

Rapport de Projet

Electronique Digitale et Analyse de Signaux

Groupe 3

A. Nilens

G.lemer

F. Degives

J-M. Tang

Mai 2020



Haute Ecole Economique et Technique

Table des matières

1	Introduction	1
2	Répartition des tâches	1
3	Estimation du temps des tâches	2
4	Choix d'implémentation du circuit électronique	4
5	Choix de developpement CCS	4
6	Choix de developpement Python	5
7	Problèmes rencontrés	6
8	État final du projet	7
9	Conclusion	7
10	Circuit Proteus	8
11	Circuit Eagle	9
12	PCB Eagle	10
13	PCB + Plan de masse	11
14	Code CCS	12
15	Code Python	17

1 Introduction

Les consignes de ce projet étaient de réaliser un circuit électronique sur un PCB contenant une puce PIC, récupérant une entrée et définissant l'état de plusieurs sorties digitales.

Pour ce faire, nous utiliserons un télémètre à ultrasons, qui envoie à la PIC la mesure d'une distance en cm. Le circuit affichera la distance reçue grâce à deux afficheurs 7 segments.

Afin de faire un lien avec notre cours de développement informatique avancé du premier quadrimestre, il nous est demandé de réaliser une application JAVA fonctionnelle communiquant avec notre puce PIC et recevant les informations de télémétrie pour les afficher et déclencher ou non l'alerte si la mesure atteint un seuil fixé.

Au vu des problèmes rencontrés durant le projet et de nos préférences de langages de programmation, nous avons finalement remplacé le programme en JAVA par un programme en Python. Nous vous expliquerons plus loin dans ce rapport nos différents choix d'implémentation et leurs raisons.

Pour ceci, nous utiliserons plusieurs composants électroniques :

- PIC : 18F458
- Contrôleur RS232/USB : type FTDI DB9-USB-D5-F
- Sonde à ultrasons : HC-SR04
- Afficheurs 7-SEG MPX1-CC
- Condensateurs : EU025 ou X50
- Cristal : X TAL18
- Résistances : R-EU_0204/7
- Bouton poussoir : Switch OMRON 10XX
- Décodeur : 4511
- Led : Led rouge et led verte

2 Répartition des tâches

Lemer Guillaume : Proteus / CCS / Java / Python

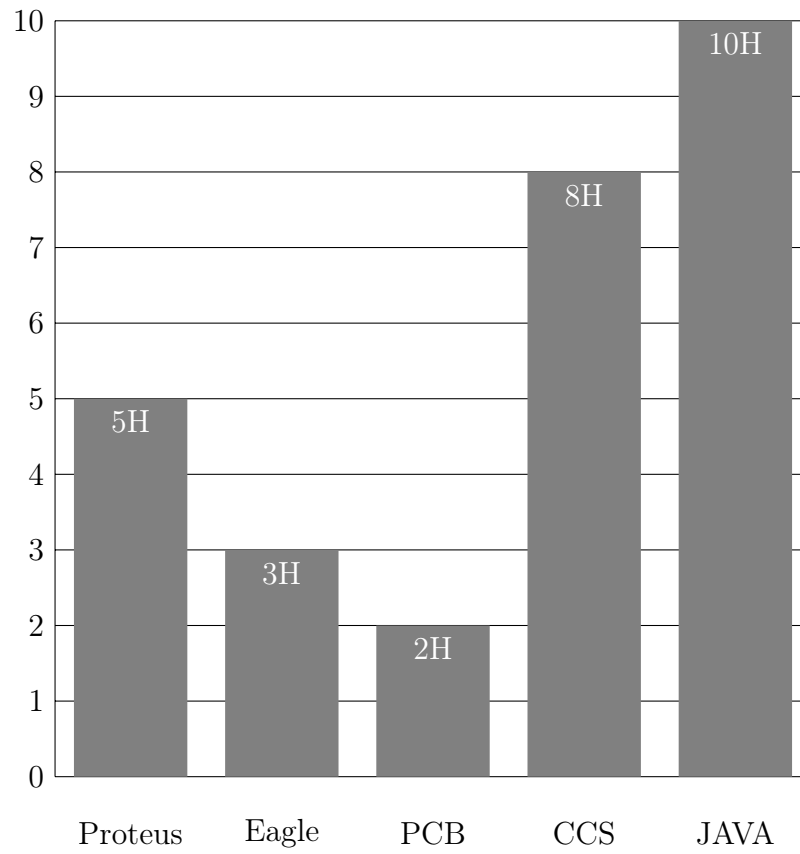
Nilens Arnaud : Proteus / Eagle / CCS / Rapport

Degives Florian :

