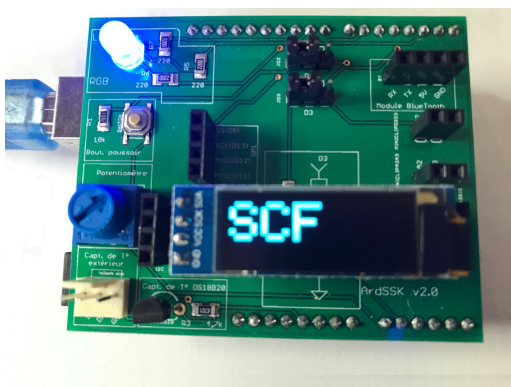


Des micro-contrôleurs dans l'enseignement de la chimie...

Atelier formation



Activité 1

Communiquer !

1 Installation préalable

Il s'agit, avant tout, d'installer l'environnement de développement (IDE) d'Arduino.

Connectez-vous sur le site arduino.cc puis cliquez sur l'onglet DOWNLOADS du menu SOFTWARE. Sur la page arduino.cc/en/Main/Software, on vous propose soit d'utiliser un éditeur web, soit d'installer l'**Arduino IDE**. Choisissez votre système d'exploitation et procédez à l'installation de l'environnement de développement.

Exécutez une première fois le programme Arduino. Ouvrir la page de préférences de l'application (Fichier > Préférences sous Windows ; Arduino > Préférences sur Mac) et notez l'emplacement du carnet de croquis sur la première zone de saisie. En général, il s'agit du dossier Arduino créé dans le dossier Documents. Ouvrir ce dossier puis :

- copier le dossier SCF (après l'avoir décompacté) dans le dossier Arduino ;
- copier le dossier Dallas-master (après l'avoir décompacté) dans le dossier libraries du dossier Arduino.

Fermez le programme de l'IDE d'Arduino. Ouvrez le à nouveau. Vous devez trouver :

- une entrée SCF dans le menu Fichier puis Carnet de Croquis ;
- une entrée DallasTemperature dans le menu Fichier puis Exemples.

C'est bon, vous être prêt pour mercredi !

2 En route vers notre premier programme. . .

La démarche de conception d'un programme est donnée ci dessous :

- on utilise le logiciel Arduino pour écrire son programme ;
- on télécharge (télé-verse) le programme dans la carte (il y a une étape de compilation intermédiaire qui vous signalera parfois quelques bugs !)
- le programme étant téléchargé, la carte fonctionne en autonomie si elle a sa propre source d'énergie, ou, bien sûr, si elle reste connectée via le port USB.

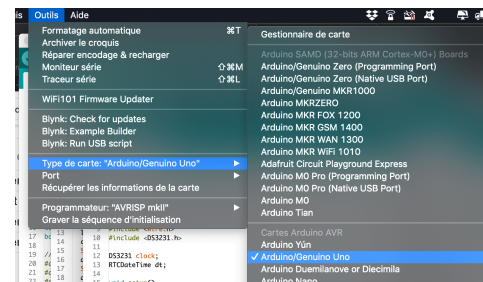
2.1 Éditez votre premier programme

a Premiers réglages

Connectez la carte Arduino à l'ordinateur via le câble USB et démarrez le logiciel Arduino (icône ci-contre). En fait, l'ordre importe peu. Une fois le logiciel démarré :



- allez dans le menu *Outils* puis *Type de carte* et vérifiez que *Arduino Uno* est bien coché (le cocher sinon) ;
- vérifiez que la carte est bien reconnue en allant dans *Outils, Port*. La carte est reconnue si un port est affiché, par exemple "Com 3" ou `/dev/cu.usbmodem14201`.



b Structure du programme

A l'ouverture du logiciel, on peut remarquer qu'un bout de code est déjà proposé :

Listing 1.1 – Programme vide

```
1 void setup() {
2     // put your setup code here, to run once:
3 }
4
5
6 void loop() {
7     // put your main code here, to run repeatedly:
8 }
9 }
```

On dispose, ici, de la structure de tout programme Arduino :

- une fonction (ou procédure) **setup** d'initialisation qui ne sera exécutée qu'une seule fois ;
- une fonction (ou procédure) **loop** qui, comme son nom l'indique également, sera exécutée dans une boucle infinie.

c Le fameux "Hello Word"

On désire afficher, sur l'écran de l'ordinateur, une seule fois "Hello Word".

