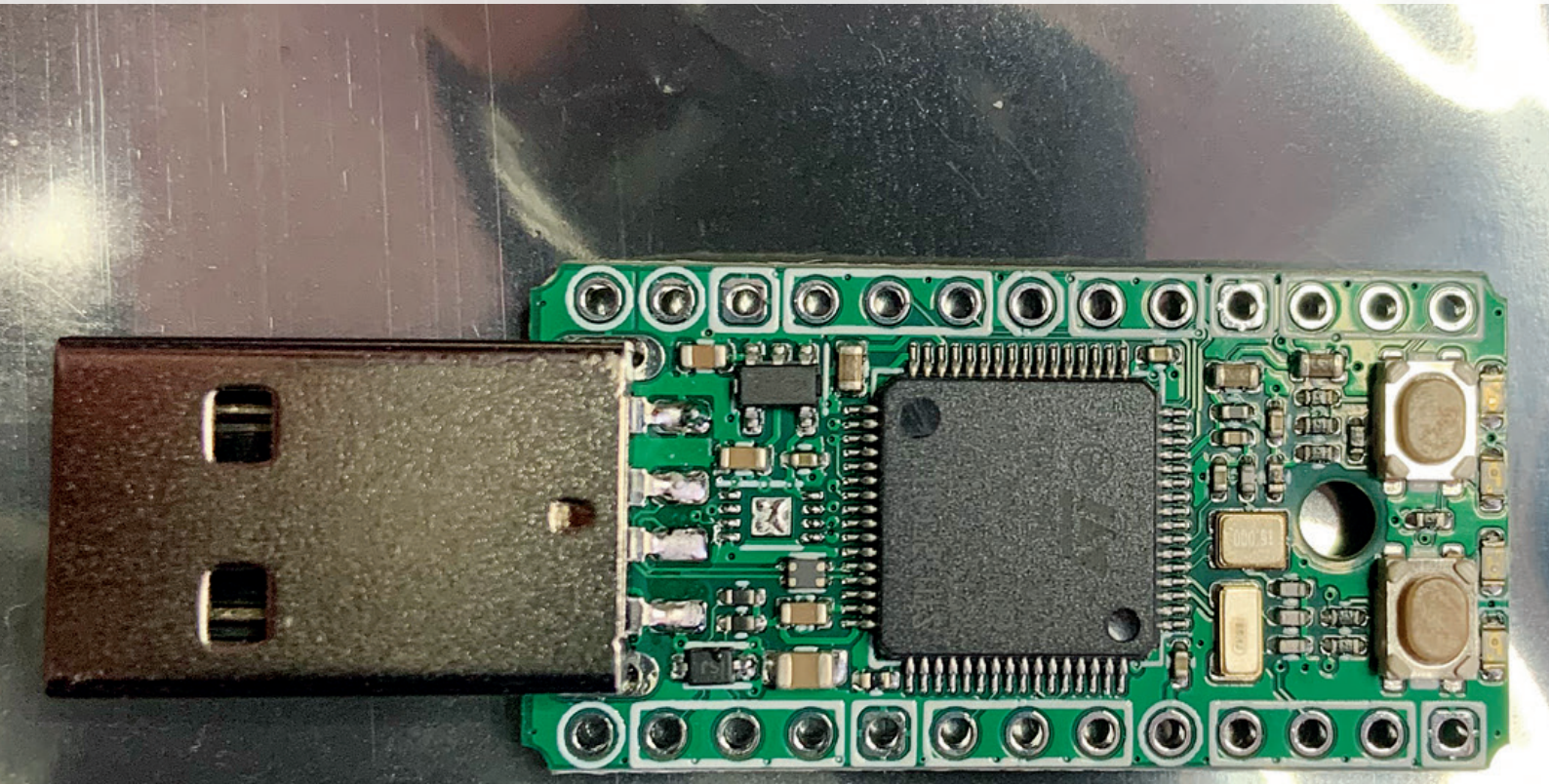
PROGRAMMEZ!

Le magazine des développeurs



SPÉCIAL
PYBSTICK 26

20 pages de tutoriels !

Créer un Pong
Contrôler un ruban de LED RGB
PYBStick + Raspberry Pi : le duo parfait
Sins : un jeu de labyrinthe

GRATUIT

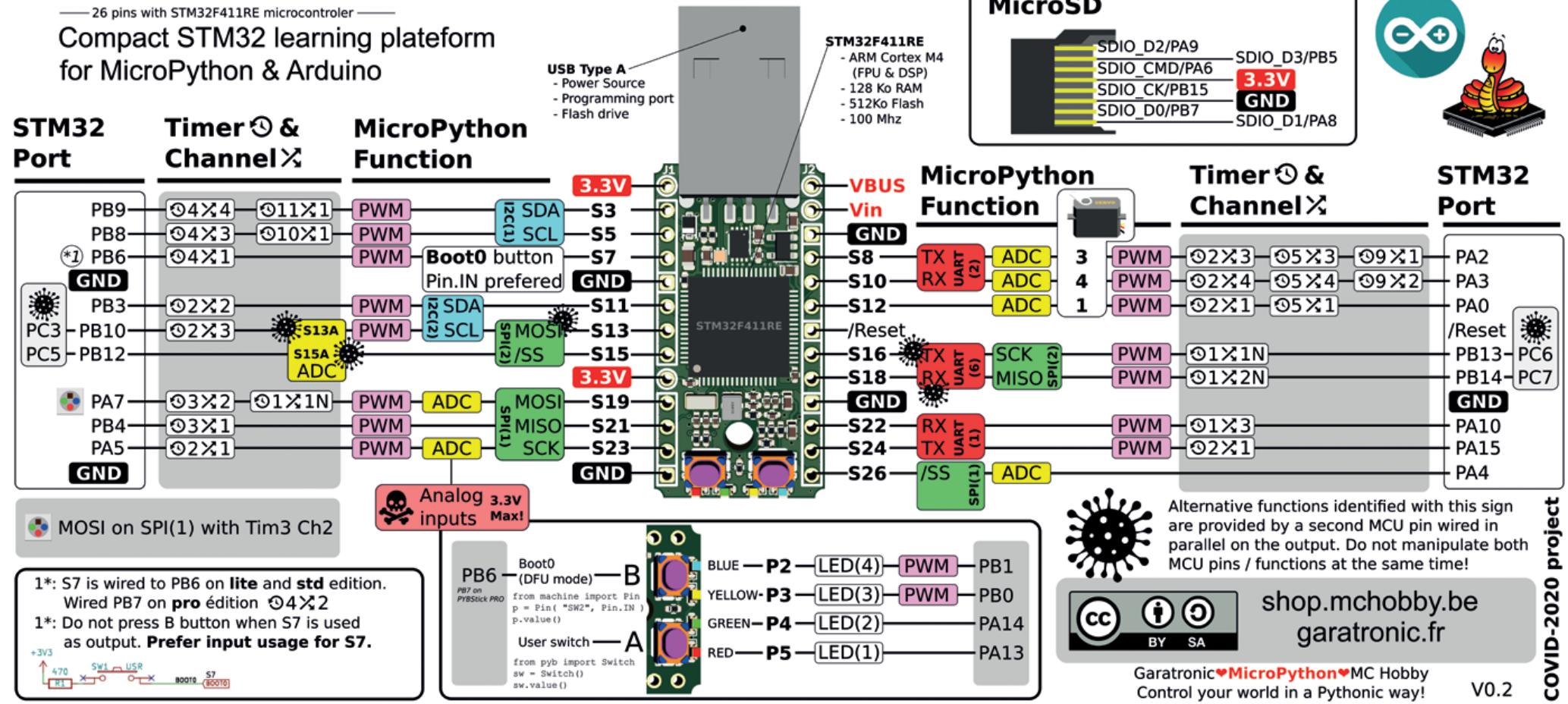
Été 2020

Imprimé en France

PYBStick Std 26

26 pins with STM32F411RE microcontroler

Compact STM32 learning plateform
for MicroPython & Arduino



1*: S7 is wired to PB6 on **lite** and **std** edition.
Wired PB7 on **pro** édition ⏲4✕2
1*: Do not press B button when S7 is used as output. **Prefer input usage for S7.**

shop.mchobby.be
garatronic.fr

Garatronic❤MicroPython❤MC Hobby
Control your world in a Pythonic way!

V0.2

COVID-2020 project

/EDITO/

Dans l'univers des cartes makers et de prototypages, le choix ne manque pas et à tous les prix ! La PYBStick Standard 26 se démarque sur plusieurs points :
1 un tarif attractif
2 dimensions réduites & poids plume
3 connecteur microSD
4 SoC STM
5 nombreuses GPIO
6 Compatible MicroPython & Arduino IDE
7 Développée en France par Garatronic & McHobby

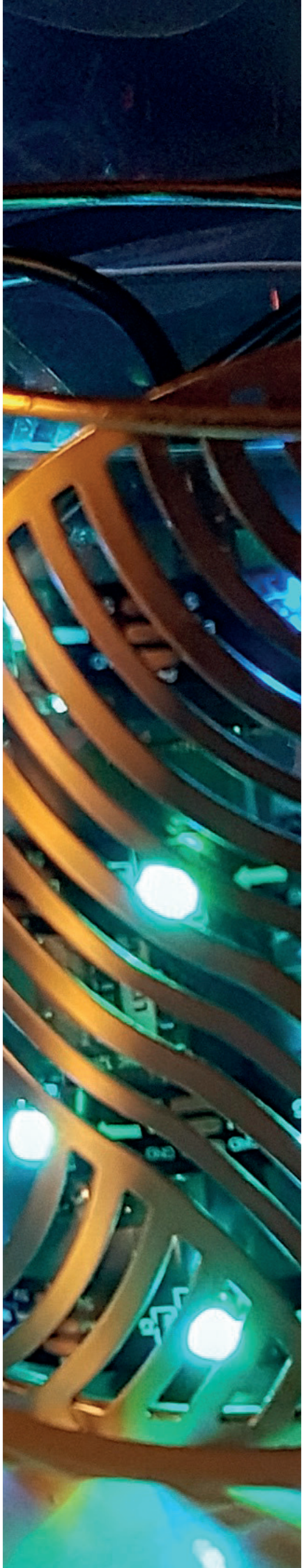
Cette carte nous a séduit immédiatement par sa taille et sa programmation en MicroPython et Arduino. La PYBStick est un bon compromis pour les débutants et les makers avertis.

Programmez! est depuis de nombreuses années partenaires d'événements makers tels que la Maker Faire ou Makeme. Nous sommes très sensibles à ce mouvement et à tout ce qui est DIY (faire soi-même). C'est pour cela qu'en partenariat avec Garatronic et McHobby, nous avons édité une série Programmez! de la PYBStick Standard 26.

Dans ce cahier maker, nous vous proposons plusieurs tutoriels et montages pour votre PYBStick. Idéal pour débuter en douceur avec cette carte sur-puissante.

