About Me (and Linux)

- Freelance Embedded Linux Kernel Hacker
- Started with Linux as a hobbyist
- Amiga, PPC, FBDev, MIPS, PS3/Cell, …
- Maintainer of the m68k architecture since 2004
- Maintainer of Renesas clock and pin control drivers since 2016
- Maintainer of Renesas ARM SoC platforms since July 2019

Connecting Simple Devices to a Linux System

- Sensors, motors, switches, LEDs, actuators, displays, solenoids, …
- Makers and Industrial Automation

Images © SIMAC Electronics GmbH, Olimex Ltd
Arduino

- Platform of choice for hooking up simple devices
- Large ecosystem
- Multiple hardware platforms (Various Arduinos, ESP32, Teensy, …)
- Lots of libraries, supporting most popular devices
- Limited processing power
- No hardware description, devices hardcoded in software
- Some common support across boards/families (pin 13 = LED)

Arduino APIs

- Pins
  - pinMode()
  - digitalRead() / digitalWrite()
  - analogRead() / analogWrite()

- Buses
  - Wire (I2C)
  - SPI
  - Serial
  - OneWire

- Devices
  - DallasTemperature
  - ...

Linux

- Platform of choice for devices with a real OS and connectivity
- Large ecosystem
- Lots of drivers supporting most popular devices
- More processing power
  - Ranging from single-core systems with 1 MiB RAM to supercomputers
- Hardware description:
  - Discoverable devices (PCI, …)
  - Devices described by firmware (ACPI, real Open Firmware)
  - Flattened Device Tree
  - Board files ⇒ FDT

Linux APIs

- Strict separation of kernel and userspace
  - (most) Drivers in kernelspace
  - Application in userspace
  - Standardized APIs

- Platform AND peripherals can be replaced without changing the application

- Swapping e.g. sensors

Images © BeagleBoard.org Foundation, Raspberry Pi Foundation, Olimex Ltd
**Example: Getting To Blinky**

**Arduino**

```c
void setup()
{
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop()
{
  digitalWrite(LED_BUILTIN, HIGH);
  delay(1000);
  digitalWrite(LED_BUILTIN, LOW);
  delay(1000);
}
```

*Can we do the same on Linux? From userspace?*

**Example: Getting To Blinky**

**Linux / sysfs GPIO**

```bash
# echo 953 > /sys/class/gpio/export
# echo out > /sys/class/gpio/gpio953/direction
# echo 1 > /sys/class/gpio/gpio953/value
# echo high > /sys/class/gpio/gpio953/direction # shorthand for out/1
# echo low > /sys/class/gpio/gpio953/direction # shorthand for out/0

▶ How to know the GPIO number? Can it change?
▶ sysfs GPIO is deprecated!
▶ Arduino-alike libs (for C, Shell, Python, …)
```

**Example: Getting To Blinky**

**Linux / chardev GPIO / libgpiod**

```bash
# gpiodetect
gpiochip0 [e6050000 gpio] (16 lines)
...
gpiochip8 [0020] (8 lines)
gpiochip9 [bd9571mwv-gpio] (2 lines)

# gpioinfo
gpiochip0 - 16 lines:
  line 0: unnamed unused input active-high
  line 1: unnamed unused input active-high
  line 2: unnamed "SDHIO VccQ" output active-high [used]
...

# gpioset gpiochip2 19=0 # LED off
# gpioset gpiochip2 19=1 # blinks very briefly?!?
# gpioset -m time -u 500000 gpiochip2 19=1
# gpioset -m wait gpiochip2 19=0 20=1 21=0 # multiple
```

