2. Mbed OS és a felhőalapú Keil Studio

Keil Studio

Added security | https://studio.keil.arm.com

File Edit Selection View Go Help

Active project

Lab04_button_interrupt

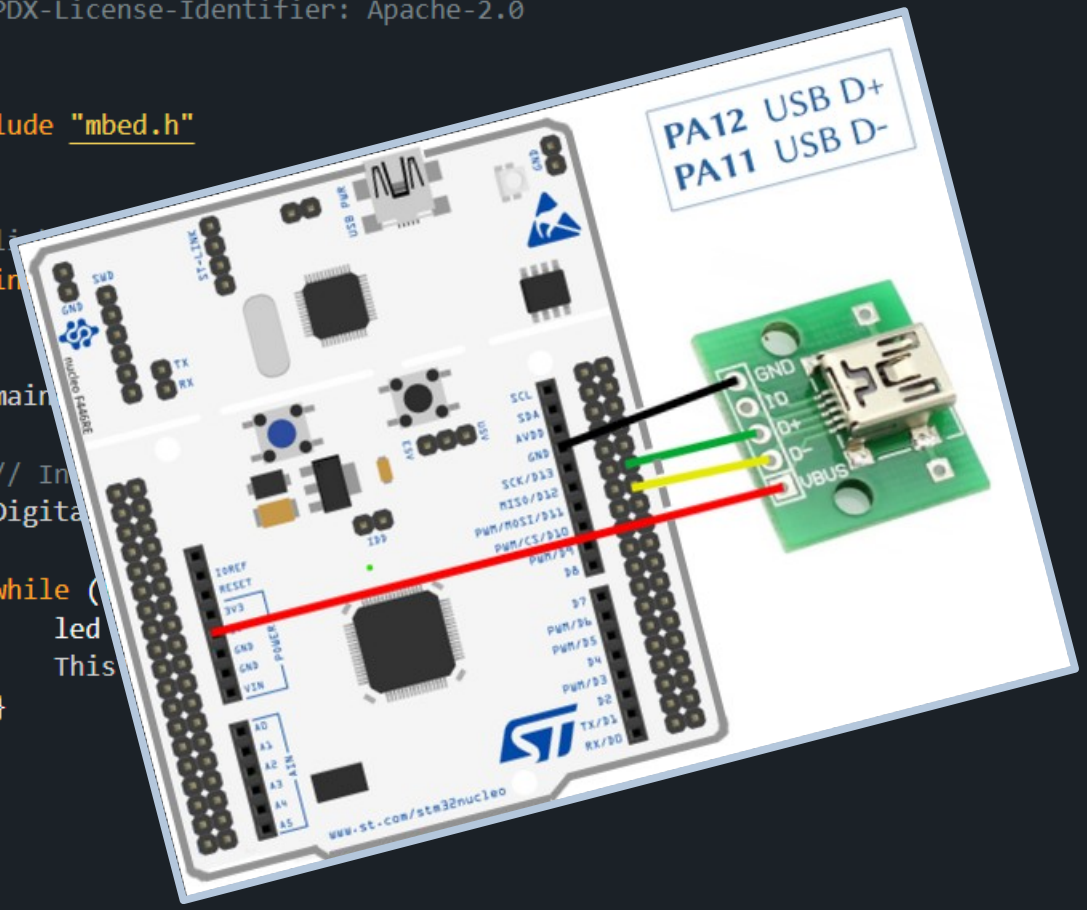
Target hardware

NUCLEO-F446RE

- Lab04_button_interrupt
  - mbed-os-example-blinky
    - .circleci
    - .github
    - BUILD
    - mbed-os
    - resources
    - .gitignore
    - .mbed
    - .travis.yml
    - CMakeLists.txt
    - compile_commands.json
    - CONTRIBUTING.md
    - LICENSE
    - main.cpp
    - mbed-os.lib

main.cpp mbed6_queue | main.cpp mbed-os-example-blinky

```
1  /* mbed Microcontroller Library
2   * Copyright (c) 2019 ARM Limited
3   * SPDX-License-Identifier: Apache-2.0
4   */
5
6  #include "mbed.h"
7
8
9  // Bli
10 #defin
11
12
13 int main
14 {
15     // In
16     Digita
17
18     while (
19         led
20         This
21     }
22 }
23
```

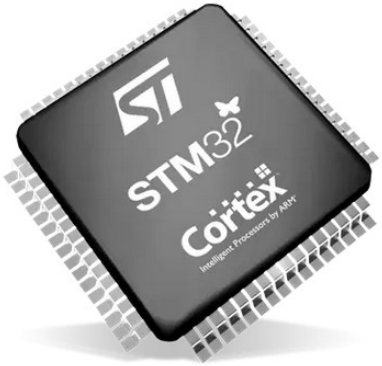


Felhasznált és ajánlott irodalom

- 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 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

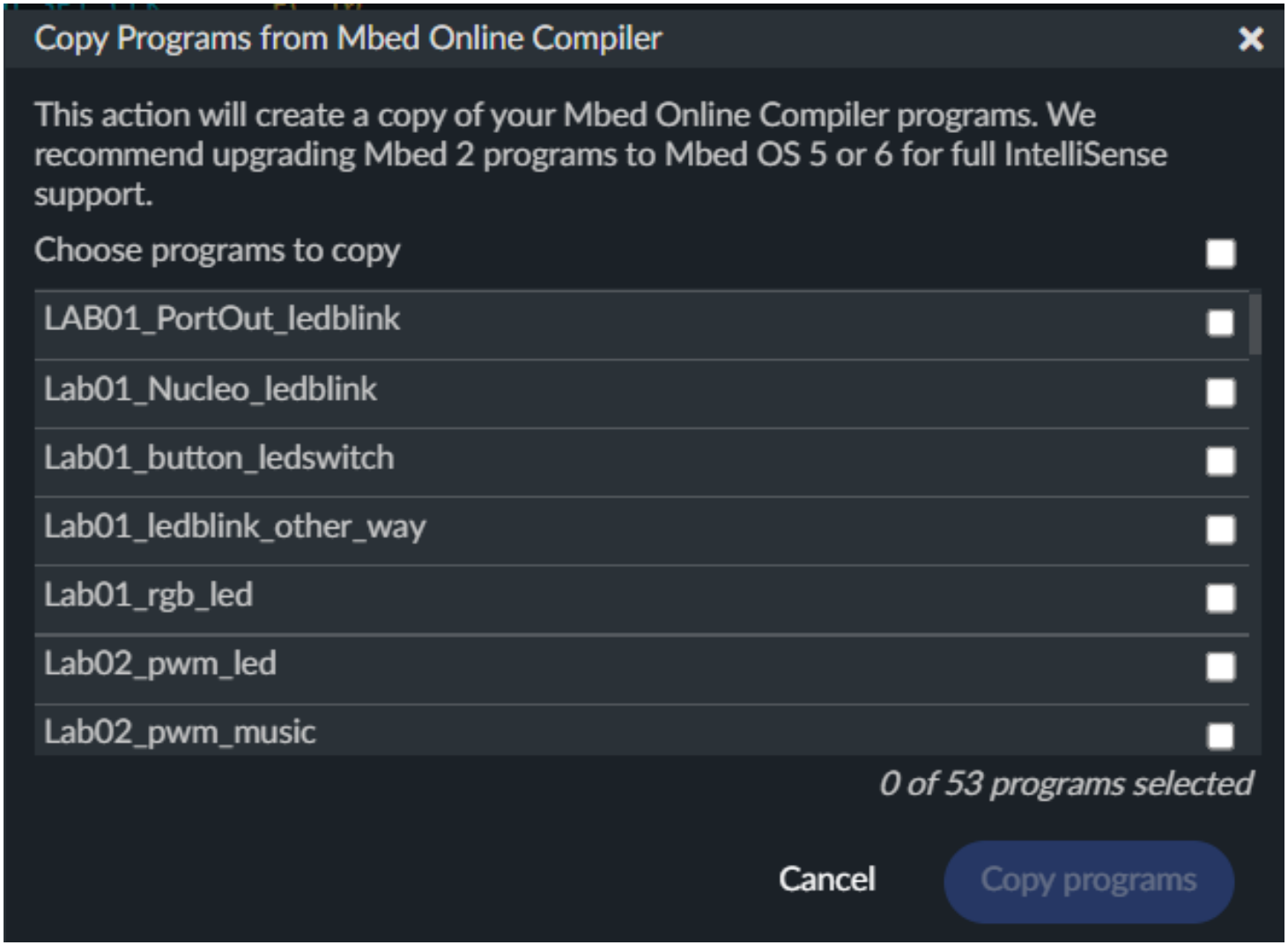
Adatlapok:

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



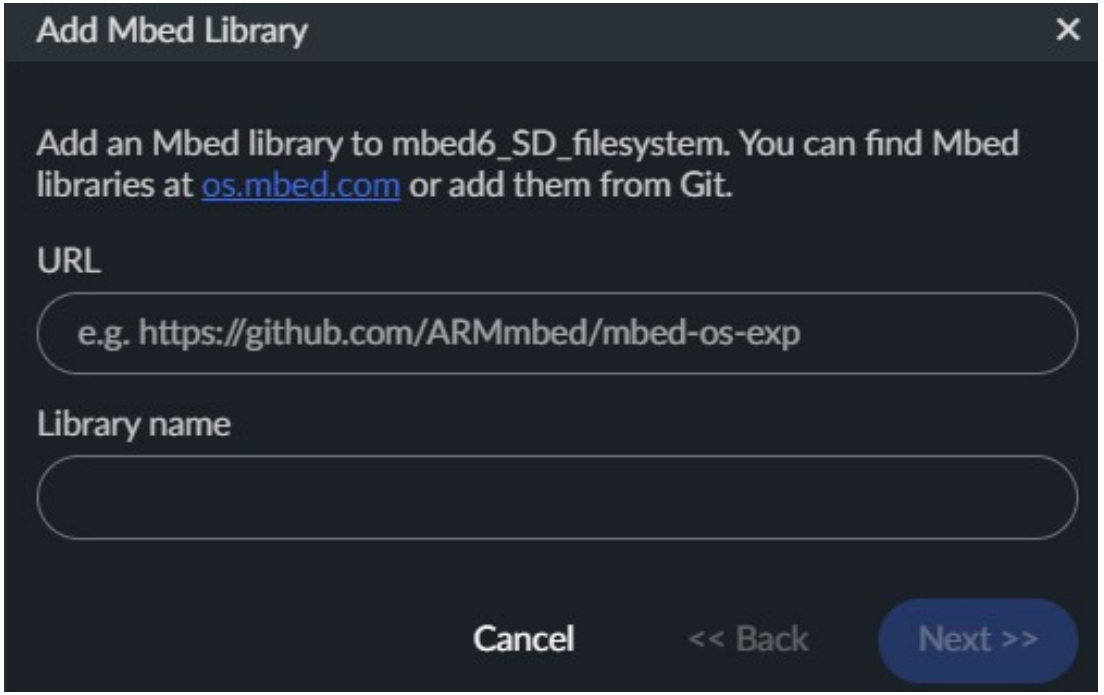
Importálás az Mbed Online Compiler-ből

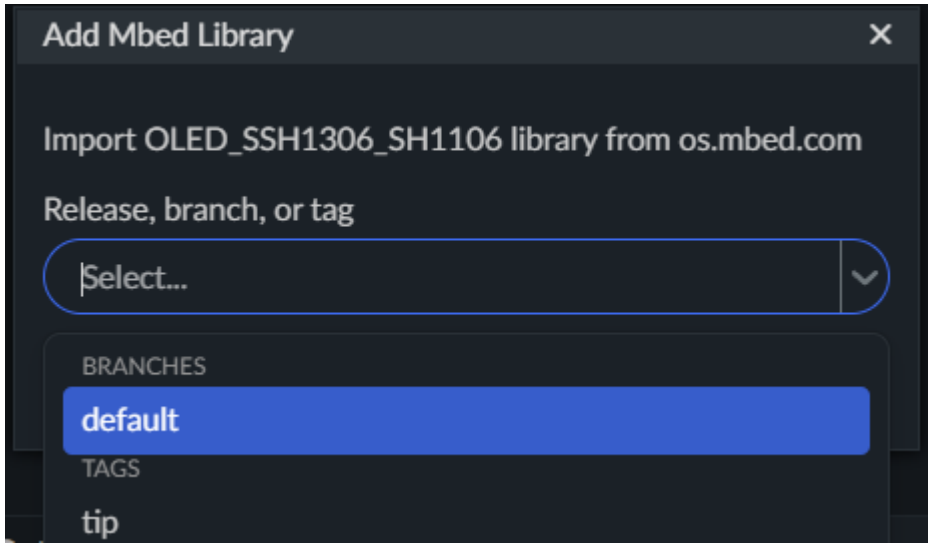
- **File → Import from Mbed Online Compiler** menüpont választása
- Jelöld be az importálni kívánt projekteket, és kattints a **Copy** gombra!



Könyvtárak importálása

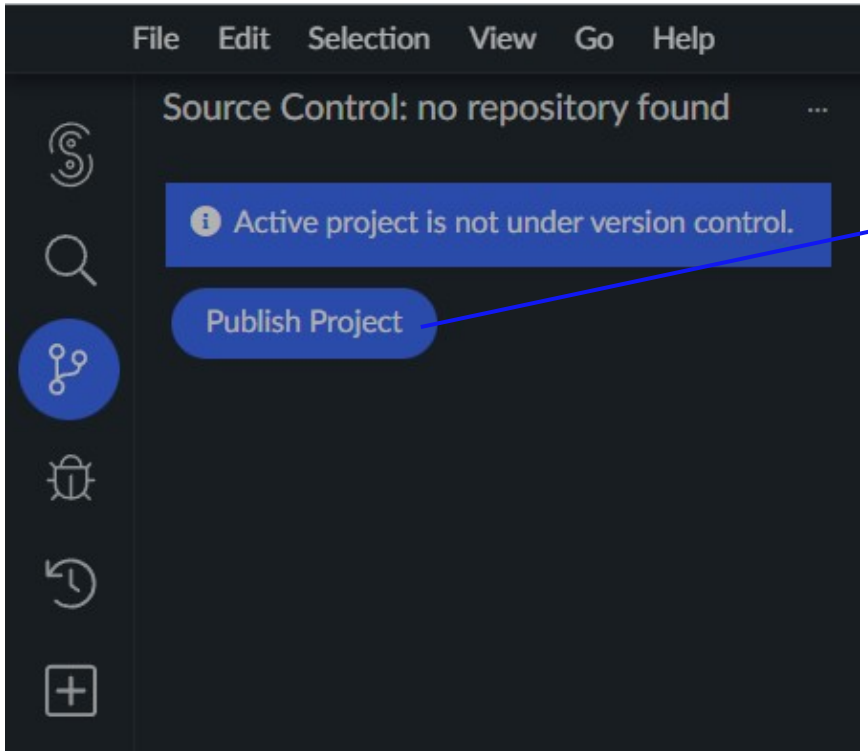
- **File → Add Mbed library to active project** a kiválasztott programkönyvtárat hozzáadja az éppen aktív projekthez
- Az URL rovatba az elérhetőséget írjuk (os.mbed.org vagy GitHub link)
- A **Library Name** rovatba a programkönyvtár mappájának új nevét kell megadni, majd **Next** után a branch (ág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

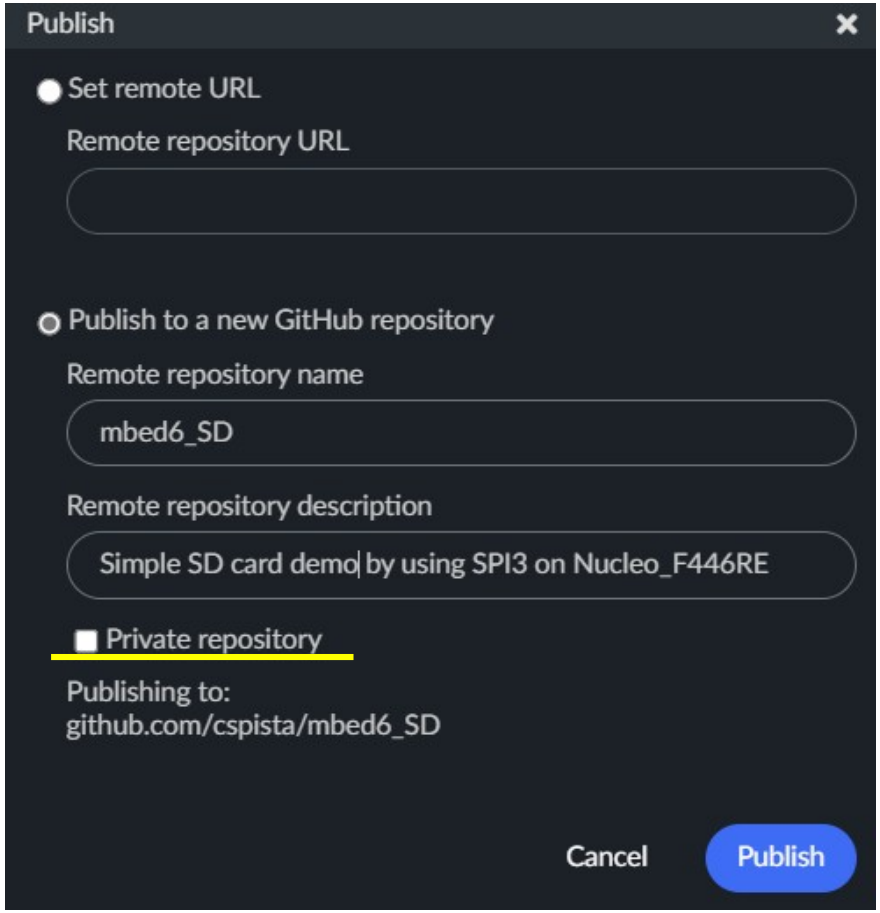




Projektek közzététele

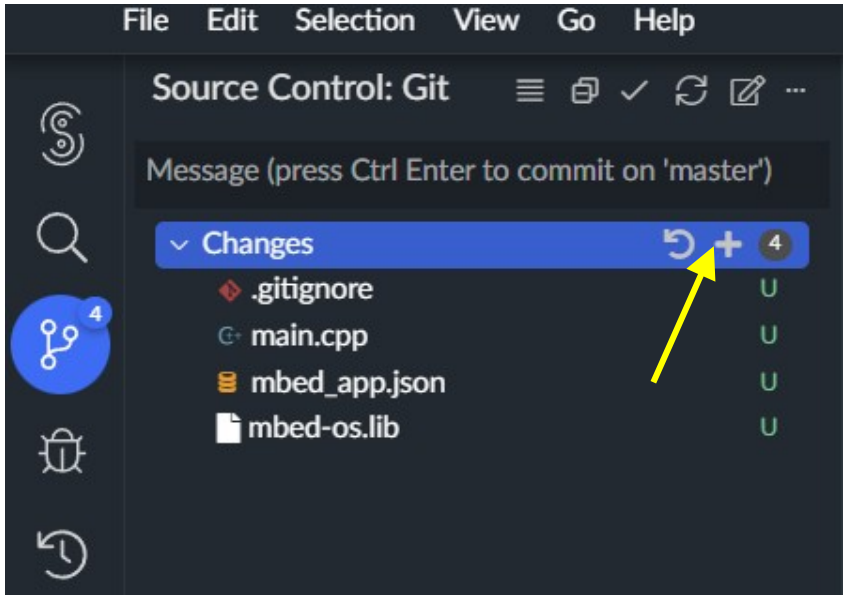
- Az **ARM Keil Studio**-ból a **GitHub** felhő alapú lerakatba tölthetjük fel projektjeinket, ami egyúttal a verziókövetést is biztosítja
- **Publish project** – első lépésként létre kell hozni egy új lerakatot az éppen aktív projekt számára a **GitHub** tárhelyen

