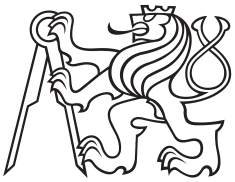
Diplomová práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra měření

Logický analyzátor s Raspberry Pi Pico

Vít Vaněček

Vedoucí: doc. Ing. Jan Fischer, CSc.
Obor: Kybernetika a robotika
Květen 2022

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Vaněček** Jméno: **Vít** Osobní číslo: **474475**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra měření**

Studijní program: **Kybernetika a robotika**

Studijní obor: **Kybernetika a robotika**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Logický analyzátor s Raspberry Pi Pico

Název diplomové práce anglicky:

Logic analyser based on Raspberry Pi Pico

Pokyny pro vypracování:

Navrhněte a realizujte logický analyzátor s modulem Raspberry Pi Pico, který pro ovládání a zobrazení výsledků bude využívat PC aplikace typu PulseView a Data Plotter. Pro verzi přístroje s DataPlotter vyřešte mimo zobrazení logických též záznam analogových signálů a jejich zobrazení. Realizujte funkci impulsního, případně též i funkčního generátoru s využitím PWM kanálů. Posuďte také možnost realizace funkce čítače s Raspberry PI Pico.
Na základě svých zkušeností získaných při vývoji tohoto přístroje vytvořte dokument popisující způsob práce s modulem Raspberry PI Pico, jeho programování i možnosti využití ve výuce programování na školách.

Seznam doporučené literatury:

Yiu, J.: The Definitive Guide to ARM Cortex -M0 and Cortex-M0+ processors
RP2040 Datasheet; Raspberry PI Trading Ltd., 2020
Raspberry PI Pico Datasheet; Raspberry PI Trading Ltd., 2020

Jméno a pracoviště vedoucí(ho) diplomové práce:

doc. Ing. Jan Fischer, CSc. katedra měření FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **08.02.2022** Termín odevzdání diplomové práce: **20.05.2022**

Platnost zadání diplomové práce: **30.09.2023**

doc. Ing. Jan Fischer, CSc. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. | Datum převzetí zadání

Podpis studenta

Poděkování

Tímto bych rád poděkoval vedoucímu práce doc. Ing. Janu Fischerovi, CSc. za ochotu a pomoc při tvorbě práce. Dále děkuji rodině a přátelům za podporu při studiích.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 20. května 2022

Abstrakt

Práce se zabývá vytvořením měřicích přístrojů s vývojovou deskou Raspberry Pi Pico. Vytvořené měřicí přístroje zahrnují logický analyzátor a osciloskop. Přístroje využívají PC aplikace pro vizualizaci naměřených dat a ovládání zařízení. Logický analyzátor využívá upravenou aplikaci PulseView a osciloskop využívá aplikaci Data Plotter. Komunikace mezi aplikacemi a Pico je zařízena pomocí USB sběrnice. Pro osciloskop byl také navržen impulsní a funkční generátor PWM. Práce obsahuje poznatky z programování Pico a prověřuje možnost vytvoření čítače. Na závěr byl vytvořen návod, jak začít programovat Pico pomocí Python, grafického programování a s pomocí C/C++.

Klíčová slova: Raspberry Pi Pico, osciloskop, logický analyzátor, mikrořadič, měřící přístroje

Vedoucí: doc. Ing. Jan Fischer, CSc.

Abstract

This thesis focuses on designing measuring devices with a Raspberry Pi Pico development board. Measuring devices include oscilloscope and logic analyzer. Devices use PC applications to visualize measured data and control the device. Logic Analyzer uses modified application PulseView, and oscilloscope uses Data Plotter. Pico uses USB bus for communication with PC applications. A configurable PWM generator has been developed as part of the oscilloscope. The thesis also contains some experiments and knowledge gained during the programming of Pico. The thesis ends with a manual on how to start programming Pico using Python, graphical programming, and C/C++.

Keywords: Raspberry Pi Pico, oscilloscope, logic analyzer, microcontroller, measuring devices

Title translation: Logic analyser based on Raspberry Pi Pico

Obsah

Obrázky

Tabulky

Výpisy

Kapitola 1
Úvod

Moderní mikrořadiče jsou vybaveny rychlými procesory a řadou periferií, které umožňují vytvoření levných měřicích přístrojů. Tyto přístroje mohou sloužit jako levná alternativa laboratorních měřicích přístrojů, které lze použít i ve výuce. Nízká cena a dostupnost mikrořadičů umožňuje zapůjčení studentům na domácí práce i koupi levného přístroje pro osobní použití. Jedním z mikrořadičů vhodným pro vytvoření takového přístroje je RP2040 v modulu Raspberry Pi Pico.

Pico je nově vydaná vývojová deska od firmy Raspberry Pi, která je založena na mikrořadiči RP2040 vyrobeným stejnou firmou. Nabízí dobré parametry za rozumnou cenu a umožňuje programování pomocí jazyka Python, který je vhodný pro začátečníky. Pico konkuruje vývojovým deskám, jako je Nucleo, které se již využívají v rámci výuky na ČVUT FEL. Díky dobrým parametrům a přístupnosti začátečníkům se Pico ukazuje jako vhodná deska, se kterou lze rozšířit arzenál měřicích přístrojů s mikrořadiči používaných ve výuce.

Cílem práce je tedy vytvořit osciloskop a logický analyzátor s Pico. Vzhledem k tomu, že se jedná o nový mikrořadič, je také motivace zaznamenat poznatky z programování. Tyto poznatky se mohou hodit dalším studentům, kteří v budoucnu budou vytvářet projekty s Pico. Dále může být vhodné vytvořit krátký návod na programování Pico pomocí jazyků Python a C/C++.

Kapitola 2

Rozbor zadání

Cílem této práce je vytvořit základní přístroje pro záznam a vizualizaci signálů, jako je logický analyzátor a osciloskop, s pomocí desky Raspberry Pi Pico (Dále jen Pico). Pico je nově vydaná vývojová deska od firmy Raspberry Pi, která je založena na mikrořadiči RP2040 vyrobeným stejnou firmou. Nabízí dobré parametry za rozumnou cenu a konkuruje vývojovým deskám, jako je Nucleo s mikrořadiči STM32 a Arduino s mikrořadiči Atmel. Díky tomu se Pico ukazuje jako vhodným kandidátem pro vytvoření levných měřicích přístrojů pro studenty a kutily. Takovéto přístroje lze využít při tvorbě elektrotechnických obvodů pro zjištění, zda obvod funguje nebo také pro názornou ukázku studentům, jak obvod pracuje.

V rámci studia na fakultě elektrotechnické se student setká s projekty vyžadující zapojení elektronických obvodů, práci se senzory a mikrořadiči. Díky možnosti použití mikrořadiče jako měřicího přístroje, může student pozorovat chování obvodu nebo komunikaci s digitálními senzory a zjistit, jak obvod funguje a případně najít chyby v projektu. Pokud samotný projekt vyžaduje použití mikrořadiče, může student dostat jeden navíc, vyhrazený jako měřicí přístroj, nebo může jiný student půjčit svůj mikrořadič pro účely měření. Výhodou je také možnost použití doma, kde student většinou nemá přístup k laboratorním přístrojům.

Měřicí přístroje, jako jsou ty, které budou vytvořeny v této práci, se již na fakultě elektronické využívají. Jsou založeny na vývojových deskách od STM32 jako je NucleoF303RE, ale tyto desky nejsou mezi začátečníky a kutily tak známé. Pico lze programovat pomocí jednoduchého jazyku Micropython a pomocí grafického programování, což z něj dělá dobrou desku i pro začátečníky. Je tedy velká šance, že lidé, kteří se začínají učit s elektronikou, budou vlastnit Pico a díky přístrojům vytvořeným v této práci si z něj budou moci udělat osciloskop či logický analyzátor.

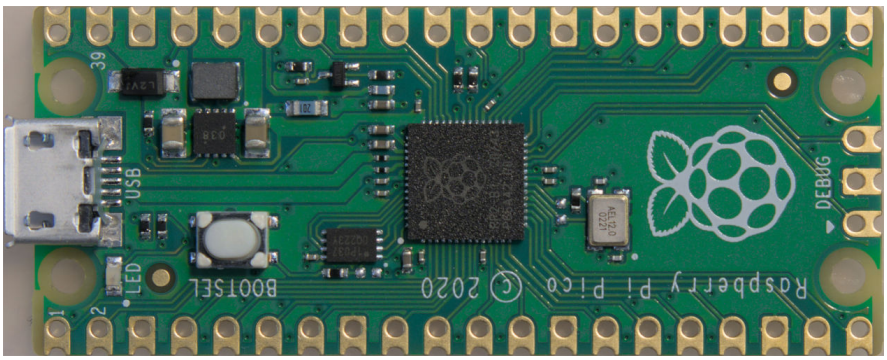
Pico samotné nemá možnost ovládání a zobrazení zaznamenaných signálů. Přístroje mají pro tyto účely využívat PC aplikace Pulseview a Dataplotter. To jsou univerzální aplikace pro ovládání zobrazení dat z měřicích přístrojů, které jsou otevřené a volně ke stažení. Pro použití Pico jako měřicího přístroje potom bude stačit pouze nahrát na desku firmware, stáhnout si program pro vizualizaci v počítači a připojit desku k počítači pomocí USB kabelu. Nahrání firmwaru na Pico nevyžaduje žádnou externí elektroniku a lze to provést pomocí počítače a USB kabelu. Přístroje s Pico budou tak samostatné a nebudou vyžadovat žádnou externí elektroniku.

Pro Pico prozatím existuje malé množství návodů a projektů, které ji využívají. Dalším cílem práce je proto sepsání poznatků, jakým způsobem lze Pico programovat. Tyto poznatky budou sloužit pro další studenty, kteří s deskou budou pracovat, nebo vytvářet další měřicí přístroje.

Kapitola 3

Představení hardwaru Raspberry Pi Pico

V této kapitole bude představeno Pico, jeho hardwarové parametry a schopnosti. Na obr. 3.1 je možné vidět samotnou desku a mikrořadič RP2040 uprostřed desky. RP2040 je hlavní část Pico a krom něho jsou na desce obsaženy pouze krystaly pro generování hodinového signálu, regulátory pro napájení mikrořadiče, USB konektor apod. Tato kapitola bude zaměřena na samotný mikrořadič a hardwarové periferie uvnitř mikrořadiče.



Obr 3.1: Vývojová deska Raspberry Pi Pico

3.1 Procesory v RP2040

Procesor je nedílnou součástí počítačů a mikrořadičů. RP2040 obsahuje dva procesory ARM Cortex-M0+ s maximální frekvencí 133MHz. Programování mikrořadiče se dvěma procesory může být složitý úkol a mikrořadič proto obsahuje hardwarové prvky, pomocí kterých lze implementovat komunikaci mezi procesory a také mutexy, pomocí kterých lze zamezit situaci, kdy oba procesory přistupují ke stejné paměti. Druhé jádro bude využito pro implementaci Osciloskopu a lze si tedy o konkrétním použití druhého jádra přečíst v kapitole 5.6

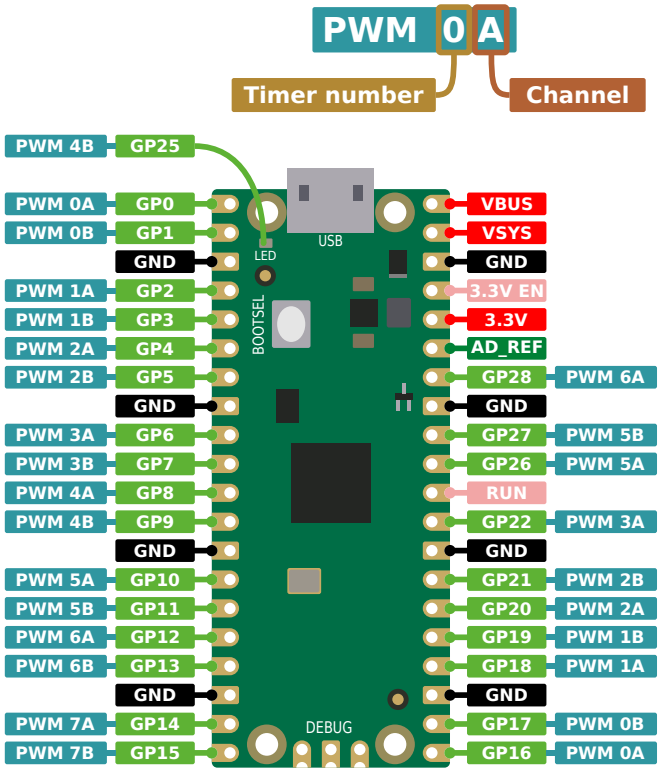
3.2 Paměťi v RP2040

Součástí RP2040 je operační paměť SRAM, která má velikost 264 kB. Paměť FLASH v mikrořadiči obsažena není a musí se napojit externě. FLASH obsažená na Pico má velikost 2 MB, ale samotný mikrořadič podporuje velikosti až 16 MB. S touto FLASH pamětí komunikuje RP2040 pomocí rozhraní QSPI. Pro čtení dat z FLASH obsahuje mikrořadič systém execute-in-place (XIP) s pamětí o velikosti 16 kB. Do této paměti se ukládají nedávno přečtená data z FLASH a urychluje tak opakované čtení. [1]

[1] Při programování v C/C++ s PicoSDK je také možné nastavit zkopírování obsahu FLASH do SRAM na začátku programu a nemusí se tak číst instrukce z externí FLASH. [2, s. 288]

3.3 Čítače pro PWM

Pulsně šířková modulace (PWM) je jedním ze základních signálů, které se v mikrořadičích využívají. PWM lze použít pro regulaci jasu LED, regulaci příkonu topného tělíska, ovládání servo motorů a mnoho dalšího. RP2040 obsahuje 7 čítačů, vyhrazených pro generování PWM, které mají každý 2 výstupní kanály. Frekvenci generovaného PWM signálu určuje maximální hodnota PWM čítače a střídu určuje hodnota v registru výstupního kanálu. Toto tedy znamená, že GPIO piny připojené ke stejnému PWM čítači mají vždy stejně nastavenou frekvenci a piny připojené ke stejnému výstupnímu kanálu mají stejně nastavenou střídu. V dokumentaci je příslušnost pinů ke kanálům PWM čítačů vypsána pomocí tabulky [6, s. 543]. Pro lepší znázornění příslušnosti pinů byla vytvořena vizualizace na obr. 3.2.



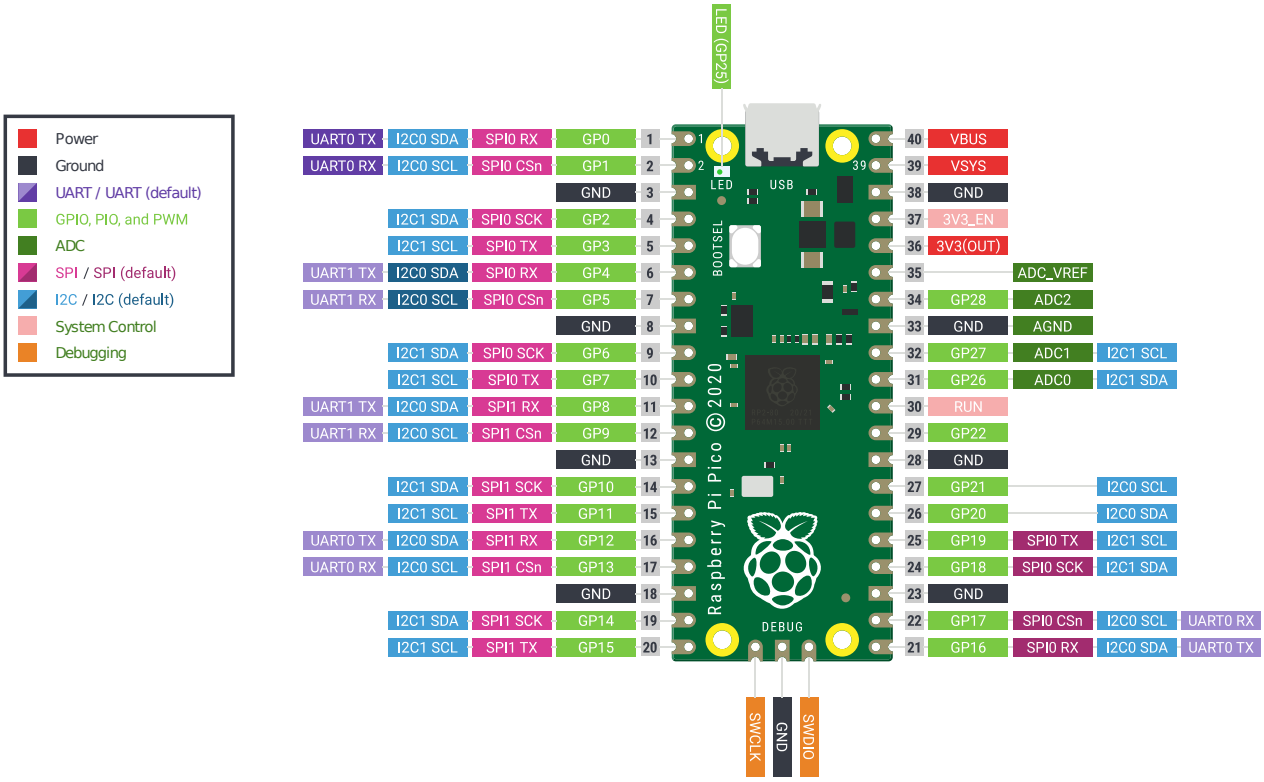
Obr 3.2: Vizualizace příslušnosti pinů k PWM čítačům na Pico

Krom generování PWM lze kanál B nastavit také jako vstupní. Vstupní kanál lze nastavit do 3 režimů, kterými jsou:

- čítač běží, pokud je vstup kanálu B ve stavu High,
- čítač jednou přičte, pokud na vstupu kanálu B nastane náběžná hrana,
- čítač jednou přičte, pokud na vstupu kanálu B nastane sestupná hrana.

3.4 Digitální komunikace s RP2040

Pro komunikaci s digitálními senzory, jinými mikrořadiči a externími obvody se často používají digitální komunikace typu UART, I2C a SPI. Pro každou z těchto komunikací obsahuje RP2040 dvě periferie, které lze přiřadit k různým pinům. Možnosti přiřazení k pinům je znázorněna na obr. 3.3.



Obr 3.3: Raspberry Pi Pico Pinout [4]

3.5 Převodník AD v RP2040

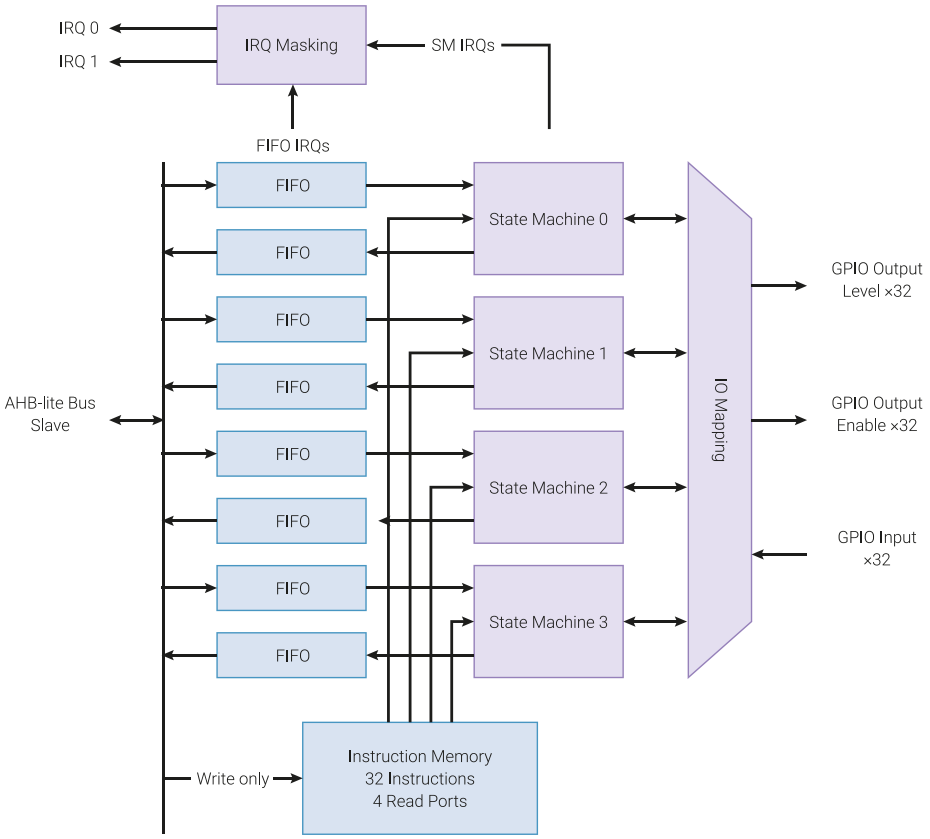
Senzory jako teplotní senzory a potenciometry poskytují informaci ve formě napětí, které je potřeba změřit. Podobně pro vytvoření osciloskopu je vhodné měřit hodnotu napětí a tu převést na digitální hodnotu. Pro účely čtení napětí slouží analogově-digitální převodní (ADC). RP2040 obsahuje jeden 12bitový ADC s 5 vstupy. Z těchto vstupů jsou 4 připojeny k digitálním pinům (Na Pico dostupné pouze 3) a jeden je připojen k internímu teplotnímu senzoru. Při práci s ADC lze nastavit, ze kterého vstupu bude přivedeno napětí pro převod, nebo lze také nastavit automatické přepínání mezi vstupy. To se hodí, pokud je potřeba digitalizovat napětí z více vstupů najednou. Maximální dosažitelná frekvence digitalizace je 500kHz a tuto frekvenci lze snížit pomocí interního čítače. Vstupy, ze kterých lze digitalizovat napětí, jsou znázorněny na obr. 3.3.

3.6 Řadiče DMA v RP2040

Direct Memory Access (DMA) slouží k rychlému přenášení dat nezávisle na procesoru. Toto je užitečné například při čtení analogových hodnot z ADC. Díky použití DMA lze přenášet naměřené hodnoty z ADC do paměti bez zatížení procesoru, který může mezi tím pracovat na jiných věcech. RP2040 obsahuje 12 DMA kanálů, které lze nezávisle použít. U DMA kanálu lze nastavit požadavky na přenos od ostatních periferií nebo také neustálý pravidelný přenos. Tímto lze například zařídit, aby DMA přenos nastal pouze pokud je v ADC připravena hodnota nebo pokud UART přijal nová data. Krom toho je poskytnuta možnost zřetězení kanálů, kdy se po dokončení přenosu jednoho DMA kanálu automaticky spustí jiný kanál.

3.7 Bloky PIO v RP2040

Programable IO (PIO) si lze představit jako další malé procesory, které jsou určeny pro naprogramování vlastních periferií. Lze si pomocí něj naprogramovat komunikace jako UART apod., ale také OneWire nebo komunikaci s digitální LED WS2812B. Tyto komunikace vyžadují přesné časování a díky PIO není nutné je řešit hlavním procesorem, čímž se uvolní výkon pro jiné procesy. RP2040 obsahuje 2 PIO bloky a každý z těchto bloků obsahuje 4 stavové automaty. Stavové automaty se programují pomocí jazyka PIO Assembler, který zahrnuje 9 instrukcí. Komunikace mezi procesorem a stavovým automatem probíhá pomocí FIFO registrů, navíc lze pomocí stavového automatu vyvolat přerušení procesoru. Diagram PIO bloku je znázorněn na obr. 3.4 a ukázka použití PIO je v kapitole 6.3.



Obr 3.4: Diagram PIO bloku [6, s. 330]

Kapitola 4

Experimenty s Raspberry Pi Pico a poznatky z programování

Při vytváření této práce bylo zjištěno několik poznatků, které budou užitečné pro vytvoření osciloskopu a logického analyzátoru s Pico. V této kapitole budou popsány některé tyto poznatky při programování Pico pomocí C++ s PicoSDK a bude na ně odkazováno z kapitol, kde bude popsána implementace osciloskopu a logického analyzátoru. Tyto poznatky mohou být užitečné i pro další studenty, kteří budou s Pico vytvářet přístroje a projekty a je jim proto vyhrazena vlastní kapitola.

4.1 Přenos binárních dat pomocí USB

Komunikace mezi počítačem a mikrořadičem běžně probíhá pomocí USB a UART-USB převodníku. Mikrořadič v tomto případě posílá data pomocí UART a převodník tyto data přijme a přepošle je do počítače pomocí USB. Moderní mikrořadiče mají integrovanou USB periferii, díky které lze data posílat pomocí USB přímo z mikrořadiče, bez nutnosti převodníku, díky čemuž může být komunikace spolehlivější a rychlejší. I RP2040 obsahuje tuto periferii a bude použita pro implementaci měřicích přístrojů v této práci.

Při programování v C/C++ s pomocí PicoSDK lze pro odesílání dat prostřednictvím USB využít funkce ze standardní knihovny `#include <stdio.h>`. Těmi jsou funkce `printf()`, `putchar()` a `puts()`. Příklad těchto funkcí je znázorněn ve výpisu 4.1. Po nahrání na Pico a připojení přes sériový terminál dostane terminál zprávy jako ve výpisu 4.2.

```
#include "pico/stdlib.h"
#include <stdio.h>

int main() {
    stdio_usb_init();

    while (1) {
        if (stdio_usb_connected()) {
            printf("Test number: %03d\n", 25);
            puts("Test string");
            putchar('F');
            putchar('\n');
        }
        sleep_ms(1000);
    }
}
```

Výpis 4.1: Odesílání zpráv prostřednictvím USB

```
Test number: 025
Test string
F
```

Výpis 4.2: Zprávy přijaté terminálem v počítači

Tyto funkce jsou vyhovující pro posílání textových zpráv. Pro přenášení binárních dat, jako je digitalizovaný signál, jsou ovšem nevhodné, protože provádí převod zakončení řádku `'\n'` na `'\r\n'`. To může způsobovat problémy při odeslání vzorku s hodnotou 10 nebo 13, které odpovídají `'\n'` a `'\r'`. Pro tyto účely jsou součástí PicoSDK funkce `putchar_raw()` a `puts_raw()`, které převod zakončení řádku neprovádí. Další možností je vypnout převod zakončení řádků pro všechny funkce. To se provede nastavením v CMakeLists.txt, což je ukázáno ve výpisu 4.3.

```
add_executable(rpi_pico_usb
    src/main.cpp
)
# Vypne prevod na crlf
target_compile_definitions(rpi_pico_usb PRIVATE PICO_STDIO_ENABLE_CRLF_SUPPORT=0)
```

Výpis 4.3: Vypnutí převodu zakončení řádku

V případě měřicích přístrojů bude potřeba odesílat prostřednictvím USB pole naměřených hodnot v binární podobě. Způsob, jakým toto je možné udělat s PicoSDK je znázorněn ve výpisu 4.4.

```
inline void send_measured_data(uint8_t *buff, size_t buff_size) {
    for (size_t i{0}; i < buff_size; i++) {
        putchar_raw(buff[i]);
    }
}
```

Výpis 4.4: Odeslání naměřených hodnot pomocí `puthcar_raw()`

V tomto případě se ovšem musí odeslat každá hodnota zvlášť, což muže být pomalé. Byla proto snaha nalézt v PicoSDK funkce, které odešlou data najednou. Existuje funkce `_write()`, jejíž použití je znázorněno ve výpisu 4.5. [1] Tato funkce ovšem provádí převod zakončení řádku a při bližším průzkumu byla nalezena další možnost.

```
extern "C" int _write(int handle, char *buffer, int length);

inline void send_data(uint8_t *buff, size_t buff_size) {
    _write(1, (char *)buff, buff_size);
}
```

Výpis 4.5: Odeslání naměřených hodnot pomocí `_write()`

V PicoSDK jsou nejzákladnější funkce pro odeslání a příjem přes USB uloženy do struktury `stdio_driver_t stdio_usb`, jejíž použití je znázorněno ve výpisu 4.6.

[1] `extern "C"` je v tomto případě potřeba pouze pokud se funkce `_write()` používá v C++ kódu

```
#include "pico/stdio/driver.h"
#include <stdio.h>
#include "pico/stdlib.h"

inline void send_data(uint8_t *buff, size_t buff_size) {
    stdio_usb.out_chars((char *)buff, buff_size);
}
```

Výpis 4.6: Odeslání naměřených hodnot pomocí `stdio_usb.out_chars()`

Nutno upozornit, že je potřeba přidat include souboru `"pico/stdio/driver.h"`. Tato funkce neprovádí žádné převody a měla by tedy být pro odesílání nejrychlejší. Toto je ovšem nutné prověřit experimentem.

4.2 Měření rychlosti funkcí pro odeslání dat pomocí USB

Pro měřicí přístroje s Pico bude potřeba ukládat digitalizované vzorky do datového pole, které je potřeba odeslat do PC aplikací pomocí USB. V předchozí kapitole 4.1 bylo zjištěno, jaké funkce je možné použít pro odeslání takového datového pole. Nyní bude vyzkoušeno, která z těchto funkcí je nejrychlejší pro odeslání velkého množství dat.

