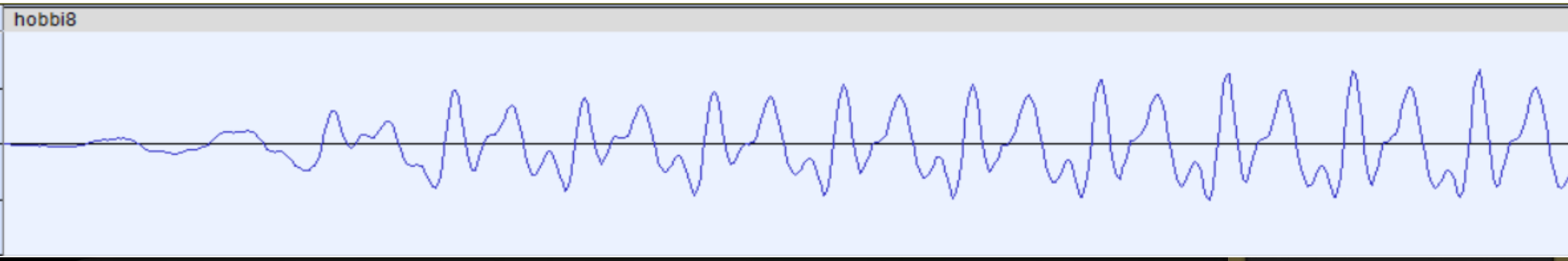
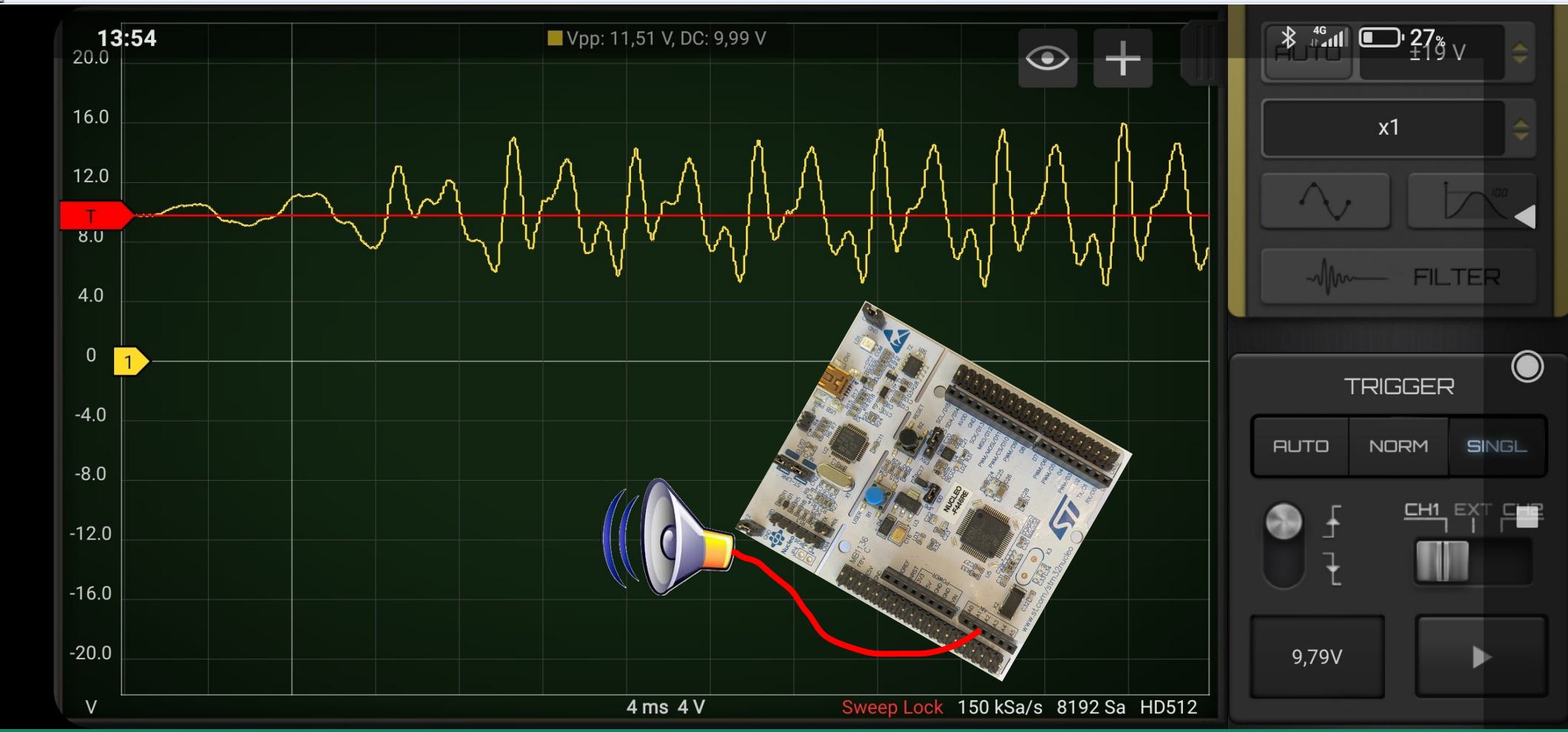
10. Hangkeltés és hanglejátszás Mbed OS alatt



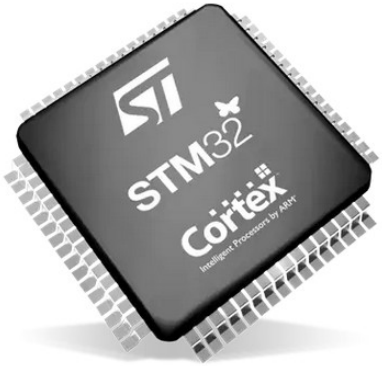


Felhasznált és ajánlott irodalom

- Rob Toulson and Tim Wilmhurst:
  Fast and Effective Embedded Systems Design: Applying the ARM mbed
- Perry Xiao: Designing Embedded Systems and the Internet of Things (IoT) with the ARM mbed
- Cserny István: A FRDM-KL25Z kártya programozása mbed környezetben
- **ARM mbed honlap: https://os.mbed.com/**
- **ARM Keil Studio: https://studio.keil.arm.com/**
- **ARM mbed: Audioplayer library for Mbed**
- **ARM mbed: USB WAV audio player tutorial**
- ARM mbed OS Documentation: https://os.mbed.com/docs/mbed-os/
- Keil Studio manual: https://developer.arm.com/documentation/102 497/latest
- ARM mbed forráskód: https://github.com/ARMmbed/mbed-os

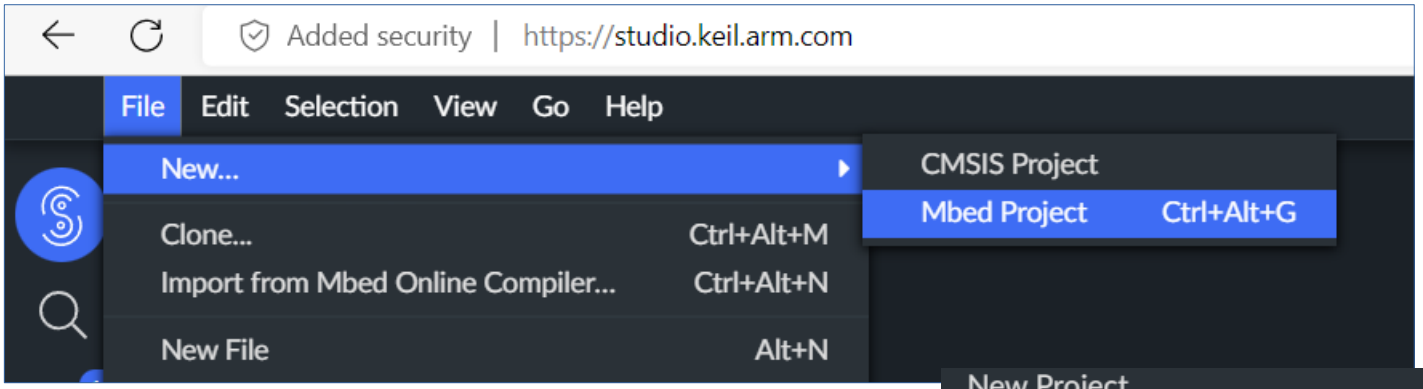
Adatlapok:

- **STM32F446RE** adatlap és termékinfo
- **STM32F446** Family Reference Manual

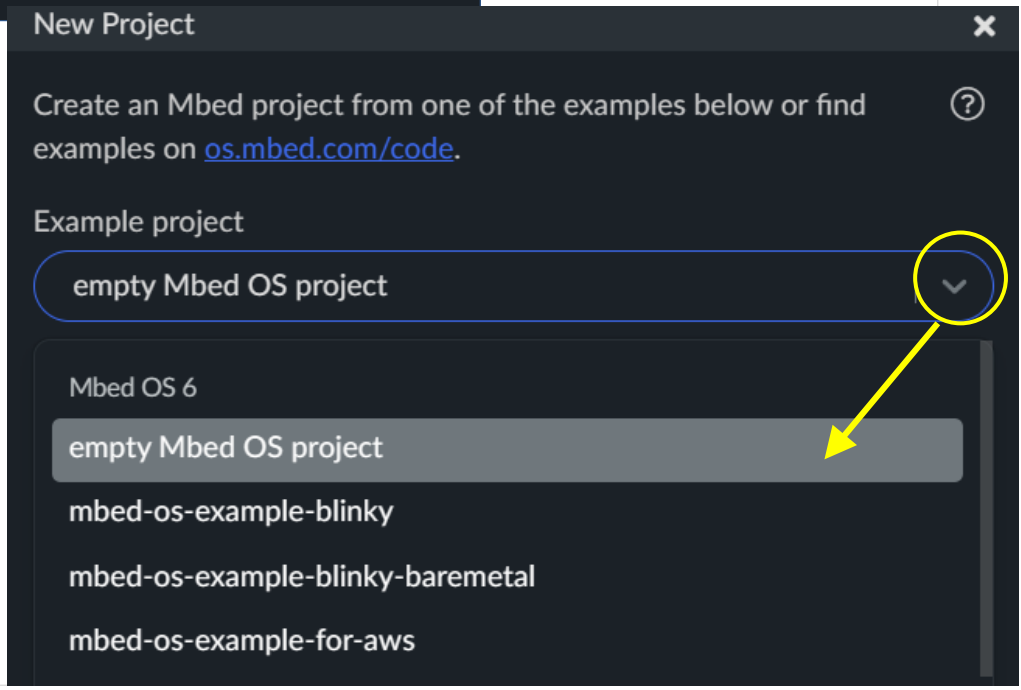


Emlékeztető: Új projekt létrehozásának lépései

- Projektjeinket az online **Keil Studio**-ban hozzuk létre
- A Keil Studio főmenüjében: **File→New→Mbed Project**