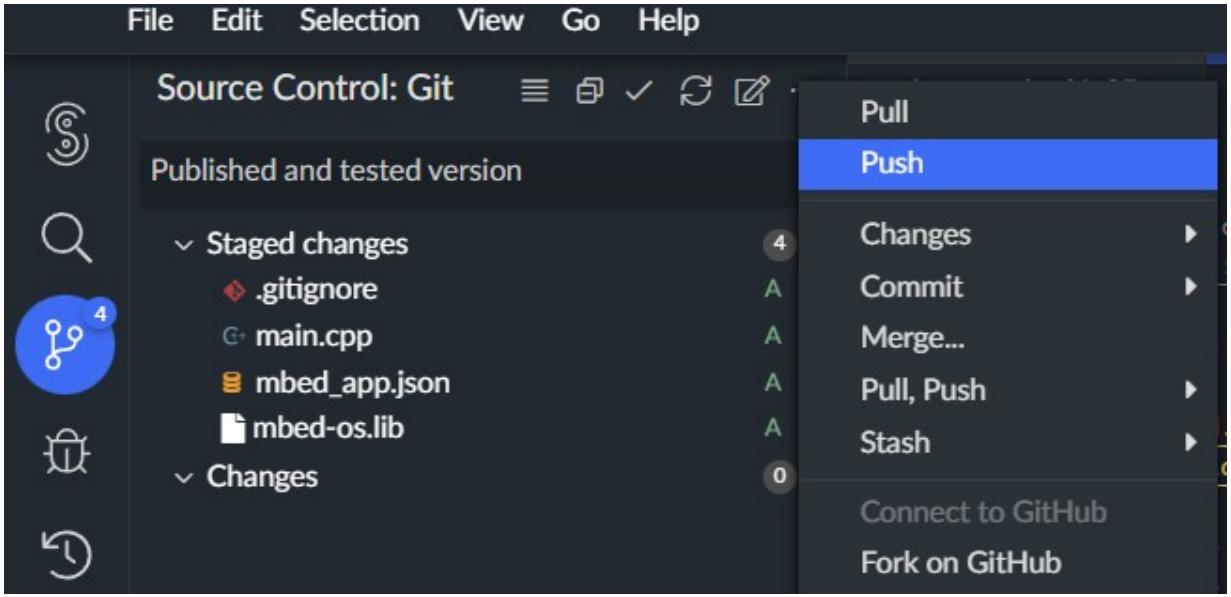




Projektek közzététele

- A változások jüváhagyása után *Message* helyére megjegyzést írhatunk, majd **Ctrl + Enter** vagy a pipa ikonra kattintva **Commit** (véglegesítés) hajtható vége (lokális véglegesítés)
- A **Push**, vagy **Push to** a GitHub lerakatban is érvényesíti a lokális változtatásokat (kivéve, amikor valamilyen titokzatos okból nem akarja végrehajtani a távoli érvényesítést…)
- Leírás: Keil Studio User Guide: Work with Git source control





Példaprogramok

mbed6_SD – SD kártya blokkos kezelés

mbed6_SD_filesystem – SD kártya fájlkezeléssel

mbed6_usbCDC – virtuális soros port

mbed6_usbSerial – virtuális soros port/printf

mbed6_usbHID_PnP – egyedi HID eszköz

mbed6_usbMSD – USB SD kártya-olvasó

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

Projektek konfigurálása

- Az **Arm Mbed OS** konfigurációs rendszere összegyűjti és értelmezi a céláramkörre a **targets.json**-ben megadott konfigurációt, az összes **mbed_lib.json** fájlt és az alkalmazásszintű **mbed_app.json** fájlt
- Nekünk elsősorban az utóbbival lesz dolgunk, amelyben felüldefiniáljuk vagy kiegészítjük az alapértelmezett konfigurációt
- Az **mbed_app.json** fájljainkban többnyire ***requires, config,*** vagy ***target_overrides*** elemek szerepelnek

mbed6_baremetal_queue/mbed_app.json

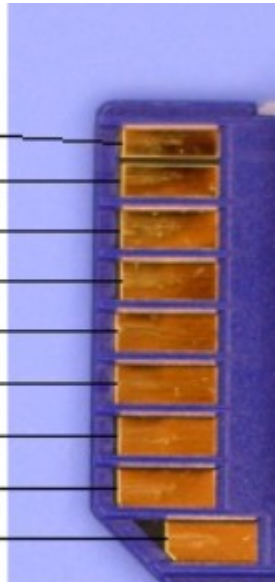
```
{
    "requires": ["bare-metal","events"],
    "target_overrides": {
      "*": {
        "target.c_lib": "small",
        "target.printf_lib": "minimal-printf",
        "platform.minimal-printf-enable-floating-point": false,
        "platform.stdio-minimal-console-only": true,
        "platform.stdio-baud-rate": 115200
      }
    }
}
```

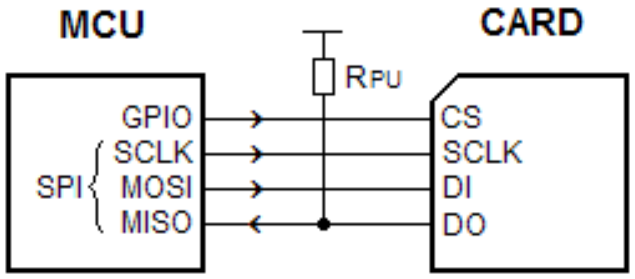
SD kártya használata SPI módban

- A **Secure Digital Memory Card** *de facto* szabvány a hordozható eszközök memóriakártyái közül
- Meghajtható **SPI** módban, illetve 1, vagy 4 adatvonalas **SDIO** módban (az **STM32F446RE** mikrovezérlő rendelkezik **SDIO** perifériával is), az adatok 512 bájtos blokkokba vannak szervezve

SDC

No	SD	SPI
8	DAT1	
7	DAT0	DO
6	Vss2	
5	CLK	SCLK
4	Vcc	
3	Vss1	
2	CMD	DI
1	CAT3	CS
9	DAT2	



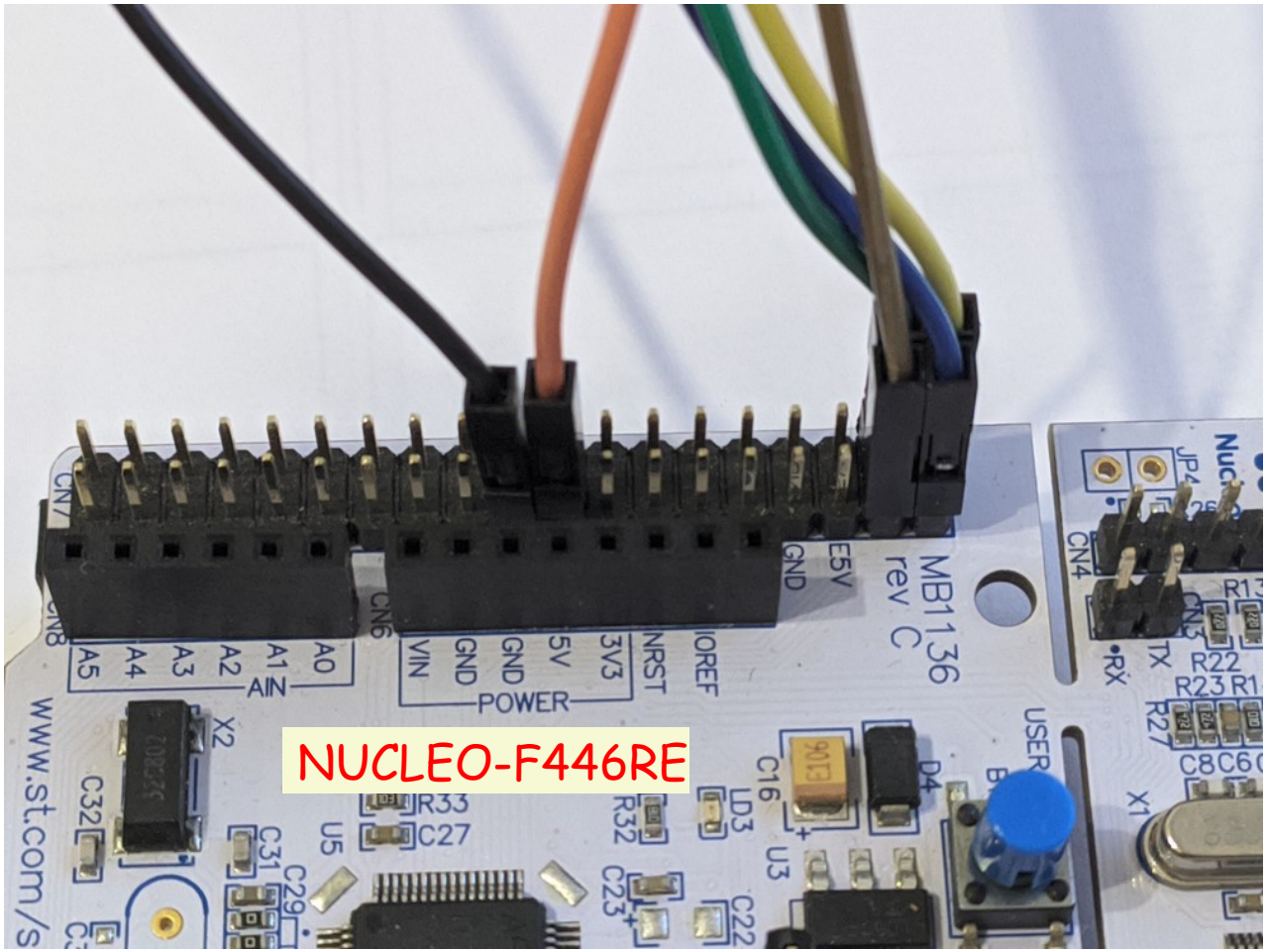


NUCLEO-F446RE kártya

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



mbed6_SD/mbed_app.json

- A **Data storage API** kategóriában 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, ennek konfigurálásához az **mbed_app.json** fájlban az alábbi beállítás szükséges ("*" helyett valójában "NUCLEO_F446RE"-et kellene írni)

```
{
  "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"
    }
  }
}
```

