Een NTP-klok met CircuitPython

waarom zou je deze programmeertaal gebruiken?

Michael Bottin (Frankrijk)

We nemen een duik in een van de nuttige alternatieven voor Python die licht genoeg zijn om op microcontrollers te draaien: CircuitPython. Hier bouwen we een project rond een Raspberry Pi Pico W, dat de tijd ophaalt van een NTP-server en deze weergeeft op een LCD.

Python is een van de populairste programmeertalen. Omdat het een geïnterpreteerde taal is, maakt het de ontwikkelingsfase veel sneller, in tegenstelling tot talen als C/C++, waar je je programma moet compileren voordat je het kunt uitvoeren. Aan de andere kant betekent dit niet-compileren dat vooraf een uitvoeringsomgeving op het target moet worden geïnstalleerd.
Al meer dan 30 jaar is Python in de voorste gelederen bij computers en cloud computing. Het bestrijkt verschillende gebieden zoals wetenschappelijke berekeningen, webontwikkeling (back-end), systeemscripting, machine learning en big data (gegevensanalyse en visualisatie). Python wordt door veel bedrijven gebruikt, onder andere door Google ("Python waar het kan, C++ waar het moet"), YouTube, Instagram, Spotify, Intel, Facebook, Dropbox, Netflix, Pixar en Reddit.
Het is dan ook geen verrassing dat Python zijn weg heeft gevonden naar de wereld van de embedded elektronica. Maar om dat mogelijk te maken, moest er een lichtgewicht versie komen die op een microcontroller kon draaien. Maar die versie moest ook hardware-georiënteerd zijn om de verschillende periferie te gebruiken die bij een microcontroller beschikbaar is (GPIO, PWM, I²C, SPI, UART...).
Er zijn twee versies ontwikkeld:

- MicroPython, door Damien George in 2013 [1] gemaakt;
- CircuitPython, gemaakt door Limor Fried (Adafruit CEO), Scott Shawcroft en vele anderen [2].

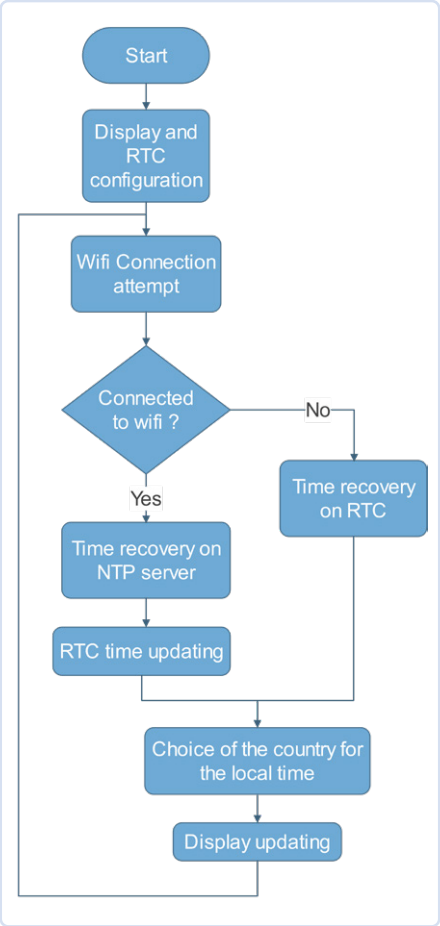
Deze twee talen delen dezelfde implementatie van de programmeertaal Python (CPython). CircuitPython is een open-source fork van MicroPython. Elke evolutie van MicroPython heeft directe gevolgen voor CircuitPython. Voor zover ik weet zijn er in dit tijdschrift al verschillende artikelen verschenen over het gebruik van MicroPython, maar geen over het gebruik van CircuitPython.

Voordelen van CircuitPython boven MicroPython?

Laten we eerlijk zijn, als je als IT-specialist al bekend bent met Python en je wilt deze taal gebruiken in de wereld van de embedded elektronica, dan is CircuitPython niet interessant voor je. In dat geval is het eigenlijk veel beter om MicroPython te gebruiken. Ben je echter een maker, student of docent zonder veel kennis van Python, dan kan CircuitPython een interessante en geschikte keuze zijn. CircuitPython biedt namelijk niet alleen een gebruiksvriendelijk ontwikkeltraject, maar ook een extra abstractielaag. Opmerkelijk is ook dat de meeste informatie over deze taal online te vinden is via bibliotheken, tutorials en forums.
Daarom heb ik voor dit artikel CircuitPython gekozen. Ik wilde een eenvoudige aanpak bieden ten behoeve van lezers die Python op microcontrollers willen gaan gebruiken. De CircuitPython-community is bovendien erg actief. Er worden regelmatig nieuwe versies uitgebracht (de huidige versie is 8.x) en nieuwe bibliotheken, en veel op microcontrollers gebaseerde boards ondersteunen deze taal. CircuitPython kan ook worden uitgevoerd op SBC's zoals Raspberry Pi, BeagleBone, Odroid of Jetson met behulp van Blinka API.

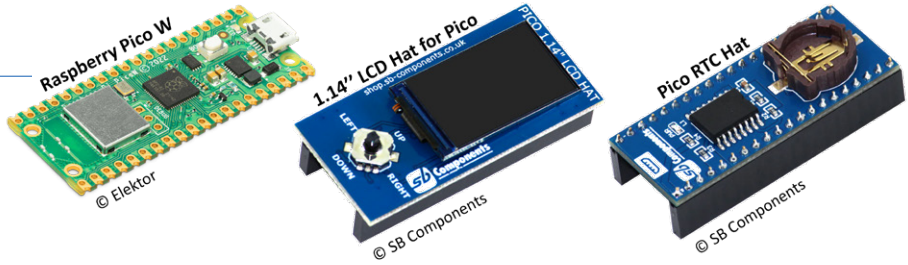
Doel van dit artikel

Dit artikel is bedoeld om de lezers een kans te geven CircuitPython te ontdekken en vertrouwd te raken met deze taal door middel van een eenvoudige toepassing.
Ik zal je laten zien hoe je met CircuitPython heel eenvoudig een internet-gesynchroniseerde NTP-klok kunt maken. Met dit project kun je ook de lokale tijd kiezen voor meer dan twintig landen over de hele wereld. De klok heeft een WiFi-verbinding nodig (SSID en wachtwoord vooringesteld)



Figuur 1. Algemeen flowdiagram van de NTP-klok.

om de tijd te synchroniseren, en een speciale NTP-server (Network Time Protocol) om de timestamp op te halen.
Deze klok kan de tijd ook bijhouden als er geen netwerk beschikbaar is, dankzij een ingebouwde real-time klok (RTC), die de tijd opslaat als het netwerk beschikbaar is of zelf de tijd aangeeft als er geen netwerkverbinding is. Een batterij zorgt ervoor dat de RTC up-to-date blijft wanneer de voeding uitvalt. Het algemene flowdiagram van de klok is getekend in **figuur 1**.



