Gemstone2 LSP在启动时在log中会report下面的错

```
----------------------------------------------------------

I2C: i2c-1: PXA I2C adapter

I2C: i2c-2: PXA I2C adapter

udevd[1252]: timeout: killing '/sbin/modprobe of:Ni2cT<NULL>Cmrvl,pxa-i2c' [1388]



udevd[1252]: '/sbin/modprobe of:Ni2cT<NULL>Cmrvl,pxa-i2c' [1388] terminated by signal 9 (Killed)



edt_ft5x06 3-0038: touchscreen probe failed

edt_ft5x06: probe of 3-0038 failed with error -121

I2C: i2c-3: PXA I2C adapter

I2C: i2c-4: PXA I2C adapter

I2C: i2c-5: PXA I2C adapter

----------------------------------------------------------
```

从log看是udevd报错的。udevd通过modprobe载入i2c-pxa.ko时报的错。

edt_ft5x06 touch screen device是挂在i2c-pxa bus上的slave device.

但edt_ft5x06 driver是builtin driver，而i2c-pxa bus driver是dynamic module(i2c-pxa.ko)。

edt_ft5x06 driver当然是依赖于i2c-pxa.ko的。

粗看好像有点问题。

builtin driver的初始化是在start_kernel()中进行的。

```
start_kernel()
       |
       |
      \|/
rest_init()
       |
       |
      \|/
kernel_init()
       |
       |
      \|/
kernel_init_freeable()
       |
       |
      \|/
do_basic_setup()
       |
       |
      \|/
do_initcalls()
       |
       |
```

\|/

do_initcall_level(level)

edt_ft5x06 driver在do_initcall_level(6)时运行。

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

```
static struct i2c_driver edt_ft5x06_ts_driver = {
        .driver = {
                .owner = THIS_MODULE,
                .name = "edt_ft5x06",
                .of_match_table = of_match_ptr(edt_ft5x06_of_match),
                .pm = &edt_ft5x06_ts_pm_ops,
        },
        .id_table = edt_ft5x06_ts_id,
        .probe    = edt_ft5x06_ts_probe,
        .remove   = edt_ft5x06_ts_remove,
};

module_i2c_driver(edt_ft5x06_ts_driver);
```

in include/linux/i2c.h

```
#define module_i2c_driver(__i2c_driver) \
    module_driver(__i2c_driver, i2c_add_driver, \
            i2c_del_driver)
```

==>

module_driver(edt_ft5x06_ts_driver, i2c_add_driver, i2c_del_driver) 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 int __init edt_ft5x06_ts_driver_init(void)
{
        return i2c_add_driver(&edt_ft5x06_ts_driver);
}
module_init(edt_ft5x06_ts_driver_init);
static void __exit edt_ft5x06_ts_driver_exit(void)
{
        i2c_del_driver(&edt_ft5x06_ts_driver);
}
module_exit(edt_ft5x06_ts_driver_exit);
```

对于builtin driver而言

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

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

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

当

```
i2c_add_driver(&edt_ft5x06_ts_driver);
```

运行时，i2c-pxa.ko都还根本没有载入(i2c-pxa.ko的载入是在kernel初始化完成后，启动udevd,由udevd daemon通过modprobe来载入的)。

edt_ft5x06 driver初始化时，它所依赖的i2c-pxa.ko还没有载入，这有点搞?!

从kernel启动的log看(添加"initcall_debug" kernel parameter)

```
calling  edt_ft5x06_ts_driver_init+0x0/0x10 @ 1

initcall edt_ft5x06_ts_driver_init+0x0/0x10 returned 0 after 37 usecs
```

在do_basic_setup()中edt_ft5x06_ts_driver_init被调用了，而且返回值是0，表示成功了!

但有一点可以肯定edt_ft5x06 driver的实质性初始化函数edt_ft5x06_ts_probe()肯定没被调用，因为要是调用肯定失败(i2x-pxa bus driver都还在SD card上，没有载入内存呢)

在其后的udevd的log中看到如下报错

