horloge NTP en CircuitPython

pourquoi utiliser ce langage de programmation ?

Michael Bottin (France)

Nous découvrons l'une des alternatives utiles à Python, conçue pour être suffisamment simple pour fonctionner sur des microcontrôleurs : CircuitPython. Ici, nous réalisons un projet avec un Raspberry Pi Pico W, qui récupère l'heure d'un serveur NTP et l'affiche sur un écran LCD.

Python est un des langages de programmation parmi les plus populaires. C'est un langage interprété ce qui rend la phase de développement plus rapide contrairement à un langage comme le C/C++ qui nécessite d'être compilé avant son exécution. En revanche, cette absence de compilation oblige à ce qu'un environnement d'exécution soit installé sur la cible au préalable. Depuis plus de 30 ans, ce langage continue son avancée dans le monde des PC et du cloud. Il couvre des domaines comme le calcul scientifique, le développement web (*back-end*), le développement logiciel, l'écriture de scripts système, le Machine Learning, le Big Data (analyse et visualisation de données). Il est utilisé par de nombreuses compagnies comme Google (« Python où nous pouvons, C ++ où nous devons »), Youtube, Instagram, Spotify, Intel, Facebook, Dropbox, Netflix, Pixar, Reddit… Il était donc prévisible que ce langage fasse son entrée dans le monde de l'électronique embarquée. Mais pour cela, il a fallu créer une version allégée pour qu'elle puisse tourner sur un microcontrôleur mais également une version tournée vers le hardware pour qu'elle puisse piloter les différents périphériques disponibles dans un microcontrôleur (GPIO, PWM, I²C, SPI, UART…). Deux versions ont vu le jour :

- MicroPython développé par Damien George dès 2013 [1]
- CircuitPython développé par Limor Fried (PDG d'Adafruit), Scott Shawcroft et beaucoup d'autres contributeurs dès 2017 [2].

Ces deux langages partagent tous deux la même implémentation du langage Python (CPython). CircuitPython est un dérivé open-source de MicroPython. Toute évolution de MicroPython est rapidement répercutée sur CircuitPython. Il y a déjà eu plusieurs articles sur l'utilisation de MicroPython dans cette revue, mais à ma connaissance, aucun sur CircuitPython.

Quel est l'intérêt de CircuitPython vis-à-vis de MicroPython ?

Soyons clair, pour un informaticien habitué au Python souhaitant utiliser ce langage dans le monde de l'embarqué, aucun. Il est même préférable d'utiliser dans ce cas MicroPython. Mais pour un maker, un étudiant ou un enseignant n'ayant pas de connaissances particulières en Python, le choix de CircuitPython peut être intéressant. La chaîne de développement est très conviviale, le langage apporte une couche d'abstraction supplémentaire et la majorité des informations liées à ce langage (documentation, bibliothèques, tutoriels, forum…) sont centralisées sur le web.
C'est pour cette raison que j'ai choisi d'utiliser CircuitPython pour cet article, afin d'offrir aux lecteurs désirant démarrer en Python sur microcontrôleur une porte d'entrée facile. La communauté autour de CircuitPython est également très dynamique. De nouvelles versions sortent régulièrement (version actuelle 8.x), de nouvelles bibliothèques également. Un grand nombre de cartes populaires à base de microcontrôleur supportent ce langage. CircuitPython peut également être employé sur des SBC comme Raspberry Pi, BeagleBone, Odroid ou encore Jetson via l'API Blinka.

Ambition de cet article

Cet article constitue une découverte de l'utilisation du langage CircuitPython à travers une simple application.
Je vais donc vous présenter comment vous pouvez très rapidement mettre en œuvre une horloge synchronisée sur le web grâce à CircuitPython. Le projet propose également de choisir l'heure locale d'une vingtaine de pays dans le monde. L'horloge se synchronisera grâce à une connexion wifi

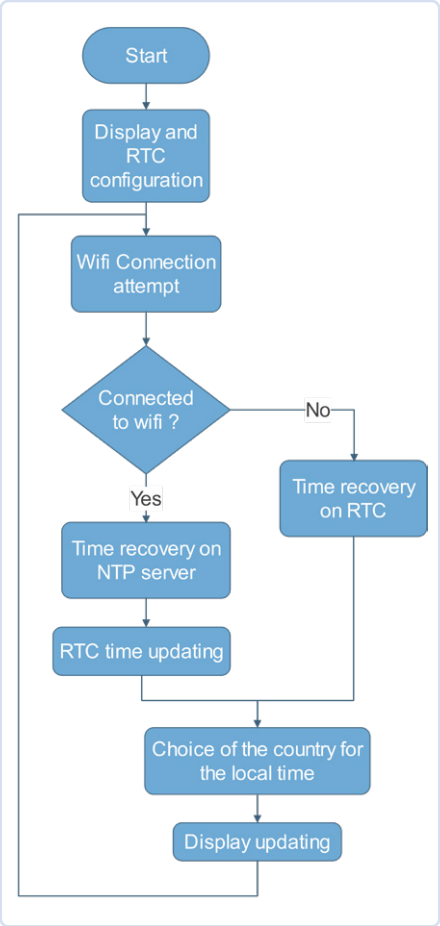

Figure 1. Algorigramme général du fonctionnement de l'horloge NTP.

(préconfigurée au niveau SSID et mot de passe) et récupérera son horodatage via un serveur NTP (*Network Time Protocol*) dédié. Cette horloge sera aussi en mesure de maintenir l'heure à jour en absence de réseau grâce à l'horloge temps réel (RTC) embarquée qui sauvegardera l'heure réseau lorsque celui-ci est disponible et qui fournira elle-même l'heure lorsque le réseau est indisponible. Une pile garantira également le maintien de l'heure de la RTC à jour en absence d'alimentation.

L'algorigramme général du fonctionnement de l'horloge est présenté à la **figure 1.**

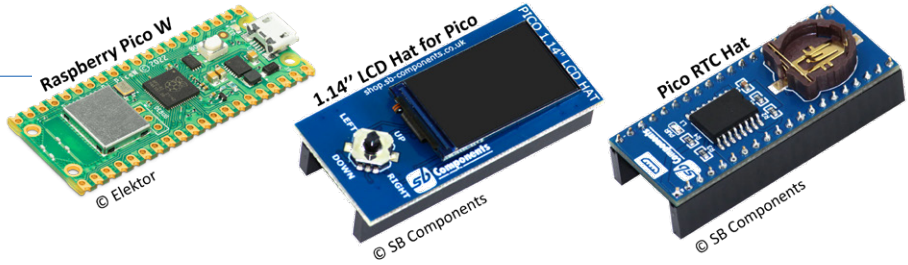

Figure 2. Modules utilisés.

