2. Device-Tree

NXKR\_YoungSikYang

Exported on 2024-02-23 11:35:27

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 5

3.1.1 Add a device(mydevice@1) in the .dtsi (s5p6818.dtsi) 5

3.1.2 driver code (misc driver) 5

3.2 Probe 7

4 Device tree directory 8

4.1 /proc/device-tree/ 8

4.2 /sys/firmware/devicetree/base/ 8

4.2.1 Checking the properties of a device 8

4.2.2 Reading the information of a device 8

# Understanding the device tree

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

# 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.

# 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.

## How to connect a device to a driver

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

### driver code (misc driver)

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

|  |
| --- |
| #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"); |

Code Block 1 drivers/misc/my\_misc.c

## Probe

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



# Device tree directory

## /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.

## /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.

### Checking the properties of a device

|  |
| --- |
| ls /sys/firmware/devicetree/base/mydevice@1 |

### Reading the information of a device

|  |
| --- |
| cat /sys/firmware/devicetree/base/mydevice@1/name  cat /sys/firmware/devicetree/base/mydevice@1/compatible |