C5-160551-INFO
Objets Connectés :
Programmation Microcontrôleurs - TP6

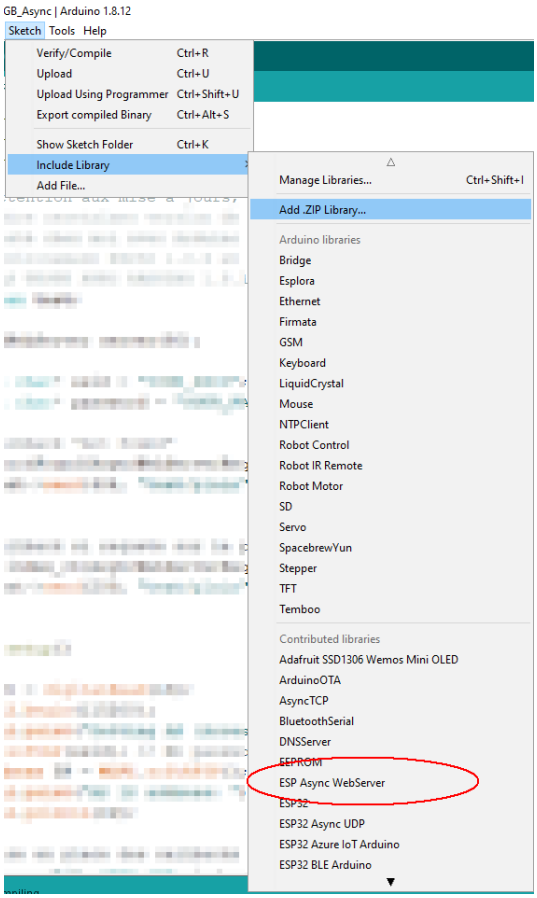
©Bernard Besserer

année universitaire 2021-2022

1 Contrôle d'une LED simple

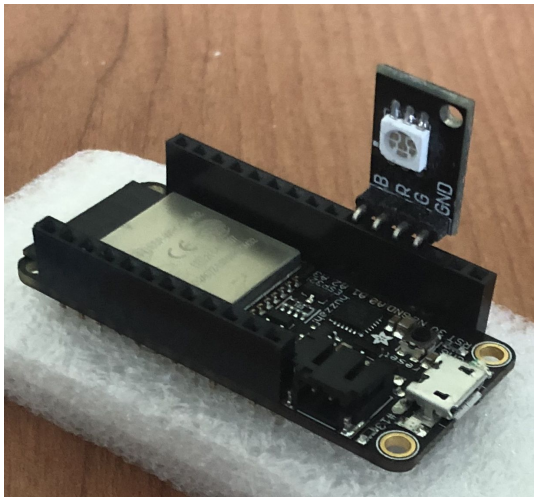
- Mettre l'ESP32 en tant que point d'accès Wifi (HotSpot) avec un SSID correspondant à l'un des noms de famille du binôme, en majuscule,
- Faire tourner un serveur HTTP sur l'ESP32, permettant à l'aide de deux boutons d'allumer ou d'éteindre la LED (LED_BUILTIN). L'action sur chaque bouton doit recharger la page HTML affichée et le texte affiché doit refléter l'état de la LED (avec un texte "la LED est allumée" ou "La LED est eteinte"). Montrer le fonctionnement.

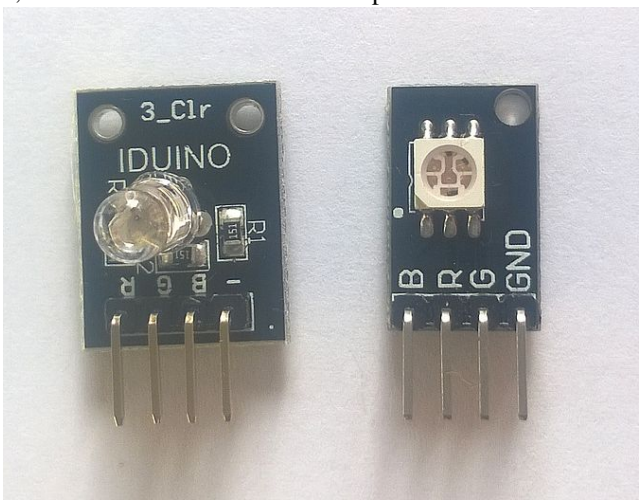
Pour la réalisation, reprenez les pages HTML réalisés lors du TEA, et comme serveur HTTP, vous pouvez utiliser le serveur WEB "standard" du package Arduino pour l'ESP32 (objet WebServer), en partant du code indiqué à la section 4 du TP5 (passage en mode automatique de l'asservissement) mais voir aussi plusieurs exemples qui sont disponibles ... voir copie d'écran ci-dessous, à gauche.
Vous pouvez aussi utiliser la bibliothèque plus performante nommée AsyncWebServer de Hristo Gochkov :
`github.com/me-no-dev/ESPAsyncWebServer`
Helas, celle-ci ne s'installe plus via le gestionnaire de bibliothèque, et le site GitHub n'indique que la procédure d'installation pour l'environnement PlateformeIO. Toutefois, en téléchargeant le ZIP depuis le site GitHub, la bibliothèque (et la dépendance nécessaire AsyncTCP) peut s'installer en passant par la commande "Add Zip Library", voir ci dessous à droite (à remarquer, la bibliothèque est déjà installée ici !).

