in drivers/uio/uio_pdrv_genirq.c

```
#ifdef CONFIG_OF
static struct of_device_id uio_of_genirq_match[] = {
        { /* This is filled with module_parm */ },
        { /* Sentinel */ },
};
MODULE_DEVICE_TABLE(of, uio_of_genirq_match);
module_param_string(of_id, uio_of_genirq_match[0].compatible, 128, 0);
MODULE_PARM_DESC(of_id, "Openfirmware id of the device to be handled by uio");
#endif
```

uio_of_genirq_match[2]的array是empty，没有任何用于match uio device的compatible string。

在pegmatitle.dtsi

```
        pip_irq@f9100000  {
                compatible = "generic-uio";
                id = <0>;
                reg = <0 0xf9100000 0 0x10000>;
                interrupt-parent = <&gic>;
                interrupts = < 0 206 4 >;
        };
```

当Gemstone2 board起来后，uio_pdrv_genirq driver找到了pip_irq device!

Why ?

uio_pdrv_genirq driver的作者有如下说明(https://github.com/Xilinx/linux-xlnx/commit/7ebd62dbc727ef343b07c01c852a15fc4d9cc9e5)

```
Revert "microblaze: UIO setup compatible property"
This reverts commit 3976b66.

This patch removes "generic-uio" compatible string from the driver.
This hack is in our tree for a long time and it is time
to fix it. For ensuring the same behaviour please add
uio_pdrv_genirq.of_id="generic-uio" to bootargs.

Feel free to use different compatible string for your purpose.

Signed-off-by: Michal Simek <michal.simek@xilinx.com>
```

---

u-boot传给Linux kernel的boot cmd如下

```
bootcmd: mmc dev 2;ext2load mmc 2:2 0x400000 /boot/uImage;ext2load mmc 2:2 0xf000
00 /boot/mv6270-ffc.dtb;setenv bootargs $bootargs root=/dev/mmcblk1p2 uio_pdrv_ge
nirq.of_id=generic-uio rootwait;bootm 0x400000 - 0xf00000
```

这里的uio_pdrv_genirq.of_id=generic-uio就作为uio_pdrv_genirq driver的module parameter.

uio_pdrv_genirq driver有如下module parameter定义

```
module_param_string(of_id, uio_of_genirq_match[0].compatible, 128, 0);
```

Kernel会把uio_pdrv_genirq.of_id=generic-uio

拆成key = uio_pdrv_genirq.of_id

value = generic-uio

然后把value赋值给uio_of_genirq_match[0].compatible,也就是uio_of_genirq_match[2]的第一个entry。这是在uio_pdrv_genirq driver初始化前做的，所以当kernel的driver framework在binding uio_pdrv_genirq与pip_irq device时，能够match。

下面是详细分析过程：

uio_pdrv_genirq.of_id module parameter的定义

```
module_param_string(of_id, uio_of_genirq_match[0].compatible, 128, 0);
```

```
#define module_param_string(name, string, len, perm)                    \
        static const struct kparam_string __param_string_##name         \
                = { len, string };                                       \
        __module_param_call(MODULE_PARAM_PREFIX, name,                   \
                            ¶m_ops_string,                               \
                            .str = &__param_string_##name, perm, -1, 0);\
        __MODULE_PARM_TYPE(name, "string")
```

==>

```
        static const struct kparam_string __param_string_of_id
                = { 128, uio_of_genirq_match[0].compatible };
                \                       ①
        __module_param_call(MODULE_PARAM_PREFIX, of_id,                  \
    ②
                            ¶m_ops_string,                               \
                            .str = &__param_string_of_id, perm, -1, 0);\
        __MODULE_PARM_TYPE(of_id, "uio_of_genirq_match[0].compatible")
```

①

```
/* Special one for strings we want to copy into */
struct kparam_string {
        unsigned int maxlen;
        char *string;
};
```

