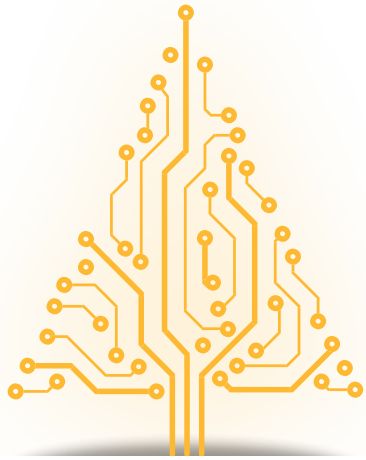


guirlandes de LED avec ESP32 et FreeRTOS

clignotantes et scintillantes

Serge Sussel (France)

Pour les jours de fête et les soirées, des rubans de LED en 24 V CC qui clignotent et scintillent peuvent être un régal pour les yeux. Il suffit d'un ESP32 pour piloter un système complet avec 13 paramètres différents. Avec FreeRTOS, l'exécution simultanée de plusieurs tâches est possible.

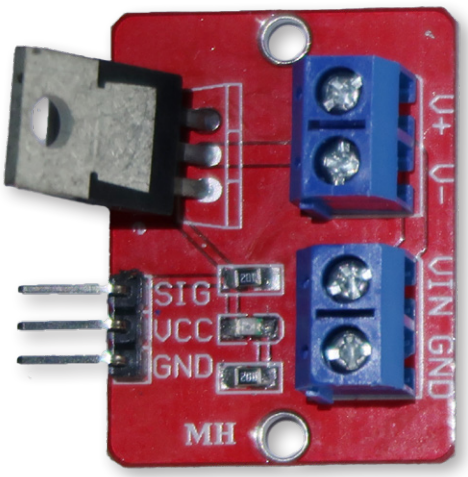


Figure 1. Module à FET.

Pour les fêtes de fin d'année, je voulais décorer et illuminer le sapin de Noël. J'avais de très vieilles guirlandes lumineuses avec des ampoules à incandescence en série, chacune encapsulée dans un petit bulbe en plastique coloré. Pour donner vie à ce chapelet de lampes, un rudimentaire interrupteur thermique était connecté entre le secteur et les lampes. De plus, la faible longueur de la guirlande limitait les possibilités d'animation de l'arbre.

J'ai donc cherché sur le web un site commercial proposant des guirlandes utilisant la technologie LED, qui consomme moins d'énergie et ne chauffe pas autant. J'en ai trouvé un (Lumitronix) et j'ai opté pour l'achat de deux rubans de LED multicolores beaucoup plus longs avec les transformateurs adéquats. C'est un système sous 24 V avec un redresseur qui peut alimenter plusieurs rubans de LED. Seulement, l'allumage des rubans multicolores reste fixe, sans animation - un peu triste pour des illuminations de sapin de Noël.

J'ai également rénové mon ancienne guirlande en remplaçant les ampoules à filament par des LED de couleur identique aux bulbes en plastique, rallongé les fils entre chaque lampe, et fait l'adaptation pour 24 V grâce à une résistance de limitation de courant montée en série avec les LED dans le connecteur. Cela m'a permis de produire trois rubans lumineux.

Enfin, pour animer le tout, j'ai pensé à programmer et utiliser un microcontrôleur, et piloter les rubans avec une interface à MOSFET pour faire clignoter ou scintiller les LED. Je me suis également instruit en lisant des livres sur le C/C++ et sur l'Arduino pour me former et apprendre de nouvelles choses.

Au début de ce projet, je me suis demandé comment utiliser plusieurs sorties MLI simultanément, mais de manière asynchrone, afin d'avoir une animation différente pour chaque ruban. Je pensais développer un automate à états finis pour cela, mais je trouvais cela un peu compliqué.

