Université Mohammed Seddik BENYAHIA – JIJEL

A.N : 2020 / 2021

Faculté des Sciences et de la Technologie

Département d'Electronique

TP ESE23/27 *Master* II

SYSTEMES EMBARQUÉS

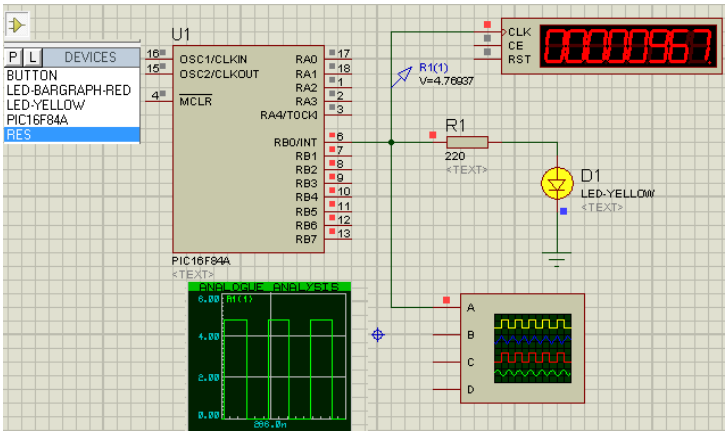
OBJECTIFS :

1-Maîtriser la programmation en assembleur pour PIC ,la simulation et le debugging
2- Apprendre à gérer les temporisations et à les calculer en utilisant les timers ou des boucles simples ou imbriqées.
3-Bien assimiler le principe de gestion d'une interruption sur TMR0.

EXERCICE .I :Calcul du temps d'exécution

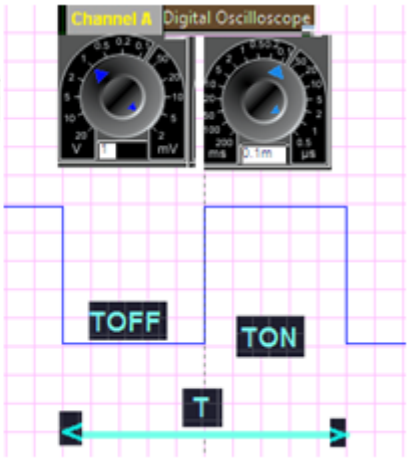
```
        list p=16F84A
        #include<p16f84A.inc>
      __CONFIG _CP_OFF&_PWRTE_ON&_WDT_OFF&_XT_OSC

        ORG 0
        GOTO START
        ORG 04
        RETFIE
START:  BSF 3,5    ;*************************************************
        CLRF TRISB
        BCF 3,5    ;*************************************************
ETQ:
        CLRF PORTB
        CALL ms
        COMF PORTB,1
        CALL TEMPO
        ETQ1 GOTO ETQ
TEMPO   MOVLW 255
        MOVWF 0x21
TT:     NOP
        NOP
       NOP
        DECFSZ 0x21,1
        GOTO TT
        RETURN
        END
```



Travail à faire :

1-Mesurer Ton et Toff utilisant l'oscilloscope.
2-Calculer le temps de la temporisation à la main.
3-vérifier que Ton=Toff=temps de la temporisation TEMPO.
4-Expliquer en qqs mots que fait le programme.



5-Remplacer la routine de temporisation par le code suivant :

```
TEMPO   MOVLW 255
        MOVWF 0x21
        MOVWF 0x22
TT:     NOP
        DECFSZ 0x21,1
        GOTO TT
        DECFSZ 0x22,1
        GOTO TT
        RETURN
```

Mesurer Ton et Toff du signal carré généré sur la broche RB0 et les comparer avec le temps de la TEMPO que vous calculez à la main.

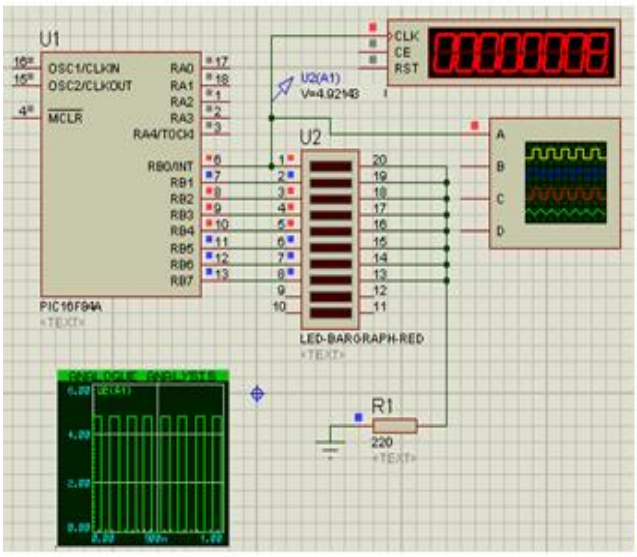
6-Quelle est la valeur qu'il faut charger dans W pour avoir un signal carré de fréquence 10HZ.

