1. debug non-init function

we could get the function address from /proc/kallsyms

for example:

```
$ cat /proc/kallsyms | grep i2c_pxa_probe
```

没有任何输出，因为i2c-pxa.ko还没有载入

```
root@granite2:~# modprobe i2c-pxa
I2C: i2c-0: PXA I2C adapter
I2C: i2c-1: PXA I2C adapter
I2C: i2c-2: PXA I2C adapter
I2C: i2c-3: PXA I2C adapter
I2C: i2c-4: PXA I2C adapter
I2C: i2c-5: PXA I2C adapter
root@granite2:~# cat /proc/kallsyms | grep i2c_pxa_probe
bf25a2fc t i2c_pxa_probe[i2c_pxa]
```

i2c_pxa_probe() 位于**0xbf25a2fc**

root@granite2:~# modinfo i2c-pxa

filename: /lib/modules/3.18.7-yocto-standard/kernel/drivers/i2c/busses/i2c-pxa.ko

alias: platform:pxa2xx-i2c

license: GPL

alias: platform:ce4100-i2c

alias: platform:pxa3xx-pwri2c

alias: platform:pxa2xx-i2c

alias: of:N*T*Cmrvl,mmp-twsi*

alias: of:N*T*Cmrvl,pwri2c*

alias: of:N*T*Cmrvl,pxa-i2c*

depends:

intree: Y

vermagic: 3.18.7-yocto-standard SMP preempt mod_unload ARMv7 p2v8

由于ko是obj file，还未链接（链接在kernel载入ko时动态完成），所以只有offset，还没有address。

```
walterzh@walterzh-Precision-T1650:~/gerrit2/linux/3.18.7+gitAUTOINC+3c5efe25e7-r0/image/lib/modules/3.18.7-yocto-standard/kernel/drivers/i2c/busses$ nm i2c-pxa.ko | grep " i2c_"
00000000 t i2c_adap_pxa_exit
00000000 t i2c_adap_pxa_init
         U i2c_add_numbered_adapter
         U i2c_del_adapter
00000358 r i2c_pxa_algorithm
00000364 r i2c_pxa_dev_pm_ops
00000000 d i2c_pxa_driver
00000000 r i2c_pxa_dt_ids
000001fc t i2c_pxa_functionality
000007fc t i2c_pxa_handler
000003c0 r i2c_pxa_id_table
000007bc t i2c_pxa_master_complete
0000034c r i2c_pxa_pio_algorithm
00000cf4 t i2c_pxa_pio_xfer
000002fc t i2c_pxa_probe
00000274 t i2c_pxa_remove
00000000 t i2c_pxa_reset
00000208 t i2c_pxa_resume_noirq
00000250 t i2c_pxa_suspend_noirq
00000f14 t i2c_pxa_xfer
```

计算ko载入地址

以i2c_pxa_probe()为例

000002fc t i2c_pxa_probe

0xbf25a2fc - 0x2fc = 0xbf25a000,即kernel把i2c-pxa.ko载入到了0xbf25a000地址。

为了在XDB中调试i2c-pxa.ko，需要告诉xdb,i2c-pxa.ko的载入地址(0xbf25a000)。

2. debug probe() function

由于ko的probe()是在载入之时被kernel调用的，所以上面的方法没办法使用。

可行的方法是：在kernel载入ko后，调用probe() callback的地方设置断点。但由于各个driver的probe()是由自己所属的bus负责的。

即不同bus下的driver，设断点不同。

① platform bus

in drives/base/platform.c

```
1.  static int platform_drv_probe(struct device *_dev)
2.  {
3.          struct platform_driver *drv = to_platform_driver(_dev->driver);
4.          struct platform_device *dev = to_platform_device(_dev);
5.          int ret;
6.
7.          ret = of_clk_set_defaults(_dev->of_node, false);
8.          if (ret < 0)
9.                  return ret;
10.
11.         ret = dev_pm_domain_attach(_dev, true);
12.         if (ret != -EPROBE_DEFER) {
13.                 ret = drv->probe(dev);
14.                 if (ret)
15.                         dev_pm_domain_detach(_dev, true);
16.         }
17.
18.         if (drv->prevent_deferred_probe && ret == -EPROBE_DEFER) {
19.                 dev_warn(_dev, "probe deferral not supported\n");
20.                 ret = -ENXIO;
21.         }
22.
23.         return ret;
24. }
```

