Obsah obrázku symbol, logo, Písmo, Grafika

Obsah vygenerovaný umělou inteligencí může být nesprávný.

Kalkulačka v postfixové notaci na čipu RP2040

Křesťanské gymnázium, Kozinova 1000, 102 00 Praha 10 – Hostivař

Ondřej Běhal, 2025/26

|  |  |
| --- | --- |
| **Maturitní práce z informatiky**  ZADÁNÍ | |
| **Školní rok:** 2025/26 | **Škola:**  Křesťanské gymnázium  Kozinova 1000  Praha 10 - Hostivař  102 00 |
| **Jméno žáka:** | Ondřej Běhal |
| **Název maturitní práce:** | Kalkulačka v postfixové notaci na čipu RP2040 |
| **Termín odevzdání:** | 9. dubna 2026 |
| **Vedoucí maturitní práce:** | Mgr. Ivona Spurná |
| **Oponent maturitní práce:** |  |
| **Délka obhajoby:** | 15 min |
| **Způsob zpracování maturitní práce:** | Žák zpracuje **maturitní práci v elektronické podobě** v programu vhodném ke zpracování – dle charakteru práce.  Průběžné výstupy i finální výstup odevzdá elektronicky do vyhrazeného místa určeném vedoucím maturitní práce.  Dále žák připraví **prezentaci**, která bude dokumentovat danou práci, její průběh a závěrečný stav. Tato prezentace bude taktéž odevzdána jako soubor do vyhrazeného místa určeném vedoucím maturitní práce.  Dále žák připraví textovou zprávu – **dokumentaci** – v rozsahu a obsahu definovaném ve *Specifikaci textové části maturitní práce z informatiky*, která je na školních webových stránkách. Tuto dokumentaci odevzdá jak elektronicky v editovatelné podobě (MS Word) a verzi pro tisk (PDF) do vyhrazeného místa určeném vedoucím maturitní práce, tak v tištěné podobě v jednom výtisku.  Z hlediska autorství musí být práce provedena výše uvedeným žákem. Uvede použité zdroje v dokumentaci i prezentaci. |
| **Hodnocení:** | Z provedení maturitní práce (tj. vlastní praktická práce, soubory prezentace a dokumentace) získá žák na konci školního roku známku odpovídající celkovému bodovému zisku z výstupů.  100–90 bodů – výborný 89–75 bodů – chvalitebný 74–50 bodů – dobrý 49–25 bodů – dostatečný 24–0 bodů – nedostatečný  Celkové hodnocení maturitní zkoušky z informatiky: celková známka bude reflektovat z 50 % získané hodnocení z provedení maturitní práce (viz výše) a z 50 % ústní obhajobu této práce u maturitní zkoušky. Žádná z těchto známek nesmí být 5 (nedostatečná), jinak je celkové hodnocení 5 (nedostatečná). |
| **Výstupy:** | 1. výstup – 1. část maturitní práce – max 15 bodů 2. výstup – 2. část maturitní práce – max 15 bodů 3. výstup – 3. část maturitní práce – max 15 bodů 4. výstup – 4. část maturitní práce – max 15 bodů 5. výstup – soubory prezentace a dokumentace – max 40 bodů |
| **Termíny výstupů:** | 1. výstup – 13. října 2025 2. výstup – 1. prosince 2025 3. výstup – 12. ledna 2026 4. výstup – 23. února 2026 5. výstup – 9. dubna 2026 |
| **Zadání maturitní práce:** | Cíl práce:  Vytvořit jednoduchou kalkulačku využívající postfixovou notaci matematických výrazů (také známá jako Reverzní polská notace – RPN), jež je založena na zásobníku (stack) a nevyžaduje používání závorek ani vyhodnocování precedence operací, neboť pořadí operací je již v zápisu samotném. Kalkulačka není programem běžícím pod operačním systémem, ale je naprogramovaná na „holém železe“ (bare metal) přímo na mikrořadiči RP2040 od britské firmy Raspberry Pi Ltd., konkrétně na vývojové desce Raspberry Pico. Výstup ukazuje uživateli na 128x64bodovém OLED displeji založeném na čipu SSD1306. Vstup obdrží přes klávesnici přes sériové rozhraní UART nebo SWD.  Průběžné výstupy:   1. výstup: Žák dokáže využít již existující ekosystém knihoven a nástrojů, aby dokázal napsat kód pro mikrořadič, který posléze umí zkompilovat, nahrát na mikrořadič přes ladící sériové rozhraní SWD a spustit svůj program. 2. výstup: Žák dokáže pomocí vhodných knihoven komunikovat s OLED displejem přes sběrnici I2C, dokáže na displej vykreslovat jednoduché obrazce i text. 3. výstup: Žák dokáže vytvořit objekt reprezentující výše zmíněný zásobník. Dokáže ho inicializovat, přidat položku na jeho vrchol, odebrat položku z jeho vrcholu, dotázat se na vrchní položku bez jejího odstranění, a dotázat se na prázdnost zásobníku. 4. výstup: Žák dokáže vykreslovat zásobník na displej, i když se na displej nevejde celý; dokáže na něj přidávat nové číselné hodnoty a na nich provádět matematické operace jako sčítání, odčítání, násobení, dělení. |

