SYSTEMES EMBARQUES 1

TP03– INTRODUCTION AUX ENTREES/SORTIES (I/0)

Samuel Mertenat
Classe T2f

Fribourg, les 28 octobre et 11 novembre 2014
13h – 16h35

TABLE DES MATIERES

SYSTEMES EMBARQUES 1

TP03– INTRODUCTION AUX ENTREES/SORTIES (I/0)

BUTS ET OBJECTIFS DU TRAVAIL PRATIQUE

Ce troisième travail pratique est l'occasion de découvrir le fonctionnement des entrées / sorties, qui se composent, dans notre cas, d'un affichage 7-segments, de LEDs et de boutons poussoir.

Les objectifs de ce travail sont :

- Coder un petit programme en assembleur (planifier, concevoir et tester)
- Comprendre le fonctionnement des entrées / sorties
- Lire la « datasheet » d'un composant

Spécifications du travail à effectuer :

- La gauge devra être capable de compter entre 0 et 255
- Le bouton poussoir FPGA de gauche sert à décrémenter la gauge
- bouton poussoir FPGA du centre sert à remettre à zéro (reset) la gauge
- Le bouton poussoir FPGA de droite sert à incrémenter la gauge
- L'affichage de l'état de la gauge se fera sur l'affichage 7-segments de la FPGA en hexadécimal
- En cas de dépassement de capacité de la gauge, celle-ci conservera sa valeur maximale, respectivement minimale
- Si on maintient la pression sur un bouton poussoir, l'incrémentation/la décrémentation se fera automatiquement à intervalle de 0.5 seconde environ
- L'état des boutons poussoir sera représenté sur les LED de la FPGA

ANALYSE

INITIALISER LE MATERIEL

Pour initialiser le matériel (boutons, 7-segments, LEDs), nous devons définir les « PIN » en mode « read » ou « write ». Pour cela, nous utilisons les adresses mémoires de contrôle (CTRL) des différents « PIN ».

Nous utilisons le « GPIO_0 » pour l'affichage 7-segments et le « GPIO_1+ pour les LEDs et les boutons poussoirs.

Les boutons doivent être configurés en mode « entrée » (PIN = 0) et les LEDs et l'affichage 7-segments en mode « sortie » (PIN = 1).

IP	Adresse	Registre	Accès	Description
GPIO_0	0xd6000008	RW	r/w	Lecture/écriture des valeurs de chaque pin
	0xd600000a	CTRL	r/w	Configuration: entrée (0) / sortie (1) des pins
	0xd600000c	ID	r	Identifiant unique de l'IP → 2
GPIO_1	0xd6000010	RW	r/w	Lecture/écriture des valeurs de chaque pin
	0xd6000012	CTRL	r/w	Configuration: entrée (0) / sortie (1) des pins
	0xd6000014	ID	r	Identifiant unique de l'IP → 3

Figure 1: Adresses nécessaires afin d'initialiser le matériel

Initialisation de l'affichage 7-segments :

```
ldrh r0, =FPGA_BASE                    // load the fpga base's address
ldrh r1, =0x3FF                        // load 0b1111111111
strh r1, [r0, #FPGA_SEG7_CTRL]         // set each segment as output
```

Initialisation des boutons et des LEDs :

```
ldrh r0, =FPGA_BASE                    // load the fpga base's address
ldrh r1, =0x0FF                        // 0b00011111111
strh r1, [r0, #FPGA_LED_SW_CTRL]       // set the LEDs as output [0-7] and buttons as input [8-10]
```

LIRE ET FILTRER L'ETAT DES BOUTONS POUSSOIR

Pour lire la valeur d'un bouton poussoir, nous devons prendre en compte la table ci-dessous :

PIN	GPIO0 -Périphériques (7-segments)	GPIO1-Périphériques (FPGA-LED + Switch)
0	DIG1	FD0
1	DIG2	FD1
2	DPx	FD2
3	A	FD3
4	B	FD4
5	C	FD5
6	D	FD6
7	E	FD7
8	F	F_S0
9	G	F_S1
10	nc	F_S2
11→ 15	nc	nc

Figure 2: Liste des PINs nécessaires pour l'utilisation du matériel

Dans la table ci-dessus, le nom des boutons débute par « F_S x », les LEDs par « FDx » et les segments sont représentés par une lettre majuscule (A à G).