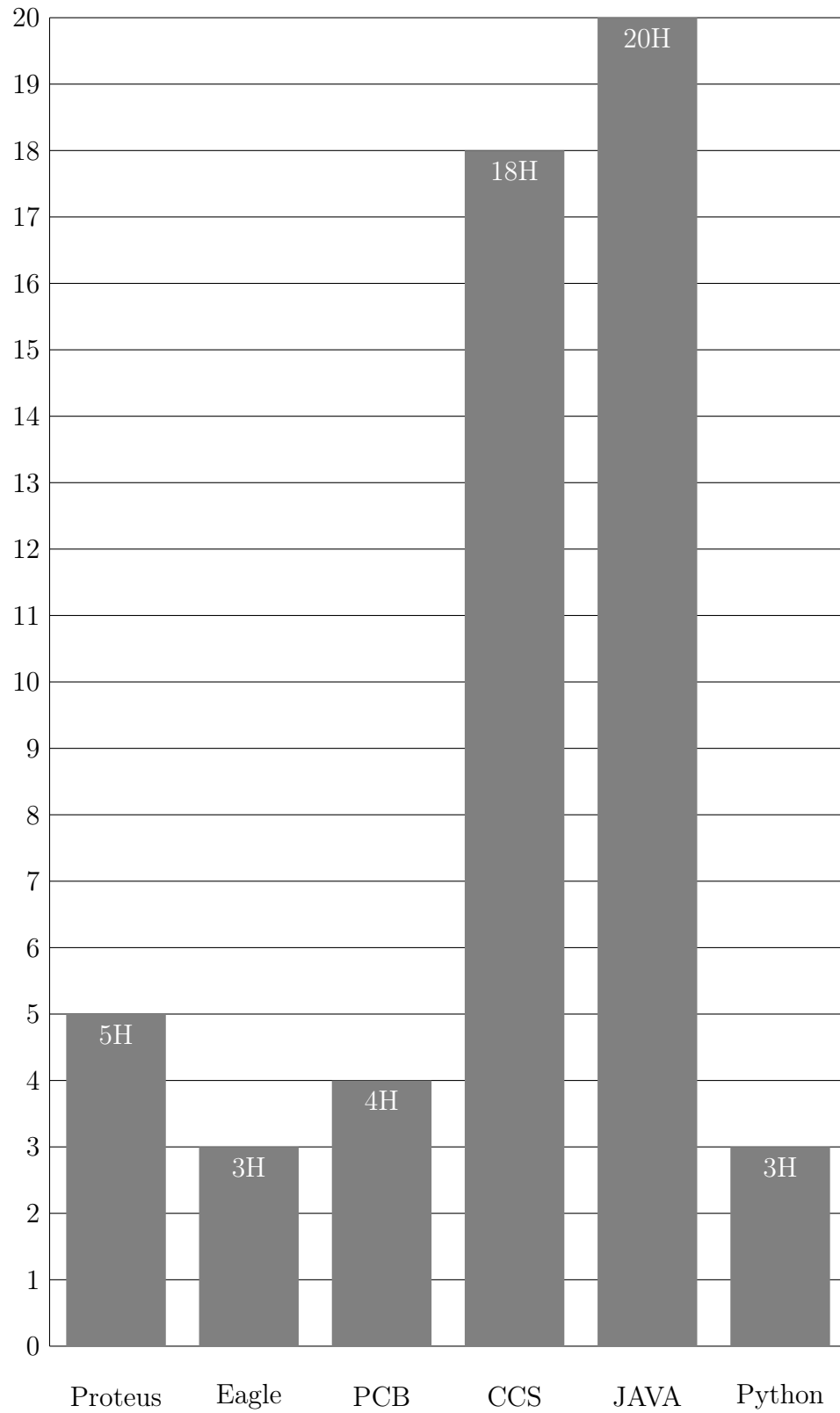
Tang Jean-Michaël :

3 Estimation du temps des tâches

Estimation :



Temps réellement accordé aux tâches :



4 Choix d'implémentation du circuit électronique

Au niveau du circuit, les schémas étant assez similaires dans tous les groupes, nous allons vous expliquer les quelques éléments qui diffèrent probablement et pourquoi nous avons pris ces choix. Premièrement, nous avons pris la décision de mettre 2 décodeurs 7 segments qui sont chacun dédiés exclusivement à un afficheur. Si nous nous sommes dirigés vers cela c'est pour un besoin de réactivité et pour avoir une gestion plus facile de l'affichage étant donné que les deux afficheurs sont gérés indépendamment l'un de l'autre.

Pour finir, nous avons intégré un afficheur LCD à la simulation (celui-ci n'est pas présent sur le schéma du PCB Eagle) pour permettre l'affichage des données contenues dans la puce PIC. Cette implémentation nous permet de faire une comparaison des données du code python et celles de la puce. De plus, cela nous a fortement aidé dans les phases de debugging du code CCS.

5 Choix de developpement CCS

Notre code CCS¹ fonctionne comme suit :

1. Nous commençons par définir les paramètres de liaison au port COM1, nos variables, plusieurs fonctions et le fonctionnement d'interrupt RDA permettant de récupérer les données du port COM1.
2. Dans la première partie de la fonction mais nous initialisons la puce PIC ainsi que notre écran LCD.
3. Ensuite, nous rentrons dans une boucle infinie qui est la partie principale du programme. C'est dans celle-ci que nous gérons l'affichage sur l'écran LCD et les afficheurs sept segments, le calcul, la gestion des led, les comparaisons de variables ainsi que leur envoi au programme Python.