Présentation du matériel

Un grand nombre de cartes de développement (et donc de microcontrôleurs) peuvent être facilement programmées avec le langage CircuitPython. J'ai choisi ici la carte Raspberry Pico W à la fois car elle est très populaire, peu coûteuse, disponible chez un grand nombre de revendeurs dont Elektor [3] et qu'elle dispose d'une interface Wifi. Toutefois, elle ne dispose ni d'un écran d'affichage, ni d'une horloge temps-réel. Il est donc nécessaire d'ajouter quelques composants supplémentaires pour notre projet.

Plutôt que de réaliser une carte dédiée, j'ai choisi ici d'assembler des modules du commerce pour réaliser ce projet. Trois modules empilés suffisent au projet (cf. **figure 2**) :

- Un module Raspberry Pico W (attention à bien prendre la version W qui dispose d'une interface Wifi)
- Un module d'affichage avec un écran LCD 240x135 pixels et un joystick pour la navigation (SB Components [4])
- Un module avec une horloge temps réel DS3231 (SB Components [5])

Remarque : le microcontrôleur RP2040 dispose également en interne d'une horloge temps réel RTC. Vous pourriez l'utiliser à la place du module *Pico RTC Hat*, cependant la RTC du RP2040 n'est pas aussi précise dans le temps et ne dispose pas d'une connexion vers une batterie de sauvegarde.

Schéma de câblage

Même si l'on utilise des modules du commerce qui facilitent la mise en œuvre de la partie hardware, on a besoin de connaître les connexions entre ces modules afin de pouvoir écrire les lignes de code du programme de notre application. La **figure 3** présente les interconnexions réalisées entre les différents modules ainsi que les noms donnés à ces connexions, noms qui seront réutilisés dans le programme. (Notez que le brochage indiqué dans la documentation GitHub de SB Components est incorrect au moment de la rédaction de cet article, mais leur code source indique l'affectation correcte des broches illustrée dans la **figure 3**).

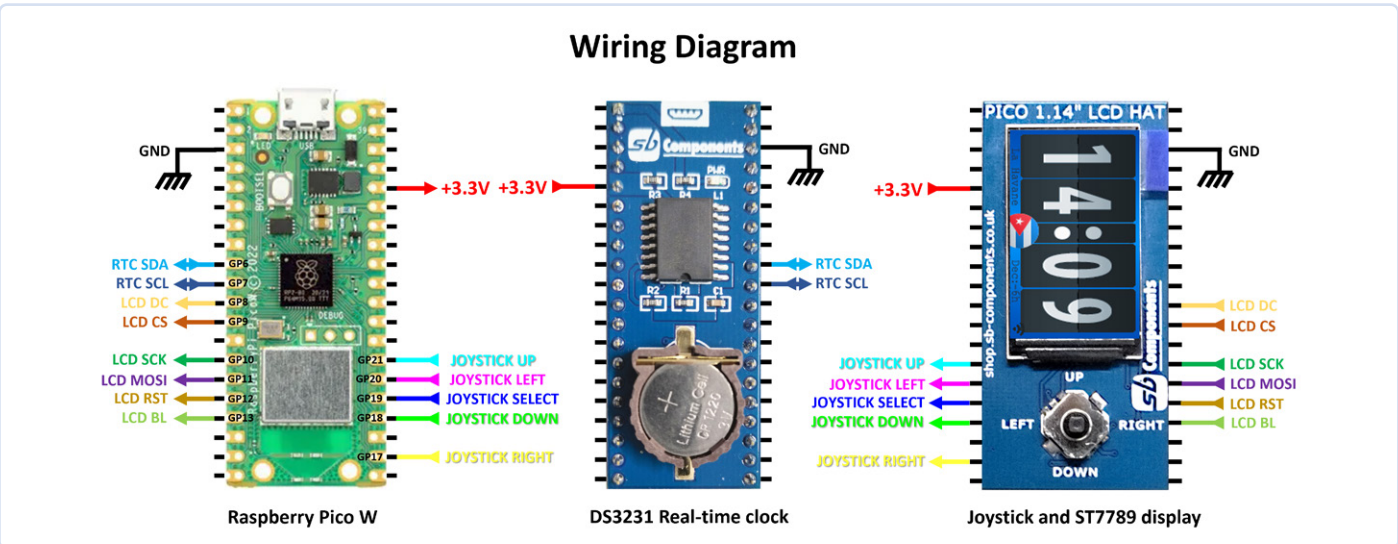

Figure 3. Interconnexions réalisées entre les différents modules.

Figure 4. Site web de CircuitPython.

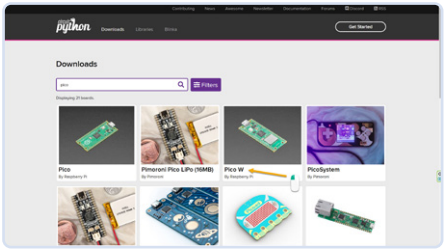

Figure 5. Sélection du Pico W parmi les cartes compatibles avec CircuitPython.

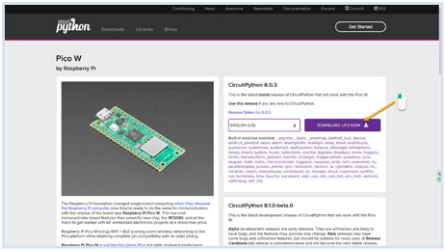

Figure 6. Téléchargement du fichier .UF2.

Installation CircuitPython

Le microcontrôleur RP2040 de la Raspberry Pico W n'intègre pas CircuitPython lorsque vous achetez le module. Avant toute chose, vous allez donc devoir le « flasher » sur la carte.

