INSTYTUT INFORMATYKI
Wydział Informatyki AGH

Ćwiczenie laboratoryjne

Raspberry Pi i taśmy LED ze sterownikiem WS2812B

Nazwa kodowa: **`rpi-ws`**. Wersja **`20231120.0`**

Ada Brzoza
ada.brzoza@agh.edu.pl

Cel ćwiczenia

Celem ćwiczenia jest zapoznanie się z wybranymi zagadnieniami tworzenia oprogramowania dla komputerów jednopłytkowych z mikroprocesorem aplikacyjnym korzystając z wysokopoziomowego języka programowania.

Tym razem zajmiemy się oprogramowaniem pasków z diodami LED posiadającymi układy sterowników. Dzięki wbudowanym sterownikom, każda dioda LED na pasku może świecić na inny kolor i dla każdej diody LED na pasku możemy niezależnie ustawiać jasności składowych koloru czerwonego, zielonego i niebieskiego w zakresach od 0 do 255. A to wszystko przy pomocy jednego wyprowadzenia cyfrowego :) Daje nam to ogromne możliwości budowy ciekawych efektów świetlnych.

Materiały

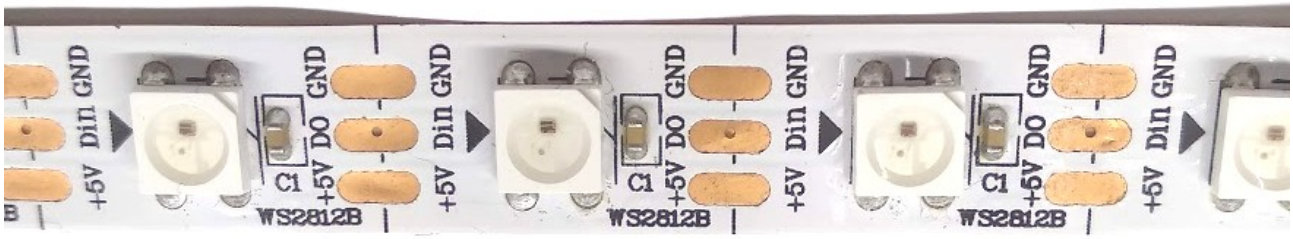
Wymienione poniżej materiały są dostępne w laboratorium, w którym prowadzone są zajęcia z platformą Raspberry Pi. W przypadku samodzielnej lub zdalnej realizacji ćwiczenia należy zaopatrzyć się w te nie we własnym zakresie. Potrzebujemy następujące elementy:

1. komputer Raspberry Pi 4B (a jeśli zaczynamy od instalacji nowego obrazu, można użyć właściwie dowolnego modelu Raspberry Pi),
2. monitor komputerowy z interfejsem HDMI – raczej dla ułatwienia, ponieważ z Raspberry Pi można łączyć się także przez SSH,
3. klawiatura i mysz z interfejsami USB (alternatywnie można uzyskać dostęp do Raspberry Pi przez interfejs sieciowy i SSH, jednak jest to nieco trudniejsze niż podejście opisane w niniejszej instrukcji),
4. taśma diod LED ze sterownikami WS2812B lub pasujący gadżet o nazwie handlowej Neopixel,
5. opcjonalnie przycisk chwilowy typu tact-switch,
6. typowe akcesoria, takie jak przewody połączeniowe do płytek breadboard, zasilania Raspberry Pi, etc.