# Prohlášení

Prohlašuji, že jsem maturitní práci zpracoval samostatně a že jsem uvedl všechny použité informační zdroje a literaturu v seznamu použitých zdrojů.

# Poděkování

# Abstrakt

Tato maturitní práce se zabývá vytvořením jednoduché kalkulačky vytvořené v programovacím jazyce Rust. Program kalkulačky je spouštěn na mikrořadiči RP2040 od společnosti Raspberry Pi Ltd., resp. na jeho vývojové desce Raspberry Pi Pico od stejného výrobce, a svůj výstup vykresluje na OLED displeji. Kalkulačka používá tzv. postfixovou notaci matematických výrazů (též známá jako reverzní polská notace – RPN), kdy operátor (například +) následuje *po* obou hodnotách, na kterých má být operace vykonána. Cílem práce bylo se naučit pracovat s čipem RP2040, obzvláště při jeho programování v jazyce Rust, a ne v jazycích C a C++, které podporuje oficiální balíček pro vývoj softwaru od výrobce čipu.

## Klíčová slova

Kalkulačka, Rust, RP2040, Raspberry Pi Pico, nízkoúrovňové programování, postfixová notace

# Abstract (in English)

This graduation thesis focuses on creating a simple calculator created in the programming language Rust. The programme is run on the RP2040 microcontroller made by Raspberry Pi Ltd., respectively its development board Raspberry Pi Pico by the same maker, and it renders its output on an OLED display. The calculator uses the so-called postfix notation of mathematical expressions (also known as reverse Polish notation – RPN), when the operator comes *after* both operands, on which the operation shall be carried out. The goal of the project was to learn to work with the RP2040 chip, especially when programming it in the Rust language, and not the C or C++ languages, which are supported by the official software development kit made by the chip maker.

## Keywords

Calculator, Rust, RP2040, Raspberry Pi Pico, low-level programming, postfix notation

Obsah

[Prohlášení 4](#_Toc209783467)

[Poděkování 5](#_Toc209783468)

[Abstrakt 6](#_Toc209783469)

[Klíčová slova 6](#_Toc209783470)

[Abstract (in English) 6](#_Toc209783471)

[Keywords 6](#_Toc209783472)

[1. ÚVOD 9](#_Toc209783473)

[2. TEORETICKÁ ČÁST 10](#_Toc209783474)

[2.1 Mikrořadiče 10](#_Toc209783475)

[3. PRAKTICKÁ ČÁST 11](#_Toc209783476)

[4. ZÁVĚR 12](#_Toc209783477)

[Zdroje 13](#_Toc209783478)

# 1. ÚVOD

Tato maturitní práce se zaměřuje na vytvoření funkční kalkulačky. Hlavním cílem je vytvořit program, který dokáže přijímat vstup od uživatele, provádět matematické operace a vykreslovat výstup zpět na displej.

Součástí práce je také použití již existujícího *toolchainu* jazyka Rust, jako i mnoha knihoven a projektů, pro jednodušší tvorbu samotného programu a jeho následného nahrání na paměť flash, ze které čip čte své instrukce. Projekt slouží jako praktická ukázka nízkoúrovňového programování v jazyce Rust namísto obvyklého C nebo C++.

