Corrections

TD2 - exo 3

```
      CBLOCK 0x00 ; début de la zone variables en ACCESS RAM
            somme_pdsfort :1     ; Zone de 1 byte
            somme_pdsfaible : 1     ; zone de 1 bytes
            compteur :1  ; zone de 1 byte
            moyenne :1 ; le resultat sera ici
      ENDC ; Fin de la zone
      ...
debut
      movlw 0x0
      movwf somme_pdsfort
      movwf somme_pdsfaible
      movlw 0x4
      movwf compteur       ; initialiser compteur
boucle
      movf PORTB,W            ; PORTB -> W
      addwf somme_pdsfaible,f ;somme_pdsfaible + W ->somme_pdsfaible
      btfsc STATUS,C          ; test si pas retenue
      incf somme_pdsfort,f    ; sinon incremente somme_pdsfort
      decfsz compteur , f  ; décrémenter compteur et tester sa valeur
      goto boucle
      ; il faut diviser par 4
      bcf STATUS,C            ;C<-0
      rrf somme_pdsfort,f     ;division par 2
      rrf somme_pdsfaible,f   ;division par 2
      bcf STATUS,C            ;C<-0
      rrf somme_pdsfort,f     ;division par 2
      rrf somme_pdsfaible,f   ;division par 2
; on a la moyenne dans somme_pdsfaible
      movf somme_pdsfaible , w  ; on charge la valeur obtenue dans w
      movwf moyenne,f   ;on range en 14 comme demandé
      goto debut
END
```

TD3 - exo 1

```
CBLOCK 0x00 ; début de la zone variables en ACCESS RAM
      somme_pdsfort :1     ; Zone de 1 byte
      somme_pdsfaible : 1     ; zone de 1 byte
      compteur :1  ; zone de 1 byte
      tab:4       ;zone 4 bytes
ENDC ; Fin de la zone
      ...
debut
      movlw 0x0
      movwf somme_pdsfort
      movwf somme_pdsfaible
      movlw 0x4
      movwf compteur       ; initialiser compteur
      movlw tab
      movwf FSR               ; W -> FSR
        ; mettre aussi STATUS,RP1=0 et STATUS,IRP=0 pour compatibilité
boucle
      movf PORTB,w            ; PORTB -> W
      movwf INDF             ;(FSR)<-W
      incf FSR,f
      decfsz compteur , f  ; décrémenter compteur et tester sa valeur
      goto boucle
```

```
; initialisation boucle 2
      movlw 0x4
      movwf compteur       ; initialiser compteur
      movlw tab
      movwf FSR
boucle2
      movf INDF,w         ; W <-(FSR)
      incf FSR,f
      addwf somme_pdsfaible,f ;somme_pdsfaible + W ->somme_pdsfaible
      btfsc STATUS,C          ; test si pas retenue
      incf somme_pdsfort,f    ; sinon incremente somme_pdsfort
      decfsz compteur , f  ; décrémenter compteur et tester sa valeur
      goto boucle2
      ; il faut diviser par 4
      bcf STATUS,C            ;C<-0
      rrf somme_pdsfort,f     ;division par 2
      rrf somme_pdsfaible,f   ;division par 2
      bcf STATUS,C            ;C<-0
      rrf somme_pdsfort,f     ;division par 2
      rrf somme_pdsfaible,f   ;division par 2
; on a la moyenne dans somme_pdsfaible
      movf somme_pdsfaible , w  ; on charge la valeur obtenue dans w
      movwf 0x14  ;on range en 14 comme demandé
      goto debut
END
```