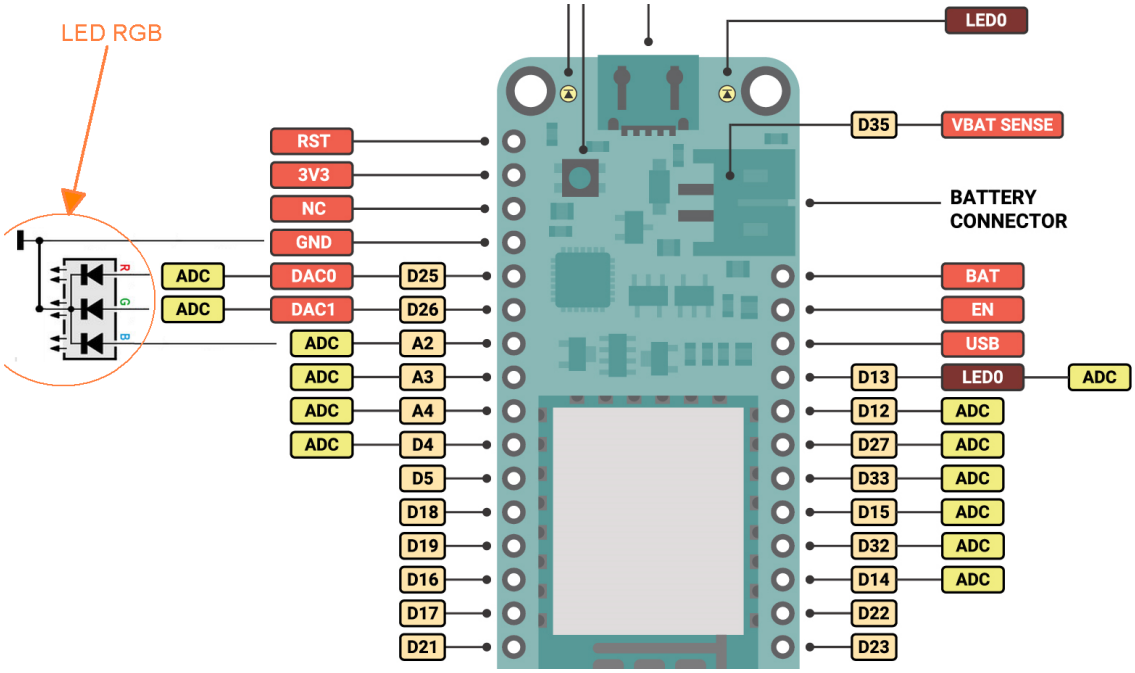


2 Commande d'une LED RGB

La LED RGB dispose de 4 broches (**R**ed, **G**reen, **B**lue et une masse commune) et **les résistances de protection sont intégrées**. On peut éviter le câblage et brancher la LED sur broches A2, D25, D26 et GND de la platine ESP32 (voir photos et schéma. Vérifier que la broche GND de la LED RGB est bien connectée au GND). Il y deux modèles de LED, mais dans les deux cas, le GND est au même endroit, mais les canaux couleur ne sont pas dans le même ordre.







2.1 Utilisation de la LED RGB sans bibliothèque, avec 2 canaux

Vérifiez si, a part la LED placée comme illustré sur la photo, vous n'avez pas d'autres connexions sur D25, D26 et A2. Brancher la LED à cet endroit permet d'utiliser les deux canaux DAC (Digital to Analog Converter) dont dispose l'ESP32, accessible sur D25 et D26, donc les canaux B et G si vous avez le type de la LED de gauche (donc Bleu, Vert et Cyan si mélange) et les canaux R et G si vous avez le type de la LED de droite (Rouge, Vert et Jaune si mélange).