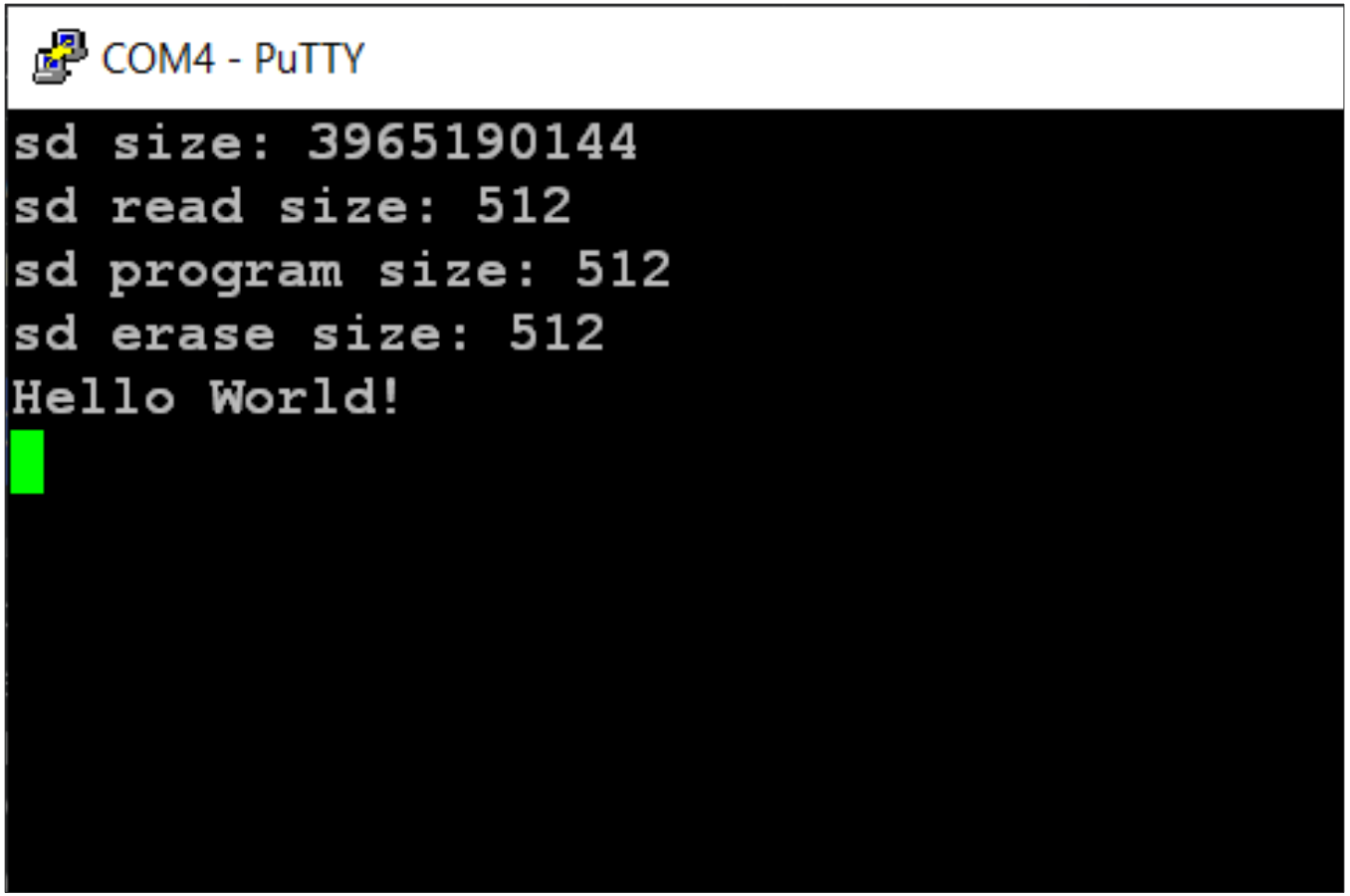
mbed6_SD/main.cpp

```
#include "mbed.h"
#include "SDBlockDevice.h"
SDBlockDevice sd(MBED_CONF_SD_SPI_MOSI, MBED_CONF_SD_SPI_MISO,
MBED_CONF_SD_SPI_CLK, MBED_CONF_SD_SPI_CS);
uint8_t block[512] = "Hello World!\n";

int main() {
  if (0 != sd.init()) { printf("Init failed \n");  return -1;  }    // Call the  initialisation method
  printf("sd size: %llu\n",       sd.size());
  printf("sd read size: %llu\n",   sd.get_read_size());
  printf("sd program size: %llu\n", sd.get_program_size());
  printf("sd erase size: %llu\n",   sd.get_erase_size());

  if (0 != sd.frequency(5000000)) {  printf("Error setting frequency \n"); }    // Set the frequency
  if (0 != sd.erase(0, sd.get_erase_size()))  { printf("Error Erasing block \n");  }
  if (0 == sd.program(block, 0, 512)) {           // Write data block to the device
    if (0 == sd.read(block, 0, 512)) {            // Read the data block from the device
      printf("%s", block);                        // Print the contents of the block
    }
  }
  sd.deinit();                  // Call the SDBlockDevice instance de-initialisation method
}
```

A program forrása: https://os.mbed.com/docs/mbed-os/v6.15/apis/sdblockdevice.html

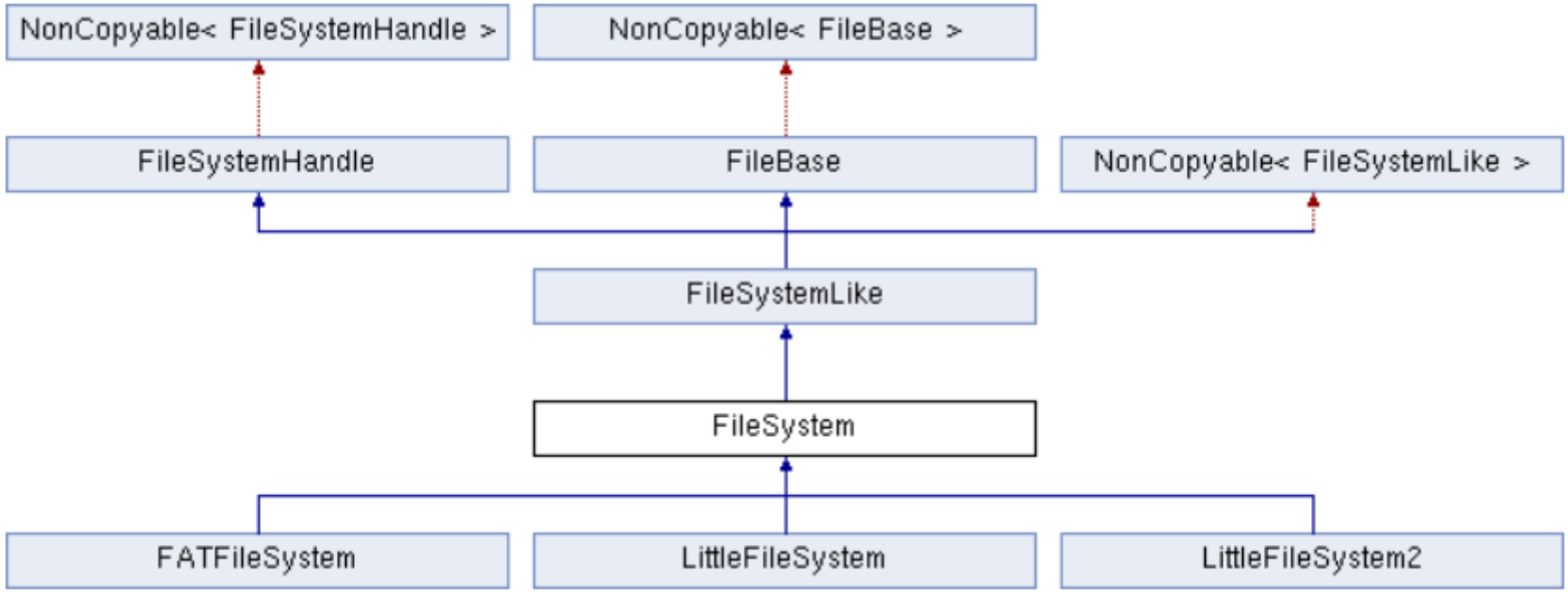
mbed6_SD futási eredménye

- Az alábbiakban egy 4 GB-os SD kártyával futtattuk a programot
- A törlés utáni írás és visszaolvasás során egyetlen blokkot írunk, amelyben csak a **Hello World!** szöveg nullától különböző



Fájlrendszer használata

- A **FATFilesytem**, vagy valamelyik **LittleFileSystem** választható (mi most a LittleFileSystem2-t fogjuk kipróbálni)
- A Windows/Linux/MacOS rendszerekkel csak a **FATFilesytem** kompatibilis, ám ennek erőforrásigénye valamivel nagyobb
- Dokumentáció: Mbed OS File system APIs



mbed6_SD_filesystem/main.cpp - 1/3

- Mbed official example: Getting started with the SD-driver

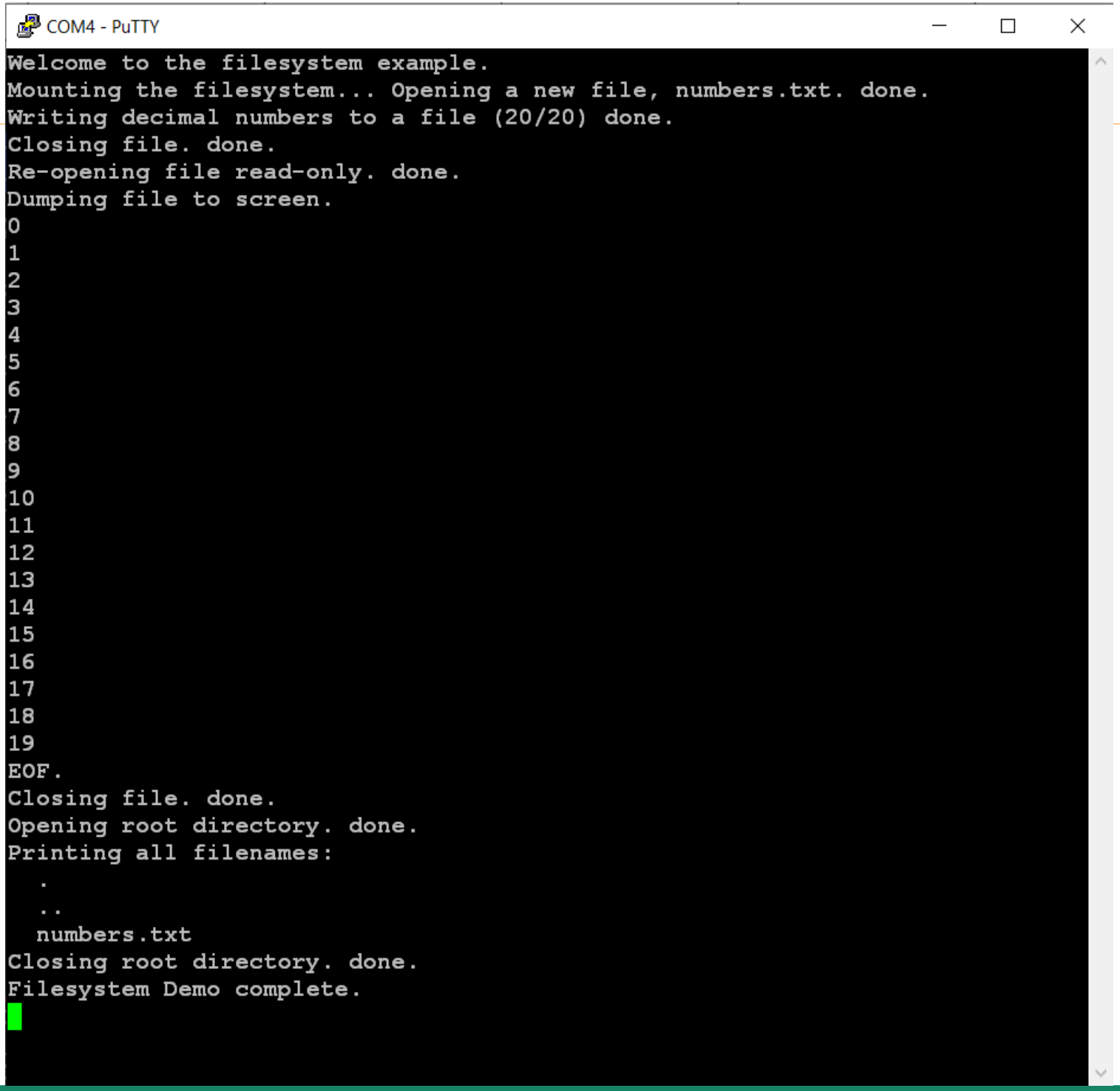
```
#include "mbed.h"
#include "LittleFileSystem2.h"
#include "SDBlockDevice.h"
#include <stdio.h>
#include <errno.h>
#include "platform/mbed_retarget.h"
SDBlockDevice sd(MBED_CONF_SD_SPI_MOSI,MBED_CONF_SD_SPI_MISO,
                 MBED_CONF_SD_SPI_CLK, MBED_CONF_SD_SPI_CS);
LittleFileSystem2 fs("sd", &sd);

void return_error(int ret_val) {
    if (ret_val) {
        printf("Failure. %d\n", ret_val);
        while (true) { __WFI(); }
    } else { printf("done.\n"); }
}

void errno_error(void *ret_val) {
    if (ret_val == NULL) {
        printf(" Failure. %d \n", errno);
        while (true) { __WFI(); }
    } else {
        printf(" done.\n");
    }
}
```

