Communication avancée

1 Introduction

Il est conseillé d'avoir consulté le document disponible sur le site sur l'introduction aux microcontrôleurs et à la communication avant de poursuivre.
Ce document propose quelques éléments de programmation pour une carte de type Arduino, permettant d'optimiser largement la vitesse de transfert de donnée via la liaison série, par rapport à l'utilisation du simplet *Serial.println*
La méthode *Serial.println* est souvent utilisée, car elle permet très rapidement d'afficher les données à l'écran, via le moniteur série d'Arduino. Sa simplicité d'utilisation et sa rapidité de mise en oeuvre sont ses avantages. Cependant dès qu'une commnucation rapide avec le PC est requise, notamment pour traiter les informations, les afficher en temps réel par exemple, cette approche n'est pas exploitable.

2 Mise en évidence du problème

On désire envoyer des données le plus rapidement possible vers le PC. Commençons par ce programme qui affiche dans le moniteur série d'Arduino, le nombre 2, stocké dans une variable de type *byte*, donc 1 octet. Mesurons le temps pris par l'affichage :

```
byte a = 2;
long t0, dt;
void setup() {
  Serial.begin(115200);
  for (int i = 0; i < 1000; i++) {
    t0 = micros();
    Serial.println(a);
    dt = micros() - t0;
    Serial.println(dt);
  }
}

void loop() {
}
```

Rapidement le temps se stabilise pour attendre une valeur de 252 microsecondes. théoriquement, à 115200 bauds, il faut 8,68 microsecondes pour transmettre un bit, donc pour un octet, (attention il faut transmettre les 8 bits constituant l'octet plus les bits de *start* de *stop*, donc 10 bits au total) il faut 86,8 microscondes. La méthode *println* envoie également à la fin du mot un retour à la ligne qui est le caractère $\backslash n$, soit deux octets, donc un temps de transmission de 173,6 microsecondes. Il y a environ 78 microsecondes en plus inhérentes à l'exécution complète de la méthode dont le contenu n'est pas étudié dans ce document. Déclarons maintenant a en tant que flottant :

```
float a = 2;
long t0, dt;
void setup() {
  Serial.begin(115200);
  for (int i = 0; i < 1000; i++) {
    t0 = micros();
    Serial.println(a);
    dt = micros() - t0;
    Serial.println(dt);
```

```
  }
}

void loop() {
}
```

Le moniteur série affiche maintenant 2.00 et le temps d'envoi est de 512 microsecondes, ce qui à 78 microsecondes près, correspond au temps pour envoyer 5 octets. Si $a = 1234567,89$, le temps d'envoi et d'affichage est de 1020 microsecondes, ce qui, au temps supplémentaire systématique près d'exécution de la méthode, correspond à 11 octets : la méthode println envoie les octets des caractères constituant le mot à envoyer. Alors qu'un flottant est codé sur 4 octets, quelle que soit sa valeur, le temps mis pour envoyer la variable n'est pas constant, car considéré comme un mot de type "String", donc chaque caractère est envoyé individuellement.
Il est évident que cette approche n'est pas optimale. Puisqu'un flottant est codé sur 4 octets, il serait logique d'envoyer ces 4 octets puis de décoder l'information pour reconstituer le nombre. La fonction write, est utile à cette égard.

3 La méthode *write*

Essayons ce programme :

```
byte a = 101;
void setup() {
  Serial.begin(115200);
  Serial.write(a);
}
```

En ouvrant le moniteur série, l'affichage est "e" ! Or, en prenant n'importe quelle table ASCII (disponible rapidement sur le net), on remarque que l'ordinal (valeur numérique du caractère considéré) de "e" est 101. La méthode write envoie donc de manière brute, la valeur de l'octet. Puis le moniteur série le considère comme l'ordinal d'un caractère, et affiche le caractère correspondant. En soit, la méthode est donc inexploitable... avec le moniteur série d'Arduino, mais pas avec un programme capable de le décoder (voir intro micro-contrôleur communication). Maintenant, prenons :

```
unsigned int a = 30319;
void setup() {
  Serial.begin(115200);
  Serial.write(a);
}
```

Le moniteur série affiche "o", dont l'ordinal vaut 111. Or il est important de remarquer que $30319 = 111 + 118.2^8$. 111 est l'octet de poids faible, et 118 l'octet de poids fort de 30319. En conclusion, la méthode *write* envoie (quelque soit le type de variable) uniquement l'octet de poids faible de la variable. Il faut être capable d'extraire les octets de la variable puis de les envoyer avec *write*, un à un en poids croissant.
Plusieurs techniques sont possibles.