Figuur 2. De gebruikte modules.

Presentatie van het materiaal

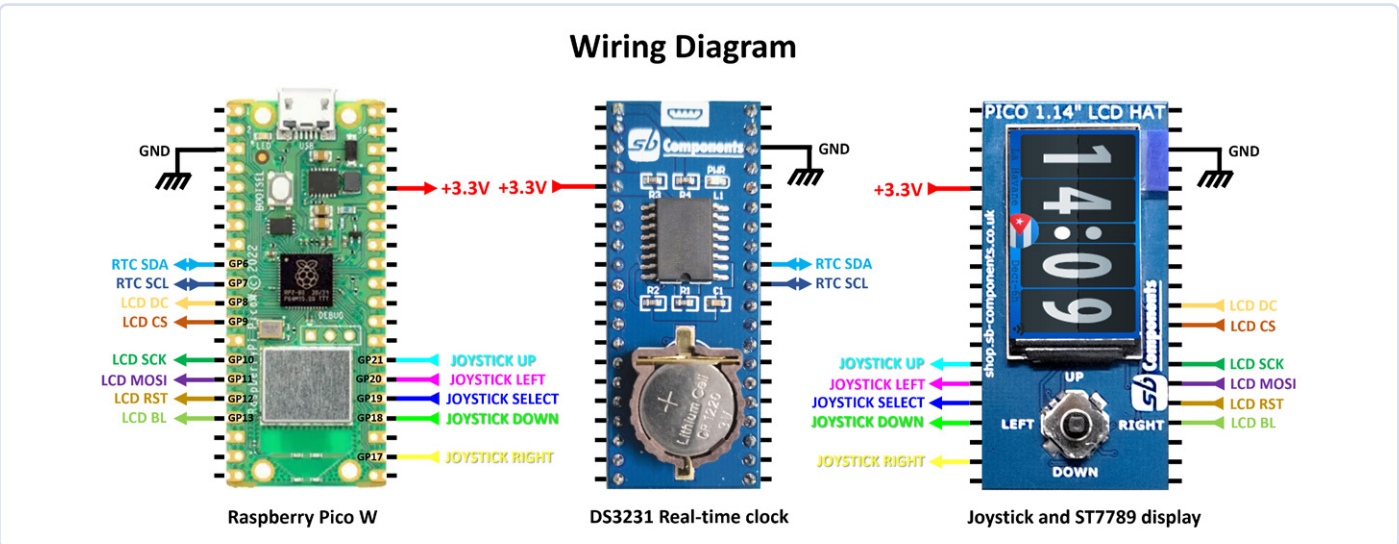
Veel ontwikkelboards (en dus ook microcontrollers) zijn gemakkelijk te programmeren met CircuitPython. Ik heb gekozen voor de Raspberry Pico W omdat het een bekend, goedkoop board is dat in de meeste shops, waaronder die van Elektor [3], verkrijgbaar is en omdat het een WiFi-interface bezit. Daar staat tegenover dat het geen display of een nauwkeurige real-time klok heeft, wat betekent dat extra componenten nodig zijn voor ons project.
In plaats van een eigen board te ontwikkelen, besloot ik voor dit project kant-en-klaar verkrijgbare modules te gebruiken. Drie op elkaar geprikte modules volstaan voor het project (zie **figuur 2**):

- een Raspberry Pico W-module (gebruik in elk geval de W-versie, die de WiFi-interface heeft);
- een displaymodule met een LCD met een resolutie van 240 × 135 en een joystick-controller voor de navigatie (SB Components [4]);
- een DS3231 real-time klokmodule (SB Components [5]).

Opmerking: de RP2040-microcontroller heeft ook een interne real-time klok die in plaats van de Pico RTC Hat-module kan worden gebruikt. Maar, zoals gezegd, de RP2040 is niet zo nauwkeurig en heeft geen back-up batterij.

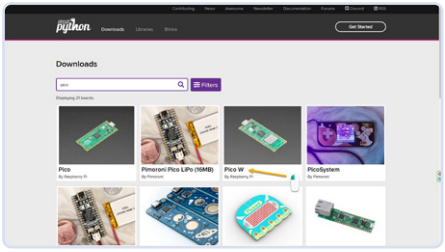
Bedradingsschema

Zelfs als we kant-en-klare modules gebruiken die de hardware-implementatie vergemakkelijken, moeten we de verbindingen tussen de modules kennen om de programmaregels van onze toepassing te kunnen schrijven. **Figuur 3** toont de diverse verbindingen tussen de verschillende modules, evenals de namen die aan deze verbindingen zijn gegeven en die we in het programma opnieuw zullen gebruiken. (Merk op dat de pinout van SB Components die in hun GitHub-documentatie wordt vermeld, op het moment van schrijven onjuist is, maar hun broncode verwijst naar de correcte pinnen in figuur 3).

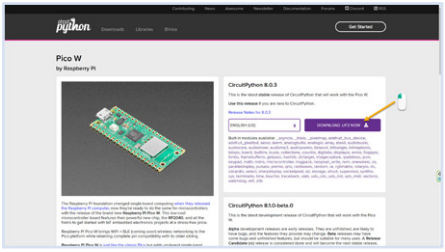


Figuur 3. De verbindingen tussen de modules.

