C5-160551-INFO
Objets Connectés :
Programmation Microcontrôleurs - TP2

©Bernard Besserer

année universitaire 2021-2022

1 Programmation sur l'ESP32 et affichage sur le terminal

Pour commencer un petit programme qui n'utilise aucune entrée/sortie sur des broches, uniquement la liaison série. Écrivez un programme qui permette de calculer la suite de Fibonacci **sans utiliser de programmation récursive**. On rappelle que :

$$\mathcal{F}_{n+2} = \mathcal{F}_{n+1} + \mathcal{F}_n$$

Chaque terme de cette suite est la somme des deux termes précédents, les 2 premiers termes étant 1 et 1, soit 1, 1, 2, 3, 5, 8, 13,... Les termes de la suite doivent être transmis à la machine hôte pour affichage via la liaison série qui sert au debug.

Manipulation à effectuer :

1. Commencez par écrire le programme qui calcule la suite jusqu'à un rang N arbitraire, disons les 20 premiers termes de la suite. C'est une boucle et à chaque tour vous écrivez une valeur vers le terminal série.
   Il fut initialiser la liaison série dans le setup : `Serial.begin(115200);`
   Puis, dans la boucle : `Serial.println(value);` `value` étant la variable qui contient le terme de la suite. Insérez un délai de 500ms dans la boucle, et typez les variables utilisées pour calculer la suite comme des `short`
2. Modifier le programme en calculant la suite de Fibonnacci au rang N, N étant une valeur entière saisie par l'utilisateur dans le terminal série. Votre programme doit d'abord afficher un message du type : "saisir le rang :" dans le terminal série. Ensuite, l'utilisateur doit entrer une valeur qui correspond au rang N pour le calcul de la suite qui commence aussitôt.

   Il y plusieurs façon de lire un nombre de plusieurs chiffres depuis le terminal, n'oubliez pas que le microcontrôleur reçoit une suite de caractères ASCII, qu'il faut examiner et convertir en nombre. Plusieurs solutions sont possibles. Examinez le code suivant :

```
long nb;
Serial.setTimeout(5000);            // attente de 5sec max pour Serial.parseInt, puis on passe à la suite
do {
   nb = Serial.parseInt();          // Serial.parseInt detecte les caractères qui ne sont pas des chiffres
   if (Serial.read() == '\n')       // si le caractère est un retour à la ligne (touche ''entrée'')
      break;                        // alors on quitte la boucle
   } while(Serial.available() > 0); // tant qu'il y a des caractères dans le buffer
Serial.println(nb);                 // Pour verification : affiche la valeur avec un saut de ligne
```

Pour que ce code fonctionne, il faut que le terminal série soit réglé pour transmettre un saut à la ligne (*newline*) après la saisie.

Questions : Si les variables qui sont utilisés pour calculer les termes de la suite sont typés comme des `short`, à partir de quel rang les résultats sont incorrects ? Pourquoi ? (répondre sur moodle)
Ajoutez dans la boucle un test permettant de détecter des résultats incorrects et modifier votre code afin que le programme calcule la suite de Fibonacci jusqu'à ce que le résultats devient incorrect.
Si les variables qui sont utilisés pour calculer les termes de la suite sont typés comme des `int`, le cacul est possible jusqu'à quel rang ? (répondre sur moodle)
Si les variables sont typées comme des `long` ?
Si les variables sont typées comme des `long long` ?

