TP2 – STR : Gestion des E/S par interruption

1. Introduction

Nous avons montré, dans le TP précédent que la méthode interrogation traite les événements dans un ordre fixé par le programmé. Cette séquentialité du traitement des événements ne garantit pas l'exécution en temps réels des tâches critiques ou à faible échéances.

Faisant appel aux interruptions, nous espérons dans ce TP de palier les inconvénients de la méthode interrogation. Les microcontrôleurs PIC16/PIC18 n'utilisent pas le mécanisme de vectorisation des interruptions. Toutes les interruptions pointent vers une seule adresse (ou deux pour les PIC18); les sources d'interruptions sont déterminées par interrogation dans la routine d'interruption (ISR : Interrupt Service Routine).

Etant donné le programme suivant : les événements associés aux entrées RB0 et RB1 sont traité par interruption (INT0 et INT1).

```
#include <xc.h>
#define _XTAL_FREQ 8000000
#pragma config OSC = HS, WDT = OFF, LVP = OFF
#define LED1 PORTCbits.RC0
#define LED2 PORTCbits.RC1
#define LED3 PORTCbits.RC2
#define Flag_IT0 INTCONbits.INT0IF
#define Flag_IT1 INTCON3bits.INT1IF


void Init(void) {
  ADCON1 = 0x0F;
  TRISC = 0x00;
  TRISB = 0xFF;
  INTCONbits.INT0IE = 1
  INTCON3bits.INT1IE = 1;
  INTCONbits.GIE = 1;
}
void Traite_IT0(void){
   Flag_IT0 = 0;
   __delay_us(200);
}
```

```
void Traite_IT1(void){
   Flag_IT1 = 0;
   __delay_us(150);
}
void __interrupt()isr(void) {
LED3 = 1;
    if(Flag_IT0 == 1)
     {LED1=1; Traite_IT0(); LED1=0;}
    else
    {
      if(Flag_IT1 == 1)
        {LED2=1; Traite_IT1(); LED2=0;}
    }
 LED3 = 0;
}
void main()
{
   Init();
   while(1) {
   }
}
```

On suppose que les interruptions sont générées périodiquement avec P1 = P2 = 400us et, on prend pour une largeur d'impulsion de 10us pour chacun des événements.

Travail demande

1. Donner le temps d'exécution de chaque Tâche.
2. Donner le temps d'exécution de la routine d'interruption. (On ne considère que le temps d'exécution des fonctions « delay »).
3. Tapez le programme ci-dessus sous MPLAB XC8.
4. Faites le schéma de simulation avec le logiciel ISIS conformément la figure 1. Les évènements sont générés par une horloge DCLOCK. Utilisez le mode graphique DIGITAL pour visualiser les signaux des entrées/sorties (fixez le temps de simulation à 1.2ms).

