基于eBPF的设备驱动测试框架

华为 OS内核实验室

魏勇军

设计初衷

OpenEuler

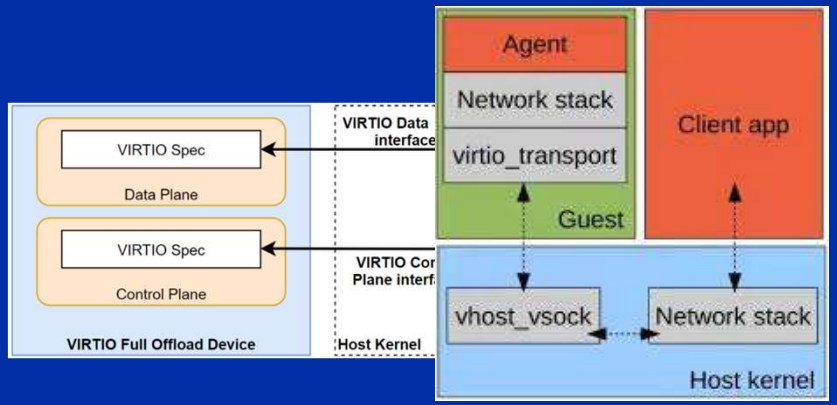
代码量	测试手段	小型化设备
驱动代码约~50% CVE/bugfix超半数	测试依赖物理硬件 静态检查发现问题	受资源限制， KASAN等检测手段 难部署

