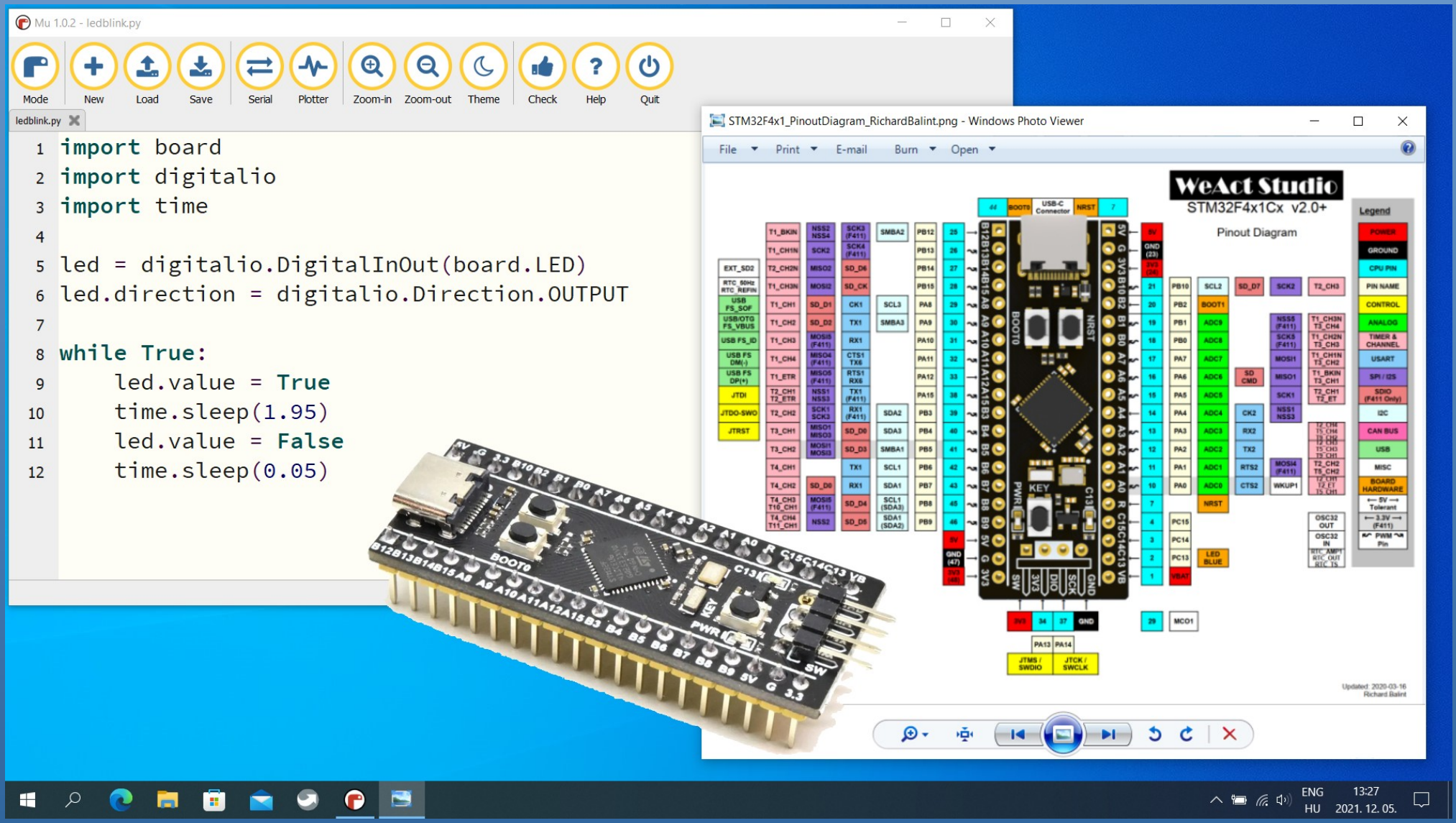
CircuitPython tanfolyam



11. Hangkeltés és hanglejátszás (audiopwmio)

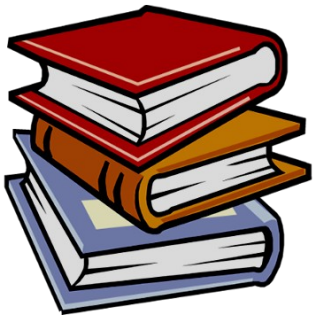
Felhasznált és ajánlott irodalom

Python:

- Mark Pilgrim/Kelemen Gábor: Ugorj fejest a Python 3-ba!
- P. Wentworth et al. (ford. Biró Piroska, Szeghalmy Szilvia és Varga Imre): Hogyan gondolkozz úgy, mint egy informatikus: Tanulás Python 3 segítségével

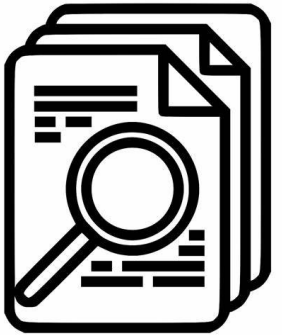
CircuitPython:

- Adafruit: https://circuitpython.org/downloads
- Adafruit: Adafruit CircuitPython API Reference
- Learn Adafruit: CircuitPython Essentials
- Adafruit: github.com/adafruit/Adafruit CircuitPython Bundle



Adatlapok és dokumentáció:

- **STM32F411CE** adatlap és termékinfo
- **STM32F411xC/E** Family Reference Manual
- **WeAct Studio: STM32F4x1 MiniF4**
- **Texas Instruments: LM2904 adatlap**



Felhasznált és ajánlott irodalom

Építési leírások és magyarázatok:

- Adafruit Learn: Todbot's CircuitPython Tricks - Audio
- Next-hack.com: How to play a video on Arduino Uno (5/6): playing a 16 bit 20ksps audio from the SD card
- Uwe Zimmermann: About filters and cut-off frequencies
- Matt Giordano: Arduino Hi Fi Audio PWM