**Example: Getting To Blinky**

**Linux / sysfs LED**

```bash
# echo 0 > /sys/class/leds/led6/brightness
# echo 1 > /sys/class/leds/led6/brightness

# cat /sys/class/leds/led6/trigger
[none] usbport timer oneshot disk-activity disk-read disk-write ide-disk mtd nand-disk heartbeat gpio cpu cpu0 cpu1 ...

# echo activity > /sys/class/leds/led6/trigger
# echo pattern > /sys/class/leds/led6/trigger
# echo 0 500 255 500 > /sys/class/leds/led6/pattern # PNM
# echo 0 500 0 0 255 500 255 0 > /sys/class/leds/led6/pattern # Binary

▶ Kernel drivers to the rescue
▶ Needs hardware description
Linux
The Right Stuff

- Out-of-Tree Linux
  - Do whatever you want
  - Custom solutions
  - Less sharing

- Upstream Linux
  - More sharing
  - Long Term vision and maintenance
  - Work with the community to get your work accepted
  - Follow the (un)written/. . . rules

There’s only one way of Linux, and that’s upstream, upstream, upstream!!!

Image © Warner Bros. Entertainment Inc.

Userspace Drivers

- Userspace
  - /dev/* /dev/* /sys/*

- App + Driver(s)
  - /dev/i2c-* (bus)
  - /dev/gpiochip* /dev/spidev* /dev/ttyS*

- High-Level Generic
  - i2cdev
  - gpio/lib-cdev
  - spidev
  - tty
  - pwm-sysfs

- Low-Level Bus
  - I2C
  - GPIO
  - SPI
  - UART

- Device
  - A B C D E F G ...

Comparison: Userspace vs. Kernelspace Drivers

- Userspace Drivers
  - ✓ Simpler to get something to work
  - ✓ Licensing
  - ✓ Custom microcontroller protocols
  - X Reuse and sharing
  - X Interrupts
  - X Overhead

- Kernelspace Drivers
  - ✓ Efficiency
  - ✓ Reuse
  - ✓ Abstraction (replace hardware, keep interface)
  - ✓ Integration with other subsystems
  - ✓ Interrupts
  - X More complex
  - X Upstreaming effort

Too many userspace drivers in these Arduino/<Fruit>Pi/96boards/... days
**Example: HD44780 Character LCD**

[Image of HD44780 Character LCD]

**Userspace: 5+ drivers/libs**

1. GPIO (4-bit/8-bit)
2. I2C
3. SPI
4. W1
5. Custom microcontroller protocol

**Kernel space: One "new" driver**

1. Panel driver → GPIO
   - I2C/SPI/W1 I/O Expander drivers
   - Custom frontend?

---

**Industrial IO (IIO)**

- Sensors, ADC, DAC, ...
- Mostly I2C/SPI
- sysfs / libio

