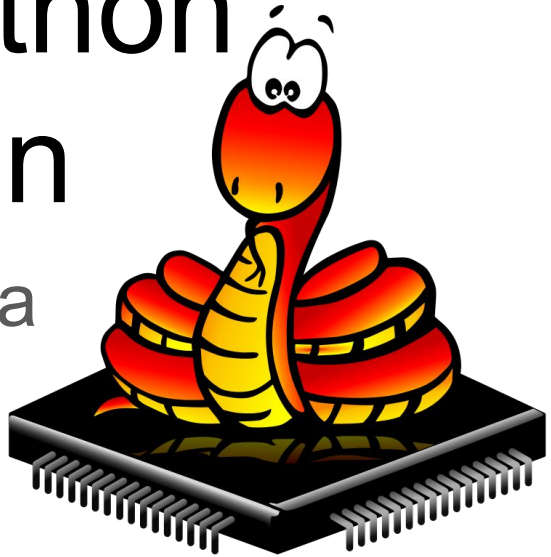
FT232H USB vezérlő programozása Python programnyelven

I2C protokoll használata



Felhasznált irodalom

- Adafruit: CircuitPython Libraries on any Computer with FT232H
- Adafruit: Adafruit FT232H Breakout
- FTDI: User Guide For libMPSSE – I2C
- Ben Gillett : I2C and more with the FTDI FT2232H Mini Module and an Arduino

Adatlapok

- Future Technology Devices International Ltd: FT232H Single Channel HiSpeed USB to Multipurpose UART/FIFO IC
- Measurement Specialties: HTU21D(F) Sensor

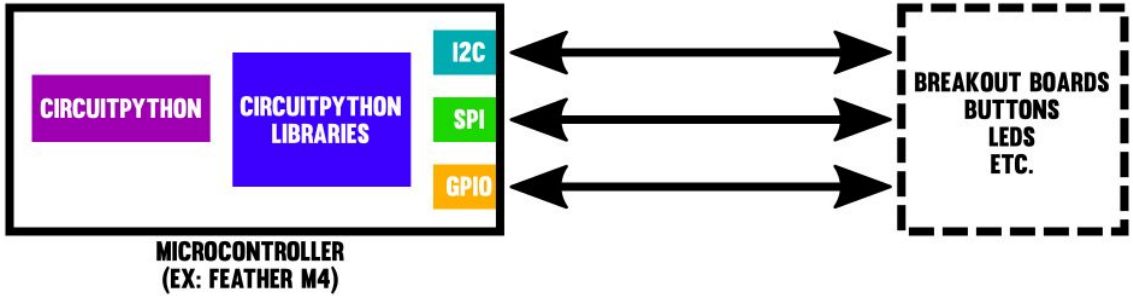
Közvetlen kapcsolat PC és alacsony szintű hardvereszköz között

Ha alacsony szintű hardver eszközzel szeretnénk kialakítani kapcsolatot az asztali számítógépről, akkor soros kommunikáción keresztül ez egyszerűen megvalósítható egy USB soros átalakítóval. Ellenben ha a hardvereszköz csak más soros kapcsolatra képes (I2C vagy SPI), akkor ehhez egy mikrovezérlő beiktatása szükséges amely közvetlenül kommunikál az eszközzel majd az eredményt kiértékeli és soros kapcsolaton keresztül elküldi a számítógép irányába. Ellenben van mód a PC-ről közvetlenül felvenni a kapcsolatot az eszközzel egy USB soros multiprotokoll eszközzel, ilyen pl az FT232H IC.