TD3 - exo 2

```
      CBLOCK 0x00 ; début de la zone variables en ACCESS RAM
            somme_pdsfort :1     ; Zone de 1 octet
            somme_pdsfaible : 1     ; zone de 1 octet
            compteur :1  ; zone de 1 octet
            tab:4       ;zone 4 octets
            Max:1 ;;;;;;;;zone 1 octet
      ENDC ; Fin de la zone
      ...
debut
      movlw 0x0
      movwf somme_pdsfort
      movwf somme_pdsfaible
        movwf Max       ;;;;;;;; initialisation de Max
      movlw 0x4
      movwf compteur       ; initialiser compteur
      movlw tab
      movwf FSR               ; W -> FSR
        ; mettre aussi STATUS,RP1=0 et STATUS,IRP=0 pour compatibilité
boucle
      movf PORTB,w            ; PORTB -> W
      movWf INDF             ;(FSR)<-W
      subwf Max,w       ;;;;;;;;;;Max - W -> W
      btfsS STATUS,C          ;;;;;;;;;;c=1 si >0 et on saute
      GOTO suite
      GOTO suite2
suite  movf INDF,w         ;;;; W <-(FSR) on recherche derniere valeur
      movwf Max       ;;;;;;;;;;pour mettre dans Max
suite2 incf FSR,f
      decfsz compteur , f  ; décrémenter compteur et tester sa valeur
      goto boucle
; initialisation boucle 2
      movlw 0x4
      movwf compteur       ; initialiser compteur
      movlw tab
      movwf FSR
```

```
boucle2
      movf INDF,w         ; W <-(FSR)
      incf FSR,f
      addwf somme_pdsfaible,f ;somme_pdsfaible + W ->somme_pdsfaible
      btfsc STATUS,C          ; test si pas retenue
      incf somme_pdsfort,f    ; sinon incremente somme_pdsfort
      decfsz compteur , f  ; décrémenter compteur et tester sa valeur
      goto boucle2
      ; il faut diviser par 4
      bcf STATUS,C            ;C<-0
      rrf somme_pdsfort,f     ;division par 2
      rrf somme_pdsfaible,f   ;division par 2
      bcf STATUS,C            ;C<-0
      rrf somme_pdsfort,f     ;division par 2
      rrf somme_pdsfaible,f   ;division par 2
; on a la moyenne dans somme_pdsfaible
      movf somme_pdsfaible , w  ; on charge la valeur obtenue dans w
      movwf 0x14  ;on range en 14 comme demandé
      goto debut
END
```

TD4 - exo1

La modification simpliste suivante fonctionne. Le sous-programme correct est appelé mais les tous tests suivants (l'appel) seront effectués pour rien !

```
              processor 16f84
STATUS      EQU   0x03
Z           EQU   2
C           EQU   0
W           EQU     0
f           EQU     1
choix EQU 0x0C
ORG 0
debut
;si Choix=0 faire le sous programme spgm0 si Choix=1 faire spgm1
      ; Choix contient ici une valeur <=N
      movlw 0x0
      subwf Choix,w           ;Choix - W -> W
      btfsC STATUS,Z          ;Z=1 si = 0 et on saute
      CALL spgm0
        ;GOTO suite
      movlw 0x1
      subwf Choix,w           ;Choix - W -> W
      btfsC STATUS,Z          ;Z=1 si = 0 et on saute
      CALL spgm1
        ;GOTO suite
      movlw 0x2
      subwF Choix,w           ;Choix - W -> W
      btfsC STATUS,Z          ;Z=1 si = 0 et on saute
      CALL spgm2
      GOTO suite
      ; erreur ici !!!
suite   GOTO debut
spgm0       ;code ici
      RETURN
spgm1       ;code ici
      RETURN
spgm2       ;code ici
      RETURN
END
```

On peut donc procéder comme ceci :

```
ORG 0
debut
;si Choix=0 faire le sous programme spgm0 si Choix=1 faire spgm1
```

```
      ; Choix contient ici une valeur <=N
      movlw 0x0
      subwf Choix,w          ;Choix - W -> W
      btfsC STATUS,Z         ;Z=1 si = 0 et on saute
      GOTO CALL_spgm0
      movlw 0x1
      subwf Choix,w          ;Choix - W -> W
      btfsC STATUS,Z         ;Z=1 si = 0 et on saute
      GOTO CALL_spgm1
        ;GOTO suite
      movlw 0x2
      subwF Choix,w          ;Choix - W -> W
      btfsC STATUS,Z         ;Z=1 si = 0 et on saute
      GOTO CALL_spgm2
      ; si on est ici c'est pas normal : erreur !!!
suite   GOTO debut
; debut des appels
CALL_spgm0
        CALL spgm0
        GOTO suite
CALL_spgm1
        CALL spgm1
        GOTO suite
CALL_spgm2
        CALL spgm2
        GOTO suite
; code des sous-programmes
spgm0       ;code ici
      RETURN
spgm1       ;code ici
      RETURN
spgm2       ;code ici
      RETURN
END
```

