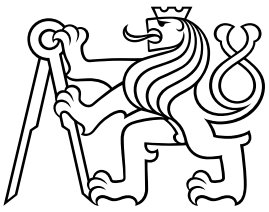
Bakalářská práce



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

Abstrakt

Výuka základů elektroniky vyžaduje nástroje, které studentům umožní experimentovat s realnými obvody a pochopit principy jejich fungování. Tradiční konvenční nástroje postrádají flexibilitu pro výukové účely a mohou být příliš komplikované pro osobu, která teprve objevuje vlastnosti elektronických obvodů. Tato práce reaguje na tuto potřebu návrhem multifunkční logické sondy, která kombinujee funkce logického analyzátoru, generátoru signálů a testeru komunikačních rozhraní. Její klíčovou výhodou je možnost jednoduchého sestavení s využitím dostupných mikrořadičů, což ji předurčuje pro využití ve výuce.

Sonda existuje ve dvou variantách: plnohodnotné na bázi STM32 a omezené na bázi Raspberry Pi Pico. V základním řežimu poskytuje detekci logických úrovní, měření frekvence, generaci impulsů a měření napětí. Rozšířená verze poskytuje diagnostiku známých seriových rozhraní (UART, I2C, SPI, Neopixel). Integrace s PC terminálem umožňuje používání pokročilých funkcí, zatímco lokální režim slouží pro rychlou analýzu bez nutnosti PC.

Hardwarový návrh je optimalizován pro minimalizaci externích komponent s důrazem na využití interních periferií mikrořadičů, což umožňuje sestavení zařízení na nepájivém kontaktním poli. Součástí práce je firmware, dokumentace a návody pro studenty, které pokrývají sestavení sondy i příklady jejího využití. Výsledkem je open-source řešení, které lze dále rozšiřovat a přizpůsobovat specifickým vzdělávacím potřebám.

Klíčová slova: STM32, Raspberry Pi Pico, logická sonda,

Abstract

Teaching the fundamentals of electronics requires tools that allow students to experiment with real circuits and understand their principles of operation. Traditional conventional tools lack flexibility for teaching purposes and may be too complicated for a person who is just discovering the properties of electronic circuits. This work addresses this need by designing a multifunctional logic probe that combines the functions of a logic analyzer, signal generator, and communication interface tester. Its key advantage is the possibility to be assembled simply using available microcontrollers, which makes it suitable for use in teaching.

The probe exists in two variants: a full-featured STM32-based and a limited Raspberry Pi Pico-based. In basic mode, it provides logic level detection, frequency measurement, pulse generation and voltage measurement. The extended version provides diagnostics of known serial interfaces (UART, I2C, SPI, Neopixel). Integration with a PC terminal allows the use of advanced functions, while local mode is used for fast analysis without the need for a PC.

The hardware design is optimized to minimize external components with an emphasis on the use of internal microcontroller peripherals, allowing the device to be assembled on a non-soldering contact array. The thesis includes firmware, documentation and student tutorials that cover the probe build and examples of its use. The result is an open-source solution that can be further extended and adapted to specific educational needs.

Keywords: STM32, Raspberry Pi Pico, logic probe, measurement tools, …

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.

Prohlášení

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

v Praze, 11. 05. 2025

8 TODOs remaining

Obsah

Seznam obrázků

Seznam tabulek

Seznam úryvků kódu

Seznam zkratek

SOP	Small Outline Package
TSSOP	Thin Shrink Small Outline Package
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

TODO: ROZPRACOVÁNO, PŘEDELAT Vzdělávání v oblasti elektrotechniky a elektroniky vyžaduje nejen hluboké teoretické znalosti, ale také praktické dovednosti a umění si poradit s naskytnutým problémem. Pro řešení nejrůznějších překážek při navrhování elektronických obvodů je podstatné vědět, jak používat specializované nástroje, které umožňují chybu odhalit. Nástroje jsou také důležité ve školách, kde student může pomocí praktických příkladů zjistit teoretické zákonitosti. Nástroje studentům pomáhají pochopit chování a vlastnosti elektronických obvodů v realném světě. Jedním z těchto nástrojů je logická sonda, zařízení používané k analýze digitálních signálů a diagnostice obvodů. Aby nástroj přispěl ke vzdělávání, je žádoucí aby takové zařízení bylo intuitivní, multifunkční a přizpůsobené náležitostem laboratorní výuky.

Tato bakalářská práce se zaměřuje na návrh a realizaci logické sondy, která bude navržena a optimalizována pro využití ve výuce středoškolských oborů či ve výuce vysokoškolských oborů. Cílem je vytvořit zařízení, které umožní méně zkušeným studentům provádět klíčové diagnostické úkony, jako například, nastavování úrovní signálů, odchytávání signálů, měření frekvence, měření napětí, generování signálů a další. Sonda bude navržena s důrazem na jednoduché ovládání a to prostřednictvím UART, aby byla snadno použitelná i pro studenty, kteří se s elektronikou setkávají poprvé.

V této práci budou představeny požadavky na zařízení a realizace této logické sondy.

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]. Pro

[1] Např. PuTTY, GTKTerm…

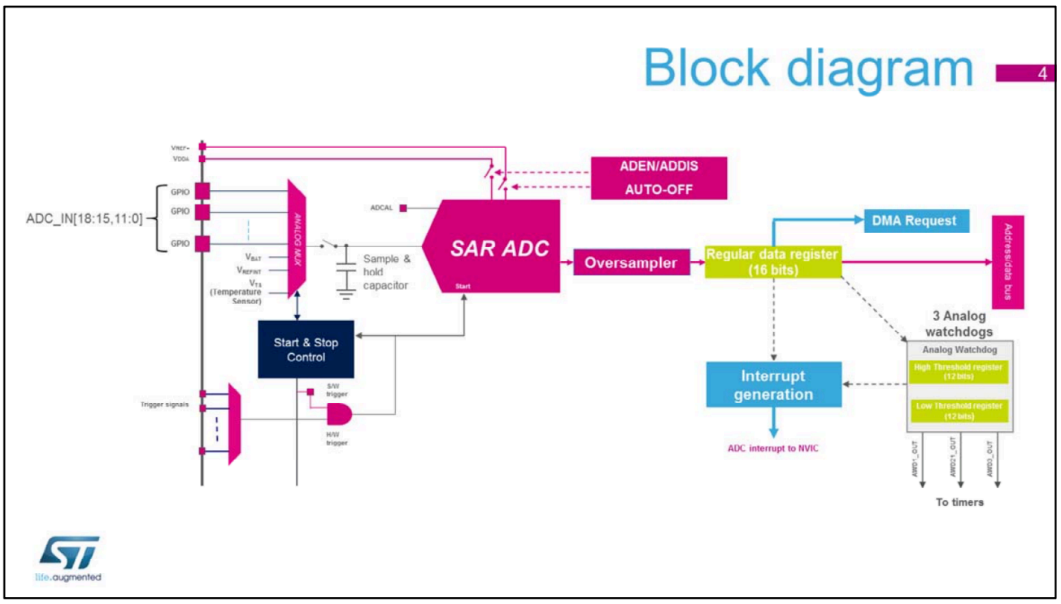
řadu G031 jsou typické kompaktní rozměry ať už vývojové Nucleo desky, tak typové pouzdra jako například **TSSOP20** nebo **SOP8**, což poskytuje snadnou integraci do kompatního hardwarového návrhu [8]. Obě zmíněná pouzdra jsou použita pro implementaci logické sondy, o které pojednává kapitola 3.

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 ($V_{\text{REF}+}$). Referenční 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



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

$V_{\mathrm{REF+}}$ 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], [10].
- 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. [11].

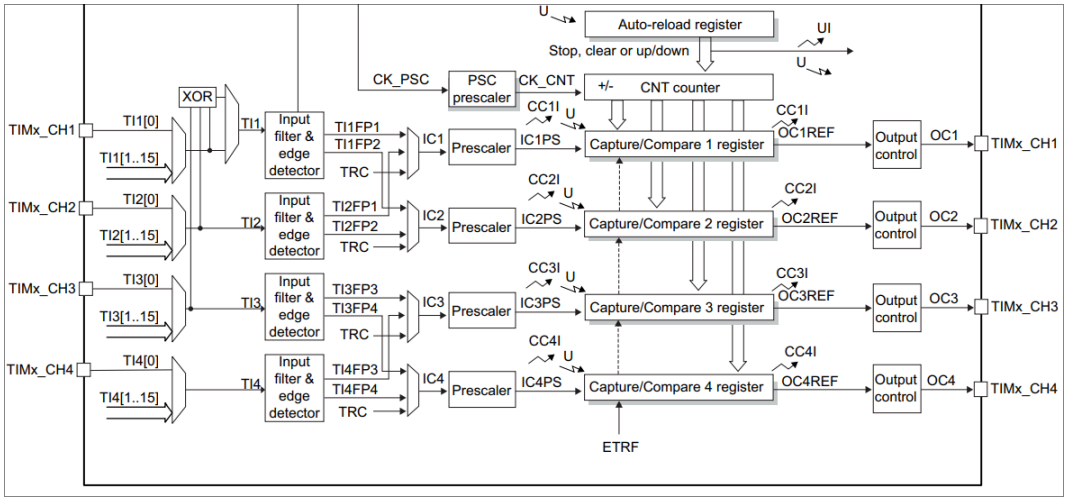
Před spuštěním časovače je potřeba nastavit, jak často má časovač čítat. Frekvenci časovače nastavuje tzn. prescaler, neboli „předdělička“. Prescaler dělí s konstantou, která je zvolena, frekvenci hodin dané periferie. Pro případ STM32G0 je to 64 MHz. Frekvence časovače určuje, jak často časovač inkrementuje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

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

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

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

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



Obrázek 2: Blokové schéma STM32G031 časovače [2]

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.

