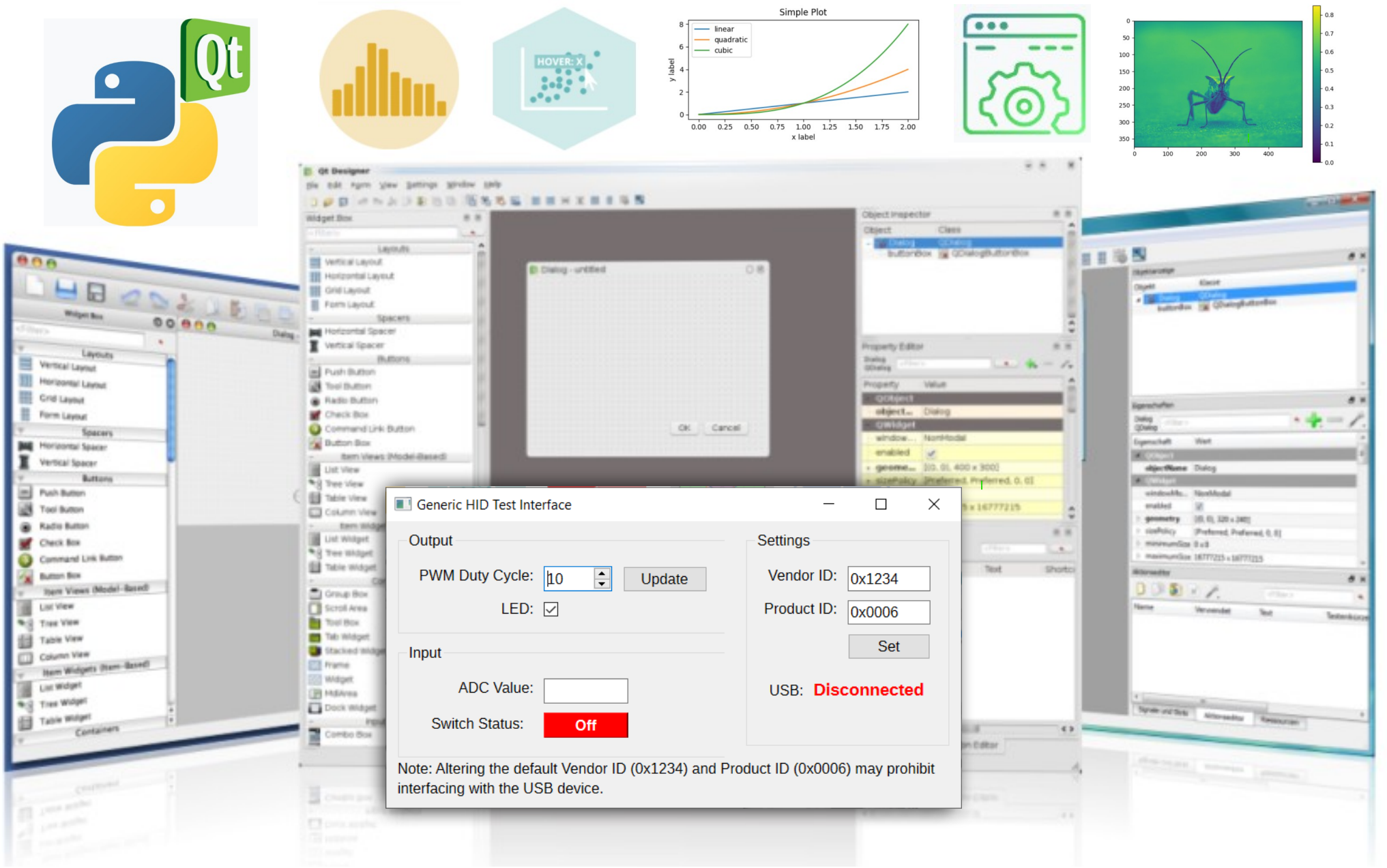
5. Python grafikus bővítmények használata – 2. rész



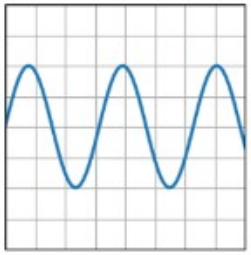
Felhasznált és ajánlott irodalom

- Mark Pilgrim/Kelemen Gábor: Ugorj fejest a Python 3-ba!
- P. Wentworth et al. (ford. Biró Piroska, Szeghalmy Szilvia és Varga Imre): Hogyan gondolkozz úgy, mint egy informatikus: Tanulás Python 3 segítségével
- Burkhard A. Meier: Python GUI Programming Cookbook
- Martin Fitzpatrick: Create GUI Applications with Python and Qt6 (Pyside6)
- **René Aguirre: pywinusb: USB/HID windows helper library**
- wxPython Homepage
- wxPython Tutorial
- **Pyside6 tutorial**
- **Qt6 for Python Documentation**

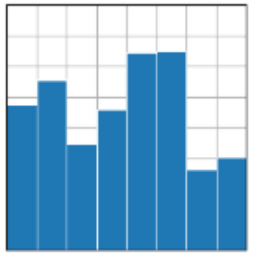
Grafikus könyvtárak és keretrendszerek

- A Python programok grafikus elemekkel való kiegészítése többféle módon és célirányban végezhető

Adatok ábrázolása (diagramok)

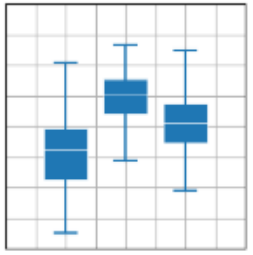


plot(x, y)



bar(x, height)

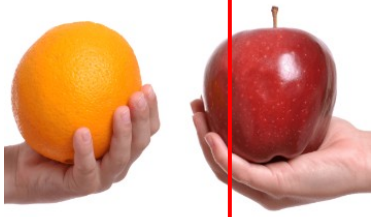
contour(X, Y, Z)

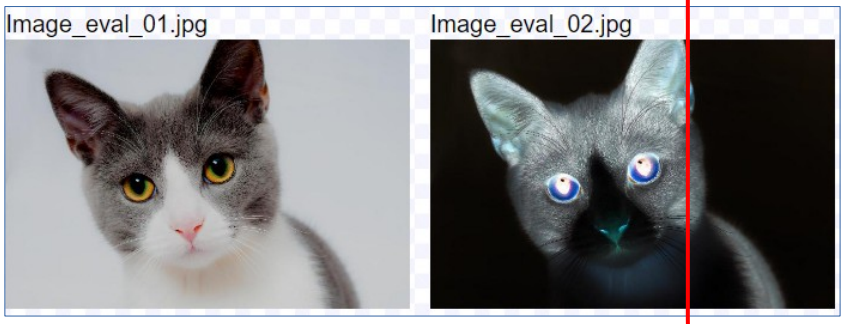


boxplot(X)

(Matplotlib, Pandas)

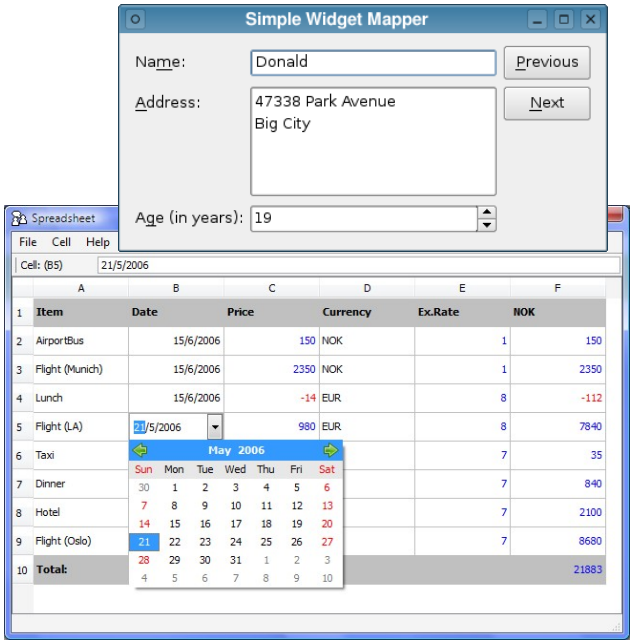
Médiafájlok kezelése és feldolgozása





(PIL, Pillow)

