- **pupdr** correspondant à **GPIOx_PUPDR** pour choisir une résistance de rappel à Vcc, à la masse, aux deux ou pas du tout.

Dans **iomux.c** les adresses sont utilisées sous la forme d'une structure **stm32_gpio_base** :

```
static const unsigned long stm32_gpio_base[] = {
  STM32F2_GPIOA_BASE, STM32F2_GPIOB_BASE,
  STM32F2_GPIOC_BASE, STM32F2_GPIOD_BASE,
  STM32F2_GPIOE_BASE, STM32F2_GPIOF_BASE,
  STM32F2_GPIOG_BASE, STM32F2_GPIOH_BASE,
  STM32F2_GPIOI_BASE
};
```

qui, dans la fonction **stm32f2_gpio_config(struct stm32f2_gpio_dsc *dsc, enum stm32f2_gpio_role role)** sera utilisée pour obtenir, dans **gpio_regs** un pointeur vers la fameuse structure **stm32f2_gpio_regs** avec **gpio_regs = (struct stm32f2_gpio_regs *)stm32_gpio_base[dsc->port]**. **dsc** est une structure regroupant le port sous forme d'entier non signé ainsi que la broche :

```
struct stm32f2_gpio_dsc {
  u32 port;  /* GPIO port */
  u32 pin;   /* GPIO pin  */
};
```

Enfin, **stm32f2_gpio_role** est une simple énumération utilisée au sein de **stm32f2_gpio_config()**, sous la forme d'un **switch/case** afin de procéder à la la configuration des registres **GPIOx_OTYPER** (**moder**), **GPIOx_OSPEEDR** (**speedr**) et **GPIOx_PUPDR** (**pupdr**).

Pour finir, c'est dans la fonction **stm32_iomux_init()** qu'est utilisé à de nombreuses reprises **stm32f2_gpio_config()** en fonction de la définition des macros de configuration du noyau (**CONFIG_STM32_SPI5**, **CONFIG_STM32_USART1**, **CONFIG_STM32_I2C3**, etc), et ce, dans un **switch/case** basé sur la plateforme détectée en fonction de la valeur retournée par **stm32_platform_get()** (qui tire ses informations du paramètre passé par le bootloader au noyau lors du démarrage).

Il en résulte, dans **iomux.c** et **iomux.h**, une magnifique imbrication de mapping, de conditions et de déclarations, pour le moins déroutante. Cependant, pour nos usages dans ce cas précis, la majeure partie du travail est faite et nous n'avons qu'à nous pencher sur le code de **stm32_iomux_init()** dans la condition **case PLATFORM_STM32_STM32429_DISCO:**. vous trouverez là une série de **#if defined()** regroupant tous les paramètres de configuration du noyau qu'il vous suffira alors de compléter. La majeure partie de ces macros se limitent à **#error IOMUX for STM32 I2C1 undefined** provoquant une erreur à la compilation si la fonctionnalité n'a pas été implémentée. C'est à vous qu'il revient de remplir les trous...

Ainsi, la première chose à faire pour pouvoir utiliser l'un des bus i2c mis à votre disposition consiste à repérer la macro **#if defined(CONFIG_STM32_I2C3)**. Nous allons en effet utiliser ici le bus sur lequel est connecté le contrôleur de *touchscreen* STMPE811 (cf les schémas du devkit). Nous implémenterons donc :

```
#if defined(CONFIG_STM32_I2C3)
  gpio_dsc.port = 0;      /* SCL */
  gpio_dsc.pin  = 8;      /* PA8 */
  stm32f2_gpio_config(&gpio_dsc, STM32F2_GPIO_ROLE_I2C3);

  gpio_dsc.port = 2;      /* SDA */
  gpio_dsc.pin  = 9;      /* PC9 */
  stm32f2_gpio_config(&gpio_dsc, STM32F2_GPIO_ROLE_I2C3);
#endif
```

Un bus i2c (ou TWI) n'utilise que deux lignes (en plus de l'alimentation et la masse bien entendu) pour communiquer. Cette caractéristique en fait un bus très intéressant car économe en GPIO par comparaison au SPI par exemple. Les composants sur un bus i2c sont adressés et donc ne nécessitent pas de ligne /CS supplémentaire (bref, c'est un bus). On retrouve donc ici la désignation des ports PA8 en guise d'horloge et PC9 pour les données (bidirectionnel). On précise également via **STM32F2_GPIO_ROLE_I2C3** l'utilisation alternative des deux GPIO qui, dès lors, ne se comporteront plus comme de simples entrées/sorties.

2 Installation du drivers i2c dans linux

Cette modification n'est bien entendu pas suffisante et nous devons nous pencher sur la déclaration du ou des périphériques pour notre plateforme dans **arch/arm/mach-stm32/i2c.c**. Le support de la fonctionnalité elle-même est déjà implémenté par EmCraft dans **drivers/i2c/busses/i2c-stm32.c**. Il s'agit là du pilote à proprement parler qui a pour tâche de prendre en charge le périphérique au plus bas niveau. Notre travail consistera donc uniquement à enregistrer (*register*) le ou les bus qui nous intéressent. A noter cependant que tmk ne semble pas tenir à jour son dépôt GitHub par rapport aux modifications portées sur le code d'EmCraft. Or le commit **432a7b76eea**, poussé par Vladimir Khusainov, corrige un problème important dans le pilote i2c pour STM32F4. Il sera de bon ton de récupérer la version EmCraft du fichier **i2c-stm32.c** pour écraser votre version actuelle (ou éventuellement appliquer vous-même les correctifs). Le problème concerne l'ordre dans lequel les initialisations sont faites et il s'avère que le contrôleur i2c était enregistré auprès du noyau avant d'être pleinement initialisé.

Le fichier **arch/arm/mach-stm32/i2c.c** a également fait l'objet d'une modification puisque, depuis le fork de tmk, EmCraft a ajouté le support d'un PCAL6416A de NXP (*expander* 16 E/S) en guise de démonstration/validation de l'implémentation.