Tout comme il y a des versions différentes de compilateur C selon la cible utilisée, il y a autant de versions de CircuitPython qu'il y a de cartes qui le supportent. Au moment où ces lignes sont écrites, plus de 380 cartes du commerce sont compatibles avec CircuitPython. Il y a bien évidemment la plupart des cartes récentes d'Adafruit, mais également de Pimoroni, d'Expressif, de SeeedStudio, de Waveshare, de LilyGo, de Cytron, de DFRobot, de Wiznet…

La majorité d'entre elles utilisent des microcontrôleurs de chez Microchip (SAMD21/SAMD51), de chez Nordic (nRF52840), d'Expressif (ESP32) et de la Fondation Raspberry (RP2040). Vous devez donc télécharger la version de CircuitPython qui correspond à la carte Raspberry Pico W :

- Dans un navigateur, rendez-vous sur le site web de CircuitPython [6] (**figure 4**).
- Cliquez sur le lien *Downloads* puis recherchez la carte Pico W (**figure 5**).
- Téléchargez alors le fichier UF2 disponible sur la page affichée (**figure 6**).

Le microcontrôleur RP2040 de la carte Raspberry Pico W contient déjà un bootloader qui facilite l'installation de CircuitPython dans ce dernier. Voici la démarche pour installer la version de CircuitPython que vous venez de télécharger :

1. Déconnectez la carte Raspberry Pico W de votre ordinateur de travail
2. Maintenez le bouton *BOOTSEL* appuyé de la carte Raspberry Pico W pendant que vous la connectez à votre ordinateur via un câble micro-USB (compatible avec un transfert de données).
3. Un lecteur *RPI-RP2* doit apparaître dans votre explorateur de fichier.
4. Copiez alors le fichier UF2 que vous avez téléchargé sur le lecteur *RPI-RP2*.

Vous n'avez rien d'autre à faire. Une fois CircuitPython flashé sur la carte Raspberry Pico W, un lecteur *CIRCUITPY* doit apparaître dans votre navigateur.

Utilisation du lecteur *CIRCUITPY*

Voici quelques points importants à respecter lors de l'utilisation de ce lecteur *CIRCUITPY* :

- TIl s'utilise comme une clef USB. Vous pouvez ajouter, supprimer des fichiers/dossiers depuis votre explorateur de fichiers. Sa capacité représente la mémoire Flash disponible sur le microcontrôleur RP2040 pour y stocker votre code et vos ressources (images, audio…).
- Il n'y a pas de précaution à prendre lorsque vous branchez le Raspberry Pico W à votre ordinateur. **En revanche, il est plus que souhaitable de l'éjecter à la manière d'une clef USB lorsque vous voulez la débrancher ; faute de quoi, vous risquez de corrompre le système de fichiers.**
- En ouvrant le fichier *boot.txt*, vous pouvez contrôler quelle version de CircuitPython est installée sur la carte Raspberry Pico W.
- Dès que la carte Raspberry Pico W est alimentée via son port micro-USB (grâce à votre ordinateur ou un chargeur secteur), le code CircuitPython s'exécute. Mais seuls les fichiers nommés *code.py* ou *main.py* vont s'exécuter. Prenez bien soin de nommer votre fichier de travail *code.py*.
- Il est préférable (mais pas obligatoire) de copier les fichiers images dans un dossier *./images*, les fichiers de bibliothèques dans un dossier *./lib*,… Cela permet d'avoir une arborescence propre pour vos projets.

Utilisation de l'EDI

CircuitPython étant un langage interprété, il n'y a donc pas de phase de compilation. Si un fichier *code.py* contenant du code CircuitPython existe sur le lecteur CIRCUITPY, il s'exécute automatiquement. De ce fait, on pourrait tout simplement éditer ce fichier dans un éditeur de texte et l'enregistrer pour qu'il s'exécute. Toutefois, l'écriture d'un programme s'accompagne généralement d'une phase de débogage. L'utilisation d'un EDI permet alors d'accéder à des outils comme une console pour obtenir un retour sur ses erreurs.

L'EDI que l'on va utiliser ici est Mu Editor [7]. C'est une solution logicielle simple disponible gratuitement sous Windows, Mac OSX et Linux. Il est également possible d'utiliser Thonny, VS Code, Atom ou encore PyCharm en installant l'extension CircuitPython

La **figure 7** présente l'interface de l'EDI Mu Editor. La barre d'outils est réduite à l'essentiel :

- *Mode* : choix du langage de programmation. Veillez à bien être dans le mode *CircuitPython*.
- *New* : création d'un nouveau fichier vide
- *Load* : chargement d'un fichier existant. En mode *CircuitPython* et si votre carte Raspberry Pico W est bien branchée, le logiciel vous ouvre automatiquement le dossier du lecteur *CIRCUITPY*.
- *Save* : sauvegarde du fichier en cours (onglet actif). Un petit point rouge à côté du nom du fichier dans l'onglet vous indique qu'il y a eu des modifications depuis la dernière sauvegarde.
- *Serial* : affiche/masque la console. Il est préférable de toujours afficher la console (REPL) car c'est dans celle-ci que s'afficheront les messages d'erreurs ou les informations de votre programme.
- *Plotter* : affiche/masque un graphe déroulant permettant de visualiser des données numériques envoyées depuis le code.
- *Zoom-in* et *Zoom-out* : augmente ou réduit la taille de la police d'affichage.
- *Theme* : bascule entre un thème clair (*Day*), sombre (*night*) ou à fort contraste.
- *Check* : recherche les erreurs de votre code, même celles qui n'empêchent pas son exécution.
- *Tidy* : nettoie votre code en supprimant,

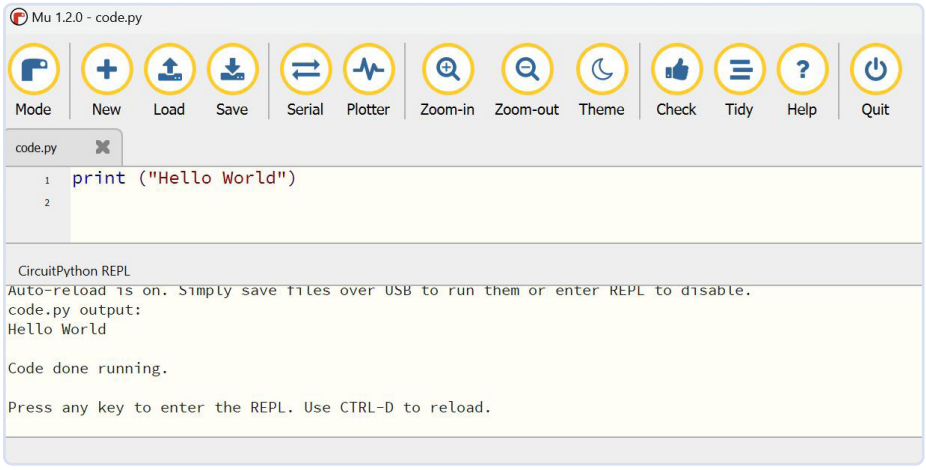

Figure 7. L'interface de l'EDI Mu Editor.

par exemple, les espaces inutiles ou les lignes vides inutiles.
- *Help* : aide en ligne.
- *Quit* : quitte l'éditeur.

