPIO pinmux node
- Link to ADC channel 0 for the thumbwheel
- Link to **GPIO1_13** for the button

© CC-BY SA4

AM335x GPIO1_13 Pin Mux Register

- Note that **GPIO1_13** is at offset **0x834**
- https://www.ti.com/lit/ug/spruh73p/spruh73p.pdf

800h	conf_gpmc_ad0	See the device datasheet for information on default pin mux configurations. Note that the device ROM may change the default pin mux for certain pins based on the SYSBOOT mode settings.	Section 9.3.1.50
804h	conf_gpmc_ad1		Section 9.3.1.50
808h	conf_gpmc_ad2		Section 9.3.1.50
80Ch	conf_gpmc_ad3		Section 9.3.1.50
810h	conf_gpmc_ad4		Section 9.3.1.50
814h	conf_gpmc_ad5		Section 9.3.1.50
818h	conf_gpmc_ad6		Section 9.3.1.50
81Ch	conf_gpmc_ad7		Section 9.3.1.50
820h	conf_gpmc_ad8		Section 9.3.1.50
824h	conf_gpmc_ad9		Section 9.3.1.50
828h	conf_gpmc_ad10		Section 9.3.1.50
82Ch	conf_gpmc_ad11		Section 9.3.1.50
830h	conf_gpmc_ad12		Section 9.3.1.50
834h	conf_gpmc_ad13		Section 9.3.1.50
838h	conf_gpmc_ad14		Section 9.3.1.50
83Ch	conf_gpmc_ad15		Section 9.3.1.50
840h	conf_gpmc_a0		Section 9.3.1.50

Figure 4.1: **AM335x Pin Mux Registers**

© CC-BY SA4

Implementation

- User button GPIO pinmux configuration:

```
gpio1_13_pin: pinmux-gpio1-13-pin {
        pinctrl-single,pins = <
                AM33XX_IOPAD(0x0834, PIN_INPUT | MUX_MODE7)
        >;
};
```

- Paddle device node:

```
paddle {
        compatible = "e-ale,baconbits-paddle";
        pinctrl-0 = <&gpio1_13_pin>;
        io-channels = <&am335x_adc 0>;
        io-channel-names = "thumbwheel";
        button-gpios = <&gpio1 13 GPIO_ACTIVE_LOW>;
};
```


© CC-BY SA4

Deploying

- Already pre-patched into the am335x_pocketbeagle.dts and is therefore present in the am335x_pocketbeagle.dtb used to boot the standard course kernel image.
- No action required here!

© CC-BY SA4

Overlay

- An advanced user may choose instead to apply DT changes via an overlay as detailed in the U-Boot tutorial. The following is an example of the same data in overlay format:

```
/dts-v1/;
/plugin/;

#include <dt-bindings/gpio/gpio.h>
#include <dt-bindings/pinctrl/omap.h>

/ {
        fragment@0 {
                target = <&am33xx_pinmux>;
                __overlay__ {
                        gpio1_13_pin: pinmux-gpio1-13-pin {
                                pinctrl-single,pins = <
                                        AM33XX_IOPAD(0x0834, PIN_INPUT | MUX_MODE7)
                                >;
                        };
                };
        };

        fragment@1 {
                target = <&l4_wkup>;
                __overlay__ {
                        paddle {
                                compatible = "e-ale,baconbits-paddle";
                                pinctrl-0 = <&gpio1_13_pin>;
                                io-channels = <&am335x_adc 0>;
                                io-channel-names = "thumbwheel";
                                button-gpios = <&gpio1 13 GPIO_ACTIVE_LOW>;
                        };
                };
        };
};
```


© CC-BY SA4

DT Binding References

- https://www.kernel.org/doc/Documentation/devicetree/bindings/gpio/gpio.txt
- https://www.kernel.org/doc/Documentation/devicetree/bindings/iio/iio-bindings.txt
- https://www.kernel.org/doc/Documentation/devicetree/bindings/pinctrl/pinctrl-bindings.txt

© CC-BY SA4

Chapter 5

Driver

e-ale

5.1 Driver

What is needed?

