```
        i2c3: i2c@d4033000 {
                pinctrl-0 = <&i2c4_pins>;
                pinctrl-names = "default";
                status = "okay";
                polytouch: edt-ft5x06@38 {
                        compatible = "edt,edt-ft5x06";
                        reg = <0x38>;
                        pinctrl-names = "default";
                        interrupt-parent = <&gpio0>;
                        interrupts = <35 0>;
                        num-x = <1024>;
                        num-y = <600>;
                        invert-y = <1>;
                        invert-x = <0>;
                        reset-gpios = <&gpio0 36 0>;
                };
        };
```

interrupt-parent = <&gpio0>;

interrupts = <35 0>;

in drivers/gpio/gpio-pxa.x

const struct irq_domain_ops pxa_irq_domain_ops = {

.map = pxa_irq_domain_map,

.xlate = irq_domain_xlate_twocell,

};

即对

interrupts = <35 0>;

2个参数指定一个gpio的中断

```
        gpio@d4019000 {
                compatible = "marvell,peg-gpio";
                #address-cells = <0x2>;
                #size-cells = <0x2>;
                reg = <0x0 0xd4019000 0x0 0x1000>;
                gpio-controller;
                #gpio-cells = <0x2>;
                interrupts = <0x0 0x24 0x4 0x0 0x77 0x4 0x0 0x78 0x4 0x0 0x79 0x4 0x0 0x7a 0x4 0x0 0x7b 0x4 0x0 0xdd 0x4 0x0 0xde 0x4>;
                interrupt-names = "gpio_mux";
                interrupt-controller;
                #interrupt-cells = <0x2>;

                ......

};
```

这里的

#interrupt-cells = <0x2>;

就决定了引用gpio interrupt的format,2个数字指定gpio interrupt。

对这2个数字解释的是irq_domain_xlate_twocell() function。

in kernel/irq/irqdomain.c

```
/**
 * irq_domain_xlate_twocell() - Generic xlate for direct two cell bindings
 *
 * Device Tree IRQ specifier translation function which works with two cell
 * bindings where the cell values map directly to the hwirq number
 * and linux irq flags.
 */
int irq_domain_xlate_twocell(struct irq_domain *d, struct device_node *ctrlr,
                        const u32 *intspec, unsigned int intsize,
                        irq_hw_number_t *out_hwirq, unsigned int *out_type)
{
        if (WARN_ON(intsize < 2))
                return -EINVAL;
        *out_hwirq = intspec[0];
        *out_type = intspec[1] & IRQ_TYPE_SENSE_MASK;
        return 0;
}
```

这里intspec[0] = 35,hardware interrupt number

intspec[1] = 0, 即触发interrupt的条件

```
 * IRQ_TYPE_NONE                - default, unspecified type
 * IRQ_TYPE_EDGE_RISING         - rising edge triggered
* IRQ_TYPE_EDGE_FALLING - falling edge triggered
 * IRQ_TYPE_EDGE_BOTH           - rising and falling edge triggered
 * IRQ_TYPE_LEVEL_HIGH          - high level triggered
 * IRQ_TYPE_LEVEL_LOW           - low level triggered
 * IRQ_TYPE_LEVEL_MASK          - Mask to filter out the level bits
 * IRQ_TYPE_SENSE_MASK          - Mask for all the above bits

        IRQ_TYPE_NONE           = 0x00000000,
        IRQ_TYPE_EDGE_RISING    = 0x00000001,
        IRQ_TYPE_EDGE_FALLING   = 0x00000002,
        IRQ_TYPE_EDGE_BOTH      = (IRQ_TYPE_EDGE_FALLING | IRQ_TYPE_EDGE_RISING),
        IRQ_TYPE_LEVEL_HIGH     = 0x00000004,
        IRQ_TYPE_LEVEL_LOW      = 0x00000008,
        IRQ_TYPE_LEVEL_MASK     = (IRQ_TYPE_LEVEL_LOW | IRQ_TYPE_LEVEL_HIGH),
        IRQ_TYPE_SENSE_MASK     = 0x0000000f,
        IRQ_TYPE_DEFAULT        = IRQ_TYPE_SENSE_MASK,
```

intspec[1] = IRQ_TYPE_NONE

什么意思?