Voici donc le déroulement classique du développement d'un programme dans l'EDI :
- Vous ouvrez le fichier *Code.py* du lecteur *CIRCUITPY*
- Vous modifiez le code dans la zone d'édition.
- Vous enregistrez les modifications du fichier.
- Vous observez le résultat de l'exécution dans la console. Vous repartez à l'étape n°2 si des erreurs sont survenues durant l'exécution.

Vous trouverez davantage d'informations à l'adresse suivante :
- *Start Here!* (concernant l'interface) [8]
- *What is a REPL?* (concernant la console REPL) [9]
- *Plotting Data with Mu* (concernant le graphe déroulant) [10]
- Raccourcis clavier [11]

Maintenant que ces informations sont données, vous allez pouvoir vous intéresser sur le projet, à savoir l'horloge NTP.

Test de la connexion wifi

CircuitPython gère le wifi en natif sur la carte Raspberry Pico W. Aucune bibliothèque supplémentaire n'est donc à installer. La connexion à un réseau wifi nécessite de fournir le SSID du réseau et son mot de passe. Pour éviter que ces informations apparaissent directement dans le code, CircuitPython utilise des variables d'environnement.
Un fichier *settings.toml* [12] situé à la racine du lecteur *CIRCUITPY* va donc contenir ces informations « secrètes ». On pourrait y stocker toute information confidentielle comme des clefs d'API. Dans notre cas, il ne contient que le *SSID* et le *PW* du réseau auquel vous souhaitez connecter votre Raspberry Pico W (cf. **figure 8**). (Remarque : ce fichier n'est pas éditable dans l'EDI, vous devez utiliser un éditeur de texte pour le créer et le remplir)
Le code pour se connecter au réseau est présenté dans le **listage 1.** Vous pouvez rapidement constater que le code est très compact :
- **Ligne 2** : importation de la bibliothèque *os* qui permet d'accéder aux

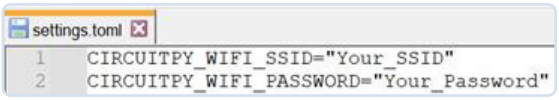

Figure 8. SSID et password dans le fichier settings.toml.

Listage 1. Code pour se connecter au réseau wifi

```
 1 # CircuitPython's own libraries
 2 import os
 3 import ipaddress
 4 import wifi
 5
 6 print()
 7 print("Connecting to Wi-Fi")
 8
 9 # connect to your Wi-Fi network with SSID/PASSWORD in 'settings.toml'
10 wifi.radio.connect(os.getenv('CIRCUITPY_WIFI_SSID'), os.getenv('CIRCUITPY_WIFI_PASSWORD'))
11
12 print("Connected to Wi-Fi")
13
14 # pings Google DNS server
15 ipv4 = ipaddress.ip_address("8.8.8.8")
16 print("Ping google.com: %f ms" % (wifi.radio.ping(ipv4)*1000))
```

Figure 9. Téléchargement des bibliothèques (la date figurant dans le nom du fichier sera évidemment différente.

variables d'environnement du fichier *settings.toml*.

- **Ligne 4** : importation de la bibliothèque *wifi* qui permet la connexion au réseau.
- **Lignes 6,7 et 12** : ces lignes ne sont pas obligatoires, elles permettent juste d'informer l'utilisateur dans la console.
- **Line 10** : une seule ligne suffit ici à la connexion physique au réseau.
- **Lines 3, 14–16** : ces lignes ne sont pas obligatoires, elles permettent de faire un petit test (*ping*) en interrogeant un serveur sur le web et ainsi vérifier que la connexion wifi fonctionne correctement puis afficher le résultat sur la console série.

Ajout des bibliothèques utiles

CircuitPython contient un grand nombre de bibliothèques fondamentales, mais il ne peut pas contenir toutes les librairies disponibles car cela saturerait inutilement la mémoire Flash du microcontrôleur. Il est donc nécessaire, selon les exigences de vos projets, d'installer des bibliothèques supplémentaires.

L'installation de bibliothèques supplémentaires se résume en CircuitPython à copier les fichiers correspondants sur le lecteur *CIRCUITPY*. Pour n'avoir à le faire qu'une seule fois, vous allez installer ici toutes les bibliothèques utiles pour la version finale du projet.

Voyons la démarche :

1. Créez un dossier *lib* sur votre lecteur *CIRCUITPY* s'il n'existe pas déjà.
2. Toutes les bibliothèques disponibles sont téléchargeables sous forme d'une archive unique sur le site de CircuitPython (remarque : elles sont aussi disponibles individuellement sur Github)
3. Sur la page du site, cliquez sur le lien *Libraries*.
4. Téléchargez alors l'archive correspondant à votre version de CircuitPython, 8.x dans notre cas (cf. **figure 9**).
5. Dézippez la où vous le souhaitez sur votre ordinateur.
6. Dans le dossier de l'archive dézippée, ouvrez le dossier *lib*.
7. Sélectionnez :
   a. Les dossiers :
      i. *adafruit_register*
      ii. *adafruit_imageload*
      iii. *adafruit_display_text*
   b. Les fichiers :
      i. *adafruit_debouncer.mpy*
      ii. *adafruit_ds3231.mpy*
      iii. *adafruit_ntp.mpy*
      iv. *adafruit_st7789.mpy*
      v. *adafruit_ticks.mpy*
8. Copiez l'ensemble de votre sélection dans le dossier *lib* du lecteur *CIRCUITPY*.

Remarque : CircUp est un outil écrit en Python qui permet de vérifier la version de vos bibliothèques, de les mettre à jour et d'installer leurs dépendances. [13].

