TP de Linux Embarque de JiangboWANG et KaixuanJIANG

TP1 prise en main

1.1 Exploitation des dossiers

/sys/class/leds/fpga_ledX/ :

```
➜  / cd /sys/class/leds/
➜  leds ls
fpga_led0  fpga_led2  fpga_led4  fpga_led6  fpga_led8  hps_led0
fpga_led1  fpga_led3  fpga_led5  fpga_led7  fpga_led9  i2c0-mux
➜  leds cd fpga_led1
➜  fpga_led1 ls
brightness  device  max_brightness  power  subsystem  trigger  uevent
```

Ce répertoire contient généralement l'interface de contrôle des LED connectées à un FPGA . À l'intérieur de ce répertoire, nous pouvons trouver le fichier brightness, où l'écriture de 0 ou 1 permet de contrôler l'état de la LED.

/proc/ioports :

```
➜  /proc cacat /proc/ioports
1/              1792/           1967/           4/              850/            driver/         mtd
10/             1802/           1976/           472/            857/            execdomains     net@
11/             1805/           1979/           489/            872/            fb              pagetypeinfo
1113/           1809/           1999/           490/            873/            filesystems     partitions
12/             1838/           2/              5/              875/            fs/             self@
13/             1841/           2001/           553/            888/            interrupts      slabinfo
14/             1842/           2027/           6/              896/            iomem           softirqs
1446/           1851/           2600/           663/            9/              ioports         stat
1461/           1859/           2620/           667/            asound/         irq/            swaps
15/             1863/           2622/           685/            buddyinfo       kallsyms        sys/
1516/           1864/           2635/           689/            bus/            key-users       sysrq-trigger
1580/           1869/           2889/           692/            cgroups         keys            sysvipc/
1584/           1873/           2960/           693/            cmdline         kmsg            thread-self@
1590/           1877/           2985/           7/              config.gz       kpagecount      timer_list
1640/           1880/           3/              734/            consoles        kpageflags      tty/
1667/           1883/           328/            744/            cpu/            loadavg         uptime
1693/           1917/           329/            754/            cpuinfo         locks           version
1709/           1923/           331/            8/              crypto          meminfo         vmallocinfo
1768/           1925/           332/            822/            device-tree@    misc            vmstat
1783/           1929/           334/            823/            devices         modules         zoneinfo
1786/           1941/           389/            845/            diskstats       mounts@
```

Ce fichier fournit des informations sur les ports d'entrée/sortie (I/O) du système. Chaque ligne correspond généralement à une plage de ports I/O, affichant la plage d'adresses et les périphériques ou ressources associés. Cela est utile pour examiner l'allocation des ressources I/O dans le système.

/proc/iomem:

```
➜  /proc cat iomem
00000000-2fffffff : System RAM
  00008000-009d5393 : Kernel code
  00a62000-00b0bd4f : Kernel data
ff200000-ff20001f : /sopc@0/i2s@0
ff200020-ff20003f : /sopc@0/i2s@0
ff201000-ff201007 : /sopc@0/bridge@0xc0000000/sysid@0x100001000
ff231000-ff23107f : /sopc@0/bridge@0xc0000000/vip@0x100031000
ff235000-ff23501f : /sopc@0/bridge@0xc0000000/spi@0x100035000
ff250000-ff25001f : /sopc@0/bridge@0xc0000000/i2c-ocores@0x100050000
ff252000-ff25201f : /sopc@0/bridge@0xc0000000/i2c-ocores@0x100052000
ff254000-ff25401f : /sopc@0/bridge@0xc0000000/i2c-ocores@0x100054000
ff256000-ff25601f : /sopc@0/bridge@0xc0000000/spi@0x100056000
ff258000-ff25801f : /sopc@0/bridge@0xc0000000/i2c-ocores@0x100058000
ff702000-ff703fff : /sopc@0/ethernet@0xff702000
ff704000-ff704fff : /sopc@0/flash@0xff704000
ff706000-ff706fff : axi_slave0
ff708000-ff7080ff : /sopc@0/gpio@0xff708000
ff709000-ff7090ff : /sopc@0/gpio@0xff709000
ff70a000-ff70a0ff : /sopc@0/gpio@0xff70a000
ffb40000-ffb7ffff : /sopc@0/usb@0xffb40000
ffb90000-ffb900ff : axi_slave1
```

Ce fichier fournit des informations sur le mappage mémoire du système. Chaque ligne correspond généralement à une plage de mémoire, affichant la plage d'adresses et son utilisation.

/proc/device-tree/sopc@0

```
➜  /proc cd device-tree/sopc@0
➜  sopc@0 ls
#address-cells       ethernet@0xff700000  gpio@0xff709000  ranges              timer@0xffc08000
#size-cells          ethernet@0xff702000  gpio@0xff70a000  rl3regs@0xff800000  timer@0xffc09000
L2-cache@0xfffef000  flash@0xff704000     i2c@0xffc04000   rstmgr@0xffd05000   timer@0xffd00000
bridge@0xc0000000    flash@0xff705000     i2c@0xffc05000   scu@0xfffec000      timer@0xffd01000
bus-frequency        flash@0xff900000     i2c@0xffc06000   sdr-ctl@0xffc25000  timer@0xffd02000
can@0xffc00000       fpgabridge@0         i2c@0xffc07000   serial@0xffc02000   timer@0xffd03000
can@0xffc01000       fpgabridge@1         i2s@0            serial@0xffc03000   timer@0xfffec600
clkmgr@0xffd04000    fpgabridge@2         intc@0xfffed000  sound               usb@0xffb00000
compatible           fpgabridge@3         leds             spi@0xfff00000      usb@0xffb40000
device_type          fpgamgr@0xff706000   name             spi@0xfff01000      usbphy@0
dma@0xffe01000       gpio@0xff708000      pmu0             sysmgr@0xffd08000   vcc3p3-regulator
➜  sopc@0
```

Ce fichier fournit des informations liées à l'arbre des périphériques, où sopc@0 pourrait être un

nœud dans l'arbre des périphériques. L'arbre des périphériques est une structure de données décrivant la configuration matérielle et structurelle d'un système embarqué. Sous ce nœud, nous pouvons trouver des informations sur des blocs matériels tels que les FPGA.

/proc/device-tree/sopc@0 à comparer avec le fichier iomem

/proc/device-tree/sopc@0 fournit des informations plus spécifiques sur la configuration matérielle et les connexions, notamment celles liées à un composant spécifique, comme un FPGA.

/proc/iomem donne une vue d'ensemble des plages mémoire utilisées dans le système, sans se concentrer spécifiquement sur la configuration détaillée des composants matériels.