Výsledkem je jednoduchá kalkulačka v postfixové notaci, která lze pak dále rozšířit či upravit. Zároveň představuje zkušenost v programování mikrořadičů, což je mnohdy složitější pro začátečníky a pro což není online zdaleka tolik tutoriálů.

# 2. TEORETICKÁ ČÁST

Není žádným tajemstvím, že celý náš současný svět je postaven na počítačích. Ale ne každý počítač musí být krabice ležící na stole. Existují i drobné počítače, které zabírají prostor jednotek centimetrů krychlových, které mají omezený výkon a specifický účel. Těmto typům počítačů říkáme mikropočítače. Najdeme je prakticky všude kolem nás: od domácích spotřebičů jako pračka nebo mikrovlnka, přes robotické vysavače a chytré žárovky, až po industriální procesy, kde se čas měří na mikrosekundy a kde to nejdrobnější opoždění může mít vážné následky.

S těmito mikropočítači se zachází zcela jinak než s běžnými počítači, jinak se napájí, jinak s nimi komunikujeme, jinak je programujeme. A i to, jakým způsobem jsou programovány, se může mnohdy lišit od klasického způsobu programování.

## 2.1 Mikrořadiče

Konkrétní čipy, které jsou pomyslnými srdci těchto mikropočítačů, nazýváme mikrořadiče (nebo mikrokontroléry, z anglického „microcontroller“). Tyto čipy do sebe integrují procesor, paměť, a obvykle i další periferie usnadňující komunikaci s jinými zařízeními.

Procesor („CPU = central processing unit“) je mozkem samotného mikrořadiče, vykonává strojový kód – instrukce v jazyku symbolických adres („assembly“) odpovídající instrukční sadě daného CPU. Ačkoli současné počítače používají až na pár výjimek (jako např. M-čipy v nových Apple počítačích) instrukční sadu x86-64, v kontextu mikrořadičů se používají častěji jednodušší a energeticky efektivnější sady jako ARM, Atmel AVR nebo RISC-V. Procesor pracuje s registry, což je několik míst přímo uvnitř procesoru, které uchovávají data. Obvykle je jich sice velmi málo (několik desítek) a každý z nich dokáže uchovat jen jedno „slovo“ („word“ – jednotka informace závislá na typu procesoru, u 32bitových procesorů je rovna 32 bitům, u 64bitových je to 64 bitů atd.), ale jsou velmi těsně přimknuty k procesoru, a tudíž velmi rychlé, a jen s daty v nich dokáží instrukce pracovat. Ostatní data musí být do nich zkopírována z adresního prostoru (viz níže).

Paměť dělíme na paměť flash, RAM, ROM a OTP. Paměť flash je přepisovatelná, ale poněkud pomalá, takže ji CPU používá ke čtení dat, které se nemění (jako třeba instrukce, které má vykonávat). Její výhodou je schopnost uchovat si data i po odpojení napájení, tedy říkáme, že je „nevolatilní“. Paměť RAM („=random access memory“ – náhodně přistupovaná paměť) si sice data po vypnutí neuchovává, je „volatilní“, ale je možné do ní poměrně rychle přistupovat a přepisovat ji např. pro mezivýsledky operací a jiných dat, se kterými procesor pracuje. Procesor s ní pracuje tak, že ji namapuje do svého adresního prostoru (viz níže). Paměť ROM („=read only memory“ – paměť pouze pro čtení) je nepřepsatelná a její obsah je určen již během výroby čipu, proto do ní ukládá data výrobce čipu, ať už jsou to informace o čipu samém, nebo kód pro inicializaci čipu a spuštění uživatelského programu. Do OTP („=one-time programmable“ – jednorázově programovatelná) paměti se sice zapisovat dá, ale pouze jednou: po prvním přepsání již nelze změnit. Uchovávají se v ní např. kryptografické klíče nebo jiná data, které si uživatel čipu nastaví obvykle již v továrně při vložení čipu do koncového produktu, a které uživatel koncového produktu nesmí změnit.