```bash
$ iio_info -n h3-salvator-xs
Library version: 0.10 (git tag: v0.10) Compiled with backends: local xml ip usb serial IIO context created with network backend. ...
IIO context has 2 devices:
iio:device0: max9611 5 channels found:
current: (input)
  attr 0: input value: 2214.500000000
  attr 1: shunt_resistor value: 0.005000
power: (input)
  attr 0: input value: 1854.375600000
  attr 1: shunt_resistor value: 0.005000
temp: (input)
  attr 0: raw value: 75
  attr 1: scale value: 480.076812289
voltage0: (input)
  attr 0: raw value: 75
  attr 1: scale value: 480.076812289
voltage1: (input)
  attr 0: raw value: 62
  attr 2: scale value: 14
```

---

**Existing Drivers**

- Input (evdev, /dev/input/event*)
  - gpio-keys(pollled)
  - rotary-encoder
  - touchscreen

- Output
  - gpio-leds
  - pwm-leds
  - Various displays

- Bit-banged bus implementations
  - i2c-gpio
  - spi_gpio
  - w1-gpio

---

**Pulse-Width Modulation (PWM)**

- pwms-leds
- PWM in-kernel mostly for display backlight
- Userspace: sysfs /sys/class/pwm/ (cfr. legacy GPIO)
  - $ echo 0 > /sys/class/pwm/pwmchip0/export
  - $ echo 50000 > /sys/class/pwm/pwmchip0/pwm0/period # in ns
  - $ echo 30000 > /sys/class/pwm/pwmchip0/pwm0/duty_cycle # in ns
  - $ echo 1 > /sys/class/pwm/pwmchip0/pwm0/enable

- No chardev UAPI or libpwmd yet
RGB LEDs?

It's complicated...

- Multicolor Framework v30
  - https://lwn.net/Articles/826046/
  - v31 made it into v5.9-rc1!
  - Control global/local brightness of a string of LEDs

⇒ RGB LEDs need more

- NeoPixels (WS2812) and DotStars (APA102) etc?
  - NeoPixels: precise timing, SPI abuse?
  - Upstream?

Motors & Actuators?

- No gpio-motors
- No gpio-relays
- No gpio-actuators
  - Workaround/abuse: gpio-leds

- Motor Control / H-Bridge
  - Generic H-Bridge driver?
    (forward/backward/stop/brake, speed)

Hardware Description

- PCI, W1: Autoprobe
- I2C: Can be manual for simple devices:
  - # echo pcf8574 0x24 > /sys/bus/i2c/devices/i2c-1/new_device
  - /dev/i2c-%u, /dev/ttyS%u: Always available

- Devices described in Device Tree
  - Identification through compatible values
  - Resources: reg, interrupts
  - Properties (may be device-specific)
  - Phandles
  - Subnodes

GPIO Aggregator (v5.8+)

- GPIO access control uses permissions on /dev/gpiochip*
  - All or Nothing

- Solution: Aggregate existing GPIOs, and expose them as a new gpiochip
  - # echo 'e6052000.gpio 19 e6050000.gpio 20-21' > /sys/bus/platform/drivers/gpio-aggregator/new_device
  - # gpioinfo gpio-aggregator.0
    - gpiochip12 - 3 lines:
      - line 0: unnamed unused input active-high
      - line 1: unnamed unused input active-high
      - line 2: unnamed unused input active-high
  - # chown geert /dev/gpiochip12
  - Started as a way to group GPIOs for export to a VM
  - Generic GPIO Driver (cfr. i2cdev/spidev)
Sample Device Tree Snippet: SPI Controller and Device

```c
spi1: spi@e6e10000 {
  compatible = "renesas,msiof-r8a7791", "renesas,rcar-gen2-msiof";
  reg = <0 0xe6e10000 0 0x0064>;
  interrupts = <GIC_SPI 157 IRQ_TYPE_LEVEL_HIGH>;
  clocks = <&cpg CPG_MOD 208>;
  #address-cells = <1>;
  #size-cells = <0>;
  status = "disabled";
};

&spi1 {
  status = "okay";
  flash0: eeprom@0 {
    compatible = "microchip,25lc040", "atmel,at25";
    reg = <0>;
    pagesize = <16>;
    size = <512>;
    address-width = <9>;
    spi-max-frequency = <100000>;
  };
};
```

"I need spidev in DT!"

```c
&spi1 {
  status = "okay";
  spidev0 {
    compatible = "spidev";
    reg = <0>;
    spi-max-frequency = <100000>;
  };
};
```

Rule #1: DT describes hardware, not software policy

- Use a proper compatible value!
- Add to `drivers/spi/spidev.c:spidev_dt_ids[]`
- Or bind manually:
  ```bash
  # echo spidev > /sys/class/spi_master/spi1/spi1.0/driver_override
  # echo spi1.0 > /sys/bus/spi/drivers/spidev/bind
  and /dev/spidev1.0 appears
  ```

Enabling My Device in DT

```
myvendor,mydevice
```

1. Add DT bindings (preferably json-schema / YAML)
   - Vendor prefix
     Documentation/devicetree/bindings/vendor-prefixes.yaml
   - Device DT bindings
     Documentation/devicetree/bindings/.../myvendor,mydevice.yaml
   - Or trivial device
     Documentation/devicetree/bindings/trivial-devices.yaml

2. Driver
3. Device node in board DTS
4. Validate
   ```bash
   $ make dt_binding_check dtbs_check
   ```

DT Bindings Example: GPIO-operated Door