Figuur 4. De CircuitPython-webpagina.



Figuur 5. Selecteer de Pico W uit deCircuitPython-compatibele boards.



Figuur 6. Download het .UF2-bestand.

Installatie van CircuitPython

De RP2040-microcontroller van de Raspberry PI Pico W is niet voorzien van CircuitPython wanneer je de module koopt, dus moet je hem eerst op het board 'flashen'. Net zoals er verschillende C-compilerversies zijn afhankelijk van het target dat je gebruikt, zijn er evenveel CircuitPython-versies als boards die het ondersteunen. Op dit moment zijn er meer dan 380 CircuitPython-compatibele commerciële boards. Veel van de meest recente boards van Adafruit zijn uiteraard compatibel, maar veel andere zijn dat ook, zoals de boards van Pimoroni, Expressif, Seeedstudio, Waveshare, LilyGo, Cytron, DFRobot en Wiznet.

De meeste gebruiken microcontrollers van Microchip (SAMD21/SAMD51), Nordic (nRF52840), Expressif (ESP32) of van de Raspberry Pi Foundation (RP2040). Je moet de passende Raspberry Pico W CircuitPython-versie downloaden:

- ga in je browser naar de website van CircuitPython [6] (**figuur 4**);
- klik op de link *Downloads* en zoek naar het Pico W-board (**figuur 5**);
- download het UF2-bestand van de getoonde pagina (**figuur 6**).

De RP2040-microcontroller van de Raspberry Pi Pico bevat al een bootloader die de installatie van CircuitPython vergemakkelijkt. De onderstaande instructies tonen hoe je de gedownloade CircuitPython-versie kunt installeren.

1. Ontkoppel het Raspberry Pico W-board van je computer.
2. Houd de BOOTSEL-knop van het Raspberry Pico W-board ingedrukt terwijl je het board weer op je computer aansluit via een micro-USB-kabel die geschikt is voor gegevensoverdracht.
3. Er moet nu in de file explorer een nieuw station met de naam RPI-RP2 verschijnen.
4. Laad het gedownloade CircuitPython UF2-bestand in RPI-RP2.

Dat is alles. Zodra CircuitPython met succes op het Raspberry Pico W bord is geflasht, moet er een *CIRCUITPY*-drive verschijnen.

Het gebruik van de CIRCUITPY-drive

Hier volgen enkele belangrijke zaken waar je op moet letten bij het gebruik van het *CIRCUITPY*-station:

- Het station gedraagt zich als een USB-station. Je kunt bestanden/mappen toevoegen of verwijderen vanuit een file explorer. De capaciteit is die van het flash-geheugen dat beschikbaar is op de RP2040-microcontroller om je code en resources (afbeeldingen, audio…) op te slaan.
- Je kunt het Raspberry Pico W-board op uw computer aansluiten zonder bijzondere voorzorgsmaatregelen te nemen. **Aan de andere kant verdient het sterke aanbeveling om het uit te werpen als een USB-station wanneer je het wilt ontkoppelen om geen risico te lopen dat het bestandssysteem beschadigd raakt.**
- Nadat je het bestand *boot.txt* hebt geopend, kun je controleren welke versie van CircuitPython op het Raspberry Pico W-board is geïnstalleerd.
- Zodra het Raspberry Pico W-board via de USB-poort wordt gevoed (door je computer of door een DC-voeding), draait de CircuitPython-code. Maar alleen bestanden met de naam *code.py* of *main.py* worden uitgevoerd; zorg ervoor dat je je bestand voor dit project de naam *code.py* geeft.
- Het is raadzaam (maar niet verplicht) om de afbeeldingen te kopiëren naar een directory *./images*, de bibliotheekbestanden naar een directory *./lib* enzovoort. Dit helpt om je projecten in een overzichtelijke boomstructuur te organiseren.

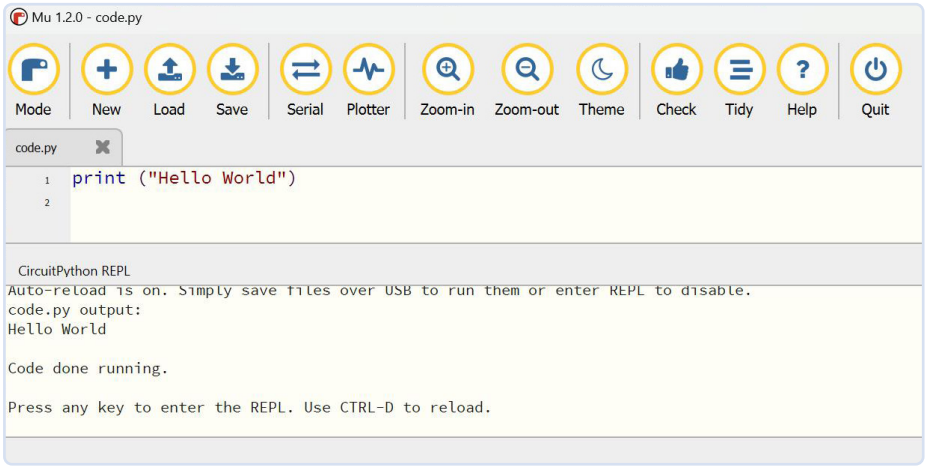
Gebruik van de IDE

CircuitPython is een geïnterpreteerde taal, dus er hoeft niet gecompileerd te worden. Als er een *code.py*-bestand met CircuitPython-code aanwezig is op de *CIRCUITPY*-drive, dan wordt het automatisch uitgevoerd. Dat betekent dat we dat bestand gewoon in een tekstverwerker kunnen bewerken en opslaan voordat we het uitvoeren, maar vergeet niet: programma's schrijven gaat vaak hand in hand met debuggen. Het gebruik van een IDE geeft je toegang tot hulpmiddelen zoals een console om feedback te krijgen over fouten.

De IDE die we voor dit project zullen gebruiken is Mu Editor [7]. Het is een eenvoudige gratis softwareoplossing die beschikbaar is voor Windows, Mac OSX en Linux. (Je kunt ook Thonny, VS Code, Atom of PyCharm gebruiken als je de CircuitPython-extensie installeert).