到底是那种情况会触发gpio controller产生interrupt?

```
reset-gpios = <&gpio0 36 0>;
```

edt-ft5x06 使用<&gpio 26 0> pin作为reset该device的控制脚。当该pin被拉低时，edt-ft5x06被

reset。

&gpio是指向gpio@d4019000 device_node的phandle。

in drivers/input/touchscreen/edt-ft5x06.c

```
tsdata->reset_pin = of_get_named_gpio(np, "reset-gpios", 0);
```

np --> edt-ft5x06@38 device_node

```
                polytouch: edt-ft5x06@38 {
                        compatible = "edt,edt-ft5x06";
                        reg = <0x38>;
                        pinctrl-names = "default";
                        interrupt-parent = <&gpio0>;
                        interrupts = <35 0>;
                        num-x = <1024>;
                        num-y = <600>;
                        invert-y = <1>;
                        invert-x = <0>;
                        reset-gpios = <&gpio0 36 0>;
                };
```

```
static inline int of_get_named_gpio(struct device_node *np,
                                    const char *propname, int index)
{
        return of_get_named_gpio_flags(np, propname, index, NULL);
}
```

of_get_named_gpio_flags(np, "reset-gpios", 0, NULL);

in drivers/gpio/gpiolib-of.c

```
int of_get_named_gpio_flags(struct device_node *np, const char *list_name,
                            int index, enum of_gpio_flags *flags)
{
        struct gpio_desc *desc;

        desc = of_get_named_gpiod_flags(np, list_name, index, flags);

        if (IS_ERR(desc))
                return PTR_ERR(desc);
        else
                return desc_to_gpio(desc);
}
```

of_get_named_gpiod_flags(np, "reset-gpios", 0, NULL);

```c
/**
 * of_get_named_gpiod_flags() - Get a GPIO descriptor and flags for GPIO API
 * @np:         device node to get GPIO from
 * @propname:   property name containing gpio specifier(s)
 * @index:      index of the GPIO
 * @flags:      a flags pointer to fill in
 *
 * Returns GPIO descriptor to use with Linux GPIO API, or one of the errno
 * value on the error condition. If @flags is not NULL the function also fills
 * in flags for the GPIO.
 */
struct gpio_desc *of_get_named_gpiod_flags(struct device_node *np,
                     const char *propname, int index, enum of_gpio_flags *flags)
{
        /* Return -EPROBE_DEFER to support probe() functions to be called
         * later when the GPIO actually becomes available
         */
        struct gg_data gg_data = {
                .flags = flags,
                .out_gpio = ERR_PTR(-EPROBE_DEFER)
        };
        int ret;

        /* .of_xlate might decide to not fill in the flags, so clear it. */
        if (flags)
                *flags = 0;

        ret = of_parse_phandle_with_args(np, propname, "#gpio-cells", index,
                                         &gg_data.gpiospec);
        if (ret) {
                pr_debug("%s: can't parse '%s' property of node '%s[%d]'\n",
```

```
                        __func__, propname, np->full_name, index);
                return ERR_PTR(ret);
        }

        gpiochip_find(&gg_data, of_gpiochip_find_and_xlate);

        of_node_put(gg_data.gpiospec.np);
        pr_debug("%s: parsed '%s' property of node '%s[%d]' - status (%d)\n",
                 __func__, propname, np->full_name, index,
                 PTR_ERR_OR_ZERO(gg_data.out_gpio));
        return gg_data.out_gpio;
}
```

of_parse_phandle_with_args(np, "reset-gpios", "#gpio-cells",0, &gg_data.gpiospec);

