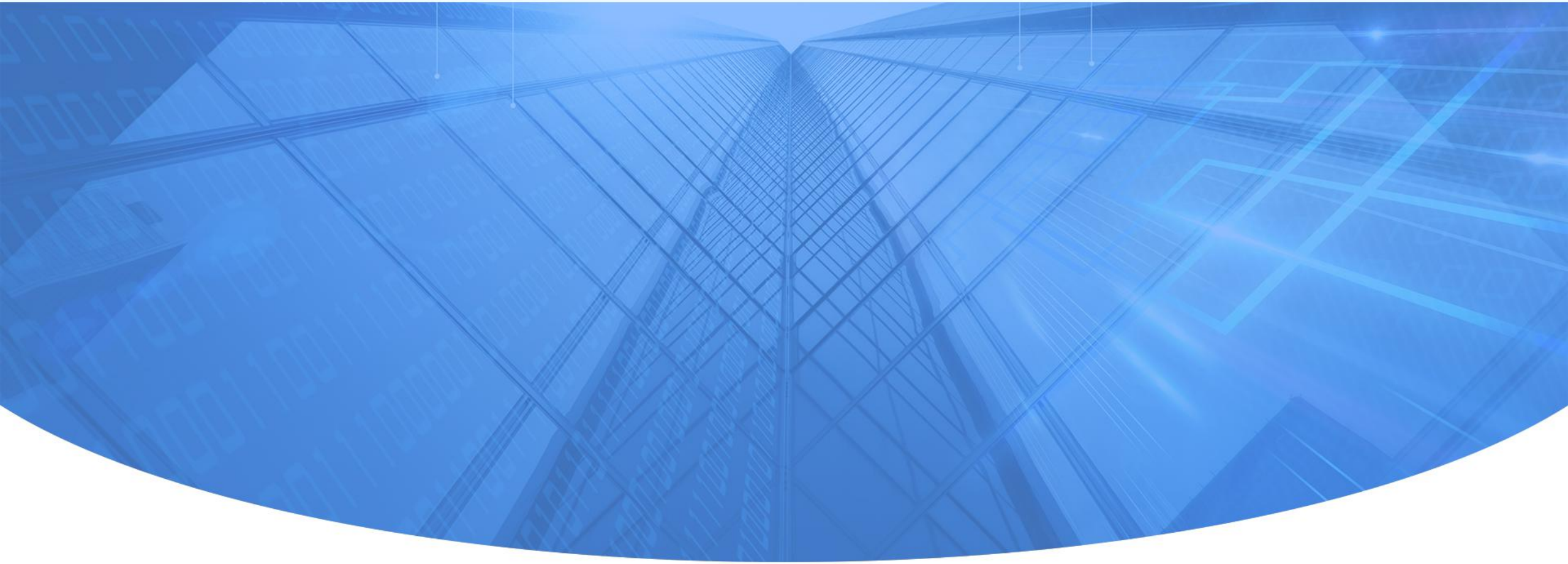

QEMU设备模拟支持：以I2C为例

PLCT实验室 李威威

2020/5/17

CONTENT

目录

QEMU QOM设备支持

- QOM，即QEMU Object Module，是QEMU提供的一套面向对象编程的模型
  - 几乎所有的设备如CPU、内存、总线、外设等都是利用这一面向对象的模型来实现的
  - 主要提供了ObjectClass、Object、TypeImpl等数据结构（基类）
    - MachineClass， BusClass， DeviceClass等
    - MachineState， BusState， DeviceState等

参考：《qemu对象模型——QOM实现分析》http://blog.chinaunix.net/uid-28541347-id-5784376.html

QEMU QOM设备支持——设备SPEC

Address	Register Name[3]	Description	Size
0x7E21 5000	AUX_IRQ	Auxiliary Interrupt status	3
0x7E21 5004	AUX_ENABLES	Auxiliary enables	3
0x7E21 5040	AUX_MU_IO_REG	Mini Uart I/O Data	8
0x7E21 5044	AUX_MU_IER_REG	Mini Uart Interrupt Enable	8
0x7E21 5048	AUX_MU_IIR_REG	Mini Uart Interrupt Identify	8
0x7E21 504C	AUX_MU_LCR_REG	Mini Uart Line Control	8
0x7E21 5050	AUX_MU_MCR_REG	Mini Uart Modem Control	8
0x7E21 5054	AUX_MU_LSR_REG	Mini Uart Line Status	8
0x7E21 5058	AUX_MU_MSR_REG	Mini Uart Modem Status	8
0x7E21 505C	AUX_MU_SCRATCH	Mini Uart Scratch	8