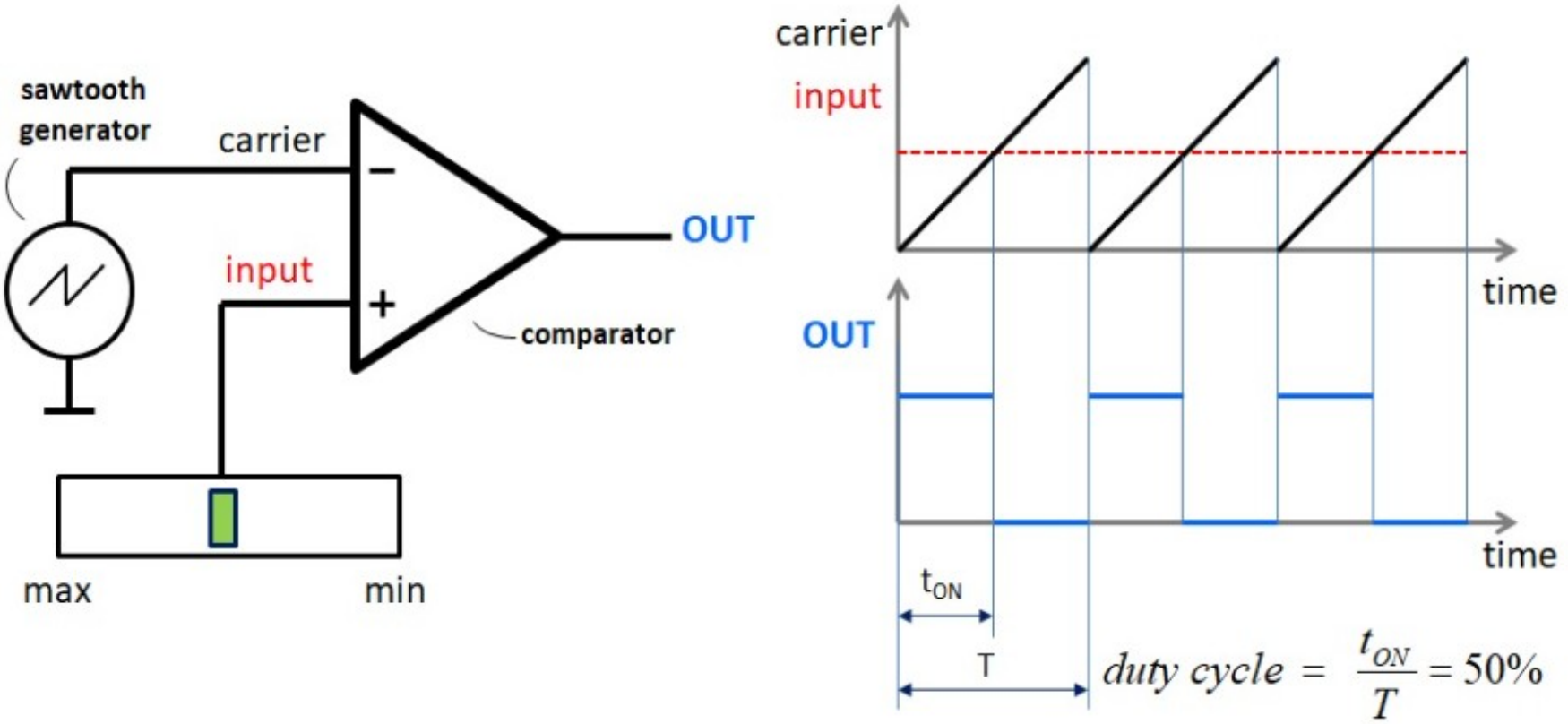
Hangkeltés CircuitPythonban

- Az egyszerű négyszöghullám jelek keltésén (**simpleio.tone()** függvény – lásd: 2021. dec. 9-i előadás) túlmenően **CircuitPythonban** komolyabb lehetőségek is vannak hangkeltésre
- A hardver adottságoktól függően az alábbiak jöhetnek szóba:
  - Beépített DAC használata – **audioio**
  - PWM használata – **audiopwmio** (a kimeneten RC szűrés kell)
  - I2S periféria – **audiobusio** (külső I2S dekóderrel)
- A fentieken kívül szükség lesz az **audiocore** és **audiomp3** modulokra is (ezek is beépített modulok)
- Az **STM32F411CE-blackpill-with-flash** beépített moduljainak listája:

```
_asyncio , _bleio , _pixelmap , adafruit_bus_device , adafruit_pixelbuf , aesio ,
analogio , array , atexit , audiocore , audiomp3 , audiopwmio , binascii ,
bitbangio , bitmaptools , board , builtins , busio , collections , digitalio ,
displayio , errno , fontio , framebufferio , getpass , json , keypad , math ,
microcontroller , msgpack , neopixel_write , onewireio , os , pulseio , pwmio ,
rainbowio , random , re , rtc , sdcardio , select , sharpdisplay , storage ,
struct , supervisor , synthio , sys , terminalio , time , touchio , traceback ,
usb_cdc , usb_hid , usb_midi , vectorio , zlib
```

