Getting To Blinky: Virt Edition

Making Device Pass-Through Work on Embedded ARM

Geert Uytterhoeven
geert@linux-m68k.org

GLIDER bvba

FOSDEM 2019 / Virtualization and IaaS devroom

Glider.be

Table of Contents

About Me (and Computers)

Hobbyist

1985 Commodore 64

1988 Commodore Amiga 500

1994 Linux/m68k on Amiga

1997 Linux/PPC on CHRP

1997 Linux FBDev

Sony

2006 Linux on PS3/Cell

SONY

GLIDER bvba

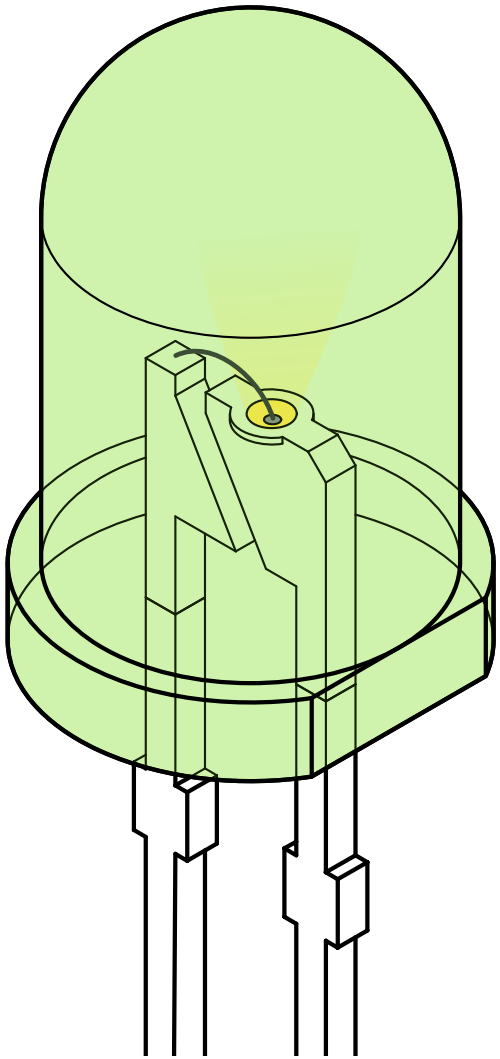
2013 Renesas ARM-based SoCs

RENESAS

About Me (and FOSDEM)

2001 OSDEM
2002 FOSDEM
2003 FOSDEM
2004 FOSDEM, Embedded Track Program Committee
...
...
2019 FOSDEM, still going strong!

LED = Light-Emitting Diode



Holy grail of embedded engineers!

Getting To Blinky: Virt Edition

Inspiration for the title

Getting To Blinky 4.0 by Contextual Electronics*

> *Getting to Blinky 4.0 is a short video series introducing the key concepts of using the open source ecad software KiCad. You will be building a small blinking board to cover each of these concepts.*
> `https://contextualelectronics.com/courses/getting-to-blinky/`

Designing your own PCB with KiCad is cool and fun, but . . .

This is the **Virtualization** Devroom!

*I am not affiliated to CE. I did enjoy their videos.

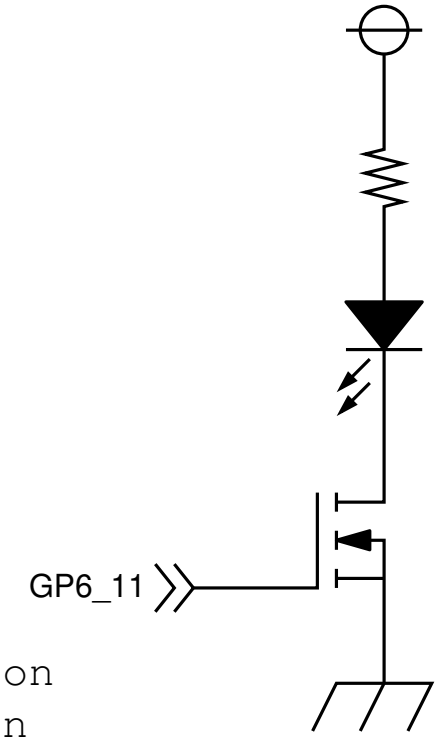
Getting To Blinky: Virt Edition

Renesas Salvator-XS with R-Car H3

- 4x Cortex-A57, 4x Cortex-A53
- GPIO
- IOMMU, SATA, ...