Pro přístup k čemukoli jinému, než vlastním registrům potřebuje procesor adresy – číselné hodnoty jednoznačně označující nějaké místo v paměti (nebo jinde, viz později). Rozsah všech těchto číselných adres se nazývá adresní prostor. V případě paměti označuje adresa konkrétní bajt, ke kterému přistupujeme, ale adresní rozsah neobsahuje jen paměť. Komunikaci s periferiemi zvládá totiž CPU tak, že periferie svoje vstupy/výstupy zpřístupňují jako registry, ke kterým CPU přistupuje jako k čtveřicím bajtů v paměti. Každý registr má tudíž přiřazenou adresu ze stejného adresního rozsahu jako paměť. Když nějaké skupině bajtů přidělíme číselný rozsah v adresním prostoru, říkáme, že tuto skupinu namapujeme do adresního rozsahu.

Periferie jsou další jednotky na čipu, které slouží k jiné funkci než vykonávání instrukcí a uchovávání dat a které lze ovládat z programu právě tak, že jsou namapovány do adresního prostoru CPU. Některé z nich jsou zodpovědné za základní funkce nutné ke správnému fungování mikrořadiče, další za různé interní funkce, jiné za komunikaci s okolním světem skrze elektrické signály. Interními periferiemi může být třeba časovač počítající mikrosekundy od startu systému. Periferie komunikující s okolním světem jsou například univerzální vstup-výstup („GPIO = general purpose input/output“), periferie implementující určitý komunikační protokol jako I2C nebo UART, nebo i generátory signálu PWM („=pulse width modulation“ – signál modulovaný šířkou pulzu) a analogově-digitální převodníky („ADC = analog-digital converter“). GPIO je asi nejjednodušší z nich, dokáže pouze číst digitální elektrický signál zapnuto-vypnuto na pinech nastavených jako vstup, nebo podobný signál vyvolávat na pinech nastavených jako výstup.

## 2.2 Specifika čipu RP2040

Čip RP2040 je mikrořadič od britské společnosti Raspberry Pi Ltd., která je známá vyráběním jednodeskových počítačů Raspbery Pi. Poněkud nepřekvapivě, první dvě písmena „RP“ v jeho názvu jsou iniciály společnosti, ale číslo 2040 je poněkud zajímavější. V datasheetu (dokumentu, který výrobce vydává pro svůj čip a ve kterém uvádí všechny detailní informace o jeho používání a fungování) v sekci 1.1 říká výrobce, že první číslice označuje počet jader samotného procesoru (zde 2), druhá číslice udává zhruba typ procesoru (zde ARM Cortex-M0+), a třetí a čtvrté číslice udávají skrze složitější matematický vzoreček množství RAM a nevolatilní paměti na čipu (zde 4 a 0, protože konkrétní čip má 264 kiB RAM a žádnou nevolatilní paměť na čipu). Je to první pokus tohoto výrobce o tvorbu mikrořadiče, a vzhledem k jeho popularitě, široké adopci naskrz průmyslem a podpoře skrze knihovny i dokumentaci, jsem ho zvolil jako vhodnou platformu pro tento projekt. Obzvláště jeho datasheet je ten nejupravenější, nejkrásnější, nejobsáhlejší, nejčitelnější dokument, kterým kdy byl můj zrak požehnán.

Avšak tento konkrétní mikrořadič se od své konkurence v mnoha bodech výrazně liší, ať už pozitivně nebo negativně, a je nutno na tyto vlastnosti brát ohled. Mezi ně patří například periferie PIO, dvě procesorová jádra, specifický a poměrně složitý proces zavádění, rozsáhlá podpora USB, otevřený zdrojový kód ROM nebo absence FPU či analogových periferií mimo již zmíněný ADC, který má navíc pověst nízké kvality.

PIO („=programmable input-output“ – programovatelný vstup-výstup) je asi nejvýraznějším, a mezi ostatními mainstreamovými mikrořadiči unikátním, rozdílem. Jedná se o dvojici periferií zahrnujících celkem 8 konečných automatů („state machine“). V podstatě se jedná o jakési jednoduché procesory úzce specializované na vstupně-výstupní operace, se zaměřením na determinismus a přesné časování. Dokáží využívat jakékoli GPIO piny a jdou programovat v jejich vlastním jazyce symbolických adres poskytujícím celkem 9 16bitových instrukcí. Protože každá instrukce trvá jen jeden cyklus hodinového signálu ovládaného děličem frekvence systémových hodin, pokud není nastaveno jinak, umožňuje PIO implementaci mnohých komunikačních protokolů, včetně těch, které neimplementují jiné periferie na čipu. Ačkoli je tato periferie velmi flexibilní, výkonná, a užitečná, je poměrně komplexní a v tomto projektu pro ni nemáme využití.

