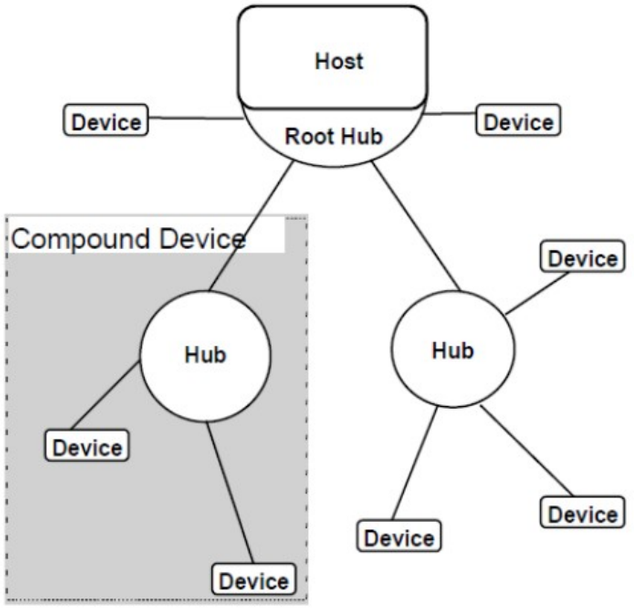
STM32 mikrovezérlők programozása STM32CubeIDE környezetben

11. Az USB HID eszközosztály

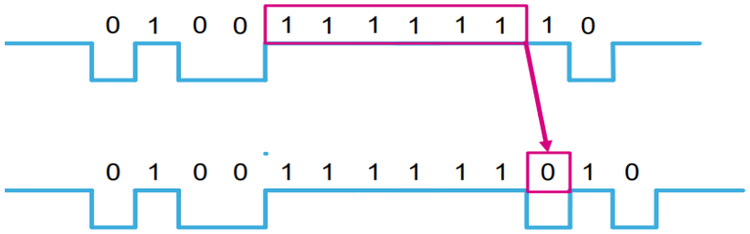
Felhasznált és ajánlott irodalom

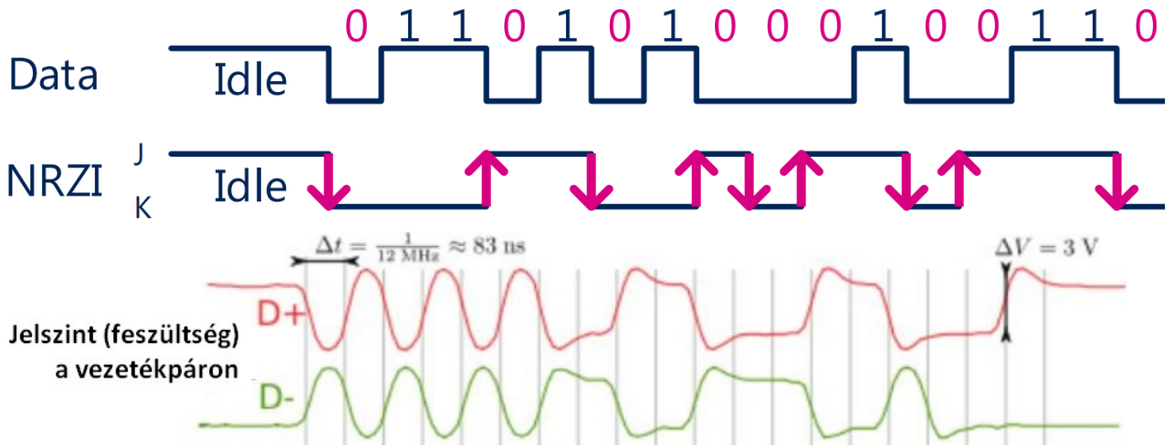
- STMicroelectronics: STM32 USB training (27 video + letölthető anyagok)
  - Youtube lejátszási lista
  - Letölthető anyagok (ZIP fájl)
- STMicroelectronics: UM1734 - STM32Cube™ USB device library
- Jan Axelson: USB complete (könyv)
- Jan Axelseon: The HID page (weblap)
- Cypress: AN57 294 – An Introduction to Universal Serial Bus 2.0
- Microchip: Microchip Libraries for Applications (innen egy PC alkalmazást veszünk át)
- Oliv (ioPush,net): STM32 - custom USB HID device step by step

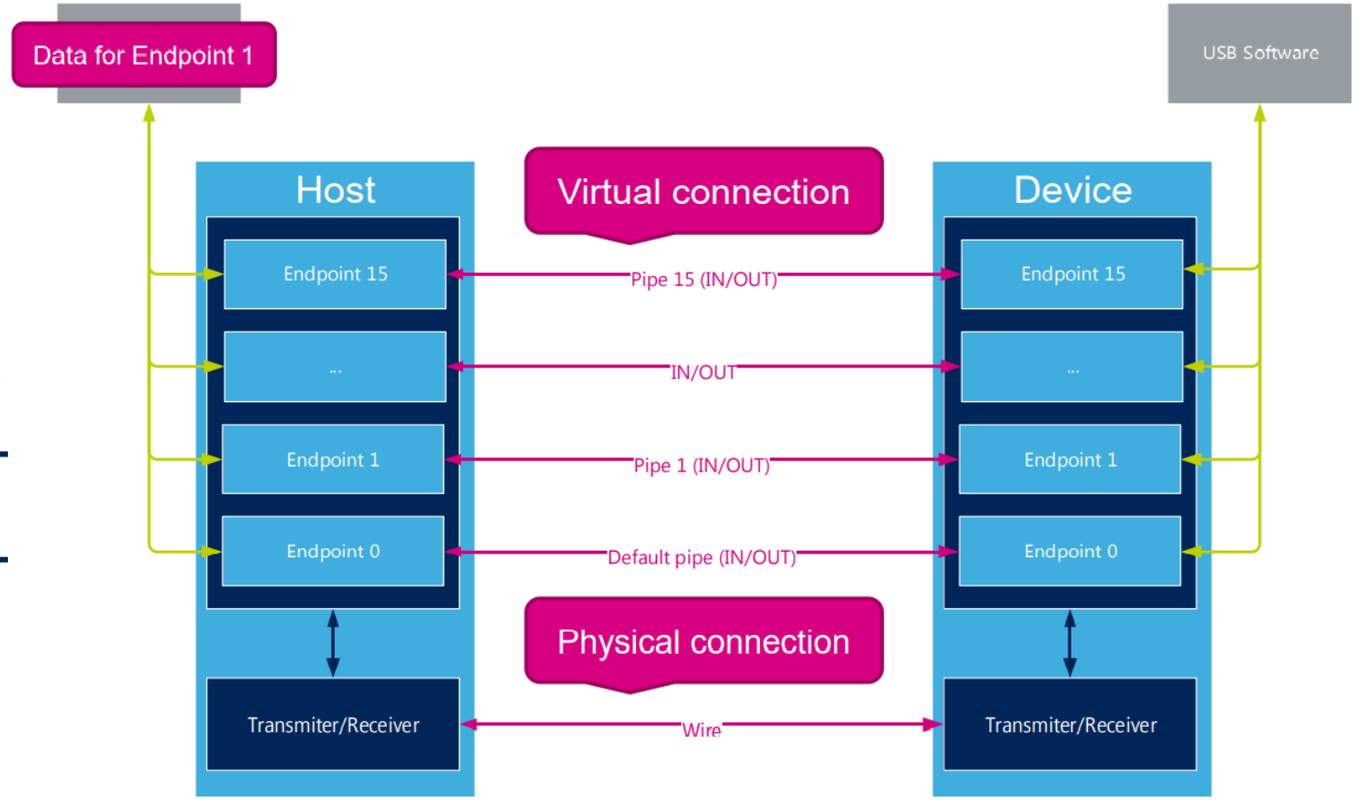
Emlékeztető



- USB: többszintű, csillag topológiájú hálózat, host - device viszony
- Jeltovábbítás: differenciális vezetékpár, bit stuffing, NRZI kódolás
- Az eszközök 7 bites címet kapnak, az eszközökben max. 16 végpont
- A tranzakciók virtuális kapcsolaton a host és egy kiválasztott eszköz megadott végpontjai között zajlanak (virtuális csővonal)



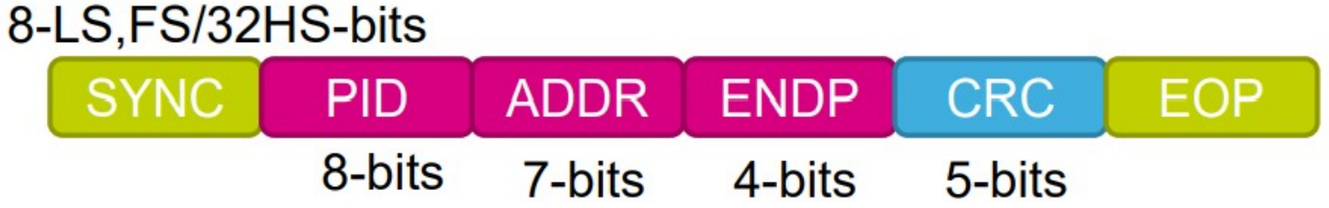




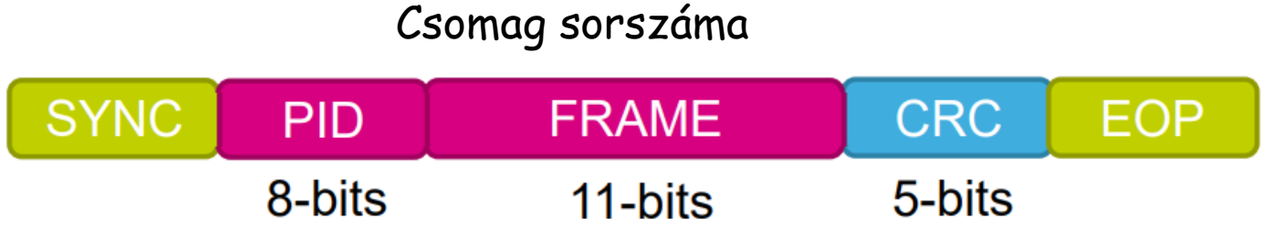
Csomagok és struktúrák

A tranzakciók építőkövei a csomagok, amelyek típus szerint az alábbiak lehetnek:

- **Token** (jelzés) csomag
  - **SYNC** – szinkronizálás
  - **PID** – csomagtípus jelzése (4bit)
  - **ADDR** – cél (néha a forrás) címe
  - **ENDP** – a cél végpont száma
  - **CRC** – ellenőrző kód (token/SOF 5bit, data 16 bit)
  - **EOP** – csomag vége jelzés



- **Start Of Frame** (SOF, keret kezdete)
  - FRAME – a csomag sorszáma



- **Data** (adat) csomag
  - **DATA** – adat, 0–1024 bájt

- **Acknowledge** (visszajelzés)
  - **PID** lehetséges értékei: ACK, NAK, STALL, NYET

Forrás: STM32 USB training

PID – csomagtípus jelző

- Az első 4 bit adja meg a csomag típusát, a második 4 bit biztonsági másolat

PID type	PID name	PID[3:0]
Token	OUT	0001B
	IN	1001B
	SOF	0101B
	SETUP	1101B
Data	DATA0	0011B
	DATA1	1011B
	DATA2	0111B
	MDATA	1111B

Tranzakció elején

Tranzakció közben

HS átvitelnél

PID type	PID name	PID[3:0]
Handshake	ACK	0010B
	NAK	1010B
	STALL	1110B
	NYET	0110B
Special	PRE	1100B
	ERR	1100B
	SPLIT	1000B
	PING	0100B
	Reserved	0000B

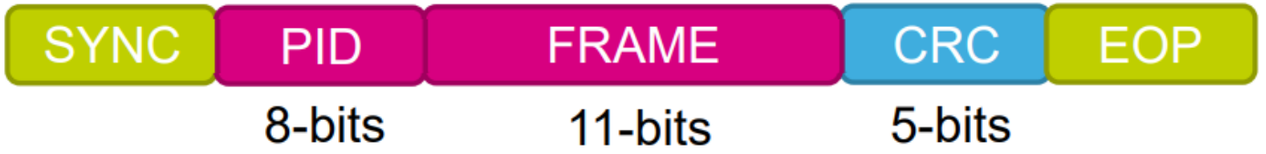
Tranzakció végén

Specifikus csomagok

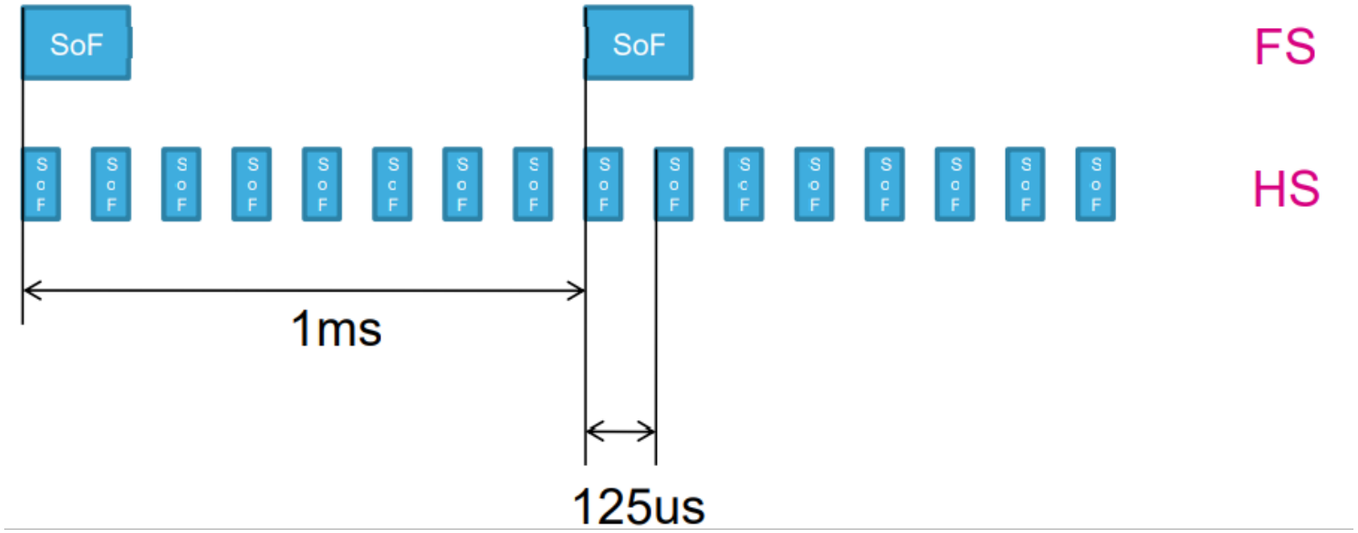
Forrás: STM32 USB training

Start Of Frame – keret kezdete jelzés

- A host egyenletes időközönként kiküldi
  - 1ms időközönként LS/FS módban
  - 125µs időközönként a HS eszközöknek



- Referencia időalapot szolgáltat az USB busz számára
- Szinkronizálásra szolgál (pl. az audio eszközöknél)
- A kristály nélküli mikrovezérlők ezzel a jellel szinkronizálják a belső órajelet
- **Suspend** parancsot jelent, ha 3 ms-ig nem érkezik SOF jelzés (pl. PC hibernáláskor)



Forrás: STM32 USB training

Tranzakciók

- A tranzakciók csomagokkal történő kommunikációból épülnek fel
  - Először egy TOKEN (jelzés) csomagot küld a host (SETUP, OUT, IN stb.)
  - Azután a DATA csomagot küld vagy fogad
  - A tranzakciót visszajelzési csomag zárja (ACK, NAK, STALL, NYET)

A tranzakció három fázisa

- Az adatátviteli irány a host nézőpontjából értendő
  - **IN** az eszköztől a host felé, **OUT** a host-ról az eszköz felé irányuló adatátvitel
- A tranzakciók alaptípusai:
  - Vezérlő (control) tranzakciók és vezérlő szekvenciák
  - Tömeges (bulk) adatátviteli tranzakció
  - Valósidejű (isochronous) tranzakció
  - Megszakítás (interrupt) tranzakció

Forrás: STM32 USB training

Tranzakció visszajelzési csomagok

- A visszajelzési csomagok az alábbiak lehetnek (vagy nincs visszajelzés)