Analóg kimenet helyettesítése PWM-mel

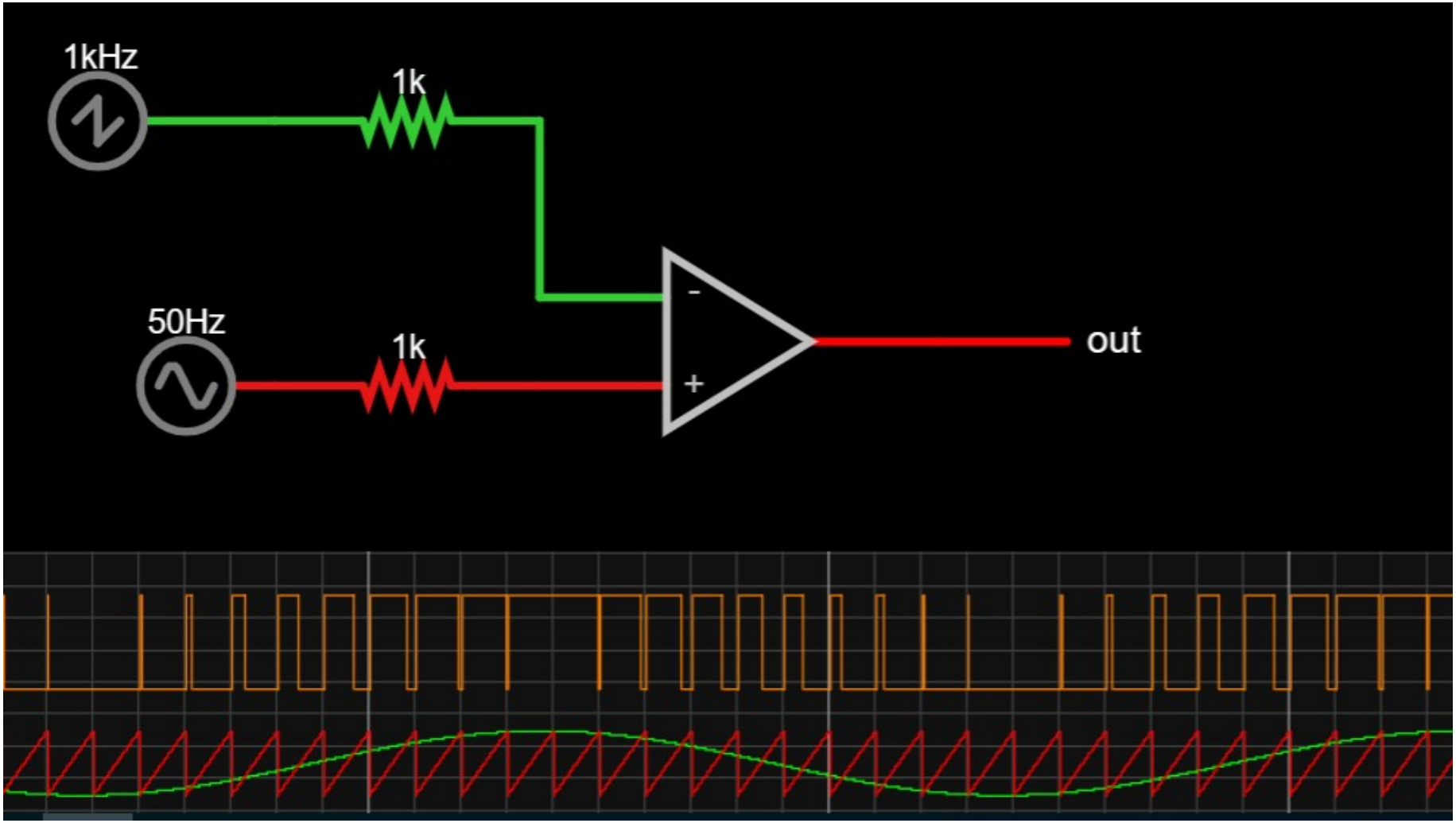
- Ha nincs DAC periféria, akkor **PWM** jellel kell dolgoznunk.
  De hogy lesz **PWM** jel a hanghullámból?
- **Analóg módszer** (például a D osztályú erősítőkben használják):
  A bemenő jelet egy fűrészfog jellel hasonlítja össze egy komparátor