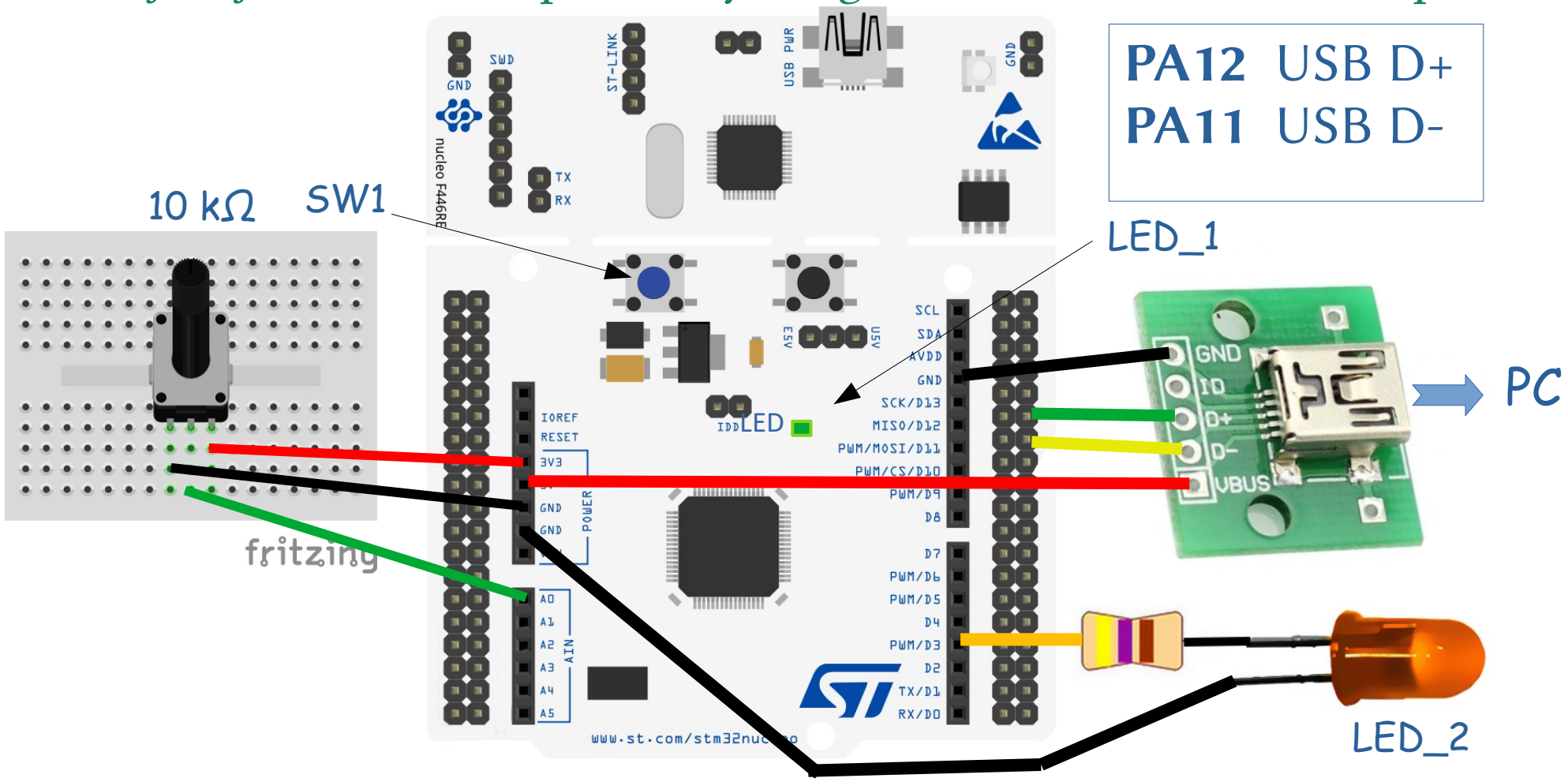
GUI alkalmazások (grafikus felhasználói felület)



(Tkinter, wxPython, PyQt)

mbed6_usbHID_PWM

- USB generic HID eszközként egy NUCLEO-F446RE kártyát használunk, amelyen a korábban bemutatott **mbed6_usbHID_PWM** program fut
- Az USB eszközön 8 bájtos csomagokkal vezérelhetjük a két LED-et, illetve 8 bájtos jelentéseket kapunk a nyomógomb és az **A0** bemenet állapotáról



mbed6_usbHID_PWM

- **Input Report:** a mikrovezérlő küldi automatikusan, 1 ms-onként, első bájtja a **B1** (SW1) nyomógomb bemenet állapota, a következő 4 bájt (high endian sorrendben) az **A0** analóg csatorna ADC konverziójának eredménye (kössünk ide egy potmétert, vagy egy analóg szenzort). A 4 bájtból mi csak az utolsó kettőt fogjuk használni, mivel az eszközünk 16 bites eredményeket küld.
- **Output Report:** (a PC küldi) vezérlési funkciót lát el: az első bájt a beépített LED-et vezérli (0: ki, 1: be), a második bájt pedig a **D3** kimenetre kötött LED-et vezérli (1 - 100 közötti érték), ami a **PWM** jel százalékos kitöltését adja meg). LED_1 tehát csak ki/be kapcsolgatható, LED_2-nek pedig a fényereje impulzus-szélesség modulációval (PWM) állítható, 100 fokozatban.

A 8 bájtos adatcsomagokat csak a firmware és a PC alkalmazás fogja tudni értelmezni! Az ADC adat nálunk 16 bites...

Input Report Data

Button Status	:Byte 1
~~ADC_Data[31..24]~~	:Byte 2
~~ADC_Data[23..16]~~	:Byte 3
ADC_Data[15..8]	:Byte 4
ADC_Data[7..0]	:Byte 5
Unused (0x00)	:Byte 6
Unused (0x00)	:Byte 7
Unused (0x00)	:Byte 8

Output Report Data

LED Control	:Byte 1
PWM Duty Cycle	:Byte 2
Unused (0x00)	:Byte 3
Unused (0x00)	:Byte 4
Unused (0x00)	:Byte 5
Unused (0x00)	:Byte 6
Unused (0x00)	:Byte 7
Unused (0x00)	:Byte 8

pywinusb

- **René Aguirre: pywinusb: USB/HID windows helper library**
- Legutolsó kiadása: **ver 0.4.2** (2016. december 12)
  egyelőre csak Windows alatt használható
- Telepítése (ha a Python már telepítve van): `pip install pywinusb`
- Importálása: `from pywinusb import hid`
- Kompatibilitás: **Python 2.7,** illetve **Python 3.5** (eddigi tapasztalataink szerint maga a könyvtár **Python 3.10** alatt is működik, a mintapéldák azonban nem)
- Szűkszavú leírás a **pywinusb wiki oldalán** található
- Mintapéldák: a **github.com/rene-aguirre/pywinusb** címen elérhető programcsomag **examples** mappája tartalmaz néhány mintapéldát, azonban kompatibilitási okoból ezek többségét módosítások nélkül nem tudjuk használni (a mai előadásban bemutatott programokat már módosítottuk, hogy **Python 3.10** alatt működjenek)

Pywinusb egyszerű használata

Többnyire az alábbi lépéseket kell követnünk

- Importáljuk a **pywinusb.hid** alcsomagot:

```
from pywinusb import hid
```

- Definiáljunk egy **HidDeviceFilter()** objektumot az USB eszközünk kiválasztásához! Például:

```
# using vendor id and product id
filter = hid.HidDeviceFilter(vendor_id=0x1234, product_id=0x0006)
```

- Kérjük le a kiszűrt HID eszközök listáját, vegyük pl. a legelsőt:

```
all_devices = filter.get_devices()
device = all_devices[0]
```