Handshake	Description
ACK	Receiver accepts error-free data packet
NAK	Receiving device cannot accept data or transmitting device cannot send data
STALL	Endpoint is halted or a control pipe request is not supported
NYET	HS only: No response yet from receiver
Missing handshake	No handshake from receiver, error during transmission

Forrás: STM32 USB training

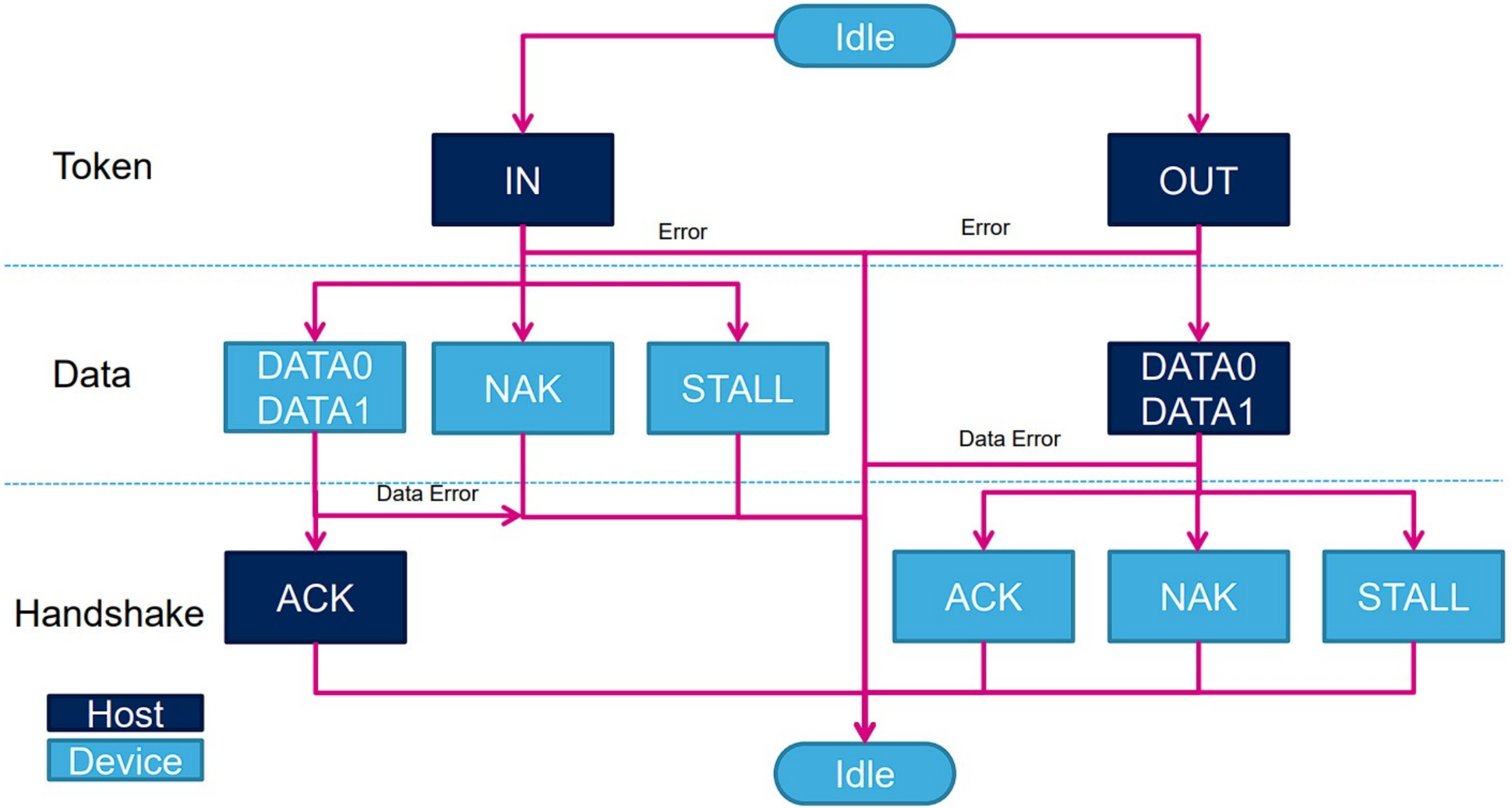
Interrupt tranzakciók

Megszakításvezérelt HID eszközök számára (egér, billentyűzet, botkormány, …)

- Periodikus adatbeolvasás
- Korlátozott méretű adatcsomagok
- Időrés: 1ms, vagy annak többszörösei
- Az 1 ms-os keret legfeljebb 90% használható interrupt vagy isochron átvitelre
- Mindegyik USB sebességnél rendelkezésre áll (LS/FS/HS)
- Hibadetektálás és javítás

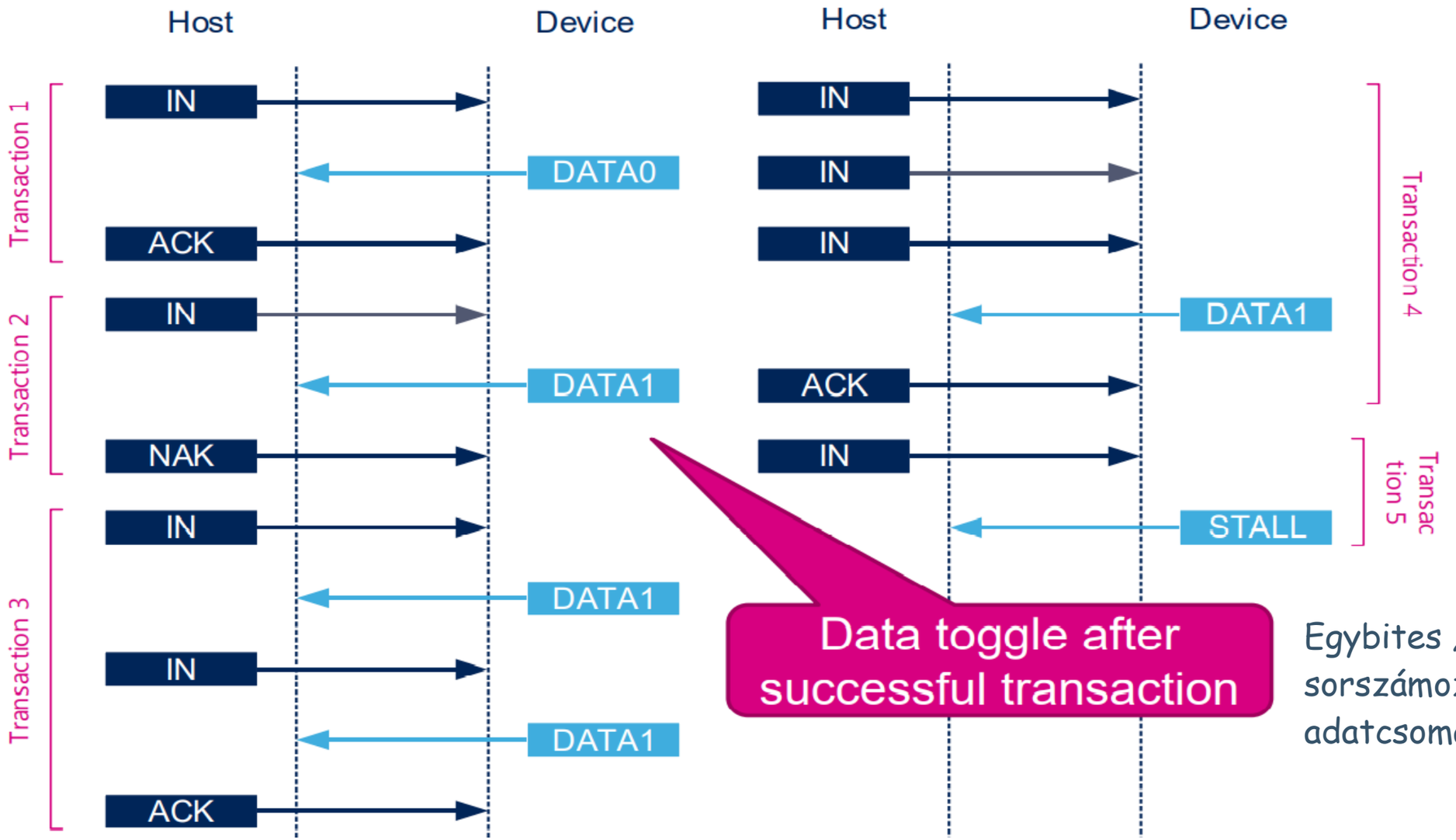
Forrás: STM32 USB training

Interrupt tranzakciók



Forrás: [STM32 USB training]

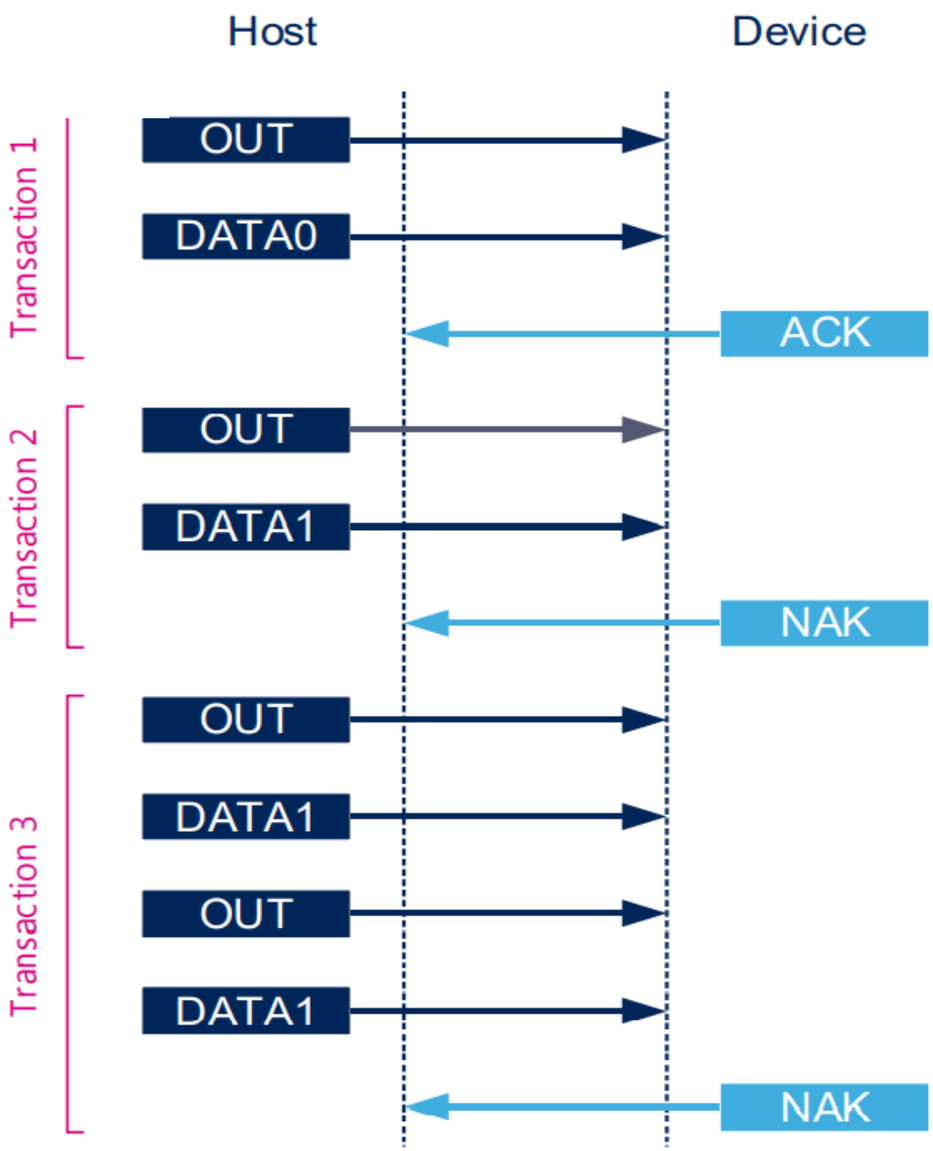
Interrupt IN tranzakció

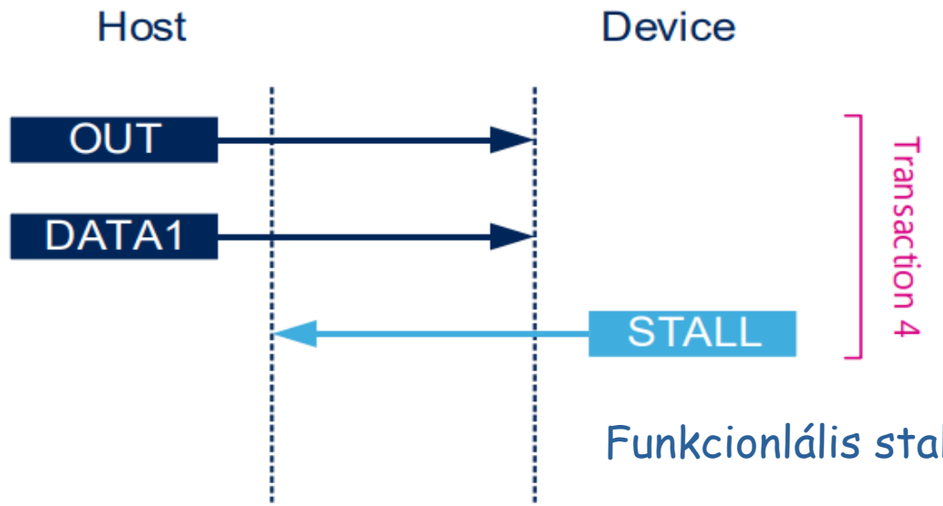


Egybites „számlálóval" sorszámozzuk az adatcsomagokat...

Forrás: [STM32 USB training]

Interrupt OUT tranzakció





Funkcionlális stall

STALL

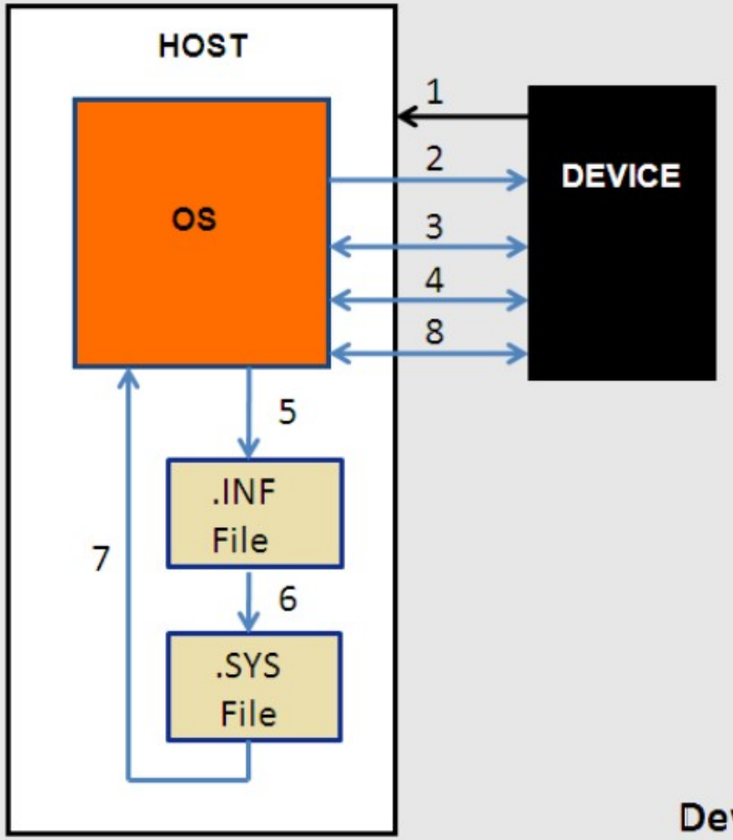
protocol stall
a funkció nem támogatott vagy nem hajtható végre

functional stall
a végpont Halt állapotban (nem tud fogadni/küldeni)

