TP - OS KERNEL

Introduction:

Nous souhaitons, à l'aide d'une carte BeagleBone Black, piloter un moteur grâce à une modulation de largeur d'amplitude (PWM) et être capable de connaitre sa vitesse de rotation grâce à un capteur compteur de tour.

Prise en main de la carte BeagleBoneBlack :

Dans un premier temps, nous avons à prendre en main la carte BeagleBone Black. C'est une carte à faible coût capable de réaliser des calculs arithmétiques de vecteurs avec virgule flottante. Elle est donc toute indiquée pour la réalisation de robots ludiques pour la gestion de leur motorisation ainsi que des différents capteurs ou caméras.

Une distribution Linux Debian est flashée dans la mémoire eMMC et un port microSD est présent, ce qui va nous permettre de pouvoir ajouter des drivers.

Pour comprendre comment cette carte fonctionne, commençons d'abord par piloter une simple broche numérique en sortie.

Nous avons utilisé ce tutoriel :

http://www.armhf.com/using-beaglebone-black-gpios/

On a branché les anodes aux pins 12, 14, 15, 16 du header P9 et les cathodes à DGND.

Grâce à cette simple première utilisation, nous commençons déjà à comprendre comment fonctionne la carte BBB et comment l'on peut interagir avec.

Essayons à présent de piloter une LED à l'aide d'une PWM.

Nous avons utilisé ce tutoriel :

http://www.blaess.fr/christophe/2013/07/06/beaglebone-black-et-pwm/avec la LED déjà branchée sur la pin 14.

Et pour faire clignoter la LED, nous avons suivi les valeurs présentes sur ce site (period/duty) : http://elinux.org/EBC_Exercise_13_Pulse_Width_Modulation

Compilation de fichier:

Maintenant que cette première approche de la carte est terminée, nous avons besoin de pouvoir compiler du code.

Nous avons completé le PATH du Kernel dans le fichier Makefile :

KDIR := /home/sasl/shared/EISE4-OSK-LSB/src/bb-kernel/KERNEL

Et compiler tel que :

 $3305058@pc4161: {\tt ~/Documents/STB/srcTP/linux-gpio-irq-latency-test\$ make ARCH=arm CROSS_COMPILE=/home/sasl/shared/EISE4-OSK-LSB/bin/gcc-linaro-arm-linux-gnueabihf-4.9-2014.06_linux/bin/arm-linux-gnueabihf-$

Interruption sur la BeagleBone Black:

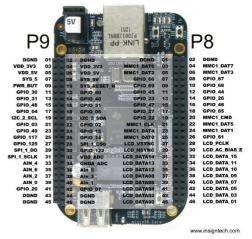
Nous utilisons le fichier test-irq-latency.c où l'on configure les pins utilisées (celle qui va servir à générer l'interruption, et celle qui sera utilisée dans l'interruption pour vérifier le bon fonctionnement) Nous utilisons donc le registre AM33XX_CONTROL_BASE (fonction setup_pinmux())

Pour comprendre comment l'interruption fonctionne, ce site nous a très bien expliqué (choix des pins, décalage pour la configuration voulue sur le registre) :

http://derekmolloy.ie/gpios-on-the-beaglebone-black-using-device-tree-overlays/

Ajout du driver :

- 1- On ajoute la carte SD sur la carte BeagleBone Black
- 2- On relie les câbles pour la communication série (Debug)



Accès série : screen /dev/ttvUSB0 115200

- 3- Puis l'on redémarre la carte BBB pour boot sur la carte SD
- 4- On se connecte avec l'identifiant « root » et password « root »
- 5- On copie notre fichier .ko dans le dossier /tmp/test/ (scp 3305058@192.168.7.1:<fichier>)

Si ReadOnly, il faut faire un fsck /dev/mmcblk0p1 puis mount -oremount,rw /dev/mmcblk0p1 /

- 6- On installe ensuite grâce à : insmod <nom du module>
- 7- On créer le fichier correspondant : mknod /dev/<nom du module> c 124 0
- 8- Puis on donne l'acces : chmod 700 /dev/<nom du module>

Char Device:

Nous avons implémenté les fonctions(open, close, read, write) vues durant les TPs précédents. Pour tester notre code, nous allons commencer à LIRE le compteur généré dans l'interruption déclenché par la connexion de la broche 15 au VDD3V3 de la carte.