需要一种独立于物理硬件的驱动测试手段

现有方案

编写qemu虚拟设备驱动

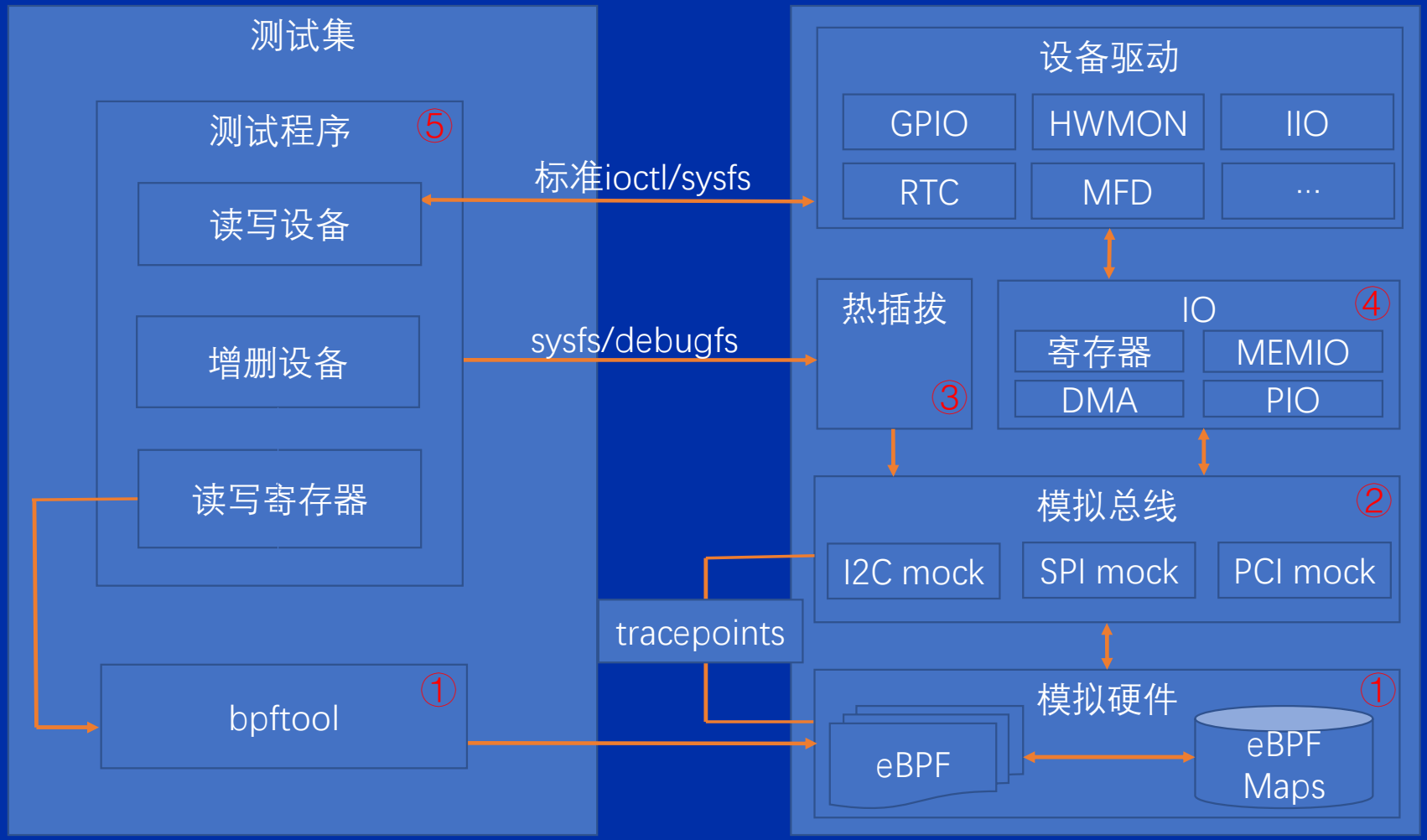
- 缺乏灵活性
- 实现设备模拟工作量大



roadtest demo

- 依赖virtio：依赖virtio标准
- UML下运行：功能缺失

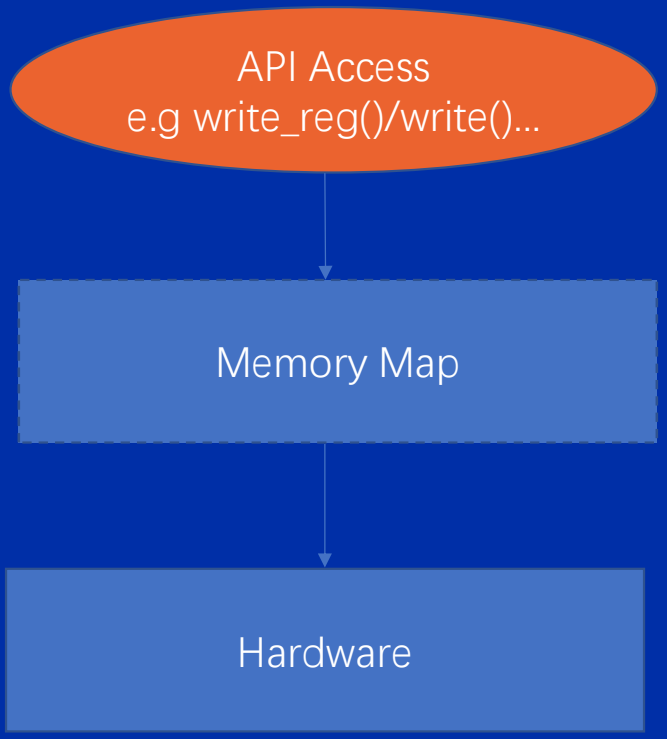
基于eBPF的测试框架



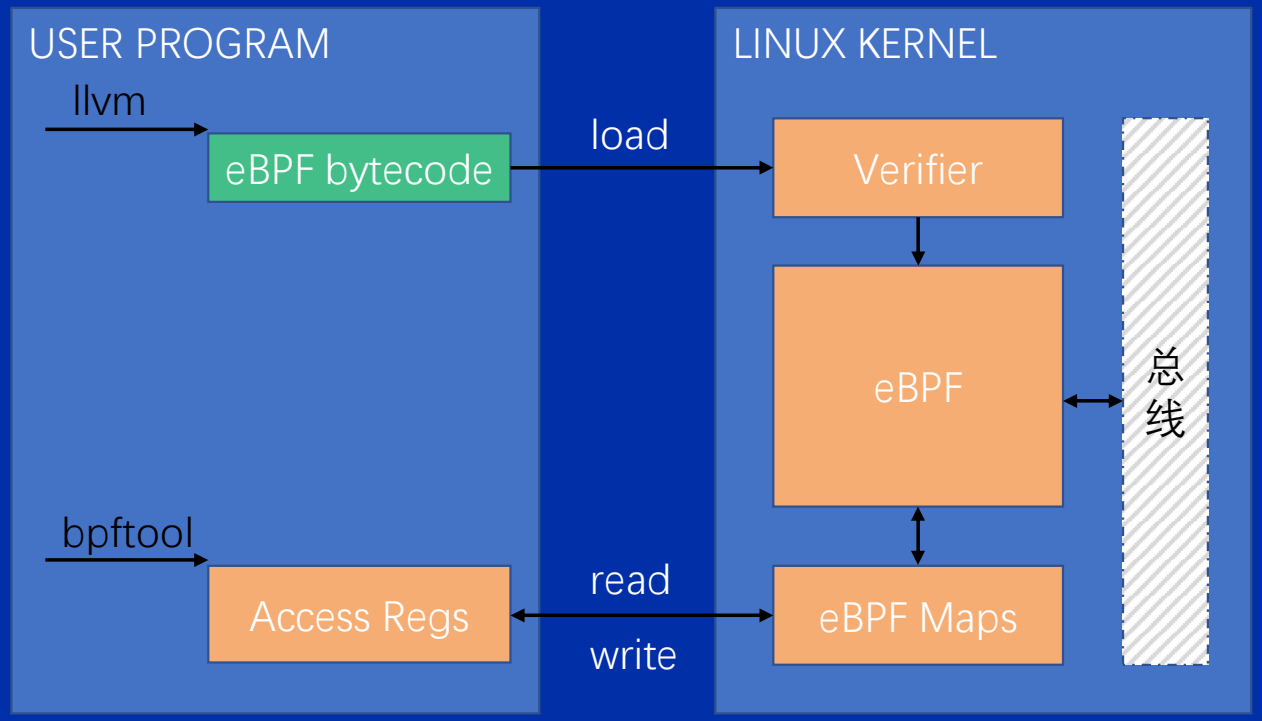
预备知识

- 设备驱动框架
- DTS
- tracepoint
- eBPF基础知识

模拟硬件



真实硬件访问模型



基于eBPF的虚拟硬件

eBPF TRACEPOINT机制

OpenEuler

DECLARE_TRACE_WRITABLE

提供功能：

- 提供基于TRACEPOINT的eBPF可写机制

使用限制：

- 只有第一个参数内存读写
- 读写固定长度内存，不支持变长内存

定义上下文数据

```
struct bpf_testmod_test_writable_ctx {
        bool early_ret;
        int val;
};
```

定义tracepoint

```
BPF_TESTMOD_DECLARE_TRACE(bpf_testmod_test_writable_bare,
        TP_PROTO(struct bpf_testmod_test_writable_ctx *ctx),
        TP_ARGS(ctx),
        sizeof(struct bpf_testmod_test_writable_ctx)
);
```

引入tracepoint

```
noinline ssize_t
bpf_testmod_test_read(struct file *file, struct kobject *kobj,
                      struct bin_attribute *bin_attr,
                      char *buf, loff_t off, size_t len)
{
        ...
        if (bpf_testmod_loop_test(101) > 100)
                trace_bpf_testmod_test_read(current, &ctx);

        /* Magic number to enable writable tp */
        if (len == 64) {
                struct bpf_testmod_test_writable_ctx writable = {
                        .val = 1024,
                };
                trace_bpf_testmod_test_writable_bare(&writable);
                if (writable.early_ret)
                        return snprintf(buf, len, "%d\n", writable.val);
        }

        return -EIO; /* always fail */
}
```

