ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA ELEKTROTECHNICKÁ

KAE

BAKALÁŘSKÁ PRÁCE

Příklady použití MicroPython

Michal Rendl **2019**

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta elektrotechnická
Akademický rok: **2018/2019**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Michal RENDL**
Osobní číslo:	**E15B0167P**
Studijní program:	**B2644 Aplikovaná elektrotechnika**
Studijní obor:	**Aplikovaná elektrotechnika**
Název tématu:	**Příklady použití MicroPython**
Zadávající katedra:	**Katedra elektromechaniky a výkonové elektroniky**

Zásady pro vypracování:

Připravte sadu vzorových příkladů použití jazyka MicroPython a ověřte jejich činnost.

.

1. Uvažujte HW prostředky dostupné na KAE - např. kity s procesory ARM řady STM32.
2. Uvažte při návrhu příkladů možnost nasazení do výuky mikroprocesorové techniky.

Rozsah grafických prací:	**podle doporučení vedoucího**
Rozsah kvalifikační práce:	**30 - 40 stran**
Forma zpracování bakalářské práce:	**tištěná/elektronická**

Seznam odborné literatury:

Student si vhodnou literaturu vyhledá v dostupných pramenech podle doporučení vedoucího práce.

Vedoucí bakalářské práce:	**Ing. Petr Weissar, Ph.D.** Katedra aplikované elektroniky a telekomunikací
Datum zadání bakalářské práce:	**5. října 2018**
Termín odevzdání bakalářské práce:	**13. června 2019**

Prof. Ing. Zdeněk Peroutka, Ph.D.
děkan

L.S.

Prof. Ing. Václav Kůs, CSc.
vedoucí katedry

V Plzni dne 5. října 2018

Abstrakt

Předkládaná bakalářská práce je zaměřena na vytvoření vzorových příkladů pro kity s procesory ARM řady STM32. Využit bude především programovací jazyk MicroPython. Práce si klade za cíl seznámit čtenáře s celým procesem programování v MicroPythonu. To zahrnuje mimo jiné vytvoření firmwaru, flashování desky, zpracování dat, základní práci s registry, práci se soubory, sériovou komunikaci, bezdrátovou komunikaci, ovládání displeje, nebo optimalizaci kódu.

Klíčová slova

MicroPython, Mikrokontrolér, STM32, Sériová komunikace, SPI, I2C, UART, SSD1306, REPL, webREPL, UM2121, DS18B20, ILI9341

Abstract

The presented bachelor thesis is focused on creating exemplary examples for kits with ARM STM32 series processors. In particular, the MicroPython programming language will be used. The aim of the thesis is to introduce the reader to the whole process of programming in MicroPython. This includes, but is not limited to, firmware creation, board flashing, data processing, basic registry work, file handling, serial communications, wireless communications, display control, or code optimization.

Key words

MicroPython, Mikrocontroller, STM32, Serial communication, SPI, I2C, UART, SSD1306, REPL, webREPL, UM2121, DS18B20, ILI9341

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně, s použitím odborné literatury a pramenů uvedených v seznamu, který je součástí této bakalářské práce.

Dále prohlašuji, že veškerý software, použitý při řešení této bakalářské práce, je legální.

.....

podpis

V Plzni dne 11.6.2019 Michal Rendl

Poděkování

Tímto bych rád poděkoval vedoucímu bakalářské práce doc. Ing. Petru Weissarovi, Ph.D. za cenné profesionální rady, připomínky a zapůjčení hardwaru. Taktéž bych chtěl poděkovat Ondřeji Urbanovi a Josefu Navrátilovi za cenné připomínky.

Obsah

Seznam symbolů a zkratek

CPU ..... Centrální procesorová jednotka (Central Processing Unit)

DFU ..... Aktualizace firmwaru zařízení (Device Firmware Upgrade)

GPIO ..... Obecný vstup/výstup (General Purpose Input/Output)

IDE ..... Integrované vývojové prostředí (Integrated Development Environment)

IOT ..... Internet věcí (Internet of Things)

OS ..... Operační systém (Operating Systém)

PWM ..... Pulzně šířková modulace (Pulse Width Modulation)

SPI ..... Sériové periferní rozhraní (Serial Peripheral Interface)

SWD ..... Ladění sériové linky (Serial Wire Debug)

USB ..... Univerzální sériová sběrnice (Universal Serial Bus)

UART ..... Univerzální asynchronní přijímač/vysílač

..... (Universal Asynchronous Receiver/Transmitter)

I²C ..... Druh sériové sběrnice - Inter Integrated Circuit

REPL ..... Interaktivní prostředí - Read Eval Print Loop

STM ..... Výrobce integrovaných zařízení - STMicroelectronics

Úvod

Pokud se kdokoliv rozhodne, že se chce věnovat programování mikrokontrolérů, nenaskytuje se mu moc možností, který programovací jazyk zvolit. Mikrokontroléry jsou obecně programovány v jazyce C, který přistupuje a manipuluje s periferními obvody přímo pomocí registrů. Tento jazyk je nízkoúrovňový, a proto není psaní kódu v tomto jazyce snadné. Chce to velkou trpělivost a mnoho volného času, proto mnoho lidí brzo práce s tímto jazykem znechutí a zanechají této činnosti. Nejen pro tyto lidi se nabízí možnost začít programovat mikrokontroléry v mnohem jednodušším jazyce na pochopení, kterým je MicroPython. Ten poskytuje, ve srovnání s jazykem C, snadnou práci se slovníky, seznamy, strukturami a některé další věci typické pro vysokoúrovňové programování. Tyto věci lze napsat i v jazyce C. Problém je, že jsou založené na ukazatelích (pointrech). Při nesprávné práci s ukazateli je aplikace náchylná k nechtěnému přepsání paměti a následnému pádu aplikace. Jedna z mála nevýhod oproti programu v jazyku C je rychlost programu napsaného v MicroPythonu. Tuto nevýhodu do určité míry kompenzuje možnost použití inline assembleru.

Z prvního odstavce jsou patrné výhody jazyka MicroPython, přesto se v současné době nevěnuje tomuto jazyku dostatečná pozornost, a proto není na tento jazyk mnoho kvalitních návodů v češtině. Cílem této bakalářské práce je sepsat podrobný postup jak pracovat s tímto programovacím jazykem a zhotovit základní vzorové příklady. Jelikož není tento jazyk primárně podporován překladačem, práce se bude nejdříve zabývat vytvořením a nahráním firmware na desku. Potom bude probrána základní práce s tímto jazykem. Největší důraz bude kladen na vysvětlení sériové komunikace, protože ji pro přenos dat využívá mnoho senzorů, displejů a další periferní obvody.

1 MicroPython

MicroPython je softwarová implementace programovacího jazyka Python3 napsaného v jazyce C, který je optimalizován pro běh na mikrokontrolérech. Uživatel má k dispozici interaktivní programovací prostředí nazvaný REPL, aby okamžitě spustil podporované příkazy. Je zde zahrnut i výběr základních knihoven Pythonu. MicroPython obsahuje knihovny, které umožňují programátorovi přístup k nízko úrovňovému hardwaru. MicroPython je původně vytvořený australským programátorem a fyzikem Damienem Georgeem, po úspěšné podpoře kampaně na Kickstarteru v roce 2013. Od té doby je provozován na hardwaru, kterým je například Arduino, ESP8266 nebo ESP32. V roce 2016 byla vytvořena verze MicroPythonu pro BBC Micro Bit. MicroPython je překladač jazyka Python s částečnou funkcí nativní kompilace. MicroPython poskytuje celou syntaxi Pythonu 3.4 a navíc klíčová slova z Pythonu 3.5. Kód firmwaru je křížově kompilován (cross-compiled), vytvořen pro cílovou architekturu mikrokontroléru a flashován pomocí vhodného programátoru. MicroPython sice provádí všechny tyto kroky, avšak je tu omezení ve vlastnosti mikrokontrolérů pracovat v reálném čase. MicroPython proto není vhodný pro aplikace, kde je kladen vysoký důraz na práci v reálném čase.

1.1 Rozdíly mezi Pythonem a MicroPythonem

Některé knihovny nemohou být u některých mikrokontrolérů implementovány. To je způsobené například vysokou spotřebou paměti (např. Sqlite3), nebo nedostatkem určité požadované hardwarové funkce (např. paralelní zpracování dat). Pokud některý modul chybí, zapříčiní to neuplatnitelnost tohoto modulu pro použití v integrovaném řadiči, proto MicroPython podporuje pouze část standardní knihovny pythonu. Úplný seznam standardních knihoven pythonu lze nalézt v standartní knihovně Python 3.4. Mezi CPython3, který je považován za referenční implementaci jazyka Python3 a MicroPythonem existuje několik rozdílů. [1]

1.1.1 Omezené systémy

MicroPython je určen pro omezená prostředí, zejména pro mikrokontroléry, které mají řádově menší výkon a paměť, než desktopové systémy, na kterých běží CPython. To znamená, že MicroPython musí být navržen s ohledem na toto omezené prostředí, důsledkem toho je vynechání funkcí, které jednoduše nebudou odpovídat cílovým systémům. Některé z funkcí

jsou potenciálně předmětem budoucího vývoje, ale mnoho z nich by ovlivnilo velikost a výkon jakékoliv implementace. [1]

1.1.2 Garbage collector

Na rozdíl od CPythonu, který používá referenční počítání, MicroPython používá jako primární způsob správy paměti tzv. garbage collection. Garbage collection funguje tak, že speciální algoritmus nazvaný garbage collector vyhledává a uvolňuje úseky paměti, které již program nebo proces nepoužívá. [1]

1.1.3 Deskriptory

MicroPython neimplementuje úplný objektový datový model CPython, ale pouze jeho podmnožinu. Pokročilé využití metody dědičnosti, metoda __new__ nemusí fungovat. Pořadí rozlišování metod je odlišné. Metatřídy nejsou zatím podporovány. Návrh MicroPythonu není založen na deskriptivních objektech. Deskriptory jsou považovány za "naddynamické" funkce a jsou v rozporu s cílem být rychlé a efektivní. Navzdory tomu je zjednodušená podpora deskriptorů implementovaná jako funkce opt-in. [1]

1.1.4 Ostatní rozdíly

MicroPython podporuje pouze minimální podmnožinu funkcí introspekce a reflexe jako jsou názvy objektů, docstring a tak podobně. Například print() nekontroluje rekurzivní datové struktury stejným způsobem jako CPython. Existuje však kontrola využití zásobníku, takže tisková rekurzivní struktura dat nepovede ke zhroucení kvůli přetečení zásobníku. Dále nejsou podporovány Buffery I/O streamů. [1]

1.2 Knihovny

CPython podporuje celou řadu knihoven (funkcí a tříd), ale jejich přenášení přímo do mikrokontroléru není snadné, protože nejsou optimalizovány pro běh s menší pamětí RAM. Aby užitečné a již existující knihovny Pythonu byly k dispozici v MicroPythonu, jsou upraveny tak, aby byly lépe optimalizovány pro běh na omezeném hardwaru.

1.2.1 Import knihoven

Aby se mohla použít knihovna, je potřeba ji nejdříve importovat. Pro to je použit příkaz import a název knihovny, např. `import math`. Pokud je potřeba pouze určitá funkce

v knihovně, je vhodnější načíst pouze onu funkci příkazem `from` „název knihovny“ `import` „název funkce“.

```
>>> from math import cos,pi
>>> cos(5*pi)
-1.0
```

Obr. 1 Příklad importu funkce z knihovny

Ke zjištění dostupných vestavěných knihoven, které lze použít, stačí do prostředí REPL zadat příkaz `help('modules')`. Tento výpis knihoven se liší v závislosti na konkrétní sestavě MicroPythonu a na konkrétní desce.

```
>>> help("modules")
__main__          json              socket            umachine
_onewire          lcd160cr          stm               uos
array             lcd160cr_test     struct            urandom
binascii          machine           sys               ure
builtins          math              time              uselect
cmath             micropython       ubinascii         usocket
collections       network           ucollections      ustruct
errno             onewire           uctypes           utime
framebuf          os                uerrno            utimeq
gc                pyb               uhashlib          uzlib
hashlib           random            uheapq            zlib
heapq             re                uio
io                select            ujson
Plus any modules on the filesystem
>>>
```

Obr. 2 Výpis všech knihoven na desce NUCLEO-F411RE

array - pole číselných dat

cmath - matematické funkce pro komplexní čísla

gc - ovládání garbage collector

math - matematické funkce

sys - specifické funkce systému

ubinascii - binární / ASCII konverze

ucollections - kolekce a kontejner

uheapq - algoritmus fronty haldy

uos - základní služby "operačního systému"

utime - funkce související s časem

machine - funkce týkající se hardwaru

micropython - přístup a ovládání vnitřních prvků MicroPythonu

network - nastavení sítě

pyb - funkce týkající se hardwaru, specifické pro desku pyboard

Nejpoužívanější knihovny pyb a machine umožňují přístup k interním periferiím čipu mikrokontroléru, kterými jsou například GPIO, ADC, PWM, funkce související s časem, sériovou komunikací, resetem, přerušením, napájením a ovládáním dalších periferií.

1.2.2 MicroPython-lib

MicroPython neposkytuje rozsáhlou standardní knihovnu modulů. Není možné a ani nemá smysl poskytovat kompletní knihovnu CPython. Mnoho modulů není použitelných ani užitečných v kontextu vestavěných systémů a není dostatek paměti pro nasazení celé knihovny na malé zařízení. Proto se MicroPython řídí minimalistickým přístupem. To znamená, že jsou do překladače zahrnuty pouze základní datové typy a knihovny specifické pro konkrétní hardware. Veškeré standardní knihovny jsou podporovány v modulu micropython-lib. [2]

1.3 Způsoby programování

MicroPython podporuje hned několik způsobů programování (Paradigmat).

Imperativní programování

Paradigma popisuje výpočet pomocí posloupnosti příkazů a určuje přesný postup (algoritmus), jak danou úlohu řešit.

Funkcionální programování

Paradigma chápe výpočet jako vyhodnocení matematických funkcí. Funkcionální programovací jazyky, především čistě funkcionální, se používají spíše v akademickém než komerčním prostředí.

Objektově orientované programování

Výkonný kód je přidružen k datům (metody jsou zapouzdřeny v objektech), což umožňuje snadnější přenos kódu mezi různými projekty (abstrakce a zapouzdření). Propojení umožnilo zavést dědičnost, ale kvůli zjednodušení si vyžádalo zavedení polymorfismu.

2 Flashování

Pro programování v MicroPythonu a používání prostředí REPL je potřeba nejdříve nahrát firmware na desku pomocí bootloaderu. Tento proces se nazývá flashování. Firmware pro desku WROOM-32 a NUCLEO-F411RE je možné stáhnout z oficiálních stránek MicroPythonu, tím se usnadní mnoho práce. U desky NUCLEO-F413ZH tato možnost není a proto je nutné vytvořit konfigurační soubory a firmware vygenerovat v linuxovém prostředí.

2.1 Vygenerování firmware v OS Linux

Aby se mohl vygenerovat firmware za pomocí konzole, je potřeba nejdříve stáhnout projekt MicroPython z webové služby GitHub.

```
$ sudo apt-get install git
$ git clone https://github.com/micropython/micropython
```

Dále je nutné nainstalovat toolchain[1] z repositáře

```
$ sudo add-apt-repository ppa:team-gcc-arm-embedded/ppa
$ sudo apt-get update
$ sudo apt-get install gcc-arm-embedded
```

Následně je potřeba inicializovat projekt micropython.