```
I2C: i2c-1: PXA I2C adapter

I2C: i2c-2: PXA I2C adapter

hub 2-0:1.0: 1 port detected

mv-ehci d4292100.ehci: successful find EHCI device with regs 0xf0726140 irq 62 working in Host mode

initcall ehci_hcd_init+0x0/0xa0 [ehci_hcd] returned 0 after 778059 usecs

edt_ft5x06 3-0038: touchscreen probe failed

edt_ft5x06: probe of 3-0038 failed with error -121

I2C: i2c-3: PXA I2C adapter

I2C: i2c-4: PXA I2C adapter

I2C: i2c-5: PXA I2C adapter

initcall i2c_adap_pxa_init+0x0/0x14 [i2c_pxa] returned 0 after 30469058 usecs
```

显然udevd通过modprobe utility在载入i2c-pxa driver的过程中触发了edt_ft5x06_ts_probe() function的运行。而触发其运行的好像是i2c-pxa module的i2c_adap_pxa_init() function。由于edt_ft5x06_ts_probe()失败并耗费了相当长时间，所以i2c_adap_pxa_init()打印出的时间为30469058微妙，即30秒。

---

edt-ft5x06 driver载入analyse：

in drivers/i2c/i2c-core.c

```c
int i2c_register_driver(struct module *owner, struct i2c_driver *driver)
{
        int res;

        /* Can't register until after driver model init */
        if (unlikely(WARN_ON(!i2c_bus_type.p)))
                return -EAGAIN;

        /* add the driver to the list of i2c drivers in the driver core */
        driver->driver.owner = owner;
        driver->driver.bus = &i2c_bus_type;            ①

        /* When registration returns, the driver core
         * will have called probe() for all matching-but-unbound devices.
         */
        res = driver_register(&driver->driver);        ②
        if (res)
                return res;

        /* Drivers should switch to dev_pm_ops instead. */
        if (driver->suspend)
                pr_warn("i2c-core: driver [%s] using legacy suspend method\n",
                        driver->driver.name);
        if (driver->resume)
                pr_warn("i2c-core: driver [%s] using legacy resume method\n",
                        driver->driver.name);

        pr_debug("i2c-core: driver [%s] registered\n", driver->driver.name);

        INIT_LIST_HEAD(&driver->clients);
        /* Walk the adapters that are already present */
        i2c_for_each_dev(driver, __process_new_driver);

        return 0;
}
```

①指定edt-ft5x06是挂在i2c bus上的。

②driver_register()本质上就是要把edt-ft5x06 driver挂到i2c bus的driver list上。

in drivers/base/driver.c

```
int driver_register(struct device_driver *drv)
{
        int ret;
        struct device_driver *other;

        BUG_ON(!drv->bus->p);

        if ((drv->bus->probe && drv->probe) ||
            (drv->bus->remove && drv->remove) ||
            (drv->bus->shutdown && drv->shutdown))
                printk(KERN_WARNING "Driver '%s' needs updating - please use "
                        "bus_type methods\n", drv->name);

        other = driver_find(drv->name, drv->bus);
        if (other) {
                printk(KERN_ERR "Error: Driver '%s' is already registered, "
                        "aborting...\n", drv->name);
                return -EBUSY;
        }

        ret =bus_add_driver(drv);
        if (ret)
                return ret;
        ret = driver_add_groups(drv, drv->groups);
        if (ret) {
                bus_remove_driver(drv);
                return ret;
        }
        kobject_uevent(&drv->p->kobj, KOBJ_ADD);

        return ret;
}
```

in drivers/base/bus.c

```
/**
 * bus_add_driver - Add a driver to the bus.
 * @drv: driver.
 */
```

```
int bus_add_driver(struct device_driver *drv)
{
        struct bus_type *bus;
        struct driver_private *priv;
        int error = 0;

        bus = bus_get(drv->bus);
        if (!bus)
                return -EINVAL;

        pr_debug("bus: '%s': add driver %s\n", bus->name, drv->name);

        priv = kzalloc(sizeof(*priv), GFP_KERNEL);
        if (!priv) {
                error = -ENOMEM;
                goto out_put_bus;
        }
        klist_init(&priv->klist_devices, NULL, NULL);
        priv->driver = drv;
        drv->p = priv;
        priv->kobj.kset = bus->p->drivers_kset;
        error = kobject_init_and_add(&priv->kobj, &driver_ktype, NULL,
                                     "%s", drv->name);
        if (error)
                goto out_unregister;