Az ábra forrása: https://next-hack.com/wp-content/uploads/2019/02/analogpwm.jpg

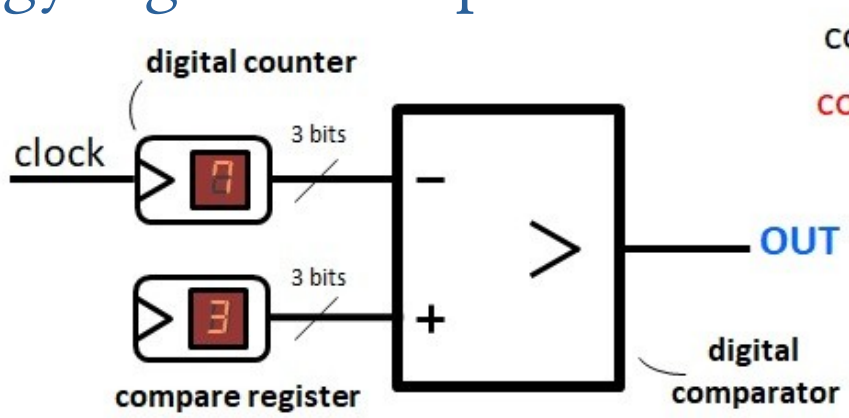
Analóg kimenet helyettesítése PWM-mel

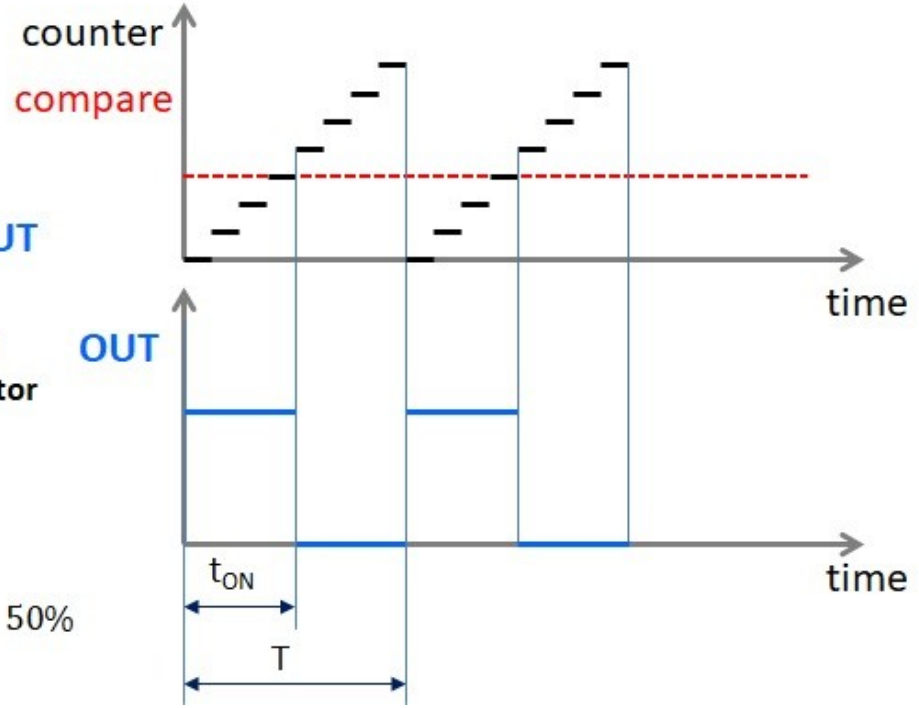
- Az ábrán egy áramköri szimuláció eredménye látható (https://falstad.com/circuit)
- **Az elmentett terv:** circuit-analog-PWM.circuitjs.txt



Analóg kimenet helyettesítése PWM-mel

- **Digitális módszer:** ebben az esetben nem analóg bemenőjel van, hanem egy szám (az egyenlő időközönként mintavételezett jel soron következő értéke) és egy **PWM** csatorna
- Fűrészfogjel helyett most egy számlálónk van, ami 0-tól a maximumig számol (3-bit esetén 7-ig, 8 bit esetén 255-ig), s van egy összehasonlító regiszter, amibe a bemenő értéket írjuk, illetve van egy digitális komparátor



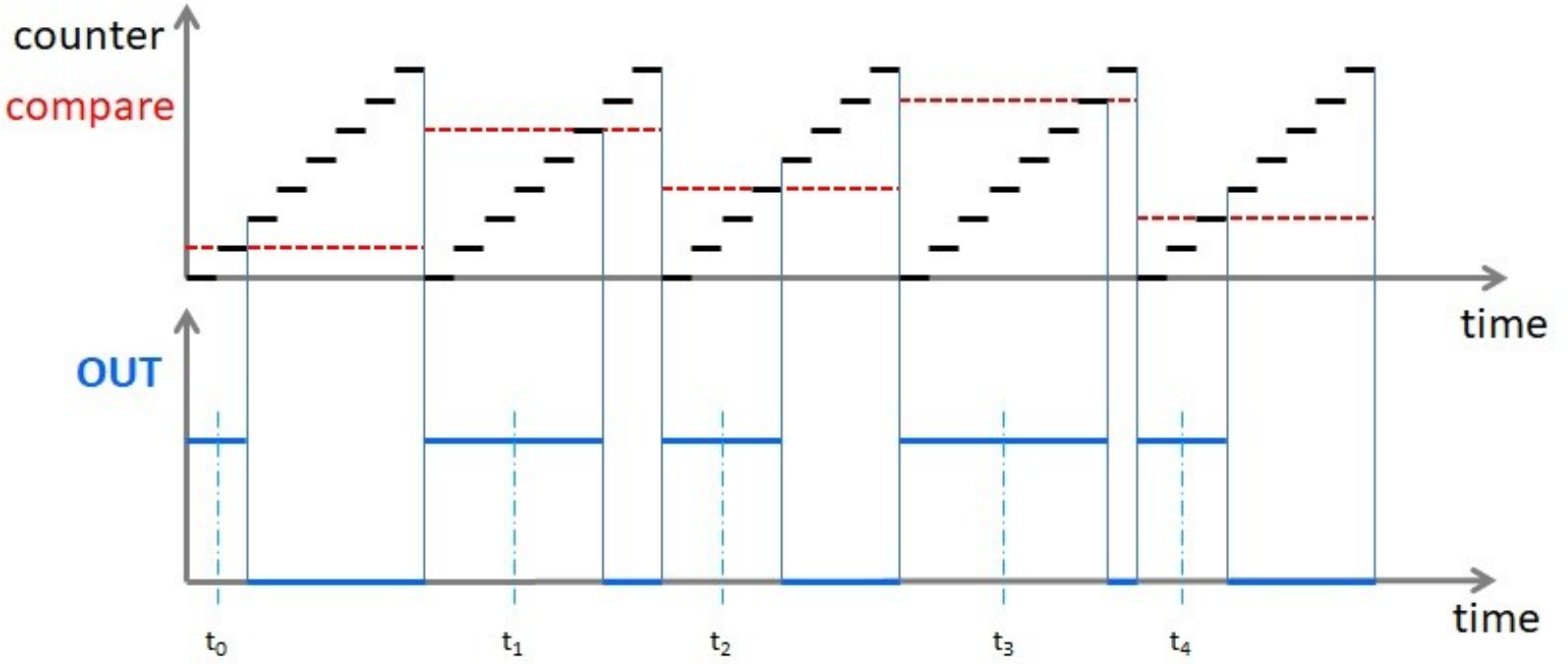


$$duty\ cycle = \frac{t_{ON}}{T} = \frac{compare + 1}{2^n} = 50\%$$

Az ábra forrása: https://next-hack.com/wp-content/uploads/2019/02/3bitdpwm.jpg

A PWM jel előállításának buktatói

- **Fázishiba:** a szokásos ***él-igazított*** (edge aligned) módú jelkeltésnél az impulzusok súlypontja a kitöltéstől függően „vándorol”, ami zajjelet kelt a kimeneten

