STM32 mikrovezérlők programozása STM32CubeIDE környezetben

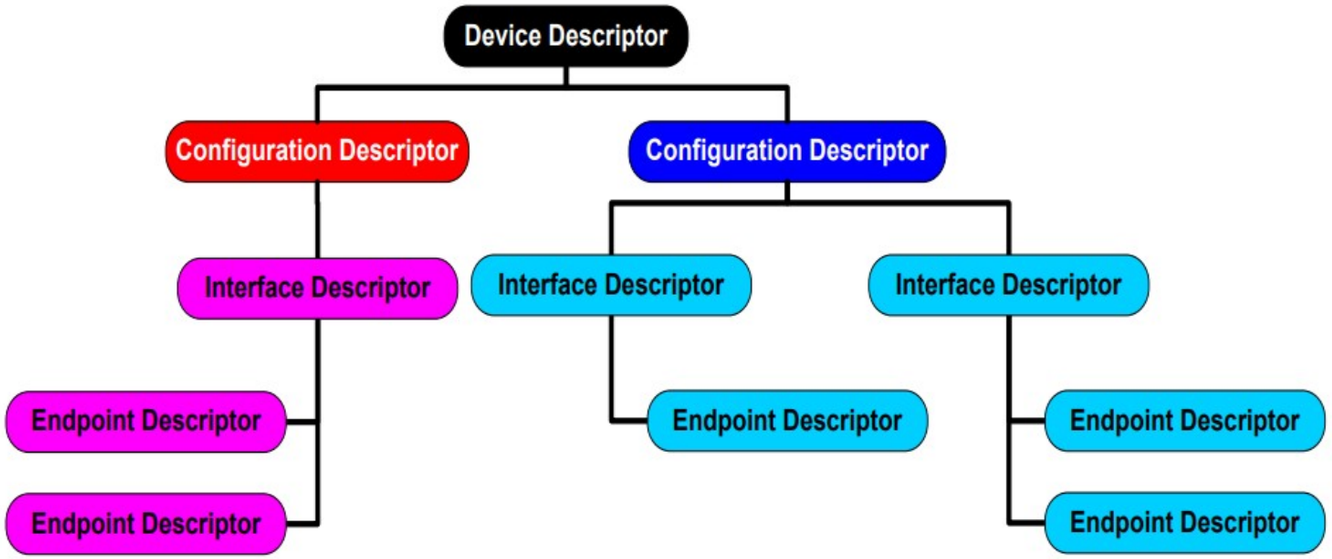
12. Az USB HID eszközosztály – 2. rész

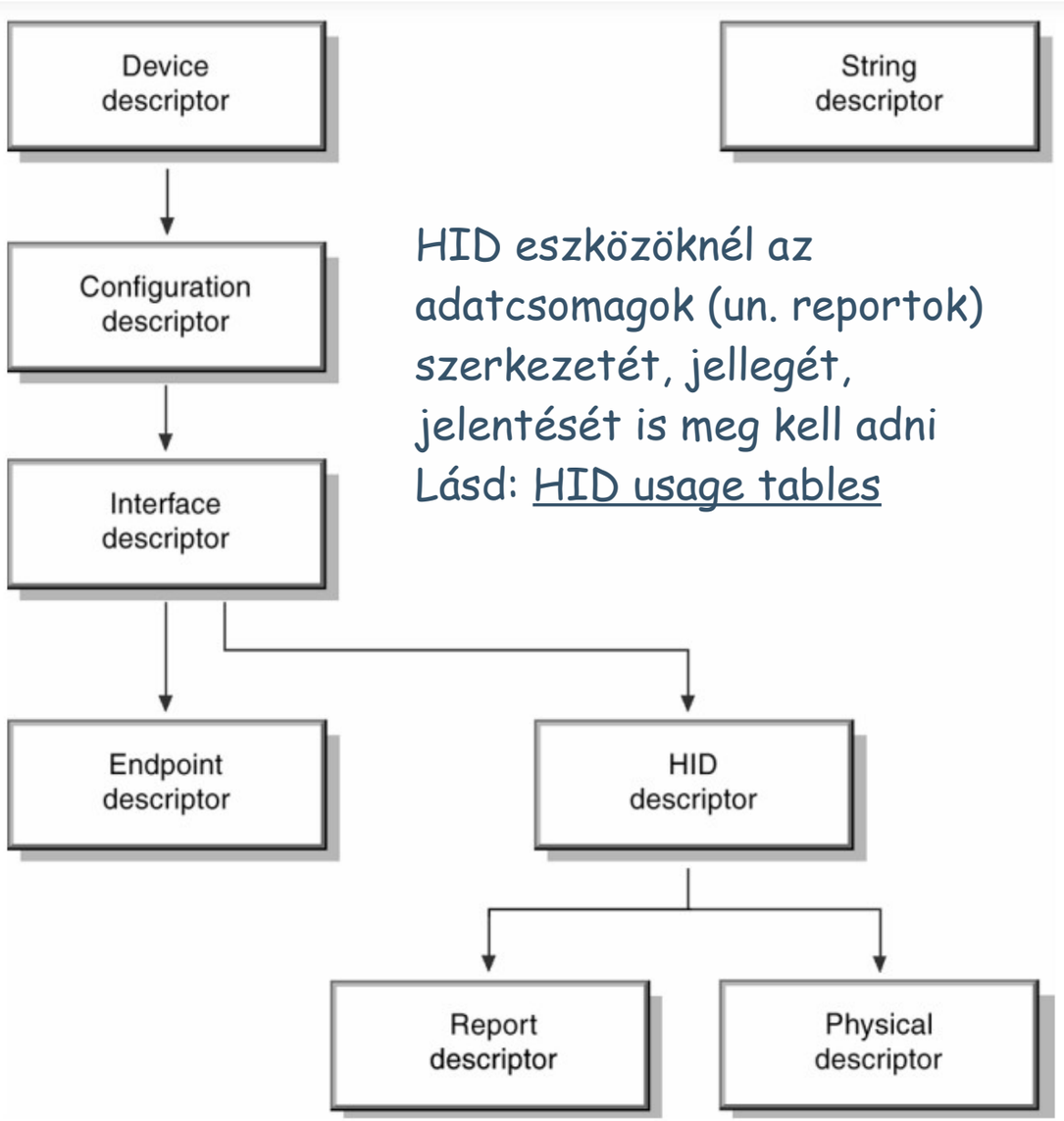
Felhasznált és ajánlott irodalom

- 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 Axelson: The HID page (weblap)
- USB-IF: USB 2.0 Specification
- USB-IF: USB Device Class Definition for Human Interface Devices (HID)
- USB-IF: HID Usage Tables for USB
- Brave Engineer: STM32F103 USB емулятор HID клавіатури

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

Device descriptor

- A Device descriptor az a leíró adatsor, amelyet kapcsolódáskor előséként kér le és olvas el a host
- Példaképpen nézzük meg egy olyan USB HID eszköz leíró táblázatait, amely billentyűzet és pointer is egyben (a HID eszközosztályt leíró **hid1_11.pdf** mintapéldája)
- A mi programunk ettől annyiban tér el, hogy egy pointer eszközt írunk át billentyűzetnek (csak egy interface lesz ….)

Part	Offset/Size (Bytes)	Description	Sample Value
bLength	0/1	Numeric expression specifying the size of this descriptor.	0x12
bDescriptorType	1/1	Device descriptor type (assigned by USB).	0x01
bcdUSB	2/2	USB HID Specification Release 1.0.	0x100
bDeviceClass	4/1	Class code (assigned by USB). Note that the HID class is defined in the Interface descriptor.	0x00
bDeviceSubClass	5/1	Subclass code (assigned by USB). These codes are qualified by the value of the *bDeviceClass* field.	0x00
bDeviceProtocol	6/1	Protocol code. These codes are qualified by the value of the *bDeviceSubClass* field.	0x00
bMaxPacketSize0	7/1	Maximum packet size for endpoint zero (only 8, 16, 32, or 64 are valid).	0x08
idVendor	8/2	Vendor ID (assigned by USB). For this example we'll use 0xFFFF.	0xFFFF
idProduct	10/2	Product ID (assigned by manufacturer).	0x0001
bcdDevice	12/2	Device release number (assigned by manufacturer).	0x0100
iManufacturer	14/1	Index of String descriptor describing manufacturer.	0x04
iProduct	15/1	Index of string descriptor describing product.	0x0E
iSerialNumber	16/1	Index of String descriptor describing the device's serial number.	0x30
bNumConfigurations	17/1	Number of possible configurations.	0x01

Configuration descriptor

Part	Offset/Size (Bytes)	Description	Sample Value
bLength	0/1	Size of this descriptor in bytes.	0x09
bDescriptorType	1/1	Configuration (assigned by USB).	0x02
wTotalLength	2/2	Total length of data returned for this configuration. Includes the combined length of all returned descriptors (configuration, interface, endpoint, and HID) returned for this configuration. This value includes the HID descriptor but none of the other HID class descriptors (report or designator).	0x003B
bNumInterfaces	4/1	Number of interfaces supported by this configuration.	0x02
bConfigurationValue	5/1	Value to use as an argument to Set Configuration to select this configuration.	0x01
iConfiguration	6/1	Index of string descriptor describing this configuration. In this case there is none.	0x00
bmAttributes	7/1	Configuration characteristics 7 Bus Powered 6 Self Powered 5 Remote Wakeup 4..0 Reserved (reset to 0)	10100000B
MaxPower	8/1	Maximum power consumption of USB device from bus in this specific configuration when the device is fully operational. Expressed in 2 mA units—for example, 50 = 100 mA. The number chosen for this example is arbitrary.	0x32

Interface descriptor (keyboard)

Part	Offset/Size (Bytes)	Description	Sample Value
bLength	0/1	Size of this descriptor in bytes.	0x09
bDescriptorType	1/1	Interface descriptor type (assigned by USB).	0x04
bInterfaceNumber	2/1	Number of interface. Zero-based value identifying the index in the array of concurrent interfaces supported by this configuration.	0x00
bAlternateSetting	3/1	Value used to select alternate setting for the interface identified in the prior field.	0x00
bNumEndpoints	4/1	Number of endpoints used by this interface (excluding endpoint zero). If this value is zero, this interface only uses endpoint zero.	0x01
bInterfaceClass	5/1	Class code (HID code assigned by USB).	0x03
bInterfaceSubClass	6/1	Subclass code. 0 No subclass 1 Boot Interface subclass	0x01
bInterfaceProtocol	7/1	Protocol code. 0 None 1 Keyboard 2 Mouse	0x01
iInterface	8/1	Index of string descriptor describing this interface.	0x00

HID descriptor (keyboard)

Part	Offset/Size (Bytes)	Description	Sample Value
bLength	0/1	Size of this descriptor in bytes.	0x09
bDescriptorType	1/1	HID descriptor type (assigned by USB).	0x21
bcdHID	2/2	HID Class Specification release number in binary-coded decimal—for example, 2.10 is 0x210).	0x101
bCountryCode	4/1	Hardware target country.	0x00
bNumDescriptors	5/1	Number of HID class descriptors to follow.	0x01
bDescriptorType	6/1	Report descriptor type.	0x22
wDescriptorLength	7/2	Total length of Report descriptor.	0x3F

- Amikor a saját HID eszközünkhöz egyedi HID report descriptort készítünk, akkor annak méretéhez kel igazítani a fenti paramétert!