② I2C bus

in drivers/i2c/i2c-core.c

```c
static int i2c_device_probe(struct device *dev)
{
        struct i2c_client       *client = i2c_verify_client(dev);
        struct i2c_driver       *driver;
        int status;

        if (!client)
                return 0;

        driver = to_i2c_driver(dev->driver);
        if (!driver->probe || !driver->id_table)
                return -ENODEV;

        if (!device_can_wakeup(&client->dev))
                device_init_wakeup(&client->dev,
                                   client->flags & I2C_CLIENT_WAKE);
        dev_dbg(dev, "probe\n");

        status = of_clk_set_defaults(dev->of_node, false);
        if (status < 0)
                return status;

        status = dev_pm_domain_attach(&client->dev, true);
        if (status != -EPROBE_DEFER) {
                status = driver->probe(client, i2c_match_id(driver->id_table,
                                   client));
                if (status)
                        dev_pm_domain_detach(&client->dev, true);
        }

        return status;
}
```

③ SPI bus

in drivers/spi/spi.c

```
static int spi_drv_probe(struct device *dev)
{
        const struct spi_driver         *sdrv = to_spi_driver(dev->driver);
        int ret;

        ret = of_clk_set_defaults(dev->of_node, false);
        if (ret)
                return ret;

        ret = dev_pm_domain_attach(dev, true);
        if (ret != -EPROBE_DEFER) {
                ret = sdrv->probe(to_spi_device(dev));
                if (ret)
                        dev_pm_domain_detach(dev, true);
        }

        return ret;
}
```

④ usb bus

in drivers/usb/core/driver.c

```
/* called from driver core with dev locked */
static int usb_probe_interface(struct device *dev)
{
        struct usb_driver *driver = to_usb_driver(dev->driver);
        struct usb_interface *intf = to_usb_interface(dev);
        struct usb_device *udev = interface_to_usbdev(intf);
        const struct usb_device_id *id;
        int error = -ENODEV;
        int lpm_disable_error;

        dev_dbg(dev, "%s\n", __func__);

        intf->needs_binding = 0;

        if (usb_device_is_owned(udev))
                return error;

        if (udev->authorized == 0) {
                dev_err(&intf->dev, "Device is not authorized for usage\n");
                return error;
        }

        id = usb_match_dynamic_id(intf, driver);
        if (!id)
                id = usb_match_id(intf, driver->id_table);
        if (!id)
                return error;

        dev_dbg(dev, "%s - got id\n", __func__);

        error = usb_autoresume_device(udev);
        if (error)
                return error;

        intf->condition = USB_INTERFACE_BINDING;

        /* Probed interfaces are initially active.  They are
         * runtime-PM-enabled only if the driver has autosuspend support.
         * They are sensitive to their children's power states.
         */
        pm_runtime_set_active(dev);
        pm_suspend_ignore_children(dev, false);
        if (driver->supports_autosuspend)
                pm_runtime_enable(dev);

        /* If the new driver doesn't allow hub-initiated LPM, and we can't
         * disable hub-initiated LPM, then fail the probe.
         *
         * Otherwise, leaving LPM enabled should be harmless, because the
         * endpoint intervals should remain the same, and the U1/U2 timeouts
         * should remain the same.
         *
         * If we need to install alt setting 0 before probe, or another alt
```

```
54.             * setting during probe, that should also be fine.  usb_set_interface()
55.             * will attempt to disable LPM, and fail if it can't disable it.
56.             */
57.            lpm_disable_error = usb_unlocked_disable_lpm(udev);
58.            if (lpm_disable_error && driver->disable_hub_initiated_lpm) {
59.                    dev_err(&intf->dev, "%s Failed to disable LPM for driver %s\n.",
60.                                    __func__, driver->name);
61.                    error = lpm_disable_error;
62.                    goto err;
63.            }
64.
65.            /* Carry out a deferred switch to altsetting 0 */
66.            if (intf->needs_altsetting0) {
67.                    error = usb_set_interface(udev, intf->altsetting[0].
68.                                    desc.bInterfaceNumber, 0);
69.                    if (error < 0)
70.                            goto err;
71.                    intf->needs_altsetting0 = 0;
72.            }
73.
74.            error = driver->probe(intf, id);
75.            if (error)
76.                    goto err;
77.
78.            intf->condition = USB_INTERFACE_BOUND;
79.
80.            /* If the LPM disable succeeded, balance the ref counts. */
81.            if (!lpm_disable_error)
82.                    usb_unlocked_enable_lpm(udev);
83.
84.            usb_autosuspend_device(udev);
85.            return error;
86.
87.     err:
88.            usb_set_intfdata(intf, NULL);
89.            intf->needs_remote_wakeup = 0;
90.            intf->condition = USB_INTERFACE_UNBOUND;
91.            usb_cancel_queued_reset(intf);
92.
93.            /* If the LPM disable succeeded, balance the ref counts. */
94.            if (!lpm_disable_error)
95.                    usb_unlocked_enable_lpm(udev);
96.
97.            /* Unbound interfaces are always runtime-PM-disabled and -suspended */
98.            if (driver->supports_autosuspend)
99.                    pm_runtime_disable(dev);
100.           pm_runtime_set_suspended(dev);
101.
102.           usb_autosuspend_device(udev);
103.           return error;
104.   }
```