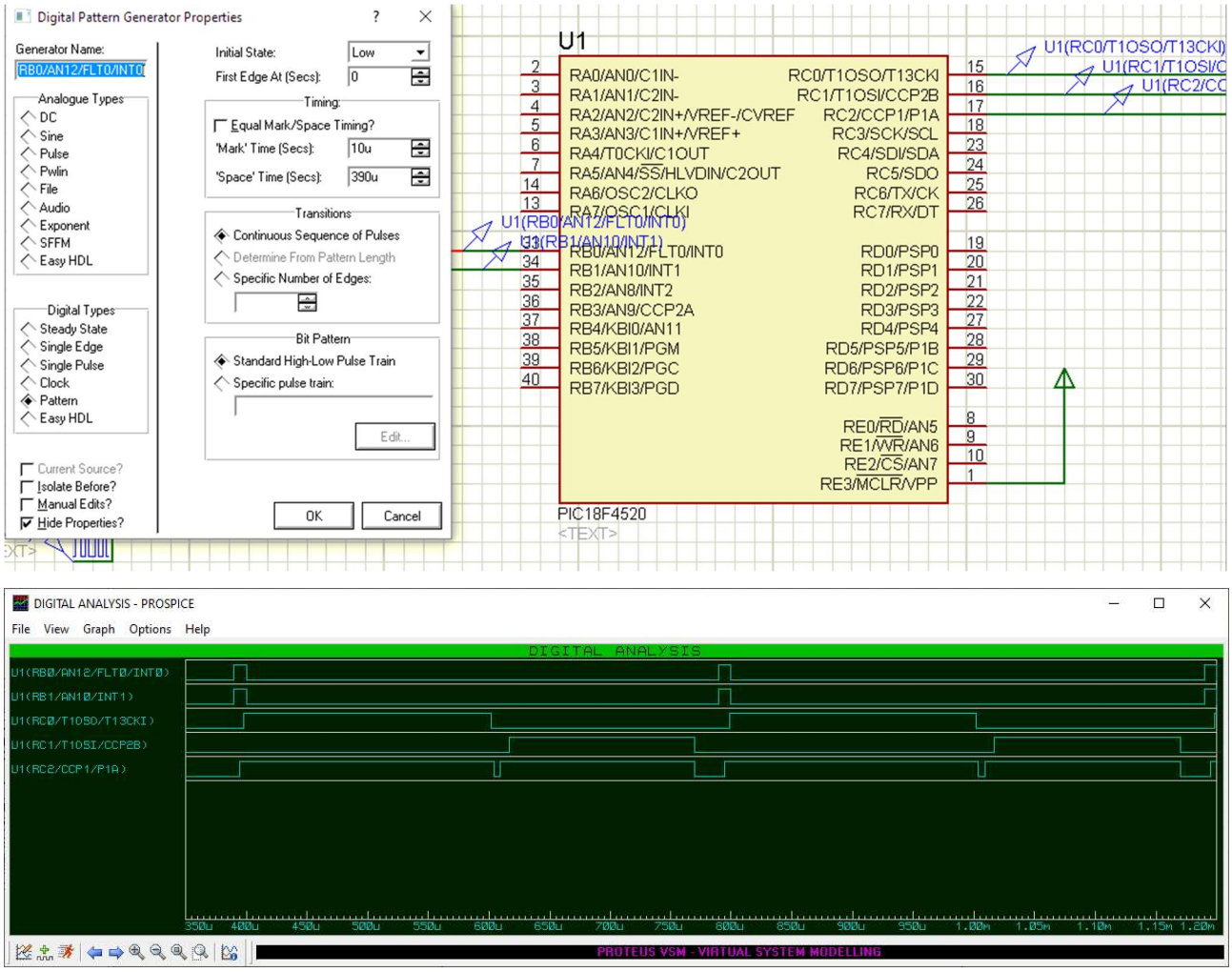


Figure 1

5. Estimer le temps d'exécution de la boucle (Cb).
6. Que peut-on dire du graphe obtenu ? Autrement-dit, y a-t-il des événements ratés par le système.
7. Changez les largeurs des impulsions des événements en maintenant toujours une période de 400µs.
8. Interpréter les signaux de la figure 1 ?

2. Interruptions vectorisées

Nous allons émuler les interruptions vectorisées en introduit la priorité des interruptions. Dans ce cas, chaque interruption aura son propre vecteur d'interruption. L'INT0 aura l'adresse 0x0008 (haute priorité) et INT1 l'adresse 0x0018 (Base priorité).

```
#include <xc.h>
#define _XTAL_FREQ 8000000
#pragma config OSC = HS, WDT=OFF, LVP=OFF
#define LED1 PORTCbits.RC0
#define LED2 PORTCbits.RC1
#define LED3 PORTCbits.RC2
#define Flag_IT0 INTCONbits.INT0IF
#define Flag_IT1 INTCON3bits.INT1IF
void Init(void) {
  ADCON1 = 0x0F;
  TRISC = 0x00;
  TRISB = 0xFF;
  RCONbits.IPEN = 1;
  INTCON3bits.INT1P = 0;
```