Endpoint descriptor (keyboard)

Part	Offset/Size (Bytes)	Description	Sample Value
bLength	0/1	Size of this descriptor in bytes.	0x07
bDescriptorType	1/1	Endpoint descriptor type (assigned by USB).	0x05
bEndpointAddress	2/1	The address of the endpoint on the USB device described by this descriptor. The address is encoded as follows: Bit 7 Direction ← Bit 0..3 The endpoint number	10000001B 0 - OUT endpoint 1 - IN endpoint
bmAttributes	3/1	This field describes the endpoint's attributes when it is configured using the bConfigurationValue. 11	00000011B Interrupt
wMaxPacketSize	4/2	Maximum packet size this endpoint is capable of sending or receiving when this configuration is selected.	0x0008
bInterval	6/1	Interval for polling endpoint for data transfers. Expressed in milliseconds.	0x0A

HID Report descriptor (keyboard)

- A mai példaprogramjainkhoz egy ugyanilyen HID report leíróra lesz szükségünk
- A report formátuma:

0	Módosítók (Shift, Alt, Ctrl, stb)
1	reserved
2	Billentyű kód 1
3	Billentyű kód 2
4	Billentyű kód 3
5	Billentyű kód 4
6	Billentyű kód 5
7	Billentyű kód 6

Item		Value (Hex)
Usage Page (Generic Desktop),		05 01
Usage (Keyboard),		09 06
Collection (Application),		A1 01
Usage Page (Key Codes);		05 07
Usage Minimum (224),		19 E0
Usage Maximum (231),		29 E7
Logical Minimum (0),		15 00
Logical Maximum (1),		25 01
Report Size (1),		75 01
Report Count (8),		95 08
Input (Data, Variable, Absolute),	;Modifier byte	81 02
Report Count (1),		95 01
Report Size (8),		75 08
Input (Constant),	;Reserved byte	81 01
Report Count (5),		95 05
Report Size (1),		75 01
Usage Page (Page# for LEDs),		05 08
Usage Minimum (1),		19 01
Usage Maximum (5),		29 05
Output (Data, Variable, Absolute),	;LED report	91 02
Report Count (1),		95 01
Report Size (3),		75 03
Output (Constant),	;LED report padding	91 01
Report Count (6),		95 06
Report Size (8),		75 08
Logical Minimum (0),		15 00
Logical Maximum(101),		25 65
Usage Page (Key Codes),		05 07
Usage Minimum (0),		19 00
Usage Maximum (101),		29 65
Input (Data, Array),	;Key arrays (6 bytes)	81 00
End Collection		C0

Példaprogramok

Blue pill kártya:

F103_USB_HID_PnP – LED vezérlés és adatbeolvasás

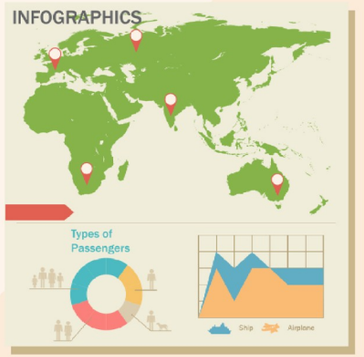
F103_HID_keyboard – USB billentyűzet emuláció (HU)

F103_HID_kwikboard – PC vezérlés nyomógombokkal (HU)

Nucleo-F446RE kártya:

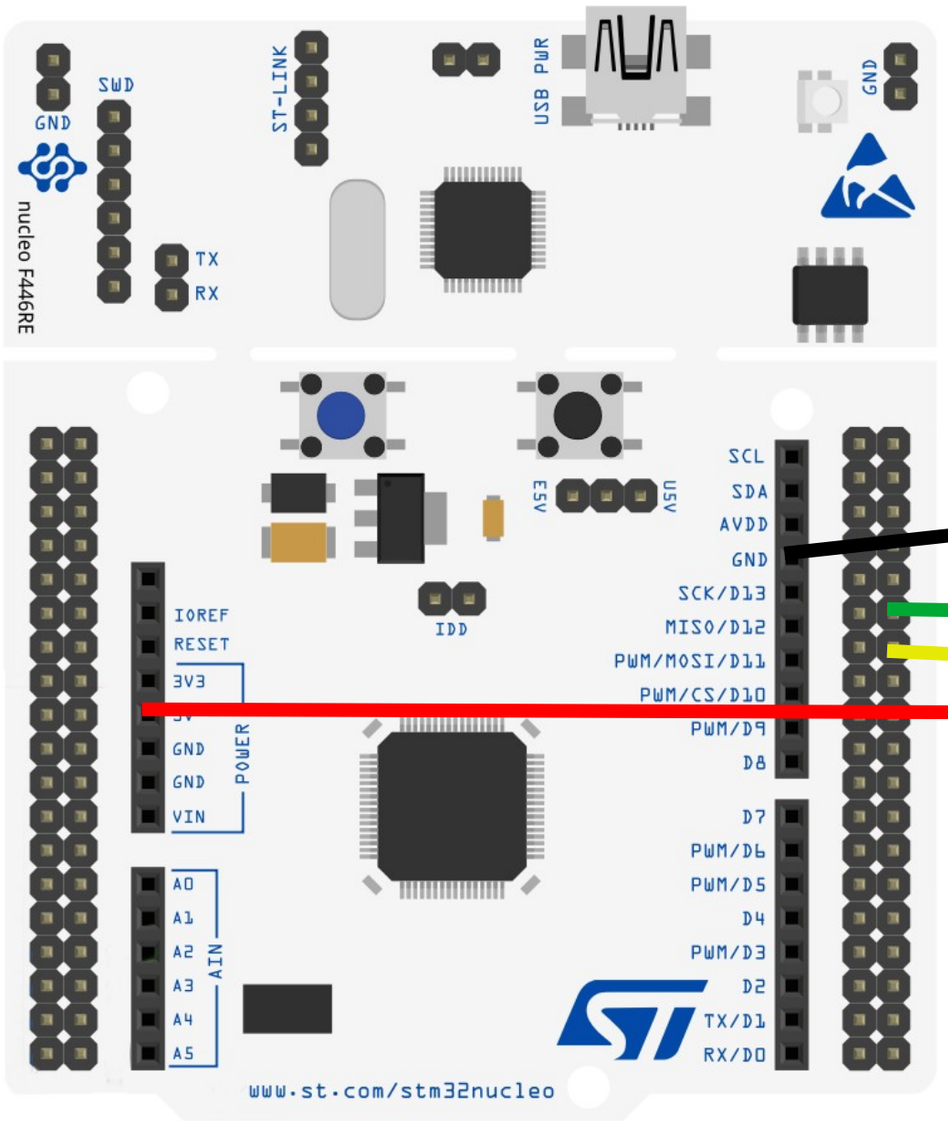
F446RE_HID_keyboard – USB billentyűzet emuláció (US)

F446RE_HID_kwikboard – PC vezérlés nyomógombokkal (HU)

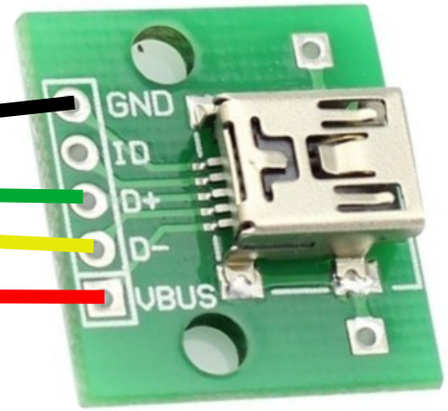


freepik.com

Bekötési vázlat

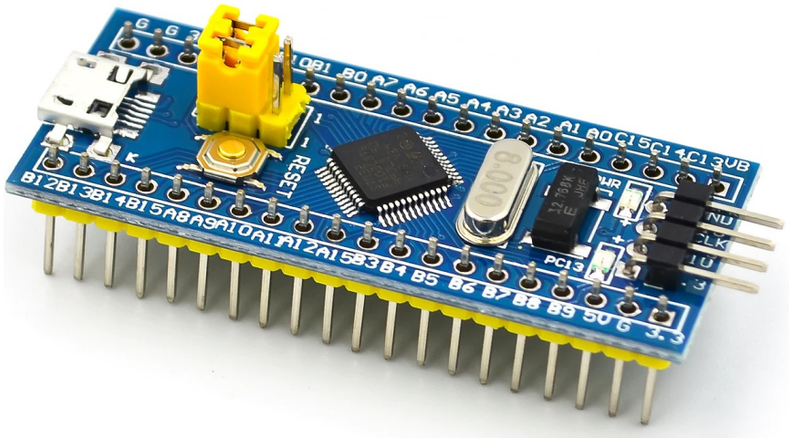


PA12 USB D+
PA11 USB D-



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

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



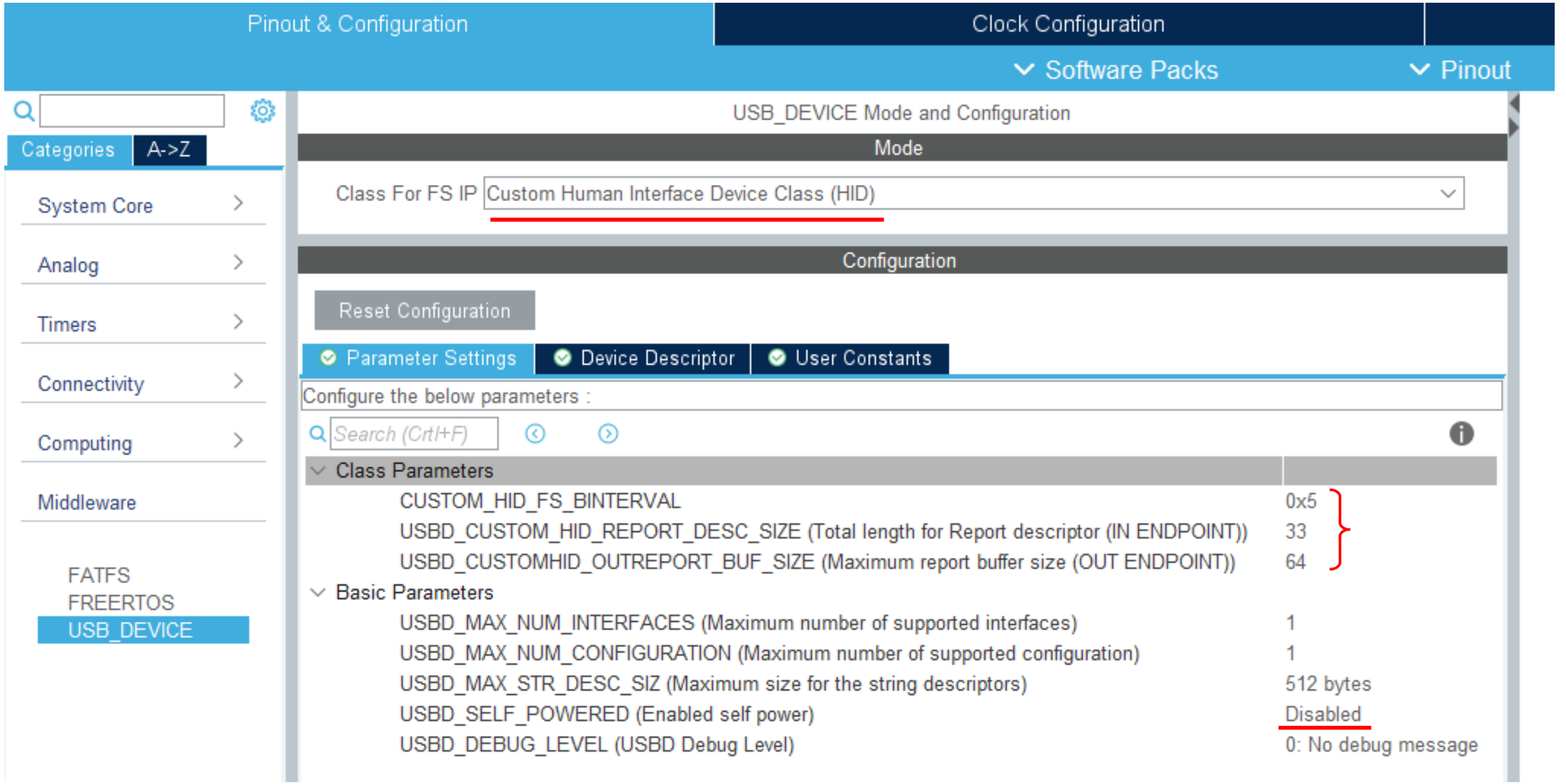
F103_USB_HID_PnP

- Az előző előadásban bemutatott projektet az **STMF103C8** MCU-ra is adaptáljuk: a program a PC felől érkező 64 bájtos üzenetek első bájtját az alábbi táblázat szerint értelmezi