4.2.1 Metoda měření rychlosti přenosu dat

Měření bude probíhat v počítači pomocí Python scriptu, který přijme 30000 bajtů od Pico, připojeného pomocí USB. Pole v Pico bude typu `uint8_t` a bude inicializováno postupně zvyšující se hodnotou od 0 do 254 a po překročení 254 se hodnota vynuluje a bude se zase zvyšovat. Hodnota 255 bude vyhrazena pro označení konce datového pole. Inicializace pole je znázorněna ve výpisu 4.7.

```
constexpr size_t data_array_size{30000};
uint8_t data_to_send[data_array_size];

int main() {
    uint8_t temp{0};
    for (uint8_t &byte : data_to_send) {
        byte = temp;
        temp = (temp + 1) % 0xFF;
    }
    data_to_send[data_array_size - 1] = 0xFF;

    stdio_usb_init();

    while (1) {
```

Výpis 4.7: Inicializace pole dat pro měření ryclosti přenosu

V nekonečné smyčce, znázorněné ve výpisu 4.8, bude Pico přijímat znaky a na ně příslušně odpovídat. Po přijetí znaku `'I'` odešle do počítače název funkce, kterou nyní používá pro odeslání pole dat. Po přijetí `'S'` odešle samotné pole dat pomocí funkce `send_data()`, jejíž implementace se bude měnit podle aktuálně použité funkce pro odeslání pomocí USB. Ukázka funkce `send_data()` je ve výpisu 4.9. Pico navíc změří uběhnutý čas při odesílání v μs a ten pošle po přijetí symbolu `'T'`. Samotná rychlost komunikace se bude měřit v Python scriptu a čas změřený v Pico slouží pouze pro kontrolu.

```
while(1) {
        if (stdio_usb_connected()) {
        char c = getchar();

        if (c == 'I') {
            printf("%s", test_name);
            putchar_raw(0xFF);
        } else if (c == 'S') {
            start_time = time_us_32();
            send_data(data_to_send, data_array_size);
            end_time = time_us_32();
        } else if (c == 'T') {
            printf("%u %u ", start_time, end_time);
            putchar_raw(0xFF);
        }
    }
}
```

Výpis 4.8: Nekonečná smyčka v Pico pro měření rychlosti odesílání

```
inline constexpr char test_name[]{"putchar_raw"};
inline void send_data(uint8_t *buff, size_t buff_size) {
    for (size_t i{0}; i < buff_size; i++) {
        putchar_raw(buff[i]);
    }
}
```

Výpis 4.9: Funkce pro odeslání datového pole

Python script využije knihovny `serial` pro komunikaci s Pico a knihovny `pandas` pro uložení naměřeného času do csv souboru. Python script vždy nejdřív odešle symbol `'I'` pro získání jména funkce, kterou Pico zrovna využívá, následně odešle `'S'`, přijme datové pole a zkontroluje, zda mají data správnou podobu. Nakonec skript získá naměřený čas z Pico pomocí `'T'` a nová data přidá do csv souboru. Část Python scriptu je znázorněna ve výpisu 4.10.

```
with serial.Serial('/dev/ttyACM0', dsrdtr=True) as SER:
    SER.write(b'I')
    function_name = SER.read_until(expected=b'\xFF')

    SER.write(b'S')
    start_time = time.time_ns()
    raw_data = SER.read_until(expected=b'\xFF')
    end_time = time.time_ns()

    SER.write(b'T')
    pico_time = SER.read_until(expected=b'\xFF')
```

Výpis 4.10: Část Python scriptu pro měření rychlosti USB

Měření proběhlo s notebookem Dell s operačním systémem Fedora 35. Pico je připojeno microUSB kabelem do interního USB portu, nikoliv přes USB Hub. S každou funkcí proběhlo měření 5× a výsledná hodnota byla zprůměrována. U funkcí, které dokáží odeslat více bajtů najednou, bylo také

otestováno, zda není výhodnější odesílat data po méně bajtech, než je velikost datového pole. Pro možnost otestování funkcí jako je putchar a printf, byl pomocí CMake také vypnut převod zakončení řádku.

4.2.2 Výsledky měření rychlosti přenosu

V tab. 4.1 jsou zaznamenány změřené rychlosti přenosu mezi Pico a PC. Rychlost odesílání byla spočítána pomocí rovnice 4.1. Vypočítané rychlostí byly zprůměrovány a následně vytvořen graf 4.1.

$$s_{tx} = \frac{30000}{t_{py}} \tag{4.1}$$

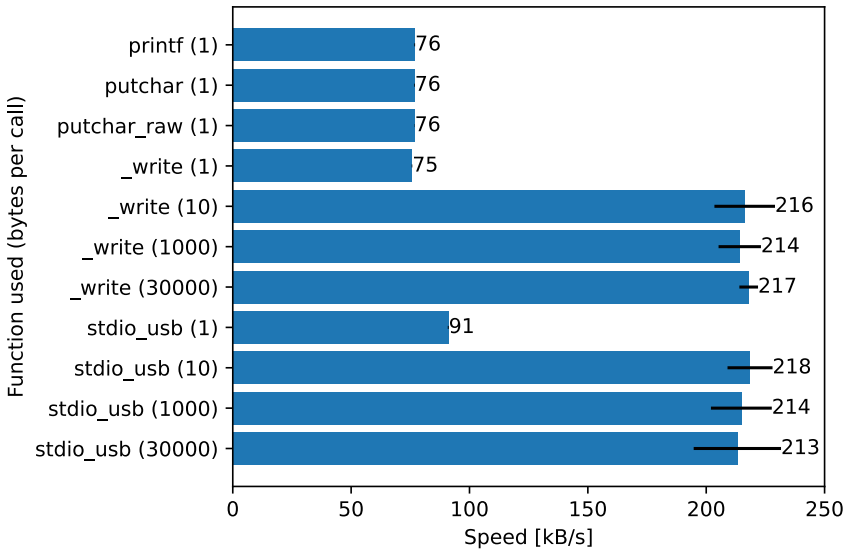
Použitá funkce	Bajtů odeslaných najednou	t_{py} [ns]	t_{pico} [ns]	s_{tx} [B/s]
printf	1	390046303	389523000	76913
printf	1	389487943	388791000	77024
printf	1	389986681	388984000	76925
printf	1	390113701	389364000	76900
printf	1	392025393	389496000	76525
putchar	1	391240090	389387000	76679
putchar	1	391960511	389400000	76538
putchar	1	391128861	389017000	76701
putchar	1	389912826	389149000	76940
putchar	1	389985771	389154000	76925
putchar_raw	1	390513175	389695000	76821
putchar_raw	1	389950779	388927000	76932
putchar_raw	1	390646604	388834000	76795
putchar_raw	1	390887808	389195000	76748
putchar_raw	1	391120348	389566000	76702
_write	1	396735490	393540000	75617
_write	1	396071144	393789000	75743
_write	1	397116392	393671000	75544
_write	1	395012343	394199000	75946
_write	1	396149614	393638000	75728
_write	10	132853916	107878000	225811
_write	10	144999361	139726000	206897
_write	10	152648191	147370000	196530
_write	10	129604485	108363000	231473
_write	10	135997226	111544000	220592
_write	1000	141365005	108334000	212216
_write	1000	150728154	141788000	199033
_write	1000	137465660	107990000	218236
_write	1000	132483067	112076000	226444
_write	1000	139602006	112537000	214896
_write	30000	133832061	108753000	224161
_write	30000	137261668	112520000	218560
_write	30000	138484967	131247000	216630
_write	30000	137305793	111871000	218490
_write	30000	141583458	135858000	211889
stdio_usb	1	330558718	327557000	90755

Pokračování na další straně

Tabulka 4.1: Naměřená data pro zjištění rychlosti přenosu USB z Pico do PC

Použitá funkce	Bajtů odeslaných najednou	t_{py} [ns]	t_{pico} [ns]	s_{tx} [B/s]
stdio_usb	1	329291431	327402000	91104
stdio_usb	1	329870946	327310000	90944
stdio_usb	1	329831455	327253000	90955
stdio_usb	1	328358006	327466000	91363
stdio_usb	10	140336478	109994000	213771
stdio_usb	10	145499497	114445000	206186
stdio_usb	10	138885988	112326000	216004
stdio_usb	10	127866685	107489000	234619
stdio_usb	10	135168772	110654000	221944
stdio_usb	1000	130788479	106884000	229378
stdio_usb	1000	146628635	140199000	204598
stdio_usb	1000	150605455	144387000	199195
stdio_usb	1000	130133789	107054000	230531
stdio_usb	1000	142495738	113310000	210532
stdio_usb	30000	142249149	113789000	210897
stdio_usb	30000	133744367	109877000	224308
stdio_usb	30000	132895238	110295000	225741
stdio_usb	30000	132325290	108461000	226714
stdio_usb	30000	168452741	162167000	178091

Tabulka 4.1: Naměřená data pro zjištění rychlosti přenosu USB z Pico do PC



Obr 4.1: Graf rychlosti přenosu z Pico do PC pomocí USB

4.2.3 Závěr měření rychlosti USB

Z naměřených dat je možné vidět, že pro odesílaní velkých datových polí jsou nejrychlejšími funkce `_write()` a `stdio\_usb()`. Dále, že není výhodnější odesílat data po méně bajtech najednou, než je velikost datového pole. Funkce `_write()` má oproti `stdio\_usb()` nevýhodu v tom, že vyžaduje vypnutí převodu zakončení řádku. Z toho bylo usouzeno, že pro realizaci osciloskopu a logického analyzátoru bude nejvýhodnější využít funkci `stdio\_usb()`.

4.3 Příznak DTR při použití USB

Knihovna PicoSDK využívá příznak Data Terminal Ready (DTR) pro zjištění, zda je pomocí USB připojena aplikace, která bude s Pico komunikovat. Příznak DTR aktivuje samotná aplikace a tím značí, že je připravena ke komunikaci a PicoSDK neodešle data přes USB, dokud příznak není aktivní. Uživatel má také možnost zjistit, zda je příznak aktivní pomocí funkce `stdio_usb_connected()`. Problém ovšem může nastat, pokud je požadována komunikace s aplikací, která příznak DTR neaktivuje.

Byla proto nalezena možnost, jak deaktivovat kontrolu příznaku DTR v PicoSDK. Deaktivace vyžaduje úpravu zdrojového souboru **pico-sdk/lib/tinyusb/src/class/cdc/cdc_device.h**. V tomto souboru lze upravit funkci `tud_cdc_connected()` na řádku 169, aby nekontrolovala příznak DTR a Pico ho tak uvidí jako neustále aktivní. Funkci lze upravit z původní implementace 4.11 na implementaci 4.12. Nutno ovšem upozornit, že funkce `stdio_usb_connected()` bude po úpravě vracet vždy hodnotu `true` a nelze lehce rozpoznat připojení k PC aplikaci. V této práci proto nebude úprava PicoSDK využita.

```
static inline bool tud_cdc_connected (void)
{
    return tud_cdc_n_connected(0);
}
```

Výpis 4.11: Funkce na kontrolu DTR v PicoSDK

```
static inline bool tud_cdc_connected (void)
{
    return tud_ready();
}
```

Výpis 4.12: Upravená funkce na kontrolu DTR v PicoSDK

4.4 Cyklický režim DMA

DMA je velice užitečná periferie pro rychlý přenos dat nezávisle na procesoru. V některých případech, jako je čtení hodnost z AD převodníku se může hodit cyklický režim. V tomto režimu DMA kanál nepřestane zapisovat hodnoty na konci datového pole, ale vrátí se na začátek pole a přepisuje staré hodnoty. DMA v RP2040 ovšem cyklický režim přímo nenabízí a byla proto snaha ho zařídit jiným způsobem. DMA kanál lze po dokončení přenosu požadovaného množství hodnot znovu spustit a množství hodnot se resetuje na předchozí hodnotu. Adresy pro zápis a čtení se ovšem opětovným spuštění neresetují. To není problém, pokud se adresa nemění, ale může být problémové, pokud je adresa inkrementována po každém zápisu, jako např. při zápisu do datového pole. Před opětovném spuštění je tedy potřeba znovu adresu datového pole nastavit.

RP2040 dává možnost zřetězení kanálů, kdy se po dokončení přenosu spustí jiný kanál automaticky. Krom toho je možnost spustit kanál zápisem do jednoho z registrů, pomocí kterého se DMA kanál nastavuje. Poté by ovšem nebylo možné nastavit kanál bez toho, aby se spustil. RP2040 má proto pro každý kanál 4 aliasy a v každém aliasu spouští DMA kanál pouze jeden trigger registr. Aliasy registrů DMA kanálu jsou znázorněny v tabulce 4.2. Pomocí trigger registrů a řetězení kanálů tedy můžeme zařídit cyklický režim pomocí 2 DMA kanálů. První kanál slouží pro samotný přenos a druhý pro opětovné spuštění prvního a resetovaní adresy, která se inkrementuje.

Offset	0x00	0x04	0x08	0x0C (Trigger)
0x00 (Alias 0)	READ_ADDR	WRITE_ADDR	TRANS_COUNT	CTRL_TRIG
0x10 (Alias 1)	CTRL	READ_ADDR	WRITE_ADDR	TRANS_COUNT_TRIG
0x20 (Alias 2)	CTRL	TRANS_COUNT	READ_ADDR	WRITE_ADDR_TRIG
0x30 (Alias 3)	CTRL	WRITE_ADDR	TRANS_COUNT	READ_ADDR_TRIG

Tabulka 4.2: Aliasy registrů DMA kanálu [6, s. 94]

Nastavení cyklického režimu pomocí 2 DMA kanálů je ukázáno ve výpisu 4.13. V tomto programu čte hlavní DMA kanál hodnotu ADC a ukládá ji do datového pole `test_array`. Druhý DMA kanál přesune adresu začátku datového pole do registru WRITE_ADDR_TRIG. Tím se tedy hlavní kanál opětovně spustí a adresa datového pole se resetuje. Nutno upozornit, že pro druhý DMA kanál je potřeba vytvořit novou proměnnou typu pointer, která obsahuje adresu na počátek pole. To je z důvodu, že druhý kanál požaduje adresu na místo v paměti, kde je uložena adresa datového pole a na to lze použít proměnnou typu `void *`.

```
#include <stdint.h>
#include "hardware/dma.h"

int main() {
    uint16_t write_array[10];
    uint16_t read_val{0};
    void *array_addr = &write_array[0];
    const uint main_chan = dma_claim_unused_channel(true);
    const uint ctrl_chan = dma_claim_unused_channel(true);

    /* Nastaveni hlavniho DMA kanalu */
    dma_channel_config chan_cfg = dma_channel_get_default_config(main_chan);
    channel_config_set_transfer_data_size(&chan_cfg, DMA_SIZE_16);
    channel_config_set_chain_to(&chan_cfg, ctrl_chan);
    channel_config_set_write_increment(&chan_cfg, true);
    channel_config_set_read_increment(&chan_cfg, false);
    dma_channel_configure(main_chan,
                          &chan_cfg,
                          &write_array[0],
                          &read_val,
                          10,
                          false);

    /* Nastaveni kanalu pro restart hlavniho */
    chan_cfg = dma_channel_get_default_config(ctrl_chan);
    channel_config_set_transfer_data_size(&chan_cfg, DMA_SIZE_32);
    channel_config_set_read_increment(&chan_cfg, false);
    channel_config_set_write_increment(&chan_cfg, false);
    dma_channel_configure(ctrl_chan,
                          &chan_cfg,
                          &dma_channel_hw_addr(main_chan)->al2_write_addr_trig,
                          &array_addr,
                          1,
                          false);
```

Výpis 4.13: Příklad cyklického režimu pomocí 2 DMA kanálů

Pro zastavení DMA v tomto příkladě lze použít funkci `dma_channel_abort(main_chan)`, což přeruší DMA okamžitě. Další možností je využít tzv. null trigger, kdy se do registru pro opětovné spuštění zapíše nula. Ve výpisu 4.13 by to znamenalo zapsat do proměnné `array_addr` nulu, místo adresy datového pole. Tím by se DMA ukončilo po posledním zápisu do datového pole.

4.5 Zapisování do registru 16 a 8bitově

Při experimentování s DMA bylo zjištěno, že zápis do registru 16 nebo 8bitově bude hodnotu opakovat tak, aby se zaplnilo celých 32 bitů registru. Toto lze ukázat na příkladě ve výpisu 4.14. Výstup tohoto programu je možné vidět ve výpisu 4.15.

```
const uint8_t value = 0xAB

volatile uint32_t *reg_32 = &pwm_hw->slice[0].cc;
*reg_32 = value;
printf("Reg 32: 0x%08X\n", pwm_hw->slice[0].cc);

volatile uint16_t *reg_16 = (io_rw_16 *)&pwm_hw->slice[0].cc;
*reg_16 = value;
printf("Reg 16: 0x%08X\n", pwm_hw->slice[0].cc);

volatile uint8_t *reg_8 = (io_rw_8 *)&pwm_hw->slice[0].cc;
*reg_8 = value;
printf("Reg 8: 0x%08X\n", pwm_hw->slice[0].cc);
```

Výpis 4.14: Zápis do registru 32 16 a 8bitově

```
Reg 32: 0x000000AB
Reg 16: 0x00AB00AB
Reg 8: 0xABABABAB
```

Výpis 4.15: Výsledek zápisů do registru

Příkladem kdy takové chování může být problémové je např. při generování signálu pomocí PWM na výstupním kanálu. Generování může probíhat čtením z datového pole cyklicky pomocí DMA a zapisování těchto hodnot do registru výstupního kanálů čítače (registr CC). V registru CC odpovídá spodních 16 bitů kanálu **A** a horních 16 bitů kanálu **B**. Datové pole, ze kterého bude DMA kanál číst by tedy mohlo být typu `uint16_t`. DMA v tomto případě bude nastaveno na 16bitové přenosy a zapsaná hodnota se v registru bude opakovat. Výsledkem tedy bude, že se hodnota nenastaví pouze pro kanál **A**, ale také pro kanál **B**.

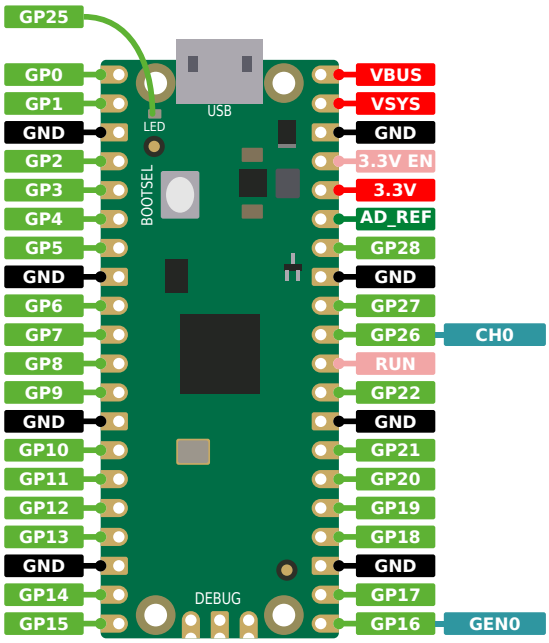
Kapitola 5

Návrh osciloskopu s Raspberry Pi Pico

Osciloskop je jedním ze základních měřicích přístrojů, který umožňuje vizualizaci analogového signálů v čase. To je užitečné pro ověření funkce elektronických obvodů, ale také pro názorné vysvětlení, jak obvod funguje. Pro studenty na ČVUT FEL, jsou osciloskopy dostupné v laboratořích, ale pro práci na jednoduchých obvodech může být dostačující i levný osciloskop sestrojený z mikrořadiče. Takovýto osciloskop navíc umožní práci mimo laboratoře. V této kapitole bude popsána realizace osciloskopu s pomocí Pico a PC aplikace Data Plotter.

Pico lze programovat pomocí programovacích jazyků Python a C/C++. Python je vhodný pro začátečníky, ale pro napsání složitého firmwaru vyžadující přímý přístup k periferiím mikrořadiče, je vhodnější použít jazyky C a C++. Se samotným C/C++ by ovšem programování bylo značně složité, protože ovládání periferií by vyžadovalo zapisování dat na správné adresy. Pro zjednodušení existuje knihovna PicoSDK, která obsahuje funkce, které práci s periferiemi značně zjednodušují. Firmware, který bude psán pro Pico bude tedy využívat jazyků C a C++ a knihovny PicoSDK.

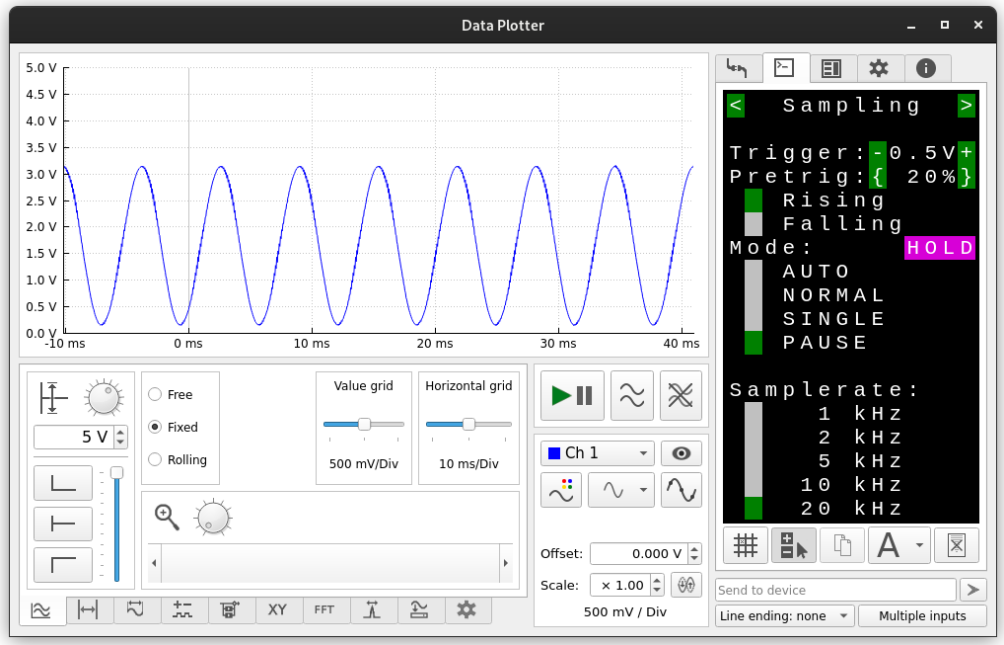
Na obr. 5.1 jsou znázorněny piny přiřazené pro osciloskop s Pico. Pin **CH0** značí vstup osciloskopu a pin **GEN0** značí generátor PWM.



Obr 5.1: Přiřazení pinů pro osciloskop s Pico

5.1 Popis PC aplikace Data Plotter

Pro sestrojení osciloskopu je potřeba řešit způsob nastavení parametrů měření a vizualizace naměřených dat. Pico samotné nemá takovéto možnosti, ale díky integrovanému USB konektoru, lze uskutečnit komunikaci mezi Pico a počítačem. Nabízí se tedy možnost vytvoření počítačové aplikace, která umožní ovládání i vizualizaci. Vytvoření takovéto aplikace může být však pracné a jednodušší možností je použít již existující aplikaci. Aplikaci, kterou lze využít pro vytvoření osciloskopu již vytvořil Jiří Maier v rámci své bakalářské práce pod názvem Data Plotter [1].



Obr 5.2: Ukázka aplikace Data Plotter

Aplikace Data Plotter slouží jako universální prostředí pro zobrazení analogových a digitálních signálů. Krom toho obsahuje aplikace terminálové rozhraní s podporou ANSI sekvencí, které slouží pro ovládání přístroje. Pro aplikaci je definován komunikační protokol, pomocí kterého přístroj posílá naměřená data a podobu terminálového rozhraní. Komunikační protokol využívá odesílání ASCI symbolů, což ho dělá lehce pochopitelný a jeho implementace není složitá. Data Plotter je proto vhodnou aplikací pro přístroje vytvořené s mikrořadiči a bude využita i pro osciloskop s Pico. Podobu aplikace Data Plotter, včetně terminálového rozhraní je znázorněno na obr. 5.2.

5.1.1 Komunikační protokol aplikace Data Plotter

Komunikační protokol pro Data Plotter obsahuje následující zprávy [1, s. 125]:

- **$$P** přidání bodu do grafu
- **$$C** přidání celého kanálu do grafu
- **$$L** přidání logického kanálu do grafu
- **$$B** přidání logického bodu do grafu
- **$$T** výpis do terminálu
- **$$S** nastavení
- **$$I** výpis informační zpráv

- **$$W** varovná zpráva
- **$$X** chyba zařízení (zobrazí zprávu a odpojí zařízení)
- **$$E** echo (pošle přijatý text zpět do zařízení)
- **$$U** neznámý (data jsou zahozena, nezpracují se)

Z těchto zpráv jsou pro osciloskop s Pico nejdůležitější **$$C** a **$$T**. Pomocí zprávy **$$C** posílá osciloskop digitalizovaný signál do Data Plotter. Celá zpráva má následující podobu [1, s. 127]:

$$C(ch),(časový krok),(délka),(bity),(min),(max),(index nuly);U?(data....);

Samotná naměřená data lze posílat v binární podobě a pomocí symbolu **?** v **u?** se označuje kolik bajtů zabírá jedno číslo. V případě 12bitového ADC převodníku v Pico bude jedna hodnota zabírat 2 bajty a data budou tedy označena **u2**.

Pomocí zprávy **$$T** definuje osciloskop podobu terminálu.
Například po odeslání zprávy `"$$TTest\e[1E\e[42m<\e[0m  Sampling  \e[42m>\e[0m"` bude mít terminál podobu jako na obr. 5.3.

Obr 5.3: Ukázka terminálu Data Plotter po odeslání zprávy **$$T**

Barva pozadí symbolů < a > je nastavena pomocí ANSI sekvence `"\e[42m"`. Změna barvy pozadí navíc udělá ze symbolu interaktivní tlačítko. Jakmile na symbol uživatel klikne, bude odeslán do mikrořadiče. Tímto způsobem lze zařídit, aby mikrořadič reagoval na kliknutí uživatele a vytvořit tak rozhraní, pomocí kterého může uživatel přístroj ovládat. V případě, kdy je požadováno barevné pozadí, které není interaktivní, lze toto nastavit pomocí zprávy `"$$Snoclickclr:"`. Tímto lze nastavit, aby konkrétní barvy nebyly interaktivní.

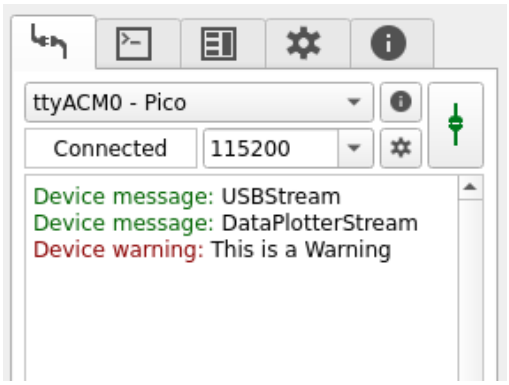
5.2 Implementace komunikace mezi Pico a Data Plotter

Komunikaci mezi Pico a Data Plotter je možné uskutečnit pomocí USB. Na základě poznatků z kapitoly 4.2 bude pro odesílání dat z Pico použita funkce `stdio_usb.out_chars()`. Pro přijetí dat z Data Plotter lze použít funkci `getchar()` ze standardní knihovny `<stdio.h>`. Tato funkce ovšem zastaví běh programu, dokud nejsou přijata nějaká data. PicoSDK proto dodává funkci `getchar_timeout_us()`, která požaduje jako parametr čas v milisekundách. Pokud do tohoto času nejsou přijata žádná data, vrací funkce záporné číslo [1].

Pro zjednodušení práce s USB a odesílání příkazů zpráv pro Data Plotter byly vytvořeny třídy `comm::USBStream` a `comm::DataPlotterStream`. `USBStream` slouží pouze jako abstrakce pro funkce

[1] Proměnná, do které se ukládá vrácená hodnota tedy musí být typu `int` nebo `signed char`.

na práci s USB a `DataPlotterStream` obsahuje metody pro odesílání konkrétních zpráv pro Data Plotter. Ukázka použití těchto tříd je ve výpisu 5.1. Po připojení Pico k Data Plotter se zobrazí zprávy jako na obr. 5.4.



Obr 5.4: Zprávy přijaté v Data Plotter s použitím pomocných tříd

```
int main() {
    comm::USBStream usb_stream{stdio_usb};
    comm::DataPlotterStream dataplotter{usb_stream};

    usb_stream.init();
    while (1) {
        if (usb_stream.connected()) {
            usb_stream.send("$$IUSBStream");
            dataplotter.send_info("DataPlotterStream");
            dataplotter.send_warning("This is a~Warning");
        }
        sleep_ms(1000);
    }
}
```

Výpis 5.1: Ukázka použití pomocných tříd pro komunkaci s Data Plotter

5.3 Implementace knihovny pro tvorbu terminálového rozhraní

Pro zjednodušení vytváření a úprav terminálového rozhraní byla vytvořena knihovna, pomocí které lze rozhraní stavět modulárně. Základem této knihovny je třída `dt::Terminal`, která obsahuje metody na vykreslení terminálového rozhraní a také na přepínání obrazovek. Pro vytvoření jednotlivých prvků terminálu slouží třídy `dt::StaticPart` a `dt::DynamicPart`. Ve třídě `dt::StaticPart` jsou uloženy statické prvky terminálu v podobně string a navíc je ve třídě uložen odkaz na objekt typu `dt::DynamicPart`. Vytvořené objekty typu `dt::StaticPart` se následně uloží do objektu `dt::Terminal` a pomocí něho lze prvky vykreslit.