GPIO és soros kommunikáció kialakításának módjai

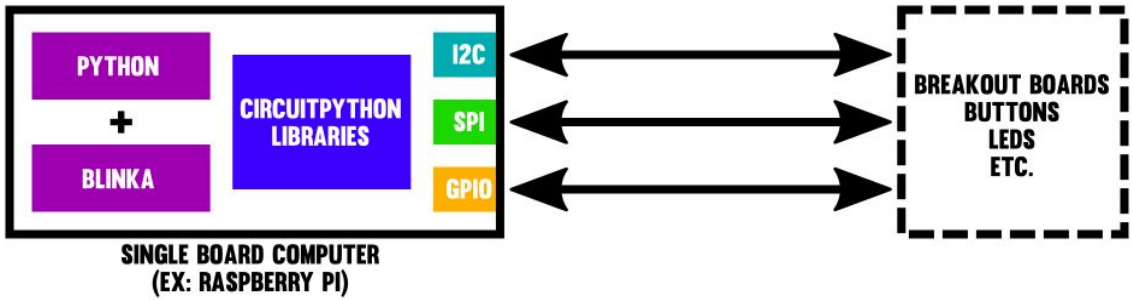
Mikrovezérlő programozása

A korábbi előadások témája már többször foglalkozott a soros kommunikáció kialakításával és a GPIO elérésével mikrovezérlők esetén. Arduino környezet használatával C++ programnyelven vagy fejlettebb mikrovezérlők esetén LUA, micropython esetleg circuitpython programnyelv használatával.
Ebben az esetben a mikrovezérlőre feltöltésre kerül a program, az a mikrovezérlőn fut és a PC felé kommunikálhat, de ilyenkor a kommunikációt is meg kell határozni és le kell programozni



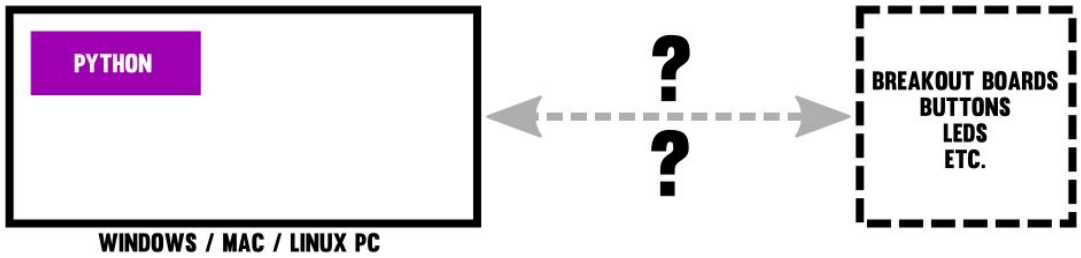
Single Board Computer (SBC) segítségével (Raspberry)

Ebben az esetben a program az SBC-n fut. Használható szintén C vagy C++ programnyelv, de itt már használható scriptnyelv is pl. python mivel ebben az esetben már nagyobb processzor teljesítmény és memória áll rendelkezésre. A PC-vel való kapcsolatra szintén használható soros kommunikáció vagy hálózaton keresztül is lehet kommunikálni, TCP/IP



PC közvetlen soros kommunikáció

Itt használható az FT232H vezérlő, hogy közvetlenül elérhető legyen egy érzékelő vagy GPIO be vagy kimenet. Az FT232H programozásával kihasználható a PC sebessége és gyors és széleskörűen beállítható hálózati elérése is. Közvetlenül C, C++, Delphi (minta kód elérhető az FTDI oldalán) vagy python programnyelvből elérhetőek az érzékelők SPI, I2C vagy JTAG kapcsolaton keresztül.



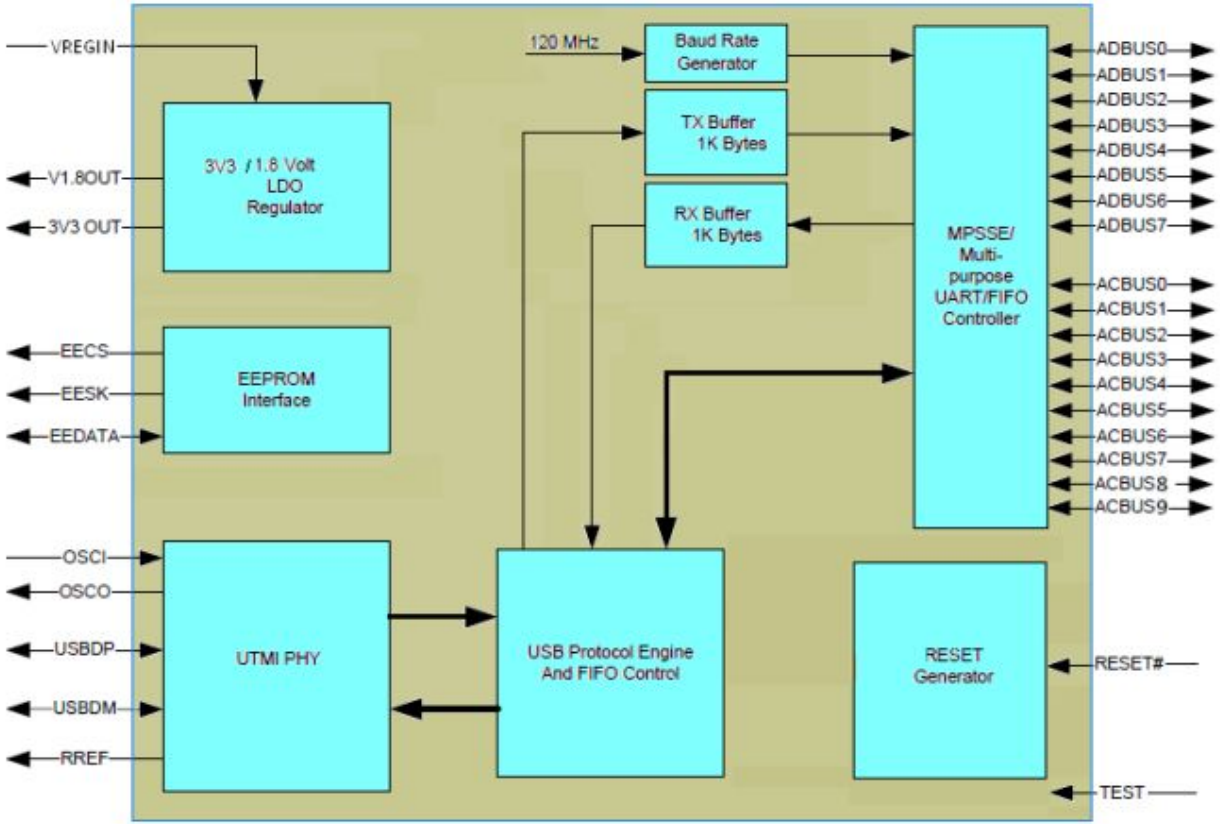
FT232H Single Channel HiSpeed USB to Multipurpose UART/FIFO IC