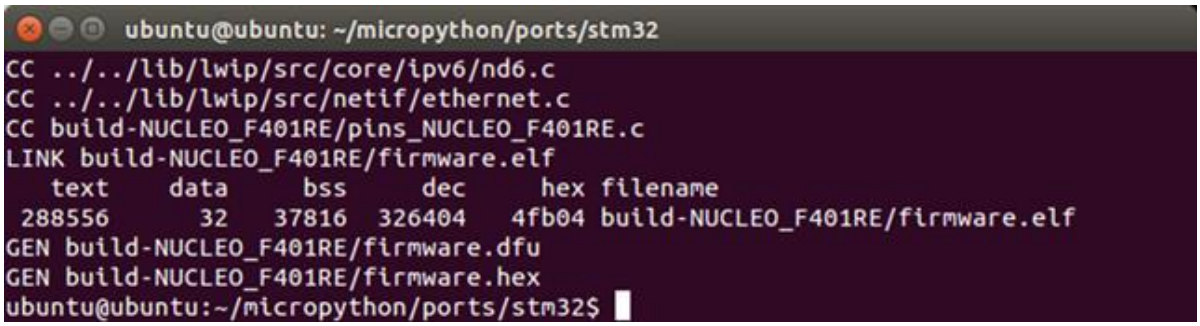
```
$ cd micropython
$ git submodule update --init
```

Nakonec se dostane do složky `stm32`, kde se vygenerují potřebné soubory příkazem `make`

```
$ cd ports/stm32/boards
$ make BOARD=NUCLEO_F411RE
```

Zde se samozřejmě `NUCLEO_F411RE` nahradí požadovanou deskou, je potřeba dávat pozor na mezery u znaménka =, protože tu žádná mezera nesmí být! Pokud vše proběhlo správně, vygeneruje se firmware ve třech podobách a to s koncovkou elf, dfu a hex.

[1] Toolchain není součástí repozitáře Ubuntu od vereze 18.10



Obr. 3 Výpis vygenerovaných souborů

2.2 Přizpůsobení MicroPythonu k rozdílné desce

Pro přizpůsobení firmware MicroPython na jinou desku s jádrem STM32, je nutné nejdříve vytvořit složku uvnitř projektu MicroPython v podadresáři stm32/boards s názvem desky, například NUCLEO_F413ZH. Potom se do této složky umístí následující 4 upravené soubory:

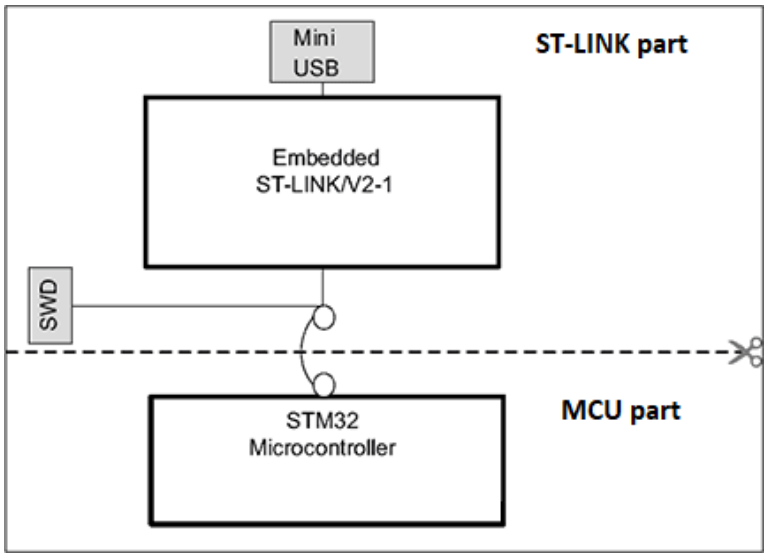
- mpconfigboard.h - popisuje desku směrnicí, tu v podstatě tvoří #define direktivy
- mpconfigboard.mk - odkaz na tabulky alternativních funkcí (AF) a konfigurační skript GNU LD
- pins.csv - seznam použitých vývodů (záleží na rozložení hardwaru desky)
- stm32f4xx_hal_conf.h - deklarace definice a importu pro STM32Cube HAL (ten může být generovaný nástrojem STM32CubeMX)

Mezi hlavní rozdíly mezi jednotlivými deskami patří například různý počet uživatelských LED, latence externího oscilátoru (HSE), frekvence CPU, nebo rozdílné umístění pinů pro UART, SPI, I2C.

2.3 Nahrání firmware na desku

2.3.1 ST-Link

ST-Link je debugger a programátor pro obvody mikrokontrolérů STM8 a STM32. ST-Link pro komunikaci s jakýmikoliv mikrokontrolérem STM8 nebo STM32 používají rozhraní SWD a JTAG / sériové ladění (SWD). Pokud deska nemá zabudovaný ST-LINK programátor, je možné ho dokoupit, nebo lze použít levnější dongle od výrobců třetích stran. Pro nahrání firmware na desku přes ST-Link je potřeba vygenerovaný soubor hex, nebo bin. Tento soubor se nahraje do desky programem STM32 ST-LINK Ubility.



Obr. 4 Zjednodušené schéma ST-Link programátoru připojeného k desce (převzato a upraveno z [3])

2.3.2 USB - FTDI sériový modul

Tento modul se připojí k libovolnému rozhraní UART na desce, v tomhle případě se může použít i bezdrátový Bluetooth modul. Po připojení modulu se přes program Flash Loader Demonstrator nahraje vygenerovaný firmware, který je opět ve formátu hex, nebo bin. V operačním prostředí Linux lze k flashování desky použít nástroj příkazového řádku, známý jako Esptool.

2.3.3 USB OTG FS

Poslední možností, jak flashnout desku, je použít USB konektor nazvaný DFU, pokud ho deska obsahuje. Pro tenhle způsob flashování desky je zapotřebí firmware ve formátu dfu, ten se nahraje do desky přes programem Flash Loader Demonstrator.

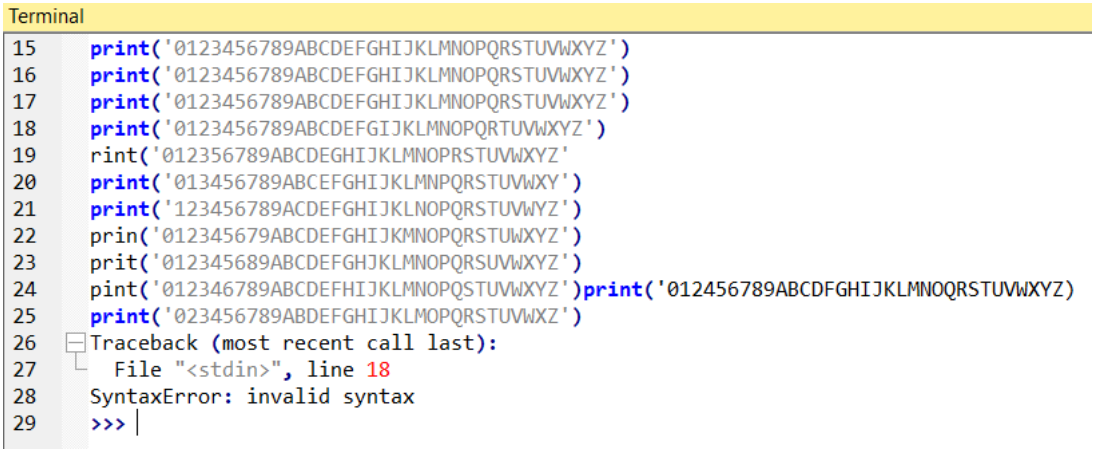
3 REPL

Způsob, kterým lze ovládat mikrokontrolér s microPythonem, je používání REPL. REPL je jednoduché interaktivní počítačové prostředí, které přijímá uživatelské vstupy (jednotlivé příkazy), vyhodnocuje je a vrátí výsledek uživateli. Program napsaný v prostředí REPL se provádí po částech. Běžné příklady zahrnují řádkové shelly, které vytvářejí příkazový řádek (CLI) a jsou zvláště charakteristické pro skriptovací jazyky. Použití REPL je zdaleka nejjednodušší způsob, jak vyzkoušet kód a spouštět příkazy.

3.1 Ovládání přes sériový port

První způsob, kterým lze ovládat prostředí REPL je prostřednictvím kabelového připojení přes asynchronní sériové rozhraní UART, které je detailněji popsané v kapitole 5.1.1. Modulační rychlost (baud rate) je v základu nastavená na 115200 Bd. Při odeslání rozsáhlejšího kódu do desky přes rozhraní UART může nastat problém s vyrovnávací pamětí, proto je možné tuhle paměť upravit metodou `init` s parametrem `read_buf_len`. [4]

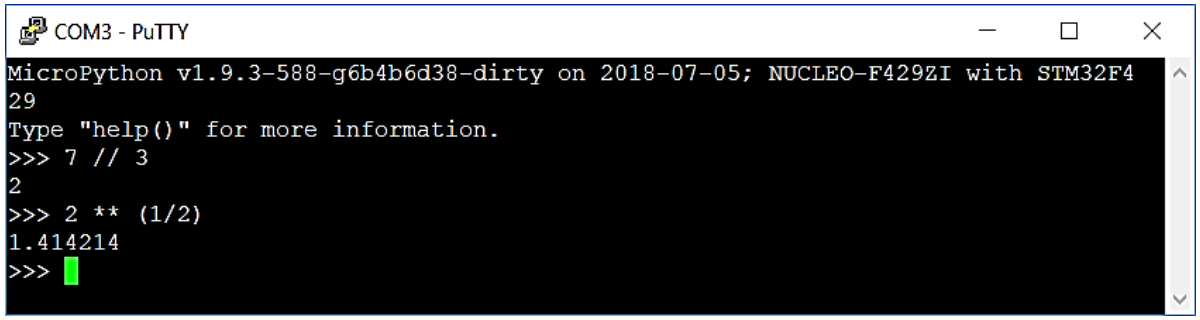
```
import pyb
pyb.repl_uart().init(baudrate = 115200, read_buf_len = 300)
```



Obr. 5 Důsledek nedostatečné vyrovnávací paměti

3.1.1 PuTTY

V prostředí Windows se dá ovládat REPL přes sériové rozhraní využitím emulátoru terminálu. Existuje celá řada těchto emulátorů a lze samozřejmě použít kterýkoliv z nich. Zde je pro demonstraci ukázaný program nazvaný PuTTY, který je volně dostupný ke stažení z oficiálního webu. Po zapnutí programu PuTTY a resetování desky se na prvním řádku terminálu vypíše verze MicroPythonu, desky a jádra. Na druhém řádku bude vypsán příkaz pro nápovědu. Nakonec se na třetím řádku vypíše série znaků `>>>`, ty jsou indikací, že se uživatel nachází v již zmíněném prostředí REPL.



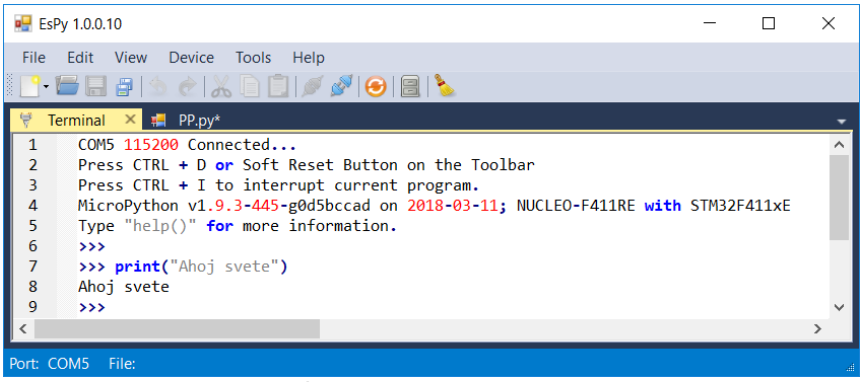
Obr. 6 Program PuTTY

3.1.2 IDE

Jedná se o softwarovou aplikaci, která poskytuje programátorovi nástroje pro vytváření, testování a ladění kódu. IDE většinou obsahuje editor zdrojových kódu, který umožňuje snadné formátování textu, vyhledávání klíčových frází nebo přejmenování názvu tříd, metod a proměnných napříč celým projektem. Některé IDE umožňují i verzování projektů.

3.1.2.1 EsPy

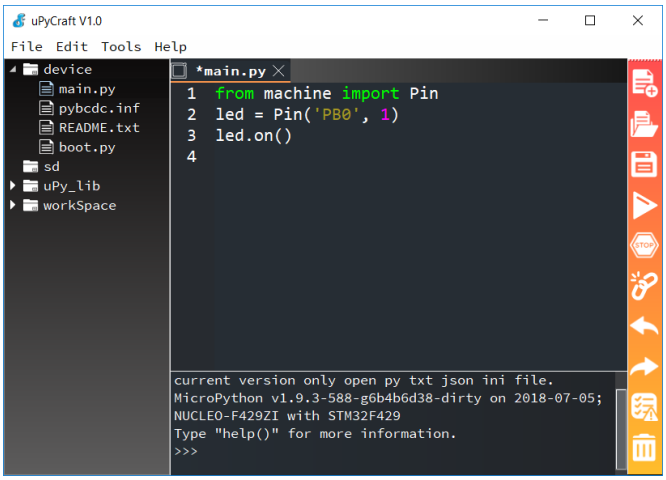
EsPy je IDE, vytvořené v programovacím jazyce C#, které využívá sadu knihoven pro vytváření grafického rozhraní Windows Forms. EsPy obsahuje řadu funkcí pro základní používání mikrokontroléru s MicroPythonem, kterými jsou například úprava kódu, interaktivní terminál, správce souborů, nebo program pro nahrání firmware do desky nazvaný Esptool. [5]



Obr. 7 IDE EsPy 1.0.0.10

3.1.2.2 UPyCraft

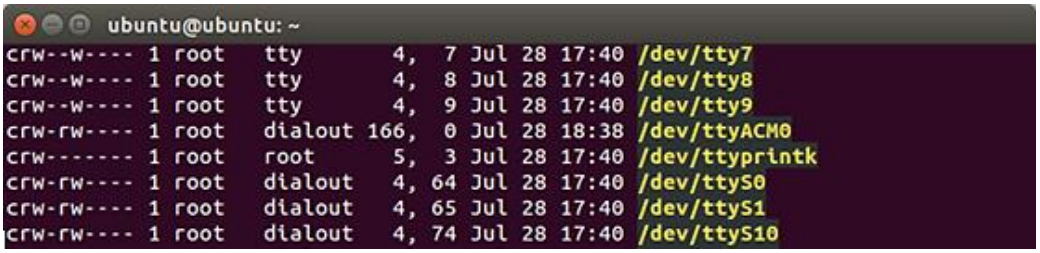
UPyCraft je stejně jako EsPy IDE, volně dostupný ke stažení na webové službě GitHub a stejně tak obsahuje veškeré základní funkce pro práci s mikrokontrolérem. UPyCraft je vytvořený v programovacím jazyce Python. [6]



Obr. 8 IDE UPyCraft V1.0

3.1.3 OS LINUX

Pro připojení k prostředí REPL v OS Linux je potřeba nejdříve zjistit pod jakým názvem se jeví připojená deska v počítači. Nejjednodušší způsob, jak to zjistit je vypsat seznam všech připojených jednotek. To se v terminálu provede příkazem `$ ls -l /dev/tty*`, potom se připojí deska s MicroPythonem a znovu se vypíše seznam připojených jednotek. V druhém výpisu se vypíše o 1 řádek více, ten je potřeba najít. V tomhle případě je to řádek `/dev/ttyACM0`



Obr. 9 Terminál v prostředí LINUX

Samotné připojení k REPL se provede příkazem screen s parametrem modulační rychlosti 115200 Bd.

```
$ sudo apt-get install screen
$ sudo screen /dev/ttyACM0 115200
```

Pokud vše funguje správně, zobrazí se výzva Pythonu `>>>`, kde lze psát Python příkazy.