4 Extraire les octets d'une variable

4.1 Avec *lowByte* et *highByte*

Dans le document "*Introduction aux micro-contrôleurs et à la communication*", nous avions déjà utilisé ces fonctions mais pour l'extraction d'un type *int*. Comment faire pour une variable dont la place en mémoire est de 4 octets par exemple ?

Considérons le nombre $a = 1684234849$, qui s'écrit aussi $a = 97 + 98.2^8 + 99.2^{16} + 100.2^{24}$. L'octet de poids faible vaut 97, celui de poids fort 100.
Si on essaie ce programme :

```
void setup() {
  Serial.begin(115200);
  unsigned long a = 1684234849;
  byte b0 = lowByte(a);
  byte b1 = highByte(a);
  Serial.println("b0 vaut " + String(b0));
  Serial.println("b1 vaut " + String(b1));
}

void loop() {
}
```

On obtient :

```
b0 vaut 97
b1 vaut 98
```

$b0$ correspond bien à l'octet de poids faible, mais $b1$ n'est pas l'octet de poids fort de a, mais l'octet de poids fort du doublet (mot de 16 bits) de poids faible de a, ce qui correspond bien à la description de la fonction *highByte* donnée sur le site de référence d'Arduino. Pour réaliser ce qui est attendu, il est possible d'utiliser les opérateurs de décalage (on trouve toutes sortes de ressource sur le net à leurs sujets, ils sont très simples à comprendre, leur introduction n'est pas l'objet de ce document). En effet, il faut récupérer le doublet de poids fort de a, puis lui appliquer *lowByte* et *highByte* pour en extraire les deux octets manquants. Ainsi, on peut proposer :

```
void setup() {
  Serial.begin(115200);
  unsigned long a = 1684234849;
  byte b0 = lowByte(a);
  byte b1 = highByte(a);
  unsigned long b=a>>16;
  byte b2=lowByte(b);
  byte b3=highByte(b);
  Serial.println("b0 vaut " + String(b0));
  Serial.println("b1 vaut " + String(b1));
  Serial.println("b2 vaut " + String(b2));
  Serial.println("b3 vaut " + String(b3));
}

void loop() {
}
```

La sortie est :

```
b0 vaut 97
b1 vaut 98
b2 vaut 99
b3 vaut 100
```

Ce qui est attendu.

4.2 Avec les pointeurs

Cependant le langage C propose un outil extrêmement puissant : les pointeurs. Le concept est légèrement plus difficile à assimiler que ce qui vient d'être proposé jusqu'à maintenant. Afin de continuer, il est préférable de se documenter. Voici quelques liens plutôt pédagogiques que techniques, et bien faits :

- `https://www.youtube.com/watch?v=Qra-0U_jEKs`
- `https://www.rocq.inria.fr/secret/Anne.Canteaut/COURS_C/chapitre3.html`
- `https://openclassrooms.com/fr/courses/19980-apprenez-a-programmer-en-c/15417-a-lassaut-des-p`

Les pointeurs sont utilisable dans Arduino, puisque basé sur le langage C :

```
void setup() {
  Serial.begin(115200);
  unsigned long a = 1684234849;
  unsigned long *pa = &a;
  unsigned long b = pa;
  unsigned long c = *pa;
  Serial.println(b);
  Serial.println(c);
  *pa = 10;
  Serial.println(a);
}

void loop() {
}
```

Dont la sortie après compilation est :

```
8695
1684234849
10
```

Détaillons ligne à la ligne.

```
unsigned long *pa=&a;
```

Cette ligne permet de déclarer un pointeur **pa** pointant sur la variable **a**. **pa** a pour valeur l'adresse de **a**, alors que ***pa** vaut la valeur de **a**.Pour afficher l'adresse de **a**, et donc **pa**, on est tenté d'écrire :

```
Serial.println(pa);
```

Mais Arduino ne l'accepte pas. Ainsi, pour l'afficher, on peut simplement créer une variable **b** à laquelle on affecte la valeur de **pa**, donc l'adresse de **a**. C'est une petite astuce qui peut être parfois utile. Il s'agit de la ligne :

```
unsigned long b = pa;
```

Ici, l'adresse de **a** vaut 8695. Cette valeur ne sera pas nécessairement la même lors d'une modification du code, lorsqu'on débranche/rebranche la carte, etc. Pour afficher la valeur de la variable pointée, soit ***pa**, on utilise le même contournement :

```
unsigned long c = *pa;
```