Forrás: STM32 USB training

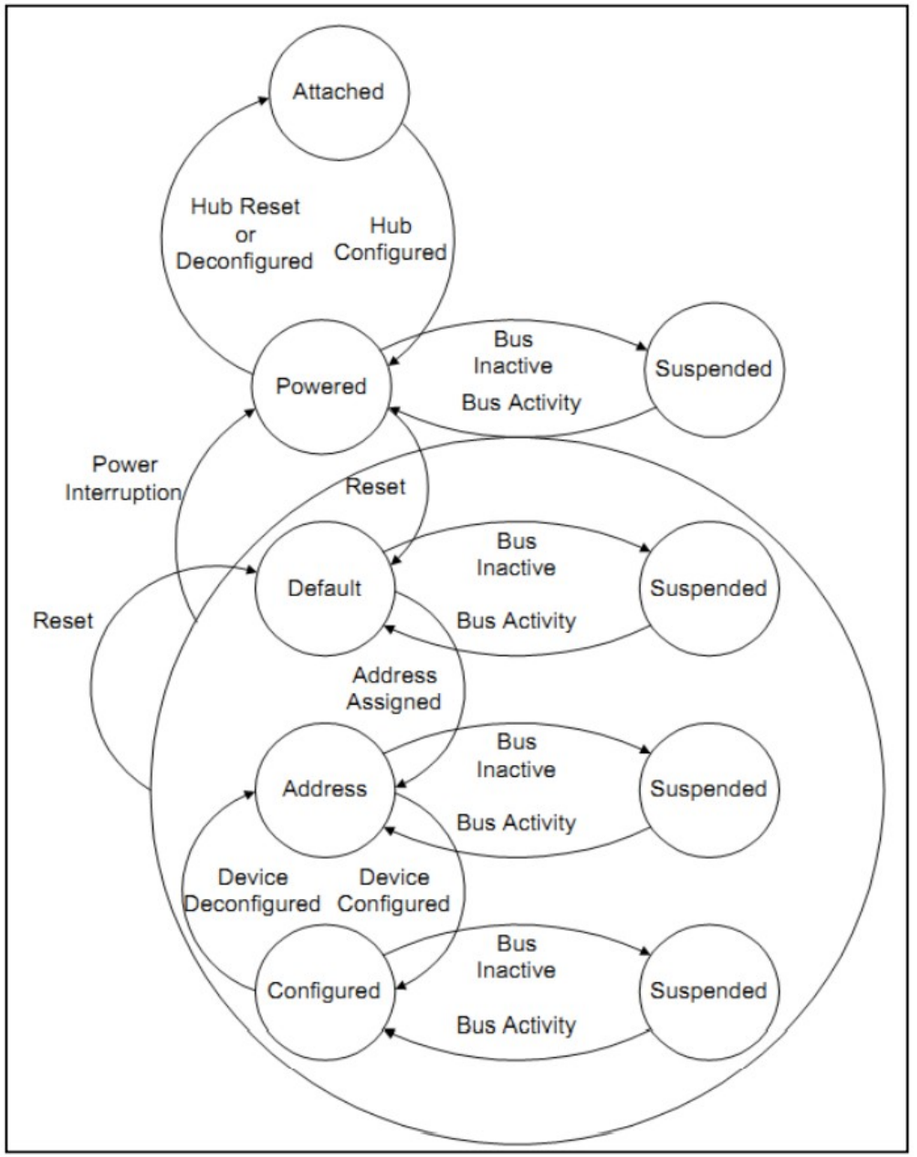
Enumeráció és állapotok

- **Enumeration** (számbavétel) az a folyamat, amikor a felcsatlakoztatott eszközt a host kikérdezi, azonosítóval látja el és konfigurálja. A „kikérdezés” az eszközön tárolt leírótáblák beolvasását és értelmezését jelenti



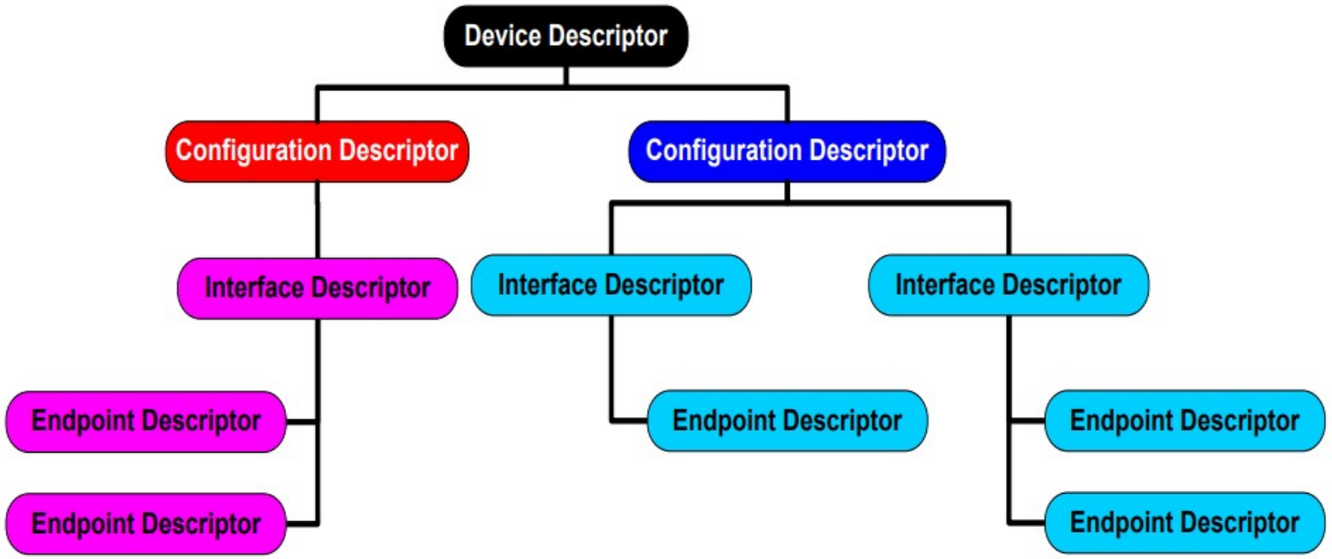
1. Device Connected to Host
2. Host resets Device then requests Device Descriptor
3. Device responds to request and Host sets new address
4. Host requests Device Descriptor using new address and Device responds
5. Host locates and reads INF file
6. INF specifies Device driver
7. Driver loaded on host
8. Host selects configuration

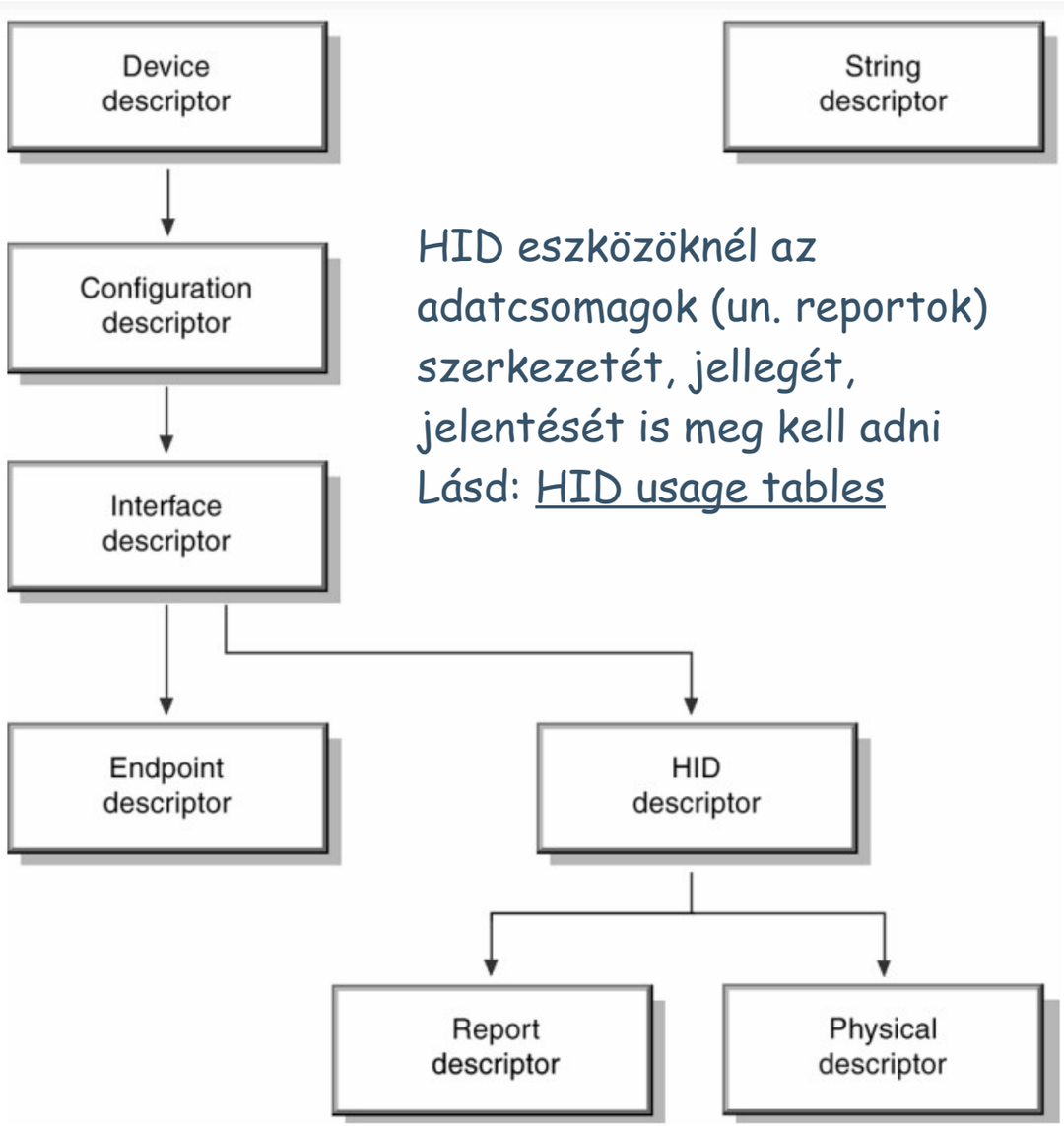
Device is configured and ready to use



USB eszközleíró táblák

- Az eszközleíró táblázatok (descriptors) struktúrált adatsorok, amelyek az eszköz jellemzőit adják meg a host által értelmezhető módon
- **Device descriptor table** – ezt olvassa elsőként a host
- **Configuration** – egyidejűleg csak egy lehet aktív
- **Interface** – több is lehet aktív (pl. HID +MSC)
- **Endpoint** – végpontok mérete, iránya (IN/OUT)





HID eszközöknél az adatcsomagok (un. reportok) szerkezetét, jellegét, jelentését is meg kell adni
Lásd: HID usage tables

Példaprogramok

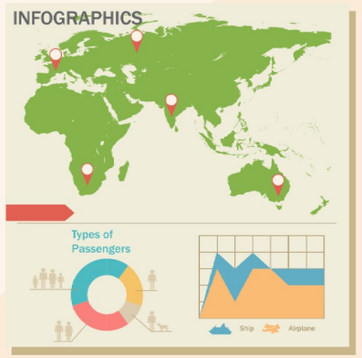
Blue pill kártya:

F103_USB_HID – USB mouse demo: az egérkurzor léptetése

Nucleo-F446RE kártya:

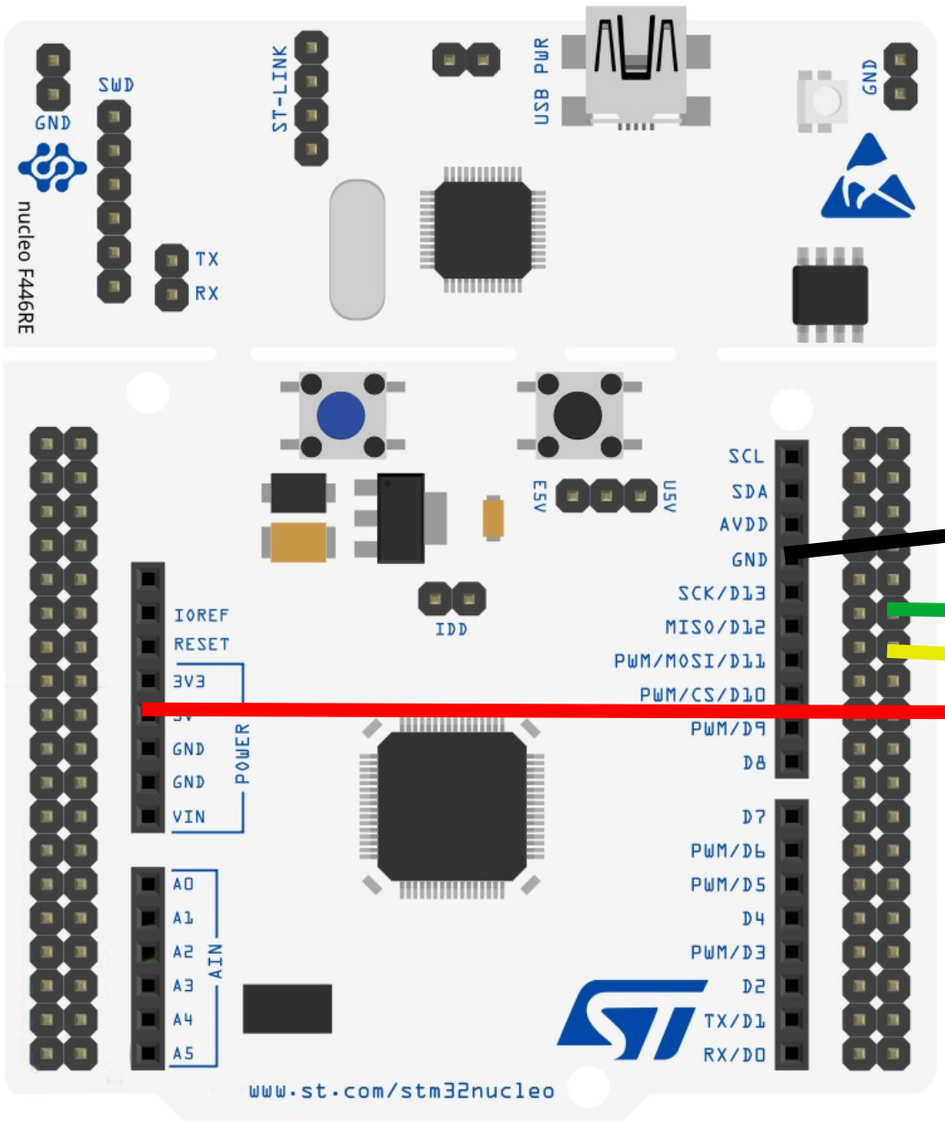
F446RE_USB_HID – kommunikáció egyedi HID eszközzel

F446RE_USB_HID_PnP – LED vezérlés és adatbeolvasás egyedi HID eszközzel

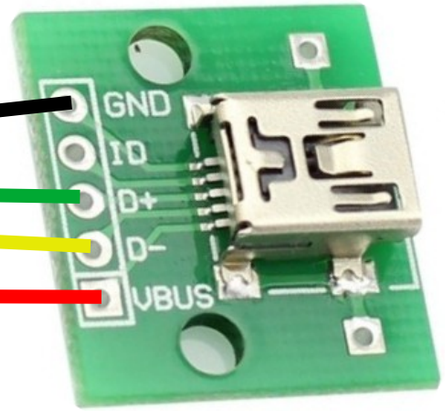


freepik.com

Bekötési vázlat

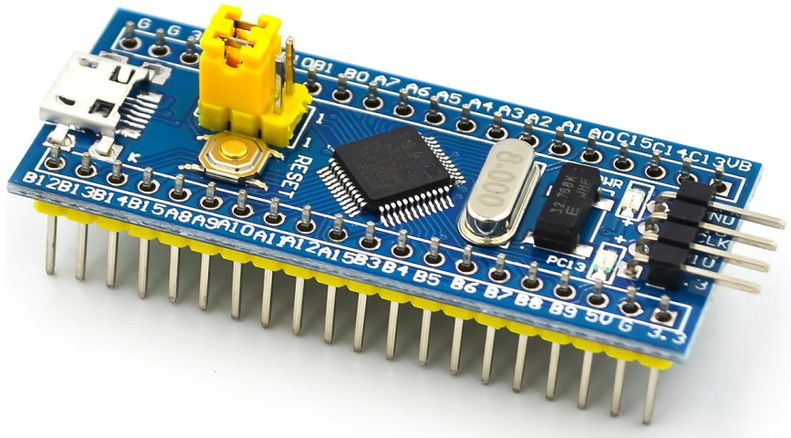


PA12 USB D+
PA11 USB D-



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

STM32F103C8: a **D+** vonalat 1,5 kΩ-mal 3.3 V-ra fel kell húzni!