        klist_add_tail(&priv->knode_bus, &bus->p->klist_drivers);
        if (drv->bus->p->drivers_autoprobe) {                ③
                error =driver_attach(drv);
  ④
                if (error)
                        goto out_unregister;
        }
        module_add_driver(drv->owner, drv);

        error = driver_create_file(drv, &driver_attr_uevent);
        if (error) {
                printk(KERN_ERR "%s: uevent attr (%s) failed\n",
                        __func__, drv->name);
        }
        error = driver_add_groups(drv, bus->drv_groups);
        if (error) {
                /* How the hell do we get out of this pickle? Give up */
                printk(KERN_ERR "%s: driver_create_groups(%s) failed\n",
                        __func__, drv->name);
        }

        if (!drv->suppress_bind_attrs) {
                error = add_bind_files(drv);
                if (error) {
                        /* Ditto */
                        printk(KERN_ERR "%s: add_bind_files(%s) failed\n",
                                __func__, drv->name);
                }
        }

        return 0;

out_unregister:
        kobject_put(&priv->kobj);
        kfree(drv->p);
        drv->p = NULL;
out_put_bus:
        bus_put(bus);
        return error;
}
```

③ drv->bus->p->drivers_autoprobe is 1. drv->bus = i2c bus

in i2c_register_driver()

driver->driver.bus = &i2c_bus_type;

The initialization of i2c_bus_type

in i2c-core.c

```c
struct bus_type i2c_bus_type = {
        .name           = "i2c",
        .match          = i2c_device_match,
        .probe          = i2c_device_probe,
        .remove         = i2c_device_remove,
        .shutdown       = i2c_device_shutdown,
        .pm             = &i2c_device_pm_ops,
};

static int __init i2c_init(void)
{
        int retval;

        retval =bus_register(&i2c_bus_type);
        if (retval)
                return retval;
#ifdef CONFIG_I2C_COMPAT
        i2c_adapter_compat_class = class_compat_register("i2c-adapter");
        if (!i2c_adapter_compat_class) {
                retval = -ENOMEM;
                goto bus_err;
        }
#endif
        retval = i2c_add_driver(&dummy_driver);
        if (retval)
                goto class_err;
        return 0;

class_err:
#ifdef CONFIG_I2C_COMPAT
        class_compat_unregister(i2c_adapter_compat_class);
bus_err:
#endif
        bus_unregister(&i2c_bus_type);
        return retval;
}

/**
 * bus_register - register a driver-core subsystem
 * @bus: bus to register
 *
 * Once we have that, we register the bus with the kobject
 * infrastructure, then register the children subsystems it has:
 * the devices and drivers that belong to the subsystem.
 */
int bus_register(struct bus_type *bus)
{
        int retval;
        struct subsys_private *priv;
        struct lock_class_key *key = &bus->lock_key;

        priv = kzalloc(sizeof(struct subsys_private), GFP_KERNEL);
        if (!priv)
```

```
                return -ENOMEM;

        priv->bus = bus;
        bus->p = priv;

        BLOCKING_INIT_NOTIFIER_HEAD(&priv->bus_notifier);

        retval = kobject_set_name(&priv->subsys.kobj, "%s", bus->name);
        if (retval)
                goto out;

        priv->subsys.kobj.kset = bus_kset;
        priv->subsys.kobj.ktype = &bus_ktype;
        priv->drivers_autoprobe = 1;                                     （A）

        retval = kset_register(&priv->subsys);
        if (retval)
                goto out;

        retval = bus_create_file(bus, &bus_attr_uevent);
        if (retval)
                goto bus_uevent_fail;

        priv->devices_kset = kset_create_and_add("devices", NULL,
                                                 &priv->subsys.kobj);
        if (!priv->devices_kset) {
                retval = -ENOMEM;
                goto bus_devices_fail;
        }

        priv->drivers_kset = kset_create_and_add("drivers", NULL,
                                                 &priv->subsys.kobj);
        if (!priv->drivers_kset) {
                retval = -ENOMEM;
                goto bus_drivers_fail;
        }

