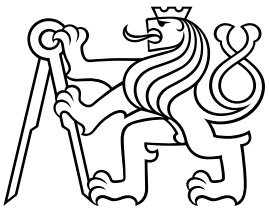
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra kybernetiky

Multifunkční diagnostická logická sonda

Milan Jiříček

Vedoucí práce: doc. Ing. Jan Fischer, CSc.
Studijní program: Otevřená informatika
Specializace: Základy umělé inteligence a počítačových věd
2025

assignment page 1

assignment page 2

Prohlášení

Prohlašuji, že jsem závěrečnou práci vypracoval(a) samostatně a uvedl(a) všekeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací a Rámcovými pravidly používání umělé inteligence na ČVUT pro studijní a pedagogické účely v Bc a NM studiu.

Prohlašuji, že jsem v průběhu příprav a psaní závěrečné práce použil nástroje umělé inteligence. Vygenerovaný obsah jsem ověřil. Stvrzuji, že jsem si vědom, že za obsah závěrečné práce plně zodpovídám.

v Praze, 14. 05. 2025

Abstrakt

Tato práce představuje návrh a realizaci multifunkční diagnostické logické sondy pro výuku práce s logickými obvody a vestavnými systémy. Zařízení nahrazuje komereční laboratorní přístroje, které jsou složité a finančně náročné, a umožňuje studentům měřit základní veličiny (napětí, odpor, frekvenci), generovat pulzy a diagnostikovat komunikační rozhraní (UART, I2C SPI, Neopixel). Sonda je realizována ve dvou variantách: verze na bázi STM32 a zjednodušená pro Raspberry Pi Pico. Lokální režim s RGB LED a tlačítkem slouží pro rychlou diagnostiku, zatímco terminálový režim s ANSI TUI poskytuje pokročilé funkce včetně ovládání a zobrazování hodnot skrze UART periferii. Návrh využívá STM HAL a Raspberry Pi Pico C SDK pro prenositelnost kódu a minimalizuje externí komponenty, aby studenti mohli sondu snadno sestavit na nepájivém kontaktním poli. Hlavním přínosem je urychlení identifikace chyb při praktických cvičeních, jako jsou odhalení vadné součástky nebo nesprávné zapojení periferií.

Klíčová slova: STM32, STM32G0, Raspberry Pi Pico, logická sonda, UART, I2C, SPI, vzdělávací nástroje, diagnostika chyb, C/C++, ANSI escape sekvence

Abstract

This work presents the design and implementation of a multifunctional diagnostic logic probe for teaching logic circuit and embedded system applications. The device replaces complex and costly commercial laboratory instruments, enabling students to measure basic parameters (voltage, resistance, frequency), generate pulses, and diagnose communication interfaces (UART, I2C, SPI, Neopixel). The probe is implemented in two variants: an STM32-based version and a simplified version for Raspberry Pi Pico. A local mode with an RGB LED and button provides quick diagnostics, while a terminal mode with an ANSI-based text user interface (TUI) offers advanced functions, including control and data visualization via UART. The design leverages the STM32 HAL and Raspberry Pi C SDK for code portability and minimizes external components, allowing students to assemble the probe easily on a breadboard. The primary contribution lies in accelerating error identification during practical exercises, such as detecting faulty components or incorrect peripheral wiring.

Keywords: logic probe, STM32, STM32G0, Raspberry Pi Pico, error diagnostics, UART, I2C, SPI, embedded systems, educational tools, ANSI escape sequences

Title translation: Multifunctional Diagnostic Logic Probe

Poděkování

Rád bych tímto poděkoval panu doc. Ing. Janu Fischerovi, CSc., mému vedoucímu práce, za jeho cenné rady, odbornou pomoc a ochotu sdílet své znalosti. Děkuji mu také za věnovaný čas, podnětné připomínky a trpělivost, které mi poskytoval během celého procesu tvorby této práce.

Dále bych chtěl vyjádřit největší poděkování své rodině a mé přítelkyni za jejich neochvějnou podporu, povzbuzení v náročných momentech a pochopení, jež mi dávali najevo po celou dobu mého studia. Bez jejich lásky a motivace by tato práce nevznikla.

Nemohu opomenout ani své přátele, kteří mi po celou dobu studia pomáhali udržet optimismus, sdíleli se mnou radosti i starosti, a svou přítomností mi vytvářeli potřebný odstup od pracovního vypětí. Jejich přátelství a ochota být mi oporou v osobním životě významně přispěly k tomu, abych mohl tuto práci úspěšně dokončit.

1 TODOs remaining

Obsah

Seznam obrázků

Seznam tabulek

Seznam úryvků kódu

Seznam zkratek

SOP	Small Outline Package
TSSOP	Thin Shrink Small Outline Package
PLL	Phase Locked Loop
POF	Point Of Failure
FPGA	Field-Programmable Gate Array
SRAM	Static Random Access Memory
ADC	Analog Digital Converter
MSPS	Milion Samples Per Second
DMA	Direct Access Memory
PWM	Pulse Width Modulation
HAL	Hardware Abstraction Library
GPIO	General Purpose Input/Output
I2C	Inter-Integrated Circuit
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Reciever Transmiter
CMSIS	Cortex Microcontroller Software Interface Standard
NVIC	Nested Vectored Interrupt Controller
IOT	Internet Of Things
EEPROM	Electrically Erasable Programmable Read-Only Memory
MSB	Most Significant Bit
LSB	Least Significant Bit
ASCII	American Standard Code for Information Interchange
TUI	Terminal User Interface
GUI	Graphical User Interface
CMOS	Complementary Metal–Oxide–Semiconductor
MCU	Microcontroller Unit
SWD	Serial Wire Debug
FOSS	Free Open Source Software
SSH	Secure Shell

Kapitola 1
Úvod

Výuka základů elektroniky vyžaduje nástroje, které studentům umožní experimentovat s realnými logickými obvody a prakticky ověřovat teoretické znalosti. Tradiční diagnostické přístroje, jako osciloskopy nebo logické analyzátory, jsou však pro začínající studenty složité, finančně náročné a nabízejí funkce nad rámec základních výukových potřeb. Během praktických cvičení se studenti často dostávají do situací, kdy například čítač pulzů s výstupem na 7-segmentový displej přestane fungovat, aniž je zřejmá příčina. Mohou váhat, zda chyba spočívá v softwaru (nesprávně naprogramovaném časovači), hardwaru (spálené LED, vadném senzoru) nebo třeba v prohození vodičů UART (Tx/Rx). Bez vhodného nástroje pak tráví hodiny hledáním závady, což demotivuje a zpomaluje výuku. Tento problém vede k potřebě vytvořit jednoduché, multifunkční a dostupné řešení, které by usnadnilo osvojení principů práce s logickými a komunikačními obvody.

Cílem této bakalářské práce je návrh a realizace multifunkční diagnostické logické sondy, která kombinuje funkce logického analyzátoru, generátoru signálů a testeru komunikačních rozhraní typické pro výuku (UART, I2C, SPI, Neopixel). Hlavní inovací řešení je integrace vzdělávacího aspektu do samotného návrhu zařízení - sonda je navržena ve dvou variantách: plnohodnotná verze založená na mikrokontroleru STM32 a omezená verze využívající Raspberry Pi Pico pro naplnění potřeby středních škol, které nejsou specializováné na výuku elektroniky. Hlavní výhodou řešení je minimalizace externích komponent, které umožňují sestavení zařízení na nepájivém kontaktním poli studentem a integrace s PC terminálovou aplikací pro pokročilé funkce bez nutnosti instalace specialních aplikací a ovladačů.

Klíčovými prvky návrhu jsou jednoduché ovládání (lokální režim s RGB LED a tlačítkem, terminálový režim ovládaný skrze terminálovou aplikaci), měření napětí, odporu, frekvence a střídy PWM signálů, generování pulsů a diagnostika komunikačních periferií. Důraz je kladen na open-source přístup, který umožní další rozšiřování a přizpůsobení vzdělávacím potřebám.

Text práce je strukturován do osmi kapitol. Po úvodu a rozboru problematiky následuje popis hardwarového návrhu sondy, implementace terminálového a lokálního režimu na základu STM32 a realizace omezené verze na Raspberry Pi Pico. Závěr shrnuje dosažené výsledky práce. Součástí práce jsou také přílohy s ukázkami kódu a uživatelská příručka.

Kapitola 2

Rozbor problematiky

2.1 Motivace

Během laboratorních cvičení zaměřených na logické obvody a vestavné systémy studenti navrhují digitální obvody a programují mikrokontroléry (MCU). Při vývoji však mohou narazit na situaci, kdy jejich řešení úlohy náhle přestane fungovat podle očekávání, aniž by byla zjevná příčina problému. Najít závadu může být velice časově náročné jak pro studenta, tak pro vyučujícího.

Při návrhu softwaru pro mikrokontroléry je klíčové průběžně ověřovat funkčnost pomocí pulsů. Studenti tak mohou například zjistit, zda je generován výstupní pulz požadované frekvence, zda obvod správně reaguje na vstupní impuls, nebo zda je signál přenášen přes daný vodič. Praktickým příkladem je vývoj čítače pulzů s výstupem na 7-segmentový displej – zde sonda umožňuje okamžitě validovat, zda software správně zpracovává vstupy a aktualizuje výstup. Studenti při sestavování obvodů také často čelí problémům jako nefunkční komponenty (spálené LED, vadné senzory) nebo chybám v zapojení – například prohození Tx/Rx vodičů UART, chybějící pull-up rezistory na I2C sběrnici, nebo nesoulad s referenčním schématem. Takové chyby vedou k časově náročnému hledání závad.

Standardní logická sonda je elektronické zařízení sloužící k diagnostice logických obvodů. Pomáhá určovat logické úrovně a detekovat pulsy. Je to jeden ze standardních nástrojů pro elektrotechniky pracující s FPGA, mikrokontrolery či logickými obvody. Výhoda logické sondy je cena pořízení a flexibilita použití. Logická sonda je jedním z prvních nástrojů, který dokáže najít základní problém v digitálním obvodu. Další běžné nástroje pro diagnostiku logických obvodů jsou osciloskop a logický analyzátor. Tyto nástroje jsou vhodné pro diagnostiku např. I2C sběrnice nebo SPI rozhraní, kdy uživatel může vidět konkrétní průběh signálu. Pro výukové účely však mají zásadní nevýhody: Pořizování analyzátorů a osciloskopů může být velice nákladné, jejich ovládání vyžaduje pokročilé znalosti, a student musí nejprve pochopit, jak s přístrojem zacházet. Navíc nabízejí spoustu funkcí, které jsou pro účely výuky nadbytečné a mohou začátečníky dezorientovat.

Multifunkční diagnostická logická sonda (dále jen sonda), která je navržena v rámci této bakalářské práce má za cíl, minimalizovat zmíněné problémy konvenčních diagnostických nástrojů a obecně zpřístupnit diagnostiku studentům, kteří jsou stále ve fázi učení. Sonda, která je vyvinuta, přináší levné řešení, které obsahuje potřebné funkce pro základní diagnostiku logických obvodů a snaží se studentovi zjednodušit celý proces hledání problému v řešení úlohy i bez hlubokých předchozích znalostí s používáním pokročilých diagnostických nástrojů.

Student si může tak osvojit metodiku debugování od základních kontrol napájení, přes odchytávání pulsů po analýzu komunikačních periferií. Možnost sestrojení sondy na nepájivém kontaktním poli poskytuje flexibilitu vyučujícím vytvořit rychle multifunkční logickou sondu. Jelikož návrh bere zřetel na možnost realizace studentem, je při sesta-

vování použito minimální počet externích součástek. Tímto je možno dosáhnout úspory času na straně vyučujícího, kdy vyučující odkáže na použití sondy při hledání problému. Použití MCU typu STM32 a RP2040 umožňuje transparentnost, a možnost hlubšího pochopení fungování sondy z důvodu velkého množství manuálů a návodů na internetu.

2.2 Požadavky

V rámci bakalářské práce bude navržena a realizována multifunkční diagnostická logická sonda (dále jen sonda) na platformě STM32. V návrhu sondy je potřeba zohlednit následující klíčové oblasti: jednoduchost ovládání i uživateli, kteří nemají zkušenosti s používáním pokročilých diagnostických nástrojů, rychlá realizovatelnost sondy na nepájivém kontaktním poli a praktičnost ve výuce. Aby nástroj nebylo komplikované sestavit, je nutné aby bylo využito co nejméně externích součástek. Tím je redukován čas sestavení a také je sníženo množství POF.

Firmware a hardware sondy jsou primárně navrženy pro mikrokontrolér STM32G030 v pouzdrech SOP8 a TSSOP20, které díky integrovaným periferiím (UART, GPIO, časovače) umožňují jednoduché sestavení i na nepájivém kontaktním poli. Sonda bude s omezenou verzí realizována i na Raspberry Pi Pico. Sonda bude vybavena tzv. „lokálním režimem“ a „terminálovým režimem“.

Lokální režim bude sloužit pro rychlou základní diagnostiku obvodů s indikací pomocí RGB LED a ovládání skrze jedno tlačítko. Bude fungovat bez nutnosti připojení zařízení k PC skrze UART-USB převodník. Tlačítkem bude uživatel přepínat módy, kanály a úrovně. Lokální režim bude mít následující vlastnosti: nastavení úrovní kanálů, odchytávání pulsů, prověření logické úrovně a generace pravidelných pulsů.

Terminálový režim bude poskytovat konkrétní měření veličin logického obvodu a diagnostiku sběrnic. Sonda bude v tomto režimu ovládána odesíláním symbolů za pomocí periferie UART skrze převodník UART-USB. Sonda takto poskytne uživatelské rozhraní, které se vygeneruje na straně mikrokontroleru a zobrazí skrze terminálovou aplikací podporující tzn. ANSI sekvence[1]. Oproti ovládání prostřednictvím specializované aplikace vyvinuté namíru sondě, tento přístup zajišťuje přenositelnost napříč operačními systémy a nenutí uživatele instalovat další software, což je výhodné zejména na sdílených zařízeních, jako jsou fakultní počítače.

Sonda v tomto režimu bude nabízet funkce základní a pokročilé. Mezi základní funkce patří: detekce logických úrovní, detekce impulsů, určení jejich frekvence, nastavení logických úrovní, generace impulsů, měření napětí a měření odporu. Mezi pokročilé náleží diagnostika sběrnic UART, I2C, SPI a Neopixel. Sběrnice sonda bude pasivně poslouchat nebo aktivně vysílat. Získaná data budou zobrazována skrze terminálovou aplikaci.

2.3 Volba mikrokontrolerů pro realizaci sondy

2.3.1 MCU STM32G031

Pro návrh v této bakalářské práci byl zvolen mikrořadič STM32G031 od firmy STMicroelectronics [9]. Tento mikrořadič je vhodný pro aplikace s nízkou spotřebou. Je postavený na 32bitovém jádře ARM Cortex-M0+, které je energeticky efektivní a nabízí dostatečný výkon pro běžné vestavné aplikace. Obsahuje 64 KiB flash paměť a 8 KiB SRAM [2]. MCU

[1] Např. PuTTY, GTKTerm…

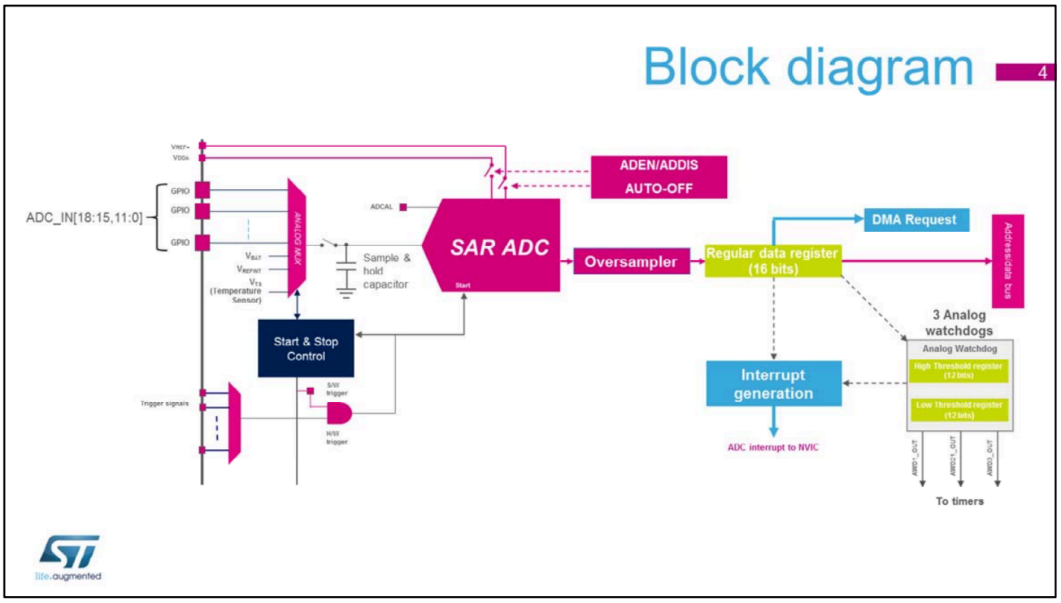
umožňuje frekvenci až 64 MHz, kterou je možné měnit pomocí PLL [2], [10]. Mikrokontroler nabízí i periferie jako USART, I2C nebo SPI pro rychlou seriovou komunikaci s dalším zařízením nebo senzory. Pro řadu G031 jsou typické kompaktní rozměry ať už vývojové Nucleo desky, tak typové pouzdra jako například **TSSOP20** nebo **SOP8**, což poskytuje snadnou integraci do kompatního hardwarového návrhu [8]. Obě zmíněná pouzdra jsou použita pro implementaci logické sondy, o které pojednává kapitola 3. V rámci realizace je použité MCU **STM32G030**, které je s kompatibilní s návrhem logické sondy.

Analogo-digitální převodník

Mikrokontrolér STM32G031 je vybaven analogo-digitálním převodníkem[2], který obsahuje 8 analogových kanálů o rozlišení 12 bitů. Maximální vzorkovací frekvence převodníku je 2 MSPS. Při měření kanálů se postupuje sekvenčně, která je určená pomocí tzv. ranků[3]. Při požadavku o měření převodník nejprve změří první nastavený kanál, při dalším požadavku druhý a až změří všechny, tak pokračuje opět od počátku. Aby během měření bylo dosaženo maximální přesnosti, převodník podporuje tzv. oversampling[4]. Převodník obsahuje **accumulation data register**, který přičítá každé měření a poté pomocí data shifteru vydělí počtem cyklu, kde počet cyklů je vždy mocnina dvojky z důvodu složitého dělení na MCU [1]. Tato metoda zamezuje rušení na kanálu.

$$\text{měření} = \frac{1}{2^m} \times \sum_{n=0}^{n=N-1} \text{Konverze}(t_n) \tag{1}$$

AD převodník, po dokončení měření vzorků, vrací hodnotu, která není napětí. Tato hodnota je poměrná hodnota vůči napájecímu napětí vyjádřena 12 bitově[5]. Pro převedení hodnoty převodníku na napětí je nutné znát referenční napětí systému ($V_{\text{REF}+}$). Referen-



Obrázek 1: Blokový diagram AD převodníku [1]

[2]Neboli ADC

[3]Rank určuje v jakém pořadí je kanál změřen.

[4]Proběhne více měření a následně jsou výsledky např. zprůměrovány aby byla zajištěna větší přesnost.

[5]Např. hodnota 4095 značí, že naměřené napětí je stejně velké jako napájecí napětí.

ční napětí může být proměnlivé, hlavně pokud systém využívá VDDA[6] jako referenci, která může být 2 V až 3.6 V a také může kolísat vlivem napájení nebo zatížení. Pro výpočet $V_{\mathrm{REF+}}$ se používá interní referenční napětí V_{REFINT} kalibrační data uložená během výroby mikrořadiče a naměřené hodnoty z ADC [2].

Vztah pro výpočet je následující:

$$V_{\mathrm{REF+}} = \frac{V_{\mathrm{REFINT\_CAL}} \times 3300}{V_{\mathrm{REFINT\_ADC\_DATA}}} \tag{2}$$

kde:

- $V_{\mathrm{REFINT\_CAL}}$ je kalibrační hodnota interního referenčního napětí, která je uložená ve flash paměti mikrořadiče během výroby. Tato hodnota představuje digitální hodnotu, kdy $V_{\mathrm{REF+}}$ je přesně `3.3 V`. Hodnota se získává čtením z pevné adresy[7] [2], [11].
- 3300 je konstanta odpovídající referenčnímu napětí při kalibraci ve výrobě vyjádřená v milivoltech.
- $V_{\mathrm{REFINT\_ADC\_DATA}}$ je aktuální naměřená hodnota na AD převodníku. Tato hodnota závisí na aktuálním napětí na napájení.

Po zjištění referenčního napětí dle Rovnice 2, lze získat na základě referenčního napětí, velikosti převodníku a hodnoty naměřené převodníkem, dle Rovnice 3. Rozlišení v případě tohoto zařízení bude 12 bitů. Počet bitů je podstatný pro určení, jaká hodnota je maximální, neboli referenční napětí.

$$V_{\mathrm{CH}} = \frac{V_{\mathrm{CH\_ADC\_DATA}}}{2^{\mathrm{rozlišení}} - 1} \times V_{\mathrm{REF+}} \tag{3}$$

kde:

- V_{CH} je napětí naměřené na daném kanálu.
- $V_{\mathrm{CH\_ADC\_DATA}}$ je digitální hodnota získaná z AD převodníku.
- rozlišení je počet bitů AD převodníku.
- $V_{\mathrm{REF+}}$ je referenční hodnota napětí.