Figuur 7 toont de interface van de Mu Editor IDE. De werkbalk is minimalistisch:

- *Mode:* keuze van de programmeertaal. Zorg ervoor dat je de CircuitPython-modus selecteert.
- *New:* aanmaken van een nieuw, leeg bestand.
- *Load:* opent een bestaand bestand. Als de CircuitPython-modus is geselecteerd en je Raspberry Pico W-board correct is aangesloten, zou de *CIRCUITPY*-drive automatisch in het dialoogvenster moeten verschijnen.
- *Save:* slaat het huidige bestand op (het actieve tabblad). Een kleine rode stip naast de bestandsnaam in het tabblad geeft aan dat het bestand is gewijzigd sinds de laatste back-up.
- *Serial:* toont/verbergt de console. Ik raad je aan de console (REPL) altijd weer te geven, omdat deze foutmeldingen toont en wat je wilt afdrukken van de code die op de Pico draait.
- *Plotter:* toont/verbergt een scrollende grafiek met digitale gegevens die je code kan aanmaken.
- *Zoom-in* en *Zoom-out:* maakt het lettertype groter of kleiner.



Figuur 7. De interface van de Mu Editor IDE.

- *Theme:* schakelt tussen drie verschillende displaymodi: *Day* (licht thema), *Night* (donker thema) en *High-Contrast*.
- *Check:* zoekt naar fouten in de code, zelfs als deze geen invloed hebben op de werking ervan.
- *Tidy:* ruimt je code op door bijvoorbeeld nutteloze spaties of lege regels te verwijderen.
- *Help:* Online-helpfunctie.
- *Stop:* Sluit de editor.

Deze stappen moeten dus worden gevolgd om een programma te ontwikkelen in de IDE:

- Open het *code.py*-bestand van het *CIRCUITPY*-station.
- Wijzig de code in het editorvenster.
- Sla de wijzigingen in het bestand op.
- Bekijk het resultaat van de uitvoering in de seriële console. Ga terug naar stap 2 als er tijdens de uitvoering een fout is opgetreden.

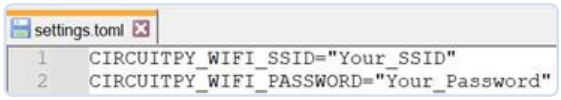
Meer informatie vind je via deze links:

- Begin hier! (over de interface) [8].
- Wat is een REPL? (de console) [9].
- Data plotten met Mu [10].
- Sneltoetsen [11].

Hiermee heb je alles om je project te starten. Dit is waar het nu om gaat: de NTP-klok.

WiFi-verbindingstest

CircuitPython ondersteunt native WiFi op het Raspberry Pico W-board, dus je hoeft geen extra bibliotheken te installeren. Om verbinding te maken met WiFi moet je de SSID en het wachtwoord van je netwerk opgeven. Om te voorkomen dat deze informatie direct in de code moet worden opgenomen, maakt CircuitPython gebruik van omgevingsvariabelen.
In de root van *CIRCUITPY* staat een bestand *settings.toml* [12] en deze gegevens blijven 'geheim'. Alle vertrouwelijke gegevens, zoals API-keys, kunnen in dit bestand worden opgeslagen; in ons geval bevat het alleen de SSID en het wachtwoord van het netwerk waarmee je je Raspberry Pico W-board wilt verbinden (zie **figuur 8**). (Opmerking: dit bestand kan niet bewerkt worden in de IDE; je moet een tekstverwerker gebruiken om het te maken en in te vullen.)
De code om verbinding te maken met het netwerk is gegeven in **listing 1**. Je ziet meteen al dat de code erg compact is:



Figuur 8. SSID en wachtwoord in settings.toml.

Listing 1. Code voor de WiFi-verbinding.

```
# CircuitPython's own libraries
import os
import ipaddress
import wifi

print()
print("Connecting to Wi-Fi")

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

print("Connected to Wi-Fi")

# pings Google DNS server
ipv4 = ipaddress.ip_address("8.8.8.8")
print("Ping google.com: %f ms" % (wifi.radio.ping(ipv4)*1000))
```

Figuur 9. Downloaden van de bibliotheken (de datum in de bestandsnaam zal natuurlijk anders zijn).

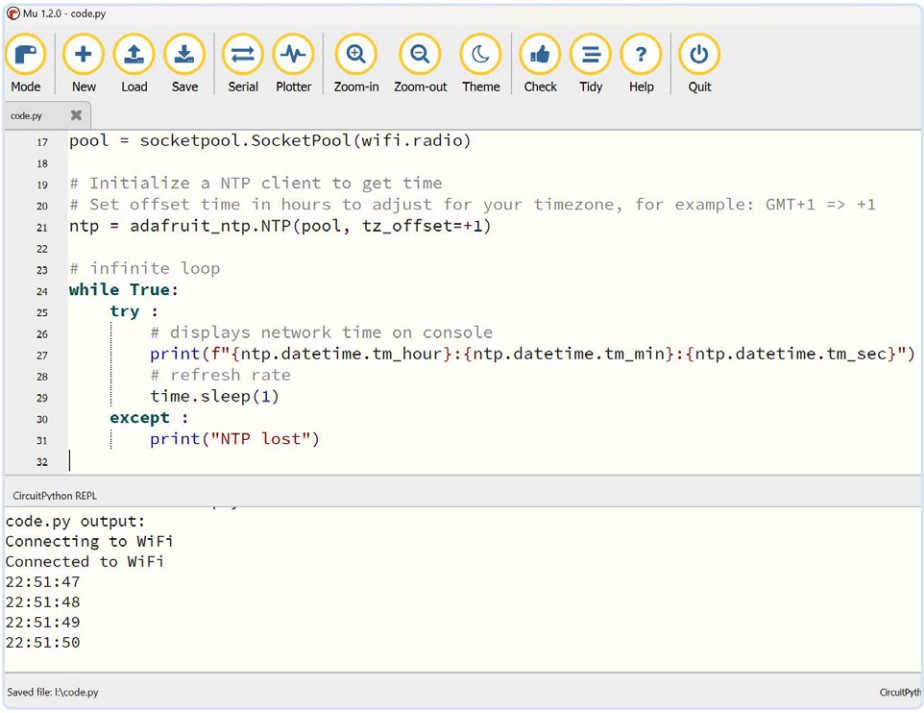
- **Regel 2:** importeren van de *os*-bibliotheek, waarmee de omgevingsvariabelen van het bestand *settings.toml* kunnen worden gelezen.
- **Regel 4:** importeren van de *wifi*-bibliotheek, die de netwerkverbinding mogelijk maakt.
- **Regels 6-7, 12:** deze regels zijn niet noodzakelijk, maar geven de gebruiker informatie via de seriële console.
- **Regel 10:** deze ene regel is voldoende om de fysieke netwerkverbinding tot stand te brengen.
- **Regels 3, 14-16:** deze regels zijn niet noodzakelijk, maar ze stellen je in staat een test (ping) uit te voeren door een server te ondervragen en zo de status van je WiFi-verbinding te controleren, en vervolgens het resultaat uit te voeren naar de seriële console.