J'ai donc commencé le projet avec un Arduino Nano et une programma-

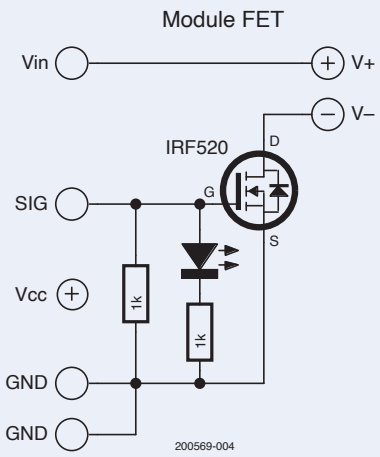


Figure 2. Schéma du module à FET.

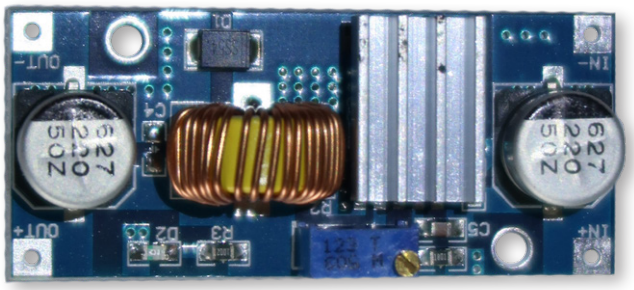


Figure 3. Convertisseur CC/CC pour obtenir une sortie de 5 V.

tion simple qui pilotait une seule sortie MLI avec l'interface à MOSFET. J'ai ensuite dupliqué tout cela afin de pouvoir piloter deux guirlandes.

Comme j'avais trois rubans, le troisième avait la même animation que l'un des deux autres. Tout en testant le projet, j'ai aussi amélioré les paramètres d'animation du programme à plusieurs reprises afin d'améliorer le rendu visuel.

Le projet ESP32

En découvrant et en lisant les articles de Warren Gay dans Elektor sur FreeRTOS (www.elektormagazine.com/warrengay), ainsi qu'après l'acquisition de son livre sur ce sujet (www.elektor.fr/19341), j'ai essayé d'adapter FreeRTOS à l'Arduino Nano, mais j'ai rapidement trouvé ses limites. Après plusieurs essais, je n'ai pas pu exécuter plus d'une tâche RTOS à la fois sur le Nano. Il y avait sûrement des optimisations à faire.

Je me suis donc tourné vers l'ESP32 afin de porter mon projet sur cette plateforme et d'implémenter FreeRTOS et des tâches asynchrones indépendantes. J'ai commencé par y écrire une tâche, et découvert les différences entre ce microcontrôleur (ESP32) et l'Arduino Nano en matière de programmation C/C++.

Après plusieurs itérations de programmation et de test, je n'avais plus d'erreurs de compilation, et la tâche fonctionnait. Avec mon oscilloscope sur la broche de sortie MLI, j'obtenais les signaux voulus. Et, avec FreeRTOS, si on n'utilise pas de mécanisme d'attente et synchronisation, chaque tâche est indépendante des autres. C'est exactement ce que je voulais faire.

Ainsi, l'animation de chaque ruban est tirée au sort aléatoirement. Néanmoins, même dans les rares cas avec deux animations identiques au même moment, elles ne commenceront pas en même temps en raison des décalages des animations précédentes.

J'ai pu apprécier la puissance de l'ESP32 et, après plusieurs essais et tests, j'ai réussi à créer deux, puis trois, puis quatre tâches qui s'exécutent simultanément. J'ai donc utilisé FreeRTOS sur l'ESP32 pour piloter quatre sorties via quatre tâches. L'ESP32 est cependant capable de piloter plus de sorties grâce à ses autres broches MLI disponibles.

J'utilise les sorties MLI pour faire varier progressivement la luminosité des rubans de LED, mais aussi pour les allumer et les éteindre, entre autres paramètres d'animation. J'ai monté mon projet sur une carte à trous et j'ai tout mis dans un boîtier en plastique. J'ai également posté ce projet sur elektormagazine.fr/labs.

Composants nécessaires

Pour trouver des composants pour mon projet, j'ai consulté l'Internet d'Extrême-Orient et j'ai trouvé de petits modules tous équipés de MOSFET, de résistances et de LED. En revanche, les délais de livraison étaient plutôt en semaines qu'en jours. J'ai donc procédé à la rétro-ingénierie du module (voir la **figure 1** et le schéma de la **figure 2**).