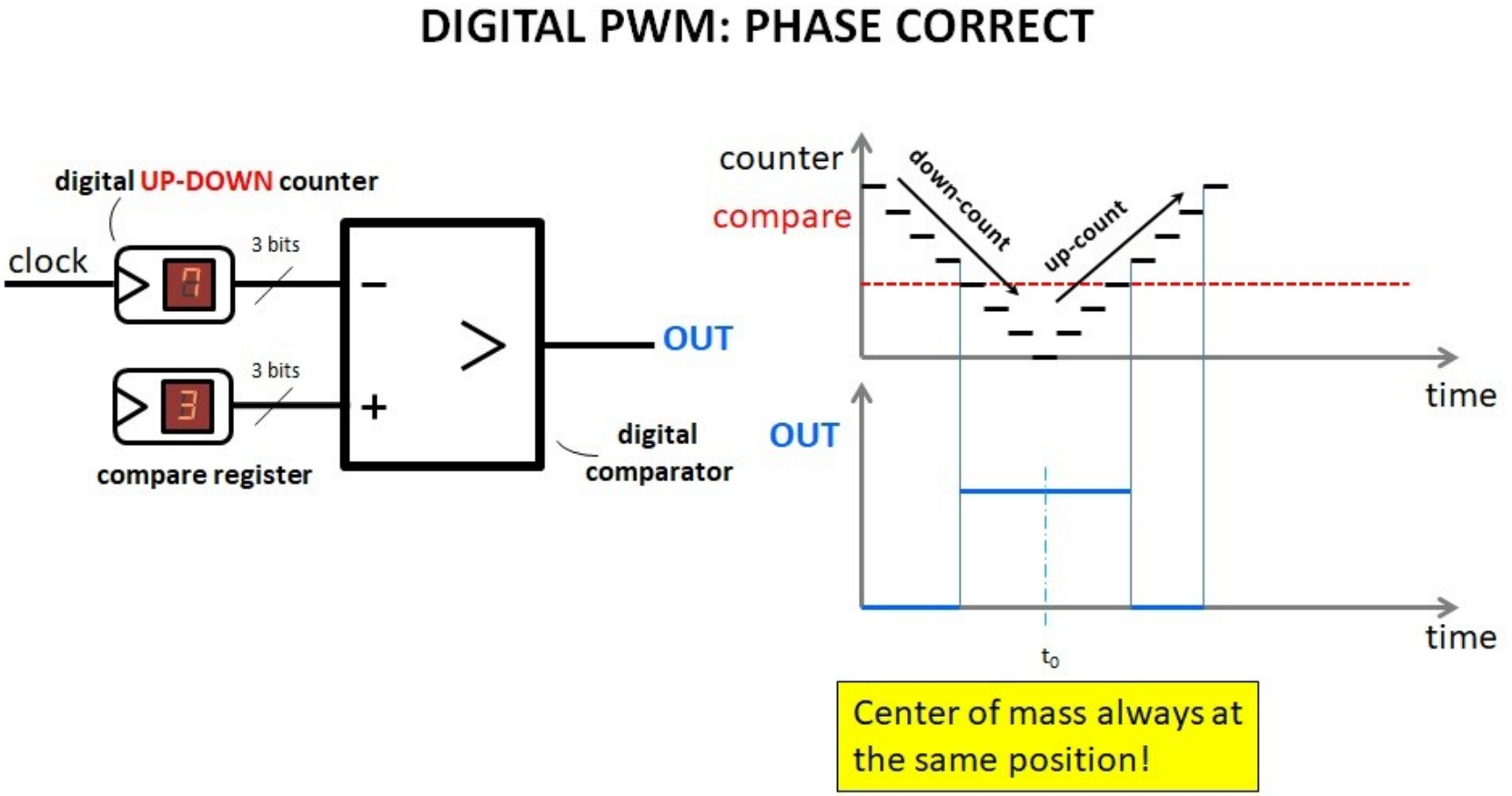


CENTER OF MASS “t_i” OF EACH PULSE NOT EQUALLY SPACED IN TIME! ➔ ADDITIONAL NOISE!

Az ábra forrása: https://next-hack.com/wp-content/uploads/2019/02/phaseerror.jpg

A PWM jel előállításának buktatói

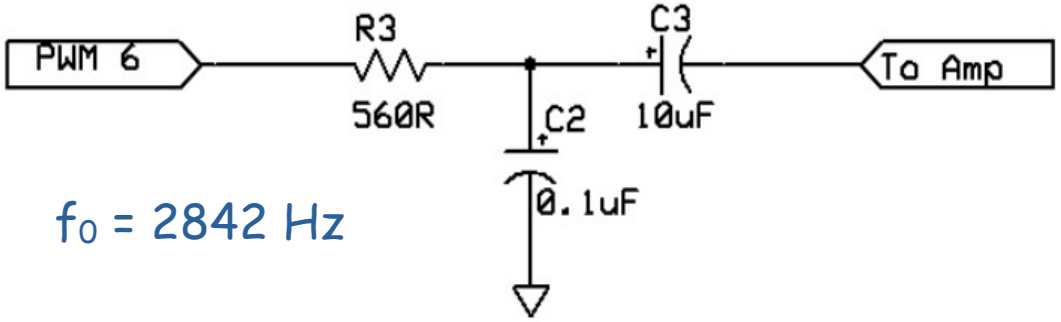
- **A fázishiba kiküszöbölése:** a ***középre igazított*** (center aligned) módú jelkeltésnél az impulzusok súlypontja helyben marad, de ennek az ára az oda-vissza számlálási mód, ami **megfelezi** az időegységenkénti periódusok (a lejátszott minták) számát

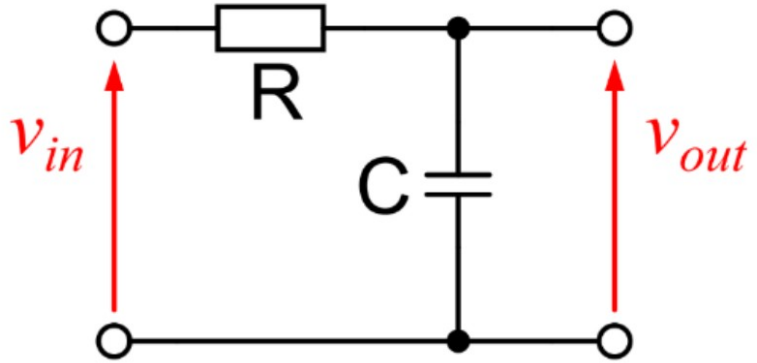


Az ábra forrása: https://next-hack.com/wp-content/uploads/2019/02/phasecorrect.jpg

