TP03 – « Serpentine »

Rapport

Version 1.0 du 10 novembre 2016

Rial Jonathan, Alan Sueur
Classe I-2

Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

Table des matières

1 Données

1.1 Configuration des GPIO

Pour la configuration des ports GPIO, nous avons créé la structure de données suivante :

GPIO base adress	Segment bitset	State	gpio offset in padmux

Voici le code :

```
gpio_config:
        .long GPIO0, SEGA, 0, PAD_OFS_SEGA
        .long GPIO0, SEGB, 0, PAD_OFS_SEGB
        .long GPIO0, SEGC, 0, PAD_OFS_SEGC
        .long GPIO0, SEGD, 0, PAD_OFS_SEGD
        .long GPIO0, SEGE, 0, PAD_OFS_SEGE
        .long GPIO0, SEGF, 0, PAD_OFS_SEGF
        .long GPIO0, SEGG, 0, PAD_OFS_SEGG
        .long GPIO2, DIG1, 1, PAD_OFS_DIG1
        .long 0, 0, 0, 0
```

La dernière ligne qui a comme valeur que des zéros permet de détecter la fin de la structure de données.

1.2 Séquence de LED

Pour afficher le serpent sur l'affichage 7 segments, nous avons créé la structure de données similaire à celle de la configuration dont voici le code :

```
snake:
        .long SEGA
        .long SEGB
        .long SEGG
        .long SEGE
        .long SEGD
        .long SEGC
        .long SEGG
        .long SEGF
        .long 0
```

Là aussi la dernière ligne permet de détecter la fin de la structure de données.

1.3 Constante

Nous avons ajouté une constante en plus de celles déjà fournies qui est le nombre qu'il faut décrémenter pour le « sleep » :

```
#define TIMEOUT         0x8ffff
```

2 Méthodes

2.1 Configuration des GPIO

2.1.1 Configuration de l'état du pin

Voici notre code qui nous permet de configurer l'état du pin :

```
cmp     r2, #1
streq   r1, [r0,#SET]
strne   r1, [r0,#CLEAR]
```

Dont les registres sont :

- r0 : adresse de base du GPIO
- r1 : bitset du segment
- r2 : état voulu du GPIO

Equivalence en langage C :

```
if(state==on)
        gpio.set = seg;
else
        gpio.clear = seg;
```

2.1.2 Configuration des GPIO en « output »

Voici le code qui permet de mettre le GPIO en mode « output » :

```
ldr r4, [r0,#OE]
bic r4, r1
str r4, [r0,#OE]
```

Dont les registres sont :

- r0 : adresse de base du GPIO
- r1 : bitset du segment
- r4 : stockage momentané

Equivalence en langage C :

```
gpio.oe &= ~seg;
```

2.1.3 Configuration du PADMUX

Voici le code qui permet de configurer le PADMUX :

```
ldr r4, =0x4f
ldr r1, =PADMUX
str r4, [r1,r3]
```

Dont les registres sont :

- r1 : adresse de base du PAD multiplexer
- r3 : offset de segment dans le PADMUX
- r4 : stockage momentané

Equivalence en langage C :

```
padmux[ofs] = 0x4f;
```

2.2 Sleep

Voici le code qui permet de faire un « sleep » :

```
sleep:  nop
1:      subs    r5, #1
        bpl     1b
        bx      lr
```

Dont le registre r5 est le nombre à décrémenter.

2.3 Allumage ou extinction d'un segment

Voici le code qui permet d'allumer ou d'éteindre un segment :

```
turn_seg:       nop
                cmp     r1, #0
                ldr     r1, =GPIO0
                streq   r0, [r1,#CLEAR]
                strne   r0, [r1,#SET]
                bx      lr
```

Dont les registres sont :

- r0 : bitset du segment
- r1 : état voulu du segment et ensuite l'adresse de base du GPIO

2.4 Allumage d'un segment

Voici le code qui permet d'allumer un segment :

```
turn_on:        nop
                mov     r10, r14 // save pointer for return
                mov     r1, #1
                bl      turn_seg
                bx      r10
```

Dont les registres sont :

- r1 : état voulu du segment
- r10 : adresse de retour (vers la méthode principale)

2.5 Extinction d'un segment

Voici le code qui permet d'éteindre un segment :

```
turn_off:   nop
            mov     r10, r14 // save pointer for return
            mov     r1, #0
            bl      turn_seg
            bx      r10
```

Dont les registres sont :

- r1 : état voulu du segment
- r10 : adresse de retour (vers la méthode principale)

2.6 Principale

2.6.1 Initialisation des GPIO

Voici le code qui permet d'initialiser les GPIO :

```
mov     r0, #0
bl      am335x_gpio_init
mov     r0, #2
bl      am335x_gpio_init
```

2.6.2 Configuration de tous les GPIO

Voici le code de la méthode principale qui configure tous les GPIO :

```
        ldr     r8, =gpio_config
1:      ldmia   r8!, {r0-r3}
        cmp     r0, #0
        beq     2f
        bl      config_gpio
        b       1b
2:      nop
```

2.6.3 Affichage du serpent

Voici ci-dessous le code de la méthode principale qui permet d'afficher le serpent. Il s'agit d'une boucle infinie.

```
1:      ldr     r4,     =snake
2:      ldr     r0, [r4], #4
        cmp     r0, #0
        beq     1b
        bl      turn_on
        ldr     r5, =TIMEOUT
        bl      sleep
        bl      turn_off
        b       2b
```

Equivalence en langage Java :

```
long[] snake = {SEGA, ..., 0};
while(true) {
        int i = 0;
        while(true) {
                i++;
                if(snake[i] == 0) break;
                turnOn(snake[i]);
                sleep();
                turnOff(snake[i]);
        }
}
```