5.3.1 Vykreslení statických prvků terminálového rozhraní

Ve výpisu 5.2 je ukázka použití tříd `Terminal` a `StaticPart` pro vykreslení jednoduchého terminálového rozhraní v Data Plotter, které je znázorněno na obr. 5.5. Třída `StaticPart` požaduje pro inicializaci počet řádků, který bude obsazovat a následně string, který se na terminál vypíše. Tím vytvoříme statické prvky terminálu, které se vloží do objektu `Terminal` pomocí metody

`push_terminal_part`. Pro vykreslení stačí zavolat metodu `print_static_elements`, která podle informace o počtu obsazených řádku statické prvky správně rozmístí. Pro rozmisťování uvnitř statického prvku nelze použít absolutní pozici `"\e[H"`.

Obr 5.5: Vykreslené terminálové rozhraní pomocí tříd `Terminal` a `StaticPart`

```
int main() {
    comm::USBStream usb_stream{stdio_usb};
    comm::DataPlotterStream dataplotter{usb_stream};
    dt::Terminal dterminal{dataplotter};
    dt::StaticPart static_example0{4, "\e[1E\e[1CExample\e[1E\e[1CStatic Part"};
    dt::StaticPart static_example1{1, "Interactive: \e[42m!"};

    usb_stream.init();
    dterminal.push_terminal_part(static_example0);
    dterminal.push_terminal_part(static_example1);
    while (1) {
        if (usb_stream.connected()) {
            dataplotter.send_info("Sending static parts!");
            dterminal.print_static_elements(true);
        }
        sleep_ms(1000);
    }
}
```

Výpis 5.2: Ukázka použití tříd pro vykreslení terminálového rozhraní

5.3.2 Vykreslení dynamických prvků terminálového rozhraní

Krom statických prvků v terminálovém rozhraní jsou užitečné také dynamické prvky, které umožní udělat rozhraní více interaktivní. Základem dynamických prvků je třída `dt::DynamicPart`, od které jsou odvozené další třídy pro konkrétní dynamický prvek. Těmi jsou

- `MultiButton` - několik tlačítek, ze kterých je aktivní pouze jedno,
- `IntNumber` - celé číslo,
- `FloatNumber` - číslo s desetinou čárkou,
- `Strings` - přepínání mezi texty.

Dynamický prvek je vždy součástí statického prvku a třída `Terminal` obsahuje metodu pro vykreslení dynamických prvků. Tato metoda vykreslí pouze dynamické prvky, které se změnily, ale vykreslení lze vynutit např. při prvním připojení Pico k Data Plotter.

Ve výpisu 5.3 je příklad použití dynamických prvků `MultiButton` a `IntNumber`. Uživatel může kliknout na jedno ze tlačítek, které po kliknutí zezelená a prvek `IntNumber` zobrazí číslo aktivního tlačítka (číslováno od 0). Na obrázcích 5.6a a 5.6b lze vidět podobu terminálu při kliknutí na první a druhé tlačítko.

(a) : Aktivní tlačítko 0 **(b)** : Aktivní tlačítko 1

Obr 5.6: Vykreslené terminálové rozhraní s dynamickými prvky

```
dt::MultiButton dtselector{2, 0, "ab", 0, comm::ansi::btn_pressed_str_green};
dt::StaticPart dtselector_part{3, "\e[3COption0\e[1E\e[3COption1", &dtselector};
dterminal.push_terminal_part(dtselector_part);

dt::IntNumber dtoption_num{11, 0, 0, 2, 0, false};
dt::StaticPart dtoption_num_part{1, "Selected:", &dtoption_num};
dterminal.push_terminal_part(dtoption_num_part);
while (1) {
    if (usb_stream.connected()) {
        if (first_connection) {
            /* Pri prvnim pripojeni se vykresli staticke i dynamicke prvky */
            dataplotter.send_info("Connected!");
            dterminal.print_static_elements(true);
            dterminal.print_dynamic_elements(true);
            first_connection = false;
        }
        int rx_char = usb_stream.receive_timeout(100);
        /* Byl prijat symbol a patri nekteremu z tlacitek? */
        if (rx_char > 0 && dtselector.button_pressed_char(rx_char) >= 0) {
            /* Nastavi cislo na aktualne aktivni tlacitko */
            dtoption_num.set_value(dtselector.get_active_button());
        }
        /* Aktualizuje dynamicke prvky pokud se zmenili */
        dterminal.print_dynamic_elements();
    } else {
        first_connection = true;
    }
}
```

Výpis 5.3: Ukázka použití dynamických prvků terminálového rozhraní

5.4 Navržené terminálové rozhraní pro osciloskop

S pomocí knihovny pro tvorbu terminálové rozhraní, bylo vytvořeno rozhraní pro ovládání osciloskopu. Rozhraní je rozděleno do 4 stránek, kterými jsou

- Sampling - nastavení režimu vzorkování a trigger,
- ADC Freq - nastavení přesné frekvence vzorkování,
- PWM Gen - nastavení režimu PWM generátoru,
- About - informace o firmwaru.

Rozdělení do několika stránek bylo zvoleno pro zpřehlednění ovládání osciloskopu. Přepínání mezi stránkami je vždy v horní části a provádí se pomocí tlačítek se symboly <, >. Pro přepnutí do stránky About je vyhrazeno tlačítko se symbolem **?**.

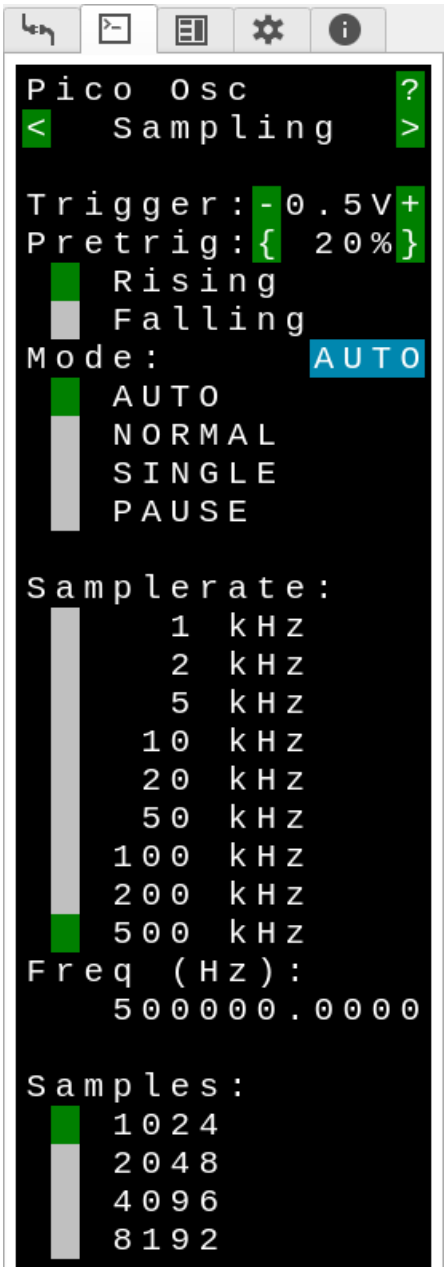
5.4.1 Stránka About terminálového rozhraní

Tato stránka obsahuje informace o osciloskopu. Na začátku stránky jsou napsány přiřazené piny. Následuje verze firmwaru a datum kompilace firmwaru. Nakonec stránka obsahuje informace o tom, kde firmware vznikl.

Obr 5.7: Ukázka stránky About

5.4.2 Stránka Sampling terminálového rozhraní

První stránkou po připojení k Pico z aplikace Data Plotter je Sampling. Podobu stránky je možné vidět na obr. 5.8. Pod výběrem stránky se nachází nastavení trigger, kde lze nastavit úroveň triggeru pomocí symbolů + a -. Dále lze nastavit počet vzorků před trigger a 4 režimy. V režimu SINGLE, bude osciloskop čekat, dokud nenastane trigger podmínka a poté co nastane, odešle data do Data Plotter a vzorkování se vypne. Režim NORMAL funguje podobným způsobem, ale vzorkování se opětovně spustí. V režimu AUTO se data odešlou do Data Plotter také pokud trigger nenastal v daném množství vzorků. Režim PAUSE zastaví vzorkování. Pod nastavení triggeru se nachází nastavení vzorkování, jako je frekvence vzorkování a počet vzorků, který se digitalizuje. Kolonka „Freq (Hz)“ zobrazuje aktuální vzorkovací frekvenci, což může být také frekvence nastavená na stránce ADC Freq.

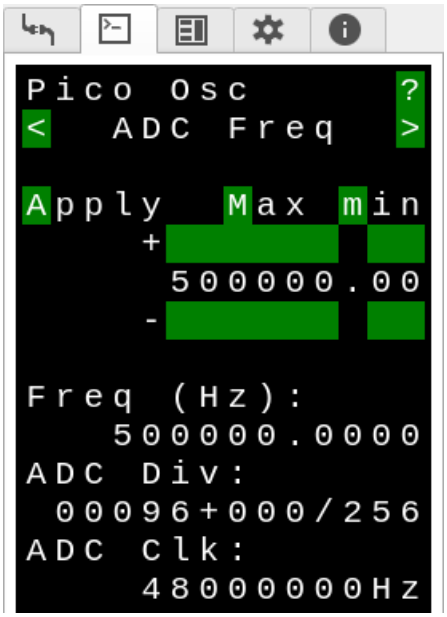


Obr 5.8: Ukázka stránky Sampling

5.4.3 Stránka ADC Freq terminálového rozhraní

Tato stránka je vyhrazena pro přesné nastavení vzorkovací frekvence osciloskopu. Frekvence se nastavuje kliknutím na tlačítko nad nebo pod číslicí. Kliknutím nad číslici se zvětší o 1 a pod číslici se sníží o 1. Kliknutím na tlačítko Apply se nastaví aktuálně nastavené číslo jako vzorkovací frekvence. Pokud tato frekvence není dosažitelná pomocí děliče, je nastavena nejbližší možná. Pomocí tlačítek Max a min lze také nastavit maximální a minimální možnou vzorkovací frekvenci.

Pod nastavením frekvence jsou zobrazeny informace o aktuální vzorkovací frekvenci, hodnota děličky hodinového signálu pro ADC a frekvence hodinového signálu. Jakmile se nastaví frekvence ve stránce ADC Freq, zešediví všechna tlačítka pro výběr vzorkovací frekvence na stránce Sampling, aby nenastala situace, kdy uživatel neví, která vzorkovací frekvence je nastavena. Podoba stránky ADC Freq je znázorněna na obr. 5.9.



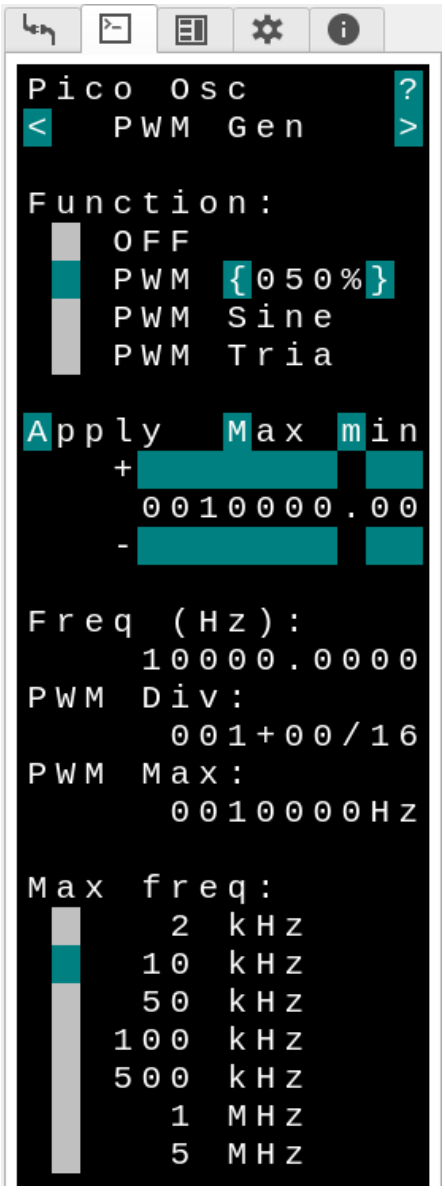
Obr 5.9: Ukázka stránky ADC Freq

5.4.4 Stránka PWM Gen terminálového rozhraní

Osciloskop s Pico umožňuje také generování PWM signálu pomocí PWM čítače. Na stránce PWM Gen lze nastavit parametry tohoto PWM signálu. V části Function lze generátor vypnout, nastavit na PWM signál s pevnou střídou nebo také signál sinus a trojúhelník. Po nastavení signálů sinus a trojúhelník se střída PWM mění automaticky, aby střední hodnota signálu přibližně odpovídala zvolenému signálu. Pod nastavením funkce lze nastavit frekvenci PWM signálu podobným způsobem, jako se nastavuje frekvence vzorkování na stránce ADC Freq (kapitola 5.4.3). Signály sinus a trojúhelník jsou složené z 64 hodnot a jejich frekvence se spočítá pomocí vzorce 5.1, kde f_{sig} je frekvence signálu a f_{pwm} je frekvence PWM.

$$f_{sig} = \frac{f_{pwm}}{64} \tag{5.1}$$

Pod nastavení frekvence jsou zobrazeny informace o aktuální frekvenci PWM signálu, hodnota děličky PWM čítače a maximální frekvence, která je určena maximální hodnotou PWM čítače. Při generování PWM signálu s pevnou střídou lze maximální frekvenci měnit v kolonce Max freq, což umožňuje nastavit větší rozsah frekvencí PWM signálu.



Obr 5.10: Ukázka stránky PWM Gen

5.5 Vzorkování a digitalizace signálu pomocí AD převodníku

Pro realizaci osciloskopu je potřeba pravidelně vzorkovat a digitalizovat aktuální hodnotu napětí na vstupu. Digitalizace napětí probíhá pomocí AD převodníku, ze kterého se hodnota vyčte a uloží do paměti. AD převodník v RP2040 je možné nastavit do tzv. Free Running režimu, kdy se po každém převodu spustí další převod v pravidelném intervalu. Tento interval lze nastavit pomocí děličky a díky tomu lze realizovat vzorkování signálu s volitelnou frekvencí. AD převodník zapisuje digitalizované hodnoty do registru RESULT, ale je také možné zapnout FIFO, ve kterém se hodnoty ukládají, pokud nejsou vyčteny včas. S těmito funkcemi lze zajistit vzorkování signálu pro osciloskop.

5.5.1 Vyčítání digitalizovaných hodnot z AD převodníku

Samotné vyčítání hodnot a ukládání do paměti lze zajistit pomocí procesoru mikrořadiče, ale lepším způsobem se jeví použít DMA řadič, který dokáže hodnoty vyčítat a ukládat nezávisle na procesoru, čímž se uvolní výkon pro další procesy. Pro vyčítání hodnot tedy bude vyhrazen DMA kanál, který lze nastavit, aby provedl přenos, jakmile je dostupná hodnota z AD převodníku. Pro tento účel se musí zapnout také FIFO a nastavit úroveň FIFO, která vyvolá DMA přenos. Tato úroveň bude nastavena na 1, aby se vyčetla vždy aktuální hodnota.

Frekvence vzorkování hodnot se nastaví pomocí děličky, která dělí hodinový signál AD převodníku. Ten odpovídá hodinovému signálu pro USB periferii a má frekvenci 48MHz. Frekvenci vzorkování spočítáme pomocí vzorce 5.2, kde f_{osc} je frekvence vzorkování a N_{div} je hodnota děličky.

$$f_{osc} = \frac{48 \times 10^6}{N_{div}} \tag{5.2}$$

Hodnota děličky je rozdělena na celou část n_{int} a racionální část n_{frac}. Celou část lze nastavit na hodnotu 0-65535 a racionální na hodnotu 0-255. Hodnota děličky se spočítá pomocí vzorce 5.3. Minimální hodnota děličky je 96 a určuje maximální dosažitelná frekvence vzorkování 500000 kHz. Maximální hodnota děličky odpovídá ≈ 65536.99609 a určuje minimální frekvenci vzorkování ≈ 732.4107 Hz.

$$N_{div} = 1 + n_{int} + \frac{n_{frac}}{256} \tag{5.3}$$

5.6 Použití druhého jádra pro vzorkování a trigger

Jednou ze základních funkcí osciloskopů je trigger. Tato funkce umožňuje zachytit situaci, kdy napětí na vstupu osciloskopu překročí nad nebo klesne pod nastavenou hodnotu. To umožňuje lépe pozorovat periodické signály a také zachytit signály, které se objevují jednou za delší dobu. V mikrořadičích od firmy STM lze trigger řešit pomocí funkce analog watchdog, kdy lze nastavit vyvolání přerušení, jakmile napětí na vstup AD převodníku překročí nastavenou hodnotu. AD převodník v RP2040 takovouto funkci ovšem nenabízí a je potřeba řešit trigger jiným způsobem.

Jedním způsobem, jak hledat trigger v digitalizovaném signálu je číst procesorem každou novou hodnotu, kterou DMA přenese z AD převodníku. Procesor ovšem v tu chvíli nesmí dělat příliš mnoho jiných procesů, aby se stíhaly číst všechny hodnoty a trigger se neminul. Procesor by tedy nesměl spravovat terminálové rozhraní, komunikaci pomocí USB apod. Díky druhému procesoru v RP2040 je ovšem možné předat činnost hledání trigger podmínky vedlejšímu procesoru a nezatěžovat tak hlavní procesor. Vedlejší procesor bude muset i ukončit vzorkování na základě toho kdy trigger podmínku našel a z toho důvodu bude správa AD převodníku a příslušnému DMA kanálu také ponechána vedlejšímu procesoru.

5.6.1 Komunikace mezi procesory RP2040

Komunikace s Data Plotter a správa terminálového rozhraní bude ponechána hlavnímu jádru, vedlejší jádro tak ovšem nemá možnost poznat, kdy má započít vzorkování a s jakými parametry. Je tedy potřeba zařídit komunikaci mezi procesory. Jedním způsobem je vytvoření globálních proměnných, ke kterým přistupují oba procesory. To je ovšem nebezpečné a mohou nastat chyby v situacích, kdy oba procesory zapisují nebo čtou stejnou proměnnou najednou. RP2040 na zamezení takovéto situace disponuje funkcemi pro zařízení komunikace a pro zamezení chybového přístupu do paměti oběma procesory.

Hlavním způsobem, kterým lze posílat zprávy mezi procesory jsou FIFO registry. RP2040 obsahuje 2 tyto registry, které jsou jednosměrné a lze pomocí nich zařídit odesílání 32bitových čísel mezi procesory. Zprávy, které se mezi jádry budou posílat byly definovány jako datový typ enum. Vytvořené datové typy jsou ve výpisu 5.4. Zprávy typu `core0_message` slouží pro oznámení vedlejšímu jádru, že má začít vzorkování a také vynucené zastavení vzorkování. Zprávy `core1_message` slouží k oznámení hlavnímu jádru, že bylo vedlejší jádro úspěšně spuštěno a oznámení dokončení vzorkování. Knihovna PicoSDK dodává funkce pro zapsání zprávy do FIFO `multicore_fifo_push_blocking()` a funkce pro čtení z FIFO `multicore_fifo_pop_blocking()`. Pro zjištění, zda FIFO obsahuje hodnotu, bude využita funkce `multicore_fifo_rvalid()`.

```
enum core0_message : uint32_t {
    START_ADC_AUTO = 0x00000000U,
    STOP_ADC,
    START_ADC_SINGLE,
};

enum core1_message : uint32_t {
    CORE1_STARTED = 0x80000000U,
    ADC_DONE,
};
```

Výpis 5.4: Zprávy pro komunkaci mezi procesory

FIFO registry jsou dostačující pro odeslání jednoduchých příkazů, ale pro získání parametrů vzorkování vedlejším jádrem bude potřeba více informací. Jedním možným způsobem, jak toto zařídit je queue dodávaná v PicoSDK. Ta umožňuje odesílat mezi procesory i pole proměnných a struktury. Pro osciloskop byl ovšem zvolen jednodušší způsob, kterým jsou globální proměnné a zámek mutex z knihovny PicoSDK. Mutex složí jako ochrana dat, ke kterým přistupují oba procesory. Pokud si mutex přivlastní jeden procesor, druhý musí počkat, než se mutex uvolní, aby mohl pokračovat. Pokud tedy vytvoříme globální proměnnou, ke které procesor přistupuje pouze pokud si přivlastnil mutex, zamezíme chybovým situacím, kdy oba procesory k této proměnné přistupují.

Pro odesílání informací o vzorkování byly vytvořeny třídy `DataForCore0` a `DataForCore1`, které obsahují potřebné informace a také mutex. Pro přivlastnění a uvolnění mutex obsahují třídy metody `lock()` a unlock `unlock()`. Ukázka použití těchto tříd je ve výpisu 5.5. Hlavní jádro před spuštěním vzorkování zamkne a zapíše data do globálního objektu `DataForCore1`, následně objekt odemkne a zašle zprávu pro spuštění vzorkování druhému jádru. Druhé jádro si objekt také zamkne, vyčte z něj parametry, odemkne a započne vzorkování.

```
mutex_t datac1_mutex;
DataForCore1 datac1{&datac1_mutex};

int main(){
    datac1.init_mutex();

    datac1.lock();
    datac1.number_of_samples = 1000;
    datac1.unlock();
```

Výpis 5.5: Zamčení objektu typu DataForCore1

5.6.2 Spuštění druhého procesoru

Ve výchozím stavu je spuštěn pouze hlavní procesor a je tedy potřeba vedlejší procesor spustit. Pro to slouží funkce `multicore_launch_core1()`, které se jako parametr dodá funkce, která se má na druhém procesoru spustit. Ukázka zapnutí druhého procesoru je ve výpisu 5.6. Hlavní procesor v tomto případě spustí na vedlejším procesoru funkci `core1_main()` a následně čeká na zprávu, že bylo vedlejší jádro úspěšně spuštěno.

```
void core1_main() {
    multicore_fifo_push_blocking(CORE1_STARTED);
    while(1){
    }
}

int main(){
    multicore_launch_core1(core1_main);

    core1_message msg = (core1_message)multicore_fifo_pop_blocking();
    if (msg != CORE1_STARTED) {
        panic("$$XUnexpectred message form Core1");
    }

    while(1){
    }
}
```

Výpis 5.6: Spuštění druhého procesoru

5.6.3 Implementace funkce trigger pro osciloskop

Jak již bylo zmíněno, zapnutí AD převodníku, DMA a hledání trigger v digitalizovaném signálu bude řešit druhý procesor. Po přijetí zprávy, že se má spustit vzorkování, tedy procesor spustí AD převodník, příslušný DMA kanál a následně musí číst data přenesená pomocí DMA, aby zkontroloval, zda nenastal trigger. Z registrů DMA kanálu lze vyčíst, kolik hodnot zbývá přenést do konce datového pole. Pomocí toho může tedy procesor zjistit, že je v datovém poli nový vzorek a ten porovnat s předchozím. Pokud procesor pozná, že nastal trigger, uloží index vzorku a ukončí AD převodník po zvoleném množství vzorků. Aby byla ošetřena situace, kdy je nalezen trigger ke konci DMA přenosů, bude použit další DMA kanál, který ten hlavní opětovně spustí. Tím zařídíme cyklický režim DMA, který je blíže popsán v kapitole 4.4.

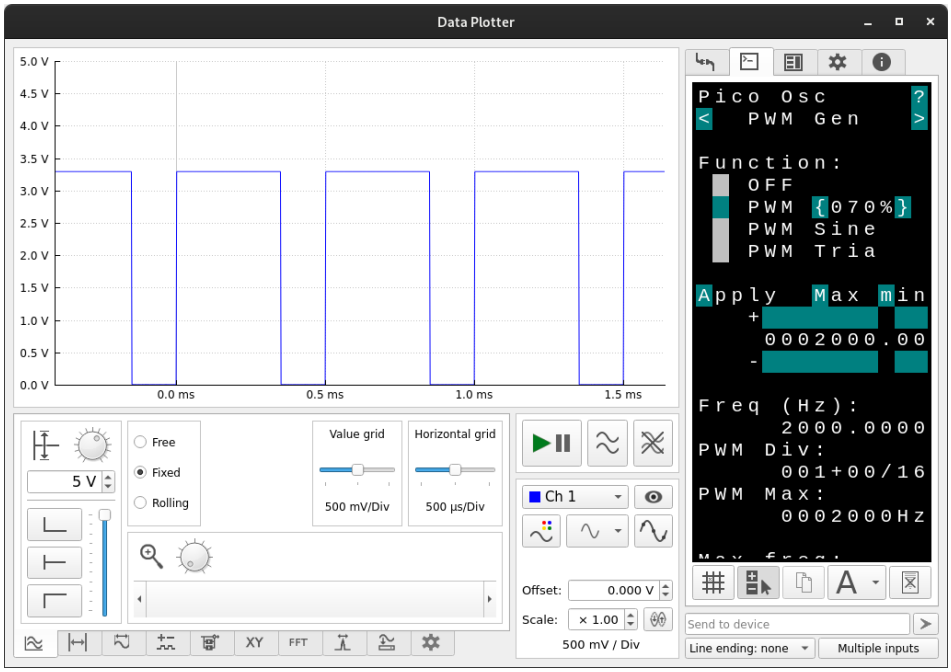
5.6.4 Problém s kontrolou dokončení přenosů DMA kanálu v cyklickém režimu

Vzorkování je potřeba ukončit, pokud nastala trigger podmínka a poté bylo zaznamenáno dostatečné množství vzorků. Pokud trigger nastal dostatečně brzy, stačí zamezit, aby se spustil další DMA cyklus. Pokud trigger nastal pozdě, je potřeba DMA kanál ukončit po daném množství vzorků v dalším cyklu. To bylo nejdříve řešeno nastavením registru TRANSFER_COUNT hlavního DMA kanálu, což nastaví počet přenosů pro další cyklus DMA kanálu. Pro rozpoznání, zda DMA kanál dokončil přenosy byla využita funkce `dma_channel_is_busy(adc_chan)`. V situacích, kdy byly potřeba vzorky z dalšího cyklu, nebyl signál správně vzorkován.

Tento problém byl původně přidružen tomu, že procesor nestíhá vzorky kontrolovat a správně nastavit DMA. Ukázalo se ovšem, že problém je opačný. V situaci, kdy hlavní DMA kanál dokončí přenos daného počtu vzorků a je opětovně spuštěn se kanál jeví jako vypnutý. Kanál se opět jeví jako zapnutý, jakmile AD převodník dodá nový vzorek a DMA kanál provede první přenos v novém cyklu. Procesor situaci, kdy se před prvním přenosem jeví kanál jako vypnutý, stihl odchytit a vzorkování se tak bralo jako dokončené. Místo kontrolování funkce `dma_channel_is_busy(adc_chan)` bylo proto využito čtení aktuálního počtu přenosů DMA kanálu a kanál je ukončen procesorem pomocí `dma_channel_abort(adc_chan)`.

5.7 Generátor PWM

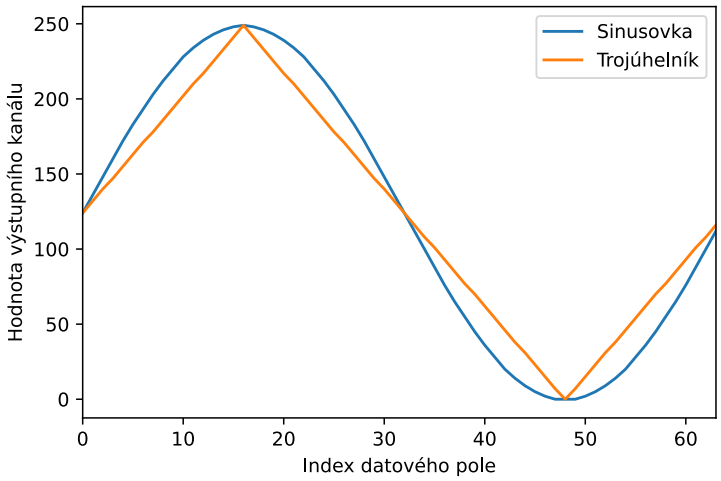
Pro práci s osciloskopem může být užitečný také generátor, kterým lze například osciloskop vyzkoušet nebo generovat signál pro měřený obvod. Pro tyto účely byl využit PWM čítač, pomocí kterého lze generovat PWM signál s nastavitelnou střídou a frekvencí. Ukázka záznamu PWM signálu je na obr. 5.11. Výstup generátoru **GEN0** je v tomto případě zapojený přímo na vstupní kanál **CH0**.