Nuttige bibliotheken toevoegen

De meeste basisbibliotheken zijn te vinden op CircuitPython (en niet alle beschikbare bibliotheken, anders zou het flash-geheugen van de microcontroller overbelast worden). Het is dus noodzakelijk, afhankelijk van de projecten waar je aan werkt, om enkele bibliotheken toe te voegen.
Het installeren van extra bibliotheken in CircuitPython bestaat gewoon uit het kopiëren van de bijbehorende bestanden naar de *CIRCUITPY*-drive. Installeer alle bibliotheken die nuttig kunnen zijn voor de uiteindelijke versie van het project, zodat je het maar één keer hoeft te doen.
Zo kun je ze installeren:

1. Maak een map *./lib* aan op je *CIRCUITPY*-drive (als die nog niet bestaat).
2. Alle beschikbare bibliotheken kunnen als één enkel archief worden gedownload van de CircuitPython-website (opmerking: ze zijn ook afzonderlijk beschikbaar op GitHub).
3. Klik op de website op de link *Libraries*.
4. Download de bundel die overeenkomt met jouwCircuitPython-versie: in ons geval versie 8.x (zie **figuur 9**).
5. Pak het uit naar een wilekeurige locatie op je computer.
6. Open de map *lib* in de uitgepakte bundel.
7. Selecteer:
   a. deze mappen:
      i. *adafruit_register*
      ii. *adafruit_imageload*
      iii. *adafruit_display_text*
   b. deze bestanden:
      i. *adafruit_debouncer.mpy*
      ii. *adafruit_ds3231.mpy*
      iii. *adafruit_ntp.mpy*
      iv. *adafruit_st7789.mpy*
      v. *adafruit_ticks.mpy*
8. Kopieer de selectie naar de map *./lib* op de *CIRCUITPY*-drive.

Opmerking: Circup is een tool (geschreven in Python) dat de versie van je bibliotheken kan controleren, ze kan bijwerken en hun dependencies kan installeren [13].



Figuur 10. NTP-klok weergegeven in de seriële console.

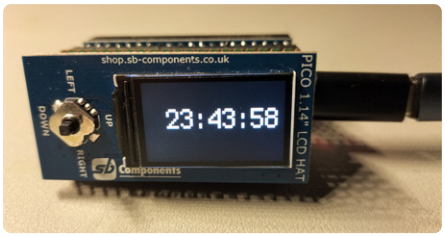
Tijdsweergave in de console

In de bovenstaande code heb je je Raspberry Pico W-board verbonden met het WiFi-netwerk. Nu ga je een speciale NTP-server bevragen zodat deze je de tijd en datum levert. De code waarmee je de tijd kunt ophalen en weergeven in de console (**figuur 10**) is gegeven in **listing 2**.
Laten we eens kijken naar de regels die zijn toegevoegd:

- **Regels 5 en 18:** importeren van de *socketpool* native library en aanmaken van een socket die de client/server-verbinding verzorgt.
- **Regel 22:** instantie van een NTP-cliënt via de socket. De parameter `tz_offset` wordt gebruikt om het tijdsverschil tussen UTC (Coordinated Universal Time [14]) en je eigen tijdzone in te stellen. In Frankrijk gebruiken we CET, wat eigenlijk UTC+1 is, vandaar de waarde `+1`. Je moet deze aanpassen voor je eigen tijdzone. De standaard server is de Adafruit-server (maar dat kan worden aangepast) en de standaard time-out is 10 seconden.
- **Regels 24-32:** in een oneindige lus wordt de tijd in de console weergegeven met de gewenste verversingssnelheid (in ons geval 1 seconde – regel 29). Hiertoe gebruiken we de parameter `ntp.datetime`. Deze parameter retourneert een tupel met naam van de tijdsklasse die de datum- en tijdinformatie van de server bevat. We benaderen elk element van de tupel via zijn naamveld [15]. We gebruiken de `try...except` constructie om een uitzondering af te vangen die betekent dat de gegevens niet zijn ontvangen zodra de time-out is bereikt.

Listing 2. NTP-request code.

```
# 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")
```



Figuur 11. NTP-klok weergegeven op het LCD.