L'amélioration demandée peut être faite en remarquant dans le code ci-dessus quelque chose qui ressemble à la déclaration de tableau :

```
CALL_spgm0
        CALL spgm0
        GOTO suite
CALL_spgm1
        CALL spgm1
        GOTO suite
CALL_spgm2
        CALL spgm2
        GOTO suite
```

On va le transformer en

```
tableau_Appels
        BCF STATUS,C  ;C<-0
        RLF choix,f   ; choix=choix x 2
        MOV choix,w
        addwf PCL , f ; ajouter w à PCL
        CALL spgm0
        GOTO suite
        CALL spgm1
        GOTO suite
        CALL spgm2
        GOTO suite
```

soit :

```
              processor 16f84
STATUS            EQU   0x03
PCL             EQU     0x02
Z           EQU   2
```

```
C           EQU   0
W               EQU     0
f               EQU     1
choix EQU 0x0C
ORG 0
debut
;si Choix=0 faire le sous programme spgm0 si Choix=1 faire spgm1
      ; Choix contient ici une valeur <=N
         GOTO tableau_Appels
suite   GOTO debut

; tableau en mémoire programme des appels
tableau_Appels
        BCF STATUS,C  ;C<-0
        RLF choix,f   ; choix=choix x 2
; les tests de dépassement ne sont pas faits
        movf choix,w
        addwf PCL , f ; ajouter w à PCL
        CALL spgm0
        GOTO suite
        CALL spgm1
        GOTO suite
        CALL spgm2
        GOTO suite
; code des sous-programmes
spgm0       ;code ici
      RETURN
spgm1       ;code ici
      RETURN
spgm2       ;code ici
      RETURN
END
```