Voici les différentes fonctions que nous avons définies :

1. RDA_isr : Fonction de récupération des données sur le port COM1. Elle charge caractère par caractère, dans un buffer, le tableau qu'elle reçoit si celui-ci commence par '!'.
2. outputValueParser : Fonction qui transforme la valeur qu'on lui envoie en base 16 sous forme de nombre en base 10 pour permettre l'affichage sur les afficheurs 7 segments.
3. triggerSonde : Fonction de trigger de la sonde.
4. init_lcd : Fonction d'affichage au démarrage de l'écran LCD qui affiche les informations du projet.

1. Vous pourrez retrouver le code du programme en fin de rapport.

6 Choix de developpement Python

Après de nombreux test avec un code JAVA et une interface associée, le tout sans succès, nous nous sommes replié sur un code simple et efficace que peut nous fournir Python.

Python et la librairie PySerial permettent de gérer les interaction entre notre code et la PIC. Nous avons intégrer au tout une interface grace a Tkinter, certes un peu vieux, mais efficace.

En résumé, si nous avons choisi de développer notre programme en Python c'est avant tout dans un soucis de performances, simplicité et efficacité car Python est moins gourmand en ressources et moins compliqué à comprendre. De plus, c'est un langage qui nous intéressait énormément et que nous désirions apprendre et utiliser dans un projet depuis longtemps. Nos motivations étaient donc nombreuses.

Concernant l'interface du programme, nous avons 2 grandes parties distinctes :

- Une première servant a l'entrée utilisateur de la distance minimale, accompagnée d'un bouton servant a la transmettre ainsi qu'un champ contenant la valeur de celle-ci (par défaut de 100cm).
- Et une deuxième partie étant l'affichage, un string 'ok' si la distance est bien inférieure a celle programmée dans la pic ou 'Alerte', inversement, ainsi qu'un autre string affichant la distance en temps réel reçue depuis la PIC en CM.

Pour le code en lui-même, nous utilisons 2 grandes fonctions :

- readValue() : Consiste en la lecture de la valeur reçue sur le port serial, cette valeur pouvant être :
 1. Si la distance est supérieure à la distance minimale.
 2. Si la distance est supérieure à la distance minimale.
 3. Un nombre X, si la PIC nous transfert la valeur de cette dite distance.

Cette même fonction s'occupe de gérer l'affichage Tkinter des données reçue pour plus de réactivité de l'affichage. De plus la fonction de lecture va "vider" le port serial de réception afin d'éviter un surplus de valeur étant donné que la PIC envoie les données calculées en continu.

- sendMinValue() : Consiste en la fonction de gestion de l'envoi des données à la PIC par le port serial. La seule subtilité de cette fonction est la gestion du caractère d'échappement gérer en coopération avec le code C et étant '!'. Ce caractère est ajouté au début des données envoyé afin de signaler à la PIC la réception de la trame de données.

Le corps de notre code Python gère la réception des données par une boucle infinie accompagnée d'un "try catch", si des données sont disponibles a la lecture, celles-ci sont lues par la 'fonction readValue()' et l'affichage est ensuite mis a jour afin que l'utilisateur puisse voir les données. Nous

prenons bien soin d'ouvrir le port serial (COM2) au début de notre connexion, afin d'éviter une erreur de connexion le port est d'abord fermé avant d'être ouvert. En ce qui concerne la fermeture du port serial, celui-ci est fermé dans le 'finally' de notre 'try catch' ce qui permet d'assurer la fermeture de celui-ci quoi qu'il arrive.

7 Problèmes rencontrés

Lors de la première partie de projet nous avons rencontré les problèmes suivants :

- Problèmes sur Eagle lors de la création de la board.
- Prises en main des fonctions spécifiques à la puce PIC sur CCS.
- Erreurs de branchements sur Proteus et Eagle.

Concernant la deuxième partie du projet se focalisant sur le développement des 2 parties de code nous avons eu de nombreux problèmes : Pour commencer, en JAVA :

- Nous avons commencé par utiliser Jssc mais les membres de notre groupe n'y sont pas arrivés correctement au début.
- Nous nous sommes heurtés au problème (maintenant connu de tout le cours) de JRE, qui empêche la lecture par RXTX des ports série due à une violation d'accès mémoire de JAVA JRE 14.

Les problèmes rencontrés avec java nous a poussé à recommencer plusieurs fois le code, pour enfin décider de changer de langage et de passer à une implémentation Apprenez à prononcer de l'interface graphique sur le PC en Python.