        INIT_LIST_HEAD(&priv->interfaces);
        __mutex_init(&priv->mutex, "subsys mutex", key);
        klist_init(&priv->klist_devices, klist_devices_get, klist_devices_put);
        klist_init(&priv->klist_drivers, NULL, NULL);

        retval = add_probe_files(bus);
        if (retval)
                goto bus_probe_files_fail;

        retval = bus_add_groups(bus, bus->bus_groups);
        if (retval)
                goto bus_groups_fail;

        pr_debug("bus: '%s': registered\n", bus->name);
        return 0;

bus_groups_fail:
        remove_probe_files(bus);
bus_probe_files_fail:
        kset_unregister(bus->p->drivers_kset);
bus_drivers_fail:
        kset_unregister(bus->p->devices_kset);
bus_devices_fail:
        bus_remove_file(bus, &bus_attr_uevent);
bus_uevent_fail:
        kset_unregister(&bus->p->subsys);
out:
        kfree(bus->p);
        bus->p = NULL;
        return retval;
}
```

(A)

initialize priv->drivers_autoprobe to 1.

④

```
    if (drv->bus->p->drivers_autoprobe) {
        error = driver_attach(drv);
        if (error)
            goto out_unregister;
    }
```

in drivers/base/dd.c

```
int driver_attach(struct device_driver *drv)
{
        return bus_for_each_dev(drv->bus, NULL, drv, __driver_attach);
}
```

drv --> edt-ft5x06 device driver

drv->bus --> i2c bus

bus_for_each_dev(), enumerate all devices attached to i2c bus and make drv to identify them.

```
foreach(device attached on i2c bus)
        __driver_attach(dev, drv);
```

```c
/**
 * bus_for_each_dev - device iterator.
 * @bus: bus type.
 * @start: device to start iterating from.
 * @data: data for the callback.
 * @fn: function to be called for each device.
 *
 * Iterate over @bus's list of devices, and call @fn for each,
 * passing it @data. If @start is not NULL, we use that device to
 * begin iterating from.
 *
 * We check the return of @fn each time. If it returns anything
 * other than 0, we break out and return that value.
 *
 * NOTE: The device that returns a non-zero value is not retained
 * in any way, nor is its refcount incremented. If the caller needs
 * to retain this data, it should do so, and increment the reference
 * count in the supplied callback.
 */
int bus_for_each_dev(struct bus_type *bus, struct device *start,
                     void *data, int (*fn)(struct device *, void *))
{
        struct klist_iter i;
        struct device *dev;
        int error = 0;

        if (!bus || !bus->p)
                return -EINVAL;

        klist_iter_init_node(&bus->p->klist_devices, &i,
                             (start ? &start->p->knode_bus : NULL));
        while ((dev = next_device(&i)) && !error)
                error = fn(dev, data);
        klist_iter_exit(&i);
        return error;
}
```

When edt-ft5x06 driver initialize, bus->p->klist_devices is empty ! **Why ???**

所以在initcall level 6,edt-ft5x06 driver的初始化会成功。edt-ft5x06 driver根本无edt-ft5x06 device来probe!

在这里edt-ft5x06 driver只是添加到i2c bus的driver list上而已。

在mv6220-toc.dts中

```
pxai2c4: i2c@d4033000 {
    pinctrl-0 = <&i2c1_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>;
    };
};
```

这里只是建立了edt-ft5x06@38 device是i2c@d4033000 device的child，具体的i2c bus之间bus - client之间的关系不是能表达的。

in drivers/i2c/i2c-core.c

```c
static int i2c_register_adapter(struct i2c_adapter *adap)
{
        ......

exit_recovery:
        /* create pre-declared device nodes */
of_i2c_register_devices(adap);
        acpi_i2c_register_devices(adap);
        acpi_i2c_install_space_handler(adap);

        if (adap->nr < __i2c_first_dynamic_bus_num)
                i2c_scan_static_board_info(adap);

        /* Notify drivers */
        mutex_lock(&core_lock);
        bus_for_each_drv(&i2c_bus_type, NULL, adap, __process_new_adapter);
        mutex_unlock(&core_lock);

        return 0;

out_list:
        mutex_lock(&core_lock);
        idr_remove(&i2c_adapter_idr, adap->nr);
        mutex_unlock(&core_lock);
        return res;
}
```

i2c_register_adapter()用于向i2c framework注册i2c bus.

