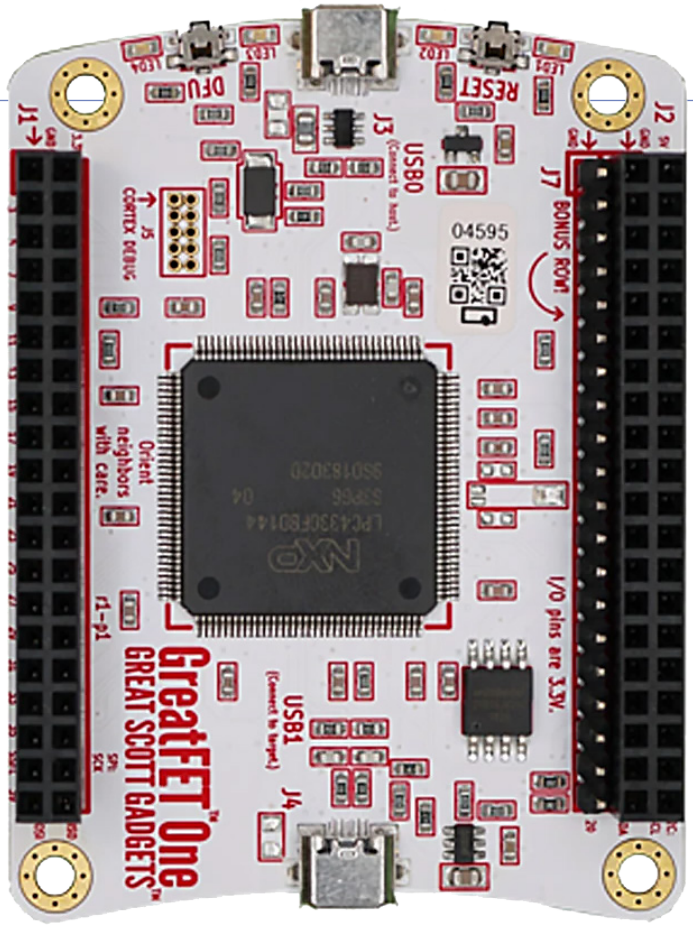
carte d'interface GreatFET One



Farfouiller sous les jupes de l'USB… pourquoi se gêner ?

Tam Hanna (Slovénie)

Si vous cherchez un moyen rapide d'effectuer des tâches simples de mesure ou de commande ou de régulation avec un PC, vous trouverez une solution techniquement intéressante et facile à utiliser pour les Pythonneurs.

Autrefois, avant que les PC parlent USB, nous passions notre temps culbuter les bits du port parallèle et à communiquer par RS-232. Aujourd'hui, nous sous-traitons cette activité à un processeur auquel nous parlons USB. La carte GreatFET One, de *Great Scott*, non seulement parle USB elle aussi, mais vous permet de farfouiller sous les jupes de l'USB avec du code Python.
Avant que l'USB devienne le port de communication des PC, vous passiez probablement par le port parallèle de votre PC pour l'interfacer avec le reste du monde. Cette possibilité a disparu. L'interaction avec le port de jeu, l'interface série ou l'interface imprimante était relativement facile comparée à l'usage de l'interface USB qui demande le support de puces spécifique et de bonnes connaissances en programmation.
Avec sa carte GreatFET One, *Great Scott Gadgets* propose une carte de développement USB qui vous permet de contrôler les signaux GPIO d'un PC comme vous pouviez tripatouiller les broches d'un port parallèle de PC au bon vieux temps.

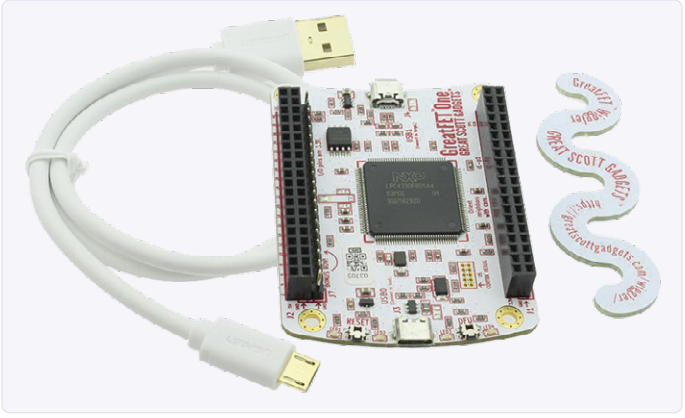


Figure 1. Le GreatFET One juste sorti de sa boîte !

