Vous devrez le créer. Ceci nous donne accès au contrôle des leds D3 et D4 sur le devkit mais la configuration n'est, là encore, pas correct. En effet, le script **start** fait appel à **echo** mais il s'agit d'une commande interne du shell BusyBox et non un lien symbolique dans **/bin**, **/bin/echo** n'a donc aucun sens et ne peu fonctionner, un simple **echo** suffit.

Mais le contenu de ce fichier est très intéressant, car il montre qu'il est possible de contrôler les GPIO du devkit depuis l'espace utilisateur et même tout simplement depuis le shell via **/sys**. Les leds soudées sur la carte, respectivement sur les broches PG13 et PG14 correspondent, au sein de la nomenclature du noyau aux GPIO 109 et 110. Le script **start** ne fait que configurer l'exportation de ces deux lignes, les passer en « sortie » et les initialiser à 0 (*off*). Ainsi pour allumer ces leds, tout ce que vous avez à faire c'est :

```
echo 1 > /sys/class/gpio/gpio109/value
echo 1 > /sys/class/gpio/gpio110/value
```

Mais il est possible de faire quelque chose de plus intéressant avec ces leds que de tenter de les contrôler de cette façon (qui est davantage adaptée à de simples GPIO non connectée à quoi que ce soit et surtout des leds). En effet, µClinux comme Linux met à disposition une infrastructure spécifique pour ce type de leds de signalisation et en particulier lorsqu'elles sont connectées à des GPIO....

4 Blink me ! Blink me !

A l'idée de changer l'état d'une led je vois votre oeil s'allumer. Oui tout le monde aime les leds mais le point ici n'est pas d'utiliser 225 DMIPS pour faire clignoter un tel composant (pour cela certains diraient qu'il existe déjà l'Arduino *Due* ou l'Intel *Galileo*) mais de montrer qu'il peut être intéressant de disposer d'une API et d'un noyau aussi courant que Linux/µClinux. Ainsi, en guise de témoin d'activité ou d'état de tout ou partie du système, le noyau intègre la classe de périphériques leds. Celle-ci se traduit en espace utilisateur en une structure placée dans **/sys/class/leds** disposant d'un répertoire par leds pilotés contenant différents fichiers :

```
~ # ls /sys/class/leds/LD3/
brightness      max_brightness  trigger
device          subsystem       uevent
```

brightness par exemple, nous permet de contrôler l'intensité lumineuse d'une led entre **0** et le contenu de **max_brightness**, qui dans le cas d'une simple led (sans PWM) sera **1** (on/off). Une simple utilisation de la commande **echo** suivie d'une valeur et redirigée vers le fichier **brightness** impactera l'état de la led en question. Mieux encore, le contenu de **trigger** est plus intéressant :

```
~ # cat /sys/class/leds/LD4/trigger
none timer [heartbeat]
```

Ce pseudo-fichier contient le nom d'un déclencheur (*trigger*) permettant de changer l'état. Ici **none** nous donne le comportement spécifié précédemment, **timer** nous permet de définir un clignotement avec une durée d'allumage et d'extinction et **heartbeat** simule un battement de coeur au rythme de la charge du système. Le contrôle s'opère de la même manière via **cat** et **echo** en listant le contenu du fichier ou en y plaçant l'un ou l'autre nom de déclencheur.

Imaginons maintenant que nous souhaitons opérer de la sorte avec nos deux leds qui, pour l'instant, sont directement pilotés par le système de gestion GPIO d'µClinux. La structure même du noyau ainsi que le travail déjà effectué sur les sources par Emcraft Systems et Robutest/tmk sur les GPIO nous facilite la tâche au point de n'avoir finalement qu'à déclarer l'existence des leds en question auprès du sous-système dédié.

On commencera donc par écrire le code permettant l'enregistrement de nos périphériques dans un nouveau fichier nommé **arch/arm/mach-stm32/leds.c** :

```
#include <mach/leds.h>
#include <mach/platform.h>

#include <linux/leds.h>
#include <linux/platform_device.h>

// déclaration des broches et noms
static struct gpio_led stm32f429idisco_led_pins[] = {
  {
    .name           = "LD3",
    .gpio           = 109,
  },
  {
    .name           = "LD4",
    .gpio           = 110,
  },
};

// déclaration du périphérique leds
static struct gpio_led_platform_data stm32f429idisco_led_data = {
  .num_leds             = ARRAY_SIZE(stm32f429idisco_led_pins),
  .leds                 = stm32f429idisco_led_pins,
};

// déclaration du périphérique
static struct platform_device stm32f429idisco_leds = {
  .name                 = "leds-gpio",
  .id                   = -1,
  .dev.platform_data    = &stm32f429idisco_led_data,
};

// intégration
static struct platform_device *stm32f429idisco_devices[] __initdata = {
```