

**Bakalářská práce**



**České  
vysoké  
učení technické  
v Praze**

**F3**

**Fakulta elektrotechnická  
Katedra měření**

## **Multifunkční diagnostická logická sonda**

**Milan Jiříček**

**Vedoucí práce: doc. Ing. Jan Fischer, CSc.  
Studijní program: Otevřená informatika  
April 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 kombinuje 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 režimu 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 periférií 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.

## 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.

## 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, 17. 04. 2025

**9 TODOs remaining**

## Obsah

1 Úvod .....	1
2 Rozbor problematiky .....	2
2.1 Technické požadavky .....	2
2.2 Využití ve výuce .....	2
2.3 Volba mikrokontrolerů .....	3
2.3.1 STM32G030 .....	3
2.3.2 STM HAL .....	5
2.4 Měření veličin digitálního obvodu .....	6
2.5 Analýza komunikačních rozhraní .....	6
2.5.1 UART .....	7
2.5.2 I2C .....	7
2.5.3 SPI .....	9
2.5.4 Neopixel .....	9
2.6 Grafické řešení .....	10
2.6.1 Ansi sekvence .....	10
3 HW návrh logické sondy STM32 .....	12
3.1 SOP8 .....	12
3.2 TSSOP20 .....	14
4 SW návrh logické sondy STM32 .....	15
4.1 Logika nastavení módů .....	15
4.2 Lokální mód .....	15
4.3 Terminálový mód .....	17
5 Realizace logické sondy .....	19
5.1 Grafické rozhraní .....	19
5.1.1 Odesílání zpráv .....	19
5.1.2 Přijímání zpráv .....	23
5.2 Měření napětí a zjišťování logické úrovně .....	24
5.3 Odchytání pulzů a frekvence .....	28
5.4 Generování pulzů .....	31
6 Závěr a zhodnocení .....	33
Citace .....	34



## 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 reálném 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 náležitostí 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 Technické požadavky

V rámci bakalářské práce bude navržena a realizována multifunkční diagnostická logická sonda na platformě STM32. V návrhu sondy je potřeba zohlednit následující klíčové oblasti: jednoduchost ovládání i uživateli, které 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<sup>1</sup>.

Firmware a hardware bude vyvinut pro STM32G030 v pouzdře SOP8 a TSSOP20, které svou nabídkou periférií jsou vhodné pro jednoduché sestavení. Také bude vyvinuta v omezené míře na Raspberry Pi Pico. Sonda je vytvořena za účelem použití při výuce. Hlavní využití Sonda bude vybavena tzn. „lokálním režimem“ a „terminálovým režimem“.

Lokální režim bude sloužit pro rychlou základní analýzu obvodů s indikací pomocí WS2812 RGB LED a ovládání skrze jedno tlačítko. Bude fungovat bez nutnosti připojení zařízení k PC skrze USB nebo USB převodník. Tlačítkem bude uživatel přepínat módy, kanály a úrovně. Lokální režim bude mít následující vlastnosti: nastavení úrovně kanálů, odchyťování pulsů, prověření logické úrovně a generace pravidelných pulsů.

Terminálový režim bude poskytovat konkrétní měření veličin digitálního obvodu a testování sběrnic. Logická sonda bude v tomto režimu ovládána UART pomocí převodníku UART/USB. Sonda takto poskytne uživatelské rozhraní, které se vygeneruje na straně mikrokontroleru a zobrazí skrze terminálovou aplikaci podporující tzn. ANSI sekvence<sup>2</sup>.

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.

### 2.2 Využití ve výuce

Logická sonda je elektronické zařízení sloužící k diagnostice a analýze digitálních obvodů. Pomáhá určovat logické úrovně, detekovat pulsy, měřit frekvenci a další. Je to jeden ze standardních nástrojů pro elektrotechniky pracující s FPGA, mikrokontrolery či logickými obvody. Výhoda logické sondy je cena pořízení a flexibilitou použití. Logická sonda je jedním z prvních nástrojů, který dokáže najít základní problém v digitálním obvodu.

Další ze základních nástrojů ve výuce je multimetr. Multimetr kombinuje několik funkcí jako např. voltmetr, ampermetr, ohmmetr. Tyto nástroje jsou podstatné pro určení základních veličin v obvodu, proto je tento nástroj nezbytný při sestavování obvodu ve

<sup>1</sup>Point of failure - Bodů selhání

<sup>2</sup>Např. PuTTY, GTKTerm...

výuce. Student například využije multimetr pro zjištění odporu rezistoru pokud nezná barevné kódování na obalu, pro zjištění kontaktování na nepájivém kontaktním poli či pro ověření správného napětí na napájení.

