C5-160551-INFO
Objets Connectés :
Programmation Microcontrôleurs - TP5

©Bernard Besserer

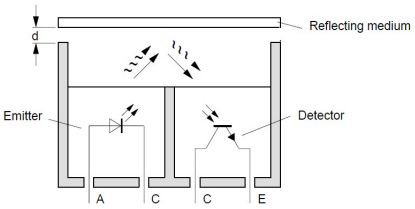
année universitaire 2021-2022

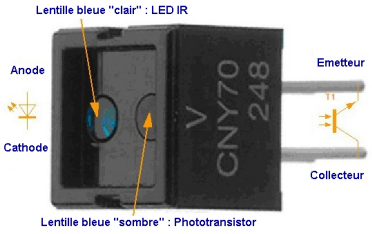
Ce qu'on va apprendre :
- Exploiter la veille profonde (*Deep Sleep*) du microcontrôleur.
- Sortir de veille de façon régulière grâce au timer.
- Récupérer une mesure à intervalles réguliers et transmettre cette mesure à un serveur (HTTP et/ou MQTT).

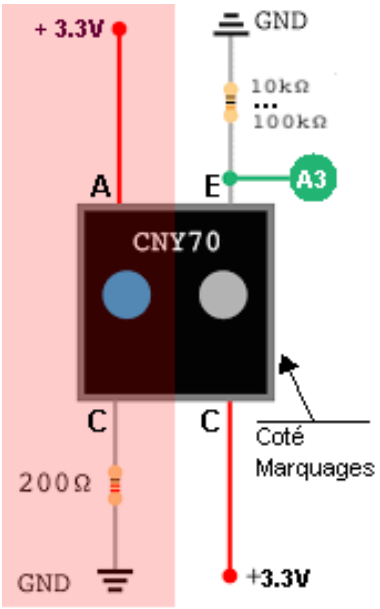
1 Câblage et vérification du fonctionnement des capteurs

Vous devez mettre en oeuvre un capteur de luminosité et un capteur de température. Il faut commencer par câbler et tester ces capteurs avant de transmettre les mesures vers le serveur de collecte.

1.1 Capteur de luminosité





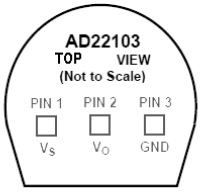


Vue de dessus

Nous allons utiliser un capteur de lumière beaucoup plus sensible et réactif que la LED détournée en tant que élément photovoltaïque du TP2. Le composant CNY70 intègre dans un boîtier une LED infrarouge et un photo-transistor, car il est destiné à détecter un objet par réflexion (quand l'objet est proche, la lumière émise par la LED est réfléchie par l'objet et vue par le photo-transistor). **Nous utiliseront QUE LA PARTIE PHOTOTRANSISTOR du CNY70, donc pas la partie en rouge**.
- Il faut placer le composant tel que les broches C (Collecteur) et E(Emetteur) soient sur des rails de connexions différents.
- Le coté C (Collecteur) est à relier au +3.3V, le coté E (Emetteur) est à relier à GND en passant par une résistance de l'ordre de quelques dizaines KΩ, une centaine de KΩ max. (3eme bague orange ou jaune)
- Du coté E (Emetteur) part un fil vers l'entrée analogique A3 de l'ESP32

1.2 Capteur de température

Comme capteur de température, nous utiliseront un capteur muni de son conditionneur intégré, qui peut s'alimenter sous 3.3V et donne une tension de sortie qui est une fonction **linéaire** de la température : le AD22103 de Analog Devices. Ce capteur est conditionné dans un petit boîtier noir type transistor, et nécessite 3 branchements. La vue représente le capteur **vu de dessus, les broches étant en-dessous**. Vous devez relier la broche VS au 3.3V de l'ESP32, et la broche GND du AD22103 au GND de l'ESP32. La sortie du capteur AD22103 (donc une tension proportionnelle à la température) **notée VO, sera à brancher sur l'entrée analogique A4**.



