

pliki:

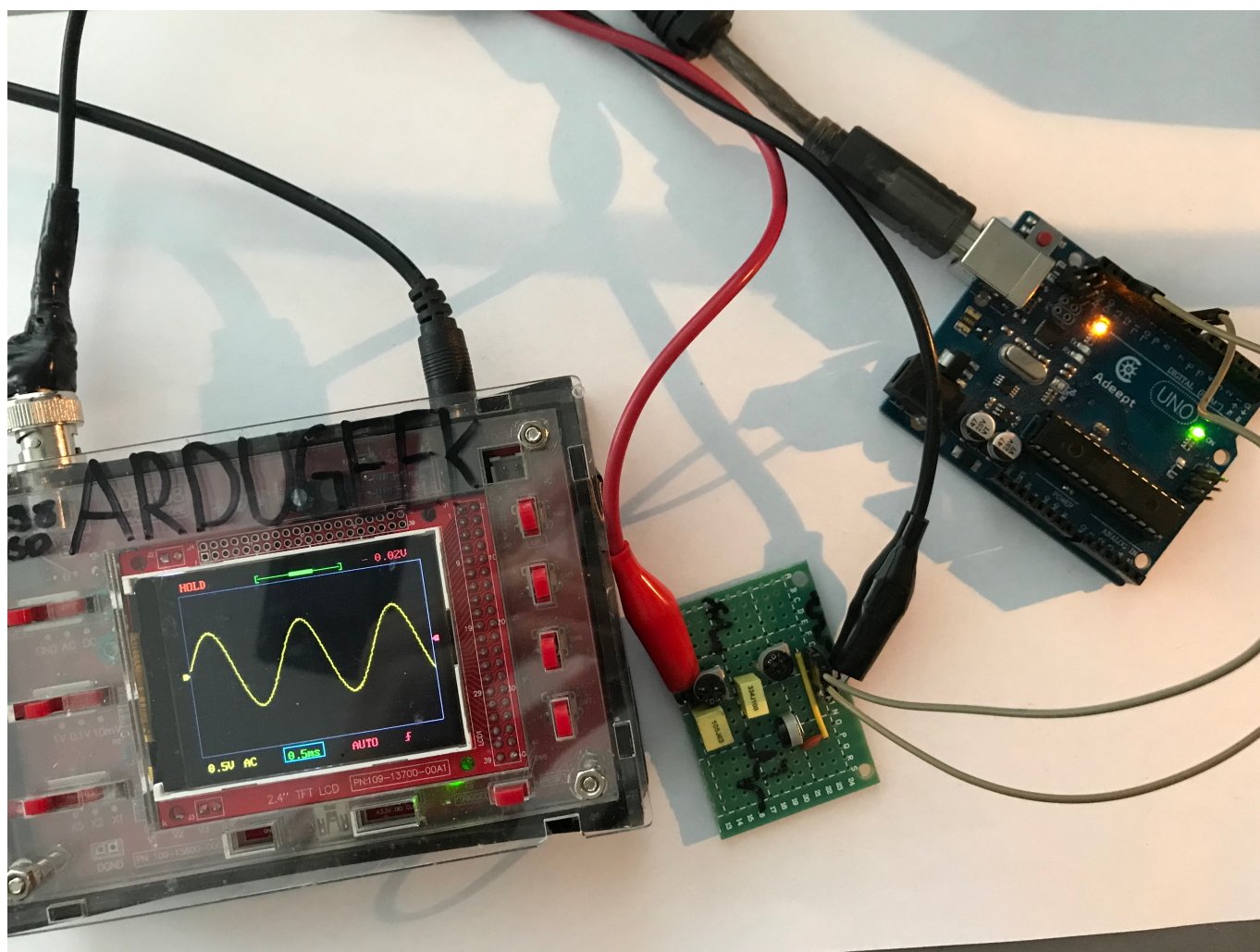
trans_ard_wav.py

translate_to_pwm.py

wave_file_writer.py

ArdunioADCs-Hack

Proste rozwiązanie do przekształcania wygenerowanych przez Pythona fal do wartości PWM dla Arduino oraz konwertowania wartości wejścia analogowego Arduino na pliki WAV.



Arduino

Spis treści:

1. W jakich projektach możesz użyć tego narzędzia?
2. Jakie są części tego narzędzia?

3. Wyjaśnienie części narzędzi
4. Jak korzystać z tego narzędzia?
5. Lista rzeczy do zrobienia
6. Wnioski

Gdzie możesz użyć tego narzędzia?

Stworzyłem ten zestaw narzędzi, ponieważ pomyślałem, że będzie to interesujące, aby rejestrować fale za pomocą wejść analogowych Arduino, a następnie używać np. Audacity do edytowania i inspekcji zarejestrowanej fali za pomocą np. FFT. Może być także używane do rejestrowania odczytów z czujnika, można zostawić Raspberry Pi z prostym skryptem PyFirmata, który będzie odczytywał wartości i zapisywał je do pliku tekstowego, a potem po dłuższym czasie możesz wrócić, skopiować plik, wprowadzić go do mojego skryptu, który go przekonwertuje, a następnie narysować go lub zapisać w pliku WAV.