Časovače

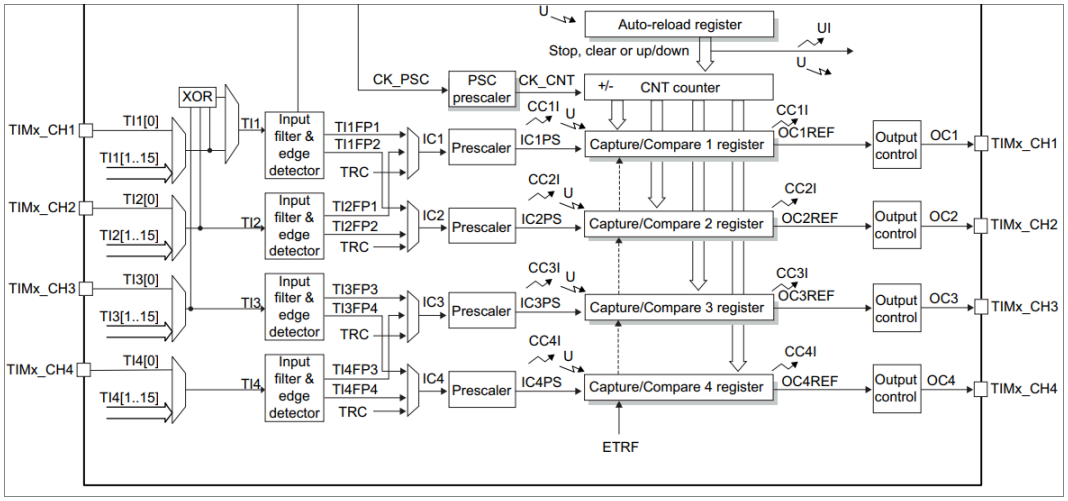
STM32G031 obsahuje několik časovačů (timeru), které se dají využít pro logickou sondu. Mikrořadič má zabudovaných několik základních a jeden pokročilý časovač. Základní časovače jsou 16 bitové a jsou vhodné pro měření doby či generování jednoduchých PWM signálů. Pokročilý časovač je na tomto mikrokontroleru 32bitový a poskytuje více kanálů. Tyto časovače také podporují nejen generování signálů na výstup, ale také zachytávání signálů a měření délky pulzů externího signálu. Pokročilý časovač nabízí řadu nastavení např. nastavování mezi normálním a inverzním výstupem PWM, generovat přerušení při dosažení specifické hodnoty časovače apod. [12].

Před spuštěním časovače je potřeba nastavit, jak často má časovač čítat. Frekvenci časovače nastavuje tzn. prescaler, neboli „předdělička“. Prescaler dělí s konstantou, která je zvolena, frekvenci hodin dané periferie. Pro případ STM32G0 je to 64 MHz. Frekvence časovače určuje, jak často časovač inkrementuje svou hodnotu za jednu sekundu [2].

$$F_{\mathrm{TIMx}} = \frac{F_{\mathrm{clk}}}{\mathrm{Prescaler} + 1} \tag{4}$$

[6] VDDA je označení pro analogové napájecí napětí v mikrokontrolérech STM32.

[7] Např. u STM32G0 je adresa kalibrační hodnoty: 0x1FFF75AA



Obrázek 2: Blokové schéma STM32G031 časovače [2]

Velikost čítače časovače, zda je 16bitový nebo 32bitový[8], souvisí s jeho tzv. periodou. Perioda určuje hodnotu, při jejímž dosažení se čítač automaticky resetuje na 0. Tuto hodnotu lze nastavit podle potřeby vývojáře. V kombinaci s prescalerem lze nastavit konkrétní časový interval, který je požadován. Časový interval lze vypočítat pomocí rovnice 5.

$$T = \frac{(\text{Prescaler} + 1) \times (\text{Perioda} + 1)}{F_{\text{clk}}} \tag{5}$$

2.3.2 Knihovna STM HAL

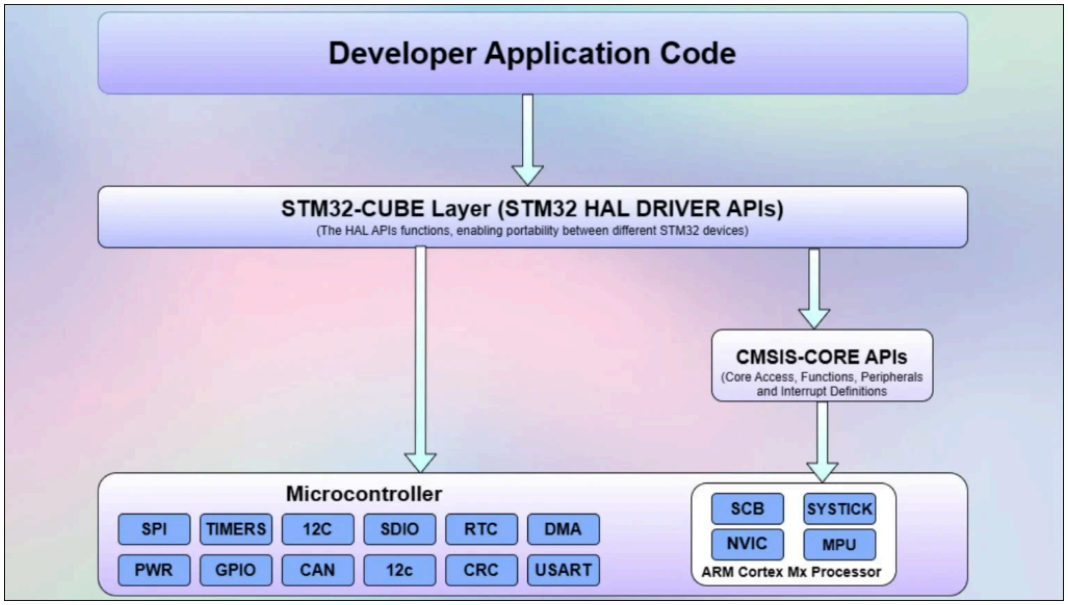
Hardware Abstraction Layer (HAL) od společnosti STMicroelectronics je knihovna určená pro mikrořadiče řady STM32, která slouží jako abstrakční vrstva mezi aplikací a hardwarem zařízení. Jejím hlavním cílem je zjednodušit vývojářům práci s periferiemi, jako jsou GPIO piny, komunikační rozhraní USART, SPI nebo I2C, a to bez nutnosti přímého zápisu do hardwarových registrů procesoru. HAL je součástí širšího ekosystému STM32Cube, který zahrnuje také nástroje pro konfiguraci mikrokontrolérů (např. STM32CubeMX) a generování kódu [13].

Mezi klíčové vlastnosti HAL je přenositelnost kódu. Protože různé modely MCU STM32 mohou mít odlišné mapování paměti nebo specifické hardwarové vlastnosti, HAL tyto rozdíly abstrahuje. Pokud vývojář potřebuje převést aplikaci na jiný čip z řady STM32, nemusí ručně upravovat adresy registrů a měnit logiku ovládání periferií, ale stačí využít nástroje na konfiguraci jako STM32CubeMX zatímco aplikační kód zůstává nezměněn. Tento přístup šetří čas a snižuje riziko chyb při portování projektů. Obrázek 3 znázorňuje diagram, který znázorňuje architekturu HAL [14].

Součástí HALu je tzv. CMSIS, což je sada standardizovanách rozhraní, které umožňují konfiguraci periferií, správu jádra, obsluhu přerušení a další [15]. CMSIS je rozdělen do modulárních komponent, kdy vývojář může využít pouze části, které potřebuje. Např. CMSIS-CORE, která poskytuje přístup k jádru Cortex-M a periferiím procesoru, obsahuje

[8]U 16 bitového časovače je maximální perioda 65535, zatímco u 32 bitového časovače je to 4294967295.

definice registrů, přístup k NVIC apod. [15]. Hlavní rozdíl mezi CMSIS a HALu[9] STMicroelectronics je ten, že CMSIS je poskytnuto přímo ARM a slouží pouze na ovládání Cortex M procesorů zatímco část od STMicroelectronics poskytuje abstrakci periferií.



Obrázek 3: STM32CubeMX HAL architektura [3]

2.3.3 Raspberry Pi Pico

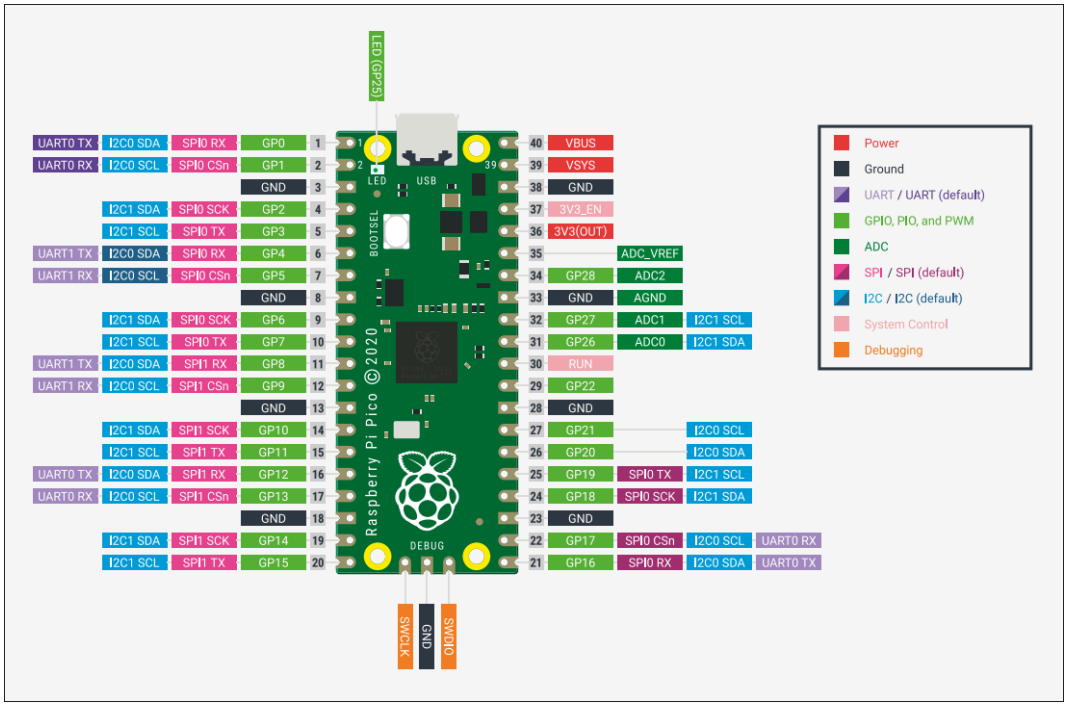
Pro omezenou verzi sondy byl zvolen mikrokontroler Raspberry Pi Pico. Tento kontroler obsahuje RP2040 s flash pamětí o velikosti 2 MiB, s celkem 40 piny [16]. RP2040 čip je navržen nadací Raspberry Pi Foundation a je postaven na Dual-core ARM cortex M0+, které dosahují frekvencí až 133 MHz, kterou je možné, stejně jako u STM32G031, měnit pomocí PLL [10], [16]. Tento mikrokontroler je poměrně populární mezi skupinou lidí, která tvoří projekty volnočasově zejména díky ceně, jednoduchého nahrávání programů do mikrokontroleru a komunitní podpoře.

Mikrokontrolér disponuje 26 GPIO piny s napětím 3,3 V (ne 5 V tolerantními), které podporují funkce jako pull-up/pull-down rezistory, hardwarová přerušení, PWM či komunikaci přes UART, SPI nebo I2C. Vestavěný 12-bitový ADC umožňuje měření napětí na třech analogových pinech s volitelnou referencí (interní 3,3 V nebo externí). Raspberry Pi Pico je navržené tak, aby bylo možné jej napájet z USB nebo i z externích zdrojů jako baterie.

PIO

Programmable Input/Output blok je unikátní funkcí MCU RP2040, která poskytuje implementaci vlastního rozhraní. RP2040 má tzv. 2 PIO bloky, kde každý blok se dá přirovnat k nezávislému malému procesoru, kde můžou běžet instrukce nezávisle na Cortex-M0+ jádře. Tyto bloky umožňují spravovat vstupy a výstupy pinů s velice přesným časováním nezávisle na zátěži CPU. Každý blok má 4 stavové automaty, které mohou nezávisle na sobě spouštět instrukce, které jsou uloženy ve sdílené instrukční paměti. Každý stavový

[9] STMicroelectronics do svého HALu zabaluje i CMSIS od ARM.



Obrázek 4: Raspberry Pi Pico vývojová deska

automat může manipulovat s GPIO a přenášet data do CPU a číst data poslané z CPU. PIO blok má speciální assembler, který obsahuje celkem 9 instrukcí (JMP, WAIT, SET atd.).

2.4 Měření veličin testovaného obvodu

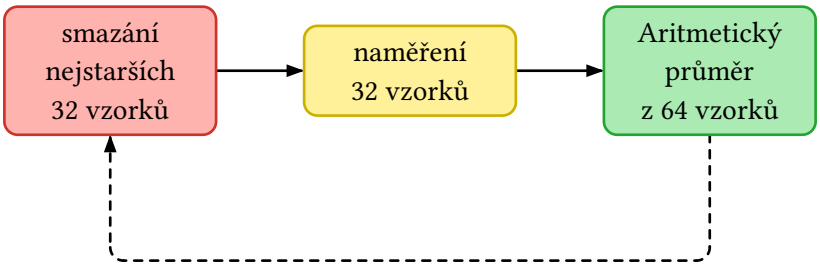
2.4.1 Měření napětí a logických úrovní

Pro měření napětí, jak již zmiňuje kapitola 2.3.1.1, je využíván AD převodník. Při měření napětí může docházet k šumu na vstupu kanálu a naměřená hodnota nemusí odpovídat realitě. Pro snížení vlivu šumu je použito tzn. sliding window. Do okna se uloží 32 vzorků měření do dvou bloků tj. 64 vzorků celkem. Každých 250 ms se provede průběžné měření 32 vzorků (vzorkovací frekvence ∼128 Hz). Nejstarší blok 32 vzorků je odstraněn a nahrazen novými daty.

$$V = \frac{\sum_{i=0}^{2^5}(V_{\text{staré i}}) + \sum_{i=0}^{2^5}(V_{\text{nové j}})}{2^6} \tag{6}$$

Tento přístup kombinuje stabilitu dlouhodobého průměru s reakcí na aktuální změny. Po aktualizaci okna, které probíhá každých 250 ms, se vypočítá aritmetický průměr z celého okna (64 vzorků), který reprezentuje výsledné napětí[10]. Počet vzorků byl zvolen v mocninách dvojky z důvodu, že dělení může probíhat jako bitový posun, jelikož dělení na MCU je pomalé a paměťově náročné. Měření s frekvencí vyšší než 100 Hz zajistí, že dojde k potlačení rušení 50 Hz, které se může na vstupu vyskytnout[11].

[10] Jedná se o klouzavý průměr.
[11] Dojde k eliminaci aliasingu.



Obrázek 5: Diagram způsobu sbírání vzorků z ADC

Pro zjistění stavu logické úrovně, je nutné vědět nejvyšší možné napětí nízké logické úrovně a nejnižší možné napětí vysoké logické úrovně. Pro CMOS logiku, Rovnice 7 popisuje definici nejvyššího napětí nízké logické úrovně a Rovnice 8 definuje nejnižší napětí logické vysoké úrovně [17]. Po změření napětí na kanále pomocí sliding window bude zkontrolováno zda napětí odpovídá logické úrovni nebo je napětí v nedefinované oblasti neboli v oblasti: $V_{\mathrm{ILmax}} < V < V_{\mathrm{IHmin}}$. Napětí v této oblasti v realném obvodu může vést k nepredikovatelnému chování, proto logická sonda toto napětí detekuje.

$$V_{\mathrm{ILmax}} = 0.3 \times V_{\mathrm{dd}} \tag{7}$$

$$V_{\mathrm{IHmin}} = 0.7 \times V_{\mathrm{dd}} \tag{8}$$

2.4.2 Měření odporu

Pro měření neznámého odporu R_x je využit AD převodník (viz kapitola 2.3.1.1) v kombinaci s napěťovým děličem, jehož schéma je znázorněno na obrázku Obrázek 6. Rezistory $R_{\mathtt{norm}}$ (normálový rezistor) a R_x (měřený odpor) jsou zapojeny v sérii, čímž tvoří uzavřenou smyčku. Podle Kirchhoffova napěťového zákona platí, že součet úbytků napětí na obou rezistorech je roven napájecímu napětí [18], [19]:

$$V_{\mathrm{dd}} = V_{R_{\mathtt{norm}}} + V_{R_x} \tag{9}$$

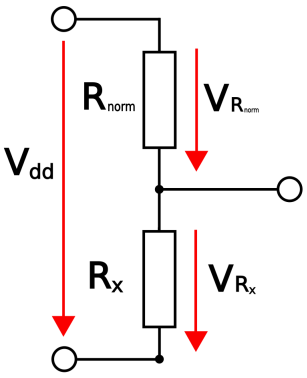
Pomocí pravidla pro napěťový dělič lze vztah mezi napětími a odpory vyjádřit jako:

$$V_{R_x} = V_{\mathrm{dd}} \times \frac{R_x}{R_{\mathrm{norm}} + R_x} \tag{10}$$

Za předpokladu, že $R_{\mathtt{norm}}$ a napájecí napětí V_{dd} jsou známé, a hodnota V_{R_x} je změřena ADC, lze neznámý odpor R_x vypočítat z upravené rovnice 10 [18]:

$$R_x = R_{\mathrm{norm}} \times \frac{V_{R_x}}{V_{\mathrm{dd}} - V_{R_x}} \tag{11}$$

V praxi probíhá měření neznámého odporu R_x následujícím způsobem: Uživatel sestaví napěťový dělič skládající se z normálového rezistoru $R_{\mathtt{norm}}$ (typicky 10 $k\Omega$) a měřeného rezistoru R_x. Normálový rezistor je připojen mezi referenční napětí V_{dd} a vstup ADC (kanál 1), zatímco R_x je zapojen mezi tento vstup a zem (viz schéma Obrázek 6). Sonda následně změří napětí V_{dd} na kanálu 1 ADC a pomocí rovnice 11, vypočítá hodnotu neznámého odporu [18].



Obrázek 6: Schéma děliče napětí

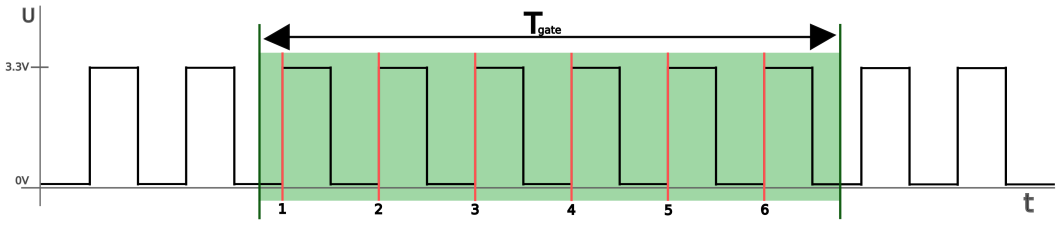
2.4.3 Měření frekvence

Metoda hradlování

Pro měření frekvencí v řádu KHz a MHz je využívána metoda hradlování. Tato metoda využívá čítače, který registruje počet náběžných nebo sestupných hran měřené frekvence N, za určitý čas T_{gate}. Čas, po který jsou počítány hrany se nazývá hradlovací (angl. gate time). Frekvence f_{gate} touto metodou je vypočítán pomocí rovnice 12. Délka hradlovacího času může mít vliv na výsledek a proto není vhodné volit jeden čas, pro všechny druhy frekvencí. Pokud bude zvolen čas příliš dlouhý, může to zpomalovat měření a také může nastat problém na straně omezenosti hardwaru, kdy při měření vysoké frekvence může dojít k přetečení čítače. V případě příliš krátkého času dojde k nepřesnosti měření a případě nízkých frekvencích nemusí dojít k zachycení správného počtu hran. Proto v případě sondy bude čas volitelný uživatelem.

$$f_{\text{gate}} = \frac{N}{T_{\text{gate}}} \tag{12}$$

Pro měření metodou hradlování je využit časovač v režimu čítání pulzů, a další časovač pro měření času hradlování, kde tyto dva časovače jsou mezi sebou synchronizovány aby nedocházelo k velké odchylce mezi časem zahájení resp. ukončení činosti čítače a časovače hradlovacího času, případě odchylky může dojít k počítání hran, které nejsou v okně hradlovacího času. Časovač pro měření hradlovacího času je závislý na oscilátoru MCU, kde odchylka frekvence oscilátoru periferie může ovlivnit realný čas měření a ve finále také výslednou frekvenci. Prakticky tato metoda neumožňuje změřit střídu PWM signálu, protože jsou počítány pouze pulzy.