对该function有如下comments:

```
/**
 * of_parse_phandle_with_args() - Find a node pointed by phandle in a list
 * @np:         pointer to a device tree node containing a list
 * @list_name:  property name that contains a list
 * @cells_name: property name that specifies phandles' arguments count
 * @index:      index of a phandle to parse out
 * @out_args:   optional pointer to output arguments structure (will be filled)
 *
 * This function is useful to parse lists of phandles and their arguments.
 * Returns 0 on success and fills out_args, on error returns appropriate
 * errno value.
 *
 * Caller is responsible to call of_node_put() on the returned out_args->node
 * pointer.
 *
 * Example:
 *
 * phandle1: node1 {
 *      #list-cells = <2>;
 * }
 *
 * phandle2: node2 {
 *      #list-cells = <1>;
 * }
 *
 * node3 {
 *      list = <&phandle1 1 2 &phandle2 3>;
 * }
 *
 * To get a device_node of the `node2' node you may call this:
 * of_parse_phandle_with_args(node3, "list", "#list-cells", 1, &args);
```

```
 */
```

```
int of_parse_phandle_with_args(const struct device_node *np, const char *list_name,
                               const char *cells_name, int index,
                               struct of_phandle_args *out_args)
```

这里list_name = "reset-gpios", cell_name = "#gpio-cells", index = 0

reset-gpios = <&gpio0 36 0>;

由np 的device_node中的"reset-gpios" property获得phandle，即指向gpio0 device_node,然后从gpio0 device_node中的"#gpio-cells" property获得指定某根gpio pin的specifier的format。即2个number，也就是&gpio0 后面的<36 0>.

drivers/of/base.c中的__of_parse_phandle_with_args() function即实现了上面的parsing.

```
1.  static int __of_parse_phandle_with_args(const struct device_node *np,
2.                                         const char *list_name,
3.                                         const char *cells_name,
4.                                         int cell_count, int index,
5.                                         struct of_phandle_args *out_args)
6.  {
7.          const __be32 *list, *list_end;
8.          int rc = 0, size, cur_index = 0;
9.          uint32_t count = 0;
10.         struct device_node *node = NULL;
11.         phandle phandle;
12.
13.         /* Retrieve the phandle list property */
14.         list = of_get_property(np, list_name, &size);            ①
15.         if (!list)
16.                 return -ENOENT;
17.         list_end = list + size / sizeof(*list);                          ②
18.
19.         /* Loop over the phandles until all the requested entry is found */
20.         while (list < list_end) {
    ③
21.                 rc = -EINVAL;
22.                 count = 0;
23.
24.                 /*
25.                  * If phandle is 0, then it is an empty entry with no
26.                  * arguments.  Skip forward to the next entry.
27.                  */
28.                 phandle = be32_to_cpup(list++);                                  ④
29.                 if (phandle) {
30.                         /*
```

```
31.                          * Find the provider node and parse the #*-cells
32.                          * property to determine the argument length.
33.                          *
34.                          * This is not needed if the cell count is hard-coded
35.                          * (i.e. cells_name not set, but cell_count is set),
36.                          * except when we're going to return the found node
37.                          * below.
38.                          */
39.                         if (cells_name || cur_index == index) {
40.                                 node = of_find_node_by_phandle(phandle);
                                    ⑤
41.                                 if (!node) {
42.                                         pr_err("%s: could not find phandle\n",
43.                                                 np->full_name);
44.                                         goto err;
45.                                 }
46.                         }
47.
48.                         if (cells_name) {
49.                                 if (of_property_read_u32(node, cells_name,
                                        ⑥
50.                                                          &count)) {
51.                                         pr_err("%s: could not get %s for %s\n",
52.                                                 np->full_name, cells_name,
53.                                                 node->full_name);
54.                                         goto err;
55.                                 }
56.                         } else {
57.                                 count = cell_count;
58.                         }
59.
60.                         /*
```

```
61.                          * Make sure that the arguments actually fit in the
62.                          * remaining property data length
63.                          */
64.                         if (list + count > list_end) {
                    ⑦
65.                                 pr_err("%s: arguments longer than property\n",
66.                                         np->full_name);
67.                                 goto err;
68.                         }
69.                 }
70.
71.                 /*
72.                  * All of the error cases above bail out of the loop, so at
73.                  * this point, the parsing is successful. If the requested
74.                  * index matches, then fill the out_args structure and return,
75.                  * or return -ENOENT for an empty entry.
76.                  */
77.                 rc = -ENOENT;
78.                 if (cur_index == index) {
    ⑧
79.                         if (!phandle)
80.                                 goto err;
81.
82.                         if (out_args) {
83.                                 int i;
84.                                 if (WARN_ON(count > MAX_PHANDLE_ARGS))
85.                                         count = MAX_PHANDLE_ARGS;
86.                                 out_args->np = node;
                ⑨
87.                                 out_args->args_count = count;
88.                                 for (i = 0; i < count; i++)
89.                                         out_args->args[i] = be32_to_cpup(list++);
90.                         } else {
```

```
                                of_node_put(node);
                        }

                        /* Found it! return success */
                        return 0;
                }

                of_node_put(node);
                node = NULL;
                list += count;
                cur_index++;
        }

        /*
         * Unlock node before returning result; will be one of:
         * -ENOENT : index is for empty phandle
         * -EINVAL : parsing error on data
         * [1..n]  : Number of phandle (count mode; when index = -1)
         */
        rc = index < 0 ? cur_index : -ENOENT;
 err:
        if (node)
                of_node_put(node);
        return rc;
}
```