1.2 Hello

Tout d'abord, nous utilisons arm-linux-gnueabihf-gcc dans Linux sous VirtualBox pour compiler helloworld.c afin d'obtenir le fichier exécutable helloworld.o qui peut être exécuté sur un processeur ARM. Ensuite, nous utilisons scp command pour envoyer helloworld.o vers un FPGA, où il est exécuté et obtient le résultat « Hello world! ».

```
➜ TP1 arm-linux-gnueabihf-gcc helloworld.c -o helloworld.o
➜ TP1 ls
helloworld.c  helloworld.o
➜ TP1 scp ./helloworld.o  root@192.168.88.47:/home/root/JiangboWANG_KaixuanJIANG_ESE_LiunxEmbarque
helloworld.o
➜ TP1
```

```
➜ JiangboWANG_KaixuanJIANG_ESE_LiunxEmbarque ./helloworld.o
Hello world!
```

1.3 Accès au matériel

Nous pouvons entrer la commande suivante dans le terminal pour contrôler l'allumage et l'extinction de la LED.

```
➜ ~ echo "1" > /sys/class/leds/fpga_led1/brightness
➜ ~ echo "0" > /sys/class/leds/fpga_led1/brightness
```

1.4 Chenillard

Nous devons créer un programme de lumière clignotante. Pour cela, nous avons créé une fonction nommée ***ledset()***. Cette fonction reçoit deux paramètres : ***ledNum*** qui représente le numéro de la LED, et ***state*** qui indique si cette LED est allumée ou éteinte.

```c
void ledSet(int ledNum, int state)
{
    FILE *ledFile;
    char filename[100];
    snprintf(filename, sizeof(filename), "/sys/class/leds/fpga_led%d/brightness", ledNum);
    ledFile = fopen(filename, "w");
    if (ledFile == NULL)
    {
        perror("Error opening LED files");
        exit(EXIT_FAILURE);
    }
    fprintf(ledFile, "%d", state);
    fflush(ledFile);
    fclose(ledFile);
}
```

TP2 Modules kernel

2.1 Accès aux registres

2.1.1 La fonction mmap()

L'utilisation de la fonction `mmap()` pour accéder au registre GPIO connecté aux LED à l'adresse `0xFF203000` présente certains avantages, mais aussi des problèmes et des limites :

2.1.2 La avantages et limites

Avantages :

1. Accès direct à la mémoire : `mmap()` permet d'obtenir un accès direct à une zone de la mémoire physique du système. Cela peut être plus efficace que d'autres méthodes d'accès aux registres, car cela évite les opérations de lecture/écriture dans des fichiers ouverts.

2. Haute performance : L'accès direct à la mémoire via `***mmap()***` peut être plus rapide que certaines autres méthodes, car il évite des opérations supplémentaires associées à l'utilisation de fonctions standard de fichiers.

Problèmes et Limites :

1. Sécurité : L'utilisation de `***mmap()***` pour accéder directement à la mémoire peut poser des problèmes de sécurité, en particulier si le programme est exécuté avec des privilèges élevés. L'accès direct à la mémoire peut potentiellement causer des problèmes de sécurité si cela n'est pas géré correctement.

2. Portabilité : Cette méthode peut ne pas être portable entre différentes plates-formes matérielles. Les adresses physiques des registres varient d'une architecture à l'autre, et donc le code basé sur `***mmap()***` peut nécessiter des adaptations pour fonctionner sur différents matériels.

3. Gestion des permissions :L'accès direct à la mémoire nécessite généralement des privilèges élevés, ce qui peut compliquer la gestion des permissions. Les programmes nécessitant des privilèges élevés peuvent poser des risques de sécurité s'ils ne sont pas correctement gérés.

4. Possibilité de plantage du système : Une mauvaise utilisation de `***mmap()***`, telle qu'un accès à une adresse invalide, peut provoquer des plantages du système ou des comportements imprévisibles.

5. Dépendance au matériel : Cette méthode dépend fortement de la structure matérielle spécifique du système. Si la configuration matérielle change, le code basé sur `***mmap()***` peut devenir obsolète.

2.2 Compilation de module noyau sur la VM

> Utilisez `modinfo`, `lsmod`, `insmod` et `rmmod` pour tester votre module (à utiliser avec `sudo`) : chargez le et vérifiez que le module fonctionne bien (`sudo dmesg`).
> Pour la suite, tester les programmes suivants (voir cours)

Utilisez la commande modinfo ici, les informations affichées sont comme indiqué dans le schéma ci-dessous

```
ensea@VM-SOC:~/src/RessourcesTP2$ sudo modinfo hello.ko
filename:       /home/ensea/src/RessourcesTP2/hello.ko
description:    Hello world Module
author:         Christophe Barès
license:        GPL
depends:
retpoline:      Y
name:           hello
vermagic:       4.19.0-6-amd64 SMP mod_unload modversions
ensea@VM-SOC:~/src/RessourcesTP2$
```

Insmod : Lorsque nous utilisons la commande insmod, nous pouvons voir la sortie de helloworld en utilisant la commande dmesg

```
[ 1240.556664] 22:20:56.551406 control  Stopping all guest processes ...
[ 1240.556687] 22:20:56.551442 control  Closing all guest files ...
[ 1242.664253] 12:53:34.222230 timesync vgsvcTimeSyncWorker: Radical host time change: 570 764 980
00 000ns (HostNow=1 701 262 414 219 000 000 ns HostLast=1 700 691 649 239 000 000 ns)
[ 1252.665986] 12:53:44.223864 timesync vgsvcTimeSyncWorker: Radical guest time change: 570 765 567
191 000ns (GuestNow=1 701 262 424 223 837 000 ns GuestLast=1 700 691 658 656 646 000 ns fSetTimeLas
Loop=true )
[ 2318.056555] Hello world!
ensea@VM-SOC:~/src/RessourcesTP2$
```

Rmmod :

```
ensea@VM-SOC:~/src/RessourcesTP2$ sudo rmmod hello.ko
[ 2721.832919] Bye bye...
ensea@VM-SOC:~/src/RessourcesTP2$ AA_
```

utilisation de paramètres au chargement du module

Nous ajoutons ces lignes de code pour transmettre des paramètres lors du chargement du module.

```
static int param;

module_param(param,int,0);
MODULE_PARM_DESC(param,"Un paramètre de ce module");
```

Nous pouvons voir dans la sortie de dmesg que nous transmettons un entier de type int lors de l'insmod du module, et sa valeur est 6.

```
[ 2318.056555] Hello world!
[ 2721.832919] Bye bye...
[ 3378.107293] Hello world!
[ 3378.107294] le paramètre est=6
ensea@VM-SOC:~/src/RessourcesTP2$ _
```

création d’un entrée dans /proc

```c
static int __init my_module_init(void) {
    // Création d’un fichier

    sub_entry = proc_mkdir(SUB_DIR, NULL );
    if (!sub_entry) {
        printk(KERN_ERR "Failed to create subdirectory\n");
        remove_proc_entry(SUB_DIR, NULL);
        return -ENOMEM;
    }
    proc_entry = proc_create(PROC_ENTRY, 0644, sub_entry, &proc

    return 0;
}
```