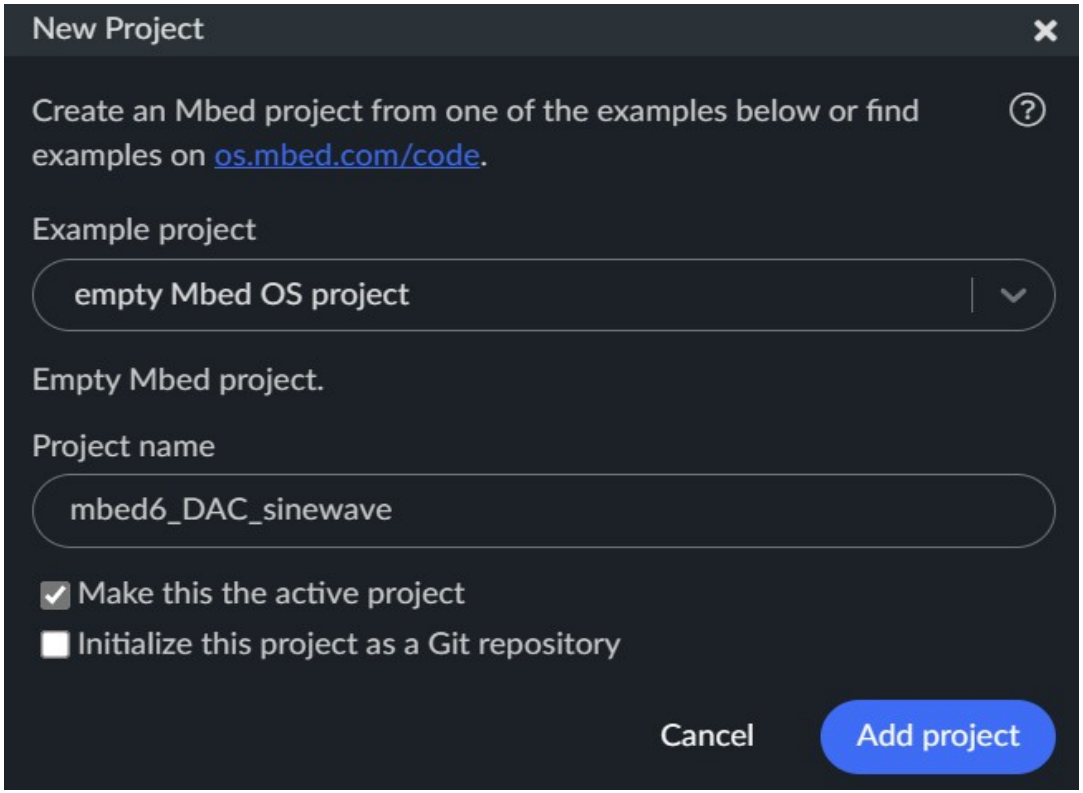


- Válasszuk ki a listából azt a minta-projektet, amit majd átszabunk az új projektünkhöz
- Többnyire az „üres” (empty) projektből célszerű kiindulni



Új projekt létrehozásának lépései

- Írjuk be az új projekt nevét! Pl. **mbed6_DAC_sinewave**
- Állítsuk be, hogy ez legyen az új aktív projekt!
- A **GitHub** verziókövetés elől vegyük ki a pipát!
- Végül kattintsunk az **Add projekt** gombra!



Példaprogramok

mbed6_DAC_sinewave – hangkeltés DAC-kal

mbed6_PWM_sinewave – hangkeltés PWM-mel

mbed6_DAC_wavplayer – DAC wav-lejátszó

mbed6_PWM_wavplayer – PWM wav-lejátszó

A mintaprogramok az https://github.com/cspista oldalon is megtalálhatók

Hangkeltés DAC-kal

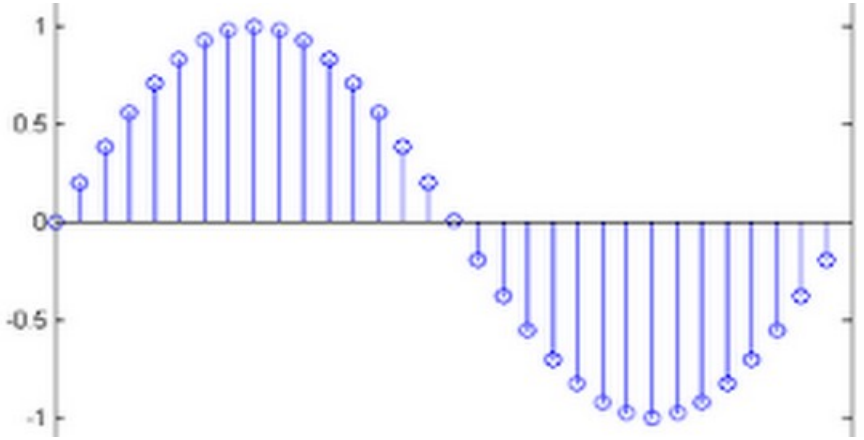
- Az **STM32F446RE** mikrovezérlő két 12-bites DAC kimeneti csatornával rendelkezik, a **PA_4** és **PA_5** kivezetésekre kötve (bővebben lásd a 2020. december 17-i és 2021. január 22-i előadásokban)
- Az **AnalogOut** osztály a **DAC** konfigurálására és kezelésére szolgál. Mivel része a standard **mbed API**-nak, nem kell külön importálni.
- Az **AnalogOut** osztály legfontosabb tagfüggvényei:

Függvény	Használat
AnalogOut név(pin)	Létrehoz egy "név" nevű **AnalogOut** objektumot és a pin paraméterrel megadott kivezetést a DAC analóg kimeneteként konfigurálja. A pin értéke csak **PA_4**, vagy **PA_5** lehet
write(*float* data)	A DAC kimenő feszültségét beállítja data*VREF értékre (data értéke 0.0 és 1.0 közötti szám, VREF = 3.3V)
write_u16(*uint16_t* data)	A DAC kimenő feszültségét beállítja data*VREF/2^{16} értékre (data 0 és 0xffff közötti szám, VREF = 3.3V)

- A mai előadás példáiban csak a **PA_4** kimenetet fogjuk használni (ez az **A2** Arduino kompatibilis kivezetés)

mbed6_DAC_sinewave

- Szinuszhullám keltéséhez először kiszámoljuk és a ***sample*** nevű tömbben eltároljuk egy teljes hullám mintavételezett adatait
- A szinusz függvény –1 és +1 közötti értékeket vehet föl, ezt transzformálni kell a 0 – 65 535 tartományba (*uint16_t*), mert bár a DAC 12 bites, az **AnalogOut write_u16** tagfüggvénye 16 bites adatot vár
- Mivel hullámonként 36 adatot küldünk ki, egy 440 Hz-es hanghullámhoz t = 1/ (440Hz*36) = 63.1313... µs-onként kell kiküldeni a következő adatot
- Az időzítést mi a `wait_us(63)` késleltetéssel oldottuk meg, ami nyilvánvalóan pontatlansághoz vezet, mivel nem veszi figyelembe az utasítások végrehajtási idejét
- Pontosabb időzítéshez timer megszakítást vagy DMA adatátvitelt kellene használni



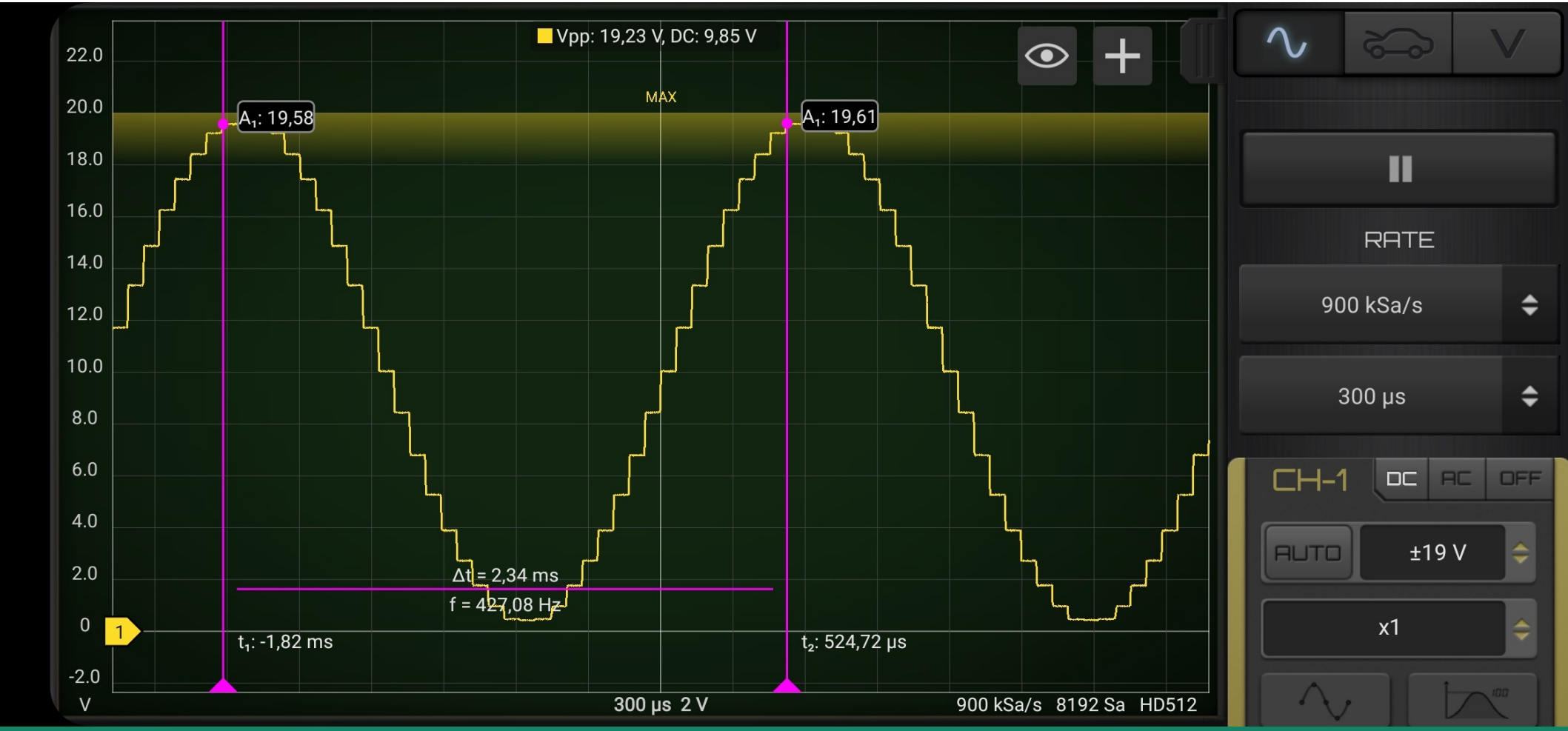
mbed6_DAC_sinewave/main.cpp

- A programban egy 36 pontból álló hullám mintáit küldjük ki 63 µs-onként, ami névleg 440 Hz-es frekvenciát eredményez

```
#include "mbed.h"
const double pi = 3.14159265358979323846;
const double amplitude = 32767;                 // 2^15-1

AnalogOut aout(PA_4);

int main() {
    double rads = 0.0;
    uint16_t sample[36] = {0};
    for (int i = 0; i < 36; i++) {
        rads = (pi * i*10) / 180.0f;
        sample[i] = (uint16_t)(amplitude * (1.0 + sin(rads)));
    }

    while (1) {
        for (int i = 0; i < 36; i++) {
            aout.write_u16(sample[i]);
            wait_us(63);                        // 1.0/(440Hz*36) = 63.13 us
        }
    }
}
```