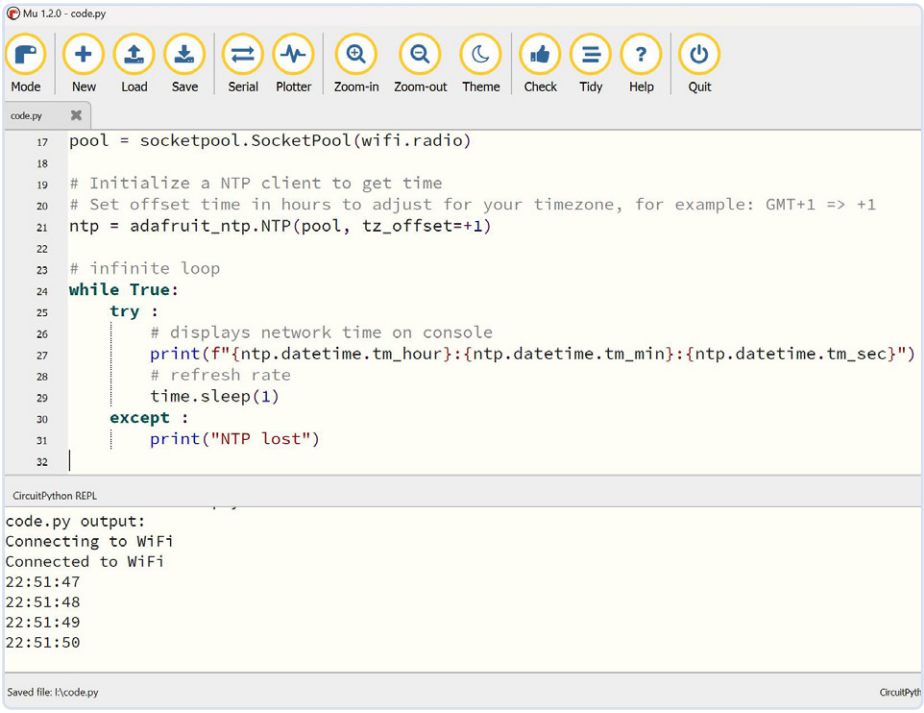

Figure 10. L'horloge NTP affichée dans la console série.

Affichage de l'heure dans la console

Dans le précédent programme, vous avez connecté votre Raspberry Pico W au réseau wifi. Vous allez maintenant interroger un serveur NTP dédié pour qu'il vous fournisse la date et l'heure. Le code permettant la récupération de l'heure et son affichage dans la console (**figure 10**) est disponible dans le **listage 2**.

Voyons ce qui a été ajouté :

- **Lignes 5 et 18** : importation de la bibliothèque native *socketpool* et création d'un socket qui permettra le maintien de la connexion client-serveur.
- **Ligne 22** : instanciation d'un client NTP via le socket. L'attribut `tz_offset` permet de régler le décalage horaire entre le fuseau horaire GMT (*Greenwich Mean Time*) et votre fuseau horaire. Étant en France, mon fuseau horaire CET correspond au fuseau horaire GMT +1, d'où la valeur +1. Vous devez l'ajuster à votre fuseau. Le serveur par défaut est celui d'Adafruit (mais on peut le modifier) et le *timeout* est de 10 secondes par défaut.
- **Lignes 24-32** : dans une boucle infinie, on affiche l'heure dans la console au rythme du rafraîchissement souhaité (ici 1 seconde – ligne 29). Pour cela, on utilise l'attribut `ntp.datetime`. Celui-ci renvoie un tuple nommé de la classe `time` qui contient les informations de date et d'heure du serveur. On accède à chaque élément du tuple par son champ nommé [15]. On utilise l'instruction `try…except` pour intercepter une exception qui correspondrait à la non réception d'une donnée une fois que la durée de *timeout* est écoulée.

Listage 2. Code permettant la récupération de l'heure.

```
# CircuitPython own's libraries
import time
import os
import wifi
import socketpool

# CircuitPython external libraries
import adafruit_ntp

print("Connecting to WiFi")

# connect to your WiFi network with SSID/PASSWORD in 'settings.toml'
wifi.radio.connect(os.getenv('CIRCUITPY_WIFI_SSID'), os.getenv('CIRCUITPY_WIFI_PASSWORD'))

print("Connected to WiFi")

# Setting up a socket
pool = socketpool.SocketPool(wifi.radio)

# Initialize a NTP client to get time
# Set offset time in hours to adjust for your timezone, for example: GMT+1 => +1
ntp = adafruit_ntp.NTP(pool, tz_offset=+1)

# infinite loop
while True:
    try :
        # displays network time on console
        print(f"::")
        # refresh rate
        time.sleep(1)
    except :
        print("NTP lost")
```

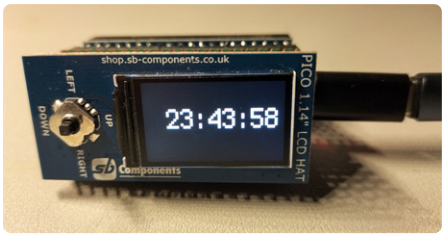

Figure 11. L'horloge NTP affichée sur l'écran LCD.

Affichage de l'heure sur l'ecran LCD

Pour cette étape, nous allons utiliser l'écran plutôt que la console pour l'affichage de l'heure. CircuitPython dispose d'une bibliothèque native générique appelée *displayio* qui permet de fournir des méthodes et des attributs communs pour tous les écrans supportés par CircuitPython.
Ici, l'écran embarqué sur le module LCD couleurs a les caractéristiques suivantes :

- Il utilise le protocole SPI
- Sa définition est de 240x135 pixels
- Son *driver* est le ST7789

Vous pouvez télécharger le code via le lien à la fin de l'article. Il serait long de le détailler ici, mais voici les principales étapes pour aboutir au résultat de la **figure 11** :

1. Bibliothèques natives à CircuitPython :
   board : pour accéder aux noms des broches du Raspberry Pico W
   displayio : pour la gestion des graphiques
   busio : pour le bus de communication SPI
   terminalio : pour disposer d'une police de caractères pour l'affichage
2. Bibliothèques externes à CircuitPython :
   adafruit_st7789 : pour la gestion de l'écran LCD
   adafruit_display_text : pour l'affichage de zone de texte sur l'écran

On configure ensuite tout l'affichage :