Obrázek 7: Signál při měření metodou hradlování

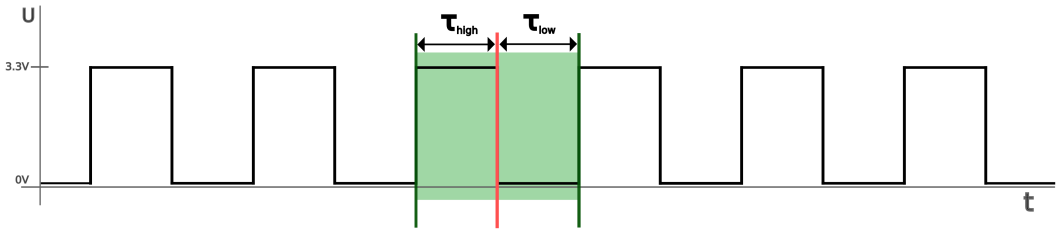
Reciproční frekvence

Reciproční frekvence vhodná pro měření nízkých frekvencí f_{rec} (typicky < 1 KHz). Na rozdíl od hradlování nepočítá hrany za pevný čas, ale měří periodu signálu T, ze které frekvenci dopočítá vztahem 13. Perioda je detekována pomocí náběžné/sestupné hrany, kdy se zahájí měření a měření je ukončeno při další náběžné/sestupné hraně. Během této doby se počítají pulsy interního oscilátoru MCU.

$$f_{\text{rec}} = \frac{1}{T} \tag{13}$$

Výhoda této metody je možnost výpočtu střídy PWM signálu. V případě, že místo měření celé periody, může být změřen čas od náběžné hrany k sestupné hraně a poté od sestupné k náběžné hraně. Tímto je možno získat šířku pulzu ve vysoké logické úrovni a šířku pulzu v nízké logické úrovni. Pomocí rovnice 14 je možné dopočítat střídu PWM.

$$D = \frac{\tau_{\text{high}}}{\tau_{\text{high}} + \tau_{\text{low}}} \tag{14}$$



Obrázek 8: Signál při měření reciproční frekvence

2.5 Analýza komunikačních rozhraní

Kapitola 2.1 zmiňuje testování hardwarových částí obvodu. Analýza seriové komunikace je častá nutnost při hledání chyby v implementaci studenta či vývojáře nebo jako otestování funkčnosti součástky. Logická sonda poskytne prostředí pro pasivní poslouchání komunikace, které pomůže vývojáři nalézt chybu v programu nebo studentovi při realizaci školního projektu.

2.5.1 UART

Universal Asynchronous Reciever Transmiter je rozhraní, kde data jsou odesílána bez společného hodinového signálu mezi odesílatelem a přijemcem. Místo toho je podstatný baudrate[12], což určuje počet přenesených bitů za sekundu. UART podporuje nastavení různých protokolů komunikace jako například RS-232 a RS-485. UART také umí full duplex komunikaci [20], [21].

Data jsou přenášena v tzv. rámcích, které jsou strukturovány následovně: [21]

- **Start bit** - Každý rámec začíná start bitem, který určuje začátek rámce. Bit je vždy „0“.
- **Slovo dat** - Poté následuje 8 bitů dat[13].
- **Paritní bity** - Paritní bity slouží k detekci chyby v přenosu. Parita nám dokáže pouze detekovat chybu rámce pouze v případech, kdy nevznikne chyb více[14].

[12] Počet bitů přenesených za sekundu

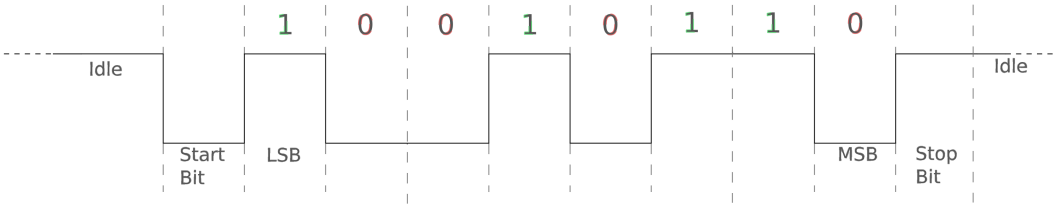
[13] Lze používat i 7 bitů nebo 9 bitů dat.

[14] Chyba z dat 1101 na 1000 nelze detekovat, protože lichá parita má paritní bit v obou případech 0.

- **Stop bit** - Stop bit[15] signalizuje konec přenosu rámce. Obvykle logická „1“.

Pokud rozhraní neodesílá žádné bity, na vodičích se nachází vysoká úroveň. Této vlastnosti bude využito později v návrhu logické sondy.

V logické sondě je UART využíván, ke komunikaci s PC a také logická sonda umí toto rozhraní pasivně sledovat i aktivně odesílat testovací sekvence Obrázek 9 ukazuje způsob zpracování signálů [22]. Testování tohoto rozhraní je potřeba pokud student či vývojář potřebuje najít chybu např. při implementaci seriové komunikace mezi dvěma mikrokontrolery, kde se právě často využívá UART.

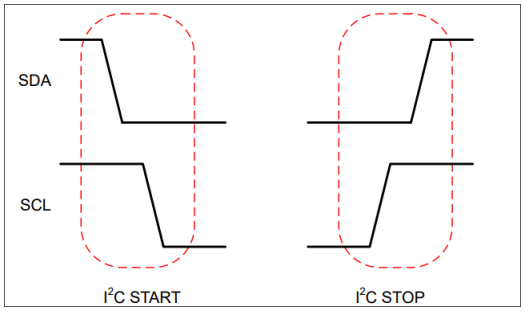


Obrázek 9: Způsob zpracování signálu UART bez parity

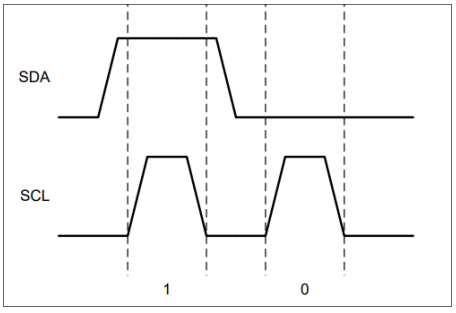
2.5.2 I2C Bus

Inter-Integrated Circuit je seriová komunikační sběrnice, který byl vytvořen Philips Semiconductor jako nízkorychlostní sběrnice pro propojení zařízení jako např. mikrokonkrolery a procesory se senzory, periferiemi apod. Od roku 2006 implementace protokolu nevyžaduje licenci a proto se začal široce používat např. v IOT. Výhoda sběrnice je, že pro komunikaci potřebuje pouze dva vodiče, na které je možné připojit až 128 zařízení najednou, jelikož využívá systém adres [4].

SCL vodič, slouží jako hodinový signál a SDA vodič slouží jako datový vodič. Protokol rozlišuje zařízení typu master a slave. Master řídí hodinový signál a protože je I2C obousměrný half duplex protokol, tak master zahajuje a zastavuje komunikaci aby nedocházelo ke konfliktům. Oba vodiče mají otevřený kolektor, z důvodu, že je na lince připojeno více zařízení a vodiče jsou pull up rezistorem přivedeny na společný zdroj napětí, což znamená, v klidovém režimu, jsou na vodičích vysoké logické úrovně [4].

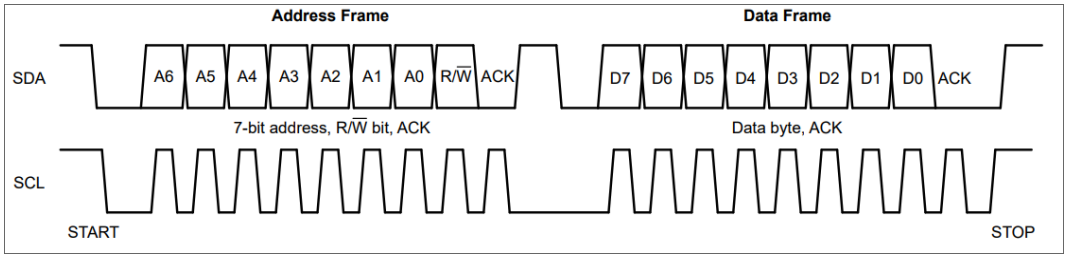


Obrázek 10: Zahájení a ukončení komunikace v I2C [4]



Obrázek 11: Logická jednička a logická nula v I2C [4]

[15] Stop bitů může být i několik



Obrázek 12: Rámce I2C [4]

Pro zahájení komunikace, master is zarezervuje sběrnici posláním `I2C START`. Nejprve master přivede vodič SDA do nízké logické úrovně a následně tam přivede i SCL. Tato sekvence indikuje, že se master zahajuje vysílání a všechny zařízení poslouchají. Pro ukončení je nejprve uvolněn SCL a až poté SDA. Tím je signalizováno, že komunikace je ukončena a jiný master může komunikovat.

Obrázek 11 ukazuje způsob odesílání jednotlivých bitů. Pro odeslání logické jedničky SDA uvolní vodič, aby byla přivedena přes pull up rezistor vysoká logická úroveň. Pro logickou nulu, vysíláč stáhne vodič do nízké úrovně. Přijimač zaznamená bit v momentě, kdy SCL zapulsuje.

Protokol rozděluje bity do rámců. Rámec má vždy 8 bitů. Nejprve pošle adresový rámec, který identifikuje, který slave má reagovat. Součástí rámce je také read-write bit. Pokud slave přečte adresový rámec a daná adresa mu nepatří, ignoruje komunikaci. V opačném případě odpoví `ACK` bitem, kdy nízká úroveň znamená potvrzení. Vysoká úroveň nastane, když slave nezareaguje a vodič zůstane v klidu, tzn. vysoká úroveň.

Po identifikaci se zahájí odesílání datových rámců, které se skládají z 8 bitů a jsou zakončeny `ACK`. Pokud byl read-write bit nastaven na read, master většinou zašle adresu z které chce číst a slave pošle obsah paměti. Při write, master zašle adresu na kterou chce zapisovat a následně data, které chce zapsat [4].

2.5.3 SPI

Serial peripherial interface je jeden z nejvíce využívaných rozhraní používaný mezi mikrokontrolery a periferiemi jako např. AD převodníky, SRAM, EEPROM apod. Rozhraní nemá definované jaké napětí se používají a ani velikosti rámců. Typicky se používá 8 bitů. Oproti UART a I2C vyniká rychlostí komunikace, která je v řádu MHz.

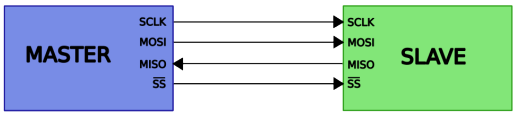
SPI má vždy jedno master zařízení a i několik podřízených slave zařízení. SPI je synchronní full duplex rozhraní, které má celkem 4 vodiče[16]. `SCLK`, což je hodinový signál, který určuje synchronizaci dat, `MOSI`, neboli vodič, kde probíhá komunikace od masteru ke slave, `MISO`, kde probíhá komunikace od slave k masteru a poté `SS`[17], neboli slave select. Ten určuje, se kterým slave zařízením probíhá komunikace, každé zařízení má vlastní `SS` pin [23]. Obrázek 13 ukazuje způsob zapojení vodičů v případě jednoho slave zařízení. Pro zahájení komunikace nastaví logicky nízkou úroveň na `SS`[18]. Master zahájí generování hodinového signálu, podle kterého se synchronizuje komunikace. Jelikož je

[16] 4 vodiče má v případě jednoho slave zařízení. S každým dalším slave zařízením musí být připojen další `SS`.

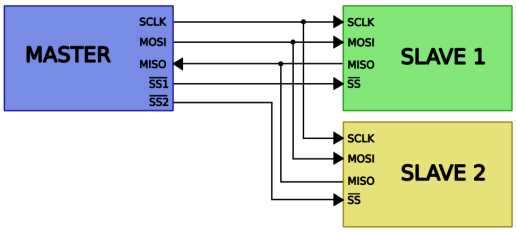
[17] Někdy také `CS` jako chip select.

[18] Jelikož se spíná vodič do log. 1, tak má značka `SS` nad sebou negaci.

komunikace full duplex, komunikace začne probíhat mezi master a slave oběma směry tzn. na vodičích `MISO` a `MOSI`. Po dokončení komunikace master ukončí vysílání hodinového signálu a nastaví `SS` na vysokou logickou úroveň. Obrázek 14 znázorňuje připojení více slave zařízení. V tomto případě master využívá více `SS` a podle přivedení nízké logické úrovně určuje směr komunikace [23].

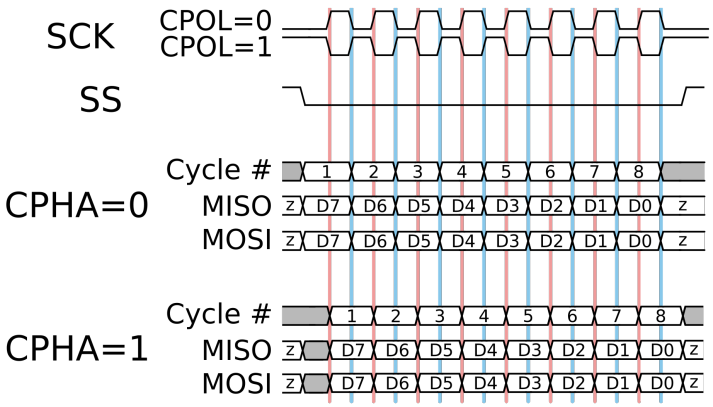


Obrázek 13: Diagram SPI komunikace s jedním slave zařízením



Obrázek 14: Diagram SPI komunikace s více slave zařízeními

Zjištění logické 0 a 1 vychází z přečtení logické úrovně v momentě vzestupné nebo sestupné hraně hodinového signálu. Vztah mezi hodinovým signálem a daty tzn. `CPOL` bitem a `CPHA` bitem. `CPOL` bit určuje, jakou logickou úroveň má klidový stav hodinového signálu. Při log. 0 je klidová úroveň nízká a hodinový signál započne náběhovou hranou, při log. 1 naopak. `CPHA` určuje, jaká hrana má určovat logickou úroveň signálu, při log. 0 je čtena hodnota při první hraně signálu, při log. 1 je čtena hodnota při druhé hraně hodinového signálu[19].

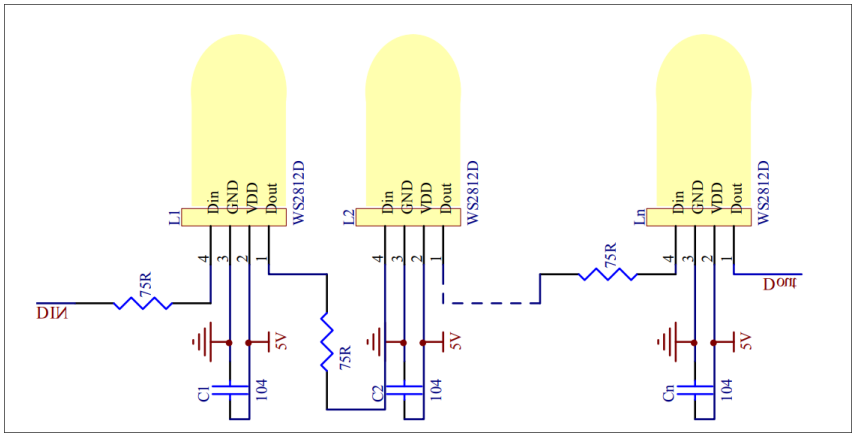


Obrázek 15: Časový diagram SPI zobrazující úroveň a posun hodinového signálu [5]

2.5.4 Neopixel

Neopixel je název pro kategorii adresovatelných RGB LED. Dioda má celkem 4 vodiče: ground, Vcc, DIn a DOut. LED má vlastní řídící obvod, který ovládá barvy diody na základě signálu z vodiče DIn. Výhoda LED je možnost připojit diody do serie, a jedním vodičem ovládat všechny LED v sérii [6]. Obrázek 16 znázorňuje zapojení více LED do série a schopnost ovládání jedním vodičem.

[19]Pokud bude `CPOL` bit = 0 a `CPHA` = 0, tak signál bude čten při náběžné hraně, při `CPOL` = 0 a `CPHA` = 1 bude čtena při sestupné hraně.



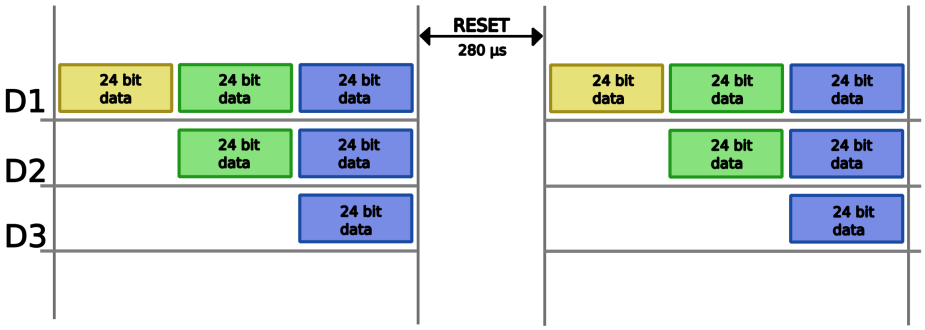
Obrázek 16: Způsob zapojení RGB LED do série [6]

Data do LED se zasílají ve formě 24 bitů, kdy každých 8 bitů reprezentuje jednu barevnou složku. Parametry pořadí složek, časování apod. se může lišit v závislosti na konkrétním verzi a provedení LED. V této práci je vycházeno z WS2812D. První bit složky je, v případě WS2812, MSB[20].

R7	R6	R5	R4	R3	R2	R1	R0	G7	G6	G5	G4	G3	G2	G1	G0	B7	B6	B5	B4	B3	B2	B1	B0

Tabulka 1: Pořadí bitů pro nastavení barvy v WS2812 [6]

Neopixel nepracuje na sběrnici s časovým signálem, proto je nutné rozpoznávat logickou jedničku a nulu jiným způsobem. Na pin DIn je přivedena vysoká úroveň na určitou dobu, poté je na určitou dobu přivedena nízká úroveň. Kombinace těchto časů dává řídícímu obvodu v LED možnost rozpoznat, jaký bit byl poslán diodě. Pro ovládání n LED, na DIn první LED je zasláno $n \times 24$ bitů. Dioda zpracuje prvních 24 bitů, a na Dout odešle $(n-1) \times 24$ bitů. Tento proces se opakuje pro každou LED v sérii a tím je dosaženo rozsvícení všech diod na požadovanou barvu. Aby řídící obvod rozpoznal, které data má poslat dále a která jsou už nová iterace barev pro LED, je nutné dodržet tzn. RESET time, kdy po uplynutí tohoto času, řídící obvod, už neposílá data dále, ale zpracuje je Tabulka 2 ukazuje časování pro WS2812D. Po testování odesílání dat do RGB LED bylo zjištěno, pro



Obrázek 17: Diagram posílání dat pro zapojené WS2812D v sérii [6]

[20] Most significant bit

LED jsou stěžejní časy vysokých úrovní, nicméně časy nízkých úrovní jsou více benevolentní a bylo například zjištěno, že nízká úroveň u log. 1 je možné zkrátit i na čas 330 ns.

Bit	Typ času	t[ns]
0	vysoká úroveň napětí	$220 \sim 380$
0	nízká úroveň napětí	$580 \sim 1000$
1	vysoká úroveň napětí	$580 \sim 1000$
1	nízká úroveň napětí	$580 \sim 1000$
RESET	nízká úroveň napětí	$> 280\ 000$

Tabulka 2: Časování logických úrovní pro zaslání bitů WS2812D [6]

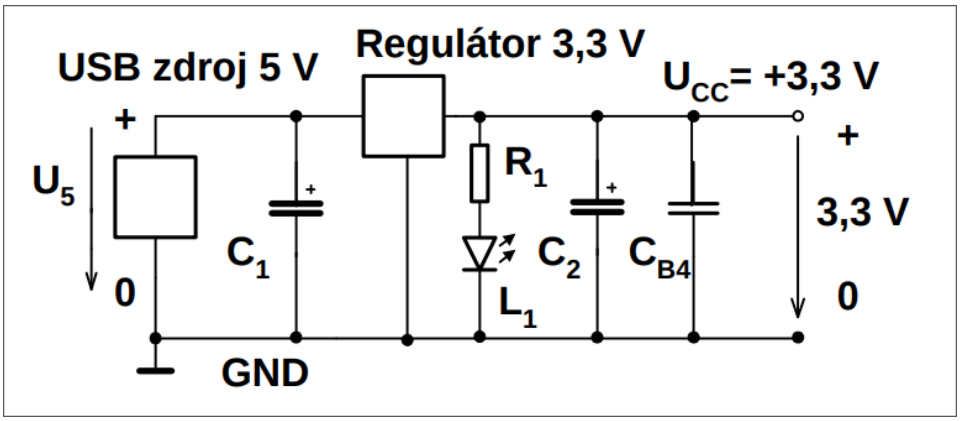
Kapitola 3

HW návrh logické sondy STM32

Návrhy obsahují, co nejméně komponent, aby student byl schopný zařízení jednoduše sestavit. Tzn. například pull up nebo pull down rezistory jsou řešeny interně na pinu. Logická sonda musí být ideálně co nejvíce kompatibilní mezi oběma pouzdry, tak aby byla zaručena přenositelnost a pravidla pro sestavení byla co nejvíce podobná.

3.1 Sdílené vlastnosti mezi návrhy pouzder

Sonda je napájena skrze UART/USB převodník. Jelikož USB pracuje s napětím 5 V ale STM32G030 vyžaduje napájecí napětí $1.7 \sim 3.6$ V [2] je nutné napětí snížit. Proto byl použit linearní stabilizátor **HT7533**, který stabilizuje napětí na 3.3 ± 0.1 V. Ke vstupu je připojen kondenzátor `C1` k potlačení šumu o velikosti 10 μF. K výstupu je připojen keramický kondenzátor[21] `C2` k zajištění stability výstupu o velikosti také 10 μF [24].



Obrázek 18: Zapojení regulátoru pro napájení STM32G030 [7]

Návrh zohledňuje implementaci lokálního režimu. Pro tuto implementaci je na pin `PA13` zapojeno tlačítko pro interakci s uživatelem vůči zemi s interním pull up rezistorem na pinu. Připojení vůči zemi minimalizuje riziko zkratu chybným zapojení uživatelem.