```
#if IS_ENABLED(CONFIG_OF)
static void of_i2c_register_devices(struct i2c_adapter *adap)
{
        void *result;
        struct device_node *node;

        /* Only register child devices if the adapter has a node pointer set */
        if (!adap->dev.of_node)
                return;

        dev_dbg(&adap->dev, "of_i2c: walking child nodes\n");

        for_each_available_child_of_node(adap->dev.of_node, node) {
                        (A)
                struct i2c_board_info info = {};
                struct dev_archdata dev_ad = {};
                const __be32 *addr;
                int len;

                dev_dbg(&adap->dev, "of_i2c: register %s\n", node->full_name);
                     (B)

                if (of_modalias_node(node, info.type, sizeof(info.type)) < 0) {
                          (C)
                        dev_err(&adap->dev, "of_i2c: modalias failure on %s\n",
                                node->full_name);
                        continue;
                }

                addr = of_get_property(node, "reg", &len);
                                                                     (D)
                if (!addr || (len < sizeof(int))) {
                        dev_err(&adap->dev, "of_i2c: invalid reg on %s\n",
                                node->full_name);
                        continue;
                }

                info.addr = be32_to_cpup(addr);
                if (info.addr > (1 << 10) - 1) {
                        dev_err(&adap->dev, "of_i2c: invalid addr=%x on %s\n",
                                info.addr, node->full_name);
                        continue;
                }

                info.irq = irq_of_parse_and_map(node, 0);
                                                                        (E)
                info.of_node = of_node_get(node);
                info.archdata = &dev_ad;

                if (of_get_property(node, "wakeup-source", NULL))
                        info.flags |= I2C_CLIENT_WAKE;

                request_module("%s%s", I2C_MODULE_PREFIX, info.type);
```

```
                                （F）

                result = i2c_new_device(adap, &info);

   (G)
                if (result == NULL) {
                        dev_err(&adap->dev, "of_i2c: Failure registering %s\n",
                                node->full_name);
                        of_node_put(node);
                        irq_dispose_mapping(info.irq);
                        continue;
                }
        }
}
```

(A)

enumerate i2c@d4033000的child device node

(B)

这里node->full_name应该为"/i2c@d4033000/edt-ft5x06@38"

(C)

这里info.type = "edt-ft5x06"。从compatible = "edt,edt-ft5x06";提取出的。

(D)

得到edt-ft5x06@38 device的地址：0x38

(E)

对

  interrupt-parent = <&gpio0>;

  interrupts = <35 0>;

的解释，获得virtual irq number。

(F)

request_module("i2c:edt-ft5x06");

(G)

i2c_new_device() function才会真正触发i2c client device 与client driver之间的match与probe action。

```c
/**
 * i2c_new_device - instantiate an i2c device
 * @adap: the adapter managing the device
 * @info: describes one I2C device; bus_num is ignored
 * Context: can sleep
 *
 * Create an i2c device. Binding is handled through driver model
 * probe()/remove() methods.  A driver may be bound to this device when we
 * return from this function, or any later moment (e.g. maybe hotplugging will
 * load the driver module).  This call is not appropriate for use by mainboard
 * initialization logic, which usually runs during an arch_initcall() long
 * before any i2c_adapter could exist.
 *
 * This returns the new i2c client, which may be saved for later use with
 * i2c_unregister_device(); or NULL to indicate an error.
 */
struct i2c_client *
i2c_new_device(struct i2c_adapter *adap, struct i2c_board_info const *info)
{
        struct i2c_client       *client;
        int                     status;

        client = kzalloc(sizeof *client, GFP_KERNEL);
        if (!client)
                return NULL;

        client->adapter = adap;

        client->dev.platform_data = info->platform_data;

        if (info->archdata)
                client->dev.archdata = *info->archdata;

        client->flags = info->flags;
        client->addr = info->addr;
        client->irq = info->irq;

        strlcpy(client->name, info->type, sizeof(client->name));