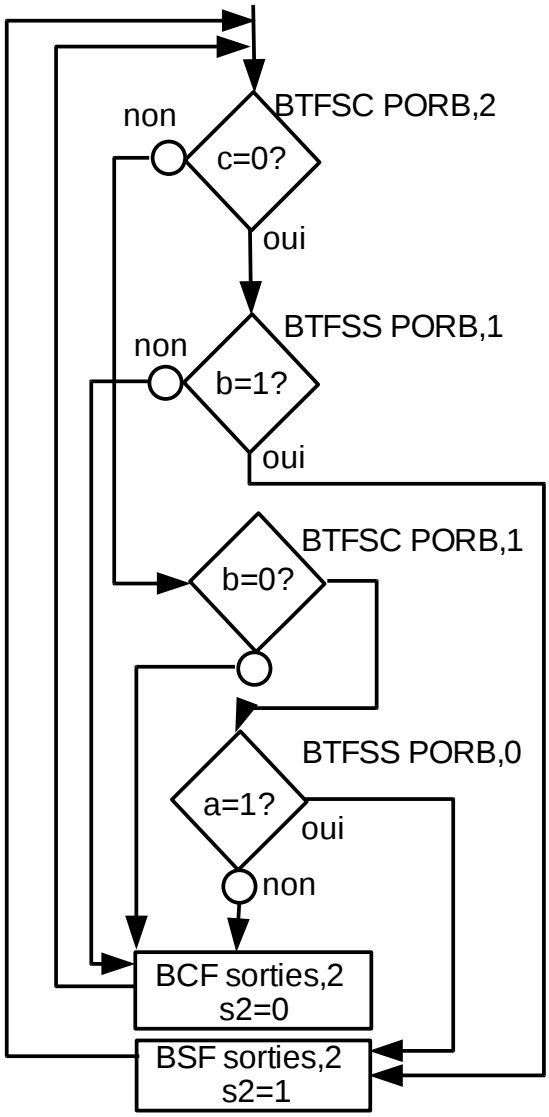
TD5 Question 1

L'entrée est dans le portB et la sortie est dans une variable (appelée sorties):

```
              processor 16f84
PORTB      EQU     0X06
PCL         EQU   0x02
W          EQU     0
f          EQU     1
sorties EQU 0X0D
ORG 0
debut
        movf PORTB,W          ; PORTB -> W
        ANDLW 0x07
        CALL tableau
        MOVWF sorties           ; W->entrees
        GOTO debut
tableau
      addwf PCL , f ; ajouter w à PCL
      retlw      2               ;  premier élément = 2
      retlw      3               ;  deuxième élément = 3
      retlw      4               ;  troisième élément = 4
      retlw      5               ;  quatrième élément = 5
      retlw      2               ;  cinquième élément = 2
      retlw      6               ;  sixième élément = 6
      retlw      0               ;  septième élément = 0
      retlw      3               ;  huitième élément = 3
END
```

TD5 Question 2

On refait le diagramme de test pour qu'il soit linéaire. Par exemple :




```
              processor 16f84
PORTB           EQU     0X06

sorties EQU 0X0D

ORG 0
debut
        BTFSC PORTB,2  ; c=0?
        GOTO b_0       ;non
        GOTO b_1       ;oui
b_1     BTFSS PORTB,1  ; b=1?
        GOTO s2_0      ;non
        GOTO s2_1      ;oui
b_0     BTFSC PORTB,1  ; b=0?
        GOTO s2_0      ;non
        GOTO a_1       ;oui
a_1     BTFSS PORTB,0  ; a=1?
        GOTO s2_0      ;non
```

```
        GOTO s2_1      ;oui
s2_1    BSF sorties,2
        GOTO debut
s2_0    BCF sorties,2
        GOTO debut
END
```

Remarque : les goto avec l'étiquette juste en-dessous peuvent être enlevés.

TD8 Clavier et ports

TD9 Timer0

Exercice 1

1°)

```
unsigned int div10(unsigned int A){
      unsigned int Q; /* the quotient */
      Q = ((A >> 1) + A) >> 1; /* Q = A*0.11 */
      Q = ((Q >> 4) + Q)     ; /* Q = A*0.110011 */
      Q = ((Q >> 8) + Q) >> 3; /* Q = A*0.00011001100110011 */
        /* either Q = A/10 or Q+1 = A/10 for all A < 534,890 */
      return Q;
}
```

2°)

```
// initialisation du timer dans cet ordre
      TMR0H = 0x00;
      TMR0L = 0x00;
      T0CONbits.TMR0ON = 1; // c'est parti


// Arret du timer
      T0CONbits.TMR0ON = 0; // c'est fini
// unsigned int res; unsigned char timer0L;
      timer0L=TMR0L;// obligatoire en premier
      res=TMR0H;
      res<<=8;
      res += timer0L;
      printf( "Temps ecoule alogo 1 : %d \n",res);
```

3°)

```
// preparation du timer 0
// quartz comme source
      T0CONbits.T0CS = 0;
// si 0 pas besoin diviseur suivant
      T0CONbits.PSA = 1;
// division par 256
      T0CONbits.T0PS0 = 1;
      T0CONbits.T0PS1 = 1;
      T0CONbits.T0PS2 = 1;
// passage en 16 bits
      T0CONbits.T08BIT = 0;
```

Exercice 2

Question 1

```
// quartz comme source
      T0CONbits.T0CS = 0;
// si 0 pas besoin diviseur suivant
      T0CONbits.PSA = 1;
//  division par 1
      T0CONbits.T0PS0 = 0;
      T0CONbits.T0PS1 = 0;
      T0CONbits.T0PS2 = 0;
// passage en 8 bits
```

```
            T0CONbits.T08BIT = 1;
//          TMR0H = 0x00; inutile car en 8 bits
            TMR0L = 0x00;
            T0CONbits.TMR0ON = 1; //C'est parti
            while(1) {
                  INTCONbits.TMR0IF = 0;
                  while (INTCONbits.TMR0IF == 0);
                  printf("Overflow \n");
            }
```

Question 2

```
            while(1) {
            //    TMR0H = 0x00; inutile car en 8 bits
                  TMR0L = 255-100;
                  INTCONbits.TMR0IF = 0;
                  while (INTCONbits.TMR0IF == 0);
                  printf("Overflow \n");
            }
```