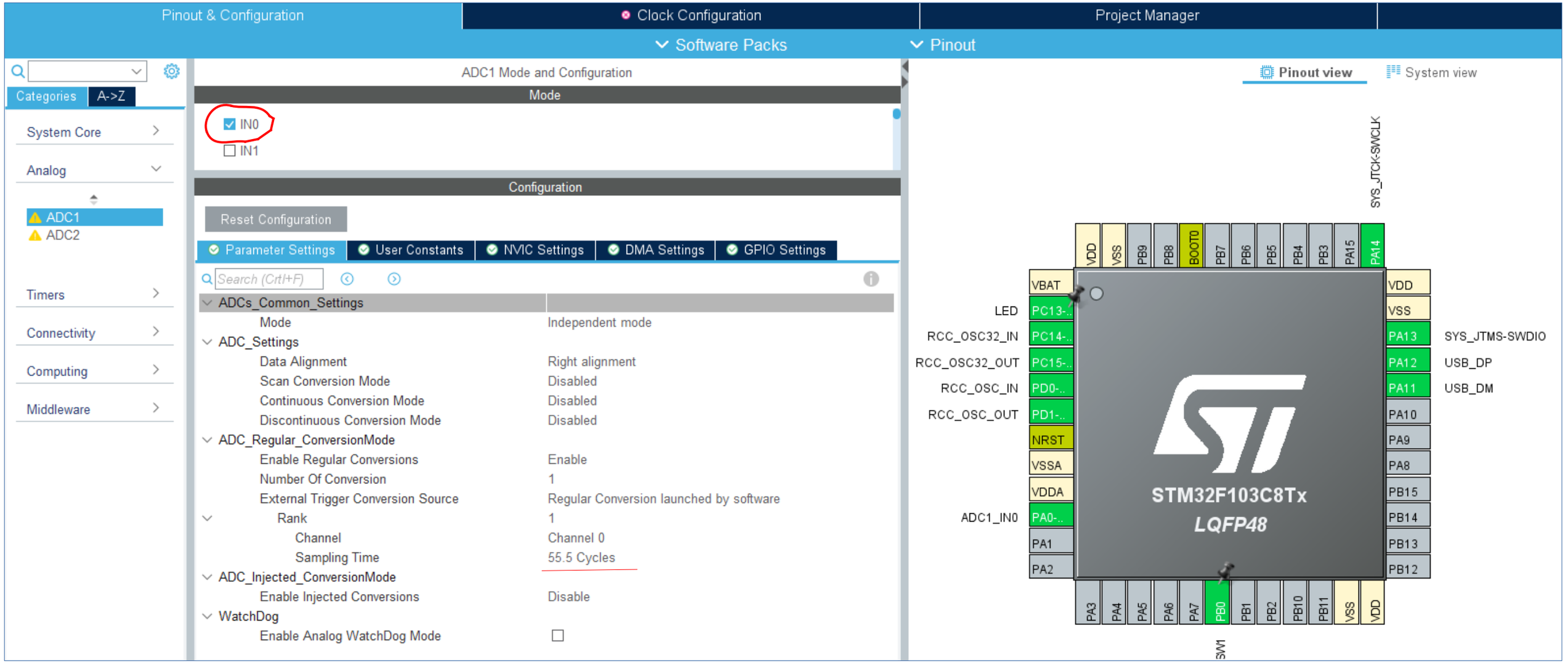
Parancs	Válasz	Szöveges leírás
0x80	nincs	LED állapot átbillentése (LED a PC13 kivezetésre van kötve)
0x81	0x81 1/0	Nyomógomb állapotának lekérdezése (SW1 a PB0 kivezetésre van kötve)
0x37	0x37 LSB MSB	ADC IN0 csatorna értékének lekérdezése (IN0 a PA0 kivezetésre van kötve)
egyéb	0xFF	Nem értelmezett parancs

- A **0x80** parancsra nem küldünk választ
- A **0x81** parancsnál a második bájt 1 vagy 0, a **PB0** 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)

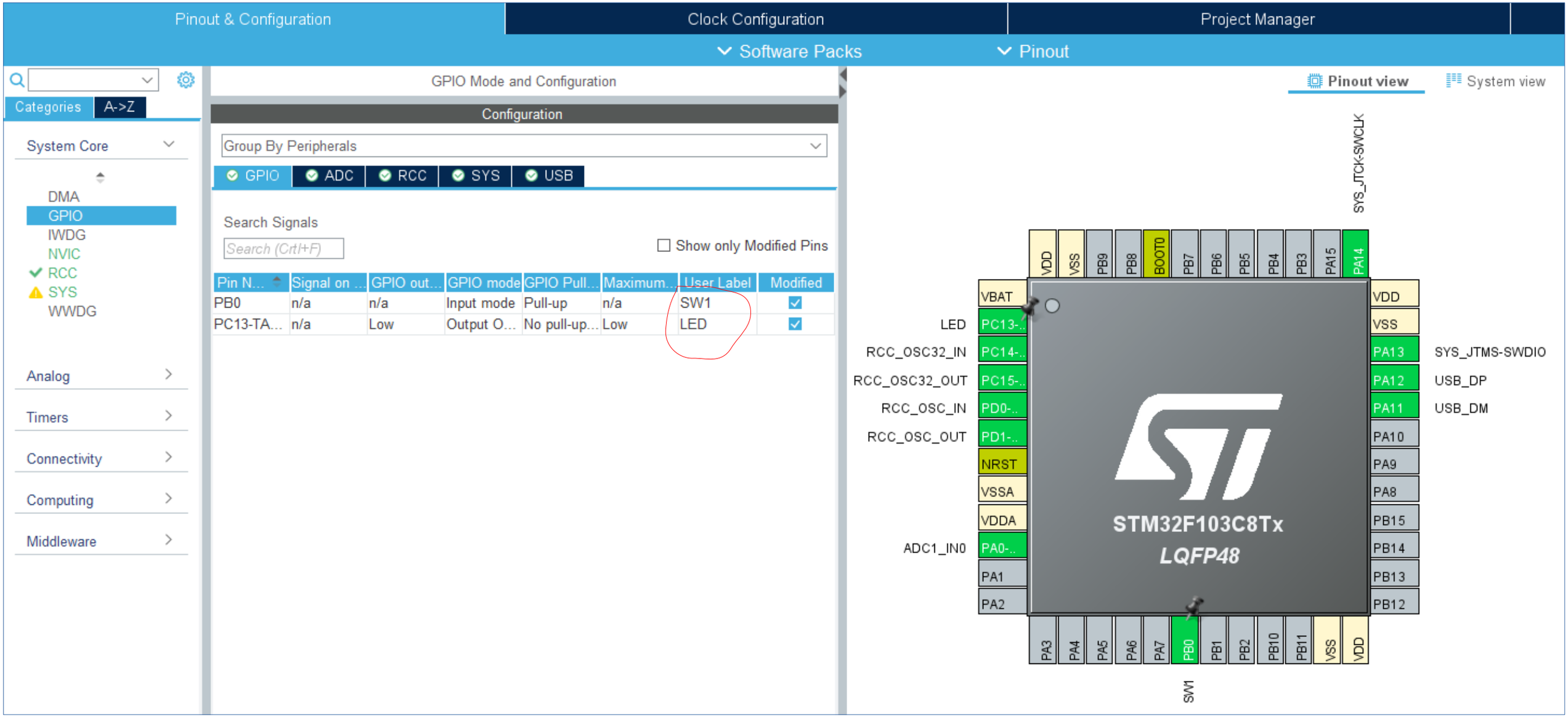
F103_USB_HID_PnP: custom HID konfigurálás



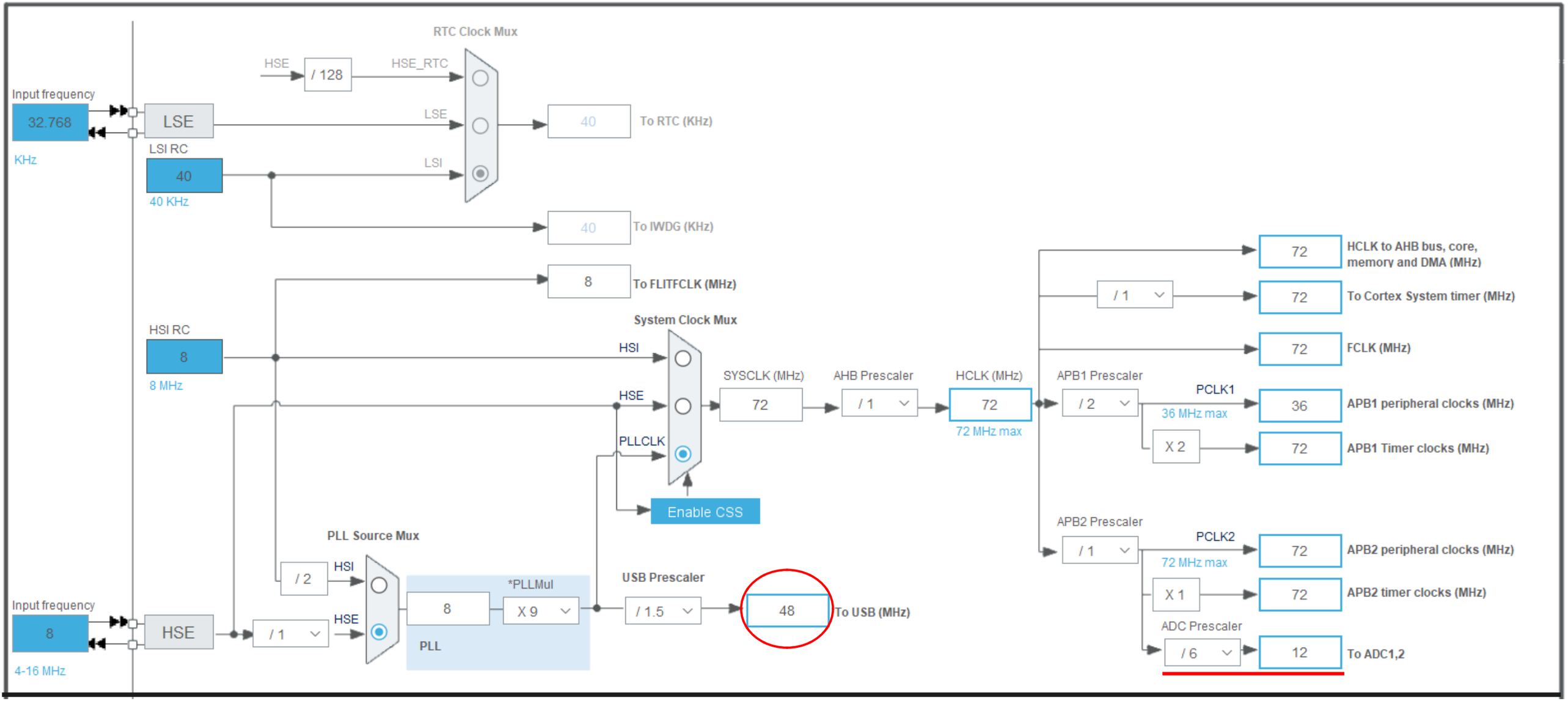
F103_USB_HID_PnP: ADC konfigurálás



F103_USB_HID_PnP: GPIO konfigurálás



F103_USB_HID_PnP: Az órajelek konfigurálása



F103_USB_HID_PnP: 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:** Ellenőrizzük OUT report és a HID report deszkriptor méretét!

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

Ezeket elvileg a Custom HID Class konfigurálásnál már beállítottuk

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

F103_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:** tegyük elérhetővé a **main.c**-ben definiált 64 bájtos ki- és bemeneti buffereket és a jelzőt az üzenetek kezeléséhez!

```
/* Private variables ---------------------------------------------------------*/
/* USER CODE BEGIN PV */
  extern uint8_t TX_buffer[0x40];       //--- USB customhid
  extern uint8_t RX_buffer[0x40];
  extern uint8_t RX_buffer_flag;
/* USER CODE END PV */
```

F103_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 */
};
```

F103_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 (ne legyen ***static*** függvény!)

```
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 (ne legyen ***static*** függvény!)