①

list = of_get_property(np, list_name, &size);

np points to edt-ft5x06 device_node

list_name points to "reset-gpios"

该function 返回3个成员的数组。size = 3

list[0] = &gpio0, gpio0 devioce_node的phandle

list[1] = 36

list[2] = 0

②

list_end = list + size / sizeof(*list);

list_end points to list + 3，即数组的尾部

③

while (list < list_end) {

}

对list数组成员进行处理

④

phandle = be32_to_cpup(list++);

phandle = list[0] = &gpio0

⑤

node = of_find_node_by_phandle(phandle);

由&gpio0 phandle找到gpio@d4019000 device_node

node points to gpio@d4019000 device_node

```
        gpio@d4019000 {
                compatible = "marvell,peg-gpio";
                #address-cells = <0x2>;
                #size-cells = <0x2>;
                reg = <0x0 0xd4019000 0x0 0x1000>;
                gpio-controller;
                #gpio-cells = <0x2>;
                interrupts = <0x0 0x24 0x4 0x0 0x77 0x4 0x0 0x78 0x4 0x0 0x79 0x4 0x0 0x7a 0x4 0x0 0x7b 0x4 0x0 0xdd 0x4 0x0 0xde 0x4>;
                interrupt-names = "gpio_mux";
                interrupt-controller;
                #interrupt-cells = <0x2>;
                clocks = <0x2a>;
                ranges;

                gpio@d4019000 {
                        reg = <0x0 0xd4019000 0x0 0x4>;
                };

                gpio@d4019100 {
                        reg = <0x0 0xd4019100 0x0 0x4>;
                };

                gpio@d4019200 {
                        reg = <0x0 0xd4019200 0x0 0x4>;
                };

                gpio@d4019300 {
                        reg = <0x0 0xd4019300 0x0 0x4>;
                };

```

```
                gpio@d4019400 {
                        reg = <0x0 0xd4019400 0x0 0x4>;
                };

                gpio@d4019500 {
                        reg = <0x0 0xd4019500 0x0 0x4>;
                };

                gpio@d4019600 {
                        reg = <0x0 0xd4019600 0x0 0x4>;
                };

                gpio@d4019700 {
                        reg = <0x0 0xd4019700 0x0 0x4>;
                };
        };
```

⑥

```
                        if (of_property_read_u32(node, cells_name,
                                                 &count)) { }
```

node points to gpio@d4019000 device_node

cells_name points to "#gpio-cells"

The function makes count variable is equal to 2.