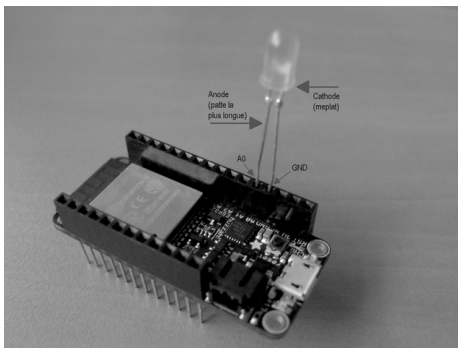
2 Mesure d'une grandeur analogique

Nous allons utiliser une LED de manière peu conventionnelle : Lorsque du silicium semi-conducteur est illuminé, celui-ci fonctionne comme une cellule photovoltaïque et produit une (très faible) tension. Votre LED va être utilisée comme cela (on s'en servira donc comme capteur de lumière).
La tension généré par la LED est de l'ordre de la centaine de μvolts seulement (disons 0 dans le noir et 400 μV en pleine lumière). Mais l'étage d'entrée du CAN[1] intégré dans l'ESP comprend un amplificateur à gain programmable sur 4 niveaux ($\times 1$, $\times 4$, $\times 8$, $\times 16$).
Si le circuit CAN utilise sa référence de tension interne (2.5 V) pour sa pleine échelle et sachant que la résolution du CAN est sur 12 bits, combien de niveaux différents de luminosités sera-t'il possible de détecter ?
Soit le programme ci-dessous :

```
void loop()
{
  cumul = 0;
  for(count =0; count < 8 ; count++) // do 8 measures, compute average to reduce noise
  {
    value = analogRead(A0);
    delay(1);                        // short delay because A-D conversion need time
    cumul = cumul + value;
  }
  cumul = cumul >> 3;                // A vous de determiner ce qui est fait avec cette ligne de code...
  Serial.println(cumul);
  delay(1000);
}
```

Utilisez une LED verte et réalisez le montage ci-dessous (Attention, la LED est un composant polarisé, veillez à la brancher dans le bon sens). Ensuite, recopiez le programme ci-dessus et ajoutez la fonction setup(), **les déclarations nécessaires (variables)**, etc... et testez. En faisant plus ou moins obstacle à la lumière, les valeurs relevés et transmise sur la liaison série doivent changer.
Si les valeurs restent toujours trop élevées, ajouter éventuellement dans le setup une commande `analogSetAttenuation(ADC_6db);`

```
analogSetAttenuation(attenuation): sets the input attenuation for all ADC pins. Default is ADC_11db.
Accepted values:
ADC_0db: sets no attenuation (1V input = ADC reading of 1088).
ADC_2_5db: sets an attenuation of 1.34 (1V input = ADC reading of 2086).
ADC_6db: sets an attenuation of 1.5 (1V input = ADC reading of 2975).
ADC_11db: sets an attenuation of 3.6 (1V input = ADC reading of 3959).
```



2.1 Moyenne glissante (ou moyenne mobile)

Dans votre code, on calcule un moyenne simple sur 8 mesures répétitives très proche (le calcul de la moyenne minimise le "bruit" de mesure). Gardez ce code mais modifiez la fonction `loop()` afin de calculer la moyenne glissante (ou moyenne mobile, voir `fr.wikipedia.org/wiki/Moyenne_mobile`) sur les 4 dernières mesures moyennées. La valeur de `cumul` (**celle qui est affichée**) représente une mesure par seconde, il faut donc calculer la moyenne mobile sur les 4 dernières valeurs de `cumul` ($e_{n-1}, e_{n-2}, e_{n-3}, e_{n-4}$).
A la seconde suivante, arrivée d'une nouvelle mesure e_n, la moyenne est recalculée avec les valeurs ($e_n, e_{n-1}, e_{n-2}, e_{n-3}$)
Pour implémenter cet algorithme, **au moins les 3 dernières mesures doivent être mémorisées, il faut donc des variables supplémentaires**. A l'arrivée d'un nouvelle mesure et après calcul de la moyenne, la mesure e_{n-3} peut être supprimée car elle n'est plus nécessaire, mais la mesure e_{n-2} devient la mesure e_{n-3}, etc...
Écrivez le code qui calcule la moyenne glissante et modifiez l'affichage sur la liaison série en écrivant sur une seule ligne, séparés par des point-virgules, les 4 valeurs utilisés pour le calcul ainsi que la moyenne, comme ceci :

```
542;544;543;537;moyenne mobile
544;543;537;522;moyenne mobile
```

La dernière valeur devra être affichée avec `Serial.println` qui ajoute un saut de ligne après affichage.