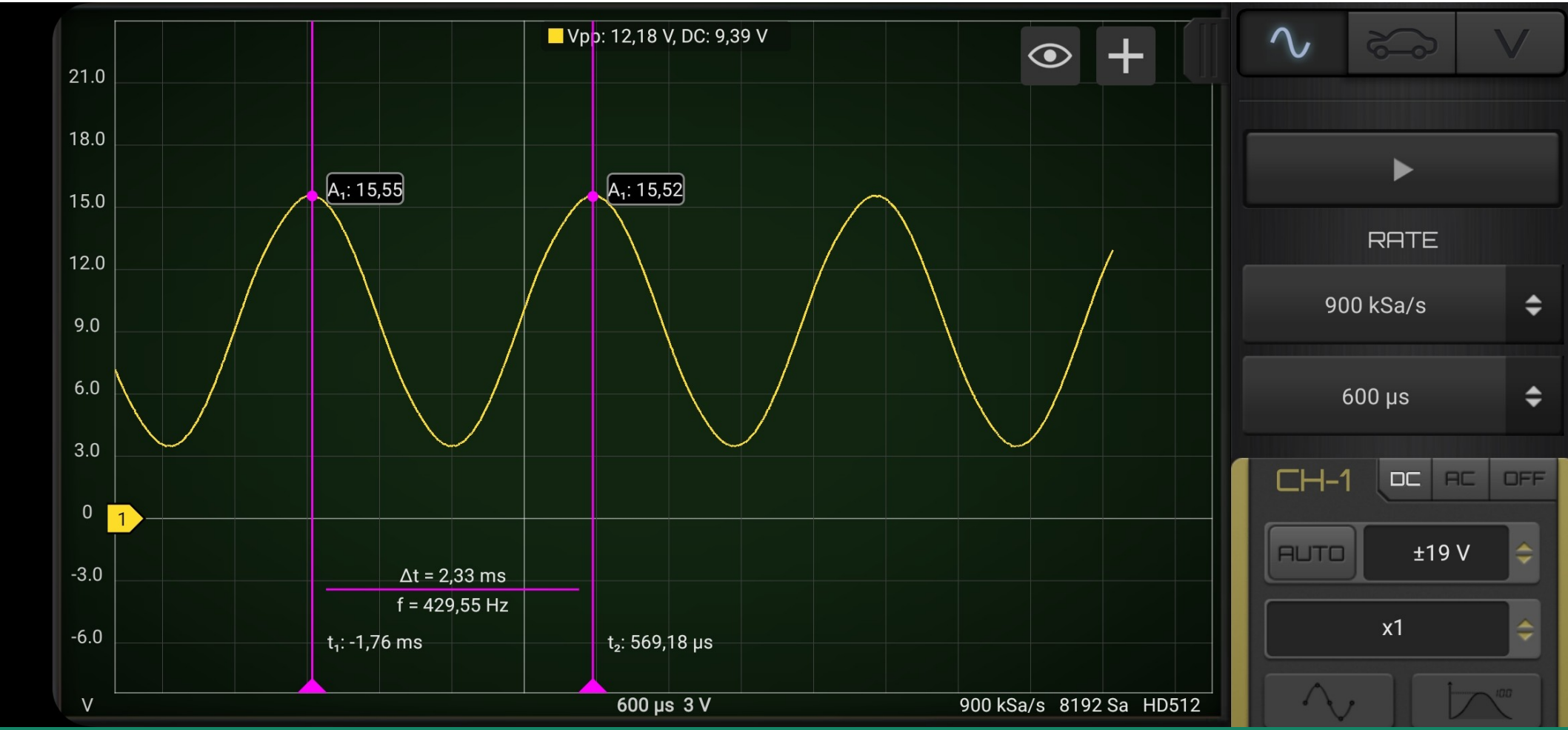
mbed6_DAC_sinewave eredménye

- A **PA_4** kivezetésen kapott jel láthatóan lépcsős
  (a 0–20 skála valójában 0–3,3 V-nak felel meg)
- A frekvencia a várakozásnak megfelelően a névlegesnél valamival kevesebb, kb. 427 Hz



mbed6_DAC_sinewave – jelsimítással

- Egy RC alulászteresztő szűrő (R = 1k, C = 470n) szépen „kisimítja” a jelet, ugyanakkor észrevehetően csökkenti a jel amplitúdóját (Vpp 3,3 V-ról kb. 2 V-ra csökken)



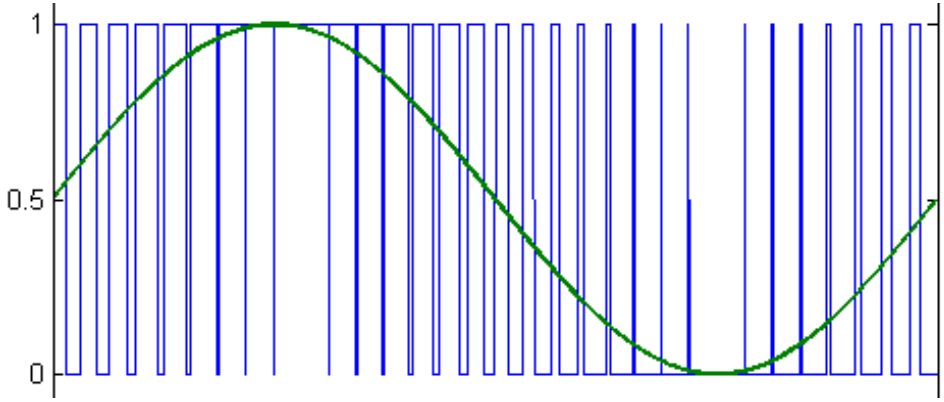
Hangkeltés PWM-mel

- Az **STM32F446RE** mikrovezérlő PWM kimenetekkel is rendelkezik (bővebben lásd a 2021. október 28-i előadásban)
- A **PwmOut** osztály a PWM csatornák konfigurálására és kezelésére szolgál. Mivel része a standard **mbed API**-nak, nem kell külön importálni. A **PwmOut** osztály legfontosabb tagfüggvényei:

Függvény	Használat
PwmOut név(pin)	Létrehoz egy "név" nevű **PwmOut** objektumot és a pin paraméterrel megadott kivezetést PWM kimeneteként konfigurálja (pin csak PWM képes kimenet lehet)
write(*float* data)	A PWM kitöltési tényezőjét beállítja data*100% értékre (data értéke 0.0 és 1.0 közötti szám lehet)
period(*float* data) **period_ms**(*int* data) **period_us**(*int* data)	A PWM periódusidejét állítja be *data* értéke szerint
pulsewidth(*float* data) **pulsewidth_ms**(*int* data) **pulsewidth_us**(*int* data)	Beállítja a PWM impulzus szélességét (a magas szintű állapot idejét) *data* értéke szerint. A periódus idejét nem változtatja meg
suspend() / **resume**()	Leállítja, illetve újraindítja a PWM csatorna működését

mbed6_PWM_sinewave

- Szinuszhullám keltéséhez ugyanúgy kiszámoljuk és egy tömbben eltároljuk egy teljes hullám mintavételezett adatait, mint a DAC-nál.
- A szinusz függvény –1 és +1 közötti értékeket vehet föl, ezt most a 0 – 1.0 tartományba (*float*) transzformáljuk, mert a **PwmOut write()** tagfüggvénye ilyen adatot vár a kitöltés megadásához
- Mivel hullámonként 36 adatot küldünk ki, egy 440 Hz-es hanghullámhoz t = 1/ (440Hz*36) = 63.1313... µs-onként kell kiküldeni a következő adatot
- Az időzítést most is a `wait_us(63)` késleltetéssel oldottuk meg, ami nyilvánvalóan pontatlan, mivel nem veszi figyelembe az utasítások végrehajtási idejét



