2. Device-Tree

NXKR_YoungSikYang

Exported on 02/23/2024

Table of Contents

1 Understanding the device tree 3
2 How the device tree is loaded by Bootloader 4
3 Platform device - Platform driver 5
3.1 How to connect a device to a driver 6
3.1.1 Add a device(mydevice@1) in the .dtsi (s5p6818.dtsi) 6
3.1.2 driver code (misc driver) 6
3.2 Probe 9
4 Device tree directory 10
4.1 /proc/device-tree/ 10
4.2 /sys/firmware/devicetree/base/ 10
4.2.1 Checking the properties of a device 10
4.2.2 Reading the information of a device 10

1 Understanding the device tree

Refer to the page [`u-boot/1. Device Driver/1`]

2 How the device tree is loaded by Bootloader

During the boot process, the secondary bootloader loads the DTB into memory. The specific method of loading can vary:

- **Directly from Storage**: The bootloader reads the DTB from its storage location into RAM.
- **Packaged with the Kernel**: In some configurations, the DTB may be appended to the kernel image. The bootloader loads the entire package into memory, and the kernel extracts the DTB.
- **User Selection or Automatic Detection**: In systems with multiple possible hardware configurations, the bootloader may present a selection to the user or automatically detect the hardware configuration to choose the correct DTB.

3 Platform device - Platform driver

- **Static Configuration**: Information about the device (like memory addresses, IRQ numbers, etc) is specified within static data like the Device Tree.
- **No Hardware Enumeration**: These devices are typically directly integrated into the motherboard and do not have an enumeration mechanism unlike PCI or USB devices.

3.1 How to connect a device to a driver

3.1.1 Add a device(mydevice@1) in the .dtsi (`s5p6818.dtsi`)

```
{
	model = "nexell soc";
	compatible = "nexell,s5p6818";
	#address-cells = <0x1>;
	#size-cells = <0x1>;

	aliases {
		serial0 = &serial0;
		serial1 = &serial1;
		serial2 = &serial2;
		serial3 = &serial3;
		serial4 = &serial4;
		serial5 = &serial5;
		i2s0    = &i2s_0;
		i2s1    = &i2s_1;
		i2s2    = &i2s_2;
		spi0    = &spi_0;
		spi1    = &spi_1;
		spi2    = &spi_2;
		i2c0    = &i2c_0;
		i2c1    = &i2c_1;
		i2c2    = &i2c_2;

		pinctrl0 = &pinctrl_0;
	};

	mydevice@1 {
		compatible = "yang,mydevice";
		/* Other necessary properties */
	};
```

3.1.2 driver code (misc driver)

The driver code should match the device name(mydevice@1) and 'compatible'(yang,mydevice)

drivers/misc/my_misc.c

```c
#include <linux/miscdevice.h>
#include <linux/fs.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/init.h>

/* Connect the device from the dts to the driver */
#include <linux/moduleparam.h>
#include <linux/platform_device.h>

static struct of_device_id mydevice_match[] = {
    { .compatible = "yang,mydevice",},
    {}
};
//MODULE_DEVICE_TABLE(of, );


static int mydevice_probe(struct platform_device *pdev)
{
    printk("===yang's device[%s][L:%d]", __func__, __LINE__);
    return 0;
}

static struct platform_driver mydevice_driver = {
    .probe = mydevice_probe,
    .driver = {
        .owner = THIS_MODULE,
        .name = "mydevice@1",
        .of_match_table = mydevice_match,
    },
};

static int __init mydevice_debug_init(void)
{
    printk("===yang's device[%s][L:%d]===\n", __func__, __LINE__);
    return platform_driver_register(&mydevice_driver);
}

late_initcall(mydevice_debug_init);

/*
** This function will be called when we open the Misc device file
*/
static int etx_misc_open(struct inode *inode, struct file *file)
{
    pr_info("EtX misc device open\n");
    return 0;
}
```

```
/*
** This function will be called when we close the Misc Device file
*/
static int etx_misc_close(struct inode *inodep, struct file *filp)
{
    pr_info("EtX misc device close\n");
    return 0;
}

/*
** This function will be called when we write the Misc Device file
*/
static ssize_t etx_misc_write(struct file *file, const char __user *buf,
                size_t len, loff_t *ppos)
{
    pr_info("EtX misc device write\n");

    /* We are not doing anything with this data now */

    return len;
}

/*
** This function will be called when we read the Misc Device file
*/
static ssize_t etx_misc_read(struct file *filp, char __user *buf,
                    size_t count, loff_t *f_pos)
{
    pr_info("EtX misc device read\n");

    return 0;
}

//File operation structure
static const struct file_operations fops = {
    .owner          = THIS_MODULE,
    .write          = etx_misc_write,
    .read           = etx_misc_read,
    .open           = etx_misc_open,
    .release        = etx_misc_close,
    .llseek         = no_llseek,
};

//Misc device structure
struct miscdevice etx_misc_device = {
    .minor = MISC_DYNAMIC_MINOR,
    .name = "simple_etx_misc",
    .fops = &fops,
};

/*
```

```
** Misc Init function
*/
static int __init misc_init(void)
{
    int error;

    error = misc_register(&etx_misc_device);
    if (error) {
        pr_err("misc_register failed!!!\n");
        return error;
    }

    pr_info("misc_register init done!!!\n");
    return 0;
}

/*
** Misc exit function
*/
static void __exit misc_exit(void)
{
    misc_deregister(&etx_misc_device);
    pr_info("misc_register exit done!!!\n");
}

module_init(misc_init)
module_exit(misc_exit)

MODULE_LICENSE("GPL");
MODULE_AUTHOR("EmbeTronicX <embetronicx@gmail.com>");
MODULE_DESCRIPTION("A simple device driver - Misc Driver");
MODULE_VERSION("1.29");
```

3.2 Probe

```
dmesg | grep -i yang
```

```
-# dmesg | grep -i yang
] ===yang's device[mydevice_debug_init][L:47]===
] ===yang's device[mydevice_probe][L:31]
```

4 Device tree directory

4.1 /proc/device-tree/

There are various directories and files corresponding to the nodes and properties defined in the Device Tree within **/proc/device-tree/**

Reading these files allows software to obtain information about the system's hardware configuration, such as peripheral addresses, interrupt numbers, and device parameters.

4.2 /sys/firmware/devicetree/base/

This path is like **/proc/device-tree/.** The sysfs interface is often preferred because it's more structured and designed to be easier to navigate programmatically.

4.2.1 Checking the properties of a device

```
ls /sys/firmware/devicetree/base/mydevice@1
```

4.2.2 Reading the information of a device

```
cat /sys/firmware/devicetree/base/mydevice@1/name
cat /sys/firmware/devicetree/base/mydevice@1/compatible
```