Microcontrôleur ARM Cortex M3
TD 2 : Gestion du temps

Ce TD doit vous permettre de comprendre comment effectuer des temporisations élémentaires en langage C et de réaliser quelques programmes simples :

- *Structure d'une boucle d'attente,*
- *Durée de temporisation,*
- *Utilisation d'un Timer,*
- *Comportement temporel d'un bouton poussoir,*
- *Exercices pour le TP.*

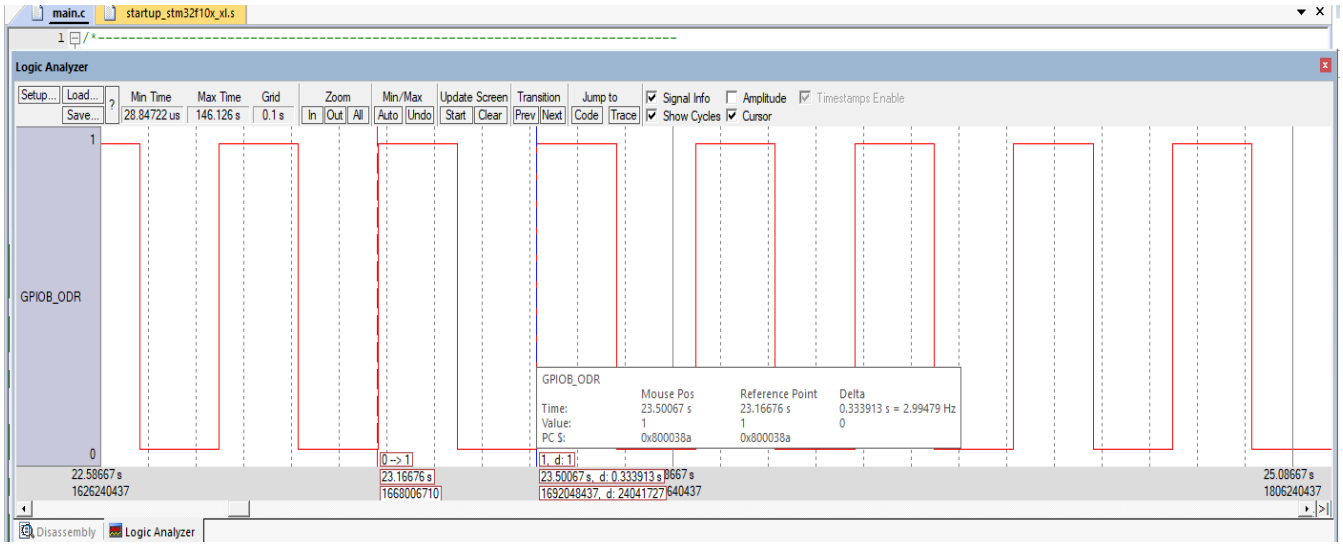
Exercice 1 : Temporisation

D'après le programme ci-dessous :

```
/*-------------------------------------------------------------------------
 * Name:    Clign_tempo.c
 * Purpose: GPIO usage for STM32
 * Version: V2.00
 *-------------------------------------------------------------------------*/


#include <stm32f10x.h>        // STM32F10x Library Definitions

#define LEDS_PORTB   GPIOB->ODR

void TempoCochon (unsigned long i)
{
    while (i--);
}
void Enable_GPIO(void)
{
  RCC->APB2ENR |= (1 << 3); // Enable GPIOB clock

}

void Init_GPIO(void)
{
  GPIOB->CRH = 0x33333333; // Mode=0b11 (50MHz) et CNF=0b00 (Push-Pull) pour PB8 à PB15 (Leds 8 à 15)

}
int main (void)
{
    Enable_GPIO(); // Validation des ports
    Init_GPIO(); // Initialisation des ports tels que définis au TD-TP1

    LEDS_PORTB = 0x0100;

    while (1)
    {
    LEDS_PORTB ^= (1 << 8 );
      TempoCochon(1000000);
}
}
```

- *Etablir l'organigramme complet de ce programme*
- *Expliquer ce que réalise la fonction TempoCochon()*
- *A quoi correspond la valeur du paramètre passé à cette fonction ?*

Voici la capture écran de la simulation de ce programme avec l'analyseur logique de ukeil5 :



- *En déduire la valeur en seconde de cette temporisation.*
- *Comment peut-on prévoir la durée d'exécution de cette fonction ?*
- *En déduire le comportement de la LED8 .*