3.2 Ovládání přes WebREPL

Jedna z unikátních vlastností MicroPythonu na deskách, které podporují vytváření sítí, je WebREPL. Ten umožňuje ovládat mikrokontrolér prostřednictvím webového prohlížeče bez nutnosti sériového připojení k desce. Připojení k WebREPL lze provést na již existující WiFi síti, nebo lze použít desku jako samostatný přístupový bod (access point), v tomhle případě deska vytvoří vlastní síť, ke které se lze připojit.

3.3 Raw-REPL

Raw-REPL se používá jako součást testovací sady pro běh testů na desce, ale může být i užitečný nástroj pro vývoj aplikací. Raw-REPL umožňuje zadávání a spuštění python kódu na desce ze skriptu. V prostředí raw-REPL lze také přesně určit, jaký je skutečný výstup příkazu (např. print). Pokud uživatel v prostředí REPL zadá klávesovou zkratkou CTRL+A, série znaků `>>>` se změní na `>`, to znamená, že se uživatel nachází v prostředí raw-REPL.

4 Souborový systém

Desky Nucleo, používané v této bakalářské práci, jsou nedostačující, co se týče přístupu k vnitřnímu souborovému systému. To trochu omezuje jejich užitečnost pro práci se soubory, protože neexistuje žádný rychlý způsob, jak nahrát zdrojové soubory pythonu, jako je tomu u desek PyBoard. Způsob, kterým se zde přistupuje k souborům, je pomocí rozhraní REPL. [7]

4.1 Práce se soubory

MicroPython podporuje standardní způsob přístupu k souborům pomocí funkce open(), kde je první parametr název souboru a druhý parametr je oprávnění. Bez zadání druhého parametru je soubor přístupný pouze pro čtení, proto je potřeba zadat 'w', tím se umožní zápis do souboru. Obsah souboru lze přečíst metodou write() a zavření souboru se provede příkazem close().

```
>>> f = open('data.txt', 'w')
>>> f.write('Write data') #po zadání se vypíše počet zapsaných byte
10
>>> f.read()
'Write data'
>>> f.close()
```

Pro kontrolu nad souborovým systémem se používá knihovna s názvem `os`, ta se importuje příkazem `>>> import os`. V této knihovně lze provádět několik základních činností:
Vytvoření složky metodou mkdir:

```
>>> os.mkdir('dir')
```

Smazání souboru metodou remove:

```
>>> os.remove('data.txt')
```

Zobrazení souborů metodou listdir:

```
>>> os.listdir()
['main.py', 'pybcdc.inf', 'README.txt', 'boot.py']
```

4.2 Skripty po spuštění desky

Po připojení mikrokontroléru k PC se deska začne napájet a provede se zaváděcí (bootovací) proces. Při tomto procesu se mikrokontrolér nastaví tak, aby obsahoval souborový systém. Tento souborový systém je ve formátu FAT a je přístupný v jednotce /flash, ta se nachází ihned za firmwarem MicroPython.

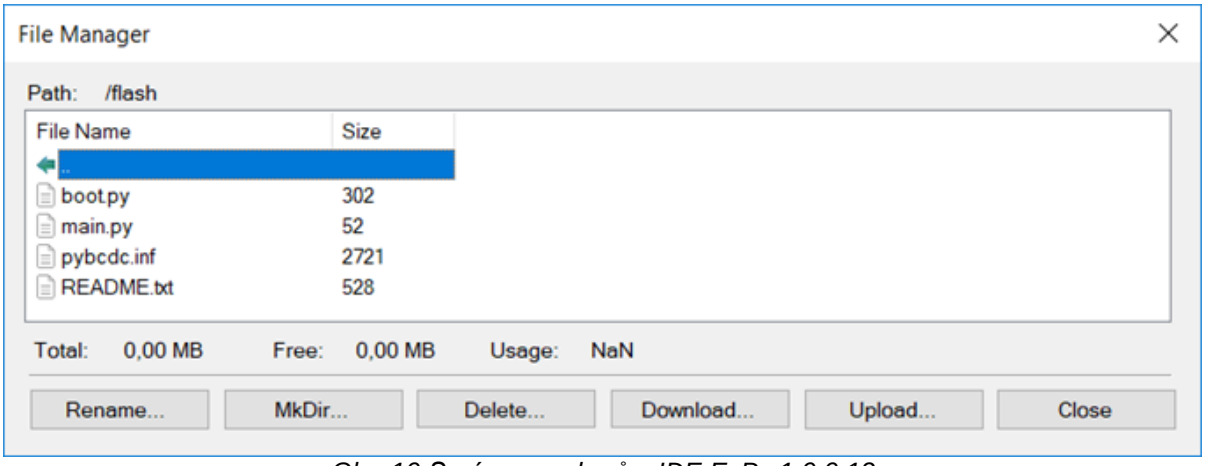
V jednotce /flash se nachází následující 4 soubory: [7] [8]

- **boot.py** - tento skript je spuštěn při zavádění desky. Nastavuje různé konfigurace pro desku
- **main.py** - hlavní skript, který bude obsahovat Python program
- **README.txt** - obsahuje několik velmi základních informací o tom, jak začít s deskou
- **pybcdc.inf** - soubor ovladače systému Windows, který slouží ke konfiguraci sériového zařízení USB

Nachází se zde 2 soubory, které jsou ve formátu python, ty se spustí při zapnutí zařízení. Nejdříve se spustí soubor se skriptem boot.py, potom následuje soubor main.py. Tyto soubory lze libovolně upravovat podle libosti. boot.py by měl obecně obsahovat zaváděcí parametry, kterými jsou například ladění sériového rozhraní, nebo ke kterému přístupovému bodu Wi-Fi se má deska připojit. V souboru main.py by se měla nacházet smyčka s hlavním kódem. [7] [9]

4.3 Úprava souborů

Jak je napsané v úvodu o souborovém systému, desky Nucleo mají obtížný přístup k souborům, proto je vhodnější namísto prostředí REPL použít správce souborů, který je součástí IDE.



Obr. 10 Správce souborů v IDE EsPy 1.0.0.12

4.3.1 Knihovna pyboard,py

Knihovna pyboard.py umožňuje vzdálené spouštění příkazů na cílové desce z počítače. Soubor pyboard.py je dostupný uvnitř projektu MicroPython v podadresáři nástrojů (tools).

4.3.1.1 Spuštění souborů umístěných v pc na desce

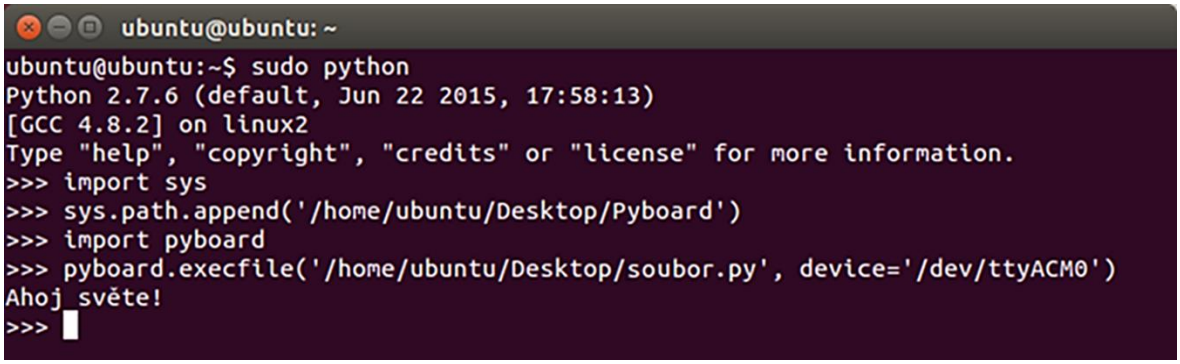
Pokud je potřeba na desce vykonat soubor napsaný v pythonu, který se nachází v pc, např. `soubor.py`, který obsahuje `print('Ahoj světe!')`, je k dispozici knihovna pyboard s příkazem `execfile`. Tato knihovna soubor načte a následně ho vykoná.

Postup ve Windows:

Nainstalování pyserial, otevření python 3.6 s oprávněním správce a zadání příkazů:

```
import sys
sys.path.append('C:/Users/Michal/Desktop')
import pyboard
pyboard.execfile('C:/Users/Michal/Desktop/soubor.py', device='COM3')
```

Alternativně lze podobný postup aplikovat v prostředí Linux s tím rozdílem, že název zařízení se jmenuje jinak. Prostředí python s oprávněním správce se otvírá příkazem `sudo python`.



Obr. 11 Postup spuštění souboru umístěného v OS Linux

4.3.1.2 Přenos dat z desky do pc

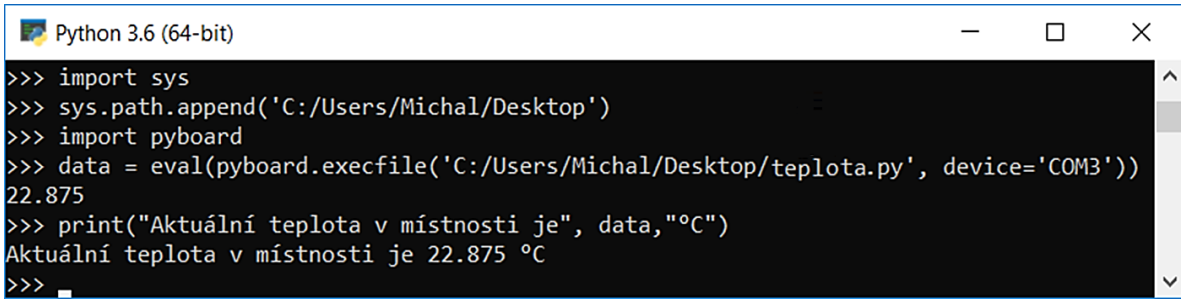
Pokud je potřeba přenést data z desky do pc pro další zpracování, je možné k tomu využít upravenou funkci execfile, která je součástí knihovny pyboard.py. Problém s modulem `pyboard.execfile` je takový, že výstup spíše zobrazí, než aby ho vrátil. Funkce `pyb.execfile` je v podstatě přesná kopie, která ale naopak vrátí výstup, než aby ho zobrazila. Proto se může upravit soubor `pyboard.py` tak, že se přidá řádek `return output` do funkce `execfile` následujícím způsobem:

```
def execfile(filename, device='/dev/ttyACM0'):
    pyb = pyboard.Pyboard(device)
    pyb.enter_raw_repl()
    output = pyb.execfile(filename)
    pyb.exit_raw_repl()
    pyb.close()
    return output
```

Po této úpravě lze spouštět soubory s tím, že se výstupní data uloží do proměnné v pc, kde se mohou dále zpracovávat.

Python 3.6 (program v pc)

```
import sys
sys.path.append('C:/Users/Michal/Desktop')
import pyboard
# použití funkce eval vyhlazuje rozdíl mezi python2 a python3
data = eval(pyboard.execfile('C:/Users/Michal/Desktop/teplota.py',
device='COM3'))
```



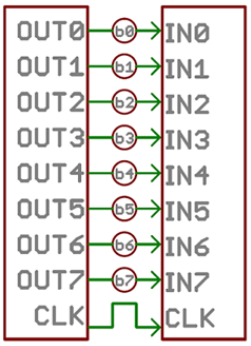
Obr. 12 Přenos dat z desky do proměnné v programu Python 3.6

5 Sériová komunikace

Aby si jednotlivé mikropočítačové moduly mohly vyměňovat své informace, musí sdílet společný komunikační protokol. Existují desítky komunikačních protokolů, které lze obecně rozdělit na paralelní a sériové.

Paralelní rozhraní

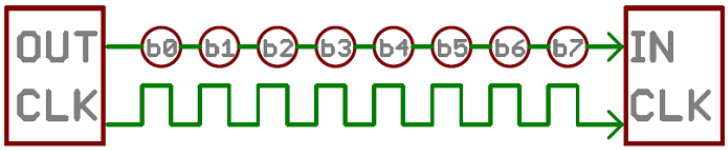
Paralelní rozhraní přenášejí více bitů najednou. Obvykle vyžadují sběrnice s daty, vysílající přes osm, šestnáct nebo více drátů. Data jsou přenášena v obrovských vlnách 1 a 0. Paralelní komunikace má jistě své výhody. Je rychlá a její implementace je relativně snadná. Vyžaduje však mnoho dalších vstupních a výstupních vývodů. [10]



Obr. 13 Příklad paralelního rozhraní, který vysílá jeden byte každý časový impuls. (převzato z [10])

Sériová rozhraní

Na rozdíl od paralelního rozhraní, sériové posílá data po jednom bitu, k tomu může využívat pouze jediný vodič. To umožňuje zmenšit rozměry pouzdra i desky, protože odpadá prostorově náročné propojování velkých počtů vodičů. Mezi nejznámější sériové rozhraní patří například USB, Ethernet, SPI, CAN a I2C. Každé z těchto sériových rozhraní lze roztřídit do jedné ze dvou skupin a to synchronní nebo asynchronní. [10]



Obr. 14 Příklad sériového rozhraní, který vysílá jeden bit každý časový impuls. (převzato z [10])

Sériová komunikace může být buď jednosměrná (simplexní), nebo obousměrná (duplexní).

Jednosměrná komunikace

Jednosměrná komunikace se využívá mezi vysílacím a přijímajícím zařízením. Využívají ji například sériové LCD displeje, které nepotřebují posílat žádné údaje zpět do řídicího zařízení.

Obousměrná komunikace

Obousměrná komunikace se dělí na poloviční a plný duplex. Rozdíl mezi nimi je ten, že při plném duplexu probíhá vzájemná komunikace současně. Při polovičním duplexu mohou obě strany vysílat i přijímat, ale ne současně, v libovolný čas probíhá přenos pouze jedním směrem.

5.1 Asynchronní sériová komunikace

Asynchronní sériový protokol má řadu pravidel, které pomáhají zajistit bezchybný přenos dat. Tyto mechanismy, které zavádíme, abychom mohli vyloučit vodič s hodinovým signálem, jsou synchronizační bity, paritní bity a modulační rychlost.

Zarámování dat

Každý blok (obvykle byte) přenášených dat je skutečně odeslán v paketu nebo v rámci bitů. Rámce jsou vytvářeny přidáním synchronizačních a paritních bitů do našich dat.