Question 3

On va en fait générer 2kHz et changer sans arrêt un bit RA_0 du **PORTA**.
1MHz / 2kHz=500. Si l'on divise par 2 on peut rester en 8 bits avec comme valeur 250-2=248 à compter donc une initialisation à 256-248=8. Ceci n'est qu'une valeur théorique qui dépend du compilateur. Il faut donc faire des essais pour trouver la valeur exacte. Une autre technique pour être quasi indépendant du compilateur c'est de remplacer :

```
TMR0L = 8;
```

par

```
TMR0L = TMR0L+8;
```

qui dépend moins du tmps que l'on a mis pour réaliser les instructions.

```
#include <p18f452.h>
#include <stdio.h>
#pragma config WDT = OFF
void main(void) {
// configure USART
      SPBRG = 25; // configure la vitesse (BAUD) 9600 N 8 1
      TXSTA = 0x24;
      RCSTA = 0x90;     // active l'USART
// quartz comme source
      T0CONbits.T0CS = 0;
// si 0 pas besoin diviseur suivant
      T0CONbits.PSA = 1;
//  division par 2
      T0CONbits.T0PS0 = 0;
      T0CONbits.T0PS1 = 0;
      T0CONbits.T0PS2 = 1;
// passage en 8 bits
      T0CONbits.T08BIT = 1;
      T0CONbits.TMR0ON = 1; //C'est parti
// bit RA0 du PORTA en sortie
      TRISAbits.TRISA0=0;
      while(1) {
      //    TMR0H = 0x00; inutile car en 8 bits
            TMR0L = TMR0L+8; // peut être le mieux
            INTCONbits.TMR0IF = 0;
            while (INTCONbits.TMR0IF == 0);
            PORTAbits.RA0 = PORTAbits.RA0 ^1;
      }
}
```

Question 4

Période diminue => fréquence augmente.

```
TMR0L = TMR0L+8;
```

est remplacé par

```
TMR0L = TMR0L+delta;
```

avec unsigned char delta; initialisé à 8 et augmenté progressivement.

Question 5

on génère 4kHz et :

```
#include <p18f452.h>
#include <stdio.h>
#pragma config WDT = OFF
void main(void) {
// configure USART
      SPBRG = 25; // configure la vitesse (BAUD) 9600 N 8 1
      TXSTA = 0x24;
      RCSTA = 0x90;     // active l'USART
// quartz comme source
      T0CONbits.T0CS = 0;
// si 0 pas besoin diviseur suivant
      T0CONbits.PSA = 1;
//  division par 2
      T0CONbits.T0PS0 = 0;
      T0CONbits.T0PS1 = 0;
      T0CONbits.T0PS2 = 1;
// passage en 8 bits
      T0CONbits.T08BIT = 1;
      T0CONbits.TMR0ON = 1; //C'est parti
// bit RA0 du PORTA en sortie
      TRISAbits.TRISA0=0;
      while(1) { //4kHz=125 125+2=127
            TMR0L = TMR0L+127; // peut être le mieux
            INTCONbits.TMR0IF = 0;
            while (INTCONbits.TMR0IF == 0);
            PORTAbits.RA0 = 1;
            TMR0L = TMR0L+127; // peut être le mieux
            INTCONbits.TMR0IF = 0;
            while (INTCONbits.TMR0IF == 0);
            PORTAbits.RA0 = 0;
            TMR0L = TMR0L+127; // peut être le mieux
            INTCONbits.TMR0IF = 0;
            while (INTCONbits.TMR0IF == 0);
            PORTAbits.RA0 = 0;
            TMR0L = TMR0L+127; // peut être le mieux
            INTCONbits.TMR0IF = 0;
            while (INTCONbits.TMR0IF == 0);
            PORTAbits.RA0 = 0;
      }
}
```

TD10 Comparaison

Exercice 1

division par 8 donne 125 kHz et le timer divise encore par 65536 ce qui donne 1,91 Hz

Exercice 2

1°) 9,53674 Hz alors que l'on veut 2 Hz donc division par 5.