La sortie confirme que **pa** pointe bien vers **a**.
Enfin, en modifiant ***pa**, on modifie **a**.
La variable **a**, de type *unsigned long*, est codé sur 4 octets dans une Arduino Mega, ou Uno. On peut afficher l'adresse de deux variables de ce même type :

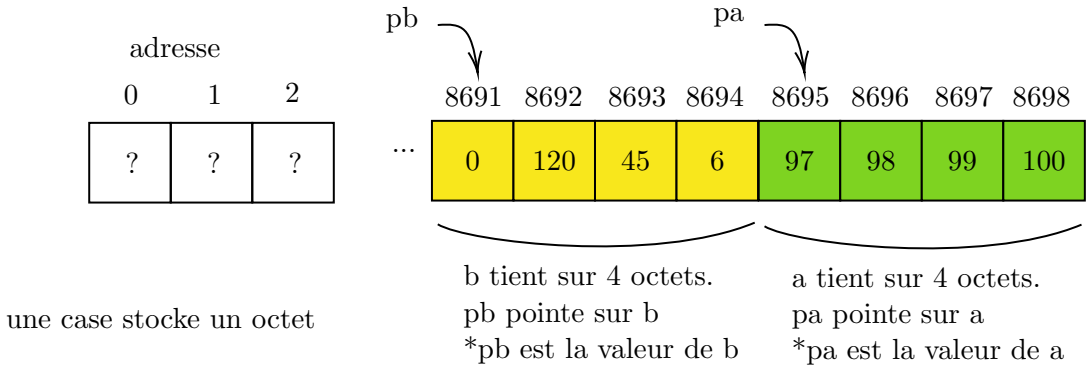
```
void setup() {
  Serial.begin(115200);
  unsigned long a = 1684234849;
  unsigned long b = 103643136;
  unsigned long *pa = &a;
  unsigned long *pb = &b;
  unsigned long adra = pa;
  unsigned long adrb = pb;
  Serial.println(adra);
  Serial.println(adrb);
}

void loop() {
}
```

Dont la sortie est :

```
8695
8691
```

L'adresse de **b** est 8191, puis 4 case mémoires plus loin , on trouve celle de **a**. Ce qui conforte ce qui est vient d'être dit sur le type *unsigned long*. Pour comprendre comment ces deux variables sont positionnées dans la mémoire de la carte, on peut dresser le schéma suivant (ici $b = 103643136 = 0+120.2^8+45.2^{16}+6.2^{24}$) :



L'adresse d'une variable pointée correspond donc à l'adresse de l'octet de poids faible. Comment accèder à chaque octet individuellement ? Commençons déjà par une constatation en considérant le programme suivant :

```
void setup() {
  Serial.begin(115200);
  unsigned long a = 1684234849;
  unsigned long b = 103643136;
  unsigned long *pa = &a;
  unsigned long *pb = &b;
  unsigned long adra = pa;
  unsigned long adrb = pb;
  unsigned long c = *pb;
  unsigned long d = *(pb + 1);
  Serial.println(adra);
  Serial.println(adrb);
  Serial.println(c);
  Serial.println(d);
}
```

```
void loop() {
}
```

Ce qui donne :

```
8695
8691
103643136
1684234849
```

***pb** est égale à la valeur de la variable vers laquelle **pb** pointe, c'est à dire **b**. Mais ***(pb+1)** a la valeur de la variable dont l'adresse est situé à 1 "longueur d'un *unsigned long* plus loin que celle de b, c'est à dire 4, car le pointeur est de type *unsigned long*. Remarquons que :

- ***(pb+1)** est la valeur de la variable située immédiatement après l'adresse de **b** (si ***pb** était un *int*, il aurait pointé "deux cases mémoire" plus loin
- ***pb+1** vaudrait la valeur de b, à laquelle on rajoute 1, ce qui est complètement différent.

Si l'on veut extraire les 4 octets de **a** en utilisant les pointeurs, on peut proposer :

```
void setup() {
  Serial.begin(115200);
  unsigned long a = 1684234849;
  byte *p = (byte *)&a;
  byte p0 = *p;
  byte p1 = *(p + 1);
  byte p2 = *(p + 2);
  byte p3 = *(p + 3);
  Serial.println(p0);
  Serial.println(p1);
  Serial.println(p2);
  Serial.println(p3);
}

void loop() {
}
```