截自：《BCM2837 ARM Peripherals》

QEMU QOM设备支持——设备定义

定义设备所需维护的状态，总线/后端接口，中断等

```
typedef struct {
    /*< private >*/
    SysBusDevice parent_obj;
    /*< public >*/
    MemoryRegion iomem;

    CharBackend chr;
    qemu_irq irq;

    uint8_t read_fifo[BCM2835_AUX_RX_FIFO_LEN];
    uint8_t read_pos, read_count;
    uint8_t ier, iir;
} BCM2835AuxState;
```

QEMU QOM设备支持——设备操作

```
static void bcm2835_aux_write(void *opaque,
hwaddr offset, uint64_t value,  unsigned size)
{
    BCM2835AuxState *s = opaque;
    unsigned char ch;

    switch (offset) {
    ...
    case AUX_MU_IO_REG:
        ch = value;
        qemu_chr_fe_write_all(&s->chr, &ch, 1);
        break;
    …
}
```

```
static uint64_t bcm2835_aux_read(void *opaque,
hwaddr offset, unsigned size)
{
    BCM2835AuxState *s = opaque;
    uint32_t c, res;

    switch (offset) {
    ...
    case AUX_MU_IO_REG:
        c = s->read_fifo[s->read_pos];
         ....
        qemu_chr_fe_accept_input(&s->chr);
        bcm2835_aux_update(s);
        return c;
    …
}
```

如何跨设备操作？如对其它设备的访问控制或者影响其它设备的功能？

QEMU QOM设备支持——设备实例初始化

初始化设备IO地址空间注册回调函数，初始化中断

```
static void bcm2835_aux_init(Object *obj)
{
    SysBusDevice *sbd = SYS_BUS_DEVICE(obj);
    BCM2835AuxState *s = BCM2835_AUX(obj);
    memory_region_init_io(&s->iomem, OBJECT(s), &bcm2835_aux_ops, s,
                          TYPE_BCM2835_AUX, 0x100);
    sysbus_init_mmio(sbd, &s->iomem);
    sysbus_init_irq(sbd, &s->irq);
}
```

```
static const MemoryRegionOps bcm2835_aux_ops = {
    .read = bcm2835_aux_read,
    .write = bcm2835_aux_write,
    .endianness = DEVICE_NATIVE_ENDIAN,
    .valid.min_access_size = 4,
    .valid.max_access_size = 4,
};
```

QEMU QOM设备支持——设备Class初始化

初始化class相关接口，以及需要维护VMState，属性等

```
static void bcm2835_aux_realize(DeviceState *dev, Error **errp) {
    BCM2835AuxState *s = BCM2835_AUX(dev);
    qemu_chr_fe_set_handlers(&s->chr, bcm2835_aux_can_receive,
                             bcm2835_aux_receive, NULL, NULL, s, NULL, true);
}

static Property bcm2835_aux_props[] = {
    DEFINE_PROP_CHR("chardev", BCM2835AuxState, chr),
    DEFINE_PROP_END_OF_LIST(),
};

static void bcm2835_aux_class_init(ObjectClass *oc, void *data) {
    DeviceClass *dc = DEVICE_CLASS(oc);
    dc->realize = bcm2835_aux_realize;
    dc->vmsd = &vmstate_bcm2835_aux;
    set_bit(DEVICE_CATEGORY_INPUT, dc->categories);
    dc->props = bcm2835_aux_props;
}
```

QEMU QOM设备支持——设备注册

定义TypeInfo，注册设备（type_register_static + type_init ）

```
static const TypeInfo bcm2835_aux_info = {
    .name          = TYPE_BCM2835_AUX,
    .parent        = TYPE_SYS_BUS_DEVICE,
    .instance_size = sizeof(BCM2835AuxState),
    .instance_init = bcm2835_aux_init,
    .class_init    = bcm2835_aux_class_init,
};

static void bcm2835_aux_register_types(void)
{
    type_register_static(&bcm2835_aux_info);
}

type_init(bcm2835_aux_register_types)
```

class_init中无法输出调试信息？

QEMU QOM设备支持——设备创建

1）初始化OBJECT

```
object_initialize(&s->aux, sizeof(s->aux), TYPE_BCM2835_AUX);
object_property_add_child(obj, "aux", OBJECT(&s->aux), NULL);
qdev_set_parent_bus(DEVICE(&s->aux), sysbus_get_default());
```

2）设置属性，设置memory region，连接中断

```
qdev_prop_set_chr(DEVICE(&s->aux), "chardev", serial_hd(1));
object_property_set_bool(OBJECT(&s->aux), true, "realized", &err);

memory_region_add_subregion(&s->peri_mr, UART1_OFFSET,
        sysbus_mmio_get_region(SYS_BUS_DEVICE(&s->aux), 0));
sysbus_connect_irq(SYS_BUS_DEVICE(&s->aux), 0,
        qdev_get_gpio_in_named(DEVICE(&s->ic), BCM2835_IC_GPU_IRQ,
                INTERRUPT_AUX));
```