Mít rovnou dvě procesorová jádra je v této cenové kategorii poměrně vzácnou vlastností, a umožňuje to skutečně paralelní zpracovávání, jako i vyšší výkon dostupný pro složité operace, avšak v zájmu jednoduchosti nevyužíváme v tomto projektu druhé jádro, jen to první. Hned po startu je skrze zavaděč první fáze v ROM uvedeno druhé jádro do režimu spánku, dokud ho uživatelský program z prvního jádra neprobudí.

Proces zavádění (startování) čipu je také poměrně neobvyklý. Celkem má technicky vzato tři fáze. První fáze je hned po startu, kdy začne procesor vykonávat instrukce rovnou z ROM. ROM obsahuje zavaděč první fáze, který načte prvních 256 bajtů z flash paměti a zkontroluje jejich kontrolní součet. Pokud odpovídá, spustí zavaděč druhé fáze, pokud ne, spustí USB zavaděč, který se z pohledu připojeného počítače chová jako standardní USB externí disk, na který lze zkopírovat program ve formátu UF2, který bude nahrán do paměti flash a spuštěn (to je jednodušší způsob, jak tento mikrořadič programovat). Pomocí tlačítka BOOTSEL lze také vynutit přechod rovnou do USB zavaděče. Zavaděč druhé fáze, známý jako boot2, je zprostředkován uživatelem, ačkoli výrobce poskytuje několik ověřených verzí pro různé flash čipy. Jeho jediným úkolem je inicializovat flash čip podle specifik jeho modelu, aby z něj šel kód spouštět co nejvýkonněji. Kód je z něj spouštěn v režimu XIP, kdy je celý rozsah paměti namapován do adresního rozsahu jako paměť pouze pro čtení a z pohledu softwaru je více méně transparentní. Protože však každý flash čip vyžaduje jinou sekvenci příkazů pro svůj nejvýkonnější mód, musí být inicializační kód dodán uživatelem a ne výrobcem v ROM, aby šel měnit. A nakonec, po dokončení zavaděče druhé fáze, se spustí zavaděč třetí fáze, taktéž známý jako „reset handler“, který je specifický pro konkrétní programovací jazyk a který inicializuje běhové prostředí („runtime environment“), aby konečně mohl být spuštěn uživatelský program v specifikovaném vstupním bodě („entry point“). Například programovací jazyk C inicializuje globální proměnné a zkopíruje ty měnitelné z flash do RAM v sadě rutin známých jako crt0.

Podpora USB již byla zmíněna. ROM obsahuje plně funkční zavaděč typu USB MSC („=mass storage class“ – třída velkokapacitního úložiště), který se chová jako externí disk a který umožňuje programovat celý mikrořadič jednoduše přes USB. Ale kromě toho dokáže USB periferie komunikovat jako jakékoli jiné USB zařízení, od třídy MSC (externí disky) přes HID (klávesnice, myši, herní ovladače) až po CDC (síťové karty, sériové porty, modemy); nebo dokonce dokáže spravovat USB sběrnici sama, převzít roli počítače a komunikovat s jinými USB zařízeními.

Fakt, že zdrojový kód celé ROM je otevřený a každý do něj smí nahlížet nebo s ním pracovat, se sice může na první pohled zdát jako nevýznamné, avšak opak je pravdou: umožňuje to větší transparentnost a zjednodušuje to práci s ROM, když programátor ví, co se v ní děje. Navíc pak ROM obsahuje několik vysoce optimalizovaných funkcí např. pro matematické operace na číslech s plovoucí desetinnou čárkou („=floating-point numbers“), kterýmiž částečně nahrazuje absenci dedikované jednotky FPU („=floating-point unit“ – jednotka pro plovoucí desetinnou čárku / matematický koprocesor), která by tyto operace prováděla přímo a výkonněji. Subsystém SIO však obsahuje alespoň celočíselný dělič, který během 8 cyklů hodinového signálu vypočítá celočíselný podíl a zbytek po dělení (zároveň). Použitý procesor Cortex-M0+ zároveň obsahuje zabudovaný celočíselný násobič.

