TP1 Capteurs / Actionneurs

SI4 – Polytech Nice Sophia

Vous devrez répondre sur l'énoncé aux questions R1, R2, …
Vous devrez écrire, tester puis montrer à l'enseignant les codes C1, C2, …

Ouvrez le logiciel Nios-II SBT for Eclipse, créer une nouvelle « Nios-II application and BSP from template ».
Choisissez :

- SOPC information file name : fichier a extension .sopcinfo fourni sur moodle
- name : un nom de projet sans espace (le chemin vers le projet ne doit pas contenir d'espace non plus)
- template : Hello_world

Cliquez sur Finish, cross-compiler le projet et tester le code (cf. annexe 1).

1) Gestion des périphériques

Réaliser un premier programme gérant les périphériques suivants :

- 10 LED rouges,
- les afficheurs 7 segments de 0 à 5,
- 10 les switches.

L'objectif de ce premier code étant de retrouver la table d'affichage des 10 chiffres décimaux sur les afficheurs 7 segments, vous écrirez la configuration lue sur les switches sur les LED rouges et sur le premier afficheur 7-segments (à droite), les autres restant à 0.

Pour cela utiliser les deux fonctions de lecture et écriture dans les périphériques mappés en mémoire (inclure altera_avalon_pio_regs.h) :

- `IOWR_ALTERA_AVALON_PIO_DATA (address, data)`
- `data = IORD_ALTERA_AVALON_PIO_DATA (address)`

Le type de data dépend du nombre de bits du périphérique utilisé.
Les adresses des périphériques sont définies comme des macros dans le fichier system.h (dans le projet BSP).

R1. A partir de ce fichier system.h, lister l'ensemble des périphériques de ce micro-contrôleur (hors mémoires).

Périphérique	Adresse

C1. Tester les valeurs sur les switches.
R2. Remplissez la table d'allumage des afficheurs (annexe 2) suivante.
L'attente entre 2 lectures/écritures se fera par la fonction usleep(microsecondes) de `<unistd.h>`.

Chiffre décimal	Configuration binaire	Configuration en héxadécimal
0		
1		
2		
3		
4		
5		
6		
7		
8		
9		

Ce tableau doit vous servir à définir une table de traduction ou LUT (Look Up Table) :
int LUT[10] = {config0, config1, config2, … config9} ;
où chaque configuration sera écrite en hexadécimale, par exemple : 0xff.
C2. Tester en envoyant la valeur d'un compteur de 0 à 9999 sur les 4 afficheurs 7 segments.

2) Programmation des interruptions

Maintenant que les périphériques sont bien configurés, nous allons programmer les interruptions du processeur pour lire la configuration des boutons poussoirs.
Comme pour toute lecture de capteur ou de périphérique d'entrée, deux méthodes de lectures sont possibles :

- par polling,
- par interruption.

R3. Rappeler les avantages et inconvénients de chaque méthode.

Méthode de lecture de périphérique	Avantage ou inconvénient
Polling	
Interruptions	

Pour la programmation des interruptions, nous allons suivre la démarche vue en cours en s'appuyant sur le code fourni en annexe 3 et en remplaçant les constantes ADDRESS_BASE et NUMERO_IRQ par les noms définis dans system.h (dans le projet BSP).

R4. A quoi sert les préfixes suivants, souvent utilisés en embarqué : *volatile* sur la variable d'état des boutons poussoirs, *static* sur le routine d'interruption ?

Préfixe	Rôle
Volatile	
Static	

C3. Faire un code qui indique le numéro du bouton pressé sur les afficheurs 7-segments. Les LED rouges doivent s'allumer pour indiquer les switches en position haute. Cette configuration des switches sera utilisée pour programmer le temps d'attente dans la boucle principale (plus l'utilisateur lève de switches, plus le temps max d'attente est long.

3) Test de réflexe

C4. Maintenant que les périphériques sont programmables, réaliser un code répondant au cahier des charges suivants :

- pour lancer le test, l'utilisateur doit appuyer sur le bouton 1,
- toutes les LED rouges s'allument alors au bout d'un temps aléatoire,
- pendant ce temps les LED rouges clignotent régulièrement (Leds pairs puis impairs…),
- l'utilisateur doit appuyer aussi vite que possible sur le bouton 4 (à gauche),
- le système mesure le temps de réaction de l'utilisateur entre l'allumage et l'appuie sur le bouton, et affiche cette valeur : secondes et millisecondes sur les 7 segments. Utiliser pour cela le périphérique timer (annexe 4).
- les switches servent à fournir le temps d'attente maximum avant l'allumage,
- un nouveau test de réflexe est lancé en appuyant sur le bouton 1,
- en appuyant sur le bouton 2, le système affiche le temps moyen de réaction depuis le début des tests.