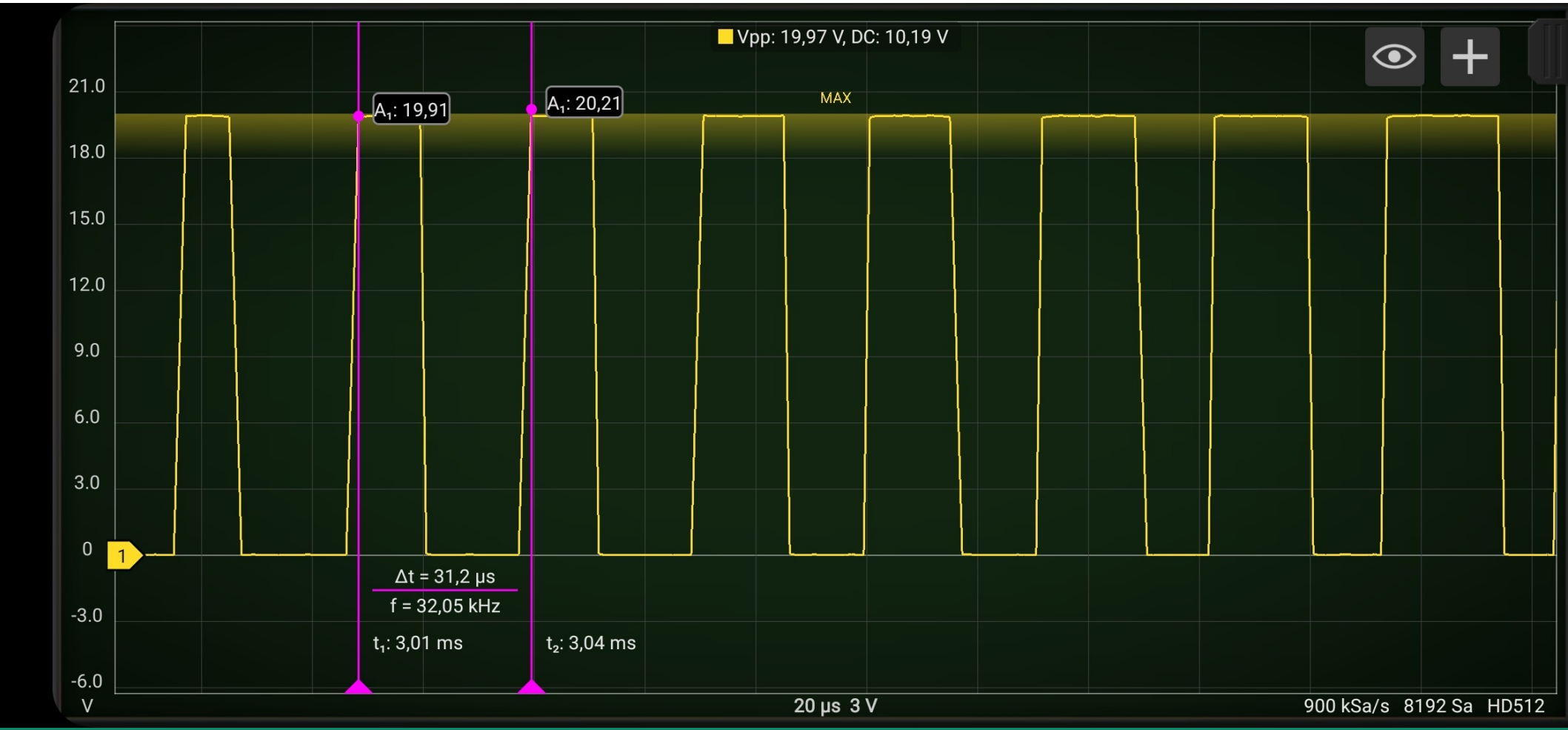
- A PWM periódust a fenti időtartam felére állítjuk be (`0,0000315 s`), így minden beírási ciklusra két PWM periódus esik, így garantáltan minden beírás érvényre jut legalább egyszer

mbed6_PWM_sinewave/main.cpp

```
#include "mbed.h"
const double pi = 3.14159265358979323846;
const double amplitude = 0.5f;

PwmOut aout(PB_0);                          // A3 Arduino kompatibilis kivezetés

int main() {
    double rads = 0.0;
    float sample[36] = {0};

    // sinewave output
    for (int i = 0; i < 36; i++) {
        rads = (pi * i*10) / 180.0f;
        sample[i] = (amplitude * (1.0 + sin(rads)));
    }
    aout.period(0.0000315);                 // 1.0/(2*440Hz*36) = 31.5 us
    while (1) {
        for (int i = 0; i < 36; i++) {
            aout.write(sample[i]);
            wait_us(63);                    // 1.0/(440Hz*36) = 63.13 us
        }
    }
}
```

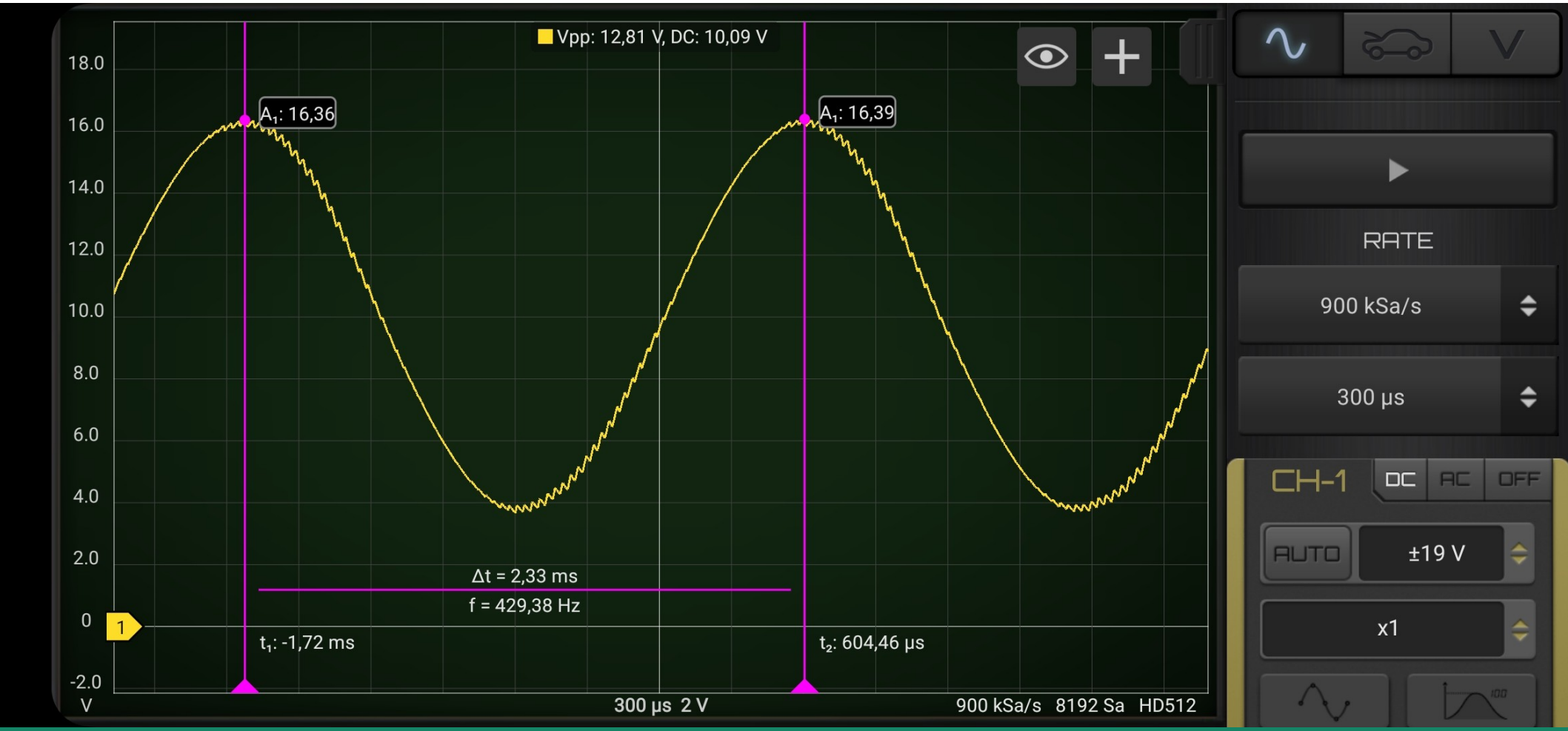
mbed6_PWM_sinewave – a nyers kimenő jel

- A **PB_0** kivezetésen kapott jelben szemmel nem ismerhető fel a szinusz, mivel az a változó szélességű impulzusokba van kódolva
- A PWM periódus a névlegesnek megfelelő (31.2 µs vs. 31.5 µs), az eltérés kisebb, mint a mintavételezés és a leolvasás okozta bizonytalanság



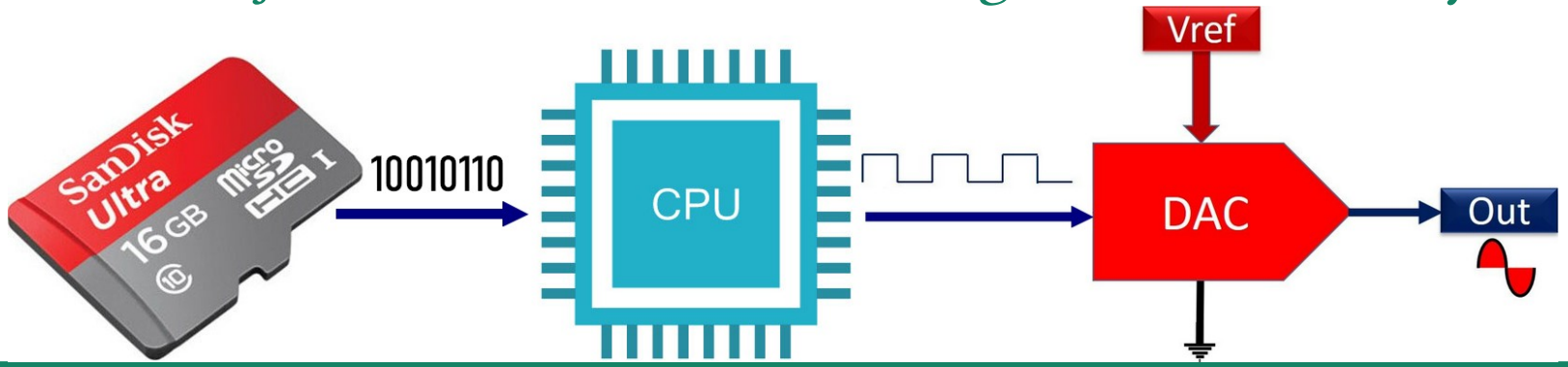
mbed6_PWM_sinewave – a „simított” jel

- Az RC aluláteresztő szűrő (R = 1k, C = 470n) itt is „kisimítja” a jelet, bár marad némi „érdesség”, és lecsökkenti a jel amplitúdóját (Vpp 3,3 V-ról kb. 2,1 V-ra csökken)