C'est quoi ce truc ?

GreatFET One est conçu comme du matériel ouvert ; le logiciel est décrit dans un wiki [**1**], les fichiers du matériel utilisables par KiCad sont également disponibles sur Github [**2**]. Si vous achetez le *GreatFET One*, vous recevez ce que montre la **fig. 1** - en plus de la carte (prête à l'emploi et équipée de tous les connecteurs), vous recevez également un câble USB et un outil Wiggler de type serpent pour séparer en toute sécurité les cartes d'extension enfichables. La documentation relative à l'API d'interaction principale appelée *LibGreat se* trouve sous [**3**].
Et si vous vous demandez pourquoi il est question d'un transistor à effet de champ, sachez qu'ici FET signifie *Flash Emulation Tools*. Vous devinez l'idée (originale) sous-jacente ? C'est d'avoir un dispositif USB capable de communiquer avec un PC et d'échanger avec lui des signaux comme s'il s'agissait d'une clé USB. Une fois que vous avez ça, vous pouvez bien sûr utiliser votre émulateur pour parler avec le PC et lui faire croire que vous êtes n'importe quel dispositif USB… vous ne trouvez pas qu'on dirait un outil de hacker ?

Installation rapide !

L'environnement de programmation Python est un bon choix pour un développeur de système qui veut bricoler du code pour tester le matériel. J'ai préparé cette démo rapide en quelques étapes avec

```
tamhan@TAMHAN18:~$ greatfet info
Found a GreatFET One!
  Board ID: 0
  Firmware version: 2018.12.1
  Part ID: a0000a30584f66
  Serial number: 000057cc67e6303a6757
```

Figure 2. La carte GreatFET One a été reconnue.

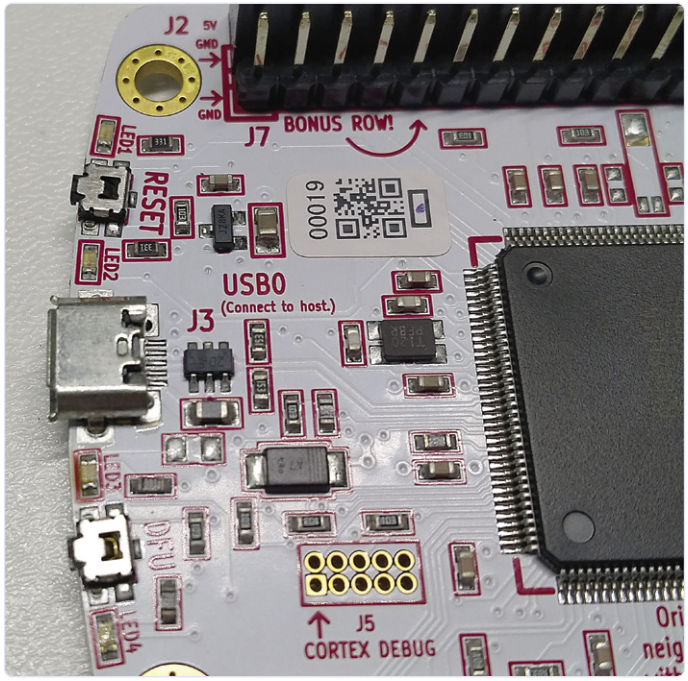


Figure 3. Un étiquetage clair est un atout. Ce port USB se connecte à l'hôte.