Pour aller plus loin :
- Présentation de la carte dans Programmez spécial été 2020
- Sur le GitHub de McHobby : https://github.com/mchobby/pybstick-projects

François Tonic
Rédacteur en chef
Maker



/2/ **Schéma de la carte**

/3/ **Sommaire + édito**

/4/ **Débuter en MicroPython avec la PYBStick26 et la Raspberry Pi**

/8/ **Ruban de leds RGB contrôlé par une PYBStick26**

/12/ **Vive Pong !**

/17/ **Sins : un jeu de labyrinthe**

Une publication **NEFER-IT**
57 rue de Gisors 95300 Pontoise
09 86 73 61 08 / ftonic@programmez.com
Directeur de la rédaction & rédacteur en chef : François Tonic
Chromie : Pierre Sandré
Ont collaboré à ce numéro : François Mocq, Olivier Bersot, Arnaud Aujon Chevalier

Relecture : F. Hue, la rédaction
Imprimé en France par Imprimerie Jaurès
Dépôt légal : à parution

Publication spéciale : GRATUIT

Débuter en MicroPython avec la PYBStick26 et le Raspberry Pi

La carte PYBStick26 rassemble dans un volume réduit tout ce qu'il faut pour réaliser un projet Maker. Utilisée avec le Raspberry Pi, elle offre la possibilité de développer des scripts MicroPython sans mettre en œuvre un PC encombrant et gourmand en énergie.

Préparer le Raspberry Pi

Pour la rédaction de cet article, j'ai utilisé un Raspberry Pi 4 qui est sur le bureau. Comme toujours avec Raspberry Pi OS, avant d'installer un nouveau logiciel il convient de faire la mise à jour du système. Dans une fenêtre de terminal tapez :

```
sudo apt-get update
sudo apt-get upgrade
```

Pour dialoguer avec MicroPython, l'OS du Raspberry Pi ne dispose pas en natif d'un émulateur de terminal. J'ai choisi d'installer screen qui est un gestionnaire multi-terminaux (multiplexeur de terminaux) que vous pouvez découvrir plus en avant sur la page que lui consacre Ubuntu [1].

```
sudo apt-get install screen
```

et… c'est tout. On peut maintenant connecter la carte au Raspberry Pi

Connecter la PYBStick26 au Raspberry Pi

Ben… Comment vous dire ? Il suffit de connecter la prise USB de la carte PYBStick26 dans une des prises USB du Raspberry Pi. (Image 1)

La détection de la carte par le Raspberry Pi

Dès que vous connectez la carte PYBStick26 au port USB du Raspberry Pi, vous voyez apparaître la carte comme si vous aviez inséré une clé USB nommée PYBFLASH. La fenêtre qui s'ouvre vous invite à ouvrir le gestionnaire de fichiers sur cette clé. (Image 2).

Comme vous le voyez il n'y a pas de manip compliquée à faire, puisque le simple fait de connecter cette carte sur un port USB du Raspberry Pi vous permet d'accéder à la mémoire de la PYBStick26 et d'y déposer votre script. Cliquez sur le bouton Valider et le gestionnaire de fichiers vous ouvre une fenêtre sur les fichiers de la PYBStick26. Le programme qui nous intéresse ici est main.py qui sera exécuté automatiquement par le STM32 lors de la mise sous tension de la carte.

Écrire un Script en MicroPython

Le programme principal main.py

L'objet de cet article n'est pas de faire un cours sur MicroPython, mais de vous montrer comment accéder à la carte et exécuter vos programmes. Si vous cherchez des infos sur MicroPython, il y a

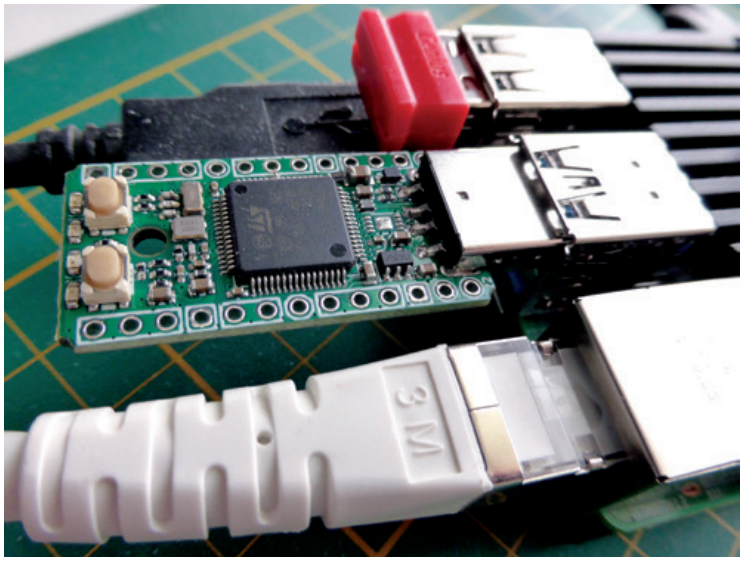


Image 1

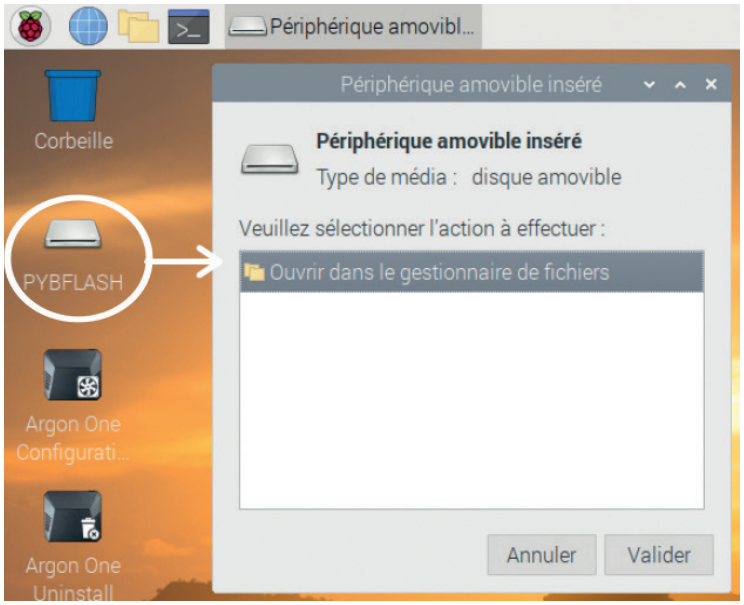


Image 2

le livre de Dominique Meurisse (MCHOBBY) paru aux Éditions ENI qui vous donnera toutes les informations utiles : MicroPython et Pyboard [2]. Pour commencer on va ouvrir le programme main.py en double cliquant dessus : IMAGE 3

Par défaut le programme est vide et ne contient qu'une ligne qui vous indique que c'est ici que vous pouvez écrire votre code.

Attention : pas gentil, Thonny

Attention, comme on ouvre un programme avec l'extension .py (Python) c'est l'IDE Python par défaut du Raspberry Pi qui s'ouvre. Thonny fonctionne avec le Python du Raspberry Pi (vous me suivez toujours ?) et pas avec le MicroPython de la carte ! Ça veut dire que si vous exécutez votre programme main.py dans Thonny vous allez avoir plein d'erreurs car ce n'est pas le bon Python qui exécute votre programme… Je vous aurai prévenu(e)…

Que la lumière soit !

La carte PYBStick26 embarque 4 LED de couleurs différentes (bleu, jaune, vert et rouge) (IMAGE 4). Ce sont ces LED qui vont nous servir pour notre première tentative d'écriture de programme. Et n'oubliez pas que le "Hello World" de l'électronicien c'est de savoir allumer une LED. Ça prouve qu'on est capable de gérer (encore imparfaitement, mais quand même) la chaîne de fonctionnement d'un microcontrôleur. La LED c'est la porte d'entrée du microcontrôleur…

Saisissez le script test1.py (IMAGE 5) dans main.py. Comme vous pouvez le constater je ne me suis pas foulé, c'est du Python de base mais ça fonctionne. Ici pas de boucle, juste du linéaire. On fait simple pour commencer. Les programmes de cet article sont disponibles en ligne au format zip [3]

Si vous regardez le code : on importe les bibliothèques qui vont nous être utiles (LED et sleep), on allume les 4 LED en les mettant à ON, après 0,5 seconde on éteint tout le bazar et on lance le chenillard qui allume les LED les unes après les autres plusieurs fois de suite. On termine en beauté en rallumant toutes les LED et… c'est fini. Le programme s'arrête. On verra après comment faire un programme qui tourne indéfiniment.

Enregistrez le programme, éjectez proprement la "clé USB" avec l'icône d'éjection en haut à droite à côté de l'icône Bluetooth. Redémarrez le Raspberry Pi ou alors plus simple, débranchez et rebranchez la carte PYBStick26.

Si tout va bien, vous verrez la séquence lumineuse décrite ci-dessus. Si ça ne fonctionne pas revérifiez soigneusement ce que vous avez écrit.

J'ai un bouton qui me démange

Enfin non, le bouton il est sur la carte. Il y a deux boutons un pour passer en mode DFU (pour programmer la carte – mais c'est une autre histoire) et un autre qui est réservé à l'utilisateur (IMAGE 4). Pour utiliser le bouton, nous allons mettre un autre script dans main.py. Rouvrez main.py et remplacez le programme précédent par test2.py (IMAGE 6).

Cette fois, les LED sont éteintes au démarrage et si on appuie sur le bouton elles s'allument toutes les 4. En même temps, le programme envoie une information sous forme de texte vers le port de sortie. Elle s'affichera à l'écran… dans la prochaine étape. Soyez patient(e) on ne peut pas tout voir en même temps.

Ici on ajoute la bibliothèque Switch qui va nous servir à lire l'état du bouton poussoir. C'est la boucle while qui fait tourner le programme indéfiniment puisque la condition est toujours vraie (True). Ensuite on teste dans la boucle si le bouton est appuyé. S'il l'est, on allume les LED. Sinon on les éteint.

Comme tout à l'heure enregistrez le script, éjectez proprement la "clé USB" avec l'icône d'éjection en haut à droite à côté du Bluetooth. Redémarrez le Raspberry Pi ou débranchez et rebranchez la carte PYBStick26.

Cette fois vous devriez allumer les LED en appuyant sur le bouton A (IMAGE 4).

Si vous voulez en savoir plus je vous conseille de lire la page de Dominique [4] qui donne plein d'exemples en REPL (on y vient tout de suite) mais qui sont facilement transposables dans main.py, en plus ça fait un bon exercice.

Utilisation de l'interface REPL

REPL, c'est quoi donc ?

L'interface USB de la carte PYBStick26 permet également à la PYBStick d'exposer un port série pour communiquer avec le Raspberry Pi ainsi qu'un interpréteur de commande Python interactif appelé REPL.
L'interpréteur de commande REPL permet:
• De voir les messages envoyés par vos scripts MicroPython,
• D'interagir avec vos scripts en cours de fonctionnement (arrêter le script, inspecter l'état de variables)
• De tester des scripts en cours de développement (importation à la volée avec la commande import).
• De saisir des instructions Python pour tester des fonctionnalités de la plateforme microcontrôleur de façon interactive.

