

2022

# Projet PIC



Abderrachid Bellaali

Ikram Arous

Marina Quadu

Patson Tiako

Groupe A

17/05/2022

# 1. Introduction

Un microcontrôleur PIC est une unité de traitement de l'information à laquelle on a ajouté des périphériques internes de communication avec l'extérieur permettant de réaliser des montages sans nécessiter l'ajout de composants externes ou du moins avec un nombre restreint de composants. Il peut fonctionner de manière autonome après programmation.

## 2. Objectifs

L'application est un télémètre à ultrasons qui envoie au PIC la mesure d'une distance en cm.

- La programmation du PIC 18F458 doit se faire via l'interface RS232 avec un convertisseur vu qu'il n'intègre pas de contrôleur USB.
- L'affichage de la distance se fait sur 2 afficheurs 7-segments à anodes communes.
- Si la distance est sur 2 chiffres, les 2 chiffres apparaissent. Si c'est sur 3 chiffres, la distance est convertie en m avec la partie entière séparée de la partie fractionnaire par le point de l'afficheur.
- Une LED rouge clignotante signale l'alerte en cas de dépassement de la distance définie par l'utilisateur ou dans le cas contraire, d'atteinte du seuil critique également défini par l'utilisateur.
- Une LED verte doit rester en permanence allumée quand aucune alerte n'est en cours.
- Le PIC sera programmé en C.
- Le PIC doit communiquer avec une application Python qui affichera la distance mesurée et qui permettra de définir le seuil de déclenchement d'alerte.

### **3.Éléments techniques**

#### **Circuit imprimé**

Pour le PCB, nous utilisons le logiciel de conception de circuits imprimés Eagle. Nous avons réalisé le circuit sur base d'une simulation Proteus avec les composants suivants :

- Un microcontrôleur PIC 18F458
- Une sonde à ultrasons HC-SR04
- Un cristal
- Un bouton poussoir
- Une interface RS232 avec convertisseur USB
- Une LED rouge
- Une LED verte
- Deux décodeurs 74LS47
- Deux afficheurs 7-segments à anodes communes

#### **Code C**

Le code C va permettre de programmer le PIC à l'aide d'un compilateur. Le compilateur CCS que nous avons utilisé a donc généré un fichier .hex sur base du fichier .c dans lequel sont consignées toutes les instructions de programmation nécessaires au fonctionnement du PIC 18F458.

#### **Code Python**

Le code Python sert d'interface entre le microcontrôleur PIC programmé et l'autre côté du canal de transmission c'est-à-dire l'utilisateur et son ordinateur. L'interface nous permet de visualiser la distance entre la sonde et l'objet détecté mais également si alerte il y a en cas de dépassement du seuil de déclenchement. Elle doit également permettre de changer le seuil auquel l'alarme va se déclencher.

## 4. Les tests effectués, leurs résultats et les solutions apportées

### Sur le PCB

- Estimation du temps de réalisation : plus ou moins 8h
- Temps réel de réalisation : 15h

Nous avons testé les composants, les pins et les connexions.

Tout d'abord, le logiciel tinybld198 ne reconnaissait pas le PIC, il a fallu vérifier toutes les soudures et reprendre celles qui ne semblaient pas très nettes. Après quelques efforts et sueurs froides, il l'a enfin reconnu !

Ensuite, les afficheurs 7-segments ne s'allumaient pas. En effet, au moment de l'assemblage, nous avons travaillé avec des décodeurs HCF4511 et des afficheurs 7-segments à cathodes communes. Mais la simulation a été réalisée avec des décodeurs 74LS47 et leurs afficheurs 7-segments à anodes communes correspondants.

Après réalisation de notre distraction, nous avons effectué les changements adéquats et les afficheurs ont, bien évidemment, fonctionné convenablement.

### Le code C

- Estimation : plus ou moins 20h
- Temps réel de réalisation : 28h

Le programme a été réalisé au tout début du projet mais pas finalisé à ce moment-là. Nous avons alors une première ébauche que nous avons fait évoluer au fur et à mesure et nous l'avons finalisée quand nous avons pu assembler et tester notre PCB.