Vývojová deska, kterou používáme, Raspberry Pi Pico, obsahuje kromě samotného čipu také podpůrný hardware jako flash čip, napájecí zdroj a USB konektor, který dohromady umožňuje zakoupit hotový produkt a začít vyvíjet pro tento čip.

## 2.3 Protokoly

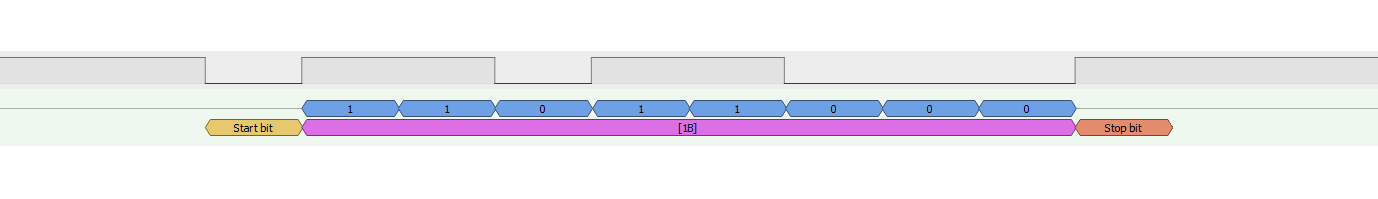
Pro nás projekt využíváme množství protokolů pro komunikaci, zde budou stručně popsány.

Pro příjem vstupu od uživatele používáme jednoduchý sériový protokol UART, pro komunikaci s displejem používáme sběrnici I2C a pro nahrávání a ladění programu používáme protokol SWD.

### 2.3.1 UART

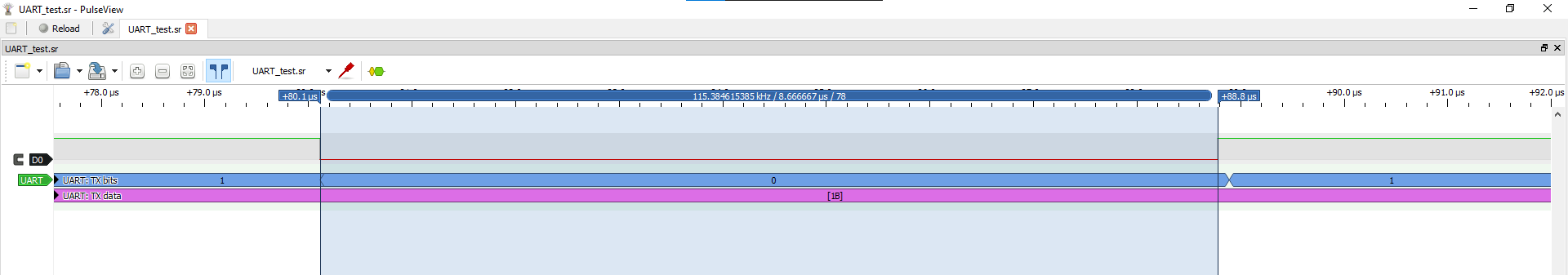
UART („=universal asynchronous receiver-transmitter“ – univerzální asynchronní přijímač-vysílač) je asi jeden z nejjednodušších způsobů, jak přenášet data mezi dvěma systémy – přenáší holé bajty, bez synchronizace, bez nějakých vyšších protokolů kromě základního způsobu binární elektrické signalizace.

Pro signalizaci používá protokol UART pouze hodnoty 0 a 1. Pokud se aktuálně žádná data nepřenáší, zůstává linka na hodnotě 1. Pro začátek komunikace je vyslán jeden tzv. „start bit“ o hodnotě 0, aby příjemce věděl, že začíná přenos; poté je vysláno několik bitů přímo tak, jak je máme v úmyslu přenést (bit 1 jako 1, bit 0 jako 0), nejméně významný bit (bitu na pozici jedniček v čísle dvojkové soustavy) jako první; poté volitelný paritní bit, který však můžeme vynechat; a nakonec jeden nebo více tzv. „stop bitů“, které jsou vždy o hodnotě 1. Stop bit může trvat nejčastěji dobu jednoho, jednoho a půl nebo dvou normálních bitů.