Logická sonda je sestrojena za účelem sloučení funkcí těchto standartních nástrojů, do jednoho multifunkčního zařízení, což zjednoduší studentovi celkovou analýzu. Pro nástroj používaný ve výuce je podstatné aby nástroj bylo pokud možno využívat i bez předchozích zkušeností. Student si může osvojit metodiku debugování od základních kontrol napájení až po analýzu časových signálů.

Pokud navrhne softwarové řešení zadaného problému, je žádoucí toto řešení během vývoje testovat pulzy. Jedná se například o testování zda je pulz vyslán po vodiči, zda je na výstupu správná frekvence nebo testování reakce na vyslaný impuls. Typické využití následujících nástrojů může být během tvorby čítače pulzů a zobrazování hodnoty na 7-segmentovém displeji. Sonda vzhledem k těmto vlastnostem může sloužit jako univerzální nástroj pro studentské projekty od jednoduchých čítačů po řídicí systémy.

Během sestavování hardwarového řešení je nutné vyloučit nefunkčnost hardwarových součástí obvodu. Potíže mohou činit například spálené RGB LED, nefunkční displeje, či senzory. Kromě vyloženého selhání hardwaru se vývojář může setkat s chybným zapojením součástky, která neodpovídá referenčního manuálu. Typicky: prohození vodičů na UART, zapomenutí pull up rezistorů na vodičích I2C apod. Zmíněné problémy mohou způsobit zdoluhavé hledání chyby. Pro zjištění problému je podstatné ověřit správnosti komunikace mezi částmi obvodu a tak vyloučit případný hardwarový problém.

Pořizování běžných analyzátorů může být velice nákladné, protože je těchto přístrojů je potřeba velký počet. Sestrojení sondy na nepájivém kontaktním poli poskytuje flexibilitu vyučujícím vytvořit rychle logickou sondu. Jelikož návrh bere zřetel na možnost realizace studentem, je při sestavování použito minimální počet externích součástek. Použití MCU typu STM32 a RP2040 umožňuje transparentní výuku hardwarového designu.

## ■ 2.3 Volba mikrokontrolerů

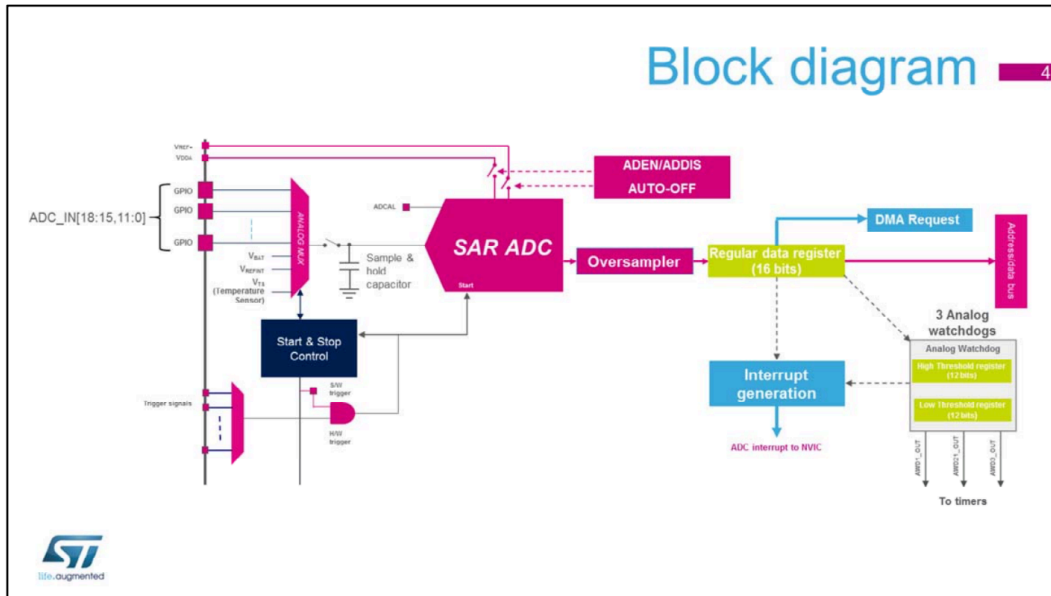
### ■ 2.3.1 STM32G030

**TODO: „doc. Fischer hovořil, že není vhodné leaknout problem s G031 a G030 mám tu psát o G030 nebo G031“** Pro návrh v této semestrální práci byl zvolen mikrořadič STM32G030 od firmy STMicroelectronics [1]. 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 32 KiB flash paměť a 8 KiB SRAM. [2]

Pro řadu G030 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. [3] Obě zmíněné pouzdra jsou použity pro implementaci logické sondy, o které pojednává Kapitola 5.