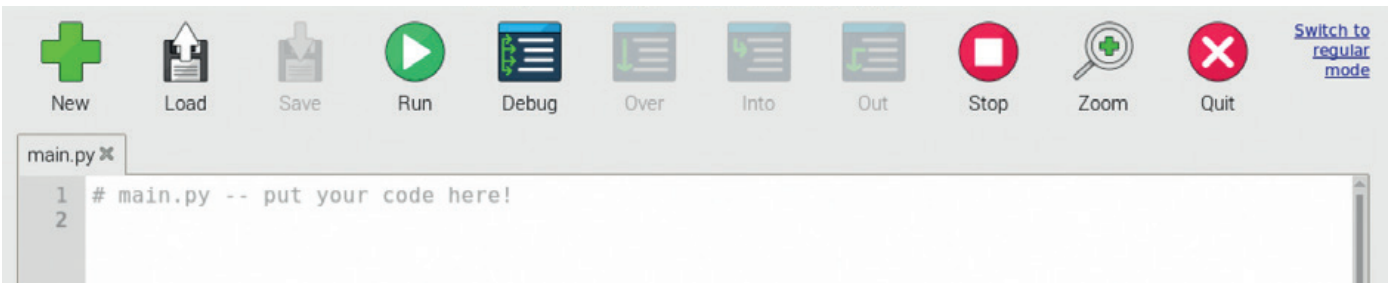
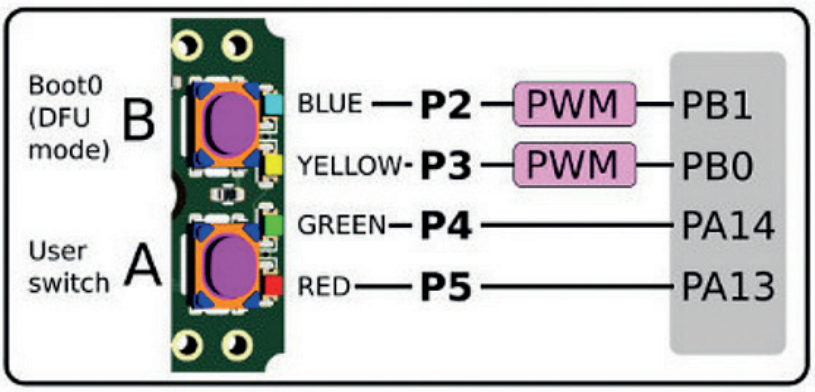


Image 3

Image 4



Démarrer l'interface REPL

Au début de cet article, je vous ai fait installer screen. Si ce n'est pas encore fait c'est le moment ! Pour les premiers essais avec screen, effacez le contenu du fichier main.py sur la PYBStick26. Rebootez le Raspberry Pi.

Ouvrez screen en tapant la commande suivante dans une fenêtre terminal :

```
screen /dev/ttyACM0
```

Vous obtenez l'affichage suivant envoyé sur son port série par la carte PYBStick :

```
MPY: can't mount SD card
MicroPython v1.12-256-geae495a-dirty on 2020-03-31;
PYBSTICK26_LITE with STM32F411CE
Type «help()» for more information.
>>>
```

Le prompt de Python, composé de trois signes >>> apparait et attend vos instructions. L'indication à propos de la carte SD est normale, car il n'y a pas de carte SD dans le lecteur. Pour tester et vérifier que la carte répond essayez ceci :

```
>>> import sys
>>>Print (sys.version)
3.4.0
>>>
```

On importe la bibliothèque sys et on affiche la version du système. Tout va bien. Vous pouvez maintenant si vous le souhaitez réaliser les nombreuses manips que vous propose Dominique sur la page github de la carte PYBStick26 [4].

Affichage d'erreur sur REPL

Nous disposons maintenant d'un outil connecté à la PYBStick26. Je vous propose de créer une erreur dans le programme pour voir ce qui se passe. C'est intéressant de voir ceci car des erreurs… on en fait forcément quand on teste des programmes, alors autant commencer par ça.

Ça marche pas !

Commencez par ouvrir main.py (il devrait être vide maintenant) et saisissez le programme suivant :

```
# main.py -- put your code here!
from pyb import LED
from time import sleep
l1 = LED(1)

while True :
   led1.on()
   sleep(0.5)
   led1.off()
   sleep(0.5)
```

```
1  from pyb import LED
2  from time import sleep
3
4  l1 = LED(1)
5  l2 = LED(2)
6  l3 = LED(3)
7  l4 = LED(4)
8
9  # Allume
10 l1.on()
11 l2.on()
12 l3.on()
13 l4.on()
14 sleep (0.5)
15 l1.off()
16 l2.off()
17 l3.off()
18 l4.off()
19 sleep (0.5)
20
21
22 # Chenillard
23 for i in range (1, 6):
24     l1.on()
25     sleep(0.5)
26     l1.off()
27
28     l2.on()
29     sleep(0.5)
30     l2.off()
31
32     l3.on()
33     sleep(0.5)
34     l3.off()
35
36     l4.on()
37     sleep(0.5)
38     l4.off()
39
40 # Allume  toutes les LED
41 l1.on()
42 l2.on()
43 l3.on()
44 l4.on()
45 sleep (0.5)
46 l1.off()
47 l2.off()
48 l3.off()
49 l4.off()
50 sleep (0.5)
```

Image 5

Image 6

```
# main.py -- put your code here!
from pyb import LED
from pyb import Switch
from time import sleep

l1 = LED(1)
l2 = LED(2)
l3 = LED(3)
l4 = LED(4)

sw = Switch()

while True:
    # Si le bouton A = User Switch est appuyé
    if sw.value():
        # Allume les LED
        l1.on()
        l2.on()
        l3.on()
        l4.on()
        # Informe l'utilisateur
        print ("Bouton appuyé")

    # Si le bouton est relâché, éteindre les LED
    else:
        l1.off()
        l2.off()
        l3.off()
        l4.off()
        # Informe l'utilisateur
        print ("Bouton relaché")
```

Un simple programme pour faire clignoter la LED 1. La boucle infinie while True fera s'exécuter ce programme en permanence. Il n'aura pas échappé à votre sagacité que la LED créée au début du programme s'appelle l1 et que j'ai utilisé led1 dans la boucle… donc… ça ne fonctionne pas !

Redémarrez le Raspberry Pi (ou débranchez/rebranchez la PYBStick26) et observez le résultat dans le terminal (il faut relancer screen /dev/ttyACM0). Vous obtenez ceci :

```
MPY: can't mount SD card
Traceback (most recent call last):
File «main.py», line 7, in

NameError: name 'led1' isn't defined
MicroPython v1.12-256-geae495a-dirty on 2020-03-31;
PYBSTICK26_LITE with STM32F411CE
Type «help()» for more information.
>>>
```

Il vous indique bien que led1 n'est pas défini et que la première erreur est à la ligne 7. Ceci vous permettra de corriger le programme. Remettez le programme en ordre (modifiez led1 en l1 aux deux endroits) et… supprimez les deux points après le while true (oui, je sais, c'est pas gentil). Redémarrez le Raspberry Pi (ou débranchez/rebranchez la PYBStick26) et observez le résultat dans le terminal (il faut relancer screen /dev/ttyACM0).
Vous obtenez ceci :

```
MPY: can't mount SD card
Traceback (most recent call last):
File «main.py», line 7
SyntaxError: invalid syntax
MicroPython v1.12-256-geae495a-dirty on 2020-03-31; PYBSTICK26_LITE
with STM32F411CE
Type «help()» for more information.
>>>
```


Hein ? encore la ligne 7 ? Eh oui, faites marcher votre neurone. L'interpréteur Python après le while attend le signe : qu'il ne trouve pas à la ligne 6. Il se dit "tiens ! il doit être à la ligne 7…". Du coup, il passe à la ligne 7 et là pas de signe : en vue ! Il s'arrête et vous dit : "J'ai trouvé une erreur et je suis à la ligne 7". A vous de remonter la piste… C'est comme ça que parfois pour une parenthèse ou une accolade manquante on se retrouve avec une erreur 20 lignes plus bas. Mais vous verrez, au début on traite l'interpréteur de tous les noms… Et puis on s'y fait et on finit par admettre que s'il signale une erreur il n'a pas tort (donc il a raison) et que la panne (comme souvent) se situe entre la chaise et le clavier, pas dans l'ordinateur.

Allez, réparez une dernière fois le main.py (à moins que vous n'ayez rajouté involontairement une autre erreur, ça arrive souvent !) et vous devriez obtenir un magnifique clignotement de la LED 1

Conclusion

Une taille réduite, un prix abordable et une documentation importante et EN FRANÇAIS font de la PYBStick26 un formidable outil d'apprentissage. Elle ravira les makers, mais aussi les FabLab ou les collèges/lycées/IUT qui trouveront avec cette micro-carte un outil polyvalent à intégrer à leurs projets.

Notess

[1] https://doc.ubuntu-fr.org/screen
[2] https://m.editions-eni.fr/livre/micropython-et-pyboard-python-sur-microcontroleur-de-la-prise-en-main-a-l-utilisation-avancee-9782409022906
[3] https://www.framboise314.fr/wp-content/uploads/2020/06/PYBSTICK26_12.zip
[4] https://github.com/mchobby/pyboard-driver/tree/master/PYBStick

François Mocq
Créateur du blog
www.framboise314.fr
Auteur du livre Raspberry Pi 4
aux Editions ENI

Ruban de leds RGB contrôlé par une PYBStick26

Dans ce tutoriel nous allons fabriquer des décors lumineux à base d'un ruban de leds RGB néopixels d'Adafruit qui s'anime dès qu'on le branche. Pour cette réalisation nous allons utiliser une PYBStick26, fabuleuse carte MicroPython qui va nous permettre de contrôler un ruban grâce à quelques-unes de ses 26 broches entrées/sorties, ainsi qu'une bibliothèque MicroPython spécialement conçue pour cet usage.

Matériel nécessaire

J'ai conçu une petite carte d'extension pour la PYBStick26 qui va permettre de piloter un ruban de leds RGB de votre choix. Attention à bien prendre un modèle RGB (et non pas RGBW) neopixels. Dans mon cas, j'ai utilisé un ruban d'1m RGB souple noir de 30 leds que j'ai enroulé à l'intérieur d'un décors de type support de bougie.

En plus du ruban, je conseille de prendre aussi une paire de connecteurs jst-sm 3 pôles mâle/femelle pour pouvoir brancher/débrancher le ruban en toute sécurité sans se tromper avec les polarités, car il n'y survivrait pas.

Selon le nombre de leds que vous voulez éclairer, prenez un transformateur 5V avec prise Jack DC de bonne qualité, entre 2A (pour 30 leds ou moins) jusqu'à 5A pour piloter des rubans plus grands. En principe une led consomme au max 60 ma. Les animations ne les éclairent jamais toutes en full white en même temps : tablez sur une alimentation 5v 30 ma par led. Ça tient largement.

Bien entendu il vous faudra une PYBStick26. Un modèle LITE est suffisant pour cette réalisation (c'est ce que j'utilise). Un modèle STANDARD est parfaitement compatible.

Quelques composants supplémentaires à prévoir:
1 carte d'extension, à faire fabriquer chez n'importe quel fabriquant de PCB à partir des fichiers GERBER fournis: il s'agit du fichier GERBER_LedStick_v1.1.zip
1 level-shifter 74AHCT125, avec un support 2*7 broches à souder
1 condensateur polarisé 470uF
1 condensateur céramique 100nF
1 petit bornier 3 pôles, ou bien une prise jst femelle à souder, impérativement au pas de 2,54mm: le ruban y sera connecté. Je préfère un bornier sur lequel je visse le câble avec embout jst-sm femelle libre: ça laisse plus de longueur pour y connecter le ruban avec son connecteur jst-sm mâle.
1 Barrel-Jack DC mâle 3 points à souder: l'alim 5V y sera branchée.
2 barrettes à souder 13 pin mâles pas 2,54mm (j'achète des barrettes 40 pins sécables).
1 diode de redressement ou Schottky de petite taille, type 1N4001 (ce que j'ai pris) ou SB560.