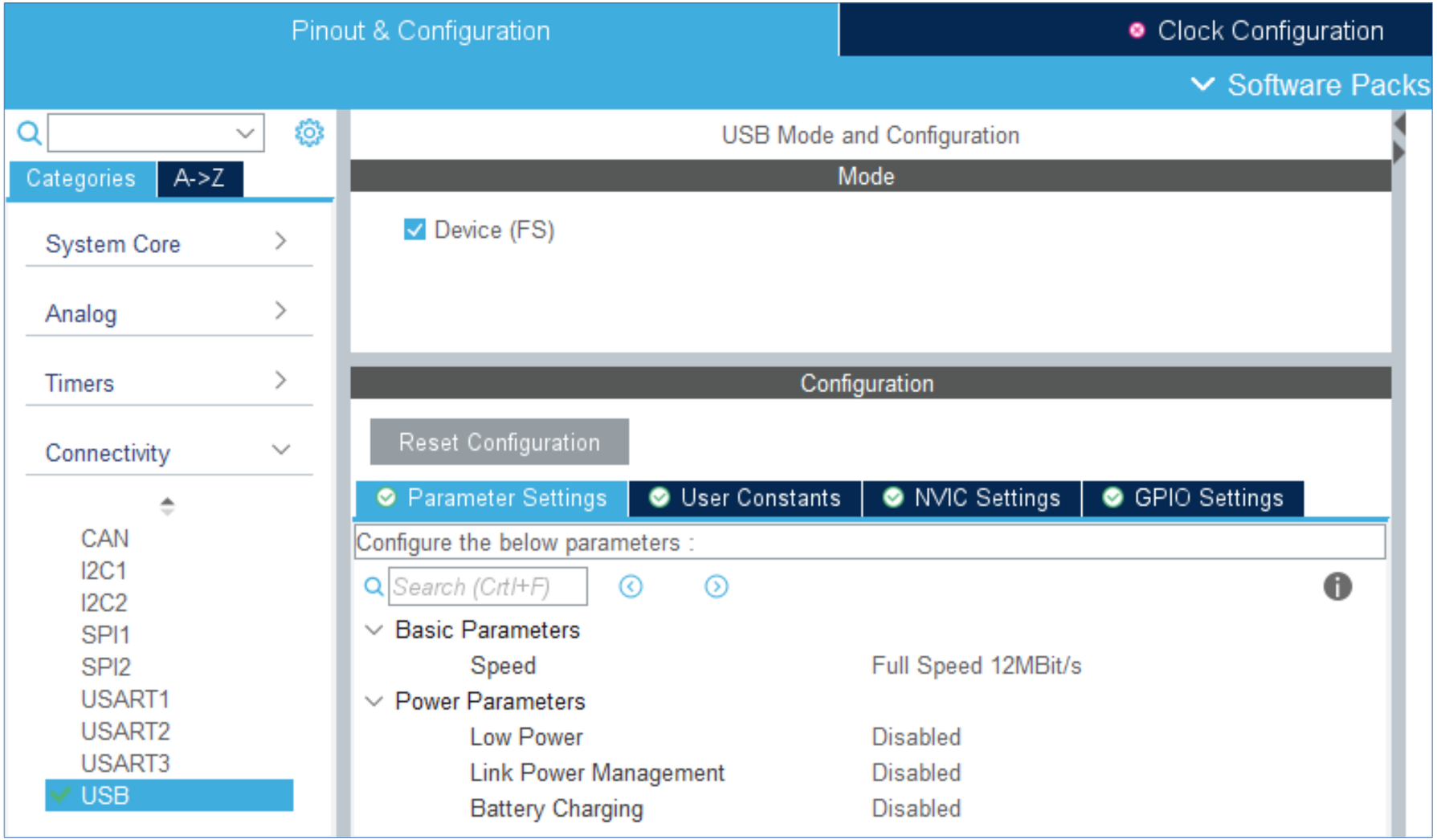


F103_USB_HID – USB egér demó

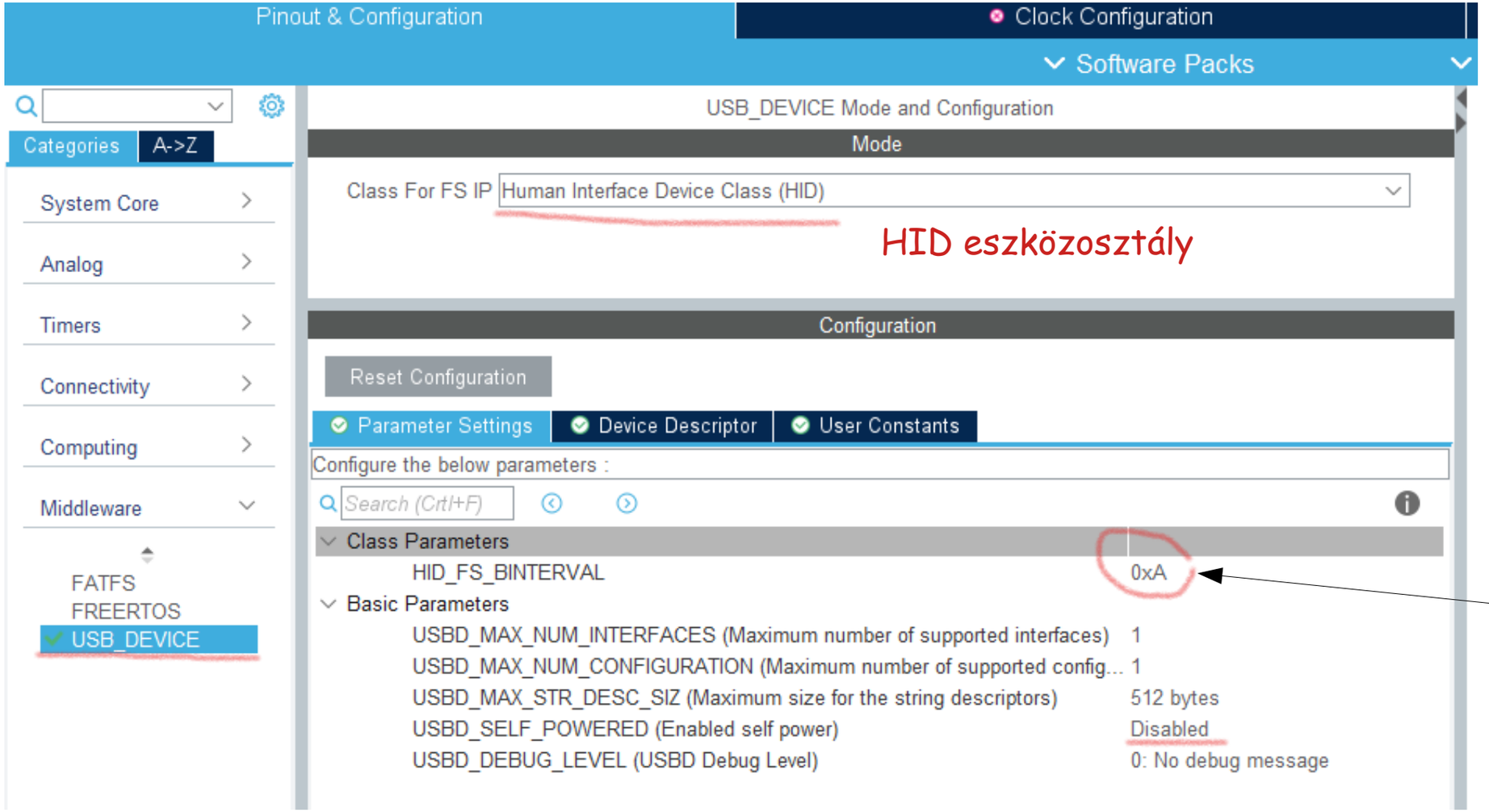
- Ebben a projektben az **STM32F103C8** mikrovezérlő (Blue pill kártya)
  - USB **HID** eszközként kapcsolódik a számítógéphez
  - USB egérként azonosítja magát
  - Másodpercenként automatikusan jobbra lépteti az egérkurzort (egy négybájtos üzenettel)
- A projektet az **STMCubeIDE** grafikus környezetben hozzuk létre és konfiguráljuk, a következő oldalakon bemutatott módon
- Az automatikusan generált kódot a **main.c** állományban néhány sorral kell csak kiegészíteni (többé-kevésbé az STM32 - custom USB HID device step by step cikk útmutatását követve)

F103_USB_HID: az USB periféria konfigurálása

- Az USB kivezetések **PA12** (D+) és **PA11** (D-), **PA12** 1.5 kΩ-mal 3.3 V-ra legyen felhúzva!



F103_USB_HID: az USB periféria könyvtár konfigurálása

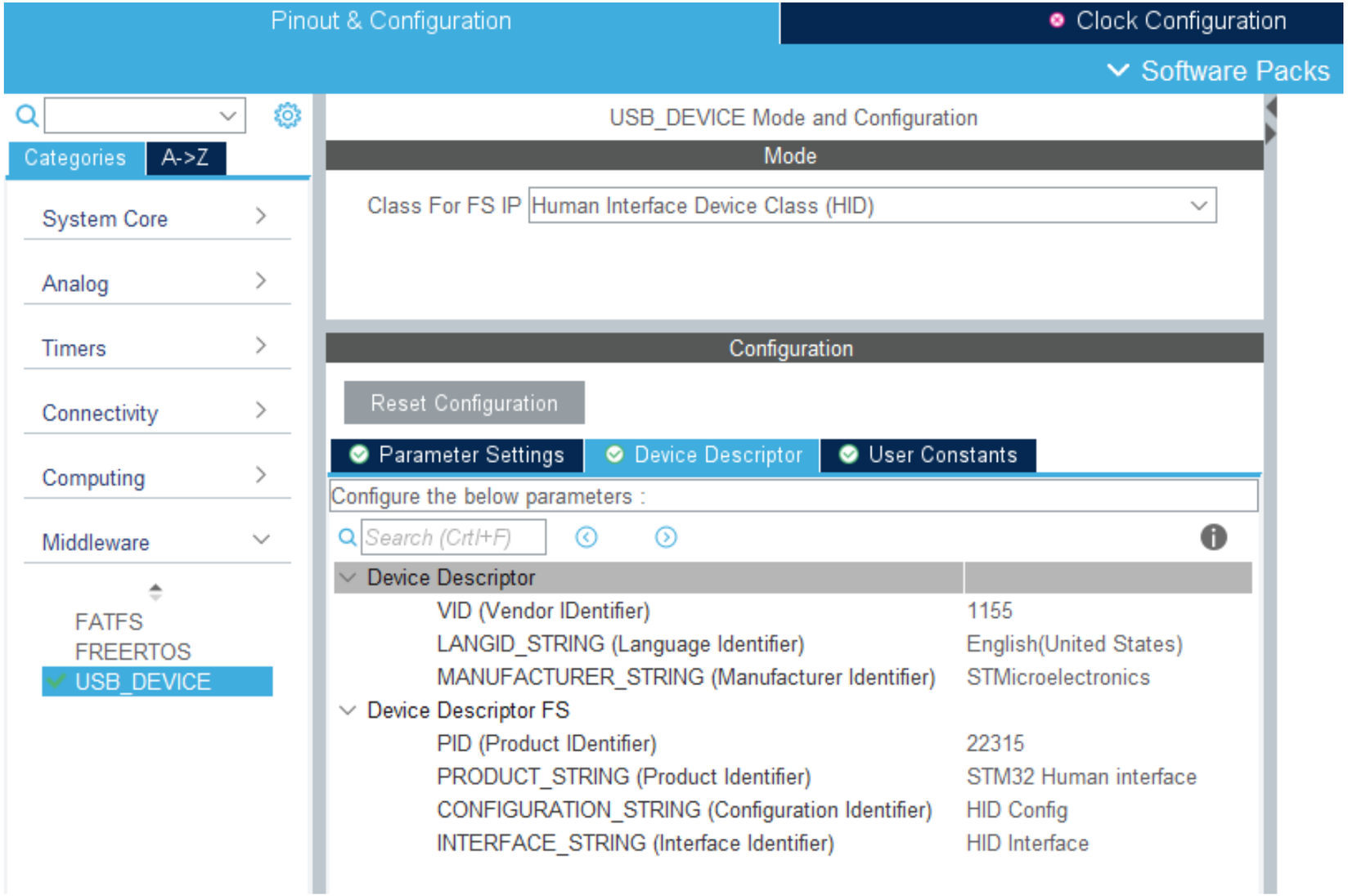


Milyen időközönként történjen lekérdezés?

Az ábrán látható alapértelmezett értékkel 10 ms-onként

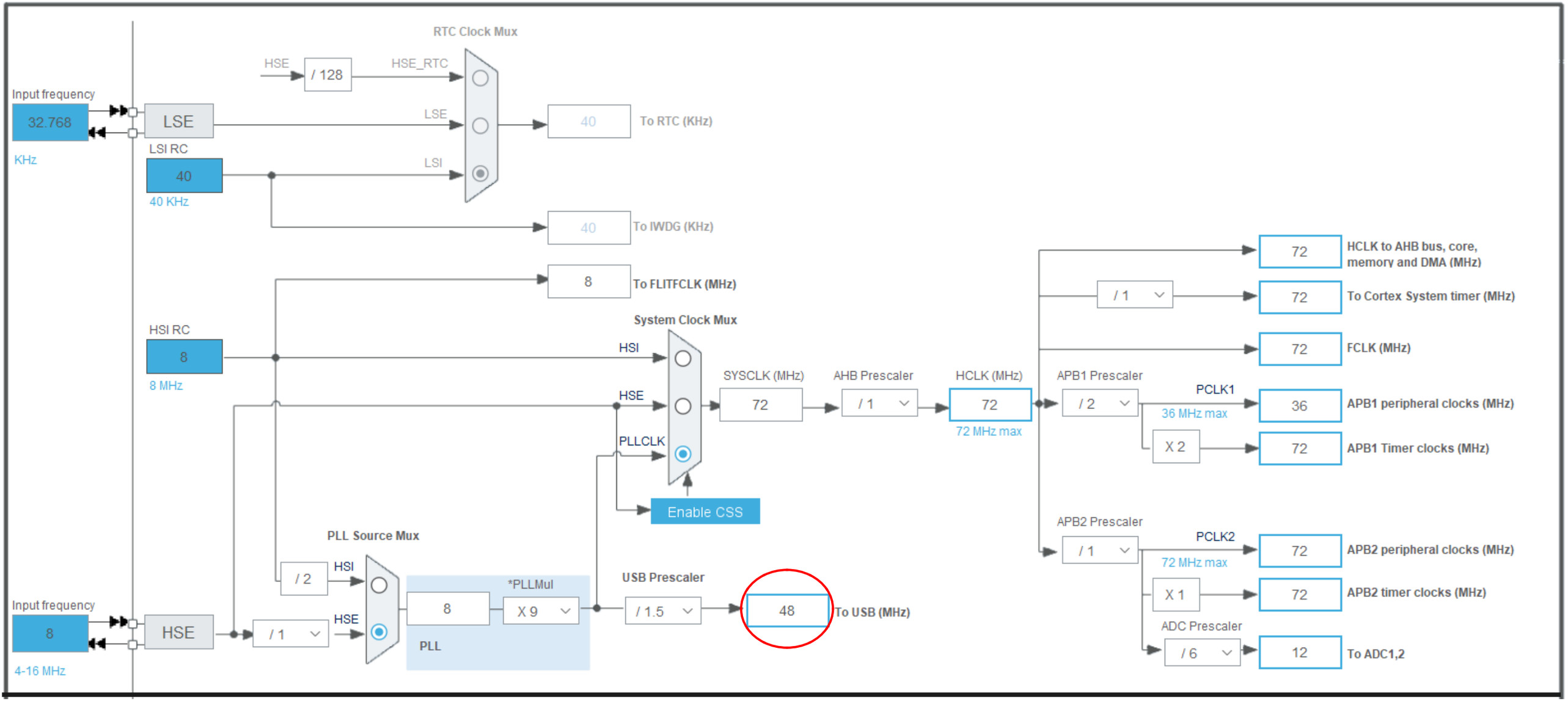
A maximális gyakoriság: 1 ms-onként

F103_USB_HID: az USB periféria könyvtár konfigurálása



Az eszközt a
VID = 0x483
PID = 0x572B
páros azonosítja

F103_USB_HID: Az órajelek konfigurálása



F103_USB_HID: main.c (részlet)

```c
#include "main.h"
#include "usb_device.h"

#include "usbd_hid.h"
extern USBD_HandleTypeDef hUsbDeviceFS;

int main(void) {
  struct mouseHID_t {          // HID report típus definiálása
    uint8_t buttons;           // egérgombok (1-1 bit)
    int8_t x;                  // x irányú elmozdulás
    int8_t y;                  // y irányú elmozdulás
    int8_t wheel;              // görgetés
  };
  struct mouseHID_t mouseHID; // HID report példányosítás és kezdőérték definiálás
  mouseHID.buttons = 0; mouseHID.x = 10; mouseHID.y = 0; mouseHID.wheel = 0;
  HAL_Init();   SystemClock_Config(); MX_GPIO_Init(); MX_USB_DEVICE_Init();

  while(1) {
    mouseHID.x = 10;       // 10 egységgel jobbra mozgatjuk a kurzort
    USBD_HID_SendReport(&hUsbDeviceFS, (uint8_t*)&mouseHID, sizeof(struct mouseHID_t));
    HAL_Delay(1000);
  }
}
```

Ne legyen **static** függvény!