1. On crée le bus de communication SPI `SPI_bus`.
2. On crée le bus `display_bus` qui relie le Raspberry Pico W à l'écran LCD. Il contient le bus SPI mais également deux signaux supplémentaires (*Command* et *chip_select*).
3. On crée enfin l'écran `display` en lui fournissant le bus précédent et en précisant sa définition (`width=135`, `height=240`) et les offsets en x (`rowstart=40`) et y (`colstart=53`). Les valeurs de ces deux derniers paramètres s'expliquent par le fait que la définition maximale du driver ST7789 est 320x240 et que celle de notre écran n'est que de 240x135 (Cf. **figure 12**).
4. En CircuitPython, tout élément graphique doit appartenir à un groupe. On commence donc par créer un groupe `display_group` puis une zone de texte `time_label`. On ajoute la zone de texte au groupe. Et enfin on affiche le groupe sur l'écran.
5. Pour mettre à jour l'affichage de l'heure sur le LCD, il suffit, dans la boucle infinie, de modifier l'attribut `text` de notre zone de texte `time_label` en y copiant la chaîne de caractères que l'on affichait précédemment dans la console.

Listage 3. Code pour la RTC.

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

# Simple demo of reading and writing the time for the DS3231 real-time clock.
# Change the if False to if True below to set the time, otherwise it will just
# print the current date and time every second.  Notice also comments to adjust
# for working with hardware vs. software I2C.

import time
import board
import busio
import adafruit_ds3231

i2c = busio.I2C(scl=board.GP7, sda=board.GP6)
rtc = adafruit_ds3231.DS3231(i2c)

# Lookup table for names of days (nicer printing).
days = ("Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday")

# pylint: disable-msg=using-constant-test
if True:  # change to True if you want to set the time!
    #                     year, mon, date, hour, min, sec, wday, yday, isdst
    t = time.struct_time((2023, 03, 09, 14, 58, 15, 3, -1, -1))
    # you must set year, mon, date, hour, min, sec and weekday
    # yearday is not supported, isdst can be set but we don't do anything with it at this time
    print("Setting time to:", t)  # uncomment for debugging
    rtc.datetime = t
    print()
# pylint: enable-msg=using-constant-test