En utilisant les fonctions proc_mkdir et proc_create, nous pouvons créer un répertoire dans le dossier proc et y ajouter un fichier.

```
ensea@VM-SOC:~/src/RessourcesTP2$ sudo insmod proc.ko
ensea@VM-SOC:~/src/RessourcesTP2$ ls /proc
1     17   24   33   407   7667       diskstats    kcore        mounts        swaps
10    182  25   34   416   8          dma          keys         mtrr          sys
11    19   255  345  4200  9          driver       key-users    my_sub_dir    sysrq-trigger
12    197  26   35   483   acpi       execdomains  kmsg         net           sysvipc
120   199  27   371  54    buddyinfo  fb           kpagecgroup  pagetypeinfo  thread-self
121   2    28   372  55    bus        filesystems  kpagecount   partitions    timer_list
123   20   29   373  56    cgroups    fs           kpageflags   sched_debug   tty
14    200  3    375  57    cmdline    interrupts   loadavg      schedstat     uptime
15    21   30   381  6     consoles   iomem        locks        self          version
1534  22   31   4    66    cpuinfo    ioports      meminfo      slabinfo      vmallocinfo
16    23   315  400  7     crypto     irq          misc         softirqs      vmstat
168   233  32   406  7170  devices    kallsyms     modules      stat          zoneinfo
ensea@VM-SOC:~/src/RessourcesTP2$ cd /proc/my_sub_dir/
ensea@VM-SOC:/proc/my_sub_dir$ ls
my_proc_entry
```

utilisation d’un timer

```
[ 5423.378395] my_init: Timer module loaded
[ 5423.378396] my_init: Setup timer to fire in 2s (4296248168)
[ 5425.393437] my_timer_callback called (4296248672)
```

```c
static int __init my_init(void)
{
    int ret;

    pr_info("%s: Timer module loaded\n", __func__);

    timer_setup(&my_timer, my_timer_callback, 0);
    pr_info("%s: Setup timer to fire in 2s (%ld)\n", __func__, jiffies);

    ret = mod_timer(&my_timer, jiffies + msecs_to_jiffies(2000));
    if (ret)
        pr_err("%s: Timer firing failed\n", __func__);

    return 0;
}
```

La fonction `timer_setup` est utilisée pour initialiser la structure du minuteur dans le noyau, et la fonction `mod_timer` est utilisée pour modifier le temps d'expiration du minuteur. Cela signifie que le noyau appellera automatiquement la fonction `my_timer_callback` créée précédemment avec `timer_setup` deux secondes après l'appel de la fonction actuelle.

2.3 Cross Compilation de modules noyau

Notez le chemin vers ces compilateurs : whereis arm-linux-gnueabihf-gcc

```
ensea@VM-SOC:~/src/RessourcesTP2$ whereis arm-linux-gnueabihf-gcc
arm-linux-gnueabihf-gcc: /usr/bin/arm-linux-gnueabihf-gcc
```

La mention <chemin_arm-linux-gnueabihf sans l'ajout de gcc à la fin est une référence à un chemin spécifique dans le contexte du développement logiciel pour une architecture ARM avec l'ensemble d'outils GCC (GNU Compiler Collection) pour l'ABI (Application Binary Interface) appelé gnueabihf.

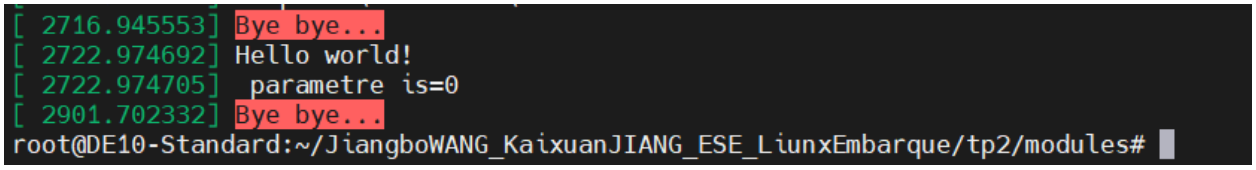
— Quel est le rôle des lignes commençant par export ?

Les lignes commençant par `export` dans un script shell ont pour rôle de définir des variables d'environnement pour être utilisées par le script lui-même ou par d'autres processus lancés à partir du script. Dans le contexte spécifique de votre question, le `export` est utilisé pour définir une variable d'environnement nommée `CRDSS_COMPILE` avec la valeur du chemin vers le compilateur croisé (cross-compiler) pour l'architecture ARM.

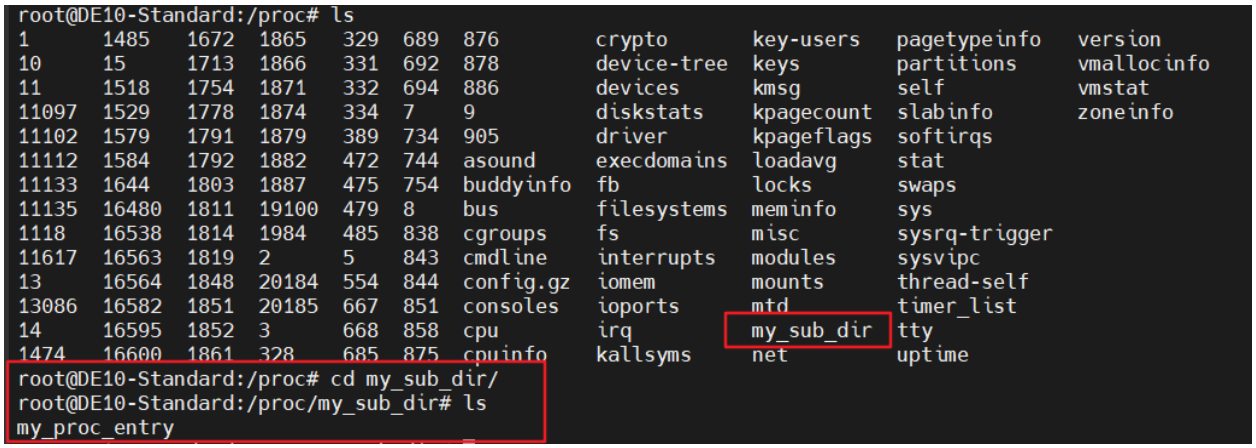
— Pourquoi le chemin fini par un tiret "-" ?

le chemin qui se termine par un tiret `-`, cela indique qu'il s'agit du préfixe du compilateur croisé. Cela permet généralement d'indiquer que le chemin spécifié est le préfixe à utiliser pour les outils de compilation croisée, et que le nom réel du compilateur (par exemple, `gcc`) sera ajouté par la suite en fonction du contexte d'utilisation.
le chemin spécifié est `/usr/bin/arm-linux-gnueabihf-`, cela signifie que les outils spécifiques comme `gcc` seront construits en utilisant ce préfixe. Ainsi, le compilateur complet serait `/usr/bin/arm-linux-gnueabihf-gcc`.