Control LED connected to GPIO on real hardware from Linux from sysfs[†]

```
# echo 371 > /sys/class/gpio/export
# echo high > /sys/class/gpio/gpio371/direction
# echo low > /sys/class/gpio/gpio371/direction
```

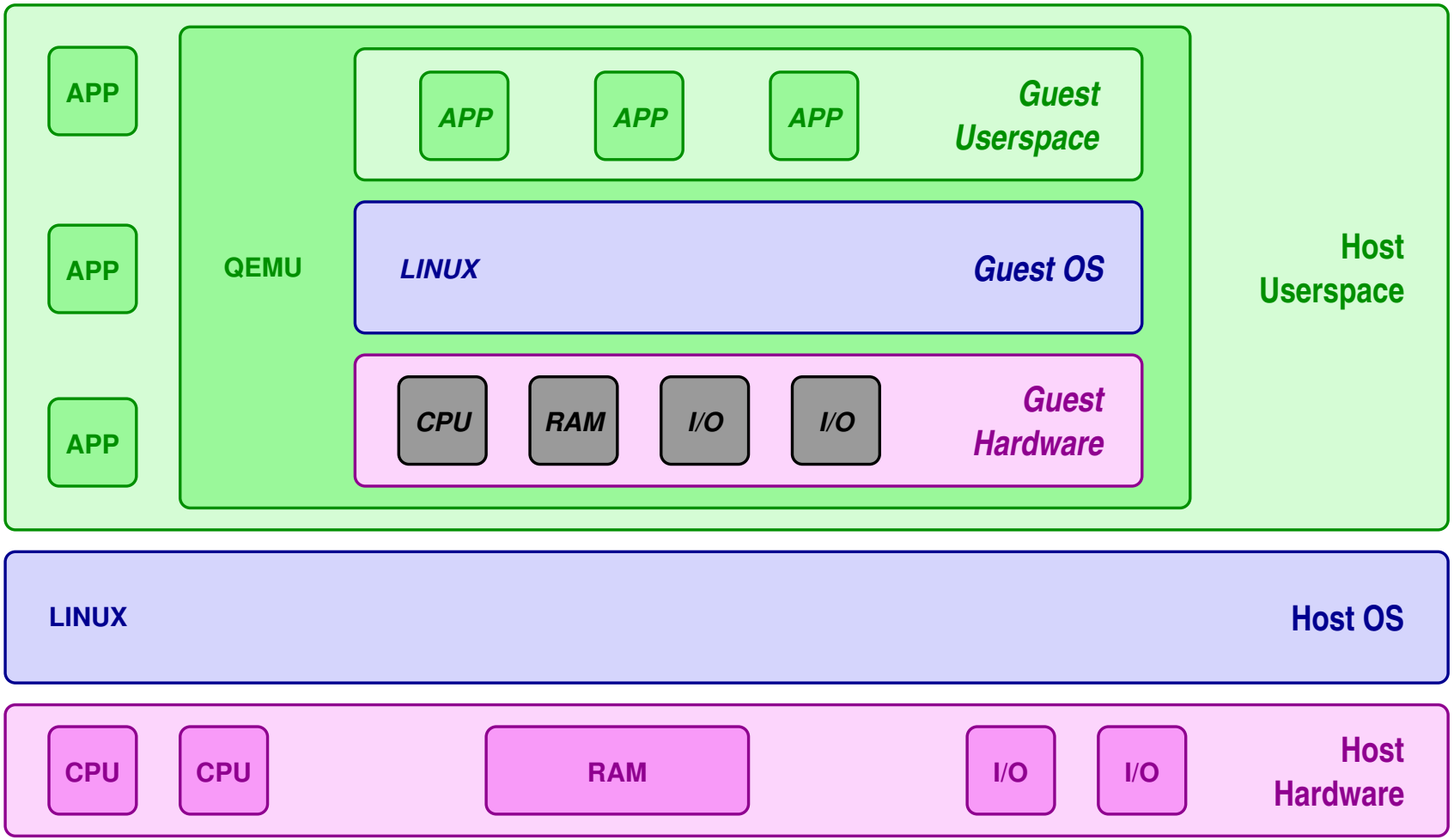


Can we control the LED from a guest, too?

[†]Or use the new chardev GPIO API.