        /* Check for address validity */
        status = i2c_check_client_addr_validity(client);
        if (status) {
                dev_err(&adap->dev, "Invalid %d-bit I2C address 0x%02hx\n",
                        client->flags & I2C_CLIENT_TEN ? 10 : 7, client->addr);
                goto out_err_silent;
        }

        /* Check for address business */
        status = i2c_check_addr_busy(adap, client->addr);
        if (status)
                goto out_err;

        client->dev.parent = &client->adapter->dev;
        client->dev.bus = &i2c_bus_type;
        client->dev.type = &i2c_client_type;
        client->dev.of_node = info->of_node;
        ACPI_COMPANION_SET(&client->dev, info->acpi_node.companion);

        i2c_dev_set_name(adap, client);
        status =device_register(&client->dev);

    (H)
        if (status)
                goto out_err;

        dev_dbg(&adap->dev, "client [%s] registered with bus id %s\n",
                client->name, dev_name(&client->dev));

        return client;

out_err:
        dev_err(&adap->dev, "Failed to register i2c client %s at 0x%02x "
                "(%d)\n", client->name, client->addr, status);
out_err_silent:
        kfree(client);
        return NULL;
}
```

(H)

这才会真正trigger edt_ft5x06_ts_probe()的运行。edt-ft5x06 driver早在initcall level 6期间就已经被添加到i2c bus的driver list中，由于那时i2c bus的device list为空，所以什么都没发生。而这里的device_register()会把新创建的i2c client device添加到i2c bus的device list中。在添加时，该device会让i2c bus上的drivers来match，match成功后就是紧接着的probe action.

in drivers/i2c/busses/i2c-pxa.c

i2c-pxa i2c bus driver的probe() function。在G2 LSP中i2c-pxa i2c bus driver是动态载入的。

```c
static int i2c_pxa_probe(struct platform_device *dev)
{
        struct i2c_pxa_platform_data *plat = dev_get_platdata(&dev->dev);
        enum pxa_i2c_types i2c_type;
        struct pxa_i2c *i2c;
        struct resource *res = NULL;
        int ret, irq;

        i2c = kzalloc(sizeof(struct pxa_i2c), GFP_KERNEL);
        if (!i2c) {
                ret = -ENOMEM;
                goto emalloc;
        }

        /* Default adapter num to device id; i2c_pxa_probe_dt can override. */
        i2c->adap.nr = dev->id;

        ret = i2c_pxa_probe_dt(dev, i2c, &i2c_type);
        if (ret > 0)
                ret = i2c_pxa_probe_pdata(dev, i2c, &i2c_type);
        if (ret < 0)
                goto eclk;

        res = platform_get_resource(dev, IORESOURCE_MEM, 0);
        irq = platform_get_irq(dev, 0);
        if (res == NULL || irq < 0) {
                ret = -ENODEV;
                goto eclk;
        }

        if (!request_mem_region(res->start, resource_size(res), res->name)) {
                ret = -ENOMEM;
                goto eclk;
        }

        i2c->adap.owner   = THIS_MODULE;
        i2c->adap.retries = 5;

        spin_lock_init(&i2c->lock);
        init_waitqueue_head(&i2c->wait);

        strlcpy(i2c->adap.name, "pxa_i2c-i2c", sizeof(i2c->adap.name));

        i2c->clk = clk_get(&dev->dev, NULL);
        if (IS_ERR(i2c->clk)) {
                ret = PTR_ERR(i2c->clk);
                goto eclk;
        }

        i2c->reg_base = ioremap(res->start, resource_size(res));
        if (!i2c->reg_base) {
                ret = -EIO;
                goto eremap;
```