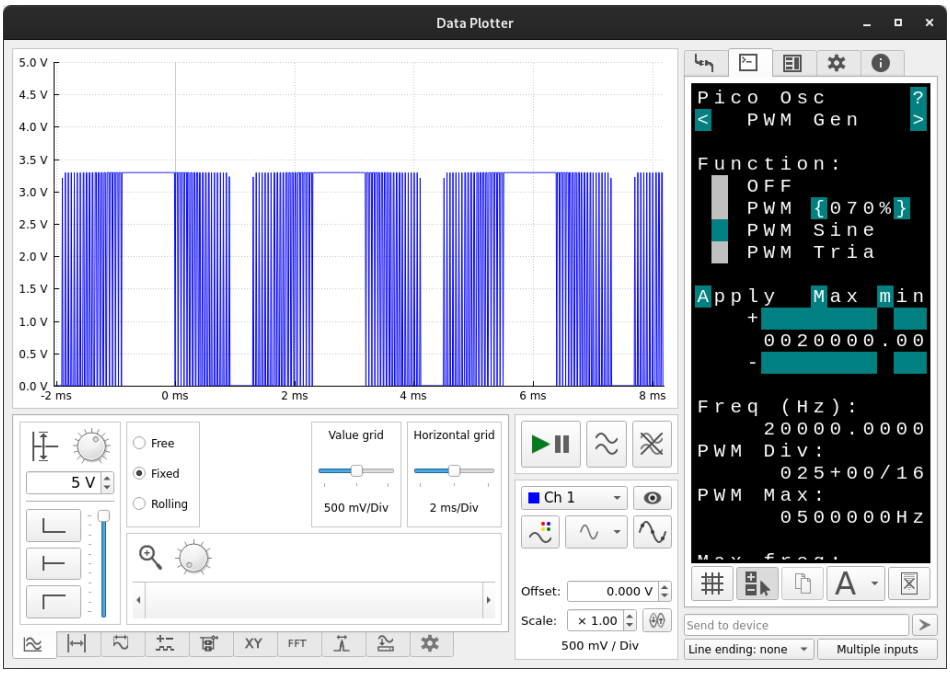
Obr 5.11: Výstupní PWM signál generátoru

DMA řadič RP2040 umožňuje nastavit požadavek pro přenos DMA kanálem, jakmile PWM čítač dosáhne maximální hodnoty. Tímto se nabízí vytvořit generátor PWM signálu, jehož střída se změní po každé periodě. Jakmile PWM čítač dosáhne maximální hodnoty, spustí se přenos DMA kanálu z datového pole do registru výstupního kanálu čítače. Tím se díky DMA změní střída výstupního signálu. DMA v tomto případě musí být v cyklickém režimu, který lze zařídit pomocí dalšího DMA kanálu, což je blíže popsáno v kapitole 4.4.

Střída PWM signálu se může měnit podle určité funkce, jako je např sinusovka. Bude tedy potřeba vygenerovat pole s hodnotami střídy, ze kterého bude DMA kanál číst. Toto pole bylo vygenerováno pomocí Python skriptu, který využívá knihovnu numpy pro vygenerování pole. Pro PWM čítač byla zvolena maximální hodnota 250 a počet prvků pole je 64. Krom sinusovky byl také vygenerován trojúhelník. Vygenerovaná pole v Python skriptu byla vykreslena do grafu na obr. 5.12.

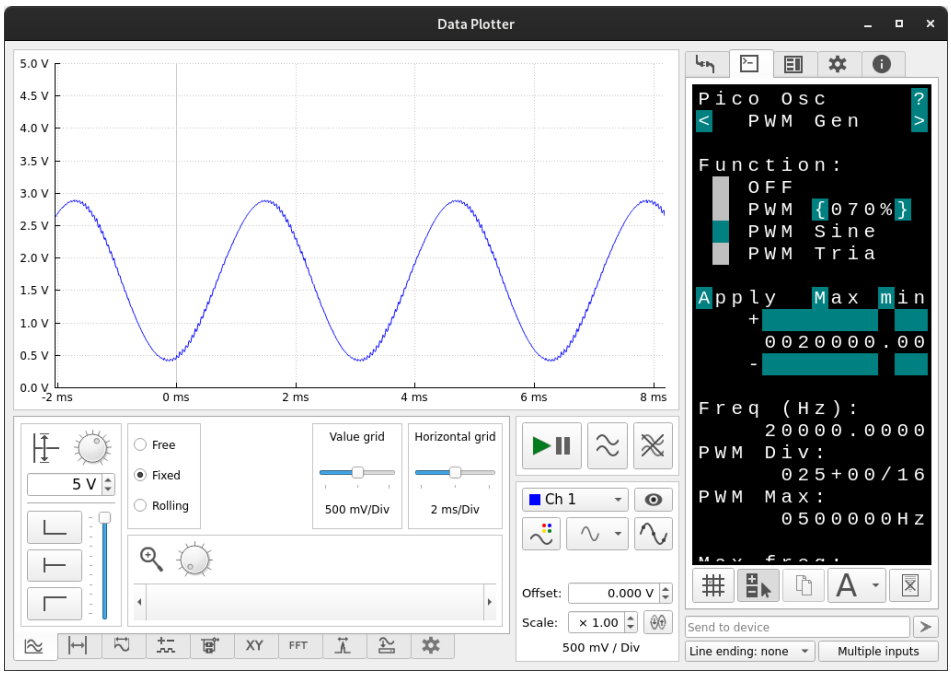


Obr 5.12: Graf vykreslených z datových polí s hodnotami pro výstupní kanál PWM čítače

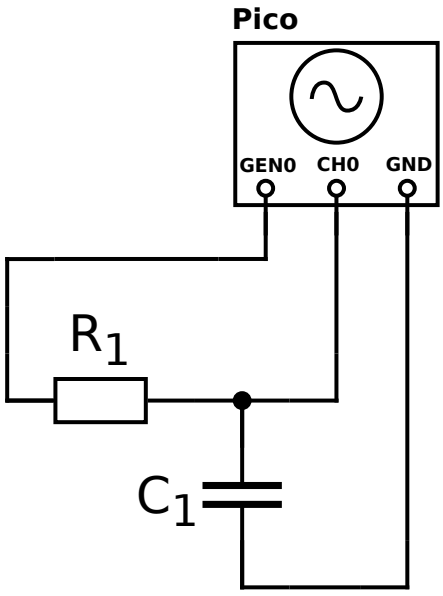


Obr 5.13: Výstupní PWM signál modulovaný sinusovkou

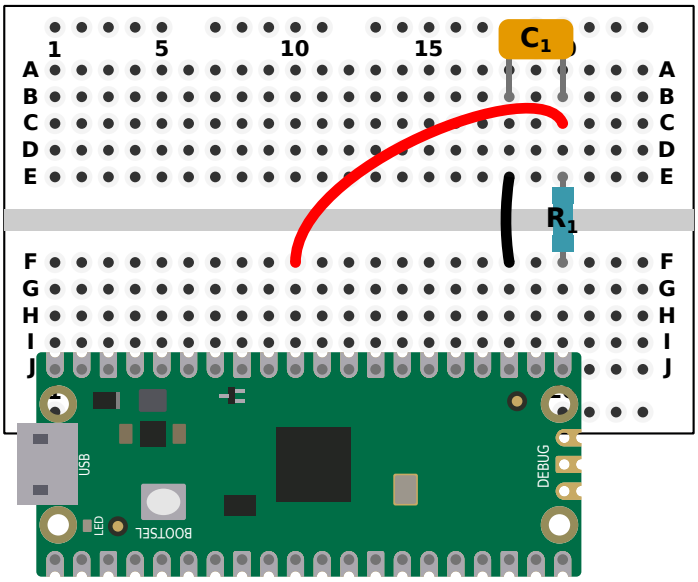
Po nastavení generátoru na funkci sinus v terminálovém rozhraní je možné signál jako na obr. 5.13. Výstup generátoru je v tomto případě zapojen přímo na vstup osciloskopu. Na obr. 5.14 je ukázka zaznamenaného signálu se stejným nastavením generátoru a vzorkování, ale navíc se zapojeným RC filtrem mezi výstup generátoru a vstup osciloskopu. Schéma zapojení s RC článkem je znázorněno na obr. 5.15a a ukázka zapojení na nepájivém poli je na obr. 5.15b.



Obr 5.14: Výstupní PWM signál modulovaný sinusovkou se zapojeným RC filtrem



(a) : Schéma zapojení generátoru s RC filtrem



(b) : Zapojení generátoru s RC filtrem na nepájivém poli

Obr 5.15: Zapojení generátoru s RC filtrem

Frekvenci sinusovky f_{sine} spočítáme pomocí vzorce 5.4, kde f_{pwm} je frekvence PWM, kterou můžeme vyčíst z terminálového rozhraní v Data Plotter. V případě znázorněném na obr. 5.14 je $f_{pwm} = 20$ kHz a frekvence sinusovky je tedy $f_{sine} = 312.5$ Hz. Pro RC článek byli zvoleny hodnoty součástek $R_1 = 4.7\text{k}\Omega$ a $C_1 = 100\text{nF}$. Časová konstanta RC článku se spočítá pomocí rovnice 5.5. V tomto případě je časová konstanta 470μs což je 94× větší než perioda PWM signálu.

$$f_{sine} = \frac{f_{pwm}}{64} \tag{5.4}$$

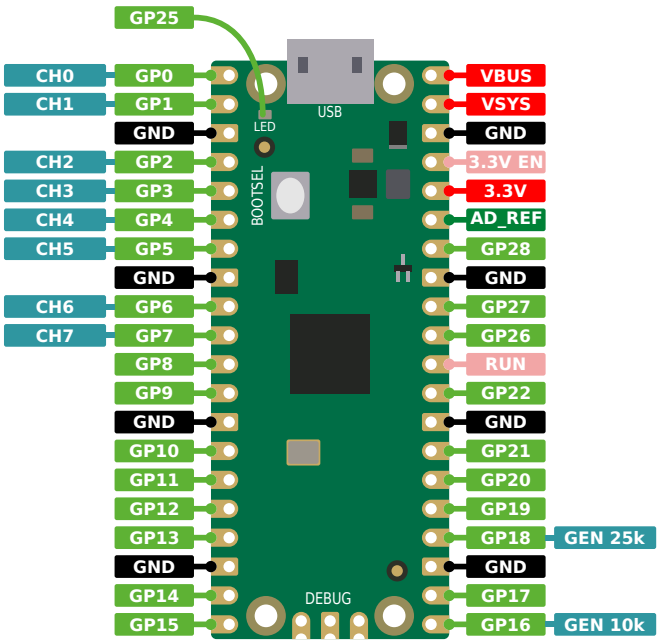
$$\tau = R_1 \times C_1 \tag{5.5}$$

Kapitola 6

Návrh logického analyzátoru s Raspberry pi Pico

Logický analyzátor umožňuje vizualizovat digitální signály a případně také tyto signály analyzovat. Analýza digitálních signálů může být užitečná například pro ověření funkce digitální komunikací mezi mikrořadiči nebo komunikaci s digitálními senzory. Na rozdíl od osciloskopu neměří logický analyzátor hodnotu napětí na vstupu, ale pouze stav 0 a 1. Díky tomu není nutné použití AD převodníku a lze použít digitální vstupy Pico. V této kapitole bude popsána realizace logického analyzátoru s Pico a aplikací PulseView. Stejně jako v případě osciloskopu, bude Pico programováno v C++ s knihovnou PicoSDK.

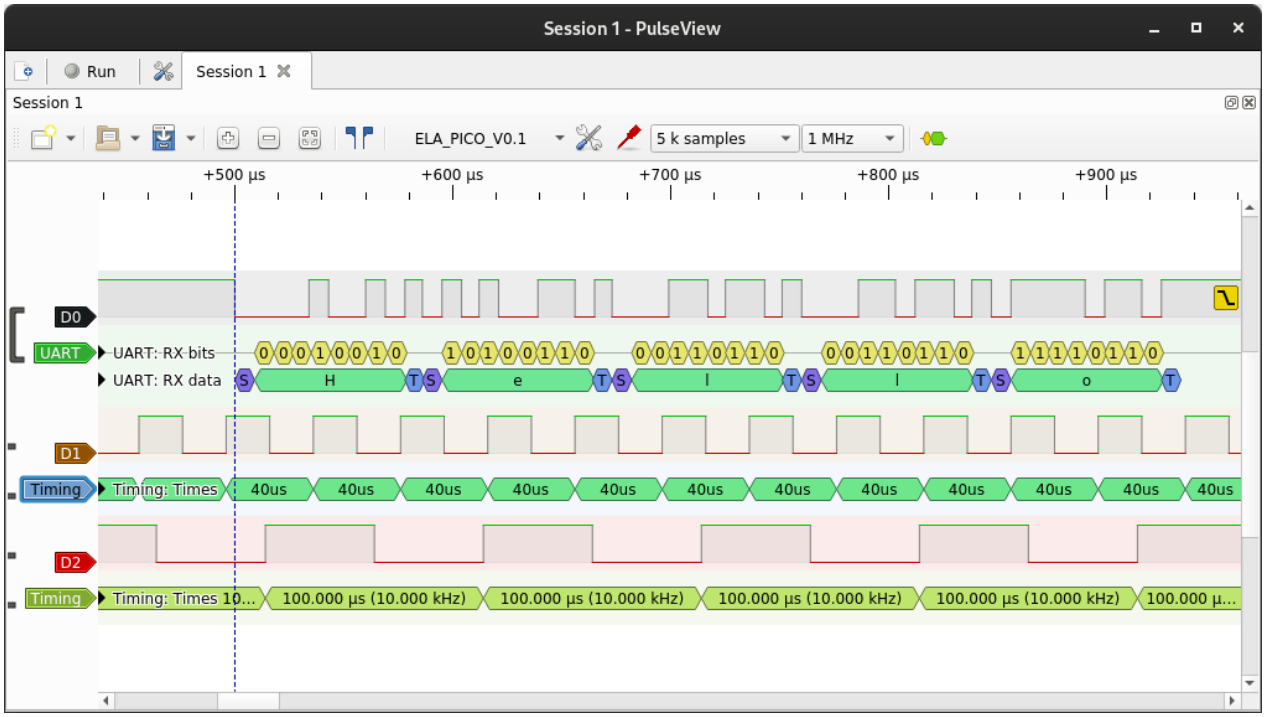
Na obr. 6.1 jsou znázorněny piny přiřazené pro logický analyzátor s Pico. Piny **CH0 - CH7** značí vstupy logického analyzátoru. Piny **GEN 10k** a **GEN 25k** značí generátory PWM signálů s frekvencí 10 kHz a 25 kHz.



Obr 6.1: Přiřazení pinů pro logický analyzátor s Pico

6.1 Popis PC aplikace PulseView

Logický analyzátor vyžaduje možnost nastavení parametrů vzorkování, vizualizaci digitálních signálů a vhodná je také možnost analyzovat digitální komunikace, digitální kódy apod. Pro tyto účely je možné využít PC aplikaci, která s Pico komunikuje pomocí USB. Vytvořit aplikaci umožňující všechny tyto požadavky by bylo pracné a stejně jako v případě osciloskopu, bude využita již existující aplikace. Pro logický analyzátor bude využita otevřená aplikace PulseView, jejíž hlavní výhodou je možnost analyzovat digitální signál a např. zobrazit data odesílaná pomocí sériových komunikací jako jsou UART a SPI. Podoba PulseView je znázorněna na obr. Aplikace je součástí projektu Sigrok, jehož součástí jsou i další otevřené aplikace a knihovny [7].



Obr 6.2: Ukázka aplikace PulseView

6.2 Komunikace s PulseView

Pro komunikaci s PulseView byl využit ELA protokol znázorněný v tabulce 6.1. To vyžaduje upravenou verzi aplikace PulseView, která byla zkompilovaná s upravenou knihovnou libsigrok v rámci bakalářské práce [8]. Knihovna libsigrok vyžadovala další úpravy pro podporu Pico. Jednou z těchto úprav je zapnutí příznaku DTR, který Pico vyžaduje pro uskutečnění komunikace, jak bylo popsáno v kapitole 4.3. Při úpravě byly také opraveny některé chyby s alokací paměti a možnost vyšších vzorkovacích frekvencí, kterých je Pico schopno.

Příkaz	Podpříkaz	Množství dat [B]
Reset		1
Handshake		1
Start		1
Stop		1
Set	Samplerate	6
	Sample-Count	6
	Pin-Mode	6
	Pretrig-Count	6
Get	Samplerate	2
	Sample-Count	2
	Pin-Mode	4
	Pretrig-Count	2
	Metadata	2
	Sampled-Data	2
Report	Samplerate	6
	Sample-Count	6
	Pin Mode	6
	Pretrig-Count	6
	Metadata	≥ 13
	Sampled-Data	≥ 10

Tabulka 6.1: Seznam zpráv ELA protokolu [8, s. 37]

6.3 Čtení digitálních vstupů pomocí PIO a DMA

V případě osciloskopu s Pico probíhá záznam signálů pomocí AD převodníku, ze kterého se vyčítají digitalizované hodnoty pomocí DMA. Pro logický analyzátor nepotřebujeme znát velikost napětí, ale pouze stav 0 značící, že napětí na vstupu je blízké nule a stav 1 značící, že napětí na vstupu je blízké napájecímu napětí mikrořadiče. Tyto stavy lze číst z registrů digitálních vstupů mikrořadiče. Stavy je možné číst pomocí procesoru, ale výhodnější je použití DMA, které dokáže stavy vstupů číst nezávisle na procesoru a periodu odečítání hodnot lze přesně nastavit interními čítači DMA řadiče nebo pomocí PWM čítače.

V mikrořadičích jako STM32 je možné pomocí DMA číst přímo vstupní GPIO registry. S RP2040 by DMA se adresa čtení jako ve výpisu 6.1. To ovšem nefunguje, protože řadič DMA k tomuto registru nemá přístup. Registr GPIO_IN je možné číst pouze pomocí procesorů, ale číst stavy vstupních pinů má možnost také periferie PIO.

```
dma_channel_set_read_addr(gpio_dma_chan, &sio_hw->gpio_in, false);
```

Výpis 6.1: Nastavení DMA pro čtení z GPIO_IN registru

6.3.1 Jednoduché čtení digitálních vstupů pomocí PIO

Jak již bylo zmíněno v kapitole 3.7, PIO je programovatelná periferie sloužící pro práci s GPIO. Každý PIO blok obsahuje 4 stavové automaty, které lze programovat pomocí 9 instrukcí. Každá z těchto instrukcí je garantována, že proběhne v jednom hodinovém cyklu. Počet hodinových cyklů lze pro každou instrukci prodloužit, což umožňuje vytvořit programy s velice přesným načasováním. Program lze psát pomocí jazyka pioasm v souboru .pio, který se při kompilaci programu přeloží na hlavičkový soubor a ten lze použít v C/C++.

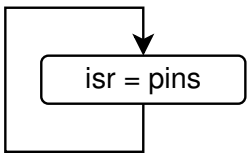
Každému stavovému automatu jsou přiřazeny vstupní a výstupní FIFO registry, ze kterých může číst i DMA kanál. Stavový automat má možnost nastavit výstupní piny ale také číst stav vstupních pinů. Bude tedy možné vytvořit program, který přečte stav pinů, hodnotu vloží do FIFO registrů a z nich bude hodnota automaticky přečtena pomocí DMA kanálu. Takovýto program je znázorněn ve výpisu 6.2 a odpovídajícímu vývojovému diagramu na obr. 6.3. Program využívá instrukci `in pins`, která přečte stavy pinů a tuto hodnotu uloží do interního registru `isr`. Z registru `isr` je možné přesunout hodnotu do FIFO registrů pomocí instrukce `push`. Je ovšem možné nastavit automatický přesun do FIFO, jakmile se v `isr` objeví nová hodnota. Díky tomu bylo možné zařídit čtení pinů jednou instrukcí.

Program ve výpisu 6.2 je schopen číst stav pinů s frekvencí, která odpovídá systémovému hodinovému signálu. Frekvence tohoto hodinového signálu je ve výchozím stavu 125MHz. Pro logický analyzátor by se ovšem hodilo nastavit vzorkovací frekvenci, což lze zařídit pomocí děličky hodinového signálu, kterou lze nastavit pro každý stavový automat. Při rychlém vzorkování může nastat problém, kdy DMA nestíhá přenášet nové hodnoty. Pro omezení takového problému, byla zvýšena priorita DMA přenosů na interní sběrnici nastavením registrů sběrnice jako ve výpisu 6.3

```
.program pins_input
.define public PIN_COUNT 8

.wrap_target
    in pins, PIN_COUNT
.wrap
```

Výpis 6.2: Program pro čtení digitálních vstupů pomocí PIO



Obr 6.3: Vývojový diagram PIO programu s čtením pinů po 1 instrukci

```
bus_ctrl_hw->priority = BUSCTRL_BUS_PRIORITY_DMA_W_BITS |
                        BUSCTRL_BUS_PRIORITY_DMA_R_BITS;
```

Výpis 6.3: Zvýšení priority DMA přenosů na sběrnici

6.3.2 Čtení vstupů a ukončení čtení pomocí PIO

Logický analyzátor s Pico bude vzorkovat vždy nastavený počet vzorků okamžitě po spuštění, nebo po tom co nastala trigger podmínka. Vzorkování je tedy nutné ukončit po nastaveném počtu vzorků. V případě osciloskopu s Pico bylo ukončení vzorkování vyřešeno procesorem, který kontroluje počet přenosů DMA kanálu. Čtení analogových hodnot v osciloskopu je ovšem omezeno rychlostí AD převodníku a procesor je dostatečně rychlý na to, aby DMA kanál stíhal hlídat. V případě logického analyzátoru může být čtení hodnost značně rychlejší, než v případě osciloskopu a problém ukončení vzorkování je potřeba řešit jiným způsobem. Vzhledem k nutnosti použití PIO pro čtení vstupu se nabízí možnost vyřešit problém pomocí samotného PIO.

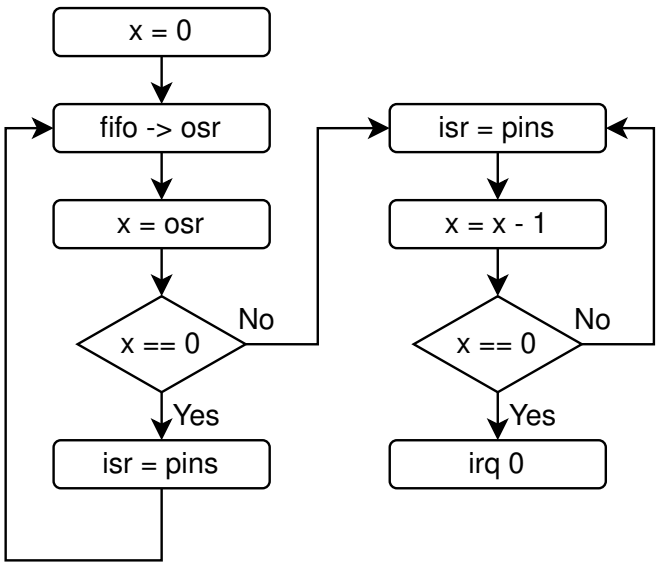
Ve výpisu 6.4 je znázorněna první verze PIO programu, který krom čtení vstupu umožňuje ukončení po daném počtu vzorků. Program čte stavy pinů a ukládá je do výstupního FIFO v nekonečné smyčce. Program navíc kontroluje, zda se objevila hodnota ve vstupním FIFO. Pokud se ve FIFO objevila hodnota, je přenesena do registru `osr` pomocí instrukce `pull noblock`. Registr `osr` je zkopírován do interního registru `x` a program skáče do odpočítávající smyčky. Zde se každý cyklus krom čtení vstupů snižuje `x` o 1. Pokud registr `x` dosáhne 0 je program ukončen a vyvolá se přerušení procesoru. Program je dále znázorněn pomocí vývojového diagramu na obr. 6.4.

Díky programu ve výpisu 6.4 stačí, aby procesor vložil do FIFO registru počet zbývajících vzorků a stavový automat se po tomto počtu sám ukončí. Stavový automat navíc vyvolá přerušení pomocí instrukce `irq nowait 0`, aby procesor poznal, že je vzorkování dokončeno. Program čte hodnotu vstupů každou 5 instrukci a maximální dosažitelná vzorkovací frekvence je tedy $125\text{MHz}/5 = 25\text{MHz}$. Aby bylo čtení každých 5 instrukcí zajištěno pro nekonečnou i odečítající smyčku, je použita instrukce `nop [2]`, která nic neprovede a zabere přesně tři hodinové cykly. Po napsání a otestování tohoto programu byla snaha program vylepšit, aby bylo možné dosáhnout vyšší vzorkovací frekvence.

```
.program pins_input
.define public PIN_COUNT 8

    set x, 0
start:
    pull noblock
    mov x, osr
    jmp !x infinite
finite:
    in pins, PIN_COUNT
    nop [2]
    jmp x-- finite
    jmp end
infinite:
    in pins, PIN_COUNT
    jmp start
end:
    irq nowait 0
end_loop:
    jmp end_loop
```

Výpis 6.4: PIO program pro čtení digitálních vstupů každou 5. instrukci a ukončení čtení



Obr 6.4: Vývojový diagram PIO programu s čtením pinů po 5 instrukci

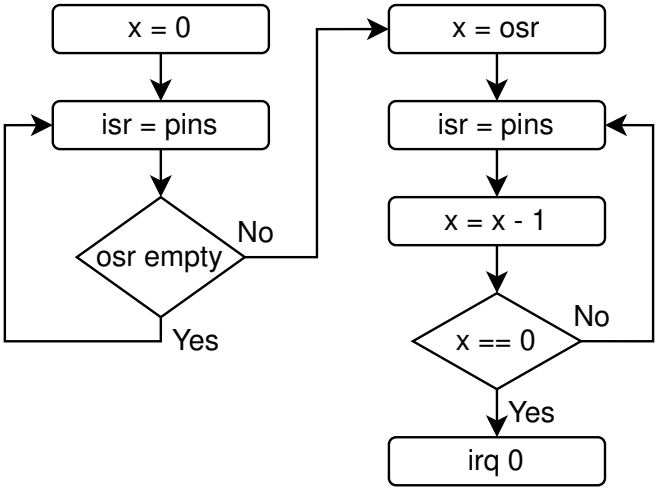
6.3.3 Rychlejší čtení vstupů a ukončení čtení pomocí PIO

Podobně jako v případě zápisu do registru `isr`, je možné i pro registr `osr` nastavit automatický přenos hodnoty z FIFO. Toho bylo využito v programu znázorněném ve výpisu 6.5. V nekonečné smyčce se v tomto programu kontroluje pouze to, zda registr `osr` obsahuje novou hodnotu, která se automaticky přenesla ze vstupního FIFO. Pokud hodnotu obsahuje, je zkopírována do registru `x` a začne odečítající smyčka. Díky automatickému přenosu ze vstupního FIFO bylo možné zařídit čtení vstupů každou 3 instrukci bez změny funkcí programu. Díky tomu je možné dosáhnout vzorkovací frekvence odpovídající 125MHz/3 ≈ 41.667MHz. Vhodnější by ovšem byla frekvence bez desetinné čárky a systémová frekvence byla proto snížena na 120MHz. Maximální vzorkovací frekvence tedy 125MHz/3 = 40MHz. Funkce programu je opět znázorněna pomocí vývojového diagramu na obr. 6.5.

```
.program pins_input
.define public PIN_COUNT 8

.wrap_target
    set x, 0
start:
    in pins, PIN_COUNT
    jmp !osre finite
    jmp start
finite:
    mov x, osr
finite_loop:
    in pins, PIN_COUNT
    nop
    jmp x-- finite_loop
    irq nowait 0
end_loop:
    jmp end_loop
.wrap
```

Výpis 6.5: PIO program pro čtení digitálních vstupů každou 3. instrukci a ukončení čtení



Obr 6.5: Vývojový diagram PIO programu s čtením pinů po 3 instrukci

6.3.4 Další zrychlení programu pro čtení vstupů pomocí PIO

Stejně jako ostatní PIO instrukce i instrukce `jmp` zabere právě 1 hodinový cyklus. Každý stavový automat ovšem nabízí možnost nastavit jedno místo jako `.wrap`, který funguje stejně jako funkce `jmp`, ale nezabírá žádný hodinový cyklus. Místo na které `.wrap` skáče se označuje pomocí `.wrap_target`. Pokud tedy v nekonečné smyčce programu nahradíme instrukci `jmp` funkcí `.wrap`, dosáhneme čtení pinů každou 2. instrukci. Funkce programu je v tomto případě stejná jako v programu 6.5, krom přidaného čtení vstupů před zkopírováním `osr` do registru `x`. To slouží pro splnění čtení vstupů každou 2. instrukci.

Dosažitelná vzorkovací frekvence je se sníženou systémovou frekvencí a novým programem 125MHz/2 = 60MHz. Při testování nového programu ovšem nastal problém, kdy při použití trigger chyběly v záznamu 2 vzorky. Příčina tohoto problému nebyla nalezena a pro logický analyzátor byl proto použit program, který čte vstupy každé 3 instrukce.

```
.program pins_input
.define public PIN_COUNT 8

    set x, 0
.wrap_target
    in pins, PIN_COUNT
    jmp !osre finite
.wrap
finite:
    in pins, PIN_COUNT
    mov x, osr
finite_loop:
    in pins, PIN_COUNT
    jmp x-- finite_loop
    irq nowait 0
end_loop:
    jmp end_loop
```

Výpis 6.6: PIO program pro čtení digitálních vstupů každou 2. instrukci a ukončení čtení

6.4 Reset PIO programu

Po úspěšném dokončení vzorkování se vytvořený PIO program zacyklí a přestane provádět další operace. Před započetím dalšího vzorkování je nutné použitý stavový automat resetovat, aby nezbývaly ve FIFO staré vzorky nebo v registrech staré hodnoty. To se provede pomocí funkcí z PicoSDK na počátku výpisu 6.7. Funkce pro resetování ovšem neresetují instrukci, na které stavový automat začíná. Po spuštění bude stavový automat začínat na poslední instrukci a v tomto případě bude tedy zacyklený. Díky funkci v PicoSDK je možné vykonat instrukci ve stavovém automatu, čehož lze využít pro vykonání instrukce `jmp` jako na konci výpisu 6.7. Proměnná `_pio_offset` obsahuje v tomto případě pozici první instrukce, která je získána při inicializaci PIO programu.

```
pio_sm_clear_fifos(_pio, _sm);
pio_sm_restart(_pio, _sm);

pio_sm_exec(_pio, _sm, pio_encode_jmp(_pio_offset));
```

Výpis 6.7: Resetování PIO stavového automatu

6.5 Trigger při hraně na vstupu analyzátoru

Při použití logického analyzátoru na analýzu PWM a podobných signálů, které jsou neustále přítomné, je možné spustit vzorkování okamžitě a signál bude odchycen. Některé signály se ovšem objevují pouze s vysokou periodou, nebo nepravidelně. To může být například komunikace s digitálními senzory každou vteřinu, odeslání dat na základě externí události apod. V těchto případech by bylo vhodné mít možnost nastavit logický analyzátor, aby vzorkoval signál až poté, co se objeví. To lze zařídit pomocí trigger, který reaguje na hranu na vstupu.