BOITIER VU DU DESSUS

Pour les deux capteurs, la lecture se fait avec analogRead(), voir code ci-dessous que vous pouvez tester, pour vous assurer que les deux capteurs réagissent.

```
void setup() {
  Serial.begin(115200);
  analogSetPinAttenuation(A3, ADC_0db);   // eventuellement, pour le capteur de luminosite
  // ne rien faire sur l'entrée A4 : 1V = valeur 1088
  }

float convert2Temp(int val)  {
  // a écrire, à la fin TP. Pour l'instant, la valeur est transmise sans modification
return ((float) val);
}

void loop() {
  int valA3 = analogRead(A3);    // resultat de la conversion A->N, correspondant a la tension sur A3
  int valA4 = analogRead(A4);    // resultat de la conversion A->N, correspondant a la tension sur A4

  float temp = convert2Temp(valA4);
  Serial.print("Temperature :");
  Serial.print(temp);
  Serial.print("   Luminosite : ");
  Serial.println(valA3);
  delay(1000);
}
```

A faire : Effectuer le câblage, tester le bon fonctionnement : il suffit de poser le doigt sur le capteur de température pour voir évoluer l'affichage, idem si l'on masque le capteur de luminosité.

2 Deep Sleep et timer

Comme vu en TD, pour mettre l'ESP32 en veille profonde, il suffit d'invoquer la fonction :

```
esp_deep_sleep_start();
```

Évidemment, si vous n'avez rien prévu pour le réveil, vous risquez d'attendre longtemps... Pour que la veille profonde soit interrompue après un temps donné, il faut appeler la fonction :
`esp_sleep_enable_timer_wakeup(time_to_sleep);` **AVANT** de rentrer en veille profonde.

La valeur time_to_sleep dépend de la fréquence de l'horloge et d'un éventuel *pre-scaler*. Par défault, en général, ce sont des micro-secondes. Il faut donc multiplier par un million pour avoir des valeurs en secondes.

La sortie de veille profonde est assimilée à un reset. En programmant l'ESP32 avec le framework Arduino, on repasse donc par la fonction setup(). A la fin de la fonction setup(),

A faire : Modifier votre code pour que les mesures de température et de luminosité se fassent toute les 10 secondes, avec une mise en veille profonde entre deux mesures. Vous pouvez déplacer entièrement le code dans le setup() ou en laisser une partie dans loop(). Après un affichage des valeurs sur la liaison série (Terminal), mettre un delay(100) pour laisser le temps à la transmission de se terminer avant d'entrer en veille.
Effectuer une mesure de la consommation électrique en utilisant les "testeurs USB" disponibles pour savoir si le mode *Deep Sleep* est bien effectif.
Modifier votre code pour que le sortie du mode Deep Sleep soit aussi possible par un appui sur un bouton-poussoir. On considère que le branchement adéquat d'un bouton-poussoir doit maintenant être maitrisée (cf. TP1 et TP3)
Attention, toute les broches ne conviennent pas pour la sortie de veille : *The ext0 wake up source option uses RTC GPIOs to wake up.* Vous avez donc le choix entre les GPIOs suivants : 0,2,4,12,13,14,15,25,26,27,32,33,34,35,36,39 (il s'agit des broches au niveau de la puce, à adapter selon la platine d'expérimentation que vous utilisez (par ex. Adafruit Huzzah 32) car toutes les broches ne sont pas forcement ramenés vers le bord de la platine. Dans le code, la notation doit être du type : `GPIO_NUM_X` *in which X represents the GPIO number of that pin.*

3 IOT : transmission d'information selon le protocole HTTP

3.1 Connection en tant que client HTTP

Commencez un nouveau projet pour tester la connexion de l'ESP32 sur un Access Point existant (soit un Access Point qui sera disponible dans la salle de TP, SSID = "OCPM", et pas de mot de passe, soit votre smartphone en mode "partage de connexion"). C'est le code qui a déjà été donné pour les requêtes auprès d'un serveur NTP. Si c'est un réseau ouvert et qu'il n'y a pas de mot de passe, la fonction `WiFi.begin` ne prend qu'un seul paramètre.