# Main loop:
while True:
    t = rtc.datetime
    # print(t)     # uncomment for debugging
    print(
        "The date is {} {}/{}/{}".format(
            days[int(t.tm_wday)], t.tm_mday, t.tm_mon, t.tm_year
        )
    )
    print("The time is {}:{:02}:{:02}".format(t.tm_hour, t.tm_min, t.tm_sec))
    time.sleep(1)  # wait a second
```

Test de l'horloge temps réel RTC

On pourrait s'arrêter là car notre horloge synchronisée sur le web fonctionne bien, mais que se passe-t-il si la connexion au réseau est interrompue ou si l'on coupe l'alimentation du montage pendant un certain temps ?

Pour éviter de perdre l'heure actuelle, une horloge temps réel va être ajoutée. Couplée à une batterie de sauvegarde (pile CR1220), elle maintiendra l'heure à jour pendant des années en absence d'alimentation via le port micro-USB. Cette RTC utilise le circuit populaire DS3231 qui emploie le bus I²C pour sa communication avec le module Raspberry Pico W.

Avant de combiner la RTC avec le serveur NTP, vous allez pouvoir la tester seule. C'est la bibliothèque *adafruit_ds3231* qui est nécessaire. Comme la majorité des bibliothèques CircuitPython, elle dispose d'exemples et de tutoriels en ligne [16].

Dans le code proposé dans ce tutoriel, vous devez juste modifier les premières lignes conformément au **listage 3**. En effet, de nombreuses cartes Raspberry Pico W ne supportent pas le protocole I²C [17]. Il faut donc préciser quelles sont celles qui sont physiquement reliées au composant RTC DS3231 (cf. **figure 3**). Lors de l'exécution, vous devriez obtenir dans la console un résultat similaire à celui de la **figure 13**.

Code final

La fonction première du projet est maintenant réalisée. Le code final est disponible en téléchargement sur [24]. la **figure 14** présente l'aspect général de l'affichage sur l'écran. On peut constater que l'aspect

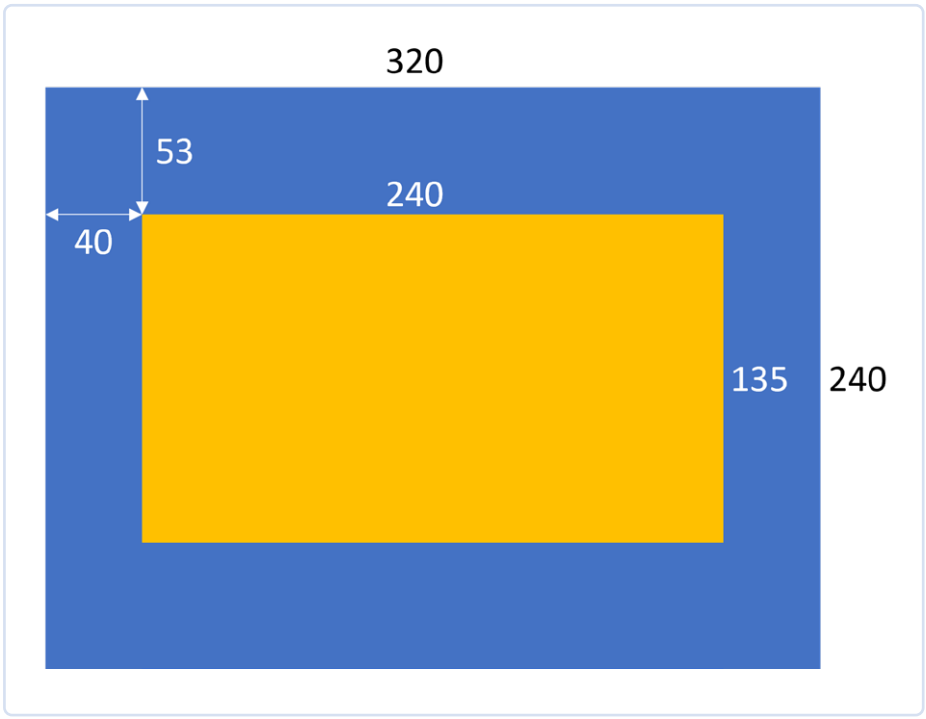

Figure 12. Offset de l'écran ST7789.

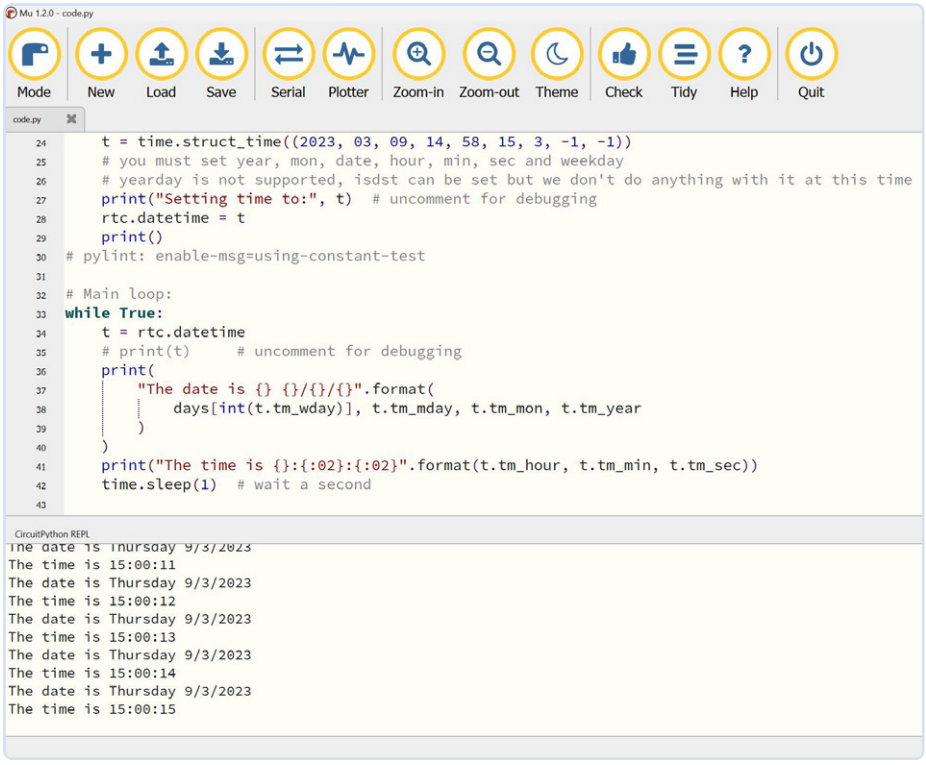

Figure 13. Résultats de l'horloge RTC.

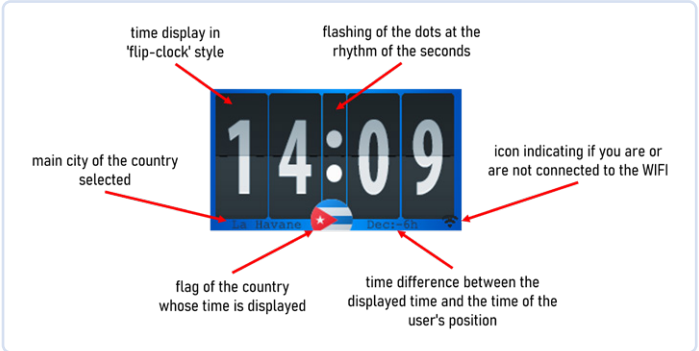

Figure 14. Interface finale.

Figure 15. Spritesheet des digits.

graphique a été amélioré et que des éléments nouveaux figurent sur l'écran. Encore une fois, une explication complète du code serait fastidieuse. Un tutoriel, disponible sur [18], fournit toutes les bases nécessaires.

Qu'est-ce qui change vis-à-vis des précédents codes ?

- Le code de gestion de l'horloge temps réel RTC a été mixé avec celui de la récupération de l'heure via le serveur NTP.
- L'affichage de l'heure se fait maintenant à l'aide d'images bitmap préparées dans un logiciel dédié. On utilise notamment la bibliothèque *adafruit_imageload* pour les charger en mémoire. Pour éviter de charger une image à chaque changement de digit (risque de ralentissement), on utilise ici une *spritesheet* comme en animation (**figure 15**). On définit ensuite quelle portion de l'image on désire afficher. On procède de même pour les points clignotants.
- Avec le joystick disponible sur le module *1.14" LCD hat* for Pico, on peut sélectionner un pays et voir ainsi l'heure actuelle dans ce pays et le décalage horaire engendré. Pour éviter les rebonds du joystick, on utilise ici une bibliothèque *adafruit_debouncer*. Cette dernière a besoin de la bibliothèque *adafruit_ticks* pour fonctionner.

Figure 16. Spritesheet des drapeaux.

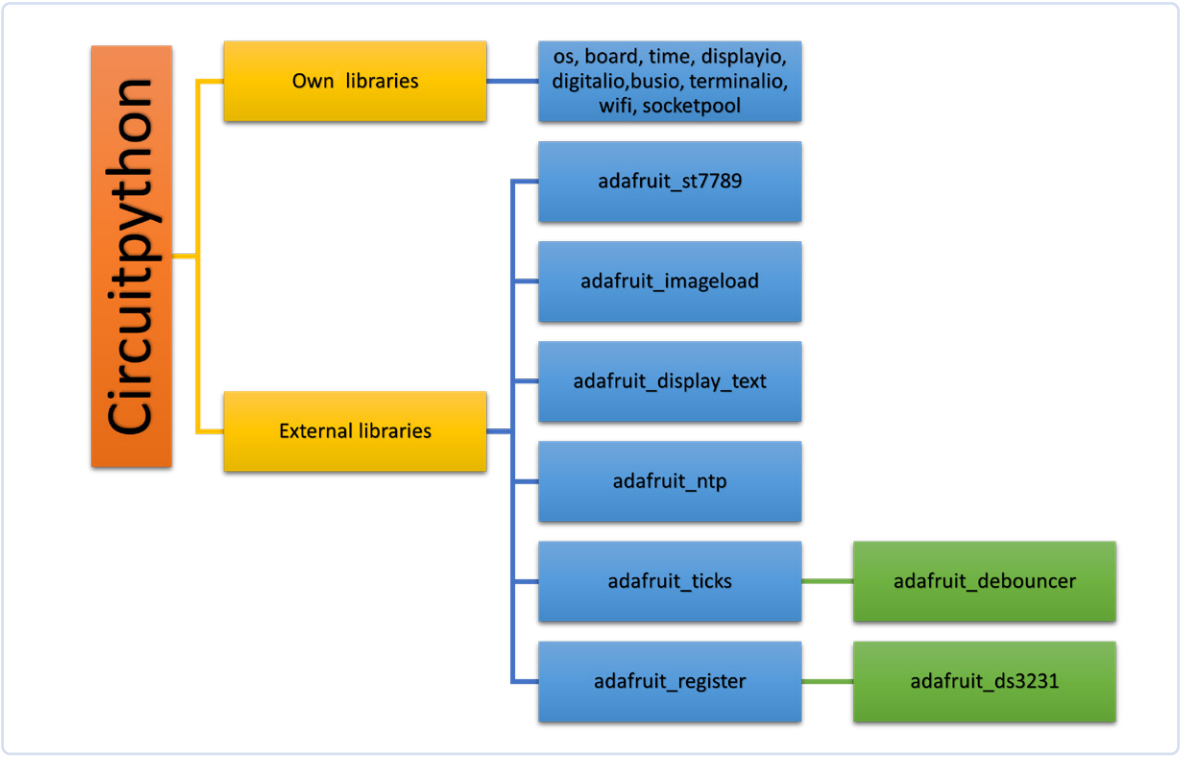

Figure 17. Arborescence des bibliothèques.

- L'affichage du drapeau utilise aussi un bitmap sous forme de *spritesheet* avec 25 pays disponibles (**figure 16**).
- Les textes correspondant à la ville et au décalage horaire utilisent des labels comme on l'a fait lorsque l'on a affiché l'heure sur le LCD précédemment.

L'arborescence des bibliothèques internes et externes utilisées dans le projet est représentée sur la **figure 17**.

Conclusion