Obr. 15 Sériový rámec pro asynchronní komunikaci. *(převzato z [10])*

Synchronizační bity

Synchronizačními bity se myslí 1 start bit a 1, nebo 2 stop bity. Tyto bity označují začátek a konec paketu a používají se k synchronizaci vysílače s přijímačem. Pro každých 8 bitů odeslaných dat je tedy požadováno 10 bitů vysílacího času, které se negativně promítají do celkové přenosové rychlosti. [10]

Datový blok

Množství dat v každém paketu může být nastaveno na cokoli od 5 do 9 bitů. Nejběžnější velikost dat je 8, protože se přenáší celý byte. V případě, že se posílají pouze znaky v ASCII, je vhodnější použít 7-bitový datový blok. Po odsouhlasení délky znaku se obě sériová zařízení také musí shodnout na endianci svých dat. Posílají se data od nejvýznamnějšího bitu (msb) po nejméně významného, nebo naopak.

Paritní bity

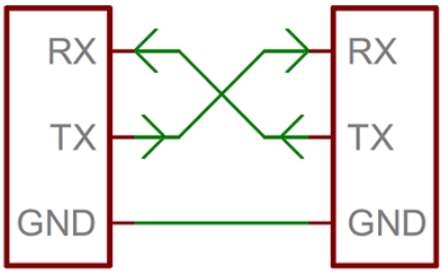
Parita je velmi jednoduchá kontrola chyb. Může se jednat o kontrolu sudosti nebo lichosti počtu jedniček v datovém bloku. Parita je nepovinná a není ani příliš rozšířená. Může být užitečná pro přenos dat v místě se zvýšeným elektromagnetickým zářením. Nevýhodou paritních bitů je zpomalení přenosové rychlosti. Přijaté údaje, které selhaly, musí být většinou znovu odeslány. [10]

Modulační rychlost

Modulační rychlost (Baud rate) určuje, jak rychle jsou data odeslána přes sériovou linku. Pokud se invertuje modulační rychlost, lze zjistit, jak dlouho trvá přenos jednoho symbolu. Tato hodnota určuje, jak dlouho má vysílač sériovou linku vysokou, nebo nízkou hodnotu, nebo v jakém časovém okamžiku přijímací zařízení vzorkuje svou linku. Čím vyšší je modulační rychlost, tím rychleji jsou data odesílána, nebo přijímána, ale existují limity na to, jak rychle je možné přenést data. Obvykle se u mikrokontrolérů používá modulační rychlost 115200 Bd. [10]

Propojení a hardware

Sériová sběrnice se skládá pouze ze dvou vodičů - jeden pro odesílání dat a druhý pro příjem. Sériová zařízení by proto měla mít dva sériové vývody: přijímací, RX a vysílací TX. Je důležité si uvědomit, že RX z jednoho zařízení by měl jít na TX druhého a naopak. Může se to zdát divné, ale má to logiku. Vysílač by měl vždy komunikovat s přijímačem a nikoliv s jiným vysílačem. [10]



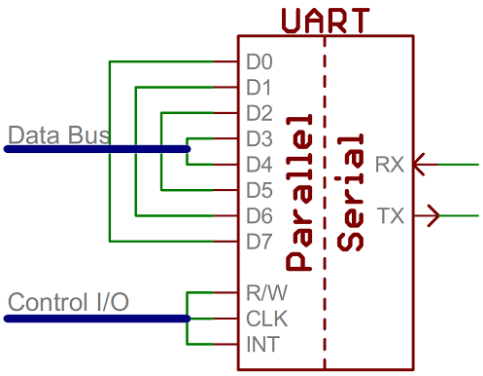
Obr. 16 Propojení sériové sběrnice (převzato z [10])

Nevýhody

Hlavní problém v asynchronních sériových portech spočívá v tom, že jsou samy o sobě vhodné pro přenos dat pouze mezi dvěma zařízeními. Přestože je možné připojit více zařízení k jedinému sériovému portu, problém sběrnice nastane, když se dvě zařízení pokoušejí současně řídit stejnou linku. Tento problém musí být pečlivě řešen, aby nedošlo k poškození dotyčných zařízení, obvykle prostřednictvím externího hardwaru. [10]

5.1.1 UART

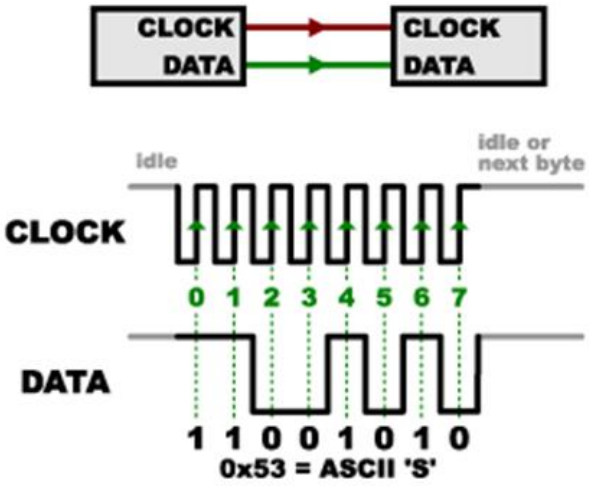
Když se připojí deska s MicroPythonem k počítači pomocí USB kabelu, je možné komunikovat se zařízením pomocí prostředí REPL, to je umožněný zařízením UART. Jedná se o univerzální asynchronní přijímač a vysílač, který je součástí mikrokontroléru. Hardware UART shromažďuje signály, které jsou posílány přes interní sběrnici pro další zpracování mikroprocesorem. V podstatě UART funguje jako prostředník mezi paralelním a sériovým rozhraním. Na jednom konci UART je sběrnice osmi datových linek (plus některé řídicí vývody) a na straně druhé jsou dva sériové vodiče - RX a TX. UART je odpovědný za odesílání a přijímání sériových dat. Na straně vysílání musí UART vytvořit datový paket - připojit synchronizační a paritní bity a poslat paket z linky TX s přesným časováním (podle nastavené modulační rychlosti). Na přijímacím konci má UART vzorkovat RX linku v rychlostech podle očekávané modulační rychlosti, vybírat synchronizační bity a rozdělit data. [10]



Obr. 17 *Zjednodušené rozhraní UART. (převzato z [10])*

5.2 Synchronní sériová komunikace

Synchronní přenos znamená, že je mezi zařízeními samostatný vodič (SCLK) s hodinovým signálem, který udržuje obě strany v synchronizaci a udává, kdy lze vzorkovat datový signál. Hodinový signál je oscilující signál, který říká přijímači, kdy přesně má vzorkovat bity datových vodičů. Vzorkovat se může na stoupající nebo klesající hraně hodinového signálu. Datasheet specifikuje, který z nich má být použit. Když přijímač detekuje tento okraj, okamžitě se podívá na datový vodič a přečte daný bit.



Obr. 18 Synchronní sériová komunikace (převzato z [11])

5.2.1 SPI

SPI je synchronní datové rozhraní, běžně používaný k odesílání dat mezi mikrokontroléry a malými periferními obvody, kterými jsou posuvné registry, čidla, vnější nebo vnitřní paměti, A/D převodníky a další obvody. Používá samostatné hodiny a datové vodiče spolu s výběrovým vodičem (ss) pro výběr zařízení, s nímž chce mikrokontrolér komunikovat. SPI je vhodný pro připojení vysokorychlostního (až do 10 MHz) plně duplexního datového přenosu.

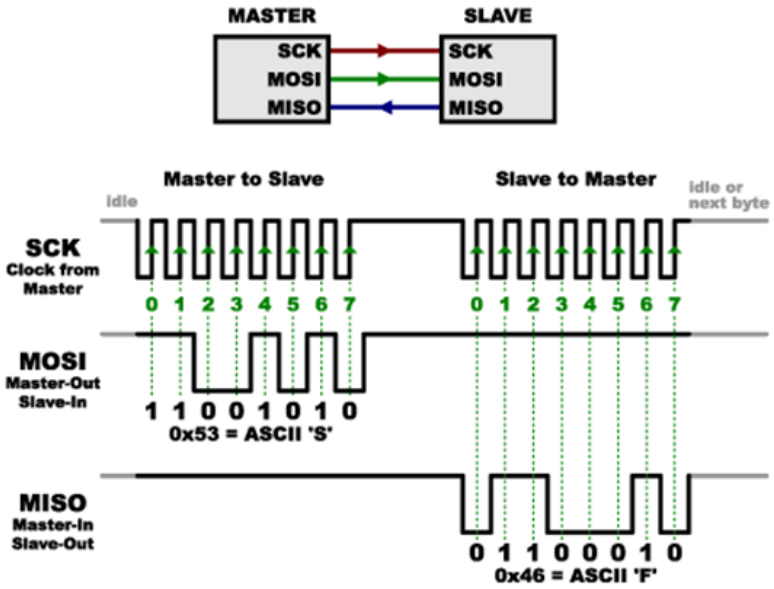
Hardware na obou koncích je obvykle velmi jednoduchý posuvný registr, který umožňuje snadnou implementaci v softwaru. [11]

Rozdíl mezi UART a SPI

UART využívá asynchronní komunikaci, která však přidává spoustu režijních nákladů ve formě synchronizačních bitů a používá poměrně komplikovaný hardware. Někdy je potřeba připojit pouze jednoduché periferie, které nemusí obsahovat takové možnosti. Proto má SPI jiný přístup, používá synchronní datovou sběrnici, která využívá hodinový signál a používá hierarchii zařízení. Z toho vyplívá, že je veškerá režie UART nahrazena jedním hodinovým signálem. Tento hodinový signál poskytuje primární zařízení (obvykle mikrokontrolér). Podobně jako má UART vodiče TX a RX, protokol SPI používá k přenosu dat vodiče MOSI a MISO. Všechna podřízená zařízení vysílají a přijímají data po těchto dvou vodičích. [11]

Přijímání dat

V SPI generuje hodinový signál (CLK, nebo SCK) pouze jedna strana, ta se nazývá "master", druhá strana je "slave". Vždy je jen jeden master (mikrokontrolér), ale může existovat několik zařízení slave. Když jsou data odesílána z masteru do slave, jsou odesílány po datovém vodiči s názvem MOSI (Master Out / Slave In). Pokud slave potřebuje odeslat odpověď zpět na master, pošle slave data přes třetí datový vodič nazvaný MISO (Master In / Slave Out), zatímco master bude nadále generovat hodinový signál.

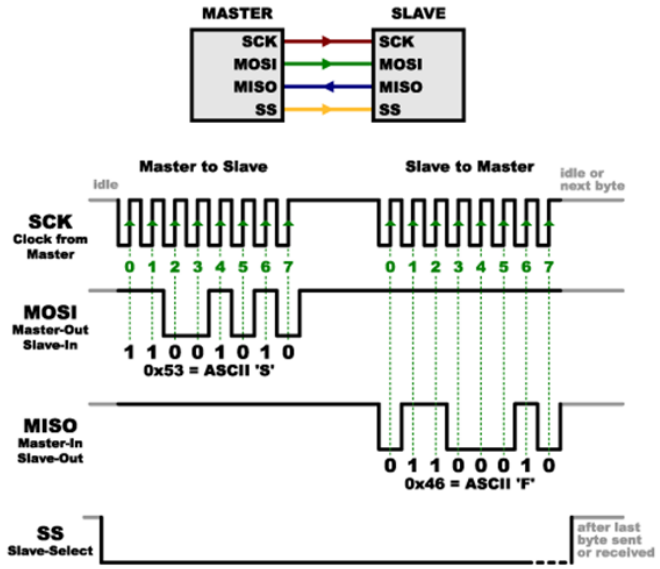


Obr. 19 Plně duplexní synchronní sériová komunikace (převzato z [11])

Vzhledem k tomu, že master vždy generuje hodinový signál, musí předem vědět, kdy slave potřebuje vrátit data a kolik dat bude vráceno. Toto je velmi zásadní rozdíl, vůči asynchronnímu sériovému přenosu, kde se může poslat náhodné množství dat v kterémkoliv směru a v libovolný čas. V praxi to není problém, jelikož SPI je obecně používán k rozhovoru se senzory, které mají velmi specifickou povelovou strukturu. Například pokud je do zařízení vyslán příkaz pro čtení dat, je pak jisté, že zařízení bude odpovídat pomocí předem napsaného pravidla. SPI je plně duplexní, a proto lze v určitých situacích vysílat a přijímat data současně.

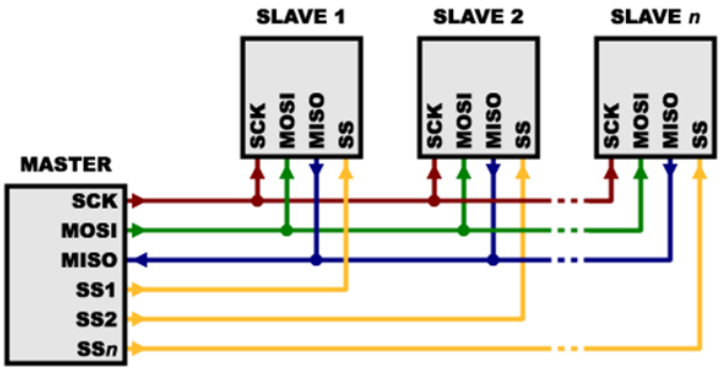
Slave select

Všechny jednotlivé obvody slave mají vtup slave select (SS), někdy nazvaný chip selct (CS). Tyto vstupy jsou propojeny samostatnými vodiči s obvodem master. Tak lze snadno vybírat obvod, se kterým má být v daném okamžiku vedena komunikace. Je-li signál vodiče SS ve vysoké úrovni, je daný slave neaktivní a jeho výstup MISO je ve stavu vysoké impedance. Pokud se však nastaví nízká úroveň, obvod slave se probudí a bude přijímat, či odesílat data. Po skončení přenosu dat se signál ve vodiči SS znovu nastaví na vysokou úroveň.



Obr. 20 Synchronní sériová komunikace s vodičem SS (převzato z [11])

Obecně platí, že každý obvod slave potřebuje samostatnou linku SS. Pokud chce master komunikovat s daným obvodem slave, nastaví signál vodiče SS na nízkou úroveň a zbytek vodičů SS nastaví vysokou úroveň. Tím se zamezí tomu, aby byli současně aktivovaný dva obvody slave a po vodiči MISO se posílali 2 signály najednou. [11]



Obr. 21 Synchronní sériová komunikace s několika vodiči SS (převzato z [11])

Programování SPI

Rozhraní může posílat data, kde je na první pozici nejvýznamnější bit (MSB) nebo nejméně významný bit (LSB). Obvody slave budou číst data buď na náběžné hraně nebo doběžné hraně hodinového impulsu. Při použití funkce SPI je nutné nastavit vývody SCK, MOSI a MISO. Navíc lze nastavit specializovaný pin SS (přinejmenším se musí nastavit jako výstupní, aby správně fungoval hardware SPI). Vzhledem k vysoké rychlosti přenášených signálů by měl být SPI používán pouze k odesílání dat na krátké vzdálenosti (až několik m). Pro odesílání dat na delší vzdálenost je vhodné zvážit snížení frekvence. Maximální frekvence hodinového signálu je 2 MHz.

Vlastnosti SPI

Výhody SPI jsou, že je rychlejší, než asynchronní sériový spojení**,** hardware pro příjem může být jednodušší, než posuvný registr a podporuje více připojených obvodů slave.

Mezi nevýhody SPI patří to, že vyžaduje více vodičů, než jiné komunikační metody, komunikace musí být předem definovaná (nelze odesílat náhodné množství dat, kdykoli je to požadováno), master musí řídit veškerou komunikaci (obvody slave nemohou komunikovat přímo mezi sebou), nebo to, že každý obvod slave obvykle vyžaduje oddělené SS linky, což může být problematické, pokud je potřeba mnoho obvodů slave.