Essayez de compiler vos autres module pour la carte SoC



Executer insmod et rmmod hello.ko ,on peut voir les resultas sur la carte



Executer insmod module proc on peut voir le fichir que on a creer se trouve sur /proc

```
→  wjbfolder sudo insmod timer_module.ko
→  wjbfolder dmesg | tail
[   32.003404] IPv6: ADDRCONF(NETDEV_UP): eth0: link is not ready
[   35.987950] socfpga-dwmac ff702000.ethernet eth0: Link is Up - 1Gbps/Full - flow control rx/tx
[   35.988869] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
[   49.736891] major = 251, minor = 0, result = 0
[   49.737969] module loading...
[   49.750303] buffer0 virtual address 0xf0ea2000
[   49.750350] buffer0 physical address 0x29800000
[  186.690913] my_init: Timer module loaded
[  186.690930] my_init: Setup timer to fire in 2s (-11334)
[  188.685231] my_timer_callback called (-11134)
→  wjbfolder
```

2.3.4 Chenillard

Le pattern depuis le fichier : /proc/ensea/chenille

En utilisant les fonctions proc_mkdir et proc_create, nous pouvons créer un répertoire ensea dans le dossier proc et y ajouter un fichier chenille.

```
root@DE10-Standard:/proc# ls
1     1715  1939  332   4935  668  850        consoles     iomem       mounts         timer_list
10    1772  1943  334   4940  685  857        cpu          ioports     mtd            tty
11    1778  1944  389   4950  689  868        cpuinfo      irq         net            uptime
13    1779  1949  4503  4972  692  877        crypto       kallsyms    pagetypeinfo   version
14    1796  1953  472   4974  693  878        device-tree  key-users   partitions     vmallocinfo
1440  1812  1955  4726  5     7    892        devices      keys        self           vmstat
1475  1828  1960  475   554   734  9          diskstats    kmsg        slabinfo       zoneinfo
15    1882  1970  4782  6     744  901        driver       kpagecount  softirqs
1522  1890  2     479   6123  754  asound     ensea        kpageflags  stat
1526  1893  2006  4795  6124  782  buddyinfo  execdomains  loadavg     swaps
1603  1897  3     4797  6137  8    bus        fb           locks       sys
1604  1926  328   4817  6388  822  cgroups    filesystems  meminfo     sysrq-trigger
1654  1929  329   485   6389  823  cmdline    fs           misc        sysvipc
1682  1930  331   4871  667   845  config.gz  interrupts   modules     thread-self
root@DE10-Standard:/proc#
root@DE10-Standard:/proc# cd ensea/
root@DE10-Standard:/proc/ensea# ls
chenille
```

La vitesse au moment du chargement du module:

La méthode que nous utilisons dans la capture d'écran ci-dessous, nous ouvrons le fichier précédemment créé dans l'espace utilisateur en appelant la fonction `open`, puis nous utilisons la fonction `write` pour modifier la vitesse dans l'espace du noyau. Dans le TP3 ultérieur, nous changeons la vitesse au moment du chargement du module.

```c
int main() {
    int fd;
    char data_to_write[] = "speed=42";

    fd = open("/proc/ensea/chenille", O_WRONLY);

    if (fd < 0) {
        perror("Error opening file");
        exit(EXIT_FAILURE);
    }

    if (write(fd, data_to_write, sizeof(data_to_write)) < 0) {
        perror("Error writing to file");
        close(fd);
        exit(EXIT_FAILURE);
    }

    close(fd);

    return 0;
}
```

Ici, nous modifions la vitesse à 42 dans l'espace utilisateur, correspondant à la fonction `proc_write` dans l'espace du noyau pour modifier la valeur correspondante. Dans le noyau, nous vérifions la chaîne transmise depuis l'utilisateur, extrayons la valeur de la vitesse, et bien sûr, dans cet exemple, nous imprimons simplement la valeur modifiée.

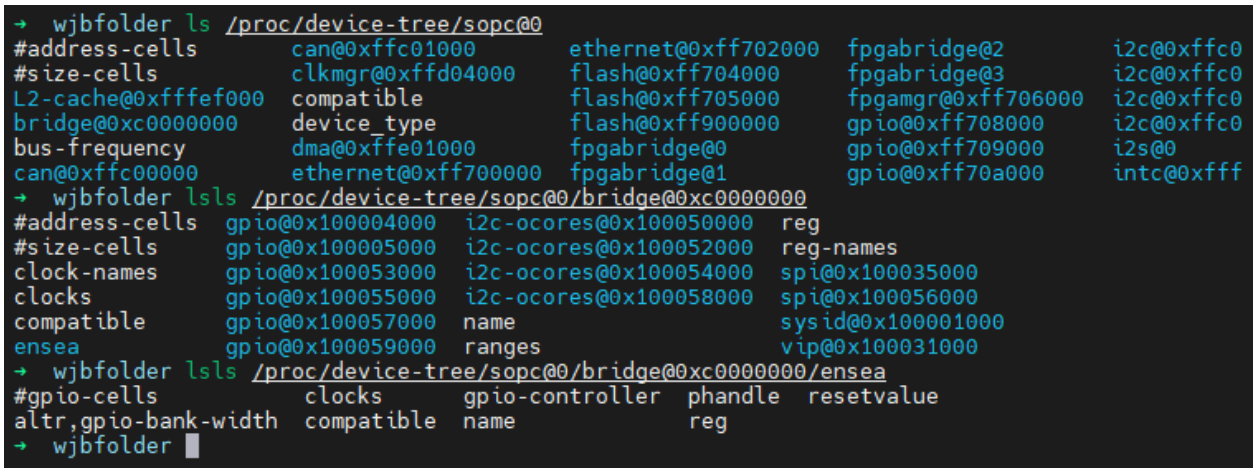
```c
static ssize_t proc_write(struct file *file, const char __user *buffer, size_t count, loff_t *offset)
    char input[10];
    char prefix[] = "speed=";

    if (count > sizeof(input) - 1) {
        printk(KERN_ERR "Input is too long\n");
        return -EINVAL;
    }

    if (copy_from_user(input, buffer, count)) {
        return -EFAULT;
    }

    input[count] = '\0';

    if (strncmp(input, prefix, strlen(prefix)) == 0) {
        if (sscanf(input + strlen(prefix), "%d", &blink_speed) == 1) {
            printk(KERN_INFO "Input is valid. Speed value: %d\n", blink_speed);
        } else {
            printk(KERN_INFO "Invalid speed value\n");
        }
    } else {
        printk(KERN_INFO "Input does not start with 'speed='\n");
    }
```

En utilisant la commande `dmesg`, nous pouvons visualiser les sorties d'impression du noyau.

```
[ 7072.380839] Open success
[ 7072.383412] Input is valid. Speed value: 42
[ 7109.353098] Bye bye...
```

TP3 Device Tree

Explorez le nouveau device tree



3.1 module accedant au LED via /dev

Quel sont les rôles des principales fonctions (probe, read,write, remove), et quand entrent-elles en action ?

leds_probe :