Trigger pro logický analyzátor byl zařízen pomocí přerušení procesoru od GPIO. Přerušení je možné nastavit při náběžné, sestupné nebo obou hranách na vstupu pinu. Vzorkování je nejdříve nastaveno do cyklického režimu, kdy se přepisují staré vzorky. Pokud nastane přerušení, je vyvolána funkce, která uloží aktuální adresu, kam DMA kanál zapisuje vzorek a do FIFO vzorkovacího stavového automatu je vložen zbývající počet vzorků. Adresa, kam DMA zapisovalo v době přerušení odpovídá vzorku, při kterém nastal trigger.

S vysokou rychlostí vzorkování, jako je 40MHz, může nastat situace, kdy adresa vzorku kam zapisuje DMA již neodpovídá vzorku, kdy nastal trigger. To znamená, že se i konec vzorkování bude později, než se očekáváno. Datové pole, do kterého se vzorky ukládají je z toho důvodu větší, než je maximální počet vzorků. Tím je vytvořena rezerva pro vzorky navíc. Místo, kde nastal trigger bude ovšem špatně zobrazeno. Byla proto vytvořena funkce, která na základě nastavení trigger a uložené DMA adresy zpětně hledá správné místo, kde trigger nastal. Tato funkce je vyvolána po každém ukončení vzorkování s nastaveným trigger.

Kapitola 7

Prověření možné implementace čítače s Raspberry Pi Pico

Osciloskop a logický analyzátor nejsou jediné přístroje, které je potencionálně s Pico možné vytvořit. Dalším přístrojem, který by mohl být sestrojen s Pico je čítač, pomocí kterého lze měřit frekvenci obdélníkového signálu, střídu PWM apod. V této kapitole budou prozkoumány možnosti využití periferií mikrořadiče RP2040 pro vytvoření takového čítače.

7.1 Použití vstupního kanálů PWM čítače

Jak již bylo zmíněno v kapitole 3.3 PWM čítače v RP2040 mají 2 kanály A a B. Kanál B lze nastavit jako vstupní, a to v režimech:

- čítač běží, pokud je vstup kanálu B ve stavu High,
- čítač jednou přičte, pokud na vstupu kanálu B nastane náběžná hrana,
- čítač jednou přičte, pokud na vstupu kanálu B nastane sestupná hrana.

Pomocí režimu přičítání při hraně na vstupu kanálu lze měřit frekvenci obdélníkového signálu. Pokud pravidelně, se známou periodou, odečteme hodnotu čítače v tomto režimu, dokážeme spočítat počet hran na vstupu za periodu a tím frekvenci signálu. Frekvence vstupu f_{sig} se v tomto případě spočítá pomocí vzorce 7.1, kde N_h je počet hran od posledního odečtení a T_o je perioda odečítání hodnoty čítače. Pravidelné odečítání hodnoty čítače je možné zařídit pomocí procesoru, ale přesněji také pomocí DMA kanálu, jehož periodu odečítání je možné nastavit čítači DMA řadiče nebo dalším PWM čítačem.

$$f_{sig} = \frac{N_h}{T_o} \tag{7.1}$$

Střídu PWM signálu na vstupu lze měřit pomocí režimu, kdy PWM čítač počítá, pokud je vstup kanálu B ve stavu High. Čítač v tomto režimu necháme počítat po známou dobu T_p a po této době čítač zastavíme. Střídu signálu spočítáme pomocí vzorce 7.2, kde D je střída v procentech, N_c je hodnota čítače a f_c je hodinový signál čítače. Dělitel $f_c T_p$ odpovídá hodnotě čítače, pokud vstup kanálu B zůstane High. Hodinový signál čítače závisí na nastavení systémového hodinového signálu a děličky PWM čítače.

$$D = \frac{N_c}{f_c T_p} 100 \tag{7.2}$$

7.2 Čítání pomocí PIO

PIO je programovatelná periferie zaměřená na práci s GPIO. V kapitole 6.3 bylo PIO využito pro logický analyzátor na vzorkování digitálních vstupů a ukončení vzorkování po nastaveném počtu vzorků. V programu pro stavový automat v PIO je možné také kontrolovat stav vstupního pinu pomocí instrukce `wait`. Tato instrukce zastaví běh programu, dokud nenastane daná podmínka. Například instrukce `wait 1 gpio 5` bude čekat, dokud pin 5 nebude ve stavu high. Pomocí instrukce `jmp x--` je možné odečítat hodnotu registru `x`. S pomocí těchto 2 instrukcí je možné dosáhnout režimu, kdy se registr `x` odečítá, pokud je pin 5 ve stavu high. Je tedy možné dosáhnout podobné funkce jako se vstupním kanálem PWM čítače.

Před započetím odečítání je potřeba nastavit registr `x` na hodnotu, od které se bude odečítat. Toho lze dosáhnout pomocí instrukce `set`, která ovšem podporuje pouze 5bitové hodnoty. Výhodné ovšem může být nastavit registr na maximální možnou hodnotu, čehož lze dosáhnout instrukcemi ve výpisu 7.2. Registr `x` je nejdříve vynulován pomocí instrukce `set` a následně je registr negován pomocí instrukce `mov`. Tím byl registr nastaven na maximální hodnotu $2^{32} - 1$.

```
set x, 0
mov x, ~x
```

Kapitola 8

Raspberry Pi Pico s Micropython

Raspberry Pi Pico (dále Pico) je nová deska od firmy Raspberry Pi, která je poháněná mikrořadičem RP2040. Na rozdíl od ostatních produktů od Raspberry Pi není deska dělaná na běh plnohodnotného operačního systému, nýbrž se programuje tzv. „bare metal“. Deska může sloužit jako „mozek“ různých elektronických projektů a je také vhodná pro výuku programování a elektroniky.

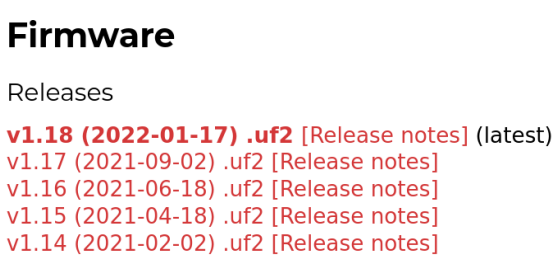
Jednou z výhod Pico je nízká cena a možnost programování pomocí interpretovaného jazyka Python, který je pro začátečníky jednodušší než kompilované jazyky jako C a C++. Tato kapitola slouží jako návod na připravení Raspberry Pi Pico pro programování pomocí jazyka Python a pro grafické programování pomocí projektu Bipes.

8.1 Nahrání firmware na Pico

Firmware je binární soubor, který si lze představit jako seznam instrukcí, které mikrořadič postupně vykonává. V případě jazyka Python potřebujeme na Pico nahrát firmware, schopný přečíst námi napsaný kód a vykonat ho. Jedním z takových firmwarů je Micropython. V této části si ukážeme, jak nahrát Micropython firmware na Pico, ale následující postup lze využít i pro jiný firmware napsaný pro Pico.

8.1.1 Stažení Micropython firmware

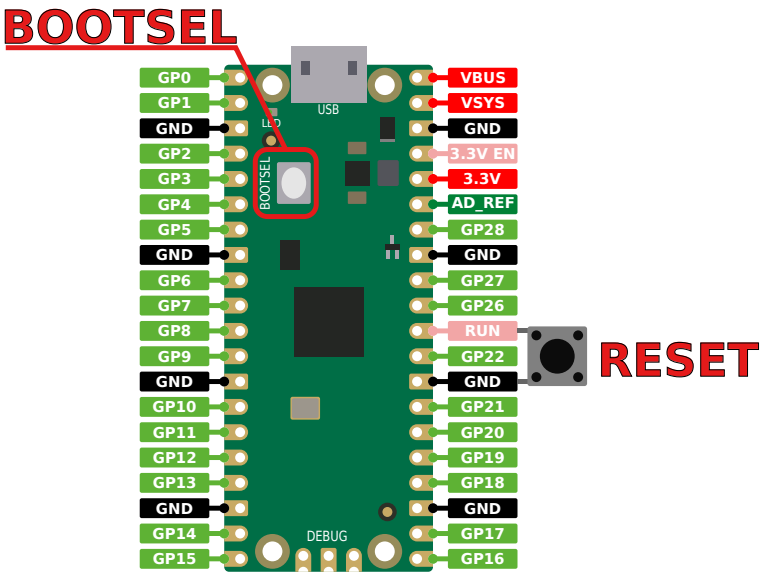
První krokem je stáhnout si připravený binární soubor ze stránek projektu Micropython. Ten nalezneme na adrese `https://micropython.org/download/rp2-pico/` ve spodní části stránky v kolonce **Releases** (obr. 8.1). Firmware stáhneme kliknutím na poslední verzi, která je v době psaní `v1.18`. Po kliknutí se stáhne soubor `.uf2`.



Obr 8.1: Stažení Micropython firmware pro Pico

8.1.2 Nahrání firmware na Pico z počítače

Nejjednodušší způsob, jakým nahrát `.uf2` firmware do Pico, je pomocí USB Mass Storage Device. Pico se v tomto případě po připojení USB kabelu zobrazí v počítači jako paměťové zařízení, do kterého lze nakopírovat zkompilovaný firmware podobně jako na Flash disk.



Obr 8.2: Tlačítko BOOTSEL a externí tlačítko pro reset

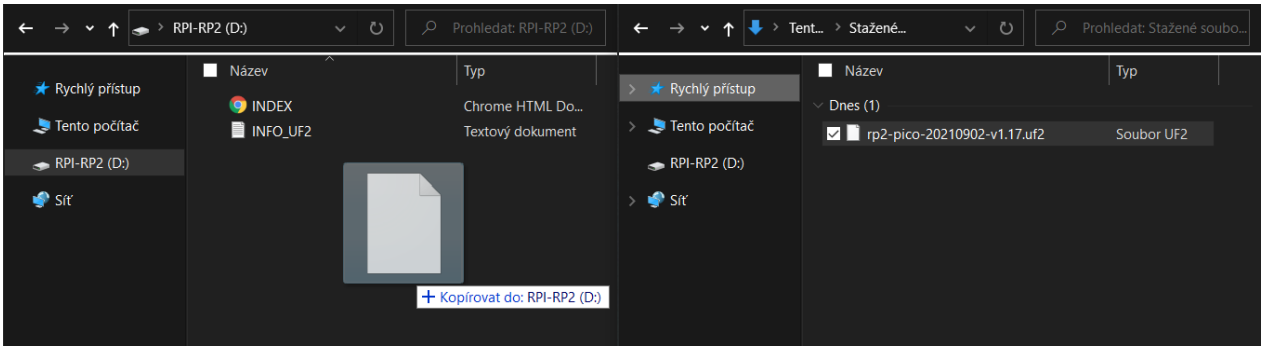
Aby se Pico zobrazilo jako paměťové zařízení, je nutné podržet tlačítko `BOOTSEL` a resetovat desku. Umístění tlačítka `BOOTSEL` je ukázáno na obr. 8.2.
Postup je následující:

1. Pokud je Pico zapojeno USB kabelem do počítače, odpojit USB kabel.
2. Pokud je Pico napájeno jiným způsobem, odpojit napájení.
3. Stisknout a držet tlačítko `BOOTSEL`.
4. Zapojit USB kabel.
5. Přestat držet tlačítko `BOOTSEL`.

Alternativně lze resetovat Pico pomocí externího tlačítka, které spojí Pin `RUN` s `GND`. Umístění externího tlačítka je znázorněno na obr. 8.2.
Postup je v tomto případě následující:

1. Spojit Pico s počítačem pomocí USB kabelu.
2. Spojit `RUN` s `GND` stisknutím externího tlačítka.
3. Stisknout a držet tlačítko `BOOTSEL`.
4. Přestat držet externí tlačítko.
5. Přestat držet tlačítko `BOOTSEL`.

Nyní by se v počítači mělo zobrazilo nové paměťové zařízení s názvem `RPI-RP2`. Ve Windows najdeme zařízení v prohlížeči souborů. Nyní můžeme do zařízení nakopírovat `.uf2` soubor jako na obr. 8.3. Po úspěšném nahrání by se mělo Pico samo resetovat a paměťové zařízení zmizet z počítače.



Obr 8.3: Kopírování `.uf2` souboru do Rapsberry Pi Pico ve Windows

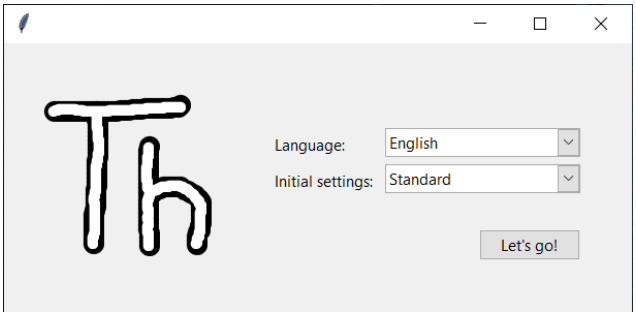
8.2 Programování Pico pomocí Python

Po úspěšném nahrání Micropython firmware je Pico připraveno na vykonávání Python kódu. V této části nainstalujeme program Thonny IDE a s jeho pomocí nahrajeme na Pico program blikající s LED na desce.

8.2.1 Příprava Thonny IDE

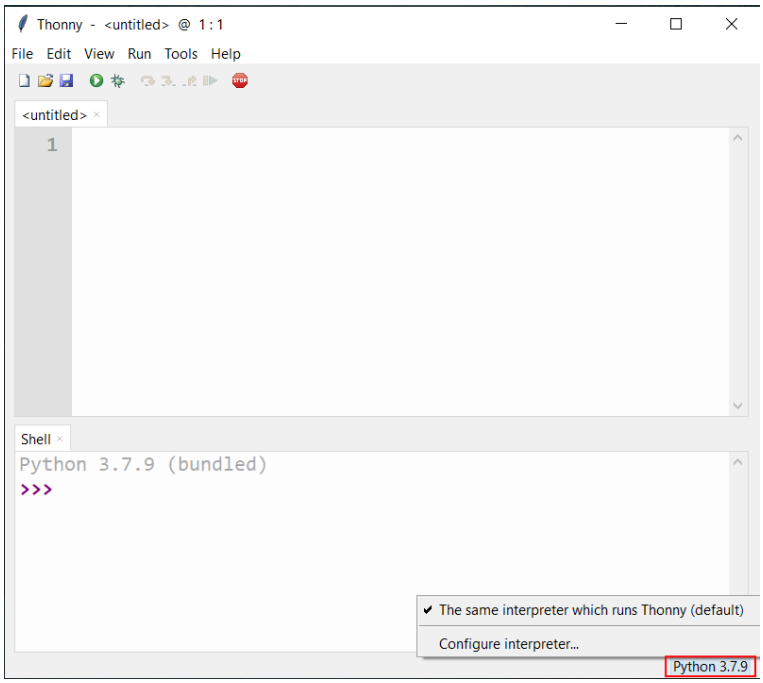
Thonny je možné získat na stránce `https://thonny.org/`. Pro Windows si stáhneme `.exe` instalátor a program nainstalujeme běžným způsobem. Při instalaci můžeme vše nechat na výchozích hodnotách a zaškrtneme políčko "Create desktop icon", pokud si přejeme ikonu na ploše.

Po prvním spuštění Thonny se zobrazí okno jako na obrázku 8.4. Zde si můžeme zvolit jazyk se kterým chceme Thonny používat a políčko `Initial Settings` necháme na `Standard`. Nyní můžeme kliknout na tlačítko `Lets Go!`.



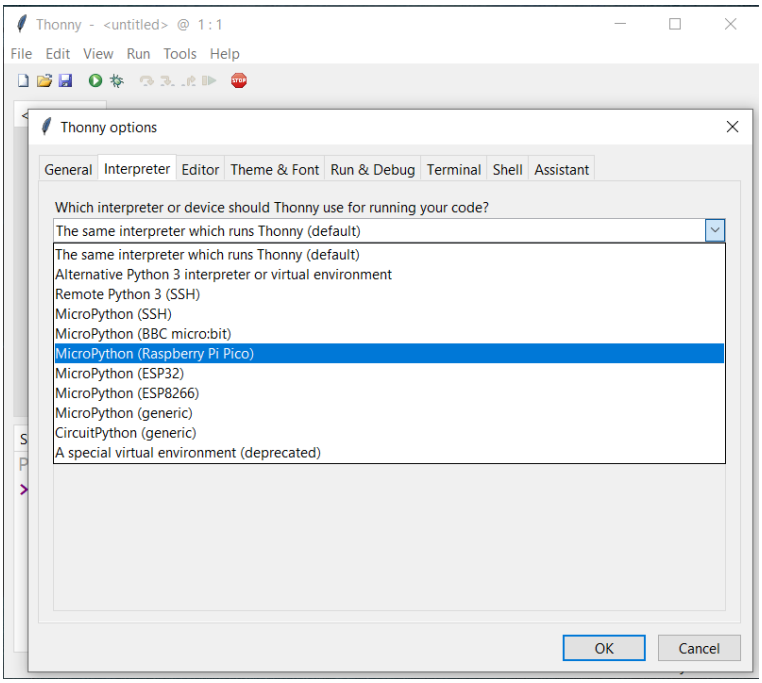
Obr 8.4: První spuštění Thonny IDE

Po zvolení prvního nastavení se nám zobrazí hlavní okno Thonny IDE. Pro spojení s Pico je potřeba zvolit, že chceme pracovat s Micropython. V pravém dolním rohu klikneme na nápis `Python 3.x.x` jako na obrázku 8.5.



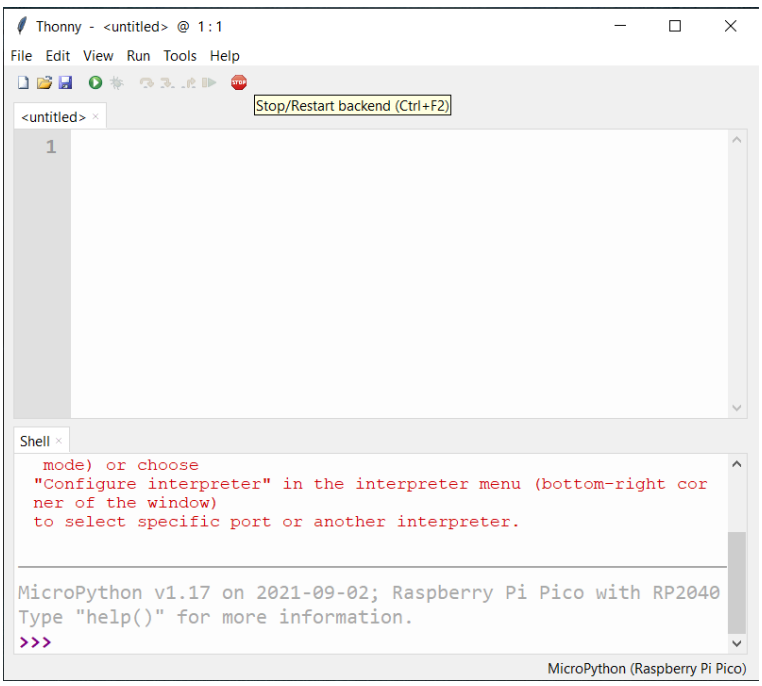
Obr 8.5: Menu pro výběr Python Interpreter

V nově zobrazeném menu vybereme možnost `MicroPython (Raspberry Pi Pico)`. Pokud se tato možnost nezobrazí, zvolíme `Configure interpreter...` . Nyní vidíme rolovací seznam, na který když klikneme, můžeme vybrat `MicroPython (Raspberry Pi Pico)` jako na obrázku 8.6 a následně kliknout na `OK`.



Obr 8.6: Výběr MicroPython (Raspberry Pi Pico) v nastavení

Thonny by se mělo spojit s Pico a ve spodní části Thonny bychom měli vidět `MicroPython` konzoli jako na obrázku 8.7. Pokud se zde objevila červená chybová zpráva, je možné, že je potřeba desku resetovat. Desku odpojíme od USB, znovu zapojíme a v horní části Thonny stiskneme tlačítko `STOP`.



Obr 8.7: Thonny připojené k Raspberry Pi Pico

Do `MicroPython` konzole je nyní možné psát Python příkazy, které deska vykoná. Pro ověření můžeme například napsat příkaz

```
print("Hello World")
```

a Pico nám odpoví textem `Hello World`.

8.2.2 Blikací program s Python

Po úspěšném nastavení Thonny IDE a spojení s deskou Raspberry Pi Pico můžeme začít psát první Python program. Pro ukázku použijeme jednoduchý program 8.1 co bude pravidelně blikat s LED na destičce.

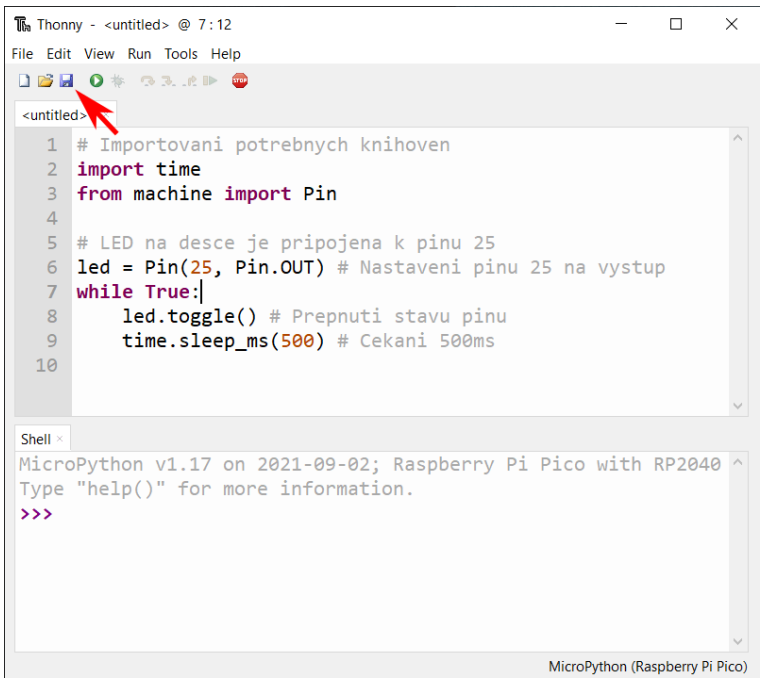
```
# Importovani potrebnych knihoven
import time
from machine import Pin

# LED na desce je pripojena k pinu 25
led = Pin(25, Pin.OUT) # Nastaveni pinu 25 na vystup

# Nekonecna smycka
while True:
    led.toggle() # Prepnuti stavu pinu
    time.sleep_ms(500) # Cekani 500ms
```

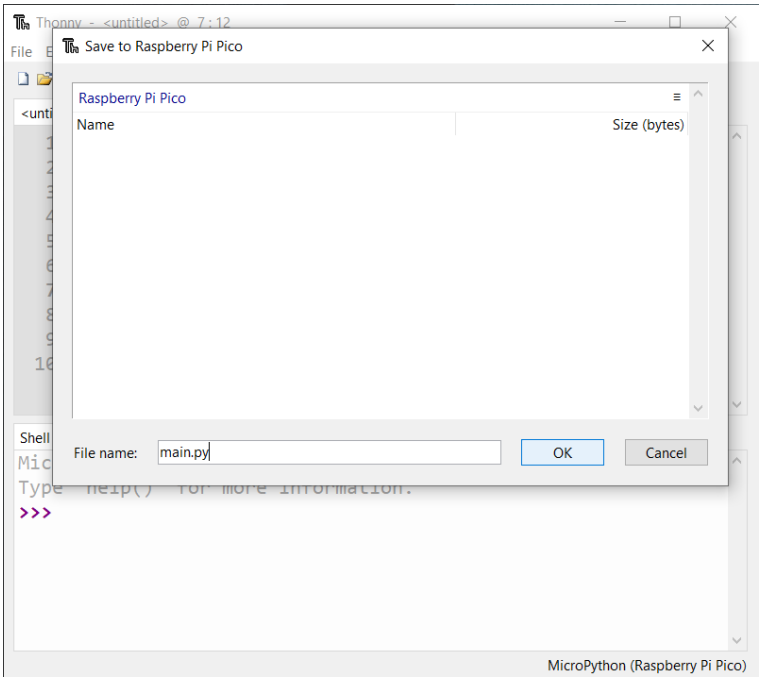
Výpis 8.1: Python blikací program

Tento program zkopírujeme do okénka Thonny jako na obrázku 8.8. Je nutné si pamatovat, že záleží na odtabování jednotlivých řádků.



Obr 8.8: Blikací program

Program nejdříve uložíme pomocí tlačítka `Save` nebo pomocí zkratky `Ctrl + S`. Thonny nám dá na výběr kam chceme program uložit. Vybereme možnost `Raspberry Pi Pico` a program uložíme pod názvem `main.py` jako na obrázku 8.9. Program nyní můžeme spustit pomocí zeleného tlačítka `Run current script`.



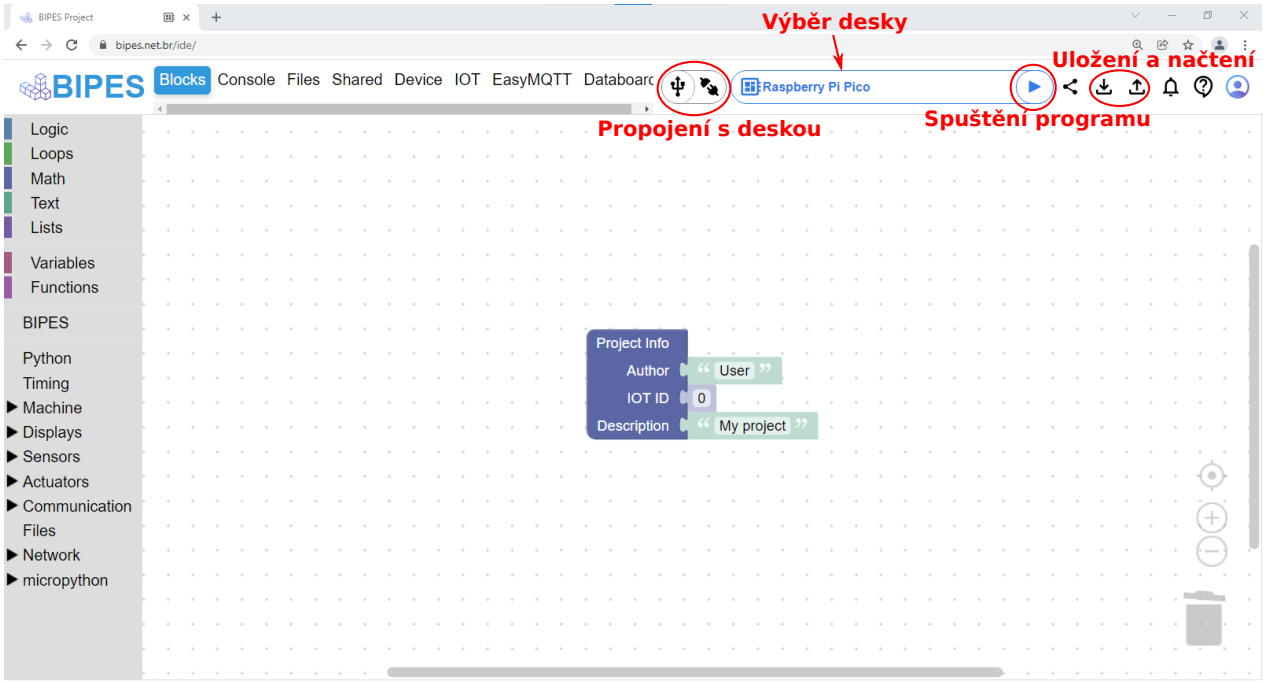
Obr 8.9: Ukládání programu na Raspberry Pi Pico

8.3 Programování graficky pomocí Bipes

Díky projektu Bipes je možné Pico programovat i pomocí spojování grafických bloků. Bipes za nás vygeneruje Python kód, který následně můžeme spustit na Pico. Vzhledem k tomu, že Bipes generuje Python kód, je potřeba nahrát na Pico Micropython firmware, pokud jste tak již neučinili. Nahrání Micropython firmware je popsáno v kapitole 8.1.

8.3.1 Představení Bipes

Bipes je webové rozhraní, které můžeme nalézt na stránce `https://bipes.net.br/ide/`. Po otevření stránky se dostáváme rovnou do prostředí, ve kterém budeme vytvářet programy. V levé straně stránky máme seznam, odkud budeme brát bloky programu. Ve vrchním panelu můžeme zvolit desku, na kterou chceme programovat, uložení aktuálního programu apod. Na obr. 8.10 je ukázka prostředí Bipes se základními popisky.

