- EXTI0 : 6,
- EXTI1 : 7,
- EXTI2 : 8,
- EXTI3 : 9,
- EXTI4 : 10,
- EXTI5-9 : 23,
- EXTI10-15 : 40.

Pour configurer une ligne d'un port comme générant une interruption, il faut :

- Choisir le registre et les bits correspondant au port utilisé. Dans le cas du bouton ici présent, c'est le port A et donc le registre **SYSCFG_EXTICR1** dont les bits **3:0** seront à **0000**, ce qui est le cas par défaut après reset.
- Configurer le masque d'interruptions pour activer celle correspondant à nos attentes. Ici, la ligne EXTI0 reliée au NVIC, générant l'interruption 6.
- Installer un gestionnaire d'interruption qui, au minimum effacera le drapeau de l'interruption concerné.

En jetant un coup d'oeil aux sources d'EmCraft, et en particulier au fichier **arch/arm/mach-stm32/exti.c**, on se rend compte que l'EXTI dispose déjà d'une implémentation ou plutôt un début d'implémentation. Nous avons en premier lieu une structure mappée à l'adresse **0x40013c00** contenant les 6 registres de configuration de l'EXTI (oui, c'est écrit « kinetis » un peu partout, pour savoir pourquoi, demandez à EmCraft) :

```
struct kinetis_exti_regs {
  u32 imr;   /* Interrupt mask register */
  u32 emr;   /* Event mask register */
  u32 rtsr;  /* Rising trigger selection register */
  u32 ftsr;  /* Falling trigger selection register */
  u32 swier; /* Software interrupt event register */
  u32 pr;    /* Pending register */
};

#define KINETIS_EXTI_BASE 0x40013c00
#define KINETIS_EXTI ((volatile struct kinetis_exti_
regs *) \
                      KINETIS_EXTI_BASE)
```

Ces registres déterminent, des bits 0 à 22, la manière de gérer les 23 lignes de l'EXTI, avec dans l'ordre :

- Le masque autorisant la génération d'interruption auprès du NVIC,
- le même masque pour la génération d'événement (non utilisé ici),
- la sélection du déclenchement sur front montant,
- et/ou sur front descendant,
- le registre permettant de provoquer une interruption par logiciel (comme si elle provenait d'une source matérielle) en passant à **1** le bit correspondant,
- et le registre contenant les drapeaux d'interruptions (ceux-là même que votre routine devra effacer).

Ce dernier point est d'ailleurs couvert par une fonction définie dans ce même fichier :

```
void stm32_exti_clear_pending(unsigned int line)
{
  if (line < STM32F2_EXTI_NUM_LINES)
        KINETIS_EXTI->pr = (1 << line);
}
EXPORT_SYMBOL(stm32_exti_clear_pending);
```

Enfin, nous avons une fonction, bien mal nommée, permettant d'activer ou désactiver l'interruption :

```
void stm32_exti_enable_int(unsigned int line,
int enable)
{
  if (line >= STM32F2_EXTI_NUM_LINES)
     goto out;

  if (enable) {
     stm32_exti_clear_pending(line);

     /* Enable trigger on rising edge */
     KINETIS_EXTI->rtsr |= (1 << line);
     /* Disable trigger on falling edge */
     KINETIS_EXTI->ftsr &= ~(1 << line);
     /* Enable interrupt for the event */
     KINETIS_EXTI->imr |= (1 << line);
  } else {
     /* Disable interrupt for the event */
     KINETIS_EXTI->imr &= ~(1 << line);
     /* Disable trigger on rising edge */
     KINETIS_EXTI->rtsr &= ~(1 << line);
     /* Disable trigger on falling edge */
     KINETIS_EXTI->ftsr &= ~(1 << line);

     stm32_exti_clear_pending(line);
  }

out:
        ;
}
EXPORT_SYMBOL(stm32_exti_enable_int);
```

En dehors du fait que ce **goto** est parfaitement horrible et brûle les yeux, on constate que cette fonction prend en paramètre la ligne reliant l'EXTI au NVIC et l'utilise pour placer à **1** le bit correspondant dans le registre de contrôle d'interruption pour enfin définir une interruption sur front montant uniquement (mais pourquoi !). Le code d'EmCraft se limite à cela et, comme vous pouvez le voir, il est fait totalement abstraction de la sélection de GPIO et donc de la configuration de **SYSCFG_EXTICR***.

Dans le cas qui nous occupe présentement ce n'est pas un problème, le bouton USER est connecté à PA0 et donc d'ors et déjà initialisé à **0000** après reset. Nous pouvons donc sereinement nous pencher sur la création d'un fichier **uclinux/arch/arm/mach-stm32/button.c** contenant, en premier lieu notre routine d'interruption :

```
#include <linux/init.h>
#include <linux/platform_device.h>
#include <linux/gpio.h>
#include <linux/interrupt.h>
#include <mach/stm32.h>
#include <mach/platform.h>
#include <mach/exti.h>
#include <mach/button.h>

#define STM32429_DISCO_BUTTON_INT 6
#define STM32429_DISCO_BUTTON_PORT 0
#define STM32429_DISCO_BUTTON_PIN 0
#define STM32429_DISCO_BUTTON_EXTILINE 0

static irqreturn_t stm32_bouton_isr(int irq, void *dev_id)
{
  printk("INFO: irq on blue button !\n");

  /* clear irq */
  stm32_exti_clear_pending(STM32429_DISCO_BUTTON_EXTILINE);

  return IRQ_HANDLED;
}
```

Rien de bien complexe ici, puisque le noyau met à notre disposition tout ce qui nous est nécessaire pour cette tâche. Notez cependant l'utilisation de la fonction **stm32_exti_clear_pending()** permettant de signifier au matériel que l'interruption a été traitée. Nous pouvons donc nous pencher sur la fonction d'initialisation de notre fonctionnalité :