### ■ Analogo-digitální převodník

Mikrokontrolér STM32G030 je vybaven ADC, který obsahuje 8 analogových kanálů o rozlišení 12 bitů. Maximální vzorkovací frekvence převodníku je 2 MSPS. Při měření ka-



Obrázek 1: Blokový diagram AD převodníku

nálů se postupuje sekvenčně, která je určena pomocí tzv. ranků<sup>3</sup>. 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 tzn. oversampling<sup>4</sup>. Převodník obsahuje **accumulation data register**, který akumuluje měření a poté pomocí data shifteru vydělí počtem cyklu. [4]

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

AD převodník, po dokončení měření vzorků, vrací hodnotu, která není napětí. Tato hodnota je poměrná hodnota vůči napájecímu napětí vyjádřena 12 bitově<sup>5</sup>. 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<sup>6</sup> jako referenci, která může být 2 V až 3.6 V a také může kolísat vlivem napájení nebo zatížení. Pro výpočet  $V_{\text{REF}+}$  se používá interní referenční napětí  $V_{\text{REFINT}}$  kalibrační data uložená během výroby mikrořadiče a naměřené hodnoty z ADC. [2]

Vztah pro výpočet je následující:

$$V_{\text{REF}+} = \frac{V_{\text{REFINT\_CAL}} \times 3300}{V_{\text{REFINT\_ADC\_DATA}}} \quad (2)$$

kde:

- $V_{\text{REFINT\_CAL}}$  je kalibrační hodnota interního referenčního napětí, která je uložena ve flash paměti mikrořadiče během výroby. Tato hodnota představuje digitální hodnotu, kdy  $V_{\text{REF}+}$  je přesně 3.3 V. Hodnota se získává čtením z pevné adresy<sup>7</sup>. [2], [5]

<sup>3</sup>Rank určuje v jakém pořadí je kanál změřen.

<sup>4</sup>Proběhne více měření a následně jsou výsledky např. zprůměrovány aby byla zajištěna větší přesnost.

<sup>5</sup>Např. hodnota 4095 značí, že naměřené napětí je stejně velké jako napájecí napětí.

<sup>6</sup>VDDA je označení pro analogové napájecí napětí v mikrokontrolérech STM32.

- 3300 je konstanta odpovídající referenčnímu napětí při kalibraci ve výrobě vyjádřená v milivoltech.
- $V_{REFINT\_ADC\_DATA}$  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.

$$V_{CH} = \frac{V_{CH\_ADC\_DATA}}{2^{\text{rozlišení}} - 1} \times V_{REF+} \quad (3)$$

kde:

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

## ■ Časovače

STM32G030 obsahuje několik časovačů, které se dají využít pro logickou sondu. Mikrořadič má zabudovaných několik základních a jeden advanced timer. Základní timery 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 přerušování při dosažení specifické hodnoty časovače apod. [6]

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]

$$F_{TIMx} = \frac{F_{clk}}{\text{Prescaler} + 1} \quad (4)$$

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

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

## ■ 2.3.2 STM HAL

Hardware abstraction layer je knihovna poskytovaná společností STMicroelectronics pro jejich mikrořadiče řady STM32. Tato knihovna tvoří vrstvu abstrakce mezi aplikací

<sup>7</sup>Např. u STM32G0 je adresa kalibrační hodnoty: 0x1FFF75AA

<sup>8</sup>U 16 bitového časovače je maximální perioda 65535, zatímco u 32 bitového časovače je to 4294967295.

a periferiemi mikrokontroléru. Pokytuje funkce na vyšší úrovni, které usnadňují přístup např. k GPIO, USART, SPI, I2C bez nutnosti přímého přístupu k registrům procesoru. [7]