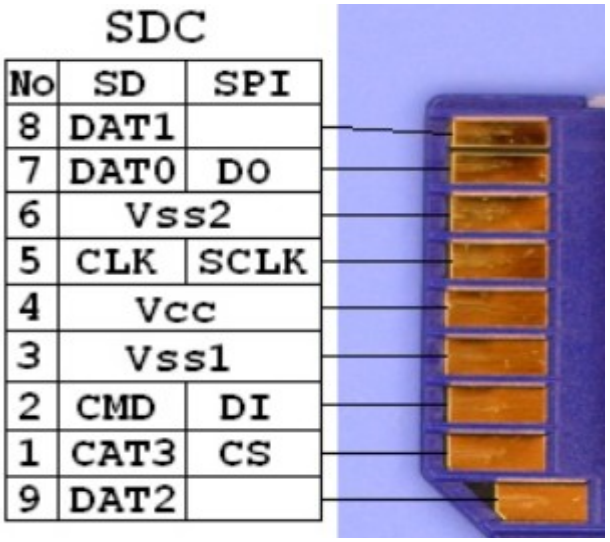
Wav fájlok lejátszása

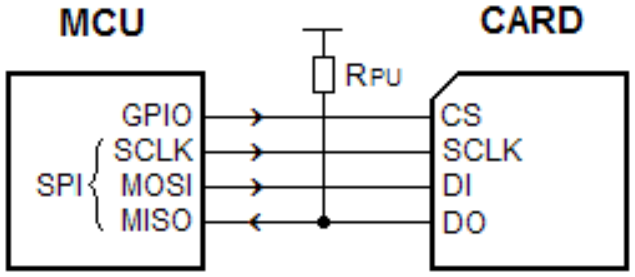
- Először fel kell elevenítenünk a 2022. szeptember 22-i előadásban elhangzottakat a fájlrendszerekkel és az SD kártya kezelésével kapcsolatban, mert SD kártyáról fogjuk betölteni a ***.wav*** fájlt
- A korábbiaktól eltérően hogy most a PC- kompatibilitás érdekében a **LittleFileSystem2** helyett a **FATFilesytem** fájlrendszert használjuk (PC-n írtuk kártyára a **hobbi8.wav** fájlt)
- A ***.wav*** fájlok értelmezéséhez telepítenünk kell az **Audioplayer** nevű könyvtári modult (ebben van a **WaveAudioStream** osztály)
- A lejátszáshoz a hangkeltést nem bízzuk az **Audioplayer**-re, hanem saját programunkban szervezzük meg, így össze-hasonlíthatjuk a **DAC** és a **PWM** hangkeltést eredményét



SD kártya használata SPI módban

- A **Secure Digital Memory Card** egy *de facto* szabvány a hordozható eszközök memóriakártyái közül. A kártya meghajtható **SPI** módban, illetve 1, vagy 4 adatvonalas **SDIO** módban az adatok 512 bájtos blokkokba vannak szervezve
- Az **STM32F446RE** mikrovezérlő rendelkezik **SDIO** perifériával is, de most az **SPI** módot és az **SPI3** csatornát fogjuk használni



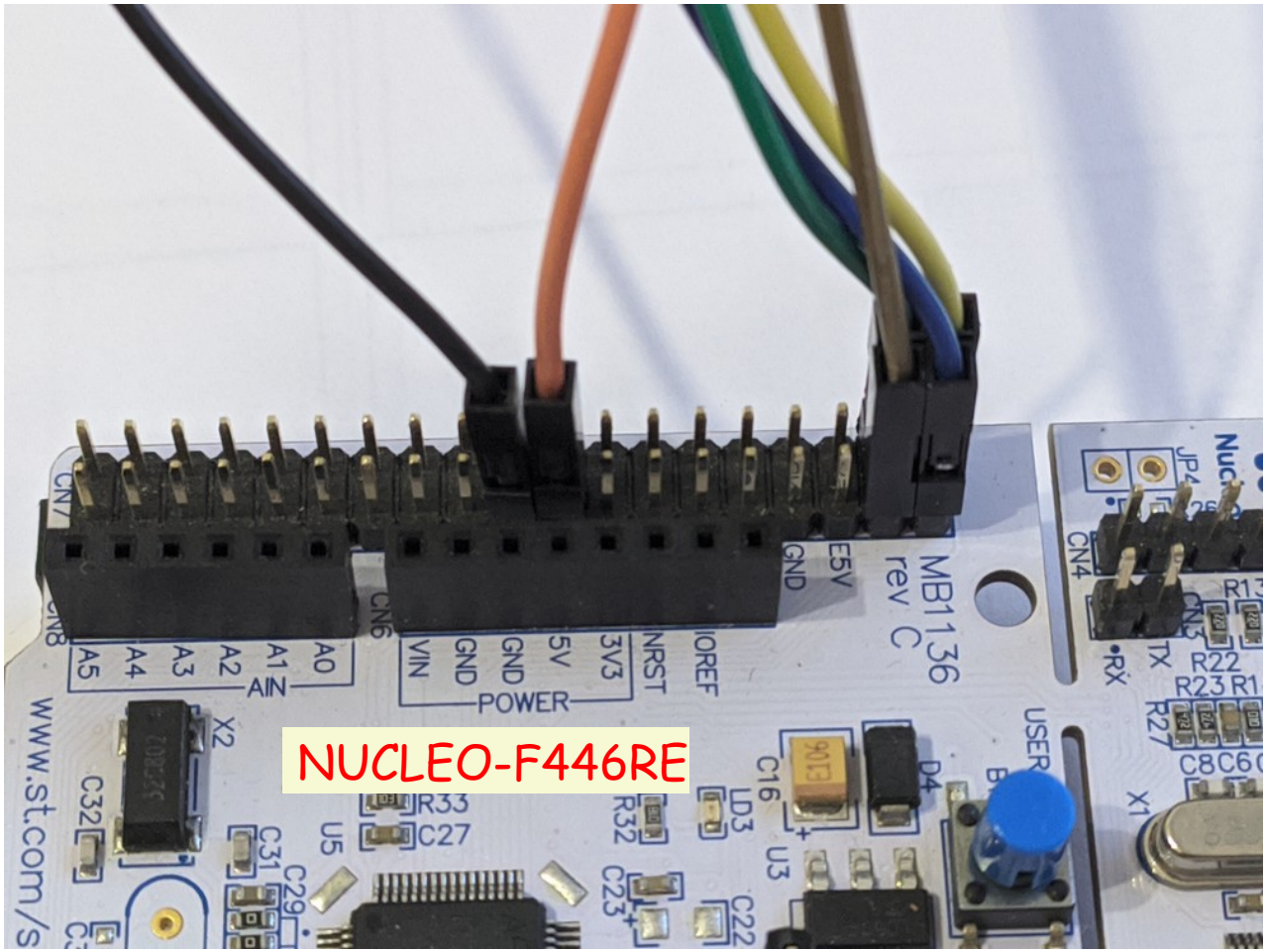


NUCLEO-F446RE kártya

SD card	SPI3	Pin
DO	MISO	PC_11
SCLK	SCLK	PC_10
DI	MOSI	PC_12
CS	CS	PD_2
VCC	+5V/3,3V	5V/3.3V
GND	GND	GND

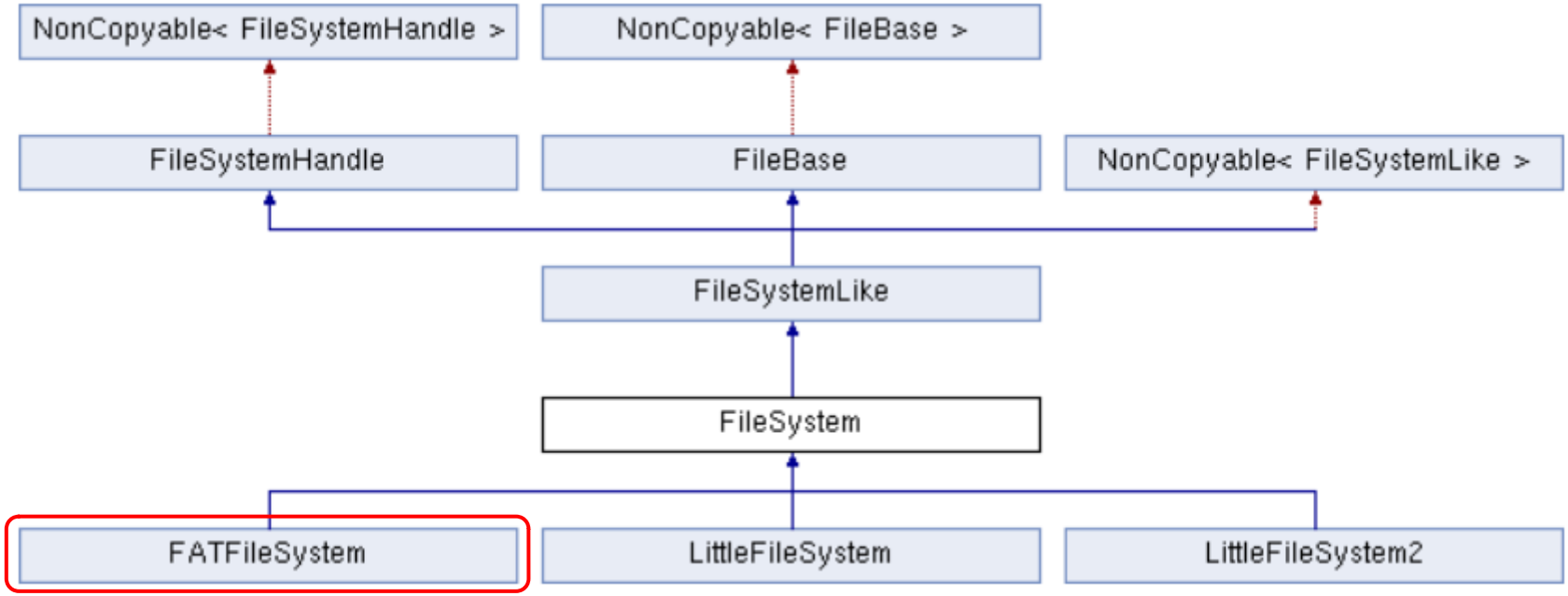
Az SD kártya bekötése

- Ügyeljünk az SD kártya modul tápfeszültségére! Esetünkben 5V, mivel saját stabilizátora van a kártyának, de általában 3.3V tápfeszültség kell



Fájlrendszer használata

- A **FATFilesytem**, vagy valamelyik **LittleFileSystem** választható (mi most a **FATFilesytem**-et fogjuk használni)
- A Windows/Linux/MacOS rendszerekkel csak a **FATFilesytem** kompatibilis, ám ennek erőforrásigénye valamivel nagyobb
- Dokumentáció: **Mbed OS File system APIs**



mbed6_DAC_wavplayer/main.cpp – 2/1.