```
# SPDX-License-Identifier: (GPL-2.0-only OR BSD-2-Clause)

---
$id: "http://devicetree.org/schemas/misc/myvendor,mydoor.yaml#"
$schema: "http://devicetree.org/meta-schemas/core.yaml#"
title: Myvendor mydoor device
maintainers:
  - J.K. Hacker <developer@myvendor.com>
properties:
  compatible:
    const: myvendor,mydoor
  gpios:
    maxItems: 2
    gpio-line-names: ["open", "lock"]
required:
  - compatible
  - gpios
  - gpio-line-names
additionalProperties: false
```

```bash
$ include <dt-bindings/gpio/gpio.h>
door {
  compatible = "myvendor,mydoor";
  gpios = <gpio2 19 GPIO_ACTIVE_HIGH>,
         <gpio2 20 GPIO_ACTIVE_LOW>;
  gpio-line-names = "open", "lock";
};
```

Disclaimer: example fails to validate!
Driver / Binding

- New driver / existing driver
- Add compatible value to driver
- Or bind manually:
  ```
  # echo gpio-aggregator > /sys/bus/platform/devices/door/driver_override
  # echo door > /sys/bus/platform/drivers/gpio-aggregator/bind
  
  # gpioinfo door
  gpiochip12 - 2 lines:
  line 0: "open" unused input active-high
  line 1: "lock" unused input active-high
  
  # gpiofind open
  gpiochip12 0
  
  # gpioinfo gpiochip12
  gpiochip12 - 2 lines:
  line 0: "open" unused input active-high
  line 1: "lock" unused input active-high
  ```

How to get it in your DT?

1. Add to board DTS
2. Create a DT overlay (more flexibility)
   2.1 Load overlay at runtime ⇒ requires out-of-tree patches (I keep them up-to-date)
      https://elinux.org/R-Car/DT-Overlays
      https://git.kernel.org/pub/scm/linux/kernel/git/geert/reneas-drivers.git/log/?h=topic/overlays
   2.2 Let U-Boot load the overlay before starting the kernel
      https://gitlab.denx.de/u-boot/u-boot/-/blob/master/doc/README.fdt-overlays
   2.3 Combine base DTS and overlays using fdtoverlay

Sample DT Overlay / Sugar Syntax

```dts-v1;
/plugin/; // Main DTS must be processed with -@

&spi1 {
  #address-cells = <1>;
  #size-cells = <0>;
  status = "okay";

  flash0: eeprom0 {
    compatible = "microchip,25lc040", "atmel,at25";
    reg = <0>;
    pagesize = <16>;
    size = <512>;
    address-width = <9>;
    spi-max-frequency = <100000>;
  }
};
```

Dynamic DT Overlays

- BeagleBone Black Cape Manager (Pantelis et al), used on Raspberry Pi, too
- What works (not)?
  - SPI, I2C, GPIO hogs, Platform devices (using of_reconfig_notifier_register())
  - Pinctrl
  - Enable/disable device
  - DT aliases
  - Endpoints?

Caveats

- Know what you're doing, with great power comes great responsibility!
- Memory leaks, locking, ...
- Frank's Evolving Overlay Thoughts
  https://elinux.org/Frank%27s_Evolving_Overlay_Thoughts

Do we really need this?

- No hotplugging in real life, but useful for prototyping
- FPGA hardware can be reconfigured at run-time
- Arduino has dynamic pin configuration
Connector Framework: Making DT Overlays Safe(r)

- Confine overlays to a subset of the system:
  - The Connector

- Candidates for a Proof-of-Concept?
  - Qwiic, Grove, UEXT, Pmod, mikroBUS, Feather Wing, Raspberry Pi HAT, BeagleBone Cape: 4–92 pins
  - Custom development board connectors: +100 pins
  - Various levels of functionality and multiplexing

- To Be Continued...

Thanks & Acknowledgements

- Renesas Electronics Corporation, for contracting me for upstream Linux kernel work,
- The Linux Foundation, for organizing this conference and giving me the opportunity to present here,
- The Linux Kernel Community, for having so much fun working together towards a common goal.

Questions & Answers

Appendix
Pin Control and Pin Muxing (pinctrl)

- Very SoC-specific
- Some common properties
- Requires DT (debugfs?)
## Pinctrl Examples