Modifiez le programme afin d'obtenir le code suivant (sans oublier les ; et en respectant la casse).

Listing 1.2 – Hello world

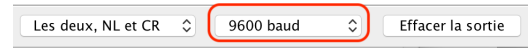
```
1 void setup() {
2     Serial.begin(9600) ;
3     Serial.println("Hello world !") ;
4 }
5
6 void loop() {
7     // put your main code here, to run repeatedly:
8 }
```

Une fois le code créé, cliquez sur l'icône de vérification du code (corrigez le code si nécessaire!).



Téléversez ensuite le programme dans la carte, puis sélectionnez *Outils/Moniteur série* ou cliquez sur l'icône loupe à droite de la barre d'icônes. Si tout se passe bien, le message apparaît (assurez vous qu'une vitesse de communication à 9600 bauds est bien sélectionnée dans le moniteur série.)

Vous pouvez tenter une variante en plaçant le code de la ligne 3 dans la boucle **loop** !



2.2 Un échange d'information bidirectionnel

Le moniteur série de l'Arduino permet non seulement d'afficher des messages reçus de l'arduino mais également d'en envoyer.

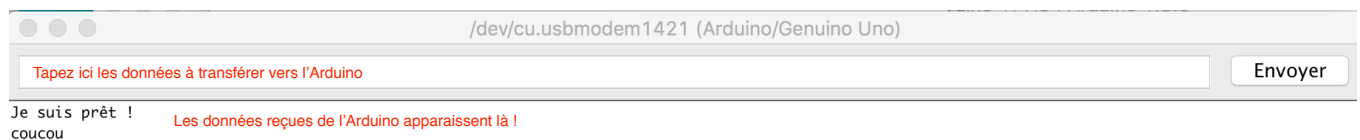
Codez le programme suivant (ou, ouvrez le fichier *Echo* via le lien *Fichier > Carnet de croquis*) > *SCF*).

Listing 1.3 – Echo

```

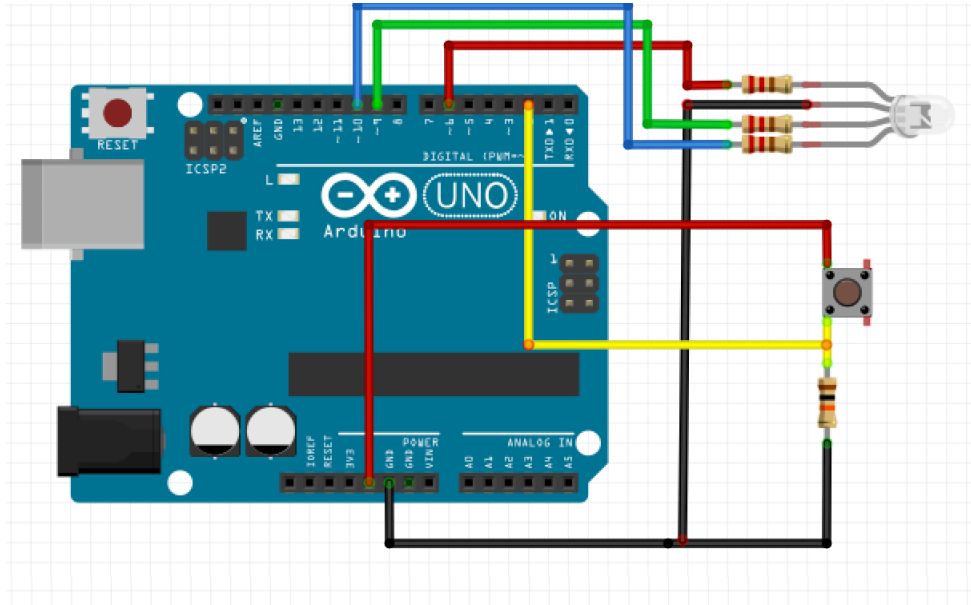
1 void setup() {
2   Serial.begin(9600) ;
3   Serial.println("Je suis prêt !") ;
4 }
5
6 void loop() {
7   if (Serial.available())
8   {
9     String message = Serial.readString() ;
10    Serial.println(message) ;
11  }
12 }
```

Dans la zone de texte en haut de l'éditeur série introduisez votre texte et cliquez sur *Envoyer* ; le message apparait alors dans la zone de réponse.



Activité 2

Commander !