- A **hobbi8.wav** állományt fogjuk lejátszani, ami egycsatornás (mono), előjel nélküli 8 bites adatokat tartalmaz, 8000 minta/s rátával
- A hangkimenet a **PA_4 (A2)** kivezetésen elérhető **DAC** csatorna

```
#include "mbed.h"
#include "SDBlockDevice.h"
#include "FATFileSystem.h"
#include "AudioPlayer.h"
#include "WaveAudioStream.h"

SDBlockDevice sd(MBED_CONF_SD_SPI_MOSI, MBED_CONF_SD_SPI_MISO,
                 MBED_CONF_SD_SPI_CLK, MBED_CONF_SD_SPI_CS);
FATFileSystem fs("sd", &sd);
AnalogOut aout(PA_4);

int main() {
    sd.frequency(5000000);        // Set SPI frequency
    uint8_t buffer[1500];         // Data buffer
    int num_bytes_read;
    File file;
    if (file.open(&fs, "hobbi8.wav") != 0) {
        error("Could not open 'hobbi8.wav'\r\n");
    }
```

mbed6_DAC_wavplayer/main.cpp – 2/2.

```
    WaveAudioStream song(&file);       //"song" is the audio data object
    if(song.get_valid() == 0){
        error("ERROR: not valid WAV file\r\n");
    }
    if(song.get_bytes_per_sample() != 1){
        error("ERROR: WAV file not 1 bytes per sample (8-bit)\r\n");
    }
    printf("Playing Audio 'hobbi8.wav'\r\n");

    while(1) {
        num_bytes_read = song.read(buffer, 1500); // read next data block
        if(num_bytes_read == -1){
            printf("Song Over\r\n");
            break;
        }
        for (int i = 0; i < num_bytes_read; i++) {
            // printf("%d\r\n",buffer[i]<<8);
            aout.write_u16(buffer[i]<<8);         // write data to DAC
            wait_us(125);
        }
    }
    song.close();                                 // Close the audio object
}
```

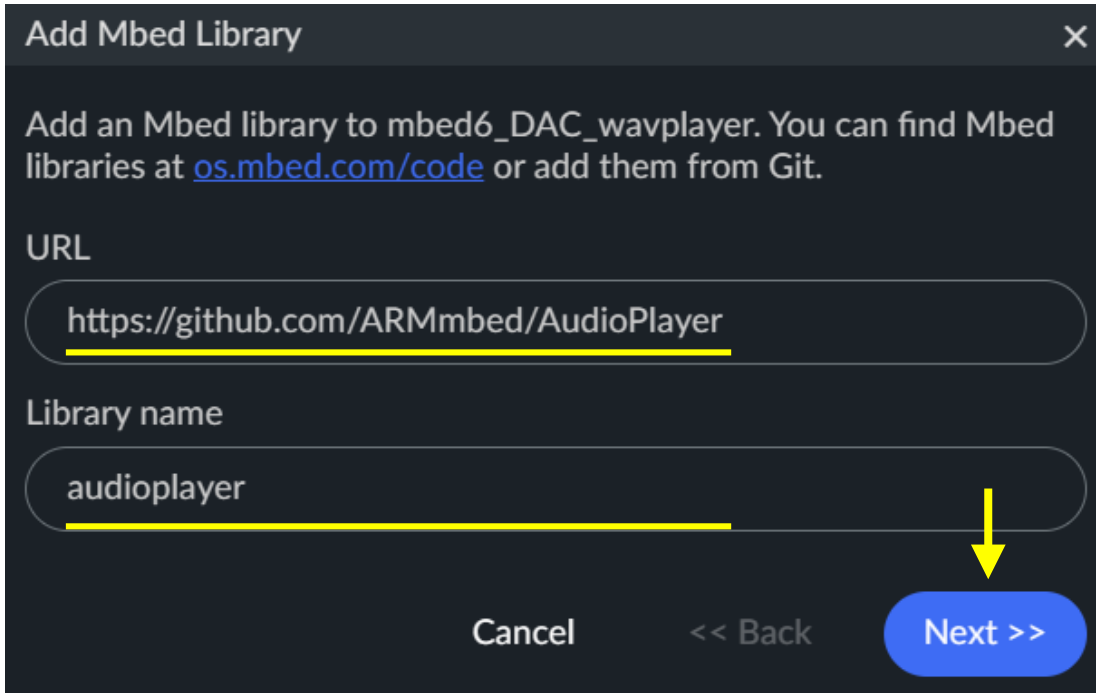
mbed6_DAC_wavplayer/mbed_app.json

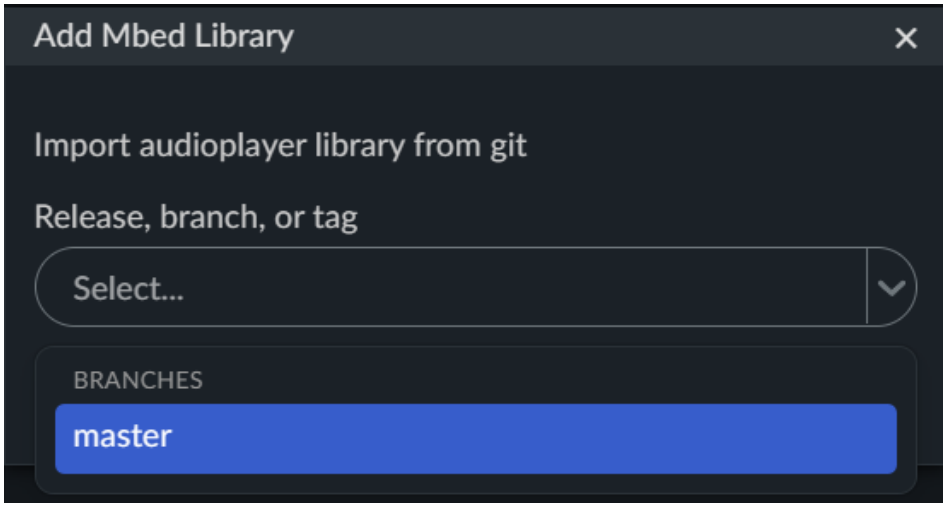
- Az Mbed OS **Data storage API** kategóriájában találjuk a fájlrendszereket és az ún. blokkos tárolóeszközök osztályait, utóbbiak közül most az **SDBlockDevice** objektumosztályt fogjuk használni, ami **SPI** módban kommunikál az SD kártyával
- A **NUCLEO-F446RE** kártyán az **SPI3** csatornát használjuk az SD kártyához, ennek konfigurálásához az új projektben hozzuk létre az **mbed_app.json** fájlt, amelybe ezt írjuk bele:

```
{
  "target_overrides": {
    "*": {
      "target.features_add": ["STORAGE"],
      "target.components_add": ["SD"],
      "sd.SPI_MOSI" : "PC_12",
      "sd.SPI_MISO" : "PC_11",
      "sd.SPI_CLK"  : "PC_10",
      "sd.SPI_CS"   : "PD_2"
    }
  }
}
```

Az Audioplayer könyvtár importálása

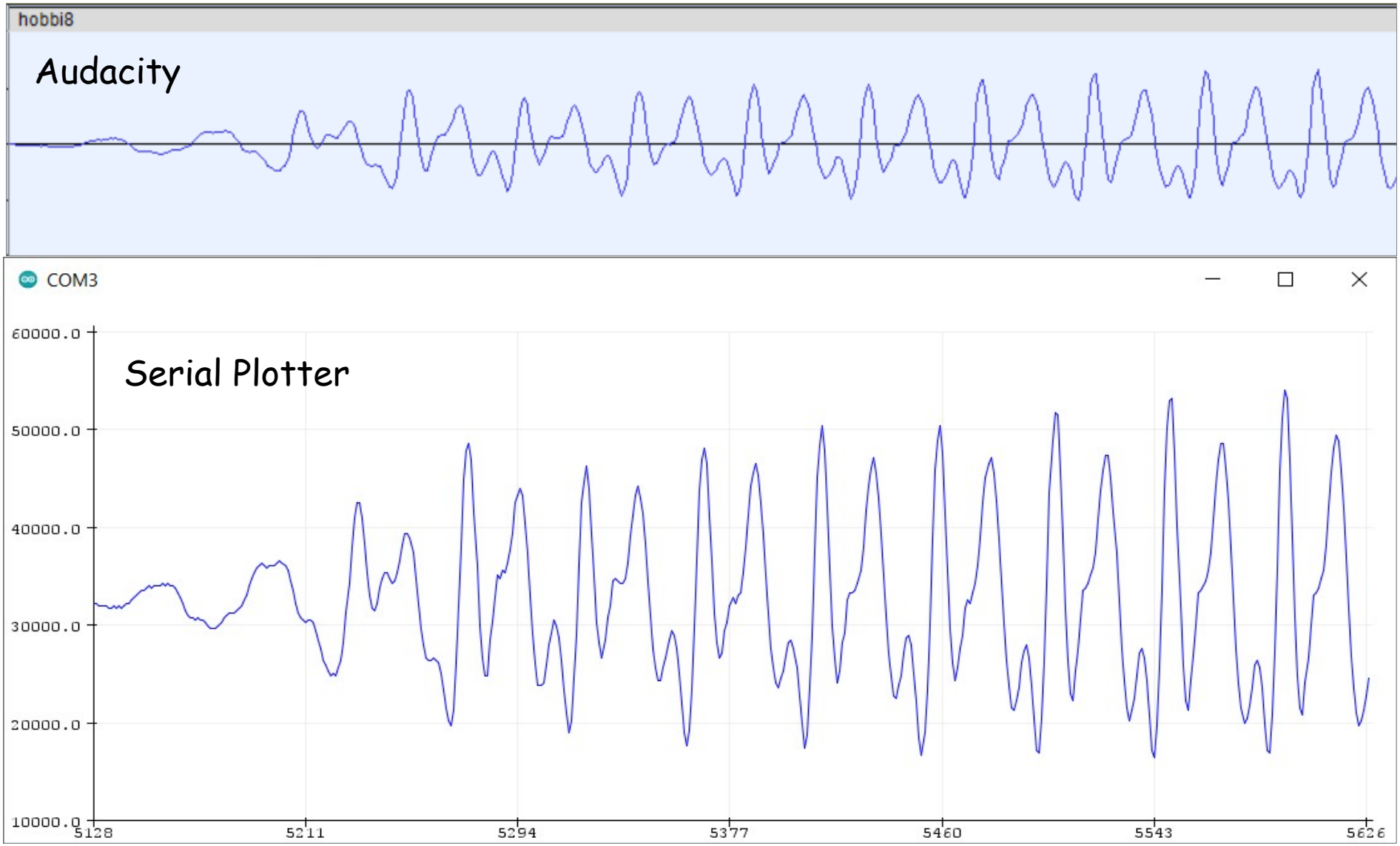
- **File → Add Mbed library to active project** a kiválasztott programkönyvtárat hozzáadja az éppen aktív projekthez
- Az URL rovatba ezt írjuk (https://github.com/ARMmbed/mbed-os-audioplayer)
- A **Library Name** rovatba a programkönyvtár mappájának új nevét kell megadni, majd **Next** után a branch (ág kiválasztása következik, (többnyire ez a *default*, vagy *master*), majd kattintsunk a **Finish** gombra!
- Tutorial: Create, import or clone an Mbed project