A PWM jel „helyreállítása”

- Az eredeti jel helyreállításához a PWM jel aluláteresztő szűrésére van szükség. Egy egyszerű egypólusú RC szűrő is megfelelő lehet, ha tolerálni tudjuk a kimenet némi hullámzását
- További javulás érhető el, ha induktivitás, vagy műveleti erősítő felhasználásával két-, vagy többpólusú szűrőt készítünk
- A követelmények azonban ellentétesek, kompromisszumot kell kötni
  - Az ingadozások kiszűréséhez nagy időállandó (alacsony letörési frekvencia) kell
  - A jelszint gyors beállásához pedig minél kisebb időállandójú szűrés szükséges
- Különböző leírásokban különböző értékekkel találkozhatunk:



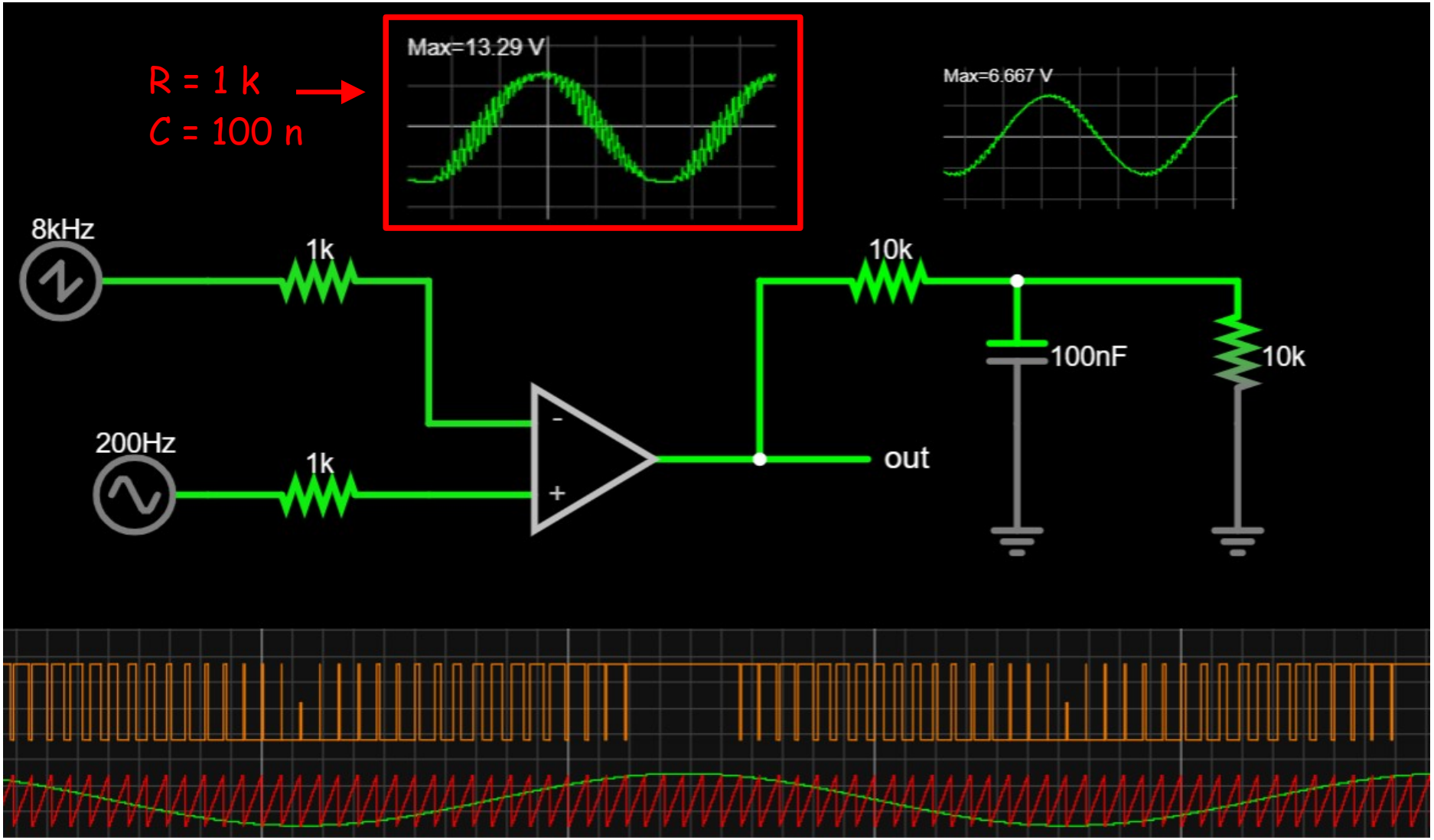


Például: Matt Giordano:
Arduino Hi Fi Audio PWM

$$\frac{v_{out}}{v_{in}} = \frac{1}{1 + j\,\omega RC} \qquad f_0 = \frac{1}{2\pi RC}$$