On pourrait bien sûr imaginer des évolutions à ce projet comme utiliser un écran LCD avec une définition plus importante (en se limitant toutefois à 320x240 vis-à-vis de la puissance du Raspberry Pico W) ou ajouter des alarmes sonores utilisateur ou encore afficher les données météos actuelles pour notre position… Il est assez aisé de programmer ces améliorations en CircuitPython à l'aide des documentations des bibliothèques en ligne, de très nombreux tutoriels existants sur le site d'Adafruit…

N'hésitez pas à vous lancer dans un projet personnel en utilisant ce langage CircuitPython, vous découvrirez par vous-même sa simplicité d'utilisation.

- Téléchargement : CircuitPython pour le Raspberry Pi Pico W [18]
- Téléchargement : bibliothèques externes [19]
- Documentation : bibliothèques natives de CircuitPython [20]
- Documentation : bibliothèques externes [21]
- Tutoriels de démarrage en CircuitPython [22], [23]

Il y a également un livre publié chez Elektor que j'ai écrit en 2020 avec la version 5.3 de CircuitPython (**figure 18**). Certes, il y a eu beaucoup de nouveautés depuis cette édition, mais les bases du langage demeurent inchangées et le code se transpose facilement à d'autres modules comme la Raspberry Pico (W) (c'est toute la force de CircuitPython avec son *unified hardware API*). Et si vous êtes bloqué, vous pouvez toujours m'envoyer vos questions à mon adresse mail.

220633-04

Figure 18. Livre « Initiation au language circuitpython et à la puce nRF52840 «.

Des questions, des commentaires ?

Si vous avez des questions ou des propositions/suggestions d'articles, envoyez un courriel à l'auteur (michael.bottin@univ-rennes.fr), ou contactez Elektor (redaction@elektor.fr).

Produits

- **Raspberry Pi Pico RP2040 W**
  https://elektor.fr/20224
- **Michael Bottin, *Initiation au langage CircuitPython et à la puce nRF52840* (livre en français)**
  https://elektor.fr/19523

LIENS

[1] MicroPython : https://fr.wikipedia.org/wiki/MicroPython
[2] CircuitPython : https://en.wikipedia.org/wiki/CircuitPython
[3] Raspberry Pico W chez Elektor : https://elektor.fr/raspberry-pi-pico-rp2040-w
[4] Module d'affichage avec un écran LCD pour le Pico : https://shop.sb-components.co.uk/products/1-14-lcd-hat-for-pico
[5] Module avec une horloge temps réel pour le Pico : https://shop.sb-components.co.uk/products/pico-rtc-hat
[6] Site web de CircuitPython : https://circuitpython.org/
[7] Mu Editor : https://codewith.mu/en/download
[8] Interface de Mu Editor : https://codewith.mu/en/tutorials/1.2/start
[9] Console REPL : https://codewith.mu/en/tutorials/1.2/repl
[10] Graphe déroulant : https://codewith.mu/en/tutorials/1.2/plotter
[11] Raccourcis clavier : https://codewith.mu/en/tutorials/1.2/shortcuts
[12] Fichier TOML : https://fr.wikipedia.org/wiki/TOML
[13] L'outil Circup : https://learn.adafruit.com/keep-your-circuitpython-libraries-on-devices-up-to-date-with-circup/
[14] UTC (Coordinated Universal Time): https://timeanddate.com/worldclock/timezone/utc
[15] Classe time : https://docs.python.org/3/library/time.html#time.struct_time
[16] Tutoriels et exmples avec la bibliothèque adafruit_ds3231 :
https://learn.adafruit.com/adafruit-ds3231-precision-rtc-breakout/circuitpython
[17] Brochage du Pico W : https://datasheets.raspberrypi.com/picow/PicoW-A4-Pinout.pdf
[18] Version de CircuitPython pour le Raspberry Pico W : https://circuitpython.org/board/raspberry_pi_pico_w/
[19] Bibliothèques externes de CircuitPython : https://circuitpython.org/libraries
[20] Bibliothèques natives de CircuitPython : https://docs.circuitpython.org/en/latest/shared-bindings/index.html#modules
[21] Tutoriels de démarrage en CircuitPython : https://docs.circuitpython.org/projects/bundle/en/latest/drivers.html
[22] Tutoriels de démarrage en CircuitPython 1 : https://learn.adafruit.com/welcome-to-circuitpython/circuitpython-essentials
[23] Tutoriels de démarrage en CircuitPython 2 : https://learn.adafruit.com/welcome-to-circuitpython/
[24] Téléchargement du logiciel : https://elektormagazine.fr/220633-04