1. En utilisant le branchement direct comme sur la photo, et sans utiliser les DAC, pourra-t'on allumer ou éteindre les 3 canaux de couleur (uniquement a l'état HIGH et LOW). On rappelle qu'il y a des limitation concernant les broches :
   GPIOs 34 to 39 are GPIs - input only pins. These pins don't have internal pull-ups or pull-down resistors. They can't be used as outputs, so use these pins only as inputs.
   Et sachant que la broche marquée "A2" est la broche 34 de l'ESP32
2. En utilisant le branchement direct comme sur la photo, écrivez un code dans le setup() qui va commuter les canaux de couleur en mode "tout ou rien", donc **sans modulation d'intensité** (couleur 1, puis couleur 1+2, puis couleur 2, puis extinction pour tester que tout est OK). Utilisez simplement des commandes digitalWrite() et delay() (pas de mise en veille ici). Montrez cette séquence.

3. Les broches notées DAC (Digital To Analog Converter) peuvent directement moduler la tension sur une broche en PWM :
   ESP32 has two 8-bit DAC (digital to analog converter) channels, connected to GPIO25 (Channel 1) and GPIO26 (Channel 2). The DAC driver allows these channels to be set to arbitrary voltages.

   Reprendre l'exercice précédent en modulant graduellement la puissance (on commence LEDs eteintes, donc noir, allumage progressif du canal couleur 1, puis allumage progressif du canal couleur 2 pour avoir le mélange, puis extinction progressive du canal couleur 1, puis extinction progressive du canal couleur 2, puis pause. Utilisation de boucles, fonctions dacWrite() et delay(). Laisser ce code dans setup(), il permet de tester le bon fonctionnement au démarrage. Montrer la séquence qui doit durer quelques secondes maximum.

   La commande du DAC est :
   ```
   dacWrite(int broche, int valeur)   // DAC 8 bits, donc valeur entre 0 et ...
   ```

4. Écrire une fonction `int GetRfromHTML(char* ColorCode)`. Le paramètre passé à la fonction sera une chaîne de caractère telle qu'utilisé pour coder les couleurs dans les langages HTML et CSS, et la fonction doit renvoyer la valeur du rouge, entre 0 et 255.

   Décliner cette fonction en GetGfromHTML(char* ColorCode) et GetBfromHTML(char* ColorCode). Tester cette fonction avec char * color = "FF0000" qui doit donner la valeur maximale pour le rouge, "00FF00" du vert et "FFFF00" du jaune (ou autre mélange, selon la LED RGB dont vous disposez). Si la chaîne débute avec le caractère #, ça doit également fonctionner.

2.2 Utilisation de la LED RGB avec bibliothèque, avec 3 canaux

Il existe plusieurs bibliothèques qui implémentent une modulation PWM sur n'importe quelle broche de type GPIO (de façon logicielle, en mettant cette broche à l'état HIGH, puis LOW, etc...). Évidemment, contrairement à l'utilisation des canaux PWM hardware du DAC que nous avons exploité précédemment, l'utilisation de la bibliothèque logicielle sollicite le CPU.

Voir les exemples disponibles dans l'IDE Arduino (Fichiers -> Exemples), concernant le pilotage d'une LED RGB.

Si ce n'est déjà fait, installez une bibliothèque de ce type et modifiez le code pour piloter les 3 canaux couleur. On utilise toujours le branchement direct comme sur la photo ; 2 canaux seront donc pilotés via les broches D25 et D26, et le 3eme canal sur une broche modulée en PWM via la classe/bibliothèque dédiée.

Pour faire cela, il faut câbler un fil allant d'un GPIO éligible vers l'entrée A2
Pas de risque de court-circuit, A2 étant uniquement une entrée.

Voir aussi cette page : `https://techtutorialsx.com/2017/06/15/esp32-arduino-led-pwm-fading/`

Dans le setup(), vous gardez votre séquence de test, qui va maintenant passer par toute les couleurs possibles (canal 1, puis 1+2, puis 2, puis 2+3, puis 3, puis 3+1, puis 1 puis extinction), que vous montrerez à l'intervenant. Ensuite, mettre en place votre serveur Web doit envoyer la page d'interface avec la saisie de couleur, récupérer la requête, et votre LED devra afficher la couleur sélectionnée.