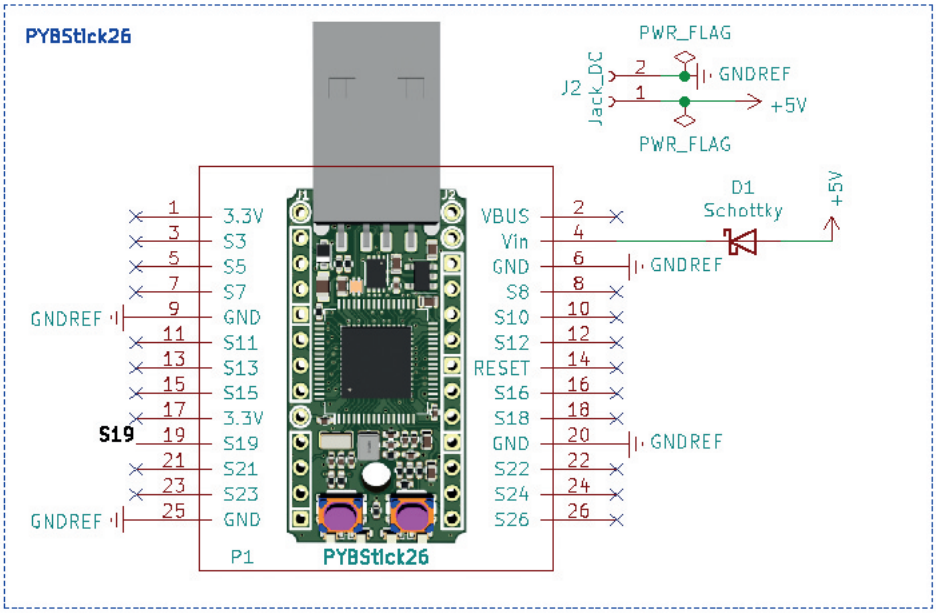
Circuit électronique : Schéma

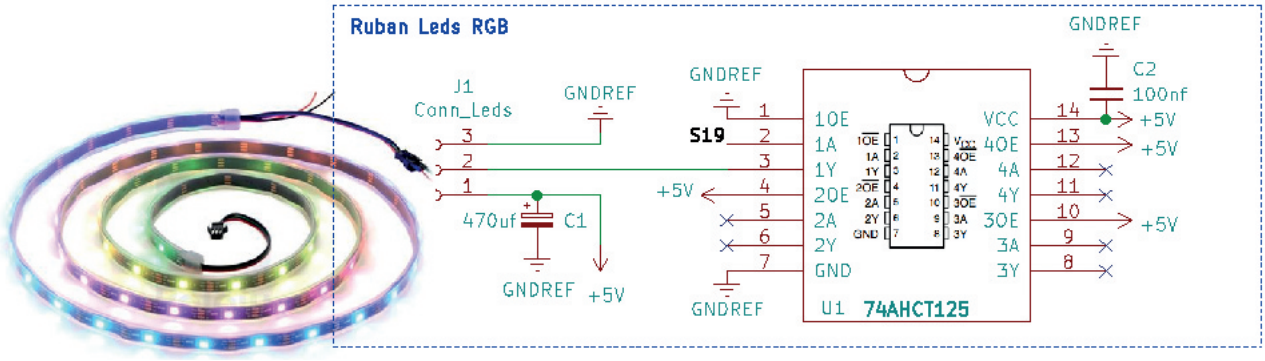
Ce circuit est réalisable sur une breadboard en mode prototype. La PYBStick26 est alimentée en externe (5V 2A) via une prise Jack DC reliée à la broche 4 Vin, avec une petite diode Schottky en série afin d'éviter que l'alimentation de la prise USB (Vbus) vienne se déverser dans l'alimentation externe. Vous me diriez «Ah oui mais l'inverse alors? Comment protéger l'alimentation USB pour éviter que l'alimentation externe ne s'y déverse?». Et bien il y en a déjà une dans la PYBStick26 qui évite cela! Ainsi nous pouvons brancher l'alimentation externe et la clé USB en toute sécurité. J'ai privilégié une petite diode 1A max (type 1N4001).

Si vous testez sur une breadboard, il n'est pas nécessaire de relier à la masse toutes les masses de la PYBStick26: une seule suffit. Il faut privilégier celle juste en dessous de Vin (pin n°6), ou bien la pin n°25 car elles sont faciles à localiser.

Le ruban est aussi alimenté en 5v avec l'alimentation externe. On ne peut pas l'alimenter avec le 5v récolté sur Vbus car c'est limité à 1A max. Il va falloir convertir la sortie S19 de la PYBStick, qui délivre un signal 3.3v, en 5v, via un circuit-intégré Level-Shifter (74AHCT125): il dispose de 4 entrées logiques qui vont pouvoir transformer 4 signaux 3.3v en 5v: un seul nous est nécessaire. On relie 1OE à la masse pour activer une conversion, puis 1A à la sortie S19 de la PYBStick (cette sortie va commander le ruban) et 1Y converti en 5v sera reliée au signal du ruban (fil du milieu). Toutes les autres entrées logiques 2OE, 3OE et 4OE doivent être reliées au +5V pour les désactiver et éviter des oscillations aléatoires de ces convertisseurs qui consommeraient du courant pour rien.

Schéma





Quelques condensateurs de découplage sont présents pour protéger le ruban de leds et le 74AHCT125, lors du branchement notamment qui peut générer des pics. Il n'y en pas besoin pour protéger la PYBStick26 car l'entrée Vin dispose d'un régulateur de tension déjà protégé par des condensateurs : on peut y mettre jusqu'à 18v sur Vin.

Attention le ruban de leds est particulièrement fragile : la moindre erreur de polarisation (vous vous trompez entre le +5v et la masse) ou la moindre erreur de tension (un chargeur 12v au lieu de 5v) et le ruban ainsi que le 74AHCT125 sont morts. Il vaut mieux vérifier 3 fois plutôt qu'une que les branchements et voltages soient corrects avant de tout mettre sous tension ...

Soudure des composants

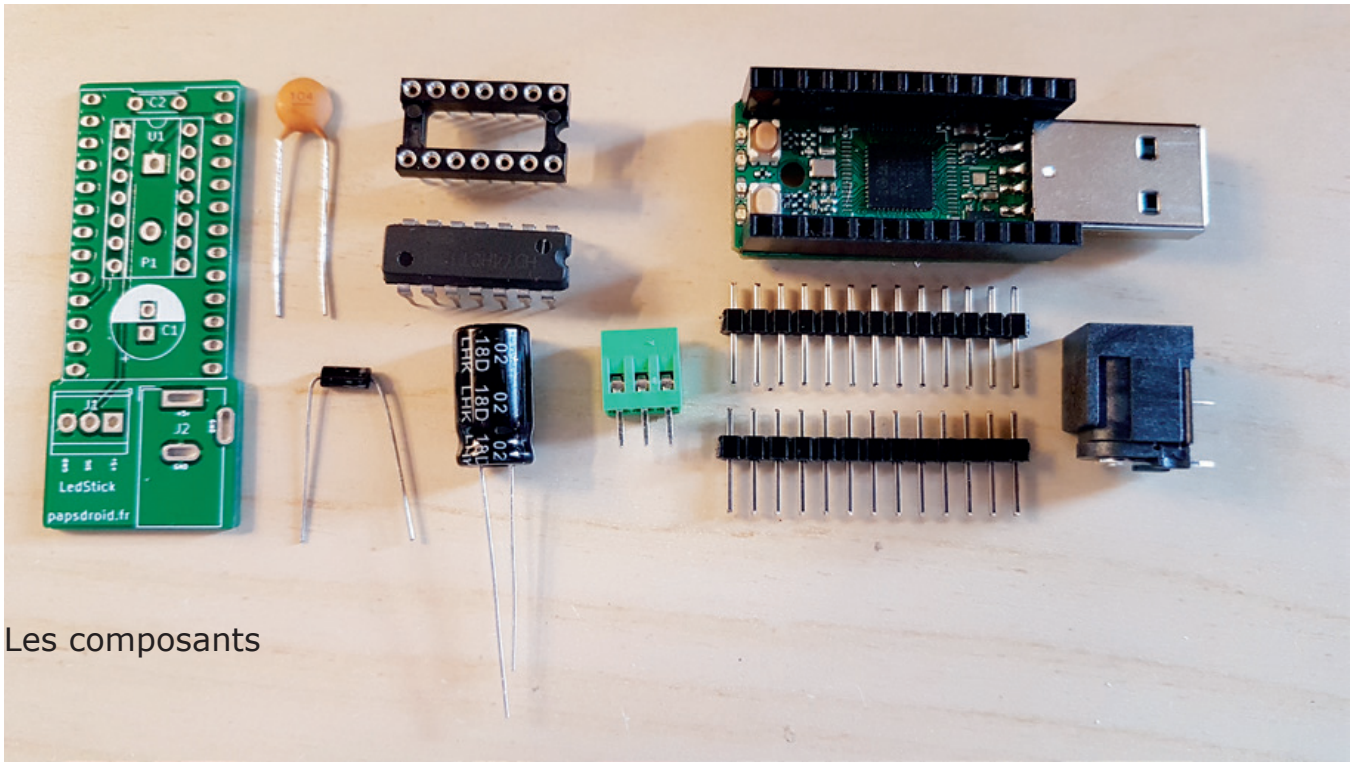
Il faut commencer par souder les barrettes femelles fournies sur la PYBStick26. Petite astuce pour les souder parfaitement perpendiculaires à leur socle : utilisez le support 2*7 broches du 74AHCT125 (si vous en avez deux, c'est encore mieux) car l'écartement entre les deux rangées correspond exactement à la longueur de ce support (6* 2,54 mm).

Commencez par souder la diode sur la carte d'extension, au choix sur la face avant ou arrière, mais attention à bien respecter le sens comme indiqué sur la sérigraphie en dessous (sens de la bague indiqué). Ensuite soudez le support du 74AHCT125 dans le bon sens, puis la petite capacité céramique, le bornier, le jack DC ainsi que le condensateur polarisé (patte la plus longue à souder sur le +). Pour finir soudez les deux rangées de 13 pins mâles orientées par dessous (soudures apparentes au dessus de la carte donc).

Vérifiez les voltages au cas où... débranchez le tout et assemblez ensuite le 74AHCT125 sur son support (dans le bon sens, attention à ne pas tordre une patte...) et enfin la carte sur la PYBStick26: le connecteur Jack DC 5V est à l'arrière de la clé USB. Connectez le ruban en vous assurant de ne pas vous tromper avec le +5V et la masse surtout (voir les sérigraphies au niveau du bornier). La PYBStick26 et sa carte d'extension sont prêtes. Il ne reste qu'à installer les scripts MicroPython.

Scripts MicroPython

Tous les fichiers mentionnés ci-dessous sont récupérables sur mon Github https://github.com/papsdroidfr/ledStick. Il faut déposer les 3 fichiers ws2812.py, rubanLeds.py et main.py à la racine du lecteur PYBFLASH lorsque vous branchez la PYBStick26 sur votre ordinateur (dans mon cas j'utilise une Pi), et c'est tout!



Les composants

Une fois les fichiers copiés sur la PYBStick26 : débranchez-la de son support USB, puis branchez le jack 5v. Les animations démarrent immédiatement.