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

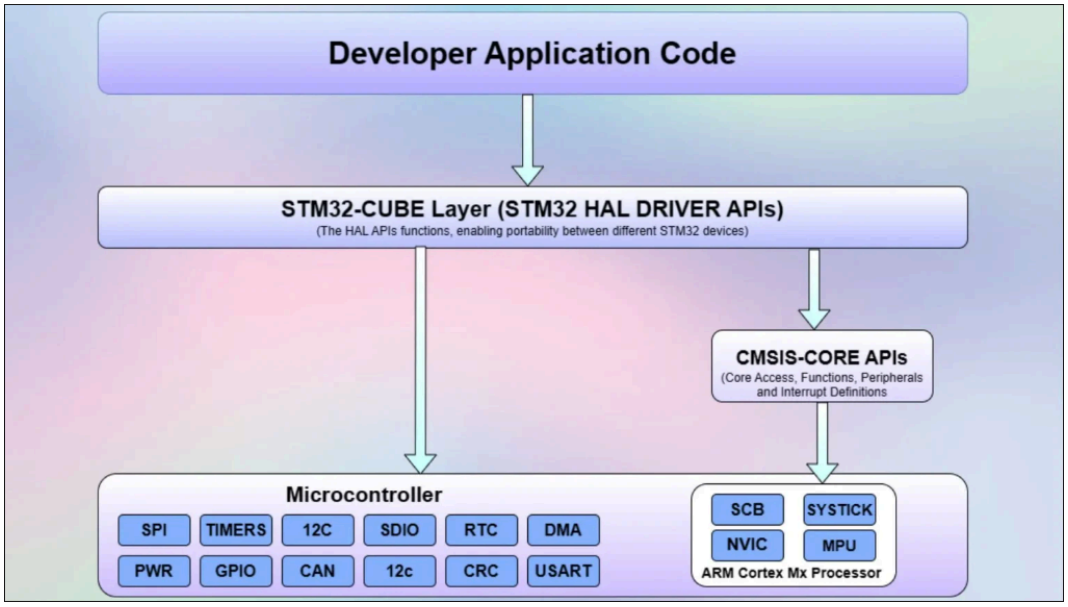
2.3.2 Knihovna STM HAL

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 [12].

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 [13].

Součástí HALu je tzv. CMSIS, což je sada standardizovanách rozhraní, které umožňují konfiguraci periferií, správu jádra, obsluhu přerušení a další [14]. 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 definice registrů, přístup k NVIC apod. [14]. 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.

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



Obrázek 3: STM32CubeMX HAL architektura [3]

2.4 Měření veličin testovaného obvodu

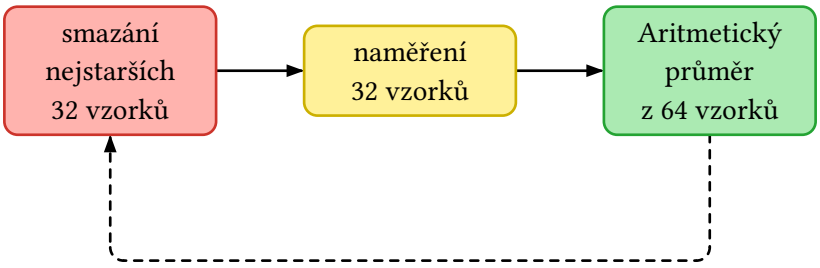
2.4.1 Měření napětí a logických úrovní

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] [15].

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



Obrázek 4: 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 [16]. 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 5. 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 [17], [18]:

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

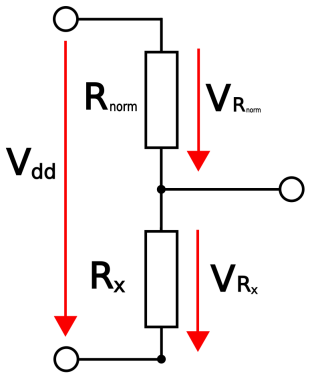
Pomocí pravidla pro napěťový dělič lze vztah mezi napětími a odpory vyjádřit jako:

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

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 [17]:

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

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 5). Sonda následně změří napětí V_{dd} na kanálu 1 ADC a pomocí rovnice 11, vypočítá hodnotu neznámého odporu [17].



Obrázek 5: Schéma děliče napětí

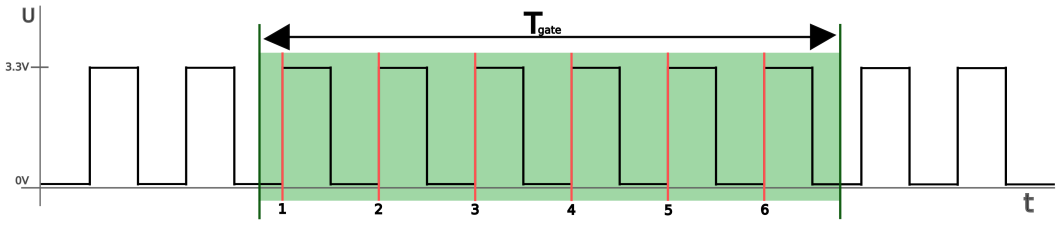
2.4.3 Měření frekvence

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 6: Signál při měření metodou hradlování

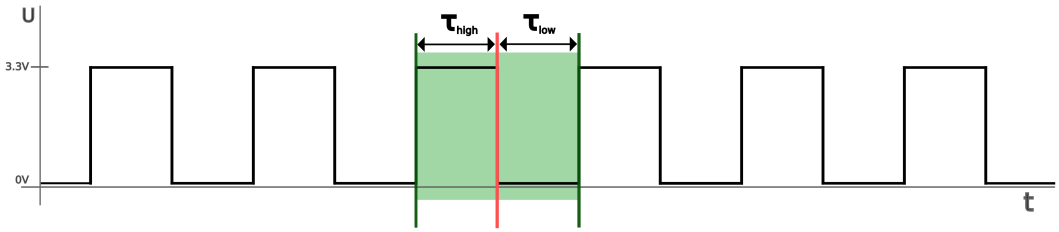
Reciproční frekvence

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 7: 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 [19], [20].

Data jsou přenášena v tzv. rámcích, které jsou strukturovány následovně: [20]

- **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

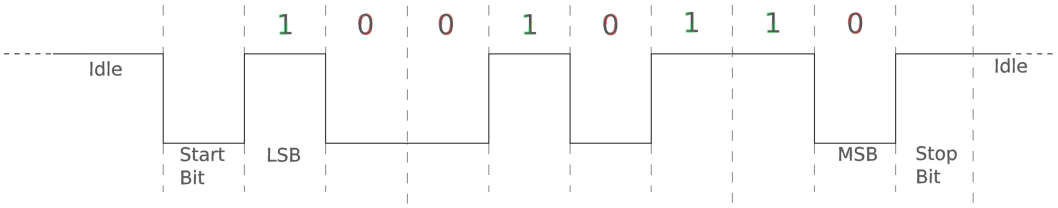
[13]Lze používat i 7 bitů nebo 9 bitů dat.

[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 8 ukazuje způsob zpracování signálů [21]. 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.

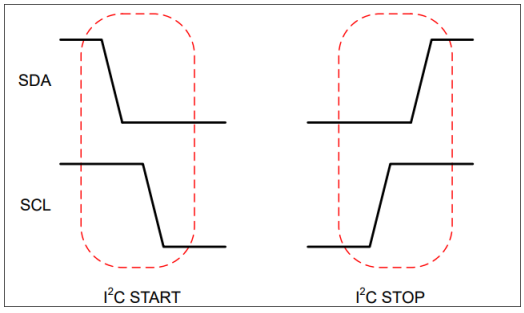


Obrázek 8: Způsob zpracování signálu UART bez parity

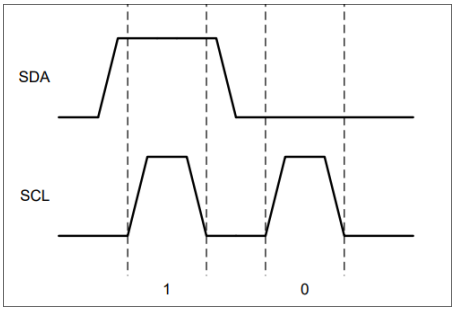
2.5.2 I2C Bus

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].