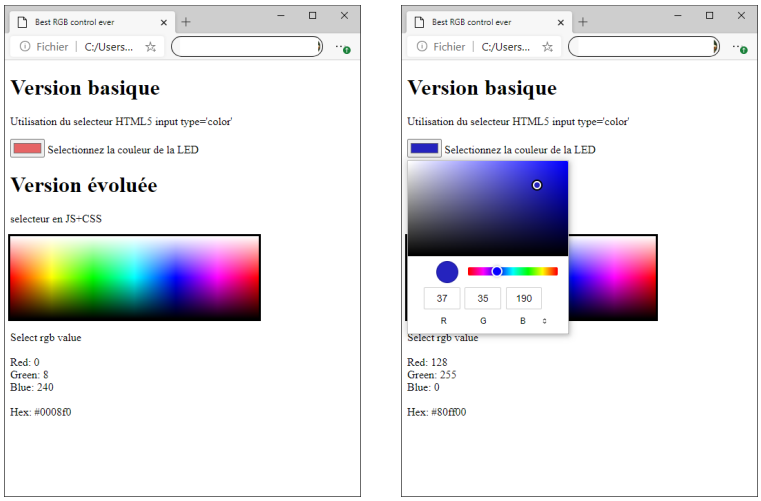
2.3 Page HTML représentant l'interface

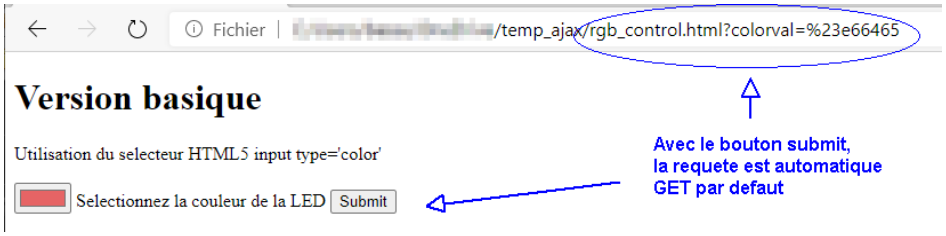
On va passer au contrôle via le smartphone. Combinez avec le code que vous avez produit pour la section 2.2 et utiliser les pages HTML produites dans le cadre du TEA :
- Interface minimale, utilisation du type de base du HTML5
- Interface évoluée, avec une plage de sélection ou une roue de sélection de couleur

L'interface minimale consiste a utiliser mettre une élément `input type=''color''`. Un rectangle s'affiche de la couleur sélectionnée et un clic sur ce rectangle ouvre le sélecteur de couleur (l'apparence du sélecteur de couleur dépends du navigateur et de la plate-forme, Android ou iOS). Mais de toute façon, vous récupérez toujours un code couleur au format hexadecimal, qui doit être transmis via la requete au serveur.

Du coté serveur, il faut récupérer cette chaîne de caractère et piloter en conséquence la LED RGB

Si vous pouvez placer l'objet `input` dans un objet `form` avec un bouton submit, alors c'est une requête GET ou POST qui est émise, avec le code hexadécimal de la couleur. Mais cette requête provoque aussi un rafraîchissement de la page. Vous pouvez ajouter une option pour ne pas provoquer le rafraîchissement ou transmettre avec une requête asynchrone sans rafraîchissement. C'est quasi nécessaire pour utiliser le sélecteur de couleur "version évoluée", la requête est émise au clic dans cette zone de couleur (donc mise en place d'une écouteur JS, etc...)





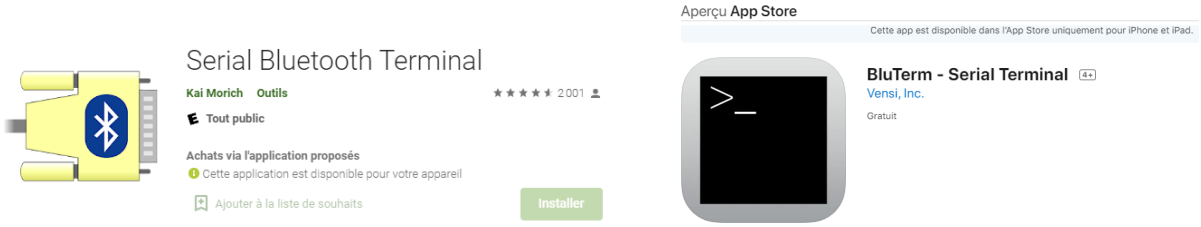
3 Utilisation du SPIFFS

On vous pouvez procéder selon vos préférences pour le "stockage" de vos contenus WEB :
- Soit encoder les fichiers comme des RAW STRING (voir TD), idéalement dans des fichiers .h, ce qui peut convenir pour des pages simple comme au 1
- Soit, si les fichiers deviennent multiples ou encombrants, utiliser le système de fichier SPIFFS comme indiqué en TD.