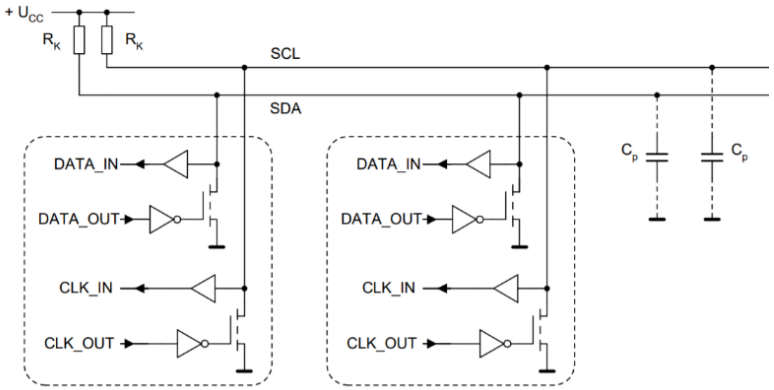
5.2.2 I²C

Protokol I2C je stejně jako SPI určený pro komunikaci pouze na krátké vzdálenosti. Jeho maximální délka je omezena nejvyšší přípustnou kapacitou na sběrnici 400 pF. I2C vyžaduje pouze dva signální vodiče pro výměnu informací a umožňuje komunikaci několika

obvodů slave s jedním, či více obvody master s tím, že hlavní zařízení nemohou vzájemně komunikovat mezi sebou. Nevýhodou I2C je, že nedosahuje rychlosti přenosu dat SPI, takže pro zařízení, která potřebují rychlé přenosy dat, je vhodnější použít SPI. Většina zařízení I2C může komunikovat na 100kHz nebo 400kHz. [12]

Vodiče

Každá I2C sběrnice využívá dva vodiče, které se nazývají SCL a SDA. SCL je vodič s hodinovým signálem a SDA je datový vodič. Hodinový signál je generován vždy aktuálním obvodem master. Některá zařízení slave mohou v určitých časech vynucovat hodiny k tomu, aby se zpozdilo odesílání dalších dat, tato činnost se nazývá "roztahování hodin".



Obr. 22 Sběrnice I^2C

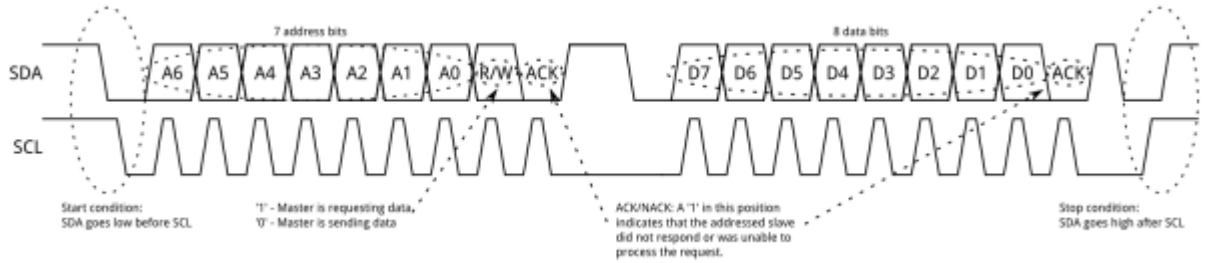
Na rozdíl od připojení UART nebo SPI, jsou ovladače sběrnice I2C v podobě otevřeného kolektoru (open drain), což znamená, že jsou signály ve výchozím stavu ve vysoké úrovni a zařízení ovlivňují signál tím, že nastaví signál na nízkou úroveň. Tím se zabrání tomu, aby jeden obvod nastavoval signál na vysokou úroveň, zatímco jiný se pokoušel o nízkou úroveň (eliminuje to situaci, kde by si konfliktní zařízení mohla způsobit vzájemné poškození, v lepším případě by pouze docházelo k nadměrným ztrátám výkonu v systému). [12]

Pull-up rezistory

Každý signální vodič má na sobě pull-up rezistor, aby obnovil signál na vysokou úroveň, když jej žádné zařízení nevyužívá. Výběr rezistorů se liší podle zařízení na sběrnici, ale většinou se začíná s hodnotou 4,7k. U dlouhých délek drátu, nebo systémů s mnoha zařízeními je lepší použít menší rezistory.

Protokol

Signál musí dodržovat určitý protokol pro zařízení na sběrnici, aby ji rozpoznal jako platnou komunikaci I2C.

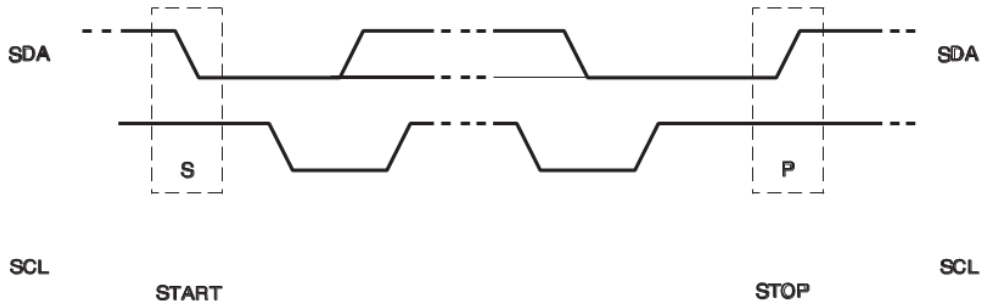


Obr. 23 Formát rámců se sedmibitovou adresou (převzato z [12])

Zprávy jsou rozděleny do dvou typů rámců. První je adresní rámec, kde master říká, ke kterému obvodu slave se má zpráva odeslat. Druhý rámec je datový, který obsahuje 8bitovou datovou zprávu předávanými z masteru na slave, nebo naopak (podle bitu R/W). Data jsou posílána po vodiči SDA, zatímco vodič SCL je držen na nízké úrovni a po vzorkování linky SCL se odebírá vzorek. Čas mezi hranou hodinového signálu a čtením/zápisem dat je definován zařízeními na sběrnici a bude se lišit od čipu k čipu. [12]

Podmínka START

Pro inicializaci rámce adresy, master opouští SCL a nastaví na SDA nízkou úroveň. To znamená, že se pošle podmínka START. Tím master upozorní všechny podřízené přístroje, že začne vysílání.



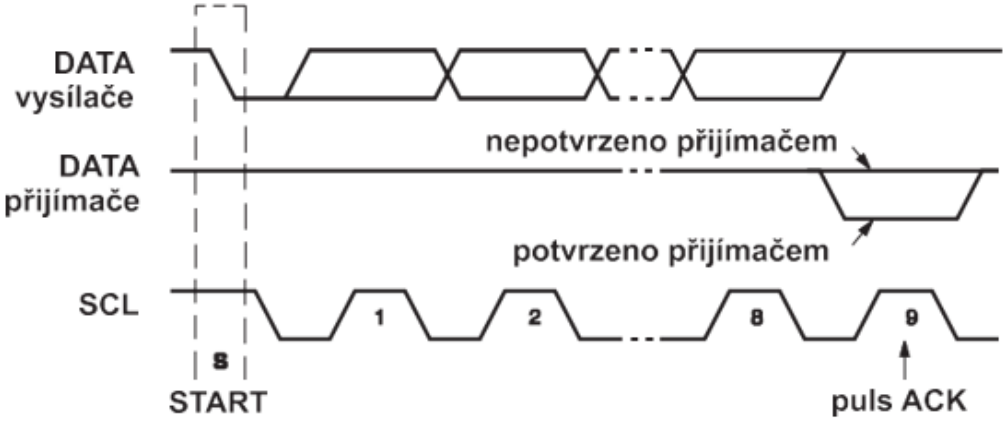
Obr. 24 Podmínky START a STOP (převzato z [13])

Adresový rámec

Každé připojené zařízení má určenou svou vlastní adresu o délce 7 nebo 10 bitů, která slouží k jeho výběru. Po 7-bitové adrese se posílá bit R / W indikující, zda se jedná o operaci čtení (1) nebo zápisu (0).

Potvrzování

9. bit rámce je bit ACK. Ten se nachází ve všech rámcích, včetně adresového. Po odeslání prvních 8 bitů rámce je zařízení pro příjem přijímáno přes SDA. Pokud přijímající zařízení nenastaví vodič SDA na nízkou úroveň před 9. hodinovým pulsem, lze předpokládat, že přijímací zařízení buď neobdrželo data, nebo nevědělo, jak zprávu analyzovat. V takovém případě se výměna zastaví a je na masterovy rozhodnout, jak postupovat. [12]



Obr. 25 Potvrzování příjmu bitem ACK (převzato z [13])

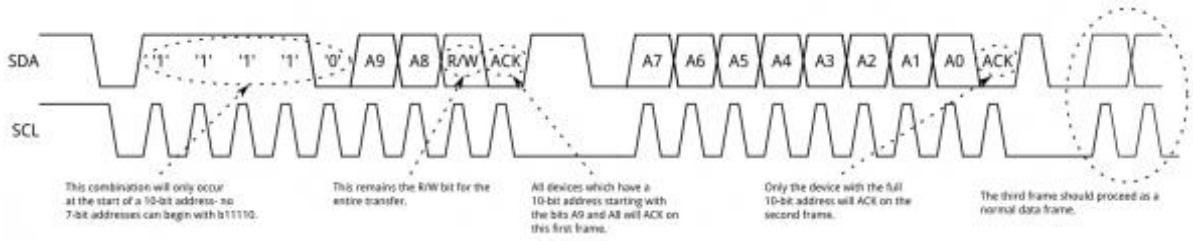
Datové rámce

Po odeslání rámce adresy se mohou přenést data. Master bude jednoduše pokračovat v generování časových impulzů v pravidelném intervalu a data budou na SDA umístěna buď masterem nebo slavem v závislosti na tom, zda bit R / W indikoval operaci čtení nebo zápisu. Počet datových rámců je libovolný a většina obvodů slave automaticky zvýší vnitřní registr, což znamená, že následující čtení nebo zápis bude pocházet z dalšího registru ve vodiči.

Podmínka STOP

Jakmile budou odeslány všechny datové rámce, master nastaví vodič SDA na vysokou úroveň, po přechodu SCL také na vysokou úroveň. To znamená, že se vytvoří podmínka STOP. Během normálního zápisu dat by se hodnota SDA neměla měnit, pokud je hodnota SCL vysoká, aby nedocházelo k falešným podmínkám zastavení.

10-bitová Adresace



Obr. 26 10-bitová adresace (převzato z [12])

V 10-bitové adresaci jsou pro přenos slave adresy vyžadovány dva rámce. První rámec se skládá z kódu b11110xyz, kde "x" je MSB adresy slave, y je bit 8 slave adresy a z je bit pro čtení / zápis, jak je popsáno výše. Bit ACK prvního snímku bude uplatněn všemi podřízenými obvody, které odpovídají prvním dvěma bitům adresy. Stejně jako při normálním 7-bitovém přenosu ihned začne jiný přenos a tento přenos obsahuje bity 7: 0 adresy. V tomto okamžiku by adresovaný slave měl reagovat bitem ACK. Pokud tomu tak není, režim selhání je stejný jako 7-bitový systém. 10-bitové adresové zařízení mohou koexistovat se 7-bitovými adresovými zařízeními, protože přední část "11110" adresy není součástí žádné platné 7-bitové adresy. [12]

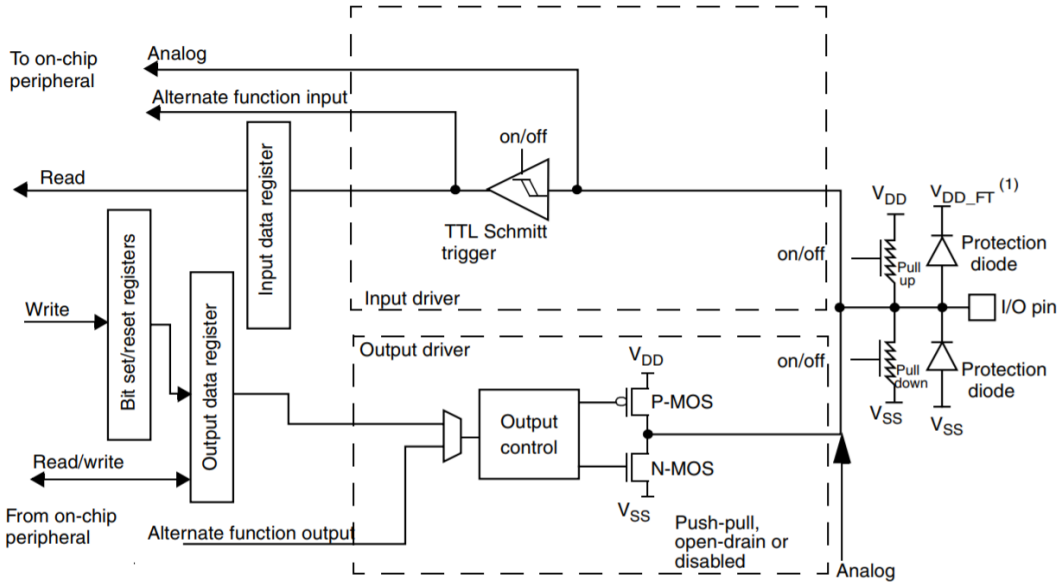
6 Vzorové příklady

6.1 NUCLEO-F411RE s rozšiřující deskou

Obr. 27 Vývojový kit NUCLEO-F411RE

6.1.1 Nastavení GPIO

GPIO jsou univerzální vstupně/výstupní vývody mikrokontroléru. Na některých vývodech GPIO lze nastavit alternativní funkce (AF) pro specifické účely, kterými jsou čítač, časovač, I2C, SPI, USART, OTG, nebo SDOI. Přesný výčet funkcí a vývodů je specifický pro jednotlivé mikrokontroléry a je uvedený v datasheetu. [14]



Obr. 28 Vnitřní nastavitelné uspořádání GPIO (převzato z [14])

Nastavovat jednotlivé vývody GPIO lze jak knihovnou machine, tak pyb. V obou případech se inicializuje objekt Pin, kterému lze nastavit základní vlastnosti. Těmi jsou například to, zda se jedná o vstup nebo výstup, open drain, případně alternativní funkce. Dále lze parametrem pull nastavit pull-up, či pull-down rezistor.