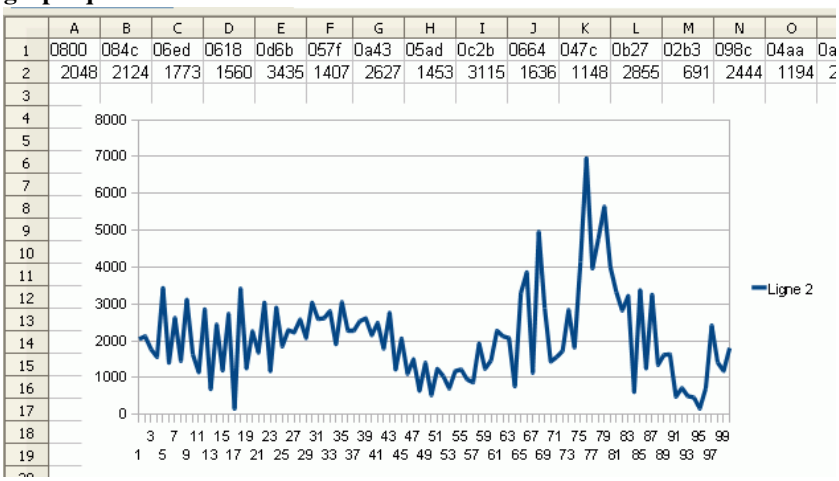
1. CAN = Convertisseur Analogique -> Numérique, ou DAC en anglais : *Analog to Digital Converter* ; c'est le périphérique interne qui est utilisé quand vous appelez la fonction `analogRead()`

2.2 Visualisation des signaux à l'aide d'un tableur

Il serait intéressant de pouvoir visualiser un relevé de mesures (ou un signal) avant et après un traitement, comme ici le calcul d'une moyenne mobile. Pour cela nous allons enregistrer dans un fichier les informations qui proviennent du port série et qui s'affichent normalement dans le terminal. Il existe plusieurs possibilités pour réaliser cela, mais l'utilisation d'un programme *freeware* de type terminal intégrant une fonctionnalité de sauvegarde va simplifier le travail.

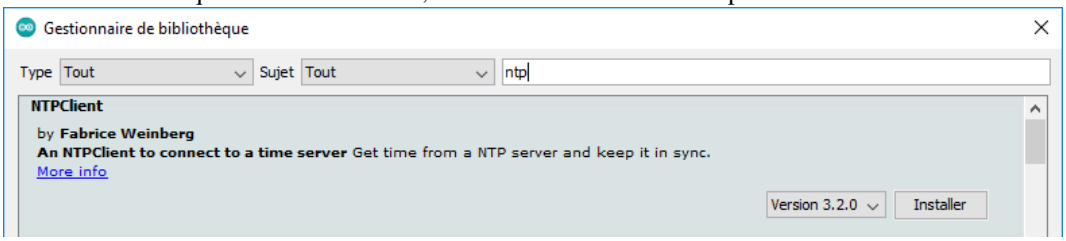
Utilisez par exemple le programme disponible ici `https://sites.google.com/site/terminalbpp/`. Il est de petite taille et ne nécessite pas d'installation avec des droits d'administration.
Développez avec l'IDE ARDUINO, mais à l'exécution du code sur le microcontrôleur, au lieu d'ouvrir le terminal intégré à l'IDE pour voir les infos transmis via les appels à Serial.print, lancez l'application Terminal. Sélectionnez le bon port, les bons paramètres (115200,7) et cliquez sur connect. Vous devez alors voir les messages transmis par la liaison série.
Et de plus, dans la partie "receive" du logiciel, vous pouvez démarrer l'enregistrement d'un fichier log. Faites des essais en occultant une ou deux fois la LED pendant la durée de votre enregistrement. Lorsque vos données sont enregistrées dans un fichier, vérifiez que ces informations sont conformes au format CSV (Comma Separated Values) :
Séparateur ';' (point-virgule) Saut de ligne après les valeurs