Ensuite, pour le code C :

- Nous avons d'abord eu du mal à trouver de la documentation sur l'implémentation d'un code C par CCS sur une PIC et ce problème ne fut jamais réglé, ce qui a rendu nos recherches laborieuses pour trouver de la documentation et des solutions à nos autres problèmes (même mineurs) rencontrés.
- Les Timers ont bloqué une bonne partie du début de notre développement, car les informations sur ceux-ci en CCS sont presque indisponibles sur internet.
- Le passage du temps à distance de la sonde semble un peu bancal car cela ne répond pas la formule classique de conversion (342,8 m/s) mais la valeur renvoyée semble correcte à une approximation près.
- La lecture sur le port Série a également posé problème au niveau de la gestion de l'événement de réception des données.

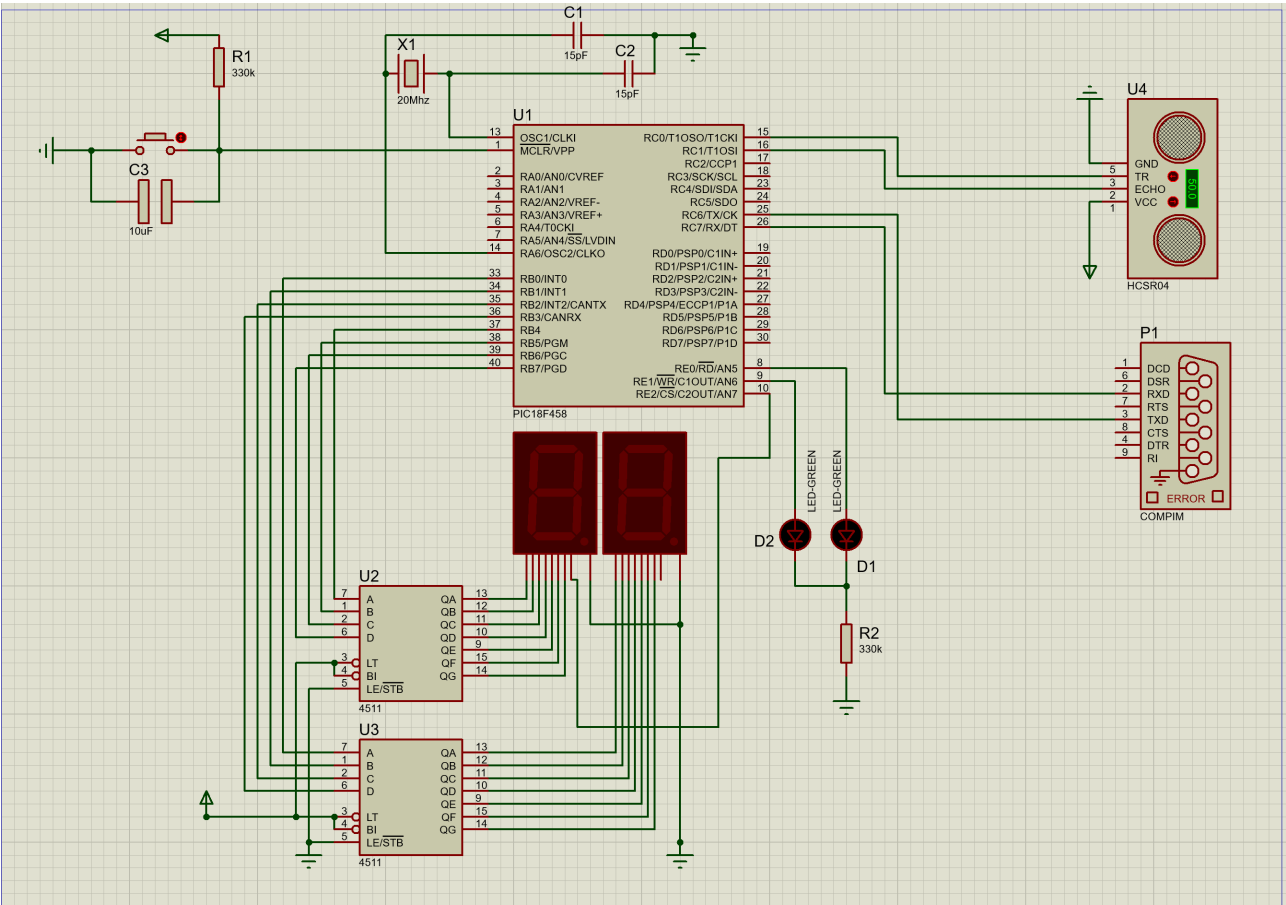
8 État final du projet

Malgré les différents problèmes rencontrés et le temps pris à les régler, nous sommes finalement arrivés à proposer un projet fonctionnel, simple d'utilisation et clair.