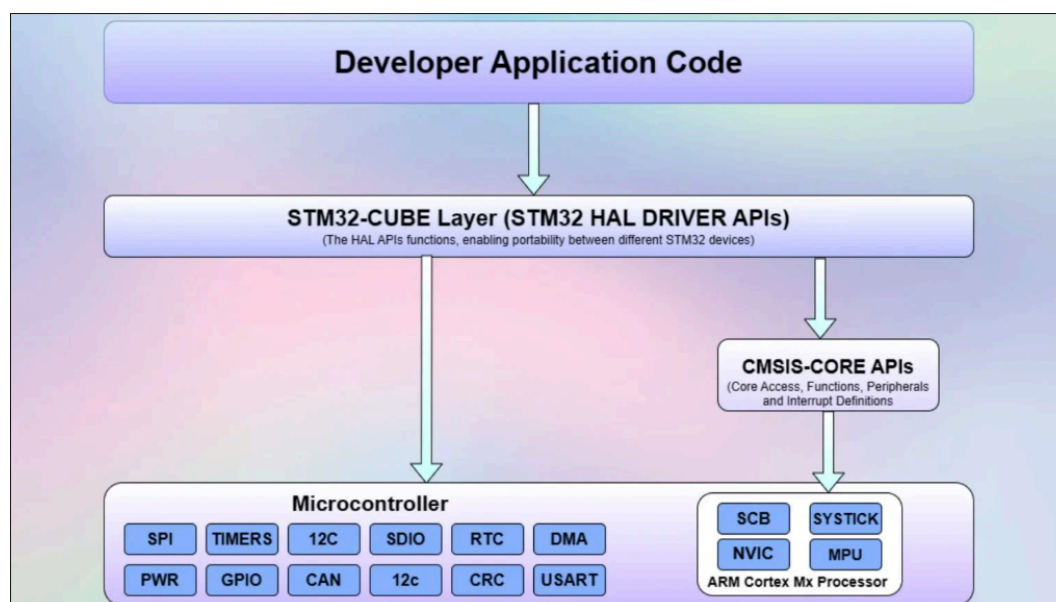
Mezi vlastnosti, kromě zmíněné jednoduchosti patří přenositelnost. Spousta mikrořadičů například využívají jiné adresy pro specifickou funkcionalitu. Pokud vývojář bude potřebovat portovat aplikaci na jiný mikrořadič, není nutné přepisovat různé adresy a logiku programu ale pouze změnit hardware a jelikož program pracuje s abstrakcí, bude nadále fungovat. Přímého přístupu k registrům procesoru. Na Obrázek 2 je znázorněn diagram, který znázorňuje architekturu HAL. [8]

Součástí HALu je tzv. CMSIS<sup>9</sup>, což je sada standardizovaných rozhraní, které umožňují konfiguraci periferií, správu procesorového jádra, obsluhu přerušení a další. [9] 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<sup>10</sup> apod. [9] Hlavní rozdíl mezi CMSIS a HALu<sup>11</sup> STMicroelectronics je ten, že CMSIS je poskytnuto přímo ARM a slouží pouze na ovládání Cortex M procesorů zatímco část od STMicroelectronics poskytuje abstrakci periferií.

## ■ 2.4 Měření veličin digitálního obvodu

## ■ 2.5 Analýza komunikačních rozhraní

Kapitola 2.2 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 poslou-



Obrázek 2: STM32CubeMX HAL architektura

<sup>9</sup>Cortex Microcontroller Software Interface Standard

<sup>10</sup>Nested Vectored Interrupt Controller

<sup>11</sup>STMicroelectronics do svého HALu zabaluje i CMSIS od ARM.

cháání komunikace, které pomůže vývojáři nalézt chybu v programu nebo studentovi při realizaci školního projektu.

### 2.5.1 UART

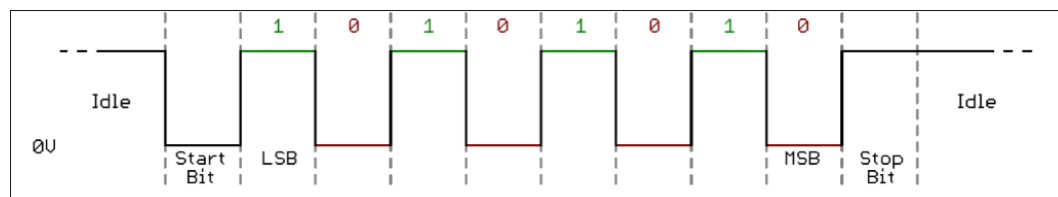
Universal Asynchronous Receiver Transmitter je rozhraní, kde data jsou odesílána bez společného hodinového signálu mezi odesílatelem a příjemcem. Místo toho je podstatný baudrate<sup>12</sup>, což určuje počet přenesených bitů za sekundu. UART podporuje nastavení různých protokolů komunikace jako například RS-232 a RS-485. UART také umí full duplex komunikaci. [10], [11]