Pour lire la valeur d'un bouton, nous devons lire la valeur stockée à l'adresse « RW » du « GPIO_1 », en extirper les 8 derniers bits et en effectuant un masque afin de déterminer quel bouton est pressé.

```
Ldr     r1, =FPGA_BASE + FPGA_LED_SW_RW + 0x1    // load the fpga buttons' address
ldrb    r0, [r1]                                 // load the current state of the buttons / leds
mov     r2, #7                                   // create a mask to filter the buttons from the rest
mvn     r0, r0                                   // inverse the loaded byte
and     r0, r2, r0                               // get the pressed button [1-2-4] (0: not pressed)
```

Nous utilisons ensuite un « cmp » afin de définir si c'est le bouton de gauche, du milieu ou de droite qui est pressé (0b001, 0b010 ou 0b100).

```
cmp r0, #0b100                  // test if the third button is pressed
```

AFFICHER UN ETAT SUR LES LEDS

Pour allumer (bit à 1) ou éteindre (bit à 0) une LED, il faut utiliser son « PIN » correspondant (cf. Figure 2 ; nom commençant par « FDx »).

Par exemple, pour allumer la LED 1, ces trois instructions nous suffisent :

```
ldrh    r0, =FPGA_BASE                  // load the fpga base's address
ldrh    r1, =0b1                        // load 0b1: first LED
strh    r1, [r0, #FPGA_LED_SW_RW]       // switch off all the leds
```

AFFICHER UNE VALEUR SUR LES 7-SEGMENTS

Pour allumer (bit à 1) ou éteindre (bit à 0) un segment, il faut nous baser sur la figure 2, où l'on peut trouver le « PIN » à utiliser. Ensuite, pour définir le « digit » à utiliser, nous définissons le « PIN » correspondant à « DIG1 » ou « DIG2 » (0 : éteint ; 1 : allumé).

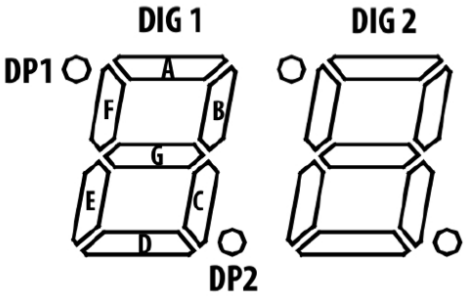

Figure 3: Affichage 7-segments

Par exemple, pour afficher la valeur « 1 » (B-C) sur le « digit » 1 (DIG1 = 1) :

```
ldrh r0, =FPGA_BASE                     // load the fpga base's address
ldrh r1, =0b11001                       // segment B-C on digit 1
strh r1, [r0, #FPGA_SEG7_RW]            // switch on the segments
```

CONCEPTION

Notre programme se décompose de la façon suivante:

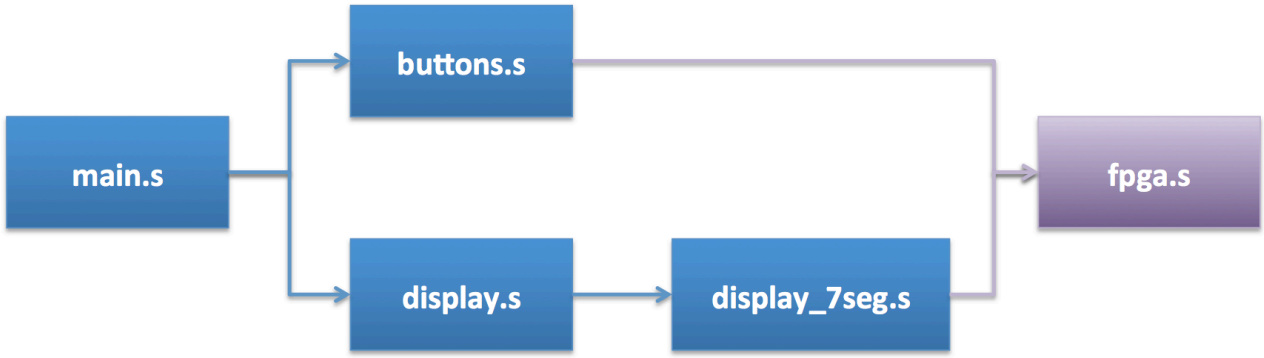

Figure 4: Structure du programme

Les classes sont :

- main.s :
  - Module principale implémentant l'algorithme de la gauge
- buttons.s :
  - Module pour la saisie de l'état des boutons poussoirs
- display.s
  - module pour l'affichage en hexa de la valeur de la gauge
- display_7seg.s
  - module pour l'affichage d'un digit sur le 7-segment de la FPGA
- fpga.s
  - Module de définition des constantes liées aux registres de la FPGA