2°) Dans la correction ci-dessous **CCPR1** = 0xFFFF mais toute autre valeur ferait l'affaire.

```
#include <p18f452.h>
// Quartz a 20 MHz on genere 2Hz sur b2 du PORTC
#pragma config WDT = OFF
void initTimer1(void);
void main(void) {
unsigned char nb,BitPORTC;
//Initialisation des variables
      BitPORTC=nb=0;
// PORTC bit b2 en sortie
```

```
      TRISCbits.TRISC2=0;
// CCP1IF seul donc 1010
      CCP1CONbits.CCP1M3=CCP1CONbits.CCP1M1=1;
      CCP1CONbits.CCP1M2=CCP1CONbits.CCP1M0=0;
// initialise le registre comparaison
      CCPR1H=CCPR1L=0xFF;
      initTimer1();
// clear le flag CCP1IF
      PIR1bits.CCP1IF=0;
      while(1){
// on attend le flag
            while(PIR1bits.CCP1IF==0);
            nb++;
// vu la frequence faible on a le temps
            if (nb==5){
                  nb=0;
            // ou exclusif pour inverser le bit
                  BitPORTC=BitPORTC^1;
                  PORTCbits.RC2=BitPORTC;
            }
// clear le flag CCP1IF
      PIR1bits.CCP1IF=0;
      }
}
void initTimer1(void){
  // division par 8
  T1CONbits.T1CKPS0=1;
  T1CONbits.T1CKPS1=1;
 // choix du quartz
   T1CONbits.TMR1CS=0;
// en synchronisé car comparaison
   T1CONbits.NOT_T1SYNC=0;
//initialisation du timer TMR1H en premier
  TMR1H = 0x00;
  TMR1L = 0x00;
// mise en route du timer1
 T1CONbits.TMR1ON=1;
}
```

3°) 2Hz => 0,5s On garde la division par 5 alors pour réaliser 2Hz=> à réaliser 0,1s
TQuartzx 4 x 8 x (x+1)=0,1s Trouver x ?
Remarque : le +1 du (x+1) est un détail de fonctionnement du PIC, la synchronisation se fait toujours à l'instruction suivante. On peut peut-être passer ce point de détail aux étudiants
x+1=62500. => x=62499.
62500 = 0xF424

```
#include <p18f452.h>
// Quartz a 20 MHz on genere 2Hz sur b2 du PORTC
#pragma config WDT = OFF
...
// CCP1IF seul et RAZ timer1 donc 1011
      CCP1CONbits.CCP1M3=CCP1CONbits.CCP1M1=CCP1CONbits.CCP1M0=1;
      CCP1CONbits.CCP1M2=0;
// initialise le registre comparaison
      CCPR1H=0xF4;
      CCPR1L=0x23; // ou 0x24
      initTimer1();
...
```

TD11 Capture

Exercice

questions 1-2) Le programme suivant m'a donné un résultat correct (en simulation) avec une horloge a 20 MHz (et même à 4MHz) et une PS2Clock à 10 kHz. Il n'y a donc pas de problème de rapidité car la construction du scancode n'est pas faite au fur et à mesure mais après la saisie des bits dans un tableau.

```
#include <p18f452.h>
#include <stdio.h>
#pragma config WDT = OFF
#define PS2DATA PORTCbits.RC1
#define PS2CLK PORTCbits.RC2
void initKBD(void);
void main(void) {
unsigned char i,result,data[11];
// configure USART
       SPBRG = 25;  // configure la vitesse (BAUD) 9600 N 8 1
       TXSTA = 0x24;
       RCSTA = 0x90;// active l'USART
       initKBD();
       for (i=0;i<11;i++) { // 11 fronts d'horloge
// attente d'un front descendant
              while( PIR1bits.CCP1IF==0);
              data[i]=PS2DATA;
       // clear le flag CCP1IF
              PIR1bits.CCP1IF=0;
       }
// maintenant on a le temps de mettre en forme
       result=0;
       for (i=1;i<9;i++)
              result+=data[i]<<(i-1);
// resultat sur PORTB comme demandé
       TRISB=0;
       PORTB=result;
// pour nos simulations sur PORT série
       printf("Le scancode est : %d\n",result);
       while(1);
}
void initKBD(void) {
// PS2CLK en entree
  TRISCbits.TRISC2=1;
// PS2DATA en entree
  TRISCbits.TRISC1=1;
// clear le flag CCP1IF
  PIR1bits.CCP1IF=0;
// detection de fronts descendants
 CCP1CONbits.CCP1M3=CCP1CONbits.CCP1M1=CCP1CONbits.CCP1M0=0;
 CCP1CONbits.CCP1M2=1;
}
```