Les bibliothèques utilisées

Deux bibliothèques sont nécessaires pour faire fonctionner le ruban avec le script MicroPython que je propose.

ws2812.py est développée par MCHobby, disponible sur leur github (section Bibliothèques - Dépendances) au demeurant très bien documenté, dans lequel il y a des exemples de codes MicroPython et de circuits pour causer avec un ruban de leds Neopixels (et plein d'autres exemples), dont je me suis très largement inspiré pour ce tuto. Cette bibliothèque permet de gérer les allumages/extinction des leds via le bus SPI de la PYBStick26 ce qui nous simplifie le travail.

rubanLeds.py : il s'agit d'une classe qui permet d'exécuter des animations colorées sur le ruban, animations que l'on va utiliser dans le script principal.

Le script principal

Il s'agit du fichier main.py que vous pouvez adapter à votre besoin avec l'import de la bibliothèque d'animation du ruban.

Usages

Il faut d'abord instancier un objet de la classe RubanLeds en précisant le nombre de leds et l'intensité souhaitée (entre 0=min et 1=max). Par exemple pour un ruban de 30 leds et intensité à 40% (100% ça troue les yeux croyez moi ! il ne faut pas les regarder de face :

```
from rubanLeds import RubanLeds
leds = RubanLeds(nb_leds=30, intensity=0.4)
```

Ensuite les principales méthodes à utiliser de la classe RubanLeds pour lancer des animations sont : leds.off() : extinction du ruban, toutes les leds en même temps.

leds.fade_off(fadein): extinction des leds les unes après les autres, en partant de la première si fadein=True (valeur par défaut), sinon en partant de la dernière.

leds.rainbow_cycle(): lance un cycle de couleur arc-en-ciel réparti sur toutes les leds. Plus le nombre de leds est grand, et plus le dégradé de couleur sera joli.

leds.random_cycle(pastel, red, green, blue): allume les leds au hasard avec des couleurs pastels si pastel=True, ou vives si pastel=False. Des composantes rouge, verte ou bleue peuvent être annulées (red=True/False, green=True/False, blue=True/False) pour donner un ton. C'est du plus bel effet quand on annule une seule composante avec des couleurs pastels.

leds.fade_wheel(fadein): allume les leds l'une après l'autre avec un effet de fade in (fadein=True) ou fade out (fadein=False) en utilisant un cycle de couleurs arc-en-ciel.

leds.fade_random(fadein, pastel, red, green, blue): allume les leds avec un effet de fade in ou fade out, couleurs vives ou pastel choisies au hasard, composantes rouge/verte/bleue activées ou non.

Montage final et effet visuel

leds.shuffle_wheel(): allume les leds dans un ordre aléatoire, en suivant un cycle arc-en-ciel

leds.shuffle_random(pastel, red, green, blue): allume les leds dans un ordre aléatoire avec de couleurs vives ou pastel, composantes rouge/vert/bleu activées ou non.

leds.mono_wheel(fadein, delay): allume toutes les leds de la même couleur mais varie en suivant les couleurs de l'arc en ciel. Démarrage avec un effet fade in ou fade out. Le delay (en secondes, 0.1 par défaut) est un temps d'attente entre chaque changement de couleur.

leds.mono_color(fadein, color, delay): allume toutes les leds avec la même couleur color=(r,v,b) rouge par défaut (255,0,0). Effet de fade in ou fade out au début. Temps d'attente delay (=0.1 secondes par défaut) avant de rendre la main.

En combinant des effets arc-en-ciel, couleurs pastels ou vives avec composantes rouge/vert/bleu désactivées ou non, du clignotement aléatoire ou des enchaînements d'allumage en fade in et extinction en fade out... : on peut créer des tonnes d'animations colorées en boucle infinie vraiment cool.

Mon usage

Dans mon cas j'utilise le ruban de leds scotché à l'intérieur d'un support de bougie. Le script main.py enchaîne pas mal d'animations colorées du plus bel effet. Un autre usage qui est très joli : fixer le ruban le long d'une plante grimpante intérieur, et l'animer avec la méthode mono_wheel(fadein=-True, delay=1). Ça génère une lumière diffuse toute verte qui vire doucement vers le jaune, puis orange, rouge, violet bleu pour redevenir vert et recommencer à l'infini. En jouant avec le paramètre delay on peut plus ou moins temporiser les changements de couleurs et c'est splendide dans une plante (attention en l'arrosant...), ou au dessus / derrière un meuble.

Les sources sont disponibles sur mon site :
https://www.papsdroid.fr/post/ledstick

Enjoy !

Olivier Bersot
Ingénieur en informatique et électronique avec une spécialisation en IA. Actuellement responsable de projets informatiques dans la finance. Il passe une grande majorité de son temps libre à partager ses tutoriels dans l'univers passionnant de l'électronique embarquée sur son site https://www.papsdroid.fr

Recréer le mythique jeu PONG

Pour commencer à utiliser la PYBStick, quoi de mieux qu'un retour aux sources du jeu, avec le célèbre PONG ? Un écran qui affiche une balle et deux raquettes, un potentiomètre pour se déplacer, du son et de la lumière : bref, un jeu qui plaira à tout le monde, tout âge confondu.

Ce projet simplissime nécessite peu de composants et son montage, réalisable en très peu de temps, ne comprend aucune difficulté. Le code est écrit en micropython, un langage puissant et intuitif, de plus en plus utilisé aujourd'hui dans l'univers des micro-contrôleurs. La définition des composants est facile à appréhender, et nous vous proposons l'intégralité du code du jeu.

Matériel

1 x PYBStick 26
1 x led verte
1 x led jaune
2 x résistance 330 Ohms
1 x afficheur OLED 128x64 1.3"
1 x Piezo buzzer
fils de prototypage (dupont)

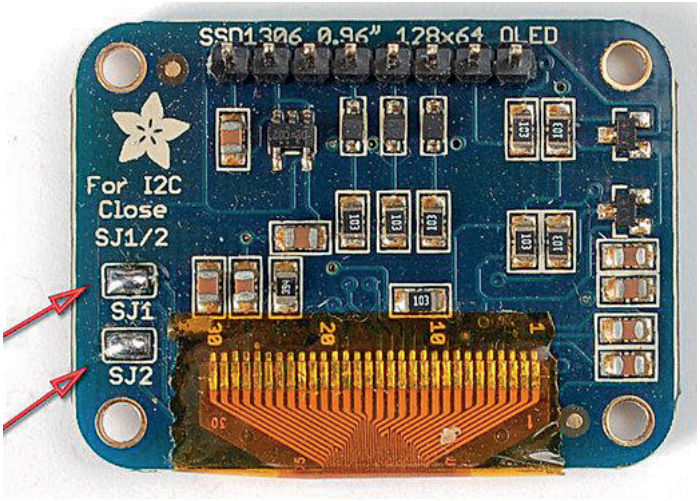
Budget : 30€ (hors PYBStick)

Pour réaliser ce projet, nous détaillons ci-dessous les composants nécessaires, leur branchement, ainsi que le code utilisé pour les faire fonctionner.

L'afficheur OLED

Nous avons ici choisi un afficheur OLED monochrome 128x64 d'une taille de 1,3" (~3,3 cm de diagonale). Il utilise un circuit de contrôle SSD1306 que l'on retrouve dans la grande majorité des petits écrans OLED. Cet afficheur fonctionne en 3,3V mais on peut aussi le connecter en 5V grâce à la présence d'un level shifter (convertisseur de niveau logique) intégré. Il a également la possibilité de fonctionner en deux modes : SPI et I2C. Nous utilisons ici le mode I2C, même s'il est moins rapide, car il ne nécessite que 2 câbles pour le transfert des données. Ces deux câbles sont appelés : SCL (Serial Clock Line), ligne d'horloge qui permet la synchronisation bidirectionnelle et SDA (Serial Data Line), ligne qui permet le transfert de données bidirectionnelles.

Pour faire fonctionner l'écran en I2C nous devons fermer les deux jumpers SJ1 et SJ2 situés à l'arrière de la carte (voir photo). Sur certains modèles, notamment les plus récents, le mode I2C est celui par défaut.



Pour câbler l'afficheur, il suffit de connecter la pin GND au GND de la carte, la pin Vin à une sortie 3,3V, la pin Data à une pin I2C SDA et enfin la pin Clk à une pin I2C SCL. Sur cet afficheur il est également nécessaire de connecter le Rst à la carte, il permet de réinitialiser l'écran afin d'obtenir un affichage stable. La PYBStick possède deux bus I2C, nous utilisons le bus I2C 1, sur les pins S3 et S5.

Le protocole I2C nécessite l'utilisation de résistances pull-down sur les pins de transfert. Elles sont activables lors de la déclaration des pins via le paramètre mode=Pin.PULL_DOWN

Il existe plusieurs bibliothèques permettant de simplifier l'utilisation d'un afficheur ssd1306, nous utilisons pour ce projet la bibliothèque officielle micropython disponible à cette adresse : https://github.com/micropython/micropython/blob/master/drivers/display/ssd1306.py

Pour commander l'afficheur, le code suivant permet de déclarer les variables :

```
import ssd1306
from machine import I2C
bus_i2c = I2C(sda=Pin(«S3», mode=Pin.IN, pull=Pin.PULL_DOWN),
```

```
scl=Pin(«S5»,  mode=Pin.IN, pull=Pin.PULL_DOWN), freq=400000)
afficheur = ssd1306.SSD1306_I2C(128, 64, bus_i2c, adresse_afficheur)
```

Rst doit avoir une valeur haute pour que l'écran fonctionne, le passage à une valeur basse pendant 10 ms permet d'effectuer la réinitialisation. Si on omet cette étape l'écran va afficher de manière aléatoire des pixels blancs et noirs.

```
rst_pin_oled = Pin(«S23», Pin.OUT)
rst_pin_oled.on()
pyb.delay(1)
rst_pin_oled.off()
pyb.delay(10)
rst_pin_oled.on()
```

La bibliothèque ssd1306 permet d'afficher simplement du texte via la méthode text(text_a_afficher, position_x, position_y, couleur), position_x et position_y représentent le coin supérieur gauche du texte à afficher. Dans le cas d'un écran monochrome couleur est toujours à 1.

Le reste des éléments à afficher : balle, raquette, séparateur au milieu de l'écran sont des rectangles qui sont dessinés grâce à la méthode fill_rect(x, y, w, h, c) qui affiche un rectangle blanc de hauteur h et largeur w à la position (x,y), c étant toujours la couleur reste à 1.

Une fois tous nos éléments dessinés, il faut utiliser la méthode afficheur.show() pour mettre à jour l'afficheur.

Dans le code, cela donne :

```
# Affichage score
afficheur.text(«%d» % score_vert, int((largeur_ecran/2)-16), 0, 1)
afficheur.text(«%d» % score_jaune, int((largeur_ecran/2)+8), 0, 1)
# Affichage balle
afficheur.fill_rect(int(position_X_balle), int(position_Y_balle), taille_balle, taille_balle, 1)
# Affichage raquette vert
afficheur.fill_rect(raquette_au_mur, position_Y_raquette_vert, largeur_raquette, hauteur_raquette, 1)
# Affichage raquette jaune
afficheur.fill_rect(largeur_ecran - largeur_raquette - raquette_au_mur , position_Y_raquette_jaune, largeur_raquette, hauteur_raquette, 1)
afficheur.show()
```

Les potentiomètres

Nous utilisons deux potentiomètres linéaires de 10 KΩ. Cela signifie que la valeur de la résistance va passer de 0 à 10kohm en augmentant de manière linéaire. A mi-course il aura donc une résistance de 5kohm. L'entrée du potentiomètre est reliée à une sortie 3,3V et la masse au GND. La tension de sortie du potentiomètre va donc varier entre 0V et 3.3V. En branchant la sortie du potentiomètre sur une entrée ADC (analog to digital converter) de la carte, on peut convertir la tension en entrée en une valeur comprise entre 0 et 4095 (0 correspondant à 0V, 4096 à 3.3V).