Nagy sebességgel (480 Mbps) működő gyors egycsatornás soros interface. Kialakítható segítségével JTAG, SPI, I2C és UART asszinkron és szinkron kommunikáció. Rendelkezik halfduplex F1248 busszal, amely lehetővé teszi 1,2,4 vagy 8 bites adatvonalon az akár 30Mbyte/s IO kommunikációt. Az IO lábak 5V tűréssel rendelkeznek.

- Egycsatornás USB soros/párhuzamos portokhoz különféle konfigurációkkal.
- A teljes USB-protokoll a chipen kezelve. Nincs szükség USB-specifikus firmware programozásra.
- USB 2.0 nagy sebességű (480 Mbit/másodperc) és teljes sebességű (12 Mbit/másodperc) kompatibilis.

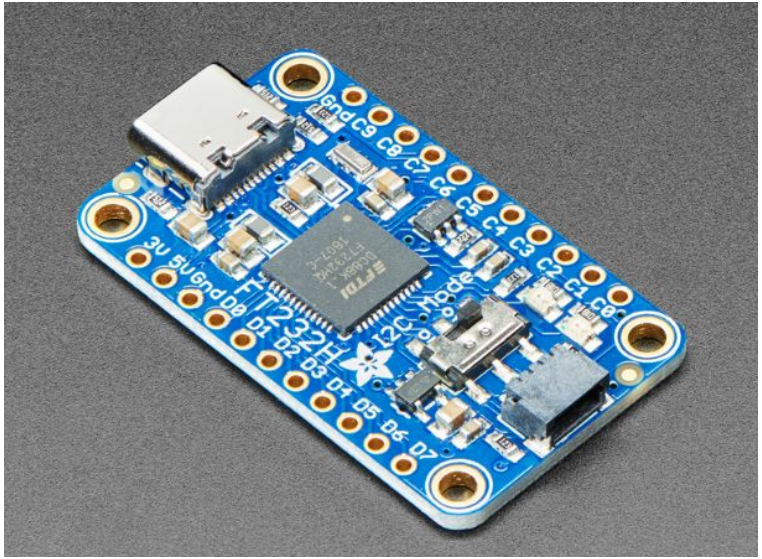
- Multi-Protocol Synchronous Serial Engine (MPSSE) a szinkron soros protokoll (USB to JTAG, I2C, SPI vagy bit-bang) tervezésének leegyszerűsítésére
- Az FTDI jogdíjmentes Virtual Com Port (VCP) és Direct (D2XX) illesztőprogramjai a legtöbb esetben kiküszöbölik az USB-illesztőprogram-fejlesztés követelményét.
- Állítható vételi puffer időtúllépés.
- Aszinkron soros UART interfész opció teljes hardveres kézfogással és modem interfész jelekkel.
- Teljesen támogatott hardveres vagy X-On / X-Off szoftveres kézfogás.
- Az UART interfész támogatja a 7/8 bites adatokat, az 1/2 stopbiteket és a páratlan/páros/jelölt/szóköz/nem paritást.
- USB tömeges adatátviteli mód (512 bájtos csomagok nagy sebességű módban).