Części tego narzędzia.

To narzędzie składa się z kilku części:

* values.txt - w tym pliku wklejasz wartości do wave_file_writer.py, powinny być w zakresie od -32768 do 32768 * input.txt - w tym pliku wklejasz lub zapisujesz wartości odczytane przez ADC Arduino, powinny być w zakresie od 0 do 1024 * pwm.txt - to plik, w którym umieszczasz wartości do translate_to_pwm.py, powinny być w zakresie od -1 do 1 * wave_file_writer.py - ten skrypt służy, jak sama nazwa wskazuje, do zapisywania danych z values.txt do pliku .wav * translate_to_pwm.py - ten skrypt konwertuje wartości z zakresu -1, 1 do zakresu 0, 255, które są wartościami dla wyjścia PWM Arduino * trans_ard_wav.py - ten plik bierze wartości wejścia analogowego Arduino w zakresie od 0 do 1024 i konwertuje je na wartości 16-bitowego pliku .wav

Wyjaśnienie części

wave_file_writer.py

cały kod skryptu

```
import wave, struct, math, random, numpy

text = open("./values.txt")
string = text.read()
text.close()

sampleRate = 44100.0
duration = 100.0
frequency = 440.0
obj = wave.open('sound_writer.wav', 'w')
obj.setnchannels(1) # mono
obj.setsampwidth(2)
obj.setframerate(sampleRate)
```

```
audio=[]
audio = string.splitlines()
audio2 = []
for i in audio:
    audio2.append(int(i))
print(audio2)
for c in audio2:
    data = struct.pack('<h', c)
    obj.writeframesraw( data )
obj.close()
```

```
text = open("./values.txt")
string = text.read()
text.close()
```

1. Skrypt otwiera plik w bieżącym katalogu o nazwie values.txt
2. Odczytuje zawartość pliku i przypisuje ją do zmiennej o nazwie *string*
3. Zamyka plik o nazwie text

```
sampleRate = 44100.0
duration = 100.0
frequency = 440.0
```

Tutaj przypisuję parametry pliku WAV

```
obj = wave.open('sound_writer.wav', 'w')
obj.setnchannels(1) # mono
obj.setsampwidth(2)
obj.setframerate(sampleRate)
```

1. Skrypt otwiera plik o nazwie sound_writer.wav z uprawnieniami do zapisu w nim i przypisuje go do zmiennej *obj*
2. Określona jest liczba kanałów w pliku
3. Określona jest długość pojedynczego próbki
4. „Częstotliwość próbkowania” pliku audio jest określona i przypisana do zmiennej *sampleRate*

```
audio=[]
audio = string.splitlines()
audio2 = []
```

1. Tworzymy listę *audio*
2. Wywołujemy metodę splitlines() na zmiennej *string*, która dzieli ją na linie i przypisuje do zmiennej *audio*
3. Tworzymy listę *audio2*

```
for i in audio:
    audio2.append(int(i))
print(audio2)
```

1. Pętla for jest wywoływana na zmiennej *audio*
2. Każdy element *i* jest konwertowany na liczbę całkowitą
3. Następnie drukujemy zmienną *audio*

```
for c in audio2:
    data = struct.pack('<h', c)
    obj.writeframesraw( data )
obj.close()
```

1. Używamy pętli for do iterowania przez listę *audio2*
2. Używamy funkcji struct.pack() do przypisania wartości do zmiennej *data*, która przygotowuje miejsce na zapis danych WAV
3. Następnie zapisujemy ramki danych do naszego pustego obiektu o nazwie *data*
4. Zamyka plik .wav

= To cały mechanizm działania tego narzędzia. Może być używane do zapisywania czegokolwiek do pliku audio, o ile jest to w pliku values.txt i mieści się w zakresie +/- 32768 =

translate_to_pwm.py

cały kod skryptu

```
import numpy as np

text = open("./pwm.txt")
string = text.read()
text.close()
lines = string.splitlines()
values = []
result = []
counter = 0
for i in lines:
    values.append(float(i))
for i in values:
    counter += 1
    result.append(int(round(np.interp
        (i, [-1, 1], [0, 255]), 0)))
print(result)
```

```
text = open("./pwm.txt")
string = text.read()
text.close()
lines = string.splitlines()
values = []
result = []
counter = 0
```

1. Skrypt otwiera plik pwm.txt i zapisuje go w zmiennej *text*
2. Plik przypisany do zmiennej *text* jest odczytywany i wynik zapisany w zmiennej *string*
3. Następnie plik *text* jest zamykany
4. Rozdzielamy zmienną *string* na linie
5. Tworzymy listę *values*
6. Tworzymy listę *result*
7. Tworzymy zmienną *counter*

```
for i in lines:
    values.append(float(i))
```

W tej pętli przechodzimy przez wartości w liście *lines* i dodajemy je do listy *values*, po konwersji na typ float

```
for i in values:
    counter += 1
    result.append(int(round(np.interp
        (i, [-1, 1], [0, 255]), 0)))
print(result)
```

