TP 3
Interface Logiciel Matériel

Objectif

Développer pour l'informatique embarquée, c'est souvent avoir à s'interfacer avec le matériel. Dans ce TP nous pratiquerons des manipulations de registres, ce nous servira à gérer les interruptions, elle même nécessaire à la manipulation de timer. Ceci nous permettra d'appeler périodiquement la fonction d'une librairie.

>> Vous produirez un rapport de TP contenant vos réalisations et observations. Les éléments indispensables à ce rapport sont signalés en bleu.

Modifier un registre

Écrire les deux fonctions suivante en s'inspirant de la slide32 du cours 4. Ces fonctions et la fonction de test unitaire associée doivent se trouver dans un fichier à part.

Écrire une fonction qui passe à 1 l'un des bit d'un registre. Elle prend en entrée un pointeur `volatile` sur `uint32_t` ainsi que le numéro d'index du bit visé. Elle ne retourne rien.

Écrire une seconde fonction qui passe un bit à 0 et utilise les mêmes arguments que la fonction précédente.

Pour tester ces fonctions, écrire une fonction qui les appelle avec différents arguments, classiques et aux limites. Après chacun des appels vérifier le résultat obtenu. Cette fonction de test ne retourne 0 que si tous les tests ont l'effet attendu sur l'espace mémoire passé en paramètre.

>> Que contiennent ces fichiers *.h et *.c ?

Gérer les interruptions

En s'appuyant sur les fonctions précédentes et sur `bcc_isr_register()` (à la place de `catch_interrupt()` présenté dans le cours), écrivez le code des fonctions dont voici les prototypes.

Vous aurez besoin d'inclure : `#include <bcc/bcc.h>`

```
/**
 * Enable an interrupt and set the handler.
 * Used the value of the macro INTERRUPT_MASK_REGISTER.
 * @param irq number of the interrupt to activate
 * @param handler function to call when the interrupt \a irq is triggered
 */
void activate_interrupt(uint32_t irq, void* handler) ;
```

```
/**
 * Disable an interrupt.
 * Used the value of the macro INTERRUPT_MASK_REGISTER
 * @param irq number of interrupt to disable
 */
void disable_interrupt(uint32_t irq) ;

/**
 * Trigger the interrupt in parameter.
 * Used the value of the macro INTERRUPT_FORCE_REGISTER
 * @param irq number of interruption to trigger
 */
void force_interrupt(uint32_t irq) ;
```

Pour tester vos fonctions, définissez un handler incrémentant un compteur global nommé `g_nb_interruptions`.

```
uint32_t compteur_test = 0;
activate_interrupt(9, votre_handler);
while ((g_nb_interruptions == 0) && (compteur_test < 100)) {
       compteur_test++;
       force_interrupt(9);
}
disable_interrupt(9) ;
force_interrupt(9) ;
```

>> Après l'exécution du code ci-dessus, quelles sont les valeurs des `compteur_test` et `g_nb_interruptions` ?

>> La commande gdb "`mon info reg`" permet d'afficher les registres. Commenter l'état des registres d'interruption, avant et après l'appel au code précédent.

Gérer un timer

En s'appuyant sur les fonctions précédentes, ainsi que sur `bcc_isr_register()` (à la place de `catch_interrupt()` présenté dans le cours), et sachant que les registres *counter*, *reload* et *control* d'un timer se suivent en mémoire (cf. slide 22 du cours), écrivez la fonction suivante :

```
/**
 * Configure les registres d'un timer pour qu'il déclenche une interruption
 * après reload_value ticks et se recharge automatiquement avec la même période.
 * @param timer_counter_register pointeur sur le registre de compteur du timer
 * @param period nombre tick entre deux déclenchement du timer
 */
void start_timer(uint32_t* timer_counter_register, uint32_t period) ;
```

Le timer 2 déclenchant l'IRQ 9, vérifier qu'appeler cette fonction permette à votre programme de sortir de la boucle suivante. Le paramètre `period` doit avoir pour valeur 5000.

```
while (g_nb_interruptions < 100) {}
```

Ajouter au handler un appel à `get_elapsed_time()` alimentant un tableau global et mesurer ainsi la période du déclenchement de interruption.

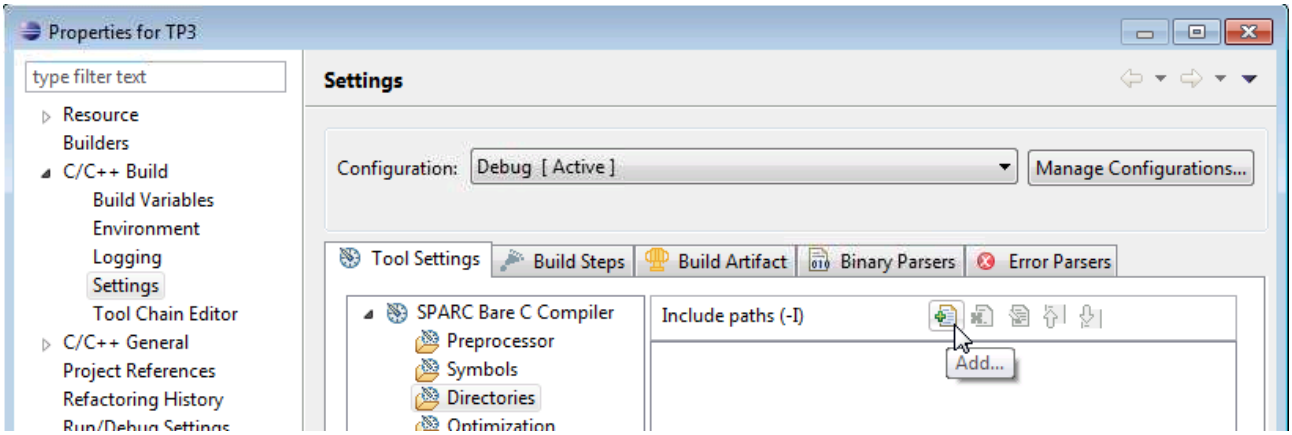
>> Quel est le temps moyen entre deux appels du handler ? Le temps mesuré le plus court, le plus long et l'écart-type ?