mbed6_SD_filesystem/main.cpp - 2/3

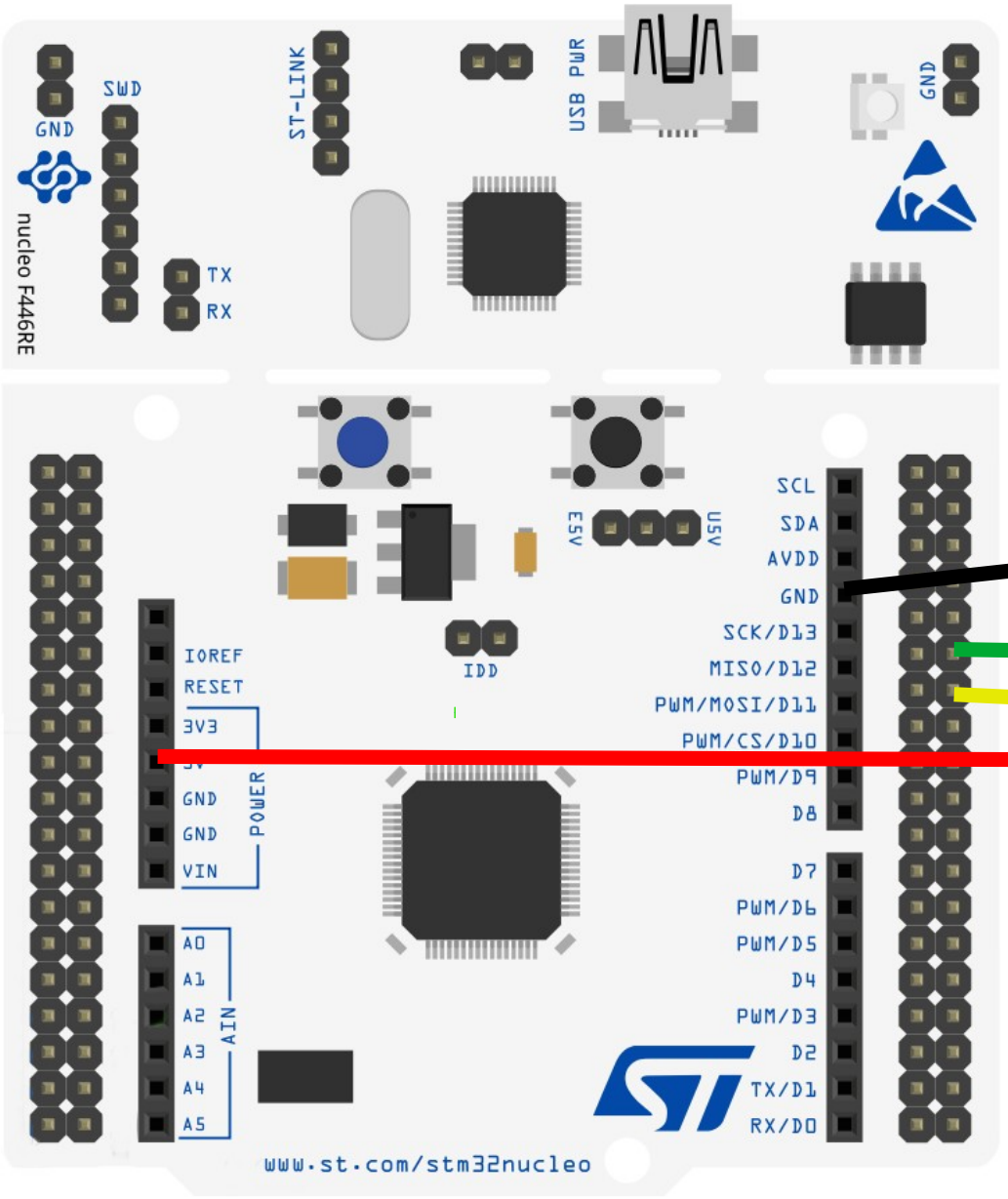
```
int main() {
    int error = 0;
    printf("Welcome to the filesystem example.\n");
    printf("Mounting the filesystem... ");    // Try to mount the filesystem
    error = fs.mount(&sd);
    if (error) {
        printf("No filesystem found, formatting... "); // Reformat if we can't mount
        error = fs.reformat(&sd); // this should only happen on the first boot
        return_error(error);
    }
    printf("Opening a new file, numbers.txt.");        // Opening new file for writing
    FILE *fd = fopen("/sd/numbers.txt", "w+");
    errno_error(fd);

    for (int i = 0; i < 20; i++) {
        printf("Writing decimal numbers to a file (%d/20)\r", i);
        fprintf(fd, "%d\n", i);
    }
    printf("Writing decimal numbers to a file (20/20) done.\n");

    printf("Closing file.");
    fclose(fd);
    printf(" done.\n");
    printf("Re-opening file read-only.");    // Reopening file in read-only mode
    fd = fopen("/sd/numbers.txt", "r");
    errno_error(fd);
```

mbed6_SD_filesystem/main.cpp - 3/3

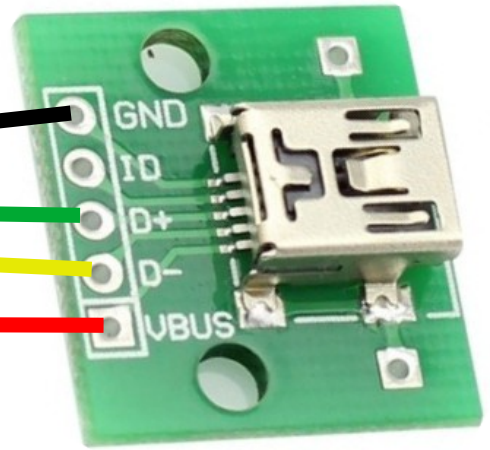
```
    printf("Dumping file to screen.\n");
    char buff[16] = {0};
    while (!feof(fd)) {
        int size = fread(&buff[0], 1, 15, fd);
        fwrite(&buff[0], 1, size, stdout);
    }
    printf("EOF.\n");
    printf("Closing file.");
    fclose(fd);
    printf(" done.\n");

    printf("Opening root directory.");
    DIR *dir = opendir("/sd/");
    errno_error(fd);

    struct dirent *de;
    printf("Printing all filenames:\n");
    while ((de = readdir(dir)) != NULL) {
        printf("  %s\n", &(de->d_name)[0]);
    }
    printf("Closing root directory. ");
    error = closedir(dir);
    return_error(error);
    printf("Filesystem Demo complete.\n");
    while (true) {}
}
```


COM4 - PuTTY
Welcome to the filesystem example.
Mounting the filesystem... Opening a new file, numbers.txt. done.
Writing decimal numbers to a file (20/20) done.
Closing file. done.
Re-opening file read-only. done.
Dumping file to screen.
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
EOF.
Closing file. done.
Opening root directory. done.
Printing all filenames:
.
..
numbers.txt
Closing root directory. done.
Filesystem Demo complete.

Az USB_FS csatorna használata



PA12 USB D+
PA11 USB D-



STM32F4xx: a **D+** vonal külső felhúzására nincs szükség!