Data jsou přenášena v tzv. rámcích, které jsou strukturovány následovně: [11]

- **Start bit** - Každý rámec začíná start bitem, který určuje začátek rámce. Bit je vždy „0“.
- **Slovo dat** - Poté následuje 8 bitů dat<sup>13</sup>.
- **Paritní bity** - Paritní bity slouží k detekci chyby v přenosu. Parita nám dokáže pouze detekovat chybu rámce pouze v případech, kdy nevznikne chyb více<sup>14</sup>.
- **Stop bit** - Stop bit<sup>15</sup> 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 3 ukazuje způsob zpracování signálů. [12] 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 3: Způsob zpracování signálu UART [12]

### 2.5.2 I2C

**Inter-Integrated Circuit** je seriový komunikační protokol, který byl vytvořen Philips Semiconductor jako nízkorychlostní protokol pro propojení zařízení jako např. mikrokontrolery a procesory se senzory, periferiemi apod. Od roku 2006 implementace protokolu nevyžaduje licenci a proto se začal široce používat např. v IOT. Výhoda protokolu je, že pro komunikaci potřebuje pouze dva vodiče, na které je možné připojit až 128 zařízení najednou, jelikož využívá systém adres. [13]

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

<sup>12</sup>Počet bitů přenesených za sekundu

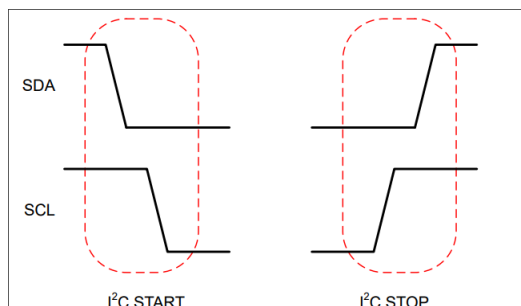
<sup>13</sup>Lze používat i 7 bitů nebo 9 bitů dat.

<sup>14</sup>Chyba z dat 1101 na 1000 nelze detekovat, protože lichá parita má paritní bit v obou případech 0.

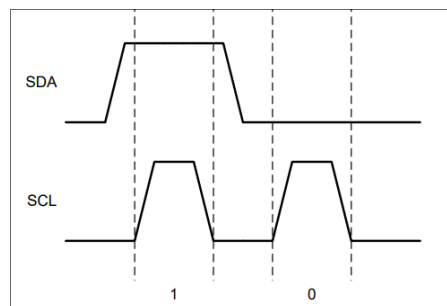
<sup>15</sup>Stop bitů může být i několik



zařízení a vodiče jsou pull up rezistorem přivedeny na společný zdroj napětí, což znamená, v klidovém režimu, jsou na vodičích vysoké logické úrovně. [13]



Obrázek 4: Zahájení a ukončení komunikace v I2C [13]



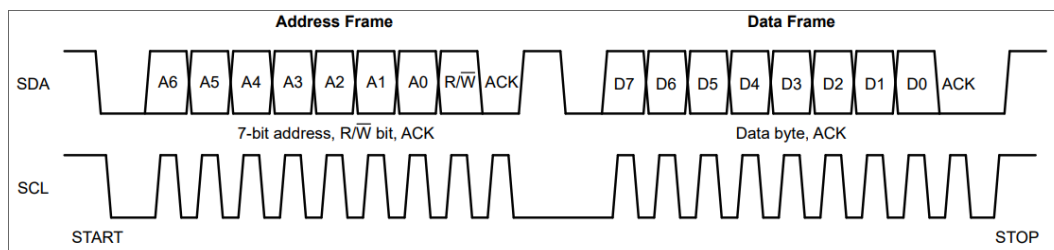
Obrázek 5: Logická jednička a nula v I2C [13]

Pro zahájení komunikace, master si 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 5 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. Příjimač zaznamená bit v momentě, kdy SCL zapulsuje.