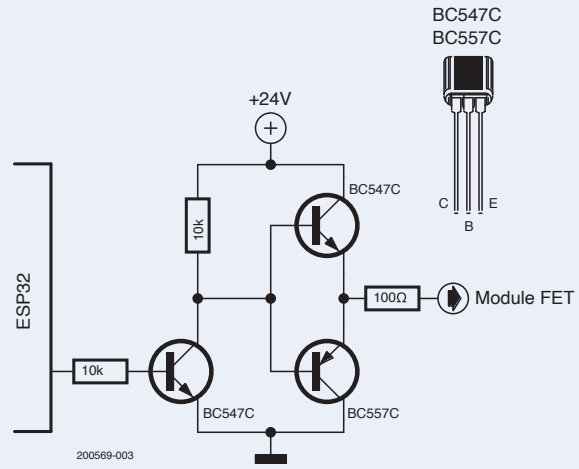


Figure 4. Pilote pour MOSFET.

Listage1. 13 jeux de paramètres pour les animations.

```
#define MAXPROGR 13   // Nombre maximum de
           //programmes LED (0 à MAXPROGR -1)

// structure pour les programmes LED
 struct  pled_t    /* structure def */
 {
     int ledvar;     /* 0 : clignotement ; 1 :
                          luminosité variable */
     int ledon;      /* durée d'allumage */
     int ledoff;     /* durée d'extinction*/
     int ledvaloff;  /* valeur pour l'état
                                  d'extinction */
     int ledxtime;   /* répéter ce programme x fois
                                                */
 };
pled_t pled [MAXPROGR];  /* mémoire réservée pour
               les paramètres des programmes LED */


...


// initialisation des paramètres des programmes LED

     pled[0].ledvar = 0;
     pled[0].ledon = 750;
     pled[0].ledoff = 900;
     pled[0].ledvaloff = 18;
     pled[0].ledxtime = 7;

     pled[1].ledvar = 0;
     pled[1].ledon = 1100;
     pled[1].ledoff = 800;
     pled[1].ledvaloff = 18;
     pled[1].ledxtime = 8;

...

     pled[5].ledvar = 1;
     pled[5].ledon = 1024;
     pled[5].ledoff = 1024;
     pled[5].ledvaloff = 0;
     pled[5].ledxtime = 6;

     pled[6].ledvar = 1;
     pled[6].ledon = 1280;
     pled[6].ledoff = 1280;
     pled[6].ledvaloff = 0;
     pled[6].ledxtime = 5;

...
     pled[12].ledvar = 1;
     pled[12].ledon = 1024;
     pled[12].ledoff = 512;
     pled[12].ledvaloff = 0;
     pled[12].ledxtime = 8;
```

Listage 2. Mode variable pour l'allumage et l'extinction en fondu.

```
// valeurs pour chaque étape dans le programme
                              // variable pour LED,
// 32 pas pour OFF vers ON et pareil pour ON vers
                                          // OFF,
// Valeurs du demi-sinus
 const int varval[] = { 0, 10, 25, 35, 45, 60,
75, 90, 100, 112, 122, 134, 145, 155, 165, 175,
184, 192, 200, 208, 215, 220, 226, 232, 236, 239,
241, 244, 247, 250, 253, 255 };
```

Listage 3. Définitions pour le matériel.

```
// Définition des types utilisateur
 // 4 guirlandes pour ce projet
 #define GUIRL_A  16 // GPIO pour la guirlande A
                     // MLI numérique - Task1Led
 #define GUIRL_B  17 // GPIO pour la guirlande B
                     // MLI numérique - Task2Led
 #define GUIRL_C  18 // GPIO pour la guirlande C
                     // MLI numérique - Task3Led
 #define GUIRL_D  19 // GPIO pour la guirlande D
                     // MLI numérique - Task4Led


 ...


// Réglage des propriétés du MLI ESP32
 const int freqpwm = 5000;    // Fréquence en Hz
 const int resolution = 8;    // 8 bits
 const int ledChannelA = 0;   // Canal
                              // de chaque
                                    // guirlande
 const int ledChannelB = 1;   //
 const int ledChannelC = 2;   //
 const int ledChannelD = 3;   //
```