On a un peu chipoté sur le point de l'afficheur 7-segment qui restait affiché à cause de l'output qui devait envoyer 0V puisque les afficheurs étaient à anodes communes.

### Le code Python

- Estimation du temps de réalisation : plus ou moins 7h
- Temps réel de réalisation : 11h

Le code Python a été développé en parallèle du code C. De même que le code C, il a évolué tout au long du projet et a été finalisé quand nous avons assemblé et testé le PCB.

Le plus difficile a été de récupérer la distance dans la chaîne de caractères produites par le code C et de la convertir en entiers.

## **5. Par rapport au cahier des charges :**

- ✓ La programmation du PIC 18F458 se fait via l'interface RS232 avec un convertisseur vu qu'il n'intègre pas de contrôleur USB.
- ✓ L'affichage de la distance se fait sur 2 afficheurs 7-segments à anodes communes.
- ✓ Si la distance est sur 2 chiffres, les 2 chiffres apparaissent. Si c'est sur 3 chiffres, la distance est convertie en mètres avec la partie entière séparée de la partie fractionnaire par le point de l'afficheur.
- ✓ Une LED rouge clignotante signale l'alerte en cas d'atteinte ou de dépassement du seuil critique.
- ✓ Une LED verte reste en permanence allumée quand aucune alerte n'est en cours.
- ✓ Le PIC communique avec une application Python qui affiche la distance mesurée et qui permet à l'utilisateur de définir le seuil de déclenchement d'alerte.

## **6. Répartition du travail**

Bien que nous ayons pratiquement tout le temps travaillé ensemble, certains d'entre nous, en fonction des affinités, se sont plus penchés sur certains aspects du projet en particulier.

### **Simulation proteus :**

Ikram et Rachid ont peaufiné la simulation Proteus en coopération avec Marina et Patson

### PCB

Marina s'est chargée du PCB et de la finalisation au niveau des soudures en collaboration avec Ikram, Rachid et Patson.

### Code C

Rachid et Patson ont principalement travaillé sur le code C en collaboration avec Ikram et Marina.

### Code Python

Ikram, Patson et Rachid ont travaillé sur le code Python en collaboration avec Marina.

### Rédaction des rapports

Marina s'est chargé de la rédaction des rapports en collaboration avec Ikram, Rachid et Patson.

## **7. Les limites du système et les améliorations**

On pourrait optimiser le PCB en travaillant avec un seul décodeur pour les 2 afficheurs 7-segments et avec des transistors en commutation.

On pourrait améliorer l'interface du code Python afin de le rendre plus agréable visuellement mais pour l'instant, il est fonctionnel, c'est tout ce qu'on lui demande.

On pourrait intégrer un buzzer pour nous avertir de l'approche du seuil critique.

## **8. Conclusions personnelles :**

### **Ikram :**

Ce projet s'est révélé très enrichissant pour moi, dans la mesure où il a consisté en une approche concrète de la conception et de la fabrication d'un circuit électronique. En effet, grâce à celui-ci, j'ai pu mettre en pratique mes connaissances vu lors des cours et renforcer mes compétences grâce aux autres membres du groupe et aux diverses difficultés rencontrées qui ont nécessité recherches et entraide.

### **Rachid :**

Pour ma part, je trouve le projet très intéressant puisqu'il met en pratique plusieurs aspects de la formation. Notamment la programmation en C et Python du PIC ainsi que la soudure de celui-ci. Il serait bien plus intéressant d'utiliser une puce plus récente que celle-ci, mais l'idée est la même.