EXPLICATION DES DIFFERENTES CLASSES

MAIN.S

- Initialisation de l'affichage 7-segments → display_fpga_ini (display_fpga.s)
- Initialisation des boutons et des LEDs → buttons_init (buttons.s)
- Initialisation des registres :
  - r12 : compteur pour la gauge
  - r11 : compteur pour la procédure de test des LEDs
- Lance la phase de test → display_fpga_test (display_fpga.s) → buttons_test (buttons.s) → main_loop (main.s)
- Boucle principale (main_loop) :
  - Récupère l'état des boutons (1-2-3 ou 0 (non pressés))
  - Effectue un masque avec la valeur correspondante à chaque bouton
    - Si c'est le bouton de gauche → decrement
    - Si c'est le bouton du milieu → reset

        - Si c'est le bouton de droite → increment
    - Affiche la valeur de la gauche (r12)
    - Redémarre la boucle (main_loop)
- decrement :
    - Charge 100000 dans r5 (delay)
    - Soustrait 1 à la gauge (r12) si celle-ci n'est pas égal à 0
    - Démarre le « delay » → main_loop
- reset :
    - Charge 100000 dans r5 (delay)
    - Met la gauge à 0 (r12)
    - Démarre le « delay » → main_loop
- increment :
    - Charge 100000 dans r5 (delay)
    - Additionne 1 à la gauge (r12) si celle-ci n'est pas égal à 255
    - Démarre le « delay » → main_loop
- delay, delayDisplayTestMode, displayButtonsTestMode :
    - Soustrait 1 à r5
    - Affichage la valeur de la gauge (r12)
    - Redémarre la boucle si le « delay » n'est pas égal à 0, sinon, retourne à l'endroit de l'appel du « delay »

BUTTONS.S

- buttons_init :
    - Définit les LEDs en tant que sortie et les boutons en tant qu'entrée
    - Extinction de toutes les LEDs
- buttons_test :
    - Allume une LED
    - Effectue un décalage vers la gauche de la valeur de la LED à allumer
    - Charge 25000 dans r5
    - Démarre le « delay »
    - Si toute les LEDs ont été allumées une à une → main_loop (main.s), sinon → buttons_test (buttons.s)
- buttons_get_state :
    - Récupère la valeur à l'adresse du FPGA et en charge les 8 derniers bits
    - Effectue un masque pour récupérer les 3 bits correspondant aux boutons
    - Stocke dans r0 cette valeur (boutons : 1,2 ou 3 ; non pressés : 0)
    - Allume la LED correspondant au bouton (si reset, toutes les LEDs)

DSPLAY.S

- display_val :
    - Charge la valeur de la gauge dans r2
    - Choisit le « digit » de gauche (r1)

- Appel display_fpga_7seg
- Choisit le « digit » de droite (r1)
- Appel display_fpga_7seg

DISPLAY_7SEG.S

- display_fpga_init :
  - Définit les segments en tant que sortie
  - Extinction de tous les segments
- display_fpga_7seg :
  - Charge l'adresse de la « table de correspondance » dans r3
  - Récupère les segments à allumer en faisant un shift de 2 de la valeur à afficher (stocke dans r3)
  - Allumer les segments correspondant à r3
- dsplay_fpga_test :
  - Tant que r12 n'est pas égal à 255, on incrémente r12 de 1. Sinon, on appel buttons_test (buttons.s)
  - Charge 50000 dans r5 (delay)
  - Démarre le « delay » (se charge d'afficher la valeur sur le 7-segments)
  - Redémarre la boucle display_fpga_test

Pour faire le lien entre la valeur de la gauge à afficher et les segments à allumer, on utilise une table de correspondance. On effectue un shift de 2 de la valeur à afficher dans la table de correspondance, ce qui nous permet de récupérer les segments à allumer.

Table de correspondance :

```
seg_7:
        .long SEG_A + SEG_B + SEG_C + SEG_D + SEG_E + SEG_F                 // 0
        .long SEG_B + SEG_C                                                 // 1
        .long SEG_A + SEG_B + SEG_D + SEG_E + SEG_G                         // 2
        .long SEG_A + SEG_B + SEG_C + SEG_D + SEG_G                         // 3
        .long SEG_B + SEG_C + SEG_F + SEG_G                                 // 4
        .long SEG_A + SEG_C + SEG_D + SEG_F + SEG_G                         // 5
        .long SEG_A + SEG_C + SEG_D + SEG_E + SEG_F + SEG_G                 // 6
        .long SEG_A + SEG_B + SEG_C                                         // 7
        .long SEG_A + SEG_B + SEG_C + SEG_D + SEG_E + SEG_F + SEG_G         // 8
        .long SEG_A + SEG_B + SEG_C + SEG_D + SEG_F + SEG_G                 // 9
        .long SEG_A + SEG_B + SEG_C + SEG_E + SEG_F + SEG_G                 // A
        .long SEG_C + SEG_D + SEG_E + SEG_F + SEG_G                         // b
        .long SEG_A + SEG_D + SEG_E + SEG_F                                 // C
        .long SEG_B + SEG_C + SEG_D + SEG_E + SEG_G                         // D
        .long SEG_A + SEG_D + SEG_E + SEG_F + SEG_G                         // E
        .long SEG_A + SEG_E + SEG_F + SEG_G                                 // F
```