- +1.8V (chipmag) és +3.3V I/O interfész (+5V toleráns).
- Kompakt 48 tűs ólommentes LQFP vagy QFN csomag



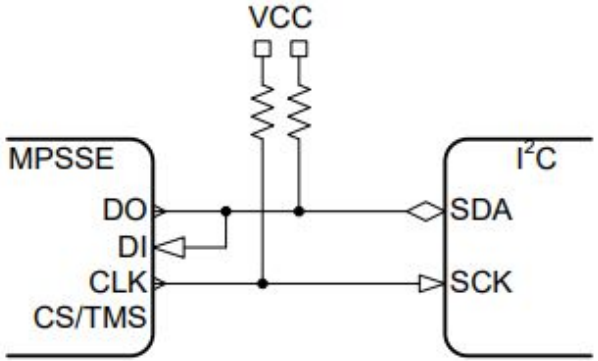
Adafruit FT232H breakout board

Az adafruit készít és értékesít FT232H IC-vel szerelt kártyát. Az Adafruittól megszokott módon komplett beüzemelési leírással, az IC vezérléséhez könyvtárakkal és példákkal. A vezérlőkönyvtár használatával további circuitpython könyvtárak elérhetőek és azon keresztül különböző érzékelők, eszközök vagy akár LCD és OLED panelek vezértelhetőek.



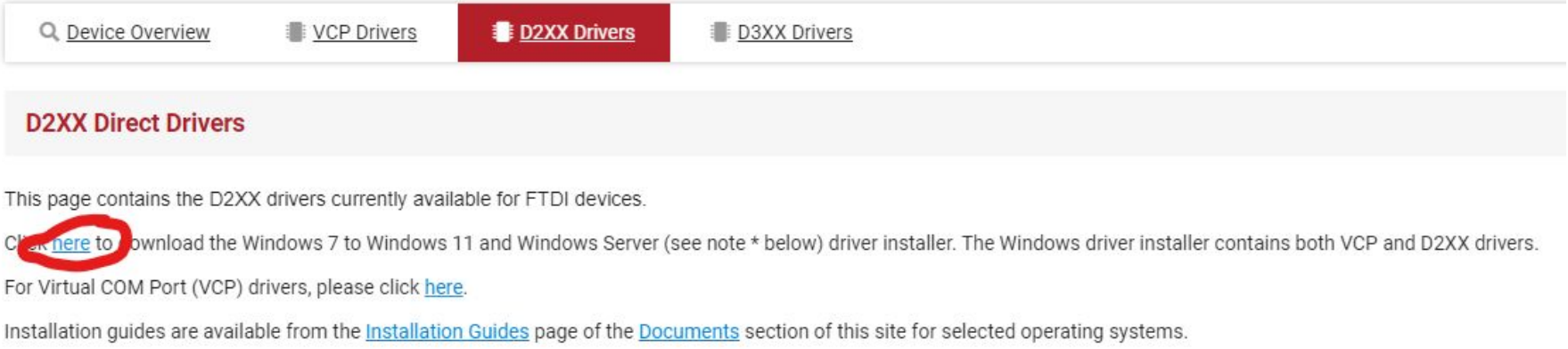
I2C bekötése

DO - D1
DI - D2
CLK- D0



d2xx driver telepítés windows alatt

Az FT232H eszköz csatlakoztatása előtt az alábbi drivert kell telepíteni.
FTDI oldaláról le kell tölteni a d2xx driver-t, jelenleg a neve CDM212364_Setup.zip, telepíteni rendszergazdai jogosultsággal lehetséges a zip fájl kitömörítése után
https://ftdichip.com/drivers/d2xx-drivers/



MPSSE könyvtár

Az FTDI Multi-Protocol Synchronous Serial Engine (MPSSE) rugalmas interfész eszközt biztosít szinkron soros eszközök kapcsolódásához egy USB-porton keresztül. Mivel az MPSSE „Multi-Protocol”, lehetővé teszi a kommunikációt sokféle szinkron eszközzel, a legnépszerűbb az SPI, az I2C és JTAG.
Az adat formázás és az óraszinkronizálás sokféleképpen konfigurálható, hogy szinte minden igényt kielégítsen. A soros kivezetések mellett további GPIO jelek is rendelkezésre állnak.