参考代码：

https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tools/testing/selftests/bpf/bpf_testmod/bpf_testmod.h
https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tools/testing/selftests/bpf/bpf_testmod/bpf_testmod-events.h
https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tools/testing/selftests/bpf/bpf_testmod/bpf_testmod.c

bpftool扩展

BPF程序

```
SEC("raw_tp.w/bpf_testmod_test_writable_bare")
int BPF_PROG(handle_raw_tp_writable_bare,
             struct bpf_testmod_test_writable_ctx *writable)
{
        raw_tp_writable_bare_in_val = writable->val;
        writable->early_ret = raw_tp_writable_bare_early_ret;
        writable->val = raw_tp_writable_bare_out_val;
        return 0;
}
```

eBPF tracepoint使用

- perf使用tracepoint+ebpf采集数据，perf退出卸载ebpf
- bpftool能load tracepoint prog, 但不能持久化（pin）

bpftool扩展

- bpftool prog命令添加autoattach功能，在load程序是pin到bpffs

BPF程序加载

$ bpftool prog load test.o /sys/fs/bpf/test autoattach

$ bpftool prog

510: raw_tracepoint_writable name handle_raw_tp_wri tag 2e13281b1f781bf3 gpl

loaded_at 2022-08-24T02:50:06+0000 uid 0

xlated 960B not jited memlock 4096B map_ids 439,437,440

btf_id 740

$ bpftool link

74: raw_tracepoint prog 510

tp 'bpf_testmod_test_writeable_bare'

[bpf-next,v9,1/3] bpftool: Add autoattach for bpf prog load|loadall - Patchwork (kernel.org)

模拟总线 - SPI总线设备

OpenEuler

SPI mockup总线驱动

注册platform驱动 ①

↓

注册SPI mockup控制器 ②

```
+static int spi_mockup_probe(struct platform_device *pdev)
+{
+	struct spi_master *master;
+	struct mockup_spi *mock;
+	int ret;
+
+	master = spi_alloc_master(&pdev->dev, sizeof(struct mockup_spi));
	...
+	master->dev.of_node = pdev->dev.of_node;
+	master->transfer_one_message = spi_mockup_transfer;

+	ret = devm_spi_register_controller(master);   ②
+	if (ret) {
+		spi_master_put(master);
+		return ret;
+	}
+
+	return 0;
+}

+static const struct of_device_id spi_mockup_match[] = {
+	{ .compatible = "spi-mockup", },
+	{ }
+};
+MODULE_DEVICE_TABLE(of, spi_mockup_match);
+
+static struct platform_driver spi_mockup_driver = {
+	.probe = spi_mockup_probe,
+	.driver = {
+		.name = "spi-mockup",
+		.of_match_table = spi_mockup_match,
+	},
+};
+module_platform_driver(spi_mockup_driver);   ①
```

[-next,0/4] spi: Introduce BPF based SPI mockup controller - Patchwork (linaro.org)

模拟总线 –读写SPI寄存器

OpenEuler

```
+++ b/include/linux/spi/spi-mockup.h
@@ -0,0 +1,13 @@
+/* SPDX-License-Identifier: GPL-2.0 */
+
+#ifndef __LINUX_SPI_MOCKUP_H
+#define __LINUX_SPI_MOCKUP_H
+
+#define SPI_BUFSIZ_MAX		0x1000
+
+struct spi_msg_ctx {
+	int ret;
+	__u8 data[SPI_BUFSIZ_MAX];
+};
+
+#endif
```

② eBPF程序数据结构

```
+DECLARE_TRACE_WRITABLE(spi_transfer_writeable,
+	TP_PROTO(struct spi_msg_ctx *msg, u8 chip_select, unsigned int len,
+		 u8 tx_nbits, u8 rx_nbits),
+	TP_ARGS(msg, chip_select, len, tx_nbits, rx_nbits),
+	sizeof(struct spi_msg_ctx)
+);
```

```
 static int spi_mockup_transfer(struct spi_controller *ctrl,
-			       struct spi_message *msg)
+			       struct spi_message *msg)
 {
-	msg->status = 0;
+	int ret = 0;
+
+	if (trace_spi_transfer_writeable_enabled())
+		ret = spi_mockup_transfer_writeable(msg);
+
+	msg->status = ret;
 	spi_finalize_current_message(ctrl);
-	return 0;
+	return ret;
 }
```