②

```
/* This is the fundamental function for registering boot/module
   parameters. */
#define __module_param_call(prefix, name, ops, arg, perm, level, flags) \
        /* Default value instead of permissions? */                     \
        static const char __param_str_##name[] = prefix #name; \
        static struct kernel_param __moduleparam_const __param_##name   \
        __used                                                          \
    __attribute__ ((unused,__section__ ("__param"),aligned(sizeof(void *)))) \
        = { __param_str_##name, ops, VERIFY_OCTAL_PERMISSIONS(perm),    \
            level, flags, { arg } }
```

即在section "__param"中填写一个struct kernel_param variable.

```
struct kernel_param {
        const char *name;
        const struct kernel_param_ops *ops;
        u16 perm;
        s8 level;
        u8 flags;
        union {
                void *arg;
                const struct kparam_string *str;
                const struct kparam_array *arr;
        };
};
```

```
struct kernel_param_ops param_ops_string = {
        .set = param_set_copystring,
        .get = param_get_string,
};
```

==>

```
{
    .name = of_id;
    .ops = param_ops_string;
    .perm = VERIFY_OCTAL_PERMISSIONS(0);
    .level = -1;
    .flags = 0;
    {
        .str = { 128, uio_of_genirq_match[0].compatible };
    }
}
```

in include/asm-gneric/vmlinux.lds.h

```
        /* Built-in module parameters. */                       \
        __param : AT(ADDR(__param) - LOAD_OFFSET) {              \
                VMLINUX_SYMBOL(__start___param) = .;            \
                *(__param)                                      \
                VMLINUX_SYMBOL(__stop___param) = .;             \
        }                                                       \
```

由__module_param_call macro定义的struct kernel_param variable被作为__start___param array的entry。

in init/main.c

```
pr_notice("Kernel command line: %s\n", boot_command_line);
parse_early_param();
after_dashes = parse_args("Booting kernel",
                  static_command_line, __start___param,
                  __stop___param - __start___param,
                  -1, -1, &unknown_bootoption);
```

这里的static_command_line来自于boot_command_line，见如下code

```
/*
 * We need to store the untouched command line for future reference.
 * We also need to store the touched command line since the parameter
 * parsing is performed in place, and we should allow a component to
 * store reference of name/value for future reference.
 */
static void __init setup_command_line(char *command_line)
{
saved_command_line=
                memblock_virt_alloc(strlen(boot_command_line) + 1, 0);
initcall_command_line=
                memblock_virt_alloc(strlen(boot_command_line) + 1, 0);
        static_command_line = memblock_virt_alloc(strlen(command_line) + 1, 0);
        strcpy(saved_command_line, boot_command_line);
        strcpy(static_command_line, command_line);
}
```

```c
/* Args looks like "foo=bar,bar2 baz=fuz wiz". */
char *parse_args(const char *doing,
                 char *args,
                 const struct kernel_param *params,
                 unsigned num,
                 s16 min_level,
                 s16 max_level,
                 int (*unknown)(char *param, char *val, const char *doing))
{
        char *param, *val;

        /* Chew leading spaces */
        args = skip_spaces(args);

        if (*args)
                pr_debug("doing %s, parsing ARGS: '%s'\n", doing, args);

        while (*args) {
                int ret;
                int irq_was_disabled;

                args = next_arg(args, &param, &val);
      ①
                /* Stop at -- */
                if (!val && strcmp(param, "--") == 0)
                        return args;
                irq_was_disabled = irqs_disabled();
                ret = parse_one(param, val, doing, params, num,          ②
                                min_level, max_level, unknown);
                if (irq_was_disabled && !irqs_disabled())
                        pr_warn("%s: option '%s' enabled irq's!\n",
                                doing, param);

                switch (ret) {
                case -ENOENT:
                        pr_err("%s: Unknown parameter `%s'\n", doing, param);
                        return ERR_PTR(ret);
                case -ENOSPC:
                        pr_err("%s: `%s' too large for parameter `%s'\n",
                               doing, val ?: "", param);
                        return ERR_PTR(ret);
                case 0:
                        break;
                default:
                        pr_err("%s: `%s' invalid for parameter `%s'\n",
                               doing, val ?: "", param);
                        return ERR_PTR(ret);
                }
        }

        /* All parsed OK. */
        return NULL;
}
```