Az előfordított MPSSE könyvtár letölthető az alábbi linkről
https://ftdichip.com/wp-content/uploads/2020/08/libMPSSE.zip

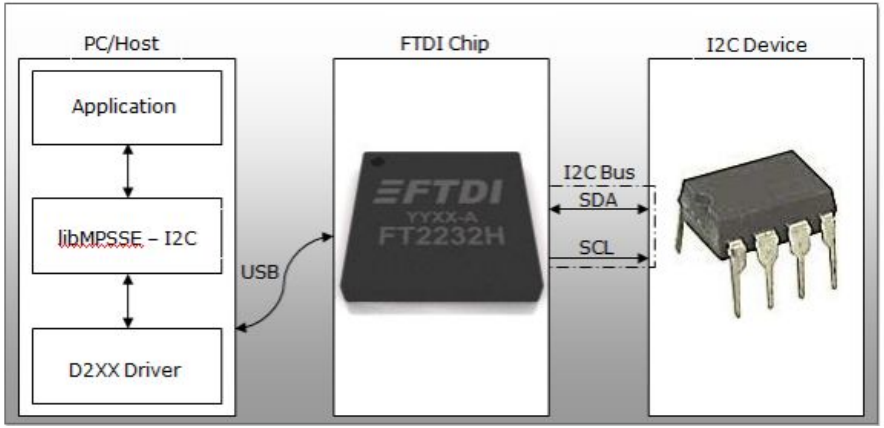
A letöltött libMPSSE.zip fájlon belül a 64 bites DLL a libMPSSE__0.6\lib\windows\visualstudio\x64\ könyvtár alatt található

MPSSE könyvtár függvény leírás

Az MPSSE könyvtár meghívható függvényeit két dokumentum tartalmazza, az I2C protokoll-hoz és az SPI protokollhoz külön dokumentáció tartozik
https://ftdichip.com/document/programming-guides/

LibMPSSE-I2C User Guide	1.4	User Guide for LibMPSSE-I2C.DLL
FTCJTAG Programmer's Guide	1.2	Lists functions available in FTCJTAG.DLL
LibMPSSE-SPI User Guide	1.1	Lists functions available in LibMPSSE-SPI.DLL

A jobb oldali képen a rendszer elemei láthatóak, az alkalmazás, a libMPSSE és a d2xx driver a PC-n fut, az FTDI chip és az eszköz IC általában egy alaplapon helyezkedik el



MPSSE könyvtár I2C függvényei a leírás alapján

FT_STATUS **I2C_GetNumChannels** (uint32 *numChannels)
This function gets the number of I2C channels that are connected to the host system. The number of ports available in each of these chips is different.
Parameters:
out *numChannels The number of channels connected to the host
Returns:
Returns status code of type FT_STATUS

A függvény meghívása python programnyelvből

```
channel_count = ctypes.c_int()
ret = libMPSSE.I2C_GetNumChannels(ctypes.byref(channel_count))
```

FT_STATUS **I2C_OpenChannel** (uint32 index, FT_HANDLE *handle)
This function opens the indexed channel and provides a handle to it. Valid values for the index of channel can be from 0 to the value obtained using I2C_GetNumChannels – 1).
Parameters:

in	Index	Index of the channel
out	Handle	Pointer to the handle of type FT_HANDLE

Returns:
Returns status code of type FT_STATUS

A függvény meghívása python programnyelvből

```
c = Channel('B', 0) # Open Channel B (index 0) for I2C
ret = libMPSSE.I2C_OpenChannel(c.index, ctypes.byref(c.handle))
```

FT_STATUS **I2C_InitChannel** (FT_HANDLE handle, ChannelConfig *config)
This function initializes the channel and the communication parameters associated with it.
Parameters:

In Handle Handle of the channel
In Config Pointer to ChannelConfig structure. Members of ChannelConfig structure contains the values for I2C master clock, latency timer and Options

Returns:
Returns status code of type FT_STATUS

A függvény meghívása python programnyelvből

```
channel_conf = ChannelConfig(400000, 25, 0) # 400KHz, 25ms latency timer
ret = libMPSSE.I2C_InitChannel(c.handle, ctypes.byref(channel_conf))
```

FT_STATUS **I2C_DeviceWrite** (FT_HANDLE handle, uint32 deviceAddress, uint32 sizeToTransfer, uint8 *buffer, uint32 *sizeTransferred, uint32 options)
This function writes the specified number of bytes to an addressed I2C slave.
Parameters:

In	Handle	Handle of the channel
In	deviceAddress	Address of the I2C slave. This is a 7bit value
In	sizeToTransfer	Number of bytes to be written
out	Buffer	Pointer to the buffer data is to be written
out	sizeTransferred	Pointer to containing the number of bytes written
In	transferOptions	This parameter specifies data transfer options.

