Blinking an LED

….The hard way.

In Linux, everything is a file

Learning about Linux through SYSFS

Thanks to Bill Gatliff

What is SYSFS?

- Virtual file system that exposes drivers to userspace
- `/sys/devices` ← driver hierarchy
- `/sys/class` ← common interfaces

- Let's go thru some examples…

What is SYSFS?

- Virtual file system that exposes drivers to userspace
- bone$ **`cd /sys/class`**
- bone$ **`ls`**

```
backlight  firmware     lcd        net            scsi_device   tty
bdi        gpio         leds       power_supply   scsi_disk     udc
block      graphics     mbox       pwm            scsi_generic  usb_device
bluetooth  hwmon        mdio_bus   regulator      scsi_host     vc
bsg        i2c-adapter  mem        rfkill         sound         video4linux
devfreq    i2c-dev      misc       rtc            spi_master    vtconsole
display    input        mmc_host   scsi_changer   spidev
```

- Let's go through some examples…

Blinking an LED

- Everything is a file in Linux

```
$ cd /sys/class/leds
$ ls -F
beaglebone:green:usr0/
beaglebone:green:usr1/
beaglebone:green:usr2/
beaglebone:green:usr3/
$ cd beaglebone\:green\:usr0
$ ls
brightness device max_brightness power
subsystem trigger uevent
```

Blinking an LED

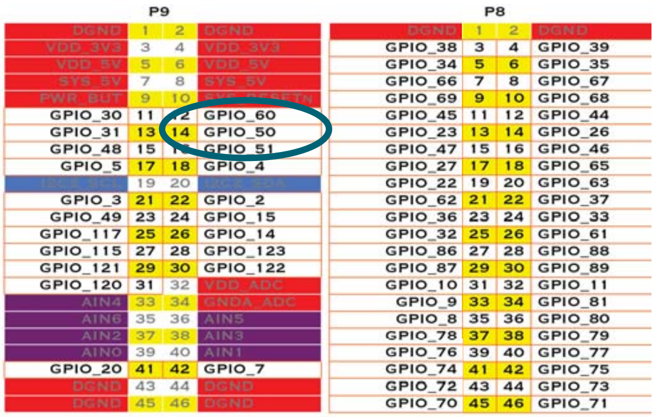
```
$ cat trigger
none nand-disk mmc0 timer oneshot [heartbeat]
     backlight gpio cpu0 default-on transient
$ echo none > trigger
$ echo 1 > brightness
$ echo 0 > brightness
```

Blinking an External LED

- The gpio pins are accessed through

```
/sys/class/gpio
```

- Earlier we used gpio **P9_14**
- The table shows which gpio pin it's assigned to



	P9		
DGND	1	2	DGND
VDD_3V3	3	4	VDD_3V3
VDD_5V	5	6	VDD_5V
SYS_5V	7	8	SYS_5V
PWR_BUT	9	10	SYS_RESETn
GPIO_30	11	12	GPIO_60
GPIO_31	13	14	GPIO_50
GPIO_48	15	16	GPIO_51
GPIO_5	17	18	GPIO_4
I2C2_SCL	19	20	I2C2_SDA
GPIO_3	21	22	GPIO_2
GPIO_49	23	24	GPIO_15
GPIO_117	25	26	GPIO_14
GPIO_115	27	28	GPIO_123
GPIO_121	29	30	GPIO_122
GPIO_120	31	32	VDD_ADC
AIN4	33	34	GNDA_ADC
AIN6	35	36	AIN5
AIN2	37	38	AIN3
AIN0	39	40	AIN1
GPIO_20	41	42	GPIO_7
DGND	43	44	DGND
DGND	45	46	DGND

