MCU et Aléatoire

Le Gourrierec Alan

October 15, 2023

Unité d'enseignement : IOT (Internet Of Thinks)

Enseignant : G. Menier / P. Charton

Établissement : ENSIBS (Vannes)

Idée et réalisation

Durant le chapitre 8 d'IOT, nous devons réaliser un programme qui nous permet, à chaque lancement de notre ESP32 de faire allumer chacune des leds de manière aléatoire.

Problème rencontré

L'énoncé paraît plutôt simple, voir même trivial, faire allumer les leds de façon aléatoire. Nous nous disons naturellement que nous allons utiliser la fonction :

math.random()

Le problème avec ceci est que cette fonction est basée sur os.time(), c'est-à-dire le temps depuis lequel le système est lancé, or ce dernier n'est pas aléatoire au lancement de l'ESP32. Il me fallait donc une autre solution pour générer ces valeurs.

Idée

J'ai donc dû réfléchir de la manière de le réaliser. Dans un premier temps, j'ai eu beaucoup d'idée :

- utiliser le wifi pour me connecter à une horloge mondiale
- utiliser les fonctions de crypto incluses dans l'ESP32
- utiliser un capteur pour obtenir une situation différente à chaque fois

Celles-ci sont des solutions qui pourraient convenir à ce que nous recherchons, cependant, elle ne m'ont pas intéresse plus que ça (particulièrement celle nécessitant l'ajout de capteurs) car je souhaitais un fonctionnement en toute condition.

La solution que j'ai choisie est l'utilisation d'un random modulaire. Le fonctionnement est plutôt simple : si nous prenons un nombre premier p qui est inférieur à $2^{31} - 1$ (8^e nombre de Mersenne) (limite d'un int) et a, un entier qui n'est pas divisible par 7. Nous utiliserons le nombre p comme modulo, car ce nombre est premier, si nous effectuons $a*n[p]$ nous atteindrons toutes les valeurs de 0 à n-1, par exemple :

Si nous prenons p = 7 et a = 6, nous obtenons :

n	0	1	2	3	4	5	6
$a * n$	0	6	5	4	3	2	1

Table 1: table de congruance de 7

Réalisation du programme

J'ai donc réalisé le code suivant :

```lua
function SaveInFile(value,PATH)
    --sauvegarde de la valeur dans le fichier
    io.output(PATH)
    io.write(""..value)
    io.close()
end

function Random(PATH,modulo)
    --[[mathématiquement parlant, si nous prenons en modulo un nombre premier
    nous obteindrons forcément pour chaque itération de notre modulo (de 0 à m-1
    une valeur différente, et je me base donc sur ce principe pour la génération)]]--

    local f = assert(io.open(PATH, "r"))
    i = f:read("*all")
    m = 131071 --nombre premier (6e nombre de Mersenne)
    nbr = 22091 --autre nombre premier
    i = math.fmod(nbr*i,m)
    f:close()
    SaveInFile(i,PATH)
    return math.fmod(i,modulo)
end
```

Figure 1: code aléatoire en lua

Ce code est donc partagé en deux parties principales, la partie réalisation de l'aléatoire et la partie stockage de la valeur obtenue afin que cette valeur soit toujours "aléatoire" (car l'aléatoire en informatique n'existe pas). J'ai choisi par ailleurs d'ajouter une fonction modulo au résultat comme dans la fonction aléatoire de base de l'esp32 pour pouvoir l'utiliser de manière plus pratique. J'ai aussi choisi de réutiliser la valeur précédente de $n * i$ en guise de ma variable i pour faire croître plus vite et "augmenter" l'aléatoire

L'autre partie de ce dernier est la fonction **SaveInFile** qui permet donc de stokers la valeur de i dans un fichier nommé *test.txt*. À chaque itération de cette fonction, elle écrase le fichier précédent et stock la valeur.

Conclusion

J'ai donc réaliser cette fonction, sans trop de difficulté en réalité et ai surtout pu voir comment stocker des variables dans des fichiers en lua. J'ai aussi pu apprendre mieux le fonctionnement de l'ESP32 et le fonctionnement du pseudo-aléatoire de l'ESP32 et en informatique en général. Cette solution n'est sans doute pas la plus performante, mais c'est celle qui m'a paru le plus évident.