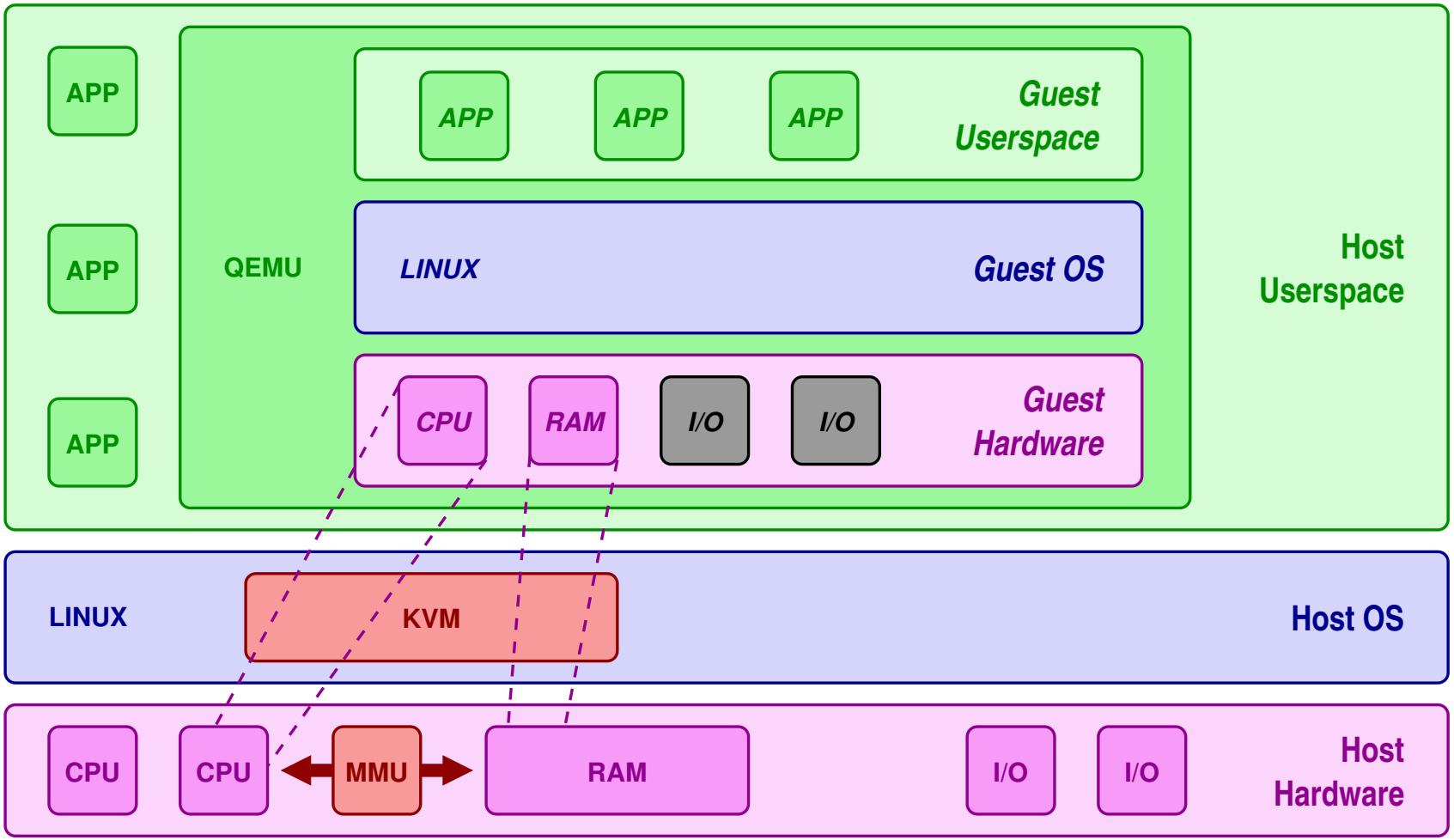
QEMU

Emulation / Paravirtualization



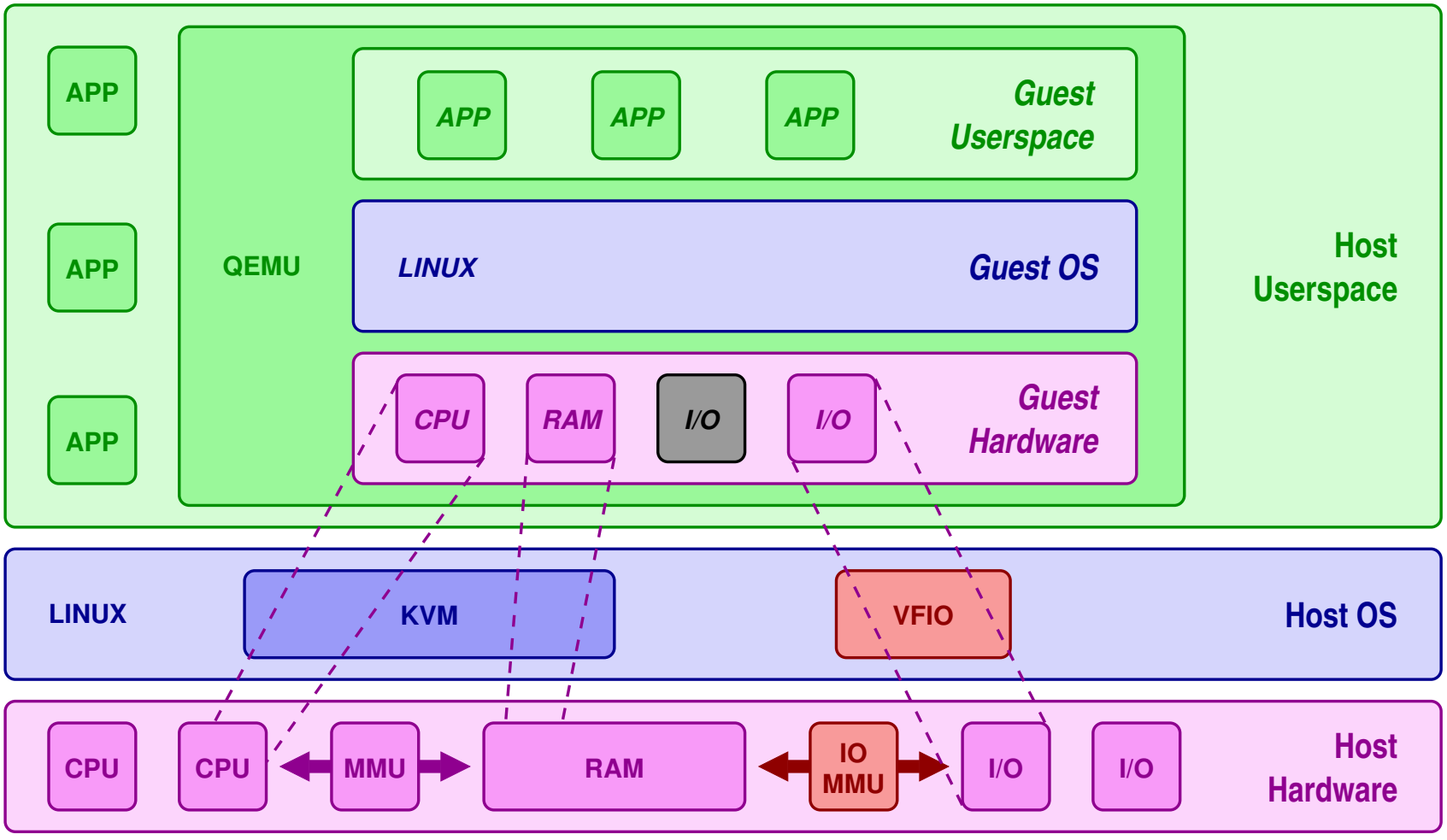
QEMU+KVM

Kernel-Based Virtual Machine / Virtualization Extensions



Device Pass-Through

Virtual Function I/O



Virtual Function I/O: PCI

- Mature, simple, and standardized
- PCI configuration space (vendor/device ID, BARs, caps)

Linux

1. Unbind PCI device from current driver:

```
# $dev=dddd:bb:ss:f
# echo $dev > /sys/bus/pci/devices/$dev/driver/unbind
```

2. Override matching of PCI device:

```
# echo vfio-pci > /sys/bus/pci/devices/$dev/driver_override
```

3. Bind PCI device to `vfio-pci` driver:

```
# echo $dev > /sys/bus/pci/drivers/vfio-pci/bind
```

QEMU

```
# qemu ... -device vfio-pci,host=bb:ss:f[,...]
```

- Other: vfio-ap, vfio-ccw, vfio-amba

Virtual Function I/O: Platform Devices

`vfio-platform`

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

  $\Rightarrow$ More complex, less standardized
- Limited hardware support
  - AMD XGBE
  - Calxeda XGMAC