Ce qui donne :

```
97
98
99
100
```

La ligne la plus importante est la suivante :

```
byte *p = (byte *)&a;
```

A cette ligne, on déclare un pointeur **p**, de type octet (*byte*), qui pointe vers le premier octet de la variable *a*. Attention, l'opération de cast s'écrit bien **(byte *)&a** et non **(byte)&a**. Cela ne modifie biensûr pas le type de **a** (*unsigned long*), mais maintenant, le pointeur est de taille 1. Ainsi ***(p+1)** pointera vers l'adresse de **a**, décalée de 1 (car le pointeur est un type byte), et ainsi de suite.
Si on modifie un des 4 pointeurs, cela modifiera la valeur de **a** en conséquence. On peut tout aussi bien créer un tableau de pointeurs, dont les éléments sont des pointeurs vers les différents octets de **a** :

```
void setup() {
  byte val;
  Serial.begin(115200);
  unsigned long a = 1684234849;
  byte *tab[4];
  for (int i = 0; i < 4; i++) {
    tab[i] = (byte *)&a + i;
    val = *tab[i];
    Serial.println(val);
  }
  *tab[1] = 1;
  *tab[0] = 0;
  *tab[2] = 0;
  *tab[3] = 0;
  Serial.println(a);
}

void loop(){
}
```

Dont la sortie est :

```
97
98
99
100
256
```

Le tableau de pointeur est nommé **tab**, de taille 4 et de dimension 1, c'est un tableau de pointeur pointant vers des variables de type *byte*, dont initialisation est réalisée par l'instruction :

```
byte *tab[4];
```

Ensuite, on affecte à chaque élément du tableau l'adresse de l'octet vers lequel il doit pointer par :

```
tab[i] = (byte *)&a + i;
```

Pour vérification, on afficher la valeur de l'octet pointé (éléments du tableau), ce qui est conforme.
On peut ensuite facilement modifier les octets de **a**. En fixant le deuxième octet à 1, et tous les autres à 0, **a** vaut bien 256.

5 Mise pour en œuvre pour la communication via la liaison série

On peut proposer des fonctions pour faciliter la transmission d'informations vers le PC. Ce qui est proposé n'est probablement pas la meilleure solution, mais elle est très efficace en terme de rapidité. Sur certaines cartes électroniques telle que la *Teensy 3.6*, il est possible d'atteindre 12Mbits/s, si le PC suit derrière évidemment.

5.1 Envoyer une valeur.

Supposons que l'on veuille envoyer une variable de type *unsigned long*. On peut le faire, par exemple avec :

```
void write_ul(unsigned long var) {
  byte len = sizeof(unsigned long);
  byte *tab[len];
  for (int i = 0; i < len; i++) {
    tab[i] = (byte *)&var + i;
  }
  Serial.write(*tab,len);
}
```

La fonction *sizeof* est très pratique car elle renvoie la taille d'une variable de type passé en argument. Un type *int* sur une Arduino Uno tient sur 2 octets, alors que sur une Arduino Due, c'est 4 ! Cette fonction s'utilise en passage de l'argument par valeur :

```
void setup() {
    Serial.begin(115200);
    unsigned long a=1684234849;
    write_ul(a);
  }

  void loop() {
  }
```

5.2 Recevoir une valeur et modifier le contenu d'un variable

Si on souhaite lire depuis l'Arduino, une série d'octet correspondant à une variable envoyée par la liaison série, puis affecter cette valeur à une variable dans l'Arduino, on peut proposer la fonction suivante :

```
void read_ul(unsigned long *var) {
  byte len = sizeof(unsigned long);
  byte *pvar=(byte *)var;
  byte val;
  for (byte i = 0; i < len; i++) {
    *(pvar+i)=Serial.read();
  }
}
```

La fonction sera utilisée en passant l'argument par adresse, et modifiera la variable pointée :

```
void setup() {
    Serial.begin(115200);
    unsigned long a;
    while(Serial.available()!=4);
    read_ul(&a);
```

```
  }

  void loop() {
  }
```

Attention, dans la fonction le pointeur de type *byte* se déclare bien comme ceci :

```
byte *pvar=(byte *)var;
```

Car *var* est déjà un pointeur, qui contient l'adresse passé en argument !

Ces fonctions sont à adapter suivant le type à lire, ou à écrire. Par exemple pour envoyer un flottant :

```
void write_f(float var) {
  byte len = sizeof(float);
  byte *tab[len];
  for (int i = 0; i < len; i++) {
    tab[i] = (byte *)&var + i;
  }
  Serial.write(*tab,len);
}
```

Et pour en recevoir un :

```
void read_f(float *var) {
  byte len = sizeof(float);
  byte *pvar=(byte *)var;
  byte val;
  for (byte i = 0; i < len; i++) {
    *(pvar+i)=Serial.read();
  }
}
```