Vypsání aktuálního nastavení vývodů

```
import machine
help(machine.Pin.board)
```

Rozsvícení LED

```
from machine import Pin
# vytvoření objektu vývodu PA5, jako výstup
led = Pin('PA5', 1)
# rozsvícení LED
led.on()
```

Nastavení GPIO jako vstup

```
# vytvoření objektu vývodu PA5, jako vstup
led = Pin('PA5', 0)
```

Blikání LED

```
from machine import Pin
led = Pin('PC0', 1)
for x in range(0, 30):
    if led.value() == 1:
        led(0)
    else:
        led(1)
    # čekání 100 ms (perioda blikání je 200 ms)
    pyb.delay(100)
```

6.1.2 Řízení frekvence CPU

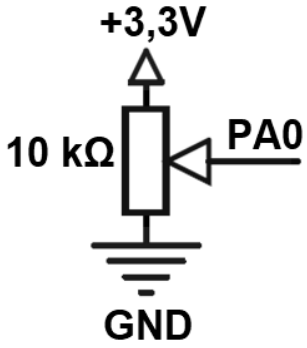
```
import pyb
sysclk =  16, 24, 30, 32, 36, 40, 42, 48, 54, 56, 60, 64, 72, 84,
96, 108, 120, 144
for s in sysclk:
    # nastavení frekvence CPU zadano v MHz
    pyb.freq(s * 1000000)
    # aktualizace modulační rychlosti UART
    pyb.repl_uart().init(baudrate=115200)
    # výpis aktuální frekvence do terminálu
    pyb.freq()
```

6.1.3 Čítač / Časovač

```
# PWM na modré LED
from pyb import Pin, Timer
tim = Timer(1, freq=1000)
ch = tim.channel(1, Timer.PWM, pin=Pin('PB13'))
ch.pulse_width_percent(50)
```

6.1.4 Čtení odporu potenciometru

Čtení odporu potenciometru lze provést A/D převodníkem, kde se hodnota odporu přímo převádí na napětí děličem.



Obr. 29 Schéma zapojení potenciometru

Čtení hodnot A/D převodníku

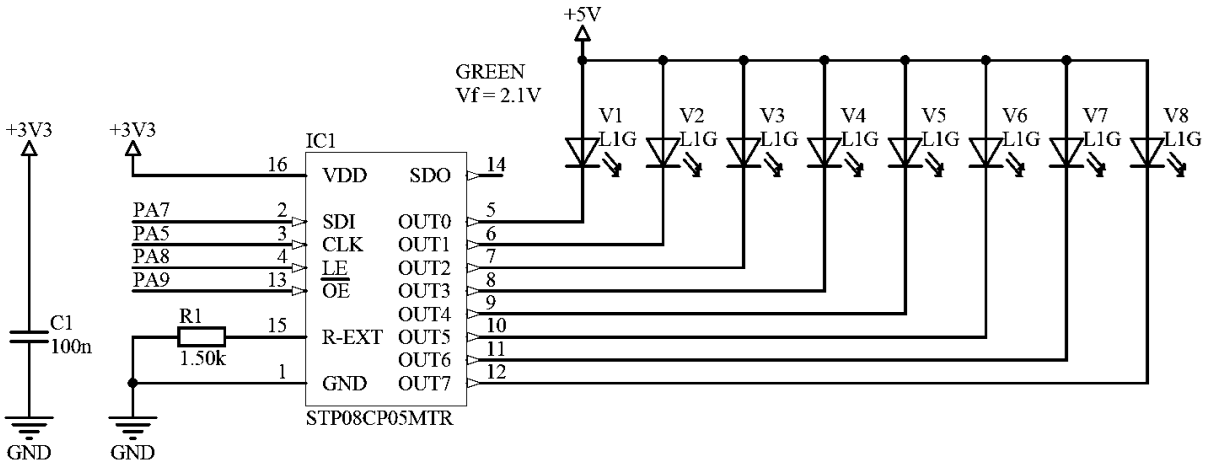
```
from pyb import Pin
pin = pyb.ADC(Pin('PA0'))
for x in range(100):
    INT = pin.read()
    HEX = hex(pin.read()
    print('Int:', INT, 'Hex:', HEX, '-' * int(INT * 0.01),'O')
    pyb.delay(100)
```

```
Int: 5    Hex: 0x2   O
Int: 2    Hex: 0x2   O
Int: 133  Hex: 0x85  - O
Int: 274  Hex: 0x113 -- O
Int: 601  Hex: 0x259 ----- O
Int: 754  Hex: 0x2f2 ------- O
Int: 1043 Hex: 0x40c ---------- O
Int: 1449 Hex: 0x5aa ------------- O
Int: 1765 Hex: 0x6e5 ----------------- O
Int: 2099 Hex: 0x839 -------------------- O
Int: 2452 Hex: 0x990 ------------------------ O
Int: 2749 Hex: 0xabe --------------------------- O
Int: 3172 Hex: 0xc60 ------------------------------ O
Int: 3540 Hex: 0xdd5 ---------------------------------- O
Int: 4093 Hex: 0xffe ----------------------------------------- O
Int: 4093 Hex: 0xfff ----------------------------------------- O
```

Obr. 30 Výstup z terminálu při ladění potenciometrem

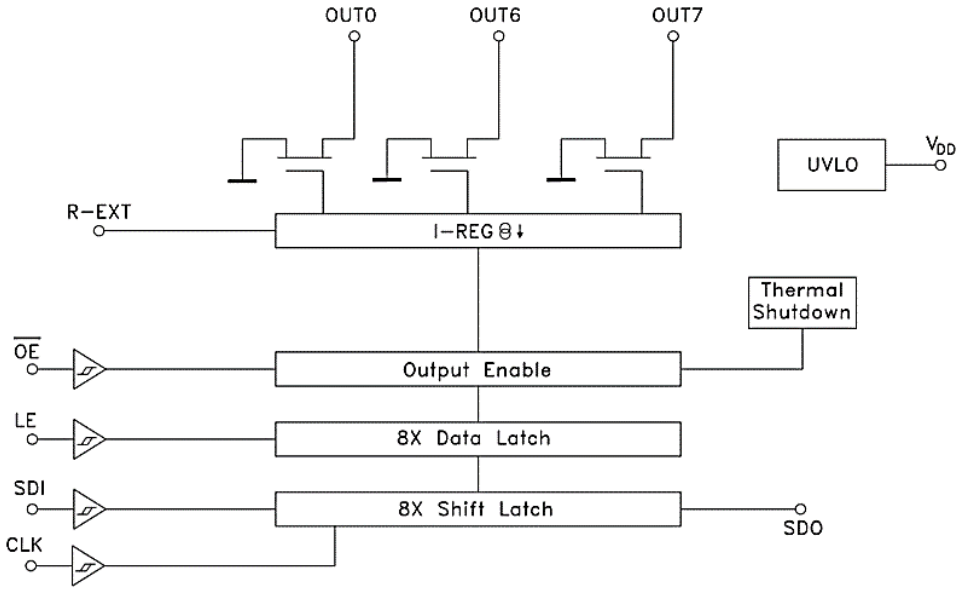
6.1.5 Ovládání posuvného registru přes SPI

Jedna ze součástek, kterou lze rozhraním SPI ovládat, je 8-bitový posuvný registr STP08CP05 s osmicí zelených LED.



Obr. 31 Schéma zapojení posuvného registru

Pro příjem dat je potřeba nejdříve nastavit vývod OE (Output Enable) do Log. 0, potom poslat data přes rozhraní SPI a nakonec data překlopit do I-registru tím, že se pošle krátký impuls na vstup LE (Input Letch).



Obr. 32 Blokový diagram posuvného registru (převzato z [15])

Inicializace rozhraní SPI

Nejdříve je nutné nastavit vývody. K tomu se využívá třída s názvem `SPI`, která obsahuje přednastavené sety různých vývodů. Tyto sety jsou reprezentovaný parametrem id.

pro id = 1 je NSS = PA4, SCK = PA5, MISO = PA6 a MOSI = PA7
pro id = 2 je NSS = PB12, SCK = PB13, MISO = PB14 a MOSI = PB15

Mezi nejdůležitější parametry, které lze při inicializaci nastavit patří:

- **mode** - specifikace obvodu SPI.MASTER, nebo SPI.SLAVE.
- **baudrate** - modulační rychlost SCK (má smysl pouze u režimu master).
- prescaler - dělička k odvození SCK z frekvence APB sběrnice. Použitím děličky se přepíše modulační rychlost SCK (argument baudrate).
- polarity - úroveň nečinnosti hodin, nabývá hodnoty 0, nebo 1.
- phase - vzorkování dat na náběžné, nebo doběžné hraně bitu, nabývá hodnot 0, nebo 1.
- bits - počet bitů v každém převedeném slovu, může nabývat hodnot 8, nebo 16.
- firstbit - nabývá hodnot SPI.MSB, nebo SPI.LSB.

Posílání dat

Pro posílání dat mezi obvodem mater a slave se používá metoda `send` s prvním povinným parametrem, kterým jsou data ve formě integer, nebo buffer objekt. Dále lze nastavit nepovinný parametr timeout, který udává časový limit v ms pro čekání na odeslání.

SPI komunikace využitím knihovny pyb

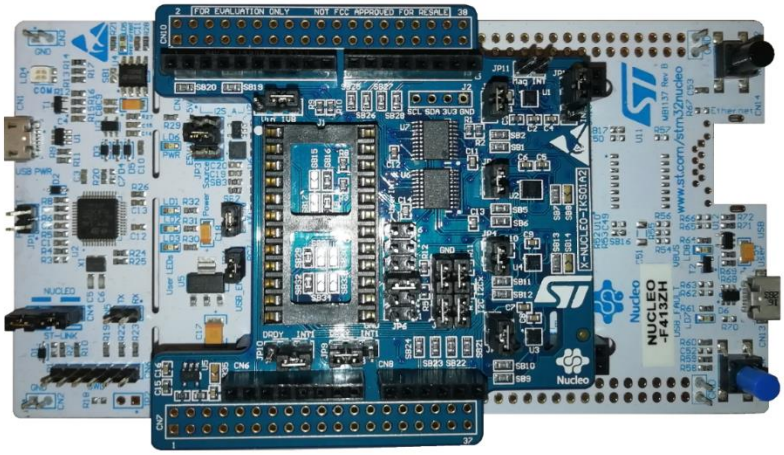
```
import pyb
LE = pyb.Pin("PA8", 1)
OE = pyb.Pin("PA9", 1)
ObjectSPI = pyb.SPI(1)
ObjectSPI.init(pyb.SPI.MASTER, baudrate=10000)
# poslání hodnoty 10 do posuvného registru [00001010](bin)
ObjectSPI.send(10)
# krátký impuls na pin LE
LE.on()
LE.off()
```

SPI komunikace pomocí knihovny machine

U většiny hardwarových bloků SPI (vybraných podle parametru id) jsou piny pevně nastavené a nelze je měnit. V některých případech tyto bloky umožňují 2-3 alternativní sady pinů. Pro tyto případy lze, na rozdíl od knihovny pyb, v knihovně machine přiřadit funkce k výstupům ručně za použití parametru id = -1. V tomhle případě se přiřadí číslo vývodu k jeho funkci (SCK, MISO a MOSI). Dalším rozdílem oproti knihovně pyb je ten, že se metoda pro posílání dat jmenuje write a v jejím argumentu může být pouze buffer. Ten se vytvoří klíčovým slovem `bytearray`.

```
import machine
from machine import Pin
ObjectSPI = machine.SPI(-1, sck=Pin("PA5"), miso=Pin("PA6"),
mosi=Pin("PA7"))
LE = Pin("PA8",1)
OE = Pin("PA9",1)
# poslání hodnoty 3 do posuvného registru [00000011](bin)
ObjectSPI.write(bytearray([3]))
LE.on()
LE.off()
```

6.2 NUCLEO-F413ZH s rozšiřující deskou UM2121



Obr. 33 Deska NUCLEO-F413ZH

6.2.1 Modul UM2121

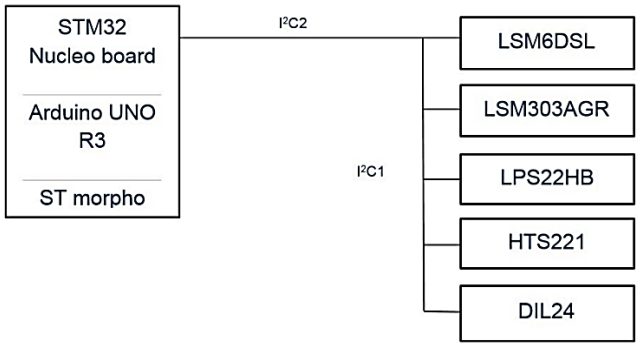
Modul UM2121, neboli X-NUCLEO-IKS01A2 obsahuje 4 senzory.

Popis	Název	Označení na DPS	Adresa na I2C sběrnici
Magnetický senzor	LSM303AGR	U1	30
3D akcelerometr a 3D gyroskop	LSM6DSL	U2	107
Senzor vlhkosti a teploty	HTS221	U3	95
Senzor tlaku	LPS22HB	U4	93
Patice DIL24			25

Umístění senzorů na DPS a sériovém rozhraní I2C

Fyzické nastavení propojek na modulu umožňuje přímý přístup k senzorům (I2C1 = I2C2):

JP7: 1-2 3-4 JP8: 1-2 3-4



Obr. 34 Fyzické propojení modulů (převzato z [16])

Přesvědčit se o správném propojení modulů lze jednoduše metodou `i2c.scan`, která skenuje všechny adresy od 0x08 do 0x77 a vrátí seznam adres zařízení, které reagují.

Přístroj reaguje, pokud nastaví vodič SDA do log. 0, a poté po něm odešle svoji adresu (včetně bitů pro zápis).

```
from machine import I2C
i2c = I2C(freq=100000, scl="PB8", sda="PB9")
i2c.scan()
```

Nastavení registrů v senzoru

Nastavení registrů přes rozhraní I2C lze provést metodou `writeto_mem` se 3 parametry, kde první parametr je adresa připojeného zařízení na sběrnici. Druhý parametr je adresa registru, který se upraví. Třetím parametrem je buffer s hodnotou, která se zapíše do registru. Například příkaz `i2c.writeto_mem(107, 0x10, b'\x52')` zapíše do zařízení, který je připojené na sběrnici pod adresou 107, do osmibitového registru 0x10 hodnotu 0x52, neboli 01010010.

Čtení registrů ze senzoru

Čtení registrů lze provádět metodou `readfrom_mem`, opět se třemi parametry. První parametr je opět adresa připojeného zařízení na sběrnici. Druhý parametr je adresa registru, který se přečte. Třetí parametr je počet byte, které se přečtou (většinou 1). Například příkaz `i2c.readfrom_mem(107, 0x22, 1)` přečte 1 byte z registru 0x22.