Virtual Function I/O: Platform Devices

`vfio-platform`

Linux

1. Unbind platform device from current driver:

```
# dev=xxxxxxxx.foo
# echo $dev > /sys/bus/platform/devices/$dev/driver/unbind
```

2. Override matching of platform device:

```
# echo vfio-platform > \
        /sys/bus/platform/devices/$dev/driver_override
```

3. Bind platform device to `vfio-platform` driver:

```
# echo $dev > /sys/bus/platform/drivers/vfio-platform/bind
```

QEMU

```
# qemu ... -device vfio-platform,host=$dev
```

Example: rcar-gpio

GPIO block controlling up to 32 GPIOs

```
gpio6: gpio@e6055400 {
    compatible = "renesas,gpio-r8a7795",
                 "renesas,rcar-gen3-gpio";
    reg = <0 0xe6055400 0 0x50>;
    interrupts = <GIC_SPI 10 IRQ_TYPE_LEVEL_HIGH>;
    #gpio-cells = <2>;
    gpio-controller;
    gpio-ranges = <&pfc 0 192 32>;
    #interrupt-cells = <2>;
    interrupt-controller;
    clocks = <&cpg CPG_MOD 906>;
    power-domains = <&sysc R8A7795_PD_ALWAYS_ON>;
    resets = <&cpg 906>;
};
```