Listages 4. Tâche se supprimant elle-même.

```
void loop()
{
 // Se supprime elle-même, inutilisée
  vTaskDelete(nullptr);
} // fin de la boucle
```

De même, j'ai trouvé un module d'alimentation entièrement assemblé et réglable acceptant une entrée de 24 V CC et, après avoir ajusté le potentiomètre multi-tours, il a fourni la tension de sortie appropriée (5 V) pour le microcontrôleur (**figure 3**).

Par ailleurs, dans un article d'Elektor, j'ai trouvé un montage intéressant pour piloter le module MOSFET, basé sur trois transistors - un couple de BC547 et un BC557 - et leurs résistances respectives (voir **figure 4**) pour améliorer les transitions sur la grille du MOSFET. Cependant, cela inverse le signal, il faut donc en tenir compte lors de la programmation. Encore une autre commande à passer et à attendre patiemment son arrivée.

Avec tous les composants enfin en main, j'ai pu assembler et effectuer les tests finaux. Heureusement, les modules à MOSFET ont été livrés par lot de 5 car je me suis retrouvé avec juste le corps d'un transistor MOSFET entre les doigts. Il a dû être plié plusieurs fois, ce qui a affaibli ses broches. Il m'a fallu un peu de rafistolage pour mettre fin au début de rupture des broches.

Ainsi, en sortie de l'ESP32 pour chaque guirlande, j'ai un module d'interface avec ses transistors BC547 et BC557 pour piloter chaque module à MOSFET, qui peut piloter un ou plusieurs rubans lumineux à LED (voir **figure 5**). J'ai tout enfermé dans un boîtier en plastique et j'ai monté des connecteurs pour l'entrée de tension CC et les sorties pour les rubans lumineux.