Deklarálja az **USBD_HID_SendReport()** fv-t

Elérhetővé teszi a **hUsbDeviceFS** „fogantyút”

Típuskényszerítés kell, lásd **USBD_HID_SendReport()** deklarálását **usbd_hid.h**-ban!

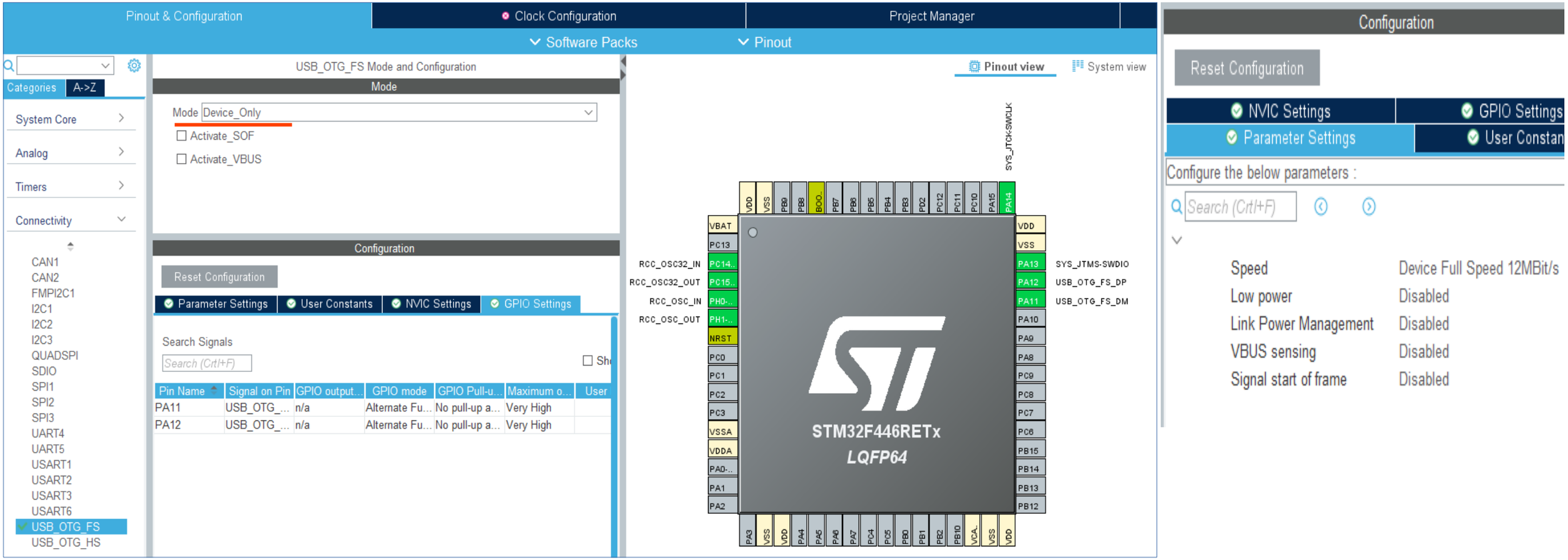
F103_USB_HID: futási eredmény

F446RE_USB_HID: „custom HID” demó

- Az **STM32F4** mikrovezérlők USB programkönyvtára különbözik az előző példában használt **STM32F1** mikrovezérlőétől: más a fájlok elnevezése, illetve a rendelkezésre álló eszköztár választéka – többek között az **egyedi (custom) HID** eszközosztály is támogatott
- Ebben a mintapéldában az STMicroelectronics: STM32 USB training egyik mintapéldáját (USB HID device - custom device lab) fogjuk követni, melynek során
  - Létrehozunk egy egyedi HID eszközt (custom HID)
  - Megmutatjuk, hogy hogyan végezhetünk kétirányú kommunikációt (IN és OUT)
  - Megmutatjuk, hogy hogyan növelhetjük meg a küldött/fogadott csomagok méretét a maximális 64 bájtra
- A létrehozott eszköz nem fog semmi hasznos dolgot végezni, csak visszatükrözi a számítógépről érkező csomagokat
- Egy picit hasznosabb egyedi HID eszközt majd a harmadik példában fogunk mutatni, ennek a példaprogramnak a kibővítésével

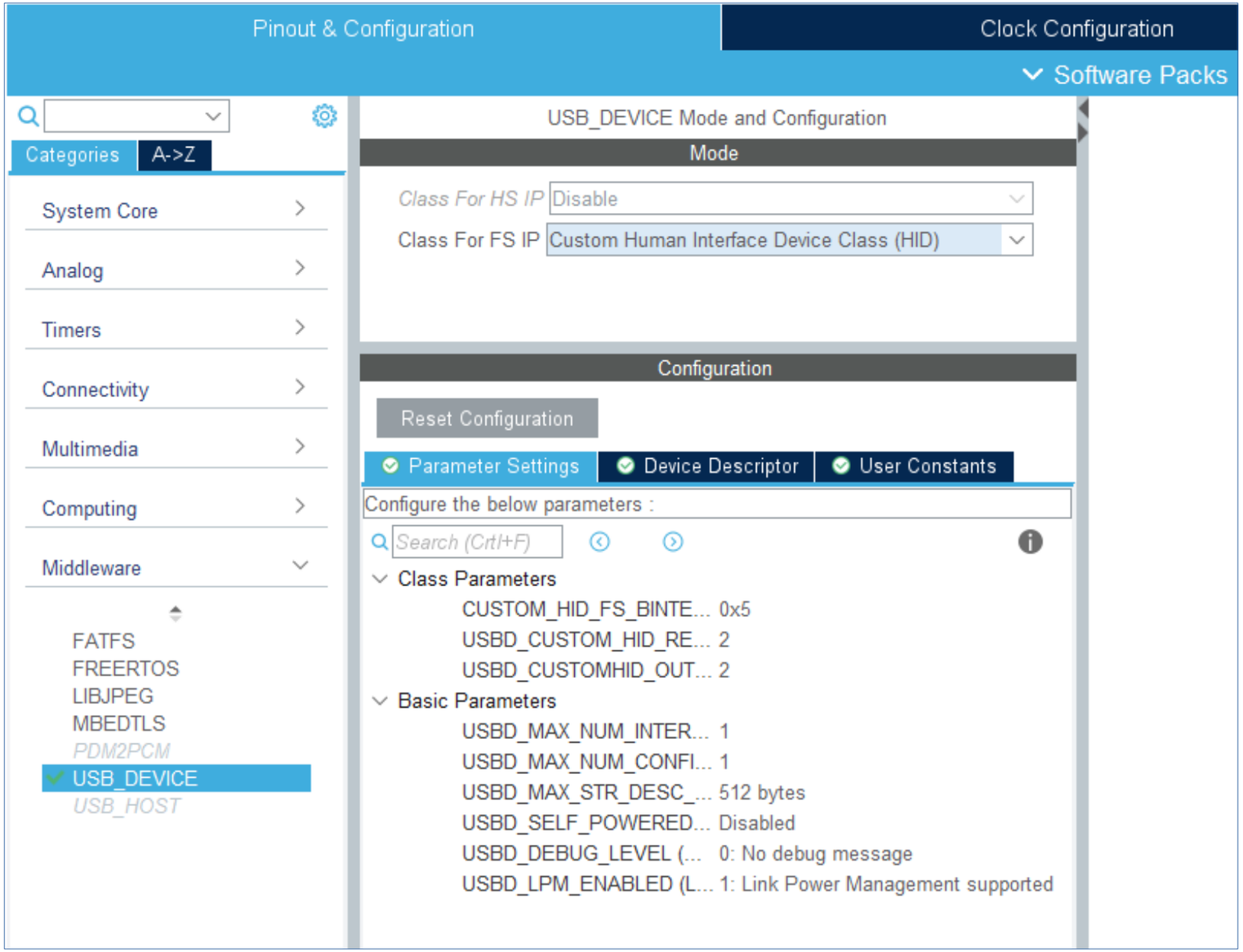
F446RE_USB_HID: USB FS konfigurálás

- Az **STM32F446RE** MCU két USB perifériával rendelkezik, most az **USB FS**-t használjuk, méghozzá „csak eszköz” (device only) módban.
- A kivezetések itt is **PA12** (D+) és **PA11** (D-), de nem kell külső felhúzás



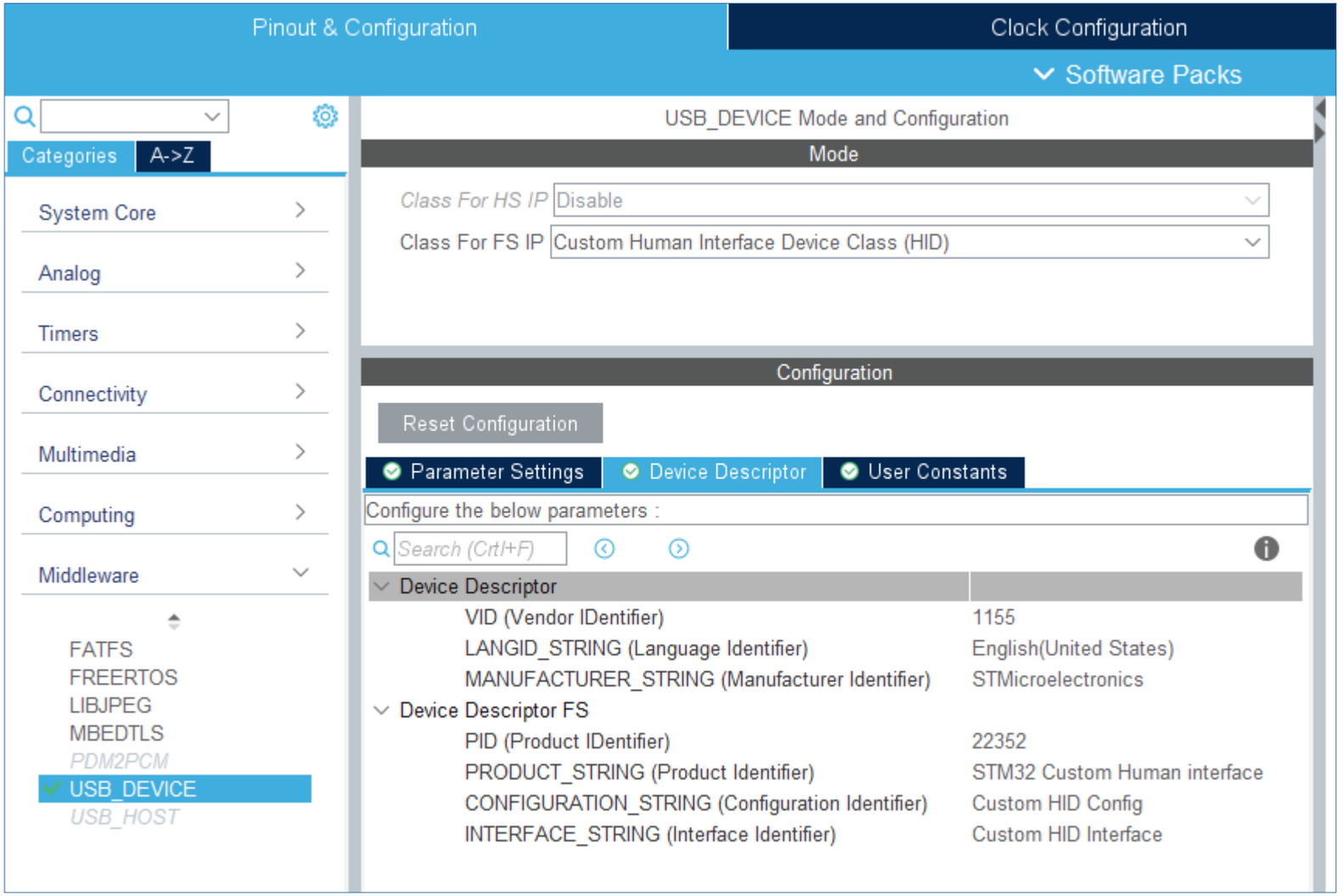
F446RE_USB_HID: USB library konfigurálás

- Az **STM32F446RE** MCU-nál a **Middleware** szekcióban USB_HOST és USB_DEVICE programcsomag is található, melyek közül most az utóbbira lesz szükségünk
- A **Custom HID** osztályt válasszuk ki!
- A default paraméter értéket hagyhatjuk (majd úgyis felülírjuk, amit kell...)

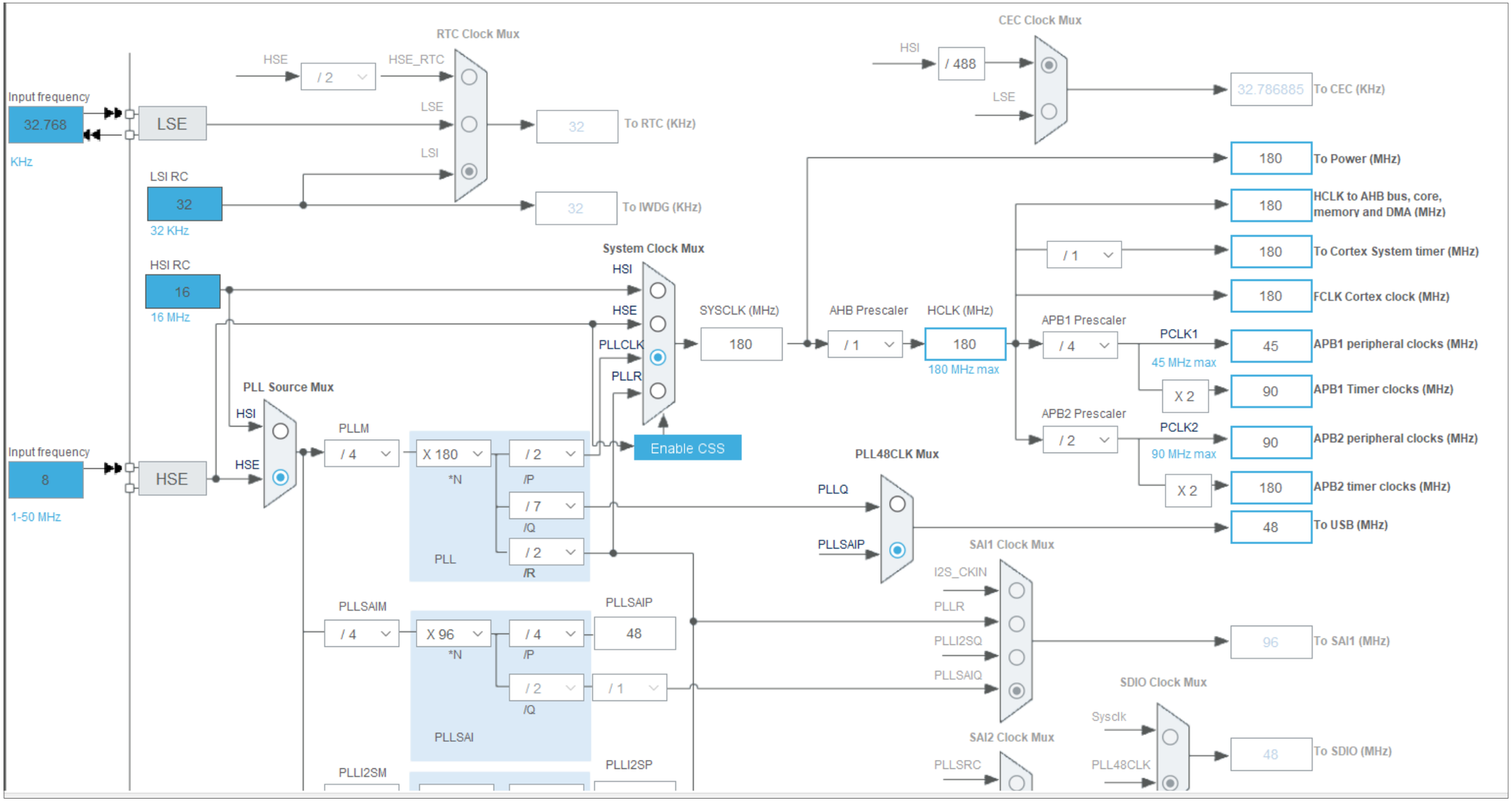


F446RE_USB_proba: USB_DEVICE konfigurálás

- A képen látható **VID/PID** pároshoz (VID = 0x483, PID = 0x5750) nem muszáj ragaszkodni, de a PC-n futó mintaalkalmazások ezt az azonosító párost feltételezik majd…



