C5-160551-INFO
Objet Connectés : Programmation microcontrôleur
TD2

©B. Besserer

année universitaire 2021-2022

1 Chaîne de production de code

1. Du coté de la cible (microcontrôleur), dans quel type de mémoire le **code** est il téléchargé ?
   ☐ RAM ☐ ROM (en général mémoire flash) ☐ registres
2. Le programmeur peut-il spécifier le type de mémoire à utiliser pour les données (constantes,variables) ?, comment ?
   ☐ Non, les données sont toujours dans la RAM
   ☐ Oui, le programmeur peut utiliser des classes de stockage et des modificateurs dans le code source
   ☐ Oui, mais il faut intervenir sur le code assembleur (pas possible avec un langage de haut niveau)

   Le code exécutable, non modifiable, sera stocké en ROM (Flash RAM) L'adresse sera définie lors de l'étape d'édition de liens, généralement à l'aide de directives (dans le code) ou à l'aide de paramètres donnés en ligne de commande (si l'éditeur de liens est invoqué en ligne de commande) ou via l'interface de l'environnement de développement. On peux spécifier l'emplacement des données (donc pas le code exécutable) via des directives en assembleur mais aussi grâce a des mots clés (classe de stockage) dans les langages évolués (`const` dans le langage C par exemple)
3. (Recherche sur internet) Le framework Arduino peut-être utilisé :
   ☐ Avec la plupart des processeurs ATMEL à l'exclusion de ceux avec un noyau ARM
   ☐ Avec la plupart des processeurs ATMEL y compris ceux avec un noyau ARM
   ☐ Avec certains processeurs de la famille MSP430 de Texas Instrument
   ☐ Avec les processeurs PIC de microchip
   ☐ Avec les processeurs core i3 et i5 de Intel

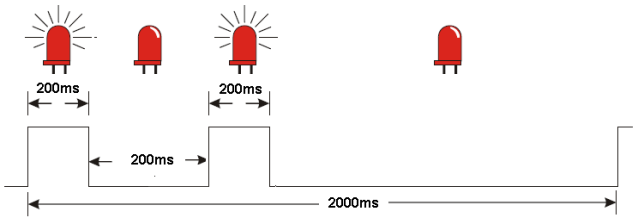
   Atmel -> processeur "natif" Arduino, donc OK
   Avec noyau ARM : La platine Arduino Due utilise un processor Atmel SAM3X8E ARM Cortex-M3, donc OK
   ENERGIA = portage de l'environnement Arduino sur certain (petits) microncontroleurs de la famille MPS430 de TI
   Pour les PIC, il n'y a pas eu de portage mais une platine qui ressemble (PINGUINO) avec un IDE qui reproduit l'environnement ARDUINO, mais PINGUINO n'est pas supporté par l'équipe d'Arduino ou le forum Arduino, et qu'il ne fonctionne pas avec l'environnement de développement d'Arduino. Toutes les librairies et shields Arduino ne sont pas compatibles avec PINGUINO
   Pour les i3, i5, ... il n'y pas de portage de l'environnement Arduino.

2 Entrées / Sorties

1. Quelles fonctions (en utilisant le framework Arduino) permettent généralement de récupérer des informations en provenance d'un capteur connecté au microcontrôleur
   ☐ digitalRead et digitalWrite ☐ digitalWrite et AnalogWrite
   ☐ digitalRead et AnalogRead ☐ digitalWrite et AnalogRead

   Votre programme s'exécute DANS le microcontrôleur et il faut LIRE l'état électrique des broches pour connaître les grandeurs mesurés A L'EXTERIEUR du microcontrôleur. Lecture, donc digitalRead() et analogRead()
2. Complétez le programme (uniquement la fonction `loop()`) qui doit fait continuellement clignoter la LED selon la séquence suivante. La LED utilisée sera sur une GPIO nommée LED_BUILTIN, ce qui correspond à la LED intégrée aux platines microcontrôleur. Celle-ci s'allume lorsqu'on écrit un état HIGH sur ce

GPIO. Utilisez la fonction delay(). Trouvez le code le plus compact possible (env. 4 lignes de code dans loop()).




```
setup()
{
pinMode(LED_BUILTIN, OUTPUT);
}

loop()
{



}
```