S'interfacer avec une bibliothèque

Récupérer la bibliothèque `libwindows-producer.a` jointe au TP, ainsi que le header `windows-producer.h` définissant l'interface de la bibliothèque.

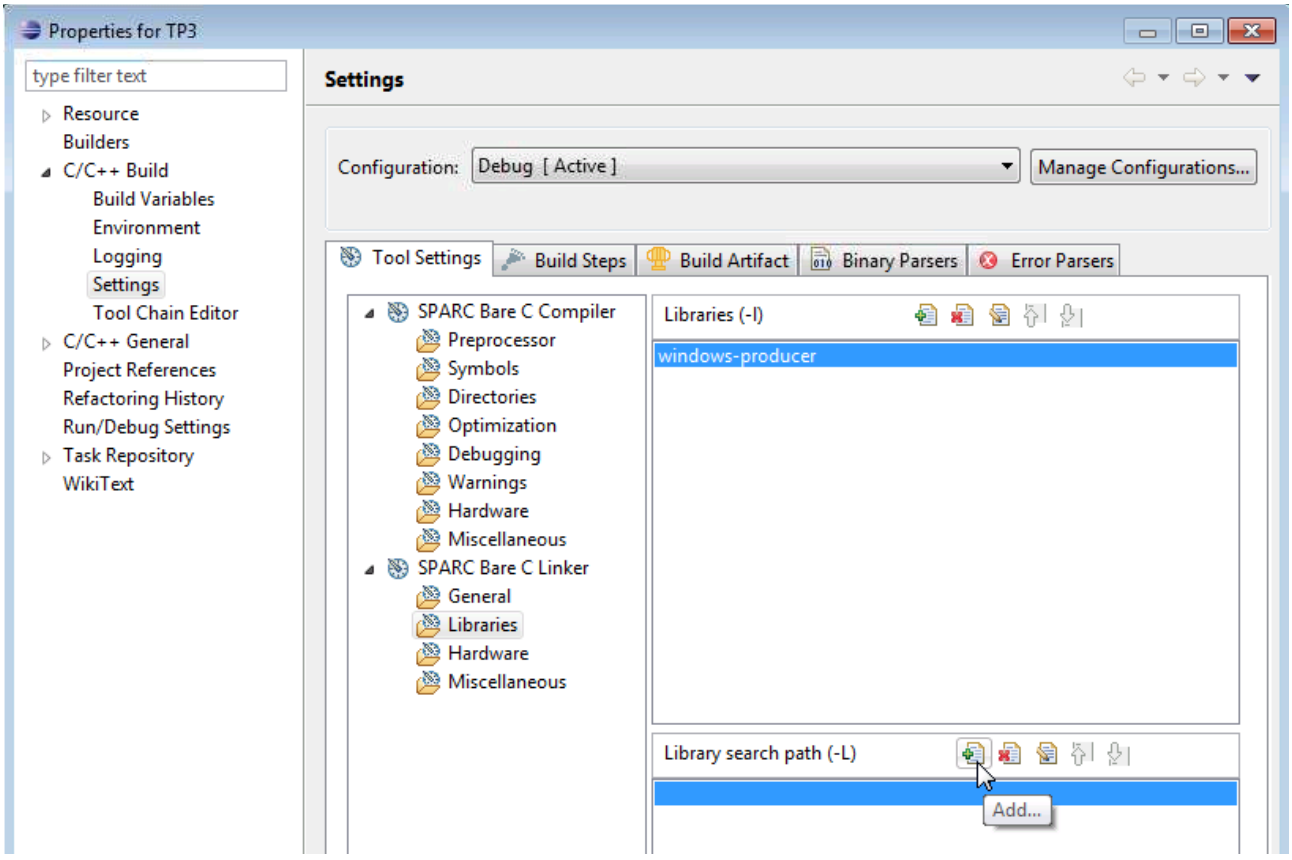
Ajouter au projet le nouvel header

Dans les paramètres du projet (`Settings > tool settings > compiler > Include paths`), indiquer l'endroit où trouver le header de la bibliothèque.



Ajouter la bibliothèque au linkage

Dans les options de linkage définir la nouvelle bibliothèque et son chemin.



Utiliser la bibliothèque pour recevoir des données

Cette bibliothèque étant compilée avec l'option soft-float, vous devrez configurer votre projet pour qu'il l'utilise aussi. Afin de laisser le développeur maître de la mémoire, cette bibliothèque ne fait pas d'allocation de mémoire. Les pointeurs en paramètre des fonctions doivent donc pointer vers des objets instanciés.

Inclure `windows-producer.h` à votre projet . Appeler `produce_images()` toutes les 100 ms. Au aurez à comprendre toute la documentation présente dans `windows-producer.h`

Définir un handler qui indiquera dans une variable globale que les nouvelles images sont prêtes.

Développer une fonction qui surveille le paramètre global précédent et calcule le flux pondéré pour chaque images 6x6 lorsque qu'elles sont prêtes.

>> Présenter dans un tableau et un graphique les variations de flux de 5 étoiles au court d'une seconde ?