Sur la platine du kit a été câblée une LED RGB (avec ses résistances de protection). Les broches rouge, bleue et verte sont respectivement connectées aux ports D6, D9 et D10 de l'Arduino.

1 Premières étapes

1.1 digitalWrite, analogWrite

a digitalWrite(port, LOW ou HIGH)

Listing 2.1 – digitalWrite

```
1 #define LED 6
2
3 void setup() {
4   pinMode(LED, OUTPUT) ;
5   digitalWrite(LED, HIGH) ;
6 }
7
8 void loop() {
9 }
```

b **analogWrite(port, 0 ≤ entier < 255)**

Ouvrir le fichier AnalogWrite.

Listing 2.2 – analogWrite

```

1  #define LED 6
2
3  void setup() {
4      pinMode(LED, OUTPUT) ;
5  }
6
7  void loop() {
8      for (int i = 0 ; i<= 255 ; i = i+1)
9      {
10         analogWrite(LED, i) ;
11         delay(20) ; // petite pause de 20 ms
12     }
13     for (int i = 255 ; i >= 0 ; i = i-1)
14     {
15         analogWrite(LED, i) ;
16         delay(20) ;
17     }
18 }

```

... avec un petit délai pour la persistance rétinienne !

1.2 **Je suis le maître du temps !****a** **delay(ms)... faute de mieux**

Exercice 1 Une led qui clignote ! Réaliser un programme qui fait clignoter la LED.

- soyons généreux pour commencer, vous avez droit à 2 instructions digitalWrite !
- un petit effort, le même avec une seule instruction digitalWrite ! (pour info, les variables HIGH et LOW sont des entiers qui valent 1 et 0 qui peuvent également être traités comme des booléens (variable de type bool)).
- pourquoi ne pas programmer un feu tricolore (orange peut être codé par RED = 255, GREEN = 165, BLUE = 0). Bon, pour voir le orange, il faut s'éloigner un peu de la LED.

b **Mesurons le temps qui passe...**

Au lieu de bloquer le système avec l'instruction `delay(ms)`, on pourrait très bien mesurer le temps écoulé depuis le dernier changement d'état de la lampe... et changer à nouveau l'état de la LED lorsque cette durée dépassera le délai souhaité.

Ouvrir le programme `MesureTempsEcoule`.

Listing 2.3 – Mesure temps écoulé

```

1  long delai = 1000 ; // en ms
2  long lastTime = 0 ;

```

```

3
4 void setup()
5 {
6     Serial.begin(9600) ;
7 }
8
9 void loop()
10 {
11     if (millis() - lastTime > delai)
12     {
13         lastTime = millis() ;
14         job() ;
15     }
16 }
17
18 void job()
19 {
20     // on fait ici ce qu'on a à faire !
21     Serial.println(lastTime) ; // pour l'exemple
22 }

```

Exercice 2 Compléter le programme précédent pour faire clignoter la LED.

c Gloire au timer !

En fait, il y a au cœur du microcontrôleur un circuit (plusieurs en réalité) qui permet de gérer des tâches à intervalle régulier : il s'agit du **Timer**. Son utilisation dépasse le cadre de cette introduction mais... vous y viendrez tôt ou tard !

2 Commander la LED

2.1 Version Hardware

a valeur = digitalRead(port)

Un bouton poussoir est câblé sur le shield. Pour l'utiliser, le cavalier **D2** doit être placé côté bouton poussoir.



L'état du bouton (appuyé ou relâché) peut être obtenu en lecture sur le port **D2**. Comme son état est binaire... vous avez deviné, bravo!, que c'est l'instruction **digitalRead** qu'il faut utiliser. Celle-ci va retourner 0 si le bouton est relâché et 255 s'il est appuyé.

Listing 2.4 – digitalRead

```

1 #define LED      6
2 #define BUTTON   2
3
4 void setup()
5 {
6     pinMode(LED, OUTPUT);

```

```

7   pinMode(BUTTON, INPUT) ;
8 }
9
10 void loop()
11 {
12     int button = digitalRead(BUTTON) ;
13     digitalWrite(LED, button) ;
14 }

```

b Bouton poussoir en mode marche/arrêt

A cause des phénomènes de rebond au niveau du bouton poussoir, cette tâche n'est pas si simple. Le programme `MarcheArret` permet de réaliser cette tâche.

c Un potentiomètre pour faire varier l'intensité

Cette approche sera abordée dans le paragraphe suivant. Il nous faudra, en effet effectuer une lecture analogique d'une tension sur un des ports de l'Arduino.