A függvény meghívása python programnyelvből

```
buf = ctypes.c_buffer(b'', 1) ; transfered = ctypes.c_int()
ret = libMPSSE.I2C_DeviceWrite(c.handle, addr, len(buf), buf, ctypes.byref(transfered), mode)
```

Példaprogram az I2C kommunikáció használatára

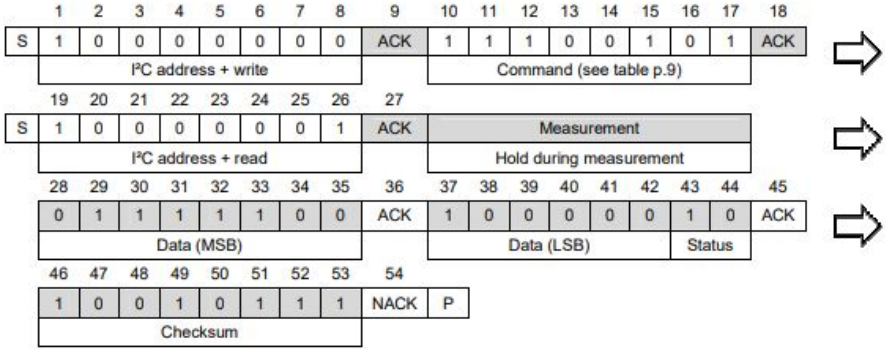
A példaprogramban az MPSSE könyvtár használatával python programnyelvből kérdezzük le a HTU21D hőmérséklet és páratartalom mérő hőmérsékleti értékét.

A visszaadott értéket az IC adatlapján megadott képlettel alakítjuk Celsius fokra

The temperature T is calculated by inserting temperature signal output S_{Temp} into the following formula (result in °C), no matter which resolution is chosen:

$$Temp = -46.85 + 175.72 \times \frac{S_{Temp}}{2^{16}}$$

Command	Code	Comment
Trigger Temperature Measurement	0xE3	Hold master
Trigger Humidity Measurement	0xE5	Hold master
Trigger Temperature Measurement	0xF3	No Hold master
Trigger Humidity Measurement	0xF5	No Hold master
Write user register	0xE6	
Read user register	0xE7	
Soft Reset	0xFE	



```
import time,ctypes
from enum import Enum
class Channel():
    def __init__(self, name, index):
        self.name = name
        self.index = index
        self.handle = ctypes.c_void_p()

class ChannelConfig(ctypes.Structure):
    _fields_ = [('ClockRate', ctypes.c_int),('LatencyTimer', ctypes.c_ubyte),('Options', ctypes.c_int)]


class ChannelInfo(ctypes.Structure):
    _fields_ = [('Flags', ctypes.c_ulong),('Type', ctypes.c_ulong),('ID', ctypes.c_ulong),
            ('LocId', ctypes.c_ulong),('SerialNumber', ctypes.c_char*16),('Description', ctypes.c_char*64),
            ('ftHandle', ctypes.c_void_p)]
# Some constants for FTDI config...
START_BIT                   = 0x01
STOP_BIT                    = 0x02
BREAK_ON_NACK               = 0x04
NACK_LAST_BYTE              = 0x08
FAST_TRANSFER_BYTES         = 0x10
FAST_TRANSFER_BITS          = 0x20
FAST_TRANSFER               = 0x30
NO_ADDRESS                  = 0x40
I2C_DISABLE_3PHASE_CLOCKING = 0x01
I2C_ENABLE_DRIVE_ONLY_ZERO  = 0x02
```

```
class FT_STATUS(Enum):
    FT_OK                           = 0
    FT_INVALID_HANDLE               = 1
    FT_DEVICE_NOT_FOUND             = 2
    FT_DEVICE_NOT_OPENED            = 3
    FT_IO_ERROR                     = 4
    FT_INSUFFICIENT_RESOURCES       = 5
    FT_INVALID_PARAMETER            = 6
    FT_INVALID_BAUD_RATE            = 7
    FT_DEVICE_NOT_OPENED_FOR_ERASE  = 8
    FT_DEVICE_NOT_OPENED_FOR_WRITE  = 9
    FT_FAILED_TO_WRITE_DEVICE       = 10
    FT_EEPROM_READ_FAILED           = 11
    FT_EEPROM_WRITE_FAILED          = 12
    FT_EEPROM_ERASE_FAILED          = 13
    FT_EEPROM_NOT_PRESENT           = 14
    FT_EEPROM_NOT_PROGRAMMED        = 15
    FT_INVALID_ARGS                 = 16
    FT_NOT_SUPPORTED                = 17
    FT_OTHER_ERROR                  = 18
    FT_DEVICE_LIST_NOT_READ         = 19


status = lambda code: FT_STATUS(code).name

sleep_us = lambda x: time.sleep( x/1000000.0)
sleep_ms = lambda x: time.sleep( x/1000.0)
sleep = time.sleep
```