Figure 5: Table de correspondance entre valeur à afficher et segments à allumer

Principe :

```
        ldr r3, =seg_7                  // load the segments' table
        ldr r3, [r3, r0, lsl #2]        // load the corresponding segments' value
        orr r3, r3, r1                  // define the digit to switch on (not the two)
        ldrh r4, =FPGA_BASE             // load the fpga base's address
        strh r3, [r4, #FPGA_SEG7_RW]    // switch on the segments
```

Test et validation

Dans ce travail pratique, il y a plusieurs points à vérifier afin de valider le bon fonctionnement du programme. La phase de tests comprend l'affichage des valeurs de 0 à 255 (test de l'affichage 7-segments) et l'allumage des 8 LEDs, l'une après l'autre. Le programme principal ajoute ensuite la validation du bon fonctionnement des boutons, en allumant la LED correspondante à celui-ci et en démarrant une action perceptible par l'utilisateur (incrémenter, mettre à zéro ou décrémenter la gauge, dont la valeur est affichée sur l'affichage 7-segemnts).

TESTS DANS LA PHASE INITIALE

Cette phase comprend tout d'abord une boucle, affichant sur l'affichage 7-segments les valeurs de 0 à FF. Ceci nous permet de confirmer le bon fonctionnement de la table de correspondance et des méthodes « display_val » (display.s), « display_fpga_7seg » et « display_fpga_init » (display_7seg.s).

```
display_fpga_test:
        cmp     r12, #255               // test if the counter is equal to 255
        beq buttons_test                // branch to the buttons test
        addne r12, #1                   // if it's not the case, +1
        ldr     r5,     =50000          // load +/- 0.25 [s]
        bl delayDisplayTestMode         // launch the timer
        bne display_fpga_test
```

Une seconde boucle, « buttons_init », permet d'attester le bon fonctionnement des LEDs et de l'initialisation des boutons. Cette boucle allume, une après l'autre, les 8 LEDs du FPGA. Attestation de la méthode « buttons_init » (buttons.s).

```
buttons_test:
        ldrh    r0, =FPGA_BASE                  // load the fpga base's address
        strh    r11, [r0, #FPGA_LED_SW_RW]      // switch off all the leds
        cmp             r11, #0b100000000       // while r11 isn't equal to 128
        moveq   r12, #0                         // if r12 is equal to 128: set to 0
        beq             main_loop               // branch to the main loop (counter initialized)
        movne   r11, r11, lsl #1                // multiply r11 by 2: the next led
        ldr             r5, =250000             // load +/- 0.25 [s]
        bl              delayButtonsTestMode    // launch the timer
        bne             buttons_test
```

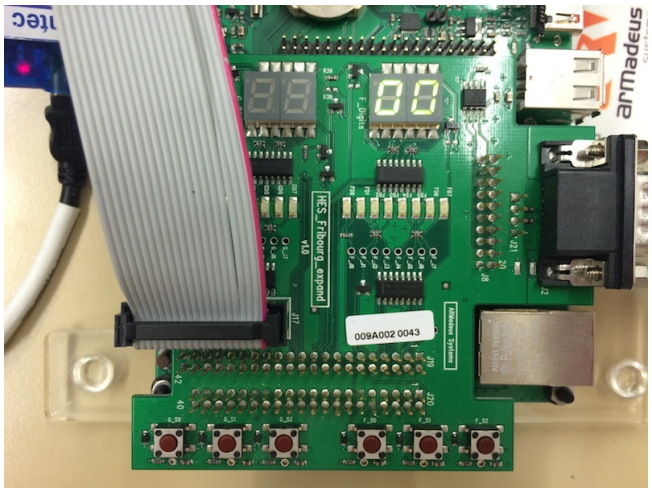

Figure 6: Affichage de la valeur 0 sur le 7-segments

Après ces tests, nous sommes assurés que le matériel a été correctement configuré (7-segments et LED en tant que sortie et boutons en tant qu'entrée). A la suite de cette phase, la gauge est mise à zéro.