VERSION 1 - testée

```
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, LOW);
}

void loop() {
  for(int i=0 ; i < 4 ; i ++) {
    digitalWrite( LED_BUILTIN, !digitalRead(LED_BUILTIN));
    delay(200);
  }
  delay(1200);
}
```

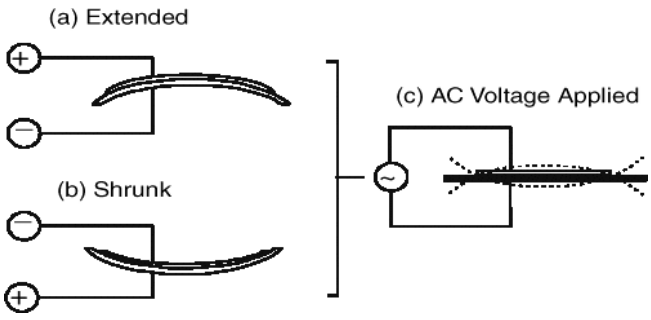
VERSION 2 - testée

```
int t;

void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
  t = 200;
}

void loop() {
  digitalWrite( LED_BUILTIN, HIGH);
  delay(200);
  digitalWrite(LED_BUILTIN,LOW);
  delay(t = ((t + 1200) % 2400));
}
```

3. Un buzzer piézoélectrique est constitué d'une lamelle d'un matériau qui est déformé lorsqu'on applique une tension, comme illustré ci-contre. A ce moment, on entend un "clic". En appliquant la polarité inverse, on entend à nouveau un "clic". Pour générer un son continu, il faut donc faire osciller la lamelle en inversant la polarité à la fréquence de son choix (dans le domaine de l'audible, disons de 500 à 5000 Hz).



Écrivez en syntaxe Arduino/Wiring le code nécessaire pour obtenir un son continu (**avec le maximum de puissance**) d'une fréquence d'approximativement 500 Hz. On suppose que le buzzer est relié entre les sorties 14 et 15 d'un microcontrôleur, soit les définitions suivantes :

```
#define BUZZER1 14    // Borne de sortie
#define BUZZER2 15    // Borne de sortie
```

L'utilisation de la fonction `delay()` est autorisée.

Si f=500Hz, alors la période est de 1/500 = 2ms. Il faut donc que la lamelle soit déformée dans un sens pendant une demi-période (1ms ou bien 1000 microsecondes), puis dans l'autre sens pendant une demi-période.

```
const int demi_periode = 1000; // en microsecondes

setup()
  {
  pinMode(14, OUTPUT);
  pinMode(15, OUTPUT);
  }

loop()
  {
```

```
  digitalWrite(14, HIGH);
  digitalWrite(15, LOW);
  delayMicroseconds(demi_periode);
  digitalWrite(14, LOW);
  digitalWrite(15, HIGH);
  delayMicroseconds(demi_periode);
  }
```

3 Mesures analogiques

Considérez le code ci-dessous.

```
void setup()
{
  Serial.begin(9600);
}
void loop()
{
  min = 4095;
  for(int count = 0 ; count < 10 ; count++)  // on effectue 10 mesures
  {
     value = analogRead(A0);
     delay(5);                  // pause très courte, mais nécessaire si le capteur reagit lentement
     if (value < min)
       min = value;
  }
  Serial.print(value);
  Serial.print("  ");
  Serial.println(min);
  delay(950);
}
```

1. Concernant les variables `min` et `value`, ajouter les déclarations au bon endroit (variable globale ou locale) et en précisant le type.

2. Est-ce que la broche A0 est utilisée en entrée ou en sortie ? ☐ Entrée ☐ Sortie
3. Pour quelle raison la "broche" A0 n'est-elle pas initialisée dans le setup() à l'aide de la fonction pinMode() ?
   ☐ La broche est toujours une entrée
   ☐ La broche est toujours une sortie
   ☐ La broche n'est pas utilisé comme un GPIO en mode entrée/sortie digital, c'est une fonction secondaire de la broche qui est utilisée, l'initialisation avec pinMode() n'est pas nécessaire.
4. Quelle est la période à laquelle s'affiche les relevés de mesure sur le terminal qui est à l'écoute sur le port série et quelles sont les informations qui sont affichées ?
5. Quel est le comportement du programme :
   ☐ Le programme effectue 10 mesures, détermine le minimum, affiche le resultat puis s'arrête
   ☐ Le programme effectue des salves de 10 mesures, détermine le minimum de ces 10 mesures, et répète l'opération toutes les secondes environ.
   ☐ Le programme effectue des mesures de façon continue, et indique le minimum des 10 dernières mesures.
   ☐ Le programme effectue des mesures tant qu'une mesure ne soit pas supérieure à 4095.
6. Pour quelle raison la variable `min` est elle initialisée à la valeur 4095 ?