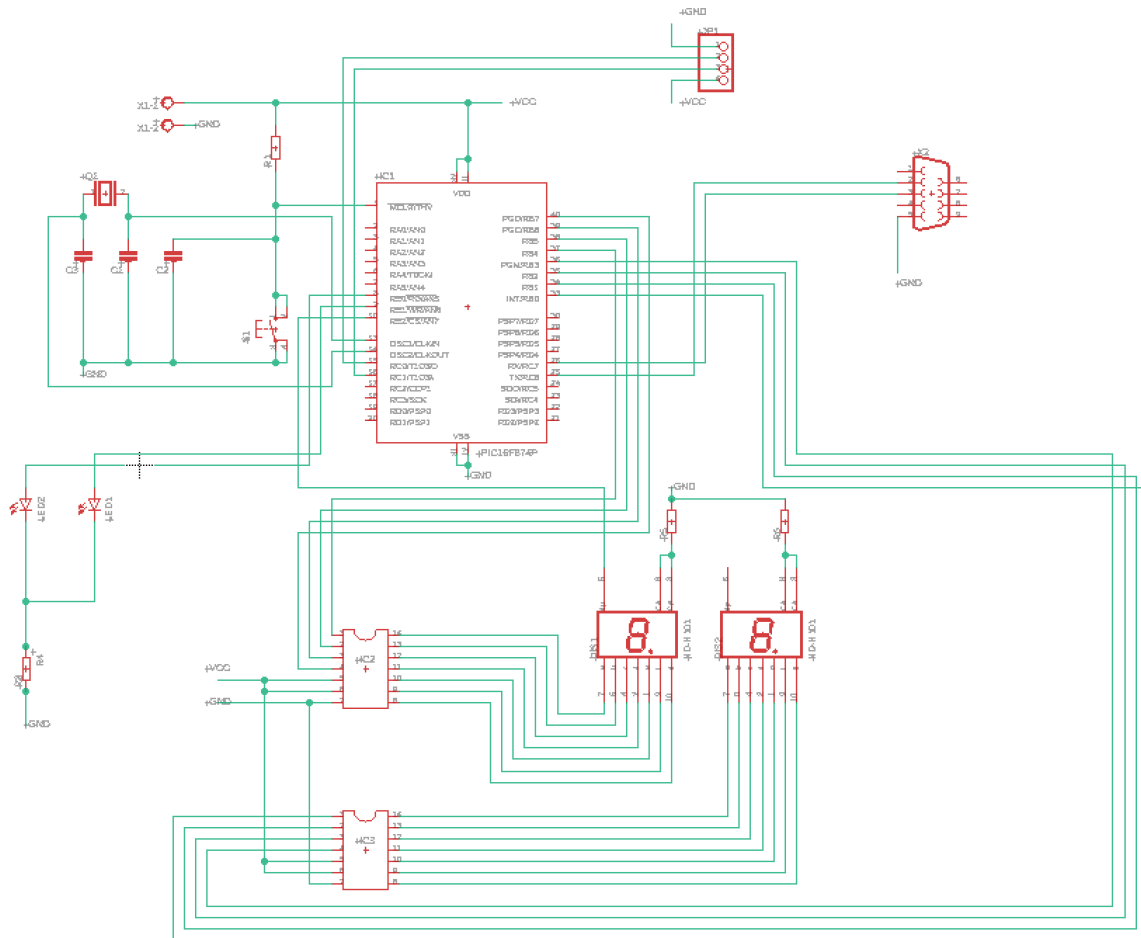
9 Conclusion

Ce projet, que nous avons d'abord abordé avec beaucoup d'enthousiasme s'est vite transformé en une corvée dû aux nombreux problèmes que nous avons rencontré, à la difficulté que nous avons eu à les résoudre ainsi qu'au manque d'informations concernant ce langage sur internet (CCS très peu documenté). La quantité de temps que ce projet nous a pris a bien dépassé nos prévisions. Nous n'étions pas préparé à une telle quantité de travail vis-a-vis des autres cours et projets que nous avions en parallèle. De plus le climat global de confinement n'a pas aidé à la mise en place d'une dynamique de groupe efficace ainsi qu'à la collaboration de tout le monde sur ce projet.