- Megnyitás, adatfogadó fv. hozzárendelés, írás, illetve lezárás:

```
device.open()
device.set_raw_data_handler(sample_handler)
out_report = device.find_output_reports()
out_report[0].set_raw_data(buffer)
out_report[0].send()
device.close()
```

küldés

Az adatokat fogadó függvény

USB HID plug and play események

- A **Pywinusb** csomag **HidPnPWindowMixin** osztálya lehetőséget biztosít az **USB HID pnp** értesítések fogadására, feltéve, ha Windows eszközt kezelő GUI alkalmazásról van szó
- Ezek az értesítések jelzik, ha egy HID eszköz csatlakozott a rendszerhez, vagy lecsatlakozott róla, s egy-egy ilyen esemény után ellenőriznünk kell, hogy az általunk használni kívánt eszköz csatlakoztatva van-e, vagy valaki kihúzta…
- A használt GUI könyvtártól függetlenül a Windows értesítési láncának eléréséhez a futó alkalmazás kiinduló ablakának „fogantyújára” lesz szükség
- A **Pywinusb** csomag alábbi pnp mintapéldáit aktualizáltuk:
  - **pnp_sample** (Python 3.10, Pywinusb 0.42, wxPython 4.2.0)
  - **pnp_sample_qt** (Python 3.10, Pywinusb 0.42, PySide6 6.3.2)
  - **pnp_qt** (Python 3.10, Pywinusb 0.42, PySide6 6.3.2)

pnp_sample.py

- Egyszerű Plug and Play példa, amely a **wxPython** GUI könyvtárat használja, de egyszerűen átírható más GUI könyvtárhoz, csak át kell adni az ablak „fogantyúját” a **HidPnPWindowMixin.__init__** inicializálási függvénynek (később bemutatjuk a Qt6-ra átírt változatot is...)
- A **MyFrame** származtatott osztály a **hid.HidPnPWindowMixin** osztály tulajdonságait és tagfüggvényeit is megörökli, így inicializálás után bekerül a HID értesítési láncba, így minden HID eszköz le- és felcsatlakoztatásáról értesül
- Az értesítési események kezeléséhez felül kell definiálni az **on_hid_pnp()** függvényt, amelyben megvizsgálhatjuk, hogy fel-, vagy lecsatlakozás történt, s hogy a saját eszközünkre vonatkozik-e a jelzett esemény
- A főprogramban példányosítani kell egy **wx.App** alkalmazást, egy **MyFrame** típusú ablakot (meg is kell jeleníteni) és futtatni kell a **wx.App** alkalmazás eseménykezelőjét

pnp_sample.py – 3/1.

A **MyFrame** osztály a **wx.Frame** és a **hid.HidPnPWindowMixin** osztályok leszármazottja

```
import wx
import pywinusb.hid as hid

target_vendor_id  = 0x1234
target_product_id = 0x0006

class MyFrame(wx.Frame,hid.HidPnPWindowMixin):
    # a device filter so we could easily discriminate wich devices to look at
    my_hid_target = hid.HidDeviceFilter(vendor_id = target_vendor_id, product_id = target_product_id)

    def __init__(self,parent):
        wx.Frame.__init__(self,parent,-1,
            title="Re-plug your USB HID device, watch the command window!",
            size = (600,300))
        hid.HidPnPWindowMixin.__init__(self, self.GetHandle())   # Itt adjuk át az ablak „fogantyúját”
        self.Bind(wx.EVT_CLOSE, self.on_close)                   # Az ablak bezárásakor lefut az on_close fv.
        # wx.EVT_CLOSE(self, self.on_close)                      # Ez a hozzárendelési mód már elavult
        self.device = None                                       # nincs még hid eszközünk

        # kick the pnp engine
        self.on_hid_pnp()
```

pnp_sample.py – 3/2.

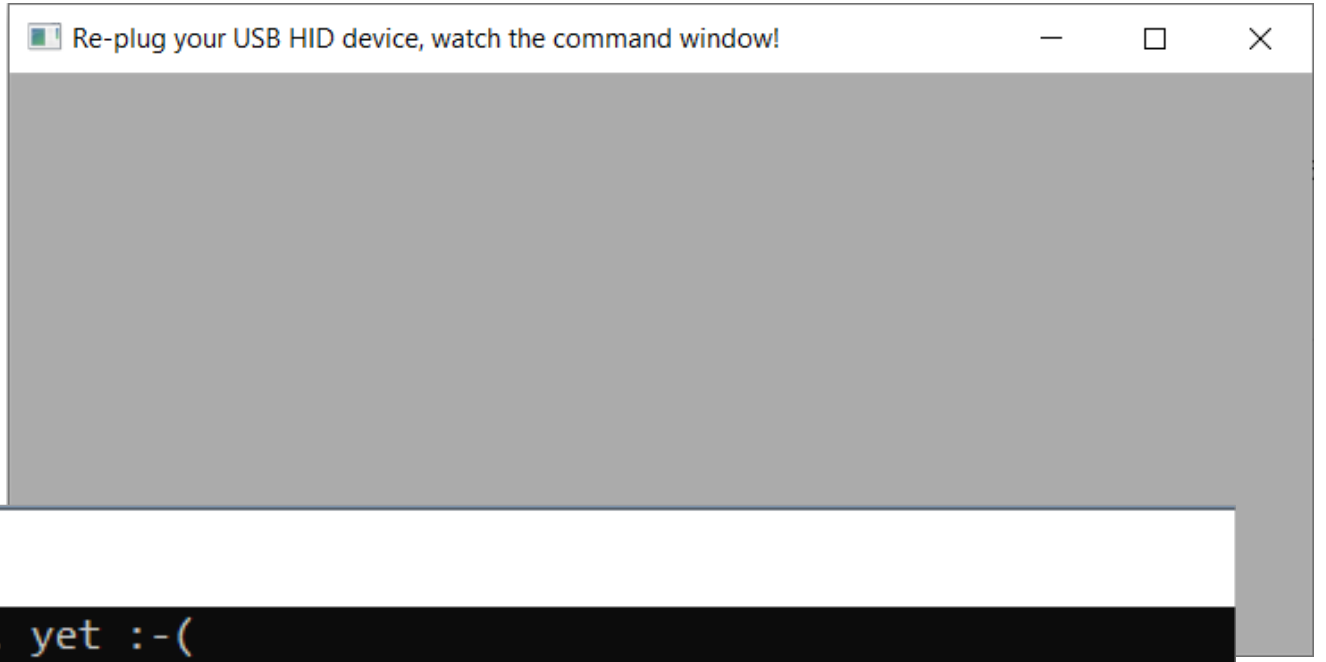
```
def on_hid_pnp(self, hid_event = None):      # Felüldefiniáljuk hid.HidPnPWindowMixin tagfüggvényét!
    old_device = self.device                 # keep old reference for UI updates
    if hid_event:
        print("Hey, a hid device just %s!" % hid_event)
    if hid_event == "connected":
        if self.device:                      # test if our device is available
            pass                             # we could detect multiple devices here
        else:
            self.test_for_connection()

    elif hid_event == "disconnected":
        if self.device and not self.device.is_plugged():
            self.device = None
            print("you removed my hid device!")
    else:
        self.test_for_connection()           # poll for devices

    if old_device != self.device:
        pass                                 # you may update ui here

def on_close(self, event):                   # Az ablak bezárásakor kilépés előtt lezárjuk a HID eszközt
    event.Skip()
    if self.device:
        self.device.close()
```

pnp_sample.py – 3/3.

```
    def test_for_connection(self):
        all_items =  MyFrame.my_hid_target.get_devices()
        if all_items:
            if len(all_items) == 1:
                self.device = all_items[0]              # this is easy, we only have a single hid device
            else:
                grouped_items = MyFrame.my_hid_target.get_devices_by_parent()
                print("%d devices now connected" % len(grouped_items))
                if len(grouped_items) > 1:
                    pass                                # you have multiple devices connected
                else:
                    pass                                # single physical device as a collection
                self.device = all_items[0]              # we just arbitrarly select the first hid object path
        if self.device:
            self.device.open()
            print("got my device: %s!" % repr(self.device))
        else:
            print("saddly my device is not here... yet :-( ")

if __name__ == "__main__":
    app = wx.App(False)
    frame = MyFrame(None)
    frame.Show()
    app.MainLoop()
```

pnp_sample.py

- A parancsablakban értesülünk a fel- és lecsatlakoztatásokról
- A grafikus ablak csak az értesítési lánc eléréséhez kell