- Rôle : Cette fonction est appelée lors de la découverte d'un nouveau périphérique qui correspond aux critères définis par le pilote (dans ce cas, un périphérique LED identifié par "dev,ensea" dans la table de compatibilité des périphériques). Elle initialise le périphérique, configure les ressources matérielles (comme l'accès à la mémoire), et prépare le périphérique pour une utilisation ultérieure.
- Déclenchement : Elle est automatiquement appelée par le système de gestion des pilotes de plateforme lorsqu'un périphérique correspondant est détecté.

leds_read :

- Rôle : Cette fonction gère les opérations de lecture du périphérique. Lorsqu'un utilisateur lit le fichier de périphérique (par exemple, /dev/ensea_leds), cette fonction est appelée pour retourner l'état actuel des LEDs.
- Déclenchement : Elle est appelée lorsque le fichier de périphérique associé au pilote est lu par un processus utilisateur.

leds_write :

- Rôle : Gère les opérations d'écriture sur le périphérique. Quand un utilisateur écrit dans le fichier de périphérique, cette fonction est appelée pour mettre à jour l'état des LEDs en fonction des données fournies par l'utilisateur.
- Déclenchement : Elle est appelée lorsqu'un processus utilisateur écrit dans le fichier de périphérique associé.

leds_remove :

- Rôle : Nettoie et libère les ressources allouées au périphérique lorsqu'il n'est plus nécessaire ou lors de la suppression du pilote. Cette fonction éteint les LEDs et désenregistre le périphérique de caractère, annulant ainsi toute interface utilisateur.
- Déclenchement : Elle est appelée lorsque le périphérique est retiré du système ou lorsque le module du pilote est déchargé (par exemple, avec la commande rmmod).

3.2 Module final

3.2.1 Cahier des charges

Choix de la vitesse de balayage par une option au moment du chargement du module

```
module_param(speed,int,0);
MODULE_PARM_DESC(speed,"Un paramètre de ce module");
```

Ici, en utilisant ces deux lignes de code, nous transmettons la valeur de la vitesse lors de l'insmod du module.

```c
static void my_timer_callback(struct timer_list *timer)
{

        static int y = 0;
        static int i = 0;

        printk("%s called (%ld)\n", __func__, jiffies);

        dev->leds_value = (0x01 << i);
        iowrite32(dev->leds_value, dev->regs);
        i++;  // Move to the next LED value

        if (i == 8) {
            i = 0;  // Reset i to 0 when all LEDs are processed
            y++;   // Move to the next iteration of the outer loop
        }

        if (y < 5) {
            // If not all iterations are completed, set the timer for the next iteration
            mod_timer(&my_timer, jiffies + msecs_to_jiffies(speed));
        } else {
            // All iterations are completed, reset y and stop the timer
            y = 0;
        }
```

Dans la fonction de rappel du timer, nous définissons la variable `speed` en tant que variable globale. En modifiant `speed`, nous ajustons ainsi l'intervalle entre chaque clignotement des leds, ce qui influence la vitesse du chenillard.

Récupération de la vitesse courante par lecture du fichier /proc/ensea/speed

```c
// Called when the driver is installed
static int leds_init(void)
{
    int ret_val = 0;
    pr_info("Initializing the Ensea LEDs module\n");
    printk(KERN_INFO "Setting blink speed to %d\n", speed);
       sub_entry = proc_mkdir(SUB_DIR, NULL);
    if (!sub_entry) {
        printk(KERN_ERR "Failed to create subdirectory\n");
        return -ENOMEM;
    }

    proc_entry_speed = proc_create(PROC_ENTRY, 0666, sub_entry, &ensea_speed_fops);
    if (proc_entry_speed == NULL) {
        pr_err("Failed to create /proc/ensea/speed entry\n");
    }
```

Dans la fonction `led_init`, nous créons le fichier `proc/ensea/speed` et l'associons à la structure `ensea_speed_fops`.

```c
static const struct file_operations ensea_speed_fops = {
    .owner = THIS_MODULE,
    .read = speed_read,
};
```

```c
static ssize_t speed_read(struct file *file, char *buffer, size_t len, loff_t *offset)
{
    int success = 0;
    char result[100] = {0};
    //speed = INTERVALLE ;
    sprintf(result, "Speed: %d\r\n", speed);
    success = copy_to_user(buffer, &result, sizeof(result));
    // If we failed to copy the value to userspace, display an error message
    if(success != 0) {
        pr_info("Failed to return current led value to userspace\n");
        return -EFAULT; // Bad address error value. It's likely that "buffer" doesn't point to a good
    }
    return 0;

}
```

Dans la fonction `speed_read`, nous utilisons la fonction `copy_to_user` pour transmettre la vitesse actuelle à l'espace utilisateur.

```c
    fd = open("/proc/ensea/speed", O_RDONLY);

    if (fd < 0) {
        perror("Error opening file speed");
        exit(EXIT_FAILURE);
    }

    if (read(fd, &data_speed, sizeof(data_speed)) < 0) {
        perror("Error read  file");
        close(fd);
        exit(EXIT_FAILURE);
    }
    printf("%s\r\n",data_speed);
    close(fd);
```

Dans l'espace utilisateur, nous utilisons la fonction `read` pour lire la vitesse actuelle.

Modification de la patern par écriture dans le fichier /dev/ensea-led

```c
// This function gets called whenever a write operation occurs on one of the character files
static ssize_t leds_write(struct file *file, const char *buffer, size_t len, loff_t *offset)
{
    int mode= 0 ;
    int y,i;
    char input[10];
    char pattern[]= "pattern=";
    char sens1[] = "LED dans le sens de gauche a droite";
    char sens2[] = "LED dans le sens de droite a gauche";
    char mode1[] = " LED en mode chenillard";
    char mode3[] = " LED en mode on";
    char mode4[] = " LED en mode off";
```

```
input[len] = '\0';
if (strncmp(input, pattern, strlen(pattern)) == 0) {
    if (sscanf(input + strlen(pattern), "%d", &mode) == 1)
    {
        printk(KERN_INFO "Input is valid. pattern value: %d\n", mode);
    switch (mode) {
        case 1:
            printk(KERN_INFO "Processing for mode 1\n");
            dev->leds_value = 0xff ;
            iowrite32(dev->leds_value, dev->regs);
            memset(copy2usr, 0, sizeof(copy2usr));
            strcpy(copy2usr, mode3);
            break;
        case 2:
            printk(KERN_INFO "Processing for mode 2\n");
            dev->leds_value = 0x00 ;
            iowrite32(dev->leds_value, dev->regs);
            memset(copy2usr, 0, sizeof(copy2usr));
            strcpy(copy2usr, mode4);
            break;
        case 3:
            printk(KERN_INFO "Processing for mode 3\n");
            dev->leds_value = (0x01 << 2);
            iowrite32(dev->leds_value, dev->regs);

            setup_timer(&my_timer, my_timer_callback, 0);
            mod_timer(&my_timer, jiffies + msecs_to_jiffies(speed));
            memset(copy2usr, 0, sizeof(copy2usr));
            strcpy(copy2usr, mode1);
            memset(directe, 0, sizeof(directe));
```

```
        strcpy(directe, sens2);
        break;
    case 4:
        printk(KERN_INFO "Processing for mode 4 ,changer le sens \n");
        setup_timer(&my_timer2, my_timer_callback2, 0);
        mod_timer(&my_timer2, jiffies + msecs_to_jiffies(speed));
        memset(copy2usr, 0, sizeof(copy2usr));
        strcpy(copy2usr, mode1);
        memset(directe, 0, sizeof(directe));
        strcpy(directe, sens1);
        break;
    default:
        printk(KERN_INFO "Invalid mode value\n");

        break;
}
    }
```

