

files:

`trans_ard_wav.py`

`translate_to_pwm.py`

`wave_file_writer.py`

Arduino: ArduinioADCs-Hack

A simple solution for converting Python-generated waveforms to PWM values for the Arduino and converting Arduino analogue input values to WAV files.

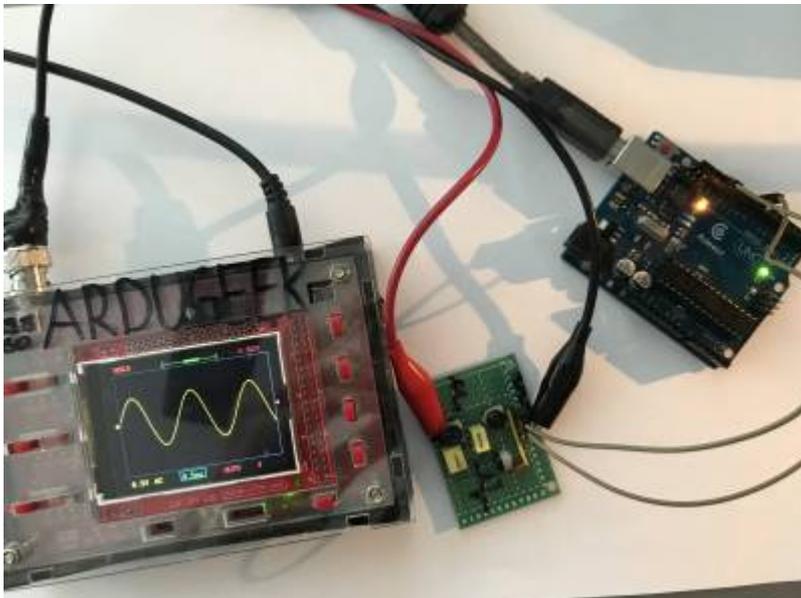


Table of contents:

1. In which projects can you use this tool?
2. What are the parts of this tool?
3. Explanation of the tool parts
4. How to use this tool?
5. To-do list
6. Applications

Where can you use this tool?

I created this toolkit because I thought it would be interesting to record waveforms using the Arduino's analogue inputs, and then use, for example, Audacity to edit and inspect the recorded waveform using, for example, the FFT. It can also be used to record sensor readings, you can leave the Raspberry Pi with a simple PyFirmat script that will read the values and write them to a text file,

and then after a long time you can go back, copy the file, feed it into my script which will convert it and then draw it or write it to a WAV file.

Parts of this tool.

This tool is made up of several parts:

- values.txt - in this file you paste the values into wave_file_writer.py, they should be in the range -32768 to 32768
- input.txt - in this file you paste or write the values read by the Arduino ADC, they should be in the range from 0 to 1024
- pwm.txt - this is the file where you put the values for translate_to_pwm.py, they should be in the range -1 to 1
- pwm.txt - this is the file where you put the values to translate_to_pwm.py, they should be in the range from -1 to 1
- wave_file_writer.py - this script is used, as the name suggests, to write data from values.txt to a .wav file
- translate_to_pwm.py - this script converts values from the range -1, 1 to the range 0, 255, which are the values for the Arduino's PWM output
- trans_ard_wav.py - this file takes the Arduino analogue input values in the range 0 to 1024 and converts them to the values of a 16-bit .wav file

Explanation of the parts

wave_file_writer.py

entire script code

<ignore>

```
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('<ignore> <h', c)
    obj.writeframesraw( data )
obj.close()
```

</ignore></ignore>.

<ignore><ignore>

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

</ignore></ignore>.

1. The script opens a file in the current directory called values.txt
2. Reads the contents of the file and assigns it to a variable named <ignore><ignore></ignore></ignore></ignore>string<ignore><ignore></ignore></ignore>
3. Closes the file named text

<ignore><ignore>

```
</ignore>.
sampleRate <ignore>=</ignore> 44100.0
duration <ignore>=</ignore> 100.0
frequency <ignore>=</ignore> 440.0
<ignore>
```

</ignore></ignore>.

Here I assign the parameters of the WAV file

<ignore><ignore>

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

</ignore></ignore>.

1. The script opens a file named sound_writer.wav with write permissions on it and assigns it to the variable <ignore><ignore></ignore></ignore></ignore>obj<ignore><ignore></ignore></ignore>.
2. The number of channels in the file is specified
3. The length of a single sample is specified

- 4. „Sampling rate” of the audio file is specified and assigned to the variable `sampleRate`

```

</ignore>.
audio=</ignore>[]
audio </ignore> string.splitlines()
audio2 </ignore> []
</ignore>

```

1. We create a list `audio`.
2. We call the `splitlines()` method on the `string` variable, which splits it into lines and assigns it to the `audio` variable.
3. We create a list of `audio2`.

```

</ignore>.
for i in audio:
    audio2.append(int(i))
print(audio2)
</ignore>

```

1. The for loop is called on the variable `audio`.
2. Each element of `audio` and `audio2` is converted to an integer
3. Then we print the variable `audio2`.

```

</ignore>.
for c in audio2:
    data </ignore> struct.pack('<ignore> <h', c)
    obj.writeframesraw( data )
obj.close()

```

1. We use a for loop to iterate through the list

`<ignore><ignore></ignore></ignore>audio2<ignore><ignore></ignore></ignore>`.