Le code la routine d'interruption de l'annexe 3 ne doit pas être modifié pour maintenir une latence d'interruption aussi courte que possible.

Temps (s,ms)	Temps Essai 1	Temps Essai 2	Temps Essai 3	Temps Essai 4	Temps Essai5
Utilisateur 1					
Utilisateur 2					

Annexe 1 – Programmation de la carte DE1-SoC / DE10-SoC

Pour Configurer le circuit FPGA sur la cible (une seule fois) :

- brancher la carte au port USB Blaster
- allumer la carte (bouton rouge)
- lancer Nios II > Quartus II Programmer,
  - autodetect : 2 circuits apparaissent
  - ajouter le fichier suivant à la place du 2e circuit
  - DE1_Quartus16_1.sof ou DE10_Quartus16è1.sof
- Cliquer sur start pour configurer le FPGA

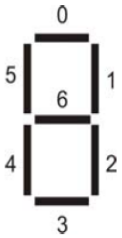
Pour télécharger l'exécutable (fichier .elf) dans la mémoire du micro-contrôleur :

- compiler le projet
- lancer Run > Run Configuration …
- New NiosII Hardware
- refresh connections
- Run

Annexe 2 – Afficheurs 7 segments

Routage des afficheurs 7-segments sur le PCB (exemple sur les Hex 0 et Hex 1)

Numérotation des bits de configuration des afficheurs :



Annexe 3 - Programmation des interruptions.

Variable globale d'état des bouttons :

```
volatile int edge_capture;
```

Routine d'interruption :

```
static void handle_button_interrupts(void* context, alt_u32 id)
{
/* Cast context to edge_capture's type. It is important that this
be declared volatile to avoid unwanted compiler optimization. */
volatile int* edge_capture_ptr = (volatile int*) context;
/* Read the edge capture register on the button PIO. Store value. */
*edge_capture_ptr =
            IORD_ALTERA_AVALON_PIO_EDGE_CAP (ADDRESS_BASE);
```

```
/* Write to the edge capture register to reset it. */
IOWR_ALTERA_AVALON_PIO_EDGE_CAP (ADDRESS_BASE, 0);
/* Read the PIO to delay ISR exit. This is done to prevent a
spurious interrupt in systems with high processor -> pio
latency and fast interrupts. */
IORD_ALTERA_AVALON_PIO_EDGE_CAP (ADDRESS_BASE);
}
```

Enregistrement de la routine d'interruption

```
static void init_button_pio()
{
/* Recast the edge_capture pointer to match the alt_irq_register() function
prototype. */
void* edge_capture_ptr = (void*) &edge_capture;
/* Enable all 4 button interrupts. */
IOWR_ALTERA_AVALON_PIO_IRQ_MASK (ADDRESS_BASE, 0xf);
/* Reset the edge capture register. */
IOWR_ALTERA_AVALON_PIO_EDGE_CAP (ADDRESS_BASE, 0x0);
/* Register the ISR. */
alt_irq_register( NUMERO_IRQ,edge_capture_ptr,handle_button_interrupts );
}
```

Annexe 4 – Timer

Première méthode: **timestamp**

Les fonctions d'accès au timestamp sont définies dans :

```
#include "sys/alt_timestamp.h"
```

alt_timestamp_start(); // intialize the timer
alt_timestamp(); // read the timer
alt_timestamp_freq() // read the frequence of the timer

Les fonctions d'accès au timer sont définies dans :

```
#include "altera_avalon_timer_regs.h"
```

Les registres du périphérique sont sur 16 bits :

- La période du timer doit être initialisée à la valeur voulue :

```
IOWR_ALTERA_AVALON_TIMER_PERIODL(address, periodLow) // 16 bits de poids faible
IOWR_ALTERA_AVALON_TIMER_PERIODH(address, periodHigh)// 16 bits de poids fort
```

- Le timer doit tout d'abord être initialisé en mode START :

```
IOWR_ALTERA_AVALON_TIMER_CONTROL(address,ALTERA_AVALON_TIMER_CONTROL_START_MSK)
```

- Pour lire la valeur du timer, il faur tout d'abord y écrire :

```
IOWR_ALTERA_AVALON_TIMER_SNAPL(address, 0x01)
t1 = IORD_ALTERA_AVALON_TIMER_SNAPL (address)
IOWR_ALTERA_AVALON_TIMER_SNAPH(address, 0x01)
t2 = IORD_ALTERA_AVALON_TIMER_SNAPL (address)
t = t2<<16 & t1
```