7. Les sketchs Arduino sont plus ou moins portables d'une plateforme à une autre, mais pas toujours. Est-ce que la valeur d'initialisation (ici 4095) sera la même si la plate-forme cible est un **Arduino uno** ou un **ESP32** ? ☐ Oui ☐ Non
8. Modifier maintenant ce code afin de garder constamment dans 2 variables `min1` et `min2` les 2 valeurs les plus faibles de chaque séquence de mesures.

1. *Les variables ne sont pas des structures complexe, de grande taille, ou des tableaux. Elle ne servent que dans la fonction loop(). Elle peuvent donc être déclarée localement.*
2. *A0 est utilisé pour une acquisition analogique (`analogRead()`). Le programme lit l'état de la broche, c'est donc une entrée.*

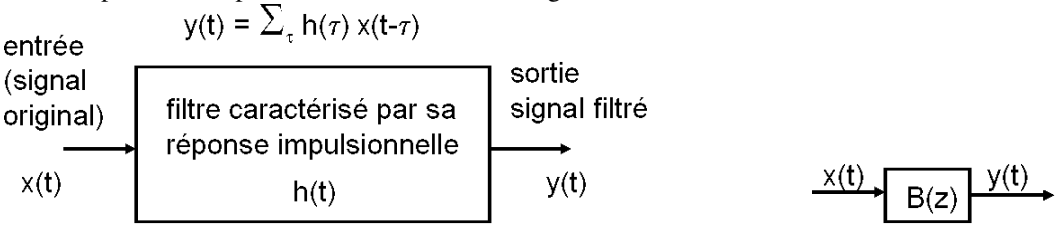
3. *Comme la broche n'est pas utilisé en mode GPIO, c'est une fonction alternative (ici, l'acquisition analogique) qui est utilisée. Utiliser l'instruction `analogRead()` du framework bascule le mode de fonctionnement de cette broche dans le mode d'acquisition analogique et permet donc de lire (donc, entrée ou input) la tension présente sur la broche. Un initialisation avec `pinMode` n'est pas nécéssaire. `pinMode` ne sert que pour les entrées/sortie "digitales" (etat HIGH ou LOW)*
4. *L'affichage des informations sur le terminal dépend des appels de la méthode `Serial.print()` et du temps d'attente entre les appels à cette fonction. Ici, les mesures sont faites dans une boucle avec un faible intervalle (10 x 5ms = 50ms) et il y un délai de 950ms entre deux séries de mesures, donc affichage des informations toutes les secondes (1000ms)*
5. *Le programme effectue des salves (batch) de 10 mesures, détermine le minimum de ces 10 mesures, et répète l'opération toutes les secondes environ.*
6. *Le convertisseur Analogique -> Numérique de l'ESP32 fonctionne sur 12 bits, soit des valeurs pouvant aller de 0 à 4095 ($2^{12} - 1$). Attention, ce n'est plus forcement la même valeur si la plateforme cible est différente.*
7. *Non, sur un Arduino Uno, le convertisseur AD possède une résolution de 10bits, donc valeurs de 0 à 1024, sur un Arduino Due ou un ESP32, la résolution est de de 4095 (12bits)*
8. *Pour mémoriser les deux valeurs les plus petites :*

```
if (value < min1)
{
   min2 = min1;
   min1 = value;
}
 else if (value < min2)
   min2 = value;
```

4 Preparation au TP : Filtrage et moyenne glissante (ou moyenne mobile)

Pas de réponses à donner sur moodle. Réaliser ces exercices vous donnera un avantage considérable lors des travaux pratiques.