2. We use the `struct.pack()` function to assign a value to the variable `<ignore><ignore></ignore></ignore></ignore>data<ignore><ignore></ignore></ignore>`, which prepares a place to store the WAV data
3. We then write the data frames to our empty object named `<ignore><ignore></ignore></ignore>data<ignore><ignore></ignore></ignore>`.
4. Closes the .wav file

`<ignore>=</ignore>` This is the whole mechanism of how this tool works. It can be used to write anything to an audio file, as long as it is in the values.txt file and is within +/- 32768 `<ignore>=</ignore>`.

`<ignore>===</ignore> translate_to_pwm.py <ignore>===</ignore>`.

entire script code

`<ignore><ignore>`

```
</ignore>.
import numpy as np

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

`</ignore></ignore>`.

`<ignore><ignore>`

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

</ignore></ignore>.

1. The script opens the file pwm.txt and stores it in the variable <ignore><ignore></ignore></ignore>text<ignore><ignore></ignore></ignore>.
2. The file assigned to the variable <ignore><ignore></ignore></ignore>text<ignore><ignore></ignore></ignore> is read and the result written to the variable <ignore><ignore></ignore></ignore>string<ignore><ignore></ignore></ignore>.
3. Then the file <ignore><ignore></ignore></ignore>text<ignore><ignore></ignore></ignore> is closed
4. We split the variable <ignore><ignore></ignore></ignore>string<ignore><ignore></ignore></ignore> into lines
5. We create a list of <ignore><ignore></ignore></ignore>values<ignore><ignore></ignore></ignore>.
6. We create a list of <ignore><ignore></ignore></ignore>result<ignore><ignore></ignore></ignore>.
7. We create the variable <ignore><ignore></ignore></ignore>counter<ignore><ignore></ignore></ignore>.

<ignore><ignore>

```
</ignore>.
for i in lines:
    values.append(float(i))
<ignore>
```

</ignore></ignore>.

In this loop, we go through the values in the list <ignore><ignore></ignore></ignore>lines<ignore><ignore></ignore></ignore> and add them to the list <ignore><ignore></ignore></ignore>values<ignore><ignore></ignore></ignore>, after converting them to float type

<ignore><ignore>

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

</ignore></ignore>.

In this loop we go through the data in the list <ignore><ignore></ignore></ignore>values<ignore><ignore></ignore></ignore>, add 1 to the variable <ignore><ignore></ignore></ignore>counter<ignore><ignore></ignore></ignore> (which is not used), add the value of the variable <ignore><ignore></ignore></ignore></ignore> and<ignore><ignore></ignore></ignore> after rounding (no decimal places) and interpolate it from the range -1, 1 to the range 0, 255 and then print the result ##### is all you need to know about

this tool, it can be used to convert any value into a PWM cycle value. Here is a small example of how to calculate sine values into a pwm.txt file to convert them to PWM

<ignore><ignore>

```

</ignore>.
Python 3.6.9 (default, Nov 7 2019, 10:44:02)
[GCC 8.3.0] on linux
Type 'help', 'copyright', 'credits' or 'license' for more information.
>>> import math as mth
>>> import numpy as np
>>> range <ignore>=</ignore> np.arange(0,6.28,0.01)
>>> result <ignore>=</ignore> []
>>> 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, # and so on ... ad infinitum ...
<ignore>

```

</ignore></ignore>.

<ignore>=</ignore> Try experimenting with different step values (the last value in the np.arange() function), e.g. 0.1; 0.5; 1. See how the quality of the waveform changes when you interpolate them to a range of +/- 32768 and save them using wave_file_writer.py to <ignore>=</ignore>.

<ignore>==</ignore> trans_ard_wav.py <ignore>==</ignore>.

<ignore><ignore>

```

</ignore>.
import numpy as np

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

```

.

OK, now let's check what this code snippet does. The first line opens the file, then we read the contents as standard and close the file. Next, we assign the text from the file under the „lines” variable, but it is split into separate lines. We create some variables that will be useful later.

```

.
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)

```

.

The first for loop reads each value and converts it from type str to int, then in the next loop we add 1 to the 'counter' variable (which is not used), round the values (no decimal places) and interpolate them from the range 0,1024 to the range of the 16-bit audio file. The final for loop prints the result of each value on a new line, and after that we can redirect the result to another file.

==== To-do list =====.

- translation from Python to PWM cycle
- translation from Arduino analogue input to WAV file
- generating simple waveforms in .wav files
- translating from Arduino analog input to a matplotlib diagram
- creating a GUI for the whole project
- creating a simple .wav tone generator based on GUI/text
- convert ASCII to digital signal in .wav format and for Arduino

If you have ideas for interesting improvements to this project, please contact me