Ubuntu 18.04 LTS. Si vous préférez un environnement Windows, vous trouverez les instructions d'installation nécessaires en ligne [**4**].
L'installation de la bibliothèque commence par l'utilisation du gestionnaire de paquets pip3 :

```
tamhan@TAMHAN18:~$ pip3 install --upgrade --user
   greatfet
```

Python a sa propre gestion de paquets depuis un certain temps déjà, avec laquelle nous pouvons charger le paquet GreatFET, y compris certaines de ses dépendances natives. Il est important d'utiliser la version *pip3* - le *pip* destiné à Python 2.X ne fonctionne pas. Dans l'étape suivante, nous vérifions si notre compte utilisateur est déjà membre du groupe PlugDev :

```
tamhan@TAMHAN18:~$ groupes tamhan
tamhan : tamhan adm dialout cdrom sudo dip plugdev
   lpadmin sambashare kvm pico
```

Avant de connecter l'ordinateur de processus à l'ordinateur, saisissons les commandes de mise à jour des règles udev de Linux :

```
tamhan@TAMHAN18:~$ sudo wget https://raw.
   githubusercontent.com/greatscottgadgets/greatfet/
   master/host/misc/54-greatfet.rules -O /etc/udev/
   rules.d/54-greatfet.rules
tamhan@TAMHAN18:~$ sudo udevadm control -reload-rules
```

Cela en vaut la peine ; il peut maintenant reconnaître un ordinateur de processus connecté (**fig. 2**) en saisissant la commande `greatfet info`.
Dans la mesure du possible, il est toujours utile de vérifier que tout nouveau kit utilise la dernière version du micrologiciel. Pour le GreatFET, nous pouvons entrer la commande suivante :

```
tamhan@TAMHAN18:~$ gf fw --auto
```

Dans les étapes suivantes, nous utilisons la version v2020 1.2 du microprogramme.
Les systèmes informatiques à processus combinatoire souffrent généralement d'une faible largeur de bande entre les différents modules. Pour un simple test, nous pouvons écrire une petite routine pour faire basculer une broche GPIO sur la carte afin de voir à quelle vitesse nous pouvons faire avancer les choses. Pour cela, nous utiliserons un fichier . py, que vous pourrez facilement éditer dans le code de *Visual Studio* :

```
tamhan@TAMHAN18:~/greatfetspace$ code worker.py
tamhan@TAMHAN18:~/greatfetspace$ python3 worker.py
```

L'IDE de *Microsoft* présente un aspect positif : le terminal revient immédiatement après l'activation - l'interpréteur Python-3 peut donc être appelé au même endroit.
Dans l'étape suivante, nous pouvons créer un objet GreatFET et créer un objet `pin` selon le schéma suivant :

```
from greatfet import GreatFET

gf = GreatFET()
pin = gf.gpio.get_pin('J1_P4')
pin.set_direction(gf.gpio.DIRECTION_OUT)
```

Une autre caractéristique intéressante est l'attribution des broches ; en travaillant avec un RPi, il peut y avoir une confusion concernant l'attribution des broches dans le logiciel et son emplacement physique sur la carte.
Great Scott contourne le problème en attribuant une étiquette unique à chaque connecteur. La chaîne utilisée ici fait alors référence, par exemple, à la broche 4 du connecteur J1 (**fig. 3**).
Cette opération est suivie d'une boucle sans fin qui fait basculer la broche et produit une forme d'onde :

```
while 1 == 1:
    pin.write(True)
    pin.write(False)
    pin.write(True)
    pin.write(False)
```

Les figures 4 et 5 montrent le signal de sortie et nous voyons que les périodes haute et basse de l'onde carrée sont approximativement égales. Le temps de commutation n'est pas particulièrement rapide ni stable.

En expérimentant, n'oubliez pas que la carte utilise une logique de 3,3 V ; le µC ne survivra probablement pas si vous essayez d'y connecter des signaux de 5 V.

Expérimentations

GreatFET One est livré avec un convertisseur analogique-numérique, plus facile à activer avec l'une des nombreuses aides en ligne de commande. La broche d'entrée par défaut du convertisseur analogique-numérique est J2_P5 :

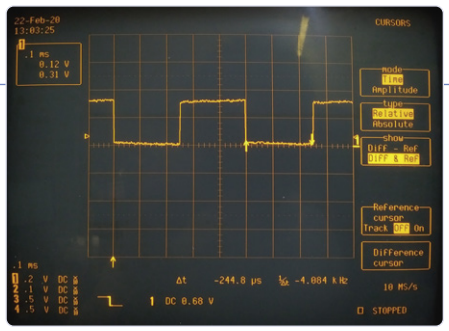


Figure 4. La forme d'onde de sortie affichée sur un oscillo…

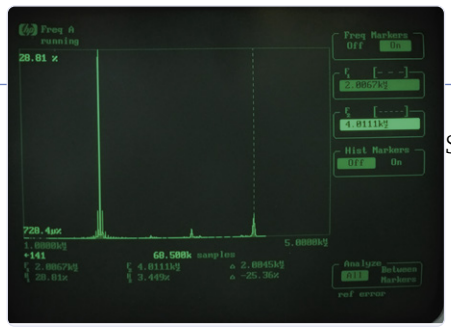


Figure 5 : … et sur un analyseur de domaine de modulation.