En appliquant un coefficient diviseur on peut donc obtenir une valeur comprise entre 0 et 53, ce qui correspond à la position du haut de la raquette sur l'écran (0 pour le haut et la hauteur de l'écran moins la hauteur de la raquette pour le bas). Ce coefficient est ici de

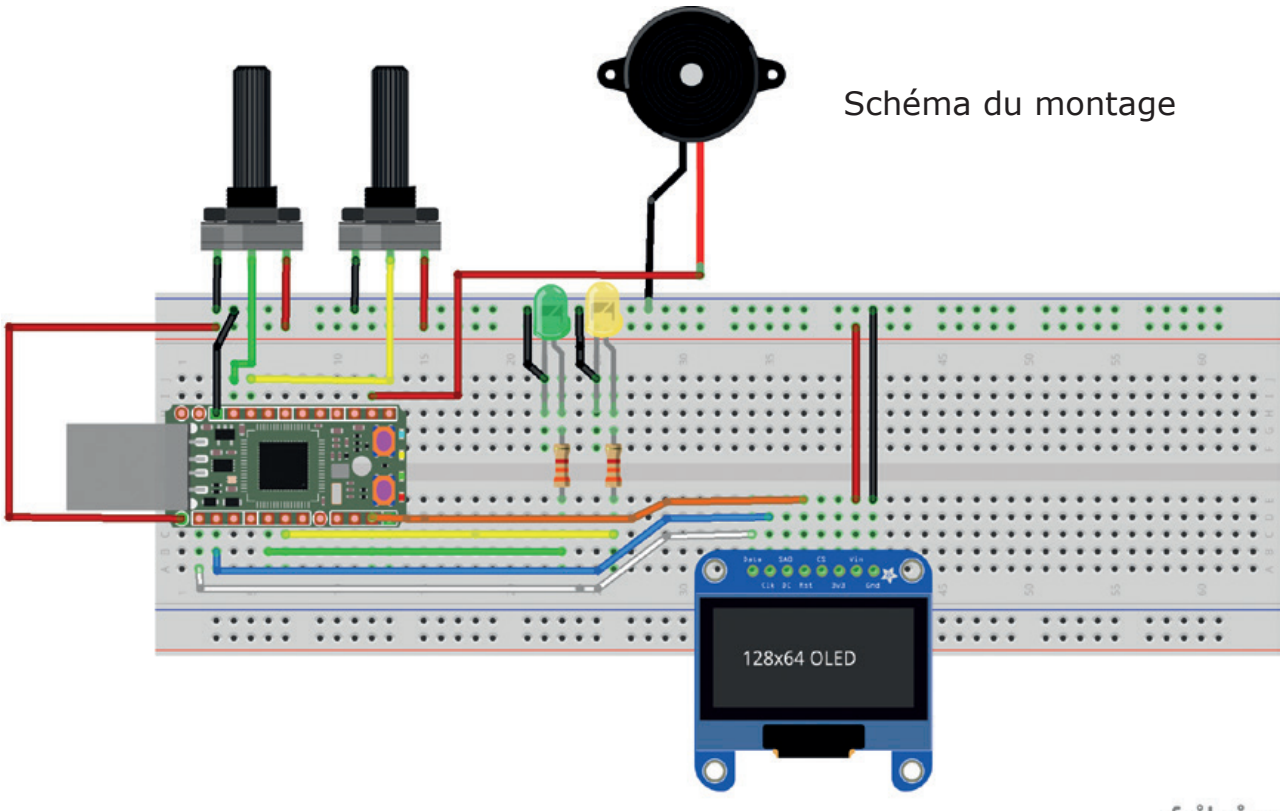
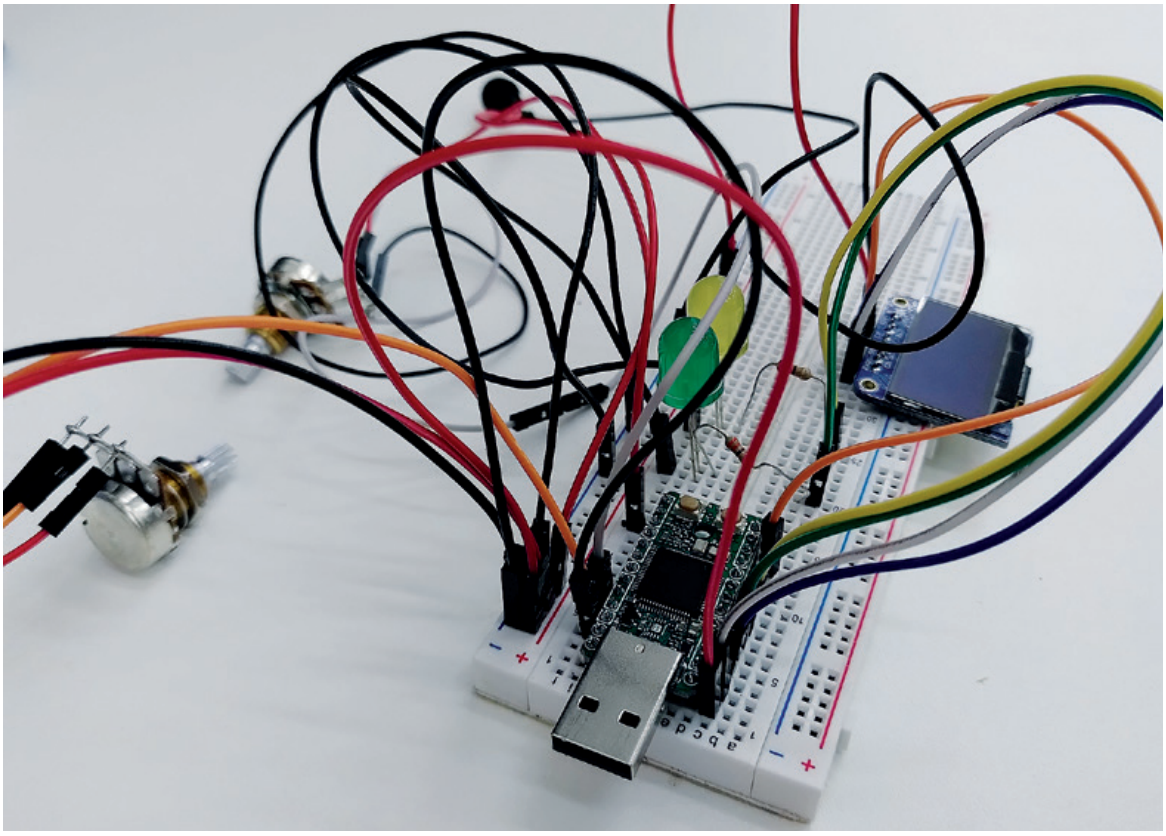


Schéma du montage



Montage du projet

77,2. Il est appelé ratio_entree_adc dans le code. Attention à bien choisir un pin possédant la capacité ADC (voir le pinout de la carte pour cela). Nous utilisons ici le S8 et S10.

Les potentiomètres sont déclarés par :

```
# declaration de l'entree ADC pour mesure position potentiomètres
from pyb import ADC
potentiometre_raquette_verte = ADC(Pin('S8'))
potentiometre_raquette_jaune = ADC(Pin('S10'))
```

Dans la boucle principale, la position de la raquette est calculée par :

```
position_Y_raquette_verte = hauteur_ecran - hauteur_raquette - int
(potentiometre_raquette_verte.read() / ratio_entree_adc)
position_Y_raquette_jaune = hauteur_ecran - hauteur_raquette - int
(potentiometre_raquette_jaune.read() / ratio_entree_adc)
```

Les LEDS

On ne peut pas brancher une led directement sur un port de la PYBStick, il faut y ajouter une résistance de limitation du courant. Les LEDS verte et jaune ont une tension passante de l'ordre de 2V. Avec une résistance de 330 ohms, on limite le courant à (3.3-2)/330 = 4mA ce que supporte le microcontrôleur STM32 (maxi -/+8mA). Pour faire clignoter les LEDs nous alternons la tension du pin grâce aux méthodes on() et off() en les espaçant de 100 ms.

Pour afficher le vainqueur du point et le vainqueur de la partie nous utilisons simplement deux LEDs déclarées sur les pins S11 et S13 :

```
led_joueur_vert = Pin(«S11», Pin.OUT)
led_joueur_jaune = Pin(«S13», Pin.OUT)
```

Le mode Pin.OUT permet d'activer le pin en sortie.

Le clignotement des LEDs est effectué par la méthode clignotement_led qui allume puis éteint la LED passée en paramètre :

```
def clignotement_point(led):
    for i in range(3):
        led.on()
        pyb.delay(100)
        led.off()
        pyb.delay(100)
```

La méthode est ensuite appelée par le code :

```
clignotement_point(led_joueur_vert)
```

Le buzzer

Le buzzer est un composant ayant une capacité piézoélectrique. La piézoélectricité est la capacité pour un matériau à produire une tension en se déformant. L'inverse est également vrai : le matériau se déforme lorsqu'il est soumis à une tension et revient à sa forme initiale quand ce n'est plus le cas. Si l'on génère des pics de tensions suffisamment rapides, le matériau oscillera et produira du son.

Le buzzer piézoélectrique permet donc d'ajouter du son au jeu. La fréquence sonore va dépendre de la fréquence du signal envoyé. Pour générer un signal pseudo-analogique nous utilisons la technique PWM (Pulse Width Modula-

tion) qui nous permet de le simuler à partir de changement rapide d'état. Afin de brancher le buzzer, il suffit de connecter le fil rouge sur un pin offrant du PWM (nous utilisons le pin S24 ici), et le fil noir au GND de la carte.

La déclaration du buzzer se fait en initialisant un timer sur la pin en lui donnant une fréquence :

```
sortie_buzzer = Pin('S24')
timer_buzzer = Timer(2, freq=2200)
pwm_buzzer=timer_buzzer.channel(1, Timer.PWM, pin=sortie_buzzer)
```

Pour l'activer ou le désactiver nous utilisons ensuite la méthode pulse_width_percent(percent) qui va définir le rapport cyclique du signal et nous permettre de changer le volume : 0% ne génère pas de son et 50% génère le son maximum, la fréquence de la note est modifiée par la méthode freq(). On peut donc définir une méthode jouer_frequence(frequence) qui va jouer une note à la fréquence donnée, nous utilisons ici un rapport cyclique de 30% pour obtenir un son de puissance moyenne :

```
def jouer_frequence( freq ):
    # si la fréquence est 0, on désactive le son
        if freq == 0:
                pwm_buzzer.pulse_width_percent( 0 )
        else:
                timer_buzzer.freq( freq )
                pwm_buzzer.pulse_width_percent( 30 )
```

Voici le tableau de correspondance entre une note et sa fréquence :

```
' ' : 0,   # Silent
'c' : 261, # Do
'd' : 294, # Ré
'e' : 329, # Mi
'f' : 349, # Fa
'g' : 392, # Sol
'a' : 440, # La
'b' : 493, # Si
'C' : 523  # Do
```

Dans le code du jeu, nous utilisons une variable duree_bip qui sera comparée à chaque tour de boucle. Si elle est positive on joue une note puis on la décrémente. On désactive le pwm si elle vaut 0. Dans le code cela se traduit par :

```
# Traitement du beep dans la boucle de $(vitesse_normale) ms
    # Traitement du beep dans la boucle de $(vitesse_normale) ms
    if (duree_bip > 0):
      jouer_frequence(261)
      duree_bip -= 1
    else:
      jouer_frequence(0)
```

Le code du jeu

La logique du code est basée sur une boucle while qui va exécuter les différentes actions du jeu (déplacement des raquettes, de la balle, émission d'un son) et dont la vitesse va pouvoir évoluer. A chaque tour de boucle, nous mesurons le temps passé et à la fin de la boucle nous le comparons à la variable vitesse_jeu pour mettre le jeu en pause le temps voulu :

```
while (True):
  debut_rafraichissement = pyb.millis()
  # calcul et actions
  while pyb.elapsed_millis(debut_rafraichissement) < vitesse_jeu:
      pass
```

Cette technique permet de garantir que le jeu se jouera toujours à la même vitesse quel que soit le microcontrôleur utilisé. Elle nous permet également de faire varier la vitesse de jeu en fonction de la durée de la partie et de la vitesse de déplacement de la raquette lorsqu'elle touche la balle. Cette caractéristique permet d'avoir un jeu qui s'adapte au niveau des joueurs.

