TD 1 : commande de leds et chenillard

Systèmes Embarqués
ING2, EILCO
P.-A. Hébert
2023–2024

Consignes

Ce premier TD machine a les objectifs suivants :

— développer un premier programme en *Langage C++* sur la carte *Arduino Uno*,
— découvrir la *boucle infinie* propre aux microcontrôleurs,
— utiliser les *ports d'entrées/sorties* (E/S) de l'*Arduino*, à « haut » puis « bas » niveau,
— et, en passant : apprendre à se méfier des calculs et mesurer l'importance des interruptions.

Dans cette première séance, on utilisera l'*Arduino* seul, sans périphériques supplémentaires, à l'exception d'un petit fil électrique.

1 Composition d'un programme *Arduino*

Un programme Arduino contient deux fonctions obligatoires.

1.1 La fonction `setup`

C'est la fonction d'initialisation du programme. Son rôle principal est d'initialiser les données du programme, ainsi que les périphériques internes mis en œuvre.

C'est la première fonction lancée par l'*Arduino*, aussitôt le programme chargé.

1.2 La fonction `loop`

Tout programme destiné à un microcontrôleur sans système d'exploitation - comme la carte *Arduino* - répète indéfiniment une tâche particulière. Cette tâche est donc placée au sein d'une boucle infinie (généralement à l'aide d'une structure de contrôle `while(1)` dans un programme C++).

Généralement, cette tâche décrit le « cycle type » de la commande d'un système (exemple pour un distributeur de boissons : une interaction complète avec l'utilisateur, de la réception de la monnaie jusqu'à la production de la boisson et au rendu de la monnaie).

La boucle infinie est nécessaire, car en l'absence de système d'exploitation, le programme doit spécifier à tout instant l'action du processeur. Celui-ci ne peut donc pas « se terminer ». Sinon, le processeur ne s'arrêtant jamais de fonctionner, il interpréterait la portion mémoire suivant le programme comme un ensemble d'instructions, et ferait vite n'importe quoi.

Sur *Arduino*, la boucle infine est cachée, et contient une seule instruction : un appel à la fonction `loop`. Cette fonction *loop* est donc le cœur du programme. Elle s'exécute de manière répétée, indéfiniment, sitôt la fonction `setup` terminée.

1.2.1 Structure d'un programme *Arduino*

```
// déclarations de bibliothèques, variables globales, fonctions externes

void setup()
{
    // initialisation
}

void loop()
{
    // instructions de la boucle infinie
}
```

2 Exercice 1 : clignotement d'une LED

2.1 Question 1

Complétez (et testez sur machine) le programme suivant, de manière à faire clignoter la LED.

Dans cette question, ne modifiez pas la fonction de pause, et travaillez exclusivement avec les fonctions présentes dans le code fourni. Celles-ci sont décrites en annexe 4.1.

```
// Declaration de variables globales
const int BROCHE_LED = 13;
const unsigned int compteur = 2500;

// Fonction d'initialisation : lancee une fois, au debut du programme
void setup() {
  pinMode(BROCHE_LED, OUTPUT);      // broche 13 de l'UNO placee en SORTIE
  Serial.begin(9600);               // initialisation de la liaison serie
}

// Calcul bidon qui fait office de pause
int pause(unsigned int cpt)
{
  int valeur = 0;
  while(cpt-->0)                    // equivalent a : while( (cpt--) > 0 )
    valeur = valeur + random(10000);
  return valeur;
}

// Fonction de boucle : relancee indefiniment
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);  // allume la LED
//digitalWrite(BROCHE_LED, 1);       // equivalent : allume la LED
  pause(compteur);                  // pause
}
```

2.2 Question 2

On souhaite accroître la durée de la pause. Pour cela, remplacez la valeur 2500 par la valeur 66000.

Que constatez-vous ? Corrigez le problème. Aide : affichez la nouvelle valeur limite du compteur.

2.3 Question 3

Remplacez la fonction de « pause » par la fonction `delay(ms)`, où `ms` correspond à la durée de la pause en millisecondes.

La LED devra être allumée 1s toutes les 1,3s.

2.4 Question 4

On souhaite ici se passer des fonctions « haut niveau » de gestion des E/S.

Réalisez la même chose en remplaçant les appels de fonction `pinMode` et `digitalWrite`, par des instructions opérant sur les registres du microcontrôleur dédiés aux E/S numériques.

Se référer à l'annexe 4.2.

3 Exercice 2 : animation d'un chenillard

3.1 Préambule

3.1.1 Cahier des charges

On souhaite réaliser un chenillard à l'aide de 4 lampes LEDs connectées à des broches contiguës, que l'on allume alternativement, de manière à évoquer le déplacement d'un point (voire d'une « chenille »).

En résumé, vous devez :

— programmer un chenillard sur 4 LEDs : elles s'allument successivement, de la première à la dernière, avec un intervalle d'environ 1s entre chaque ; elles seront connectées aux **broches 3 à 6** de l'*Arduino*, qui appartiennent à son **port D** ;

— le chenillard se lance à chaque pression d'un bouton placé sur la **broche 8** ; un appui intermédiaire ne sera pas pris en compte.

3.2 Question 1

Programmez le chenillard, de manière à le faire tourner en permanence.