Dále je připojena WS2812 RGB LED na `PB6`. Tento pin byl zvolen z důvodu přítomnosti kanálu časovače, který je využit pro posílání dat skrze PWM do LED. WS2812 dle datasheetu vyžaduje napětí $3.7 \sim 5.3$ V [6]. Pokud by WS2812 byla napájena 5 V z USB převodníku, došlo by k problému s CMOS logikou, kdy vstupní vysoká logická úroveň je definována jako $0.7 \times V_{\mathrm{dd}}$, což se rovná 3.5 V a STM32 pin při vysoké úrovni má V_{dd}, což je ~ 3.3 V [2], [17]. Z toho důvodu je navzdory datasheetu LED připojena na napětí V_{dd} mikrokontroleru. Toto zapojení bylo otestováno a je plně funkční. Problém se kterým je možné se setkat je nesprávné svícení modré barvy z důvodu vysokého prahového napětí. Mezi katodu a anodu LED je umístěn blokovací kondenzátor o velikost 100 nF.

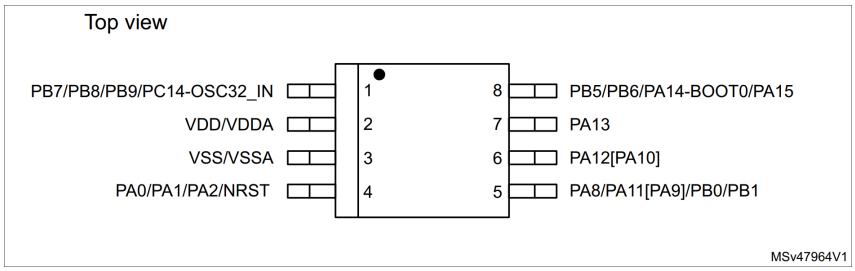
Obě pouzdra využívají pro komunikaci s PC periferii USART1. STM32 poskytuje možnost remapování pinů. Pro zjednodušení zapojení jsou piny `PA12` a `PA11` přemapované

[21] Keramický s důvodu, že LDO požadují nízké ESR

na `PA10` a `PA9`. Tyto piny jsou použity jako Tx a Rx piny UART komunikace. Pro zajištění funkce lokálního režimu je na Rx pin přiveden pull down rezistor o velikosti 10 KΩ.

3.2 SOP8

Obrázek 21[22] ukazuje zapojení STM32G030 v pouzdře SOP8. Toto pouzdro po zapojení napájení, rozhraní UART má k dispozici pouze 4 piny. Po zapojení potřebných komponent pro lokální režim, které zmiňuje Kapitola 3.1, zůstávají piny 2. Z tohoto důvodu, na pouzdro SOP8, jsou implementován pouze lokální režim a základní funkce terminálového režimu, jako měření napětí, frekvence a vysílání pulzů.



Obrázek 19: STM32G030Jx SO8N Pinout [8]

Jelikož je pouzdro malé, tak se na jednom fyzickém pinu nachází více periferií. Obrázek 19 ukazuje, že na pinu 4, kde se nachází `PA0`, má připojený i `NRST`. `NRST` požaduje aby pin byl neustále ve vysoké logické úrovni, což pro potřebu logické sondy je nepraktické protože takto není možné využít `PA0`. Funkce nresetu lze vypnout skrze tzv. **optional bits**. Kde na pozici `NRST_MODE` je potřeba nastavit `2`, aby NRST byl ignorován a `PA0` bylo použitelné. Pomocí nastavení bude zajištěno, že `NRST` bude ignorován po dobu běhu programu, nicméně při bootu je nežádoucí, aby byl přiveden na logicky nízkou úroveň, protože stále MCU v této fázi ignoruje nastavení optional bitu.

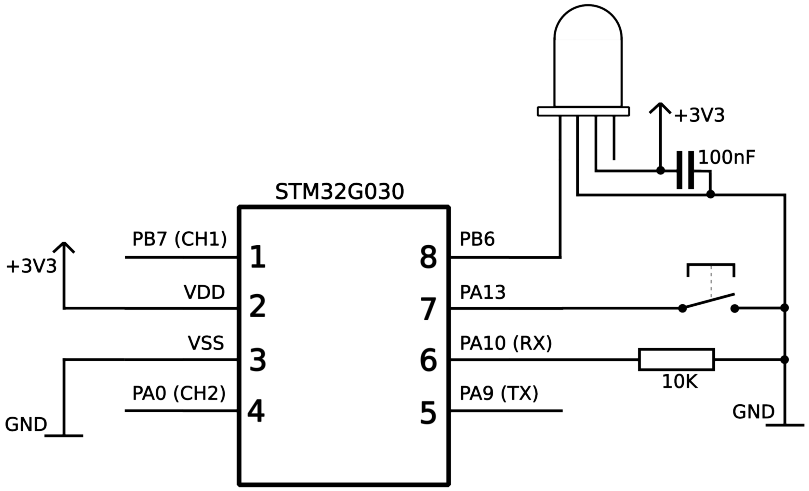
Address[1]	Corresponding option register (section)	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x1FFF7800	FLASH_OPTR (3.7.8)	Res.		IRHEN	NRST_MODE		nBOOT0	nBOOT1	nBOOT_SEL	Res.	RAM_PARITY_CHECK	DUAL_BANK	nSWAP_BANK	WWDG_SW	IWDG_STBY	IWDG_STOP	IWDG_SW	nRST_SHDW	nRST_STDBY	nRST_STOP	BORF_LEV		BORR_LEV		BOR_EN	RDP							
	Factory value	X	X	1	1	1	1	1	1	X	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	0	1	0	1	0

Obrázek 20: Paměťový prostor Flash option bits [2]

Další problém představuje pin 8, který obsahuje `PA14-BOOT0`. Při startu MCU bootloader zkontroluje bit **FLASH_ACR**, který určuje jestli je FLASH paměť prázdná. Pokud ano, MCU zapne a začne poslouchat periferie kvůli případnému stáhnutí firmwaru do FLASH paměti. Pokud FLASH prázdná není, program uložený v paměti se spustí. Pokud je na `PA14-BOOT0` ve vysoké logické úrovni, MCU se chová stejně, jako by paměť byla prázdná

[22] Schéma zapojení bylo zrealizováno pomocí nástroje *Autodesk Eagle* [25]. Komponenta Neopixel RGB LED byla použita jako externí knihovna [26].

[2]. Standartně se mikrokontroler nahrává a debuguje pomocí tzn. SWD[23], nicméně při této konfiguraci je to nepraktické, protože, by to znamenalo připojit ST-LINK k mikrokontroleru, nahrát, odpojit a poté až udělat zapojení, které ilustruje Obrázek 21. Pro jednoduchost se firmware nahraje pomocí UART. V tomto případě je ale potřeba řídit, zda má být nahráván firmware nebo spuštěn program. Optional bit `nBOOT_SEL` určuje, zda má být toto řízeno pomocí bitů `nBOOT0` a `nBOOT1` nebo pomocí úrovně `PA14-BOOT0`. V případě sondy, je potřeba druhá možnost, takže je nutné nastavit bit `nBOOT_SEL` na `0`.



Obrázek 21: Schéma zapojení STM32G030 v pouzdře SOP8

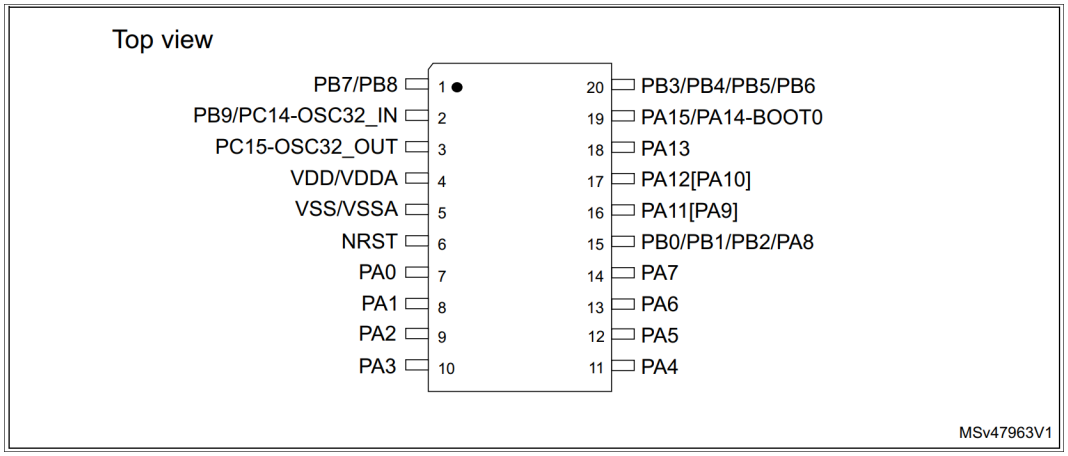
První z pinů k užívání je pin `PB7`. Tento pin slouží jako kanál AD převodníku pro měření napětí a pro měření odporu, pin je také využit pro hodinový signál pro posuvný registr. Na pinu `PA0` se nachází AD převodníkový kanál. Pin také disponuje kanály TIM2 časovače. Pin je použit jako druhý kanál AD převodníku pro měření napětí, pro posuvný registr je pin využíván pro posouvání dat do posuvného registru, měření frekvence, odchytávání Neopixel dat, detekce pulsů a generování frekvence. Pin `PB6` je použit pro odesílání dat do testované RGB LED.

3.3 TSSOP20

Pouzdro TSSOP20 nabízí oproti SOP8 výhodu většího počet pinů a tím pádem i jednodušší implementaci pokročilých funkcí. Pouzdro má celkem 20 pinů, což má za následek, že např. může být pin `NRST` (pin 6) oddělen od `PA0` a má tak vlastní pin. Z tohoto důvodu při flashování MCU není potřeba myslet na nastavení optional bits pro `NRST` a může zůstat v základním nastavení. Nicméně pin `PA14-BOOT0` musí být nastaven stejným způsobem jako u SOP8, tzn. optional bit `nBOOT_SEL` je nutné nastavit na `0` aby bylo možné při startu MCU určit, zda má být nabootován program ve FLASH paměti, nebo má poslouchat periferie pro nahrání programu.

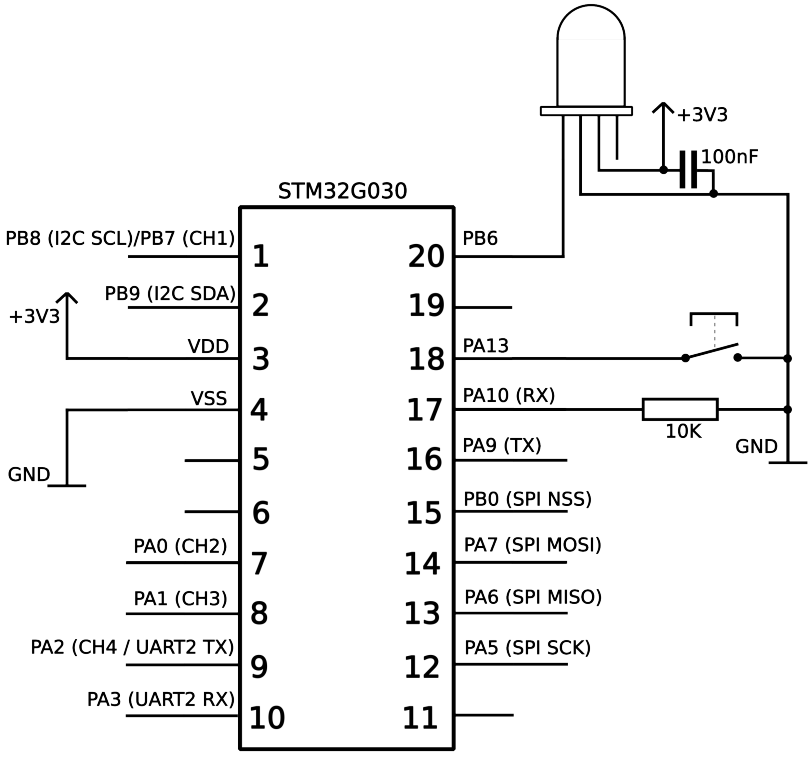
Pro zjednodušení sestavení sondy, je HW TSSOP20 návrh co nejvíce podobný návrhu SOP8. Piny `PA11` a `PA12` jsou přemapovány na `PA9` a `PA10`. Na pin `PA10` je připojen rezistor o velikosti 10 $K\Omega$ vůči zemi pro detekci komunikace s PC. Ze stejného důvodu byl zachován pin `PB6` jako výstup pro WS2812D a `PA13` pro tlačítko pro lokální režim. Obrázek 23

[23]Serial Wire Debug slouží pro jednoduší vývoj na mikrokontrolerech, je možné číst FLASH, RAM, nahrávat program, nastavovat option bity apod.



Obrázek 22: STM32G030Jx TSSOP20 Pinout [8]

ukazuje schéma zapojení s pouzdrem TSSOP20. Rozmístění pokročilých funkcí vychází z charakteristik jednotlivých pinů. Pin 1 (PB7) je využit stejně jako v pouzdře SOP8 jako první kanál ADC. Na pinu 1 (PB8) a 2 (PB9) se nachází I2C periferie a proto jsou využity pro sledování komunikace I2C sběrnice. Pin 7 (PA0) je k měření frekvence a napětí. Piny 9 (PA2) a 10 (PA3) mají USART periferii a proto jsou vhodní kandidáti na sledování UART komunikace. Piny 12 (PA5), 13 (PA6), 14 (PA7) a 15 (PB0) mají SPI rozhraní a proto jsou použity pro sledování SPI komunikace. V_{dd} je připojeno na výstup linearního stabilizátoru z obrázku 18, který má na výstopu $3.3\ V$.



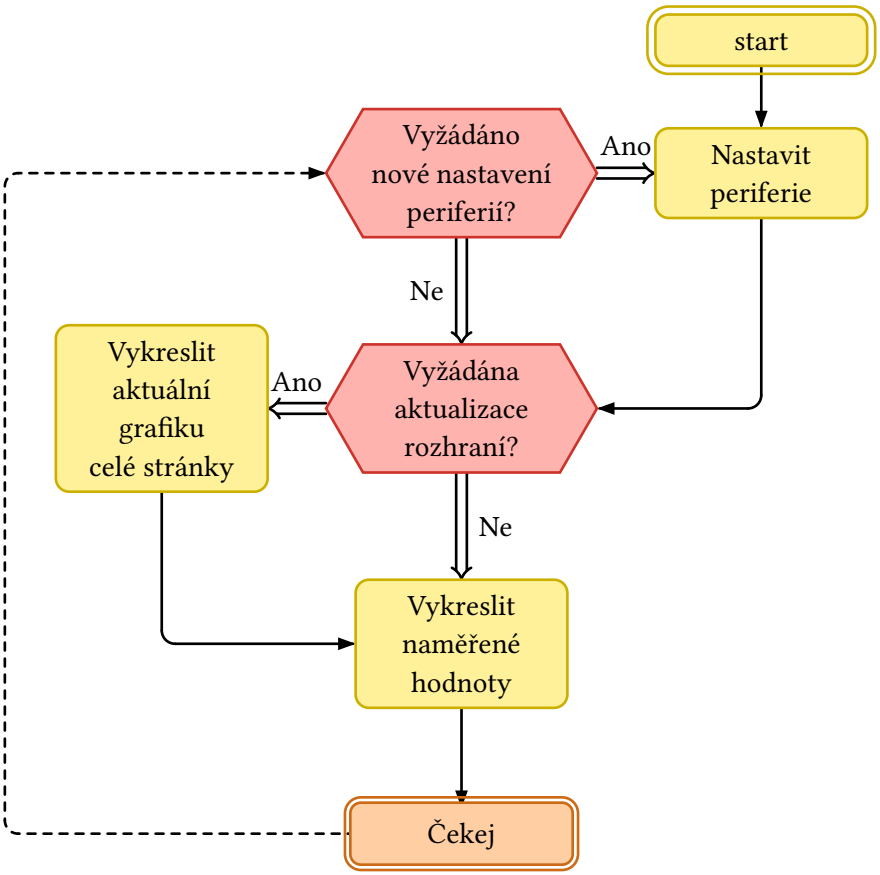
Obrázek 23: Schéma zapojení STM32G030 v pouzdře TSSOP20

Kapitola 4

Návrh terminál režimu STM32

4.1 Princip oblužní smyčky

Terminálový režim využívá rozhraní UART, pro sériovou komunikaci s PC. Způsob vstoupení do terminálového režimu rozebírá kapitola 5.1. Základ terminálového režimu běží v nekonečné smyčce, která je na konci oddělena čekáním[24]. Smyčka slouží jako obsluha akcí, které jsou vyvolány, jak uživatelem prostřednictví TUI, tak periferiemi, které momentálně běží. Obsluha při každé iteraci provede jednotlivé úkony, pokud příznaky v globální struktuře (Úryvek kódu 16) jsou nastaveny. Příznaky jsou běžně nastavovány skrze přerušení, například vyvolané uživatelem skrze odeslání symbolu seriovou komunikací. Obsluha v každé iteraci zkontroluje, zda příznak `need_frontend_update` vyžaduje vykreslit grafiku TUI (Kapitola 4.2), zda příznak `need_perif_update` vyžaduje změnit periferii (Kapitola 4.3), poté vykreslí data, které periferie získala a nakonec čeká na další smyčku. Sonda vykresluje data na základě `device_state` promněné, která určuje, jakou funkci uživatel momentálně používá.



Obrázek 24: Diagram smyčky terminálového módu

[24]Toto čekání se mění na základě zvolené funkce.

Metoda periodické obsluhy nastavování periferií a vykreslování TUI, oproti okamžité reakci přímo v přerušení, má výhodu v tom, že nemůže dojít k překrytí činností mezi hlavní smyčkou a přerušeními. Např. pokud bude stránka periodicky vykreslována, a stisk tlačítka by vyvolal přerušení k překreslení programu, může se přerušit smyčka v momentě, kdy už k překreslení dochází. V tomto případě poté dojde k rozbití obrazu vykresleného na terminál. Obdobná věc hrozí při vypínání a zapínání periferií. Kdy průběh deinicializace periferie přerušen a nastane inicializace, může dojít k nepredikovatelnému chování. Metodou obsluhy jsou definovány posloupnosti úkonů, které se nemohou překrývat.

4.2 Grafické řešení TUI

Kapitola 2.1 zmiňuje důraz na jednoduchou přístupnost ve výuce, což zahrnuje i jednoduché zobrazení informací, které uživatel potřebuje. Aby zprovoznění sondy bylo co nejvíce jednoduché, nebyla zvolena cesta ovládání skrze specialní aplikaci nebo specialní ovladač, ale byla zvolena cesta ovládání sondy skrze libovolnou terminálovou aplikaci podporující ANSI escape sekvence[25]. ANSI escape sekvence zajistí možnost grafického prostředí skrze terminál. Ke generaci rozhraní bude docházet na straně mikrokontroleru a posíláno UART periferií do PC. Tento způsob navíc zajistí nezávislost na operačním systému a je možné komunikovat se sondou na jakémkoliv populárním operačním systému.

4.2.1 Ansi sekvence

ANSI escape kódy představují standardizovanou sadu řídicích sekvencí pro manipulaci s textovým rozhraním v terminálech podporujících ANSI/X3.64 standard. Tyto kódy umožňují dynamickou úpravu vizuálních vlastností textu (barva, styl), pozicování kurzoru a další efekty, čímž tvoří základ pro tvorbu pokročilých terminálových aplikací [27].

Syntaxe

Základní syntaxe escape sekvencí pro formátování textu je:

```bash
\033[<parametry><akce>
```

- `\033` (ASCII 27 v osmičkové soustavě) označuje začátek escape sekvence[26]
- `[` je úvodní znak pro řídicí sekvence
- `<parametry>` jsou číselné kódy oddělené středníky
- `<akce>` je písmeno specifikující typ operace

Formátování textu

Pro změnu barvy a obecně textu je použito písmeno `m` jako akce. Nejčastější parametry s popisem vypisuje Tabulka 3.

[25] Například program PuTTY...
[26] Existují také `\033` N nebo `\033` \ apod.ale tyto se téměř nepoužívají.

Kód	Typ	Popis
30–37	Text · Základní	Černá, Červená, Zelená, Žlutá, Modrá, Purpurová, Tyrkysová, Bílá
90–97	Text · Světlé	Světlé varianty základních barev
40–47	Pozadí · Základní	Černé, Červené, Zelené… pozadí
100–107	Pozadí · Světlé	Světlé varianty pozadí
0	Efekt	Reset všech stylů
1	Efekt	Tučný text
4	Efekt	Podtržení
7	Efekt	Inverzní barvy

Tabulka 3: Tabulka akcí ANSI sekvencí

Manipulace s kurzorem

Sekvence také lze použít pro pohyb kurzoru, což je užitečné pro vizuál aplikace. Pro pohyb kurzoru na konkrétní pozici zajišťuje písmeno `H` a pro pohyb o relativní počet symbolů slouží písmena `A` jako nahoru, `B` jako dolu, `C` jako doprava a `D` jako doleva na pozici akce [28].

```bash
\033[<row>;<col>H // Pohyb na konkrétní pozici
\033[<posun><směr> // Posune kurzor o danou pozici
\033[10;15H // Posune kurzor na pozici 10. řádku a 15 sloupce
\033[10B // Posune kurzor o 10 řádků dolů
```

Mazání obsahu

ANSI escape kódy umožňují kromě formátování textu také dynamické mazání obsahu obrazovky nebo řádků, což je klíčové pro aktualizaci TUI. Tyto sekvence se využívají např. pro překreslování statických prvků nebo odstranění přebytečného textu [28]. .

```bash
    \033[2J // Smazání celého displeje
    \033[0K // Smazání textu od pozice kurzoru do konce řádku
    \033[1K // Smazání textu od pozice kurzoru do začátku řádku
    \033[2K // Smazání celého řádku
    \033[2KProgress: 75% // Smazání řádku a vypsání nového textu
```

4.2.2 Nastavení periferie pro zobrazování TUI

Pro komunikaci s PC je využito periferie `USART1`, která se nachází na pinech `PA11` a `PA12` respektivě `PA9` a `PA10`. Periferii je možné inicializovat pomocí programu STM32CubeMX, který po nastavení parametrů vygeneruje příslušné inicializační a deinicializační funkce. Pro komunikaci byl zvolen baudrate `115200` a 8 bitové slovo s jedním stop bitem bez parity,

```c
static void MX_USART1_UART_Init(void) {
    huart1.Instance = USART1;
    huart1.Init.BaudRate = 115200;
    huart1.Init.WordLength = UART_WORDLENGTH_8B; // velikost dat
    huart1.Init.StopBits = UART_STOPBITS_1; // počet stop bitů
    huart1.Init.Parity = UART_PARITY_NONE; // bez parity
    huart1.Init.Mode = UART_MODE_TX_RX; // Zapnut full duplex
    huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
    huart1.Init.OverSampling = UART_OVERSAMPLING_16;
    huart1.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
    huart1.Init.ClockPrescaler = UART_PRESCALER_DIV1;
    huart1.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
    if (HAL_UART_Init(&huart1) != HAL_OK) {
        Error_Handler();
    }
}
```

Úryvek kódu 1: Inicializace UART periferie

což například u programu PuTTY je základní nastavení, takže není nutné aby uživatel něco dalšího nastavoval.