F448RE_USB_HID: órajelek konfigurálása



Maximális HCLK elérésre két lehetőség nyílik:

1. HCLK = 168 MHz és PLLQ = 48 MHz (336 Mhz/7)

2: HCLK = 180 MHz, PLLSAIP = 48 MHz (2 MHzx96/4)

F446RE_USB_HID: a generált kód módosítása

- Az automatikus kódgenerálást követően az **STM32 USB training - USB HID device custom device lab** útmutatását követve, módosítanunk kell néhány forrásállományt
- **usbd_conf.h:** Növeljük meg OUT report és a HID report deszkriptor méretét!

```
#define USBD_CUSTOMHID_OUTREPORT_BUF_SIZE    64U
#define USBD_CUSTOM_HID_REPORT_DESC_SIZE     33U
```

- **usbd_customhid.h:** növeljük a végpontok méretét és módosítsuk az USBD custom HID struktúrát

```
#define CUSTOM_HID_EPIN_SIZE                 0x40
#define CUSTOM_HID_EPOUT_SIZE                0x40


typedef struct _USBD_CUSTOM_HID_Itf
{
  uint8_t                   *pReport;
  int8_t (* Init)           (void);
  int8_t (* DeInit)         (void);
  int8_t (* OutEvent)       (uint8_t* );

}USBD_CUSTOM_HID_ItfTypeDef;
```

F446RE_USB_HID: a generált kód módosítása

- **usbd_customhid.c**: szükség esetén növeljük vagy csökkentsük **bInterval** értékét (bár ezt a grafikus konfigurálásnál is megtehetjük)

```
/********************* Descriptor of Custom HID endpoints ********************/
  /* 27 */
  0x07,                                                   /* bLength: Endpoint Descriptor size */
  USB_DESC_TYPE_ENDPOINT,                                 /* bDescriptorType: */

  CUSTOM_HID_EPIN_ADDR,                                   /* bEndpointAddress: Endpoint Address (IN) */
  0x03,                                                   /* bmAttributes: Interrupt endpoint */
  CUSTOM_HID_EPIN_SIZE,                                   /* wMaxPacketSize: 2 Byte max */
  0x00,
  CUSTOM_HID_FS_BINTERVAL,                                /* bInterval: Polling Interval */
  /* 34 */

  0x07,                                                   /* bLength: Endpoint Descriptor size */
  USB_DESC_TYPE_ENDPOINT,                                 /* bDescriptorType: */
  CUSTOM_HID_EPOUT_ADDR,                                  /* bEndpointAddress: Endpoint Address (OUT) */
  0x03,                                                   /* bmAttributes: Interrupt endpoint */
  CUSTOM_HID_EPOUT_SIZE,                                  /* wMaxPacketSize: 2 Bytes max  */
  0x00,
  CUSTOM_HID_FS_BINTERVAL,                                /* bInterval: Polling Interval */
  /* 41 */
};
```

F446RE_USB_HID: a generált kód módosítása

- **usbd_customhid.c:** cseréljük le az **OUT** eseményeket kezelő függvényeket erre! (**OUT** esemény az, amikor a Host küld adatcsomagot az eszköz EPOUT-jának)

```
static uint8_t USBD_CUSTOM_HID_DataOut (USBD_HandleTypeDef *pdev, uint8_t epnum) {
  USBD_CUSTOM_HID_HandleTypeDef *hhid = (USBD_CUSTOM_HID_HandleTypeDef*)pdev->pClassData;
  ((USBD_CUSTOM_HID_ItfTypeDef *)pdev->pUserData)->OutEvent(hhid->Report_buf);
  USBD_LL_PrepareReceive(pdev, CUSTOM_HID_EPOUT_ADDR , hhid->Report_buf,
                                USBD_CUSTOMHID_OUTREPORT_BUF_SIZE);
  return USBD_OK;
}

uint8_t USBD_CUSTOM_HID_EP0_RxReady(USBD_HandleTypeDef *pdev) {
  USBD_CUSTOM_HID_HandleTypeDef *hhid = (USBD_CUSTOM_HID_HandleTypeDef*)pdev->pClassData;
  if (hhid->IsReportAvailable == 1) {
    ((USBD_CUSTOM_HID_ItfTypeDef *)pdev->pUserData)->OutEvent(hhid->Report_buf);
    hhid->IsReportAvailable = 0;
  }
  return USBD_OK;
}
```

F446RE_USB_HID: a generált kód módosítása

- **usbd_custom_hid_if.c:** adjunk hozzá egy 64 bájtos buffert az üzenetek kezelésére

```
/* USER CODE BEGIN PRIVATE_DEFINES */
uint8_t buffer[0x40];
/* USER CODE END PRIVATE_DEFINES */
```

- **usbd_custom_hid_if.c: módosítsuk a CUSTOM_HID_OutEvent_FS()** függvény (ezt hívják meg az előző oldalon módosított függvények) deklarálását és definiálását!

```
static int8_t CUSTOM_HID_OutEvent_FS (uint8_t* state);


static int8_t CUSTOM_HID_OutEvent_FS  (uint8_t* state)
{
  /* USER CODE BEGIN 6 */
  memcpy(buffer,state,0x40);
  USBD_CUSTOM_HID_SendReport(&hUsbDeviceFS,(uint8_t*)buffer,0x40);
  return (0);
  /* USER CODE END 6 */
}
```

a HID_SendReport függvénynév már ismerős lehet az előző programból...

Átmásoljuk és visszaküldjük az üzenetcsomagot

Itt lehetne valami hasznosabb dolgot is kezdeni az adatokkal

F446RE_USB_HID: a generált kód módosítása

- **usbd_custom_hid_if.c:** készítsük el az egyedi HID eszköz **Report Descriptor** tábláját

```
__ALIGN_BEGIN static uint8_t CUSTOM_HID_ReportDesc_FS[USBD_CUSTOM_HID_REPORT_DESC_SIZE] __ALIGN_END =
{
  /* USER CODE BEGIN 0 */
  0x06, 0x00, 0xff, // Usage Page(Undefined )
  0x09, 0x01,       // USAGE (Undefined)
  0xa1, 0x01,       // COLLECTION (Application)
  0x15, 0x00,       // LOGICAL_MINIMUM (0)
  0x26, 0xff, 0x00, // LOGICAL_MAXIMUM (255)
  0x75, 0x08,       // REPORT_SIZE (8)      <-- 8-bites adategységekben számolunk (gyk.: bájt)
  0x95, 0x40,       // REPORT_COUNT (64)    <-- a report mérete 64 bájt
  0x09, 0x01,       // USAGE (Undefined)
  0x81, 0x02,       // INPUT (Data,Var,Abs) <-- ez az IN report, amit a Host felé küldünk
  0x95, 0x40,       // REPORT_COUNT (64)    <-- ennek a report-nak a mérete is 64 bájt
  0x09, 0x01,       // USAGE (Undefined)
  0x91, 0x02,       // OUTPUT (Data,Var,Abs)<-- ez az OUT report, amit a Host küld
  0x09, 0x01,       // USAGE (Undefined)
  0xb1, 0x02,       // FEATURE (Data,Var,Abs)
  /* USER CODE END 0 */
  0xC0        /* END_COLLECTION */
};
```

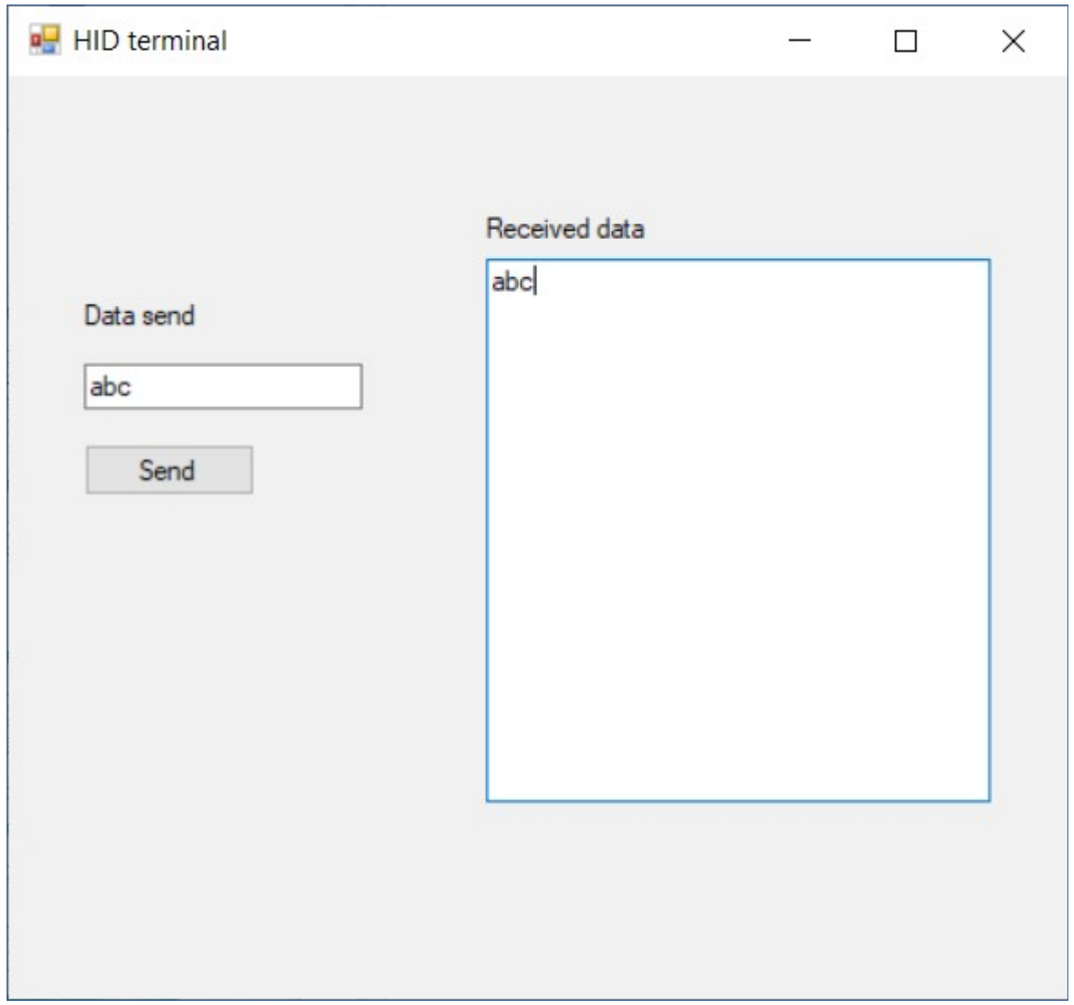
F446RE_USB_HID

- Ha idáig eljutottunk, akkor a módosított projektet lefordíthatjuk és kipróbálhatjuk
- A **main.c** állományban most kivételesen nem módosítottunk semmit, az USB kommunikáció nem a főprogramban, hanem megszakítás szinten zajlik
- A kipróbáláshoz szükségünk lesz a **hid_terminal.exe** programra, amit a kész projekttel együtt az előadás mellékletében is közreadunk, de az **STM32 USB Training** letölthető anyagaiban is megtalálható forráskódjával együtt:
  - STM32 USB Training Letölthető anyagok (ZIP fájl)
- A **hid_terminal** alkalmazás C# forráskódja a Visual Studio Community 2019 segítségével lefordítható (erre pl. akkor lehet szükség, ha módosítani akarjuk)
- A **hid_terminal** alkalmazáson kívül természetesen az általános HID kapcsolat tesztelésre készült programokkal is próbálkozhatunk a működés ellenőrzésére:
  - **Hidapi** / testgui.exe
  - **SimpleHidWrite**
  - **Device Monitoring Studio** (14 napost teszt mód) általános adatforgalom analizátor
  - **WireShark** (ingyenes) általános adatforgalom analizátor

F446RE_USB_HID és a HID terminal futtatása

- A baloldali beírósávba begépelt karaktereket a **Send** gomb megnyomásakor visszatükrözi az eszköz (ennek így sok értelme nincs, de látjuk, hogy működik)
- Értelmesebb működéshez a firmware-t tovább kell fejleszteni, hogy
  - a beérkező adat valamit vezéreljen
  - a visszaküldött adat pedig mért adat vagy egyéb hasznos információ legyen

A következő példában ezt fogjuk tenni ...



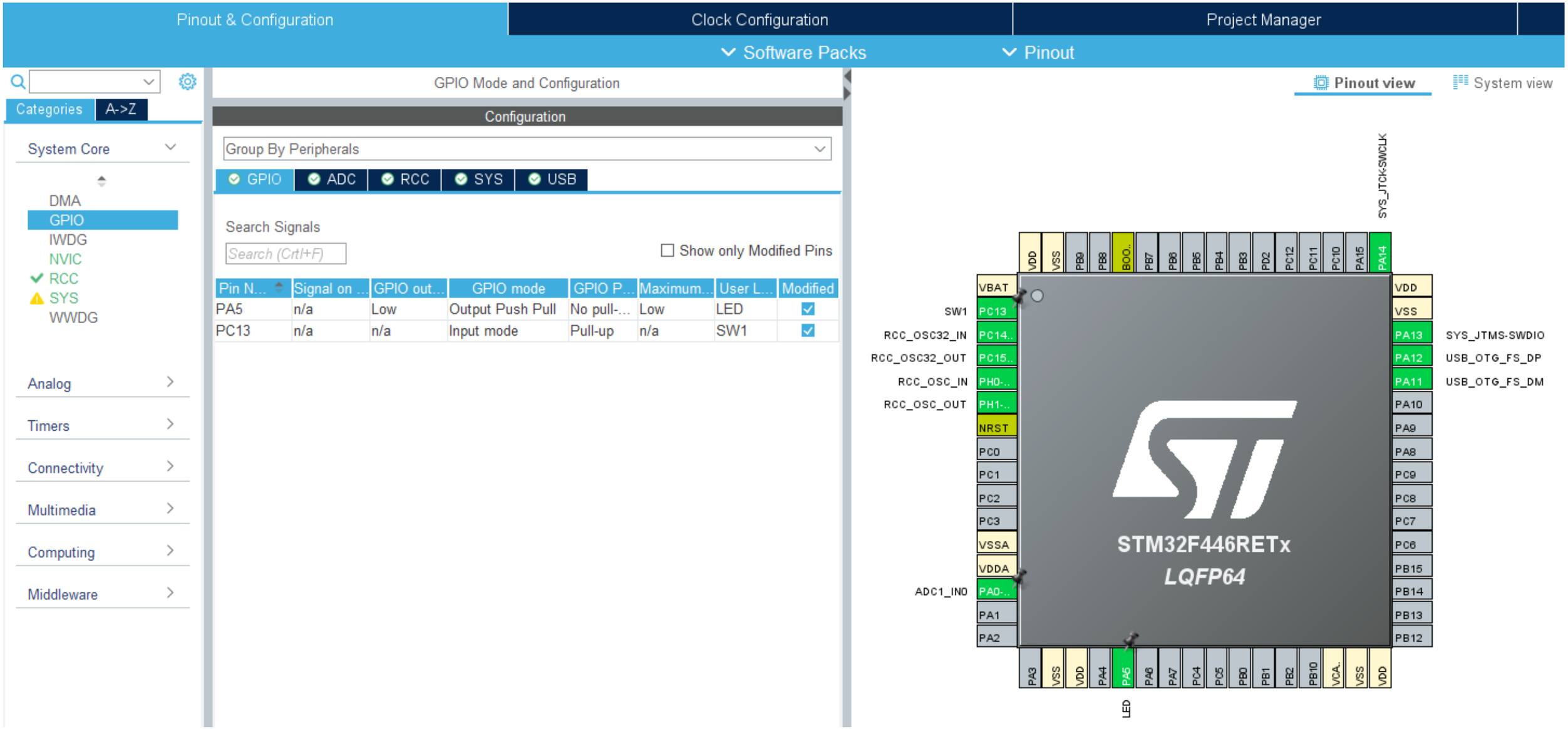
F446RE_USB_HID_PnP

- Az előző projekt továbbfejlesztésével egy olyan Plug and Play eszközt készítünk, ami a Host felől érkező 64 bájtos üzenetek első bájtját az alábbi táblázat szerint értelmezi