- Platform driver matching on DT compatible string
- Get the IIO ADC device channel corresponding to the thumbwheel
- Get the GPIO corresponding to the user button
- Register a polled input device
- Read current values from ADC and GPIO
- Report the input events

© CC-BY SA4

Implementation: Skeleton

```
#include <linux/gpio/consumer.h>
#include <linux/iio/consumer.h>
#include <linux/iio/types.h>
#include <linux/input.h>
#include <linux/input-polldev.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/of.h>
#include <linux/platform_device.h>

...

#ifdef CONFIG_OF
static const struct of_device_id paddle_of_match[] = {
        { .compatible = "e-ale,baconbits-paddle", },
        { }
};
MODULE_DEVICE_TABLE(of, paddle_of_match);
#endif

static struct platform_driver __refdata paddle_driver = {
        .driver = {
                .name = "paddle",
                .of_match_table = of_match_ptr(paddle_of_match),
        },
        .probe = paddle_probe,
        .remove = paddle_remove,
};
module_platform_driver(paddle_driver);

MODULE_AUTHOR("Matt Porter");
MODULE_DESCRIPTION("BaconBits paddle game controller input driver");
MODULE_LICENSE("GPL v2");
```


© CC-BY SA4

Implemention: probe() 1/1

```
struct paddle {
        struct input_polled_dev *poll_dev;
        char phys[32];
        struct gpio_desc *button;
        struct iio_channel *channel;
};

...

static int paddle_probe(struct platform_device *pdev)
{
        struct device *dev = &pdev->dev;
        struct paddle *p;
        struct input_polled_dev *poll_dev;
        struct input_dev *input;
        enum iio_chan_type type;
        int ret;

        p = devm_kzalloc(dev, sizeof(*p), GFP_KERNEL);
        if (!p)
                return -ENOMEM;

        p->button = devm_gpiod_get(dev, "button", GPIOD_IN);
        if (IS_ERR(p->button)) {
                ret = PTR_ERR(p->button);
                dev_err(dev, "failed to get button GPIO: %d\n", ret);
                return ret;
        }
```


© CC-BY SA4

Implementation: probe() 2/3

```
p->channel = devm_iio_channel_get(dev, "thumbwheel");
if (IS_ERR(p->channel))
        return PTR_ERR(p->channel);

if (!p->channel->indio_dev)
        return -ENXIO;

ret = iio_get_channel_type(p->channel, &type);
if (ret < 0)
        return ret;

if (type != IIO_VOLTAGE) {
        dev_err(dev, "not voltage channel %d\n", type);
        return -EINVAL;
}

poll_dev = devm_input_allocate_polled_device(dev);
if (!poll_dev) {
        dev_err(dev, "unable to allocate input device\n");
        return -ENOMEM;
}

poll_dev->poll_interval = 50;
poll_dev->poll = paddle_poll;
poll_dev->private = p;

p->poll_dev = poll_dev;
platform_set_drvdata(pdev, p);
```


© CC-BY SA4

Implementation: probe() 3/3

```
        input = poll_dev->input;
        input->name = pdev->name;
        sprintf(p->phys, "paddle/%s", input->dev.kobj.name);
        input->phys = p->phys;
        input->id.bustype = BUS_HOST;

        __set_bit(EV_KEY, input->evbit);
        __set_bit(BTN_A, input->keybit);
        __set_bit(EV_ABS, input->evbit);
        /* Hardcode min/max to the resolution of the 12-bit TSCADC */
        input_set_abs_params(input, ABS_X, 0, 4095, 0, 0);

        ret = input_register_polled_device(poll_dev);
        if (ret) {
                dev_err(dev, "unable to register input device: %d\n", ret);
                return ret;
        };

        return 0;
}
```


© CC-BY SA4

Implementation: remove()

```
static int paddle_remove(struct platform_device *pdev)
{
        struct paddle *p = platform_get_drvdata(pdev);

        input_unregister_polled_device(p->poll_dev);

        return 0;
}
```