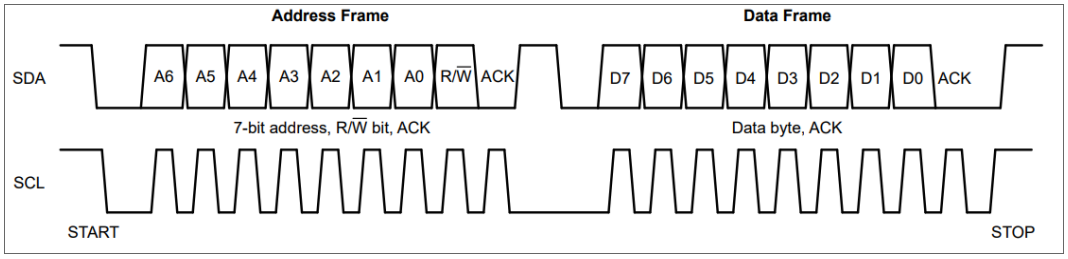


Obrázek 9: Zahájení a ukončení komunikace v I2C [4]



Obrázek 10: Logická jednička a logická nula v I2C [4]

[15] Stop bitů může být i několik



Obrázek 11: 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 10 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.

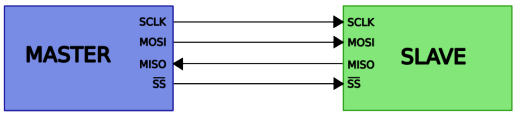
SPI má vždy jedno master zařízení a i několik podřízených slave zařízení. SPI je synchronní full duplex rozhraní, které má celkem 4 vodiče[16]. `SCLK`, což je hodinový signál, který určuje synchronizaci dat, `MOSI`, neboli vodič, kde probíhá komunikace od masteru ke slave, `MISO`, kde probíhá komunikace 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 [22]. Obrázek 12 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`.

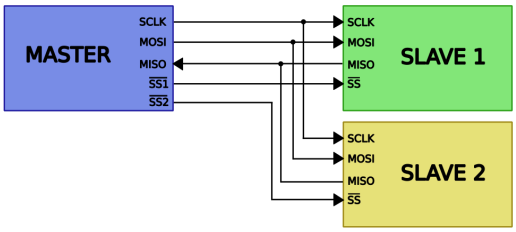
[17] Někdy také `CS` jako chip select.

[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 13 znázorňuje připojení více slave zařízení. V tomto případě master využívá více `SS` a podle přivedení nízké logické úrovně určuje směr komunikace [22].

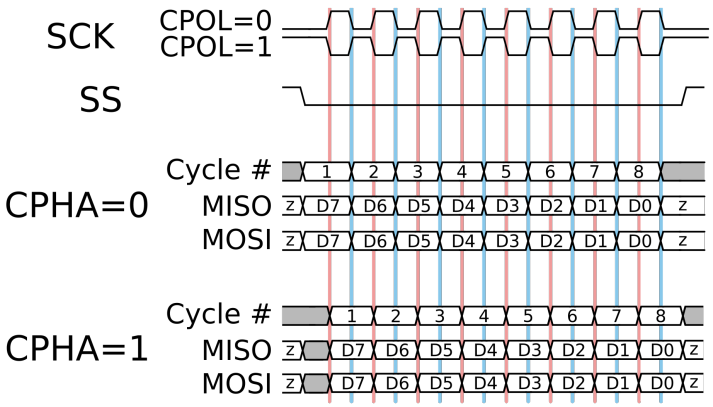


Obrázek 12: Diagram SPI komunikace s jedním slave zařízením



Obrázek 13: 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].

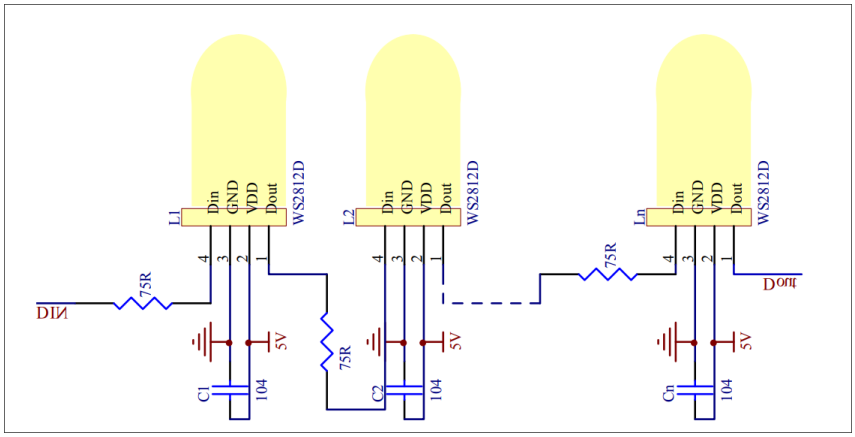


Obrázek 14: Časový diagram SPI zobrazující úroveň a posun hodinového signálu [5]

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 15 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.



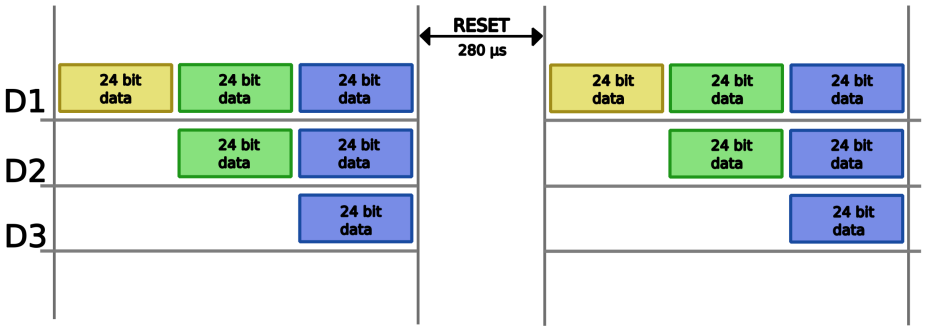
Obrázek 15: Způsob zapojení RGB LED do série [6]

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.



Obrázek 16: Diagram posílání dat pro zapojené WS2812D v sérii [6]

[20] Most significant bit

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	> 280000

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

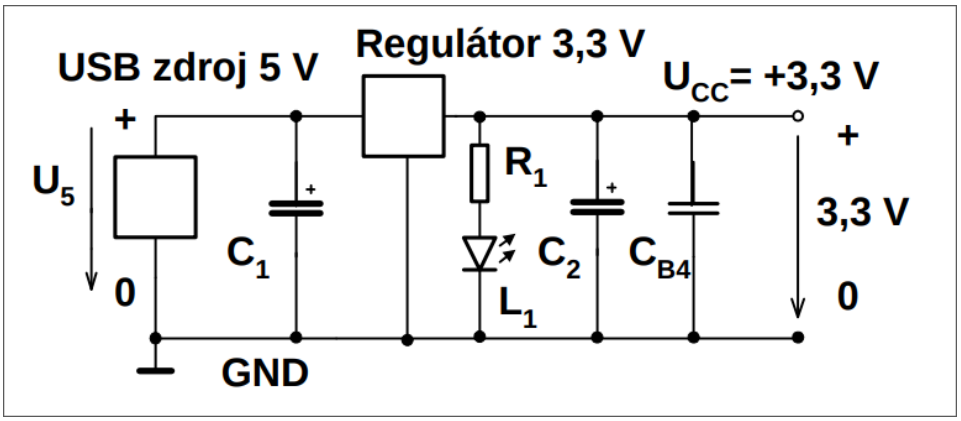
Kapitola 3

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 [23].



Obrázek 17: 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], [16]. 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.

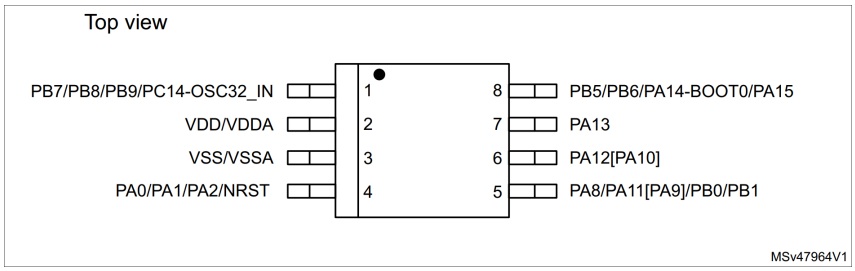
Obě pouzdra využívají pro komunikaci s PC periferii USART1. STM32 poskytuje možnost remapování pinů. Pro zjednodušení zapojení jsou piny `PA12` a `PA11` přemapované

[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 20[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 18: STM32G030Jx SO8N Pinout [8]

Jelikož je pouzdro malé, tak se na jednom fyzickém pinu nachází více periferií. Obrázek 18 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.