4.2.3 Vykreslování stránek

Stránky jsou grafická reprezentace zvolené funkce. Tyto stránky vykreslují ovládací prvky, data získané periferiemi či varovné zprávy. Každá stránka je vykreslována skrze USART1 periferii skrze jednoduchou funkci (Úryvek kódu 2), která pošle string skrze periferii o dané velikosti. Na této funkci poté staví další funkce, které dokážou sestavovat větší celky. Funkce z úryvku kódu 3 nastavuje, dle pravidel zmíněných výše, kurzor na příslušné souřadnice. Ve funkci se také nachází kontrola, zda se kurzor nachází v rámci rozměrů stránky, které jsou fixně nastavené na 80×24. Tyto rozměry jsou v terminálové aplikaci ohraničeny ASCII symboly. Pro vykreslení textu je využita funkce z úryvku kódu 17, kde k danému textovému řetězci jsou před odesláním přidány ANSI sekvence pro podbarvení pozadí, textu a nebo ztučnění symbolů. Aplikace těchto všech funkcí lze vidět na

Každá stránka má tzv. statickou část, která se po celou dobu nemění. Statická část je vždy vykreslena na začátku vstupu stránky a poté je vždy vykreslena oblužní smyčkou v momentě, kdy příznak `need_frontend_update` je nastaven. V případě nastavení příznaku obslužní smyčka odešle ANSI sekvenci `\033[2J`, která smaže celou stránku a poté vykreslí stránku odpovídající aktualně zvolené funkci. Příznak lze taky manuálně nastavit odeslá-

```c
void ansi_send_string(const char* str) {
    HAL_UART_Transmit(USART1, (uint8_t*)str, strlen(str),
    HAL_MAX_DELAY);
}
```

Úryvek kódu 2: Způsob odeslání stringu UART periferií

```c
void ansi_set_cursor(const uint8_t row, const uint8_t col) {
    if (row > TERMINAL_HEIGHT || col > TERMINAL_WIDTH) {
        Error_Handler();
        return;
    }

    char result[CURSOR_BUFF_SIZE];
    size_t ret = snprintf(result, CURSOR_BUFF_SIZE, "\033[%u;%uH",
    row, col);

    if (ret >= sizeof(result)) {
        Error_Handler();
    }
    ansi_send_string(result);
}
```

Úryvek kódu 3: Nastavení pozice kurzoru pomocí ANSI escape sekvencí

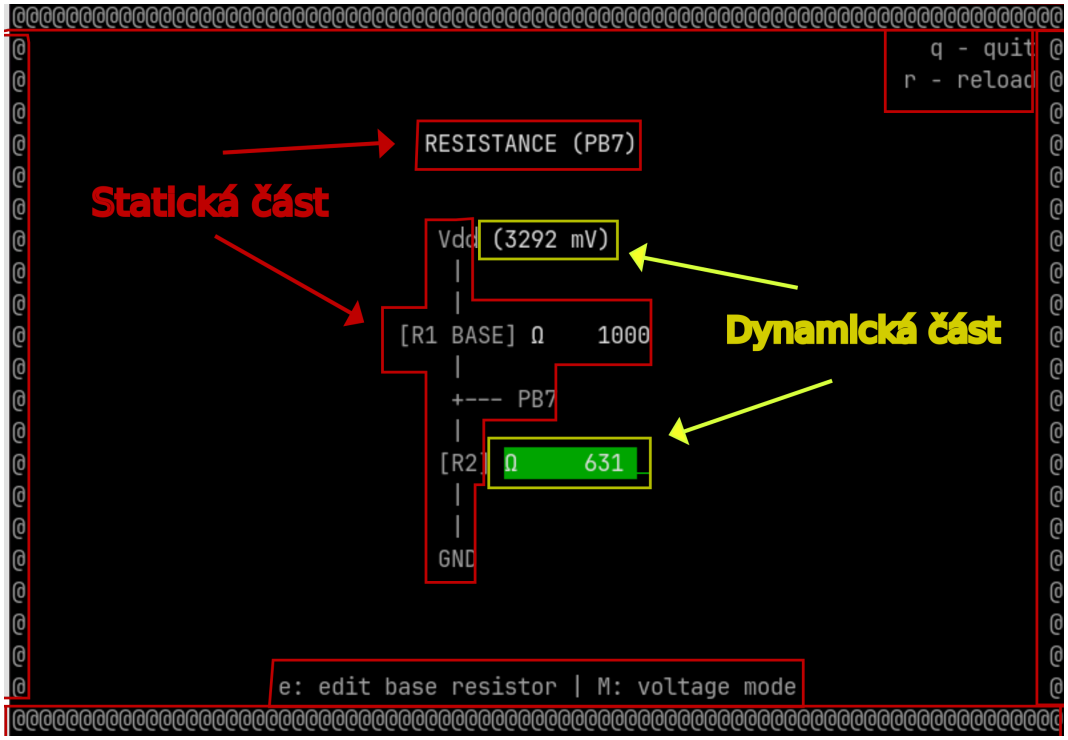
ním symbolu `R`, který je užitečný v případě, kdy se například vlivem špatného kontaktu vodiče mohou generovat náhodné symboly. Pokud by statická část vykreslovala periodicky, může nastat ke zbytečnému odesílání velkého množství dat skrze UART a to může spomalovat vykreslování. Také může dojít k rychlému blikání kurzoru v terminálové aplikaci, což je nežádoucí.

Tzv. dynamická část stránky se vykresluje každý cyklus oblužné smyčky. Do dynamické části spadá vykreslování varovných zpráv, naměřených hodnot a nebo výstupy z periferií. Hodnoty jsou vždy vykresleny s definovaným počtem číslic. Např. napětí ve voltech je vykresleno jako `printf("%4d")`, což vykreslí 4 číslice čísla a pokud má číslo méně než 4 číslice, je jsou pozice nahrazeny mezerou. Při generování upozornění, je řádek, na kterém se text vykresluje, smazán ANSI sekvencí `\033[2K` a v případě potřeby je vykreslen nový text. Na obrázku 25 je znázorněno na příkladu stránky pro měření odporu, že je ASCII ART zapojení, popisky a ohraničení vykresleno staticky a hodnoty, které jsou naměřeny jsou vykreslovány dynamicky.

4.2.4 Ovládání

Ovládat sondu lze pomocí symbolů, odesílané na rozhraní UART skrze terminálovou aplikaci. Na straně MCU jsou symboly přijímány na periferii UART, která při obdržení symbolu vyvolá přerušení. Pro implementaci zpracování symbolu je použit callback `HAL_UART_RxCpltCallback`, který je zavolán při vyvolání přerušení. Callback přečte symbol, který byl přijmut a zkontroluje, zda to není symbol, který je obecný pro všechny stránky[27]. V případě jiných symbolů je nahlédnuto do globální promněné `current_page` (viz. Úryvek kódu 16), která uchovává informaci, na které stránce se momentálně uživatel nachází a v závislosti na tom, je zvolena funkce pro provedení akce na základě přijatého symbolu. Po provedení příslušné akce je opět zapnuto přerušení pro přijetí znaku na UART periferii (viz. Úryvek kódu 4). Způsob přepínání ovládání v závislosti na stránce ukazuje

[27] Obecně je to symbol `R`, který slouží znovu vykreslení.



Obrázek 25: Ukázka vykreslování statické a dynamické části stránky

úryvek kódu 18. Způsob převedení symbolu na akci na dané stránce ukazuje příklad ovládání hlavního menu v úryvku 19. V tomto úryvku lze vidět, že pomocí přepínače je ovládání nezávislé na tom, zda uživatel posílá velká nebo malá písmena. Samotný callback nevykresluje stránku nicméně pouze nastavuje příznak `need_frontend_update` aby v dalším obslužním cyklu byla stránka vykreslena.

```c
void HAL_UART_RxCpltCallback(UART_HandleTypeDef* huart) {
    if (huart->Instance == UART_INST) {
        if (global_var.received_char == 'r' ||
            global_var.received_char == 'R') {  // reload
            ansi_clear_terminal();
            ansi_render_current_page();
            global_var.booted = true;
        } else {
            get_current_control();
        }

        HAL_UART_Receive_IT(&UART, &global_var.received_char, 1);
    }
}
```

Úryvek kódu 4: Způsob odeslání stringu UART periferií

4.2.5 Struktura TUI

Po připojení sondy je uživatel přivítán stránkou hlavního menu (obrázek 26). Tato stránka je hlavní rozbočka mezi funkcemi. Po stisknutí příslušné klávesy u funkce, je uživatel přesměrovaný na konkrétní stránku s funkcí. Hlavní menu je rozbočka pro tzv. základní módy, které jsou k dispozici, jak na SOP8 tak na TSSOP20. Při stisku písmene `A`, se uživatel v případě pouzdra TSSOP20 dostane do pokročilých funkcí, kde se nachází monitorování periferií. Univerzálně platí, že symbolem `Q` se uživatel dostane vždy to tohoto menu, ze kterého poté může zvolit jinou funkci.

Obrázek 26: TUI hlavní menu základních funkcí

Obrázek 27: TUI hlavní menu pokročilých funkcí

4.3 Princip nastavení periferií

Jelikož pouzdra SOP8 a TSSOP20 mají malý počet výstupů, není možné mít aktivované všechny periferie najednou. Další důvod je například využití časovače TIM2, který je jako jediný 32 bitový a je nutný k více funkcím. Jeden z příkladů je využití časovače TIM2 na `PA0`, ke čtení frekvence a zároveň využití TIM2 na generování pulzů. Další příklad je kolize periferie USART2 a kanálu `PB7`, kde je potřeba jiné nastavení pinů. Jeden z velkých problémů můžou činit DMA kanály, kterých je na STM32G030 pouze 5, což vyžaduje častou reinicializaci kvůli změně funkce [2].

Sonda obsahuje velké množství nastavení periferií a je velice snadné, ztratit přehled, která periferie je, a která není inicializovaná. Pro vyhnutí se tomuto problému během vývoje bylo zvoleno řešení, kdy při každém přepnutí funkce uživatelem jsou všechny periferie uvedeny do základního stavu a poté podle zvolené funkce jsou nastaveny pouze konkrétní periferie nutné pro danou funkci. Tento zpusob minimalizuje výskyt nedefinovaných chování, které během vývoje mohou nastat. Inicializace a deinicialiace periferií se řeší během obslužní smyčky, která při nastaveném příznaku `need_perif_update`, nastaví všechny periferie do původního stavu a následně dle zvolené funkce uživatelem je nastavena periferie. Nastavení příznaku je provedeno v případě, že sonda potřebuje po uběhnutém čase změnit nastavení, nebo pokud uživatel pomocí vstupu z UARTu vyvolá žádost o přepnutí funkce.

4.4 Implementace měření s ADC

4.4.1 Měření napětí a logických úrovní

Napětí je měřeno pomocí AD převodníku na dvou kanálech v případě SOP8 a na třech v případě TSSOP20 (+ kanál s referenčním napětím). Uživatel skrze TUI může vypnout či zapnout měření na určitém kanále. Jak bylo zmíněno v kapitole 2.4, ADC průběžně měří 32 vzorků za 250 ms (128 Hz). Každé měření kanálu je nastaveno na 160 cyklů, což je maximální přesnost měření.

K časování měření je využito časovače TIM3, který po uplynutí času vyvolá přerušení a je naměřena hodnota ADC. Časovač má nastavenou předdělička na $64000 - 1$, což nastaví frekvenci časovače z 64 MHz na 1 KHz (neboli časovač inkrementuje hodnotu každou 1 ms). Jelikož ADC běží na frekvenci 32 MHz a změření jednoho kanálu trvá 160 cyklů, změření jednoho kanálu trvá $\sim 5\ \mu$s. Protože frekvence měření je 128 Hz, můžeme tuto hodnotu zanedbat a nastavit periodu časovače na $7 - 1$.

Při přetečení časovače je vyvoláno přerušení, které zavolá callback z ukázky kódu 5. Funkce zastaví časovač, a sekvenčně začne měřit poměrnou hodnotu mezi napětím na kanálu a V_{dd}. Po dokončení konverze je tato hodnota uložena do dynamicky alokovaného pole `voltage_measures` o velikosti $64 \times$ počet aktivních kanálů[28]. Toto pole se chová cyklicky, tzn. při překročení počtu prvků se začne plnit od začátku. Po změření všech kanálů je resetován a nastartován časovač.

```c
void adc_measure_callback(adc_vars_t* adc_perif) {
    HAL_TIM_Base_Stop_IT(adc_perif->timer);

    HAL_ADC_Start(adc_perif->hadc);
    for (uint8_t i = 0; i < adc_perif->n_active_channels; ++i) {
        HAL_ADC_PollForConversion(adc_perif->hadc, ADC_DELAY);
        adc_perif->voltage_measures[adc_perif->measures_index++] =
        HAL_ADC_GetValue(adc_perif->hadc);
        if (adc_perif->measures_index >=
            adc_perif->n_active_channels * CHANNEL_NUM_SAMPLES) {
            adc_perif->measures_index = 0;
        }
    }

    HAL_ADC_Stop(adc_perif->hadc);
    __HAL_TIM_SET_COUNTER(adc_perif->timer, 0);
    HAL_TIM_Base_Start_IT(adc_perif->timer);
}
```

Úryvek kódu 5: Naměření vzorku z ADC (ukázka bez ošetření)

[28] ADC při nastavení více kanálu sekvenčně prochází všechny kanály dokola.

Obrázek 28: TUI měření napětí

Každých 250 ms obslužní smyčka vezme naměřené vzorky z `voltage_measures` a udělá aritmetický průměr každého kanálu. Po provedení průměru je z poměrné hodnoty vypočítáno referenční napětí a napětí každého kanálu. K výpočtům napětí je, na základě vztahů z kapitoly 2.3.1.1, využito makro z HAL knihovny (Úryvek kódu 21). Toto napětí je poté zobrazeno dynamicky na stránce. U každého kanálu je také vyhodnocené zda naměřená hodnota napětí odpovídá log. „1“, log, „0“ nebo se napětí nachází v nedefinované oblasti[29].

4.4.2 Měření odporu

Měření odporu vychází z principů měření z kapitoly 4.4.1. Pro změření odporu daných rezistorů, je změřeno napětí stejným způsobem jako v předchozí kapitole, ale pouze na prvním kanále obou pouzder. Z naměřených vzorků frekvencí 128 Hz z kanálu 1 a referenčního napětí je spočítán průměr a poté je poměrová hodnota převedena na napětí. Z hodnoty napětí je poté, na základě normálového rezistoru, vypočítán odpor měřeného rezistoru (Úryvek kódu 22).

4.5 Implementace měření frekvence a odchytávání pulzů

Stránka měření frekvence (Obrázek 30) zobrazuje `Frequency`, což je měření metodou hradlování, `Reciprocial frequency`, což je frekvence naměřená skrze šířky pulzu, `High pulse width`/`Low pulse width`, což je spočítaná šířka nízkého a vysokého pulzu a `Duty` neboli střída v případě, že by to byl signál PWM. Pro výpočet těchto hodnot, se periodicky střídá měření hradlováním a reciproční měření. Hradlovací čas je možné nastavovat z předvybraných časů.

[29] Podrobnosti je možné najít v manuálu použití.

Obrázek 29: TUI měření odporu

4.5.1 Měření frekvence hradlováním

Při měření frekvence metodou hradlováním, jsou využity dva čítače. První časovač, TIM3, je zvolen, aby byl změřen čas hradlování. Tento čítač je nastavený předděličkou na 1 KHz (incrementace každou milisekundu) a poté podle nastavené periody bude nastaven hradlovací čas. Za hradlovací čas lze volit hodnoty 50, 100, 200, 500 a 1000 milisekund. Jelikož TIM3 má promněnlivou periodu, tak je nastaven příznak `AutoReloadPreload`, který při změně periody za běhu časovače, je perioda aplikována až po přetečení časovače. Pokud by časovač měl hodnotu např. 500, a perioda by byla nastavena na 200, časovač by

Obrázek 30: TUI měření frekvence

nemohl přetéct na stal by problém. TIM3 je také nastaven jako One Pulse, což znamená, že při přetečení bude vypnut. Nastavení periferie je ukázáno v úryvku kódu 20.

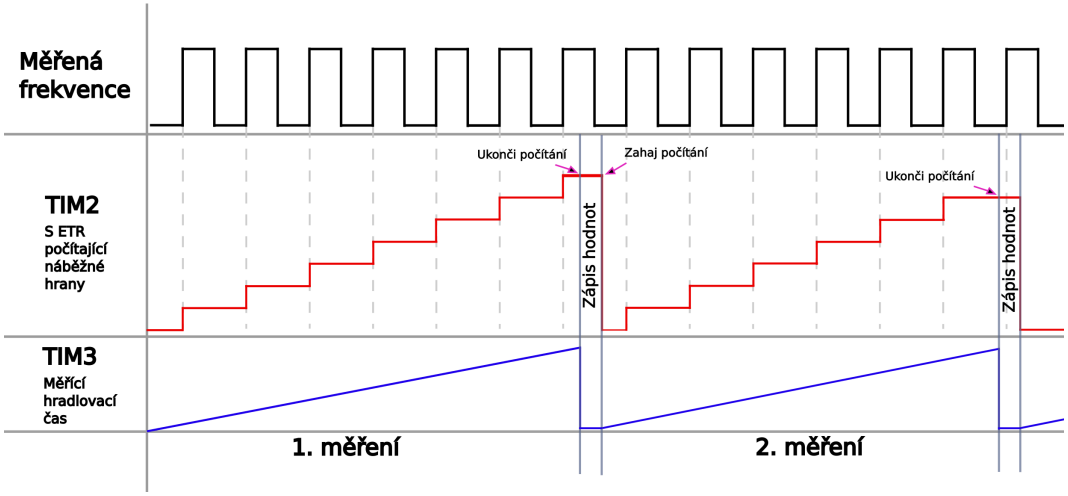
Na počítání hran je využit 32 bitový TIM2, který z důvodu své velikosti může napočítat velké množství pulsů i s poměrně dlouhým hradlovacím časem. Aby TIM2 počítal hrany, lze nastavit pro časovač tzv. ETR, neboli externí trigger. Tento trigger inkrementuje časovač pokaždé, když na ETR pinu (`PA0`) bude hrana (v tomto nastavení náběžná). Časovač v podstatě ignoruje interní hodiny, které inkrementují každý cyklus časovač, ale řídí se podle externích hodin na pinu `PA0`, které jsou v případě sondy měřená frekvence [2]. Obrázek 31 ukazuje způsob spolupráce TIM2 a TIM3.

Při vysokých frekvencích může nastat problém, kdy bude záležet i na rozdílu mezi zapnutím TIM2 a TIM3. Tyto dva časovače za normálních okolností musí být zahájeny sekvenčně a to vede k tomu, že buď bude nepatrně delší hradlovací čas a bude napočítáno více pulsů než by mělo. Design časovačů nicméně poskytuje řešení, kdy časovač může vyvolávat triggery a jiný časovač může na tyto triggery reagovat. V případě sondy byl TIM3 nastaven jako master timer, který při spuštění vyvolá trigger. TIM3 byl nastaven jako slave timer. Slave timer čeká, až dostane trigger a v momentě, kdy trigger dostane, začne počítat. Toto nastavení je ukázáno v úryvku kódu 20 a 23.

Po změření pulsů se provede výpočet frekvence. Zde nastává problém, pro získání frekvence je nutné dělit počet pulzů, hradlovým časem. Nicméně pro MCU je operace dělení poměrně drahá. Jelikož všechny hradlovací časy (kromě 1000 ms), jsou pod 1 sekundu, dojde k dělení desetiným číslem. Toto desetiné číslo je ale možné převést na zlomek a po úpravě vznikne vztah, ve kterém eliminujeme dělení. Rovnice 15 demostruje příklad na 500 ms.

$$f_{\text{gate}} = \frac{N}{T_{\text{gate}}} = \frac{N}{0.5} = \frac{N}{\frac{1}{2}} = \frac{N}{1} \times \frac{2}{1} = 2N \tag{15}$$

Měření touto metodou bylo otestováno měření nižších desítek MHz a naměřená odchylka od původní hodnoty byla ~ 0.16 %. Což pravděpodobně bude způsobeno tím, že sonda nevyužívá externího krystalu ale interního oscilačního obvodu [2], tak může



Obrázek 31: Signály při měření frekvence hradlováním

být lehká odchylka od reference. Při vyšších frekvencích zde můžou hrát roli i režie implementovaná pro časovače.