Dans la fonction `leds_write`, nous utilisons la comparaison des chaînes transmises depuis l'espace utilisateur pour décider de basculer vers le mode correspondant. Nous proposons quatre modes : le mode 1 éteint tous les LEDs, le mode 2 les allume tous, le mode 3 active le chenillard lumineux de gauche à droite, et le mode 4 de droite à gauche.

```c
static void my_timer_callback(struct timer_list *timer)
{

        static int y = 0;
        static int i = 0;

        printk("%s called (%ld)\n", __func__, jiffies);

        dev->leds_value = (0x01 << i);
        iowrite32(dev->leds_value, dev->regs);
        i++;  // Move to the next LED value

        if (i == 8) {
            i = 0;  // Reset i to 0 when all LEDs are processed
            y++;   // Move to the next iteration of the outer loop
        }

        if (y < 5) {
            // If not all iterations are completed, set the timer for the next iteration
            mod_timer(&my_timer, jiffies + msecs_to_jiffies(speed));
        } else {
            // All iterations are completed, reset y and stop the timer
            y = 0;
        }



}
```

Dans la fonction de rappel du timer, nous implémentons le chenillard lumineux en écrivant la valeur de LED différente à chaque boucle.

```c
    char data_to_write[] = "pattern=3";
    fd1 = open("/dev/ensea_leds", O_RDWR);
    if (fd1 < 0) {
        perror("Error opening file /dev/led");
        exit(EXIT_FAILURE);
    }
    if (write(fd1, data_to_write, sizeof(data_to_write)) < 0) {
        perror("Error writing to file ");
        close(fd1);
        exit(EXIT_FAILURE);
    }
    if (read(fd1, &data_pattern, sizeof(data_pattern)) < 0) {
        perror("Error read  file");
        close(fd1);
        exit(EXIT_FAILURE);
    }
    printf("%s\r\n",data_pattern);
    close(fd1);
```

Dans l'espace utilisateur, nous choisissons le mode actuel des LEDs en ouvrant le fichier du périphérique LED et en utilisant la fonction `write`.Nous récupérons également le motif actuel en lisant le fichier `/dev/ensea-led` à l'aide de la fonction `read`.

Modification du sens de balayage par écriture du fichier /proc/ensea/dir

```
    proc_entry_dir = proc_create("dir", 0666, sub_entry, &ensea_dir_fops);
    if (proc_entry_dir == NULL) {
        pr_err("Failed to create /proc/ensea/dir entry\n");
    }
```

通过proc_create函数我们在ensea文件夹下面创建dir文件 并与ensea_dir_fops结构体绑定

```
static void my_timer_callback2(struct timer_list *timer)
{

    static int y2 = 0;
    static int i2 = 0;

    printk("%s called (%ld)\n", __func__, jiffies);

    dev->leds_value = (0x80 >> i2);
    iowrite32(dev->leds_value, dev->regs);
    i2++;  // Move to the next LED value

    if (i2 == 8) {
        i2 = 0; // Reset i to 0 when all LEDs are processed
        y2++;   // Move to the next iteration of the outer loop
    }

    if (y2 < 5) {
        // If not all iterations are completed, set the timer for the next iteration
        mod_timer(&my_timer2, jiffies + msecs_to_jiffies(speed));
    } else {
        // All iterations are completed, reset y and stop the timer
        y2 = 0;
    }


}
```

```c
static void my_timer_callback(struct timer_list *timer)
{

    static int y = 0;
    static int i = 0;

    printk("%s called (%ld)\n", __func__, jiffies);

    dev->leds_value = (0x01 << i);
    iowrite32(dev->leds_value, dev->regs);
    i++;  // Move to the next LED value

    if (i == 8) {
        i = 0;  // Reset i to 0 when all LEDs are processed
        y++;   // Move to the next iteration of the outer loop
    }

    if (y < 5) {
        // If not all iterations are completed, set the timer for the next iteration
        mod_timer(&my_timer, jiffies + msecs_to_jiffies(speed));
    } else {
        // All iterations are completed, reset y and stop the timer
        y = 0;
    }


}
```

Dans différentes fonctions de rappel du timer, nous modifions la direction du chenillard en attribuant des valeurs initiales différentes au registre des LEDs.

Récupération du sens de balayage par lecture du fichier /proc/ensea/dir

```c
static ssize_t dir_read(struct file *file, char *buffer, size_t len, loff_t *offset)
{
    int success = 0;
    char result[100] = {0};
    sprintf(result, "dir: %s\r\n", directe);
    success = copy_to_user(buffer, &result, sizeof(result));
    // If we failed to copy the value to userspace, display an error message
    if(success != 0) {
        pr_info("Failed to return current led value to userspace\n");
        return -EFAULT; // Bad address error value. It's likely that "buffer" doesn't point to a good address
    }
    return 0;
```

À travers la fonction `dir_read`, nous transmettons la direction actuelle à l'espace utilisateur en utilisant la fonction `copy_to_user`.

Dans l'espace utilisateur, nous utilisons la fonction `read` pour lire la direction actuelle.

```c
    fd2 = open("/proc/ensea/dir", O_RDONLY);

    if (fd2 < 0) {
        perror("Error opening file dir");
        exit(EXIT_FAILURE);
    }

    if (read(fd2, &data_dir, sizeof(data_dir)) < 0) {
        perror("Error read  file");
        close(fd2);
        exit(EXIT_FAILURE);
    }
    printf("%s\r\n",data_dir);
    close(fd2);

    return 0;
}
```

La capture d'écran ci-dessous montre les résultats de sortie. Nous avons créé un répertoire `ensea` dans le dossier `proc` et avons créé les fichiers `dir` et `speed` à l'intérieur.

```
root@DE10-Standard:~/JiangboWANG_KaixuanJIANG_ESE_LiunxEmbarque/tp3/jkxfolder# cd /proc
root@DE10-Standard:/proc# ls
1      1551  1811  1925  331  666   827        cgroups      fb           loadavg      stat
10     16    1814  1928  332  6691  828        cmdline      filesystems  locks        swaps
10202  1610  1815  1951  334  684   829        config.gz    fs           meminfo      sys
10204  1634  1826  1980  388  688   842        consoles     interrupts   misc         sysrq-trigger
11     1665  1830  1985  389  691   855        cpu          iomem        modules      sysvipc
13     1671  1831  1995  471  692   856        cpuinfo      ioports      mounts       thread-self
14     1735  1838  2     473  7     864        crypto       irq          mtd          timer_list
1406   1747  1840  2006  488  733   881        device-tree  kallsyms     net          tty
1439   1749  1843  2022  489  743   892        devices      key-users    pagetypeinfo uptime
1485   1762  1847  2050  5    753   9          diskstats    keys         partitions   version
15     1770  1852  3     552  8     asound     driver       kmsg         self         vmallocinfo
1547   1773  1889  328   6    8149  buddyinfo  ensea        kpagecount   slabinfo     vmstat
1550   1782  1913  329   662  821   bus        execdomains  kpageflags   softirqs     zoneinfo
root@DE10-Standard:/proc# cd ensea
root@DE10-Standard:/proc/ensea# ls
dir  speed
root@DE10-Standard:/proc/ensea#
```