① SPI总线读写寄存器

```
+static int spi_mockup_transfer_writeable(struct spi_message *msg)
+{
+	struct spi_msg_ctx *ctx;
+	struct spi_transfer *t;
+	int ret = 0;
+
+	ctx = kmalloc(sizeof(*ctx), GFP_ATOMIC);
+	if (!ctx)
+		return -ENOMEM;
+
+	list_for_each_entry(t, &msg->transfers, transfer_list) {
+		memset(ctx, 0, sizeof(*ctx));
+		memcpy(ctx->data, t->tx_buf, t->len);
+		trace_spi_transfer_writeable(ctx, msg->spi->chip_select, t->len,
+					     t->tx_nbits, t->rx_nbits);
+		memcpy(t->rx_buf, ctx->data, t->len);
+	}
+
+	kfree(ctx);
+
+	return ret;
+}
```

③ eBPF程序挂载点

读写寄存器流程

序列化读/写请求

执行eBPF程序

反序列化读/写数据

模拟硬件 – eBPF程序

OpenEuler

```
SEC("raw_tp.w/spi_transfer_writeable")
int BPF_PROG(mtd_mchp23k256, struct spi_msg_ctx *msg, u8 chip,
             unsigned int len, u8 tx_nbits, u8 rx_nbits)
{
        int ret = 0;

        if (tx_nbits)
                ret = spi_transfer_write(msg, len);
        else if (rx_nbits)
                ret = spi_transfer_read(msg, len);

        return ret;
}

char LICENSE[] SEC("license") = "GPL";
```

① 程序挂载点：tracepoint
SPI总线读写寄存器时执行

```
#define MCHP23K256_CMD_WRITE_STATUS  0x01
#define MCHP23K256_CMD_WRITE         0x02
#define MCHP23K256_CMD_READ          0x03

#define CHIP_REGS_SIZE                      0x20000

#define MAX_CMD_SIZE                 4

struct {
        __uint(type, BPF_MAP_TYPE_ARRAY);
        __uint(max_entries, CHIP_REGS_SIZE);
        __type(key, __u32);
        __type(value, __u8);
} regs_mtd_mchp23k256 SEC(".maps");

static unsigned int chip_reg = 0;

static int spi_transfer_read(struct spi_msg_ctx *msg, unsigned int len)
{
        int i, key;
        u8 *reg;

        for (i = 0; i < len && i < sizeof(msg->data); i++) {
                key = i + chip_reg;

                reg = bpf_map_lookup_elem(&regs_mtd_mchp23k256, &key);
                if (!reg) {
                        bpf_printk("key %d not exists", key);
                        return -EINVAL;
                }

                msg->data[i] = *reg;
        }

        return 0;
}
```