4.5.2 Měření reciproční frekvence

Při měření pomocí reciproční frekvence je využito časovače TIM2. Časovač má nastaveny celkově 2 kanály do módu input capture. Input capture kanálu, který při hraně na vstupu uloží aktuální hodnotu časovače do registru a následně metodou DMA do paměti. Důvod inicializace dvou kanálů místo jednoho je ten, že každý kanál sice monitoruje stejný pin, ale jeden reaguje na náběžnou a druhá na sestupnou. Kanál sice umí detekovat obě najednou, nicméně pro funkci měření je nutné rozpoznat, která hrana je náběžná a která je sestupná. U frekvence je vždy změřena náběžná hrana, poté sestupná hrana a poté opět náběžná. Důvod proč jsou měřena i sestupná hrana je určení střídy v případě PWM signálu. Pokud je naměřena tato posloupnost, je možné vypočítat recipročí frekvenci, střídu a šířku pulzů.

```c
void detector_compute_freq_measures(sig_detector_t* detector) {
    uint32_t* edge_times = detector->edge_times;
    // Získání rozdílu mezi hranami
    uint32_t high_delta =
        edge_times[DET_EDGE2_FALL] - edge_times[DET_EDGE1_RISE];

    uint32_t low_delta =
        edge_times[DET_EDGE3_RISE] - edge_times[DET_EDGE2_FALL];
    // Výpočet času pulzu
    detector->widths[DET_LOW_WIDTH] = (low_delta) /
PROCESSOR_FREQ_IN_MHZ;
    detector->widths[DET_HIGH_WIDTH] = (high_delta) /
PROCESSOR_FREQ_IN_MHZ;
    uint64_t period =
        (detector->widths[DET_LOW_WIDTH] + detector-
        >widths[DET_HIGH_WIDTH]);

    // ochrana před dělením nulou
    if (period > 0) {
        detector->rec_frequency = 1000000 / period; // v ms
        detector->pwm_duty =
            ((uint64_t)detector->widths[DET_HIGH_WIDTH] * 100) /
            period;
    } else {
        detector->rec_frequency = 0;
        detector->pwm_duty = 0;
    }
}
```

Úryvek kódu 6: Funkce pro výpočet veličin na základě recipročního měření

4.5.3 Odchytávání pulzů

Obrázek 32: TUI odchytávání pulzů

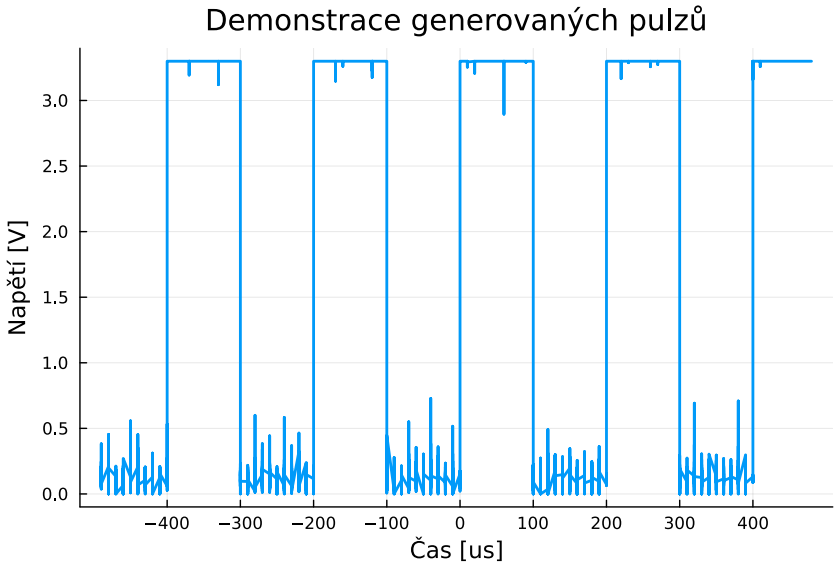
Odchytávání pulzů je podfunkce na stránce měření frekvence. Při detekci náběžné nebo sestupné hrany (dle nastavení uživatele) je nastaven příznak, který je následně vykreslen na terminál. Uživatel tento příznak poté může smazat. K detekci pulzů je využito nastavení periferií jako při recipročním měření frekvence. Časovač, který je na vstupu nastaven jako input capture ukládá do registru hodnotu a je vyvoláno přerušení. Během přerušení je zavolána funkce, která zkontroluje, zda je to odpovídající hrana a pokud ano, je nastaven příznak. K odchytávání pulzů by mohlo být využito EXTI callbacku, nicméně to by znamenalo jiné nastavení periferií a zkomplikování sondy z hlediska vývoje.

4.6 Implementace generování pulzů

Generování pulzů je realizováno za pomocí časovače TIM2, který má kanál 1 nastaven na PWM o střídě 50 %. Předdělička časovače je nastavena na $64 - 1$, což znamená, že časovač běží na frekvenci 1 MHz (inkrementace každou 1 μs). Poté pomocí periody čítače na uživatelem nastavována šířka pulzu, kdy zvolená šířka je vynásobena dvěma, protože při nastavení tomto nastavení dojde k přerušení pulzu v polovině periody. Aby časovač vyslal pouze jeden pulz, tak je inicializován tzv. One pulse, což způsobí zastavení časovače po jednom přetečení. Využití One pulse a časovače pro generaci je zajištěna velice přesná šířka jednoho pulzu.

$$f_{\text{send}} = \frac{1}{2 \times \text{perioda}} \tag{16}$$

Funkce umí generovat jeden pulz, více pulzů a generovat neustále. Generace `X` pulzů funguje na principu One Pulse, kdy po dokončení jednoho pulzu je ihned zaslán další. Takto je možné poslat přesný počet pulzů za pomocí PWM. V případě neustále generace, je One Pulse deinicializován, a je vysílána frekvence odpovídající vztahu 16. Pulz může být poslán jako pulz úrovně 3.3 V a nebo jako pulz 0 V. Nastavení časovače umožňuje nastavit pulz o šířky o řádech až mikrosekund. Obrázek 33 demonstruje přesnost, která



Obrázek 33: Demonstrace pulzů

byla naměřena pomocí osciloskopu. Generování pulzů může být vhodné pro testování čítačů, posuvných registrů a obvodů, kde je potřeba generovat hodinový signál.

4.7 Implementace nastavování úrovní

Obrázek 34: TUI nastavování úrovní

Nastavování logických úrovní funguje celkem na 4 kanálech, které je možné skrze uživatelské rozhraní vypnou nebo zapnout dle potřeby. Každý pin je inicializován skrze STM32 HAL strukturu `GPIO_InitTypeDef`, kde je nastaven `GPIO_MODE_OUTPUT_PP`. Tento mód nastaví push pull na pin a lze ho tak ovládat skrze stisk tlačítka. Obrázek 34 ukazuje vizualizaci stránky pro funkci nastavování úrovní.

4.8 Implementace diagnostiky posuvného registru

Obrázek 35: Signály při měření frekvence hradlováním

Funkce diagnostiky posuvného registru nabízí možnost nastavení jednotlivých bitů a následné odesílání dat do registru. Pro naplnění dat do posuvného registru je `PA7` připojen na jako hodinový signál registru a `PA0` jako datový pin do posuvného registru. Sonda umí posílat všech 8 bitů najednou a nebo je možné posílat bity postupně manuálně. Doba odesílání jednoho bitu je 200 ms. Tento čas nabízí možnost vizuální kontroly diagnostikovaného obvodu během odesílání. Úryvek kódu 24 ukazuje způsob odeslání jednoho bitu do posuvného registru. Funkce nabízí i kompatibilitu s posuvnými registry `SNx4HC595`, která na základě signálu `RCLK` převede data z registru na paralelní výstupy [29].

4.9 Implementace diagnostiky Neopixel

Neopixel je speciální způsob komunikace, které využívají RGB LED. Tato komunikace má svá specifika, rozebrána v kapitole 2.5.4, která jsou nutná implementovat do sondy. Uživatel díky této funkci může odesílat barvy, která otestují funkčnost RGB LED, a pasivně monitorovat barvu, která se odesílá do RGB LED. Možností je také testování LED v serii, kde uživatel může odeslat barvu, dle pravidel komunikace, na všechny LED najednou. To je užitečné v případě, kdy již jsou LED zapojeny v serii a je nutné otestovat, zda není chyba u nějaké konkrétní LED.

4.9.1 Monitorování

Pro monitorování barvy odeslané přes seriovou komunikaci je využit časovač TIM2. Časovač má nastaveny 2 kanály v nastavení input capture, kde jeden reaguje na náběžnou hranu a druhý reaguje na sestupnou hranu. Tyto hrany jsou zaznamenávány DMA metodou do paměti, kde sonda provede analýzu dat. Jelikož DMA je nastavené jako cirkulární buffer, začátek dat nutně nemusí být jako první prvek v poli. Pokud by například bylo odesláno méně bitů nebo by se na vodiči objevila parazitní hrana, začátek se posune. Proto

algoritmus začne hledat začátek tak, že prochází pole a hledá, mezi jakou sestupnou a vzestupnou hranou je větší prodleva než odpovídá `Reset` času (viz. Úryvek kódu 7).

```c
int8_t neopixel_find_start(neopixel_measure_t* data) {
    int8_t start_index = -1;
    uint8_t prev_measure = NEOPIXEL_DATA_LEN - 1;
    for (uint8_t i = 0; i < NEOPIXEL_DATA_LEN; ++i) {
        uint32_t curr_rise_edge = data->rise_edge[i];
        uint32_t prev_fall_edge = data->fall_edge[prev_measure];
        uint64_t width;
        // v případě, že časovač přetekl, přičti velikost časovače
        if (curr_rise_edge < prev_fall_edge) {
            width = (curr_rise_edge + 0xFFFFFFFF) - prev_fall_edge;
        } else {
            width = curr_rise_edge - prev_fall_edge;
        }
        if (width > NEOPIXEL_RESET) {
            // nalezen start
            start_index = i;
            break;
        }
    }
    return start_index;
}
```

Úryvek kódu 7: Funkce pro nalezení začátku komunikace neopixel

Po nalezení začátku, je zkontrolováno následujících 24 pulzů zda jejich šířka odpovídá velikosti bitové jedničky nebo nuly. Pokud je nalezen pulz, který neodpovídá velikosti, jsou data zahozeny a dále se nepokračuje. Nakonec jsou jednotlivé bity posunuty tak, aby odpovídali 8 bitovému číslu pro každou složku.

```c
pulse_width = data->fall_edge[i] - data->rise_edge[i];
if (pulse_width >= NEOPIXEL_MIN_LOW &&
    pulse_width <= NEOPIXEL_MAX_LOW) {
    bits_detected[i] = 0;
} else if (pulse_width >= NEOPIXEL_MIN_HIGH &&
    pulse_width <= NEOPIXEL_MAX_HIGH) {
    bits_detected[i] = 1;
} else {
    return;
}
```

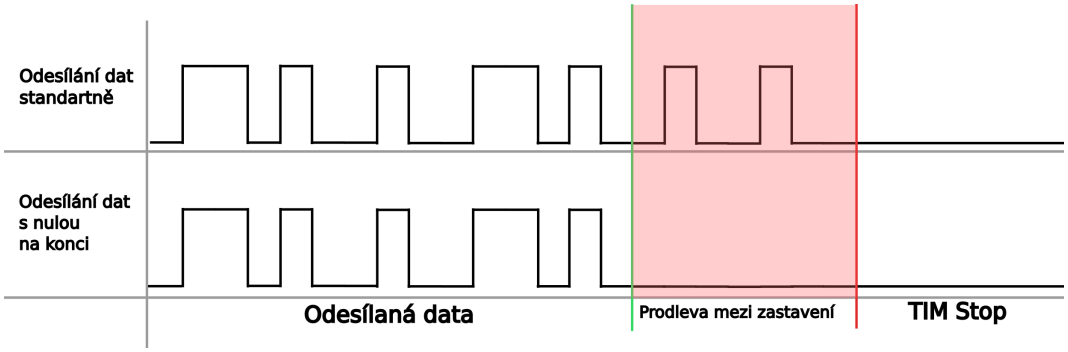
Úryvek kódu 8: Způsob definování hodnoty bitu z hran neopixel

4.9.2 Testovací signály

Pro přenos testovacích dat do Neopixelů je využit časovač TIM1, konfigurovaný v režimu PWM. Časovač pracuje s předem připraveným polem hodnot (šířek pulzů), které jsou přes DMA načítány do časovače a převáděny na PWM signál s odpovídajícími střídami. Po zadání číselných hodnot pro červenou, zelenou a modrou, jako 8 bitová čísla (0-255), jsou jednotlivá čísla složek rozděleno na bity (celkem 24 bitů). Každý bit je poté převeden na střídu PWM signálu.

Pro správně široký pulz je potřeba nastavit periodu časovače a poté určit, jaká střída bude reprezentovat log. 1 a který log. 0. Po analýze bylo zvoleno periody $80 - 1$ což při frekvenci časovače 64 MHz, odpovídá přetečení každých 1.25 μs. To odpovídá délce doby odeslání jednoho bitu [6]. Při testování RGB LED odesílání bylo zjištěno, že odeslání log. 1 odpovídá $\frac{2}{3}$ periody a log. 0 odpovídá $\frac{1}{3}$ periody.

Po převedení čísla složky odpovídající střídy, jsou přesunuty do pole aby bylo možné zahájit činnost časovače. Během testování této metody docházelo k nepredikovatelným chování, kdy při specifických barvách, neodpovídala odeslaná data a barva LED. Příčinou byla prodleva mezi vyprázdnění dat z bufferu DMA a zastavení časovače. Časovač má v CCR registru uloženo při jaké hodnotě časovače má přepnout výstup. Po dosažení hodnoty je aktivován trigger pro DMA a je načtena nová hodnota z bufferu do CCR registru [2]. Po vyprázdnění bufferu již nedojde k aktualizaci CCR registru, ale časovač neustále běží. Z tohoto důvodu jsou na začátek a na konec pole přidány nuly, který po odeslání dat nastaví do CCR registru nulu než dojde k zastavení časovače.



Obrázek 36: Neopixel signál bez ukončení a s ukončení nulou

4.10 Implementace diagnostiky UART

Diagnostika UART je implementováno za pomocí vestavěné periferie USART2 a je inicializována podobným způsobem jako USART1 pro komunikaci s PC. Rozdíl je ten, že pro tuto diagnostický nástroj je nutné nastavovat baudrate, délku slova, paritu a počet stop bitů flexibilně aby uživatel mohl přizpůsobit nastavení svým potřebám. Pro nastavování uživatel využívá klávesy a po každém nastavení je periferie znovu inicializována s novým nastavením. Periferie využívá piny `PA2` a `PA3`.

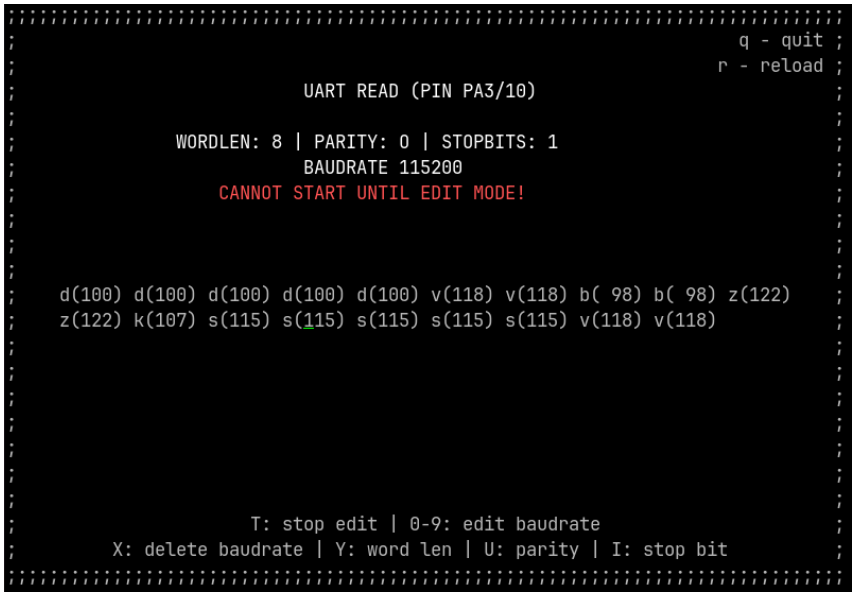
Pro nastavení periferie je nutné vstoupit do editačního módu, kdy je periferie zastavena. Uživatelovi vyskočí hláška „Cannot start until edit mode“ (viz. Obrázek 37) aby věděl, že se v tomto módu nachází. Na základě toho se změní i dolní lišta, na které se nachází nápověda ovládání. V tomto módu uživatel nastaví nastaví klávesama např. baudrate, kde

stisknutím čísla je číslo přidáno na poslední pozici a stiskem `X` je poslední pozice smazána. Po opuštění tohoto módu je periferie nastavena a může být využita.

4.10.1 Monitoring

Monitorování UART je realizováno za pomocí DMA, kdy po obdržení slova (7, 8 nebo 9 bitového) toto slovo uloženo do pole. DMA je v tomto případě nastaveno jako cirkulární. Toto pole je poté periodicky zobrazováno na terminál. Na terminál je zobrazen jak symbol, tak jeho číselná reprezentace aby uživatel, i při symbolu jako mezera, mohl vidět zda byl daný symbol zachycen. Obrázek 37 ukazuje způsob vypisování symbolů. Uživatel může symboly vyčistit klávesou `G`.

Výpis funguje způsobem scrollování, tzn. pokud se zaplní stránka, tak nejstarší symbol je smazán, jsou na stránce posunuty a nový se vykreslí jako nejnovější. Tohoto bylo dosaženo tak, že při vykreslování se zjistí DMA counter, který určuje na jakou poslední pozici periferie zapsala data. Jelikož je pole cirkulární, tak je poté iterované celé pole do doby, než se vrátí pointer na začátek. Úryvek kódu 25 ukazuje implementaci této vlastnosti.



Obrázek 37: TUI UART monitorování

4.10.2 Posílání testovacích symbolů

Diagnostika UARTu také odesílá testovací symboly, které si uživatel navolí. Celkově lze odeslat až 10 symbolů najednou. Tyto symboly jsou poté odeslány pomocí tlačítka `S` skze blokovací funkci `HAL_UART_Transmit`. Odesílaná data lze modifikovat v tzv. „Data edit mode“, který dává možnost zapisovat konkrétní symboly klávesnicí jako data a poté přepnout kurzor na další. Takto je možné upravit všechny symboly sekvenčně. Kurzor je reprezentován jako text, který je podbarven zeleně. Uživatel si může zvolit, jaké množství symbolů chce celkově odeslat.

Obrázek 38: TUI UART odesílání testovacích symbolů

4.11 Implementace diagnostiky I2C

Diagnostika I2C je funkce, která pomáhá k analýze komunikace mezi MCU a senzory. Funkce umí, detekovat slave zařízení na sběrnici, chovat se jako slave, chovat se jako master, umí bez zásahu monitorovat komunikaci mezi masterem a slavem a umí testování OLED displeje na bázi SSD1306. Tyto funkce ulehčují diagnostiku při práci se senzory a zkracují čas hledání problému, pokud například senzor posílá nekorektní data apod. Pro implementaci bylo využito zabudované I2C periferie, která je runtime přenastavována podle potřeby.

```c
void i2c_init_perif(i2c_perif_t* i2c_perif) {
    i2c_perif->hi2c->Instance = I2C1;
    i2c_perif->hi2c->Init.Timing = 0x10B17DB5; // normal mode
    i2c_perif->hi2c->Init.OwnAddress1 =
        (global_var.device_state == DEV_STATE_ADV_I2C_SLAVE)
            ? i2c_perif->slave_address << 1
            : i2c_perif->slave_address;
    i2c_perif->hi2c->Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
    i2c_perif->hi2c->Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
    i2c_perif->hi2c->Init.OwnAddress2Masks = I2C_OA2_NOMASK;
    if (HAL_I2C_Init(i2c_perif->hi2c) != HAL_OK) {
        Error_Handler();
    }
}
```

Úryvek kódu 9: I2C inicializace periferie

4.11.1 Skener adres

Obrázek 39: TUI I2C skenování adres

Skenování adres zobrazuje všechny adresy zařízeních, které se na sběrnici nachází. Skenování probíhá tak, že je funkcí `HAL_I2C_IsDeviceReady` zkontrolováno sekvenčně všech 127 adres a adresy, ze kterých je odpověď `ACK` jsou poté vykresleny do terminálu.

4.11.2 Master mód

Obrázek 40: TUI UART odesílání testovacích symbolů

Master mód je vhodný pro odesílání testovacích sekvencí senzoru. Uživatel může nastavit adresu slave zařízení na které chce odeslat data a také může nastavit read/write bit. V případě write uživatel může poslat až 10 bajtů dat najednou. Periferie nejprve odešle

první rámec, který obsahuje adresu a read/write bit na který zařízení odpoví `ACK`. Dále sonda začne odesílat data dokud všechny data úspěšně neodešle, nebo do momentu, kdy nastane error. Tento error může být například z důvodu, že zařízení neodpoví (`NACK`) nebo z jiných důvodů.

Pokud uživatel nastaví read, může odeslat na slave zařízení 1 bajt (většinou adresu paměti, ze které chce master číst) a přijme data, která mu slave pošle. Tyto data jsou reprezentovány na displeji terminálu jako hexadecimalní hodnoty. Obrázek 40 ukazuje stránku I2C diagnostiky s read master módem. Červený čtvereček ohraničuje bajt, který je poslán senzoru a zelený rámeček ohraničuje bajt, který byl přijat od slave zařízení. Pod hodnotami je zobrazené `I2C OK`, což značí, že komunikace proběhla v pořádku. Počet bajtů, které budou přečteny lze uživatelsky volit.