C:\Windows\py.exe

```
saddly my device is not here... yet :-(
Hey, a hid device just connected!
got my device: HID device (vID=0x1234, pID=0x0006, v=0x0001); mbed.org; HID DEVICE,
Hey, a hid device just disconnected!
you removed my hid device!
Hey, a hid device just connected!
got my device: HID device (vID=0x1234, pID=0x0006, v=0x0001); mbed.org; HID DEVICE,
Hey, a hid device just disconnected!
you removed my hid device!
Hey, a hid device just connected!
got my device: HID device (vID=0x1234, pID=0x0006, v=0x0001); mbed.org; HID DEVICE,
```

pnp_sample_qt.py

- Ez csak egy „ujjgyakorlat”: az előző mintapéldát átírtuk **PySide6** GUI könyvtárhoz (Qt6 for Python)
- A **MyFrame** osztály itt a **QtWidgets.QMainWindow,** valamint a **hid.HidPnPWindowMixin** osztályok leszármazottja
- A főprogramban példányosítani kell egy **QtWidgets.QApplication** alkalmazást, egy **MyFrame** típusú ablakot (meg is kell jeleníteni) és futtatni kell a **QApplication** alkalmazás eseménykezelőjét
- Itt most a főprogramba került át az **on_close** függvény hozzárendelése a kilépés eseményhez, mivel a **PySide6**-ban a **QApplication** osztály **aboutToQuit** eseményjelzője az, amelynek **connect** metódusával elvégezhetjük a hozzárendelést, s ezt csak a **QApplication** osztály példányosítása után hívhatjuk meg

pnp_sample_qt.py – 2/1.

```
import sys
from PySide6.QtCore import Qt, QSize
from PySide6.QtWidgets import QApplication, QMainWindow
import pywinusb.hid as hid

target_vendor_id  = 0x1234
target_product_id = 0x0006

class MyFrame(QMainWindow,hid.HidPnPWindowMixin):
    # a device filter so we could easily discriminate wich devices to look at
    my_hid_target = hid.HidDeviceFilter(vendor_id = target_vendor_id, product_id = target_product_id)

    def __init__(self,parent):
        QMainWindow.__init__(self,parent)
        hid.HidPnPWindowMixin.__init__(self, self.winId()) <-------- # Az ablak „fogantyújának” átadása
        self.setWindowTitle("Re-plug your USB HID device, watch the command window!...")
        self.setFixedSize(QSize(500, 300))
        self.device = None                                  #no hid device... yet

        # kick the pnp engine
        self.on_hid_pnp()
```

pnp_sample_qt.py – 2/2.

```
    def on_hid_pnp(self, hid_event = None):
    (ugyanaz, mint az előző programban)

    def test_for_connection(self):
    (ugyanaz, mint az előző programban)

    def on_close(self):
        if self.device:
            self.device.close()

if __name__ == "__main__":
    app = Qapplication(sys.argv)                    # Ehelyett   app = Qapplication([])  is megteszi...
    form = MyFrame(None)
    app.aboutToQuit.connect(form.on_close)          <-------    # Kilépési esemény hozzárendelése
    form.show()
    app.exec()
    sys.exit()                                      # Ez a sor el is hagyható...
```

pnp_sample_qt.py

- A parancsablakban értesülünk a fel- és lecsatlakoztatásokról
- A grafikus ablak csak az értesítési lánc eléréséhez kell

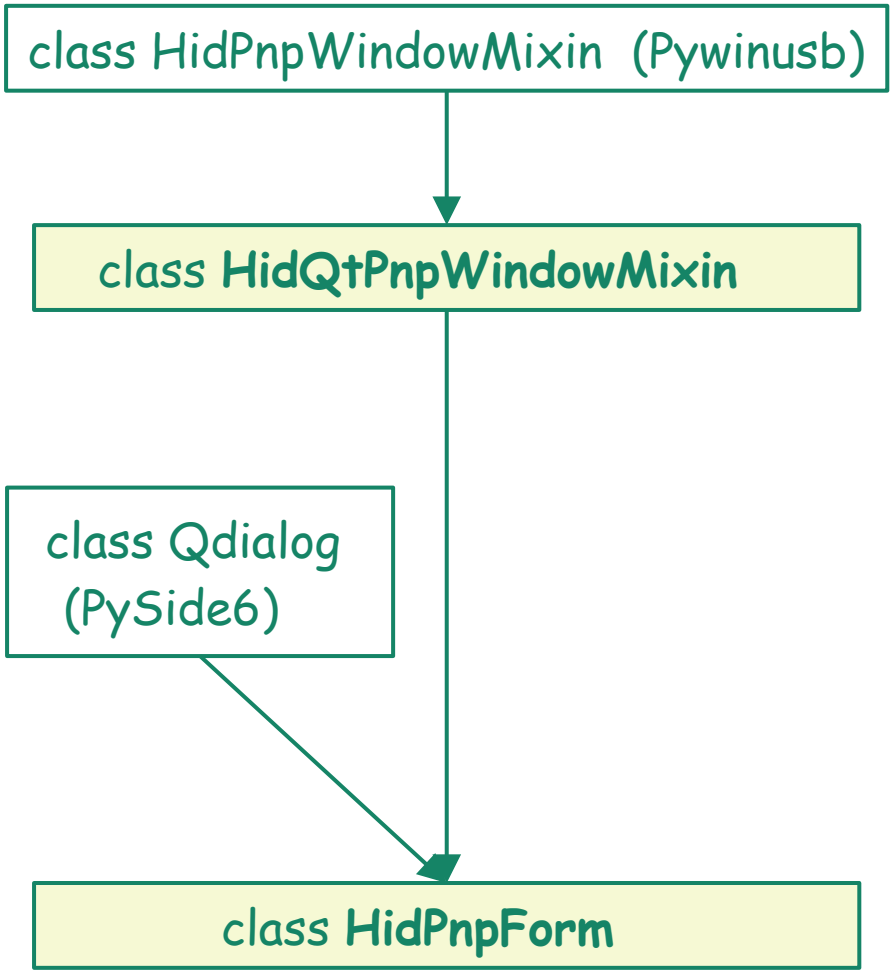
C:\Windows\py.exe

```
saddly my device is not here... yet :-(
Hey, a hid device just connected!
got my device: HID device (vID=0x1234, pID=0x0006, v=0x0001); mbed.org; HID DEVICE,
Hey, a hid device just disconnected!
you removed my hid device!
Hey, a hid device just connected!
got my device: HID device (vID=0x1234, pID=0x0006, v=0x0001); mbed.org; HID DEVICE,
Hey, a hid device just disconnected!
you removed my hid device!
Hey, a hid device just connected!
got my device: HID device (vID=0x1234, pID=0x0006, v=0x0001); mbed.org; HID DEVICE,
```

pnp_qt.py

- A **pnp_qt.py** mintapélda eredetileg **PySide 1.2.1, Pywinusb 0.3.6** és **Python 2.7.9**-hoz készült, ezért módosítani kellett, hogy **PySide 6.3.2, Pywinusb 0.4.2** és **Python 3.10** alatt működjön
- Ez a program is a **wxPython** GUI könyvtárat használó **pnp_sample.py** mintapélda egy átiratának tekinthető, de ebben már a grafikus ablak ún. widgeteket és többféle szerepet is kap:
  - Futás közben beírhatjuk a kiválasztott eszköz **Vid/Pid** azonosítóit
  - A **Set Test** gombbal vizsgálhatjuk az eszköz jelenlétét
  - A **pnp események** naplózása egy szövegablakban történik, így a terminál ablakra nincs szükségünk
- **Egy apró trükk:** a **.pyw** kiterjesztésű szkriptet terminál ablak megjelenítése nélkül futtathatjuk (ezt csak az alaposan letesztelt programnál érdemes használni, amikor már nem várható hibaüzenet)

pnp_qt.py



Properties:
pnpChanged (Qt6 Signal)
hidConnected (Qt6 Signal)
hid_device
hid_target