Question 3

Un quartz 4MHz est encore assez rapide pour évaluer la fréquence de 10kHz sans interruption mais à l'aide d'une capture (donc avec attente du positionnement du bit CCP1IF). Le programme donné ci-après ne fait qu'afficher trois valeurs de mesure. Une simulation me donne 95 (au lieu de 100) pour les trois mesures. Dans un cas réel il serait bon de faire une moyenne sans prendre la première mesure qui est n'importe quoi puisqu'elle dépend de quand arrive le premier front descendant d'horloge.

```
#include <p18f452.h>
#include <stdio.h>
#pragma config WDT = OFF
void initKBD(void);
void initTimer1(void);
void initCaptureTimer1(void);
void main(void) {
unsigned char i;
unsigned int data[11],periode,tmr1;
// configure USART
       SPBRG = 25;  // configure la vitesse (BAUD) 9600 N 8 1
       TXSTA = 0x24;
       RCSTA = 0x90;// active l'USART
       initKBD();
```

```
	initTimer1();
	initCaptureTimer1();
	for (i=0;i<11;i++) { // 11 fronts d'horloge
// attente d'un front descendant
		while( PIR1bits.CCP1IF==0);
//initialisation du timer TMR1H en premier
		TMR1H = 0x00;
		TMR1L = 0x00;
//lecture du registre de capture
		data[i]=CCPR1H;
		data[i]<<=8;
		data[i]=CCPR1L+data[i];
	// clear le flag CCP1IF
		PIR1bits.CCP1IF=0;
	}
// maintenant on a le temps de calculer
	printf("La periode est : %d %d %d\n",data[3],data[4],data[5]);
	while(1);
}
void initKBD(void) {
// PS2CLK en entree
  TRISCbits.TRISC2=1;
// PS2DATA en entree
  TRISCbits.TRISC1=1;
// clear le flag CCP1IF
  PIR1bits.CCP1IF=0;
// detection de fronts descendants
 CCP1CONbits.CCP1M3=CCP1CONbits.CCP1M1=CCP1CONbits.CCP1M0=0;
 CCP1CONbits.CCP1M2=1;
}
void initTimer1(void){
  // division par 1
  T1CONbits.T1CKPS0=0;
  T1CONbits.T1CKPS1=0;
 // choix du quartz
   T1CONbits.TMR1CS=0;
// en synchronisé car capture
   T1CONbits.NOT_T1SYNC=0;
//initialisation du timer TMR1H en premier
  TMR1H = 0x00;
  TMR1L = 0x00;
// mise en route du timer1
 T1CONbits.TMR1ON=1;
}
void initCaptureTimer1(void){
  T3CONbits.T3CCP2=0;
}
```

TD 13

MikroC	C18

`void interrupt() {`
`  // traitement interruption`
`  // positionnement du FLAG`
`}`
 	
`void main() {`
`  // traitement`
`  while(1);`
`}`
	`#include<p18f452.h>`
	`#pragma config WDT = OFF`
	`#pragma config DEBUG=ON`
	`void it_prioritaire(void) ;`
	`#pragma code address_it=0x08 //0x18 pour basse priorité`
	`void int_toto(void) {`
	`  _asm GOTO it_prioritaire _endasm`
	`}`
	`#pragma code`
	`#pragma interrupt it_prioritaire`
	`void it_prioritaire(void) {`
	`  if (INTCONbits.TMR0IF) {`
	`  // traitement interruption`
	`  PORTBbits.RB0 = !PORTBbits.RB0;`
	`  //TMR0H = 0xFF;`
	`  TMR0L = 0xF0;`