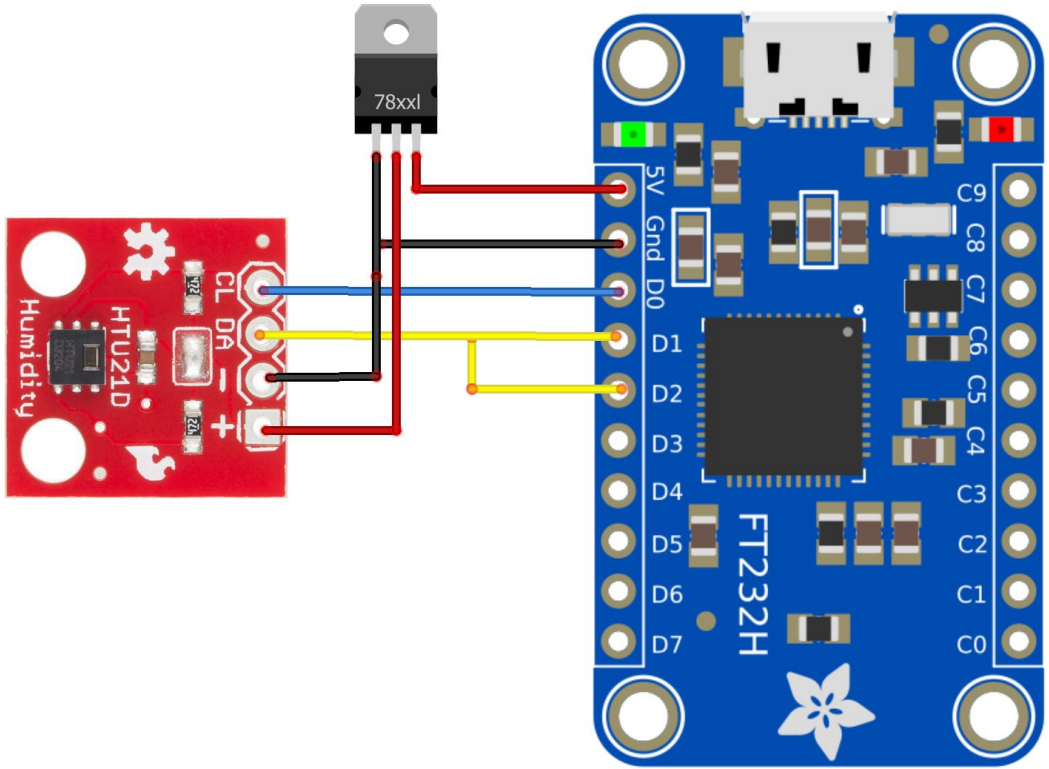
```
libMPSSE = ctypes.cdll.LoadLibrary("./libMPSSE.dll")
print('Listing channels...')
libMPSSE.Init_libMPSSE()
channel_count = ctypes.c_int()
ret = libMPSSE.I2C_GetNumChannels(ctypes.byref(channel_count))
print(f'Found {channel_count.value} channels (status {status(ret)})')
c = Channel('B', 0) # Open Channel B (index 0) for I2C

channel_info = ChannelInfo()
print(f'Getting info for channel with index {c.index}...')
ret = libMPSSE.I2C_GetChannelInfo(c.index, ctypes.byref(channel_info))
print(f'Channel description: {channel_info.Description.decode()} (status {status(ret)})')
ret = libMPSSE.I2C_OpenChannel(c.index, ctypes.byref(c.handle))
print(f'Channel {c.name} opened with handle: 0x{c.handle.value:x} (status {status(ret)})')
channel_conf = ChannelConfig(400000, 25, 0) # 400KHz, 25ms latency timer, no options
ret = libMPSSE.I2C_InitChannel(c.handle, ctypes.byref(channel_conf))
print(f'InitChannel() {c.name} (status {status(ret)})')


HTU21D_I2CADDR     = 0x40
HTU21D_READTEMP    = 0xE3
HTU21D_READHUM     = 0xE5
HTU21D_WRITEREG    = 0xE6
HTU21D_READREG     = 0xE7
HTU21D_RESET       = 0xFE

write_address = HTU21D_I2CADDR
mode = START_BIT | STOP_BIT | NACK_LAST_BYTE
```

```
buf = ctypes.c_buffer(b'', 1)
buf[0] = HTU21D_READTEMP
bytes_transfered = ctypes.c_int()
ret = libMPSSE.I2C_DeviceWrite(c.handle, write_address, len(buf), buf, ctypes.byref(bytes_transfered), mode)
print(f'Wrote {bytes_transfered.value} byte(s) (status {status(ret)})')
sleep_ms( 50 )
buf = ctypes.c_buffer(b'', 3)
bytes_transfered = ctypes.c_int()
ret = libMPSSE.I2C_DeviceRead(c.handle, write_address, len(buf), buf, ctypes.byref(bytes_transfered), mode)
print(f'Read {bytes_transfered.value} byte(s) (status {status(ret)})')
t = ( bytes(buf)[0] << 8) + ( bytes(buf)[1] & 0b11111100 )
temp = t * 175.72 / 65536 - 46.85
print( "******************\nreaded temp: {0:.3f}\n******************".format(  temp ) )
ret = libMPSSE.I2C_CloseChannel(c.handle)
print(f'CloseChannel() {c.name} (status {status(ret)})')
```