Try #1

1. Unbind GPIO6 from the `gpio-rcar` driver:

```
# echo e6055400.gpio > \
        /sys/bus/platform/drivers/gpio_rcar/unbind
gpio gpiochip6: REMOVING GPIOCHIP WITH GPIOS STILL REQUESTED
```

2. Override and bind GPIO6 to the `vfio-platform` driver:

```
# echo vfio-platform > \
        /sys/bus/platform/devices/e6055400.gpio/driver_override
# echo e6055400.gpio > \
        /sys/bus/platform/drivers/vfio-platform/bind
```

3. **Failure:**

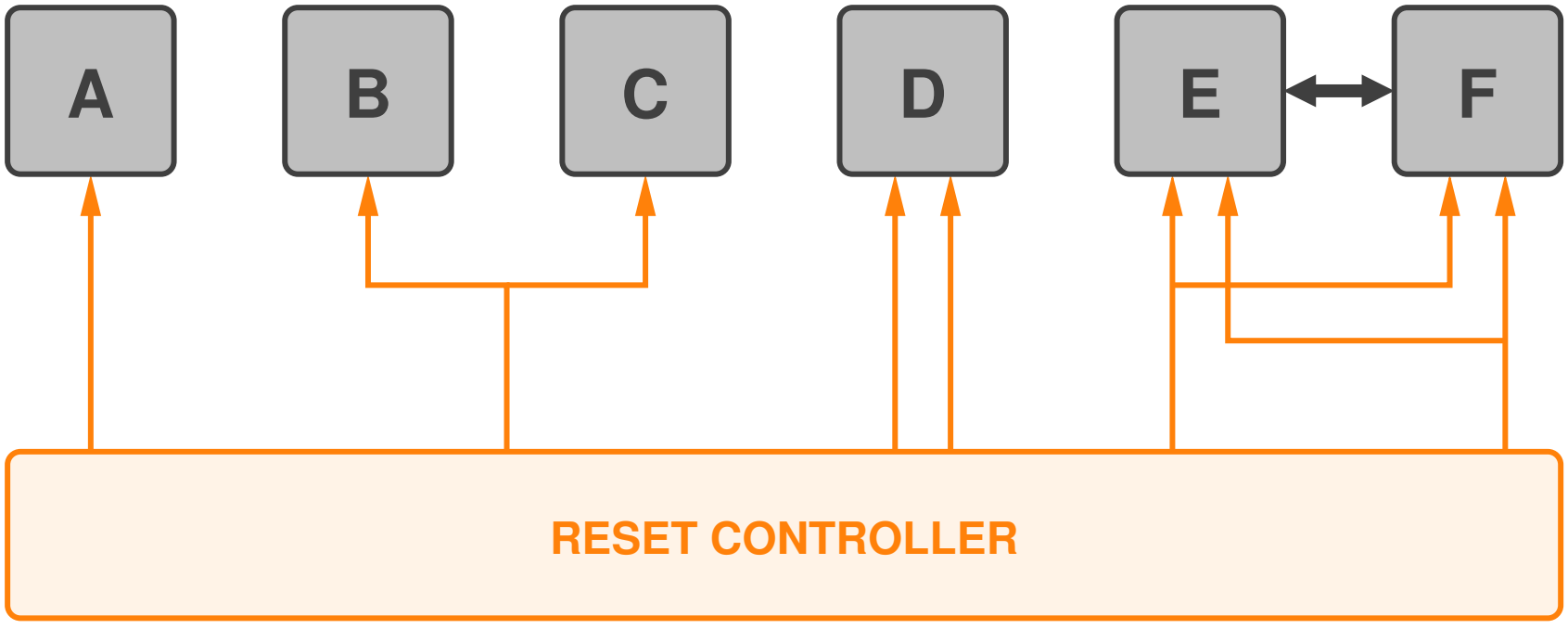
```
-bash: echo: write error: No such device

vfio: no reset function found for device e6055400.gpio
vfio-platform: probe of e6055400.gpio failed with error -2
```

Intermezzo: VFIO and Device Reset

- Why reset?
  Reset device to known state
  - **Before** guest starts using it
  - **After** guest has used it

- Device-specific VFIO reset driver, to be written for each and every device to be exported

Intermezzo: SoC Reset Topology & Linux Reset Subsystem



- Reset topology described in DT (`resets` properties)
- `include/linux/reset.h`: `reset_control_*()` API
- Generic solution for case A:
  vfio: platform: Add generic reset controller support
  (not yet accepted upstream)