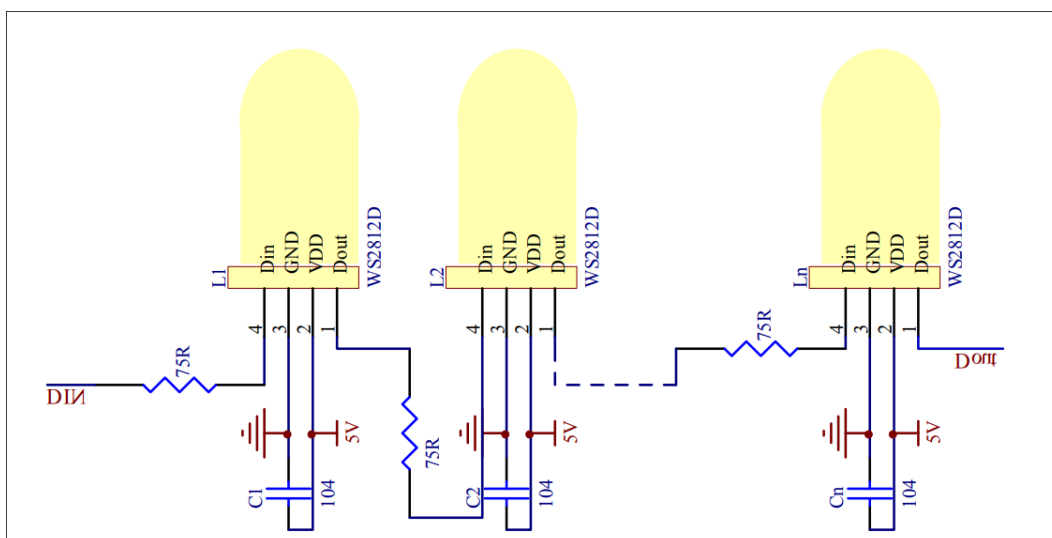
Protokol rozděluje bity do rámců. Rámec má vždy 8 bitů. Nejprve pošle adresový rámec, který identifikuje, který slave má reagovat. Součástí rámce je také read-write bit. Pokud slave přečte adresový rámec a daná adresa mu nepatří, ignoruje komunikaci. V opačném případě odpoví ACK bitem, kdy nízká úroveň 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. [13]



Obrázek 6: Rámce I2C [13]





Obrázek 7: Způsob zapojení RGB LED do série [14].

### 2.5.3 SPI

### 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 řídicí obvod, který ovládá barvy diody na základě signálu z vodiče DIn. Výhoda LED je možnost připojit diody do série, a jedním vodičem ovládat všechny LED v sérii [14]. Obrázek 7 znázorňuje zapojení více LED do série a schopnost ovládání jedním vodičem.

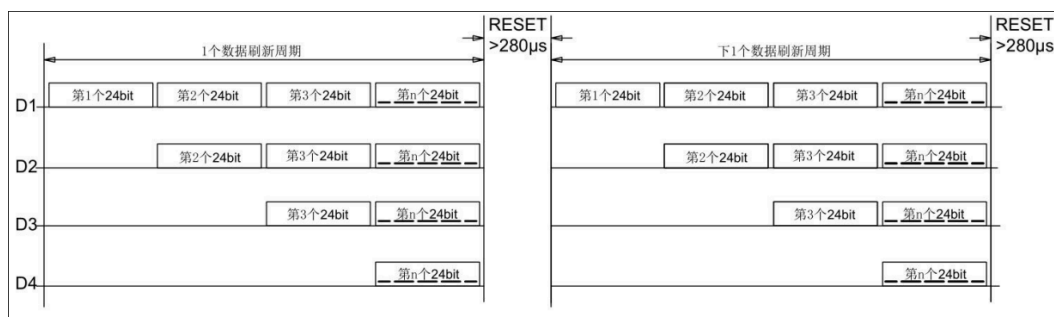
Data do LED se zasílají ve formě 24 bitů, kdy každých 8 bitů reprezentuje jednu barevnou složku. Parametry pořadí složek, časování apod. se může lišit v závislosti na konkrétním verzi a provedení LED. V této práci je vycházeno z WS2812D. První bit složky je, v případě WS2812, MSB<sup>16</sup>.

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 [14]

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 řídicímu 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 řídicí 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, řídicí obvod, už neposílá data dále, ale zpracuje je. Tabulka 2 ukazuje časování pro WS2812D.