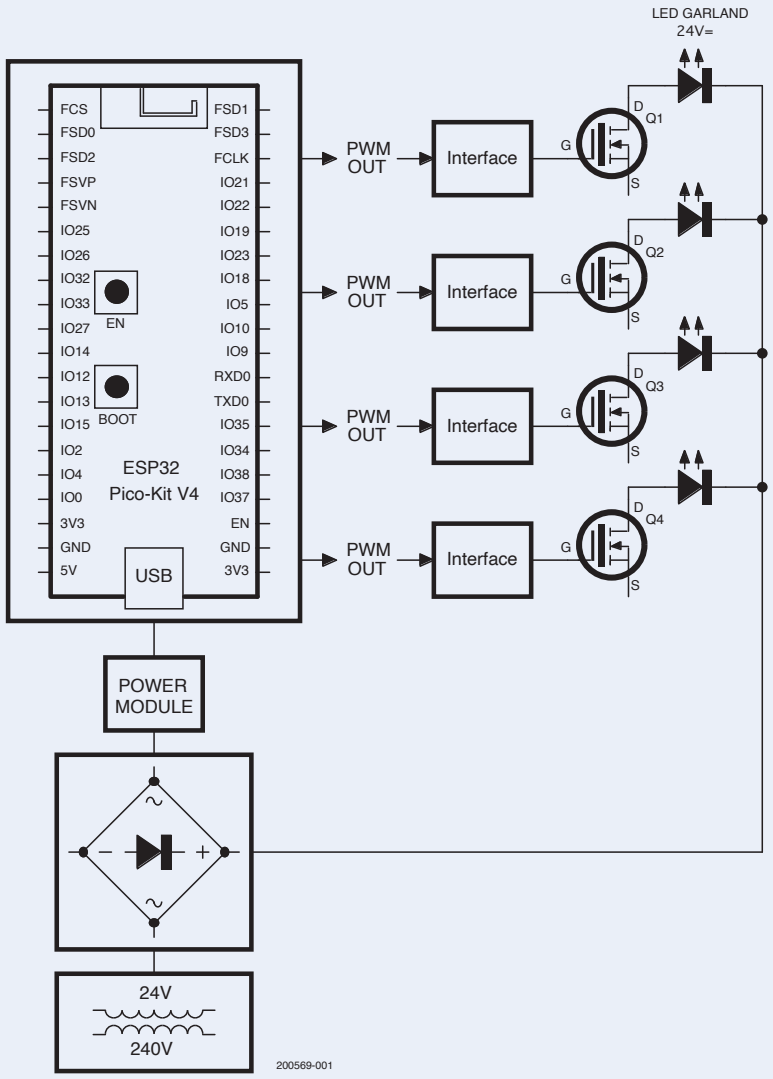


Figure 5. Schéma avec ESP32 et MOSFET.

Le code d'animation de l'ESP32

J'ai utilisé l'Arduino IDE pour développer, compiler et télécharger le programme [1] vers le microcontrôleur. Venons-en maintenant au cœur du projet, à savoir l'animation des LED. J'ai choisi de décrire les animations par un ensemble de paramètres définis dans une table contenue dans une structure(`struct`). Il y a 13 entrées dans cette table (voir **listage 1** pour la structure (`struct`) utilisée et les valeurs pour l'initialiser). On peut toutefois étendre cette table avec plus d'entrées.

J'ai défini deux types d'animations. Le premier type est une animation ON/OFF avec une option supplémentaire de clarté résiduelle en mode OFF. Chaque sortie a une durée d'allumage, une durée d'extinction et le nombre d'itérations à effectuer pour chaque cycle d'allumage/extinction. Le second type est le mode variable. Pour cela, j'ai créé un tableau de 32 éléments. Il s'agit de valeurs sinusoïdales permettant d'allumer et d'éteindre progressivement les LED (voir **listage 2**). La durée du cycle est commandée par les durées d'allumage et d'extinction dans la table de paramètres.

À la fin de ces itérations, le programme charge un autre jeu de paramètres de la table, tiré au sort avec un nombre aléatoire utilisé pour le tableau de paramètres. Si le nombre tiré est le même que le précédent, le programme refait le tirage, afin de ne pas avoir deux fois de suite la même animation pour la même guirlande. Cela améliore l'esthétique de l'arbre.

Résumé des sections du code

Au début du programme, je définis les sorties MLI utilisées et les constantes pour l'ESP32 (**listage 3**). Vient ensuite la fonction utilisée pour avoir un nombre pseudo-aléatoire (lignes 61-67 du code [1]), puis le code de la tâche RTOS (lignes 72-145), qui sera exécutée indéfiniment. Cette tâche est réutilisable, car elle est définie avec un paramètre (le numéro de la broche MLI) qui lui est passé.

Vient ensuite la fonction `setup()`, qui commence par assigner toutes les valeurs à la table des paramètres d'animation mentionnée plus haut. La section suivante gère les paramètres de sortie MLI (lignes 231-245), et la création des tâches FreeRTOS, avec leurs paramètres, pour les

activer (lignes 251-315). Dans l'ESP32, l'ordonnanceur se lance tout seul, et démarre les tâches dès qu'elles sont déclarées.

Enfin, dans la fonction `loop()`, qui est elle-même une tâche, nous la laissons se supprimer, car elle n'exécute aucun code - une tâche peut se supprimer et libérer ses ressources. Ces lignes sont présentées dans le **listage 4**. Et la féerie prend vie ! (**figure 6**)

Pour aller plus loin

Grâce à ses multiples sorties MLI, l'ESP32 peut piloter d'autres guirlandes. On peut imaginer utiliser toutes les sorties MLI pour commander des guirlandes afin d'animer un jardin ou une façade de maison. Une autre possibilité serait d'utiliser le wifi disponible sur la plateforme pour commander l'ESP32 depuis un téléphone portable via une petite interface que l'on peut développer. Place aux suggestions.

On peut trouver tout le matériel pour ce projet sur la page Elektor Labs [1].