Le filtrage numérique est une traitement appliqué à un signal échantillonné ou signal discret (formé d'une suite d'échantillons). Une fonction classique d'un filtre est par exemple de lisser un signal (filtre passe-bas). Un signal d'entrée $x(t)$ est donc modifié par le filtre pour fournir un signal de sortie $y(t)$. La modification effectué par le filtre est une convolution discrète (pour les filtres dit filtre FIR — Finite Impulse Response = Réponse Impulsionelle Finie — ou filtre non récursif). Pour ces filtres, la fonction de transfert $B(z)$ est un polynôme. Beaucoup de traitements numériques des signaux font appel à des produits de convolution discrète se traduisant par des **sommes de produits**. Par exemple, interpoler des valeurs entre 2 instants d'échantillonnage génère un nouvel échantillon de sortie par addition pondérée d'échantillons du signal d'entrée.



Prenons un exemple simple :

$$s_n = a_0 \times e_n + a_1 \times e_{n-1} + a_2 \times e_{n-2}$$

Avec $a_0 = 0.25, a_1 = 0.5, a_2 = 0.25$, cette opération réalise une moyenne pondérée de 3 échantillons consécutifs du signal d'entrée. Vous devez écrire le code qui effectue ce traitement.
Pour tester l'implémentation de l'algorithme on place 100 échantillons d'un signal dans un tableau statique (`data_in[]`). Chaque échantillon est stocké sur 16 bits (10 bits significatifs) :

```
const unsigned int data_in[] =
{512,531,443,390,858,351,656,363,778,409,287,713,172,611,298,683,539,855,313,565,420,759,294,725,462,...};
```

Les données traités par le logiciel doivent être stockées dans un tableau de sortie (`data_out`).

1. Écrivez la boucle de traitement sans tenir compte des "effets de bords" au début et à la fin de tableau. Prêtez particulièrement attention :
   - à la déclaration des différentes variables que vous utilisez
   - aux opérateurs que vous utilisez pour les calculs.

2. Concernant les “effets de bords”, proposez une solution pour traiter ces cas particuliers.
3. Pourrait-on modifier le code pour supprimer toute opération de type multiplication et division ?

— *Écriture “classique” en essayant de se rapprocher au plus près de l’équation mathématique :*

```
int data_out[100];          // a déclarer comme variable globale pour ne pas encombrer la pile

for(int i = 2 ; i < 100 ; i++)
  data_out[i] = 0.25 * data_in[i] + 0.5 * data_in[i-1] + 0.25 * data_in[i-2];
```

— *Pour remplir les 2 premières valeur du tableau de sortie, on peut faire la chose suivante :*

```
int data_out[100];          // a déclarer comme variable globale pour ne pas encombrer la pile

data_out[0] = data_in[0];    // pas de moyenne, car il n'existe pas de valeur en data_in[-1] et en data_in[-2]
data_out[1] = 0.5 * data_in[1] + 0.5 * data_in[0];  // moyenne sur deux points
                                                    // pour la suite, on peut appliquer la formule
for(int i = 2 ; i < 100 ; i++)
  data_out[i] = 0.25 * data_in[i] + 0.5 * data_in[i-1] + 0.25 * data_in[i-2];
```

— *On ne travaille pas sur des valeurs flottantes, mais on veux garder le maximum de précision. On peux multiplier tous les coefficient par 4 et diviser le résultat par 4, cela ne change rien à l’équation (pourquoi 4 : car c’est la valeur qui permet au plus petit nombre positif inférieur à 0 (donc non entier) de devenir entier : 1/0.25 = 4) :*
le coeff. 0.5 devient 2
le coeff. 0.25 devient 1
L’équation devient :

$$s_n = \frac{1}{4} \times (e_n + 2 \times e_{n-1} + e_{n-2})$$

Une multiplication par 2 peut se remplacer par une addition de deux fois la même valeur, et les division par des puissances de 2 sont facile à faire avec les décalages binaires. On peut donc reecrire :

```
int data_out[100];          // a déclarer comme variable globale pour ne pas encombrer la pile
int temp;

data_out[0] = data_in[0];    // pas de moyenne, car il n'existe pas de valeur en data_in[-1] et en data_in[-2]
temp = (data_in[1] + data_in[0]);
data_out[1] = temp >> 1;     // moyenne sur deux points
                             // pour la suite, on peut appliquer la formule
for(int i = 2 ; i < 100 ; i++)
{
  temp = data_in[i] + data_in[i-1] + data_in[i-1] + data_in[i-2];     // uniquement des additions
  data_out = temp >> 2;                                              // division par 4
}
```