VFIO for Platform Devices

Bharat Bhushan, Stuart Yoder

October 2013
Agenda

- VFIO
- Platform Devices
- vfio-platform
  - Platform bus wildcards
  - Binding issues
  - Dealing with complex devices
- QEMU

QEMU is a trademark of Fabrice Bellard.
QEMU/KVM – Device Pass-through

Goal: assign physical I/O devices to virtual machines
What is VFIO?

- Generic framework to expose I/O devices to user space

- Exposes mappable regions (e.g. PCI I/O and mem) to user space through file descriptors

- Exposes interrupts through file descriptors (eventfd)

- IOMMU support—DMA is isolated to the user space software context
VFIO

- VFIO has a layered architecture to support different IOMMUs and busses

- In kernel since 3.6

- See Documentation/vfio.txt
The Other Side of the Problem: User Space

- QEMU uses vfio mechanisms to expose device resources in virtual machine
- Guest sees devices on a virtual bus
The Linux® Driver Model

Drivers

- ata_piix
- ehci_hcd
- megaraid_sas

Device Register

- device_register()

Drivers

- 0000:00:1f.2
- 0000:00:1a.7
- 0000:03:00.0

Devices

binding

Drivers

- ata_piix
- ehci_hcd
- megaraid_sas

Device Register

- device_register()

Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.
vfio-pci binding/unbinding example

• Assume PCI device 0000:06:0d.0 is to be passed to user space

$ lspci -n -s 0000:06:0d.0

06:0d.0 0401: 1102:0002 (rev 08)

$ echo 0000:06:0d.0 > /sys/bus/pci/devices/0000:06:0d.0/driver/unbind

$ echo 1102 0002 > /sys/bus/pci/drivers/vfio-pci/new_id

vfio-pci functions as a “meta” driver—binding to any PCI device type
Anatomy of a system-on-a-chip (SoC)
Platform Bus, Platform Devices

- Most SoC I/O can’t be ‘discovered’ by an OS
- Linux is told via a device tree what devices exist
- Platform drivers register with the platform bus
- Platform devices register with the platform bus…based on parsing dev tree
vfio-platform

• Existing vfio mechanisms can be used for platform devices:
  − Exposing mappable regions
  − Exposing interrupts
  − DMA mapping

• A small handful of issues need to be solved

• Current vision of vfio-platform does not solve pass-through for all platform devices. Complicated devices with convoluted cross-device entanglements will be an issue.
Example: UART

```c
serial0: serial@4500 {
    compatible = "fsl,ns16550", "ns16550";
    reg = <0x4500 0x100>;
    clock-frequency = <200000000>;
    interrupts = <42 2 0 0>;
};
```

(modified for illustration purposes)
Bind to vfio-platform

$ echo 12ce0000.i2c > /sys/bus/platform/drivers/s3c-i2c/unbind
$ echo 12ce0000.i2c > /sys/bus/platform/drivers/vfio-platform/bind
Bind to vfio-platform

$ echo 12ce0000.i2c > /sys/bus/platform/drivers/s3c-i2c/unbind
$ echo 12ce0000.i2c > /sys/bus/platform/drivers/vfio-platform/bind

Two problems:

- Platform bus doesn’t have a ‘wildcard’ mechanism that allows vfio-platform to bind to any platform device...i.e. we need vfio-platform to act as a ‘meta’ driver.
Bind to vfio-platform

Two problems:

- Platform bus doesn’t have a ‘wildcard' mechanism that allows vfio-platform to bind to any platform device…i.e. we need vfio-platform to act as a ‘meta’ driver.

- We want vfio driver binding to devices **only** by explicit request, but the Linux driver core doesn’t support this.

  - Without this mechanism both PCI and platform vfio face the racy situation where two drivers support a device type and it is ambiguous as to which of the two drivers will bind to the device.
Platform Bus Wildcard

```c
static int platform_match(struct device *dev, struct device_driver *drv)
{
    struct platform_device *pdev = to_platform_device(dev);
    struct platform_driver *pdrv = to_platform_driver(drv);

    /* the driver matches any device */
    if (pdrv->match_any_dev)
        return 1;

    /* Attempt an OF style match first */
    if (of_driver_match_device(dev, drv))
        return 1;

diff --git a/include/linux/platform_device.h b/include/linux/platform_device.h
index ce8e4ff..2d25d50 100644
--- a/include/linux/platform_device.h
+++ b/include/linux/platform_device.h
@@ -178,6 +178,7 @@ struct platform_driver {
       int (*resume)(struct platform_device *);
       struct device_driver driver;
       const struct platform_device_id *id_table;
+      bool match_any_dev;
    };
```

http://www.spinics.net/lists/kvm/msg97195.html
sysfs_bind_only

New

struct device_driver {
    const char *mod_name; /* used for built-in modules */

    bool suppress_bind_attrs; /* disables bind/unbind via sysfs */
    + bool sysfs_bind_only; /* only allow bind/unbind via sysfs */

    const struct of_device_id *of_match_table;
    const struct acpi_device_id *acpi_match_table;

http://www.spinics.net/lists/kvm/msg97198.html
Race condition for unbound devices

$ echo 12ce0000.i2c > /sys/bus/platform/drivers/s3c-i2c/unbind

• problem: a hotplug event could cause rebind device to standard driver before vfio binds to the device
  - (mostly a problem for PCI)

• proposal: define new device flag that means ‘explicit bind only’
  $ echo 1 > /sys/devices/12ce0000.i2c/sysfs_bind_only
  $ echo 12ce0000.i2c > /sys/bus/platform/drivers/s3c-i2c/unbind
Example: DMA engine

```c

dma@101300 {
    cell-index = <0x1>;
    ranges = <0x0 0x101100 0x200>;
    reg = <0x101300 0x4>;
    compatible = "fsl,eloplus-dma";
    #size-cells = <0x1>;
    #address-cells = <0x1>;
    fsl,liodn = <0xc6>;
}

dma-channel@180 {
    interrupts = <0x23 0x2 0x0 0x0>;
    cell-index = <0x3>;
    reg = <0x180 0x80>;
    compatible = "fsl,eloplus-dma-channel";
};

dma-channel@100 {
    interrupts = <0x22 0x2 0x0 0x0>;
    cell-index = <0x2>;
    reg = <0x100 0x80>;
    compatible = "fsl,eloplus-dma-channel";
};

};

(modified for illustration purposes)
```
Dealing with Complex Devices

• For multi-node devices need a way to correlate vfio resources to device tree nodes

• RFC proposal is to extend VFIODEVICE_GET_REGION_INFO and VFIODEVICE_GET_IRQ_INFO with additional flags and some appended structs so user can do any needed correlation

• Example: VFIO_DEVTREE_REGION_INFO_FLAG_PATH

```c
struct vfio_devtree_info_path {
    u32 len;
    u8 path[];
}
```

http://www.spinics.net/lists/kvm/msg93593.html
Additional Issue: reset

- When a user space process exits, vfio expects to be able to reset a device.
- There is no standard way to do this for platform devices.
- Possible solution: device-specific reset logic in vfio somewhere
QEMU

- The other side of platform device pass-through is how QEMU exposes the platform device to user space
- No mechanism right now to dynamically add system devices and dynamically allocate IRQs. Work in progress.
- QEMU will have device specific drivers that have awareness of how to generate guest device tree nodes:

```
qemu-system-ppc
...  
-device vfio-fsl-dma,device=/sys/bus/platform/devices/ffe100300.dma
...  
```