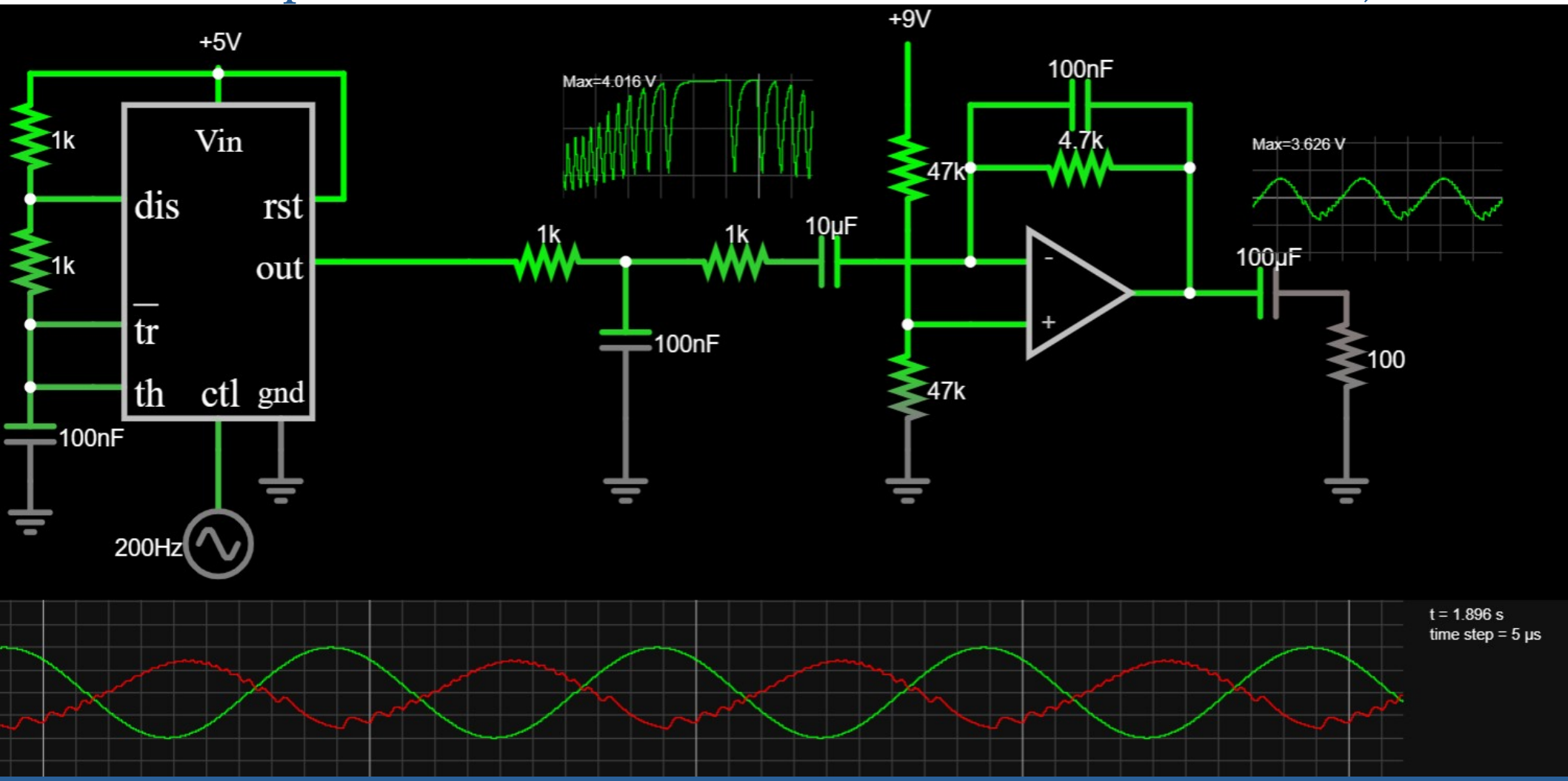
Egyszerű RC szűrő szimuláció

- Az ábrán látható adatokkal f_0 = 159Hz, a szűrés már jelentősen lecsökkenti a 200 Hz-es jel amplitúdóját. Ha lecsökkentjük az ellenállás értékét, akkor pedig a betét ábrán látható jelet kapjuk



Bonyolultabb szűrő szimuláció

- Az ábrán egy kísérleti elrendezés szimulációja látható, ahol a másodlagos szűrést egy integrátor kapcsolás végzi (ez a next-hack.com: playing a 16 bit 20ksps audio from the SD card mintakapcsolásának átméretezett változatának is tekinthető)



A nagy átverés

- Az az adatlapból is kiderül, hogy az **STM32F411CE** mikrovezérlő nem tartalmaz beépített digitális-analóg átalakítót (DAC).
- A kézenfekvő lehetőség a beépített **audiopwmio** könyvtár használata. És itt jön **a nagy átverés:** az STM32 mikrovezérlők esetében a **CircuitPython** firmware-ben nem implementáltak valódi PWM audio lejátszást, ehelyett csak egy egyszerű bitbillegtetést írtak bele („1-bites DAC”)
- A következő oldalakon bemutatjuk röviden a hanglejátszó objektumosztályok és függvények használatát, de a fent említett egybites megoldás miatt az STM32 mikrovezérlők esetén ne számítsunk a ZX Spectrumnál, vagy a „picergős kvarcjátékoknál” komolyabb hangminőségre!
- **Megjegyzés:** hanglejátszásban jobban támogatott mikrovezérlők:
  - **audioio** – SAMD51 (2x12 bit DAC), SAMD21 (1x10 bit DAC)
  - **audiopwmio** – RP2040, nRF52840

Egyszerű hangkeltés: pwmaudio_beep.py

```
import time
import array
import math
import board
import digitalio
from audiocore import RawSample
from audiopwmio import PWMAudioOut as AudioOut
button = digitalio.DigitalInOut(board.A0)
button.switch_to_input(pull=digitalio.Pull.UP)

tone_volume = 1  # Increase this to increase the volume of the tone.
frequency = 440 # Set this to the Hz of the tone you want to generate.
length = 44000 // frequency
sine_wave = array.array("H", [0] * length)
for i in range(length):
    sine_wave[i]=int((1+math.sin(math.pi*2*i/length))*tone_volume*(2**15-1))
audio = AudioOut(board.B0)
sine_wave_sample = RawSample(sine_wave, sample_rate=44000)

while True:
    if not button.value:
        audio.play(sine_wave_sample, loop=True)
        time.sleep(1)
        audio.stop()
```