Az USB_FS csatorna használata

- A **NUCLEO-F446RE** kártya esetén az USB csatornát az Mbed OS alapértelmezetten nem konfigurálja, ezt nekünk kell megtenni az **mbed_app,json** konfiguráció állományban
- Az Mbed OS **USB API** az alábbi osztályokat nyújtja:
  - USBAudio – hangadat küldés (pl. mikrofon/hangszóró)
  - **USBCDC** – kommunikációs eszközösztály, virtuális soros port
  - USBCDC_ECM – Ethernet illesztő emuláció USB-n keresztül
  - **USBHID** – általános HID (human interface device) osztály
  - USBKeyboard – USB billentyűzet
  - USBMIDI – USB-MIDI protokoll szerint üzenetek kétirányú továbbítása
  - USBMouse – USB egér
  - USBMouseKeyboard – USB billentyűzet és egér
  - **USBMSD** – USB tömegtároló osztály (pl. pendrive)
  - **USBSerial** – USBCDC-hez hasonló, de **printf**() funkcionalitással

NUCLEO-F446RE mbed_app.json

- A **NUCLEO-F446RE** kártyán az **USB_OTG_FS** csatornát használjuk, ennek konfigurálásához az **mbed_app.json** fájlban az alábbi beállítás szükséges (**"*"** helyett valójában **"NUCLEO_F446RE"**-et kellene írni)
- Ha más eszközt is konfigurálnunk kell (SD kártya, soros terminál, akkor az alábbiakat természetesen ki kell majd egészíteni a szükséges sorokkal

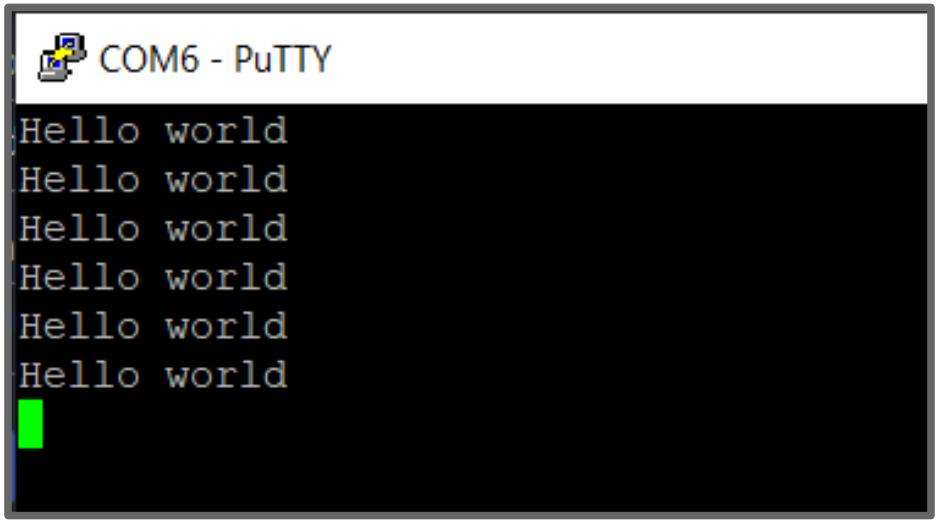
```
{
  "config": {
    "usb_speed": {
      "help": "USE_USB_OTG_FS or USE_USB_OTG_HS or USE_USB_HS_IN_FS",
      "value": "USE_USB_OTG_FS"
    }
  },
  "target_overrides": {
    "*": {
      "target.device_has_add": ["USBDEVICE"]
    }
  }
}
```

Az USB nem 'component', hanem 'device'

mbed6_usbCDC/main.cpp

- Mbed felhasználói kézikönyvi mintapélda: USBCDC Example
- Apró kompatibilitási probléma: az eredeti mintapéldában található **strlen()** függvényt a **Keil Studio** fordítója nem találja, ezért helyettesítettük azt a **std::char_traits<char>::length(msg)** függvényhívás lánccal (bővebb info)

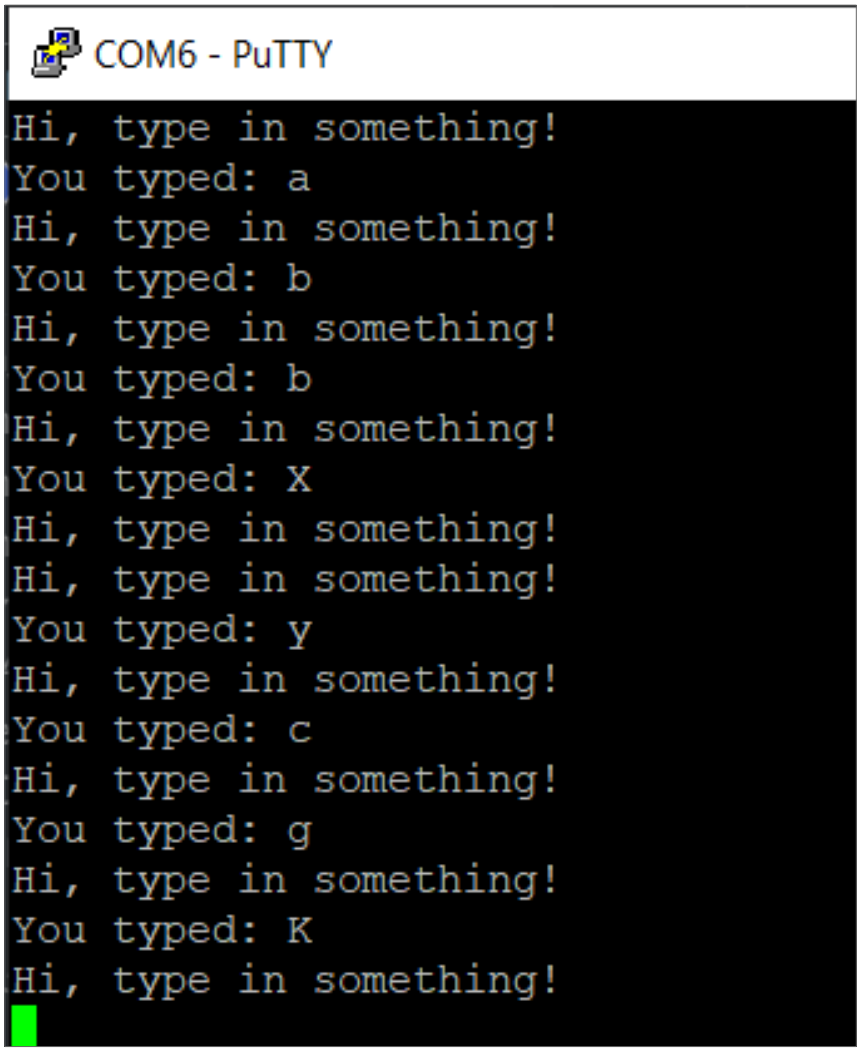
```
#include "mbed.h"
#include "USBCDC.h"
#include <string>

USBCDC cdc;

int main(void) {

  while (1) {
    char msg[] = "Hello world\r\n";
    cdc.send((uint8_t *)msg, std::char_traits<char>::length(msg));  // strlen() function has been replaced
    ThisThread::sleep_for(1000ms);
    printf("Mbed OS %d.%d.%d.\n", MBED_MAJOR_VERSION,
           MBED_MINOR_VERSION, MBED_PATCH_VERSION);
    ThisThread::sleep_for(1000ms);
  }
}
```



mbed6_usbSerial/main.cpp

- Ezzel az objektumosztállyal a **putc/getc** és a **printf/scanf** funkcionalitás is használható. Az alábbi program bemutatja a beolvasást és a kiírást

```
#include "mbed.h"
#include "usb/USBSerial.h"

USBSerial serial;     //Virtual serial port over USB
int main(void) {
  while(1) {
    serial.printf("Hi, type in something!\r\n");
    char ch = serial.getc();
    if(ch > ' ') {
      serial.printf("You typed: %c\r\n", ch);
    }
    ThisThread::sleep_for(500ms);
  }
}
```

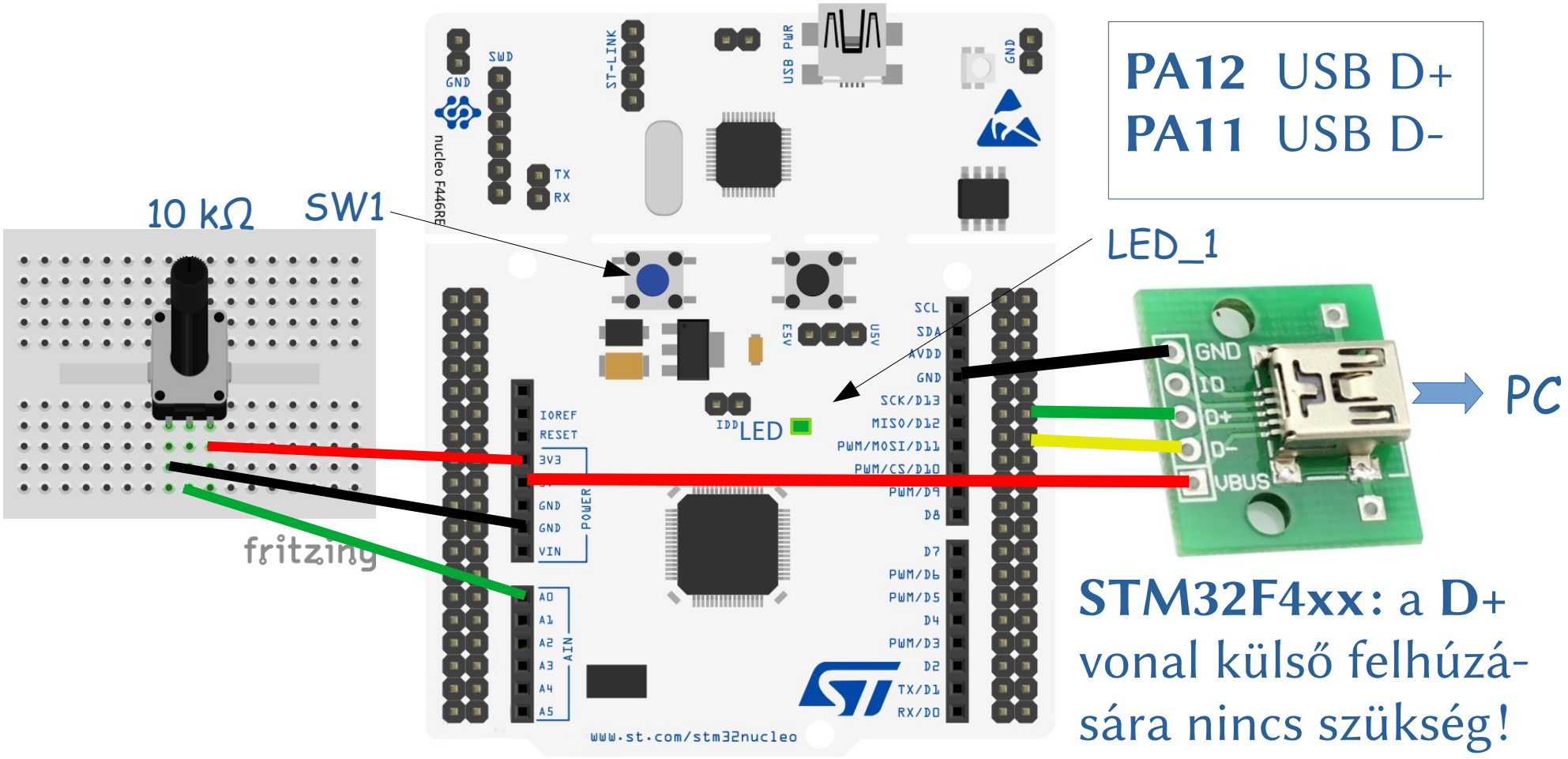


Egyedi USB HID eszköz

- Az általános (generic) **HID** eszközöknél a **HID report descriptor** csak az átvitt adatok mennyiségét mondja meg, azok értelmezése a mikrovezérlőbe töltött firmware-re és a PC-n futó alkalmazásra van bízva. Ebben az esetben tehát csak az átvitel miként szabványos, az adatok felhasználása azonban egyedi, gyártóspecifikus.
- Az **USBHID** osztály az előzőekben említett *"Általános HID Eszközt"* (Generic HID Device) valósítja meg
- A **számítógépen kell olyan alkalmazás,** amely képes kezelni az általunk készített egyedi eszköz **USBHID** adatforgalmát. Ez lehet akár egy szkript is, amihez használhatjuk a Python értelmező **pywinusb** kiegészítését. Egy másik lehetőség a **hidapi** könyvtár, amelyet C/C++ konzol alkalmazásokból vagy grafikus alkalmazásokból is használhatunk. Az **USBHID bindings** és az **USBHID C bindigs** oldalon találunk mintapéldákat mindkettő használatára.