Dans un premier temps, vous récupérez comme au point 2.1 les 4 dernières valeurs et la valeur lissée (moyenne mobile). Par la suite (2.3), vous pouvez rajouter sur la ligne d'affichage l'état de la LED (0 ou 1)
Ce fichier CVS sera ensuite ouvert par un tableur (La suite Open Office, Libre Office ou Excel si c'est installé en local sur la machine, ou alors uploadez le fichiez dans Google Drive et ouvrez-le dans Google Docs, idem avec les outils Microsoft office.live.com auxquels vous avez normalement accès en ligne avec votre compte étudiant). Les tableurs savent ouvrir les fichiers CSV, le but est d'afficher graphiquement les mesures. Il suffit de sélectionner les deux dernières colonnes pour visualiser les mesures directes et la moyenne mobile sur les 4 dernières valeurs. A vous de trouver les bon réglages dans Calc ou Excel pour afficher les mesures et les mesures filtrées sur le **même graphique**.



3 Objets connectés : synchronisation NTP et horodatage des mesures

On commence ici l'utilisation de l'ESP32 en tant qu'objet connecté. On va utiliser le protocole internet NTP (*Network Time Protocol*, voir `fr.wikipedia.org/wiki/Network_Time_Protocol`) pour demander l'heure à un serveur NTP et synchroniser l'horloge interne de l'ESP32. L'utilisation de ce protocole est simplifié grâce à une bibliothèque définissant un objet NTPClient. Il vous faudra installer cette bibliothèque. Aller dans le gestionnaire de bibliothèque de l'IDE Arduino, saisissez NTP dans le champ de recherche et installez cette bibliothèque :



Il faut aussi connecter l'ESP32 à internet, en passant par un réseau WiFi. Pas besoin de bibliothèque additionnelle pour cela, les bibliothèques WiFi sont disponible par défaut lorsqu'on installe l'environnement Arduino pour la

plateforme ESP32.
Pour ce TP, et pour éviter d'utiliser vos identifiants universitaires pour vous connecter à un réseau comme eduroam, **on vous demande d'utiliser un smartphone en mode partage de connexion.**
Récupérer le code sur Moodle, modifier le code en mettant le SSID et le mot de passe correspondant à votre partage de connexion. Examinez la suite d'instructions, essentiellement dans la fonction setup(). Vous pouvez tester ce code tel quel. Si c'est concluant et que vous récupérez l'heure actuelle (affichage sur le terminal), intégrez dans ce code les instructions pour vos mesures analogique y compris le calcul de la moyenne glissante. Modifier le code pour **afficher, en début de ligne, avant les mesures et la moyenne, l'horodatage de votre dernière mesure** (pas la date, juste l'heure).

☑ Effectuez une copie d'écran du terminal (laissez défiler quelques mesures pendant que vous occultez le capteur), déposez la capture d'écran sur Moodle.

4 Bonus : Interrupteur à déclenchement optique

Vos tracés sur le tableur doivent vous permettre d'évaluer l'écart entre la moyenne glissante et la dernière mesure lorsqu'un phénomène survient (la LED vient d'être cachée ou découverte). Connaissant approximativement cet écart, modifier le code pour qu'en cas de baisse soudaine de la luminosité, la LED_BUILTIN s'allume, et qu'en

☑ cas d'augmentation soudaine de la luminosité, la LED_BUILTIN s'éteigne. S'il n'y a pas de pas de variations importantes, l'état de la LED doit rester stable (elle ne doit pas clignoter...). Montrer le résultat.