ATTENTION VOTRE MOT DE PASSE SERA EN CLAIR DANS LE CODE. Pensez à l'effacer si vous communiquez le code.

```
#include <WiFiUdp.h>
// #include <NTPClient.h>   // si requete NTP
#include <HTTPClient.h>

const char *networkName = "XXXXXXX";     // SSID de votre access point Wifi
const char *networkPswd = "YYYYYYY";

void setup()
{
Serial.begin(115200);

  //DO NOT TOUCH
  //  This is here to force the ESP32 to reset the WiFi and initialise correctly.
  WiFi.disconnect(true);
  delay(1000);
  WiFi.mode(WIFI_STA);
  delay(1000);
  // End silly stuff !!

  // connect to Access Point with password
  WiFi.begin(networkName, networkPswd);
  // connect to Open Access Point without password
  // WiFi.begin(networkName);

  Serial.printf("Try connexion to %s\n", networkName);
  long signalStrength;
  while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
        signalStrength = WiFi.RSSI();
        Serial.printf("RSSi: %ld dBm\n",signalStrength);
    }

  // now connected to WiFi, got IP from DHCP
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

void loop()
{
}
```

A faire : Rendez fonctionnel et testez le code élémentaire donné ci-dessous. Vous pouvez utiliser delay() et vous n'utiliserez pas de mode de veille, tant que votre programme n'est pas fonctionnel. **Vous remplacerez NNNNNN par l'un des nom du binôme (nom de famille, majuscules, pas de caractères accentués ni d'espace)**

```
char request[] = "http://192.168.101.100/collecte/test.php"  // ou alors collecte.php
HTTPClient http;

// mettre ici votre code pour completer la chaine de caractères request, rajoutez au moins name=NNNNNNN

http.begin(request);
int httpResponseCode = http.GET();   // send request
```

Vérifiez sur le La machine faisant office de serveur de collecte que votre requête apparaît dans le fichier de log. (ou pour une requete en passant par "l'extérieur" : `https://bbessere.lpmiaw.univ-lr.fr/collecte/test.php`).

3.2 Référence temporelle : synchronisation sur un serveur NTP

Déjà proposé lors des TP2 & TP3. Si vous vous connectez sur le réseau ouvert OCPM, la machine accessible via le réseau OCPM dispose d'un serveur NTP sur son adresse 192.168.101.100. Pour votre requête, l'information temporelle devra être sous la forme d'une paire (clé,valeur) : time=HHHHHHH, avec HHHHHHH l'heure correctement formatée (HH :MM :SS, voir méthode getFormattedTime() de la classe NTPClient)).

3.3 Application complète

A faire : Intégrer le code dans une architecture permettant la mise en veille profonde (on peut tout mettre dans la fonction `setup`). Après un reset, donc une sortie de DeepSleep :

1. Tentative de connexion à votre AccessPoint.
2. Si OK, connexion au serveur NTP, synchronisation de l'heure.
3. Récupération des deux mesures (luminosité et température).
4. Formatage des chaînes de caractères pour obtenir une requete conforme et envoi sous forme d'une requete GET au serveur qui va collecter les mesures. N'oubliez pas de mettre le caractère de séparation & entre les deux paires clé=valeur. Verifiez sur le serveur de collecte de données.

si vous utilisez la classe HTTPClient, en cas de requêtes successives, pensez à utiliser la méthode end() sur l'instance de HTTPClient entre deux requêtes successive.

5. Puis mise en veille profonde pendant 10 secondes.

Les données formatées et la requête (requête GET) devra correspondre EXACTEMENT au format suivant :
`name=XXXXXX` : un nom du binome (pas le prénom) en majuscules, non accentuées. Pas d'espace.
`time=HH:MM:SS` : chaîne de caractère représentant l'horodatage de la mesure.
`temp=TTTT` : chaîne de caractère représentant une valeur entière, issu de la lecture de la voie analogique A4
`lum=LLLL` : chaîne de caractère représentant une valeur entière, issu de la lecture de la voie analogique A3

Requête correcte : `http://XXX.XXX.XXX.XXX/collecte/collecte.php?name=DUPONT&time=10:33:42&temp=2013&lum=1247`

4 Mise à l'échelle de la valeur issue du capteur de température

La mesure d'une température peut se faire de plusieurs manières : en utilisant des capteurs à sonde de platine dont la résistance croit de manière linéaire avec la température, des thermistances à coefficient de température négatif (mais dont la réponse n'est pas linéaire), des capteurs avec interface numérique comme le DS1820... Le composant AD22103 de Analog Devices utilisé pour ce TP dispose d'un conditionneur intégré donne une tension de sortie qui est une fonction **linéaire** de la température. Si alimentation sous 3.3V alors : $V_{OUT} = 0.25V + P \times T_A$
T_A est la température ambiante, et V_{OUT} la tension délivrée par le capteur sur sa broche Vo. P est un coefficient de proportionnalité et vaut $0.028V \text{par} \quad ^\circ\text{C}$