Obrázek 1: Příklad UART přenosu bajtu o hodnotě 27 (1B v šestnáctkové soustavě, 00011011 ve dvojkové soustavě). Pod grafem logického stavu (zelená = 1; červená = 0) lze nalézt dekódované bity i značky start a stop bitů. Schéma přenosu 8N1 – viz níže. Zdroj: vlastní, skrze logický analyzátor PulseView.

Jedním z nejdůležitějších parametrů je rychlost spojení, udávaná v počtech přenesených bitů za sekundu. Jednotka se ale nazývá baud, (potažmo s předponou SI kilobaud / kbaud nebo megabaud / Mbaud) protože zahrnuje i start, paritní a stop bity, nejen datové bity, a rychlost se obvykle nazývá „baudrate“. Obvyklými baudrate jsou 110, 300, 9600 nebo 115200. Historicky byly podporovány jen určité standardní baudrate, ale v současné době lze díky zlomkovým děličům frekvencí použít téměř jakýkoli baudrate, včetně velmi vysokých rychlostí přes 1 Mbaud. Neboť se jedná o frekvenci, perioda – trvání jednoho bitu v sekundách lze spočítat jednoduše jako .



Obrázek 2: Logický analyzátor píše, že jeden bit trvá 8.666667 µs, což by odpovídalo frekvenci 115384.615385 Hz, která je velmi blízko použitému baudrate 115200 baud. Protokol UART naštěstí dokáže tolerovat drobné odchylky od správného baudrate, protože začátkem každého bajtu se opět synchronizuje pomocí start bitu. Zároveň musíme počítat s nepřesností měření. Zdroj: vlastní

Paritní bit má 5 možností: „Odd“ („=lichý“), „Even“ („=sudý“), „Mark“ („=značka“), „Space“ („=mezera“) a „None“ („=žádný“). „Odd“ znamená, že paritní bit je 1, pokud je počet bitů hodnoty 1 v přenášených datech sudý, aby byl celkem přenesen lichý počet. „Even“ je opak – nabývá hodnoty 1, pokud je počet 1-bitů lichý, aby byl celkem přenesen sudý počet. Toto je využíváno jako primitivní kontrolní součet, který dokáže někdy detekovat chybu v přenosu, avšak není zcela spolehlivý, protože v případě chyb v sudém počtu bitů nelze chybu detekovat. „Mark“ znamená, že paritní bit je vždy 1, a „Space“ znamená, že je vždy 0, avšak tyto možnosti neposkytují kontrolní součet a jsou tedy poněkud zbytečné – v obvyklých případech je praktičtější paritní bit úplně vynechat (možnost „None“). Vynechání paritního bitu navíc marginálně zrychlí přenos dat, protože pro každý bajt dat je potřeba přenést méně bitů a tudíž více bitů z každého přenosu je datových.

Celková konfigurace se často zapisuje ve zkrácené formě, například „115200 8N1“, kdy 115200 označuje baudrate, 8 označuje, že jeden bajt je složen z 8 bitů (což historicky nemuselo vždy nutně platit – například původní ASCII tabulka byla jen 7bitová), N, že není použit paritní bit, a 1, že stop bit trvá délku jednoho normálního bitu. Dalším příkladem může být třeba „9600 7E2“ – 9600 baudů, 7 bitů na bajt, paritní bit „Even“, 2 bity dlouhý stop bit.

Detaily jako napěťové úrovně se pak mohou lišit, obvyklými standardy jsou TTL, kdy 0 a 1 jsou přímo napěťové úrovně tranzistorů (obvykle 0 V pro 0 a 1.1 V, 1.8 V, 2.5 V, 3.3 V nebo 5 V pro 1); RS-232 typickým na sériových portech starších počítačů, kdy kladné napětí mezi 3 a 15 V je 0, a stejné záporné napětí je 1; RS-485 obvyklejším v industriálním prostředí, kde přenáší data po sběrnici pomocí diferenciální signalizace – rozdílu mezi dvěmi napětími, kdy záporný rozdíl je 1 a kladný rozdíl je 0. Historicky (v době dálnopisů) se také používaly proudové smyčky, kdy přítomnost proudu značí 1 a nepřítomnost 0.