F103_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;

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_ADC1_Init(void);

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

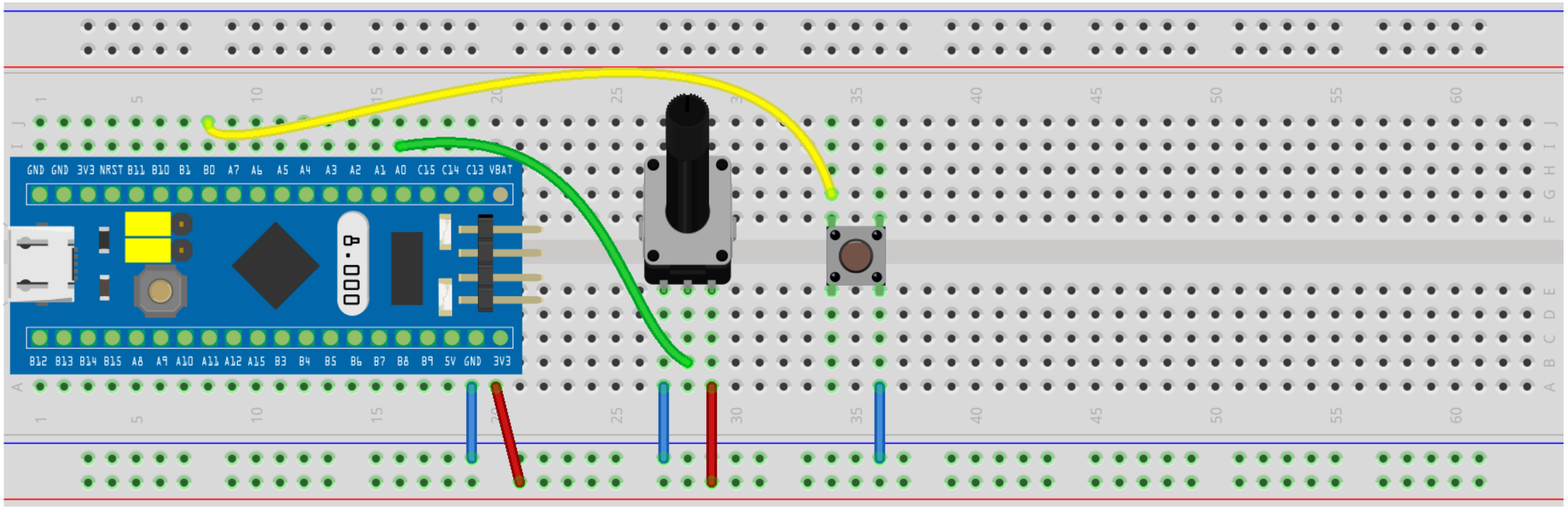
F103_USB_HID_PnP: main.c (részlet)

```
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

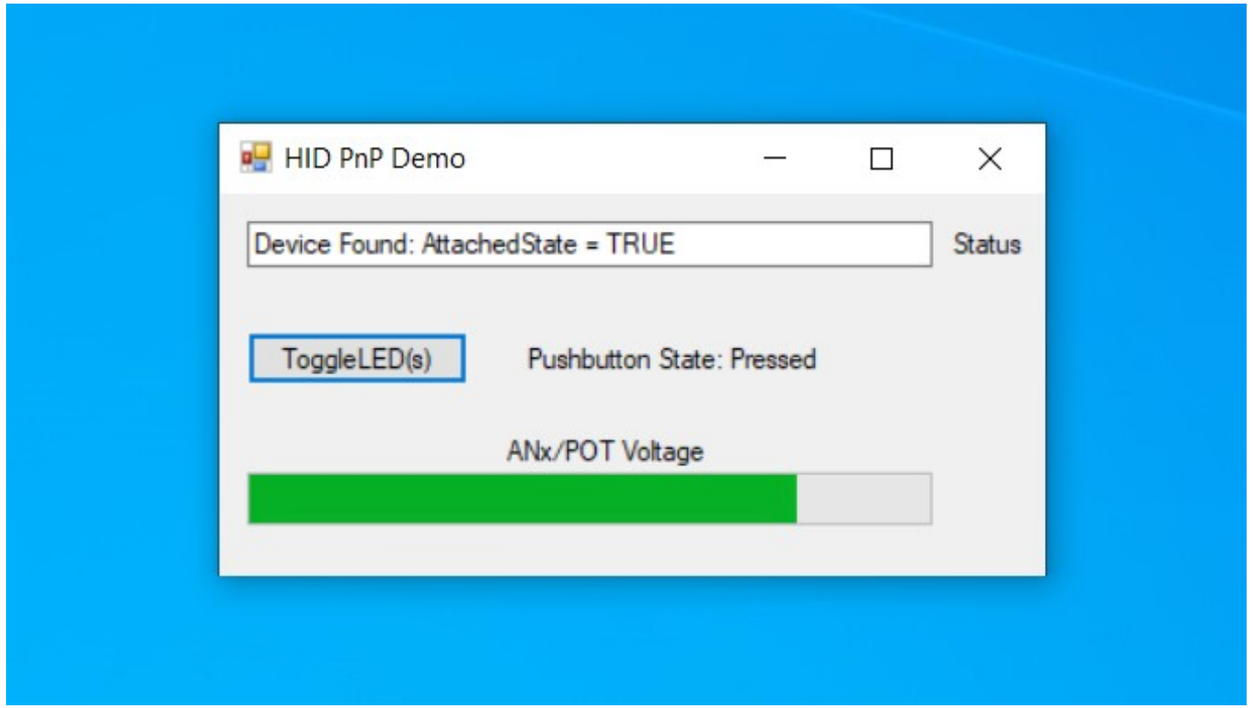
PA0 – IN0 analóg bemenet
PB0 – SW1 digitális bemenet

PA12 USB D+
PA11 USB D-



F103_USB_HID_PnP próbája a HID PnP Demo-val

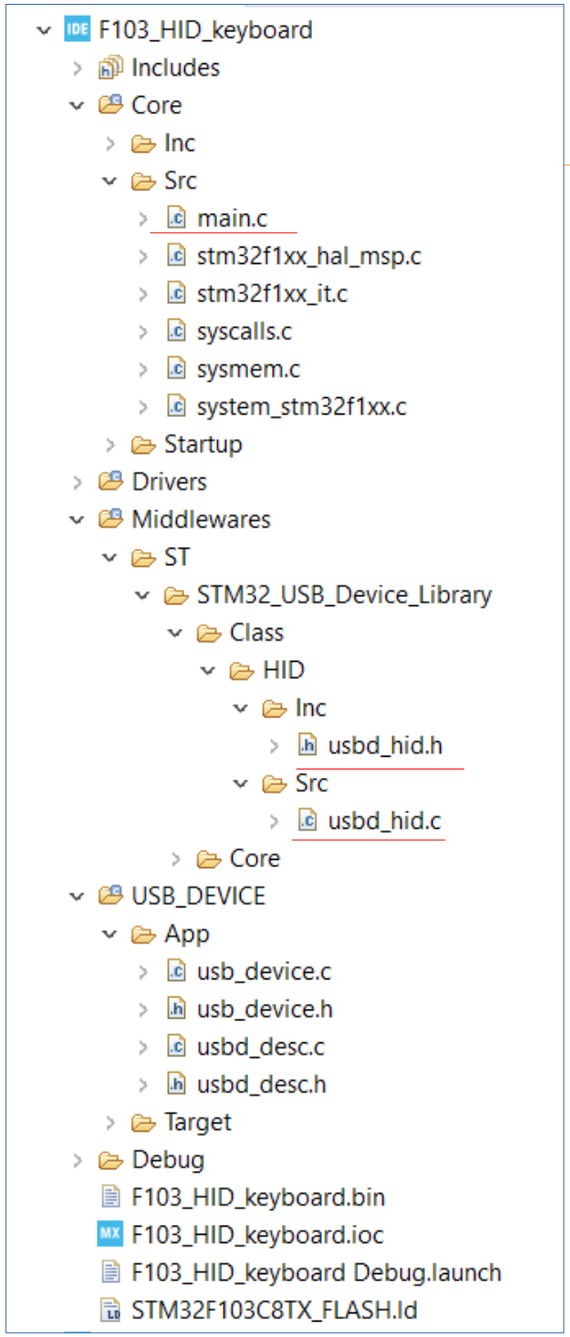
- A **HID PnP Demo** alkalmazás automatikusan megkeresi a **0483:5750** azonosítójú eszközt és kapcsolódik (vagy megszakadás után újrakapcsolódik). Kapcsolós esetén TRUE a státusz
- A LED a nyomógombbal kapcsolgatható, a nyomógomb állapota és az analóg jel pedig automatikusan frissítődik

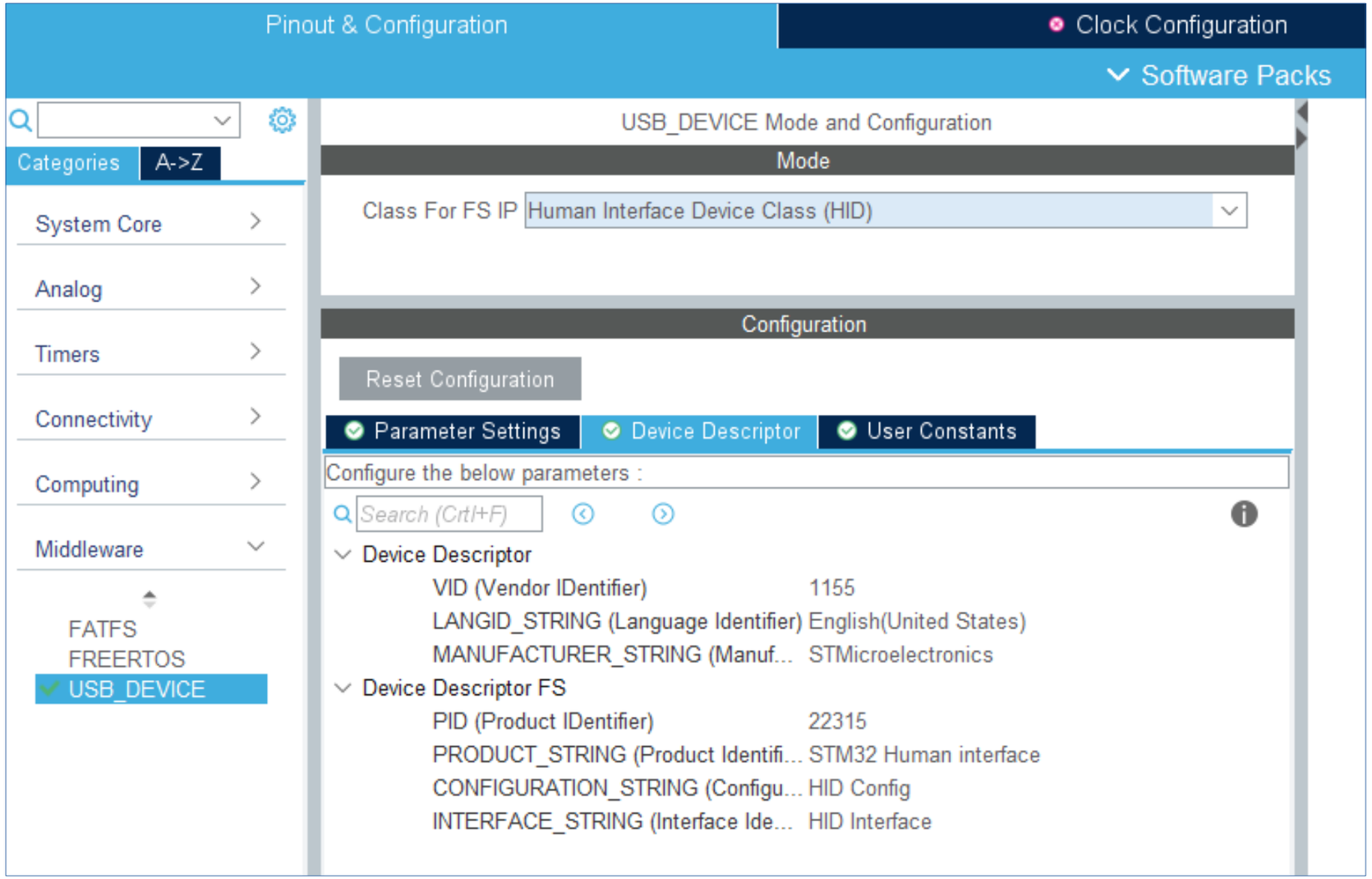


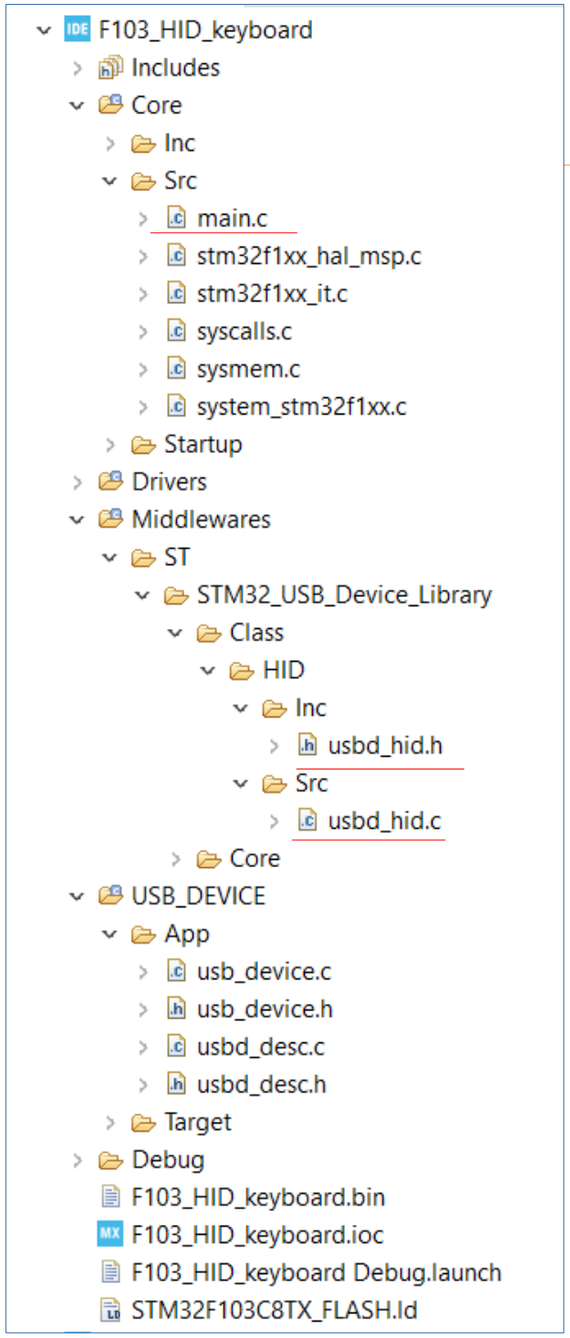
F103_HID_keyboard

- A **Brave Engineer** útmutatását (brave-engineer.space/usb-hid-keyboard/) követve, egy USB HID billentyűzetet fogunk emulálni, amellyel a számítógépbe szöveget vagy parancsot juttathatunk be
- Kezdjük azzal, hogy létrehozunk egy új projektet, amelyben egy Human Interface Device (HID) osztályú USB Full Speed eszközt konfigurálunk
- A grafikus konfiguráló program az automatikus kódgeneráláskor alapértelmezetten egy egér protokollt megvalósító eszközt definiál, ami 4 bájtos jelentéseket küld (gombok, x, y, és görgetés) – ezt írjuk át, hogy billentyűzetként ismerje fel a PC, ami 8 bájtos jelentést küld
- A descriptor táblákat hályogkovács módszerekkel csak minimális mértékben módosítjuk
- A főprogram a **WIN + R** billentyűkódot kiküldése után a lap tetején is megadott `https://brave-engineer.space/usb-hid-keyboard/` linket, **Enter**-rel lezárva küldi a számítógépnek, ami beindítja az alapértelmezett böngészőt és megnyitja a fenti URL-t
- Ebben a példában **magyar billentyűzetkiosztás** szerinti kódot küldünk, máskülönben nem érti meg a rendszer (az eredeti program USA kiosztású billentyűzettel működik)

F103_HID_keyboard: USB konfigurálás







usb_hid.h módosítások

- Az **usbd_hid.h** állományban az alábbi módosítások kellenek:

```
#define HID_EPIN_ADDR                 0x81U
#define HID_EPIN_SIZE                 0x08U //--- USB HID keyboard report size
#define USB_HID_CONFIG_DESC_SIZ         34U
#define USB_HID_DESC_SIZ                9U
#define HID_MOUSE_REPORT_DESC_SIZE     63U //--- USB HID keyboard rep desc size
```

- Az **usbd_hid.c** állományban módosítanunk kell a protokollt:

```
/* USB HID device FS Configuration Descriptor */
 . . .
  /************** Descriptor of Joystick Mouse interface ****************/
  0x09,         /*bLength: Interface Descriptor size*/
  USB_DESC_TYPE_INTERFACE,/*bDescriptorType: Interface descriptor type*/
  0x00,         /*bInterfaceNumber: Number of Interface*/
  0x00,         /*bAlternateSetting: Alternate setting*/
  0x01,         /*bNumEndpoints*/
  0x03,         /*bInterfaceClass: HID*/
  0x01,         /*bInterfaceSubClass : 1=BOOT, 0=no boot*/
  0x01,         /*nInterfaceProtocol : 0=none, 1=keyboard, 2=mouse*/
  0,            /*iInterface: Index of string descriptor*/
  . . .
```

```
__ALIGN_BEGIN static uint8_t HID_MOUSE_ReportDesc[HID_MOUSE_REPORT_DESC_SIZE]  __ALIGN_END = {
    0x05, 0x01,                    // USAGE_PAGE (Generic Desktop)
    0x09, 0x06,                    // USAGE (Keyboard)
    0xa1, 0x01,                    // COLLECTION (Application)
    0x05, 0x07,                    //   USAGE_PAGE (Keyboard)
    0x19, 0xe0,                    //   USAGE_MINIMUM (Keyboard LeftControl)
    0x29, 0xe7,                    //   USAGE_MAXIMUM (Keyboard Right GUI)
    0x15, 0x00,                    //   LOGICAL_MINIMUM (0)
    0x25, 0x01,                    //   LOGICAL_MAXIMUM (1)
    0x75, 0x01,                    //   REPORT_SIZE (1)
    0x95, 0x08,                    //   REPORT_COUNT (8)
    0x81, 0x02,                    //   INPUT (Data,Var,Abs)
    0x95, 0x01,                    //   REPORT_COUNT (1)
    0x75, 0x08,                    //   REPORT_SIZE (8)
    0x81, 0x03,                    //   INPUT (Cnst,Var,Abs)
    0x95, 0x05,                    //   REPORT_COUNT (5)
    0x75, 0x01,                    //   REPORT_SIZE (1)
    0x05, 0x08,                    //   USAGE_PAGE (LEDs)
    0x19, 0x01,                    //   USAGE_MINIMUM (Num Lock)
    0x29, 0x05,                    //   USAGE_MAXIMUM (Kana)
    0x91, 0x02,                    //   OUTPUT (Data,Var,Abs)
    0x95, 0x01,                    //   REPORT_COUNT (1)
    0x75, 0x03,                    //   REPORT_SIZE (3)
    0x91, 0x03,                    //   OUTPUT (Cnst,Var,Abs)
    0x95, 0x06,                    //   REPORT_COUNT (6)
    0x75, 0x08,                    //   REPORT_SIZE (8)
    0x15, 0x00,                    //   LOGICAL_MINIMUM (0)
    0x25, 0x65,                    //   LOGICAL_MAXIMUM (101)
    0x05, 0x07,                    //   USAGE_PAGE (Keyboard)
    0x19, 0x00,                    //   USAGE_MINIMUM (Reserved (no event indicated))
    0x29, 0x65,                    //   USAGE_MAXIMUM (Keyboard Application)
    0x81, 0x00,                    //   INPUT (Data,Array,Abs)
    0xc0                           // END_COLLECTION
};
```

F103_HID_keyboard: usb_hid.c módosítása

- Az **usbd_hid.c** állományban cseréljük le a HID report descriptort erre a táblázatra (a USB HID Device Class leírás E.6 mintapéldáját követjük)
- A satírozott rész „kakukktojás”, mert itt nem használjuk az OUT üzeneteket, amelyek egyébként a billentyűzet LED-jeit hivatottak vezérelni

F103_HID_keyboard: main.c (részlet)

```
#include "main.h"
#include "usb_device.h"
#include "usbd_hid.h"                     //--- HID class
extern USBD_HandleTypeDef hUsbDeviceFS; //--- USB device handle
uint8_t HID_buffer[8] = { 0 };            //--- HID TX buffer
ADC_HandleTypeDef hadc1;

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_ADC1_Init(void);

void USB_Keyboard_SendChar(char ch);    //--- Egy karakterhez tartozó key scan kódok kiküldése
void USB_Keyboard_SendString(char * s)  //--- Egy string kiküldése karakterenként

int main(void) {
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_ADC1_Init();
  MX_USB_DEVICE_Init();
  while (1) {
    USB_Keyboard_SendString("~https://brave-engineer.space/\n");  //--- Windows parancs kiküldése
    HAL_Delay(15000);                                             //--- 15 másodperc várakozás
  }
}
```

F103_HID_keyboard: main.c (részlet)

```
void USB_Keyboard_SendString(char * s) {
    uint8_t i = 0;
    while(*(s+i)) {
        USB_Keyboard_SendChar(*(s+i));
        i++;
    }
}

//--- Emulate keypresses
void USB_Keyboard_SendChar(char ch) {
  uint8_t ret;
  // Check if lower or upper case
  if(ch >= 'a' && ch <= 'z') {
    HID_buffer[0] = 0;
    HID_buffer[2] = (uint8_t)(4 + (ch - 'a'));
  }
  else if(ch >= 'A' && ch <= 'Z') {
    HID_buffer[0] = 2;    // Add left shift
    ch = ch - ('A'-'a'); // ch to lower case
    HID_buffer[2] = (uint8_t)(4 + (ch - 'a'));
  }
  else  {  // not a letter
    switch(ch)   {
      case ' ':  HID_buffer[2] = 44; break;
      case '~':  HID_buffer[0] = 8; // windows + r
                 HID_buffer[2] = 21; break;
      case '-':  HID_buffer[2] = 56; break;
      case '.':  HID_buffer[2] = 55; break;
      case '/':  HID_buffer[0] = 2;  // SHIFT 6
                 HID_buffer[2] = 35; break;
      case '\n': HID_buffer[2] = 40; break;
      case ':':  HID_buffer[0] = 2;  // SHIFT .
                 HID_buffer[2] = 55; break;
      case '?':  HID_buffer[0] = 2;  // SHIFT ,
                 HID_buffer[2] = 54; break;
      default:   HID_buffer[2] = 0;
    }
  }
  // press keys
  ret = USBD_HID_SendReport(&hUsbDeviceFS, HID_buffer, 8);
  if(ret != HAL_OK) Error_Handler();
  HAL_Delay(15);

