Systèmes embarqués 15/16

J.-M Friedt

FEMTO-ST/département temps-fréquence

jmfriedt@femto-st.fr

transparents à `jmfriedt.free.fr`

22 mars 2018

Communication par /sys

Création du point de communciation /sys/bus/platform/devices/gpio-simple.0

```
static struct platform_device *pdev;

static struct platform_driver gpio_simple_driver = {
  .probe          = gpio_simple_probe,
  .remove         = gpio_simple_remove,
  .driver = {.name   = "gpio-simple", },
};

static int __init gpio_simple_init(void)
{platform_driver_register(&gpio_simple_driver);
 pdev = platform_device_register_simple("gpio-simple", 0, →
     ↪NULL, 0);
 if (IS_ERR(pdev))
       {platform_driver_unregister(&gpio_simple_driver);
        return PTR_ERR(pdev);
       }
 return 0;
}

static void __exit gpio_simple_exit(void)
{platform_device_unregister(pdev);
 platform_driver_unregister(&gpio_simple_driver);
}
```

Points de communication

Ajout d'un système de fichier[1] /sys/bus/platform/devices/gpio-simple.0/value

```
static DEVICE_ATTR(value, 0444, gpio_show, NULL);

static ssize_t gpio_show(struct device *dev, struct →
    ↪device_attribute *attr, char *buf)
{return sprintf(buf, "%d\n", gpio_get_value(gpio));}


static int gpio_probe(struct platform_device *pdev)
{err = device_create_file(&pdev->dev, &dev_attr_value);
 return 0;
}

static int gpio_remove(struct platform_device *pdev)
{device_remove_file(&pdev->dev, &dev_attr_value);}

static struct platform_driver gpio_simple_driver = {
 .probe  = gpio_probe,
 .remove = gpio_remove,
 .driver = {.name   = "gpio-simple", },
};
```

Attribut de la plateforme : une fonction de callback pour la lecture

1. https://www.kernel.org/doc/htmldocs/device-drivers/API-platform-device-register-simple.html

Communication

Fonction de callback [2] lorsqu'une requête est faite vers le point de communication (selon les attributs du fichier) :

```
static DEVICE_ATTR(value_out, S_IWUSR, NULL, ecriture);

static int valeur = 0;

static ssize_t ecriture(struct device *dev, struct →
    ↪device_attribute *attr,
const char *buf, size_t count)
{if (!count) return -EINVAL;
 sscanf(buf, "%d\n", &valeur);
 return count;
}
```

Cas de l'écriture

2. K.C. Accardi, *How to Write a Device Driver*, à `ftp://ftp.polsl.pl/pub/linux/kernel/people/kristen/presentations/FISL9.0/device_driver.pdf`

Communication

```
// methode show
static DEVICE_ATTR(value1, 0440, gpio_simple_show1, NULL);
static DEVICE_ATTR(value2, 0440, gpio_simple_show2, NULL);
// methode store
static DEVICE_ATTR(value3, 0220, NULL, gpio_simple_read1);
```

se traduit par

```
# ls /sys/bus/platform/drivers/gpio-simple/gpio-simple.0/
driver driver_override modalias power subsystem uevent value1
value2 value3
```

dont les actions sont par exemple

```
int gpio_simple_show1(struct device *dev, struct →
    ↪device_attribute *attr, char *buf)
{ return sprintf(buf, "Hello World 1\n");
}
```

Plateforme v.s pilote

- Un module se charge par `init` et se décharge par `exit`
- un seul module peut être chargé à un instant donné

  `insmod: ERROR: could not insert module xxx.ko: File exists`
- un module ne peut pas recevoir de configuration au moment du chargement par le noyau
- un module peut charger une plateforme :

```
static struct platform_device *pd1,*pd2;

static int __init gpio_simple_init(void)
{pd1=platform_device_register_simple("jmf",0,NULL,0);
 pd2=platform_device_register_simple("jmf",1,NULL,0);
 return(0);
}
```

- la plateforme décrit le matériel

Plateforme v.s pilote

- les plateformes se voient passer un argument

```
static struct platform_driver jmf_driver = {
    .probe  = jmf_probe,
    .remove = jmf_rm,
    .driver = {.name   = "jmf",},
};

static int jmf_rm(struct platform_device *pdev)
{printk(KERN_ALERT "dr bye %d\n",pdev->id);return 0;}

static int jmf_probe(struct platform_device *pdev)
{printk(KERN_ALERT "dr lo %d\n",pdev->id);return 0;}
// $LINUX/Documentation/driver-model/platform.txt
//  * platform_device.id ... the device instance
// number, or else "-1" to indicate there's only one.

static int __init jmf_init(void)
{platform_driver_register(&jmf_driver);return 0;}
```

-
```
[ 1238.117164] platform init  # insmod pl
[ 1240.112258] driver init    # insmod dr
[ 1240.112283] jmf hello 0
[ 1240.112298] jmf hello 1
[ 1262.529881] jmf bye 0      # rrmod pl
[ 1262.529914] jmf bye 1
[ 1262.529922] platform exit  # rmmod dr
```

Séparation module/pilote

- un module charge plusieurs instances d'une plateforme (*driver*)

```
static struct platform_device *p1, *p2;

static int __init gpio_simple_init(void)
{p1=platform_device_register_simple("jmf",0,NULL,0);
 p2=platform_device_register_simple("jmf",1,NULL,0);
...
```

- le *driver* ne contient que la définition de la plateforme : les méthodes init et exit ne contiennent que `platform_driver_register();` et `platform_driver_unregister();`
- $\Rightarrow$ définition **implicite** des méthodes `init` et `exit` du module par `module_platform_driver(gpio_simple_driver);` avec

```
static struct platform_driver gpio_simple_driver = {
        .probe          = gpio_simple_probe,
        .remove         = gpio_simple_remove,
        .driver = {.name   = "jmf",},
};

module_platform_driver(gpio_simple_driver);
```

Application

- Partant de l'exemple de module avec *timer* en `/dev`, implémenter une méthode `read` qui se débloque à intervalles de temps réguliers (utiliser un mutex)
- Proposer le programme C correspondant qui affiche un message à intervalle de temps déterminé par le *timer*
- Remplacer `/dev/` par une plateforme dans `/sys`

Pour chaque étape, effectuer la démonstration **sur PC** puis sur carte **Redpitaya**.