⑤ pci bus

in drivers/pci/pci-driver.c

```
static long local_pci_probe(void *_ddi)
{
        struct drv_dev_and_id *ddi = _ddi;
        struct pci_dev *pci_dev = ddi->dev;
        struct pci_driver *pci_drv = ddi->drv;
        struct device *dev = &pci_dev->dev;
        int rc;

        /*
         * Unbound PCI devices are always put in D0, regardless of
         * runtime PM status.  During probe, the device is set to
         * active and the usage count is incremented.  If the driver
         * supports runtime PM, it should call pm_runtime_put_noidle()
         * in its probe routine and pm_runtime_get_noresume() in its
         * remove routine.
         */
        pm_runtime_get_sync(dev);
        pci_dev->driver = pci_drv;
        rc = pci_drv->probe(pci_dev, ddi->id);
        if (!rc)
                return rc;
        if (rc < 0) {
                pci_dev->driver = NULL;
                pm_runtime_put_sync(dev);
                return rc;
        }
        /*
         * Probe function should return < 0 for failure, 0 for success
         * Treat values > 0 as success, but warn.
         */
        dev_warn(dev, "Driver probe function unexpectedly returned %d\n", rc);
        return 0;
}
```

⑥ pcie bus

drivers/pci/pcie/portdrv_core.c

```c
/**
 * pcie_port_probe_service - probe driver for given PCI Express port service
 * @dev: PCI Express port service device to probe against
 *
 * If PCI Express port service driver is registered with
 * pcie_port_service_register(), this function will be called by the driver core
 * whenever match is found between the driver and a port service device.
 */
static int pcie_port_probe_service(struct device *dev)
{
        struct pcie_device *pciedev;
        struct pcie_port_service_driver *driver;
        int status;

        if (!dev || !dev->driver)
                return -ENODEV;

        driver = to_service_driver(dev->driver);
        if (!driver || !driver->probe)
                return -ENODEV;

        pciedev = to_pcie_device(dev);
        status = driver->probe(pciedev);
        if (status)
                return status;

        dev_printk(KERN_DEBUG, dev, "service driver %s loaded\n", driver->name);
        get_device(dev);
        return 0;
}
```

3. 如果ko的probe()没有被调用到，那由几个原因可以check。

3.1 dts中该driver对应的device的compatible string是否与driver中的匹配(match)

3.2 该driver对应的device在kernel中被create了吗？

① 在device node中如果定义了status property，同时property value不等于"okay"或"ok",则kernel会忽略该device node

② 有些bus的device，kernel是不知道怎么创建的，只有该特定的bus driver才知道怎么create。所以如果该bus driver本身还没有载入，

那么自然该device也不会被create。

比如在G2中，touch screen device是挂在i2c-pxa bus driver上的，i2c-pxa.ko没有载入，touch screen device是不会在kernel中被创建的。

```
        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>;
            };
        };
```

i2c@d4033000 bus device在kernel初始化阶段被create,但edt-ft5x06@38 device只作为i2c@d4033000 device的一个child被记录着，并不会

随着i2c@d4033000 device的创建而创建。

在G2 LSP中,i2c-pxa driver是动态载入的，但edt-ft5x06 driver却是embedded in kernel。所以初看起来两个driver的初始化顺序有点奇怪。

edt-ft5x06 依赖与i2c-pxa，但edt-ft5x06 driver是embedded in kernel，而i2c-pxa是ko。embedded driver在kernel阶段就会被初始化，而

ko则是在mannual载入时才会初始化。

一般device都先于driver创建。常规创建步骤如下：

1. 在kernel的早期阶段(setup_arch阶段)，kernel parse dtb并create device tree

2. 在kernel的后期,kernel根据device tree，create device (这时i2c@d4033000 被创建，但edt-ft5x06@38 device还没有创建)

3. 在kernel的后期, embedded driver被初始化，probe()被调用(这时edt-ft5x06@38 的driver初始化被调用，但由于该device并没有创建，

所以edt-ft5x06@38的probe function并不会被调用)

.....

4. kernel启动完毕，手工载入i2c@d4033000 driver(i2c-pxa.ko),i2c-pxa.ko会枚举其device node下的child,发觉edt-ft5x06@38 device node,

所以create edt-ft5x06@38 device，由于edt-ft5x06@38 driver is embedded，这时候才触发edt-ft5x06@38 driver的probe()。