Montrez le résultat, et, pour la version avec la roue de couleur, déposez sur moodle le projet sous forme d'archive ZIP (contenant le .ino, les eventuels .h et les fichiers .html, .js, .css)

4 Controle de la LED RGB avec le Bluetooth

Nous avons vu en cours que de nombreux protocoles peuvent être utilisés au dessus de la couche physique BT. La transmission série fait partie de ces protocoles (RFCOMM).
Sur vos smartphone Android, télécharger l'application Serial Bluetooth Terminal. N'ayant pas accès à l'écosysteme Apple, je n'ai rien pu tester, mais je suppose que l'application BluTerm remplit ce même rôle.



Le code pour tester la communication BT, du coté de l'ESP32, est très simple, il suffit de prendre le code SerialToSerialBT donné en exemple (File -> exemples -> BluetoothSerial
Sauvegardez ce code sous un nouveau nom, modifier le code en donnant un *Bluetooth Device Name* correspondant à l'un des noms du binôme, en majuscules, afin d'éviter les conflits dans la salle.

Compilez, téléversez et testez. Après avoir apparié votre ESP32 en mode BT, vous devez pouvoir dialoguer : les messages saisis sur le moniteur série (port COM du PC) sont redirigé vers l'application du smartphone via BT, et vice-versa.
On ne peut pas, dans le cadre limité de cette UE, développer une Application complete Android ou iOS native (ou même en mode hybride) intégrant une interface de type roue des couleurs et la communcation BT. On fera donc simple,

en utilisant les Applications précédemment cités, avec une interface en mode texte.
Modifiez le code en y intégrant le controle de la LED RGB (mais ni la création du Hotspot, ni le serveur HTTP). Il faut donc pouvoir contrôler la LED par BT en envoyant un code couleur au format HTML (saisie de #FF0000 dans votre application doit allumer la LED en rouge)

ANNEXE : Informations relatives à l'usage de AsyncWebServer

Dans ce modèle, on précise au serveur quelle action a faire après telle ou telle requete en associant des fonctions callback à ces requetes (des fonctions qui seront appeles lorsque la requete se présente). La syntaxe est :
`server.on("/", HTTP_GET, [](AsyncWebServerRequest *request)`
1er parametere : const char* uri : l'adresse cible de la requete
2eme parametre : WebRequestMethodComposite method : La mthode de la requete. Dans notre cas ce sera la constante HTTP_GET
3eme parametre : ArRequestHandlerFunction onRequest : La fonction callback mise ne place.

Attention, dans la plupart des exemples d'utilisation de AsyncWebServer, des expressions lambda sont utilisés pour définir ces callbacks, ou celles-ci sont déclarés *inline*. Mais on peut aussi ecrire les fonctions callback de façon "'classique"' comme-ci apres, pour la fonction notfound.
Vous pouvez aussi consulter les exemples sur techtutorialsx, par exemple : `techtutorialsx.com/2018/01/15/esp32-arduino-http-server-route-not-found-handling/`

```
#include <WiFi.h>
#include <AsyncTCP.h>
#include <ESPAsyncWebServer.h>

AsyncWebServer server(80);

const char* ssid = "YOUR_SSID";
const char* password = "YOUR_PASSWORD";

void notFound(AsyncWebServerRequest *request) {
    request->send(404, "text/plain", "Not found");
}

void setup() {
    Serial.begin(115200);
    WiFi.mode(WIFI_AP);
    WiFi.begin(ssid, password);
    if (WiFi.waitForConnectResult() != WL_CONNECTED) {
        Serial.printf("WiFi Failed!\n");
        return;
    }

    Serial.print("IP Address: ");
    Serial.println(WiFi.localIP());

    server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
        request->send(200, "text/plain", "Hello, world");
    });

    server.onNotFound(notFound);
    server.begin();
}

void loop() {
}
```