Tijdsweergave op LCD-scherm

In deze stap gaan we het display gebruiken in plaats van de console om de tijd weer te geven. CircuitPython beschikt over een native bibliotheek met de naam *displayio*. Deze bibliotheek biedt methoden en gemeenschappelijke attributen voor elk display dat door CircuitPython wordt ondersteund.
De kleuren-LCD-module die we hier gebruiken heeft de volgende features:

- SPI-protocol
- resolutie 240 × 35 pixels
- ST7789-controller

Je kunt de code downloaden via de link aan het eind van dit artikel. Het zou lang duren om alles in detail uit te leggen, maar dit zijn de belangrijkste stappen om een resultaat te bereiken zoals getoond in **figuur 11**. Je moet verschillende bibliotheken importeren:

1. Native bibliotheken van CircuitPython:
   board: voor toegang tot de namen van de Raspberry Pico W-pinnen
   displayio: voor het beheren van de graphics
   busio: voor de SPI-bus communicatie
   terminalio: om een lettertype voor het scherm te krijgen
2. Externe bibliotheken:
   adafruit_st7789: om het LC-display aan te sturen
   adafruit_display_text: om het tekstgebied op het display weer te geven

Vervolgens moet je de display-instellingen configureren:

1. Maak de SPI-communicatiebus `SPI_bus`.
2. Maak de bus `display_bus` om het Raspberry Pi Pico W-board te verbinden met het LCD. Hij bevat de SPI-bus plus twee extra signalen (*Command* en *chip_select*).
3. Maak het scherm `display` door de vorige bus beschikbaar te maken, door de resolutie ervan te definiëren (`width=135, height=240`) en door de offsets in x (`rowstart=40`) en y (`colstart=53`) aan te passen. De waarden van deze laatste twee parameters komen voort uit het feit dat de hoogste resolutie van de ST7789-driver 320 x 240 bedraagt, terwijl de resolutie van ons scherm slechts 240 x 135 bedraagt (zie **figuur 12**).
4. In CircuitPython moet elk grafisch element tot een groep behoren. Maak een groep met de naam `display_group` en maak een `time_label` tekstveld. Voeg het tekstveld toe aan de groep. Geef tenslotte de groep weer op het scherm.
5. Om de tijd op het scherm bij te werken, hoef je alleen maar in de oneindige lus het `text` attribuut van ons tekstveld-object `time_label` te wijzigen door de tekenreeks te kopiëren die we gebruikten voor weergave in de console.