Úryvek kódu 10 reprezentuje funkci, která se stará I2C komunikaci, při čtení. Nejprve je zavolána funkce `HAL_I2C_Master_Transmit`, která odešle data slave zařízení ve write módu. Zde je možné si povšimnout, že adresa musí být bitově posunuta. Důvod posunutí je přítomnost read/write bitu, které je na posledním místě bajtu. Pokud pokud dojde k problému během odesílání, odesílání ukončeno a je vypsána chyba na terminál.

```c
void i2c_read_data_master(i2c_perif_t* perif) {
    if (perif->send_data) {
        perif->send_data = 0;
        HAL_StatusTypeDef ret = HAL_I2C_Master_Transmit(
            perif->hi2c, perif->slave_address << 1,
            perif->master_read_send_data, 1, PERIF_DELAY);

        if (ret == HAL_OK) {
            ret = HAL_I2C_Master_Receive(perif->hi2c,
                                         perif->slave_address << 1,
                                         perif->slave_received_data,
                                         perif->bytes_to_catch,
                                         PERIF_DELAY);
            if (ret == HAL_OK) {
                perif->send_status = I2C_SEND_SUCCESS;
            } else {
                perif->send_status = I2C_ERROR_RECIEVE;
            }
        } else {
            perif->send_status = I2C_ERROR;
        }
        ansi_print_i2c_error(ret, perif->hi2c);
    }
}
```

Úryvek kódu 10: I2C master read funkce

Po úspěšném odeslání je opět odeslán rámec s adresou, ale tentokrát je nastaven bit jako read. Po odeslání rámce sonda čeká na odpověď o dané velikost bajtů, definovanou dobu. Tyto data jsou poté zobrazena na terminálu. V případě chyby je chyba vyparsována a na terminál je vypsána pravděpodobná příčina. Stejný princip odesílání dat je i v případě zapisování dat, ale pouze nedochází z vyžádání čtení.

4.11.3 Testování SSD1306

Obrázek 41: TUI I2C testování displeje SSD1306

Testování OLED displeje SSD1306 je funkce která, na nastavenou adresu uživatelem[30], odešle sekvenci, která inicializuje potřebné parametry displeje a rozsvítí všechny pixely na displeji, aby uživatel mohl ověři, zda displej není poškozený.

4.11.4 Monitoring

Monitoring I2C je podstatná část, která umí číst jednoduché komunikace mezi master a slave zařízením. Monitoring I2C komunikace není jednoduše realizovatelná na I2C periferii. Pokud periferie je nastavena jako slave zařízení, dojde sice ke čtení komunikace, ale také dojde k odpovídání, což je nežádaný efekt. I2C periferie STM32 nenabízí možnost monitorovat komunikaci. Aby byla komunikace I2C mohla být monitorována, bylo k tomu využito SPI rozhraní.

Propojení I2C masteru s SPI slave zařízením prostřednictvím hodinového signálu vyžaduje specifické nastavení parametrů SPI, aby bylo možné sladit odlišné vlastnosti obou rozhraní. V I2C zůstává hodinový signál (SCL) v klidovém stavu na vysoké logické úrovni (HIGH), což vyžaduje nastavení polarity SPI hodin (CPOL) na hodnotu 1. Tím se zajistí, že SPI slave zařízení bude v nečinnosti očekávat vysokou úroveň hodinového signálu, což odpovídá výchozímu stavu I2C. Dále je nutné řešit počáteční fázi komunikace: I2C zahajuje přenos tzv. start podmínkou, kdy datová linka (SDA) poklesne před poklesem SCL. Tato první hrana SCL nesmí být SPI slave interpretována jako platný hodinový takt. K jejímu ignorování slouží nastavení fáze SPI hodin (CPHA) na hodnotu 1, která způsobí,

[30]Tato adresa je nastavena stejným způsobem, jako v Master a Slave módu

Obrázek 42: TUI I2C monitoring

že data jsou čtena až na druhé hraně hodinového signálu. Tato kombinace (CPOL=1 a CPHA=1) odpovídá SPI módu 3 a umožňuje SPI zařízení správně reagovat na hodiny generované I2C masterem. V praxi se tak SPI slave synchronizuje s I2C hodinami až po start podmínce, kdy začne zpracovávat data z následujících taktů, například při zápisu I2C adresy nebo přenosu dat.

Po propojení hodinových signálů je nastaveno SPI rozhraní na 9 bitovová slova. Je to z důvodu, že I2C má slova 8 bitová, ale poté je k tomu přidán `ACK/NACK` bit, což je devátý. Na základě této metody je možné monitorovat I2C sběrnici za pomocí SPI rozhraní. Data jsou ukládána do pole za pomocí metody DMA. Po získání dat z rozhraní jsou data parsována a zobrazována na terminál podle kontextu dat. Monitor rozpozná, jestli se jedná o zápis nebo nebo čtení dat a podle toho přizpůsobí výpis. Obrázek 42 tuto skutečnost ukazuje na výpisu z terminálové aplikace. Úryvek kódu 27 ukazuje způsob inicializace SPI rozhraní.

4.12 Implementace diagnostiky SPI

Diagnostika SPI je, podobně jako u I2C, velice podstatný nástroj pro debugování komunikace mezi komponenty a MCU jako například paměti, senzory atd. STM32 má integrovanou periferii pro komunikaci SPI. SPI diagnostika obsahuje funkce jako čtení komunikace, aktivní odesílání testovacích dat a testování SSD1306 displeje. Uživatelské ovládání je co nejvíce přiblíženo k ovládání I2C diagnostiky aby uživatel se jednodušše naučil používat sondu. Pro SPI je možné nastavit jakou fázi nebo jakou polaritu má `SCK` využívat při zpracování dat.

4.12.1 Master mód

SPI periferie je konfigurována jako master, což umožňuje uživateli plně řídit komunikaci se slave zařízením. V tomto režimu může uživatel jak zapisovat data do slave zařízení, tak z něj číst odpovědi. Při zápisu sonda podporuje odeslání až 10 bajtů dat zadaných uživatelem v hexadecimalním formátu. Data jsou poté odeslány pomocí blokovací funkce `HAL_SPI_Transmit`. Ta zajistí, že mikrokontrolér čeká na dokončení celého přenosu před

dalším krokem. Pro čtení dat sonda nejprve odešle jeden bajt dat a následně pomocí blokovací funkce `HAL_SPI_Receive` zachytí odpověď slave zařízení. Úryvek kódu 11 ilustruje klíčové části této logiky a také způsob detekování chyb v průběhu odesílání. Blokovací funkce zjednodušují implementaci, ale vyžadují pečlivé časování, aby nedošlo k zablokování systému při delších operacích.

```c
void spi_transmit(spi_perif_t* perif) {
    if (!perif->send_data) {
        return;
    }

    perif->error = SPI_ERROR_SUCCESS;
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET); // NSS
    if (!perif->read_bit) {
        // only write
        if (HAL_SPI_Transmit(perif->hspi,
                             (uint8_t*)perif->data,
                             perif->bytes_count,
                             PERIF_DELAY) != HAL_OK) {
            perif->error = SPI_ERROR_SEND;
        }

    } else {
        // write
        if (HAL_SPI_Transmit(perif->hspi,
                             perif->master_send_data,
                             1,
                             PERIF_DELAY) != HAL_OK) {
            perif->error = SPI_ERROR_SEND;
        }
        //read
        if (HAL_SPI_Receive(perif->hspi,
                            perif->data,
                            perif->bytes_count,
                            100) != HAL_OK) {
            perif->error = SPI_ERROR_RECEIVE;
        }
    }

    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_SET); // NSS
}
```

Úryvek kódu 11: SPI transmit funkce

Obrázek 43: TUI SPI master mód

4.12.2 Monitoring

Sonda využívá SPI periferii konfigurovanou v slave režimu pro pasivní zachytávání dat z externího master zařízení. Uživatel může (podobně jako v master módu) nastavovat parametry hodinového signálu SCK (polaritu a fázi), aby sladil časování s požadavky systému. Pro zachycení dat, i při vysokých hodinových frekvencích, je implementováno DMA, které průběžně ukládá přijaté bajty přímo do vyhrazeného bufferu v paměti, aniž by zatěžoval jádro procesoru. Obslužná smyčka následně v pravidelných intervalech načte tato data z bufferu, převede je do hexadecimálního formátu a zobrazí na terminálovou aplikaci. Tento přístup minimalizuje riziko ztráty bajtů i při maximální rychlosti SPI komunikace a zároveň udržuje responzivitu uživatelského rozhraní.

Obrázek 44: TUI SPI monitor mód

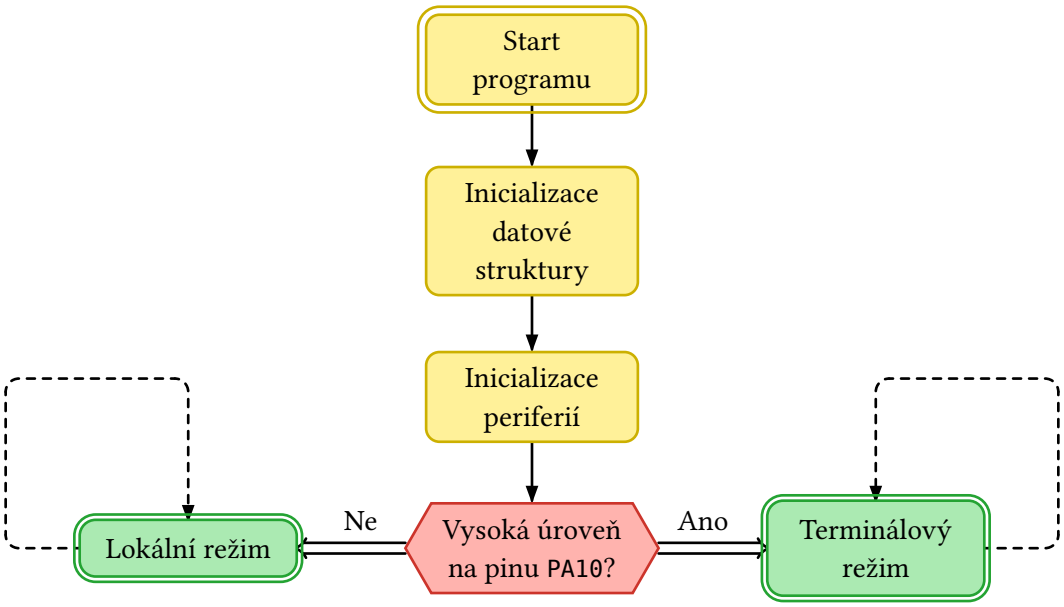
Kapitola 5

Návrh lokálního režimu STM32

5.1 Logika nastavení režimů

Při připojení MCU k napájení, dojde k bootování a pokud na pinu `PA14-BOOT0` je nízká logická úroveň, MCU načte program uložený ve FLASH paměti, který poté spustí. Při spuštění firmwaru sondy, proběhne inicializace globalních struktur, které jsou nezbytné pro chod celé sondy. Globální struktura poskytuje potřebná data různým periferiím, které například periferie využívají při přerušeních. Po inicializaci struktury, která je deklarována v úryvku kódu 16, dojde k inicializaci všech potřebných periferií, které dále jsou rozebrány v kapitole 4.3.

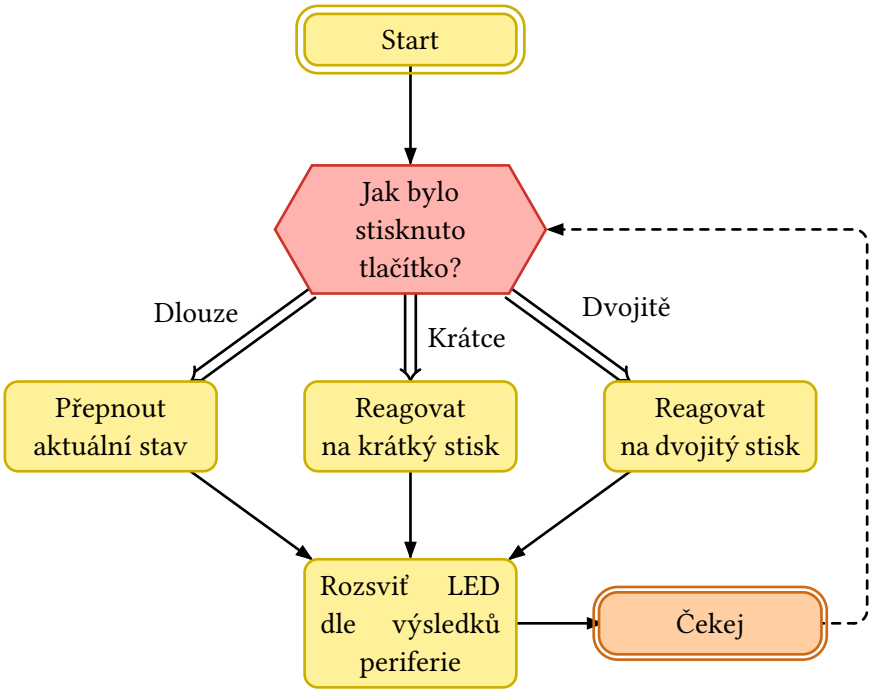
Po inicializaci periferií, sonda zkontroluje stav pinu `PA10` na kterém se nachází **Rx** USART1 periferie. Jak bylo zmíněno v kapitole 2.5.1, pokud jsou dvě zařízení propojeny a neprobíhá žádná komunikace, tak se na vodičích od `Tx` do `Rx` nachází logicky vysoká úroveň. Takto snda dokáže určit, zda je sonda připojena UART/USB převodníkem k PC, nebo je sonda pouze napájena např. skrze jiné MCU. Obrázek 21 a Obrázek 23 má v zapojení, na pinu `PA10`, rezistor o velikosti 10 $K\Omega$, který při nepřipojeném vodiči uzemní `Rx`. Obrázek 45 prezentuje způsob inicializace. Po zvolení režimu, zařízení přejde do různého nastavení, které jsou nutné pro fungování režimu. Opětovné nastavení režimu opět dojde při dalším bootu sondy, protože jednotlivé režimy běží v nekonečném cyklu dokud je zařízení napájeno.



Obrázek 45: Diagram nazpůsobu načítání režimů

5.2 Ovládání lokálního režimu

Jak kapitola 2.2 zmiňuje, lokální mód je provozní režim, v němž zařízení nekomunikuje s externím počítačem a veškerá interakce s uživatelem probíhá výhradně prostřednictvím tlačítka a RGB LED diody. Tento režim je vhodný pro prvotní rychlou diagnostiku logického obvodu. Režim se ovládá skrze tlačítko a informace jsou zobrazovány prostřednictvím RGB LED WS2812. Lokální režim běží ve smyčce, kdy se periodicky kontrolují změny a uživatelské vstupy. Způsob zobrazování barev na WS2812 je popsán v kapitole 4.9.



Obrázek 46: Diagram způsobu reakce na vstupy uživatele v lokálním módu

Při zmáčknutí tlačítka dojde k přerušení a je zavolána funkce z úryvku kódu 12, kde je zaznamenán čas zmáčknutí. Po uvolnění tlačítka dojde k přerušení náběžné hrany, a je zavolána funkce z úryvku kódu 13, kde je zaznamenán čas uvolnění a následně funkce `extern_button_check_press`, z úryvku kódu 28, porovná časy s referencí a určí, o který stisk se jedná. Funkce nastaví příznak v globální struktuře a v hlavní smyčce se poté provede příslušná akce. Tato metoda dokáže eliminovat nechtěné kmity tlačítka při stisku a uvolnění, kdy MCU zaznamenává velký počet hran v krátký moment (bouncing tlačítka).

Zařízení skrze tlačítko rozpozná tři interakce: *krátký stisk* slouží k přepínání logických úrovních na určitém kanálu, *dvojitý stisk* umožňuje cyklické přepínání mezi měřícími kanály, zatímco *dlouhý stisk* (nad 500 ms) zahájí změnu stavu. Při stisku tlačítka je signalizováno změnou barvy LED na 1 sekundu, kde barva určuje k jaké změně došlo. Tyto barvy jsou definovány v uživatelském manuálu přiložený k této práci.

5.3 Funkce lokálního režimu

Lokální režim má celkově 4 různé stavy, pro rychlou diagnostiku logického obvodu. Tyto funkce vychází ze standartních schopností komerečních logických sond. Stavy jsou přepínány dlouhým stiskem.

```c
void HAL_GPIO_EXTI_Falling_Callback(uint16_t GPIO_Pin) {
    // Pokud došlo k přerušení na pinu tlačítka
    if (GPIO_Pin == global_var.button_data->pin) {
        button_data_t* button_data = global_var.button_data;
        // zaznamenej čas, pokud tlačítko ještě nebylo stisknuto
        if (!button_data->is_pressed) {
            button_data->rise_edge_time = HAL_GetTick();
            button_data->is_pressed = true;
        }
    }
}
```

Úryvek kódu 12: Přerušení zavolané při zmáčknutí tlačítka

```c
void HAL_GPIO_EXTI_Rising_Callback(uint16_t GPIO_Pin) {
    // Pokud došlo k přerušení na pinu tlačítka
    if (GPIO_Pin == global_var.button_data->pin) {
        button_data_t* button_data = global_var.button_data;

        if (button_data->is_pressed) {
            // zaznamenej čas puštění tlačítka a zkontroluj
            // délku stisknutí
            button_data->fall_edge_time = HAL_GetTick();
            extern_button_check_press(button_data);
            button_data->is_pressed = false;
        }
    }
}
```

Úryvek kódu 13: Přerušení zavolané při uvolnění tlačítka

5.3.1 Funkce logické sondy

Při zapnutí zařízení se vždy nastaví stav **logické sondy**. Tento stav čte na příslušném kanálu periodicky, jaká logická úroveň je naměřena AD převodníkem. Jsou využity piny `PB7` a `PA0`. Logickou úroveň je možné číst také jako logickou úroveň na GPIO, nicméně to neumožňuje rozlišit stav, kdy logická úroveň je v neurčité oblasti. Pomocí měření napětí na pinu lze zjistit zda napětí odpovídá CMOS logice či nikoliv. Pokud na pinu se nachází vysoká úroveň, LED se rozsvítí zeleně, v případě nízké úrovně se rozsvítí červená a pokud je napětí v neurčité oblasti, LED nesvítí. Tlačítkem poté lze přepínat mezi jednotlivými kanály. Tento stav vychází z terminálové funkce pro měření napětí a následné zjistění logiky. Tento stav je vhodný pro rychlé zjistění úrovně na vodiči při diagnostice obvodu.

5.3.2 Funkce logických úrovní

Funkce logických úrovní je stav, který po stisku tlačítka změní logickou úroveň na opačnou, tzn. pin je nastaven jako push-pull a pokud je na pinu nízká úroveň, změní

se na vysokou a naopak. Tato úroveň lze nezávisle měnit na pinech `PB7` a `PA0`. RGB led poté barvou reprezentuje, v jakém stavu je pin nastaven. Při vstoupení do tohoto stavu uživatelem, jsou piny inicializovány funkcí z úryvku kódu 29. Tento stav je vhodný pro diagnostiku obvodu čítače, nebo klopných obvodů.

5.3.3 Funkce pulzování

Funkce pulzování je stav, kdy po stisku tlačíka je zapnuto na pinu `PA0` signál o frekvenci 1 Hz. Při zapnutí signálu na výstupu se rozsvítí RGB LED, aby si uživatel byl vědom aktivace. Po opětovném stisku tlačítka je vysílání signálu vypnuto. Takto nízká frekvence poskytuje možnost debugovat obvod, kde se například nachází posuvné registry nebo čítače. Uživatel například může vyzkoušet čítač, který zobrazuje čísla na sedmisegmentovém displeji a kontrolovat, zda se číslice mění korektně. Při vyšších frekvencích by toto znamenalo problém pro uživatele.

```c
void local_mode_pulse(void) {
    if (global_var.signal_generator->local_pulsing) {
        HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_0);
    } else {
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0, GPIO_PIN_RESET);
        neopixel_send_color(global_var.visual_output, NEOPIXEL_NONE);
    }
}
```

Úryvek kódu 14: Lokální režim pulzování ukázka obslužné smyčky

5.3.4 Funkce detekce pulzů

Detekce pulzu je stav, kdy je sledován pin `PA0` a detekována náběžná nebo sestupná hrana (detekce náběžné nebo sestupné hrany lze nastavit pomocí tlačítka). Pokud sonda detekuje hranu, rozsvítí RGB LED na 1 sekundu. Tento stav je vhodný pro detekci rychlého pulzu bez nutnosti použití osciloskopu. Sonda totiž detekuje velice krátkou hranu, ale uživatel ví, že pulz v obvodu byl, protože LED tento signál prodloužila.