### **Marina :**

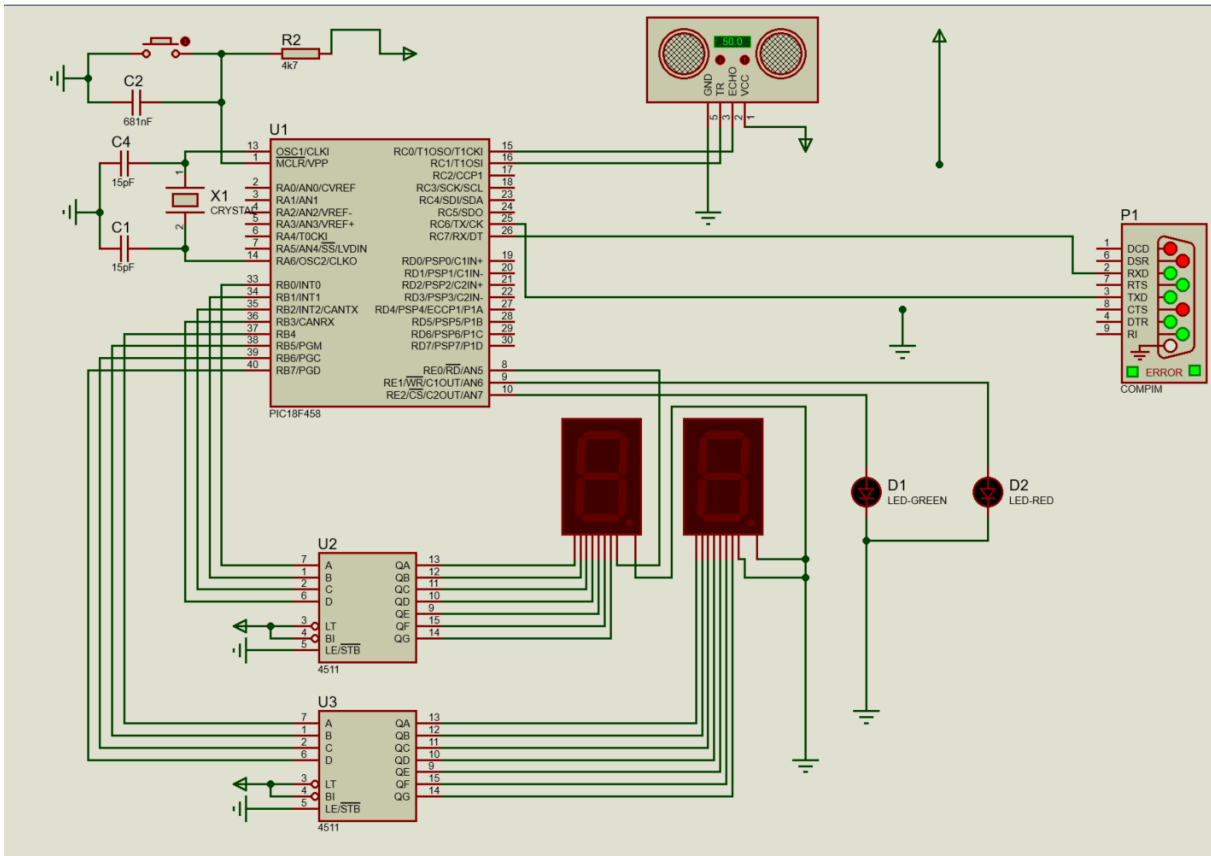
En participant à ce projet, j'ai réalisé que la communication, bien que nous ayons toujours travaillé ensemble à peu de choses près, nous a manqué. Il n'est pas évident de partager ce que l'on a fait de manière claire et exhaustive malgré les moyens que nous avons à disposition (Github, Trello...). Le projet en lui-même était très intéressant et j'en retire que je dois mieux documenter mes actions afin de me faciliter la vie et le travail de mes collègues, en plus bien évidemment d'avoir pu mettre en pratique nos connaissances acquises depuis 2 ans.