200569-04 — VF : Denis Lafourcade

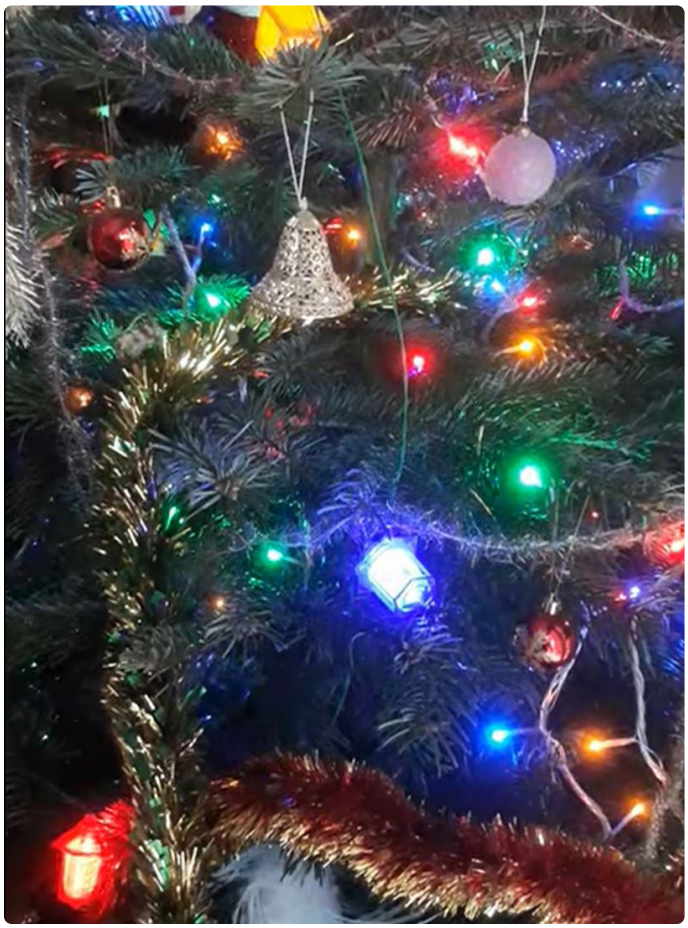


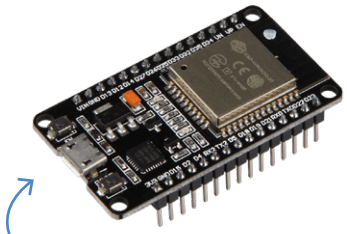
Figure 6. La féerie de lumières en action dans l'arbre.

À propos de l'auteur

Serge Sussel a découvert l'électronique au milieu des années 60 en achetant des magazines d'électronique tels que Elektor, Radio Plans, Haut-Parleur et Audiophile. Puis ses études supérieures lui ont permis de développer, entre autres, son expertise en informatique. Il a travaillé au sein de la Banque de France sur de grands systèmes au niveau du système d'exploitation (OS) et du sous-système de base de données, en développant également un préprocesseur pour compiler les éléments de sécurité des transactions. Sur le plan personnel, il s'est intéressé à l'électronique analogique dès le début, lorsqu'il est tombé dans la marmite. Il a construit des appareils de mesure, entre autres, avec des kits Heathkit, des préamplificateurs et des amplificateurs HiFi, ainsi qu'un orgue à trois claviers fourni en kit au début des années 80. Aujourd'hui à la retraite, il pratique toujours l'analogique mais aussi les plates-formes à microcontrôleurs, et il restaure des appareils très endommagés et ceux jetés par les gens qui ignorent qu'en changeant quelques composants, c'est reparti pour encore plusieurs années.

Des questions, des commentaires ?

Contactez Elektor (redaction@elektor.fr).



PRODUITS

- **Carte de développement Joy-IT NodeMCU ESP32 (SKU 19973)**
  www.elektor.fr/19973
- **Livre en anglais « FreeRTOS for ESP32-Arduino », Warren Gay, (Elektor 2020, SKU 19341)**
  www.elektor.fr/19341

LIENS

[1] Articles d'Elektor par Warren Gay : www.elektormagazine.com/warrengay
[2] Sources et matériel pour ce projet sur Elektor Labs :
www.elektormagazine.fr/labs/esp32-with-freertos-driving-some-christmas-led-garlands