	P8		
DGND	1	2	DGND
GPIO_38	3	4	GPIO_39
GPIO_34	5	6	GPIO_35
GPIO_66	7	8	GPIO_67
GPIO_69	9	10	GPIO_68
GPIO_45	11	12	GPIO_44
GPIO_23	13	14	GPIO_26
GPIO_47	15	16	GPIO_46
GPIO_27	17	18	GPIO_65
GPIO_22	19	20	GPIO_63
GPIO_62	21	22	GPIO_37
GPIO_36	23	24	GPIO_33
GPIO_32	25	26	GPIO_61
GPIO_86	27	28	GPIO_88
GPIO_87	29	30	GPIO_89
GPIO_10	31	32	GPIO_11
GPIO_9	33	34	GPIO_81
GPIO_8	35	36	GPIO_80
GPIO_78	37	38	GPIO_79
GPIO_76	39	40	GPIO_77
GPIO_74	41	42	GPIO_75
GPIO_72	43	44	GPIO_73
GPIO_70	45	46	GPIO_71

Blinking an External LED

- Here's how you turn it on

```
$ cd /sys/class/gpio
$ ls
export gpiochip0 gpiochip32 gpiochip64 gpiochip96
unexport
```

- No gpio pins are visible

```
$ echo 50 > export
$ ls
export gpio50 gpiochip0 gpiochip32 gpiochip64 …
```

- Notice **gpio50** has appeared

Blinking an External LED

- Go in a take control

```
bone$ cd gpio50
bone$ ls
active_low  direction  edge  power  subsystem  uevent
value
bone$ echo out > direction
bone$ echo 1 > value
```

- Your LED should be on

Reading a switch

- Once you know how to control an LED, reading a switch is easy
- A switch is wired to **P9_42**.Which gpio is this?

```
$ cd /sys/class/gpio
$ echo 7 > export
$ cd gpio7
$ echo in > direction
```

Reading a Switch

- Button not pushed

```
$ cat value
0
```

- Button pushed

```
$ cat value
1
```

Read in a Loop

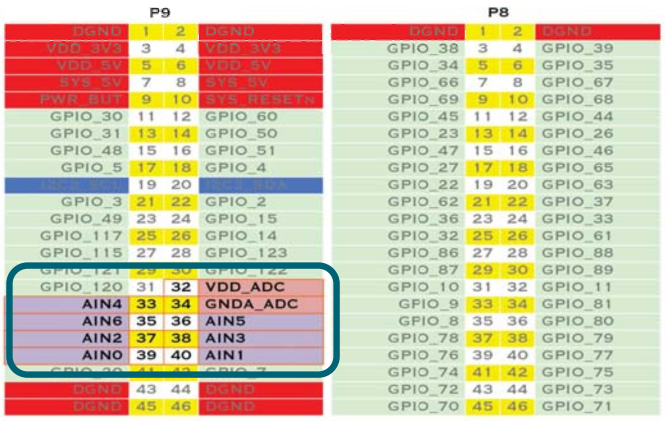
- You ca over and over

Spaces are important

```
#!/bin/ba
cd /sys/  ss/gpio
while [ 1 ]
do
    cat gpio7/value
    sleep 0.25
done
```

```
tr '\n' '\r' < gpio7/value
```

Analog In



P9			
DGND	1	2	DGND
VDD_3V3	3	4	VDD_3V3
VDD_5V	5	6	VDD_5V
SYS_5V	7	8	SYS_5V
PWR_BUT	9	10	SYS_RESETN
GPIO_30	11	12	GPIO_60
GPIO_31	13	14	GPIO_50
GPIO_48	15	16	GPIO_51
GPIO_5	17	18	GPIO_4
I2C2_SCL	19	20	I2C2_SDA
GPIO_3	21	22	GPIO_2
GPIO_49	23	24	GPIO_15
GPIO_117	25	26	GPIO_14
GPIO_115	27	28	GPIO_123
GPIO_121	29	30	GPIO_122
GPIO_120	31	32	VDD_ADC
AIN4	33	34	GNDA_ADC
AIN6	35	36	AIN5
AIN2	37	38	AIN3
AIN0	39	40	AIN1
GPIO_20	41	42	GPIO_7
DGND	43	44	DGND
DGND	45	46	DGND