Methods:
__init__()
on_hid_pnp(hid_event)
test_for_connction()
set_target(hid_filter)

Methods:
__init__()
on_test()
on_close(result)
show_hids()
on_pnp_changed(event_str)
on_connected(my_device, event_str)

Active Widgets:
vID, pID, testButton, statusLabel, logText...

pnp_qt.py (részletek)

```
import sys
from PySide6.QtCore import Qt, Signal
from PySide6.QtWidgets import  (Qapplication, Qlabel, QLineEdit, Qdialog, QpushButton, QtextEdit,
                                QgridLayout )
import pywinusb.hid as hid
default_vendor_id  = 0x1234
default_product_id = 0x0006

class HidQtPnPWindowMixin(hid.HidPnPWindowMixin):
   pnpChanged = Signal(str)                          # PnP connection/disconnection signal
   hidConnected = Signal(hid.HidDevice, str)         # PnP connection/disconnection signal
   hid_device = None

   def __init__(self):
      hid.HidPnPWindowMixin.__init__(self, self.winId())      # pass system window handle to trap PnP event
      self.hid_device = None                                  # no hid device... yet
      self._hid_target = None                                 # no filter was defined... yet
      self.on_hid_pnp()                                       # kick the pnp state machine for one shot polling

   def set_target(self, hid_filter):
      self._hid_target = hid_filter
      self.test_for_connection()
      if self.hid_device:
         self.hidConnected.emit(self.hid_device, "connected")  # in case target set while already connected
```

pnp_qt.py (részletek)

```
class HidPnPForm(QDialog, HidQtPnPWindowMixin):
  def __init__(self, parent = None):
    super(HidPnPForm, self).__init__(parent)
    HidQtPnPWindowMixin.__init__(self)          # init PnP management
 . . .
    self.testButton.clicked.connect( self.on_test )
    self.finished.connect( self.on_close )

    # custom hid signals
    self.pnpChanged.connect( self.on_pnp_changed )
    self.hidConnected.connect( self.on_connected )

  def on_test(self):
    # Triggers HID polling and sets new PnP filter
    # a device so we could easily discriminate wich devices to look at
    vId = int(self.vendorIdText.text(), 16)
    pId = int(self.productIdText.text(), 16)
    self.statusLabel.setText( "waiting device ..." )
    self.set_target( hid.HidDeviceFilter(vendor_id = vId, product_id = pId) )

  def on_close(self, result):
    if self.hid_device is not None:
      self.hid_device.close()
```

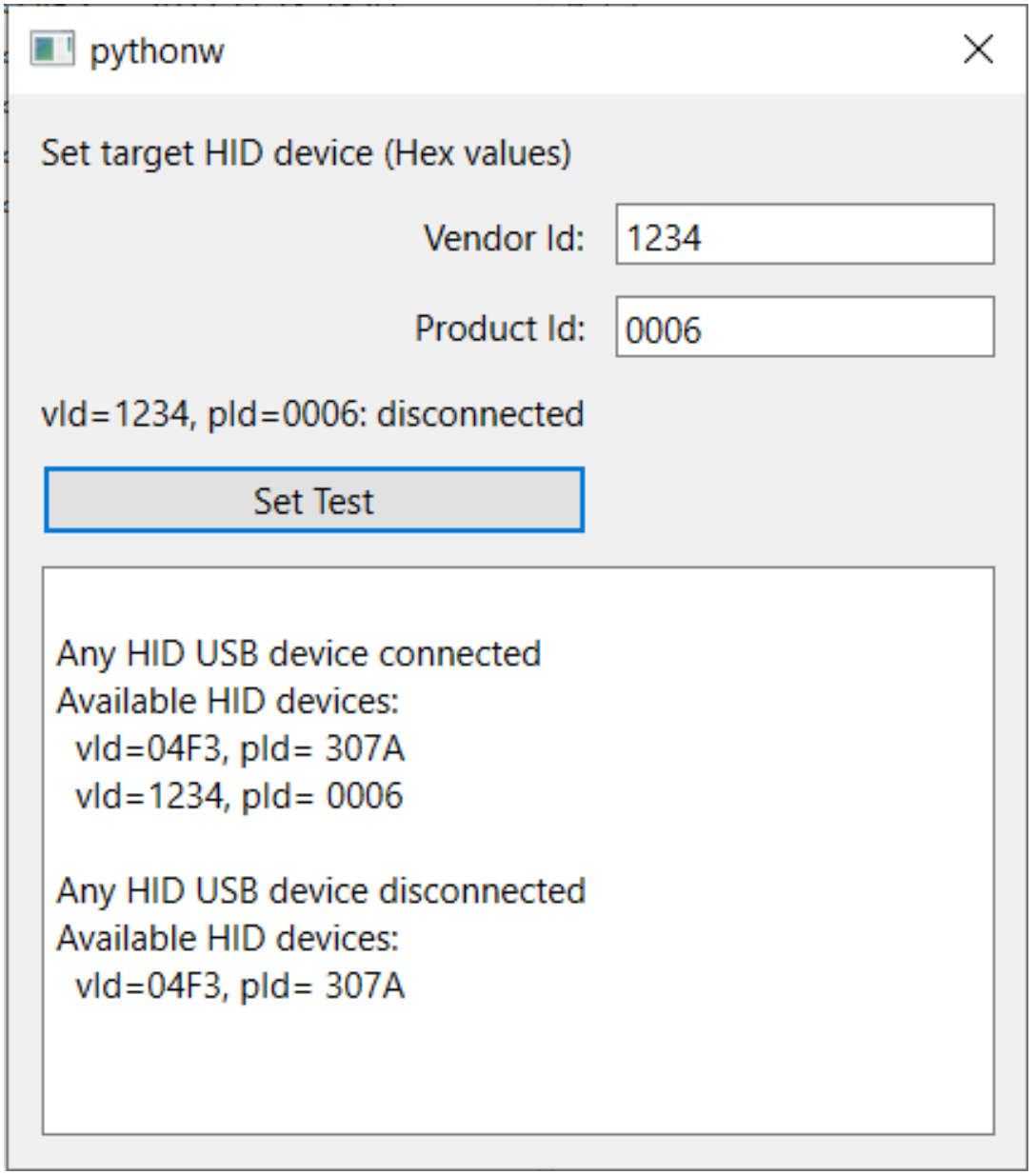
pnp_qt.py (részletek)

```
    def show_hids(self):                        # Lists all available HID devices if any...
        all_hids = hid.find_all_hid_devices()
        if all_hids:
            self.logText.append( "Available HID devices:" )
            for hid_device in all_hids:
                self.logText.append( "  vId={0:04X}, pId= {1:04X}".format(
                    hid_device.vendor_id, hid_device.product_id ))
        else:
            self.logText.append( "No HID USB devices attached now" )

    def on_pnp_changed(self, event_str):        # Lists HID connect or disconnect events
        self.logText.append( "\nAny HID USB device {}".format( event_str ))
        self.show_hids()

    def on_connected(self, my_hid, event_str): # Update statusLabel when our device state changed
        self.statusLabel.setText( "vId={0:04x}, pId={1:04x}: {2}".format(
            my_hid.vendor_id, my_hid.product_id, event_str ))

if __name__ == "__main__":
    app = QApplication(sys.argv)
    form = HidPnPForm()
    form.show()
    app.exec()
    sys.exit()
```