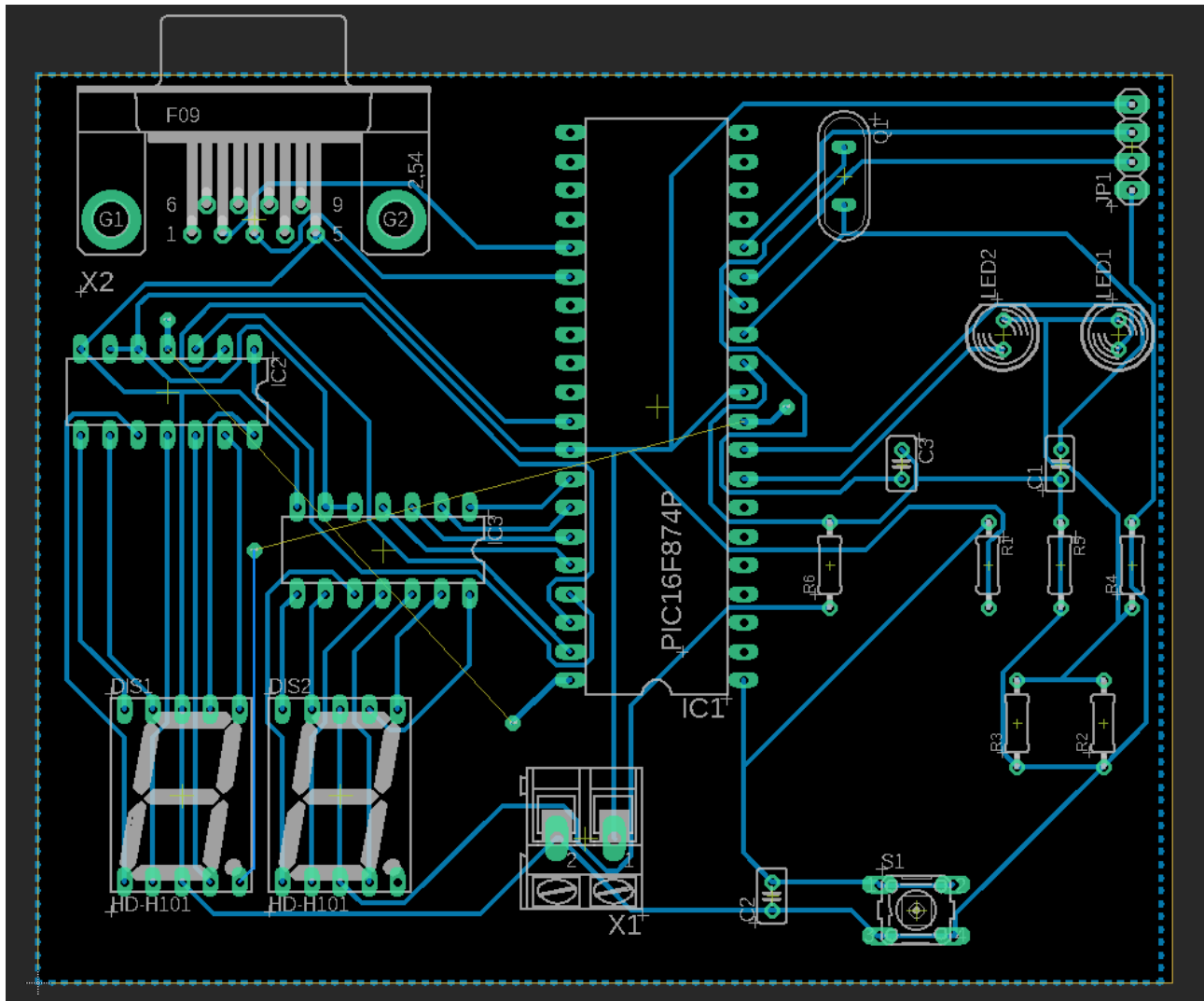
10 Circuit Proteus



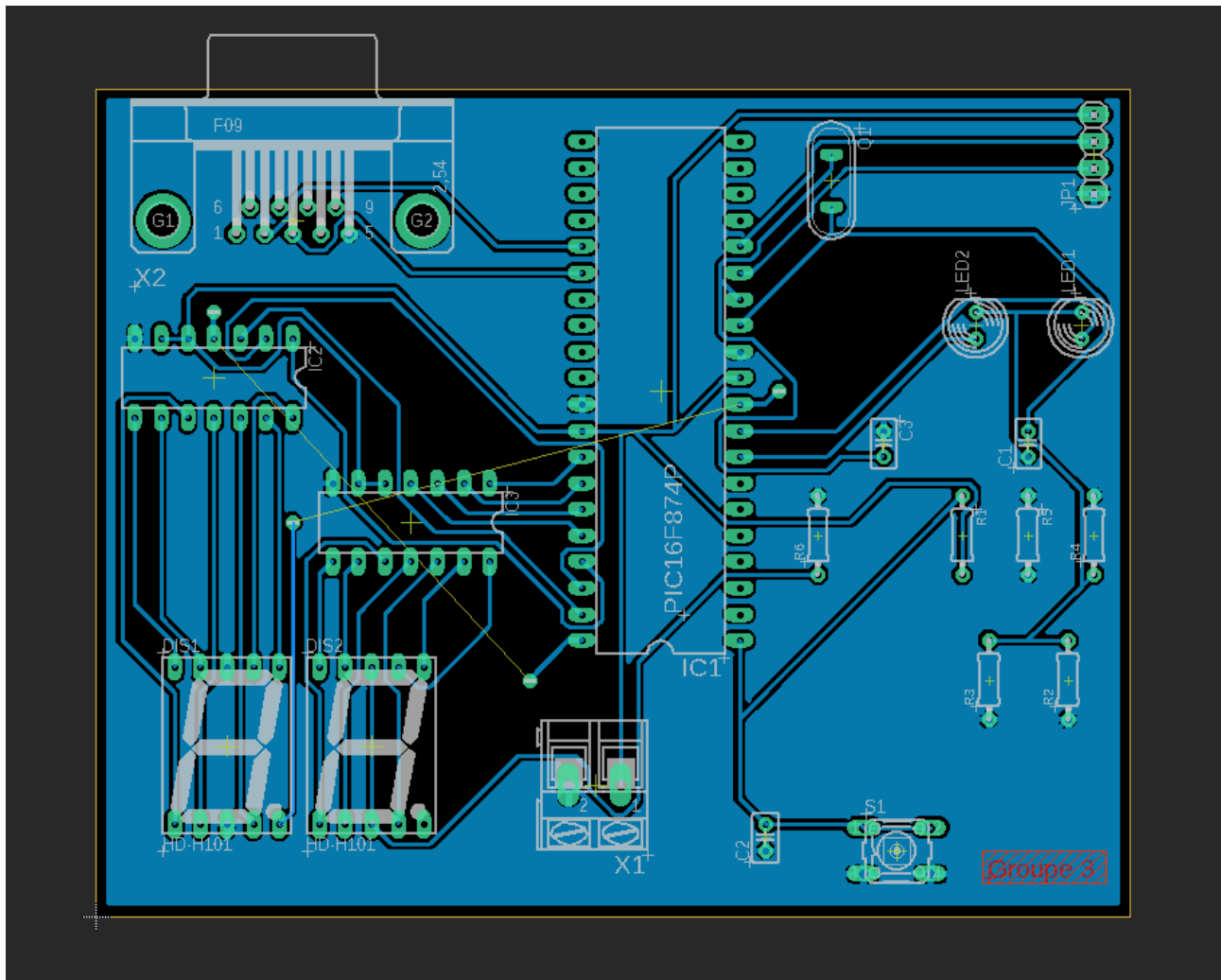
11 Circuit Eagle



12 PCB Eagle



13 PCB + Plan de masse



14 Code CCS

```
1 #include <main.h>
2 #include "LCD420.c"
3
4 #use rs232(baud=9600, parity=N, xmit=PIN_C6, rcv=PIN_C7, bits = 8, ERRORS)
5
6 #define trigger pin_C0
7 #define echo pin_C1
8 #define dot pin_E2
9 #define RX pin_C7
10 #define TX pin_C6
11 #define GREEN pin_E0
12 #define RED pin_E1
13
14 #int_TIMER1
15
16 int16 time, distance, x, i, minVal;
17 boolean flag=0;
18 char buffer[4];
19 int8 j=0;
20 int8 c,d,u;
21
22 #INT_RDA
23 void RDA_isr(void) {
24     buffer[j]=getc();
25     if(buffer[0]=='!' && flag==0) {
26         j++;
27         if(j>=4) {
28             j=0;
29             flag=1;
30         }
31     }
32 }
33
34
35 /*
36  * transofmation de la valeur en valeur base 16
37  */
38 int16 outputValueParser(int16 value){
39     int output = 0;
40     if (value < 10) { output = value;}
41     else {
42         x = value;
43         x = x % 10;
```

```

44     i = value;
45     i = i/10;
46
47     output = x + i*16;
48 }
49 return output;
50 }
51
52 /*
53 * trigger de la sonde
54 */
55 void triggerSonde(){
56     output_high(trigger);
57     delay_us(10);
58     output_low(trigger);
59 }
60
61 /*
62 * Fonctione d'initalisation du LCD
63 */
64 void init_lcd(){
65     delay_ms(500);
66     lcd_putc('\f');
67     lcd_gotoxy(1,1);