Try #2

1. Unbind GPIO6 from the `gpio-rcar` driver:

```
# echo e6055400.gpio > \
        /sys/bus/platform/drivers/gpio_rcar/unbind
gpio gpiochip6: REMOVING GPIOCHIP WITH GPIOS STILL REQUESTED
```

2. Override and bind GPIO6 to the `vfio-platform` driver:

```
# echo vfio-platform > \
        /sys/bus/platform/devices/e6055400.gpio/driver_override
# echo e6055400.gpio > \
        /sys/bus/platform/drivers/vfio-platform/bind
```

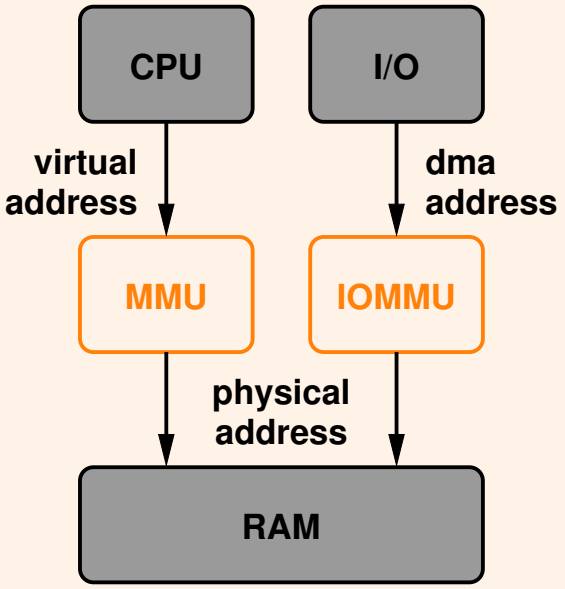
3. **Failure:**

```
-bash: echo: write error: No such device

VFIO: No IOMMU group for device e6055400.gpio
vfio-platform: probe of e6055400.gpio failed with error -22
```

Intermezzo: IOMMU and DMA

Why IOMMU?

- Address translation and protection for DMA-capable devices
- Maintain system integrity
  - Host $\leftrightarrow$ Guest
  - Guest $\leftrightarrow$ Guest



No DMA? $\Rightarrow$ VFIO No-IOMMU mode

```
CONFIG_VFIO_NOIOMMU=y
```

```
# echo 1 > /sys/module/vfio/parameters/enable_unsafe_noiommu_mode
```

Intermezzo: When does a device use DMA?

PCI

Always assumed to be DMA capable
IOMMU hierarchy known from bus hierarchy

DT

Difficult to know

- `iommus` property present: yes
- `dmas` property present: yes
  Points to DMAC, which may (not) have `iommus` property
  $\Rightarrow$ repeat question for DMAC device node
- Else: *maybe*

$\Rightarrow$ Better safe than sorry

Success #1

Linux is happy, proceed to QEMU

- QEMU has support for instantiating AMD XGBE and Calxeda XGMAC
- Let QEMU instantiate minimal rcar-gpio node:

```
platform@c000000 {
        ...
        e6055400.gpio@0 {
                reg = <0x0 0x50>;
                gpio-controller;
                #gpio-cells = <0x2>;
                interrupts = <0x0 0x70 0x4>;
                compatible = "renesas,rcar-gen3-gpio";
        };
};
```

Try #3

1. Launch QEMU

```
# qemu ... -device vfio-platform,host=e6055400.gpio
```

2. **Failure:**

```
vfio error: e6055400.gpio: failed to open /dev/vfio/0:
        No such file or directory
```

- `/dev/vfio/noiommu-0` instead of `/dev/vfio/0`
- Linux has support for *No-IOMMU Mode*
- QEMU has not!
  Needs *vfio: No-IOMMU mode support* by Xiao Feng Ren