6.2.2 3D gyroskop LSM6DSL

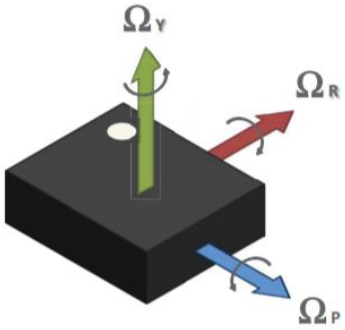
Nastavení frekvence 104 Hz pro generování dat

```
Registr CTRL2_G (11h)
ODR_G [4-7] → 104 Hz → 0100
0100 0000 (bin) = 0x40 (hex) = 64 (dec)
i2c.writeto_mem(107, 0x11, b'\x40')
```

Nastavení frekvence 104 Hz FIFO zásobníku a nepřetržité přepisování dat po přetečení zásobníku

```
Registr FIFO_CTRL5 (0Ah)
FIFO ODR [3-6] → 104Hz → 0100
FIFO MODE [0-2] → Continous mode → 110
0 0100 110 (bin) = 0x26 (hex) = 38 (dec)
i2c.writeto_mem(107, 0x0A, b'\x26')
```

Čtení registrů z gyroskopu



Obr. 35 Tří-osý gyroskop (převzato z [17])

Každá ze tří os otáčení senzoru P (pitch), R (roll) a Y (yaw) se ukládá do dvou 8-bitových registrů. V datasheetu jsou tyto registry označený následovně: [17]

```
Pitch:      OUTX_L_G (22h), OUTX_H_G (23h)
Roll:       OUTY_L_G (24h), OUTY_H_G (25h)
Yaw:        OUTZ_L_G (26h), OUTZ_H_G (27h)
```

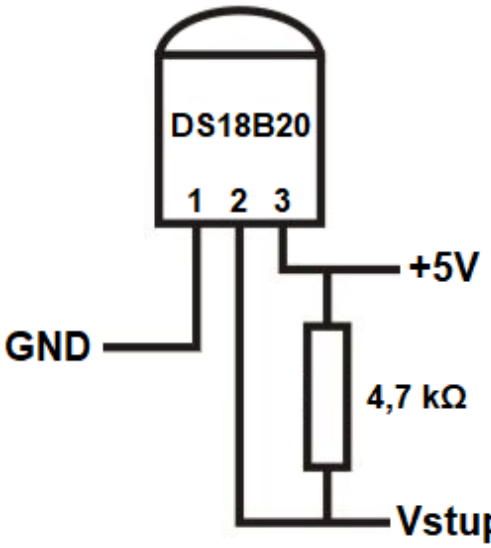
Aby se vytvořilo 16-bitové číslo, je k tomu použito formátování čísla na 16-bitové a následně bitový posuv, např. registr OUTX_H_G se posune o 8 bitů doleva a překryje se s registrem OUTX_L_G.

```
from machine import I2C
from pyb import delay
from binascii import hexlify
from math import asin, degrees
# nastavení registrů
i2c = I2C(freq=100000, scl="PB8", sda="PB9")
i2c.writeto_mem(107, 0x10, b'\x40')
i2c.writeto_mem(107, 0x11, b'\x40')
i2c.writeto_mem(107, 0x0A, b'\x20')
# vytvoření pole s daty z gyroskopu
field = []
for i in range(6):
    field.append(i2c.readfrom_mem(107, 34+i, 1))
# aktualizace dat a následné dekódování dat z bufferu
for i in range(6):
    field[i]  = i2c.readfrom_mem(107, 34+i, 1)
    field[i] = hexlify(field[i])
    field[i] = field[i].decode()
    field[i] = int(field[i], 16)
```

```
# převod hodnot gyroskopu na škálu od -90 do 90 úhlových stupňů
for x in range(6):
    field[x] = field[x*2] + (field[(x*2)+1]<<8)
    field[x] = field[x] / 65535
    if(field[x] > 0.5):
        field[x] = -(1 - field[x])
    field[x] = 4 * field[x]
    if(field[x] > 1):
        field[x] = 1
    if(field[x] < -1):
        field[x] = -1
    field[x] = degrees(asin(field[x]))
# výpis náklonu pro všechny 3 osy
print('Pitch(X): ' + str("%.1f" % field[0]) + ' deg')
print('Roll(Y): ' + str("%.1f" % field[1]) + ' deg')
print('Yaw(Z): ' + str("%.1f" % field[2]) + ' deg')
```

6.2.3 Měření teploty senzorem DS18B20

Teplotní snímač DS18B20 využívá pro přenos informací o teplotě jedno vodičovou sériovou sběrnici. MicroPython pro komunikaci přes jeden vodič používá knihovnu onewire. Ta spolu s knihovnou DS18B20 umožňuje správný odečet teploty. Aby kód fungoval, musí se nejprve správně zapojit čidlo s datovým vodičem připojeným k vývodu GPIO. Mezi napájecí a datový vodič se musí podle datasheetu připojit rezistor o hodnotě 4,7 kΩ, při této hodnotě rezistoru v mém případě nefungovala správně komunikace, problém se vyřešil výměnou rezistoru za menší o velikosti 2,2 kΩ.



Obr. 36 Zapojení teplotního čidla DS18B20

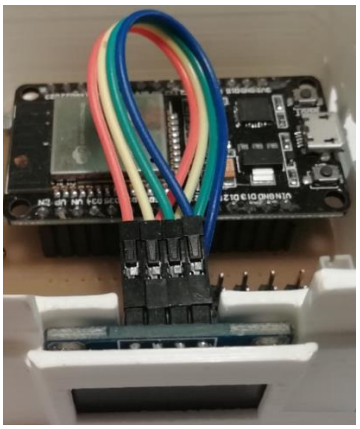
```
import time
from ds18x20 import DS18X20
from machine import Pin
from onewire import OneWire
# datový vodič vývodu PC0
pin = Pin("PC0")
ow = OneWire(pin)
ds = DS18X20(ow)
ow.reset()
# skenování připojených zařízení na sběrnici
roms = ds.scan()
print('Nalezene zarizeni:', roms)
# odečet hodnoty čidlem
ds.convert_temp()
# čekání na odečet hodnoty čidlem 1s
time.sleep_ms(1000)
# výpis naměřené teploty
for rom in roms:
    print('Aktualni teplota je: 'ds.read_temp(rom)' C')
```

```
Nalezene zarizeni: [bytearray(b'(\xd38\xe3\x05\x00\x00\xcd')]
Aktualni teplota je: 26.125 C
>>>
```

Obr. 37 Výpis teploty do konzole

6.3 ESP32-WROOM[2] s OLED displejem

Wi-Fi modul WROOM lze použít pro širokou škálu aplikací od zpracování jednoduchých senzorů, ze kterých modul vytvoří IOT, až po virtuálního asistenta se streamováním hudby.



Obr. 38 ESP32-WROOM s OLED displejem

[2] Windows 10 nemusí rozpoznat Wi-Fi modul wroom-32 korektně, dokud se nenainstaluje driver s názvem CP210x_Universal_Windows_Driver.

6.3.1 OLED SSD1306 ovládaný přes I2C

```
from machine import Pin
from machine import I2C
import machine
# inicializace I2C
i2c = I2C(scl=Pin(22), sda=Pin(21))
# nastavení displeje o velikosti 128x64 pixelů
oled = SSD1306_I2C(128, 64, i2c)
```

Vykreslování

Vykreslování objektů zde probíhá ve dvou fázích, nejdříve se změní obsah vyrovnávací paměti (bufferu) a až po vykonání příkazu `oled.show()` se změny projeví na displeji. Metody, které se zde používají, mají většinou poslední parametr, který představuje barvu. Ta má smysl hlavně u barevného displeje, u jednobarevného displeje je tento parametr vždy 0 pro zhasnutí a 1 pro rozsvícení daných pixelů.

```
# nastavení všech pixelů na 1
oled.fill(1)
# nastavení všech pixelů na 0
oled.fill(0)
# 1 pixel o souřadnicích x=2, y=5
oled.pixel(2, 5, 1)
# horizontální (vodorovná) úsečka, začínající na souřadnicích x=3,
y=9 a je dlouhá 110 pixelů.
oled.hline(3, 9, 110, 1)
# vertikální (svislá) úsečka, začínající na souřadnicích x=5, y=10 a
je dlouhá 20 pixelů.
oled.vline(5, 10, 20, 1)
# obdélník, začínající na souřadnicích x=2, y=8 a končící na
souřadnicích x=112, y=16.
oled.rect(2, 8, 112, 16, 1)
# vykreslení textu, který začíná na souřadnicích x=6, y=12.
oled.text('TEXT', 6, 12)
```

6.3.2 Wi-Fi s wroom-32

Aplikace, která je níže lze rozdělit na 2 části: 1. částí je připojení k již vytvořené síti, nebo vytvoření vlastní sítě a 2. částí je připojení k serveru, vytvoření serveru, anebo ovládání desky přes webREPL. Tyto možnosti z obou částí lze libovolně kombinovat.

Import potřebných knihoven a funkcí

```
from machine import Pin
from machine import I2C
from ssd1306 import SSD1306_I2C
import machine
import socket
import network
```

Připojení modulu k bezdrátové síti

```
sta = network.WLAN(network.STA_IF)
# aktivace rozhranní
sta.active(True)
sta.scan()
# připojení k přístupovému bodu AP
sta.connect("NAZEV_SSID", "HESLO")
s = socket.socket()
s.bind(('', 80))
s.listen(1)
```

Po připojení k síti se připojuje k přidělené ip adrese, která se zobrazí v terminálu pod názvem sta_ip po připojení desky k síti, v tomhle případě je to http://192.168.102.105.

```
>>> I (342609) wifi: connected with TP-Link, channel 3
;32mI (343829) event: sta ip: 192.168.102.105, mask: 255.255.255.0, gw: 192.168.102.1m
```

Obr. 39 Výpis z konzole po připojení k síti

Vytvoření vlastní sítě

```
ap = network.WLAN(network.AP_IF)
ap.active(True)
ap.config(essid="Nazev_SSID")
s = socket.socket()
s.bind(('', 80))
s.listen(1)
```

Po vytvoření sítě se připojuje k výchozí bráně (Default Gateway), která je ve výchozím nastavení http://192.168.4.1, případně ji lze zjistit po připojení k Wi-Fi síti otevřením příkazového řádku cmd a zadáním příkazu `ipconfig`.

```
Wireless LAN adapter Wi-Fi:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::2050:3ef7:db6d:2b37%3
   IPv4 Address. . . . . . . . . . . : 192.168.4.3
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.4.1
```

Obr. 40 Nastavení vytvořené sítě

Připojení k stávajícímu serveru

```
addr_info = socket.getaddrinfo("towel.blinkenlights.nl", 23)
addr = addr_info[0][-1]
s = socket.socket()
s.connect(addr)
while True:
    data = s.recv(500)
    print(str(data, 'utf8'), end='')
```

Natavení vlastního serveru

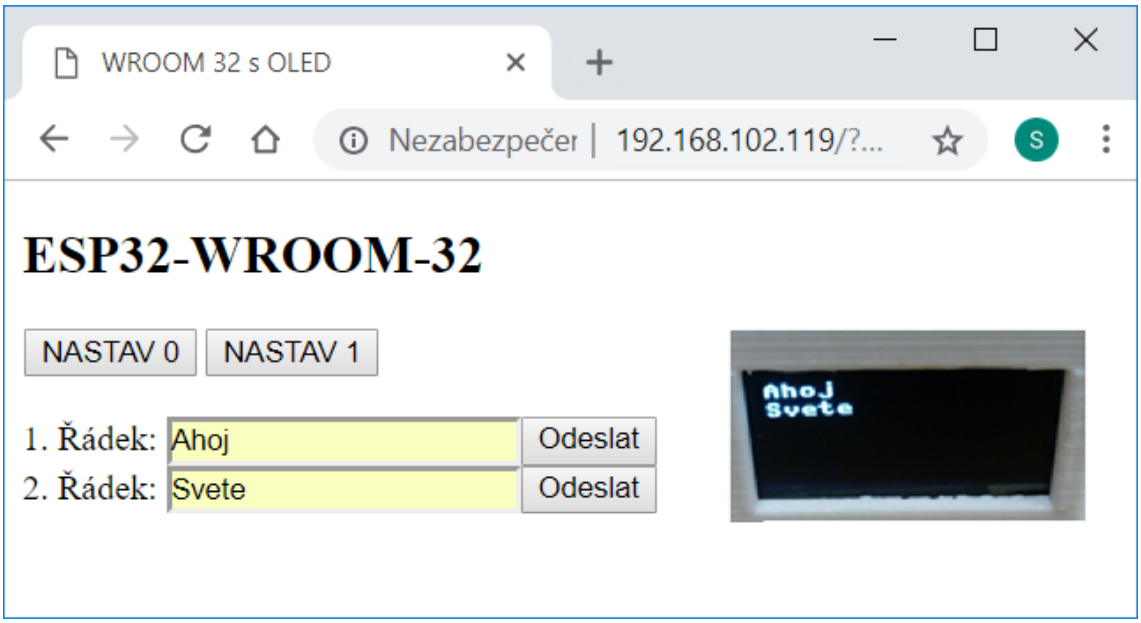
```
i2c = I2C(scl=Pin(22), sda=Pin(21))
oled = SSD1306_I2C(128, 64, i2c)
oled.fill(1)
oled.show()
```

HTML kód, který se zobrazí připojeným zařízením

```
html = """<!DOCTYPE html>
<html>
<head>
    <title>WROOM 32 s OLED</title>
    <meta charset="UTF-8" />
</head>
    <body>
    <h2>ESP32-WROOM-32</h2>
    <form>
    <button name="OLED" type="submit" value="FILL_0">NASTAV 0</button>
    <button name="OLED" type="submit" value="FILL_1">NASTAV 1</button>
    <br><br>
    </form>
    <form ENCTYPE="text/plain">
    1. Řádek: <input type="text" name="ROW1">
    <input type="submit" value="Odeslat">
    </form>
    <form>
    2. Řádek: <input type="text" name="ROW2">
    <input type="submit" value="Odeslat">
    </form>
</body>
</html>
"""
```

Zpracování webového formuláře

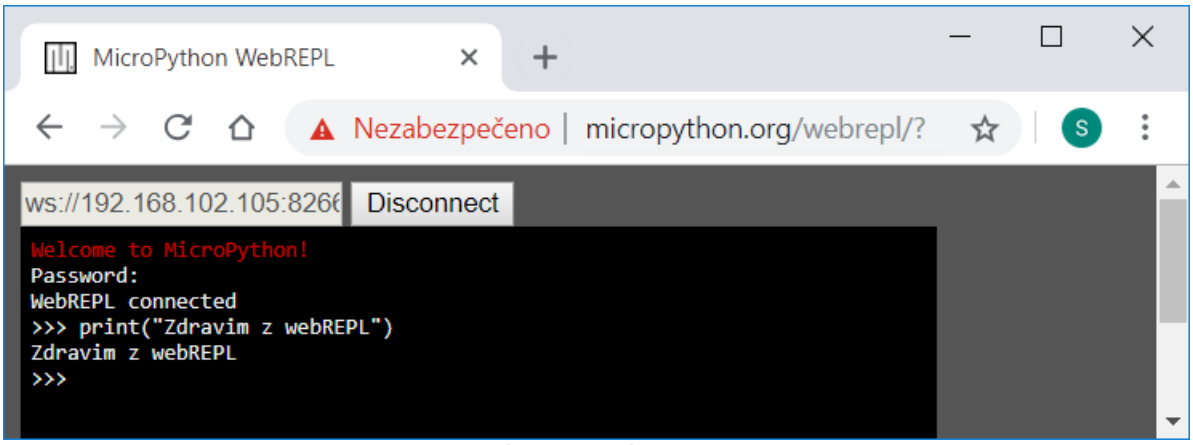
```
while True:
    conn, addr = s.accept()
    print("Got a connection from %s" % str(addr))
    request = conn.recv(1024)
    print("Got recv: %s" % request)
    request = str(request)
    print("Content = %s" % request)
    OLED_FILL_0 = request.find('/?OLED=FILL_0')
    OLED_FILL_1 = request.find('/?OLED=FILL_1')
    ROW1 = request.find('/?ROW1')
    ROW2 = request.find('/?ROW2')
    print("BUTTON1:",OLED_FILL_0)
    print("BUTTON2:",OLED_FILL_1)
    print("TEXT1  :",ROW1)
    print("TEXT2  :",ROW2)
    ROW1_TEXT = request[request.find('ROW1=') +
len('ROW1='):request.rfind('HTTP/')]
    ROW2_TEXT = request[request.find('ROW2=') +
len('ROW2='):request.rfind('HTTP/')]
    if OLED_FILL_0 == 6:
        oled.fill(0)
        oled.show()
    if OLED_FILL_1 == 6:
        oled.fill(1)
        oled.show()
    if ROW1 != -1:
        oled.text(ROW1_TEXT, 6, 12)
        oled.show()
    if ROW2 != -1:
        oled.text(ROW2_TEXT, 6, 22)
        oled.show()
    conn.send(html)
    conn.close()
```



Obr. 41 Vytvořené rozhraní

6.3.3 WebREPL

Nové verze programu MicroPythonu mají standardně funkci WebREPL vypnutou, proto se musí nejdříve povolit v klasickém REPL příkazem `import webrepl_setup`. Jakmile je webREPL povoleno a deska se připojí k síti, nebo vytvoří vlastní síť, tak se lze připojit pomocí nastaveného hesla a přistupovat k WebREPL přes webový prohlížeč. Samotné ovládání se provádí přes veřejně dostupnou webovou stránku http://micropython.org/webrepl, nebo je možné projekt WebREPL stáhnout na lokální úložiště, či nahrát na libovolné vlastní stránky. Je potřeba brát v potaz, že přístup k desce nemusí fungovat přes protokol https, ale pouze http. Na stránce s webREPL se připojuje k ip adrese, která byla zjištěna v první části této aplikace s portem 8266, přestože je v kódu nastavený port 80. Výsledná adresa má v tomhle případě tvar ws://192.168.102.105:8266/. [18]



Obr. 42 WebREPL

6.4 NUCLEO-F429I-DISCO s displejem ILI9341

Obr. 43 NUCLEO-F429I-DISCO s displejem ILI9341

6.4.1 Vykreslování na LCD displeji lLI9341

Pro usnadnění práce s displejem lze použít různé knihovny, které jsou k dispozici na webové službě github. Každé z těchto řešení je od jiného vývojáře, proto jsou napsané různými postupy a nabízí proto různé metody. V základu se vždy jedná o čtení a zápis dat přes rozhraní SPI, inicializaci displeje, zobrazení jednoduchých obrazců, výpis textu, nebo práci s obrázky. Za zmínku stojí řešení od autora s názvem ropod7 [19]. Jeho řešení využívá možnost MicroPythonu pracovat s návrhovým vzorem dekorátor. Ten umožňuje změny instancí bez nutnosti vytvoření nových odvozených tříd.