Nous avons également enregistré avec succès le périphérique de caractères misc appelé `ensea_leds`.

```
root@DE10-Standard:/dev# ls
block            initctl            pts    ram0           tty14  tty3   tty45  tty60  ttyp8    vcsa
bus              input              ptyp0  ram1           tty15  tty30  tty46  tty61  ttyp9    vcsa1
char             kmem               ptyp1  random         tty16  tty31  tty47  tty62  ttypa    vcsa2
console          kmsg               ptyp2  shm            tty17  tty32  tty48  tty63  ttypb    vcsa3
cpu_dma_latency  log                ptyp3  snd            tty18  tty33  tty49  tty7   ttypc    vcsa4
disk             mem                ptyp4  spidev32763.0  tty19  tty34  tty5   tty8   ttypd    vcsa5
ensea_leds       memory_bandwidth   ptyp5  spidev32766.0  tty2   tty35  tty50  tty9   ttype    vcsa6
f2h-dma-memory   mmcblk0            ptyp6  stderr         tty20  tty36  tty51  ttyS0  ttypf    vcsa7
fb0              mmcblk0p1          ptyp7  stdin          tty21  tty37  tty52  ttyS1  urandom  vhci
fd               mmcblk0p2          ptyp8  stdout         tty22  tty38  tty53  ttyp0  vcs      watchdog
full             mmcblk0p3          ptyp9  tty            tty23  tty39  tty54  ttyp1  vcs1     zero
i2c-0            network_latency    ptypa  tty0           tty24  tty4   tty55  ttyp2  vcs2
i2c-1            network_throughput ptypb  tty1           tty25  tty40  tty56  ttyp3  vcs3
i2c-2            null               ptypc  tty10          tty26  tty41  tty57  ttyp4  vcs4
i2c-3            psaux              ptypd  tty11          tty27  tty42  tty58  ttyp5  vcs5
i2c-4            ptmx               ptype  tty12          tty28  tty43  tty59  ttyp6  vcs6
i2c-5            ptp0               ptypf  tty13          tty29  tty44  tty6   ttyp7  vcs7
```

La capture d'écran ci-dessous montre les résultats du test. Lorsque nous exécutons le fichier usr.c, nous pouvons voir l'impression réussie de la vitesse actuelle, du mode et de la direction.

```
root@DE10-Standard:~/JiangboWANG_KaixuanJIANG_ESE_LiunxEmbarque/tp3/jkxfolder# ./usr
Speed: 100

 Description:  LED en mode chenillard
dir: LED dans le sens de droite a gauche

root@DE10-Standard:~/JiangboWANG_KaixuanJIANG_ESE_LiunxEmbarque/tp3/jkxfolder#
```

```
root@DE10-Standard:~/JiangboWANG_KaixuanJIANG_ESE_LiunxEmbarque/tp3/jkxfolder# insmod gpio_leds.ko speed=
1000
root@DE10-Standard:~/JiangboWANG_KaixuanJIANG_ESE_LiunxEmbarque/tp3/jkxfolder# ./usr
Speed: 1000

 Description:  LED en mode chenillard
dir: LED dans le sens de droite a gauche

root@DE10-Standard:~/JiangboWANG_KaixuanJIANG_ESE_LiunxEmbarque/tp3/jkxfolder#
```

4 Petit projet : Afficheurs 7 segments

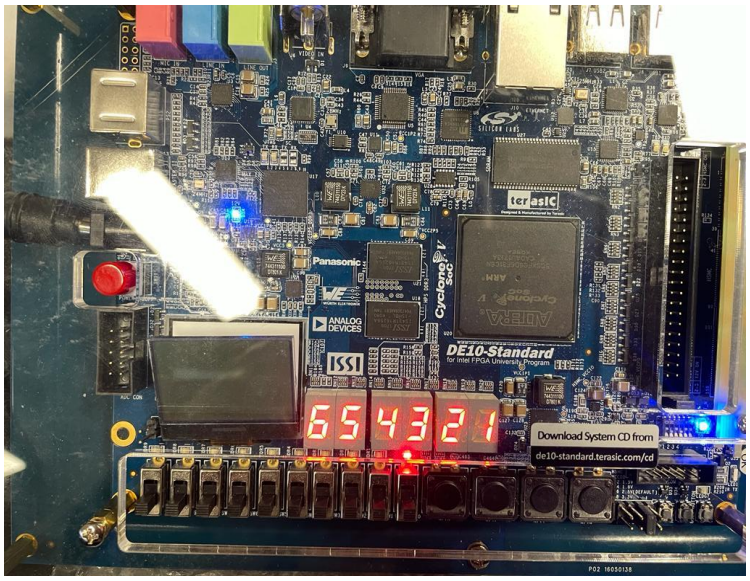
En vous inspirant du début du TP2 (Section 2.1), faites fonctionner les afficheurs 7 segments. Vous utiliserez la fonction mmap().

```c
#define HW_REGS_BASE            ( 0xfc000000 )
#define HW_REGS_SPAN            ( 0x04000000 )
#define HW_REGS_MASK            ( HW_REGS_SPAN - 1 )
#define FPGA_HEX_BASE           0x33000
#define  ALT_LWFPGASLVS_OFST    0xff200000

uint8_t *m_hex_base;

static int FPGAInit()
{
 int m_file_mem;

    m_file_mem = open( "/dev/mem", ( O_RDWR | O_SYNC ) );
    if (m_file_mem != -1){
        void *virtual_base;
        virtual_base = mmap( NULL, HW_REGS_SPAN, ( PROT_READ | PROT_WRITE ), MAP_SHARED, m_file_mem, HW_REGS_BASE );
        if (virtual_base == MAP_FAILED){
        }else{
            m_hex_base= (uint8_t *)virtual_base + ( ( unsigned long  )( ALT_LWFPGASLVS_OFST + FPGA_HEX_BASE ) & ( unsigned long)( HW_REGS_
            //0xfc000000+(0xff200000+0x33000)&(0x04000000-1)
        }
        close(m_file_mem);
    }

    return 0;
}
```

Dans l'espace utilisateur, nous ouvrons le fichier `/dev/mem` et utilisons `mmap` pour mapper l'adresse réelle du segment en une adresse virtuelle.

```
static int  HexSet(int index, int value){

    uint8_t szMask[] = {
        63, 6, 91, 79, 102, 109, 125, 7,
        127, 111, 119, 124, 57, 94, 121, 113, 64, 0
    };

    if (value < 0)
        value = 0;
    else if (value > 17)
        value = 17;

    //qDebug() << "index=" << index << "value=" << value << "\r\n";

    *((uint32_t *)m_hex_base+index) = szMask[value];
    return 0;
}

int main()
{

    FPGAInit();
    HexSet(0,1);
    HexSet(1,2);
    HexSet(2,3);
    HexSet(3,4);
    HexSet(4,5);
    HexSet(5,6);
    return 0 ;
}
```

Et en utilisant la fonction Hexset, nous avons réussi à afficher les chiffres 123456 sur le FPGA.