mbed6_usbHID_PnP

- A 2021. április 8-i előadás **F446RE_USB_HID_PnP** projektjét írjuk át **Mbed-OS v6** környezetre
- A kapcsolási vázlat az alábbi ábrán látható



mbed6_usbHID_PnP

- A program a PC felől érkező **64 bájtos üzenetekből** az első bájtot parancsként értelmezi, az alábbi táblázat szerint

Parancs	Válasz	Szöveges leírás
0x80	nincs	LED állapot átbillentése (LED1 a PA5 kivezetésre van kötve)
0x81	0x81 1/0	Nyomógomb állapotának lekérdezése (SW1 a PC13 kivezetésre van kötve)
0x37	0x37 LSB MSB	ADC IN0 csatorna értékének lekérdezése (IN0 a PA0 kivezetésre van kötve)
egyéb	0xFF	Nem értelmezett parancs

- A **0x80** parancsra nem küldünk választ
- A **0x81** parancsnál a második bájt 1 vagy 0, a **PC13** bemenet állapotától függően
- A **0x37** parancsnál az **ADC** által visszaadott érték 0 – 4095 közötti szám, amit *low endian* sorrendben küldünk ki az üzenet második és harmadik bájtjában
- PC oldalon a Microchip Library for Applications (MLA) **USB HID PnP demo** mintaalkalmazását „vettük kölcsön” és adaptáltuk (Vid/Pid csere, ProgressBar1 maximuma 1024 helyett 4096)

mbed6_usbHID_PnP/main.cpp – 2/1.

- A HID eszköz itt 64 bájtos jelentéseket küld/fogad
- A VID: 0x0483, PID: 0x5750 páros az ST Mictroelectronics STM32 Custom Human interface eszköz azonosítója

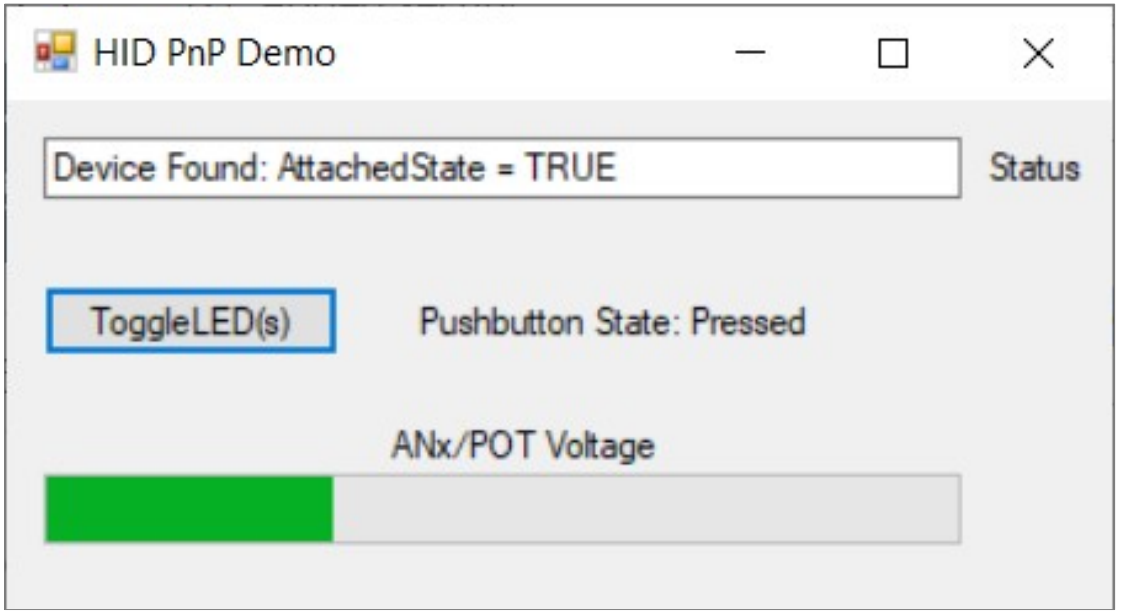
```
#include "mbed.h"
#include "usb/USBHID.h"

//We declare a USBHID device. Here the input and output reports are 64 bytes long.
USBHID hid(true,64,64,0x0483,0x5750); // Vid/Pid: STM32 Custom Human Interface Device

HID_REPORT send_report;                 //This report will contain data to be sent
HID_REPORT recv_report;                 //This report will contain data received

DigitalOut LED_1(LED1);                 //Buitin LED at PA5
DigitalIn SW1(BUTTON1,PullUp);          //Builtin button at PC13
AnalogIn adc(A0);                       //Analog input at A0 (PA0)

int main(void) {
    uint16_t val = 0;
    send_report.length = 64;
    LED_1 = 0;
    for (int i = 0; i < send_report.length; i++)                // Fill the report
        send_report.data[i] = 0x00;
    while (1) {
```

mbed6_usbHID_PnP/main.cpp – 2/2.

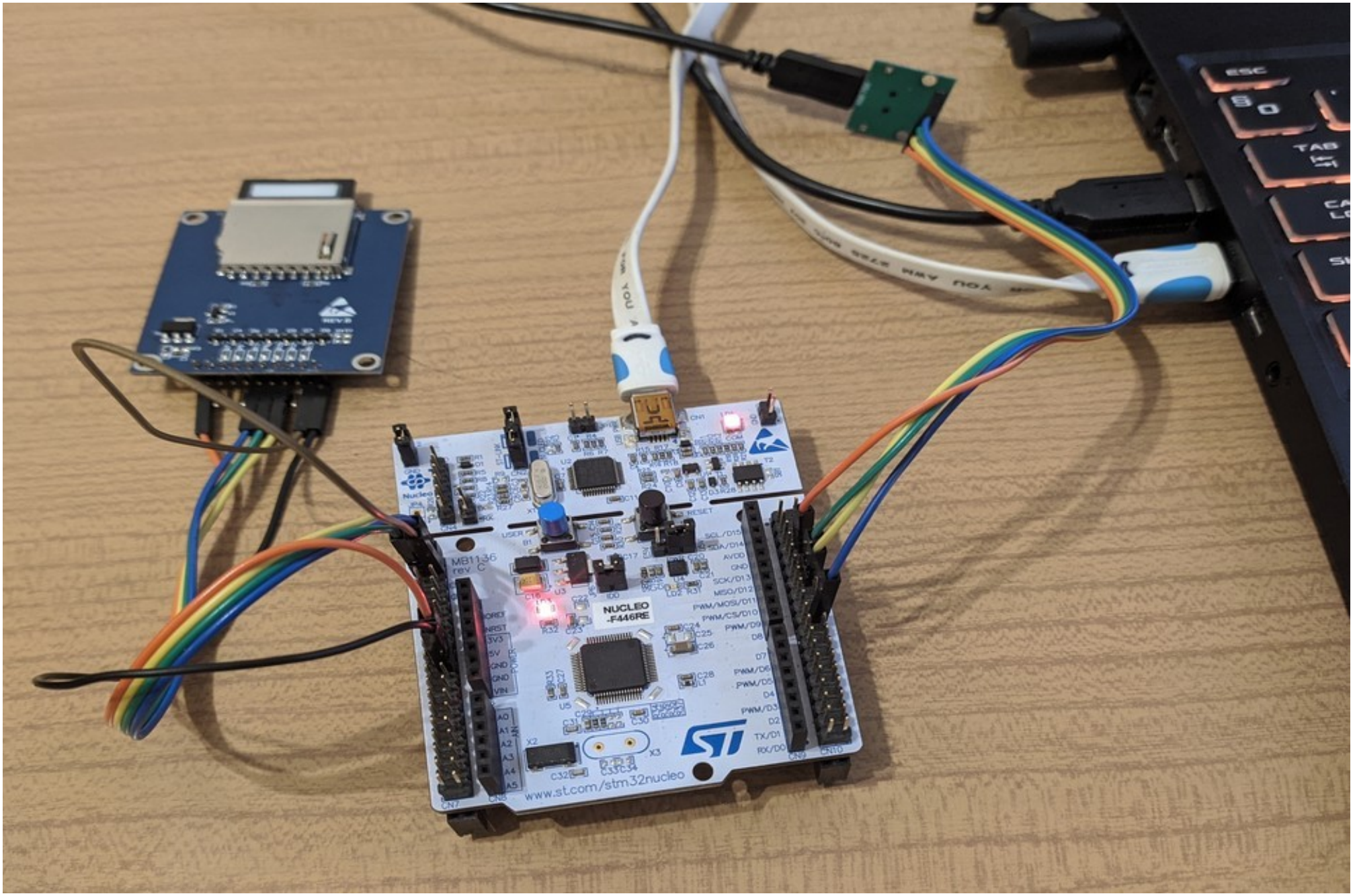
```
        if(hid.read_nb(&recv_report)) {       // read a msg (it was readNB formerly)
            switch(recv_report.data[0]) {
                case 0x80: //Toggle LED state
                    LED_1 = !LED_1;
                    break;
                case 0x81: //Get push button state
                    send_report.data[0] = recv_report.data[0];
                    send_report.data[1] = SW1;
                    send_report.data[2] = 0;
                    hid.send(&send_report);                    // Send the report
                    break;
                case 0x37: //Read POT command.
                    send_report.data[0] = recv_report.data[0];
                    val = adc.read_u16()>>4;                   // Convert to 12-bits
                    send_report.data[1] = val & 0xFF;          // Measured value LSB
                    send_report.data[2] = val >> 8;            // Measured value MSB
                    hid.send(&send_report);                    // Send the report
                    break;
                default:  {
                    send_report.data[0] = 0xFF;                // Invalid command
                    hid.send(&send_report);                    // Send the report
                }
            }
        }
    }
}
```