Podłączenie sprzętu

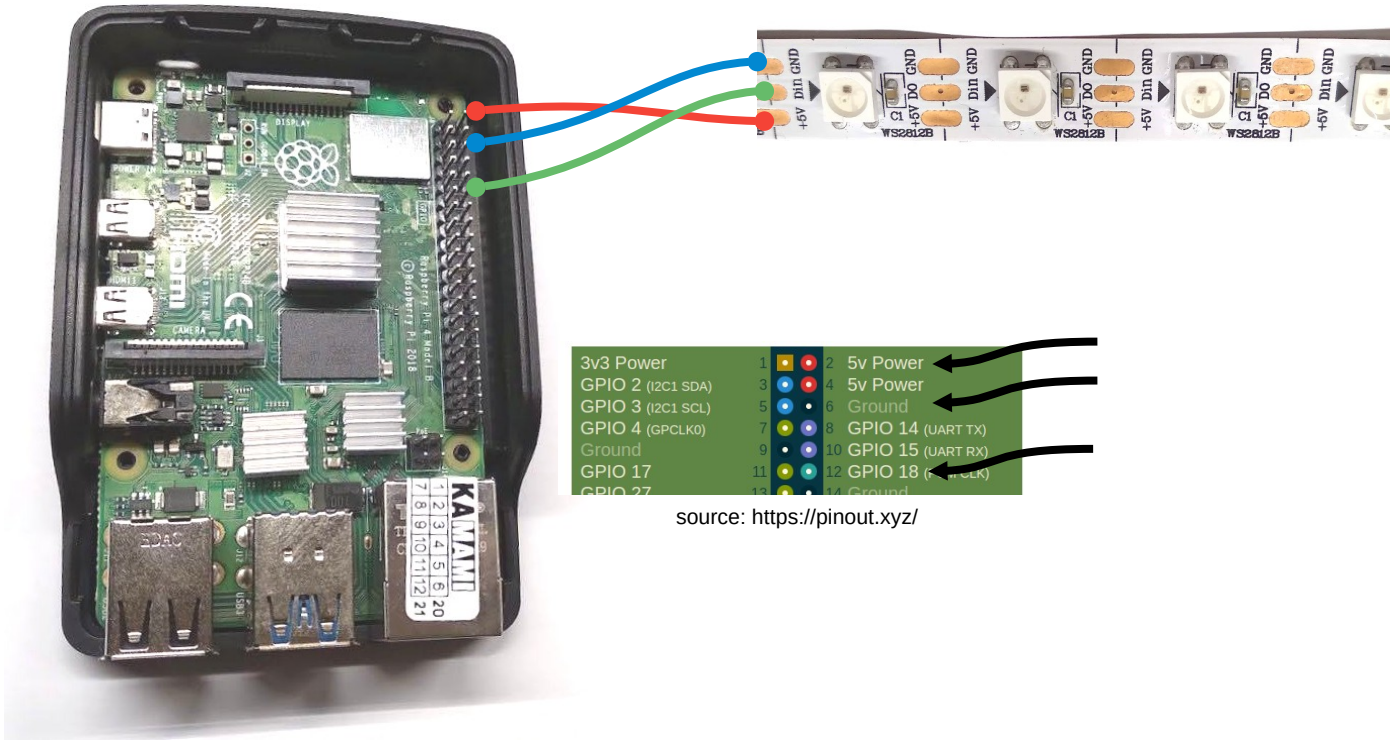
Taśmy LED ze sterownikiem WS2812B mogą być bardzo proste w podłączeniu, jeśli nie musimy zasilać dużej liczby diod LED. Taką uproszczoną sytuację mamy laboratorium, dzięki czemu zasilanie paska diod LED można pobierać bezpośrednio ze złącza szpilkowego Raspberry Pi.

Przykład fragmentu taśmy LED ze sterownikiem WS1812B został przedstawiony na fotografii poniżej.



Typowo, tego typu taśmy docina się w zaznaczonych miejscach do pożądanej długości. Do uzyskanych fragmentów dołącza się przewód albo lutując pola kontaktowe paska do przewodu albo stosując specjalne złącza. Przed podłączeniem do Raspberry Pi należy zidentyfikować złącza przewodu przylutowanego do paska na stanowisku:

- które wyprowadzenie to linia zasilania (tutaj **+5V**),
- które to wejście sygnału cyfrowego (tutaj to **Din** ale może być oznaczone jakoś inaczej, np. „in”),
- a które to masa (tutaj **GND**).



source: https://pinout.xyz/

Oprogramowanie

Oprogramowanie do sterowania diodami LED na pasku napiszemy w języku Python używając dostępnych i sprawdzonych bibliotek z Adafruit Industries, LLC:

- NeoPixel - https://github.com/adafruit/Adafruit_NeoPixel
- Warstwy ujednolicającej Blinka https://pypi.org/project/Adafruit-Blinka/

Blinka służy do tego, aby moduły tworzone dla języka CircuitPython (dla platform z mikrokontrolerami do zastosowań wbudowanych) działały również na komputerach jednopłytkowych, w szczególności Raspberry Pi oraz by mogły być zastosowane przy użyciu języka Python:

https://learn.adafruit.com/neopixels-on-raspberry-pi/python-usage

W niniejszej instrukcji opisano początkowe etapy tworzenia skryptu dla pasków diod LED niezależnie od wybranych narzędzi programowych oraz od interfejsu, którym się posługujemy - czy jest to SSH, lokalny terminal, czy edytor w GUI i terminal.

