```
  &stm32f429idisco_leds,
};

// initialisation
void __init stm32_led_init(void)
{
        int platform;
        platform = stm32_platform_get();
        if(platform == PLATFORM_STM32_STM32429_DISCO)
          platform_add_devices(stm32f429idisco_devices,
            ARRAY_SIZE(stm32f429idisco_devices));
}
```

Notez que dans la succession de déclarations nous pointons la structure précédente, ce qui nous donne une représentation arborescente plateforme / périphérique / périphérique-leds / broches. Remarquez le test effectué sur la valeur retournée par **stm32_platform_get()**. Celui-ci repose sur une information initialement transmise par le bootloader mais la déclaration de **PLATFORM_STM32_STM32429_DISCO** doit cependant être présente (**arch/arm/mach-stm32/include/mach/platform.h**). Une condition similaire (**switch/case**) est déjà utilisée par ailleurs pour le support initiale de l'USART3 du devkit.

Nous créons également le *header* pour le prototypage de la fonction d'initialisation, **arch/arm/mach-stm32/include/mach/leds.h** :

```
#include <linux/init.h>

void __init stm32_led_init(void);
```

Puis nous modifions le source principal concernant la plateforme pour y intégrer l'appel à **stm32_led_init()** (d'où la nécessité du header), conditionné par l'activation de l'option adéquate dans la configuration du noyau, **arch/arm/mach-stm32/stm32_platform.c** :

```
[...]
#include <mach/leds.h>
[...]
#if defined(CONFIG_LEDS_GPIO)
        /*
         * Register leds class device
         */
        stm32_led_init();
#endif
```

Enfin, nous n'oublions pas de modifier le **arch/arm/mach-stm32/Makefile** de la plateforme pour y ajouter notre nouveau fichier source parmi les autres :

```
[...]
obj-$(CONFIG_LEDS_GPIO)  += leds.o
[...]
```

Et.... c'est tout ! Il nous suffira ensuite de configurer notre noyau comme nous l'avons abordé précédemment dans l'article afin d'activer le support **CONFIG_NEW_LEDS** via **CONFIG_LEDS_GPIO** et les déclencheurs (*triggers*) qui nous intéressent (**CONFIG_LEDS_TRIGGER_TIMER** et **CONFIG_LEDS_TRIGGER_HEARTBEAT** en particulier). Eventuellement, afin de faire les choses proprement, nous pouvons modifier **/etc/start** afin de désactiver l'exportation des ports 110 et 109 qui ne pourront plus être utilisés (sous peine de **write error: Device or resource busy**). Dès mise à jour du noyau et éventuellement du rootfs sur le devkit, un simple **echo heartbeat > /sys/class/leds/LD4/trigger** et la led rouge battra au rythme de l'activité du système.

Je pense que ceci est très représentatif, même si relativement anodin, des avantages de l'utilisation d'un système « connu » et de la factorisation des connaissances qui en découle. Linux/µClinux fourni une infrastructure qu'il suffit de compléter. Il en va de même si vous souhaitez intégrer d'autres fonctionnalités qu'il s'agisse de triggers supplémentaires ou d'une toute autre gamme de périphériques (SPI, SD, UART, etc). Il en va de même, bien entendu pour des aspects purement logiciels comme le support de système de fichiers, d'algorithmes de chiffrement ou encore d'une pile de communication USB, Bluetooth, IP...

Conclusion (de l'article)

Vous l'avez compris, tout l'intérêt ici est de pouvoir reposer sur le travail d'autrui et surtout ses compétences. C'est tout le principe de l'opensource et du logiciel libre qui prend forme sous vos yeux. µClinux sur STM32 peut donc effectivement avoir un réel intérêt et n'est en aucun cas un simple caprice de développeur obsédé par le fait de voir fonctionner son système préféré absolument partout. L'article s'arrêtera là mais nous vous invitons à regarder cela de plus près et éventuellement compléter le travail des développeurs concernant ce devkit, avant de diffuser vos modifications comme tmk et jserv l'ont fait. Ceci présente à la fois un intérêt pratique mais également pédagogique puisqu'il s'agit là d'une très bonne façon d'apprendre à utiliser chaque sous-système de Linux/µClinux, étape par étape. Le devkit dispose en effet de bon nombre de périphériques comme le *MEMS motion sensor three-axis digital output gyroscope* L3GD20 interfacé en SPI sur le port SPI5 (avec CS sur PC1) ou encore le contrôleur pour touchscreen STM PE811 sur le troisième bus i2c (adresse 1000001) dont le support pourrait intégré le sous-système d'entrée (*input*). Le STM32F429I dispose même d'une interface Ethernet mais ceci demandera alors quelques modifications de la carte car la plupart des lignes sont d'ors et déjà utilisées pour d'autres périphériques (dont l'afficheur LCD). ■