  // release keys
  HID_buffer[0] = 0;
  HID_buffer[2] = 0;
  ret = USBD_HID_SendReport(&hUsbDeviceFS, HID_buffer, 8);
  if(ret != HAL_OK) Error_Handler();
  HAL_Delay(15);
}
```

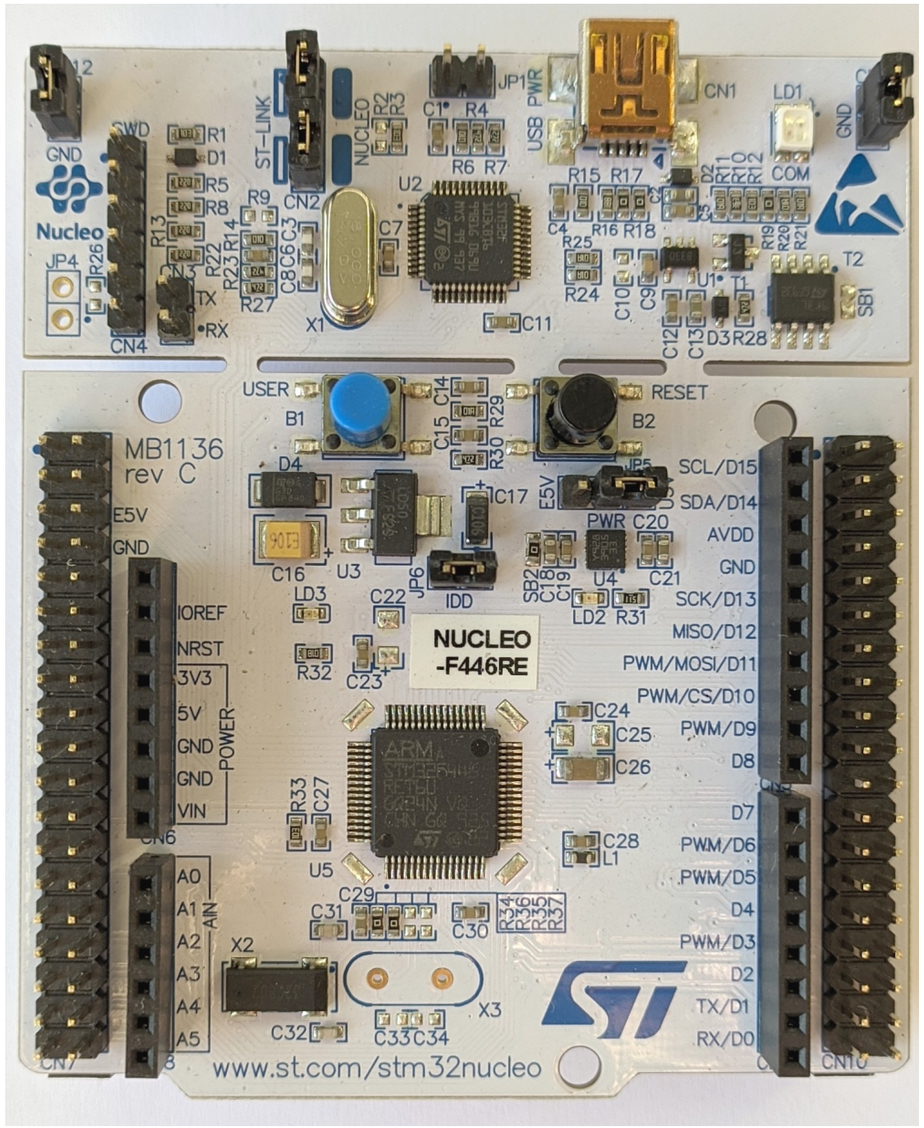
A kódokat itt magyar billentyűzetkiosztás szerint küldjük!

F103_HID_keyboard: futási eredmény

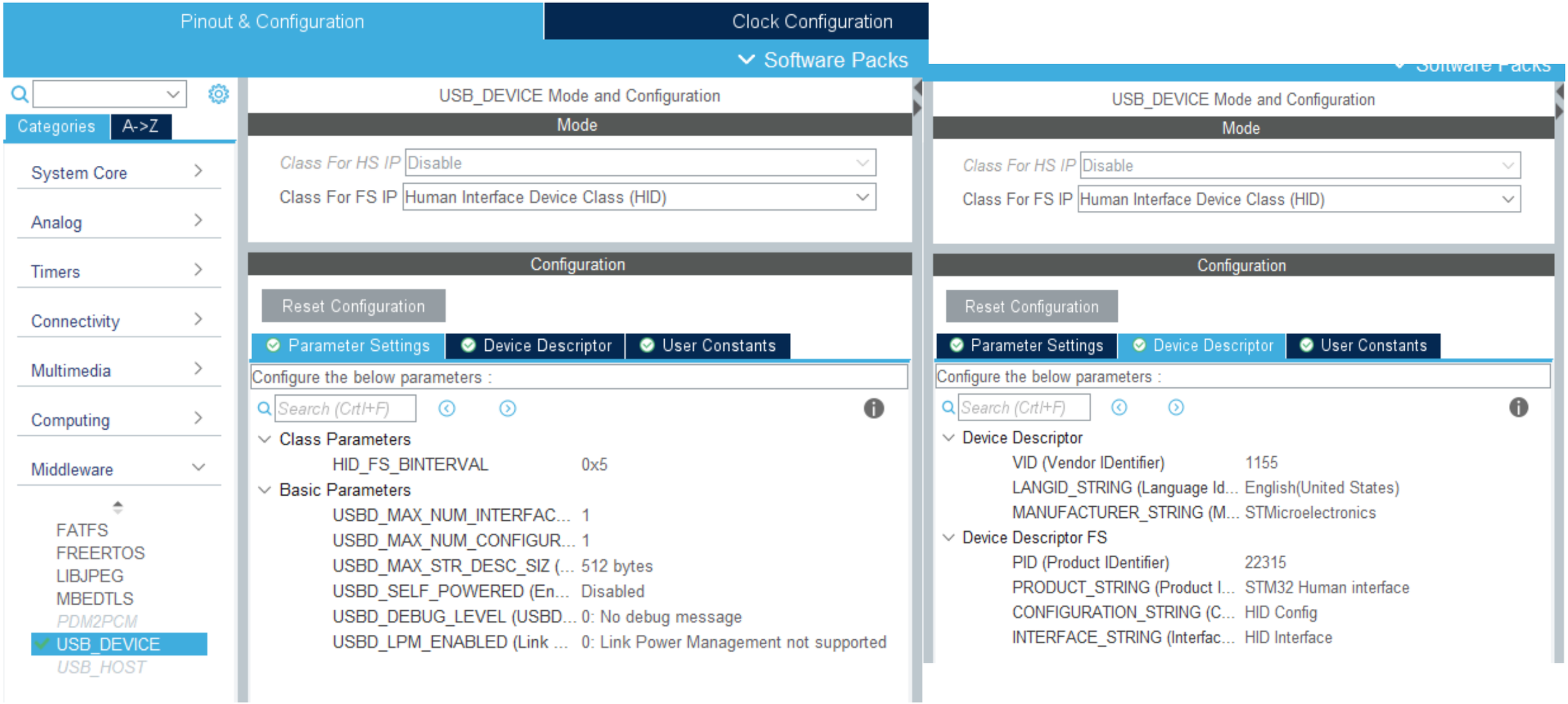
Windows RUN ablak és a kiküldött URL

F446RE_HID_keyboard

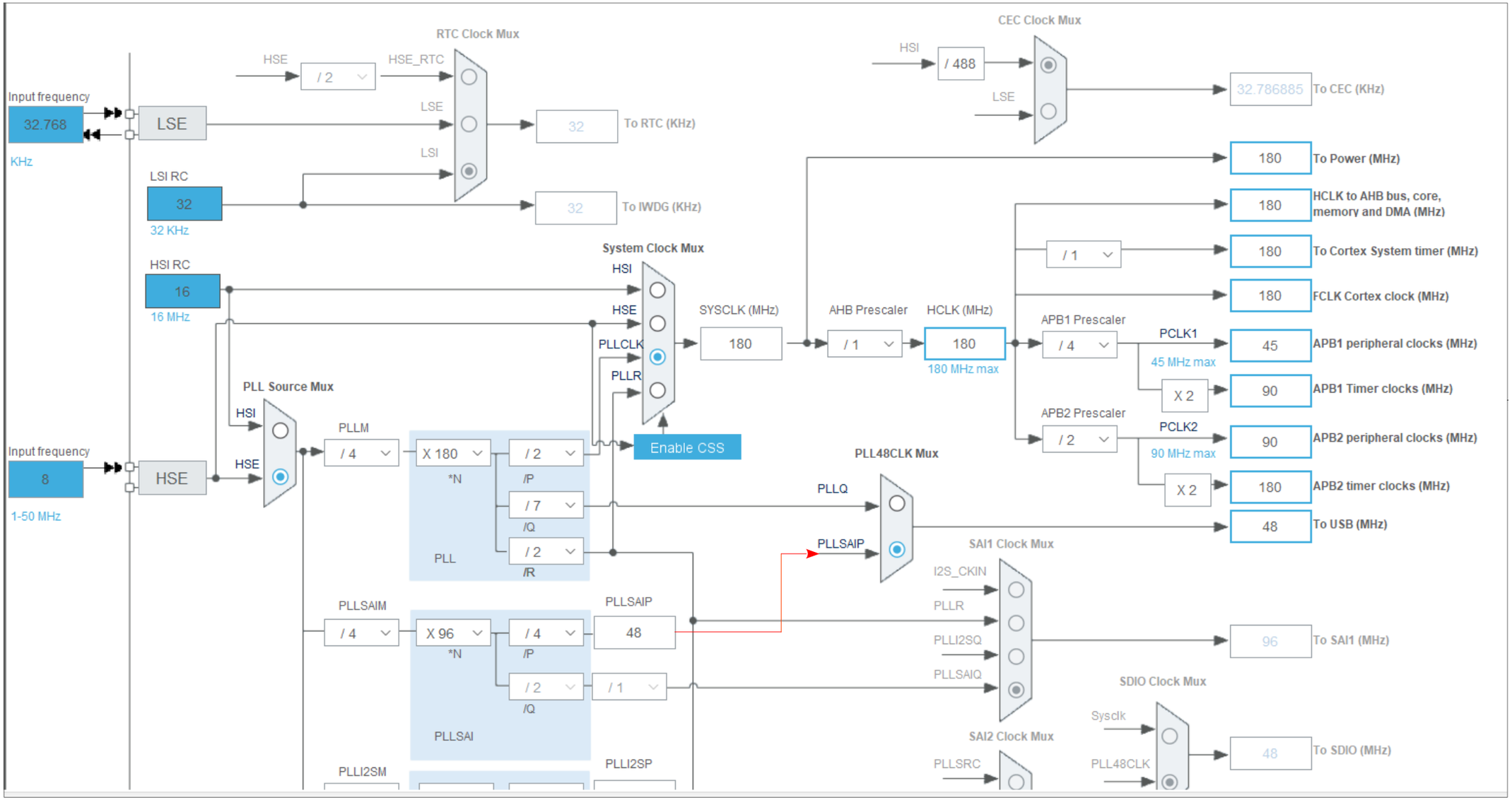
- Az **F103_HID_keyboard** programot könnyen adaptálhatjuk a **NUCLEO-F446RE** kártyára
- Kisebb eltérések csak az órajelek, illetve az USB eszköz konfigurálásnál vannak
- Az **usbd_hid.h** és **usbd_hid.c** állományokat ugyanúgy módosítani kell, mint az **STM32F103C8** mikrovezérlő esetében



F446RE_HID_keyboard: USB library konfigurálás



F448RE_HID_keyboard: 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_HID_keyboard: main.c (csak részlet)

```
void USB_Keyboard_SendString(char * s) {
    uint8_t i = 0;
    while(*(s+i)) {
        USB_Keyboard_SendChar(*(s+i));
        i++;
    }
}

void USB_Keyboard_SendChar(char ch) {
  uint8_t ret;
  // Check if lower or upper case
  if(ch >= 'a' && ch <= 'z') {
    HID_buffer[0] = 0;
    HID_buffer[2] = (uint8_t)(4 + (ch - 'a'));
  }
  else if(ch >= 'A' && ch <= 'Z') {
    HID_buffer[0] = 2;    // Add left shift
    ch = ch - ('A'-'a'); // ch to lower case
    HID_buffer[2] = (uint8_t)(4 + (ch - 'a'));
  }
  else  {  // not a letter
    switch(ch)   {
      case ' ':  HID_buffer[2] = 44; break;
      case '~':  HID_buffer[0] = 8; // windows + r
                 HID_buffer[2] = 21; break;
      case '-':  HID_buffer[2] = 45; break;
      case '.':  HID_buffer[2] = 55; break;
      case '/':  HID_buffer[2] = 56; break;
      case '\n': HID_buffer[2] = 40; break;
      case ':':  HID_buffer[0] = 2;  // SHIFT ;
                 HID_buffer[2] = 51; break;
      case '?':  HID_buffer[0] = 2;  // SHIFT /
                 HID_buffer[2] = 56; break;
      default:   HID_buffer[2] = 0;
    }
  }
  // press keys
  ret = USBD_HID_SendReport(&hUsbDeviceFS, HID_buffer, 8);
  if(ret != HAL_OK) Error_Handler();
  HAL_Delay(15);