2.2 Version Software

Nous allons, cette fois, communiquer les ordres via le port série.

a Définition d'un protocole

Pour communiquer... il faut un langage commun !

Chacun est libre de proposer son propre protocole de communication. Nous allons, ici, utiliser par la suite les conventions suivantes (sans espace entre l'ordre et la valeur).

Ordre envoyé	Signification
G	GO : on exécute en continu
S	STOP : on arrête
O	ONE : on exécute une seule fois le travail souhaité
I	INFO : on récupère une information sur le programme
D : valeur	DELAJ : on fixe le délai entre 2 actions

b Côté moniteur série

C'est très simple, il suffit d'introduire l'ordre (G par exemple) dans la zone de saisie de texte et cliquer sur **Envoyer**.

Pour exécuter la tâche souhaitée toutes les 100 ms, par exemple, on enverra D:100.

c Côté Arduino !

Charger le programme `CommandeLED`

```

1  #define LED 6
2
3  int redValue = 255 ;
4
5  void setup()
6  {
7      Serial.begin(9600) ;
8      pinMode(LED, OUTPUT);
9  }
10
11 void loop()
12 {
13     if (Serial.available())
14     {
15         String message = Serial.readString() ;
16         parse(message) ;
17     }
18 }
19
20 void parse(String msg)
21 {
22     char ordre = msg[0];
23     msg.replace(String(ordre), "") ;
24     msg.replace(":", "") ;
25     int valeur = 0 ;
26     if (msg.length() > 0)
27         valeur = msg.toInt() ;
28     switch (ordre)
29     {
30         case 'G' :
31             go() ;
32             break ;
33         case 'S' :
34             stop() ;
35             break ;
36         case 'R' :
37             redValue = valeur ;
38             go() ;
39             break ;
40     }
41 }
42
43 void go()
44 {
45     analogWrite(LED, redValue) ;
46 }
47
48 void stop()
49 {
50     digitalWrite(LED, LOW);
51 }

```

spécifique au système à l'usage !

identiques pour tous !