```
	}

	i2c->reg_ibmr = i2c->reg_base + pxa_reg_layout[i2c_type].ibmr;
	i2c->reg_idbr = i2c->reg_base + pxa_reg_layout[i2c_type].idbr;
	i2c->reg_icr = i2c->reg_base + pxa_reg_layout[i2c_type].icr;
	i2c->reg_isr = i2c->reg_base + pxa_reg_layout[i2c_type].isr;
	if (i2c_type != REGS_CE4100)
		i2c->reg_isar = i2c->reg_base + pxa_reg_layout[i2c_type].isar;

	i2c->iobase = res->start;
	i2c->iosize = resource_size(res);

	i2c->irq = irq;

	i2c->slave_addr = I2C_PXA_SLAVE_ADDR;
	i2c->highmode_enter = false;

	if (plat) {
#ifdef CONFIG_I2C_PXA_SLAVE
		i2c->slave_addr = plat->slave_addr;
		i2c->slave = plat->slave;
#endif
		i2c->adap.class = plat->class;
	}

	if (i2c->high_mode) {
		if (i2c->rate) {
			clk_set_rate(i2c->clk, i2c->rate);
			pr_info("i2c: <%s> set rate to %ld\n",
				i2c->adap.name, clk_get_rate(i2c->clk));
		} else
			pr_warn("i2c: <%s> clock rate not set\n",
				i2c->adap.name);
	}

	clk_prepare_enable(i2c->clk);

	if (i2c->use_pio) {
		i2c->adap.algo = &i2c_pxa_pio_algorithm;
	} else {
		i2c->adap.algo = &i2c_pxa_algorithm;
		ret = request_irq(irq, i2c_pxa_handler, IRQF_SHARED,
				  dev_name(&dev->dev), i2c);
		if (ret)
			goto ereqirq;
	}

	i2c_pxa_reset(i2c);

	i2c->adap.algo_data = i2c;
	i2c->adap.dev.parent = &dev->dev;
#ifdef CONFIG_OF
	i2c->adap.dev.of_node = dev->dev.of_node;
#endif
```

```c

        ret =i2c_add_numbered_adapter(&i2c->adap);
        if (ret < 0) {
                printk(KERN_INFO "I2C: Failed to add bus\n");
                goto eadapt;
        }

        platform_set_drvdata(dev, i2c);

#ifdef CONFIG_I2C_PXA_SLAVE
        printk(KERN_INFO "I2C: %s: PXA I2C adapter, slave address %d\n",
               dev_name(&i2c->adap.dev), i2c->slave_addr);
#else
        printk(KERN_INFO "I2C: %s: PXA I2C adapter\n",
               dev_name(&i2c->adap.dev));
#endif
        return 0;

eadapt:
        if (!i2c->use_pio)
                free_irq(irq, i2c);
ereqirq:
        clk_disable_unprepare(i2c->clk);
        iounmap(i2c->reg_base);
eremap:
        clk_put(i2c->clk);
eclk:
        kfree(i2c);
emalloc:
        release_mem_region(res->start, resource_size(res));
        return ret;
}

int i2c_add_numbered_adapter(struct i2c_adapter *adap)
{
        if (adap->nr == -1) /* -1 means dynamically assign bus id */
                return i2c_add_adapter(adap);

        return __i2c_add_numbered_adapter(adap);
}
```

```
/**
 * i2c_add_adapter - declare i2c adapter, use dynamic bus number
 * @adapter: the adapter to add
 * Context: can sleep
 *
 * This routine is used to declare an I2C adapter when its bus number
 * doesn't matter or when its bus number is specified by an dt alias.
 * Examples of bases when the bus number doesn't matter: I2C adapters
 * dynamically added by USB links or PCI plugin cards.
 *
 * When this returns zero, a new bus number was allocated and stored
 * in adap->nr, and the specified adapter became available for clients.
 * Otherwise, a negative errno value is returned.
 */
int i2c_add_adapter(struct i2c_adapter *adapter)
{
        struct device *dev = &adapter->dev;
        int id;

        if (dev->of_node) {
                id = of_alias_get_id(dev->of_node, "i2c");
                if (id >= 0) {
                        adapter->nr = id;
                        return __i2c_add_numbered_adapter(adapter);
                }
        }

        mutex_lock(&core_lock);
        id = idr_alloc(&i2c_adapter_idr, adapter,
                       __i2c_first_dynamic_bus_num, 0, GFP_KERNEL);
        mutex_unlock(&core_lock);
        if (id < 0)
                return id;

        adapter->nr = id;

        return i2c_register_adapter(adapter);
}
```

从上可看i2c-pxa.ko是怎样载入并触发edt-ft5x06 driver的真正的初始化,即probe()的运行。

1. udevd通过uevent发觉要载入i2c-pxa.ko (通过modprobe utility)

2. The probe() function of i2c-pxa.ko 运行(因为i2c device早在initcall level 3就根据device tree中的i2c device node创建了)

3. call down chain as follow

i2c_pxa_probe()