„connected" or „disconnected"

pnp_qt.py futtatás

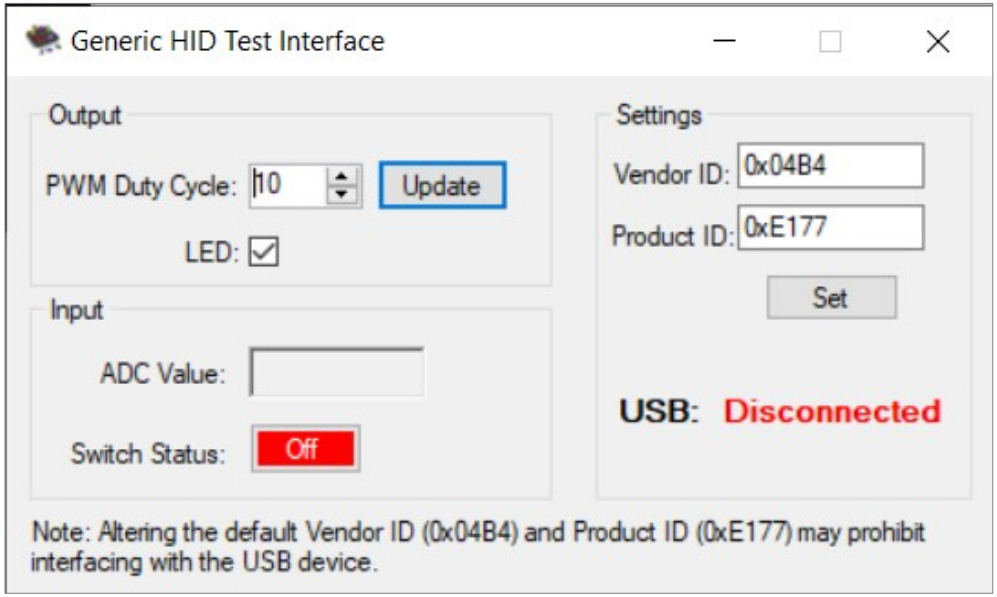
- A dialógusablakban megadhatjuk Vendor Id és Product Id értékét, ezek a **SetTest** gomb lenyomásakor érvényesülnek
- A nyomógomb fölötti kiírás jelzi a kiválasztott eszköz aktuális állapotát
- A szövegdobozban a program naplózza a HID pnp eseményeket és minden változás után kilistázza az éppen elérhető HID eszközöket (ha van ilyen)
- Ha módosítjuk a program nevét **pnp_qt.pyw**-ra, akkor nem jelenik meg a fölösleges parancs-ablak



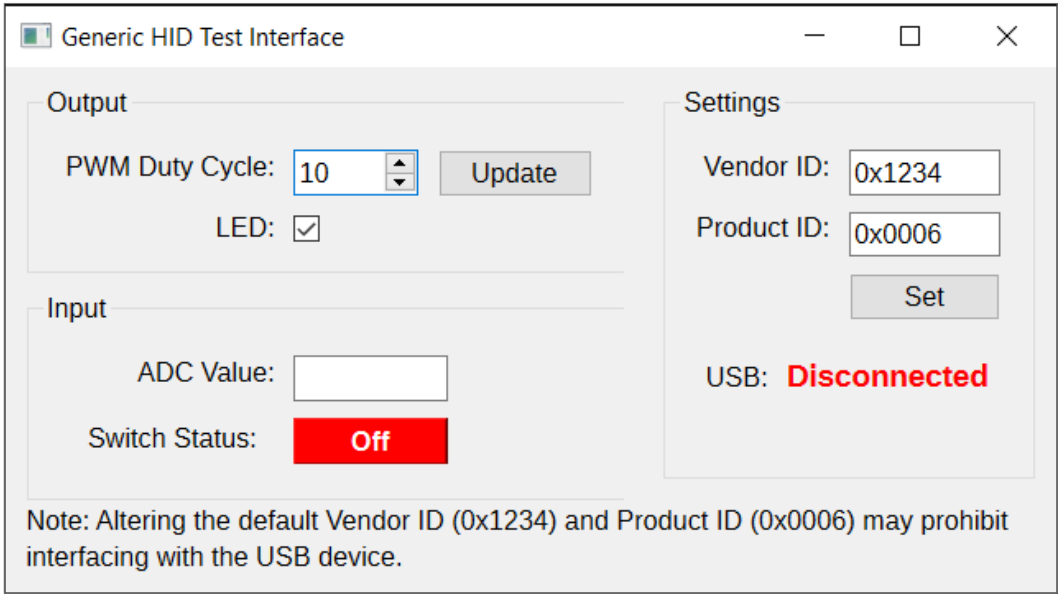
A HIDTest GUI alkalmazás

- Készítsünk a korábban bemutatott **mbed6_usbHID_PWM** programhoz GUI kezelő felületet! A cél a **Cypress AN82 072** alkalmazási mintapéldában található C# program kiváltása
  - 1. lépés: az elrendezés megtervezése **Qt Designerrel** (ld. előző előadás)
  - 2. lépés: a működtető programrész és az USB HID eszköz kezelésének megírása (ehhez a **PySide6** mellett kell a **Pywinusb** könyvtár is)