W tej pętli przechodzimy przez dane w liście *values*, dodajemy 1 do zmiennej *counter* (która nie jest używana), dodajemy wartość zmiennej *i* po jej zaokrągleniu (bez miejsc po przecinku) oraz interpolujemy ją z zakresu -1, 1 do zakresu 0, 255, a następnie drukujemy wynik ##### to wszystko, co powinieneś wiedzieć o tym narzędziu, może być używane do konwersji dowolnej wartości na wartość cyklu PWM. Poniżej znajduje się mały przykład, jak obliczyć wartości sinusoidalne do pliku pwm.txt, aby przekonwertować je na PWM

```
Python 3.6.9 (default, Nov 7 2019, 10:44:02)
[GCC 8.3.0] na linuxie
Typ "help", "copyright", "credits" lub "license", aby uzyskać więcej
informacji.
>>> import math as mth
>>> import numpy as np
>>> range = np.arange(0, 6.28, 0.01)
>>> result = []
>>> for i in range:
...     result.append(mth.sin(i))
...
>>> print(result)
[0.0, 0.009999833334166664, 0.01999866669333308, 0.02999550020249566,
0.03998933418663416, 0.04997916927067833, 0.059964006479444595,
0.06994284733753277, 0.0799146939691727, 0.08987854919801104,
0.09983341664682815, 0.10977830083717481, 0.11971220728891936,
0.12963414261969486, 0.1395431146442365, 0.14943813247359922,
0.15931820661424598, 0.16918234906699603, 0.17902957342582418,
0.18885889497650057, 0.19866933079506122, 0.20845989984609956,
0.21822962308086932, 0.2279775235351884, 0.23770262642713458,
0.24740395925452294, 0.2570805518921551, 0.26673143668883115,
0.27635564856411376, 0.28595222510483553, 0.29552020666133955,
0.3050586364434435, # i tak dalej ... w nieskończoność ...
```

= Spróbuj eksperymentować z różnymi wartościami kroków (ostatnia wartość w funkcji np.arange()), np. 0.1; 0.5; 1. Zobacz, jak jakość fali zmieni się po interpolacji ich do zakresu +/- 32768 i zapisaniu ich za pomocą wave_file_writer.py do pliku =

trans_ard_wav.py

```
import numpy as np

text = open("./input.txt")
string = text.read()
text.close()
lines = string.splitlines()
values = []
result = []
counter = 0;
for i in lines:
    values.append(int(i))
for i in values:
    counter += 1
    result.append(int(round(np.interp
        (i,[0,1024],[-32768,32768]),0)))
for i in result:
    print(i)
```

```
text = open("./input.txt")
string = text.read()
text.close()
lines = string.splitlines()
values = []
result = []
counter = 0;
```

Dobrze, teraz sprawdźmy, co robi ten fragment kodu. Pierwsza linia otwiera plik, potem standardowo odczytujemy jego zawartość i zamykamy plik. Następnie pod zmienną „lines” przypisujemy tekst z pliku, ale jest on podzielony na oddzielne linie. Tworzymy kilka zmiennych, które będą przydatne później.

```
for i in lines:
    values.append(int(i))
for i in values:
    counter += 1
    result.append(int(round(np.interp
        (i,[0,1024],[-32768,32768]),0)))
for i in result:
    print(i)
```

Pierwsza pętla for odczytuje każdą wartość i konwertuje ją z typu str na int, następnie w kolejnej pętli dodajemy 1 do zmiennej „counter” (która nie jest używana), zaokrąglamy wartości (bez miejsc po przecinku) i interpolujemy je z zakresu 0,1024 na zakres 16-bitowego pliku audio. Ostatnia pętla for drukuje wynik każdej wartości w nowej linii, a po tym możemy przekierować wynik do innego pliku.

Lista rzeczy do zrobienia

- ☒ przetłumaczenie z Pythona na cykl PWM

- ☒ przetłumaczenie z wejścia analogowego Arduino na plik WAV
- ☐ generowanie prostych fal w plikach .wav
- ☐ przetłumaczenie z wejścia analogowego Arduino na wykres matplotlib
- ☐ stworzenie GUI dla całego projektu
- ☐ stworzenie prostego generatora tonów .wav w oparciu o GUI/tekst
- ☐ konwerter ASCII na sygnał cyfrowy w formacie .wav i dla Arduino

Jeśli masz pomysły na ciekawe ulepszenia tego projektu, skontaktuj się ze mną lub po prostu wyślij swoją zmianę.

Wnioski

Moim zdaniem to był interesujący projekt. Może nauczyć trochę o cyfrowym przetwarzaniu sygnałów .wav i trochę o Arduino. Jeśli masz trochę więcej wolnego czasu i chcesz obejrzeć filmy wyjaśniające podobne projekty do tego, odwiedź moją [stronę internetową](https://wiki.ostrowski.net.pl/)

From:

<https://wiki.ostrowski.net.pl/> - **Kacper's Wiki**

Permanent link:

<https://wiki.ostrowski.net.pl/doku.php?id=projekty:arduinoadcshack&rev=1746631128>

Last update: **2025/05/07 17:18**