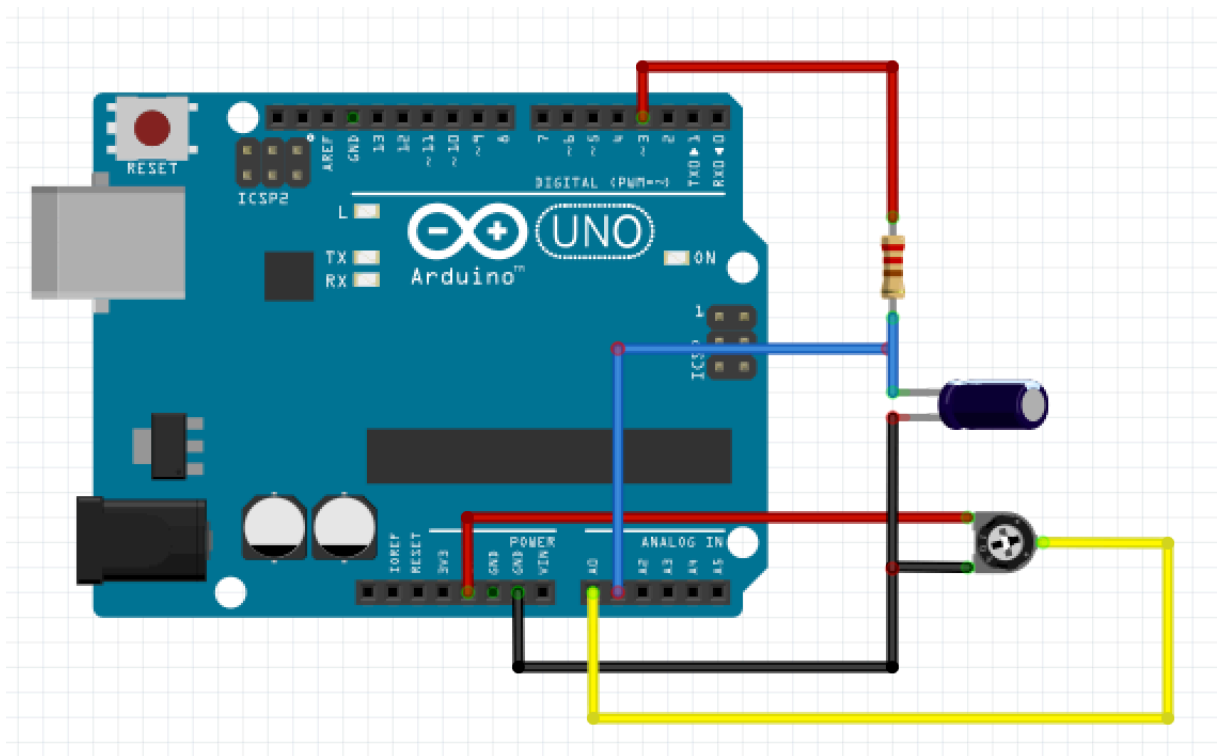
1 Lecture analogique

La platine Arduino Uno dont vous disposez comporte 6 ports dédiés à une lecture analogique (nommés A0, A1... () A5). Il s'agit de convertisseurs analogique-numérique (ADC) sur 10 bits. $2^{10} = 1023$. On peut lire une tension, en entrée, comprise entre 0 et 5 V, ce qui donne une résolution de 5 mV. La conversion demande 100 microsecondes.

Deux circuits sont câblés sur la carte et seront exploités dans ce chapitre :

- un diviseur de tension (potentiomètre rotatif de 10 k Ω) alimenté entre 0 et 5 V et dont le point milieu est connecté à l'entrée analogique **A0** ;
- un circuit RC alimenté par le port **D3** et dont le point milieu est connecté à l'entrée analogique **A1** R = 10 k Ω et C = 10 μ F.

Le schéma du circuit, utile dans ce chapitre est donné ci-dessous.



1.1 valeur = analogRead(port)... bien sûr !

Listing 3.1 – Lecture sur port analogique

```
1 #define analogPin A0
```

```

2
3 void setup()
4 {
5     Serial.begin(9600);
6 }
7
8 void loop()
9 {
10     int sensorValue = analogRead(analogPin);
11     Serial.print("sensor = ");
12     Serial.println(sensorValue);
13     delay(100);
14 }

```

Exercice 1 Éclairage variable. C'était l'objectif attendu au paragraphe c page 8. Moduler l'intensité de la LED à l'aide du potentiomètre rotatif. Attention, `analogRead` retourne un entier compris entre 0 et 1023 (10 bits) alors que `analogWrite` attend un entier compris entre 0 et 255 (8 bits). Faites la conversion, ou utilisez la fonction `valeur = map(entier,min1,max1,min2,max2)` qui convertit un entier compris entre *min1* et *max1* en un entier *valeur* compris entre *min2* et *max2*.

1.2 Sortie formatée

a Choisir un protocole pour le println !

Nous nous retrouvons quasiment face au même problème que celui rencontré paragraphe a page 8 : choisir un formatage des données. Chacun est libre de choisir le sien. Nous avons opté ici pour le format suivant :

DATA :id1 :valeur1 :id2 :valeur2...

- chaque information est séparée par `;` ;
- on commence par **DATA** ;
- id est une chaîne de caractère pour identifier la donnée : TIME, T0, U, I...
- valeur est... la valeur correspondante !

Le programme `ModeleADC` peut servir de coquille pour l'exploitation d'une lecture analogique.

Listing 3.2 – Modèle ADC

```

1 #define analogPin A0
2
3 void setup()
4 {
5     Serial.begin(9600);
6 }
7
8 void loop()
9 {
10     int sensorValue = analogRead(analogPin);
11     Serial.print("DATA:TIME:" + String(millis())) ;
12     Serial.println(":A0:" + String(sensorValue)) ;
13     delay(500);
14 }

```

b Copier dans Excel

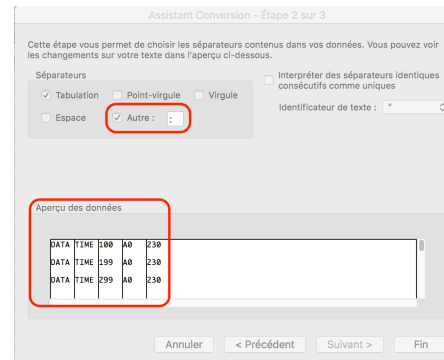
Faire une acquisition sur quelques secondes en tournant le bouton du potentiomètre rotatif.