  // release keys
  HID_buffer[0] = 0;
  HID_buffer[2] = 0;
  ret = USBD_HID_SendReport(&hUsbDeviceFS, HID_buffer, 8);
  if(ret != HAL_OK) Error_Handler();
  HAL_Delay(15);
}
```

A kódokat itt USA billentyűzetkiosztás szerint küldjük!

F103_ és F446RE_HID_kwikboard

- Az előző demóprogramot kis erőfeszítéssel hasznos alkalmazássá fejlesztjük (csak a **GPIO konfigurálás** és a **main.c** tartalma tér el az előző programtól):
  - Nyomógombokat definiálunk
  - Ezekhez tetszőleges szövegeket rendelünk
- Példák, lehetőségek:
  - Gyakran használt weboldalak megnyitása
  - Bejelentkezési jelszavak beírása
  - Online bankolás vagy bankkártya adatainak beírása

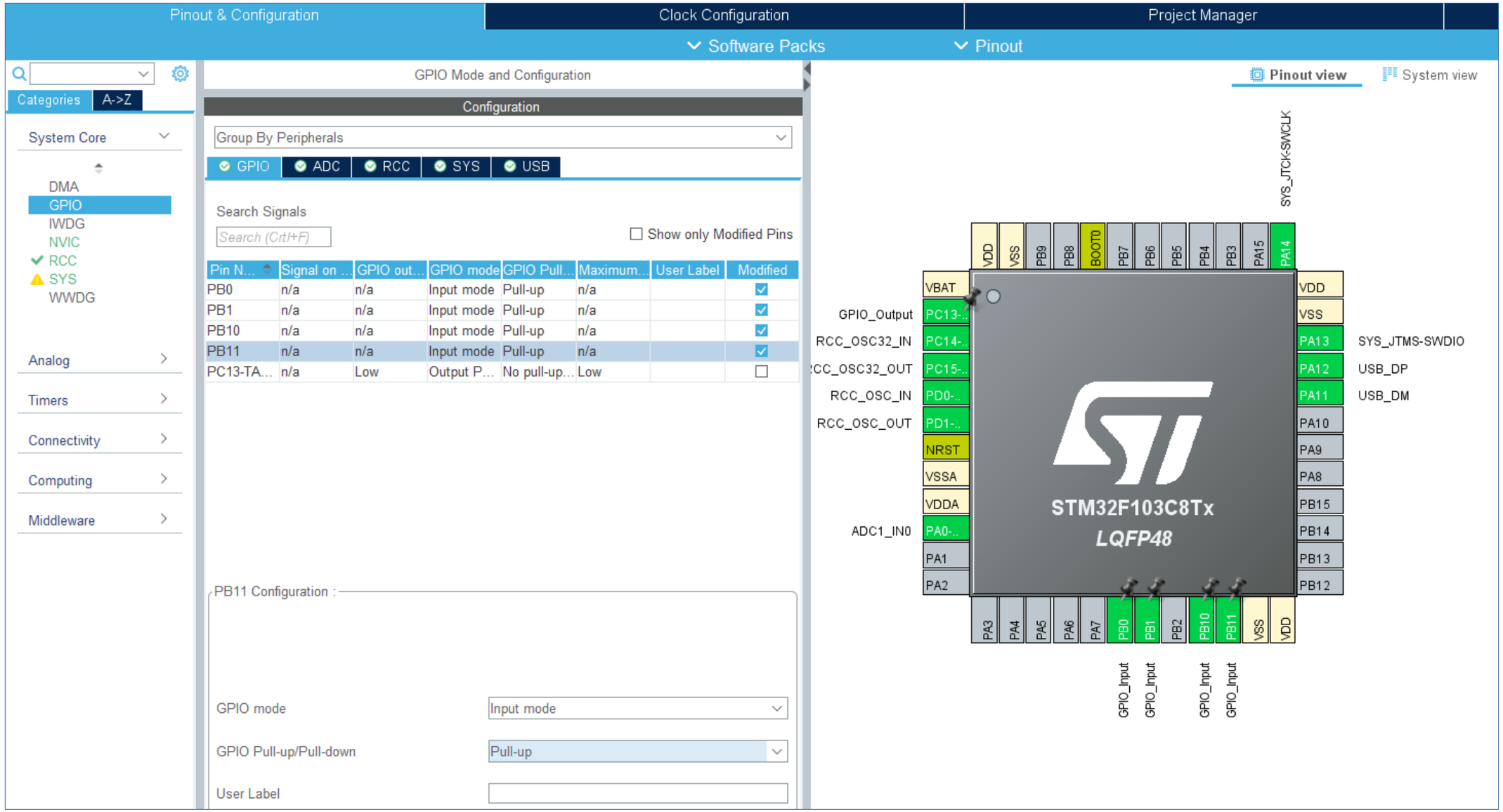
Ebben a programban mindkét mikrovezérlő esetében a magyar billentyűzetkiosztásnak megfelelő kódokat küldjük ki!

- A következő példában 4 db nyomógombot definiálunk és ezekhez webcímeket rendelünk:

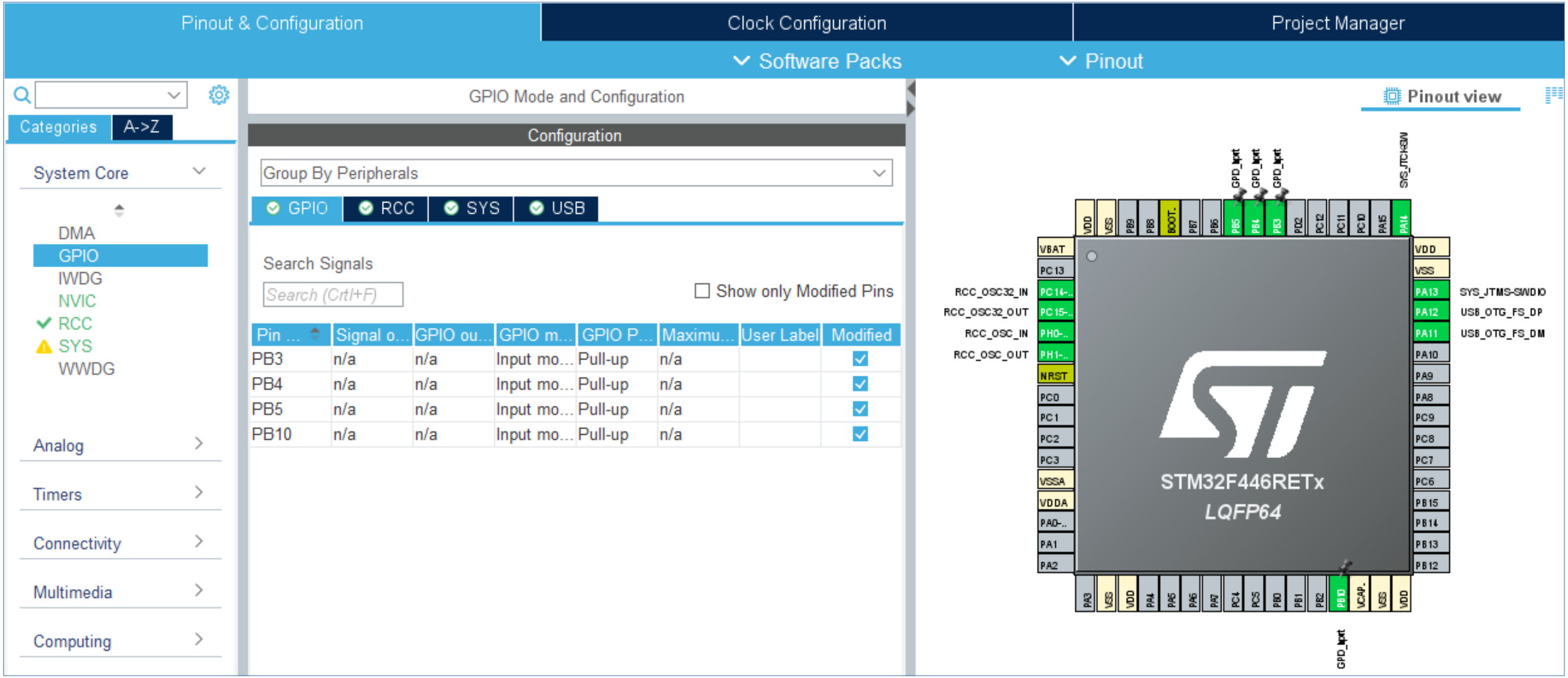
#	F103C8	F446RE	Tárolt szöveg
1	B0	B10	"~http://megtestesules.info/\n"
2	B1	B4	"~http://megtestesules.info/hobbielektronika/2020.html\n"
3	B10	B5	"~http://cspista.hu/\n"
5	B11	B3	"~http://magzarkurir.hu/\n"

Z - Y csere miatt

F103_HID_kwikboard: GPIO konfigurálás



F446RE_HID_kwikboard: GPIO konfigurálás



F446RE_HID_kwikboard: main.c (részletek)

- A **main.c** állomány változódeklarációs része így módosul:

```
uint8_t HID_buffer[8] = { 0 };                                          //--- HID TX buffer
uint8_t button_state[4] = {1, 1, 1, 1};
uint16_t SW_pin[4] = {GPIO_PIN_10, GPIO_PIN_4, GPIO_PIN_5,GPIO_PIN_3};  // B10, B4, B5, B3
char url[4] [80] = {"~http://megtestesules.info/\n",
                    "~http://megtestesules.info/hobbielektronika/2020.html\n",
                    "~http://cspista.hu/\n",
                    "~http://magzarkurir.hu/\n" };
```

- A **main()** függvény végtelen ciklusa így módosul:

```
  while (1) {
    for (int i = 0; i<4; i++) {
      if(button_state[i] ==1 && !HAL_GPIO_ReadPin(GPIOB,SW_pin[i])) {      // if waited for press and pressed
          button_state[i] = 0;                  // should wait for release next time
          USB_Keyboard_SendString(url[i]);
      }
      else if(button_state[i] == 0 && HAL_GPIO_ReadPin(GPIOB,SW_pin[i])) { // if waited for release and released
          button_state[i] = 1;                  // should wait for keypress next time
      }
    }
    HAL_Delay(50);
  }
```

F446RE_HID_kwikboard: main.c (részletek)

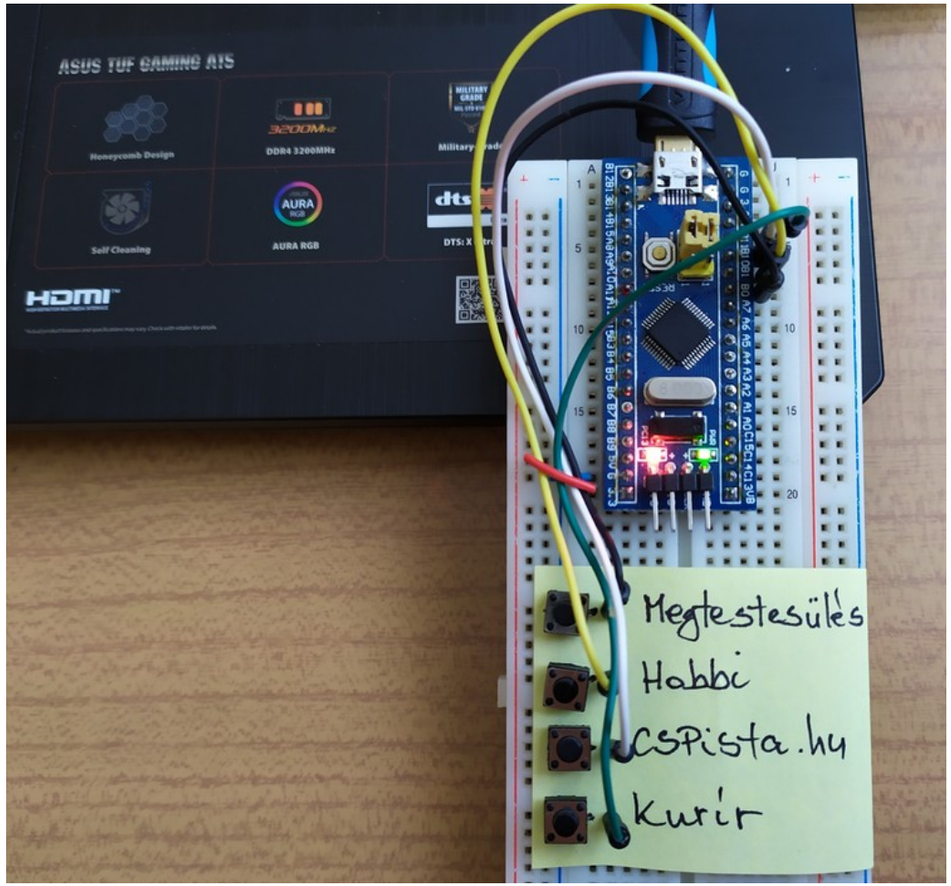
- Az **USB_Keyboard_SendChar()** függvény törzsének eleje ezzel bővül:

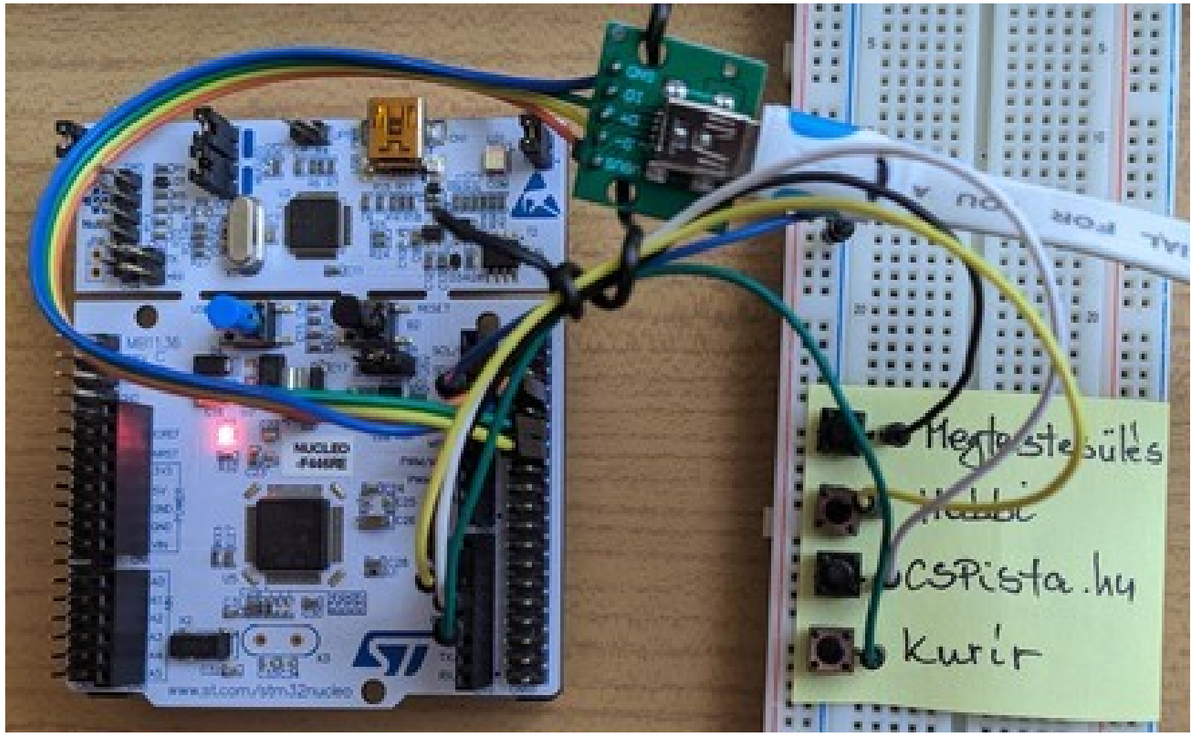
```
else if(ch > '0' && ch <= '9')
{
     HID_buffer[0] = 0;
     // convert 1-9 to scan code, starting at '1' = 0x1E
     HID_buffer[2] =(uint8_t)ch - 19;
}
```

- A **switch()** utasítást is ki kell bővíteni a nulla kezelésével:

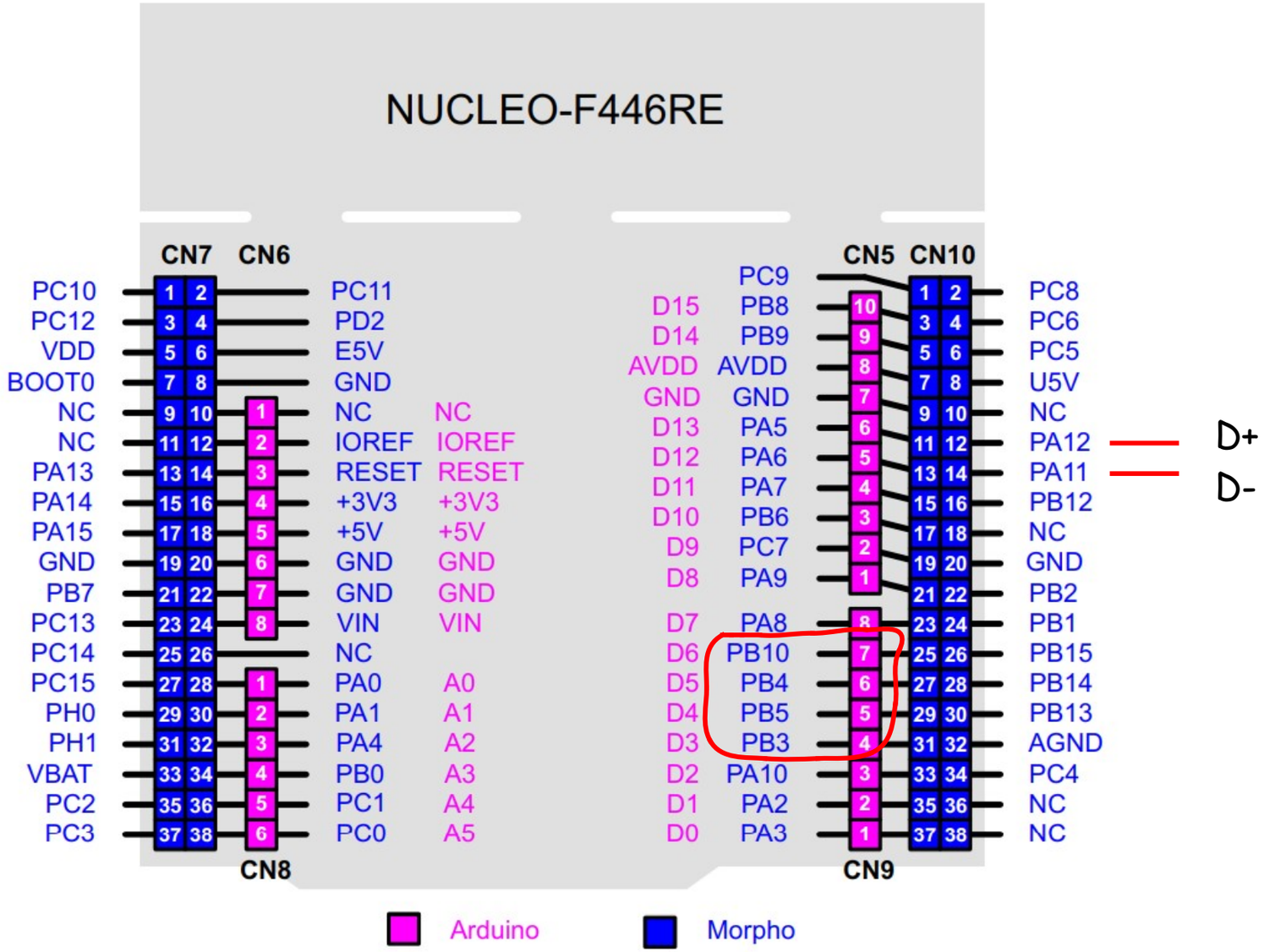
```
else // not a letter
{
     HID_buffer[0] = 0;
     switch(ch)
     {
          case '0':
          HID_buffer[2] = 0x35;
               break;
          case ' ':
               HID_buffer[2] = 44;
               break;
         . . .
```

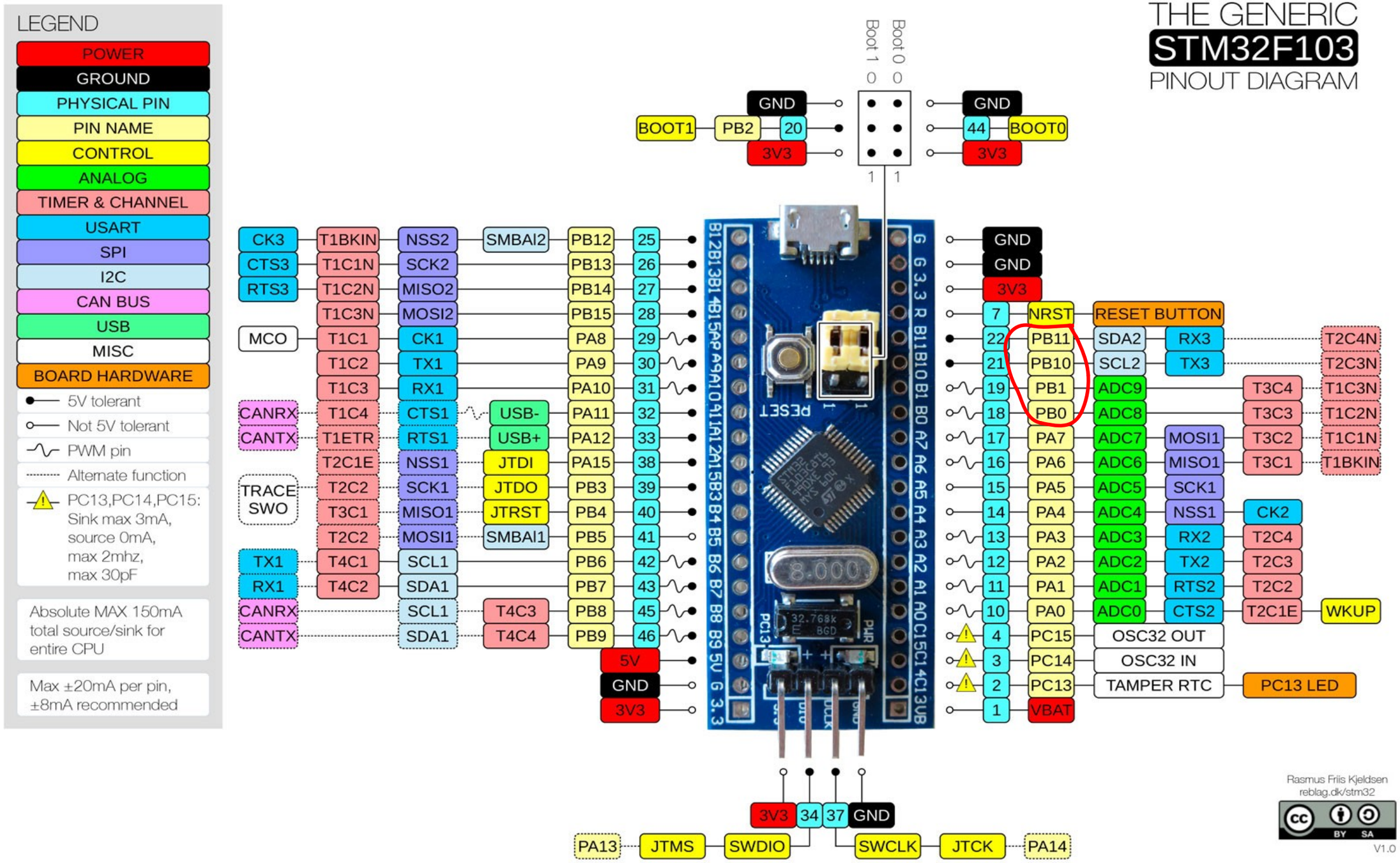
F103_ és F446RE_HID_kwikboard





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