```
    |
    |
   \|/
i2c_add_numbered_adapter()
    |
    |
   \|/
i2c_add_adapter()
    |
    |
   \|/
i2c_register_adapter()
    |
    |
   \|/
of_i2c_register_devices()
    |
    |
   \|/
i2c_new_device()   <--- edt-ft5x06 i2c client device真正的创建是从这里开始的，而非在initcall
level 3!!!
    |
    |
   \|/
device_register()
    |
    |
   \|/
device_add()
```

......

在edt_ft5x06_ts_probe() function的入口添加dump_stack()，会打印出如下call stack。

```
[<c0015568>] (unwind_backtrace) from [<c00114f8>] (show_stack+0x10/0x14)

[<c00114f8>] (show_stack) from [<c043ce70>] (dump_stack+0x80/0xc0)

[<c043ce70>] (dump_stack) from [<c031fd64>] (i2c_device_match+0x30/0xa8)

[<c031fd64>] (i2c_device_match) from [<c027b358>] (__device_attach+0x28/0x54)

[<c027b358>] (__device_attach) from [<c0279948>] (bus_for_each_drv+0x58/0x8c)

[<c0279948>] (bus_for_each_drv) from [<c027b108>] (device_attach+0x78/0x80)

[<c027b108>] (device_attach) from [<c027a7e4>] (bus_probe_device+0x84/0xa8)

[<c027a7e4>] (bus_probe_device) from [<c0278ce4>] (device_add+0x440/0x520)

[<c0278ce4>] (device_add) from [<c031ddc8>] (i2c_new_device+0x12c/0x174)

[<c031ddc8>] (i2c_new_device) from [<c031e8e0>] (i2c_register_adapter+0x454/0x48c)

[<c031e8e0>] (i2c_register_adapter) from [<bf0126b0>] (i2c_pxa_probe+0x3b4/0x4c0 [i2c_pxa])

[<bf0126b0>] (i2c_pxa_probe [i2c_pxa]) from [<c027ca50>] (platform_drv_probe+0x44/0xa4)

[<c027ca50>] (platform_drv_probe) from [<c027b194>] (really_probe+0x84/0x220)

[<c027b194>] (really_probe) from [<c027b41c>] (__driver_attach+0x98/0x9c)

[<c027b41c>] (__driver_attach) from [<c0279834>] (bus_for_each_dev+0x90/0x138)

[<c0279834>] (bus_for_each_dev) from [<c027aa64>] (bus_add_driver+0x154/0x20c)

[<c027aa64>] (bus_add_driver) from [<c027ba48>] (driver_register+0x78/0xf8)

[<c027ba48>] (driver_register) from [<c0008b40>] (do_one_initcall+0x11c/0x1c4)

[<c0008b40>] (do_one_initcall) from [<c0079348>] (load_module+0xdf8/0xf20)

[<c0079348>] (load_module) from [<c00795c8>] (SyS_finit_module+0x68/0x78)

[<c00795c8>] (SyS_finit_module) from [<c000e6a0>] (ret_fast_syscall+0x0/0x30)
```

总结大致过程如下：

1. udevd通过modprobe载入i2c-pxa.ko

2. i2c-pxa.ko载入并运行probe() function。由于i2c@d4033000 device在initcall level 3就生成了---应该是在of_platform_populate()中被创建。

3. i2c-pxa driver在初始化时会查看其device node(这里的i2c@d4033000)的child。/i2c@d4033000/edt-ft5x06@38即是其子节点。

4. i2c-pxa driver根据child device node中的xinformation创建i2c client device,并regsiter该device到i2c bus的device list中

5. step 4的action触发edt-ft5x06 driver与新创建的device的match，进而probe。