Faire un copier coller de l'intégralité des données (ou de celles qui nous intéressent) du moniteur série d'Arduino vers la première case d'une feuille Excel.

Les données sont maintenant dans la première colonne, éventuellement séparées d'une ligne blanche.

Sélectionnez la première colonne puis, dans le menu **Données** d'Excel, cliquez sur l'item **Convertir...**

Une boîte de dialogue s'ouvre, passez à la seconde page et préciser que le séparateur est : (ou autre si vous avez choisi un autre format de sortie).



Les données apparaissent alors dans chacune des premières colonnes.

Sélectionnez l'ensemble des données et cliquez sur le bouton **Trier**.

Nous avons ainsi nos données prêtes à être traitées.

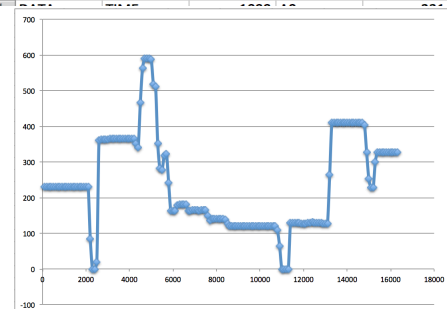


Si l'on a des nombres réels, il faudra penser à convertir les en , !

	A	B	C	D	E
1	DATA	TIME	100	A0	230
2					
3	DATA	TIME	199	A0	230
4					
5	DATA	TIME	299	A0	230
6					
7	DATA	TIME	399	A0	231
8					
9	DATA	TIME	499	A0	231
10					

	A	B	C	D	E
1	DATA	TIME	100	A0	230
2	DATA	TIME	199	A0	230
3	DATA	TIME	299	A0	230
4	DATA	TIME	399	A0	231
5	DATA	TIME	499	A0	231
6	DATA	TIME	599	A0	231
7	DATA	TIME	699	A0	231
8	DATA	TIME	799	A0	231
9	DATA	TIME	899	A0	231
10	DATA	TIME	999	A0	231

Reste alors à faire la représentation graphique...



c Exploiter les données en python

On peut très bien également copier les données du moniteur série et les sauvegarder dans un fichier texte (data.txt par exemple) .

Le programme acquisition.py permet de lire les données dans un fichier texte que l'on vient de créer, de les interpréter et de tracer la courbe.


```

1 import matplotlib.pyplot as plt
2
3 fichier = open('data.txt')
4 lignes = fichier.readlines()
5 fichier.close()
6
7 Temps=[] # temps en s
8 A0=[]    # analogRead sur port A0
9
10 for ligne in lignes :
11     ls = ligne.split(':') # on découpe la ligne
12     if len(ls) >= 5 :
13         Temps.append(float(ls[2])/1000) ;
14         A0.append(int(ls[4]))
15
16 plt.plot(Temps , A0)
17 plt.xlabel('Temps (s)')
18 plt.ylabel('A0')
19 plt.show()

```

d Acquisition/exploitation à partir d'un programme python

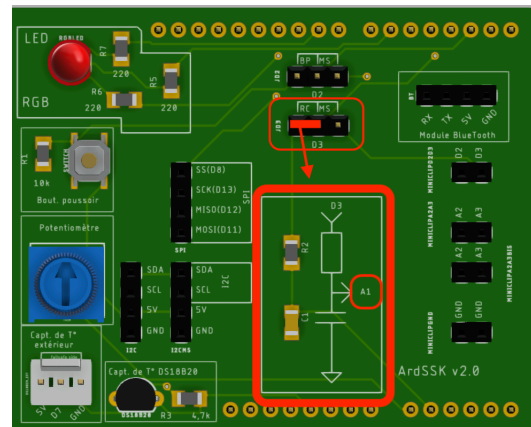
Ce point est assez technique. Nous n'avons pas le temps de l'aborder ici.

1.3 Étude et application d'un circuit RC

Nous allons utiliser, ici, le circuit électrique câblé sur la platine (un circuit RC alimenté par le port **D3** et dont le point milieu est connecté à l'entrée analogique **A1** comme reporté sur le schéma en tête de ce chapitre).



Pour cette partie, il faut mettre le cavalier D3 de la platine du kit vers la gauche, comme sur la figure ci-contre.



Le programme EtudeRC, bien que sommaire, permet l'étude de la décharge du circuit RC câblé sur la carte.