Même s'il n'y a que 4 LEDs (il pourrait y en avoir beaucoup plus !), essayez d'optimiser/structurer un minimum votre code :

— évitez la redondance de code, grâce aux boucles, fonctions, éventuellement tableaux ;

— pensez aux opérateurs C++ de décalage de bits : >> et << ;

— pensez aux opérateurs logiques bit-à-bits (distincts des opérateurs logiques booléens !), pour modifier un ensemble de bits d'un registre sans altérer les autres :

  — *OU*, pour forcer des bits à 1 : `|`

  — *ET*, pour forcer des bits à 0 : `&`

  — *NON*, pour inverser des bits : $\sim$

Paramétrez bien tous les registres impliqués. Pensez en particulier à :
— orienter correctement les 5 broches mises en œuvre, sans modifier la direction des autres broches ;
— activer la résistance de *pull-up* de la broche en entrée ;
— modifier les tensions des broches de sortie sans altérer les autres, notamment pour ne pas interférer avec les lignes utilisées par la liaison série (lignes RX et TX, respectivement connectées aux broches 0 et 1 du port D).

3.3 Question 2

Modifiez le code précédent, de manière à déclencher le chenillard sur une mise à l'état bas (= à la masse) de la tension de la broche 8.

3.4 Question 3

Observez l'incidence de l'instruction `noInterrupts()` en fin de `setup`. L'objectif de cette question est de souligner l'importance des `interruptions`, qui seront étudiées plus tard.

Vérifiez que vous obtenez le même comportement en forçant à 0 le bit `I` du registre d'état (`SREG`, cf. diapos de cours).

3.5 Question 4

Modifier le chenillard, de manière à ce qu'il fasse des allers-retours : dès que la LED allumée est à une extrémité, le chenillard repart dans l'autre sens (c'est donc la LED voisine qui s'allume ensuite, pas celle située à l'autre extrémité).

4 Annexes

4.1 Programmation « *haut niveau* » des E/S de l'*Arduino* + liaison série

Les niveaux de tension des broches externes du microcontrôleur de l'*Arduino* peuvent être contrôlés, soit en lecture (= **entrée**, *du point de vue du microcontrôleur*), soit en écriture (= **sortie**).

La fonction `pinMode` a pour rôle de spécifier le sens d'utilisation des broches. L'orientation se fait soit en sortie (valeur `OUTPUT=1`), soit en entrée (valeur `INPUT=0`), soit en entrée-« pull-up » (valeur `INPUT_PULLUP=2`). Ce dernier mode connecte la broche à la source de tension de l'arduino (5V) via une résistance de valeur importante (c'est la résistance de « *pull-up* »). Grâce à ce mécanisme, même sans la relier à quoi que ce soit, la broche dispose donc d'une tension stable, « par défaut ».

La fonction `digitalWrite` permet de définir la tension d'une sortie numérique, à condition qu'elle ait préalablement été dirigée en **sortie** par la fonction précédente.

La fonction `Serial.begin` initialise le dialogue entre l'*Arduino* et le pc par l'interface série UART, en spécifiant la vitesse de communication (qui doit être la même que celle spécifiée au niveau du pc : 9600 bauds par défaut).

4.2 Programmation « *bas niveau* » des E/S de l'*Arduino*

Les fonctions ci-dessus dédiées aux E/S opèrent en manipulant quelques *registres* du microcontrôleur.

Un **registre** est une zone de la mémoire interne destinée à piloter le microcontrôleur et ses périphériques. Il se manipule comme n'importe quelle variable : on peut aussi bien le lire que modifier son contenu, qui se trouve préserver jusqu'à sa prochaine modification. Toute modification implique aussitôt un changement du comportement du microcontrôleur.

Dans l'environnement de programmation *Arduino*, les registres sont connus du compilateur, et ne nécessitent donc pas d'inclusion de bibliothèque.

4.2.1 Programmation des sorties

Les broches en sortie peuvent être contrôlées par les *registres de donnée de port*. Ils s'utilisent tous de la même façon. Chacun de leur bit contrôle la tension d'une broche particulière :

— un bit à 0 code une tension à l'état bas,
— un bit à 1 code une tension à l'état haut.

Le *registre de donnée de port* dédié au port D (broches 0 à 7 du microC) est le registre `PORTD`.

Mais pour pouvoir travailler sur ce registre, il faut au préalable désigner au microcontrôleur les broches en **sortie** (elles sont par défaut en **entrée**).

Les directions des broches d'E/S numériques sont contrôlées par les *registres de direction de port*. Pour les 8 broches du port D, il s'agit du registre `DDRD` :

— un bit à 0 code une broche en entrée,
— un bit à 1 code une broche en sortie.

Pour résumer, l'utilisation de lignes de port en sortie nécessite :

1. d'initialiser le *registre de direction de port* pour orienter les lignes en sortie ;

2. de spécifier dans le *registre de données de port*, les tensions de ces broches.

4.2.2 Programmation des entrées

La 1ère étape est la même : il faut commencer par orienter les lignes à utiliser, cette fois-ci en **entrée**. On opère là encore sur les *registres de direction du port*, donc `DDRD` pour le port D.

Mais il y a 2 modes d'entrée : il faut ensuite spécifier si on connecte ou non la résistance interne de « pull-up » à la broche d'entrée. Cela se fait grâce aux *registres de donnée de port* : `PORTD`, pour le port D. Les bits à 1 *sur les lignes dirigées en entrées* activent la résistance pull-up des broches correspondantes. Par défaut elles ne sont pas activées (bits à 0).

Pour finir, les lectures de tension se font sur les *registres d'entrée de port*, `PIND` pour le port D :

— un bit à 0 indique un niveau de tension bas,
— un bit à 1 indique un niveau de tension haut.