Dekorátor viper

Pro použití tohoto dekorátoru stačí před funkci přidat `@micropython.viper`. Tím se optimalizuje obalená funkce a je umožněno pracovat se speciálními datovými typy. Tyto datové typy jsou typické pro nízko úrovňové programování, jedná se o int (integer), uint (unsigned integer), ptr (pointer), ptr8 (pointer na byte), ptr16 a ptr32. [20]

Ukázka práce s mezipamětí

```
@micropython.viper
def foo(self, arg: int) -> int:
    buf = ptr8(self.linebuf)
    for x in range(20, 30):
        bar = buf[x]
```

Dekorátor asm_thumb

Opět se jednoduše před funkci napíše `@micropython.asm_thumb`. To umožní práci se strojními instrukcemi, které zvyšují výkon zejména pro celočíselné aritmetické a bitové manipulace. Nevýhodou je, že programátor musí znát základy jazyka symbolických adres (assembler), ale ve výsledku je taková funkce až několikanásobně rychlejší. [20]

Ukázka práce se strojními instrukcemi

```
@micropython.asm_thumb
def _asm_get_charpos(r0, r1, r2):
    mul(r0, r1)
    adc(r0, r2)
```

Ovládání LED přes rozhraní na displeji

Jelikož se jedná o rozmanitější příklad, je vhodné psát kód tak, aby byl rozdělený na jednotlivé části. V tomto případě se vytvoří několik objektů, kdy je každý objekt zodpovědný za jasně danou činnost. Ve výsledku se vytvoří 4 objekty:

ScreenController - vykresluje rozhraní na displeji

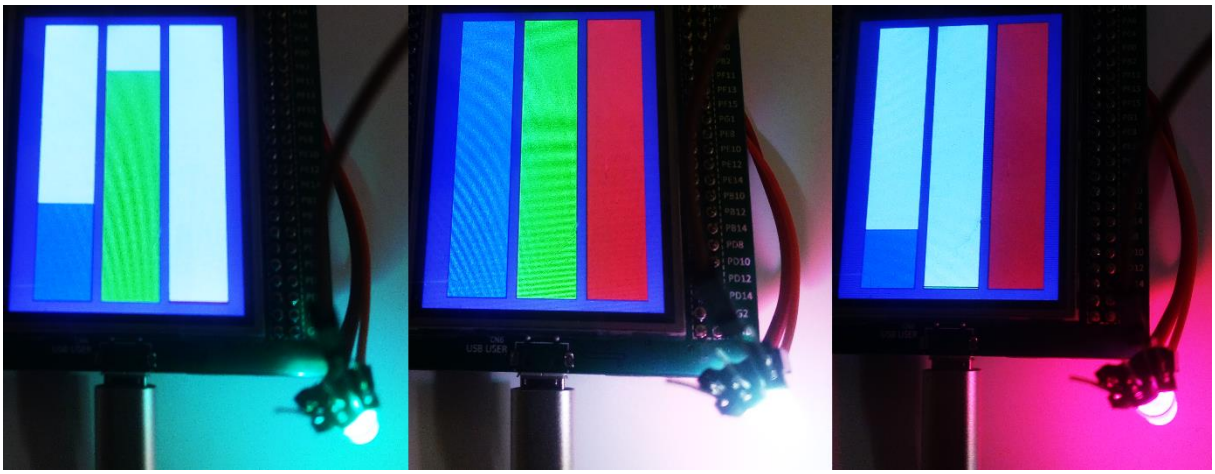
TouchpadController – detekuje a zpracovává stisk na touchpadu

LedController – obstarává časovače a tím řídí intenzitu jednotlivých barev

Časovače se zvolili podle tabulky alternativních funkcí v datasheetu: [21]

- časovač 2 s kanálem 1, přístupný na výstupu PA5
- časovač 3 s kanálem 1, přístupný na výstupu PA6
- časovač 14 s kanálem 1, přístupný na výstupu PA7

Main – obsahuje hlavní smyčku programu



Obr. 44 *Výsledné rozhraní pro ovládání tříbarevné LED*

ScreenController

```
class screenController:
    def __init__(self, display):
        self._display = display
    def drawMenu(self):
        self._display.fillMonocolor((19,19,19))
        self._display.drawRect(15,10,60,300,(0,0,0), infill = (255,0,0))
        self._display.drawRect(85,10,60,300,(0,0,0), infill = (0,255,0))
        self._display.drawRect(155,10,60,300,(0,0,0), infill = (0,0,255))
    def setColor(self, color, dutyCycle):
        dutyCycle = int(3 * dutyCycle)
        if (color == "R"):
            self._display.drawRect(15,10,60,300,(0,0,0), infill = (255,255,255))
            self._display.drawRect(15,10,60,dutyCycle,(0,0,0),infill = (255,0,0))
        if (color == "G"):
            self._display.drawRect(85,10,60,300,(0,0,0), infill = (255,255,255))
            self._display.drawRect(85,10,60,dutyCycle,(0,0,0),infill = (0,255,0))
        if (color == "B"):
            self._display.drawRect(155,10,60,300,(0,0,0), infill = (255,255,255))
            self._display.drawRect(155,10,60,dutyCycle,(0,0,0), infill=(0,0,255))
```

LedController

```
from machine import Pin
from pyb import Timer

class ledController:
    def __init__(self):
        self.TimerR = Timer(2, freq=1000)
        self.TimerG = Timer(3, freq=1000)
        self.TimerB = Timer(14, freq=1000)
        self.RChannel = self.TimerR.channel(1, Timer.PWM, pin=Pin('PA5'))
        self.GChannel = self.TimerG.channel(1, Timer.PWM, pin=Pin('PA6'))
        self.BChannel = self.TimerB.channel(1, Timer.PWM, pin=Pin('PA7'))
    def setColor(self, color, value):
        value = int(value)
        if (color == "R"):
            self.RChannel.pulse_width_percent(value)
        if (color == "G"):
            self.GChannel.pulse_width_percent(value)
        if (color == "B"):
            self.BChannel.pulse_width_percent(value)
```

TouchpadController

```
from binascii import hexlify

class touchpadController:
    def __init__(self, touchpad):
        self._touchpad = touchpad
        self._touchpad.writeto_mem(65, 0x04, b'\x00')
        self._touchpad.writeto_mem(65, 0x40, b'\x03')
    def updateValues(self, repeatingMeasurements):
        self.Z = self._touchpad.readfrom_mem(65, 0x40, 1)
        for x in range(repeatingMeasurements):
            self.X = self._touchpad.readfrom_mem(65, 0x4D, 2)
            self.Y = self._touchpad.readfrom_mem(65, 0x4F, 2)
    def getValueX(self):
        return(self._convertToInt(self.X))
    def getValueY(self):
        return(self._convertForDisplay(self._convertToInt(self.Y)))
    def getValueZ(self):
        return(self._convertToInt(self.Z) >> 7)
    def _convertForDisplay(self, y):
        if (y < 500):
            y = 0
        elif (y > 3300):
            y = 100
        else:
            y = int(y-500)
            y = int(y/30)
        y = 100-y
        return(y)
    def _convertToInt(self, var):
        var = hexlify(var)
        var = var.decode()
        var = int(var, 16)
        return(var)
```

Main

```
from lcd import LCD
from machine import I2C, Pin
from ScreenController import screenController
from TouchpadController import touchpadController
from LedController import ledController

led = ledController()
DISP = LCD()
screen = screenController(DISP)
screen.drawMenu()
TCHP = I2C(-1, scl=Pin("PA8"), sda=Pin("PC9"), freq=400000)
touchpad = touchpadController(TCHP)
for x in range(10000)
    touchpad.updateValues(200)
    x = touchpad.getValueX()
    y = touchpad.getValueY()
    z = touchpad.getValueZ()
    if (z == 1):
        if (x > 2450):
            led.setColor("B", y)
            screen.setColor("B", y)
        elif (x > 1400):
            led.setColor("G", y)
            screen.setColor("G", y)
        else:
            led.setColor("R", y)
            screen.setColor("R", y)
```

Závěr

V bakalářské práci byly podle zadání zhotoveny ukázky příkladů pro kity s procesory ARM řady STM32. Nejdříve byla vysvětlená základní práce s knihovnami, které jsou nezbytnou součástí kteréhokoliv z programů. Dále byl vysvětlen základní postup, jak zprovoznit firmware MicroPython na libovolné podporované desce. Potom byly vysvětleny možnosti, jak přistupovat k ovládání desky přes různá rozhraní. Významnou součástí práce bylo popsání sériové komunikace. Přesněji UART, přes který se ovládá prostředí REPL. SPI, kterým se ovládal posuvný registr STP08CP05 s LED a barevný displej ILI9341. Nakonec I2C, který se použilo pro čtení a zápis registrů 3D gyroskopu LSM6DSL a ovládání touchpadu u displeje.

MicroPython je programovací jazyk se snadnou syntaxí, vhodný jak pro začátečníky, tak i pro profesionály v oboru. Díky tomu, že je MicroPython volně dostupný na webové službě GitHub pod licencí MIT, se může kdokoliv podílet na vývoji tohoto jazyka. Ve výsledku MicroPython umožňuje rychlý vývoj a urychluje zavádění produktů na trh. Díky stále rostoucí popularitě Pythonu je zaručena neustále se rozrůstající komunita lidí, která bude využívat MicroPython v celé řadě aplikací.

Seznam literatury a informačních zdrojů

[1] SOKOLOVSKY, Paul. *Differences to CPython* [online]. 2018 [cit. 2018-12-05]. Dostupné z: https://github.com/micropython/micropython/wiki/Differences

[2] *MicroPython libraries* [online]. 2018 [cit. 2018-12-05]. Dostupné z: https://docs.micropython.org/en/latest/library/index.html

[3] *UM1724 User manual* [online]. 2014 [cit. 2019-06-01]. Dostupné z: https://files.amperka.ru/datasheets/nucleo-usermanual.pdf

[4] *Getting a MicroPython REPL prompt* [online]. 2018 [cit. 2018-12-05]. Dostupné z: https://docs.micropython.org/en/latest/esp8266/tutorial/repl.html

[5] JUNG, Ervin. *MicroPython IDE for ESP8266* [online]. 2018 [cit. 2018-12-05]. Dostupné z: https://github.com/jungervin/EsPy

[6] *UPyCraft* [online]. 2018 [cit. 2018-12-05]. Dostupné z: https://github.com/DFRobot/uPyCraft

[7] NOVIELLO, Carmine. *Running micropyton on STM32Nucleo-F4* [online]. 2015 [cit. 2018-12-05]. Dostupné z: https://www.carminenoviello.com/2015/06/03/running-micropyton-stm32nucleo-f4/

[8] *The internal filesystem* [online]. 2017 [cit. 2018-12-05]. Dostupné z: http://docs.micropython.org/en/v1.9.3/esp8266/esp8266/tutorial/filesystem.html

[9] *Running your first script* [online]. 2017 [cit. 2018-12-05]. Dostupné z: http://docs.micropython.org/en/v1.9.3/pyboard/pyboard/tutorial/script.html

[10] *Serial Communication* [online]. 2018 [cit. 2018-12-05]. Dostupné z: https://learn.sparkfun.com/tutorials/serial-communication

[11] *Serial Peripheral Interface (SPI)* [online]. 2018 [cit. 2018-12-05]. Dostupné z: https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi

[12] *I2C* [online]. 2018 [cit. 2018-12-05]. Dostupné z: https://learn.sparkfun.com/tutorials/i2c

[13] DUDÁČEK, Karel. *Sériová rozhraní SPI, Microwire, I2C a CAN* [online]. 2002 [cit. 2018-12-05]. Dostupné z: http://home.zcu.cz/~dudacek/NMS/Seriova_rozhrani.pdf

[14] *RM0383 Reference manual* [online]. 2018 [cit. 2018-12-05]. Dostupné z: https://www.st.com/resource/ja/reference_manual/dm00119316.pdf

[15] *STP08CP05 Datasheet* [online]. 2018 [cit. 2019-06-01]. Dostupné z: https://www.st.com/resource/en/datasheet/stp08cp05.pdf

[16] *UM2121 User manual* [online]. 2017 [cit. 2019-06-01]. Dostupné z: https://www.st.com/resource/en/user_manual/dm00333132.pdf

[17] *LSM6DSL Datasheet* [online]. 2017 [cit. 2019-06-08]. Dostupné z: https://www.st.com/resource/en/datasheet/lsm6dsl.pdf

[18] *MicroPython (na ESP8266/NodeMCU)* [online]. 2017 [cit. 2018-12-05]. Dostupné z: https://naucse.python.cz/course/mi-pyt/intro/micropython/

[19] PODGAISKI, Roman. *Pyboard drive* [online]. 2016 [cit. 2019-05-28]. Dostupné z: https://github.com/ropod7/pyboard_drive/tree/master/ILI9341

[20] *Maximising MicroPython Speed* [online]. 2017 [cit. 2019-05-28]. Dostupné z: http://docs.micropython.org/en/v1.9.3/pyboard/reference/speed_python.html

[21] *STM32F429xx Datasheet* [online]. 2018 [cit. 2019-05-28]. Dostupné z: https://www.st.com/resource/en/datasheet/dm00071990.pdf