PLM

Pierre Molinaro

23 mai 2015

Table des matières

Liste des tableaux

Liste des figures

Chapitre 1

Introduction

1.1 Cible

Chapitre 2

Déclaration des registres

La déclaration d'un registre obéit à une syntaxe particulière, ne serait-ce que parce que son adresse absolue doit y être spécifiée. Pour de nombreux registres, un bit ou un groupe de bits ont une signification particulière, et obtenir la valeur d'un champ ou modifier sa valeur est une opération courante.

À titre d'exemple, nous allons nous intéresser au registre `SYS_CSR` du processeur ARMv7-M. Le *manuel de référence de l'architecture ARMv7-M* [1] décrit ce registre comme indiqué à la figure 2.1. Cette description indique l'adresse du registre (`0xE000E010`), et les différents champs qui le composent.

2.1 Simple déclaration d'un registre

Pour déclarer le registre `SYS_CSR` (figure 2.1), on écrira simplement :

```
register SYS_CSR at 0xE000_E010 : UInt32
```

Le type `UInt32` qui est mentionné signifie que les valeurs écrites et lues de ce registre sont des entiers non signés de 32 bits. Tout type entier, signé ou non signé est autorisé.

Pour lire ou écrire ce registre, on le nomme comme s'il s'agissait d'une simple variable. Par exemple, pour activer le comptage du *Systick Timer* et l'interruption correspondante, il faut mettre à 1 les bits `CLKSOURCE`, `TICKINT` et `ENABLE`. On écrit donc :

```
SYST_CSR = (1 << 2) | (1 << 1) | (1 << 0)
```

Pour savoir si le bit `TICKINT` est activé, on écrit :

1. http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0403e.b/index.html

SysTick Control and Status Register (SYST_CSR)

Table B3-25 SysTick Control and Status Register – (0xE000E010)

Bits	R/W	Name	Function
[16]	RO	COUNTFLAG	Returns 1 if timer counted to 0 since last time this register was read. COUNTFLAG is set by a count transition from 1 => 0. COUNTFLAG is cleared on read or by a write to the Current Value register.
[2]	R/W	CLKSOURCE	0: clock source is (optional) external reference clock 1: core clock used for SysTick If no external clock provided, this bit will read as 1 and ignore writes.
[1]	R/W	TICKINT	If 1, counting down to 0 will cause the SysTick exception to be pended. Clearing the SysTick Current Value register by a register write in software will not cause SysTick to be pended.
[0]	R/W	ENABLE	0: the counter is disabled 1: the counter will operate in a multi-shot manner.

Figure 2.1 – *Registre de contrôle* `SYS_CSR` *intégré dans l'ARMv7*

```
let tickIntActif : Bool = (SYST_CSR & (1 << 1)) != 0
```

Ces écritures peuvent être rendues plus intelligibles en précisant la composition du registre `SYS_CSR` dans sa déclaration. C'est ce qui va être réalisé dans la section suivante.

2.2 Déclaration d'un registre et de ses champs

Il est possible de préciser la composition des champs entiers et booléens d'un registre :

```
register SYS_CSR at 0xE000_E010
  : UInt32 {15, COUNTFLAG, 13, CLKSOURCE, TICKINT, ENABLE}
```

Cette écriture n'est autorisée que si le type nommé (ici `UInt32`) est une type entier non signé. Les types signés (`Int32`, ...) sont interdits. Entre accolades, est décrite la composition du registre, bit par bit, en commençant par le poids fort. Un nombre entier signifie un nombre de bits inutilisés ; un identificateur nomme un bit particulier. Cette description correspond à celle de la figure 2.1 :

— les 15 bits de poids fort sont inutilisés ;
— le bit 16 est le bit `COUNTFLAG` ;

— les 13 bits suivants sont inutilisés ;
— les bits 2, 1 et 0 sont respectivement `CLKSOURCE`, `TICKINT` et `ENABLE`

Le compilateur vérifie que la description des champs définit exactement le nombre de bits du type nommé, ici les 32 bits du type `UInt32`.

2.2.1 Constantes associées aux champs booléens

Le délimiteur `::` permet de définir des constantes correspondant aux bits d'un registre. Par exemple :

```
SYST_CSR::CLKSOURCE // De type UInt32, égal à 4
```

Ces constantes ont le type du registre nommé.

Ainsi, pour activer le comptage du *Systick Timer* et l'interruption correspondante, il faut mettre à 1 les bits `CLKSOURCE`, `TICKINT` et `ENABLE`. On écrit donc :

```
SYST_CSR = SYST_CSR::CLKSOURCE | SYST_CSR::TICKINT | SYST_CSR::ENABLE
```

2.2.2 Accès en lecture

Pour accéder à la valeur d'un champ, on utilise la notation pointée en nommant ce champ :

```
let x : UInt32 = SYS_CSR.CLKSOURCE // 0 ou 4
```

Ceci effectue simplement un masquage de façon à isoler le bit demandé. Aucun décalage n'est réalisé. Comme le bit `CLKSOURCE` est le 2^e, le résultat est `0` ou `4`. Le type de l'expression `SYS_CSR.CLKSOURCE` est le type du registre `SYS_CSR`, donc `UInt32`.

Si l'on veut obtenir un résultat justifié à droite, on utilise en plus l'accesseur `.shift` :

```
let x : UInt32 = SYS_CSR.CLKSOURCE.shift // 0 ou 1
```

L'accesseur `.bool` permet d'obtenir une valeur booléenne correspondant à la valeur d'un bit d'un registre :

```
let x : Bool = SYS_CSR.CLKSOURCE.bool // false ou true
```

2.3 Champs entiers

L'accès à un champ s'effectue par la notation pointée `.`. Par exemple :

```
PORTA_PCR0.mux
```

renvoie un `UInt32` valant `0`, `0x100`, `0x200`, ..., `0x700`.

Pour obtenir une valeur justifiée à droite, on ajoute l'accesseur `.shift` :

```
PORTA_PCR0.mux.shift
```

La valeur renvoyée est alors comprise en `0` et `7`.

2.4 Attribut `@ro`

Un registre accepte l'attribut `@ro`, qui signifie qu'il est en lecture seule. Par exemple :

```
register @ro nom_registre at 0x1234 : UInt32
```

Toute tentative de faire figurer ce registre dans une construction qui provoque une écriture de celui-ci entraîne l'apparition d'une erreur de compilation.