Dans le code de référence donné, le résultat de la conversion numérique analogique est transmise directement au serveur ; la fonction `convert2Temp` ne fait rien :

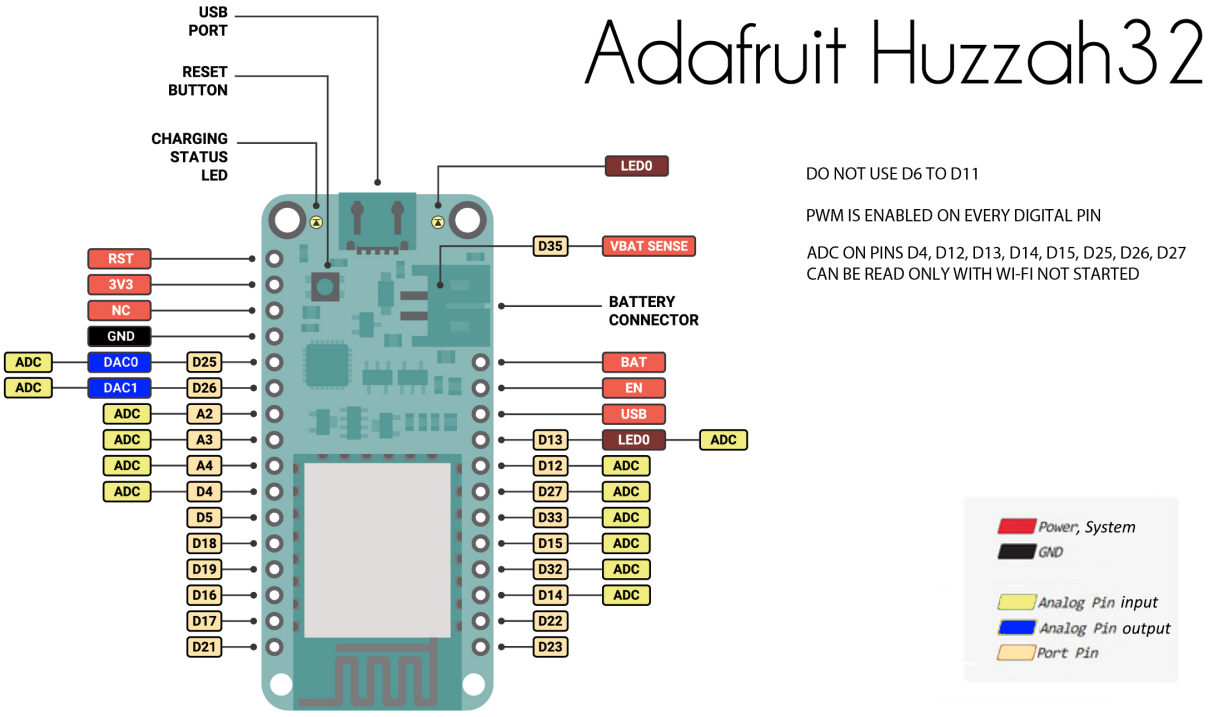
```
float convert2Temp(int val)
{
return ((float) val);
}
```

Connaissant la formule $V_{OUT} = f(T_A)$, et sachant que pour l'ESP32 dans le réglage de base, si 1V est appliqué sur une entrée analogique, alors le résultat de la conversion par analogRead() donne la valeur 1088 [1], écrire une fonction `convert2Temp` qui permet d'afficher et de transmettre une valeur en degrés Celcius. Limitez la valeur de la température à deux décimales, par exemple 19.45

Modifiez la construction de votre requête pour transmettre une valeur réelle (nombre avec virgule) pour la température. Une requête correcte est alors :

`http://XXX.XXX.XXX.XXX/collecte/collecte.php?name=DUPONT&time=12:26:16&temp=19.25&lum=2480`

Laissez votre capteur connecté transmettre vers le serveur de collecte quelques mesures dans ce nouveau format.



1. donc 3590 pour 3.3V sachant que le maximum théorique du convertisseur sur 12 bits est 4095