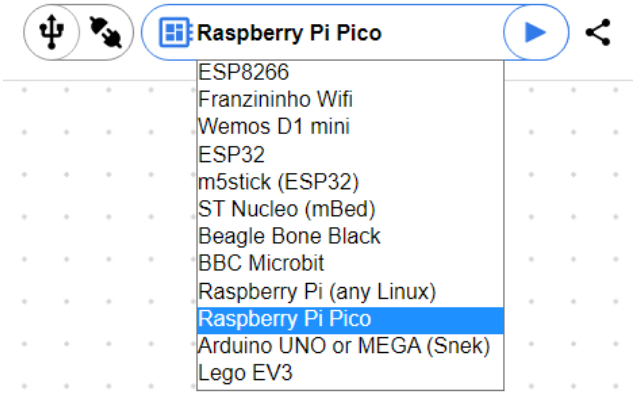


Obr 8.10: Prostředí Bipes

8.3.2 Spojení s Pico

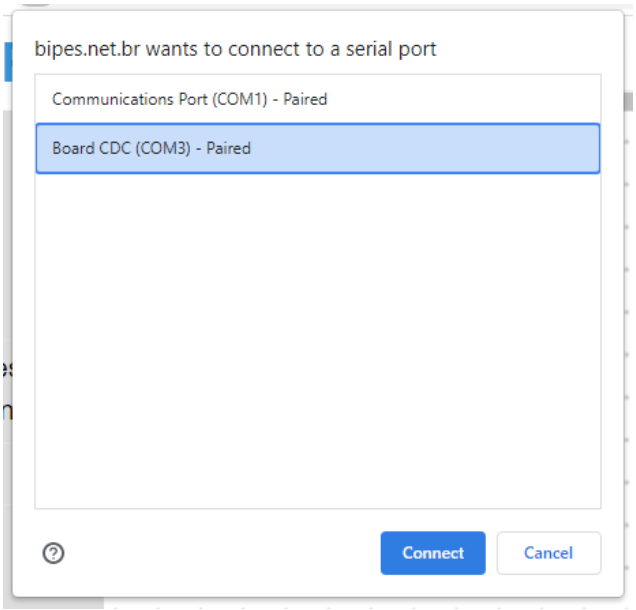
Pro vykonání námi vytvořených programů musíme nejdříve spojit Bipes s Pico. Toto v době psaní funguje pouze v prohlížečích založených na Chromium, jako je Google Chrome a nový Microsoft Edge.

Nejdříve vybereme, že chceme programovat desku Raspberry Pi Pico v pravém horním menu jako na obr. 8.11.



Obr 8.11: Nastavení desky

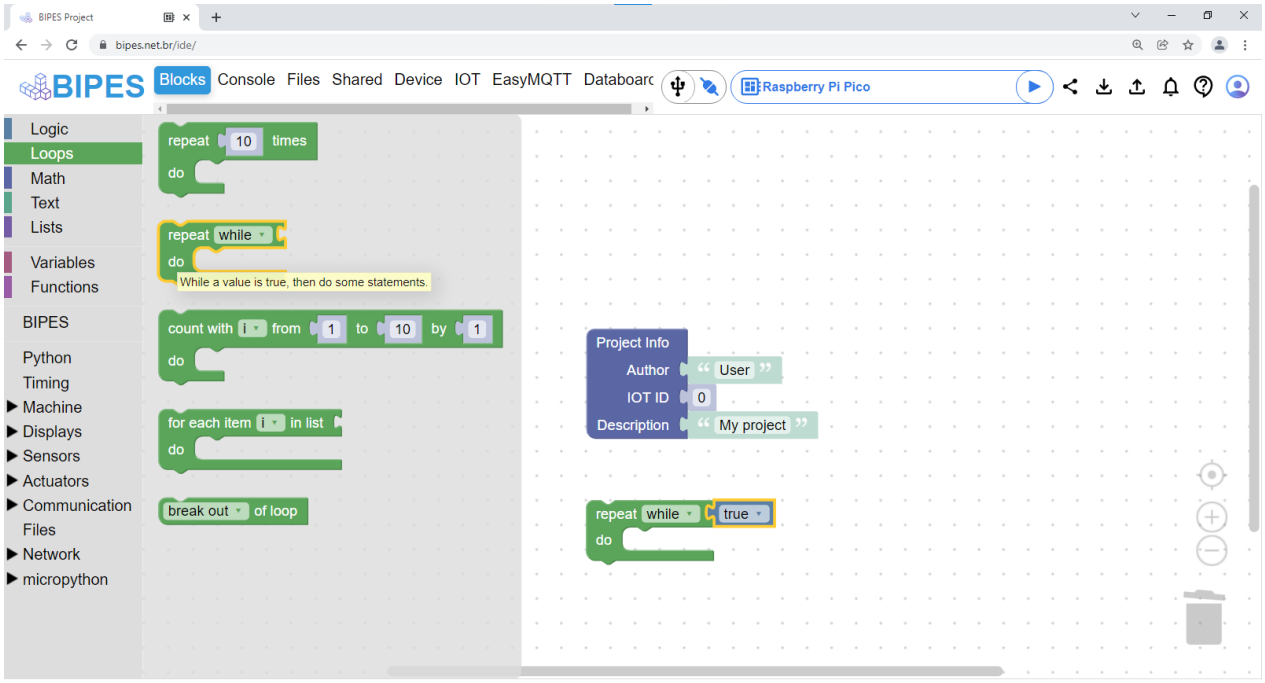
Před pokusem o připojení je vhodné se přesvědčit, že je zavřený program Thonny IDE. Pro spojení klikneme na obrázek rozpojené zásuvky. Po kliknutí se v prohlížeči objeví okénko se seznamem zařízení jako na obr. 8.12, ze kterých vybereme `Board CDC (COMx)` a klikneme na `Connect`.



Obr 8.12: Výběr zařízení pro spojení

8.3.3 Blikací program s Bipes

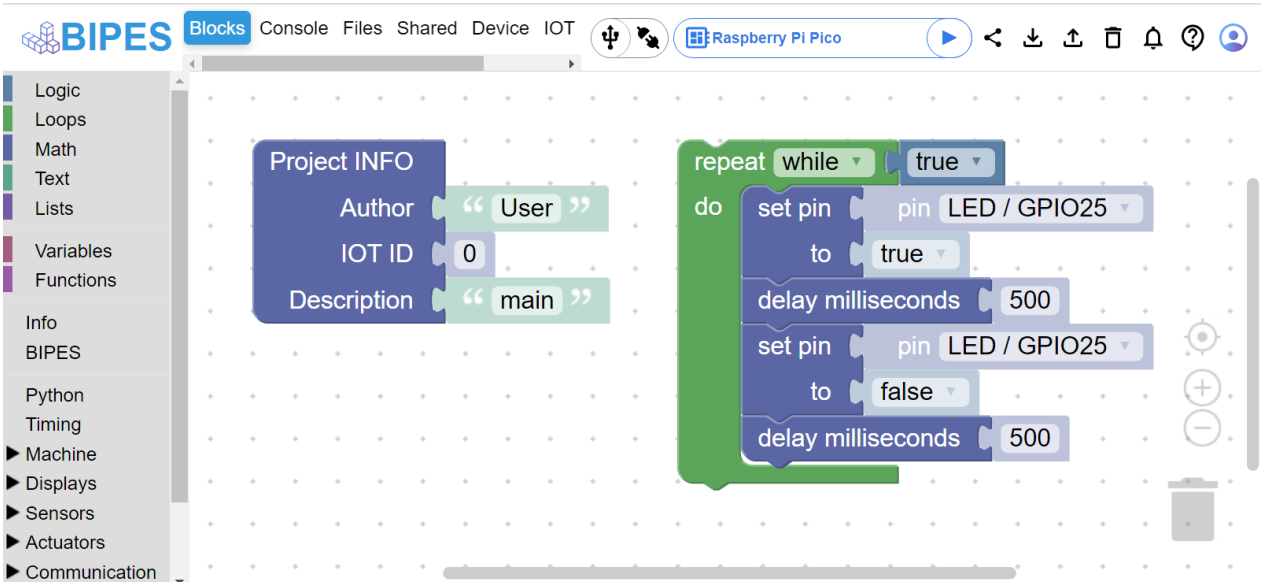
Pro vytvoření programu, který zabliká s LED na Pico musíme nejdříve vytvořit nekonečnou smyčku. Pro vytvoření nekonečné smyčky využijeme blok `repeat while`, který najdeme v kolonce **Loops**. Blok přidáme do našeho programu přetažením. K bloku navíc připojíme logickou hodnotu `true` z kolonky **Logic** jako na obr. 8.13.



Obr 8.13: Přidání bloku pro nekonečnou smyčku

Pro blikání s LED musíme přepínat pin 25 na Pico mezi hodnotami `true` a `false`. Na to najdeme blok `set output pin` v kolonce **Machine -> In/Out Pins**. Tyto bloky přidáme 2 a v jednom nastavíme `true` a v druhém `false`. V obou blocích vybereme pin `LED / GPIO25`. Dále mezi bloky pro přepínání pinu přidáme blok čekání `delay milliseconds`, který najdeme v kolonce **Timing**. Hodnotu v bloku nastavíme na 500. Program by měl nyní vypadat jako na obr. 8.14.
Program bychom nyní měli být schopni spustit tlačítkem v pravém horním rohu (obr. 8.10).

Pokud chceme program později uložit do Pico, aby se spouštěl sám, je potřeba v bloku `Project INFO` přepsat kolonku `Descrition` na `main`.

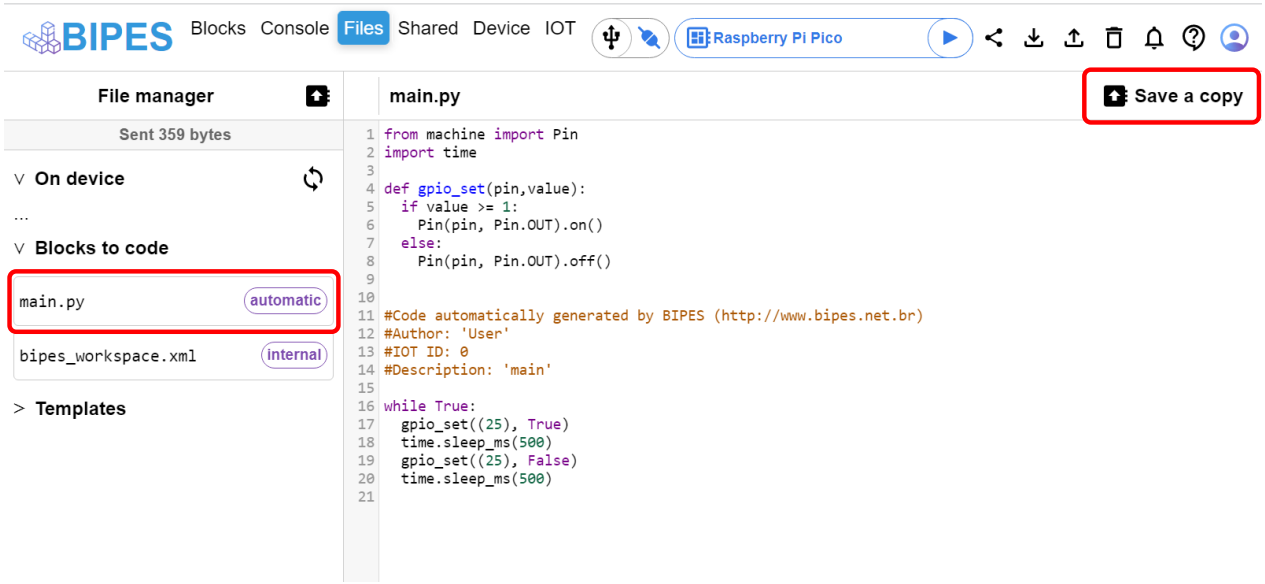


Obr 8.14: Blikací program

8.3.4 Uložení programu do Pico

V horní části stránky klikneme na kolonku **Files**, ve které najdeme Python kód vygenerovaný grafickými bloky. V levé části stránky otevřeme `main.py`, což nám ukáže vygenerovaný kód, jako na obr. 8.15. V pravém horním rohu nyní klikneme na `Save a~copy`, čímž nahrajeme soubor do Pico.

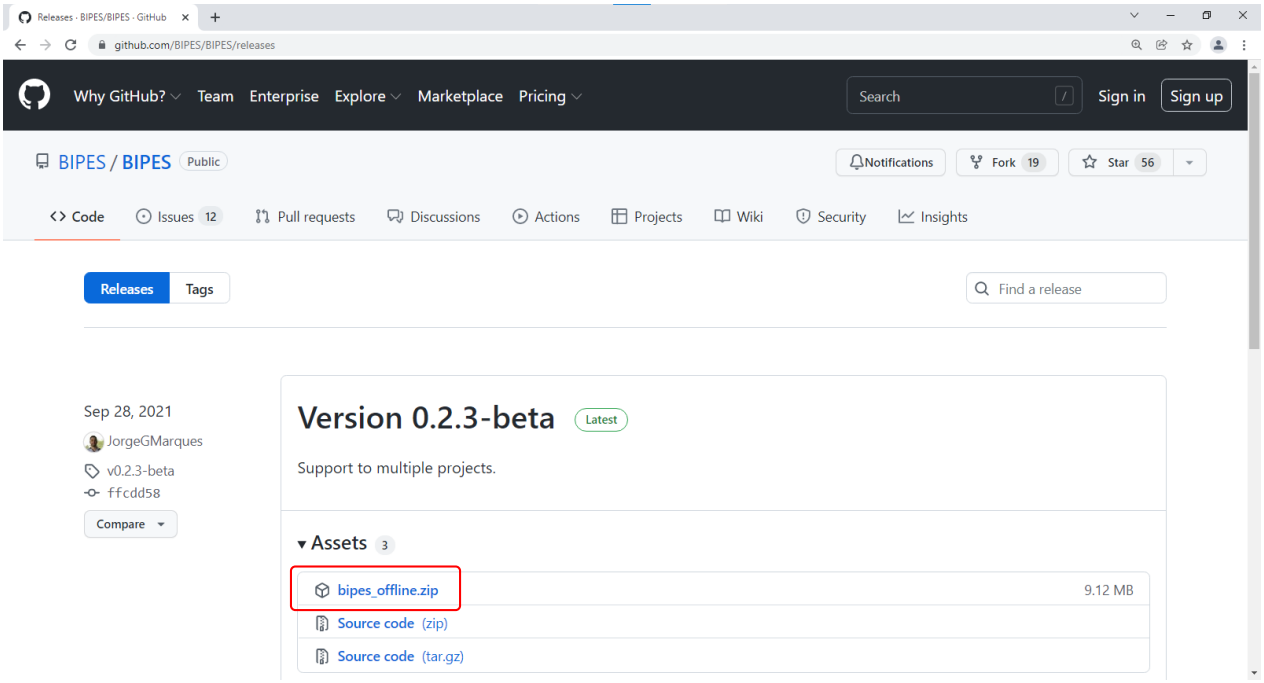
Soubor se musí jmenovat `main.py`, aby se spustil při spuštění pico. Pokud se soubor takto nejmenuje, je potřeba přepsat kolonku `Descrition` v bloku `Project INFO` jako na obr. 8.14.



Obr 8.15: Vygenerovaný Python kód

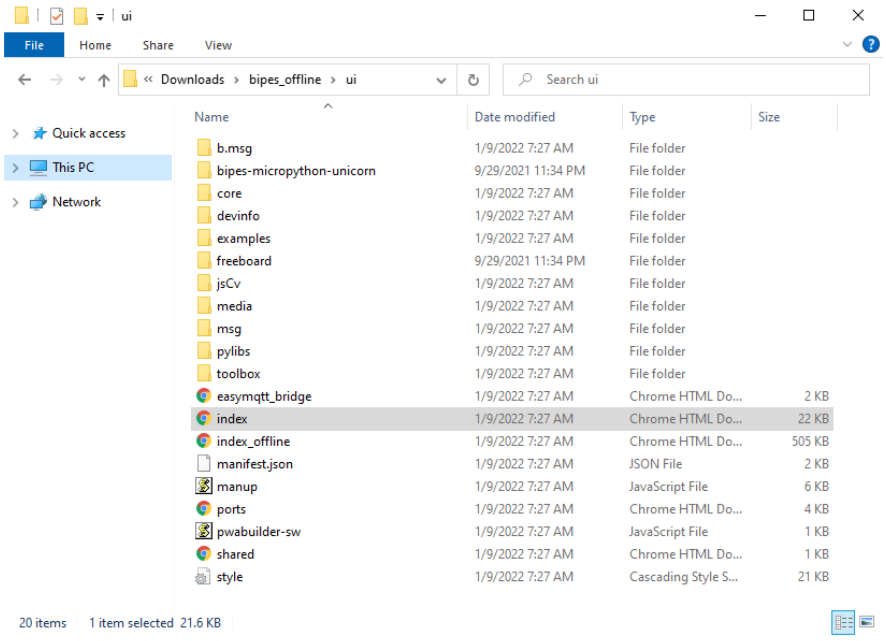
8.3.5 Bipes offline

Bipes je možné stáhnout do počítače a používat bez připojení k internetu. Stažení nalezneme na adrese `https://github.com/BIPES/BIPES/releases`. Odtud stáhneme soubor `bipes_offline.zip` (obr. 8.16), který v počítači rozbalíme do námi zvolené složky.



Obr 8.16: Stažení Bipes z Github

V rozbalené složce najdeme složku `ui` a v ní soubor `index_offline.html` (obr. 8.17). Po otevření tohoto souboru v prohlížeči se zobrazí stejné prostředí Bipes jako z webové stránky.



Obr 8.17: Spuštění Bipes offline

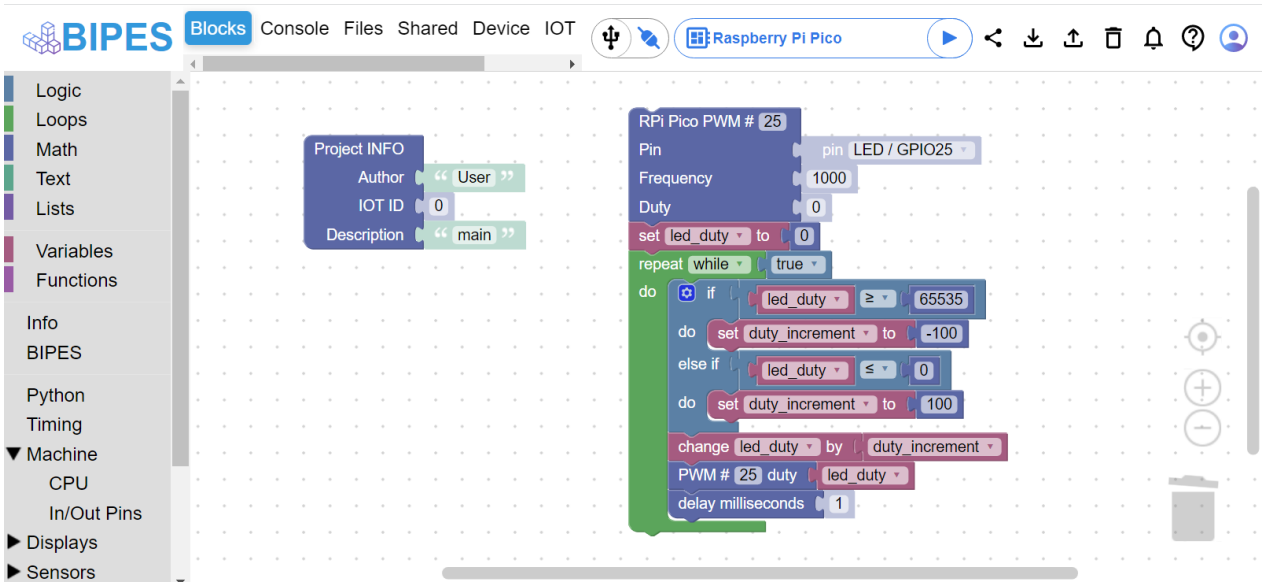
8.4 Příklady programů s micropython a bipes

V této části si ukážeme jednoduché programy v Micropython a Bipes.

8.4.1 Pulsování pomocí PWM

V tomto příkladě použijeme pulsně šířkovou modulaci, abychom plynule měnili jas LED na Pico. Na obr. 8.18 můžeme vidět implementaci v Bipes. Na počátku programu inicializujeme PWM pomocí bloku, který najdeme v **Machine>In/Out Pins**. Blokem nastavujeme pin, který bude použit, frekvenci v Hz a střídu, která se nastavuje mezi 0 - 65535. V kolonce **Variables** byly vytvořeny 2 proměnné `led_duty`, pomocí které se bude nastavovat střída a `duty_increment`, pomocí které nastavíme o kolik se střída změní každý cyklus.

Po inicializaci následuje nekonečná smyčka. Na počátku máme podmínku, která kontroluje, zda je střída větší nebo rovna 65535 (100% jas) nebo menší rovna 0 (0% jas). Pokud je jas maximální, změníme `duty_increment` na zápornou hodnotu, aby se jas začal snižovat. Pokud je jas minimální, změníme `duty_increment` na kladnou hodnotu, aby se jas začal zvyšovat. Po podmínce přečteme `duty_increment` k proměnné `led_duty` a následně zapíšeme střídu do PWM. V bloku pro nastavení střídy je nutno napsat správné číslo PWM, které vidíme v bloku kde PWM inicializujeme. Nakonec dáme zpoždění 1ms.



Obr 8.18: Pulsování s LED pomocí PWM (Bipes)

Ve výpisu 8.2 je napsána implemetace pulsování LED v micropython. Můžeme vidět že implementace je stejná jako v Bipes na obr. 8.18. Pro použití PWM musíme importovat PWM knihovnu, PWM inicializovat a uložit do proměnné `led_pwm = PWM(Pin(25))`. Nyní můžeme používat proměnnou `led_pwm` pro ovládání PWM na pinu 25.

```
from machine import Pin
from machine import PWM
import time

led_pwm = PWM(Pin(25))
led_pwm.freq(1000)
led_duty = 0
duty_increment = 0

while True:
  if led_duty >= 65535:
    duty_increment = -100
  elif led_duty <= 0:
    duty_increment = 100
  led_duty += duty_increment
  led_pwm.duty_u16(led_duty)
  time.sleep_ms(1)
```

Výpis 8.2: Pulsování s LED pomocí PWM (Micropython)

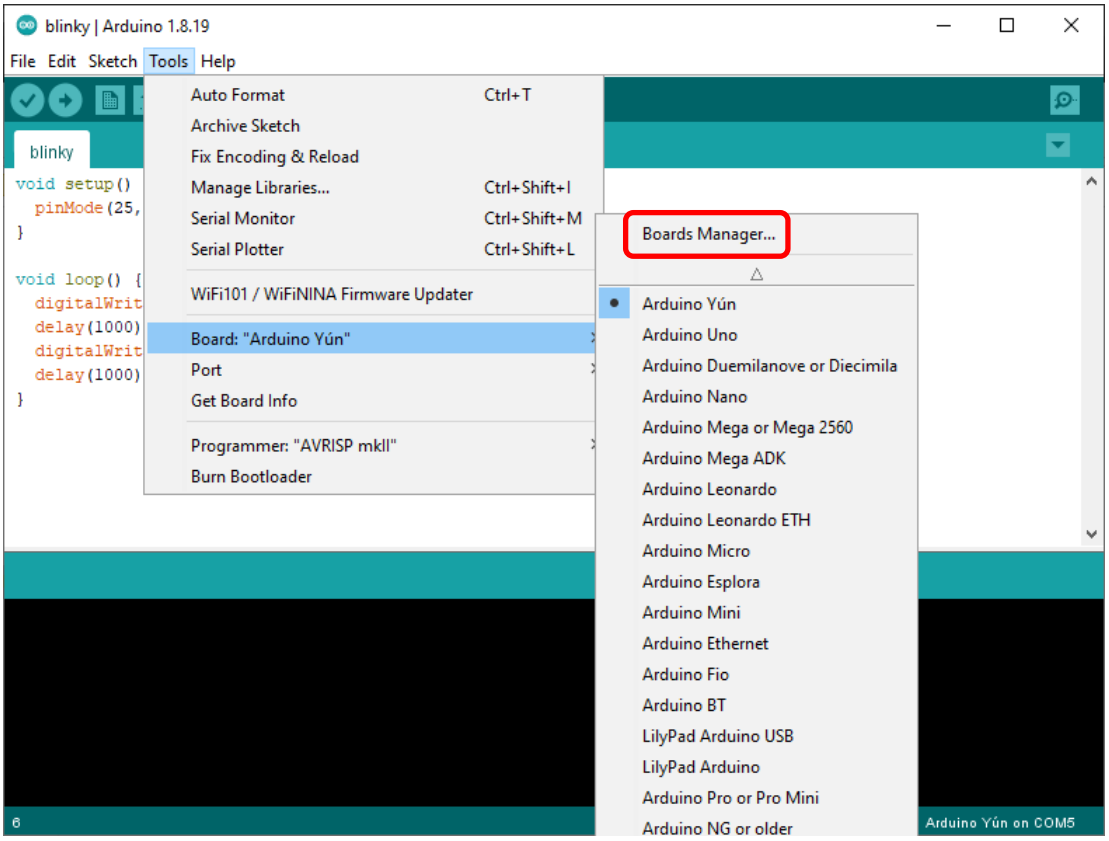
Kapitola 9

Raspberry Pi Pico s C/C++

Micropython a grafické programování jsou velice užitečné pro jednoduché projekty a pro začátečníky, co se učí s mikrořadiči. Pro pokročilejší programování mikrořadiče může být výhodnější využít kompilovaných jazyků jako je C a C++. S těmito jazyky můžeme vytvářet firmware, který lépe využívá hardware mikrořadiče a zabírá méně místa než python program s micropython firmware. V této kapitole se tedy zaměříme na programování Raspberry Pi Pico pomocí jazyku C++ s knihovnami Arduino a PicoSDK.

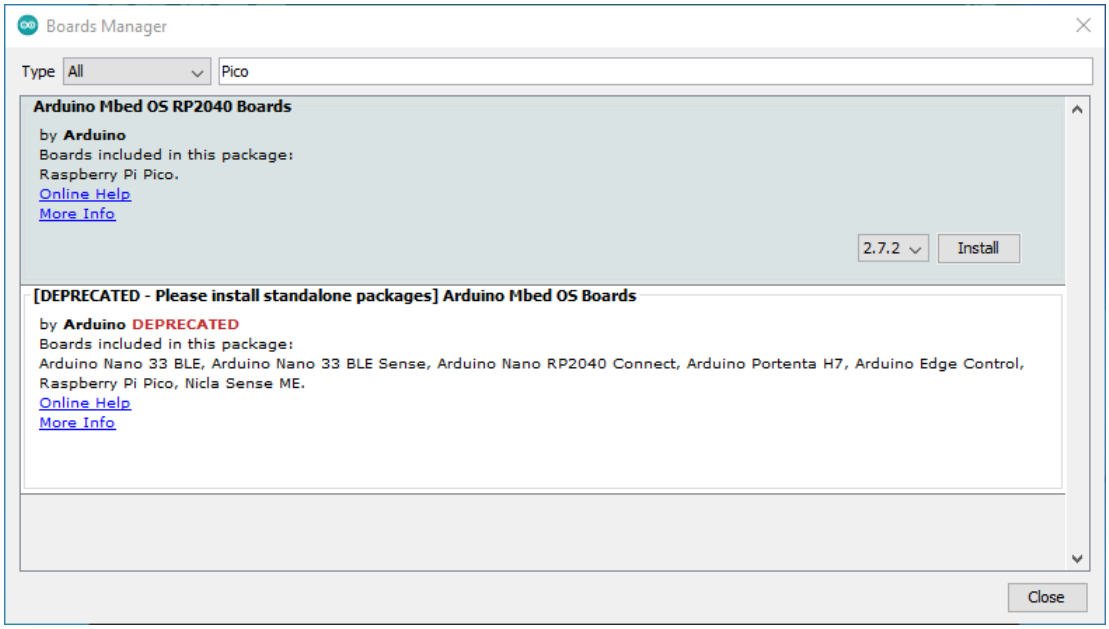
9.1 Programování Pico pomocí Arduino

Podobně jako vývojové desky Arduino je možné Pico programovat pomocí programu ArduinoIDE. Program získáme ze stránek `https://www.arduino.cc/en/software`. Po nainstalování je potřeba přidat podporu Pico pomocí `Tools > Board > Boards Manager`. Na obr. 9.1 je znázorněno, kde nalezneme `Boards Manager`.