### **Patson :**

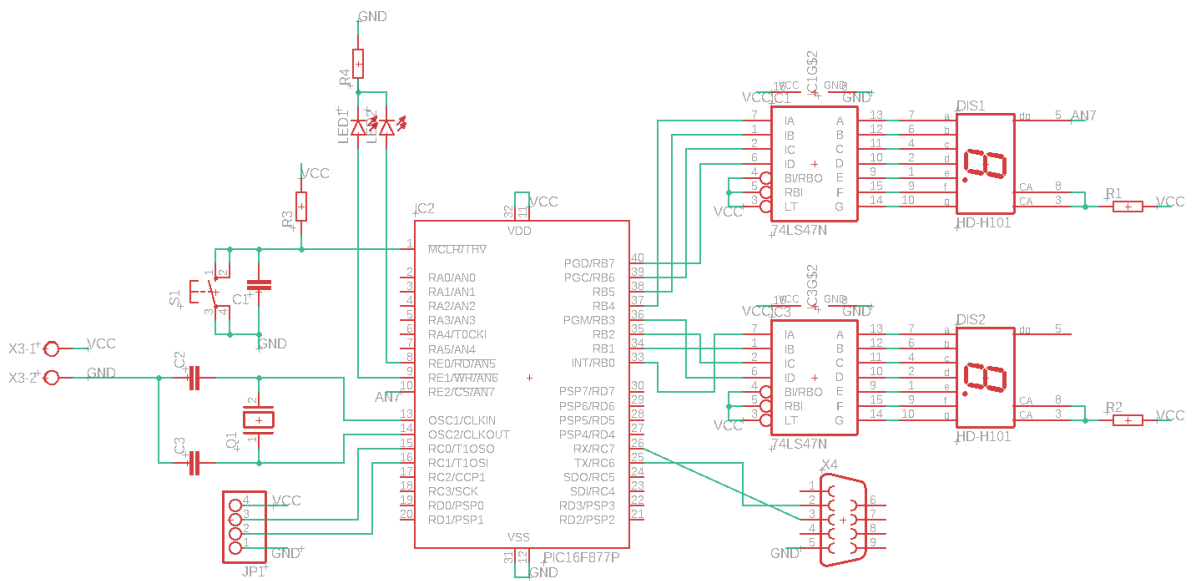
Ce projet était un bon moyen d'allier électronique et programmation. Au cours de ce projet, j'ai appris de façon ludique le fonctionnement d'un appareil électronique avec du code de bas niveau ce qui est la base du fonctionnement de tout système embarqué. Il n'était pas toujours évident de consacrer plus de temps dans la réalisation du projet à cause des autres projets à réaliser mais dans l'ensemble, réaliser ce projet m'a donné un aperçu de ce qui se trouve dans le monde de l'électronique.