© CC-BY SA4

Implementation: poll()

```
static void paddle_poll(struct input_polled_dev *dev)
{
        struct paddle *p = dev->private;
        int ret, a_val, x_val;

        a_val = gpiod_get_value(p->button);
        input_report_key(dev->input, BTN_A, a_val);

        ret = iio_read_channel_raw(p->channel, &x_val);
        if (unlikely(ret < 0))
                return;
        input_report_abs(dev->input, ABS_X, x_val);

        input_sync(dev->input);
}
```


© CC-BY SA4

Implementation: Makefile

```
obj-m := paddle.o
```


© CC-BY SA4

Build

- Example command line for out-of-tree kernel module:

```
$ export PATH=~/e-ale/gcc-linaro-7.3.1-2018.05-x86_64_arm-linux-gnueabihf/bin:$PATH
$ ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- make -C ~/e-ale/linux-kernel M=$PWD
make: Entering directory '/home/mporter/src/e-ale/linux-kernel'
  Building modules, stage 2.
  MODPOST 1 modules
  make: Leaving directory '/home/mporter/src/e-ale/linux-kernel'
$ls -l paddle.ko
  -rw-r--r-- 1 mporter mporter 120352 Oct 21 18:34 paddle.ko
```


© CC-BY SA4

Deploy

- Copy **paddle.ko** module to the rootfs. Example:

```
$ sudo mount /dev/sdb2 /mnt/tmp
$ sudo cp paddle.ko /mnt/tmp/home/root/
$ sudo umount /mnt/tmp
```

- Install the module:

```
root@pocketbeagle:~# insmod paddle.ko
paddle: loading out-of-tree module taints kernel.
input: paddle as /devices/platform/paddle/input/input0
```


© CC-BY SA4

Test: Device Nodes

```
root@pocketbeagle:~# ls -l /dev/input
total 0
drwxr-xr-x 2 root root      60 Jan  1 00:00 by-path
crw-rw---- 1 root input 13, 64 Jan  1 00:00 event0
crw-rw-r-- 1 root input 13,  0 Jan  1 00:00 js0
root@pocketbeagle:~#
```


© CC-BY SA4

Test: evtest

```
root@pocketbeagle:~# evtest /dev/input/event0
Input driver version is 1.0.1
Input device ID: bus 0x19 vendor 0x0 product 0x0 version 0x0
Input device name: "paddle"
Supported events:
Event type 0 (EV_SYN)
Event type 1 (EV_KEY)
Event code 304 (BTN_SOUTH)
Event type 3 (EV_ABS)
Event code 0 (ABS_X)
Value   4095
Min        0
Max     4095
Properties:
Testing ... (interrupt to exit)
Event: time 946685042.637203, type 1 (EV_KEY), code 304 (BTN_SOUTH), value 1
Event: time 946685042.637203, -------------- SYN_REPORT ------------
Event: time 946685042.817200, type 1 (EV_KEY), code 304 (BTN_SOUTH), value 0
Event: time 946685042.817200, -------------- SYN_REPORT ------------
Event: time 946685045.217197, type 3 (EV_ABS), code 0 (ABS_X), value 4064
Event: time 946685045.217197, -------------- SYN_REPORT ------------
Event: time 946685045.277201, type 3 (EV_ABS), code 0 (ABS_X), value 4031
Event: time 946685045.277201, -------------- SYN_REPORT ------------
Event: time 946685045.337199, type 3 (EV_ABS), code 0 (ABS_X), value 3979
Event: time 946685045.337199, -------------- SYN_REPORT ------------
Event: time 946685045.397199, type 3 (EV_ABS), code 0 (ABS_X), value 3906
Event: time 946685045.397199, -------------- SYN_REPORT ------------
```


© CC-BY SA4

Test: jstest

```
root@pocketbeagle:~# jstest /dev/input/js0
Driver version is 2.1.0.
Joystick (paddle) has 1 axes (X)
and 1 buttons (BtnA).
Testing ... (interrupt to exit)
Axes:  0: 27197 Buttons:  0:on
```


© CC-BY SA4

Reference Implementation

- https://github.com/e-ale/paddle

© CC-BY SA4