其它简化创建方法

```
static inline DeviceState *sysbus_create_simple(const char *name, hwaddr addr, qemu_irq irq)
DeviceState *sysbus_create_varargs(const char *name, hwaddr addr, ...);
```

QEMU I2C支持机制——I2C简介

- I2C是一种简单的双向二线制串行通信总线
  - 总线上设备分为主设备和从设备
  - 每个设备都有唯一的地址（7位或者10位）
  - 通信由主设备发起，通过设备地址来区分通信目标

QEMU I2C支持机制——数据结构

```
struct I2CBus {
    BusState qbus;
    QLIST_HEAD(, I2CNode) current_devs;
    uint8_t saved_address;
    bool broadcast;
};
```

```
struct I2CNode {
    I2CSlave *elt;
    QLIST_ENTRY(I2CNode) next;
};
```

```
struct I2CSlave {
    DeviceState qdev;

    /* Remaining fields for internal use by the I2C code.  */
    uint8_t address;
};
```

QEMU I2C支持机制——Master接口

```
I2CBus *i2c_init_bus(DeviceState *parent, const char *name);

int i2c_bus_busy(I2CBus *bus);

int i2c_start_transfer(I2CBus *bus, uint8_t address, int recv);
void i2c_end_transfer(I2CBus *bus);
void i2c_nack(I2CBus *bus);

int i2c_send_recv(I2CBus *bus, uint8_t *data, bool send);
int i2c_send(I2CBus *bus, uint8_t data);
int i2c_recv(I2CBus *bus);
```

QEMU I2C支持机制——Master操作流程

- 初始化i2c bus：i2c_init_bus → 设备初始化
- 数据传输流程：
  - 启动传输：i2c_start_transfer → 读写control寄存器的Start位
  - 传输数据：i2c_send/i2c_recv/i2c_send_recv → 读写fifo或者数据寄存器
  - 结束传输：i2c_end_transfer → 传输完成

QEMU I2C支持机制——Slave接口

```
DeviceState *i2c_create_slave(I2CBus *bus, const char *name, uint8_t addr);
void i2c_set_slave_address(I2CSlave *dev, uint8_t address);


typedef struct I2CSlaveClass {
    DeviceClass parent_class;

    int (*send)(I2CSlave *s, uint8_t data);

    int (*recv)(I2CSlave *s);

    int (*event)(I2CSlave *s, enum i2c_event event);

} I2CSlaveClass;
```

QEMU I2C支持机制——Slave设备模拟

```
static void smbus_device_class_init(ObjectClass *klass, void *data)
{
    I2CSlaveClass *sc = I2C_SLAVE_CLASS(klass);

    sc->event = smbus_i2c_event;
    sc->recv = smbus_i2c_recv;
    sc->send = smbus_i2c_send;
}

static const TypeInfo smbus_device_type_info = {
    .name = TYPE_SMBUS_DEVICE,
    .parent = TYPE_I2C_SLAVE,
    .instance_size = sizeof(SMBusDevice),
    .abstract = true,
    .class_size = sizeof(SMBusDeviceClass),
    .class_init = smbus_device_class_init,
};
```

HOST设备虚拟化

问题：**如何将qemu设备对接到host上的真实设备或资源？**

-serial stdio
-serial file：test
-drive file=test.img,format=raw,if=sd

I2C HOST设备虚拟化

方案一： 直接在i2c master模拟时与真实i2c通信

```
void bcm2835_i2c_setSlaveAddress (uint8_t addr)

void bcm2835_i2c_setClockDivider (uint16_t divider)

void bcm2835_i2c_set_baudrate (uint32_t baudrate)

uint8_t bcm2835_i2c_write (const char *buf, uint32_t len)

uint8_t bcm2835_i2c_read (char *buf, uint32_t len)
```

I2C HOST设备虚拟化

方案二： 沿用Slave接口，构建一个通用的slave设备

i2c_create_slave() → **bcm2835_i2c_setSlaveAddress**();
event() → ;
send() → **bcm2835_i2c_write**();
recv() → **bcm2835_i2c_read**();

必要时可以通过命令行将slave设备与host设备关联，如

-i2c-slave dev=/dev/i2c1

I2C HOST设备虚拟化

方案三： 利用chardev机制来实现一个i2c后端实现与真实设备的交互

```
typedef struct ChardevClass {
    ObjectClass parent_class;

    ...
    int (*chr_write)(Chardev *s, const uint8_t *buf, int len);
    int (*chr_sync_read)(Chardev *s, const uint8_t *buf, int len);
    int (*chr_ioctl)(Chardev *s, int cmd, void *arg);
    ...
} ChardevClass
```

谢谢

欢迎交流合作

liweiwei@iscas.ac.cn

2020/05/17