Tato historie se však promítá až do současnosti: důvod, proč nabývá neaktivní linka stálé hodnoty 1, je právě z dob proudových smyček, kdy přerušení smyčky znamenalo, že nešlo přenášet data ani jedním směrem. Když však byla neaktivní linka hodnoty 1, tak po přerušení byla neustále hodnoty 0, což šlo detekovat například tím, že tehdy mechanické dálnopisy začaly dělat velký hluk, protože neustále přijímaly, co se jim zdálo jako nulové bajty. I moderní UART využívají této skutečnosti, aby detekovaly přerušené připojení a vyvolaly signál chyby „framing error“ (když není přijat platný stop bit = stop bit nabývá hodnoty 0) nebo „break condition“ (když linka zůstane neustále na hodnotě 0 po dobu delší než dobu jednoho celého bajtu vč. start, paritních a stop bitů).

Z abstraktního pohledu lze tedy brát UART jako jednoduché tunely pro bajty – dovnitř zapíšu bajt, z druhé strany přečtu (snad) stejný bajt. Tato jednoduchost umožňuje uživateli implementovat vlastní komunikační protokoly, přenášet data nebo text.

Například terminály, které používáme v našem projektu jako vstup pro uživatele, přijímají vstupy z klávesnice a přenášejí je jako text zakódovaný nejčastěji formátem UTF-8 (zpětně kompatibilní nadstavba nad tabulku ASCII). Některé vstupy však nelze zakódovat jako textové znaky. Například klávesové zkratky Ctrl-<písmeno> odešlou bajt, jehož hodnota je rovna pořadí písmena v abecedě (například Ctrl-C tedy odešle bajt hodnoty 03, nebo Ctrl-U odešle hodnotu 21). Pro složitější vstupy jako F-klávesy (F1 až F12) nebo klávesy pro ovládání kurzoru (Home, End, Delete, PgUp, PgDn, šipky) a jejich kombinace s modifikátory (Ctrl, Alt, Shift) jsou použity ANSI escape kódy, které začínají nezobrazitelným ASCII znakem ESC (hodnota 27) a za nimiž následuje několik běžných textových znaků, které mohou reprezentovat určitou klávesu nebo klávesovou zkratku. Například klávesa Delete je reprezentována jako „<ESC>[3~“ a šipka nahoru je „<ESC>[A“. Tyto kódy však dokážou také ovládat terminál, který je přijímá, například vymazat jeho obrazovku, přemístit kurzor nebo nastavit barvu textu. Tyto kódy byly popularizované terminálem DEC VT100 z roku 1978, a staly se de facto standardním protokolem terminálů. I náš projekt tento protokol v omezené míře využívá.

### 2.3.2 I2C

AAAbbbccc

# 3. PRAKTICKÁ ČÁST

# 4. ZÁVĚR

# Zdroje

**Raspberry Pi Ltd. 2025.** RP2040 Datasheet: A microcontroller by Raspberry Pi. *Raspberry Pi Datasheets.* [Online] 20. únor 2025. [Citace: 26. říjen 2025.] https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf.

# Seznam obrázků

[Obrázek 1: Příklad UART přenosu bajtu o hodnotě 27 (1B v šestnáctkové soustavě, 00011011 ve dvojkové soustavě). Pod grafem logického stavu (zelená = 1; červená = 0) lze nalézt dekódované bity i značky start a stop bitů. Schéma přenosu 8N1 – viz níže. 13](#_Toc212452966)

[Obrázek 2: Logický analyzátor píše, že jeden bit trvá 8.666667 µs, což by odpovídalo frekvenci 115384.615385 Hz, která je velmi blízko použitému baudrate 115200 baud. Protokol UART naštěstí dokáže tolerovat drobné odchylky od správného baudrate, a zároveň musíme počítat s nepřesností měření. 13](#_Toc212452967)