68     printf(lcd_putc, "_Projet_Electronique_");
69     lcd_gotoxy(1,3);
70     printf(lcd_putc, "_2020_Groupe_3_");
71     delay_ms(1000);
72     lcd_putc('\f');
73 }
74
75
76 /*
77 * fonction principale
78 */
79 void main()
80 {
81     setup_low_volt_detect(FALSE);
82
83     setup_spi(FALSE);
84     setup_wdt(WDT_OFF);
85     setup_timer_1( T1_INTERNAL | T1_DIV_BY_1 );
86     setup_comparator(NC_NC_NC_NC);
87     setup_vref(FALSE);
88     setup_oscillator(False);

```

```

89
90  enable_interrupts(INT_RDA); // interruption sur r ception port RS232
91  enable_interrupts(GLOBAL);
92
93  lcd_init();
94  init_lcd();
95
96  minVal = 100;
97
98  while(true)
99  {
100
101  // setup des valeurs
102  time = 0;
103
104  // recuperation minValue envoye par JAVA
105  //minVal = 100;
106
107  // d clanchement de la sonde
108  triggerSonde();
109
110  // recuperation valeur temps de la sonde
111  while(input(echo) == 0){} // attente debut
112  set_timer1(0);
113  while(input(echo) == 1){} // attente fin
114  time = get_timer1();
115
116  // temps => distance
117  distance = time/285;
118
119  // envoie distance ici java
120  printf("%ld", distance);
121  printf("\n");
122
123  // si donnees recu depuis java sur RS232 > interruption INT_RSA
124  if(flag==1){
125      c=buffer[1]-48;
126      d=buffer[2]-48;
127      u=buffer[3]-48;
128      minVal=(int16) (c*100+d*10+u);
129      flag=0;
130  }
131
132
133  // creation valeurs LCD MINvalue