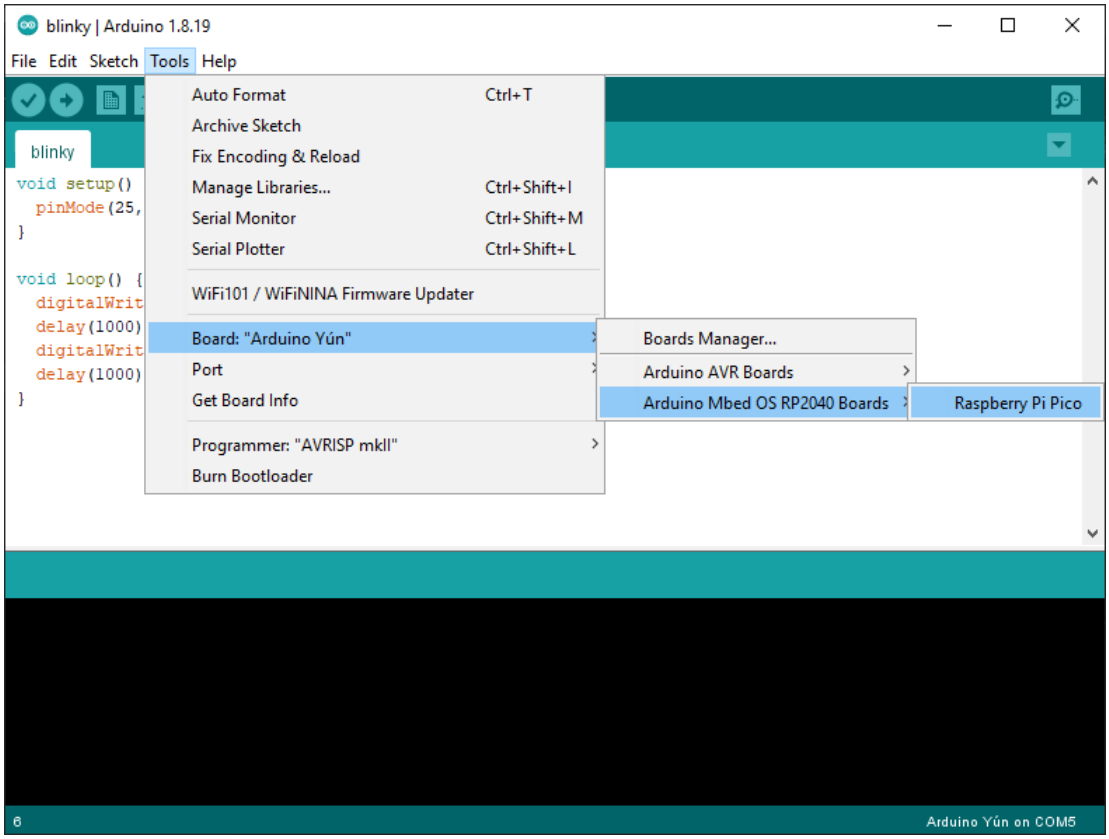
Obr 9.1: Otevření Boards Manager v ArduinoIDE

Po otevření napíšeme do vyhledávacího pole Pico a nainstalujeme první možnost jako na obr. 9.2.



Obr 9.2: Vyhledání Pico v Boards Manager

Po nainstalovaní můžeme v menu `Tools > Board` najít Raspberry Pi Pico jako na obr. 9.3. Dále vybereme správný COM port v menu `Tools > Port`.



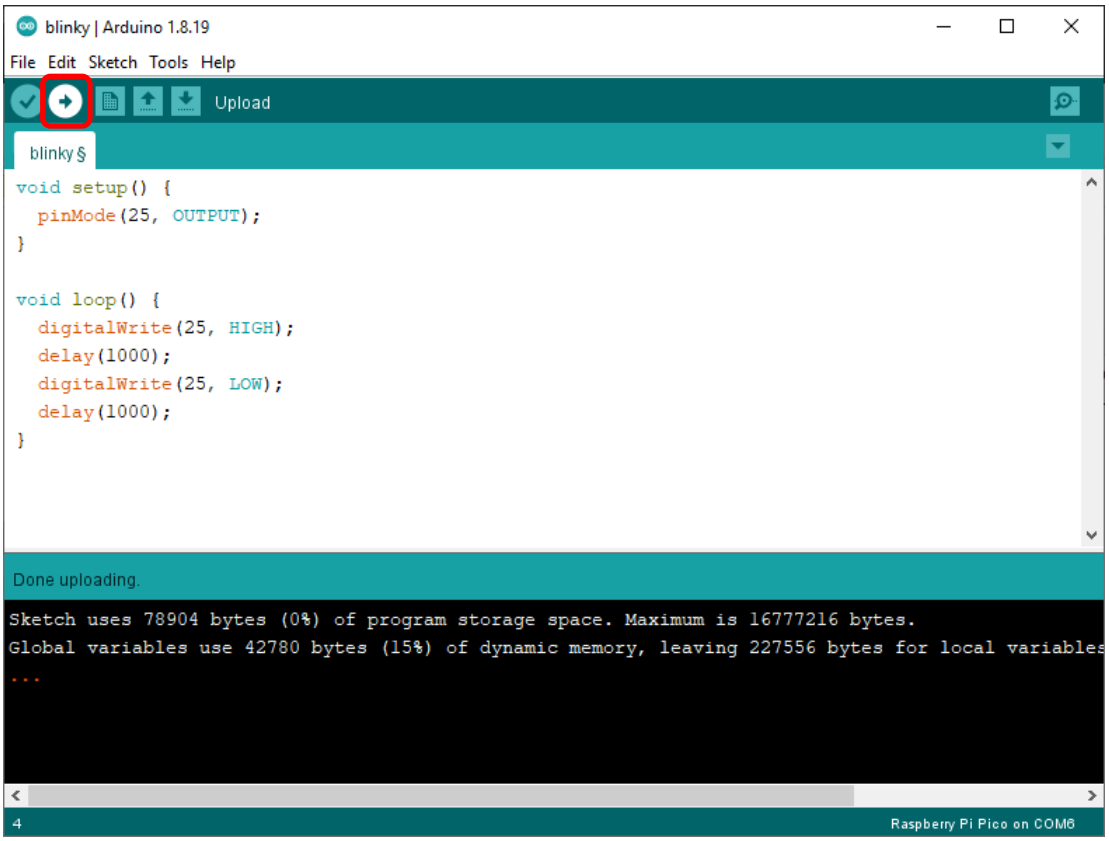
Obr 9.3: Výběr Pico v seznamu desek

Po výběru desky můžeme napsat jednoduchý program 9.1 na blikání s LED. Po napsání programu klikneme vlevo nahoře na tlačítko upload, jako na obr. 9.4. ArduinoIDE následně program zkompiluje a nahraje do Pico.

```
void setup() {
  pinMode(25, OUTPUT);
}

void loop() {
  digitalWrite(25, HIGH);
  delay(1000);
  digitalWrite(25, LOW);
  delay(1000);
}
```

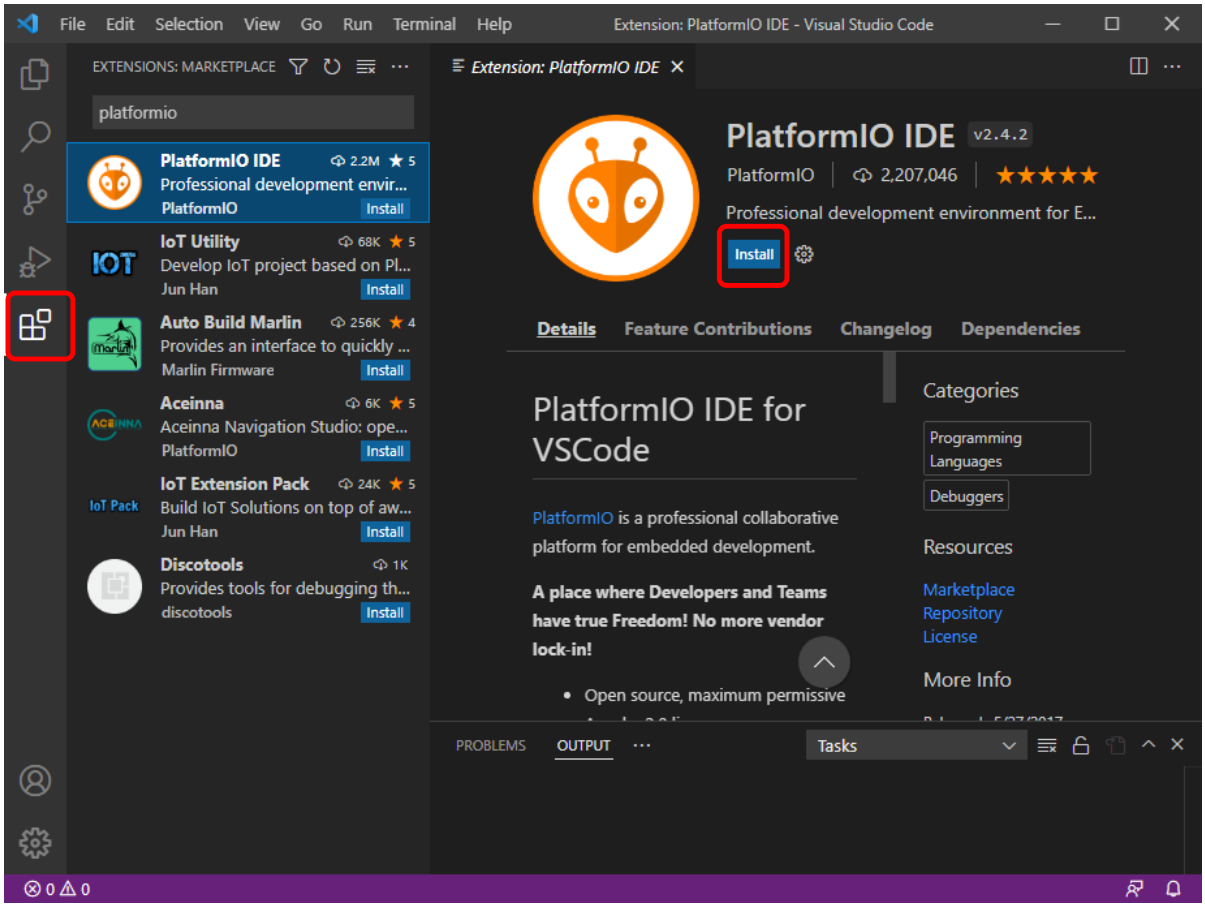
Výpis 9.1: Arduino blikací program



Obr 9.4: Blikací program v ArduinoIDE

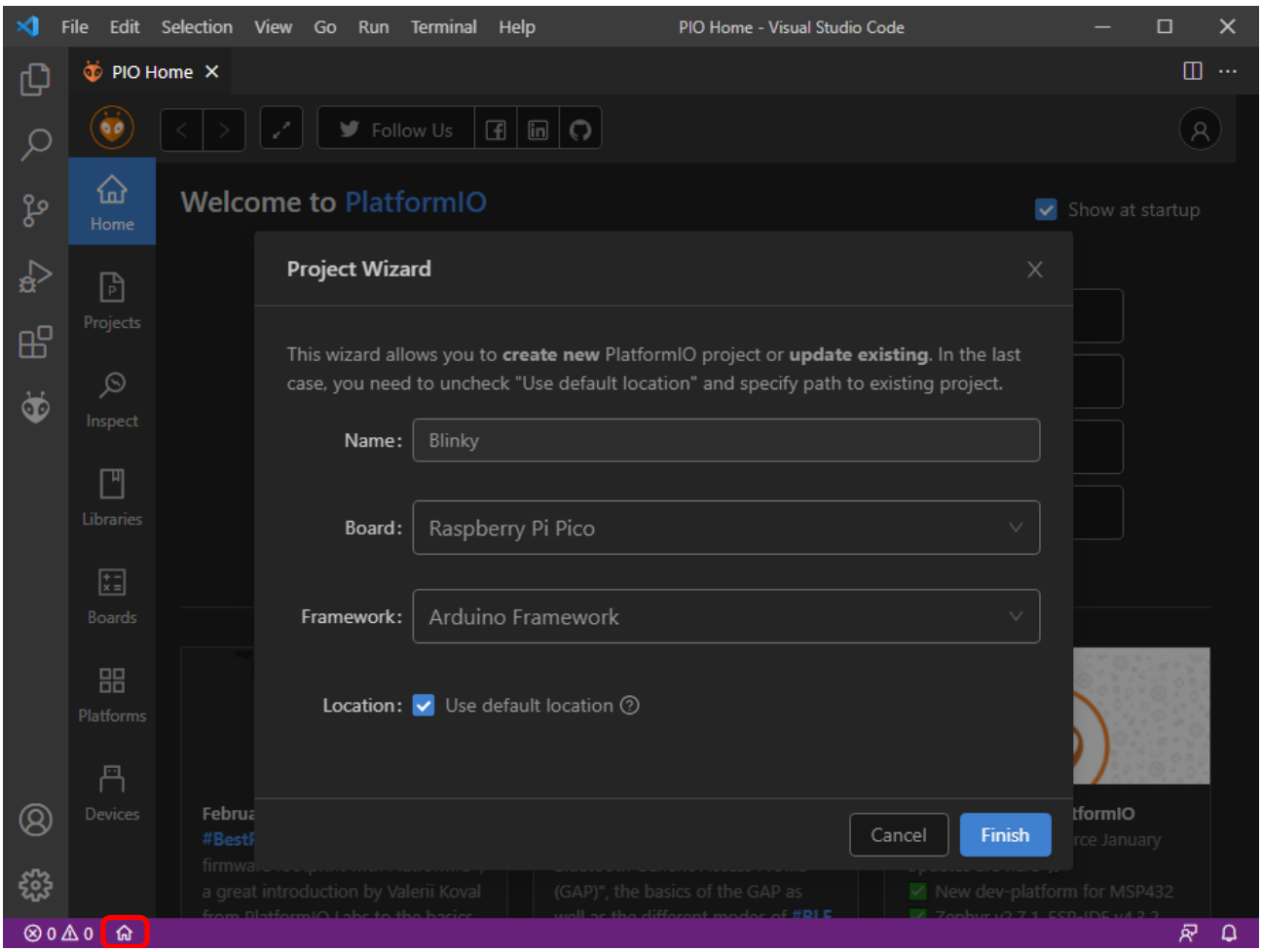
9.2 Programování Pico pomocí PlatformIO

PlatformIO je vývojové prostředí, ve kterém lze vytvářet firmware pro velké množství mikrořadičů pomocí platforem jako Arduino, Mbed, STMCube, apod. Lze v něm také programovat Raspberry Pi Pico pomocí Arduino knihoven. PlatformIO funguje jako rozšíření pro VSCode, které lze získat z `https://code.visualstudio.com/Download` . Po nainstalování VSCode otevřeme seznam rozšíření a vyhledáme a nainstalujeme PlatformIO jako na obr. 9.5.



Obr 9.5: Nalezení rozšíření PlatformIO ve VSCode

Po získání rozšíření PlatformIO nás VSCode požádá o restartování programu. V levém dolním rohu najdeme tlačítko na otevření PlatformIO Home. Zde klikneme na `New Project`, pojmenujeme projekt a vybereme desku Raspberry Pi Pico jako na obr. 9.5. Můžeme také nastavit umístění projektu nebo nechat výchozí (`Dokumenty/PlatformIO`). Po nastavení klikneme na tlačítko `Finish`. První založení projektu může trvat delší dobu, protože se stahují všechny potřebné soubory.



Obr 9.6: Založení projektu v PlatformIO

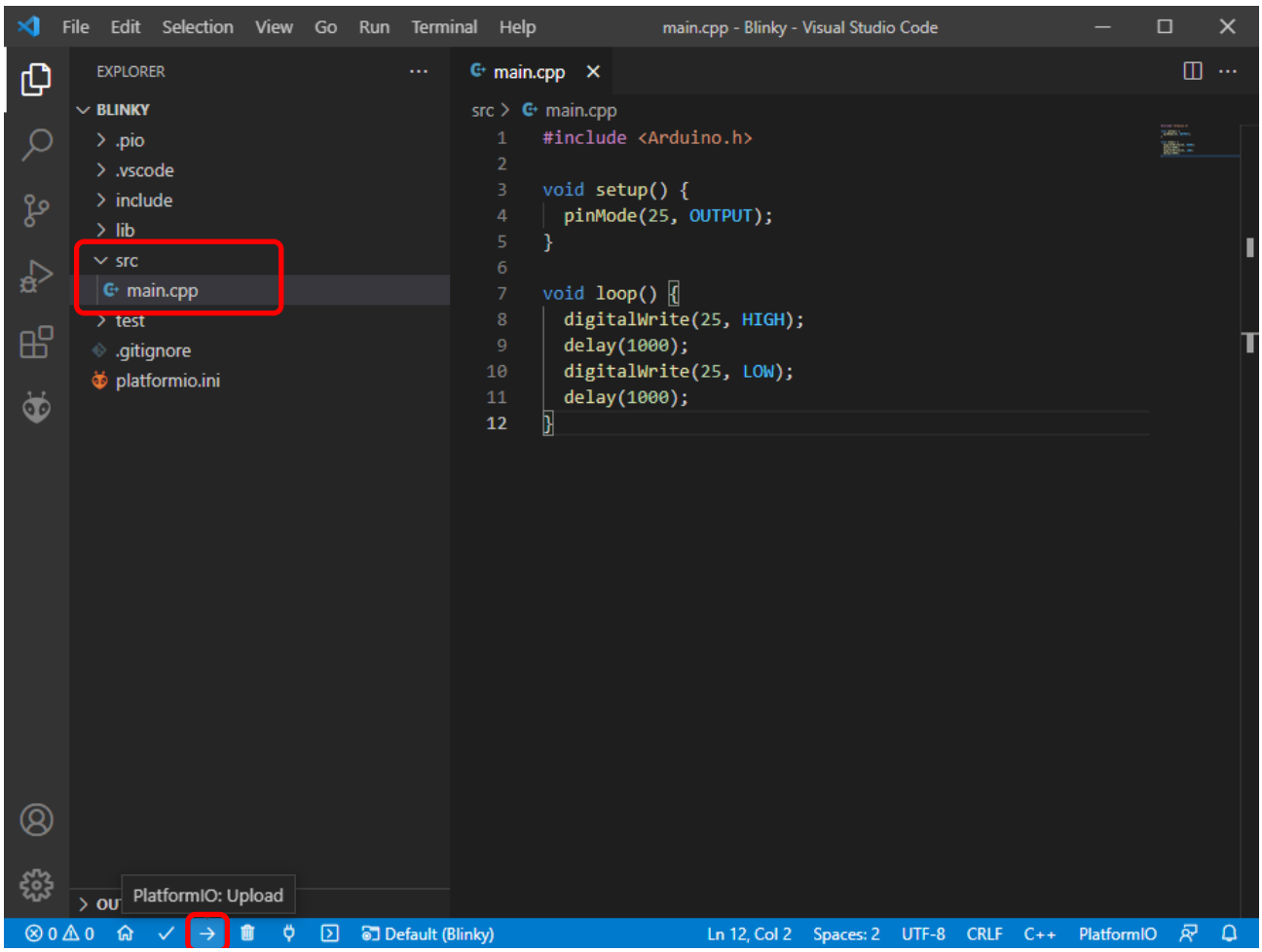
Po založení projektu můžeme otevřít soubor `src/main.cpp` a napsat blikací program 9.2 podobně jako v ArduinoIDE. Na rozdíl od ArduinoIDE však musíme na začátku programu importovat arduino knihovny. Po napsání můžeme program nahrát pomocí tlačítka v levém dolním rohu jako na obr. 9.7.

```
#include <Arduino.h>

void setup() {
  pinMode(25, OUTPUT);
  puts();
}

void loop() {
  digitalWrite(25, HIGH);
  delay(1000);
  digitalWrite(25, LOW);
  delay(1000);
}
```

Výpis 9.2: C++ blikací program



Obr 9.7: Nahrání blikacího programu v PlatformIO

PlatformIO by mělo automaticky najít správný COM port pro nahrání programu. Port lze specifikovat i manuálně pomocí konfiguračního souboru `platformio.ini`. V programu 9.3 je pro nahrání nastaven COM6.

```
[env:pico]
platform = raspberrypi
board = pico
framework = arduino
upload_port = COM6
```

Výpis 9.3: Manuální nastavení COM portu

9.3 Programování Pico pomocí Pico-SDK

Podobně jako jiné mikrořadiče, je možné vytvořit firmware pro RPI Pico v jazyce C a C++. Pro práci s periferiemi dodává firma Raspberry knihovny Pico-SDK. Tyto knihovny jsou připravené pro build systém CMake, pomocí kterého lze zařídit kompilaci celého firmwaru. V této práci se zaměříme na psaní programů ve VSCode a generování firmwaru pomocí systému CMake.

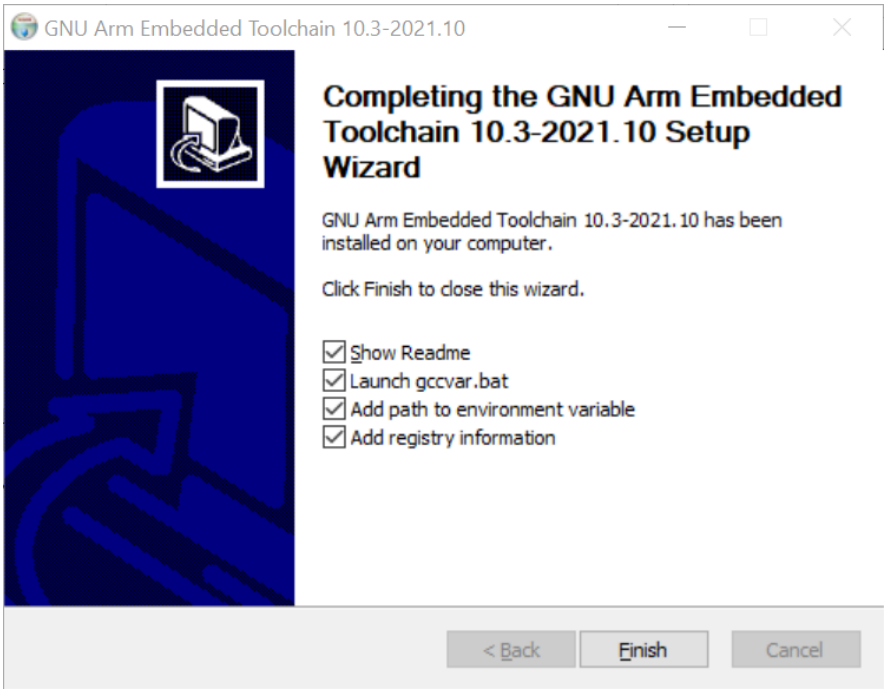
- `https://github.com/raspberrypi/pico-sdk`
- `https://cmake.org/`
- `https://code.visualstudio.com/`

9.3.1 Pico-SDK ve Windows

Abychom mohli programovat Pico pomocí Pico-SDK budeme potřebovat několik programů, nástrojů a v tomto návodu bude ukázáno, jak je získat. Návod se zaměří na programování pomocí Visual Studio Code ve Windows 10.

Instalace ARM kompilátoru

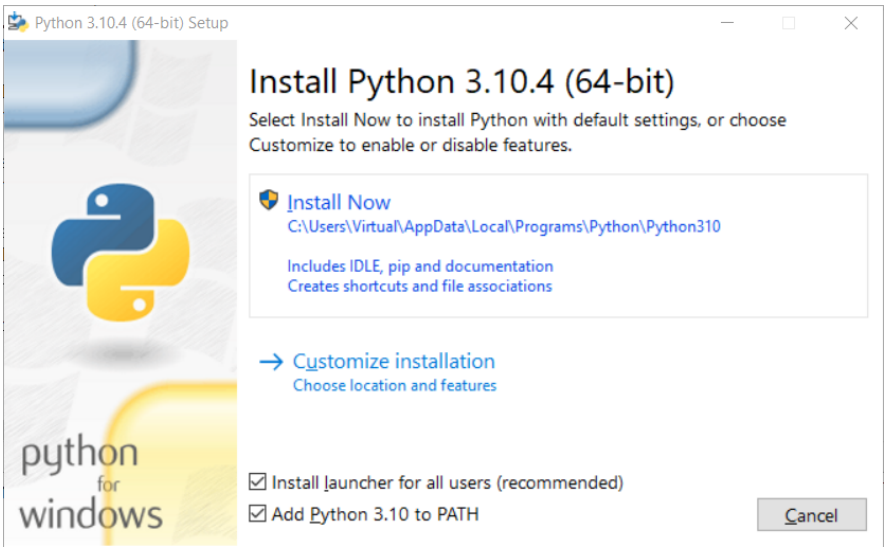
První nástrojem bude samotný kompilátor, který přeloží námi napsaný kód do binárního, kterému rozumí procesor v Pico. Instalátor nalezneme na adrese https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm/downloads . Zde stáhneme .exe instalátor, který má v době psaní název **gcc-arm-none-eabi-10.3-2021.10-win32.exe**. Po stažení nainstalujeme standardním způsobem, ale na konci instalace je důležité zatrhnout odrážku **Add path to enviroment variable**, jako na obr. 9.8



Obr 9.8: Instalace arm kompilátoru

Instalace Python

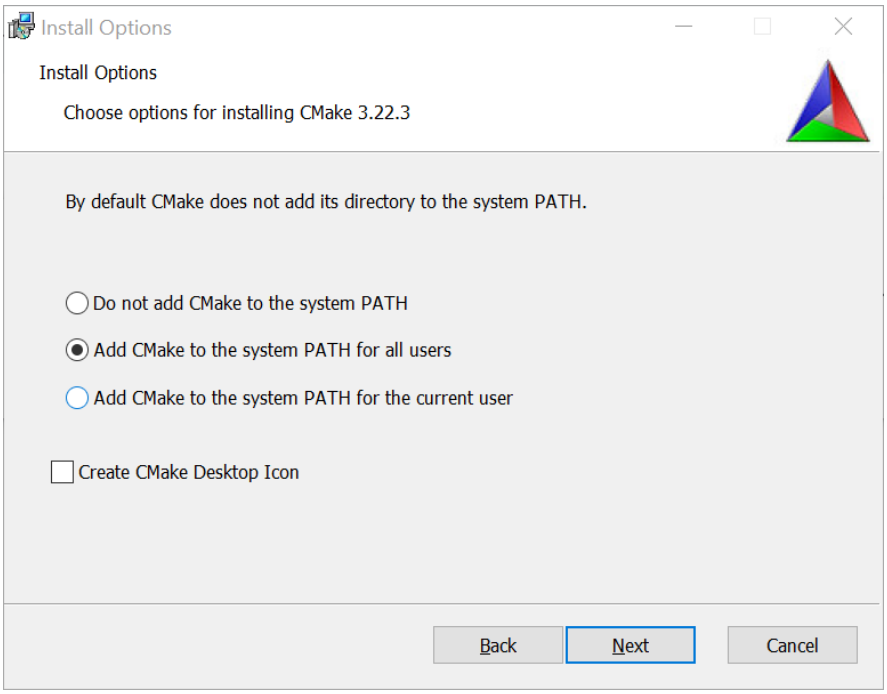
Python nalezneme na adrese `https://www.python.org/getit/`. Po stažení spustíme instalátor a nainstalujeme standardním způsobem. Na konci instalace je opět důležité zatrhnout políčko **Add Python 3.xx to PATH**, jako na obr. 9.9.



Obr 9.9: Instalace python

Instalace CMake

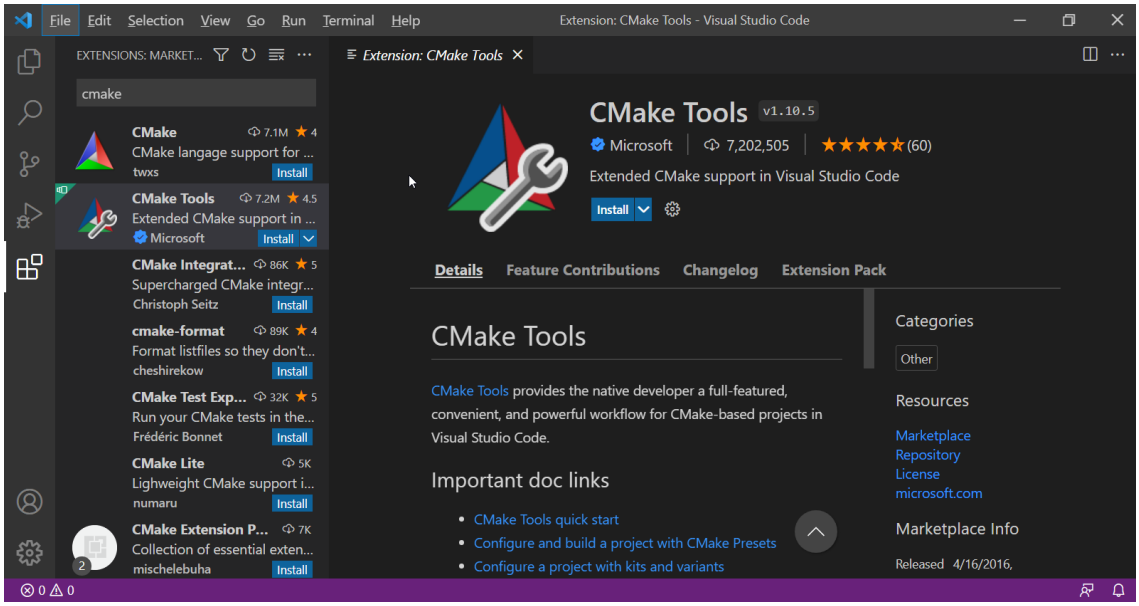
CMake nalezneme na adrese `https://cmake.org/download/`. Na stránce jsou nejdříve instalátory pro **Release Candidate**, ale doporučil bych nainstalovat stabilní verzi z kolonky **Latest Release**. Zde najdeme instalátor, který je v době psaní **cmake-3.22.3-windows-x86_64.msi**. V průběhu instalace bude opět potřeba zatrhnout kolonku **Add CMake to the system PATH**, jako na obr. 9.10.