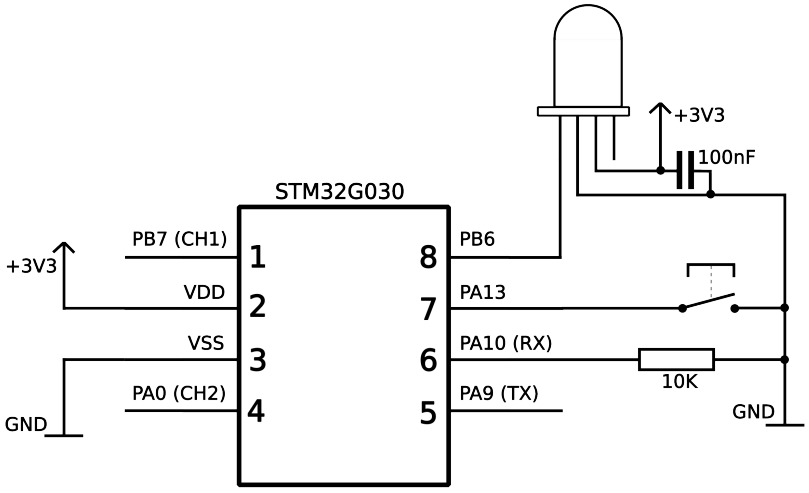
Address[1]	Corresponding option register (section)	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x1FFF7800	FLASH_OPTR (3.7.8)	Res.		IRHEN	NRST_MODE		nBOOT0	nBOOT1	nBOOT_SEL	Res.	RAM_PARITY_CHECK	DUAL_BANK	nSWAP_BANK	WWDG_SW	IWDG_STBY	IWDG_STOP	IWDG_SW	nRST_SHDW	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 19: 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* [24]. Komponenta Neopixel RGB LED byla použita jako externí knihovna [25].

[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 20. 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 20: Schéma zapojení STM32G030 v pouzdře SOP8

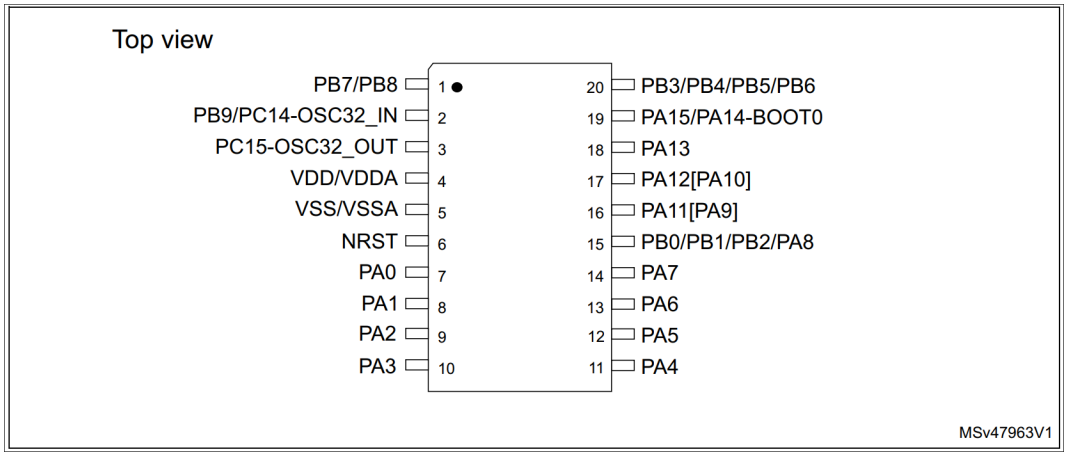
První z pinů k užívání je pin `PB7`. Tento pin slouží jako kanál AD převodníku pro měření napětí a pro měření odporu, pin je také využit pro hodinový signál pro posuvný registr. Na pinu `PA0` se nachází AD převodníkový kanál. Pin také disponuje kanály TIM2 časovače. Pin je použit jako druhý kanál AD 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.

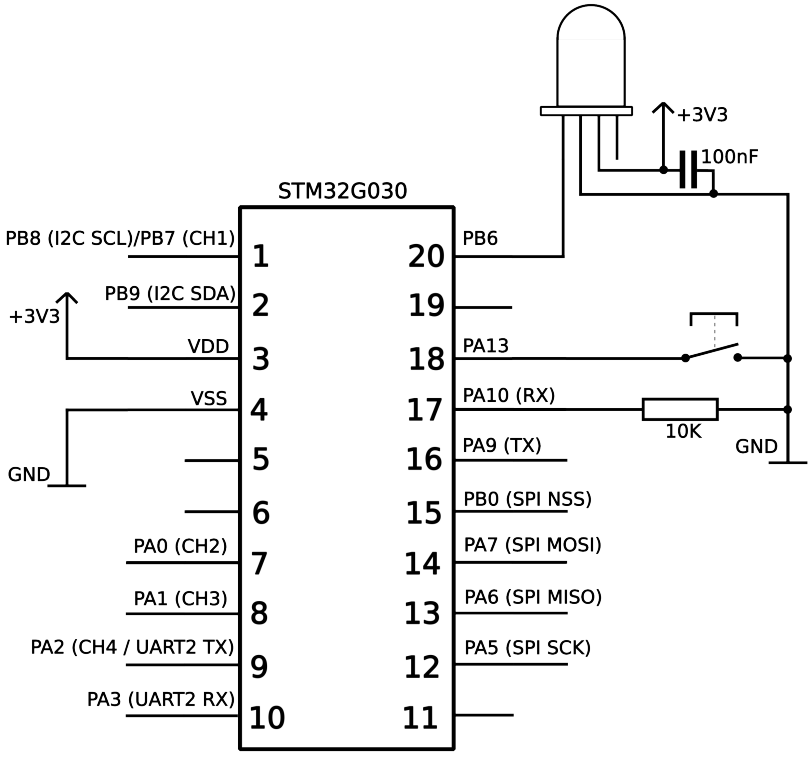
Pro zjednodušení sestavení sondy, je HW TSSOP20 návrh co nejvíce podobný návrhu SOP8. Piny `PA11` a `PA12` jsou přemapovány na `PA9` a `PA10`. Na pin `PA10` je připojen rezistor o velikosti 10 $K\Omega$ vůči zemi pro detekci komunikace s PC. Ze stejného důvodu byl zachován pin `PB6` jako výstup pro WS2812D a `PA13` pro tlačítko pro lokální režim. Obrázek 22

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



Obrázek 21: 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 17, který má na výstopu $3.3\ V$.



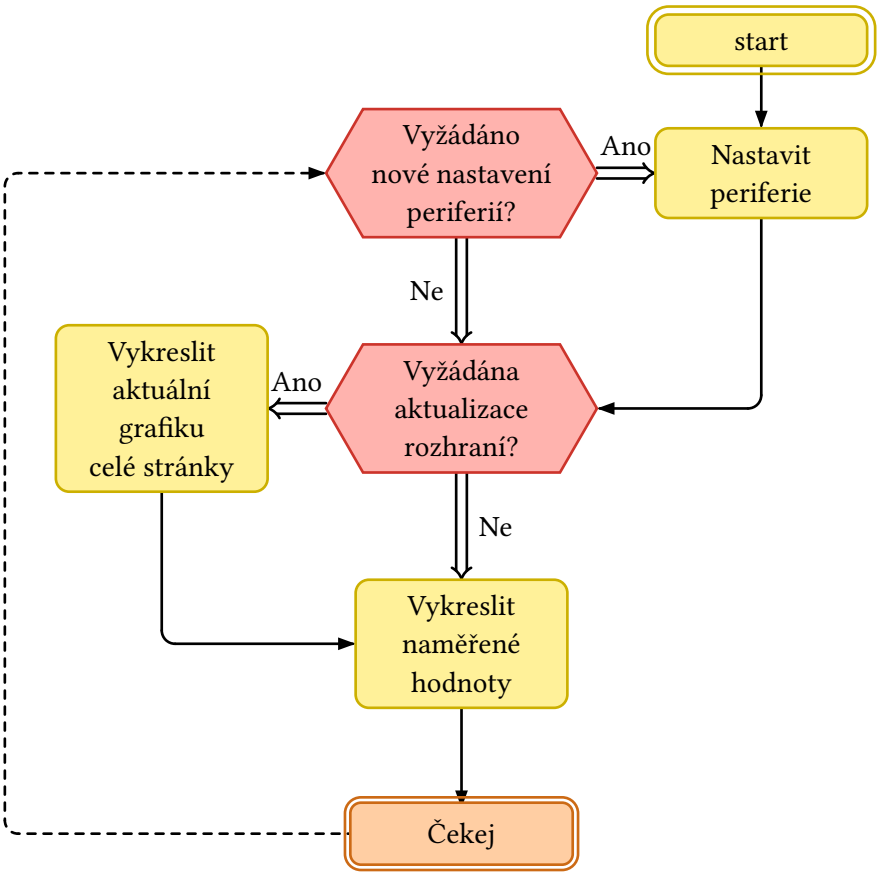
Obrázek 22: Schéma zapojení STM32G030 v pouzdře TSSOP20

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 7) 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 (**TODO: kapitola periferii**), 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 23: 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 [26].