② 使用map保存寄存器

③ 从map读取寄存器值

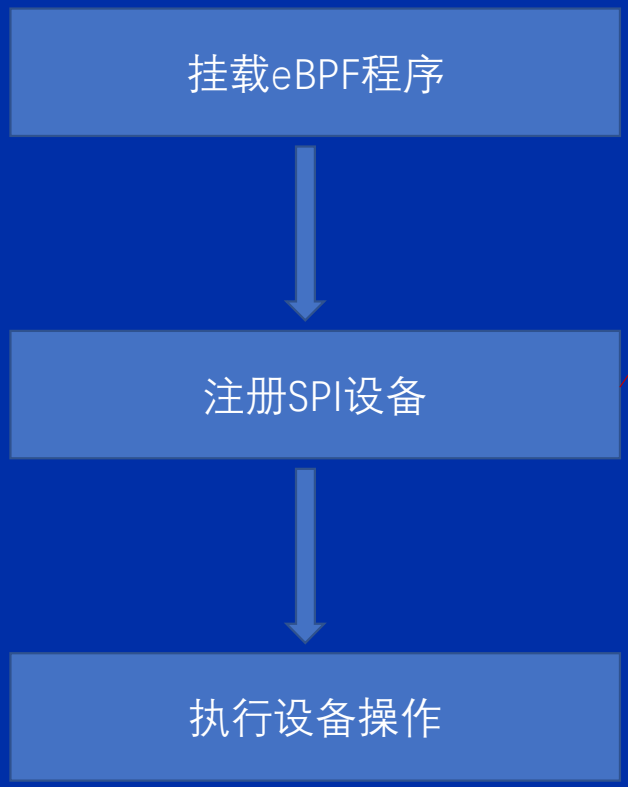
```
static int spi_transfer_write(struct spi_msg_ctx *msg, unsigned int len)
{
        u8 opcode = msg->data[0], value;
        int i, key;

        switch (opcode) {
        case MCHP23K256_CMD_READ:
        case MCHP23K256_CMD_WRITE:
                if (len < 2)
                        return -EINVAL;

                chip_reg = 0;
                for (i = 0; i < MAX_CMD_SIZE && i < len - 1; i++)
                        chip_reg = (chip_reg << 8) + msg->data[1 + i];

                return 0;
        case MCHP23K256_CMD_WRITE_STATUS:
                // ignore write status
                return 0;
        default:
                break;
        }

        for (i = 0; i < len && i < sizeof(msg->data); i++) {
                value = msg->data[i];
                key = chip_reg + i;

                if (bpf_map_update_elem(&regs_mtd_mchp23k256, &key, &value,
                                        BPF_EXIST)) {
                        bpf_printk("key %d not exists", key);
                        return -EINVAL;
                }
        }

        return 0;
}
```

④ 执行特殊command

⑤ 写寄存器值到map

[-next,4/4] spi: mockup: Add documentation - Patchwork (linaro.org)

模拟硬件 – 设备操作




```
$ bpftool prog load mtd-mchp23k256.o /sys/fs/bpf/test_prog autoattach
$ echo mchp23k256 0 > /sys/class/spi_master/spi0/new_device
$ ls /sys/bus/spi/devices/spi0.0/mtd/
  mtd0  mtd0ro
$ cat /sys/class/mtd/mtd0/name
 spi0.0
```

```
$ hexdump /dev/mtd0
0000000 0000 0000 0000 0000 0000 0000 0000 0000
*
0008000

$echo aaaa > /dev/mtd0

$ hexdump /dev/mtd0
0000000 6161 6161 000a 0000 0000 0000 0000 0000
0000010 0000 0000 0000 0000 0000 0000 0000 0000
*
0008000

$ bpftool map update name mtd_mchp23k256_ key 0 0 0 0 value 0

$ hexdump /dev/mtd0
0000000 6100 6161 000a 0000 0000 0000 0000 0000
0000010 0000 0000 0000 0000 0000 0000 0000 0000
*
0008000
```

读设备

写设备

写寄存器

模拟硬件 – 其他问题

直接读写eBPF map无法解决如下问题

寄存器编码问题

- 寄存器特定位用于标记读写
- 寄存器保留低N位用于特殊用途

单一eBPF程序对应一款芯片，eBPF后端工作量大，维护困难

IO故障注入

- 模拟寄存器读/写错误

解决方法

增加一个map用于保存配置信息，在测试程序进行配置。

```
#define BPF_CONF_REG_MASK       0x10
#define BPF_CONF_REG_SHIFT      0x11
#define BPF_CONF_FAULT          0x20

struct {
        __uint(type, BPF_MAP_TYPE_ARRAY);
        __uint(max_entries, CONF_REGS_SIZE);
        __type(key, __u32);
        __type(value, __u32);
} bpf_xfer_conf SEC(".maps");
```