```
i2c0_pins: pinmux_i2c0_pins {
    pinctrl-single,pins = <
        AM33XX_PADCONF(AM335X_PIN_I2C0_SDA, PIN_INPUT_PULLUP, MUX_MODE0) /* i2c0_sda.i2c0_sda*/
        AM33XX_PADCONF(AM335X_PIN_I2C0_SCL, PIN_INPUT_PULLUP, MUX_MODE0) /* i2c0_scl.i2c0_scl*/
    >;
}

scif2_pins: serial2 {
    pinmux = <RZA1_PINMUX(3, 0, 6)>, /* TxD2 */
             <RZA1_PINMUX(3, 2, 4)>; /* RxD2 */
};

ether_pins: ether {
    groups = "eth_link", "eth_mdio", "eth_rmii";
    function = "eth";
};
```

## Connectors

<table>
<thead>
<tr>
<th>Name</th>
<th>Pins</th>
<th>Signals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Qwiic Connect System (SparkFun Electronics)</td>
<td>4</td>
<td>power + I2C</td>
</tr>
<tr>
<td>Grove Ecosystem (Seeed Studio)</td>
<td>4</td>
<td>digital, analog, UART, or I2C</td>
</tr>
<tr>
<td>UEXT (Olimex)</td>
<td>10</td>
<td>UART + I2C + SPI</td>
</tr>
<tr>
<td>Pmod Interface (Digilent)</td>
<td>6/12</td>
<td>GPIO, PWM, SPI, UART, I2C, I2S, H-Bridge, Interrupt, Reset</td>
</tr>
<tr>
<td>mikroBUS (MikroElektronika)</td>
<td>16</td>
<td>PWM + UART + I2C + SPI + Analog + Interrupt + Reset</td>
</tr>
<tr>
<td>Feather Wing (Adafruit)</td>
<td>28</td>
<td>UART + I2C + SPI, Analog, PWM, GPIO</td>
</tr>
<tr>
<td>Raspberry Pi HAT</td>
<td>(26)40</td>
<td>UART + I2C + SPI, PWM, GPIO</td>
</tr>
<tr>
<td>BeagleBone Cape</td>
<td>2 × 46</td>
<td>Heavily multiplexed</td>
</tr>
</tbody>
</table>

## References

- [Linux GPIO: Evolution and Current State of the User API](https://elinux.org/ELC_2020_Presentations), Bartosz Golaszewski, Embedded Linux Conference 2020
- [linux/Documentation/ABI/testing/gpio-cdev](https://www.kernel.org/pub/linux/kernel/v5.x-doc/ABI/gpio-cdev.txt)
- [linux/Documentation/ABI/testing/sysfs-class-pwm](https://www.kernel.org/pub/linux/kernel/v5.x-doc/ABI/sysfs-class-pwm.txt)
- [linux/Documentation/devicetree/bindings/writing-bindings.rst](https://www.kernel.org/pub/linux/kernel/v5.x-doc/devicetree/bindings/writing-bindings.rst)
- [linux/Documentation/driver-api/gpio/drivers-on-gpio.rst](https://www.kernel.org/pub/linux/kernel/v5.x-doc/driver-api/gpio/drivers-on-gpio.rst)
- [linux/Documentation/i2c/dev-interface.rst](https://www.kernel.org/pub/linux/kernel/v5.x-doc/Documentation/i2c/dev-interface.rst)
- [linux/Documentation/i2c/i2c-dev-interface.rst](https://www.kernel.org/pub/linux/kernel/v5.x-doc/Documentation/i2c/i2c-dev-interface.rst)
- [linux/Documentation/i2c/spidev.rst](https://www.kernel.org/pub/linux/kernel/v5.x-doc/Documentation/i2c/spidev.rst)