4.2 Device Tree et module

Écrire un module permettant d’afficher l’heure sur l’afficheur.

Nous avons créé notre propre module et y avons enregistré la structure de périphérique de plateforme (`afficheur`).

```
static int __init my_module_init(void) {

        int ret_val = 0;
        ret_val = platform_driver_register(&afficheur_platform);
        if(ret_val != 0) {
            pr_err("platform_driver_register returned %d\n", ret_val);
            return ret_val;
        }

        pr_info("Ensea afficheur module successfully initialized!\n");

        return 0;
    }

static void __exit my_module_exit(void) {
    pr_info("Exiting Kernel Module\n");

    // Unregister our driver from the "Platform Driver" bus
    // This will cause "leds_remove" to be called for each connected device
    platform_driver_unregister(&afficheur_platform);

    pr_info("Ensea afficheur module successfully unregistered\n");
}
```

En même temps, nous avons défini le pilote de plateforme (`plateform_drive`) pour correspondre au périphérique défini dans le device tree.

```
// Specify which device tree devices this driver supports
static struct of_device_id ensea_afficheur_dt_ids[] = {
    {
        .compatible = "dev,time"
    },
    { /* end of table */ }
};

// Inform the kernel about the devices this driver supports
MODULE_DEVICE_TABLE(of, ensea_afficheur_dt_ids);

// Data structure that links the probe and remove functions with our driver
static struct platform_driver afficheur_platform = {
    .probe = afficheur_probe,
    .remove = afficheur_remove,
    .driver = {
        .name = "Ensea Afficheur Driver",
        .owner = THIS_MODULE,
        .of_match_table = ensea_afficheur_dt_ids
    }
};
```

Lorsque le pilote (`driver`) ici correspond avec succès à la propriété `compatible` du périphérique définie dans le device tree, la fonction `probe` sera appelée.

Dans le fichier `.dts`, nous avons ajouté un nœud pour le périphérique `afficheur`.

```
        afficheur: time {
            #address-cells = <1>;
            #size-cells = <1>;
            compatible = "dev,time";
            reg = <0xfc000000  0x04000000>;

        };
```

Dans la fonction `probe`, nous avons enregistré `afficheur` en tant que périphérique de caractères misc et l'avons associé à la structure `ensea_afficheur_fops`. Nous avons également utilisé la fonction `ioremap` pour convertir l'adresse physique en une adresse virtuelle pour l'accès.

```
static int afficheur_probe(struct platform_device *pdev)
{
        int ret_val = -EBUSY;
        struct resource *r = 0;

        pr_info("afficheur_probe enter\n");

        // Get the memory resources for this LED device
        r = platform_get_resource(pdev, IORESOURCE_MEM, 0);
        if(r == NULL) {
            pr_err("IORESOURCE_MEM (register space) does not exist\n");
            goto bad_exit_return;
        }

        // Create structure to hold device-specific information (like the registers)
        dev = devm_kzalloc(&pdev->dev, sizeof(struct ensea_afficheur_dev), GFP_KERNEL);

        // Both request and ioremap a memory region
        // This makes sure nobody else can grab this memory region
        // as well as moving it into our address space so we can actually use it
        dev->regs = devm_ioremap_resource(&pdev->dev, r);
        if(IS_ERR(dev->regs))
            goto bad_ioremap;

        // Initialize the misc device (this is used to create a character file in userspace)
        dev->miscdev.minor = MISC_DYNAMIC_MINOR;    // Dynamically choose a minor number
        dev->miscdev.name = "ensea_afficheur";
        dev->miscdev.fops = &ensea_afficheur_fops;

        ret_val = misc_register(&dev->miscdev);
        if(ret_val != 0) {
            pr_info("Couldn't register misc device :(");
            goto bad_exit_return;
        }
```

```c
static const struct file_operations ensea_afficheur_fops = {
    .owner = THIS_MODULE,
    .read = afficheur_read,
};
```

Dans la fonction `afficheur_read`, nous utilisons la fonction `ktime_get_real_ts64` pour obtenir la valeur actuelle du temps, puis nous convertissons cette valeur en année, mois, jour, et envoyons les chiffres correspondants à l'espace utilisateur.

```c
static ssize_t afficheur_read(struct file *file, char *buffer, size_t len, loff_t *offset)
{
    int success = 0;   char result[100];    int tens = 0 ;  int ones = 0 ;   struct timespec64 ts;   struct tm tm;

    ktime_get_real_ts64(&ts);
    time_to_tm(ts.tv_sec, 0, &tm);
//  [ 123.456789] Current Kernel Time: 2023-12-16 14:30:15
    pr_info("Current Kernel Time: %ld-%02d-%02d %02d:%02d:%02d\n",
            tm.tm_year + 1900, tm.tm_mon + 1, tm.tm_mday,
            tm.tm_hour, tm.tm_min, tm.tm_sec);

    sprintf(result, "Current Kernel Time: %ld-%02d-%02d %02d:%02d:%02d\n",
            tm.tm_year + 1900, tm.tm_mon + 1, tm.tm_mday,
            tm.tm_hour, tm.tm_min, tm.tm_sec);
//days
    tens = tm.tm_mday / 10;
    ones = tm.tm_mday % 10;
    HexSet(0,tens);
    HexSet(1,ones);
//hour
    tens = tm.tm_hour / 10;
    ones = tm.tm_hour % 10;
    HexSet(2,tens);
    HexSet(3,ones);
//min
    tens = tm.tm_sec / 10;
    ones = tm.tm_sec % 10;
    HexSet(4,tens);
    HexSet(5,ones);
    // Give the user the current time
    success = copy_to_user(buffer, &result, sizeof(result));
    // If we failed to copy the value to userspace, display an error message
```

Dans l'espace utilisateur, nous utilisons la fonction `read` pour lire la valeur actuelle du temps.

```c
int main() {
    int fd ;
    char data_time[100] = {0};

    fd = open("/dev/ensea_afficheur", O_RDONLY);

    if (fd < 0) {
        perror("Error opening file");
        exit(EXIT_FAILURE);
    }

    if (read(fd, &data_time, sizeof(data_time)) < 0) {
        perror("Error read  file");
        close(fd);
        exit(EXIT_FAILURE);
    }
    printf("%s\r\n",data_time);
    close(fd);
}
```