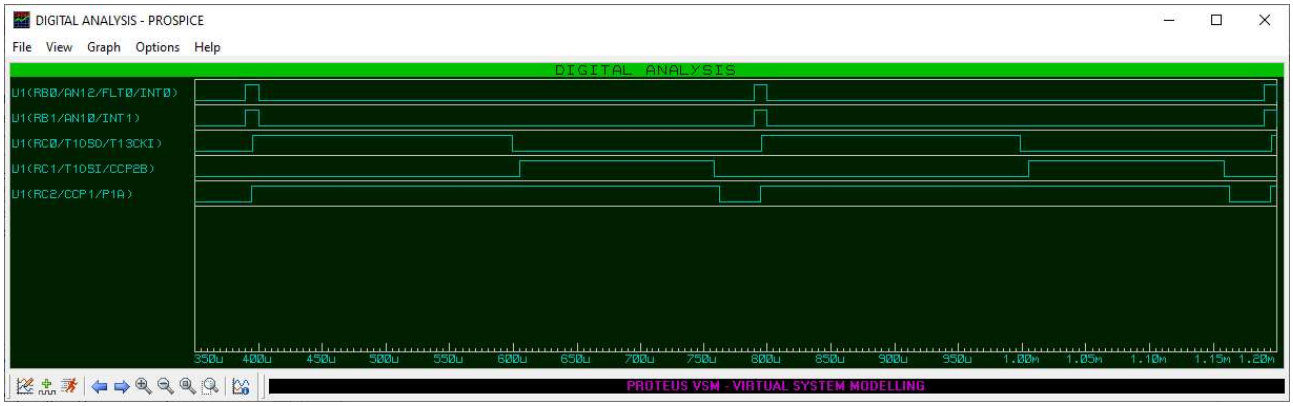
```
void Tache_fond(){
   __delay_us(500);
}


void __interrupt(high_priority) H_isr(void)
{  LED3 = 1;
   LED1 = 1;
   Traite_IT0();
   LED1 = 0;
}
void __interrupt(low_priority) L_isr(void)
{   LED3 = 1;
    LED2 = 1;
```

```
  INTCONbits.INT0IE = 1;
  INTCON3bits.INT1IE = 1;
  INTCONbits.GIEL = 1;
  INTCONbits.GIEH = 1;
}
void Traite_IT0(void){
   Flag_IT0 = 0;
   __delay_us(200);
}
void Traite_IT1(void){
   Flag_IT1 = 0;
   __delay_us(150);
}
```

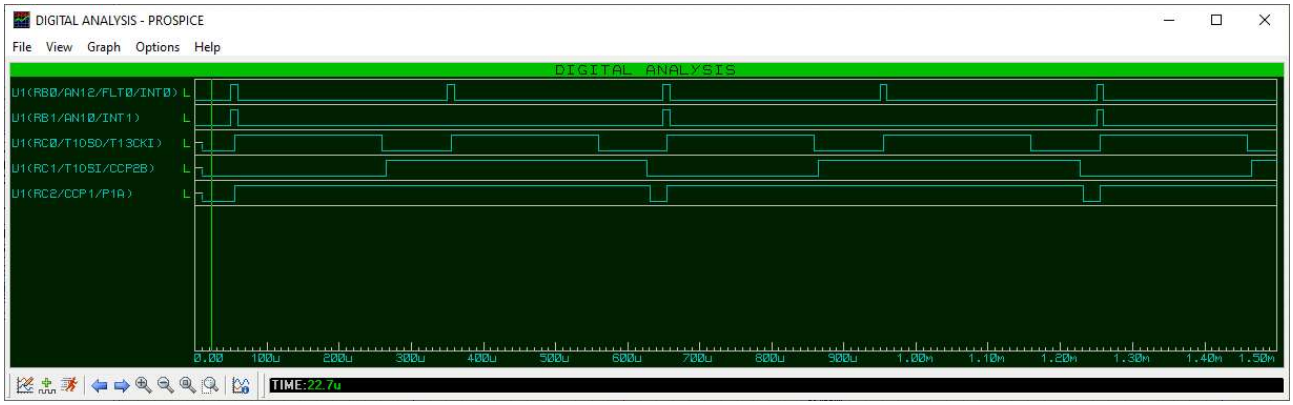
```
    Traite_IT1();
    LED2 = 0;
}
void main()
{
   Init();
   while(1)
   {
      LED3 = 0;
   }
}
```

Travail demande

1. Tapez le code ci-dessus et faites la simulation sur isis (P1 = P2 = 400µs)
2. Interpréter les signaux du graphe suivant.
3. Déterminer approximativement le temps de latence du sauvegarde et récupération de contexte.



4. Nous avons modifier les périodes des évènements P1 = 300µs (10+290) et P2 = 600µs(10+590). Interpréter les signaux du graphe suivant.
5. Déterminer le temps minimal et maximal de la boucle.



6. Apporter au programme les modifications suivantes :
   - Remplacer __delay_us(200) par __delay_ms(4) ;
   - Remplacer __delay_us(150) par __delay_ms(3) ;
   - Ajouter dans la boucle infinie une tâche de fond de 20ms.

Déterminer expérimentalement le temps maximum de la boucle (On prend P1 = P2 = 10ms).