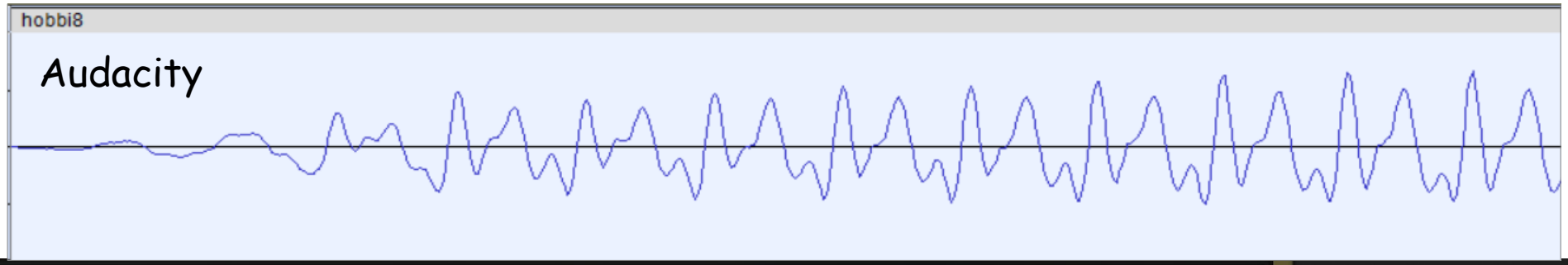


mbed6_DAC_wavplayer eredmények

- Először a ***.wav*** fájlból kiolvasott adatokat ellenőriztük



mbed6_DAC_wavplayer eredmények



Wav fájlok lejátszása PWM kimeneten

- **DAC** helyett a ***.wav*** fájlokat **PWM** kimeneten is lejátszhatjuk
- Az SD kártyán a fájl megnyitása és a ***.wav*** fájl beolvasása ugyanúgy történik, mint az előző programban
- Az audioeszköz kezelésénél ugyanúgy járunk el, mint ahogyan a szinuszhullám **PWM** lejátszásánál, tehát egy **PWM** csatornát konfigurálunk, úgy, hogy **PWM** frekvencia a hangminta mintavételezési frekvenciájának legalább kétszerese legyen, s a hangminta jelével arányos kitöltési tényezőt állítunk be
- Esetünkben 8-bites, 8 kHz-el mintavételezett hangot fogunk lejátszani, ezért a **PWM** frekvencia 16 kHz lesz, a lejátszáshoz használt késleltetési idő pedig 125 µs lesz (azaz 1/8 kHz)
- A 0–255 közé eső hangmintákat itt most 0–1.0 közé skálázzuk és a **PwmOut** osztály **write()** tagfüggvényével írjuk ki
- A projekt létrehozása után az **Audioplayer** könyvtár importálását és az **mbed_app.json** fájl létrehozását az előzőhöz hasonlóan végezzük

mbed6_PWM_wavplayer/main.cpp – 2/1.

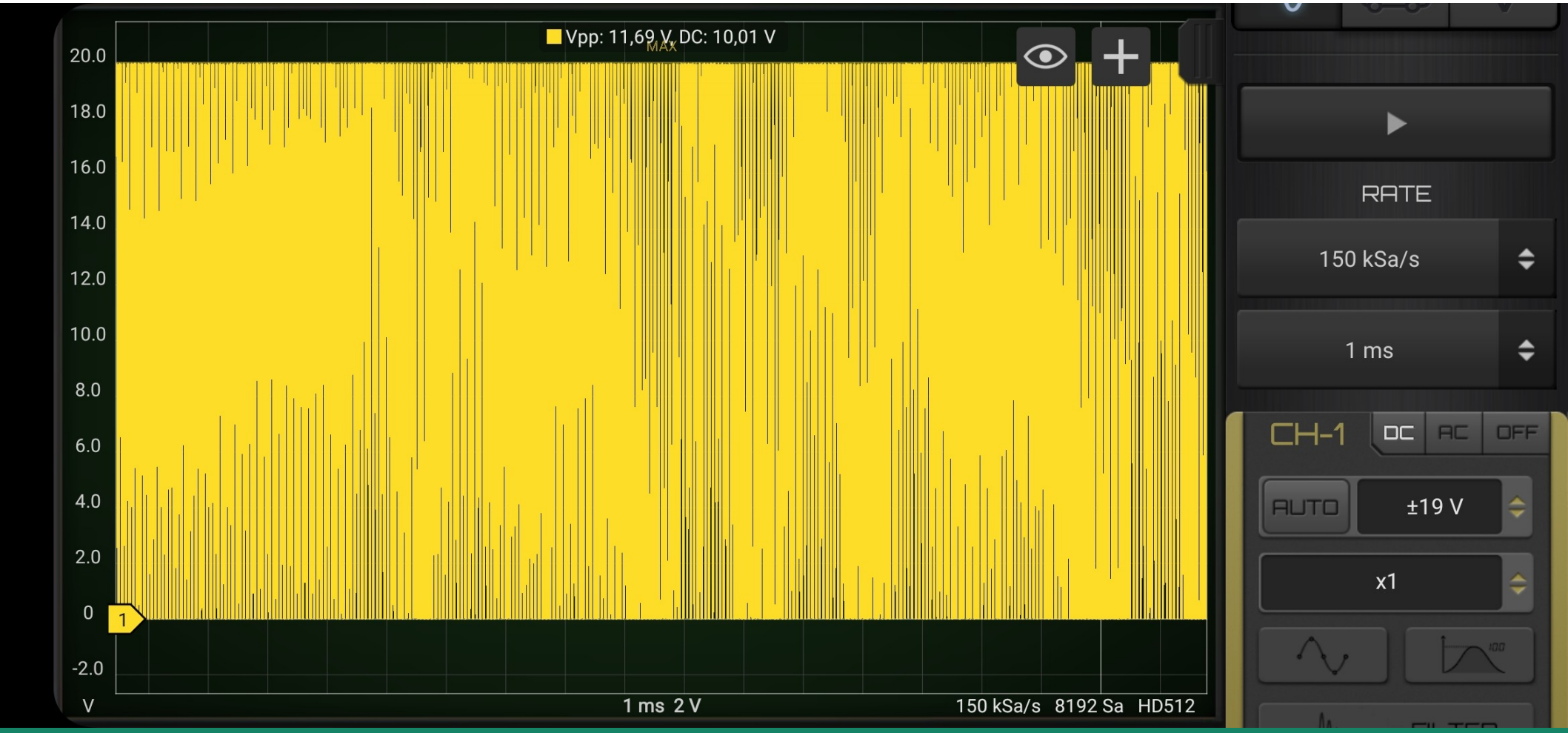
```
#include "mbed.h"
#include "SDBlockDevice.h"
#include "FATFileSystem.h"
#include "AudioPlayer.h"
#include "WaveAudioStream.h"

SDBlockDevice sd(MBED_CONF_SD_SPI_MOSI, MBED_CONF_SD_SPI_MISO,
                 MBED_CONF_SD_SPI_CLK, MBED_CONF_SD_SPI_CS);
FATFileSystem fs("sd", &sd);

PwmOut aout(PB_0);

int main() {
    sd.frequency(5000000);
    File file;
    if (file.open(&fs, "hobbi8.wav") != 0) {
        error("Could not open 'hobbi8.wav'\r\n");
    }
    WaveAudioStream song(&file);   // "song" is the audio data object
    // Check to see if file is a valid WAV file
    if(song.get_valid() == 0){
        error("ERROR: not valid WAV file\r\n");
    }
```

mbed6_PWM_wavplayer/main.cpp – 2/2.

```
    if(song.get_bytes_per_sample() != 1){       // WAV file must be 8-bit
        error("ERROR: WAV file not 1 bytes per sample (8-bit)\r\n");
    }
    uint8_t buffer[1500];
    int num_bytes_read;
    aout.period(1.0/16000.0);
    aout.write(0.5);
    printf("Playing Audio 'hobbi8.wav'\r\n");
    while(1){
        num_bytes_read = song.read(buffer, 1500);
        if(num_bytes_read == -1){
            aout.suspend();
            printf("Song Over\r\n");
            break;
        }
        for (int i = 0; i < num_bytes_read; i++) {
            // printf("%d\r\n",buffer[i]<<8);
            aout.write(buffer[i]/256.0);
            wait_us(125);
        }
    }
    song.close();//Close the WAV file
}
```