Exercice 3 : Utilisation d'un Timer

1) Afin de définir précisément la durée d'une temporisation, il est préférable d'utiliser un Timer. Le programme ci-dessous utilise le Timer1 (TIM1) du microcontrôleur pour effectuer cette temporisation :

```
/*-------------------------------------------------------------------------
 * Name:    Tempo_timer.c
 * Purpose: GPIO usage for STM32
 * Version: V1.00
 *-------------------------------------------------------------------------*/

#include <stm32f10x.h>        // STM32F10x Library Definitions

#define LEDS_PORTB   GPIOB->ODR

unsigned int temps = 0;


void Enable_GPIO(void)
{
  RCC->APB2ENR |= (1 << 3); // Enable GPIOB clock

}

void Init_GPIO(void)
{
  GPIOB->CRH = 0x33333333; // Mode=0b11 (50MHz) et CNF=0b00 (Push-Pull) pour PB8 à PB15 (Leds 8 à 15)
}

int main (void)
{
    Enable_GPIO(); // Validation des ports
    Init_GPIO(); // Initialisation des ports tels que définis au TD-TP1

    //Initialisation Timer 1 (TIM1)
  RCC->APB2ENR |= (1 << 11);      // Validation Timer1
  TIM1->CR1 |= 0x0001;            // Timer1 Enable

    LEDS_PORTB = 0x0100;

    while (1)
    {
    if (TIM1->CNT == 64000)
    {
      temps ++;
      TIM1->CNT = 0x0000;
    }

    if (temps == 225)
    {
      LEDS_PORTB ^= (1 << 8 );
      temps = 0;
      }
    }
}
```

- *Observer les instructions d'initialisation des registres du Timer1 et déterminer son comportement*
- *Analyser le déroulement du programme*
- *En considérant que l'horloge du Timer1 (CK_CNT) est pilotée par l'oscillateur interne (CK_INT) à une fréquence f_{HSI} = 72 MHz, déterminer la période de clignotement de la LED.*
- *A votre avis, pour quelle raison ce programme ne fonctionnera pas correctement ?*

2) *La méthode utilisée ci-dessus (scrutation du contenu du Timer) ne garantit pas d'obtenir une durée de temporisation constante. En effet, il se peut que le microcontrôleur se trouve dans une boucle au moment où le Timer atteint (et dépasse) la valeur désirée. Dans ce cas, il faudra attendre un tour*

complet du Timer pour atteindre (peut-être) à nouveau la valeur désirée, ce qui faussera la durée de temporisation.

Pour limiter ce problème, on préfèrera utiliser le bit UIF *(registre* TIM1_SR*) avec la fonction Auto-Reload du Timer (registre* TIM1_ARR*).*

- *Déterminer à quelle valeur devra être initialisé le registre* TIM1_ARR *pour obtenir la même durée de temporisation.*
- *A quel moment le bit* UIF *du registre* TIM1_SR *sera-t-il positionné (mis à 1)?*
- *Modifier le programme précédent en conséquence*

3) *Dans un deuxième temps, pour éviter l'emploi de la variable « temps » on utilisera le prescaler (registre TIM1_PSC):*
   - *Déterminer la valeur qui devra être chargée dans le prescaler*
   - *Modifier le programme*

Exercice 4 : préparation du TP

Pour le TP relatif aux Temporisations, vous préparerez les exercices suivants. Vous utiliserez des « tempos cochon » si besoin pour les problèmes de rebonds (la bonne solution sera vu après le TD et TP sur les interruptions):

1) Modification de la durée de clignotement :

Modifier le programme de l'exercice 3, pour que la durée de clignotement soit multipliée par 2 lors d'un appui sur le bouton BP_USER.

2) Chenillard :

Ecrire un programme qui permettra de faire défiler les LEDS de la façon suivante : LED8 à LED15 puis LED15 à LED8. La vitesse de défilement devra être de 2 LEDs/seconde.

3) Chenillard à double sens :

Ecrire un programme qui permettra de faire défiler les LEDS du chenillard vers la droite lors d'une action (impulsion) sur le bouton JOY_DROITE et vers la gauche après une action sur JOY_GAUCHE. La vitesse de défilement devra être de 2 LEDs/seconde.

Ces applications devront être préparées (et éditées sous Keil) avant le TP afin d'être testées, validées et rédigées au cours de celui-ci