①

提取bootcmd中的param = value

②

把param与__start___param array中的entry比较

```
1.  static int parse_one(char *param,                    ③
2.                      char *val,                                          ④
3.                      const char *doing,
4.                      const struct kernel_param *params,      ⑤
5.                      unsigned num_params,
6.                      s16 min_level,
7.                      s16 max_level,
8.                      int (*handle_unknown)(char *param, char *val,
9.                                            const char *doing))
10. {
11.         unsigned int i;
12.         int err;
13.
14.         /* Find parameter */
15.         for (i = 0; i < num_params; i++) {              ⑥
16.                 if (parameq(param, params[i].name)) {     ⑦
17.                         if (params[i].level < min_level
18.                             || params[i].level > max_level)
19.                                 return 0;
20.                         /* No one handled NULL, so do it here. */
21.                         if (!val &&
22.                             !(params[i].ops->flags & KERNEL_PARAM_OPS_FL_NOARG))
23.                                 return -EINVAL;
24.                         pr_debug("handling %s with %p\n", param,
25.                                 params[i].ops->set);
26.                         mutex_lock(&param_lock);
27.                         param_check_unsafe(&params[i]);
28.                         err = params[i].ops->set(val, &params[i]);      ⑧
29.                         mutex_unlock(&param_lock);
30.                         return err;
31.                 }
32.         }
33.
34.         if (handle_unknown) {
35.                 pr_debug("doing %s: %s='%s'\n", doing, param, val);
36.                 return handle_unknown(param, val, doing);
37.         }
38.
39.         pr_debug("Unknown argument '%s'\n", param);
40.         return -ENOENT;
41. }
42.
```

③

param = uio_pdrv_genirq.of_id

④

val = generic-uio

⑤

__start___param的struct kernel_param array

⑥

在__start___param array中查找kernel_param.name = uio_pdrv_genirq.of_id的entry

⑦

找到了

⑧

这里调用的是param_set_copystring()

in kernel/params.c

```
int param_set_copystring(const char *val, const struct kernel_param *kp)
{
        const struct kparam_string *kps = kp->str;

        if (strlen(val)+1 > kps->maxlen) {
                pr_err("%s: string doesn't fit in %u chars.\n",
                       kp->name, kps->maxlen-1);
                return -ENOSPC;
        }
        strcpy(kps->string, val);           ⑨
        return 0;
}
```

⑨

也就是把__start___param array中令

kernel_param.name = "uio_pdrv_genirq.of_id"

kernel_param.str.string = "generic-uio"

而这里的kernel_param.str.string == uio_of_genirq_match[0].compatible。

这样相当与

```
static struct of_device_id uio_of_genirq_match[] = {
        { /* This is filled with module_parm */ },
        { /* Sentinel */ },
};
```

==>

```
static struct of_device_id uio_of_genirq_match[] = {
    .compatible = "generic-uio";
};
```

费了半天劲，就为了这么简单的一行。

---

uio_pdrv_genirq driver的初始化

in drivers/uio/uio_pdrv_genirq.c

```
static struct platform_driver uio_pdrv_genirq = {
        .probe = uio_pdrv_genirq_probe,
        .remove = uio_pdrv_genirq_remove,
        .driver = {
                .name = DRIVER_NAME,
                .owner = THIS_MODULE,
                .pm = &uio_pdrv_genirq_dev_pm_ops,
                .of_match_table = of_match_ptr(uio_of_genirq_match),
        },
};

module_platform_driver(uio_pdrv_genirq);
```

in include/linux/platform_device.h

```
#define module_platform_driver(__platform_driver) \
        module_driver(__platform_driver, platform_driver_register, \
                      platform_driver_unregister)
```