mbed6_usbHID_PnP futási eredménye

- A Status címke mellett a kapcsolat állapotát láthatjuk
- A nyomógombbal a beépített LED állapotát kapcsolgathatjuk
- A gomb melletti felirat a beépített nyomógomb állapotát jelzi ki
- Az állapotsáv az **A0** bemenetre kapcsolt feszültséget mutatja (0 – 3,3 V a mérési/kijelzési tartomány)



mbed6_usbMSD – SD kártyával



NUCLEO-F446RE SD/USB mbed_app.json

- A **NUCLEO-F446RE** kártyán az **USB_OTG_FS** és az SD kártyához az SPI 3 csatornát kell konfigurálni (a korábbiak egyesítésével)

```
{
    "config": {
        "usb_speed": {
            "help": "USE_USB_OTG_FS or USE_USB_OTG_HS or USE_USB_HS_IN_FS",
            "value": "USE_USB_OTG_FS"
        }
    },
    "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",
            "target.device_has_add": ["USBDEVICE"]
        }
    }
}
```

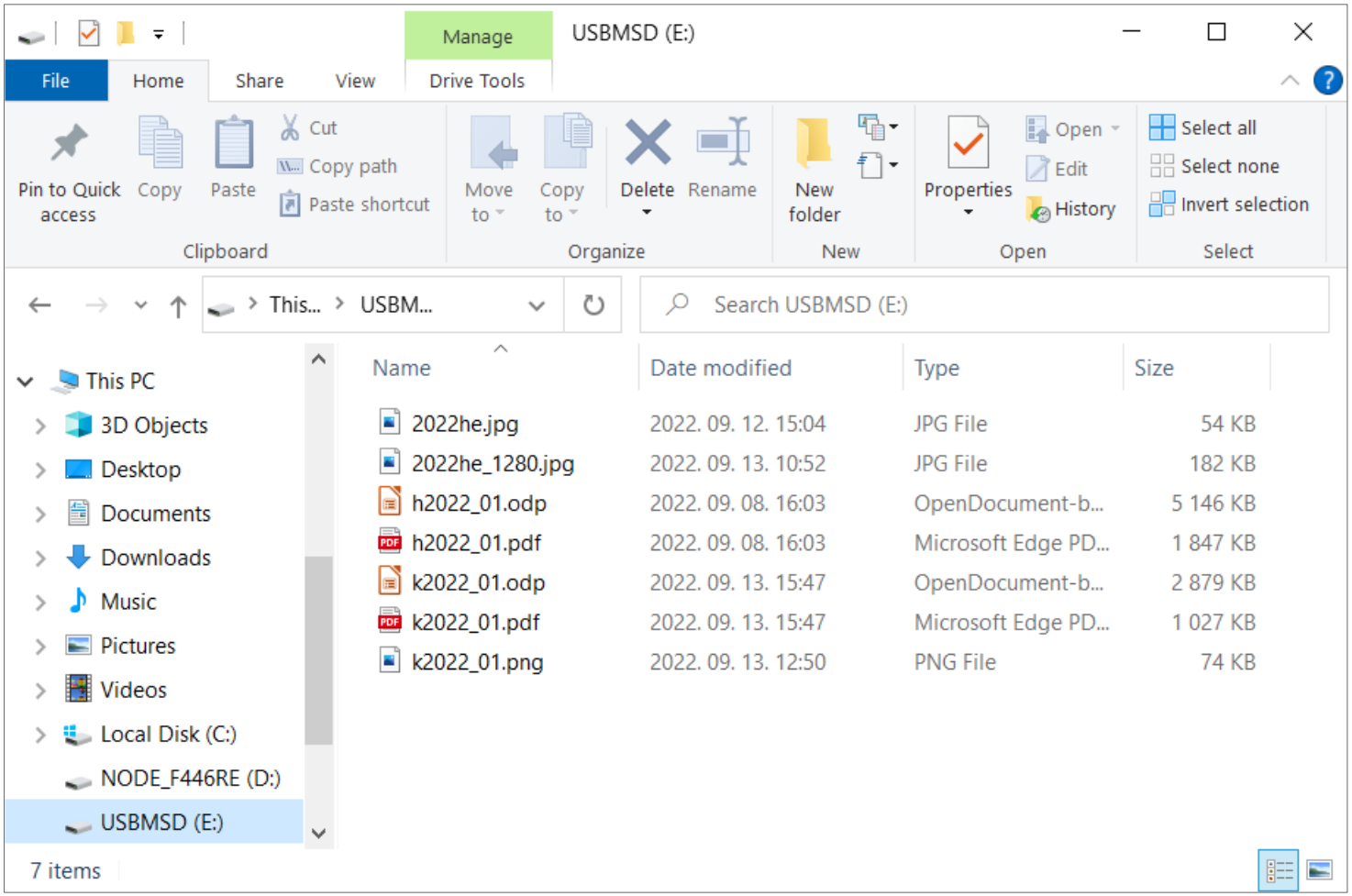
mbed6_usbMSD/main.cpp

- Az USBMSD osztály az USB tömegtároló eszközt valósítja meg, melynek példányosításához egy blokkos adatátvitelű eszközt (itt az SD kártyát) kell biztosítani
- A dokumentáció mintapéldája csak ennyi – és működik!

```
#include "mbed.h"
#include "SDBlockDevice.h"
#include "USBMSD.h"

SDBlockDevice sd(MBED_CONF_SD_SPI_MOSI, MBED_CONF_SD_SPI_MISO,
                 MBED_CONF_SD_SPI_CLK, MBED_CONF_SD_SPI_CS);
USBMSD usb(&sd);

int main() {
  while (true) {
    usb.process();
  }
  return 0;
}
```

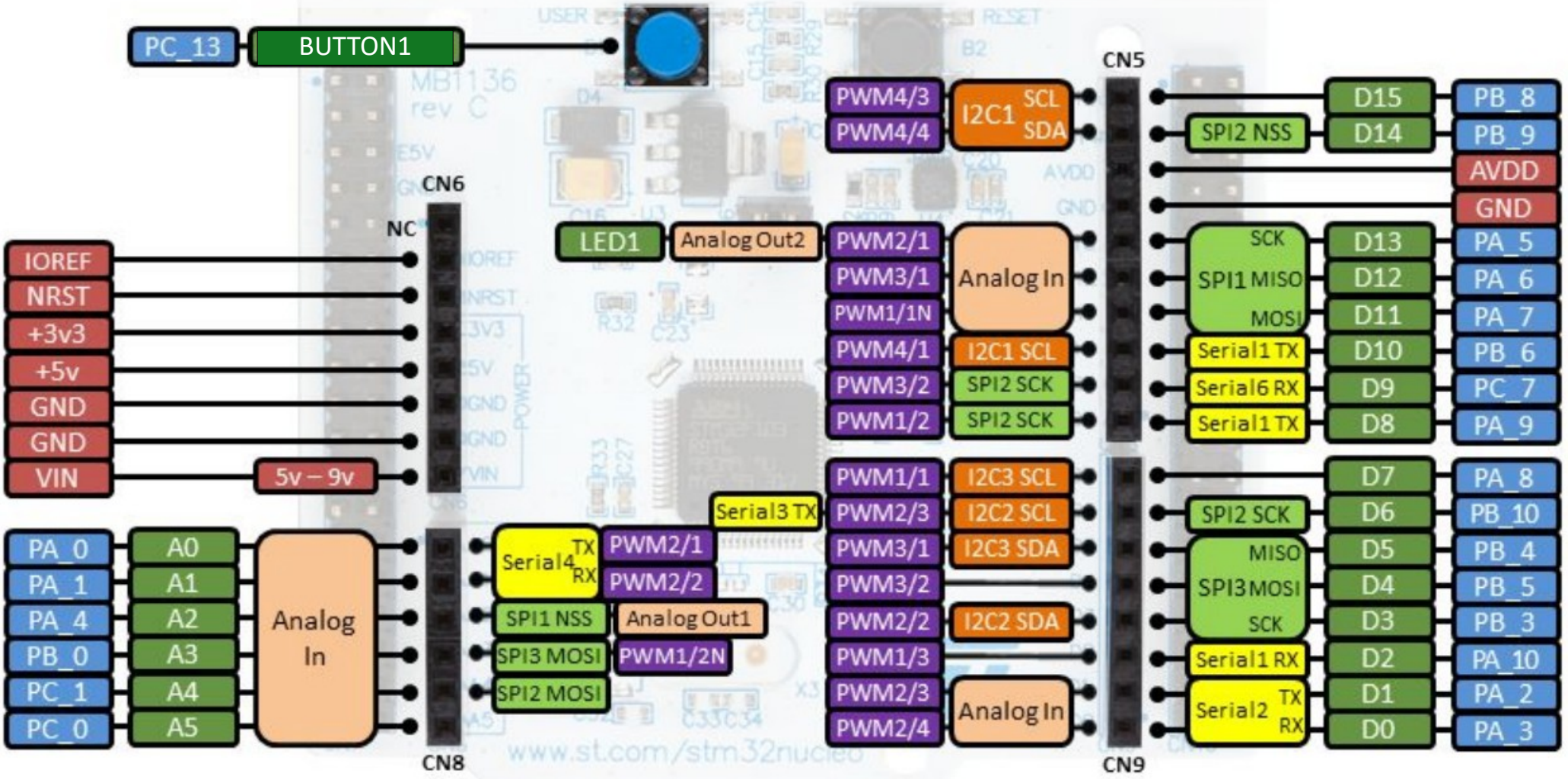
mbed6_usbMSD

- Csatlakoztatás után az SD kártyát előbb formázni kell, majd a pendrive-oknál megszokott módon írhatjuk/olvashatjuk a fájlrendszerét



Arduino kompatibilis kivezetések

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



Morpho kivezetések

Nucleo F446RE
Morpho Headers