	`// positionnement du FLAG`
	`INTCONbits.TMR0IF = 0;`
	`}`
	`}`

	`void main(void){`
	`// Initialisation`
	`TRISB = 0xFE;`
	`PORTBbits.RB0 = 0;`
	`INTCONbits.TMR0IF = 0;`
	`INTCONbits.TMR0IE = 1;`
	`INTCONbits.GIEH = 1;`
	`T0CON = 0xCF;`
	`//TMR0H = 0xFF;`
	`TMR0L = 0xF0;`
	`//INTCON2bits.TMR0IP=1; // inutile ?`
	`// traitement`
	`while(1);`
	`}`

TD 14

Compléments en vrac au TD 13

question 3 a verifier où va cette réponse

1000000/256 = 3906,25 => encore division par 4 mais se fait en mettant 255-4+2=253 dans timer0 pour chaque débordement. Mais attendre deux cycles seulement est un peu dangereux car dépendra beaucoup de l'optimisation du compilateur C. On va donc plutôt diviser par 64 et mettre 255-16+2 =241 dans timer0.

1°) (500.000/65536) = 7,63 fois

2°) en assembleur :

```
;*****************************************************************
;                     INTERRUPTION TIMER 0                       *
;*****************************************************************
inttimer
   decfsz   cmpt , f           ; décrémenter compteur de passages
   return                      ; pas 0, on ne fait rien
   BANK0                       ; par précaution
   movlw    b'00000100'        ; sélectionner bit à inverser
   xorwf    PORTA , f          ; inverser LED
   movlw    7                  ; pour 7 nouveaux passages
   movwf    cmpt               ; dans compteur de passages
   return                      ; fin d'interruption timer
```

En langage C :

```
#pragma interrupt MyInterrupt
```

```
main() {
  INTCON = INTCON | 0xA0; //GIE=1 TOIE=1
  INTCON = INTCON & 0xFB;    //T0IF=0 0b11111011
  OPTION = 0x87 // Valeur registre option
                   // Résistance pull-up OFF
                   // Préscaler timer à 256
  PORTA = 0; // initialisation à 0
  while(1);
}
/* configuration du PIC */
```

3°) /8 : donne 500000/2048 = 244,140625 passages. On initialise à 244 et l'on aura donc en une minute, 60000000/999424 = 60,034 allumages.

4°) 1000000µs => 1000000/(256*256*7*2) = 1,089913504µs
=> f=1/1,089913504µs = 0,917504 MHz => fquatrz =0,917504 * 4 = 3,670016 MHz
Le mieux est d'utiliser 1 quartz à 4MHz pouTD 10
1°) (500.000/65536) = 7,63 fois
2°) en assembleur :

```
;*****************************************************************
;                     INTERRUPTION TIMER 0                      *
;*****************************************************************
inttimer
   decfsz   cmpt , f           ; décrémenter compteur de passages
   return                      ; pas 0, on ne fait rien
   BANK0                       ; par précaution
   movlw    b'00000100'        ; sélectionner bit à inverser
   xorwf    PORTA , f          ; inverser LED
   movlw    7                  ; pour 7 nouveaux passages
```

```
   movwf    cmpt                ; dans compteur de passages
   return                       ; fin d'interruption timer
```

En langage C :

```
#pragma interrupt MyInterrupt
```

```
main() {
  INTCON = INTCON | 0xA0; //GIE=1 TOIE=1
  INTCON = INTCON & 0xFB;    //T0IF=0 0b11111011
  OPTION = 0x87 // Valeur registre option
                   // Résistance pull-up OFF
                   // Préscaler timer à 256
  PORTA = 0; // initialisation à 0
  while(1);
}
/* configuration du PIC */
```

En C (CCS)
//

3°) /8 : donne 500000/2048 = 244,140625 passages. On initialise à 244 et l'on aura donc en une minute, 60000000/999424 = 60,034 allumages.

4°) 1000000µs => 1000000/(256*256*7*2) = 1,089913504µs
=> f=1/1,089913504µs = 0,917504 MHz => fquatrz =0,917504 * 4 = 3,670016 MHz
Le mieux est d'utiliser 1 quartz à 4MHz pour le PIC et un à 3,670016 MHz pour PB0 qui peut être utilisér le PIC et un à 3,670016 MHz pour PB0 qui peut être utilisé