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 [27].

```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 [27]. .

```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 8, 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

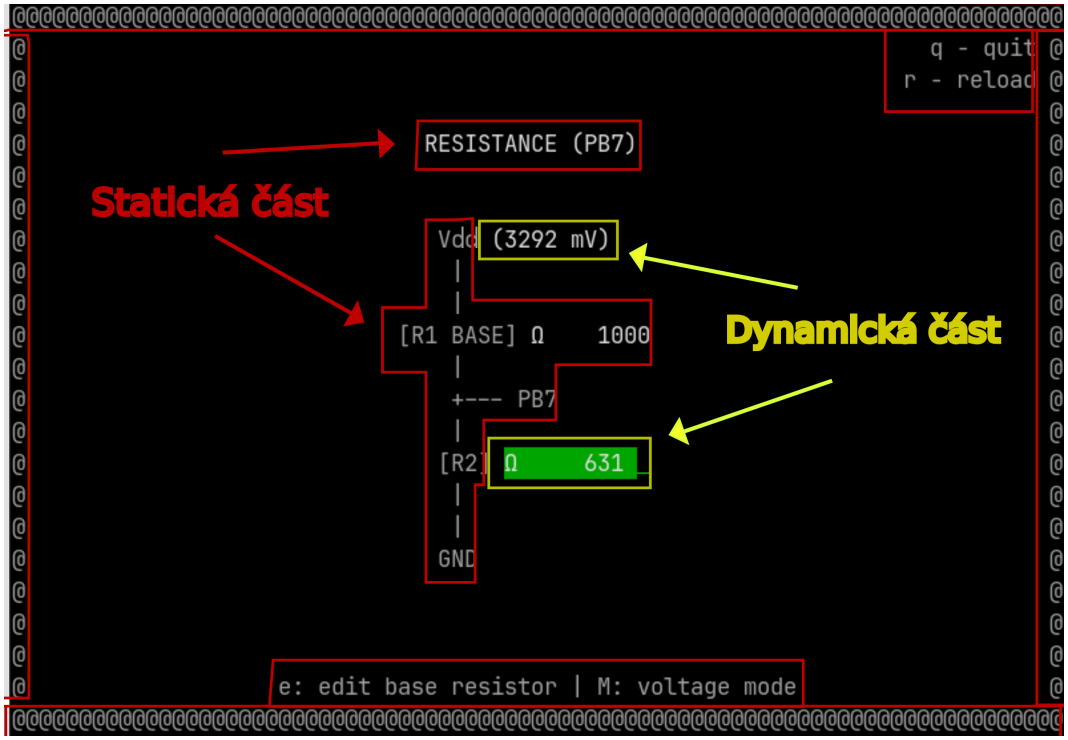
ním symbolu `R`, který je užitečný v případě, kdy se například vlivem špatného kontaktu vodiče mohou generovat náhodné symboly. Pokud by statická část vykreslovala periodicky, může nastat ke zbytečnému odesílání velkého množství dat skrze UART a to může spomalovat vykreslování. Také může dojít k rychlému 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 24 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 7), 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 24: Ukázka vykreslování statické a dynamické části stránky

úryvek kódu 9. Způsob převedení symbolu na akci na dané stránce ukazuje příklad ovládání hlavního menu v úryvku 10. 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

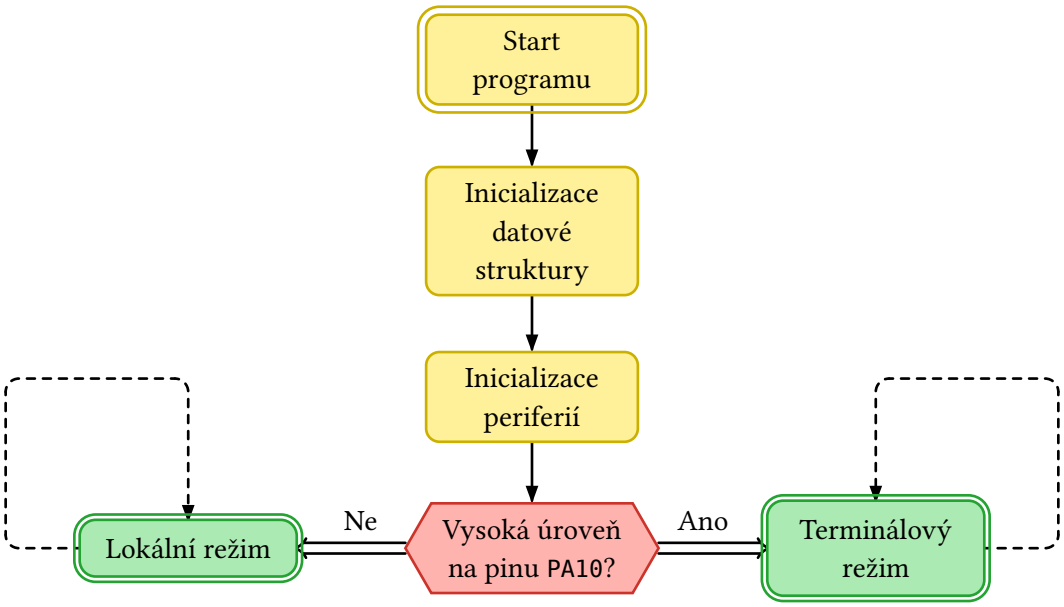
Kapitola 5

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 7, dojde k inicializaci všech potřebných periferií, které dále budou rozebrány v **TODO: add kapitolu**.

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 20 a Obrázek 22 má v zapojení, na pinu `PA10`, rezistor o velikosti 10 $K\Omega$, který při nepřipojeném vodiči uzemní `Rx`. Obrázek 25 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 25: 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 bude popsán v **TODO: dodat kapitolu**.

Při zmáčknutí tlačítka dojde k přerušení a je zavolána funkce z úryvku kódu 5, kde je zaznamenán čas zmáčknutí. Po uvolnění tlačítka dojde k přerušení náběžné hrany, a je zavolána funkce z úryvku kódu 6, kde je zaznamenán čas uvolnění a následně funkce `extern_button_check_press`, z úryvku kódu 11, 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. Stavy logické sondy jsou celkově čtyři.