La vitesse de la raquette est déterminée par la différence de position par rapport au dernier tour de boucle par :

```
vitesse_Y_raquette_verte = position_Y_raquette_verte - precedente_
position_Y_raquette_verte
```

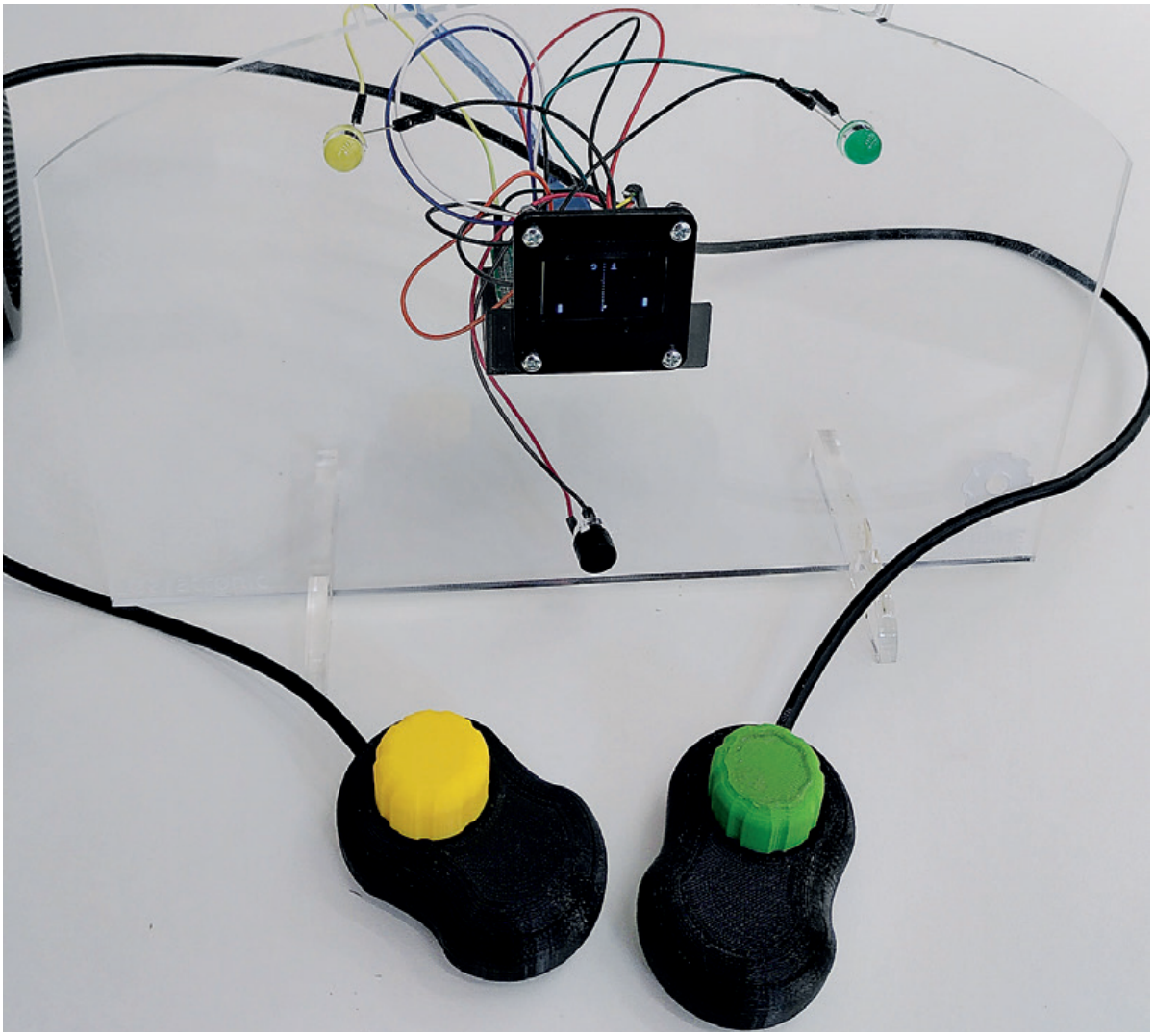
Cette information liée au déplacement est ensuite utilisée pour augmenter ou diminuer la vitesse de la balle.

Si la raquette a une vitesse supérieur à vitesse_Y_raquette_pour_effet, ici 2, c'est à dire que la raquette s'est déplacée de plus de 2 pixels vers le haut depuis le dernier tour de boucle, on accélère la balle, on la décélère si la raquette se déplace rapidement vers le bas on remet la vitesse par défaut sinon.

```
# Rebond sur raquette verte
if (position_X_balle >= raquette_au_mur) and (position_X_balle <=
raquette_au_mur + taille_balle) and (vitesse_X_balle < 0):
  if (position_Y_balle > position_Y_raquette_vert - taille_balle) and
(position_Y_balle < position_Y_raquette_vert + hauteur_raquette):
    vitesse_X_balle *= -1
    if (vitesse_Y_raquette_vert * vitesse_Y_balle) > vitesse_Y_ra-
quette_pour_effet:
      vitesse_jeu = vitesse_normale / multiplicateur_vitesse
    elif abs(vitesse_Y_raquette_vert * vitesse_Y_balle) >= vitesse_Y_
raquette_pour_effet:
      vitesse_jeu = vitesse_normale * multiplicateur_vitesse
    else:
      vitesse_jeu = vitesse_normale
```

A la fin de la partie, quand un des deux joueurs atteint 10 points, on affiche le gagnant et on va attendre 5 secondes avant de démarrer une nouvelle partie. Il ne faut pas oublier d'appeler la méthode jouer_frequence(0) pour désactiver le buzzer. Il serait également possible de jouer une mélodie.

```
# affichage du vainqueur
  if (score_vert == 10):
    afficheur.text(«vert gagne !», 10, hauteur_ecran // 2)
    clignotement_point(led_joueur_vert)
  else:
    afficheur.text(«jaune gagne !», 10, hauteur_ecran // 2)
    clignotement_point(led_joueur_jaune)
  afficheur.show()
```



Montage final et on peut commencer à jouer à Pong

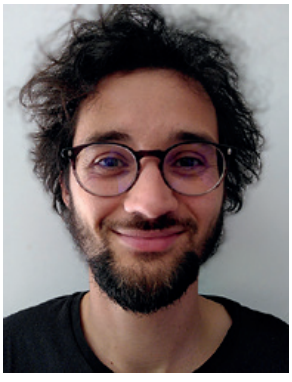
```
jouer_frequence(0)
pyb.delay(5000)
```

Pour aller plus loin

Pour rendre le jeu plus attractif nous avons créé des manettes et un support pour l'écran imprimables en 3D. Les potentiomètres se placent alors dans les manettes qui sont reliées à la carte par du câble 3 fils. Les fichiers .STL sont disponibles afin de les imprimer dans le fablab le plus proche de chez vous. Le support a été découpé et gravé dans une plaque de plexiglas grâce à une découpeuse laser.

Une amélioration rendant le jeu plus interactif est d'ajouter un retour haptique dans les manettes à chaque fois que la balle touche la raquette. Pour cela il est possible d'utiliser un vibreur dont le fonctionnement est très simple : il suffit de connecter le Vcc au 3,3V, GND au GND et le SIG à une sortie de la carte. Le vibreur s'active alors en utilisant la méthode on() et s'arrête avec la méthode off().

Arnaud Aujon Chevallier
Ingénieur logiciel dans l'Opensource.
Maker passionné et membre actif de La Bobine, Fablab Auch Gers.



SinS: labyrinthe de mémoire visuelle et auditive

Principe du jeu

Vous incarnez un héros qui doit retrouver les clés de 10 coffres à trésors cachées dans un immense labyrinthe. Dans chaque pièce où vous entrez, il y a 4 nouvelles portes (verte, bleue, jaune et rouge). Vous avez un guide qui ouvre le chemin et vous montre les 3 prochaines portes à ouvrir pour trouver une première clé. Si vous réussissez, votre guide vous montre alors les 3 portes suivantes à ouvrir y compris les 3 premières portes déjà empruntées, et ainsi de suite... Le jeu se termine quand vous avez les 10 clés, ou bien quand vous vous trompez et êtes perdu(e) dans le labyrinthe.

Plusieurs niveaux de difficultés sont proposés:
- Vert : facile. De nouvelles séquences de 3 portes à ouvrir sont rajoutées au fur et à mesure que vous trouvez les clés, sans modifier les précédentes déjà mémorisées par le guide. Chaque porte est associée à un son.

- Bleu : moyen, idem que le mode vert, mais sans les sons. Il faut se fier uniquement à sa mémoire visuelle.

- Jaune: difficile. Dans ce mode, la séquence de 3 portes à ouvrir est réinitialisée au hasard depuis le début à chaque fois que vous avez trouvé une clé. Les sons sont activés.

- Rouge: expert. Idem que le mode Jaune, mais sans les sons.

Matériel nécessaire

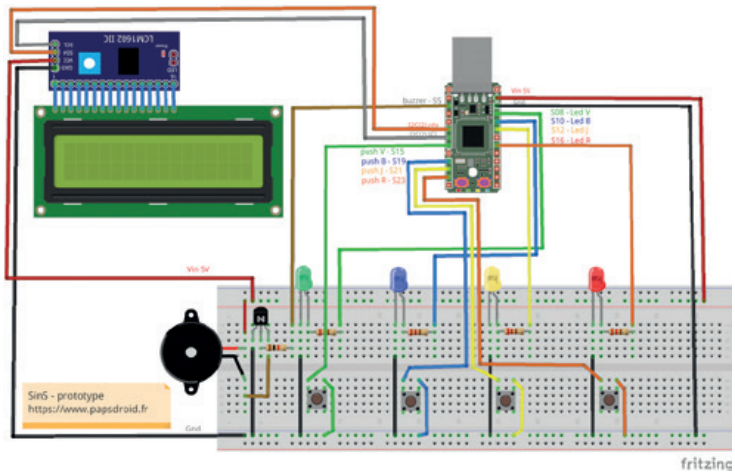
Il vous faudra réunir:
1 PYBStick26 Standard
4 leds 5mm (Verte, Bleue, Jaune, Rouge)
4 résistances 220 ohms (pour les leds)
1 petit transistor NPN (TO92)
1 résistance 1k ohms (pour le transistor)
1 petit buzzer passif
4 petits boutons poussoirs 6mm
1 écran LCD 1602 avec backpack I2C
+ câbles de connexion souples colorés mâle/mâle pour la platine d'essai

Circuit électronique : Schéma

Le prototypage sur une platine d'essai est facile à réaliser. Il faut penser à disposer les câbles de connexion judicieusement éloignés des boutons poussoirs pour ne pas qu'ils gênent. Vous trouverez les sources du projet sur Github:https://github.com/papsdroidfr/SinS. Cet été je compte concevoir un circuit imprimé sur lequel vous pourrez y souder les composants, ce qui permettra d'éviter tous les câbles.

