1 SEANCE 4 : Mise en œuvre d'une fonction d'interruption externe sur le Beck.

Rappels des objectifs :

L'idée est de pouvoir déclencher une fonction d'interruption lorsqu'on présente un front montant sur la broche INT0 (PIO13) du Beck.

2 Explication pour la mise en place d'une fonction d'IT externe INT0

La mise en place d'une IT passe par 3 étapes :

1) déclarer et écrire le code de la fonction d'interruption : c 'est l'ISR (Interrupt SDub Routine)

2) Configurer la broche INT0/PIO13 pour quelle sera d'IT active sur front montant et installer la fonction d'interruption auprès de l'OS pour qu'il puisse l'appeler à chaque front montant de INT0

3) en fin de programme principal, on peut désinstaller ce mécanisme d'IT

2.1 Prototype d'une fonction d'interruption :

```
void interrupt My_ISR_INT0(void) {
// code todo in ISR


 }
```

Le prototype d'une fonction d'IT pour le Beck est un peu particulier car c'est une fonction déclenchée par du matériel. Elle doit être précédée du mot clé « interrupt ». Cela indique que cette fonction doit prévoir une sauvegarde du contexte avant de pouvoir s'exécuter ainsi qu'une restitution du contexte en fin d'exécution.

2.2 Configuration de la broche INT0/PIO13 en tant que broche d'interruption

```
void Init_INT0_ISR(void) {
 void pfe_enable_int ( unsigned char irq );

 void pfe_set_edge_level_intr_mode ( unsigned char mode,
                                     unsigned short mask );

InterruptHandler hal_install_isr ( unsigned short irq,
unsigned short count,InterruptHandler handler );
}
```

Dans cette fonction d'initialisation de l'IT INT0, on fait appel à trois fonctions de l'API Beck :

Pfe_enable_int() => permet de configurer la broche int0/PIO13 comme broche d'IT. On lui indique en paramètre d'entrée le numéro de la broche à paramétrer (ici la broche PIO13)

Pfe_set_edge_level_intr_mode() : permet de définir si on travail sur front ou sur niveau et si on veut attendre un certain nombre de fronts avant de déclencher réellement l'ISR.

Hal_install_isr() : permet d'enregistrer notre fonction d'ISR auprès de l'OS et on lui indique la fonction à appeler (on lui passe un pointeur sur notre fonction d'ISR). ON lui indique aussi la source d'IT matérielle (définie INT0) qui doit provoquer l'appel de notre ISR.

Ces trois fonctions sont regroupées au sein d'une fonction d'Initialisation unique : Init_INT0_ISR().

Désinstallation de la fonction d'interruption en fin de programme.

```
void DeInit_INT0_ISR(void) {
InterruptHandler hal_install_isr ( unsigned short irq,
unsigned short count,                                    InterruptHandler
handler );
}
```

En fait, pour désinstaller une fonction d'IT liée à une source matérielle d'interruption il faut tout simplement réutiliser la même fonction mais en lui indiquant que la fonction d'ISR à appeler n existe plus (Pointeur sur fonction NULL). Cela suffit pour la désinstaller.

Exemple de code complet mettant en œuvre une interruption.

```
////////////////////////////////////////////////////////////////////
// Detection d'un front montant sur INT0 par interruption
// unsigned char  asm_ReadPE (void) ;
// OK le xx/yy/zzzz
////////////////////////////////////////////////////////////////////
#include <stdio.h>
#include <clib.h>

#define INT0 0

unsigned char gbyCompteur ;

unsigned char gbyFlag_INT0 ;

void interrupt  My_ISR_INT0(void) {
      // code todo in ISR
      gbyFlag_INT0=1;

 }

void Init_INT0_ISR(void) {


}


void DeInit_INT0_ISR(void) {

}


void main (void)  {

   gbyCompteur=0;
   gbyFlag_INT0=0; // set initial value of Flag
   Init_INT0_ISR();// install our ISR
   BIOS_Set_Focus ( FOCUS_APPLICATION);//set SDTIO focus to app. only
      do {
      if (gbyFlag_INT0==1) {
            gbyFlag_INT0=0; // reset flag
            printf("\n front detecte sur INT0 \n");
         }
   } while (BIOS_kbhit()!=1) ;

printf("\n fin programme ");
DeInit_INT0_ISR();// deinstall our ISR
BIOS_Set_Focus ( FOCUS_SHELL);//set SDTIO to SHELL/ application
}//end main
```