## ANNEXES

## I. Schéma Proteus

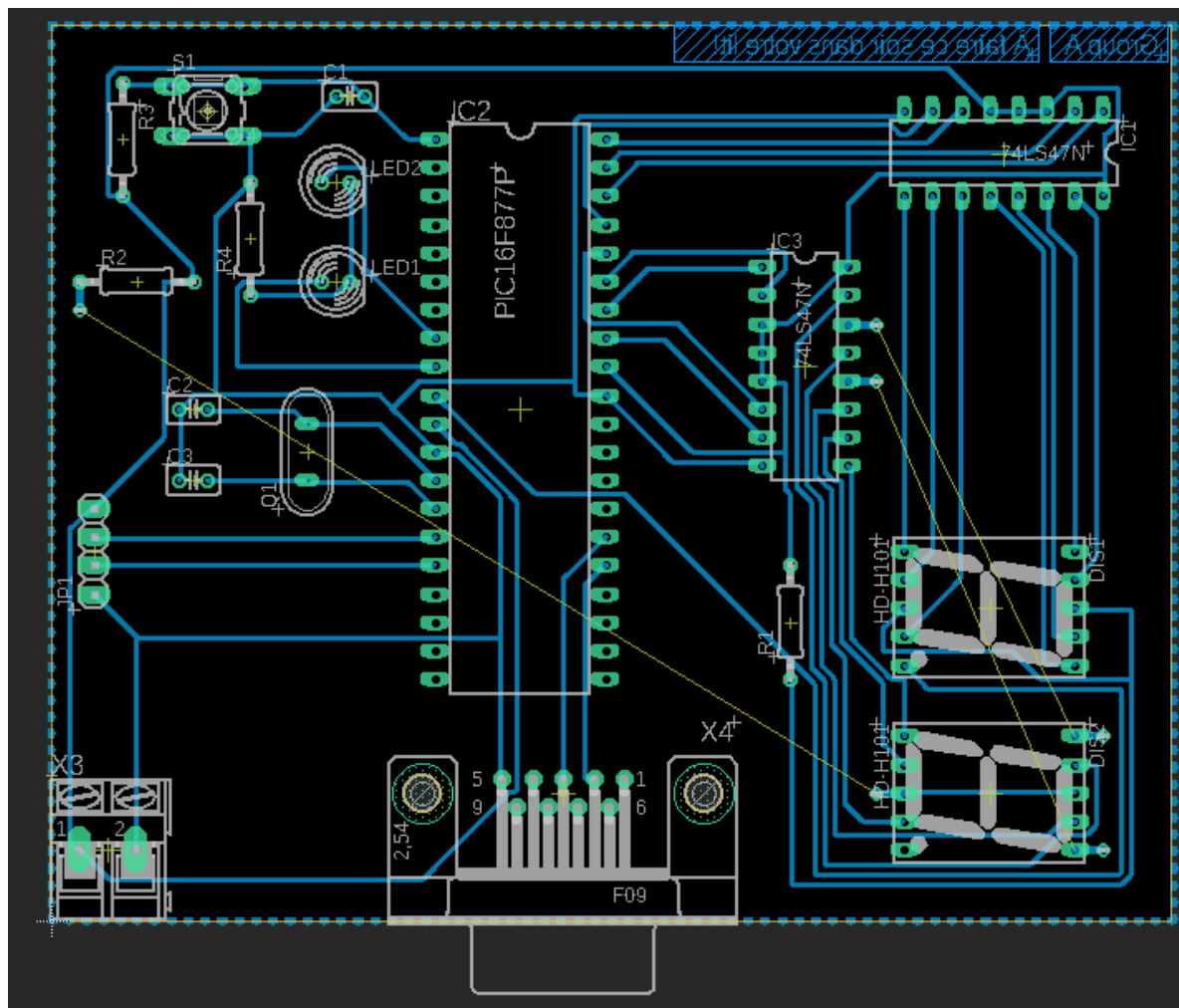


## II. Schéma Eagle





### III. Schéma PCB



#### IV. Le code C :



```
#include "C:\Userse\Documents\PIC\code_with_interrupt.h"

char buffer[4];
int i = 0;
int16 diz, unit;
boolean flag = 0;
int16 maxDist = 100;
int16 dist, time ;
int d, c, u;

#INT_TIMER1
void TIMER1_isr(void)
{
    set_timer1(0);
}
#int_RDA
void RDA_isr(void)
{
    buffer[i] = getc();
    if(buffer[0] == ':' && flag == 0 ){
        i++;
        if(i>=4){
            i = 0;
            flag = 1;
        }
    }
}

void main()
{
    setup_adc_ports(NO_ANALOGS);
    setup_adc(ADC_OFF);
    setup_psp(PSP_DISABLED);
    setup_spi(FALSE);
    setup_wdt(WDT_OFF);
    setup_timer_0(RTCC_INTERNAL);
    setup_timer_1(T1_INTERNAL | T1_DIV_BY_8);
    setup_timer_2(T2_DISABLED, 0, 1);
    setup_comparator(NC_NC_NC_NC);
    setup_vref(FALSE);
    enable_interrupts(INT_RDA);
    enable_interrupts(GLOBAL);
```

```

setup_oscillator(False);

while(true){

    //read value of sensor
    output_high(pin_c1);
    delay_us(20);
    output_low(pin_c1);

    while(!input(pin_c0)){ } //attendre l'etat haut de la pin echo

    set_timer1(0);

    while(input(pin_c0)){ } //attendre l'etat haut de la pin echo
    time = get_timer1();
    dist = time*0.028;
    printf("distance : %ld \n" , dist);
    printf("\n");
    delay_ms(100);

    if(dist<maxDist){
        output_high(PIN_E0);
        output_low(PIN_E1);
    }else{
        output_low(PIN_E0);
        output_toggle(PIN_E1);
    }

    if(dist<100){
        output_high(pin_e2);
        unit = dist%10;
        diz = ((dist - unit)/10)%100;
        output_b((diz<<4)+unit);
    }else{
        output_low(PIN_E2);
        diz = dist/100;
        unit = (dist - (diz*100))/10;
        output_b((diz<<4)+unit);
    }

    if(flag == 1){
        flag =0;
        c = buffer[1]-48;
        d = buffer[2]-48;
        u = buffer[3]-48;
    }
}