Obr 9.10: Instalace CMake

Instalace Visual Studio Code

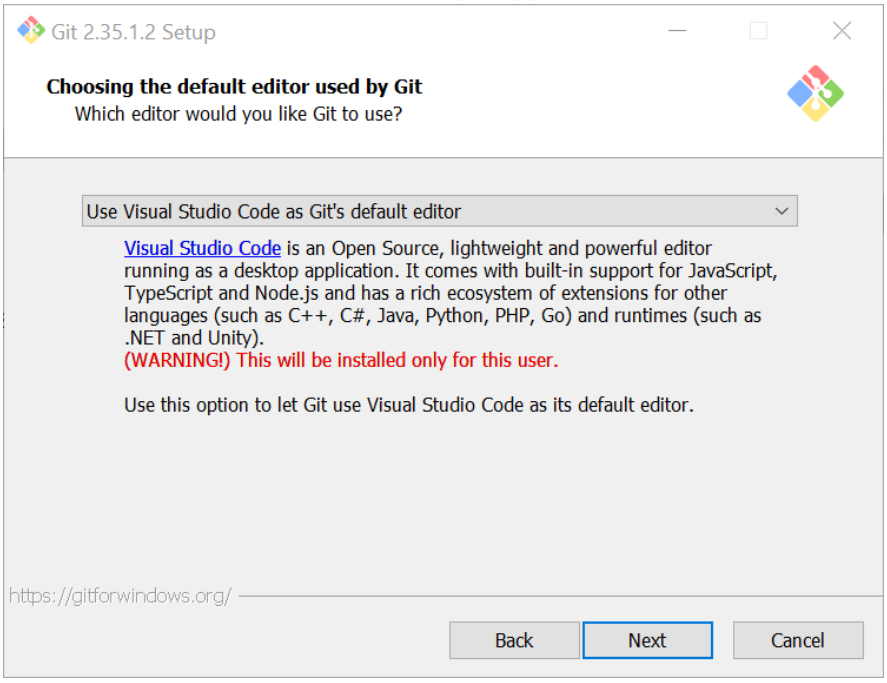
Visual Studio Code nalezneme na `https://code.visualstudio.com/Download`. Program nainstalujeme standardním způsobem. Po nainstalování a spuštění otevřeme seznam rozšíření a zde vyhledáme „cmake“. Nainstalujeme rozšíření **CMake Tools** jako na obr. 9.11.



Obr 9.11: Instalace CMake Tools ve Visual Studio Code

Instalace verzovacího systém git

Verzovací systém git nalezneme na `https://git-scm.com/download/win`. Při instalaci dává git několik možností, které můžeme nechat výchozí. Nastavení, které bych při instalaci doporučil změnit je výchozí editor z Vim, který může být složitý na použití, na Visual Studio Code nebo Notepad. Změna editoru je na obr. 9.12.



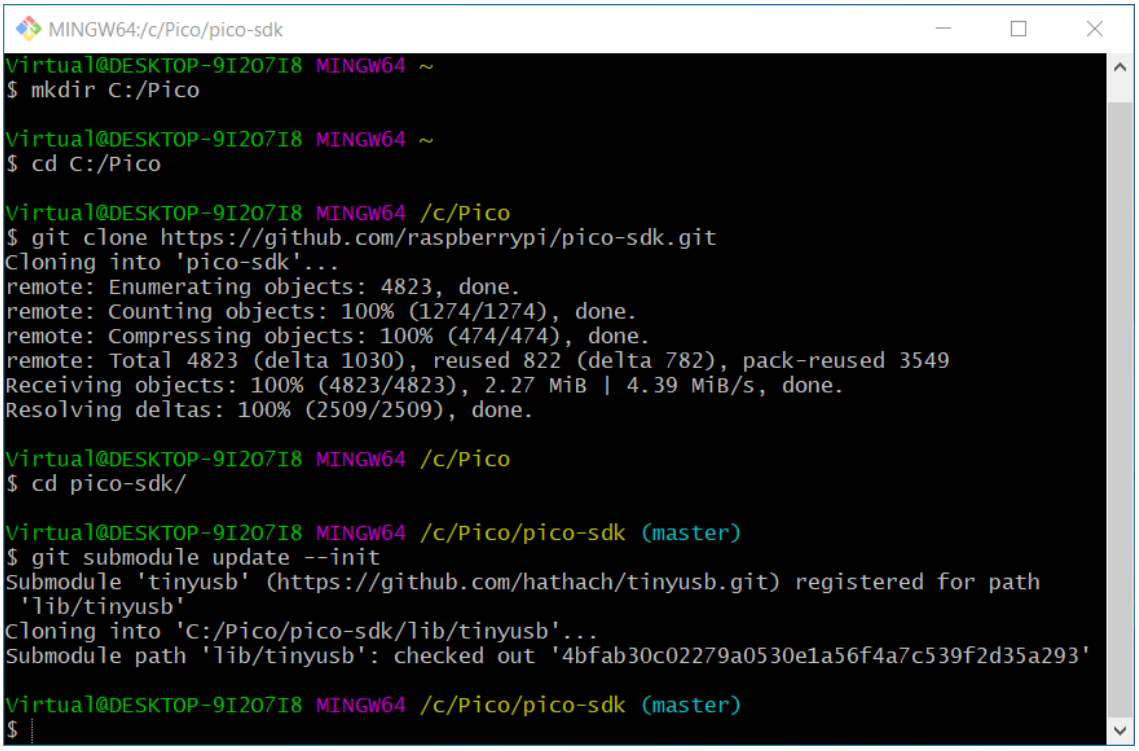
Obr 9.12: Instalace git

Pico-SDK

Nyní stáhneme samotné Pico-SDK. To provedeme pomocí git. Nejdříve otevřeme program Git Bash, který se nainstaloval společně se samotným git. Otevře se okno terminálu, do kterého postupně napíšeme příkazy vypsané ve výpisu 9.4. Ukázka příkazů napsaných v Git Bash je také na obr. 9.13.

```
mkdir C:/Pico
cd C:/Pico
git clone https://github.com/raspberrypi/pico-sdk.git
cd pico-sdk
git submodule update --init
```

Výpis 9.4: Stažení PicoSDK pomocí git



Obr 9.13: Stažení PicoSDK pomocí Git Bash

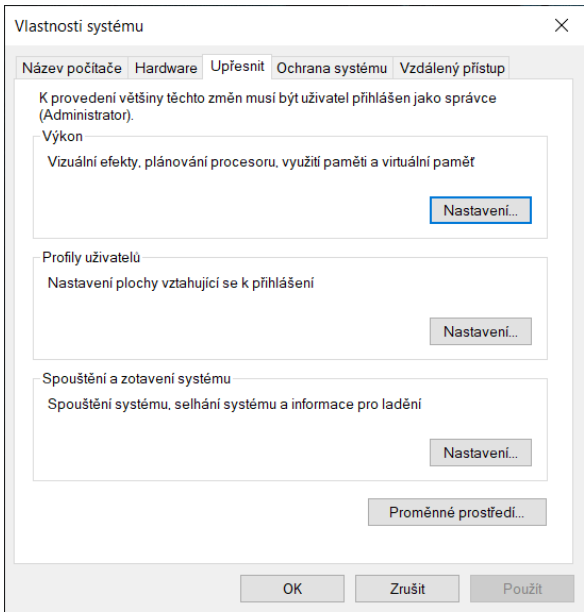
Mingw-w64

Pomocí mingw-w64 získáme nástroje, které jsou potřeba ke kompilaci Pico-SDK. Dokumentace od Raspberry pi pro tyto účely doporučuje nainstalovat **Build Tools for Visual Studio 2019**, ale instalace těchto nástrojů může být velice obsáhlá a zabírat několik GB na disku. Rozhodl jsem se použít alternativní nástroje od GNU jako je **make** a **gcc**. GNU nástroje lze nainstalovat i pomocí jiných projektů jako je **Cygwin**, **MSYS2** apod.

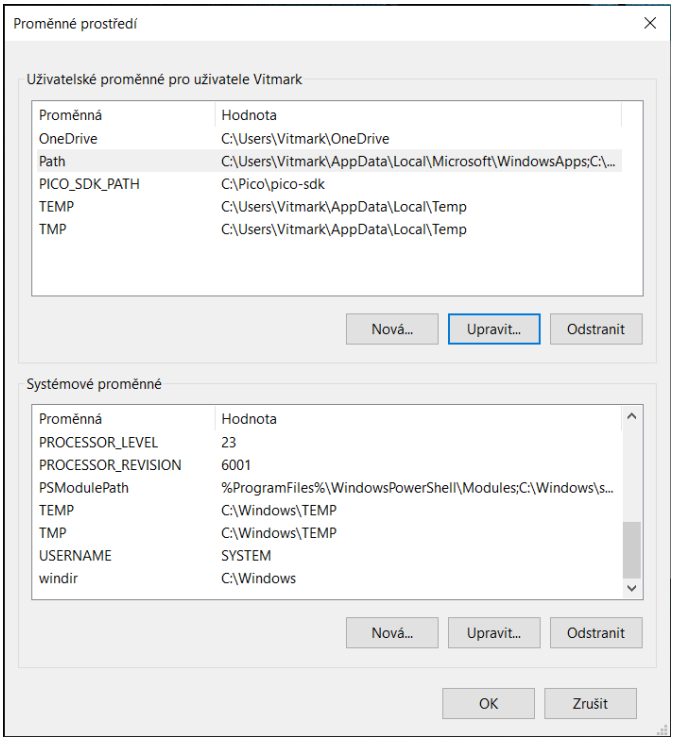
Z důvodu jednoduché instalace jsem se rozhodl použít právě mingw-w64, který nalezneme na `https://sourceforge.net/projects/mingw-w64/files/mingw-w64/`. Zde stáhneme poslední verzi, v době psaní 8.10, **x86_64-posix-sjlj**. Stáhne se soubor **x86_64-8.1.0-release-posix-sjlj-rt_v6-rev0.7z**, který rozbalíme například pomocí 7zip. Uvnitř archivu nalezneme složku **mingw64**, kterou rozbalíme například do složky **C:/Pico/**. Budeme tedy mít složku **C:/Pico/mingw64**, s GNU nástroji.

Úprava proměnných prostředí

Ve vyhledávání Windows najdeme „Upravit proměnné prostředí systému“. Po otevření dostaneme okno jako na obr. 9.14a. V tomto okně klikneme na **Proměnné prostředí** a otevře se nové okno jako na obr. 9.14b. Zde v uživatelských proměnných klikneme na **Nová...** a vytvoříme proměnnou s názvem **PICO_SDK_PATH** a jako hodnotu dáme cestu, kam jsme stáhli pico-sdk jako na obr. 9.15. Dále vybereme proměnnou **Path** a klikneme na **Upravit...**. Zde přidáme cestu k mingw64 jako na obr. 9.16.

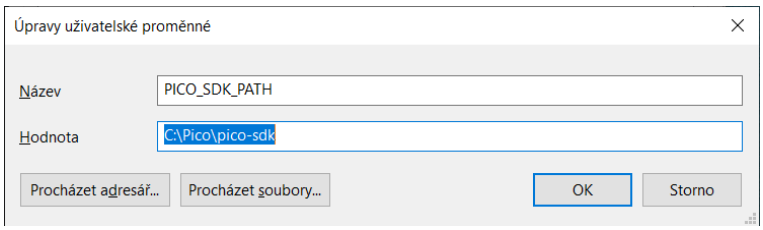


(a) : Vlastnosti systému

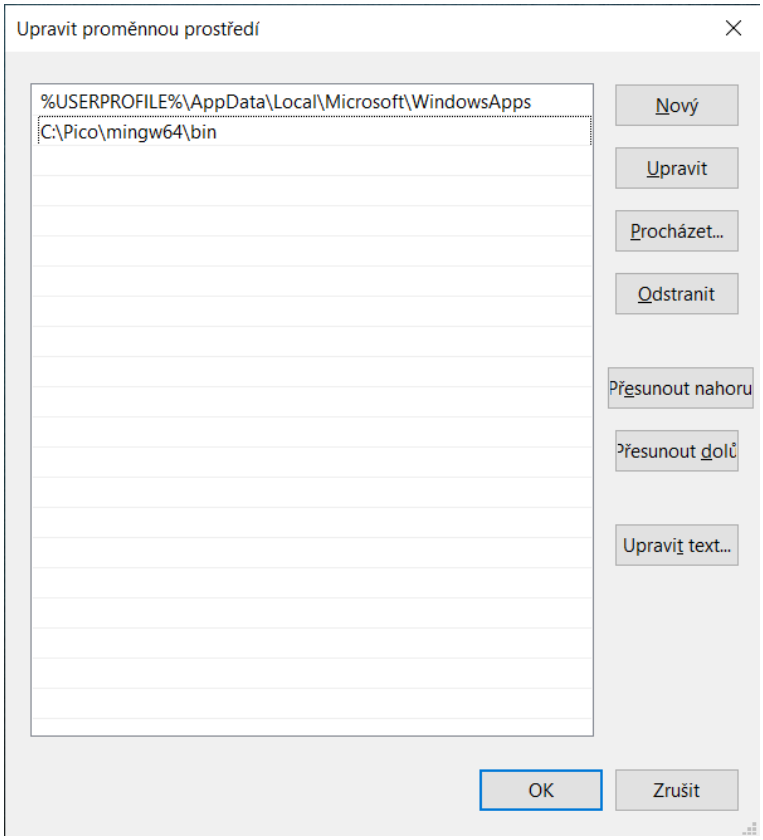


(b) : Proměnné prostředí

Obr 9.14: Okna úpravy proměnných prostředí



Obr 9.15: Přidání proměnné pro Pico-SDK



Obr 9.16: Přidání MinGW-W64 do Path

9.3.2 Založení projektu s Pico-SDK

Nejdříve vytvoříme složku, ve které bude projekt uložený. V tomto případě se bude jednat o složku **blinky**. Tuto složku otevřeme ve Visual Studio Code a vytvoříme soubory **main.c** a **CMakeLists.txt**. Dále do složky nakopírujeme soubor **pico_sdk_import.cmake**,
který najdeme v **C:\Pico\pico-sdk\external**. Složka **blinky** bude vypadat následovně:

```
blinky
├── pico_sdk_import.cmake
├── CMakeLists.txt
└── main.c
```

Do souboru **main.c** napíšeme jednoduchý blikací program jako ve výpisu 9.5.

```c
#include "pico/stdlib.h"

int main() {
    /** Inicializace LED pinu a nastaveni na vystup **/
    uint led_pin = 25;
    gpio_init(led_pin);
    gpio_set_dir(led_pin, GPIO_OUT);

    /** Nekonecna smicka na blikani s LED **/
    while (true) {
        gpio_put(led_pin, 1);
        sleep_ms(250);
        gpio_put(led_pin, 0);
        sleep_ms(250);
    }

    return 0;
}
```

Výpis 9.5: Blikací program s PicoSDK

Abychom program mohli zkompilovat, musíme napsat konfiguraci CMakeLists.txt, jako je ve výpisu 9.3.2.

: CMakeLists.txt

```
cmake_minimum_required(VERSION 3.22)
include(pico_sdk_import.cmake)

# Deklarace jmena projektu a pouzitych jazyku
project(pico_blinky C CXX ASM)
pico_sdk_init()

# Pridani zdrojovych souboru
add_executable(rpi_pico_blinky
    main.cpp
)

target_link_libraries(rpi_pico_blinky pico_stdlib)
pico_add_extra_outputs(rpi_pico_blinky)

# Vypis vyuziti SRAM a FLASH pameti
target_link_options(rpi_pico_blinky PRIVATE "-Wl,--print-memory-usage")
```

Kapitola 10
Zhodnocení dosažených výsledků

V této kapitole budou shrnuty dosažené výsledky. Hlavním cílem bylo vytvoření základních měřicích přístrojů s pomocí vývojové desky Raspberry Pi Pico. Přístroje zahrnují logický analyzátor s aplikací PulseView a osciloskop s aplikací Data Plotter. Osciloskop obsahuje i funkci generátoru PWM signálů. Na základě zkušeností získaných při vytváření přístrojů byly sepsány poznatky užitečné pro budoucí studenty, kteří budou s Pico pracovat. V práci je prozkoumána možnost vytvoření čítače pro měření frekvence a střídy digitálních signálů. Na konci práce je návod, jak začít Pico programovat.

10.1 Poznatky z programování Pico

Pico je nová vývojová deska s mikrořadičem RP2040, se kterým v době psaní není moho zkušeností, jako s mikrořadiči typu STM32 a Atmel na deskách Arduino. Byly proto sepsány poznatky s programováním periferií mikrořadiče a s knihovnou PicoSDK. Tyto poznatky se následně využily pro návrh přístrojů s Pico.

10.1.1 Přenos dat pomocí USB

Nejdříve se zjistilo, jakou funkci je možné využít pro odesílání binárních dat pomocí USB. To mohou být data jako digitalizovaný analogový signál pro osciloskop. Díky PicoSDK je možné pro odesílání dat pomocí USB použít funkce ze standardní knihovny, jako jsou `printf` a `putchar`. Tyto funkce provádí převod zakončení řádku, což může být problémové při odesílání vzorků s hodnotami 10 a 13. Jako alternativa k těmto funkcím se ukázaly funkce `puts_raw()` a `stdio_usb.out_chars()`, které převody neprovádí. Ukázalo se také, že pro vypnutí převodu zakončení řádku je možné nastavit proměnnou `PICO_STDIO_ENABLE_CRLF_SUPPORT` na 0 v `CMakeLists.txt`.

Krom toho, která funkce je vhodná na přenos binárních dat, je dobré vědět, pomocí které funkce lze data přenést nejrychleji. Byl proto proveden experiment, kde proběhlo odeslání 30000 bajtů z Pico do PC. Odeslání se provedlo s různými funkcemi a následně byla vytvořena tabulka a graf naměřených rychlostí. Pomocí experimentu se ukázalo, že nejrychlejší jsou funkce `stdio_usb.out_chars()` a `_write()` při odesílání několika bajtů najednou. Funkce `_write()` ovšem provádí převod zakončení řádku a funkce `stdio_usb.out_chars()` je proto vhodnější možností pro odesílání velkého množství binárních dat.

Nakonec se zjistilo, že Pico kontroluje příznak DTR. Tento příznak musí být aktivní, aby Pico rozpoznalo propojení s počítačem a odeslalo data. V některých případech je ovšem nutné komunikovat s aplikací která příznak DTR neaktivuje. Byla proto ukázána možnost, jak kontrolu příznaku vypnout v knihovně PicoSDK.

10.1.2 Cyklický režim DMA a zápis do registrů pomocí DMA

Pro logický analyzátor i osciloskop je vhodné mít možnost nastavit DMA kanál do cyklického režimu. V cyklickém režimu, po dosažení konce datového pole, se DMA kanál vrací na začátek. DMA řadič v RP2040 tuto možnost nenabízí a byla nalezena alternativa. Ta spočívá ve využití řetězení DMA

kanálů, kdy dokončení jednoho kanálu spustí druhý kanál. Pro uskutečnění cyklického režimu lze tedy využít 2 DMA kanály. Dále se při experimentech s použitím DMA zjistilo, že může být problémové provádět pomocí DMA 16bitové a 8bitové přenosy do registrů.

10.2 Osciloskop s Pico

Práce zahrnuje vytvoření osciloskopu, který pro nastavení vzorkování a vizualizaci využívá aplikaci Data Plotter. Aplikace umožňuje vytvoření terminálového rozhraní a pro tento účel byla vytvořena knihovna, která využívá hlavně prvků jazyka C++. Tato knihovna umožnila vytvoření terminálového rozhraní modulárně. Samotné rozhraní se skládá ze statických prvků, které se nemění, a dynamických, které se mění podle nastavených parametrů a podle interakce uživatele. Těmito prvky jsou tlačítka, měnící se texty a čísla.

Pro realizaci osciloskopu je důležitá digitalizace vstupního napětí. To se uskutečnilo pomocí AD převodníku v mikrořadiči RP2040. Tento AD převodník umožnil také nastavení vzorkovací frekvence. Krom samotného vzorkování byla realizována funkce trigger. Ten slouží ke spuštění vzorkování poté, co napětí na vstupu AD převodníku klesne nebo překročí nastavenou úroveň. Vzorkování i trigger byl realizován pomocí druhého procesoru v mikrořadiči.

Další funkcí užitečnou pro osciloskop je také generátor signálu. Toho je možné s Pico dosáhnout s pomocí PWM čítače, který generuje PWM signál s nastavitelnou střídou. Pro vylepšení schopností generátoru se také využilo DMA pro automatickou změnu střídy. Díky tomu lze měnit střídu podle funkcí, jako je sinus a trojúhelník.

Finální parametry osciloskopu jsou následující:

- 1 vstupní kanál s maximální vzorkovací frekvencí 500kHz,
- maximální počet vzorků 8192,
- trigger s nastavitelnou úrovní napětí,
- nastavitelný poměr vzorků před a po trigger,
- režimy AUTO, NORMAL a SINGLE,
- generátor PWM signálu s nastavitelnou střídou a frekvencí až 5MHz,
- generátor PWM signálu s proměnnou střídou podle funkce sins a trojúhelník.

10.3 Logický analyzátor s Pico

Dalším vytvořeným přístrojem je logický analyzátor, který pro nastavení vzorkování a vizualizaci využívá aplikaci PulseView. Pro komunikaci mezi Pico a PulseView byl využit ELA protokol a aplikace PulseView byla upravena pro přidání podpory přístroje s Pico. PulseView umožňuje zaznamenané signály také analyzovat a zobrazit například konkrétní data odeslaná pomocí digitální komunikace. Stejně jako v případě osciloskopu probíhá komunikace mezi Pico s pomocí USB a bylo využito poznatků z měření rychlosti funkcí pro přenos dat v PicoSDK.

Logický analyzátor zaznamenává pouze binární stav na vstupech a lze tedy použít digitální vstupy mikrořadiče. Ukázalo se, že pro tyto účely lze využít programovatelnou periferii PIO a prozkoumala se možnost použití PIO na více než jen čtení vstupů. Do programu pro PIO byla přidána možnost ukončení vzorkování po nastaveném počtu vzorků, Program se následně vylepšil a umožnil tak rychlejší vzorkování.

Pro logický analyzátor je také důležitá možnost trigger vzorkování, na který se využilo přerušení od GPIO. V tomto přerušení se uloží číslo vzorku a pošle se počet zbývajících vzorků PIO programu. Po ukončení vzorkování je zpětně zaznamenaný signál překontrolován, aby se nalezlo přesné místo, kdy nastal trigger.

Finální parametry logického analyzátoru jsou následující:

- 8 vstupních kanálů s maximální vzorkovací frekvencí 40MHz,
- maximální počet vzorků 50000,
- jednokanálový trigger,
- nastavitelný poměr vzorků před a po trigger,
- možnost nastavení trigger při náběžné, sestupné, nebo libovolné hraně,
- 2 generátory PWM signálu s frekvencí 10kHz a 25kHz.

10.4 Prověření implementace čítače s Pico

Na základě poznatků z návrhu osciloskopu a logického analyzátoru se prověřila možnost vytvoření čítače s Pico. Nejdříve je popsána možnost použití vstupního kanálu PWM čítače. Krom PWM čítačů je prozkoumána také možnost použití programovatelné periferie PIO a to využít čtení stavu vstupu a na základě toho odečítat hodnotu registru. Tím je dosaženo podobné funkce jako mají PWM čítače, ale do programu pro PIO je možné zapojit další logiku.

10.5 Návody na programování Pico

V závěru práce jsou návody na programování Pico s pomocí několika možností. Jednou z hlavních možností pro začátečníky je Micropython. V práci je návod, jak nainstalovat potřebné programy do PC a základní blikací program. Dále je ukázán projekt Bipes, pomocí kterého lze Pico programovat graficky. Následně byla ukázána možnost programování v C/C++ s pomocí knihoven Arduino a PicoSDK.

Kapitola 11

Závěr

Hlavním cílem práce bylo vytvořit měřicí přístroje a PWM generátor s modulem Raspberry Pi Pico. V práci je navržen osciloskop využívající aplikace Data Plotter, který umožňuje krom logických signálů také měření průběhu analogového napětí. Komunikace mezi Pico a Data Plotter je zařízena pomocí USB a nejsou potřeba dodatečné součástky. Jednou z funkcí osciloskopu je PWM generátor, který umožňuje nastavení frekvence, střídy a také měnící se střídu na základě funkcí sinus a trojúhelník.

Dalším navrženým přístrojem je logický analyzátor, který využívá aplikaci PulseView. Tato aplikace umožňuje krom vizualizace i analýzu logických signálů a digitálních komunikací. Komunikace mezi Pico a PulseView probíhá pomocí USB. Součástí logického analyzátoru jsou 2 PWM generátory s pevnou frekvencí 10kHz a 25kHz.

Na základě zkušeností z programování Pico jsou také napsány poznatky, které mohou být užitečné pro další studenty, kteří budou s Pico pracovat. Krom poznatků jsou napsány návody na možnosti programování Pico pomocí jazyka Python, C/C++, ale také graficky. Zadání práce bylo splněno v plném rozsahu.

Příloha A

Seznam zkratek

Zkratka	Význam zkratky
PWM	Pulsně šířková modulace
DMA	Periferie pro přenos dat uvnitř mikrořadiče nezávisle na procesoru
USB	Universální sériová sběrnice
PC	Osobní počítač
ADC	Převodník analogového napětí na digitální hodnotu
DTR	Příznak označující, že je datový terminál připraven přijímat data
PIO	Programovatelná periferie určená pro ovládání GPIO s přesným načasováním
GPIO	Universální vstupně/výstupní pin
UART	Asynchronní sériová sběrnice
I2C	Synchronní sériová sběrnice vyvinutá firmou Philips
SPI	Synchronní sériová sběrnice

Příloha B

Bibliografie

[1] Jiří Maier. „Univerzální GUI pro osciloskopické PC aplikace“. Bakalářská práce. Praha: ČVUT v Praze, Fakulta elektrotechnická, 2021. URL: https://dspace.cvut.cz/handle/10467/94674 (cit. 20. 05. 2022).

[2] *Raspberry Pi Pico C/C++ SDK*. URL: https://datasheets.raspberrypi.com/pico/raspberry-pi-pico-c-sdk.pdf (cit. 20. 05. 2022).

[3] *Raspberry Pi Pico Datasheet*. URL: https://datasheets.raspberrypi.com/pico/pico-datasheet.pdf (cit. 20. 05. 2022).

[4] *Raspberry Pi Pico Pinout*. URL: https://datasheets.raspberrypi.com/pico/Pico-R3-A4-Pinout.pdf (cit. 20. 05. 2022).

[5] *Raspberry Pi Pico Python SDK*. URL: https://datasheets.raspberrypi.com/pico/raspberry-pi-pico-python-sdk.pdf (cit. 20. 05. 2022).

[6] *RP2040 Datasheet*. URL: https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf (cit. 20. 05. 2022).

[7] *sigrok wiki*. URL: https://sigrok.org/wiki/Main_Page (cit. 20. 05. 2022).

[8] Vít Vaněček. „Logický analyzátor s mikrořadičem“. Bakalářská práce. Praha: ČVUT v Praze, Fakulta elektrotechnická, 2021. URL: https://dspace.cvut.cz/handle/10467/89986 (cit. 20. 05. 2022).

Příloha C

Přiložené soubory a odkazy

C.1 Obsah přiloženého disku

```
/
├── [libsigrok-ela] ..... Upravená knihovna libsigrok
├── [Logic-analyzer]
│   ├── [ela-fw-pico] ..... Zrojový kód firmwaru pro logický analyzátor
│   ├── ela-pico-pinout.png ..... Přiřazení pinů logického analyzátoru
│   └── ela-fw-pico.uf2 ..... Frimware pro logický analyzátor s Pico
├── [Oscilloscope]
│   ├── [rpi-pico-osc] ..... Zrojový kód firmwaru pro osciloskop
│   ├── pico-osc-pinout.png ..... Přiřazení pinů osciloskopu
│   └── pico-osc.uf2 ..... Frimware pro osciloskop s Pico
├── [PulseView-ELA-Linux] ..... PulseView pro Linux ve formátu .AppImage
├── [PulseView-ELA-win32] ..... PulseView pro 32bitové Windows
├── [PulseView-ELA-win64] ..... PulseView pro 64bitové Windows
├── Odkazy.txt ..... Odkazy na aktuální verze firmwarů a programů
└── Logicky-analyzator-s-RPI-Pico.pdf ..... Diplomová práce ve formátu pdf
```

C.2 Získání Data Plotter

Data Plotter je dostupný na `https://github.com/jirimaier/DataPlotter/releases`.

C.3 Odkazy na vytvořený firmware a upravené PulseView

Firmware vytvořený v této práci a upravené PulseView jsou dostupné online na stránce Github.

FW osciloskopu	`https://github.com/ela-project/rpi-pico-osc/releases`
FW logického analyzátoru	`https://github.com/ela-project/ela-fw-pico/releases`
Upravené PulseView	`https://github.com/ela-project/ela-pulseview/releases`

C.4 Požadavky pro spuštění PulseView

PulseView pro Windows vyžaduje `Microsoft Visual C++ 2010 Redistributable`. To lze získat ze stránek z `https://www.microsoft.com/en-us/download/details.aspx?id=26999` nebo pomocí winget:

- 64bitový Windows: `winget install Microsoft.VC++2010Redist-x64`
- 32bitový Windows: `winget install Microsoft.VC++2010Redist-x86`