5.3 Možné stavy lokálního režimu

TODO: dopsat tuto kapitolu

5.3.1 Stav 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. 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

```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 5: 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 6: Přerušení zavolané při uvolnění tlačítka

zeleně, v případě nízké úrovně se rozsvítí červená a pokud je napětí v neurčité oblasti, LED nesvítí. Tlačítkem poté lze přepínat mezi jednotlivými kanály.

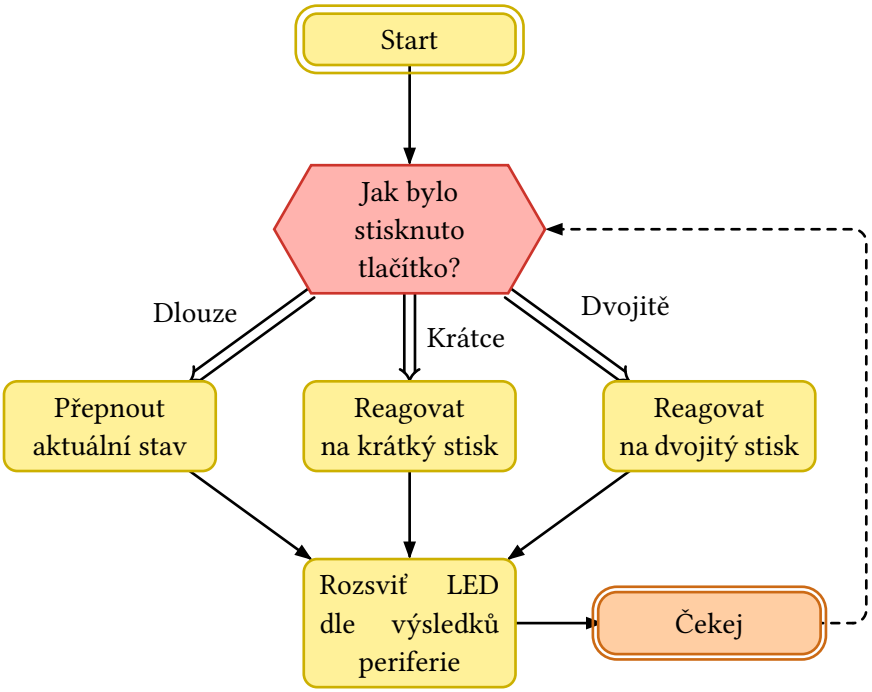
Stav logických úrovní

Další stav, který se po dlouhém stisku nastaví je **nastavování logických úrovní**. Stav při 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 všech kanálech, který má řadič v návrhu k dispozici.

5.3.2 Stav detekce pulzů

Detekování pulzů probíhá za pomocí input capture kanálu časovače. Při detekci hrany, je stav časovače uložen do registru a je vyvoláno přerušení. Přerušení poté nastaví pomocný flag, který bude zpracován při dalším cyklu smyčky. Smyčka poté na 1 sekundu rozsvítí LED jako detekci náběhové resp. sestupné hrany.

Lokální mód běží ve smyčce, kde se periodicky kontrolují změny a uživatelské vstupy. Důvod pro zvolení této metody je ten, že takto je zaručeno, že se vždy splní úkony ve správném pořadí. V **TODO: neco neco** je vysvětlen důvod podrobněji. Při začátku každého cyklu proběhne kontrola, zda uživatel dlouze podržel tlačítko. Pokud ano, přepne se stav. Poté program zkontroluje, zda bylo tlačítko zmáčknuto krátkou dobu, pokud ano, reaguje na tento úkon uživatele v závislosti na aktuálním stavu, stejně jako u dvojstisku. Je důležité podotknout, že stav tlačítka je vždy pouze jeden a nikdy se tlačítko nenachází ve více stavech zároveň. Následně po kontrole vstupní periferie proběhne kontrola hodnot a flagů aby smyčka zobrazila výstupní periferií informaci uživateli. Např. pokud je stav nastavení pulzů a flag, který symbolizuje nalezenou hranu, rozsvítí smyčka LED příslušné barvy. Po dokončení úkonů smyčka čeká určitou dobu, než zopakuje celý cyklus znovu. Doba se mění v závislosti na zvoleném stavu, tzn. detekce pulzů probíhá rychleji, než nastavování logických úrovní.


Start
Jak bylo stisknuto tlačítko?
Dlouze
Krátce
Dvojitě
Přepnout aktuální stav
Reagovat na krátký stisk
Reagovat na dvojitý stisk
Rozsviť LED dle výsledků periferie
Čekej

Kapitola 6

Realizace logické sondy

TODO: REVIZE

6.1 Měření napětí a zjišťování logické úrovně

Pro měření napětí je využíván AD převodník. Jak již bylo uvedeno v Kapitola 2.3.1.1, převodník po realizaci měření vrátí digitální hodnotu, kterou je potřeba převést na napětí. Pro získání přesného napětí je žádoucí změřit a vypočítat referenční napětí a nepoužívat odhad. V projektech, které se nezakládají na přesnosti je časté používat hodnotu `3.3 V`, což je idealizované napájecí napětí. Ve skutečnosti, je ale běžné, že napětí kolísá, popř. napětí může poklesnout, pokud je mikrořadič vytížen. Pokud je nutné, aby napětí bylo, co nepřesnější, musí být vypočteno realné referenční napětí.

Podstatné je, aby byl inicializován AD převodník. Pomocí HALu je to možné následovně:

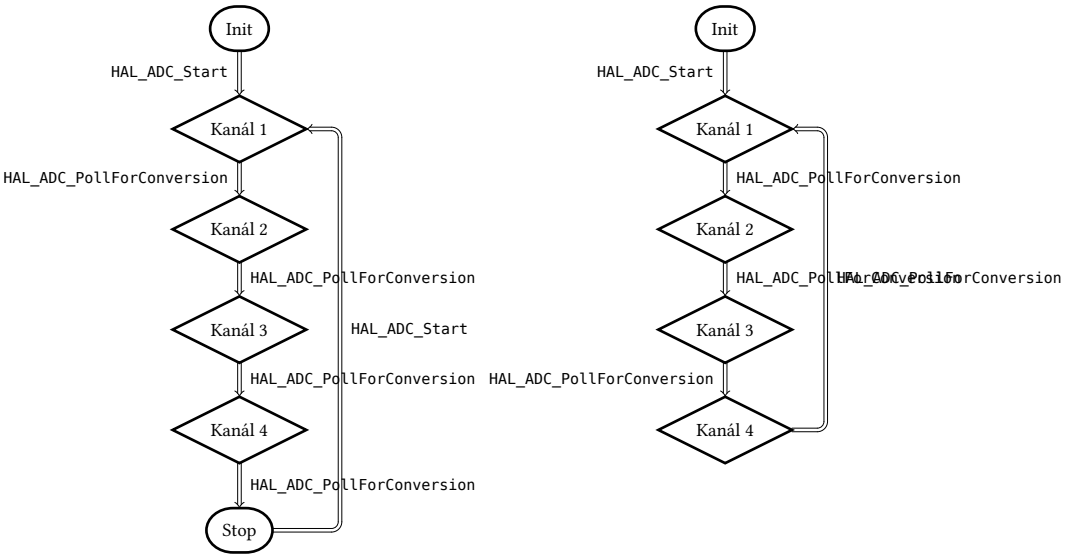
```c
static void MX_ADC1_Init(void) {
    ADC_ChannelConfTypeDef sConfig = {0};

    hadc1.Instance = ADC1;
    hadc1.Init.ClockPrescaler = ADC_CLOCK_ASYNC_DIV1; // předdělička
    hadc1.Init.Resolution = ADC_RESOLUTION_12B; // rozlišení 12 bitů
    hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
    hadc1.Init.ScanConvMode = ADC_SCAN_DISABLE;
    hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
    hadc1.Init.LowPowerAutoWait = DISABLE;
    hadc1.Init.LowPowerAutoPowerOff = DISABLE;
    hadc1.Init.ContinuousConvMode = ENABLE; // pokračuje i po konci cyklu
    hadc1.Init.NbrOfConversion = 1; // počet kanálů
    hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
    hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
    hadc1.Init.DMAContinuousRequests = ENABLE;
    hadc1.Init.Overrun = ADC_OVR_DATA_PRESERVED;
    hadc1.Init.SamplingTimeCommon1 = ADC_SAMPLETIME_160CYCLES_5;
    hadc1.Init.SamplingTimeCommon2 = ADC_SAMPLETIME_1CYCLE_5;
    hadc1.Init.OversamplingMode = DISABLE;
    hadc1.Init.TriggerFrequencyMode = ADC_TRIGGER_FREQ_HIGH;
    if (HAL_ADC_Init(&hadc1) != HAL_OK) {
        Error_Handler();
    }
```