1 Utworzenie skryptu i bazowa treść

Zaczynamy od opcjonalnego shebanga

```
#!/bin/python3.9
```

a następnie dołączamy biblioteki, których użyjemy:

```
import board
import time
import neopixel
```

Dla przejrzystości kodu można utworzyć stałą określającą liczę diod ze sterownikami w dostępnym pasku, tutaj przykładowo dla 8 diod z WS2812B:

```
pixel_count = 8
```

Następnie możemy spróbować utworzyć obiekt reprezentujący pasek diod LED:

```
pixels = neopixel.NeoPixel(board.D18, 10, brightness=0.9, auto_write=True,pixel_order=neopixel.GRB)
```

W przykładzie mamy następujące parametry:

- brightness=0.9 → Pozwala na ustawienie maksymalnej znormalizowanej jasności na 90% możliwej do uzyskania jasności, co może być wygodne, jeśli diody LED świecą zbyt intensywnie przy ustawionej pełnej jasności.
- auto_write → Jest to parametr określający, czy sterownik ma automatycznie diody w pasku LED po każdej modyfikacji koloru. Ustawienie tego parametru na False nałoży na nas konieczność wywoływania metody show(), każdorazowo, gdy chcemy wyświetlić zadane kolory i jasności. Może to mieć uzasadnienie, jeśli uzyskanie konkretnych jasności kosztuje wiele operacji zapisu.

- pixel_order → Określa kolejność kolorów w zastosowanych taśmach LED.

Utworzony obiekt **`pixel`** możemy traktować jak listę. Każdy element listy reprezentuje jedną diodę LED. Element **`pixels[0]`** reprezentuje pierwszą diodę, podłączoną „najbliżej” interfejsu Raspberry Pi.

Uwaga: aby skrypt mógł działać, należy uruchamiać go z wyższymi uprawnieniami, np.

```
sudo python mojanazwaskryptu.py
```

2 Przydatne właściwości i metody

Mając zainicjalizowany obiekt **`pixels`** typu **`NeoPixel`**, możemy w bardzo prosty sposób ustawić jasność poszczególnych składowych kolorów dla każdej diody LED w obsługiwanym pasku. Kolor piksela jest określany 3-elementową krotką, w której podajemy zawartość koloru czerwonego, zielonego i niebieskiego (R, G, B), gdzie każda składowa powinna mieścić się w zakresie 8-bitowym, tj. od 0 do 255. Przy takim schemacie kodowania np.:

- np. pixels[2] = (255,0,0) spowoduje ustawienie diody o indeksie 2 (tj. trzecia od początku paska) na najjaśniejszy kolor czerwony,
- (0,255,0) → zielony,
- (0,0,255) → niebieski,
- (255,0,255) → magenta,
- ... itd ...

Z kolei przydatne metody to m.in.:

- pixels.show() → Uaktualnia stan diod LED zależnie od informacji uprzednio wpisanych do **`pixels`** i jest przydatna, gdy ustawimy przy inicjalizacji **`auto_write`** na **`False`**.
- pixels.fill((R,G,B)) → Wpisuje zadany kolor (R,G,B) do wszystkich diod LED w pasku.

Pełne informacje na temat dostępnego API mamy na stronie projektu:

https://github.com/adafruit/Adafruit_NeoPixel

Oczywiście do własnych eksperymentów przydatna może być także metoda realizująca opóźnienie czasowe, tutaj np. na pół sekundy:

```
time.sleep(0.5)
```

3 (jeszcze) przyjemniejszy sposób nauki podstaw Pythona

Pasek z diodami LED, który jest reprezentowany w skrypcie w języku Python jako lista może być świetną motywacją do poznania i przećwiczenia w efektowny sposób podstaw iterowania

elementów listy, w tym tworzenia pętli. Jeśli jeszcze nie zajmowaliśmy się tymi zagadnieniami, możemy wykonać parę ćwiczeń dotyczących pętli przez generowanie ciekawych efektów świetlnych, np.

1. gadżet udający termometr słupkowy, wizualizujący dane z pewnego zakresu włączając coraz więcej diod LED na pasku,
2. „niby termometr słupkowy" z punktu wyżej jednak zmieniający kolor zależnie od wartości,
3. świecący punkt (1-2 diody), który „odbija się" od początku i od końca paska,
4. świecący punkt jak wyżej, który także zmienia kolor zależnie od położenia.