⑦

```
                        /*
                         * Make sure that the arguments actually fit in the
                         * remaining property data length
                         */
                        if (list + count > list_end) {
                                pr_err("%s: arguments longer than property\n",
                                         np->full_name);
                                goto err;
                        }
```

这时list = &list[1]。

list + 2 == list_end, valid

⑧

reset-gpios = <&gpio0 36 0>;

由于只有一根pin的指定，这里cur_index = index = 0

⑨

```
                                out_args->np = node;
                                out_args->args_count = count;
                                for (i = 0; i < count; i++)
                                        out_args->args[i] = be32_to_cpup(list++);
```

把parsing

reset-gpios = <&gpio0 36 0>

获得的信息通过struct of_phandle_args带回

out_args->np = node; 指向gpio0 device_node，这其实也是对"reset-gpios" property中第一个参数的parsing结果。

out_args->args_count = count; 后面是2个参数

```
for (i = 0; i < count; i++)
    out_args->args[i] = be32_to_cpup(list++);
```

out_args->args[0] = 36;

out_args->args[1] = 0;

---

in of_get_named_gpiod_flags(np, "reset-gpios", 0, NULL)

当

```
1.          ret = of_parse_phandle_with_args(np, propname, "#gpio-cells", index,
2.                                           &gg_data.gpiospec);
```

parsing result is saved in gg_data variable。

gpiochip_find(&gg_data, of_gpiochip_find_and_xlate);

进一步对<36 0>进行处理

```
1.  static int of_gpiochip_find_and_xlate(struct gpio_chip *gc, void *data)
2.  {
3.          struct gg_data *gg_data = data;
4.          int ret;
5.
6.          if ((gc->of_node != gg_data->gpiospec.np) ||
7.              (gc->of_gpio_n_cells != gg_data->gpiospec.args_count) ||        ①
8.              (!gc->of_xlate))
9.                  return false;
10.
11.         ret = gc->of_xlate(gc, &gg_data->gpiospec, gg_data->flags);        ②
12.         if (ret < 0) {
13.                 /* We've found a gpio chip, but the translation failed.
14.                  * Store translation error in out_gpio.
15.                  * Return false to keep looking, as more than one gpio chip
16.                  * could be registered per of-node.
17.                  */
18.                 gg_data->out_gpio = ERR_PTR(ret);
19.                 return false;
20.          }
21.
22.         gg_data->out_gpio = gpiochip_get_desc(gc, ret);
23.         return true;
24. }
```

①

reset-gpios = <&gpio0 36 0>中指定的gpio0 device_node要与gpio chip (controller)的device_node匹配

并且这里的gpio gpio pin specifier 为 2，也要与对应gpio chip中的gpio specifier相同。

②

in drivers/gpio/gpio-pxa.c

```
static int pxa_gpio_of_xlate(struct gpio_chip *gc,
                             const struct of_phandle_args *gpiospec,
                             u32 *flags)
{
        if (gpiospec->args[0] > pxa_last_gpio)
                return -EINVAL;

        if (gc != &pxa_gpio_chips[gpiospec->args[0] / 32].chip)
                return -EINVAL;

        if (flags)
                *flags = gpiospec->args[1];

        return gpiospec->args[0] % 32;
}
```

该function就是对<36 0>进行interpreting.

gpiospec->args[0] = 36

gpiospec->args[1] = 0

gpiospec->args[0] / 32 返回 gpio pin 36在gpio chip 1上

gpiospec->args[0] % 32 返回 gpio pin 36在gpio chip 1的hardware pin 4上。

gpio pin 36 is logical pin number.