```
25      sConfig.Channel = ADC_CHANNEL_1;
26      sConfig.Rank = ADC_REGULAR_RANK_1;
27      sConfig.SamplingTime = ADC_SAMPLINGTIME_COMMON_1;
28      if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK) {
29          Error_Handler();
30      }
31      adc1_ch = create_adc_channels(&hadc1);
32      realloc_v_measures(adc1_ch, &v_measures);
33      setup_adc_channels(&hadc1, adc1_ch, true);
34  }
```

kde je spousta nastavení, které pro tento projekt nejsou úplně podstatné nicméně některá konfigurace je stěžejní.

Parametr `hadc1.Init.NbrOfConversion` určuje, kolik měření bude provedeno, než se AD převodník zastaví. Měření může probíhat na více kanálech (V tomto případě až na 4 kanálech) ale jeden převodník nemůže měřit všechny kanály najednou. Je nutné určit pořadí v jakém bude měřit.



V levém diagramu je možno vidět, jakým způsobem funguje převodník, pokud **není** `hadc1.Init.ContinuousConvMode` nastaven na `ENABLE`. Při každém zahájení měření kanálu pomocí `HAL_ADC_PollForConversion`, převodník udělá `x` vzorků během 160,5 cyklů[28]. Po dokončení vzorkování lze zjistit hodnotu funkcí `HAL_ADC_GetValue`. Jakmile AD převodník dokončí konverzi, kanál se nastaví na 2. Při zavolání konverze znovu se již měří na druhém kanálu. Takto sekvenčně převodník pokračuje dokud nedojde k poslednímu kanálu. Pokud už další kanál nenásleduje, ad převodník zastaví svou činnost a potřeba ho znovu zapnout aby pokračoval.

[28]Tato hodnota je v STM32G0 nejvyšší a zaručuje nám tu největší možnou přesnost. Některé mikrořadiče nabízí i vyšší počet odběrů.

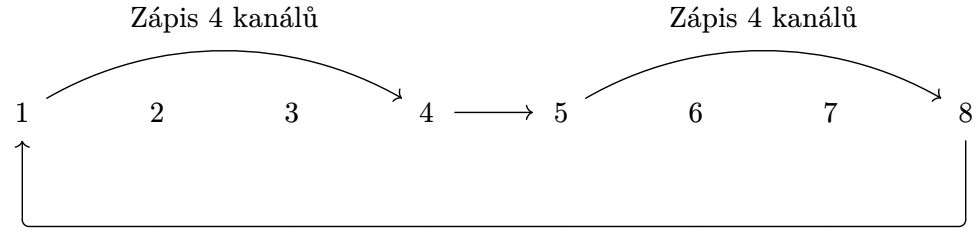
V pravém diagramu již kontinuální mód zapnutý a měření probíhá neustále dokola. Pokud AD převodník dojde k poslednímu kanálu, začne měřit opět první. K zastavení dojde pouze v případě, že je zavolána funkce `HAL_ADC_Stop`.

Logická sonda pracuje ve smyčce, kdy jednou za určitý krátký úsek zastaví klasické měření kanálů a změří referenční napětí. Tento způsob zajišťuje neustále validní referenci.

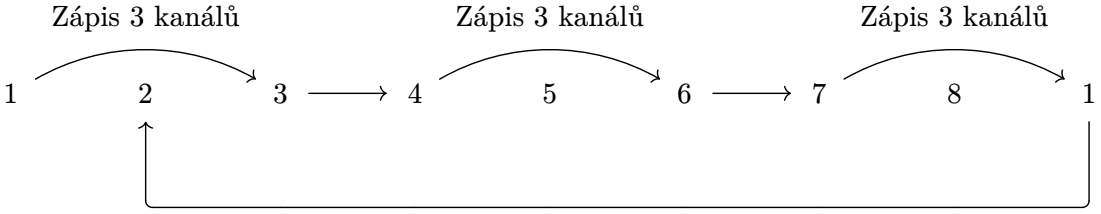
Během měření kanálů je nutné, aby nebyl zbytečně zatěžován procesor. Procesor v hlavní smyčce pracuje na vykreslování dat do terminálu a pokud by prováděl měření, mohlo by to ovlivnit rychlost měření napětí a nemuselo by být dosaženo dostatečnému sebrání vzorků pro plovoucí průměr. Toto je vyřešeno pomocí DMA[29]. HAL aktivuje DMA pro AD převodník pomocí `hadc1.Init.DMAContinuousRequests` nastavené na `ENABLE`. Parametry DMA lze nastavit pomocí STM32CubeMX. DMA je nastaveno jako cirkularní buffer, do kterého se zapisují hodnoty z AD převodníku. Procesor k těmto datům přistoupí, až bude potřebovat.

Logická sonda umí počet kanálů nastavovat dynamicky, tzn. může měřit pouze 1 kanál a nebo během běhu programu zapnout další dva. Možnost změny počtu kanálů během chodu je problém, jelikož pokud pro DMA je alokována pamět na zásobníku, velikost musí být ideálně statická.

Mějme alokovanou paměť na zásobníku, která bude 8 integerů, tzn. 2 integery pro každý kanál. Pokud jsou zapnuty 1, 2 nebo 4 kanály nenastává žádný problém, protože nakonci se začnou hodnoty ukládat opět od začátku. Takže při opakovaném zápisu nedojde k „rozjetí“ indexů. A jednodušše můžeme říct, že na indexu 6, je vždy kanál 2.



Pokud ale budeme mít aktivovány 3 kanály dojde k problému. Už při první iteraci se nám indexy posunou, což je nežádoucí.



Existuje možnost toto vyřešit pomocí nalezení nejmenšího společného násobku. Pokud, ale hledáme násobek pro 100, 200, 300, 400, tak dojdeme k tomu, že je to naprosté plýtvání pamětí, které si na STM32G0 nemůžeme dovolit.

Tento problém byl vyřešen tak, že paměť je alokována na haldě, takže je dynamicky vytvořená, a realokuje se při změně, tak aby stále odpovídal počet vzorků a nedocházelo k posunu.

[29]DMA je metoda, kdy periferie umí přímo zapisovat nebo číst z paměti. Vyhoda je, že procesor nemusí zasahovat a šetří se zdroje, které můžou být využity jinde.

Po určitém časovém úseku, procesor zpracuje data z DMA a zprůměruje hodnoty z AD převodníku, následně hodnoty převede dle metodity v Kapitola 2.3.1.1 a vykreslí na seriovou linku pomocí ANSI sekvencí zmíněné v

TUI vykresluje hodnoty na každém kanálu a poté vykresluje, zda je logická úroveň vysoká, nízká a nebo je nejasná ukazuje vizuál stránky pro měření. Je možné pozorovat, že kanál 1 na pinu `A0` měří `0,0 V` a `L` znázorňuje nízkou úroveň. Kanál 2 ukazuje napětí `3,3V` a je to vysoká úroveň. Kanál 3 je plovoucí a není připojený. Proto úroveň je nejasná a měří pouze parazitní napětí. Kanál 4 je vypnutý.

Kanály je možné zapínat a vypínat pomocí stránky `Channels` ukazuje vzhled této stránky. Uživatel pomocí klávesových zkratek 1 až 4 volí jaké kanaly aktivovat, s tím, že po zvolení kanálů je nutné nastavení uložit stisknutím klávesy S.

Všechny data ohledně kanálů AD převodníku jsou uloženy ve struktuře zvané `adc_channels`. Tato struktura drží, jaké kanály jsou aktivovány, jaké kanály jsou označeny uživatelem, ale ještě nebyly uloženy a tím pádem aplikovány, jaká byla poslední průměrná hodnota měření, jaká čísla pinů kanály osidlují a nakonec instance `ADC_HandleTypeDef`, což je HAL struktura, která je abstrakce ovládání převodníku, ukládání konfigurací apod.

```c
typedef struct {
    _Bool channel[NUM_CHANNELS]; // aktivované kanály
    _Bool channel_unapplied[NUM_CHANNELS]; // kanály neaktivované
    _Bool applied; // bylo nastavení uživatele aplikováno?
    uint32_t avg_last_measure[NUM_CHANNELS]; // poslední průměr hodnot
    unsigned int pin[NUM_CHANNELS]; // čísla pinů
    unsigned int count_active; // počet aktivních pinů
    ADC_HandleTypeDef* hadc; // instance adc pro danou strukturu
} adc_channels;
```

6.2 Odchytání pulzů a frekvence

Pro měření frekvence hraje stěžejní roli časovač. Jak bylo zmíněno v Kapitola 2.3.1.2, časovače umí tzv. input capture. Input capture poskytuje možnost měření časových parametrů vstupního signálu, jako například perioda nebo šířka signálu. Časovač inkrementuje hodnotu o dané frekvenci a v momentě kdy na vstupu je objeví náběžná nebo sestupná hrana, dojde k přerušení a aktuální hodnota čítače se uloží do speciálního registru [11].

Pro přesné zjistění frekvence musí být kladen důraz na režii. Při odběru dat by nesměl procesor provádět jakoukoliv. Toto se dá realizovat pomocí DMA podobně jako v Kapitola 6.1.

K zjištění frekvence je použita tzv. metoda hradlování. Metoda hradlování využívá periodu vzorkování, která je využita pro spočítání finální frekvence. Rovnice 15 uvádí způsob, jak spočítat frekvenci pomocí metody hradlování. Frekvence lze spočítat jako

polovinu napočítaných pulzů za časový úsek[30]. K tomuto účelu je použit 32 bitový časovač, aby bylo možné měřit, co největší frekvence.

$$F = \frac{\frac{N}{2}}{T} \tag{15}$$

Pro měření časového úseku je využit druhý časovač. Tento časovač je nastaven předděličkou tak, aby hodnotu inkrementoval za 1 ms. Uživatel poté pomocí TUI nastavuje periodu na požadovaný úsek.

```c
signal_detector.frequency =
    (signal_detector.pulse_count / 2) /
    (signal_detector.sample_times[signal_detector.sample_time_index] /
    1000);
```

Po spuštění časovač začne inkrementovat hodnotu a v momentě, kdy časovač přeteče tzn. dosáhne poslední hodnoty periody, časovač vyvolá přerušení. Při přerušení se zastaví časovač pro čítání pulzů a spočítají se potřebné hodnoty.

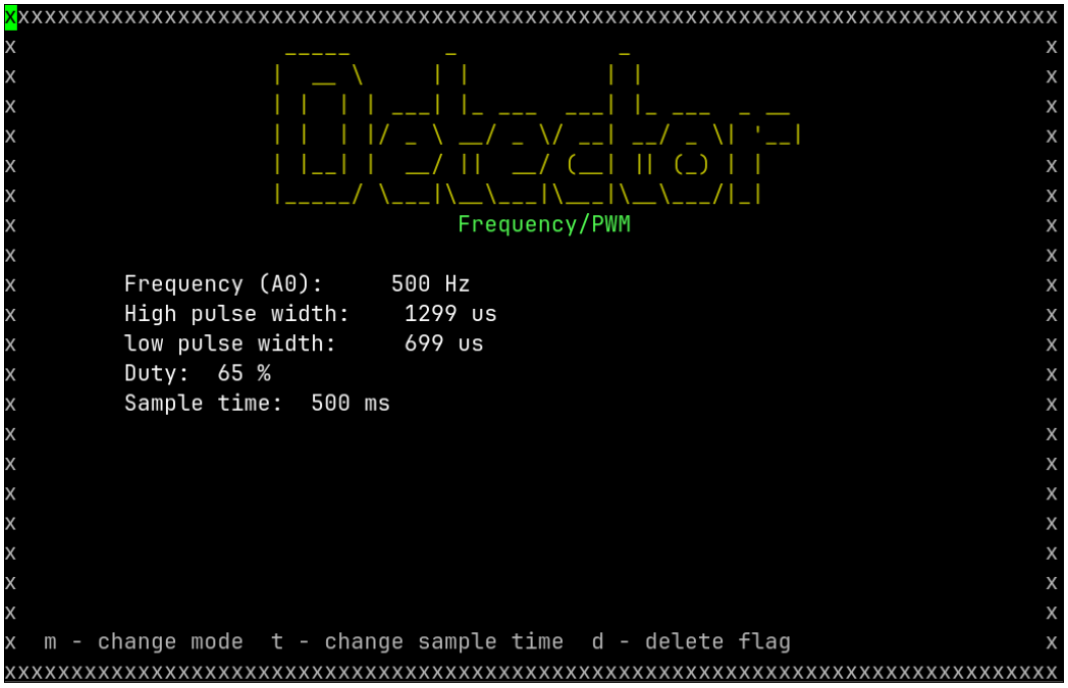
Logická sonda dle Rovnice 15 vypočítá frekvenci. Důvod, proč logická sonda počítá oba pulzy a ne pouze nástupnou nebo sestupnou hranu je ten, že při detekci obou hran dokáž sonda spočítat, jak široké pulzy jsou. Tuto skutečnost je možné využít například pro počítání Duty u PWM signálů.

Níže je možné vidět logiku, která počítá šířku pulzu pro vysokou úroveň. Pokud je čas sestupu signálu větší než vzestupu, není potřeba nic přepočítávat. Pokud čas sestupu je nižší než čas vzestupu, znamená to, že časovač přetekl a je nutné od `0xFFFFFFFF` odečíst čas vzestupu. Nakonec sonda přepočítá hodnotu časovače na čas, tzn. vynásobí konstantou, aby byl vzat potaz na frekvenci procesoru.