```

```

        maxDist = (int16)(c*100+d*10+u);
    }
    delay_ms(300);
}
}

```

## V. Le code Python :



```

import serial
import time
from tkinter import ttk
from ttkthemes import ThemedTk

serialPort = serial.Serial('COM3', baudrate=9600, timeout=1)

print("connecte au port serie: " + serialPort.portstr)

# Configuration du seuil critique
def seuilCritique():
    resultat = int(spinbox1.get())
    serialPort.write(bytes(":" + "%03d" % resultat, 'UTF-8'))
    time.sleep(3)
    return resultat

# Lecture des mesures reçues du PIC à travers le port serie
def readFromSerial():
    data = serialPort.readline().decode("ascii")

    if data != "":
        distance = str(data)
        valeur_distance = 0
        if "distance" in distance:
            tab_distance = [int(s) for s in data.split()
if is.isdigit()]
            valeur_distance = int(tab_distance[0])

        seuil = seuilCritique()
        print("seuil", seuil)
        if valeur_distance < int(seuil):
            status = "OK"
            texte = f"{distance} La distance mesurée est de
{valeur_distance} cm".format()
            label2.config(text=texte)

```

```

        label3.config(text=status)
    elif valeur_distance >= int(seuil):
        status = "ALARM"
        texte = f"{distance} cm Attention le seuil critique est
atteint!".format(
            valeur_distance)
        label2.config(text=texte)
        label3.config(text=status)

    window.after(50, readFromSerial)

window = ThemedTk(themebg=True)
window.set_theme("equilux")
window.title("Projet Programmation PIC")

message1 = ttk.Label(window, text="Veuillez sélectionner un seuil
critique:")
message1.grid(column=0, row=0, padx=10, pady=10)

spinbox1 = ttk.Spinbox(window, from_=0, to=10, width=10,
state='normal')
spinbox1.set(100)
spinbox1.grid(column=1, row=0, padx=10, pady=10)

button1 = ttk.Button(window, text="Valider", command=seuilCritique)
button1.grid(column=0, row=1, padx=10, pady=10)

label2 = ttk.Label(window, text="", width=40)
label2.grid(column=1, row=1, padx=10, pady=10)
window.update()

label3 = ttk.Label(window, text="", width=40)
label3.grid(column=1, row=2, padx=10, pady=10)
window.update()

if __name__ == '__main__':
    window.after(50, readFromSerial)
    window.mainloop()

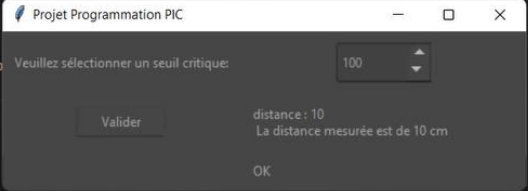
```

## VI. Interface du code Python

```
distance = str(data)
valeur_distance = 0
if "distance" in distance:
    tab_distance = [int(s) for s in data.split() if s.isdigit()]
    valeur_distance = int(tab_distance[0])

seuil = seuilCritique()
print("seuil", seuil)
if valeur_distance < int(seuil):
    status = "OK"
    texte = f"{distance} La distance mesurée est de {valeur_distance} cm".format()
    label2.config(text=texte)
    label3.config(text=status)
elif valeur_distance >= int(seuil):
    status = "ALARM"
    texte = f"{distance} : {valeur_distance} cm Attention le seuil critique est atteint!"
    label2.config(text=texte)
    label3.config(text=status)

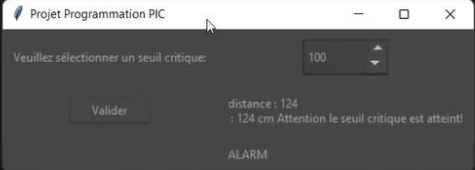
window.after(50, readFromSerial)
```