Listing 3. RTC-code.

```
# 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 van de real-time klok

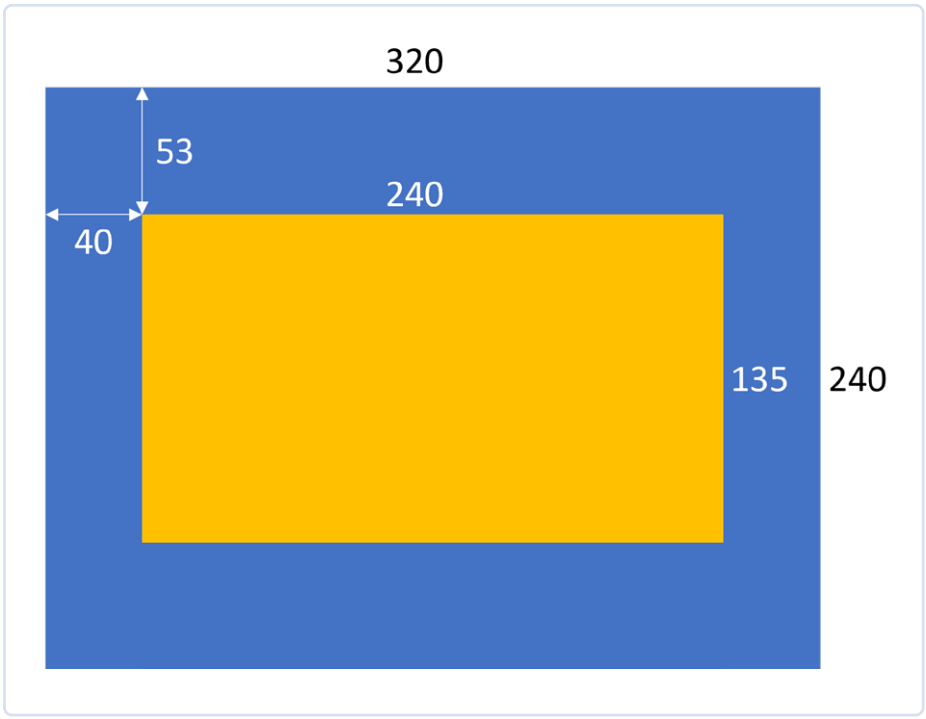
Onze webgesynchroniseerde klok werkt perfect – houden we het dus voor gezien? Nee, we zijn nog niet klaar, want wat zou er gebeuren als de netwerkverbinding werd onderbroken? Of als de stroom tijdelijk uitvalt?

Om te voorkomen dat de huidige tijd verloren gaat, voegen we een real-time klok toe. Gekoppeld aan een back-up batterij (CR1220) zal deze de tijd jarenlang bijhouden, zelfs als de klok niet gevoed wordt via de micro-USB-poort. Deze RTC gebruikt de bekende DS3231, en die gebruikt de I²C-bus om te communiceren met de Raspberry Pi Pico W. Alvorens de RTC te combineren met de NTP-server gaan we die separaat testen, en daarvoor heb je de *adafruit_ds3231* bibliotheek nodig. Zoals de meeste CircuitPython-bibliotheken biedt deze online tutorials en voorbeelden [16].

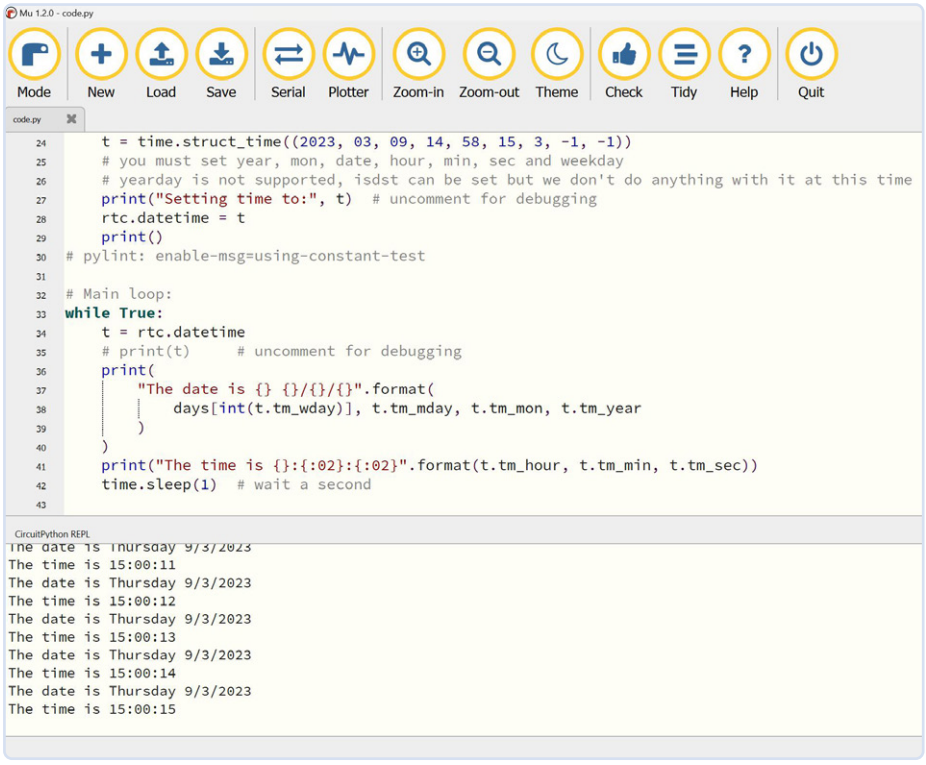
In de code die in deze tutorial wordt gepresenteerd, hoef je alleen de eerste regels aan te passen zoals getoond in **listing 3**. Veel Raspberry Pi Pico W's ondersteunen het I²C-protocol namelijk niet [17]. Je moet precies weten welke fysiek verbonden zijn met het DS3231 RTC-IC (zie figuur 3). Bij de uitvoering zou je iets vergelijkbaars moeten krijgen als in **figuur 13**.

Definitieve code

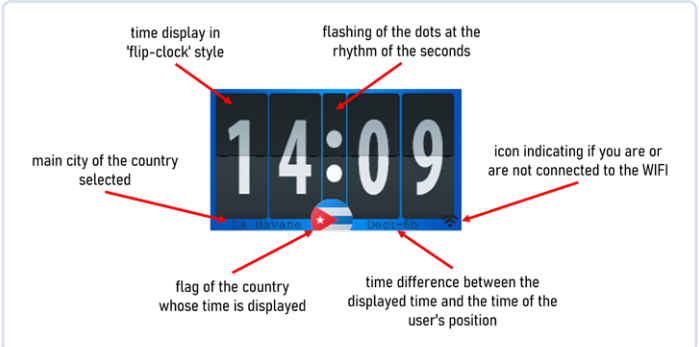
Het primaire doel van ons project is bereikt. De link om de definitieve code te downloaden is te vinden in [24]. **Figuur 14** toont de algemene interface van het display. Je ziet dat het grafische aspect is verbeterd en dat er nieuwe elementen op het scherm zijn verschenen. Nogmaals, een gedetailleerde uitleg zou langdradig worden, maar dit artikel verduidelijkt alle noodzakelijke basisprincipes [18].



Figuur 12. ST7789 display-offsets.



Figuur 13. RTC-resultaat.



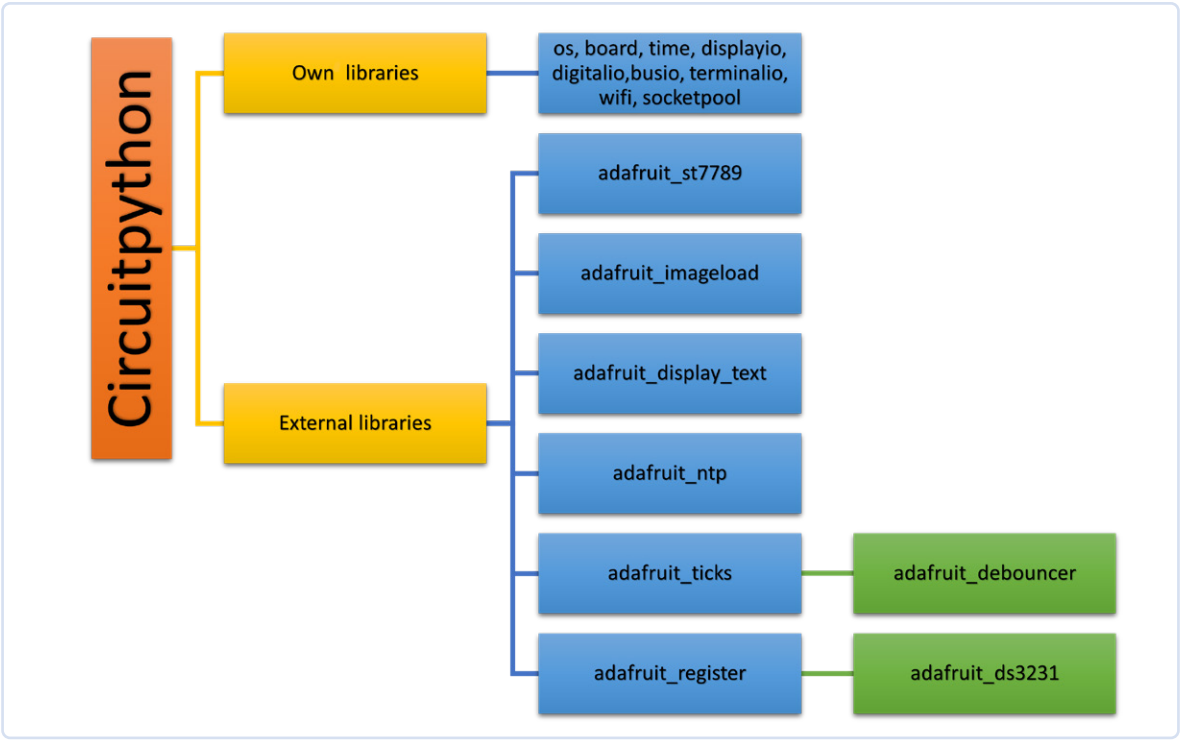
Figuur 14. De definitieve interface.

Figuur 15. Het cijfer-sprite sheet.

Wat zijn de verschillen met de vorige listings?

- De code om de real-time klok (RTC) te beheren is gecombineerd met de code die het mogelijk maakt de tijd op te halen van de NTP-server.
- De tijd wordt nu weergegeven met behulp van bitmap-afbeeldingen die eerder in speciale code zijn voorbereid. We moeten de *adafruit_imageload* bibliotheek gebruiken om ze in het geheugen te laden. Om te vermijden dat bij elke cijferwisseling een afbeelding wordt geladen (dat geeft een risico van vertraging), moeten we onze toevlucht nemen tot een 'sprite sheet' zoals ook wordt gebruikt bij animaties (zie **figuur 15**). Vervolgens moeten we bepalen welk deel van de afbeelding weergegeven moet worden. Dezelfde procedure kan worden toegepast op de knipperende punten.
- Met de joystick op de 1,14" LCD Hat for Pico-module kunnen we een land selecteren en de huidige tijd in dit land en het tijdsverschil met UTC visualiseren. Om contactdender van de joystick te voorkomen, gebruiken we de *adafruit_debouncer* bibliotheek, die op zijn beurt de *adafruit_ticks* bibliotheek nodig heeft.
- De weergave van de vlag maakt ook gebruik van een bitmap in de vorm van een sprite sheet met 25 landen (zie **figuur 16**).

Figuur 16. Het vlaggen-sprite sheet.



Figuur 17. Boomstructuur van de bibliotheken.

- De teksten die overeenkomen met de stad en het tijdsverschil gebruiken labels, net zoals we eerder deden om de tijd op het LCD weer te geven.

De boomstructuur van de in dit project gebruikte interne en externe bibliotheken is geschetst in **figuur 17**.

Conclusie