```c
uint32_t rise_pulse_ticks;
if (sig_high_end > sig_high_start) {
    rise_pulse_ticks = sig_high_end - sig_high_start;
} else if (sig_high_end < sig_high_start) {
    rise_pulse_ticks = (0xFFFFFFFF - sig_high_start) + sig_high_end;
} else {
    rise_pulse_ticks = 0;
}
float high_width = (rise_pulse_ticks * CONST_FREQ) / 1000;
```

Duty time je poté spočítaný jako poměr času vysokého signálu a nízkého signálu uvedený v procentech.

[30] Z důvodu, že časovač v tomto případě měří náběžný i sestupný pulz, je vždy počet pulzů v periodě signálu dvojnásobný, proto je zapotřebí počítat pouze s polovinou.



Obrázek 26: Stránka pro měření frekvence a PWM

Kromě frekvence, umí sonda odchytávat pulzy, jak nízké, tak vysoké. Mezi módy je možné přepínat klávesou. Pro zachytávání signálu je časovač opět nastaven v režimu input capture. Časovač je spuštěn a nyní není využito DMA, ale časovač vyvolává přerušení, pokud dojde k zachycení signálu, toto přerušení zkontroluje, zda hrana signálu je nástupná nebo sestupná. Poté rozhodne podle módu uživatele, kterou hranu má ignorovat a kterou má detekovat.

Když časovač odchytí správnou hranu, přerušení nastaví flag, který reprezentuje nalezený pulz. Tento pulz se poté promítne uživateli do TUI. Flag je možné smazat a čekat na další vyvolání přerušení. Do budoucna je plán, přidat automatické smazání flagu po určité době, aby této funkcionality mohlo být využíváno i bez nutnosti připojení na seriovou linku. Po nalezení pulzu se rozsvítí led, která po například 1 sekundě opět zhasne.

Obrázek 27: Stránka pro hledání pulzů

6.3 Generování pulzů

Při generování pulzu má uživatel možnost nastavit konkrétní délku pulzi prostřednictvím TUI. Nastavená délka určuje, jak dlouho bude pulz trvat. Po nastavení délky může uživatel stisknout příslušnou klávesu, která spustí proces generování pulzu.

Časovač v tomto případě má nastaven prescaler tak, aby se jednodušše počítal čas v periodě. Poté, co je nastavený časovač, spustí se. Po přetečení časovače se vyvolá přerušení, které značí, že časovač odměřil příslušnou dobu. Přerušení následně časovač zastaví.

Sonda umožňuje generovat pulzy dvou typů. První možností je generace nízké úrovně signálu během trvání vysoké úrovně. Druhou možností je naopak generace vysoké úrovně signálu během nízké úrovně. Uživatel si tedy může zvolit, kterou úroveň chce jako výchozí stav.

Díky těmto režimům je možné i nastavit úrovně, které uživatel nemusí nutně používat jako generování pulzů, ale jako přepínání úrovní dle potřeby. **TODO: ještě se na to kouknout znovu a upravit to neaktualni**

```c
static void MX_TIM16_Init(void) {
    htim16.Instance = TIM16;
    htim16.Init.Prescaler = 63999; // předdělička
    htim16.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim16.Init.Period = 999; // výchozí hodnota periody
    htim16.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    htim16.Init.RepetitionCounter = 0;
    htim16.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
    if (HAL_TIM_Base_Init(&htim16) != HAL_OK) {
        Error_Handler();
    }
    HAL_TIM_Base_Start_IT(&htim16);
    sig_gen_init(&signal_generator);
}
```

Obrázek 28 ukazuje, jak vypadá prozatimní zhotovení rozhraní pro ovládání generátoru pulzů. Pomocí kláves uvedené v nápovědě je možné měnit šířku pulzu a nebo měnit, jaký mód použít.

```c
void sig_gen_toggle_pulse(sig_gen_t* generator, const _Bool con) {
    generator->start = false;

    // nastavení periody, která byla nastavena uživatelem
    __HAL_TIM_SET_AUTORELOAD(&htim16, generator->period - 1);

```

Obrázek 28: Stránka pro generování signálu

```
7       if (con && generator->con) {
8           HAL_TIM_Base_Stop_IT(&htim16);
9           generator->con = false;
10      } else {
11          generator->con = con;
12          HAL_TIM_Base_Start_IT(&htim16);
13      }
14  }
```

Kapitola 7

Závěr a zhodnocení

Příloha A

Citace

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

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

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

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

[5] W. Commons, „File:SPI timing diagram msbfirst.svg — Wikimedia Commons, the free media repository". [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". [Online]. Dostupné z: https://www.tme.eu/Document/bd6b355a8705d46515a8ada0d153187b/WS2812D-F5-L.pdf

[7] doc. Ing. Jan Fischer, CSc., „ETC22 Přednáška 2". [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". [Online]. Dostupné z: https://www.farnell.com/datasheets/2882477.pdf

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

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

[11] M. Dudka, „STM32 Timery". [Online]. Dostupné z: http://www.elektromys.eu/clanky/stm_timer1/clanek.html

[12] STMicroelectronics, „STM32CubeMX". [Online]. Dostupné z: https://www.st.com/content/st_com/en/stm32cubemx.html

[13] M. Hasan, „Understanding STM32 HAL Library Fundamentals". [Online]. Dostupné z: https://embeddedthere.com/understanding--stm32-hal-library-fundamentals/

[14] A. Limited, „What is CMSIS?". [Online]. Dostupné z: https://www.keil.arm.com/cmsis

[15] H. Nazeran, „Reducing Power Line Interference in Digitised Electromyogram Recordings by Spectrum Interpolation", *Medical and Biological Engineering and Computing*, 2004.

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

[17] R. Nave, [Online]. Dostupné z: http://hyperphysics.phy-astr.gsu.edu/hbase/electric/voldiv.html

[18] G. A. Center, [Online]. Dostupné z: https://germanna.edu/sites/default/files/2022-03/Ohms%20and%20Kirchoffs%20Laws.pdf

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

[20] U. Wisconsin–Madison, „Uart Basics". [Online]. Dostupné z: https://ece353.engr.wisc.edu/serial-interfaces/uart-basics/

[21] V. H. Adams, „Universal Asynchronous Receiver Transmitter". [Online]. Dostupné z: https://vanhunteradams.com/Protocols/UART/UART.html

[22] PIYU DHAKER ANALOG DEVICES, „INTRODUCTION TO SPI INTERFACE". [Online]. Dostupné z: https://www.analog.com/EN/RESOURCES/ANALOG-DIALOGUE/ARTICLES/INTRODUCTION-TO-SPI--INTERFACE.HTML

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

[24] Autodesk, „Eagle". [Online]. Dostupné z: https://www.autodesk.com/products/eagle/overview

[25] Albert van Dalen, „addressable LED rings NeoPixel WS2812B-2020". [Online]. Dostupné

z: https://github.com/avandalen/Eagle-
-library-addressable-LED-rings
[26] [Online]. Dostupné z: https://enxstandards.
web.illinois.edu/standard/ansi-x3-4/
[27] fnky, „ANSI Escape Sequences". [Online].
Dostupné z: https://gist.github.com/fnky/
458719343aabd01cfb17a3a4f7296797

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 7: 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 8: 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 9: 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 10: Ovládání menu

```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 11: Struktura globálních promněných