7-Montrer en calculant à la main que le temps de la TEMPO suivante est de 1s approximativement.

```
TEMPO  movlw 0xFF
       movwf  0x21   ; Fill the register.
       movlw 0xC
       movwf  0x20   ; Load 12 into count.
loop2  decfsz 0x20,1
       goto loop1
       goto exit
loop1  decfsz 0x21,1
       goto loop1
       goto loop2
exit   return
```

EXERCICE .II : Utilisation du Timer0

Saisir le code **assembleur** suivant sur **bloc-notes** puis utiliser **Mpasmwin.exe** pour obtenir le fichier **.hex** pour simuler sous **Proteus** le montage suivant :




```
        list p=16F84A
        #include<p16f84A.inc>
        __CONFIG _CP_OFF&_PWRTE_ON&_WDT_OFF&_XT_OSC

        ORG 0
        GOTO START
        ORG 04
        GOTO ISR
START   BSF 3,5;********** Config**********
        CLRF TRISB
        CLRF OPTION_REG
        BSF OPTION_REG,0
        BSF OPTION_REG,1
        BSF OPTION_REG,2
        ;BCF OPTION_REG,3
        ;BCF OPTION_REG,5
        BCF 3,5;***************Retour bank0 *******
        MOVLW B'10100000'
        MOVWF INTCON;
        CLRF TMR0
HERE:   GOTO HERE     ;*******Ne fait rien,juste attendre***

ISR:    INCF PORTB,1  ;***** Routine asservissement intr *
        BCF INTCON,T0IF
        RETFIE
        END
```

1-Commenter ligne par ligne et Expliquer que fait le programme.

2-Mesurer Ton et Toff du signal carré sur RB0 puis vérifier par calcul .

3-Refaire le même travail avec le code suivant.

```
        list p=16F84A
        #include<p16f84A.inc>
        __CONFIG _CP_OFF&_PWRTE_ON&_WDT_OFF&_XT_OSC

        ORG 0
        GOTO START
        ORG 04
        GOTO ISR
START:  BSF 3,5 ;********** Config***********
        CLRF TRISB
        CLRF OPTION_REG
        BCF OPTION_REG,0
        BSF OPTION_REG,1
        BCF OPTION_REG,2
        BCF 3,5 ;**************Retour bank0 *******

        MOVLW B'10100000'
        MOVWF INTCON;

        CLRF TMR0

HERE:   MOVF 0X20,0
        MOVWF PORTB
        GOTO HERE     ;*******Ne fait rien,juste attendre***

ISR:    INCF 0X20,1   ;***** Routine asservissement intr ***

        BCF INTCON,T0IF

        RETFIE
        END
```

EXERCICE .II : Jeu de lumière utilisant RRF et RLF

Saisir le code **assembleur** suivant sur **bloc-notes** puis utiliser **Mpasmwin.exe** pour obtenir le fichier **.hex** pour simuler sous **Proteus** le même montage vu précédemment :

```
       list p=16F84A
       #include<p16f84A.inc>
       __CONFIG _CP_OFF&_PWRTE_ON&_WDT_OFF&_XT_OS

           ORG 0
           GOTO START
           ORG 04
           RETFIE
START:     BSF 3,5 ;********** Config***********
           CLRF TRISB
           CLRF OPTION_REG

           BCF 3,5 ;**************Retour bank0 ***

           BSF STATUS,C
           CLRF PORTB
           MOVLW 8
           MOVWF 0x22
LOOPR:
           RRF PORTB,1
           CALL TEMP
           DECFSZ 0x22
           GOTO LOOPR
           MOVLW 8
           MOVWF 0x22
LOOPL:     RLF PORTB,1
           CALL TEMP
           DECFSZ 0x22
           GOTO LOOPL
TEMP:

           DECFSZ 0x20,1
           GOTO TEMP
           DECFSZ 0x21,1
           GOTO TEMP
           RETURN
           END
```

1-Commenter ligne par ligne et Expliquer que fait le programme.

2-Mesurer Ton et Toff du signal carré sur RB0 puis vérifier par calcul .

3- Visualiser les signaux sur les broches du port B utilisant l'oscilloscope ou analyseur graphique analgique.

4- Modifier le code pour réaliser un jeu de lumière avec plus d'option (vitesse, séquences,...etc).

Rappel

Timer 0 is operated in 8-bit mode. The time for an interrupt is given by:

$$\text{Time} = (4 \times \text{clock period}) \times \text{Prescaler} \times (256 - \text{TMR0})$$

where Prescaler is the selected prescaler value, and TMR0 is the value loaded into timer register TMR0 to generate timer interrupts every Time period.

In our application the clock frequency is 4MHz, that is, clock period = 0.25µs, and Time = 5ms. Selecting a prescaler value of 32, the number to be loaded into TMR0 can be calculated as follows:

$$\text{TMR0} = 256 - \frac{\text{Time}}{4 * \text{clockperiod} * \text{prescaler}}$$

Comparing values

Rappel

To determine whether a given number is less than, greater than or equal to another number, it is necessary to subtract them and test the zero and carry bits generated by the ALU. If the result is negative, then the carry bit is zero. If the carry bit is 1, then the result is either positive or zero. If the result is zero, then the zero bit is 1. The carry and zero bits are bits 0 and 2 of the `STATUS` register respectively. The following code can be used:

```
movlw 0x3       ; Assume W has some value in it.
sublw 0x5       ; (k-W) into W.
btfss STATUS,0  ; Test the carry bit.
goto LT         ; carry=0, so k<W.
btfsc STATUS,2  ; carry=1, so k>W or k=W, so test Z.
goto equal      ; k=W.
goto GT         ; k>W.
```