Dans ce projet les portes sont matérialisées par des leds (verte, bleue, jaune et rouge), chacune étant commandée par une sortie de la PYBStick26 avec une petite résistance de 220Ω. Un des grands avantages de la PYBStick26 est qu'elle offre plusieurs sorties PWM (Pulse With Modulation) qui permettent de générer un signal d'une fréquence de 500hz dont on peut contrôler la durée du signal HIGH généré lors de chaque impulsion. En modulant cette durée de signal HIGH (exprimée en %) on arrive à, plus ou moins, diminuer l'intensité d'éclairage de la led, ce qui permet d'allumer ou éteindre les leds avec de jolis effets de variation au lieu du brutal on/off.

Un petit Buzzer passif va être utilisé pour générer les notes de musique liées à l'ouverture de chaque porte. Un transistor NPN va commander le passage du courant à travers le buzzer: la base du transistor étant reliée à une sortie de la PYBStick26 avec une résistance de 1 kΩ ohm en série. Pour contrôler le volume sonore nous allons encore une fois utiliser une sortie PWM de la PYBStick26, et cette fois nous allons modifier la fréquence de pulsation : 440 Hz vont permettre de générer une note de La, 523 Hz une note de Do, 659 Hz : Mi, 784 Hz : Sol. Les bluesmen reconnaîtront avec ces 4 notes un accord de Lam7.



Montage du projet

Un écran LCD de 2 lignes de 16 caractères est utilisé pour afficher les messages du jeux. Cet écran doit être connecté à un backpack I2C, ce qui offre non seulement l'avantage de le commander l'écran avec 4 fils (Vcc, Gnd, SDA, SCL) au lieu de 16 , mais aussi de considérablement simplifier la programmation via l'utilisation d'une bibliothèque MicroPython. Ainsi l'affichage d'un message sur un écran LCD est aussi simple que lcd.print('Hello world!')

Enfin quelques boutons poussoirs, un pour chaque porte, vont servir d'interface utilisateur pour capter les réponses du joueur. Au premier coup d'œil avec le schéma que je propose, tout électronicien devrait réagir : «Mais il est fou ce type ?!» il n'y a aucune résistance de protection sur les boutons poussoirs, les sorties sont reliées à rien du tout quand le bouton est en position «Off» et court-circuite direct à la masse en position «On». De plus il n'y a aucun système de filtrage pour limiter tous les «faux-rebonds» éléctro-mécaniques que génèrent à coup sûr de tels boutons poussoirs. Et pourtant je confirme que ce schéma est bon. En effet d'une part la PYBStick26 contient déjà des résistances de protection en configuration «PULL-UP» que l'on peut activer ou non sur ses entrées. En les activant, le schéma, que je propose, va relier l'entrée via une résistance interne de 10kΩ à +3.3V: donc pas besoin d'en rajouter une elle est déjà présente. D'autre part, pour ce qui concerne le filtrage anti-rebond (en appuyant sur le bouton poussoir, pendant un laps de temps court de quelques millisecondes, plusieurs connexions/déconnexions vont se faire, générant ainsi plusieurs passages on/off/on/off…), il y a deux façons de procéder : soit on résout le problème de manière électronique à l'aide d'un filtrage passe-bas composé de condensateurs et de résistances (on parle de filtrage RC), soit on résout le problème de manière logicielle en prenant en compte qu'il faut attendre un délai de stabilisation avant d'interpréter le signal. C'est ce que je propose de faire en MicroPython, donc pas besoin de filtrage RC.

Scripts MicroPython

Plusieurs scripts doivent être installés à la racine du lecteur PYBFLASH qui apparaît sur votre ordinateur lorsque vous branchez la PYBStick26 sur un port USB libre. Toutes Les sources sont récupérables sur mon Github https://github.com/papsdroidfr/SinS.

Les deux bibliothèque pwm.py et lcd2ic.py ont été développées par MCHobby. Elles permettent respectivement de prendre le contrôle d'une sortie PWM que nous allons utiliser pour les leds, et de contrôler un afficheur LCD avec son backpack I2C.

Les autres scripts MicroPython orientés objet sont des classes qui permettent de gérer les éléments du jeux.

leds.py

Ce script est composé de deux classes: Led() et RackLeds(). En instanciant un objet led de la classe Led(), des méthodes vont permettre d'allumer ou d'éteindre une led brutalement : led.on() et led.off(), ou avec un effet variateur: led.smooth_on(), led.smooth_off(). La méthode led.clignote(n) va faire clignoter la led n fois (avec un effet variateur)

Un objet de la classe Rackleds représente le rack de 4 leds du jeux, identifiées par 4 codes couleur respectifs 'V', 'B', 'J', 'R'. Les méthodes permettent d'accéder à une led précise rackleds.get_led(color) et, de les allumer toutes en même temps avec des effets de variation, clignotement etc. Enfin les méthodes rackleds.win() et rackleds.loose() vont faire clignoter la led Verte ou la Rouge quand on a gagné ou perdu.

buzzer.py

Dans ce script nous gérons la classe Buzz() qui va gérer le buzzer du jeu. Une instance buzz de la classe Buzz() est composée de méthodes lui permettant de jouer des notes et des animations musicales. Ainsi buzz.play() et buzz.mute() vont activer ou mettre en mute le buzzer. Le volume sonore est quand à lui défini dans la variable buzz.level (par défaut à 25%). La méthode buzz.chg_freq(freq) va modifier la fréquence (440 pour jouer un La, 523 un DO etc.). La méthode buzz.buzz_id(color) va jouer une note

en fonction de la couleur choisie. Enfin buzz.welcome_sound() et buzz.loose_sound() vont jouer des animations musicales qui portent bien leur nom.

buttons.py

À l'image du script leds.py, ce script regroupe deux classes Button_led() et RackButtons() pour les boutons poussoirs qui commandent chaque led.

C'est dans un objet button instancié de la classe Button_led() que l'on gère de manière logicielle la capture d'un appui sur le bouton poussoir par l'utilisateur sans se faire piéger par les faux rebonds électro-mécaniques. Pour se faire, nous allons déclarer dans notre objet une variable self.extint qui est une instance de la classe Extint. Elle permet via interruption logicielle d'appeler une fonction self.callback lorsque l'entrée configurée en PULL-UP change d'état. Dans cette fonction button.callback(line), la première chose à faire est d'attendre 50 à 100ms pour que le signal soit stabilisé, puis de tester à l'issue de ce délai que le bouton est toujours appuyé. En positionnant le délai à 50ms j'arrive à éviter l'intégralité des faux rebonds, et ce délai de 50ms est invisible pour l'utilisateur. Il faut utiliser un délai de plus de 150ms pour que ça commence à se ressentir.

Enfin l'objet RackButtons va contenir deux méthodes rackbuttons.activate_buzzer() et rackbuttons.desactivate_buzzer(). Elles vont activer ou désactiver le buzzer du jeux lorsque que l'utilisateur appuie sur les boutons poussoirs.

lcd.py

Ce script va gérer la classe Lcd() du jeux avec laquelle nous allons pouvoir écrire des messages sur l'écran LCD avec des caractères personnalisés. Les méthodes lcd.clear(), lcd.msg_centre(lig1, lig2) et lcd_msg(lig1, li2) vont respectivement effacer l'écran, et afficher deux lignes de texte centrées ou non.

Les deux méthodes lcd.write_char(label, pos) et lcd_write_key(n, l) vont respectivement afficher un caractère spécial et une série de n dessins de clés centrées à droite sur la ligne ligne l.

La bibliothèque lcd2ic.py, développée par MCHobby, permet de créer des caractères spéciaux. Ils sont en définition 8 lignes de 5 pixels, donc on utilise pour cela une liste de 8 codes binaires de 5 bits. Pour générer les codes binaires ce site est très utile: https://maxpromer.github.io/LCD-Character-Creator

main.py

C'est le script principal que la PYBStick26 exécute au démarrage. Elle est composée de la classe Jeux_sins() qui va instancier tous les objets dont nous avons besoin et gérer la boucle principale du jeux.

Au tout début, un message d'accueil avec une musique apparaît, ainsi qu'une petite animation colorée des leds. Le joueur est ensuite invité à choisir un mode de jeux en appuyant sur l'un des boutons vert, bleu, jaune ou rouge.

Le mode choisi est alors confirmé sur l'écran.

Le joueur est invité à :
- observer la séquence de portes (avec ou sans son, selon le mode choisi) à ouvrir, indiquée par le guide.
- reproduire la même séquence, à l'aide du rack de boutons poussoirs.

S'il a ouvert les portes dans le bon ordre sans se tromper: la led verte va clignoter 3 fois, une clé est trouvée, et si le record du mode est battu il est aussi mis à jour.

S'il s'est trompé, il est alors perdu dans le labyrinthe ! Une musique de la loose est jouée, la led rouge clignote 3 fois, et un message est indiqué sur l'écran. Le jeu retourne alors sur l'écran de choix du mode.

Si vous avez trouvé les 10 clés: bravo et respect ! Le jeu retourne sur l'écran de choix du mode.

Olivier Bersot
Ingénieur en informatique et électronique avec une spécialisation en IA. Actuellement responsable de projets informatiques dans la finance. Il passe une grande majorité de sontemps libre à partager ses tutoriels dans l'univers passionnant de l'électronique embarquée sur son site https://www.papsdroid.fr

MC HOBBY
MICROCONTRÔLEUR HOBBY

shop.MCHobby.be

+33 (0) 359 260 058
+32 (0) 2 387 24 78
support@mchobby.be
fb.mchobby.be
blog.mchobby.be

Raspberry-Pi et ODROID

Hardkernel

Apprendre le codage, l'informatique et l'électronique avec les nano-ordinateurs les plus célèbres du monde.

Raspberry Pi
APPROVED RESELLER

Arduino et compatibles

adafruit
feather

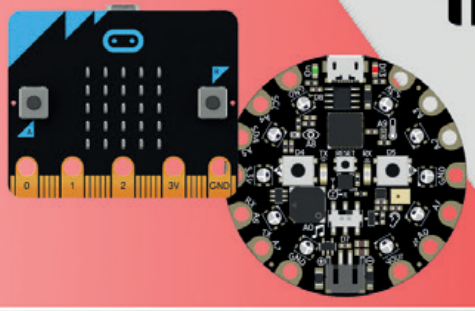
Programmer des objets en C simplifié avec les plateformes microcontrôleur les plus documentées du Net.

MicroPython

MicroPython

Programmer vos microcontrôleurs et projets électroniques avec des scripts Python. Un langage facile à apprendre et à maîtriser sur une large gamme de cartes.

Micro:bit et Circuit Playground



Ouvrir le monde de la programmation et de l'électronique aux plus jeunes avec la programmation par bloc.

Wiki documentaire

De la documentation gratuite, en Français, dans le plus pur esprit open-source.

Le savoir se partage avec passion !