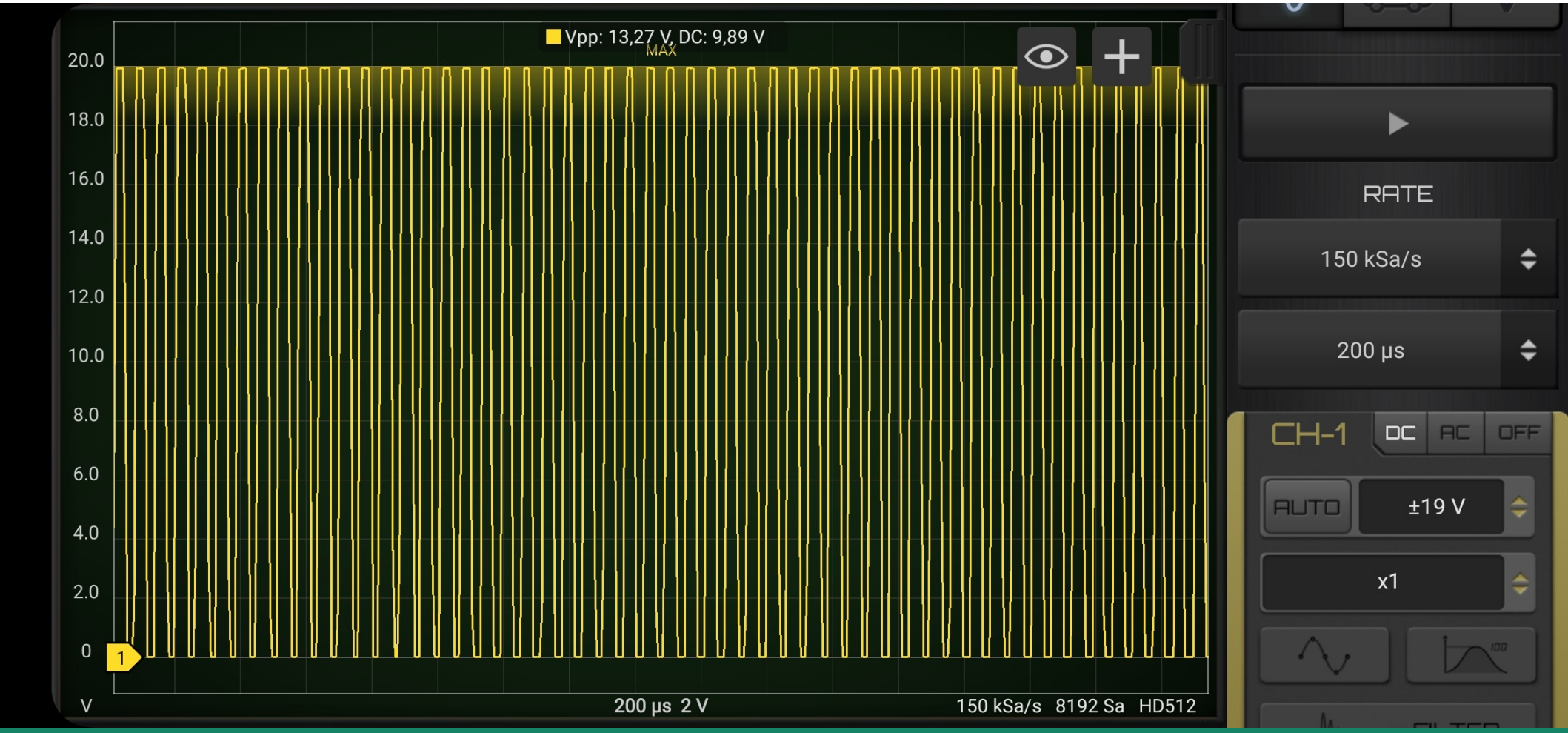
mbed6_PWM_wavplayer futási eredménye

- „Madártávlatból” nézve vehető ki legjobban a lejátszani kívánt jel (ahol a legnagyobb a sűrűsödés)



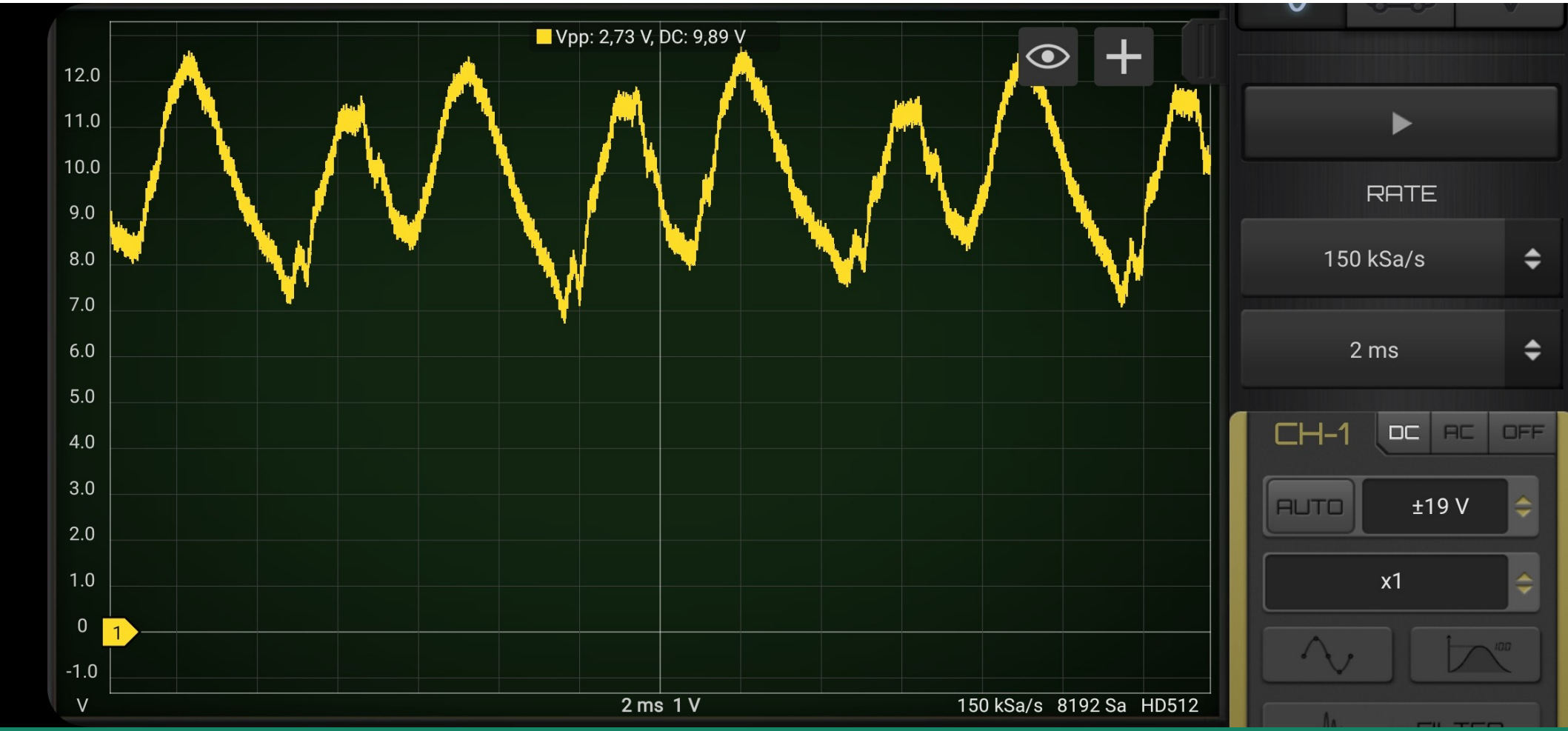
mbed6_PWM_wavplayer futási eredménye

- Ötszörös skálanyújtás után már csak szélesebb-keskenyebb négyszögjeleket látunk



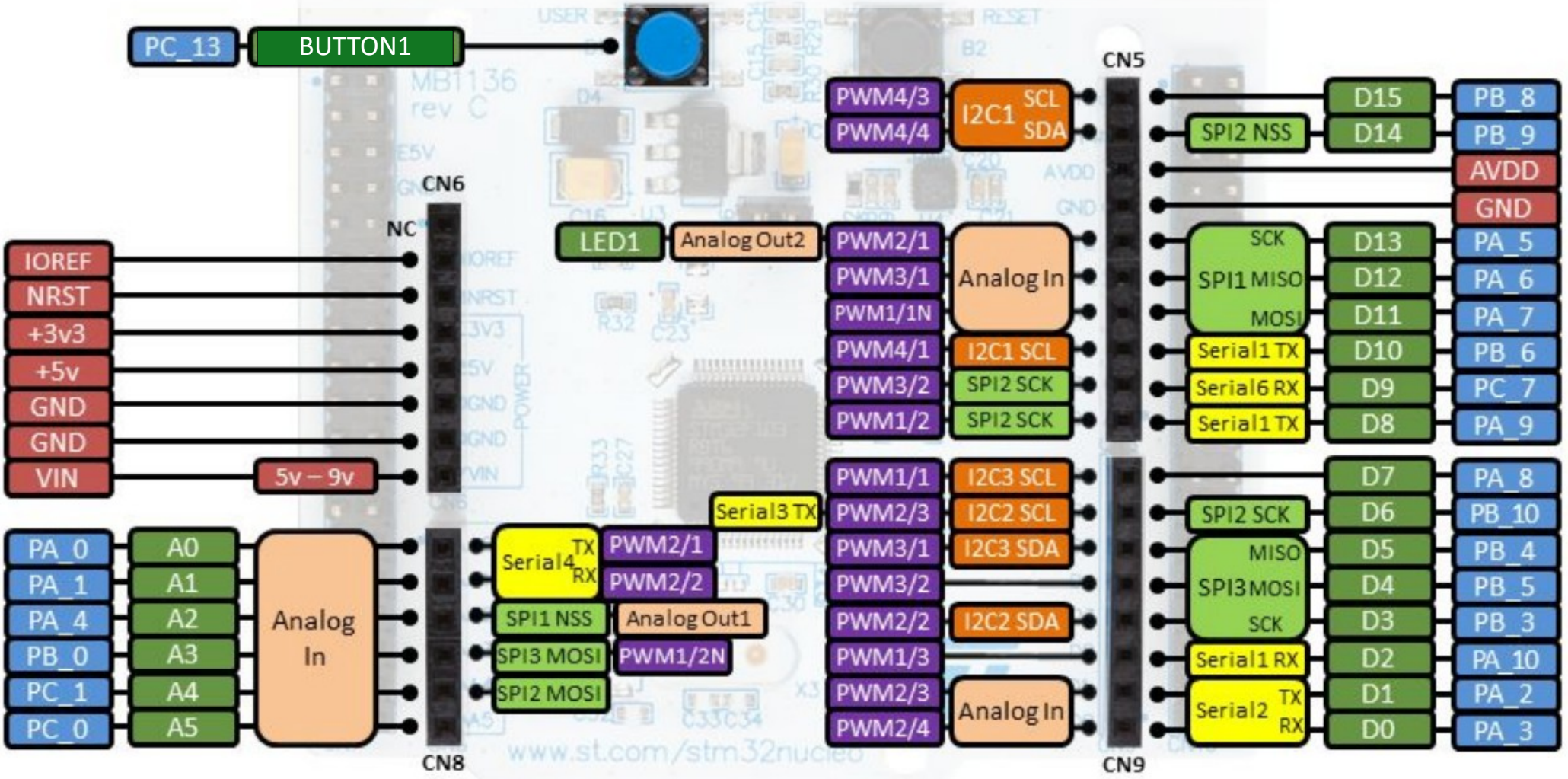
mbed6_PWM_wavplayer futási eredménye

- A szokásos RC aluláteresztő szűréssel láthatóvá válik a rekonstruált jel és többé-kevésbé eltűnik a „vivő” jel



Arduino kompatibilis kivezetések

- Kivezetés azonosításra a kék, illetve a sötétzöld címkék használhatók (pl. PA_5, D13, vagy LED1)



Morpho kivezetések

Nucleo F446RE

Morpho Headers