Przykładowe materiały dotyczące list:

https://www.w3schools.com/python/python_for_loops.asp

https://www.w3schools.com/python/python_while_loops.asp

4 Bias lighting czyli „nie-ambilight"

Dzięki temu gadżetowi możemy oświetlić ścianę za monitorem lub telewizorem na kolor zależny od tego, jaki obraz jest wyświetlany na ekranie. Z racji, że technologia o nazwie Ambilight była wprowadzona komercyjnie, użyjemy tutaj określenia technicznego ***bias lighting***, a w ramach tej części ćwiczenia zbudujemy prosty model takiego systemu działający dla Raspberry Pi. Sterowalne diody LED dostępne w formie pasków mogą zostać łatwo użyte do samodzielnego zbudowania gadżetu działającego bardzo podobnie jak oryginalny Ambilight.

Aby zmodyfikować nasz dotychczasowy projekt na model gadżetu *bias lighting* warto zacząć przede wszystkim od odczytania kolorów pikseli wyświetlanych na monitorze. Można to zrobić na wiele sposobów, z czego może dalekim od optymalnego, ale za to bardzo prostym jest zastosowanie bardzo popularnej biblioteki **Pillow** oraz screeninfo dla języka Python.

https://pypi.org/project/Pillow/

https://pypi.org/project/screeninfo/

Dopiszmy zatem na początku naszego skryptu, obok innych dyrektyw **`import`**:

```
from PIL import ImageGrab
from screeninfo import get_monitors
import numpy as np
```

Warto zacząć od pobrania informacji o rozdzielczości monitora (lub monitorów) dołączonych do Raspberry Pi:

```
for m in get_monitors():
    print(str(m))
    print('disp w=',m.width,' disp h=',m.height)
```

Następnie pobierzmy i dla ułatwienia wyświetlmy fragment obrazu z ostatniego (czyli prawdopodobnie też jedynego) monitora dołączonego do Raspberry Pi, np.:

```
left = 1/4*m.width
right = 3/4*m.width
upper = 100
lower = m.height / 5
img = ImageGrab.grab(bbox =(left,upper,right,lower,))
```

W przykładowym wywołaniu do obiektu **img** pobieramy fragment obrazu wg krotki (x0, y0, x1, y1), tj.

- od 1/4 szerokości ekranu do 3/4 szerokości ekranu,
- od 100 pikseli od góry ekranu do 1/5 wysokości ekranu licząc od góry.

Testowo możemy wyświetlić na ekranie obraz wywołując

```
img.show()
```

(jednak nie jest zalecane uzywanie **img.show()** w pętli ;))

Dość mało wyszukanym (jednak dość prostym w implementacji) sposobem obliczenia jasności pikseli dla tasiemki diod LED jest obliczenie średniej jasności wszystkich trzech składowych kolorystycznych w zarejestrowanym obrazie. W tym celu konwertujemy uzyskany obraz na macierz **numpy** oraz obliczamy średnie dla poszczególnych składowych, które są indeksowane trzecim indeksem tych macierzy (pierwsze dwa indeksy iterują numery kolumn i rzędów pikseli). Uzyskane średnie wpisujemy wprost do krotki **rgb**:

```
imarr = np.asarray(img)
rgb = (
  (int)(np.mean(imarr[:,:,0])),
  (int)(np.mean(imarr[:,:,1])),
  (int)(np.mean(imarr[:,:,2])) )
```

Następnie możemy użyć tej krotki do ustawienia wszystkich pikseli w dołączonym do Raspberry Pi i zainicjalizowanym pasku LED:

```
pixels.fill( rgb )
```

Aby uzyskać prosty efekt bias lighting, wystarczy teraz w nieskończonej pętli (np. **while True**) wywoływać metody: pobierającą obraz, konwertującą obraz do macierzy numpy, obliczający średnią dla składowych kolorów i wysterowujący diody LED na dołączonej taśmie.

W wersji ambitnej można podzielić zarejestrowany obraz na obszary i sterować osobno poszczególnymi diodami LED odpowiadającymi danemu fragmentowi obrazu. W ten sposób możemy uzyskać efekt jeszcze bliższy „prawdziwemu” oświetleniu bias lighting. Można także wybrać inny, bardziej zaawansowany sposób obliczania jasności diod LED w dołączonym pasku.