==>

```
module_driver(uio_pdrv_genirq, platform_driver_register, platform_driver_unregister)
```

in include/linux/device.h

```
#define module_driver(__driver, __register, __unregister, ...) \
static int __init __driver##_init(void) \
{ \
        return __register(&(__driver) , ##__VA_ARGS__); \
} \
module_init(__driver##_init); \
static void __exit __driver##_exit(void) \
{ \in drivers/uio/uio_pdrv_genirq.c

static struct platform_driver uio_pdrv_genirq = {
        .probe = uio_pdrv_genirq_probe,
        .remove = uio_pdrv_genirq_remove,
        .driver = {
                .name = DRIVER_NAME,
                .owner = THIS_MODULE,
                .pm = &uio_pdrv_genirq_dev_pm_ops,
                .of_match_table = of_match_ptr(uio_of_genirq_match),
        },
};

module_platform_driver(uio_pdrv_genirq);

in include/linux/platform_device.h

#define module_platform_driver(__platform_driver) \
        module_driver(__platform_driver, platform_driver_register, \
                        platform_driver_unregister)

==>

module_driver(uio_pdrv_genirq, platform_driver_register, platform_driver_unregist
er)

in include/linux/device.h

#define module_driver(__driver, __register, __unregister, ...) \
static int __init __driver##_init(void) \
{ \
        return __register(&(__driver) , ##__VA_ARGS__); \
} \
module_init(__driver##_init); \
static void __exit __driver##_exit(void) \
{ \
        __unregister(&(__driver) , ##__VA_ARGS__); \
} \
module_exit(__driver##_exit);

==>

static int __init uio_pdrv_genirq_init(void)
{
    return platform_driver_register(&uio_pdrv_genirq);
}
```

```
module_init(uio_pdrv_genirq_init);
static void __exit uio_pdrv_genirq_exit(void)
{
    platform_driver_unregister(&uio_pdrv_genirq);
}
module_exit(uio_pdrv_genirq_exit);

由于在G2 LSP中uio_pdrv_genirq is embedded driver,即

#define module_init(x)  __initcall(x);
#define module_exit(x)  __exitcall(x);

而
#define __initcall(fn) device_initcall(fn)

又
#define device_initcall(fn)             __define_initcall(fn, 6)

即uio_pdrv_genirq_init()将在init level 6阶段执行。

in init/main.c

static void __init do_initcall_level(int level)
{
        initcall_t *fn;

        strcpy(initcall_command_line, saved_command_line);
        parse_args(initcall_level_names[level],
                   initcall_command_line, __start___param,
                   __stop___param - __start___param,
                   level, level,
                   &repair_env_string);

        for (fn = initcall_levels[level]; fn < initcall_levels[level+1]; fn++)
                do_one_initcall(*fn);
}
        __unregister(&(__driver) , ##__VA_ARGS__); \
} \
module_exit(__driver##_exit);
```

==>

```
static int __init uio_pdrv_genirq_init(void)
{
    return platform_driver_register(&uio_pdrv_genirq);
}
module_init(uio_pdrv_genirq_init);
static void __exit uio_pdrv_genirq_exit(void)
{
    platform_driver_unregister(&uio_pdrv_genirq);
}
module_exit(uio_pdrv_genirq_exit);
```

由于在G2 LSP中uio_pdrv_genirq is embedded driver,即

```
#define module_init(x)  __initcall(x);
#define module_exit(x)  __exitcall(x);
```

而

```
#define __initcall(fn) device_initcall(fn)
```

又

```
#define device_initcall(fn)             __define_initcall(fn, 6)
```

即uio_pdrv_genirq_init()将在init level 6阶段执行。

in init/main.c

```
static void __init do_initcall_level(int level)
{
        initcall_t *fn;

        strcpy(initcall_command_line, saved_command_line);
        parse_args(initcall_level_names[level],
                   initcall_command_line, __start___param,
                   __stop___param - __start___param,
                   level, level,
                   &repair_env_string);

        for (fn = initcall_levels[level]; fn < initcall_levels[level+1]; fn++)
                do_one_initcall(*fn);
}
```