P8			
DGND	1	2	DGND
GPIO_38	3	4	GPIO_39
GPIO_34	5	6	GPIO_35
GPIO_66	7	8	GPIO_67
GPIO_69	9	10	GPIO_68
GPIO_45	11	12	GPIO_44
GPIO_23	13	14	GPIO_26
GPIO_47	15	16	GPIO_46
GPIO_27	17	18	GPIO_65
GPIO_22	19	20	GPIO_63
GPIO_62	21	22	GPIO_37
GPIO_36	23	24	GPIO_33
GPIO_32	25	26	GPIO_61
GPIO_86	27	28	GPIO_88
GPIO_87	29	30	GPIO_89
GPIO_10	31	32	GPIO_11
GPIO_9	33	34	GPIO_81
GPIO_8	35	36	GPIO_80
GPIO_78	37	38	GPIO_79
GPIO_76	39	40	GPIO_77
GPIO_74	41	42	GPIO_75
GPIO_72	43	44	GPIO_73
GPIO_70	45	46	GPIO_71

Analog In

- Input voltage range is 0 to 1.8V.
- These are accessed much link the gpio

```
$ export SLOTS="/sys/devices/platform/bone_capemgr/slots"
$ echo BB-ADC > $SLOTS
$ cd /sys/bus/iio/devices/iio:device0
$ ls -F
buffer/          in_voltage1_raw  in_voltage4_raw  name      scan_elements/
dev              in_voltage2_raw  in_voltage5_raw  of_node@  subsystem@
in_voltage0_raw  in_voltage3_raw  in_voltage6_raw  power/    uevent
$ cat in_voltage0_raw
3936
```

Analog In - Explore

- How did I figure this out?
- The variable NODE_PATH tells where the node modules are kept

```
bone$ echo $NODE_PATH
/usr/local/lib/node_modules
```

- See what's there

```
bone$ ls $NODE_PATH
async       i2c       node-red-node-bb-upm       npm        serialport
blessed     mraa      node-red-node-beaglebone   request    socket.io
bonescript  node-red  node-red-node-mongodb      sensortag  winston
```

Analog In - Explore

```
bone$ cd
$NODE_PATH/bonescript
bone$ ls
autorun.js                 bonescript.version  LICENSE   node_modules  server.js  test
bonescript.node_version    dts                 main.js   package.json  src
bonescript.npm_version     etc                 Makefile  README.md     systemd
bone$ cd src
bone$ ls
autorun.js     eeprom.js       hw_oldkernel.js  index.js          serial.js
bone.js        functions.js    hw_simulator.js  my.js             server.js
bonescript.js  hw_capemgr.js   hw_universal.js  parse.js          socket_handlers.js
constants.js   hw_mainline.js  iic.js           rewrite_bone.js
```

- What now?

Analog In - Explore

```
bone$ grep analog *
…
index.js: f.analogRead = function(pin, callback) {
…
```

- Look in index.js

```
 resp = hw.readAIN(pin, resp, callback);
bone$ grep readAIN *
hw_mainline.js: exports.readAIN = function(pin, resp, callback) {
```

Analog In - Explore

- In *hw_mainline.js* you find:

```
var ainPrefix = "/sys/bus/iio/devices/iio:device0";
var SLOTS = "/sys/devices/platform/bone_capemgr/slots";
var AINdts = "BB-ADC";
```

```
$ export SLOTS="/sys/devices/platform/bone_capemgr/slots"
$ echo BB-ADC > $SLOTS
$ cd /sys/bus/iio/devices/iio:device0
```

Analog In

- You can keep reading the input using

```
while [ 1 ]
do
 tr '\n' '\r' < in_voltage0_raw
done
```