Detekce funguje za pomocí časovače TIM2, který má nastavený kanál na režim input capture. Tento řežim při detekci hrany, uloží stav časovače do registru a je vyvoláno přerušení. Přerušení poté nastaví pomocný příznak, který bude zpracován při dalším cyklu smyčky. Obslužní smyčka poté rozsvítí LED. Důvod zvolení časovače místo přerušení EXTI, které vyvolá přerušení při hraně na vstupu, je ten, že tato metoda vychází z měření frekvencí v terminálovém režimu a tento způsob šetří paměťové zdroje, které jsou na tomto MCU velice omezené.

```c
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef* htim) {
    if (htim->Instance != TIM2) {
        return;
    }

    _Bool is_channel_1 = htim->Channel == HAL_TIM_ACTIVE_CHANNEL_1;
    _Bool is_channel_2 = htim->Channel == HAL_TIM_ACTIVE_CHANNEL_2;
    _Bool is_rise_edge_mode =
        global_var.detector->mode == DETECTOR_MODE_RISE_EDGE
    _Bool is_rise_edge_mode =
        global_var.detector->mode == DETECTOR_MODE_FALL_EDGE;

    if ((is_channel_1 && is_rise_edge_mode) ||
        (is_channel_2 && is_fall_edge_mode)) {
        detector->one_pulse_found = true;
    }
}
```

Úryvek kódu 15: Zachytávání pulzů v lokálním režimu

Kapitola 6

Návrh omezené verze na RPI Pico

Návrh omezené verze na Rasberry Pi Pico **TODO: dopsat**

6.1 Komunikace s PC

6.2 Měření napětí a logických úrovní

6.3 Měření frekvence a detekce pulzů

6.4 Generování pulzů

Kapitola 7

Závěr a zhodnocení

Příloha A

Citace

[1] STMicroelectronics, „STM32G0-ADC: Product training". Viděno: 20. květen 2025. [Online]. Dostupné z: https://www.st.com/resource/en/product_training/STM32G0-Analog-ADC-ADC.pdf

[2] STMicroelectronics, „STM32G0x1 Reference manual". Viděno: 20. květen 2025. [Online]. Dostupné z: https://www.st.com/resource/en/reference_manual/rm0444--stm32g0x1-advanced-armbased-32bit-mcus-stmicroelectronics.pdf

[3] Mahamudul Hasan, „Understanding STM32 HAL Library Fundamentals". Viděno: 20. květen 2025. [Online]. Dostupné z: https://embeddedthere.com/understanding--stm32-hal-library-fundamentals/

[4] Joseph Wu - TI, „A Basic Guide to I2C". Viděno: 20. květen 2025. [Online]. Dostupné z: https://www.ti.com/lit/an/sbaa565/sbaa565.pdf

[5] Wikimedia Commons, „File:SPI timing diagram msbfirst.svg — Wikimedia Commons, the free media repository". Viděno: 20. květen 2025. [Online]. Dostupné z: https://commons.wikimedia.org/w/index.php?title=File:SPI_timing_diagram_msbfirst.svg&oldid=719502258

[6] Worldsemi, „WS2812D-F5-1261 Intelligent control LED integrated lightsource". Viděno: 20. květen 2025. [Online]. Dostupné z: https://www.tme.eu/Document/bd6b355a8705d46515a8ada0d153187b/WS2812D-F5-L.pdf

[7] doc. Ing. Jan Fischer, CSc., „ETC22 Přednáška 2". Viděno: 20. květen 2025. [Online]. Dostupné z: https://embedded.fel.cvut.cz/sites/default/files/kurzy/ETC22/Prednasky_ETC22_E/ETC22E_2_pr_2024_10_7_G030_Osciloskop.pdf

[8] STMicroelectronics, „STM32G030x6/x8". Viděno: 20. květen 2025. [Online]. Dostupné z: https://www.farnell.com/datasheets/2882477.pdf

[9] STMicroelectronics, „STM32G0 Series". Viděno: 20. květen 2025. [Online]. Dostupné z: https://www.st.com/en/microcontrollers-microprocessors/stm32g0-series.html

[10] NI, „What is a phase-locked loop (PLL)?". Viděno: 20. květen 2025. [Online]. Dostupné z: https://knowledge.ni.com/KnowledgeArticleDetails?id=kA00Z000000P9T3SAK&l=cs-CZ

[11] iter, „Calibrating STM32 ADC (VREFINT)". Viděno: 20. květen 2025. [Online]. Dostupné z: https://stackoverflow.com/questions/58328342/calibrating-stm32-adc--vrefint

[12] Michal Dudka, „STM32 Timery". Viděno: 20. květen 2025. [Online]. Dostupné z: http://www.elektromys.eu/clanky/stm_timer1/clanek.html

[13] STMicroelectronics, „STM32CubeMX". Viděno: 20. květen 2025. [Online]. Dostupné z: https://www.st.com/content/st_com/en/stm32cubemx.html

[14] Mahamudul Hasan, „Understanding STM32 HAL Library Fundamentals". Viděno: 20. květen 2025. [Online]. Dostupné z: https://embeddedthere.com/understanding-stm32-hal-library-fundamentals/

[15] Arm Limited, „What is CMSIS?". Viděno: 20. květen 2025. [Online]. Dostupné z: https://www.keil.arm.com/cmsis

[16] The Rasberry Foundation, „Raspberry Pi Pico datasheet". Viděno: 20. květen 2025. [Online]. Dostupné z: https://datasheets.raspberrypi.com/pico/pico-datasheet.pdf

[17] All About Circuits, „Logic Signal Voltage Levels". Viděno: 20. květen 2025. [Online]. Dostupné z: https://www.allaboutcircuits.com/textbook/digital/chpt-3/logic-signal-voltage-levels/

[18] Nave, R., Viděno: 20. květen 2025. [Online]. Dostupné z: http://hyperphysics.phy-astr.gsu.edu/hbase/electric/voldiv.html

[19] Germanna Academic Center, Viděno: 20. květen 2025. [Online]. Dostupné z: https://germanna.edu/sites/default/files/2022-03/Ohms%20and%20Kirchoffs%20Laws.pdf

[20] Gavin Wright, „Definition USART (universal synchronous/asynchronous receiver/transmitter)". Viděno: 20. květen 2025. [Online]. Dostupné z: https://www.techtarget.com/whatis/definition/USART-Universal-Synchronous-Asynchronous-Receiver-Transmitter

[21] University of Wisconsin–Madison, „Uart Basics". Viděno: 20. květen 2025. [Online]. Dostupné z: https://ece353.engr.wisc.edu/serial-interfaces/uart-basics/

[22] V. Hunter Adams, „Universal Asynchronous Receiver Transmitter". Viděno: 20. květen 2025. [Online]. Dostupné z: https://vanhunteradams.com/Protocols/UART/UART.html

[23] Piyu Dhaker, „Introduction To SPI Interface". Viděno: 20. květen 2025. [Online]. Dostupné z: https://www.analog.com/en/resources/analog-dialogue/articles/introduction-to-spi-interface.html

[24] HOLTEK, „HT75XX-1 100mA Low Power LDO". Viděno: 20. květen 2025. [Online]. Dostupné z: https://img.gme.cz/files/eshop_data/eshop_data/3/330-201/dsh.330-201.1.pdf

[25] Autodesk, „Eagle". Viděno: 20. květen 2025. [Online]. Dostupné z: https://www.autodesk.com/products/eagle/overview

[26] Albert van Dalen, „addressable LED rings NeoPixel WS2812B-2020". Viděno: 20. květen 2025. [Online]. Dostupné z: https://github.com/avandalen/Eagle-library-addressable-LED-rings

[27] G. standards, Viděno: 20. květen 2025. [Online]. Dostupné z: https://enxstandards.web.illinois.edu/standard/ansi-x3-4/

[28] fnky, „ANSI Escape Sequences". Viděno: 20. květen 2025. [Online]. Dostupné z: https://gist.github.com/fnky/458719343aabd01cfb17a3a4f7296797

[29] Texas Instruments, „SNX4HC595 8-bit shift registers with 3-state output ". Viděno: 20. květen 2025. [Online]. Dostupné z: https://www.ti.com/lit/ds/symlink/sn74hc595.pdf

Příloha B

Dodatečné úryvky kódu

```c
typedef struct {
    _Bool booted; // true pokud uživatel připojil seriovou komunikací
    dev_setup_t device_setup; // Lokální/Terminálový mód
    dev_state_t device_state; // měřící mód (napětí, odpor atd.)
    local_state_t local_state; // lokální režim měřící mód
    local_substate_t local_substate; // lokální režim měřený kanál
    unsigned char received_char; // Znak jako vstup
    _Bool need_frontend_update; // příznak aktualizace TUI
    _Bool need_perif_update; // příznak nastavení periferií
    ansi_page_type_t current_page; // aktualní frontend stránka

    // struktury pro základní měření
    adc_vars_t* adc_vars;
    sig_detector_t* signal_detector;
    sig_generator_t* signal_generator;

    // struktury pro vstupy/výstupy uživatele
    visual_output_t* visual_output;
    button_data_t* button_data;

    // struktury pro pokročilé funkce
    neopixel_measure_t* adv_neopixel_measure;
    shift_register_t* adv_shift_register;
    uart_perif_t* uart_perif;
    i2c_perif_t* i2c_perif;
    spi_perif_t* spi_perif;
} global_vars_t;
```

Úryvek kódu 16: Struktura globálních promněných

```c
void ansi_send_text(const char* str,
                    const ansi_text_config_t* text_conf) {
    char buffer[TEXT_SEND_BUFF_SIZE];
    // offset stringu pro přidávání textu
    size_t offset = 0;
    // Při nastavení barvy přidej ansi sekvenci pro zbarvení
    if (strlen(text_conf->bg_color) != 0) {
        offset += snprintf(buffer + offset, sizeof(buffer) - offset,
        "%s",
                           text_conf->bg_color);
    }

    // Při nastavení barvy přidej ansi sekvenci pro zbarvení
    if (strlen(text_conf->color) != 0) {
        offset += snprintf(buffer + offset, sizeof(buffer) - offset,
        "%s",
                           text_conf->color);
    }

    // Nastav text tučně
    if (text_conf->bold) {
        offset +=
            snprintf(buffer + offset, sizeof(buffer) - offset, "%s",
            BOLD_TEXT);
    }

    offset += snprintf(buffer + offset, sizeof(buffer) - offset,
    "%s", str);

    // Kontrola velikosti bufferu
    if (offset >= sizeof(buffer)) {
        PrintError("Buffer overflow in text");
        return;
    }
    ansi_send_string(buffer);
    // Escapování všech nastavení na konci stringu
    ansi_clear_format();
}
```

Úryvek kódu 17: Funkce pro vykreslení barevného textu

```c
void get_current_control(void) {
    char received_char = global_var.received_char;
    switch (global_var.current_page) {
        case ANSI_PAGE_MAIN:
            control_main_page();
            break;
        case ANSI_PAGE_MAIN_ADVANCED:
            control_advanced_main_page();
            break;
        case ANSI_PAGE_VOLTAGE_MEASURE:
            control_voltage_page(received_char);
            break;
        case ANSI_PAGE_FREQUENCY_READER:
            control_frequency_reader_page(received_char,
            global_var.signal_detector);
            break;
        case ANSI_PAGE_IMPULSE_GENERATOR:
            control_impulse_generator_page(received_char);
            break;
        case ANSI_PAGE_LEVELS:
            control_levels_page(received_char);
            break;
        case ANSI_PAGE_NEOPIXEL_MEASURE:
            control_neopixel_measure_page(received_char);
            break;
        case ANSI_PAGE_SHIFT_REGISTER:
            control_shift_register_page(received_char);
            break;
        case ANSI_PAGE_UART:
            control_uart_page(received_char);
            break;
        case ANSI_PAGE_I2C:
            control_i2c_page(received_char);
            break;
        case ANSI_PAGE_SPI:
            control_spi_page(received_char);
            break;
        default:
            control_main_page();
    }
}
```

Úryvek kódu 18: Ovládání sondy skrze symboly

```c
    void control_main_page(void) {
    switch (global_var.received_char) {
        case 'v':
        case 'V':
            ansi_set_current_page(ANSI_PAGE_VOLTAGE_MEASURE);
            dev_mode_change_mode(DEV_STATE_VOLTMETER);
            break;
        case 'f':
        case 'F':
            ansi_set_current_page(ANSI_PAGE_FREQUENCY_READER);
            dev_mode_change_mode(DEV_STATE_FREQUENCY_READ);
            break;
        case 'g':
        case 'G':
            ansi_set_current_page(ANSI_PAGE_IMPULSE_GENERATOR);
            dev_mode_change_mode(DEV_STATE_PULSE_GEN);
            break;
        case 'l':
        case 'L':
            if (NOT_SOP) {
                ansi_set_current_page(ANSI_PAGE_LEVELS);
                dev_mode_change_mode(DEV_STATE_LEVEL);
            }
            break;
        case 'a':
        case 'A':
            if (NOT_SOP) {
                ansi_set_current_page(ANSI_PAGE_MAIN_ADVANCED);
                dev_mode_change_mode(DEV_STATE_NONE);
            }
            break;
    }
}
```

Úryvek kódu 19: Ovládání menu

```c
static void MX_TIM3_Init(void) {
    TIM_ClockConfigTypeDef sClockSourceConfig = {0};
    TIM_MasterConfigTypeDef sMasterConfig = {0};

    htim3.Instance = TIM3;
    htim3.Init.Prescaler = 64000 - 1;
    htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim3.Init.Period = 1000 - 1; // základní nastavení které bude změněno
    htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
    if (HAL_TIM_Base_Init(&htim3) != HAL_OK) {
        Error_Handler();
    }
    sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
    if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK) {
        Error_Handler();
    }
    if (HAL_TIM_OnePulse_Init(&htim3, TIM_OPMODE_SINGLE) != HAL_OK) {
        Error_Handler(); // Časovač se sám zastaví po přetečení
    }
    sMasterConfig.MasterOutputTrigger = TIM_TRGO_ENABLE; // trigger pro slave časovač
    sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
    if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) !=
        HAL_OK) {
        Error_Handler();
    }
}
```

Úryvek kódu 20: Inicializace TIM3 jako časovač

```c
uint32_t adc_get_voltage(const uint32_t v_ref,
                         const uint32_t measure) {
    return __HAL_ADC_CALC_DATA_TO_VOLTAGE(v_ref, measure,
                                          ADC_RESOLUTION_12B);
}

uint32_t adc_get_v_ref( const uint32_t raw_voltage_value) {
    return __HAL_ADC_CALC_VREFANALOG_VOLTAGE(raw_voltage_value,
                                             ADC_RESOLUTION_12B);
}
```

Úryvek kódu 21: Využití HAL maker pro převod poměrových hodnot na napětí

```c
uint32_t ref_voltage = adc_get_v_ref(adc_ch->avg_voltage[0]);
uint32_t measured_voltage = adc_get_voltage(ref_voltage,
                                            adc_ch->avg_voltage[1]);
uint32_t resistance = (adc_ch->base_resistor * measured_voltage)
                / (ref_voltage - measured_voltage);
```

Úryvek kódu 22: Způsob výpočtu odporu

```c
void timer_setup_slave_freq(void) {
    TIM_ClockConfigTypeDef sClockSourceConfig = {0};
    TIM_SlaveConfigTypeDef sSlaveConfig = {0};
    TIM_MasterConfigTypeDef sMasterConfig = {0};

    slave_tim->Instance = TIM2;
    slave_tim->Init.Prescaler = 0; // nastaven na maximum
    slave_tim->Init.CounterMode = TIM_COUNTERMODE_UP;
    slave_tim->Init.Period = 4294967295;
    slave_tim->Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    slave_tim->Init.AutoReloadPreload =
TIM_AUTORELOAD_PRELOAD_DISABLE;
    if (HAL_TIM_Base_Init(slave_tim) != HAL_OK) {
        Error_Handler();
    }
    // inicializace externího clocku
    sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_ETRMODE2;
    sClockSourceConfig.ClockPolarity = TIM_CLOCKPOLARITY_NONINVERTED;
    sClockSourceConfig.ClockPrescaler = TIM_CLOCKPRESCALER_DIV1;
    sClockSourceConfig.ClockFilter = 0;
    if (HAL_TIM_ConfigClockSource(slave_tim, &sClockSourceConfig) !=
HAL_OK) {
        Error_Handler();
    }

    // nastavení závislosti spuštění na TIM3
    sSlaveConfig.SlaveMode = TIM_SLAVEMODE_TRIGGER;
    sSlaveConfig.InputTrigger = TIM_TS_ITR2;
    if (HAL_TIM_SlaveConfigSynchro(slave_tim, &sSlaveConfig) !=
HAL_OK) {
        Error_Handler();
    }
    sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
    sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
    if (HAL_TIMEx_MasterConfigSynchronization(slave_tim,
&sMasterConfig) !=
        HAL_OK) {
        Error_Handler();
    }
}
```

Úryvek kódu 23: Inicializace TIM2 jako čítač hran

```c
void shift_register_send_one_signal(shift_register_t*
shift_register,
                                    const uint8_t index) {
    if (shift_register->bits[index]) {
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0, GPIO_PIN_SET);
    } else {
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0, GPIO_PIN_RESET);
    }
    HAL_Delay(SHIFT_REGISTER_LATCH);
    HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_7);
    HAL_Delay(SHIFT_REGISTER_LATCH);
    HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_7);
    HAL_Delay(SHIFT_REGISTER_LATCH);
}
```

Úryvek kódu 24: Způsob zápisu jednoho bitu do posuvného registru

```c
void ansi_render_read_vals(uart_perif_t* uart) {
    // získání pozice začátku
    uint16_t curr_buff_index =
        UART_BUFFER_SIZE - __HAL_DMA_GET_COUNTER(uart->huart->hdmarx);

    char buff[11];
    uint8_t col = 0;
    uint8_t row = 12;
    ansi_set_cursor(row, 5);

    for (size_t i = 0; i < UART_BUFFER_SIZE; ++i) {
        if (curr_buff_index >= UART_BUFFER_SIZE) {
            curr_buff_index = 0;
        }
        if (uart->received_char[curr_buff_index] == 0) {
            ++curr_buff_index;
            continue;
        }

        ++col;
        snprintf(buff, 11, " %c(%3d)", uart->received_char[curr_buff_index],
                 uart->received_char[curr_buff_index]);
        ansi_send_text(buff, &ansi_default_conf);

        if (col >= 10) {
            col = 0;
            ++row;
            ansi_set_cursor(row, 5);
        }
        ++curr_buff_index;
    }
}
```

Úryvek kódu 25: Způsob vykreslování UART symbolů scrollováním

```c
uint8_t init_sequence[] = {0xAE, 0xD5, 0x80, 0xA8, 0x3F, 0xD3,
                           0x00, 0x40, 0x8D,0x14, 0x20, 0x02,
                           0xA0, 0xC0, 0xDA, 0x12, 0x81, 0xCF,
                           0xD9, 0xF1, 0xDB, 0x40, 0xA5, 0xA6, 0xAF};

HAL_StatusTypeDef SSD1306_spi_init_display(SPI_HandleTypeDef* hspi) {
    HAL_StatusTypeDef status;

    for (uint8_t i = 0; i < sizeof(init_sequence); i++) {
        if ((status = SSD1306_spi_write_command(hspi,
        init_sequence[i]))) {
            return status;
        }
        HAL_Delay(40);
    }
    return HAL_OK;
}
```

Úryvek kódu 26: Inicializace SSD1306 displeje I2C

```c
void i2c_monitor_init(i2c_perif_t* i2c_perif, const spi_perif_t*
spi_perif) {
    spi_perif->hspi->Instance = SPI1;
    spi_perif->hspi->Init.Mode = SPI_MODE_SLAVE;
    spi_perif->hspi->Init.Direction = SPI_DIRECTION_2LINES;
    spi_perif->hspi->Init.DataSize = SPI_DATASIZE_9BIT;
    spi_perif->hspi->Init.CLKPolarity = SPI_POLARITY_HIGH;
    spi_perif->hspi->Init.CLKPhase = SPI_PHASE_2EDGE;
    spi_perif->hspi->Init.NSS = SPI_NSS_SOFT;
    spi_perif->hspi->Init.FirstBit = SPI_FIRSTBIT_MSB;
    spi_perif->hspi->Init.TIMode = SPI_TIMODE_DISABLE;
    spi_perif->hspi->Init.CRCCalculation =
SPI_CRCCALCULATION_DISABLE;
    spi_perif->hspi->Init.CRCPolynomial = 7;
    spi_perif->hspi->Init.CRCLength = SPI_CRC_LENGTH_DATASIZE;
    spi_perif->hspi->Init.NSSPMode = SPI_NSS_PULSE_DISABLE;
    if (HAL_SPI_Init(spi_perif->hspi) != HAL_OK) {
        Error_Handler();
    }
    memset(i2c_perif->monitor_data, 0, I2C_ARRAY_SIZE);
    gpio_spi_slave_init();
    HAL_SPI_Receive_DMA(spi_perif->hspi, (uint8_t*)i2c_perif-
>monitor_data,
                        I2C_ARRAY_SIZE * 2);
}
```

Úryvek kódu 27: I2C inicializace periferie pro monitoring

```c
void extern_button_check_press(button_data_t* data) {
    uint32_t time = data->fall_edge_time - data->rise_edge_time;

    // identifikace typu stisknutí
    if (time > SHORT_PRESS_TIME && time < LONG_PRESS_TIME) {
        data->short_press = true;
        data->long_press = false;
        uint32_t curr_time = HAL_GetTick();

        // detekce druhého krátkého stisknutí
        if (curr_time - data->last_short_button_time < DOUBLE_PRESS_TIME &&
            !data->double_press) {
            data->double_press = true;
            data->short_press = false;
        } else {
            data->last_short_button_time = curr_time;
        }

    } else if (time > LONG_PRESS_TIME && !data->long_press) {
        data->long_press = true;
        data->short_press = false;
        data->double_press = false;
    }
}
```

Úryvek kódu 28: Způsob detekce boucingu

```c
void gpio_init_push_pull(void) {
    gpio_deinit_pins();

    GPIO_InitTypeDef GPIO_InitStruct = {0};
    GPIO_InitStruct.Pin = GPIO_PIN_0;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

    GPIO_InitStruct.Pin = GPIO_PIN_7;
    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
    GPIO_InitStruct.Pin = GPIO_PIN_1;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7, GPIO_PIN_RESET);
}
```

Úryvek kódu 29: Inicializace pinů lokálního režimu pro stav logických úrovní

Příloha C

Uživatelská příručka