divisons ensuite l'unique fonction d'activation (/désactivation) des lignes EXTI, en une première fonction d'activation :

```
int stm32_exti_enable_int(unsigned int line, int edge)
{
        if (line >= STM32F2_EXTI_NUM_LINES)
                return -EINVAL;

        stm32_exti_clear_pending(line);

        if (edge & STM32_EXTI_RISING)
                /* Enable trigger on rising edge */
                STM32_EXTI->rtsr |= (1 << line);

        if (edge & STM32_EXTI_FALLING)
                /* Enable trigger on falling edge */
                STM32_EXTI->ftsr |= (1 << line);

        /* Enable interrupt for the event */
        STM32_EXTI->imr |= (1 << line);

        return 0;
}
EXPORT_SYMBOL(stm32_exti_enable_int);
```

Celle-ci accepte en second argument, non plus l'état *enable* ou non mais le front montant ou descendant (défini dans `exti.h`). Le `goto` a, bien entendu, disparu et la fonction retourne `0` en cas de réussite et `-EINVAL` (*invalid value*) en cas d'arguments non compatibles. Nous procédons de même pour la fonction inverse :

```
int stm32_exti_disable_int(unsigned int line)
{
        if (line >=  STM32F2_EXTI_NUM_LINES)
                return -EINVAL;

        /* Disable interrupt for the event */
        STM32_EXTI->imr &= ~(1 << line);
        /* Disable trigger on rising edge */
        STM32_EXTI->rtsr &= ~(1 << line);
        /* Disable trigger on falling edge */
        STM32_EXTI->ftsr &= ~(1 << line);

        stm32_exti_clear_pending(line);

        return 0;
}
EXPORT_SYMBOL(stm32_exti_disable_int);
```

Il ne nous reste plus ensuite qu'à ajouter une fonction permettant de configurer le multiplexeur pouvant définir quel GPIO peut déclencher une interruption :

```
int stm32_exti_set_gpio(unsigned int port, unsigned int pin) {
  if (port >= STM32_F429_EXTI_NUM_PORT)
    return -EINVAL;

  if (pin >= STM32_F429_EXTI_NUM_PIN)
    return -EINVAL;

  if(pin > 11) {
    SYSCFG->exticr4 ^= (SYSCFG->exticr4 ^ (port << 4*(pin-(11+1))))
            & (STM32_SYSCFG_EXTI_MASK << 4*(pin-(11+1)));
    return 0;
  }

  if(pin > 7) {
   SYSCFG->exticr3 ^= (SYSCFG->exticr3 ^ (port << 4*(pin-(7+1))))
          & (STM32_SYSCFG_EXTI_MASK << 4*(pin-(7+1)));
    return 0;
  }

  if(pin > 3) {
    SYSCFG->exticr2 ^= (SYSCFG->exticr2 ^ (port << 4*(pin-(3+1))))
            & (STM32_SYSCFG_EXTI_MASK << 4 *(pin-(3+1)));
    return 0;
  }

  SYSCFG->exticr1 ^= (SYSCFG->exticr1 ^ (port << 4*pin))
          & (STM32_SYSCFG_EXTI_MASK << 4*pin);

  return 0;
}
```

Là encore nous testons les arguments reçus. Ce code est perfectible, ne serait que parce qu'il est intimement lié aux valeurs de `STM32_F429_EXTI_NUM_PORT` et `STM32_F429_EXTI_NUM_PIN`, respectivement à 11 et 16, correspondantes au nombre de port (A à K) et au nombre de lignes par port sur un STM32F429ZIT. Le reste est principalement du *bit twiddling* permettant de modifier la valeur des 4 bits de sélection du port via un masque pour chaque registre EXTI se présentant ainsi :

```
SYSCFG_EXTICR1 [reserved][exti 3][exti 2][exti 1][exti 0]
SYSCFG_EXTICR2 [reserved][exti 7][exti 6][exti 5][exti 4]
SYSCFG_EXTICR3 [reserved][exti11][exti10][exti 9][exti 8]
SYSCFG_EXTICR4 [reserved][exti15][exti14][exti13][exti12]
```

De ce fait, pour connecter un GPIO sur l'EXIT, il nous suffit maintenant d'appeler `stm32_exti_set_gpio()` en passant en argument le port (A=0, B=1, etc) et la broche du port (0, 1, etc) qui nous intéresse. Attention cependant, ici rien ne nous empêche d'utiliser des GPIO déjà brochés pour d'autres fonctionnalités (LCD, DRAM, SPI, etc), vous devrez vous référer aux schémas du devkit pour trouver une ligne utilisable.

Personnellement, j'ai opté pour PC3, inutilisé même en activant deux SPI et l'i2c (cf autres articles). La simple connexion d'un bouton poussoir avec une résistance de rappel sur le même principe que le bouton USER (PA0) et l'installation d'une interruption (IRQ 9) comme précédemment se fait sans le moindre problème pour peu que l'on ajoute dans le code la configuration de l'EXTI via `SYSCFG` avec `stm32_exti_set_gpio(2,3)`.

3 Arrêtons là

Il est possible d'aller encore plus loin, par exemple en cherchant à s'assurer que le GPIO sélectionné est bel et bien configuré en entrée, mais nous avons fait déjà ample connaissance avec des fonctionnalités très intéressantes de ce microcontrôleur. Les possibilités offertes pour la génération d'interruptions sur n'importe laquelle des 168 GPIO du STM32 ouvrent des perspectives intéressantes de développement. Bien entendu, il ne s'agit là que d'une fonctionnalité basique qui prendrait la forme de quelques lignes de C en développement *bare metal* mais n'oubliez pas que nous bénéficions ici, en plus, de toute l'API Linux. ■