Dans la console, on peut vérifier que chaque connexion a incrémenté le compteur grâce à :

cat < /dev/<nom du module> (ou dans les logs de 'dmsg')

Nous avons incrémenté un char, donc nous affichons bien : a b c d

Modification code pour gérer la PWM via le code de test-irq-latency.c :

```
On déclare les pins utilisés :
#define AM33XX_CONTROL_BASE 0x44e10000
static u32 pins[] = \{
   AM33XX_CONTROL_BASE + 0x878, // test pin (60): gpio1_28 (beaglebone p9/12)
                          mode 7 (gpio), PULLUP, OUTPUT
                      //
   0x7 | (2 << 3),
   AM33XX_CONTROL_BASE + 0x840, // irq pin (48): gpio1_16 (beaglebone p9/15)
   0x7 | (2 << 3) | (1 << 5), // mode 7 (gpio), PULLUP, INPUT
   AM33XX_CONTROL_BASE + 0x848, // pwm pin (40): gpio1[18](beaglebone p9/14)
                                                      mode 6 (PWM), PULLUP, OUTPUT
   0x6 | (2 << 3),
 for (i=0; i<6; i+=2) {
   void* addr = ioremap(pins[i], 4);
Nous utilisons les pins 12(GPIO), 14(PWM) et 15(Interruption) du header P9.
Les registres que nous allons utilisés sont :
#define PWMSS1 MMAP ADDR 0x48302000
#define EPWM TBPRD 0xA
#define EPWM CMPA 0x12
#define EPWM_CMPB 0x14
#define EPWM_TBCTL 0x0
#define EPWM_TBCNT 0x8
#define TBCTL_CTRMODE_UP 0x0
Mode UP: On commence le timer à 0 et on incrémente jusqu'à la valeur de la période. Puis on reset, et on
recommence...
#define TB DIV1 0x0
Soit
        TBPRD => Période
        CMPA et CMPB => Gérer le rapport cyclique
Pour modifier la période, on écrit dans : PWMSS1 MMAP ADDR + EPWM TBPRD
Pour modifier CMPA, on écrit dans : PWMSS1 MMAP ADDR + EPWM CMPA
Pour modifier CMPB, on écrit dans : PWMSS1_MMAP_ADDR + EPWM_CMPB
Pour choisir le Mode UP : PWMSS1_MMAP_ADDR + EPWM_TBCTL = TBCTL_CTRMODE_UP
Pour la configuration de TBCTL, HSPCLKDIV et CLKDIV = TB_DIV1
Pour avoir un rapport cyclique précis, il faut que TBPRD soit le plus proche de 65535 (car l'on augmente le range
```

de CMPA et CMPB)

```
EX : 20.25% duty cycle
on TBPRD = 62500 , CMPA = 12656.25 ( .25 rejection) , real duty : 20.2496% (12656 /62500)
on TBPRD = 6250 , CMPA = 1265.625 ( .625 rejection), real duty : 20.24% (1265 6250)
on TBPRD = 500, CMPA = 101.25 ( .25 rejection), real duty: 20.2% (101/500)
```

On règle: TBPRD = 62500, CMPA = 31250, CMPB = 31250 => Rapport cyclique = 50 % Compréhension des registres grâce à :

https://github.com/VegetableAvenger/BBBIOlib/blob/master/BBBio lib/BBBiolib PWMSS.c et la documentation constructeur page 1523.

Conclusion:

Ce mini-projet nous a permis de découvrir comment nous approprier une carte du marché et de pouvoir s'en servir très rapidement. A travers ce TP, nous avons pu appliquer des notions vues en cours comme l'importation du driver et son utilisation en char device.