Try #4

1. Launch QEMU

```
# qemu ... -device vfio-platform,host=e6055400.gpio
```

2. Guest boots, GPIO driver is initialized:

```
gpio_rcar c000000.e6055400.gpio: driving 32 GPIOs
```

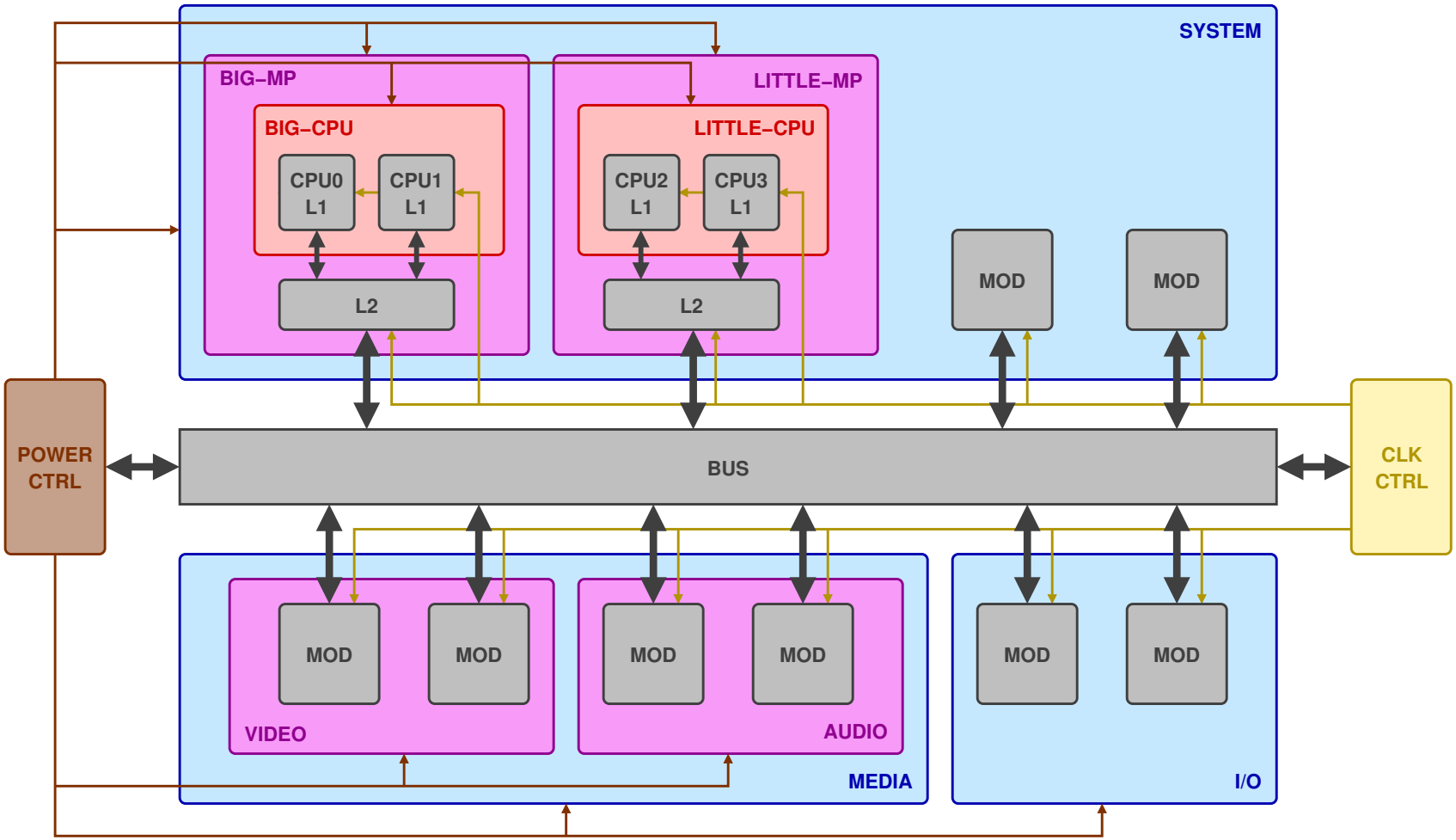
3. Let's control the LED

```
# Export GPIO used for LED
echo 491 > /sys/class/gpio/export

# Turn LED off/on
echo low > /sys/class/gpio/gpio491/direction
echo high > /sys/class/gpio/gpio491/direction
```

4. **Failure:** Nothing happens

Intermezzo: SoC Power Management Topology

Multiple Hierarchical Power Areas, Clock Domain



Try #4

Needs PM Domain (Clock/Power Domain) handling!

- Device off ⇒ **undefined behavior**
  Nothing happens, reads back zeroes, exception, crash/lock-up (of the whole system!)

- Clock Domain: ~~Explicit clock management~~ Runtime PM
- Power Area: Runtime PM
- Solution: Host calls `pm_runtime_get_sync()` when guest opens VFIO device
  vfio: platform: Fix using devices in PM Domains
  (commit 415eb9fc0e23071f in Linux v4.18)

- Delegate to guest?
  - ✓ More fine-grained power control
  - X Inherently unsafe

Success #2#3

BLINKY!!

Q: Does this work for input (button) too?

A: Worked out-of-the-box, after GIC fix
KVM: arm/arm64: vgic: Disallow Active+Pending for level interrupts
(commit 67b5b673ad4d4691 in Linux v4.17)

Other Devices?