Er zijn natuurlijk veel manieren om dit project te verbeteren, zoals het gebruik van een LC-display met een hogere resolutie (zonder echter 320 × 240 te overschrijden, want de Raspberry Pi Pico W moet het wel kunnen aansturen), het toevoegen van hoorbare alarmen of het weergeven van actuele weergegevens voor je locatie… Deze verbeteringen kunnen gemakkelijk worden gecodeerd in CircuitPython dankzij de resources die beschikbaar zijn via online-bibliotheken, of de vele tutorials die te vinden zijn op de Adafruit-website. En waarom probeer je niet je eigen CircuitPython-project te programmeren en aan den lijve te ondervinden hoe gemakkelijk het is om deze taal te gebruiken?

- Download: CircuitPython voor de Raspberry Pi Pico W [18].
- Download: externe bibliotheken [19].
- Documentatie: CircuitPython native bibliotheken [20].
- Documentatie: externe bibliotheken [21].
- CircuitPython Essentials tutorials [22] [23].

Je kunt ook andere resources vinden in een (Franstalig) boek dat ik heb geschreven over CircuitPython versie 5.3 (zie **figuur 18**). Het is gepubliceerd door Elektor in 2020, dus sinds die editie is er natuurlijk veel vernieuwd, maar de basis van de taal blijft hetzelfde en de code kan gemakkelijk worden geport naar andere modules zoals de Raspberry Pi Pico W (wat een van de belangrijkste pluspunten is van CircuitPython en zijn 'unified hardware API'!). En als je niet verder komt, kun je me per e-mail je vragen sturen!

220633-03

Figuur 18. Het boek "Initiation au langage CircuitPython et à la puce nRF52840".

Vragen of opmerkingen?

Als u technische vragen hebt of gewoon nieuwe ideeën voor een artikel wilt aandragen, kunt u per e-mail contact opnemen met de auteur via michael.bottin@univ-rennes.fr, of met de Elektor-redactie via redactie@elektor.com.

Gerelateerde producten

- **Raspberry Pi Pico RP2040 W**
  www.elektor.nl/20224
- **Michael Bottin, *Initiation au langage CircuitPython et à la puce nRF52840*,(Franstalig)**
  www.elektor.nl/19523

WEBLINKS

[1] MicroPython: https://en.wikipedia.org/wiki/MicroPython
[2] CircuitPython: https://en.wikipedia.org/wiki/CircuitPython
[3] Raspberry Pico W bij Elektor: https://elektor.nl/raspberry-pi-pico-rp2040-w
[4] LCD Hat voor Pico: https://shop.sb-components.co.uk/products/1-14-lcd-hat-for-pico
[5] Pico RTC Hat: https://shop.sb-components.co.uk/products/pico-rtc-hat
[6] Download CircuitPython: https://circuitpython.org/
[7] Mu Editor: https://codewith.mu/en/download
[8] Mu Editor Interface: https://codewith.mu/en/tutorials/1.2/start
[9] REPL-console: https://codewith.mu/en/tutorials/1.2/repl
[10] Scrollende grafiek: https://codewith.mu/en/tutorials/1.2/plotter
[11] Sneltoetsen: https://codewith.mu/en/tutorials/1.2/shortcuts
[12] TOML-bestand: https://en.wikipedia.org/wiki/TOML
[13] Circup-tool: 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] Het type van de tijdwaarde: https://docs.python.org/3/library/time.html#time.struct_time
[16] adafruit_ds3231-bibliotheek tutorials en voorbeelden: https://learn.adafruit.com/adafruit-ds3231-precision-rtc-breakout/circuitpython
[17] Pico W-pinning: https://datasheets.raspberrypi.com/picow/PicoW-A4-Pinout.pdf
[18] CircuitPython-versie voor de Raspberry Pico W: https://circuitpython.org/board/raspberry_pi_pico_w/
[19] Externe bibliotheken: https://circuitpython.org/libraries
[20] CircuitPython native bibliotheek documentatie: https://docs.circuitpython.org/en/latest/shared-bindings/index.html#modules
[21] Externe bibliotheken documentatie: https://docs.circuitpython.org/projects/bundle/en/latest/drivers.html
[22] CircuitPython Essentials Tutorials 1: https://learn.adafruit.com/welcome-to-circuitpython/circuitpython-essentials
[23] CircuitPython Essentials Tutorials 2: https://learn.adafruit.com/welcome-to-circuitpython/
[24] Software-download: https://elektormagazine.nl/220633-03