Wav lejátszás: pwmaudio_wavplay2.py

```
import board
import audiocore
import audiopwmio
import digitalio

data = open("dog_bark.wav", "rb")
wav = audiocore.WaveFile(data)
print(wav)
a = audiopwmio.PWMAudioOut(board.B0)

print("Sample rate: ",wav.sample_rate)
print("Bits per sample: ",wav.bits_per_sample)
print("Channel count: ",wav.channel_count)

print("playing")
a.play(wav)
while a.playing:
  pass
print("stopped")
```

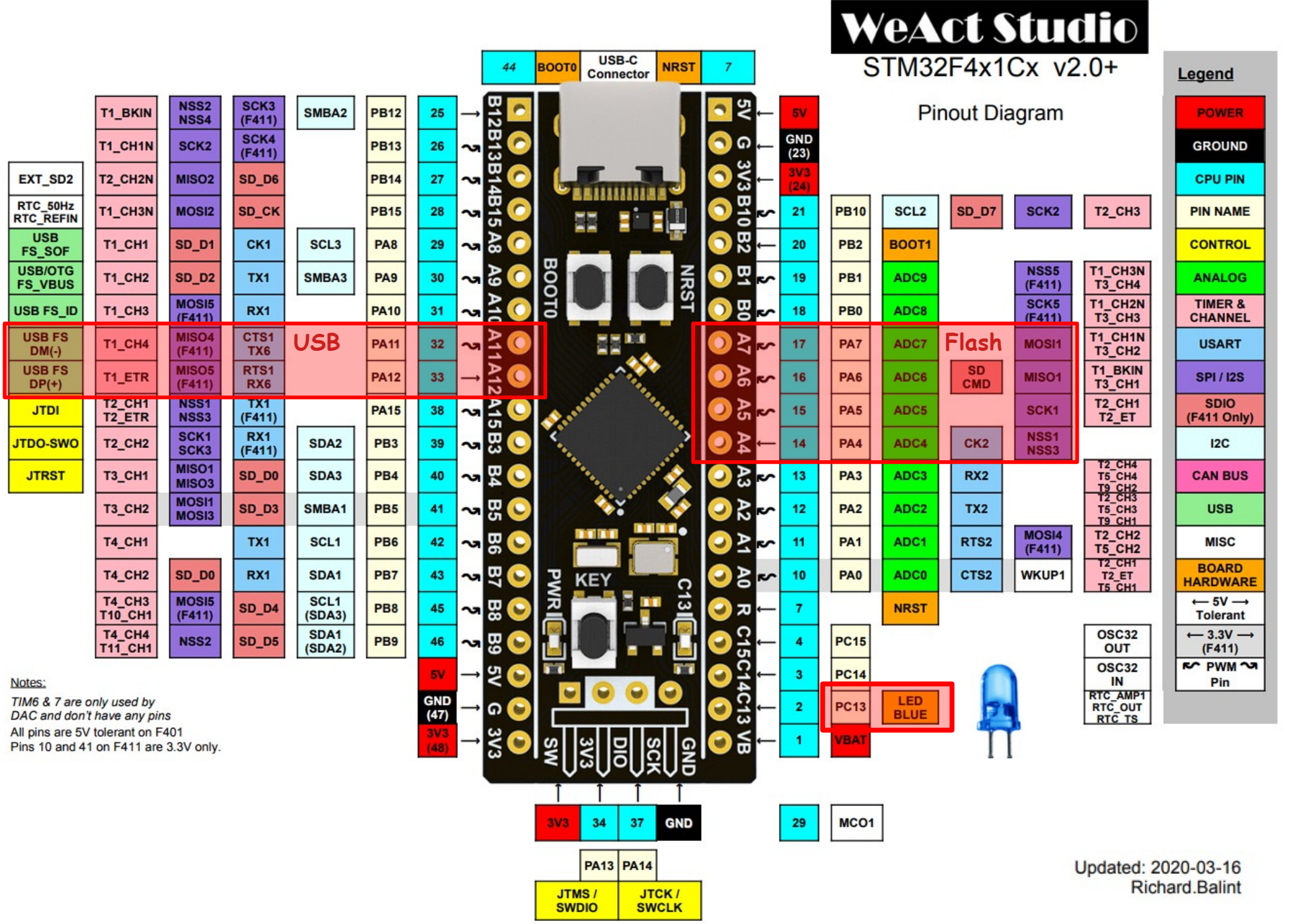
Mp3 lejátszás: pwmaudio_mp3.py

```
import board
import board
import audiocore
import audiopwmio
import audiomp3

data = open("hobbi.mp3", "rb")
mp3 = audiomp3.MP3Decoder(data)
a = audiopwmio.PWMAudioOut(board.B0)
print("Sample rate: ", mp3.sample_rate)
print("Bits per sample: ", mp3.bits_per_sample)
print("Samples decoded: ", mp3.samples_decoded)
print("Channel count: ", mp3.channel_count)



print("playing")
a.play(mp3)
while a.playing:
  pass
print("stopped")
```


WeAct Studio
STM32F4x1Cx v2.0+
Pinout Diagram
Legend
POWER
GROUND
CPU PIN
PIN NAME
CONTROL
ANALOG
TIMER & CHANNEL
USART
SPI / I2S
SDIO (F411 Only)
I2C
CAN BUS
USB
MISC
BOARD HARDWARE
5V Tolerant
3.3V (F411)
PWM Pin
USB
Flash
BOOT0
NRST
PWR
KEY
C13
SW
3V3
DIO
SCK
GND
LED BLUE
JTMS / SWDIO
JTCK / SWCLK
Notes:
TIM6 & 7 are only used by DAC and don't have any pins
All pins are 5V tolerant on F401
Pins 10 and 41 on F411 are 3.3V only.
Updated: 2020-03-16
Richard.Balint