Listing 3.4 – Etude RC

```

1 #define analogPin  A1
2 #define commandePin 3
3 long start = 0 ;
4 int sensorValue = 0 ;
5
6 void setup()
7 {
8     Serial.begin(9600);
9     pinMode(commandePin, OUTPUT) ;
10    digitalWrite(commandePin, HIGH) ;
11    delay(3000) ;

```

```

12   sensorValue = analogRead(analogPin);
13   digitalWrite(commandePin, LOW) ;
14   start = millis() ;
15 }
16
17 void loop()
18 {
19     if (sensorValue > 10)
20     {
21         sensorValue = analogRead(analogPin);
22         String msg = "DATA" ;
23         msg = msg + ":TIME:" + String(millis()-start) ;
24         msg = msg + ":AO:" + String(sensorValue) ;
25         Serial.println(msg) ;
26     }
27 }

```

1.4 On raboute tout...

En combinant la partie commande à partir du moniteur série (chapitre précédent) et acquisition formatée, on peut imaginer un programme générique permettant de servir de modèle pour tout type d'acquisition.

Ce programme pourrait avoir la structure proposée dans `ModeleAcquisition`.

Listing 3.5 – Modèle acquisition

```

① 1 // importation des bibliothèques
2
3 // variables utilisées pour tous les programmes :
4 #define BAUD 9600 // vitesse d'échange pour le port série
5 #define Info "Modèle acquisition"
6 unsigned long start = 0 ;
7 unsigned long last = 0 ;
8 long delai = 1000 ; // Delai en ms entre 2 appels de la
   fonction job
9 bool doJob = false ;
10
② 11 // variables spécifiques au système étudié
12 #define analogPin A0
13
14 // *****
15
16 void setup()
17 {
18     Serial.begin(BAUD) ;
③ 19 }
20
21 void loop()
22 {
23     if (Serial.available())
24     {
25         String message = Serial.readString() ;
26         parse(message) ;

```

```
27     }
28     if (doJob && millis() - last > delai)
29     {
30         job() ;
31         last = millis() ;
32     }
33 }
34
35 void parse(String msg)
36 {
37     msg.replace("\r", "") ;
38     msg.replace("\n", "") ;
39     msg.replace(":", "") ;
40     char ordre = msg[0];
41     msg.replace(String(ordre), "") ;
42     int valeur = 0 ;
43     if (msg.length() > 0)
44         valeur = msg.toInt() ;
45     switch (ordre)
46     {
47         case 'I' :
48             Serial.println(Info) ;
49             break ;
50         case 'G' :
51             go() ;
52             break ;
53         case 'O' :
54             job() ;
55             break ;
56         case 'S' :
57             stop() ;
58             break ;
59         case 'D' :
60             delai = valeur ;
61             break ;
62         default :
63             parse(ordre, valeur) ;
64             break ;
65     }
66 }
67 void go()
68 {
69     doJob = true ;
70     start = millis() ;
71 }
72
73 void stop()
74 {
75     doJob = false ; // on éteint la LED
76 }
77
78 // traitement spécifique au système étudié
79
80 void parse(char ordre, long valeur)
```

```

81 {
82     switch (ordre)
83     {
84         default :
85             break ;
86     }
87 }
88
89 void job()
90 {
91     int sensorValue = analogRead(analogPin);
92     String msg = "DATA" ;
93     msg = msg + ":TIME:" + String(millis() - start) ;
94     msg = msg + ":AO:" + String(sensorValue) ;
95     Serial.println(msg) ;
96 }

```

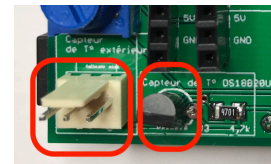
Qu'aurons nous à faire pour adapter ce programme à nos besoins ?

- Compléter les lignes 1 et 2 par l'importation éventuelle d'autres bibliothèques.
- Définir ligne 12 et suivantes les différents ports utilisés sur l'arduino, instancier différents objets associés aux capteurs également.
- Compléter la fonction `parse(char ordre, long valeur)` ligne 80 et suivantes si l'on doit passer d'autres paramètres ou informations au système.
- Et enfin, modifier la fonction `job`.
- Et c'est tout !

2

Utilisation d'un capteur et de sa bibliothèque

Nous prenons l'exemple du capteur de température DS18B20 dont un composant est soudé sur la platine d'essai.