Pour réaliser une phase de tests complète, il aurait fallu encore implémenter une séquence de tests pour les boutons, ainsi que de l'affichage de messages pertinents dans la console. Mais, étant seul et par

manque de temps, je n'ai guère eu le temps de le faire.

On aurait pu imaginer devoir quitter le mode de tests par une pression prolongée sur le bouton reset pour atteindre la boucle principale ou afficher sur la console un message notifiant la pression sur une touche, l'état de la gauge, etc.

TESTS DANS LE PROGRAMME PRINCIPAL

Le programme principal apporte l'attestation du bon fonctionnement des boutons poussoir. En effet, suite à la pression sur l'un des boutons, la LED s'y référant s'allume (tous les LEDs si c'est le bouton du milieu) et une action est réalisée (incrémenter, décrémenter ou mettre la gauge à zéro), dont le résultat est visible directement sur l'affichage 7-segments.

VALIDATION DU CAHIER DES CHARGES

- ✓ La gauge devra être capable de compter entre 0 et 255
- ✓ Le bouton poussoir FPGA de gauche sert à décrémenter la gauge
- ✓ bouton poussoir FPGA du centre sert à remettre à zéro (reset) la gauge
- ✓ Le bouton poussoir FPGA de droite sert à incrémenter la gauge

```
bl      buttons_get_state       // return 0 or 1-2-4 (buttons decrement, reset, increment)
cmp r0, #0b001                  // test if the first button is pressed
beq decrement
cmp r0, #0b010                  // test if the second button is pressed
beq reset
cmp r0, #0b100                  // test if the third button is pressed
beq increment
```

- ✓ L'affichage de l'état de la gauge se fera sur l'affichage 7-segments de la FPGA en hexadécimal
- ✓ En cas de dépassement de capacité de la gauge, celle-ci conservera sa valeur maximale, respectivement minimale

```
decrement:
        ldr     r5,     =100000             // load +/- 0.5 [s]
        cmp r12, #0                         // test if the counter is equal to 0
        subne r12, #1                       // if it's not the case, -1
        bl delay

increment:
        ldr     r5,     =100000             // load +/- 0.5 [s]
        cmp     r12, #255                   // test if the counter is equal to 255
        addne r12, #1                       // if it's not the case, +1
        bl delay
```

- ✓ Si on maintient la pression sur un bouton poussoir, l'incrémentation/la décrémentation se fera automatiquement à intervalle de 0.5 seconde environ

```
delay:
        sub     r5, #1                      // decrement the timer by 1
        bl      display_val
        cmp r5, #0                          // test if the timer is equal to 0
        bne     delay                       // if it's not the case, re-call the loop again
        bl      main_loop
```

- ✓ L'état des boutons poussoir sera représenté sur les LED de la FPGA

```
buttons_get_state:
        nop
        ldr r1, =FPGA_BASE + FPGA_LED_SW_RW + 0x1        // load the fpga buttons' address
        ldrb    r0, [r1]                        // load the current state of the buttons / leds
        mov     r2, #7                          // create a mask to filter the buttons from the rest
        mvn     r0, r0                          // inverse the loaded byte
        and     r0, r2, r0                      // get the pressed button [1-2-4] (0: not pressed)
        cmp     r0, #0b010
        moveq   r0, #0b11111111          // switch on all the leds when the reset button is pressed
        ldrh    r1, =FPGA_BASE                  // load the fpga base's address
        strh    r0, [r1, #FPGA_LED_SW_RW]       // switch on the corresponded led
        moveq   r0,     #0b010                  // restore the originale value (0b010)
        bx              lr
```

Bien que des phases de tests ont été réalisées, le meilleur moyen de vérifier le bon fonctionnement du programme est de l'exécuter sur la cible. Dès lors, on pourra être quasiment certain du bon fonctionnement de celui-ci.

CONCLUSION

Ce troisième travail pratique aura été l'occasion de se familiariser avec les entrées / sorties disponibles sur la cible (7-segments, LEDs et boutons poussoir) en réalisant une gauge et en implémentant des méthodes pouvant modifier sa valeur (incrémenter, décrémenter, mettre à zéro). D'autre part, ce travail m'aura permis d'améliorer mes connaissances en assembleur et d'apprendre à lire une « datasheet » afin d'adresser les composants de la cible de la bonne manière.

ANNEXE

Le code source du programme se trouve sur Git, à l'adresse : https://forge.tic.eia-fr.ch/git/samuel.mertenat/se12-tp.git, dans le dossier « tp3 ».