```



```

134  c = minVal/100;
135  d = (minVal-(c*100))/10;
136  u = (minVal-(c*100))-(d*10);
137  // Affichage MinValue LCD
138  lcd_gotoxy(3,1);
139  printf(lcd_putc, "_MinVal:_");
140  printf(lcd_putc, "%d", c);
141  printf(lcd_putc, "%d", d);
142  printf(lcd_putc, "%d", u);
143
144  // c r e a t i o n   v a l e u r s   D i s t a n c e   L C D
145  c = distance/100;
146  d = (distance-(c*100))/10;
147  u = (distance-(c*100))-(d*10);
148  // affichage distance LCD
149  lcd_gotoxy(3,3);
150  printf(lcd_putc, "_Distance:_");
151  printf(lcd_putc, "%d", c);
152  printf(lcd_putc, "%d", d);
153  printf(lcd_putc, "%d", u);
154
155  //   v   r i f i c a t i o n       b o r n e   m i n V a l
156  if (distance < minVal){
157      // allumer red => trop proche
158      printf("1\n");
159      output_high(RED);
160      output_low(GREEN);
161  }
162  else {
163      // allumer green => OK
164      printf("2\n");
165      output_high(GREEN);
166      output_low(RED);
167  }
168
169  // gestion du point si > que 100 !> cm -> m
170  if(distance > 99){
171      distance = distance / 10;
172      output_high(dot);
173  }
174  else {
175      output_low(dot);
176  }
177
178  // affichage sur 7seg de la distance

```

```
179     output_b(outputValueParser(distance));
180
181
182     // attente pour eviter spam
183     delay_ms(500);
184 }
185
186 }
```

15 Code Python

```
1 import time
2 import serial
3 import tkinter as tk
4
5 serPort = 'COM2'
6 baudRate = 9600
7
8 serial = serial.Serial(serPort, baudRate)
9
10 '''
11 Lecture_des_valeur_du_port_serial
12 Print_de_la_valeur_recue_dans_le_GUI
13 '''
14 def readValue() :
15     line = (serial.readline()).decode() # read a '\n' terminated line
16     line = line[:-1]
17     if (line == '1') :
18         print('OK')
19         inputVal_text.set('OK')
20         intputVal.config(fg='green')
21     elif (line == '2') :
22         print('Alerte')
23         inputVal_text.set('Alerte')
24         intputVal.config(fg = 'red')
25     else :
26         print('distance_de:_'+ line + '_cm')
27         outputVal_text.set('distance_de:_'+ line + '_cm')
28     serial.flush()
29
30 '''
31 Envois_de_la_valeur_minimal_a_la_sonde_depuis_l'entr e dans le GUI
32 '''
33 def_sendMinValue():
34     min_value=inputMinVal.get()
35     if_(int(min_value)<10):
36         min_value=str(0)+min_value
37     if_(int(min_value)<100):
38         min_value=str(0)+min_value
39     serial.write(('!'+min_value+'\n').encode())#_envois_du_string_sous_forme_!<Value>\n
40     print('Valeur '+min_value+' envoy e comme valeur minimale')
41
42 '''
43 Lancement du programme et du GUI
```

```

44 '''
45 window=tk.Tk()
46 window.title("Projet_Elec")
47 # d finition des espaces d'affichages
48 paramFrame = tk.Frame(window)
49 outputFrame = tk.Frame(window)
50 # d finition du label d'alarme
51 labelMinVal = tk.Label(paramFrame, text='Valeur_d\'alarme:')
52 labelMinVal.pack()
53 # d finition du champ d'insertion des donnees de distance_minimal
54 inputMinVal = tk.Entry(paramFrame, textvariable=int, width=6)
55 inputMinVal.insert(0, 100)
56 inputMinVal.pack()
57 # d finition du bouton d'envoi de la donnee_minimal
58 sendBtn = tk.Button(paramFrame, text='send', command=sendMinValue)
59 sendBtn.pack()
60 # d finition de l'affchage de la valeur_minimale
61 inputVal_text = tk.StringVar()
62 inputVal = tk.Label(outputFrame, textvariable=inputVal_text)
63 inputVal.pack()
64 # d finition du champ de reception de valeur
65 outputVal_text = tk.StringVar()
66 outputVal = tk.Label(outputFrame, textvariable=outputVal_text)
67 outputVal.pack()
68 # mise en place des blocs d'affichage
69 paramFrame.pack(padx=1, pady=1)
70 outputFrame.pack(padx=20, pady=20)
71 # mise a jours de la window
72 window.update()
73
74 try:
75     print("Ouverture_du_port_COM2")
76     serial.close()
77     serial.open()
78     print("Port_serial_COM2_ouvert_\n")
79
80     time.sleep(2)
81     try:
82         #print("Appuyez_sur_une_touche_pour_terminer")
83         while 1:
84             if (serial.inWaiting() > 0):
85                 readValue()
86                 window.update()
87     except KeyboardInterrupt:
88         pass

```

```
89 finally:  
90     serial.close()
```