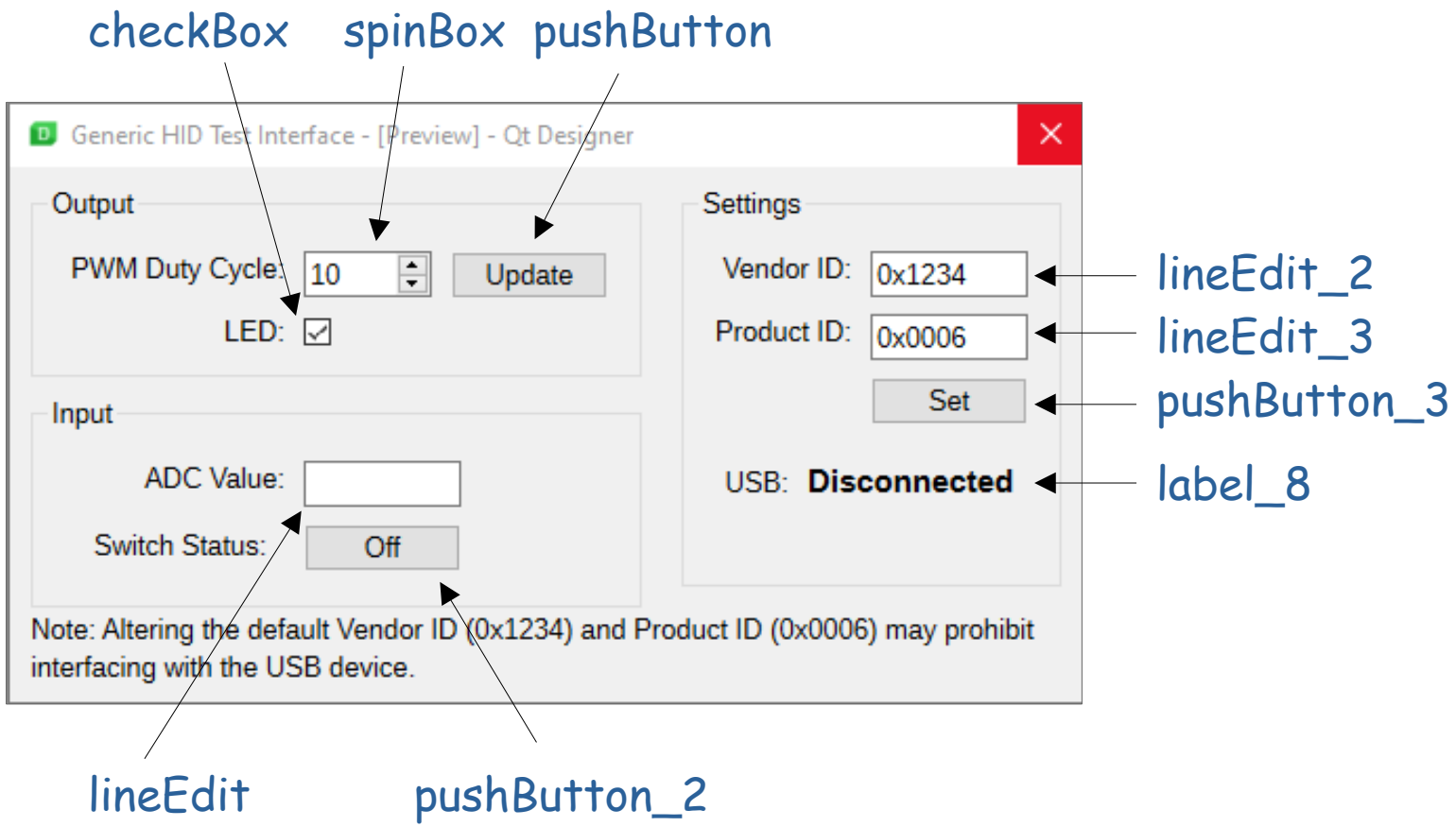
Az eredeti C# alkalmazás



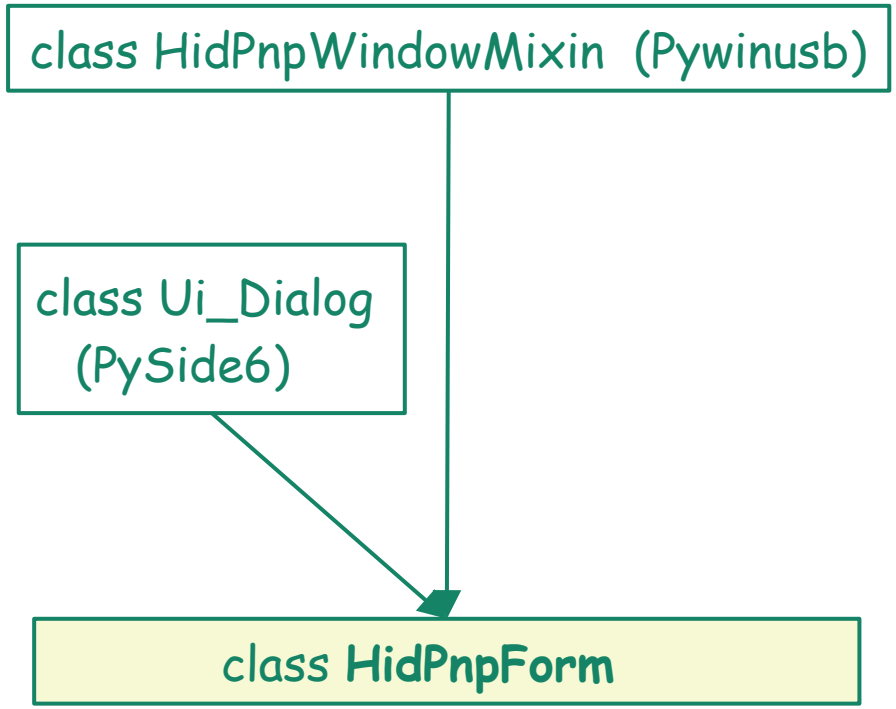
Az általunk készített Python alkalmazás



A használat során hivatkozott GUI elemek



HIDTest_app.pyw



A **pnp_sample_qt.py** programot dolgoztuk át és bővítettük ki. A GUI felületet és megvalósítását már az előző előadásban bemutattuk (HIDTest.ui, ill. HIDTest.py)

Properties:
vID, pID, pwm_data, led_state, sw1, adc, my_filter, device

Methods:

__init__()	
sample_handler(data)	(receive data)
send_command(led1,led2)	(send data)
on_update()	(pushButton)
on_set()	(pushButton_3)
on_check()	(checkBox)
on_hid_pnp(hid_event)	
test_for_connction()	

A **sample_handler(data)** és a **send_command(led1,led2)** függvényeket már az október 6-i előadásban használtuk (lásd **Lab03/usbhid_pwm.py**).
Az **on_hid_pnp(hid_event)** és a **test_for_connction()** függvények pedig már ismerősök a **pnp_sample.py** programból

HIDTest_app.pyw (részletek)

```
import sys
from PySide6 import QtWidgets
from ui_HIDTest import Ui_Dialog
import pywinusb.hid as hid
ix,sw1,adc = (0,0,0)

class MainWindow(Ui_Dialog, hid.HidPnPWindowMixin ):
    vID = 0x1234
    pID = 0x0001
    my_filter = hid.HidDeviceFilter(vendor_id = vID, product_id = pID)
    led_state = 1
    pwm_data =  10

    def __init__(self):
        super(MainWindow, self).__init__()
        self.setupUi(self)
        hid.HidPnPWindowMixin.__init__(self, self.winId())
        self.label_8.setStyleSheet("color: red")
        self.pushButton_2.setStyleSheet("color: white; background-color: red")
        self.pushButton_2.setText("Off")
        self.pushButton.clicked.connect(self.on_update)
        self.pushButton_3.clicked.connect(self.on_set)
        self.checkBox.stateChanged.connect(self.on_check)
        self.device = None      #no hid device... yet
        self.on_hid_pnp()       # kick the pnp engine
```

Események „bekötése"

HIDTest_app.pyw (részletek)

```
def sample_handler(self,data):
    global sw1;  global adc;  global ix
    sw1 = sw1 + data[1]
   adc = adc + data[4]*256 + data[5]
    ix = ix + 1
    if ix == 100:
        # self.lineEdit.setText("0x{0:04X}".format(adc//100))
        self.lineEdit.setText("{0:.3f} V".format(adc*3.3/6553600))
        if sw1>50:
            self.pushButton_2.setStyleSheet("color: white; background-color: green")
            self.pushButton_2.setText("ON")
        else:
           self.pushButton_2.setStyleSheet("color: white; background-color: red")
           self.pushButton_2.setText("Off")
        ix,sw1,adc = (0,0,0)

def send_command(self,led1,led2):   # Send the message to the Mbed board
    buffer = [0 for i in range(9)]           # The first byte is the report ID which must be 0
    buffer[1] = led1
    buffer[2] = led2
    if self.device is not None and self.device.is_plugged():
        out_report = self.device.find_output_reports()
        out_report[0].set_raw_data(buffer)
        out_report[0].send()
```

HIDTest_app.pyw (részletek)