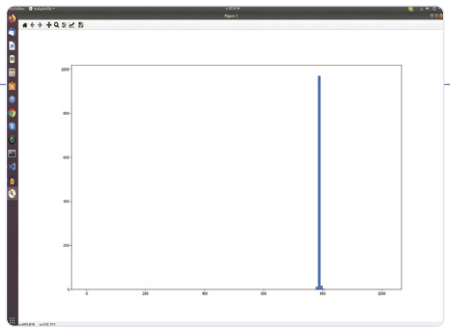


Figure 6. Histogramme simple.

```
tamhan@TAMHAN18:~/greatfetspace$ greatfet adc
3.18076171875V
```

Pour tester les performances du code Python sur le matériel, je devrai donc charger à la fois la *NumPy-Library* et la *MatPlotLib* à ce stade. Si ces deux bibliothèques manquent, installez-les manuellement. Au cours de la prochaine étape, j'utiliserai une source de référence de tension à température contrôlée. En attendant, j'utiliserai le programme suivant.
Dans un premier temps, je charge quelques bibliothèques comme d'habitude :

```
import numpy as np
from matplotlib import pyplot as plt
from greatfet import GreatFET
```

La fonction de lecture de l'ADC mise en œuvre est une question complexe. Pour plus de commodité, je crée la carte NumPy - l'utilisation de l'API n'est pas très efficace :

```
gf = GreatFET()

store = np.array(gf.adc.read_samples(1))
i = 0
while i < 1000:
    store = np.append(store, gf.adc.read_samples(1))
    i=i+1
```

À ce stade, il existe deux procédures différentes. La première méthode consiste à créer des *Bin-Ranges*, que je passe ensuite à la fonction `histogram` de NumPy :

```
binrange = range(0,1024,8)
hist, bins = np.histogram(store, binrange)
```

L'effort exigé par cette méthode est récompensé par le retour du tableau dans lequel la fréquence des valeurs dans les différents `bins` peut être collectée. Le transfert des `bins` n'est alors qu'une copie du paramètre `binrange`.

Si vous souhaitez uniquement afficher les diagrammes directement, vous pouvez sous-traiter l'affichage de l'histogramme directement à MatPlotLib selon le schéma suivant :

```
plt.hist(store, bins)
plt.show()
```

L'histogramme (**fig. 6**) vaut la peine — il faut ignorer le bruit électromagnétique.
Enfin, il convient de noter que GreatFET offre également un générateur de motifs (patterns) et même une fonction d'exportation des informations Sigrok-Log. Ces deux fonctions peuvent être activées selon le schéma suivant à l'aide de commandes :

```
tamhan@TAMHAN18:~/greatfetspace$ greatfet logic -p
   out.sr -f 2M -n 4
```

La vitesse maximale possible dépend du matériel utilisé — voir la discussion intéressante sur ce sujet [5].

Récapitulons

La documentation est un peu maigre, mais avec tout le potentiel de la carte comme outil pour farfouiller sous les jupes de l'USB, j'imagine que cette carte va faire un carton dans la communauté des makers et des hackers. Je n'ai encore pu exploiter qu'une petite partie de ce dont elle est réellement capable, son potentiel est vaste.

200124-04

LIENS

[1] **logiciel** : https://github.com/greatscottgadgets/greatfet/wiki
[2] **description du matériel** : https://github.com/greatfet-hardware/azalea
[3] **docu LibGreat** : https://github.com/greatscottgadgets/libgreat
[4] **installation sous Windows** : https://greatscottgadgets.github.io/greatfet-tutorials/windows.html
[5] **taux d'échantillonnage** : https://github.com/greatscottgadgets/greatfet/issues/286