寄存器mask, 默认0xff

寄存器shift, 默认0

是否注入故障

在读写寄存器前使用配置信息转义reg

添加删除设备

OpenEuler

DTS

```
/dts-v1/;

/ {
        clk24m: clk24m {
                compatible = "fixed-clock";
                clock-output-names = "clk24m";
                clock-frequency = <24000000>;
                #clock-cells = <0>;
        };
};
```

- 适合少量的固定设备，如SPI mockup总线设备等
- 无法创建测试集所需海量设备

设备ID

$ echo mchp23k256 0 > /sys/class/spi_master/spi0/new_device

$ echo 0 > /sys/class/spi_master/spi0/delete_device

- 使用sysfs接口创建/删除设备
- 使用Device Tree定义属性的设备无法使用

DTS Overlay

```
/dts-v1/;
/plugin/;

&{/spi} {
        spi-sram@0 {
                #address-cells = <1>;
                #size-cells = <1>;
                compatible = "microchip,mchp23k256";
                reg = <0>;
                spi-max-frequency = <20000000>;
        };
};
```

- 动态添加/删除设备

三种方式配合使用，适应不同场景

模拟中断处理

中断问题

- 模拟硬件设备需要中断处理
- 测试程序需要能控制中断触发

解决方式

- irq-sim: 提供基于worker的中断处理，无用户接口
- gpio-sim: 提供用户态控制接口，但无法触发设备中断
- irq-sim + gpio-sim：扩展支持中断控制器

[v2] allow gpio simulator be used as interrupt controller | Patchew

DTS配置：

```
gpio-sim {
        compatible = "gpio-simulator";

        irqchip0: irqchip0 {
                gpio-controller;
                #gpio-cells = <2>;
                ngpios = <16>;

                interrupt-controller;
                #interrupt-cells = <2>;
                #address-cells = <0>;

                irq0-hog {
                        gpio-hog;
                        gpios = <0 0>;
                        input;
                        line-name = "irq-sim0";
                };
};
```

触发中断：

$ echo pull-down > /sys/bus/gpio/devices/gpiochip0/sim_gpio0/pull
$ echo pull-up > /sys/bus/gpio/devices/gpiochip0/sim_gpio0/pull

测试框架

测试框架设计
- 基于python unittest

读写寄存器
- write_regs(reg, bytes)
- write_reg(reg, int)
- bytes =read_regs(reg)
- int = read_reg(reg)

触发中断
- trigger_irq()

IO故障注入
- trigger_io_fault()

```python
from kddv.core.ddunit import SPIDriverTest
from . import MTDDriver

MCHP23K256_TEST_DATA = bytes([0x78] * 16)

class TestMCHP23k256(SPIDriverTest, MTDDriver):
    name = "mchp23k256"

    @property
    def bpf(self):
        return f"mtd-{self.name}"

    @property
    def dts(self):
        return f"mtd-{self.name}"

    def test_read_data(self):
        with self.device() as dev:
            self.write_regs(0x00, MCHP23K256_TEST_DATA)
            data = self.device_read_bytes(dev, 16)
            self.assertEqual(data, MCHP23K256_TEST_DATA)

    def test_write_data(self):
        with self.device() as dev:
            self.write_regs(0x00, bytes([0] * 16))
            self.device_write_bytes(dev, MCHP23K256_TEST_DATA)
            data = self.read_regs(0x00, 16)
            self.assertEqual(bytes(data), MCHP23K256_TEST_DATA)
```

使用的bpf程序

使用的dts文件

通过bpf写寄存器

通过驱动程序读数据

通过驱动程序写数据

通过bpf读寄存器

后续计划

- PCI驱动支持
  - MEMIO
  - PIO
  - DMA
- 设备休眠唤醒

- 开源计划
  - bpftool扩展 已发送V9版
  - irq-sim & gpio-sim扩展 已发送V2版
  - SPI mockup总线设备驱动 已发送V1版
  - 驱动测试框架
  - I2C mockup总线设备驱动
  - 驱动测试集


OpenEuler