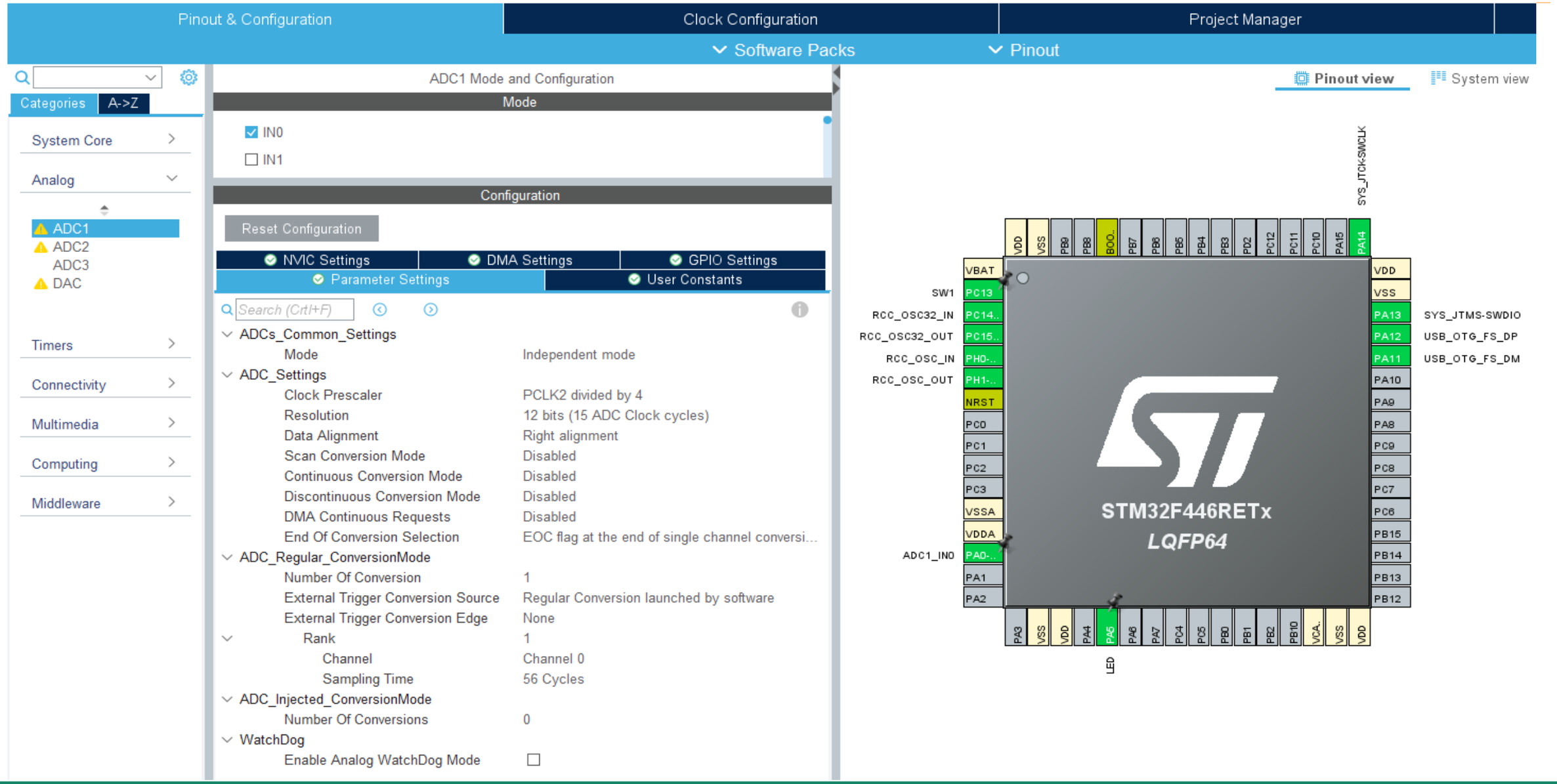
Parancs	Válasz	Szöveges leírás
0x80	nincs	LED állapot átbillentése (LED a PA5 kivezetésre van kötve)
0x81	0x81 1/0	Nyomógomb állapotának lekérdezése (SW1 a PC13 kivezetésre van kötve)
0x37	0x37 LSB MSB	ADC IN0 csatorna értékének lekérdezése (IN0 a PA0 kivezetésre van kötve)
egyéb	0xFF	Nem értelmezett 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) HID PnP demo mintaalkalmazását „vettük kölcsön” és adaptáltuk (Vid/Pid csere, ProgressBar1 maximuma 1024 helyett 4096)

A F446RE_USB_HID_PnP: GPIO konfigurálás



A F446RE_USB_HID_PnP: ADC konfigurálás



F446RE_USB_HID_PnP

- A projekt generálásának többi lépése megegyezik az előző projektével, beleértve az **usbd_conf.h, usbd_customhid.h, usbd_customhid.c és usbd_custom_hid_if.c** állományok utólagos (az automatikus kódgenerálás utáni) módosításokat is
- **A következő lépés** a beérkező és kiküldendő üzenetcsomagok szétválasztása, ehhez két külön buffert alakítunk ki a **main.c**-ben, s az **usbd_custom_hid_if.c** állományban csak hivatkozunk rá
- Az **RX_buffer_flag** jelző 1-be állításával jelezzük a főprogramnak, hogy üzenet érkezett, a főprogram pedig majd törli a jelzőt

usbd_custom_hid_if.c (részlet)

```
/* USER CODE BEGIN PV */

extern uint8_t TX_buffer[0x40];
extern uint8_t RX_buffer[0x40];
extern uint8_t RX_buffer_flag;

/* USER CODE END PV */
```

Itt a bufferek elnevezése nem a Host, hanem az alkalmazásunk szempontja szerint történt, tehát TX_buffer-ből küldjük az adatot (az IN kérésre), RX_bufferbe fogadjuk az adatot (az OUT csomagokat)

F446RE_USB_HID_PnP

- Az **usbd_custom_hid_if.c** állományban módosítsuk a `CUSTOM_HID_OutEvent_FS` függvényt

```
static int8_t CUSTOM_HID_OutEvent_FS (uint8_t* state) {
  /* USER CODE BEGIN 6 */
    memcpy(RX_buffer,state,0x40);
    RX_buffer_flag = 1;
    return (USBD_OK);
  /* USER CODE END 6 */
}
```

A beérkező üzenetcsomagot átmásoljuk RX_buffer-be és 1-be állítjuk a jelzőt

- Szintén az **usbd_custom_hid_if.c** állományban „kiszabadítjuk” a kommentből az `int8_t USBD_CUSTOM_HID_SendReport_FS(uint8_t *report, uint16_t len)` függvényt, ezzel tudunk majd üzenetet küldeni a főprogramból

```
int8_t USBD_CUSTOM_HID_SendReport_FS(uint8_t *report, uint16_t len) {
  return USBD_CUSTOM_HID_SendReport(&hUsbDeviceFS, report, len);
}
```

- Deklaráljuk az `USBD_CUSTOM_HID_SendReport_FS()` függvényt az **usbd_custom_hid_if.h** állományban, hogy ténylegesen elérhető legyen

F446RE_USB_HID_PnP: main.c (részlet)

- A főprogramban definiáltuk a ki- és bemeneti buffereket (hogy kényelmesen elérjük)
- Az `usbd_custom_hid_if.h` becsatolása révén pedig elérjük az `USBD_CUSTOM_HID_SendReport_FS` fv-t

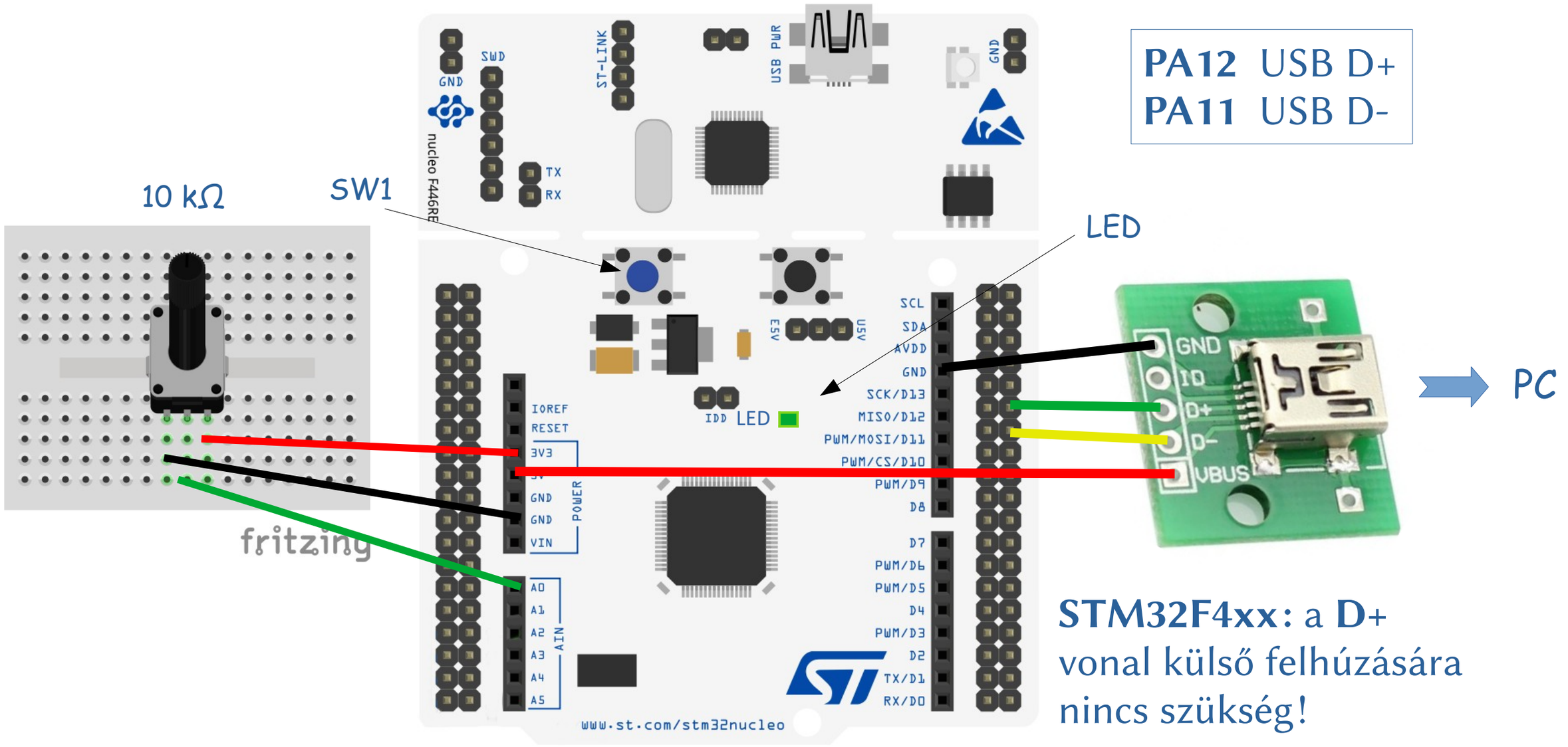
```
#include "main.h"
#include "usb_device.h"
#include "usbd_custom_hid_if.h"
ADC_HandleTypeDef hadc1;

uint8_t TX_buffer[0x40];
uint8_t RX_buffer[0x40];
uint8_t RX_buffer_flag = 0;
uint16_t val = 0;

int main(void) {
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_USB_DEVICE_Init();
  MX_ADC1_Init();
```

- Az alkalmazás az **ADC1**-et használja, az **IN0** bemeneten (**PA0**)
- A ***val*** változó szolgál az ADC-ből kiolvasott adat tárolására

F446RE_USB_HID_PnP: main.c (részlet)

```
while (1)  {
    if(RX_buffer_flag) {
      switch(RX_buffer[0]) {
        case 0x80: //Toggle LED state
             HAL_GPIO_TogglePin(LED_GPIO_Port, LED_Pin);
            break;
        case 0x81: //Get push button state
            TX_buffer[0] = RX_buffer[0];
            TX_buffer[1] = HAL_GPIO_ReadPin(SW1_GPIO_Port, SW1_Pin);
            USBD_CUSTOM_HID_SendReport_FS(TX_buffer,64);  // Send answer to host
            break;
        case 0x37: //Read POT command.
            TX_buffer[0] = RX_buffer[0];
            HAL_ADC_Start(&hadc1);                        // Start ADC1 conversion
            HAL_ADC_PollForConversion(&hadc1, 100);       // Wait for EOC
            val = HAL_ADC_GetValue(&hadc1);
            TX_buffer[1] = val & 0xFF;                    // Measured value LSB
            TX_buffer[2] = val >> 8;                      // Measured value MSB
            USBD_CUSTOM_HID_SendReport_FS(TX_buffer,64);  // Send answer to host
            break;
        default:  { TX_buffer[0] =0xFF;                   // Invalid command
            USBD_CUSTOM_HID_SendReport_FS(TX_buffer,64);  // Send answer to host
        }
      }
      RX_buffer_flag = 0;                                 // Invalidate receive buffer
    }
  }
```