2.1 La bibliothèque DallasTemperature

L'utilisation de la bibliothèque DallasTemperature est alors nécessaire.

Cette bibliothèque a été installée sur votre ordinateur lors de la copie des bibliothèques (cf ?? page ??).

Prenons le premier exemple directement dans l'IDE d'Arduino :

- sélectionner l'item **Exemple** du menu **Fichier** de l'IDE d'Arduino ;
- cliquer ensuite **DallasTemperature** vers le bas du sous menu déroulant qui vous est proposé ;
- sélectionner enfin l'exemple **Simple** dans le nouveau menu.

Le programme se charge alors et apparaît dans l'éditeur.

Il faut changer l'identificateur du port `ONE_WIRE_BUS` : pour la carte configurée, l'identificateur est **7** (et non 2!) (ligne 6).

Téléversez le programme sur la carte et ouvrez le moniteur série. Mettez la main sur le capteur, ça marche !

Le programme est le suivant :

Listing 3.6 – Programme Simple de la bibliothèque Dallas

```
1 // Include the libraries we need
2 #include <OneWire.h>
3 #include <DallasTemperature.h>
4
5 // Data wire is plugged into port 2 on the Arduino
6 #define ONE_WIRE_BUS 7
7
8 // Setup a oneWire instance to communicate with any OneWire
   devices (not just Maxim/Dallas temperature ICs)
9 OneWire oneWire(ONE_WIRE_BUS);
10
11 // Pass our oneWire reference to Dallas Temperature.
12 DallasTemperature sensors(&oneWire);
13
14 /*
15    The setup function. We only start the sensors here
16 */
17 void setup(void)
18 {
19     // start serial port
20     Serial.begin(9600);
21     Serial.println("Dallas Temperature IC Control Library Demo");
22
23     // Start up the library
24     sensors.begin();
25 }
26
27 /*
28    Main function, get and show the temperature
29 */
30 void loop(void)
31 {
32     // call sensors.requestTemperatures() to issue a global
       temperature
33     // request to all devices on the bus
34     Serial.print("Requesting temperatures...");
35     sensors.requestTemperatures(); // Send the command to get
       temperatures
36     Serial.println("DONE");
37     // After we got the temperatures, we can print them here.
38     // We use the function ByIndex, and as an example get the
       temperature from the first sensor only.
39     Serial.print("Temperature for the device 1 (index 0) is: ");
40     Serial.println(sensors.getTempCByIndex(0));
41 }
```

2.2 Il nous faut l'adapter à nos besoins

Que doit-on retenir de ce programme pour l'exploiter par la suite ?

- les lignes 2 et 3 pour l'importation des bibliothèques ;
- les lignes 9 et 11 pour l'instanciation du bus OneWire et du (ou des éventuels) capteur(s) sur ce bus ;
- ligne 24 l'initialisation du (ou des) capteur(s) ;
- ligne 35 on attend la lecture d'une température ;
- ligne 40 on lit la température (la première, et la seule ici).

a Faisons le ménage

En faisant un peu le ménage dans cette bibliothèque on pourrait concevoir le programme suivant.

Listing 3.7 – Acquisition température

```

1  #include <DallasTemperature.h>
2  #include <OneWire.h>
3
4  #define ONE_WIRE_BUS 7
5  OneWire oneWire(ONE_WIRE_BUS);
6  DallasTemperature sensors(&oneWire);
7
8  void setup()
9  {
10     Serial.begin(9600);
11     sensors.begin();
12 }
13
14 void loop()
15 {
16     sensors.requestTemperatures();
17     float T = sensors.getTempCByIndex(0);
18     String msg = "DATA" ;
19     msg = msg + ":TIME:" + String(millis() ) ;
20     msg = msg + ":T:" + String(T, 1) ;
21     Serial.println(msg) ;
22     delay(1000) ;
23 }
```

Handwritten annotations on the code listing:

- ① next to line 1
- ② next to line 5
- ③ next to line 11, with an arrow pointing to it
- ⑤ next to line 16, with an arrow pointing to it and the text "dans job()" written next to it
- Below line 22, the text "géré par ailleurs" is written with an arrow pointing up towards the loop body.

b Ou faisons un copier/coller de ce qui nous intéresse !

Nous pouvons facilement adapter le programme `ModeleAcquisition` à nos nouveaux besoins, et créer ainsi le programme `ModeleAcquisition_T`.