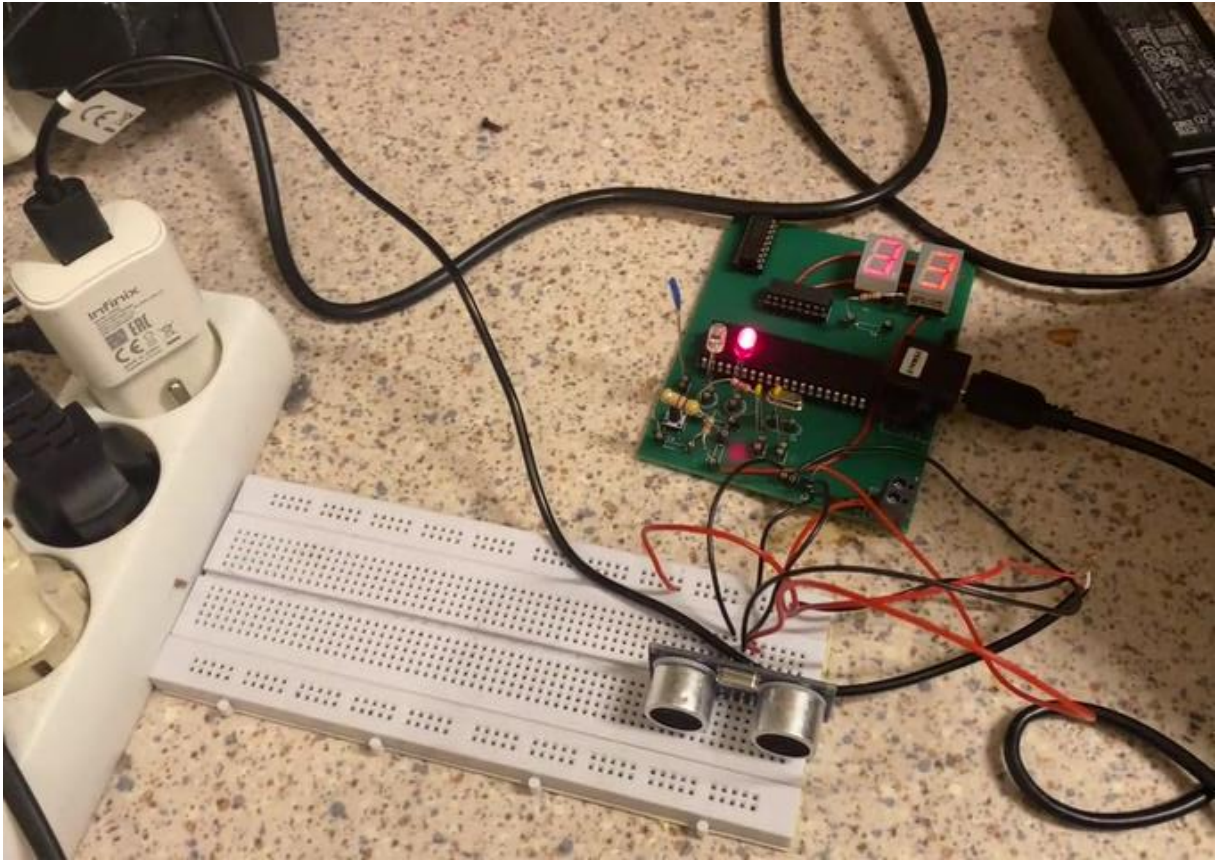
```
# Configuration du seuil critique
def seuilCritique():
    resultat = int(spinbox1.get())
    serialPort.write(bytes(":" + "%03d" % resultat, 'UTF-8'))
    time.sleep(3)
    return resultat

# Lecture des mesures reçues du PIC à travers le port serie
def readFromSerial():
    data = serialPort.readline().decode("ascii")

    if data != "":
        # if "distance" in data:
        #     tab_distance = [int(s) for s in data.split() if s.isdigit()]
        #     distance = int(tab_distance[0])
        # else :
        #     distance = int(data)
        distance = str(data)
        valeur_distance = 0
        if "distance" in distance:
            tab_distance = [int(s) for s in data.split() if s.isdigit()]
            valeur_distance = int(tab_distance[0])
```



## VII. Bricolage...



Quand on n'a pas de générateur 5V, on a des idées... 😊