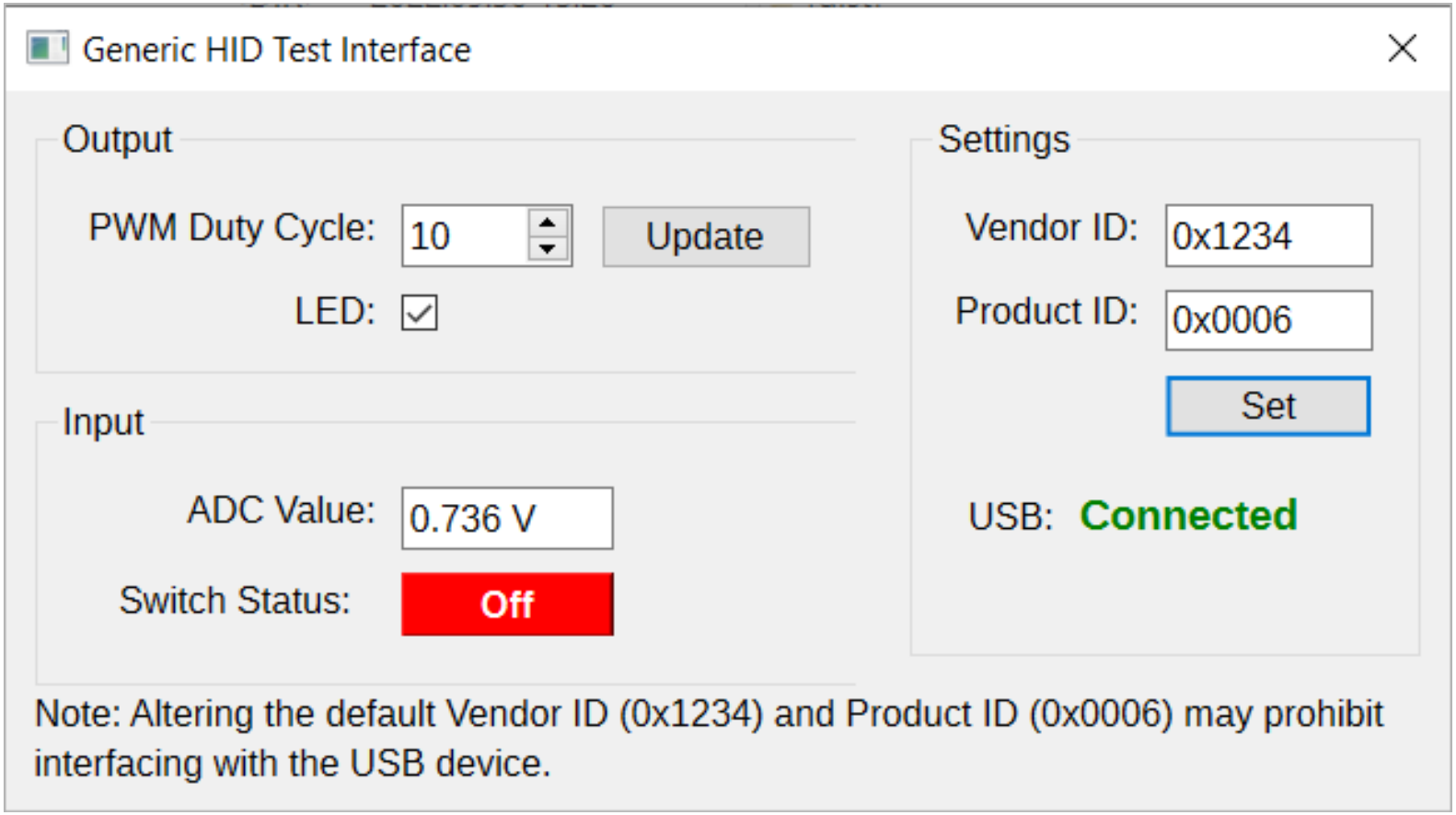
```
# Az Update nyomógomb lenyomásakor aktiválódik
def on_update(self):
    self.pwm_data = int(self.spinBox.value())
    self.send_command(self.led_state, self.pwm_data)

# A Set feliratú gomb lenyomásakor a vID és pID adatokkal aktualizáljuk my_filter-t
# és ellenőrizzük, hogy a megcímzett eszköz elérhető-e
def on_set(self):
    self.vID = int(self.lineEdit_2.text(), 16)
    self.pID = int(self.lineEdit_3.text(), 16)
    self.my_filter = hid.HidDeviceFilter(vendor_id = self.vID, product_id = self.pID)
    self.test_for_connection()

# CheckBox kattintásakor aktualizáljuk LED1 állapotát (led_state) és kiküldjük a LED vezérlő parancsot
def on_check(self,state):
    if state:
      self.led_state = 1
    else:
      self.led_state = 0
    self.send_command(self.led_state,self.pwm_data)
```

HIDTest_app.pyw (részletek)

```
    def test_for_connection(self):
        all_items =  self.my_filter.get_devices()
        if all_items:
            . . .
            self.device = all_items[0]

        if all_items and self.device:            # Ha a szűrt lista nem üres és device sem üres
            self.device.open()                   # Megnyitjuk a kiválasztott eszközt
            self.device.set_raw_data_handler(self.sample_handler)
            self.label_8.setText("Connected")
            self.label_8.setStyleSheet("color: green")
        else:
            if self.device is not None:          # Ha a korábban megnyitott eszköz már nincs a szűrőlistán
                self.device.set_raw_data_handler(None)
                self.device.close()
                self.device = None
            self.label_8.setText("Disconnected")
            self.label_8.setStyleSheet("color: red")

app = QtWidgets.QApplication([])
window = MainWindow()
window.show()
app.exec()
```

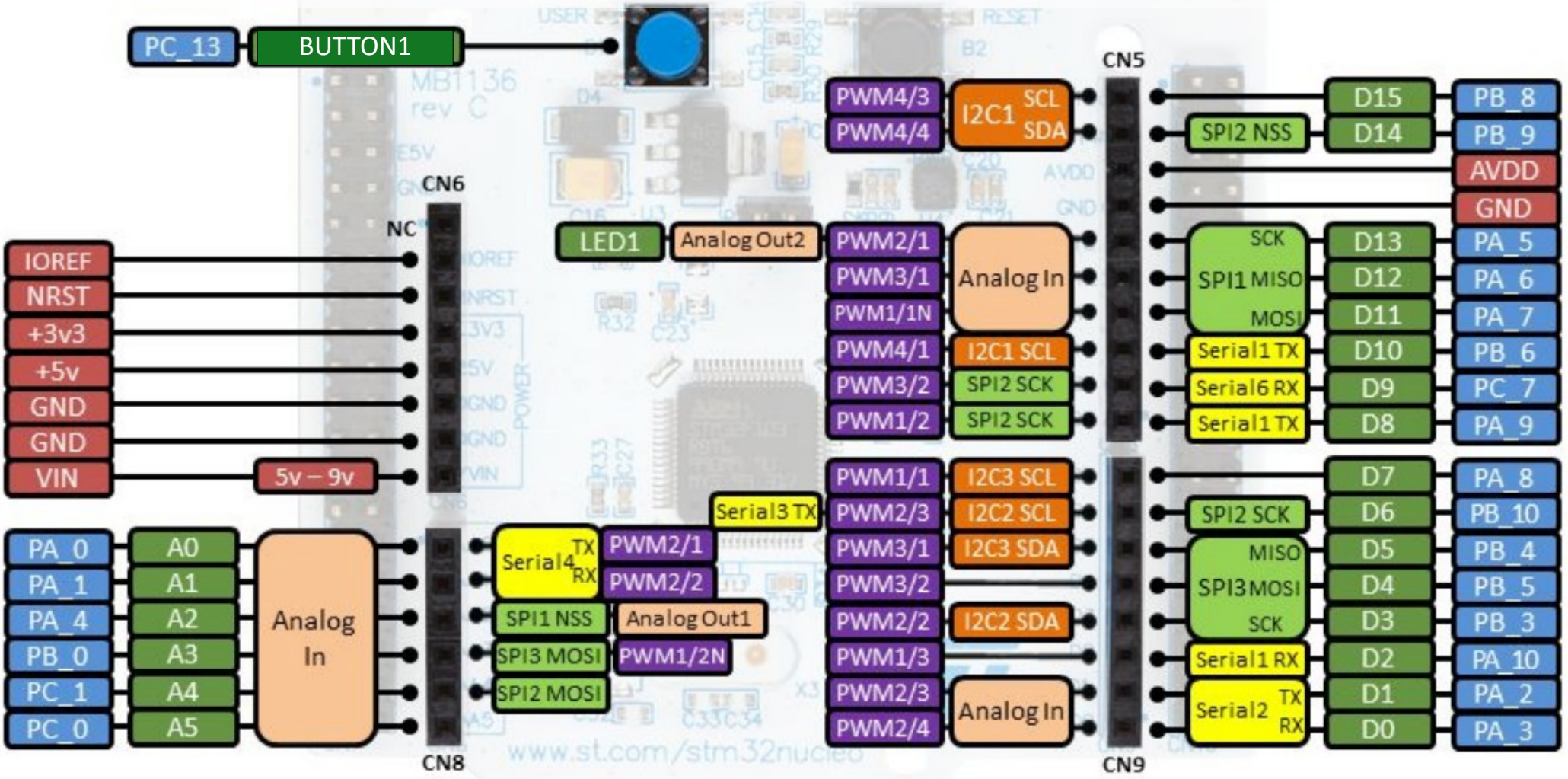
HIDTest_app.pyw futtatása

- Itte egy MCP9700A analóg hőmérő jelét kötöttük az analóg bemenetre. Az ábrán látható érték 23,6 °C-ot jelent



Arduino kompatibilis kivezetések

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



Morpho kivezetések

Nucleo F446RE

Morpho Headers