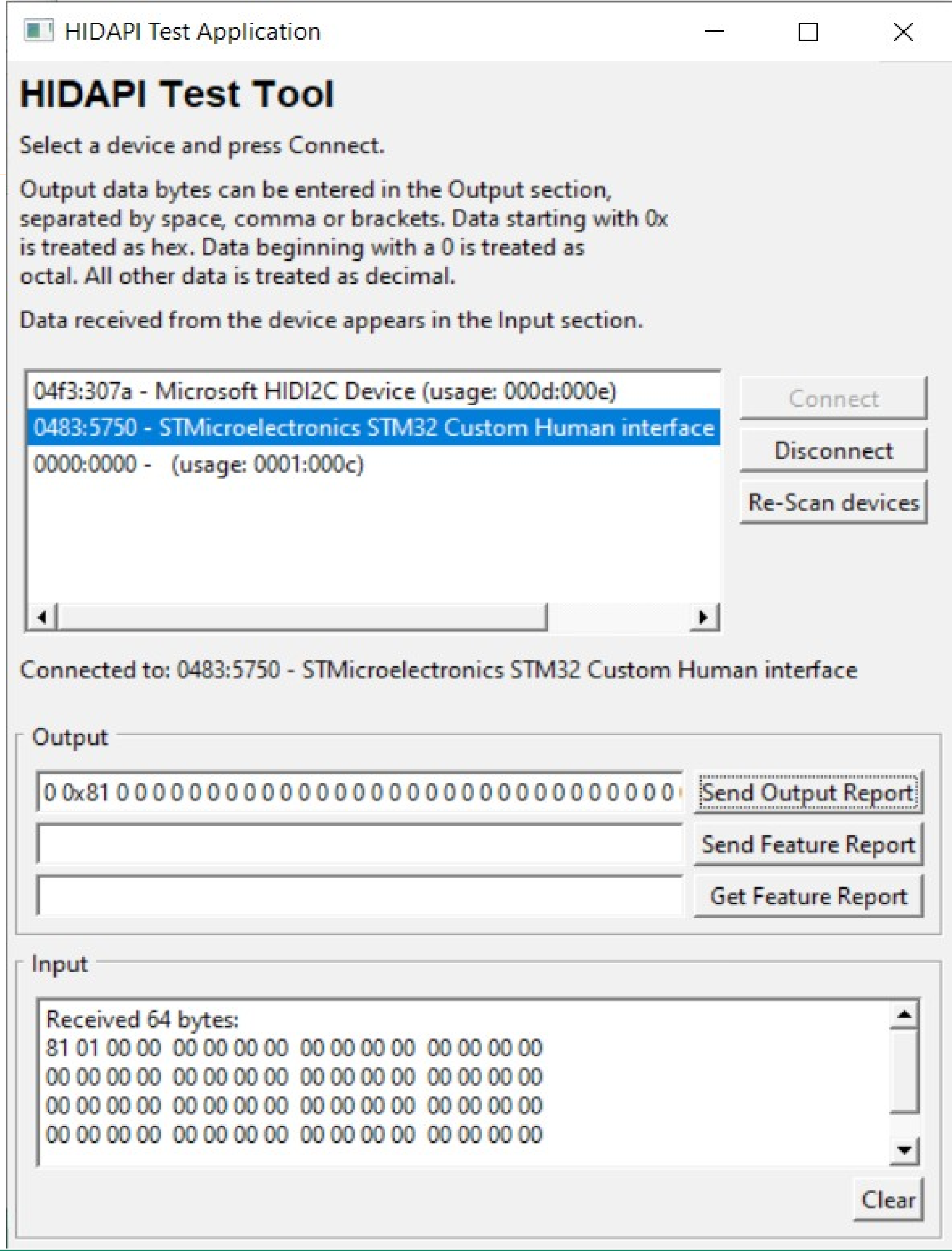
Bekötési vázlat



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

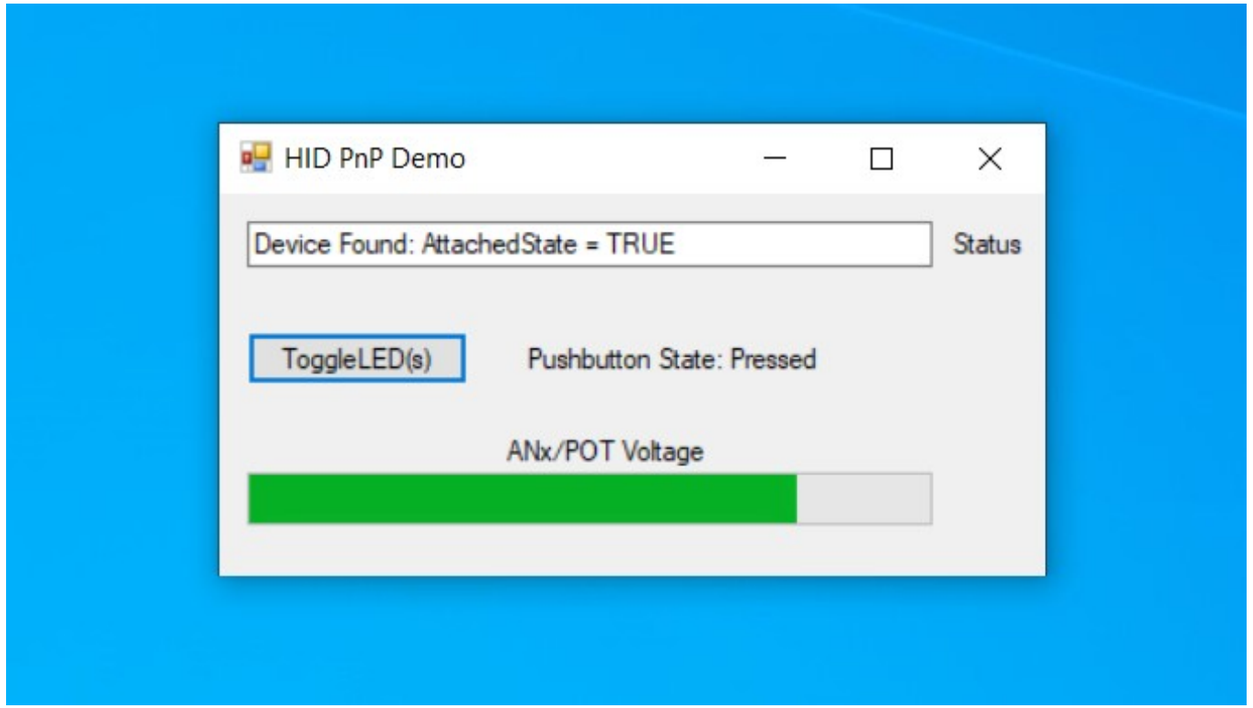
F446RE_USB_HID_PnP: tesztelés

- A HIDapi **testgui.exe** használatával tesztelünk
- A 0483:5750 eszközt kiválasztjuk, majd **Connect**
- Az **Output** ablakba 65 bájtot kell írni (az első kötelezően nulla és nem része az üzenetcsomagnak)
- A **Send Output Report** gombra kattintva küldhetjük ki az üzenetet
- A beérkező üzenetek az input ablakban jelennek meg
- Az ábrán látható válasz (0x80 01) azt jelzi, hogy nincs lenyomva a nyomógomb
- A 37 4d 09 00 kezdetű üzenet pedig azt jelezné, hogy a **PA0** bemenet jele 0x94D, ami körülbelül 1,91 V-nak felel meg

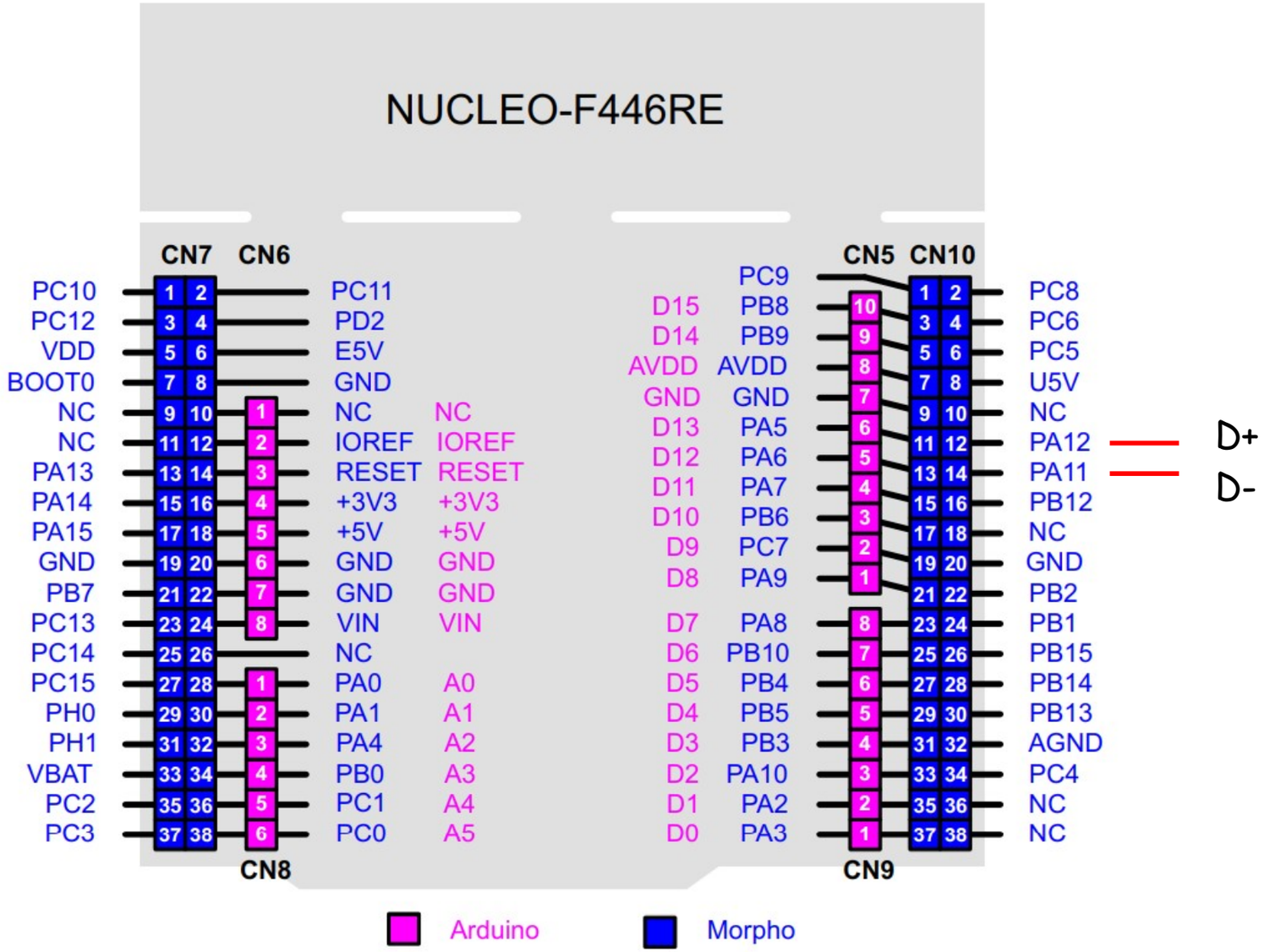


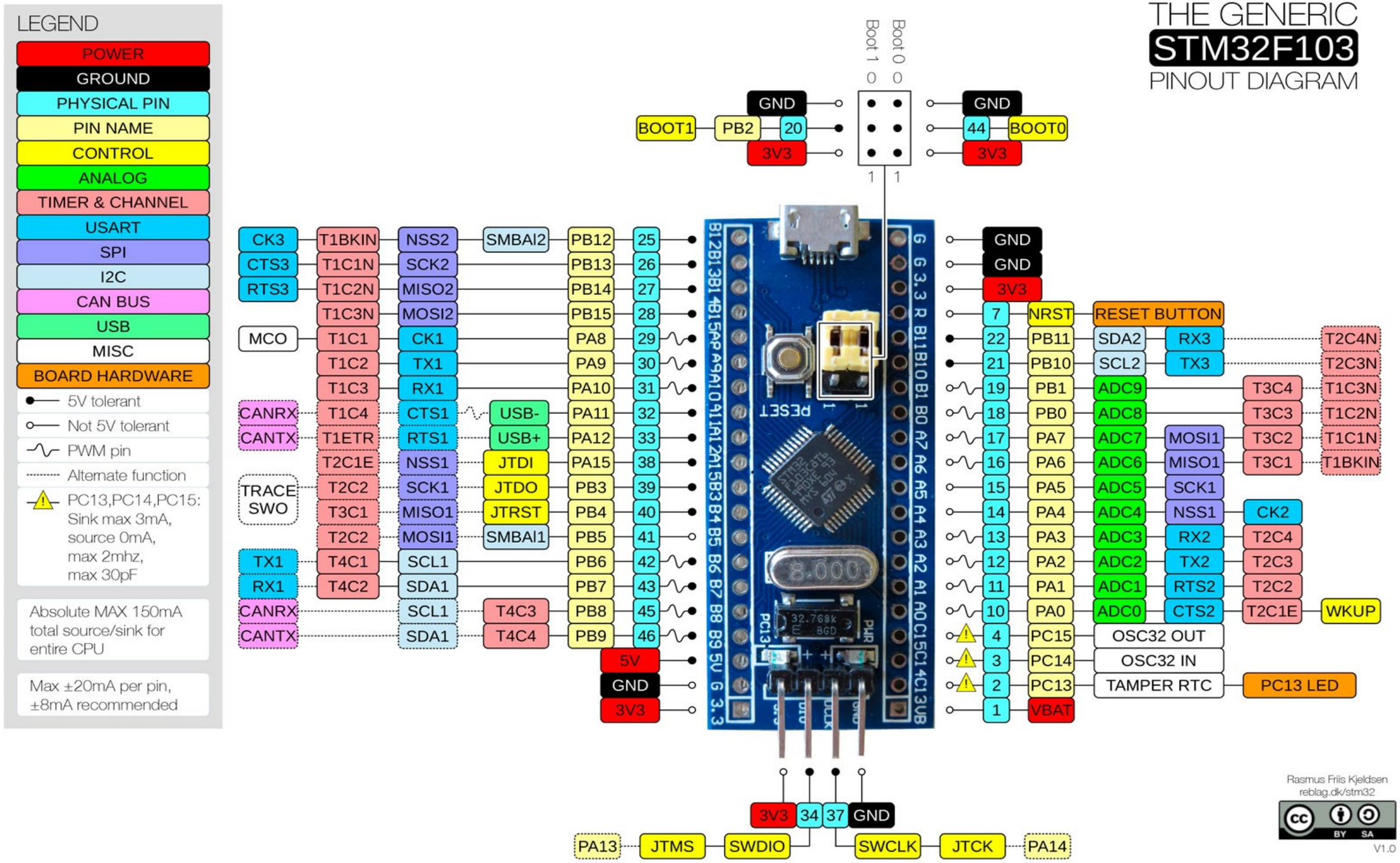
F446RE_USB_HID_PnP próbája a HID PnP Demo-val

- A **HID PnP Demo** alkalmazás automatikusan megkeresi a **0483:5750** azonosítójú eszközt és kapcsolódik (vagy megszakadás után újrakapcsolódik). Kapcsolós esetén TRUE a státusz
- A LED a nyomógombbal kapcsolgatható, a nyomógomb állapota és az analóg jel pedig automatikusan frissítődik



NUCLEO-F446RE kivezetések




LEGEND
POWER
GROUND
PHYSICAL PIN
PIN NAME
CONTROL
ANALOG
TIMER & CHANNEL
USART
SPI
I2C
CAN BUS
USB
MISC
BOARD HARDWARE
5V tolerant
Not 5V tolerant
PWM pin
Alternate function
PC13,PC14,PC15: Sink max 3mA, source 0mA, max 2mhz, max 30pF
Absolute MAX 150mA total source/sink for entire CPU
Max ±20mA per pin, ±8mA recommended
THE GENERIC
STM32F103
PINOUT DIAGRAM
Boot 1
Boot 0
GND
BOOT1
PB2
20
3V3
GND
44
BOOT0
3V3
CK3
CTS3
RTS3
MCO
CANRX
CANTX
TRACE SWO
TX1
RX1
CANRX
CANTX
T1BKIN
T1C1N
T1C2N
T1C3N
T1C1
T1C2
T1C3
T1C4
T1ETR
T2C1E
T2C2
T3C1
T2C2
T4C1
T4C2
NSS2
SCK2
MISO2
MOSI2
CK1
TX1
RX1
CTS1
RTS1
NSS1
SCK1
MISO1
MOSI1
SCL1
SDA1
SCL1
SDA1
SMBAI2
USB-
USB+
JTDI
JTDO
JTRST
SMBAI1
T4C3
T4C4
PB12
PB13
PB14
PB15
PA8
PA9
PA10
PA11
PA12
PA15
PB3
PB4
PB5
PB6
PB7
PB8
PB9
25
26
27
28
29
30
31
32
33
38
39
40
41
42
43
45
46
5V
GND
3V3
GND
GND
3V3
7
22
21
19
18
17
16
15
14
13
12
11
10
4
3
2
1
NRST
PB11
PB10
PB1
PB0
PA7
PA6
PA5
PA4
PA3
PA2
PA1
PA0
PC15
PC14
PC13
VBAT
RESET BUTTON
SDA2
SCL2
ADC9
ADC8
ADC7
ADC6
ADC5
ADC4
ADC3
ADC2
ADC1
ADC0
OSC32 OUT
OSC32 IN
TAMPER RTC
RX3
TX3
MOSI1
MISO1
SCK1
NSS1
RX2
TX2
RTS2
CTS2
T3C4
T3C3
T3C2
T3C1
CK2
T2C4
T2C3
T2C2
T2C1E
PC13 LED
T2C4N
T2C3N
T1C3N
T1C2N
T1C1N
T1BKIN
WKUP
3V3
34
37
GND
PA13
JTMS
SWDIO
SWCLK
JTCK
PA14
Rasmus Friis Kjeldsen
reblag.dk/stm32
V1.0