- `sh-sci` serial (hacked driver, PIO only, by Kieran)
- `rcar-sata` (DMA, behind IOMMU)
  - Does not work with Renesas IPMMU:

    ```
    vfio error: ee300000.sata: failed to setup container for
            group 0: No available IOMMU models
    ```

    Fixed after enabling `CONFIG_VFIO_IOMMU_TYPE1`
    drivers/vfio: Allow type-1 IOMMU instantiation with all ARM/ARM64 IOMMUs
    (commit cf3f98c7f466a7c7 in Linux v4.20)
  - `sata_rcar` does explicit clock management
    ata: sata_rcar: Add rudimentary Runtime PM support
    (commit 1ecd34ddf63ef1d4 in Linux v4.19)

- Add support for a new device?
  ⇒ Extend QEMU to instantiate the device in DT

Generic Device Instantiation

Copy DT from host to guest instead of instantiating a fixed device node:

- Remap `reg` and `interrupts` properties
- Copy simple properties (no phandles!)
  $\Rightarrow$ zero-sized or whitelisted
- Ignore properties handled by host
  (power management, isolation, pin control)
- Reject clocks if no PM
- Reject subnodes

$\Rightarrow$ Limited to simple devices

hw/arm/sysbus-fdt: Add support for instantiating generic devices
(not yet accepted upstream)

Example: rcar-sata

Host DT

```
sata: sata@ee300000 {
    compatible = "renesas,sata-r8a7795", "renesas,rcar-gen3-sata";
    reg = <0 0xee300000 0 0x200000>;
    interrupts = <GIC_SPI 105 IRQ_TYPE_LEVEL_HIGH>;
    clocks = <&cpg CPG_MOD 815>;
    power-domains = <&sysc R8A7795_PD_ALWAYS_ON>;
    resets = <&cpg 815>;
    status = "disabled";
    iommus = <&ipmmu_hc 2>;
};
```

Guest DT

```
platform@c000000 {
    ...
    ee300000.sata@0 {
        status = "okay";
        reg = <0x0 0x200000>;
        interrupts = <0x0 0x70 0x4>;
        compatible = "renesas,sata-r8a7795", "renesas,rcar-gen3-sata";
    };
};
```

Conclusions

- Devices on SoCs for embedded are usually not on a PCI bus
- IOMMUs are present, and virtualization is wanted
- Device Pass-Through makes sense for High-Bandwidth devices
  - SATA
  - Ethernet
  - GPU
  - ...
- Simple devices can be instantiated in a generic way
- More complex devices still need device-specific instantiation code
  $\Rightarrow$ Provide helpers to ease implementation

Challenges & Limitations

- Clocks: not just used for power management
- DMACs: Follow `dmas` if no iommu_group is found
- Dividing complex devices up for host and (multiple) guest access
  - GPIOs on the same gpiochip
  - DMA channels on the same DMAC
  - Devices connected to multiple DMACs
  - Display pipelines sharing devices
  - Two devices in the same 4 KiB MMIO page

  $\Rightarrow$ Hardware (SoC/board) designers should take virtualization into account
- DT binding schema checks may help in automatic classification of virt-safe devices?

Back to Blinky. . .

- Real world GPIO: relays, power switching, . . .
- No high bandwidth, no DMA
- Better solutions for GPIO

- PoC using Emulation: *[qemu POC] Add a GPIO backend*
  - Backend: `libgpiod`
  - Frontend: Existing PL061 on ARM virtual machine
  - GPIO outputs only

```
# qemu ... -gpiodev e6055400.gpio,vgpios=0:1:2,gpios=11:12:13
```

- Paravirtualization ≈ `libgpiod` API?

- Extend to similar subsystems, like PWM? ⇒ RGB Blinky ;-)
  Real world: motor control, . . .

Thanks & Acknowledgements

- **Renesas Electronics Corporation**, for contracting me for upstream Linux kernel work,
- **Eric Auger**, for his initial work for instantiating VFIO platform devices in QEMU,
- **Kieran Bingham**, for trying VFIO with SCIF serial,
- **FOSDEM and its volunteer team**, for organizing this conference and giving me the opportunity to present here,
- The **Linux Kernel and QEMU Communities**, for having so much fun working together towards a common goal.

Questions?

References

- Device Pass-Through Using VFIO on R-Car SoCs
  https://elinux.org/R-Car/Virtualization/VFIO
- Linux side
  https://git.kernel.org/pub/scm/linux/kernel/git/geert/renesas-drivers.git/log/?h=topic/rcar3-virt-gpio-passthrough-v3
- QEMU side
  https://github.com/geertu/qemu/tree/topic/rcar3-virt-gpio-passthrough-v5
- "[PATCH v5] vfio: platform: Add generic reset controller support"
  https://lore.kernel.org/lkml/20181113131508.18246-1-geert+renesas@glider.be/
- "[qemu POC] Add a GPIO backend"
  https://lists.gnu.org/archive/html/qemu-devel/2018-10/msg00554.html