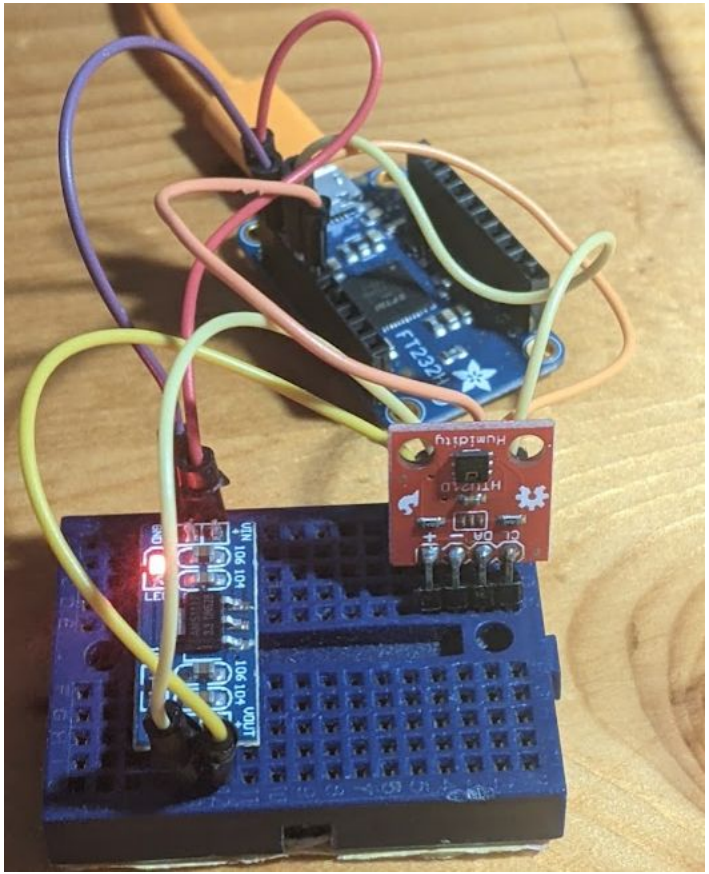
A program futtatása a következő kimenetet adja

```
Listing channels...
Found 1 channels (status FT_OK)
Getting info for channel with index 0...
Channel description: FT232H (status FT_OK)
Channel B opened with handle: 0x2310c48a940 (status FT_OK)
InitChannel() B (status FT_OK)
Wrote 1 byte(s) (status FT_OK)
Read 3 byte(s) (status FT_OK)
******************
readed temp: 25.148
******************
CloseChannel() B (status FT_OK)
```

Adafruit FT232H és HTU21D bekötése



fritzing



Adafruit FT232H Breakout board