<sup>16</sup>Most significant bit



Obrázek 8: Diagram posílání dat pro zapojené WS2812D v sérii [14]

Bit	Typ času	t[ns]
0	vysoká úroveň napětí	220 ~ 380
0	nízká úroveň napětí	580 ~ 1000
1	vysoká úroveň napětí	580 ~ 1000
1	nízká úroveň napětí	580 ~ 1000
RESET	nízká úroveň napětí	> 280000

Tabulka 2: Časování logických úrovní pro zaslání bitů WS2812D [14]

## 2.6 Grafické řešení

Kapitola 2.2 zmiňuje důraz na jednoduchou přístupnost ve výuce, což zahrnuje i jednoduché zobrazení informací, které uživatel potřebuje. Proto aby byla sonda jednoduše použitelná bez nutnosti instalace speciálního softwaru byla zvolena metoda generování TUI v terminálové aplikaci. Ke generaci rozhraní bude docházet na straně mikrokontroleru a posíláno UART periferií do PC.

### 2.6.1 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.

### Syntaxe

Základní syntaxe escape sekvencí pro formátování textu je:

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

- \033 (ASCII 27 v osmičkové soustavě) označuje začátek escape sekvence<sup>17</sup>
- [ je úvodní znak pro řídicí sekvence
- <parametry> jsou číselné kódy oddělené středníky
- <akce> je písmeno specifikující typ operace

<sup>17</sup>Existují také \033 N nebo \033 \ apod. ale tyto se téměř nepoužívají.

## ■ Formátování textu

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

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 dolů, `C` jako doprava a `D` jako doleva na pozici akce. [15]

```
\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.

```
\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
```

## 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. Některé funkce jako například buzení I2C displejů není možné na menším pouzdře realizovat z důvodu malého počtu pinů.

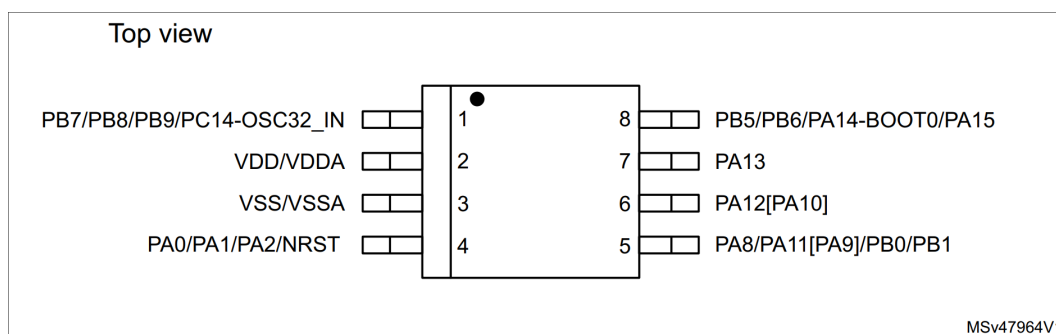
Jeden z nejpodstatnějších pinů, který potřebujeme použít pro měření je pin **PA0**. Na tomto pinu se nachází, ADC převodník, kanál 1 32 bitového časovače a ETR<sup>18</sup>. [2], [3]

**TODO: Tady navážu dále**

### 3.1 SOP8

Obrázek 11<sup>19</sup> ukazuje zapojení STM32G030 v malém pouzdře. Toto pouzdro po zapojení napájení, rozhraní UART má k dispozici pouze 4 piny. Návrh také zohledňuje realizaci lokálního módu. Tzn. pro interakci s uživatelem je připojené tlačítko na PA13 proti zemi. Tento návrh byl zvolen z důvodu snížení rizika zkratu při sestavování uživatelem. Dále je připojena RGB Neopixel LED na PB6, tento pin byl zvolen z důvodu přítomnosti časovače, o softwarové realizaci poté pojednává ... **TODO: doplnit odkaz na realizaci** WS2812 požaduje napětí 3.7 ~ 5.0 V, nicméně v návrhu bylo otestováno, že tyto diody tolerují bez potíží i 3.3 V. Mezi katodu a anodu je umístěn blokovací kondenzátor o velikosti 100 nF.

Jelikož je pouzdro malé, tak se na jednom fyzickém pinu nachází více periférií. Obrázek 9 ukazuje, že na pinu 4, kde se nachází PA0, má připojený i NRST. NReset 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.



Obrázek 9: STM32G030Jx SO8N Pinout [3]

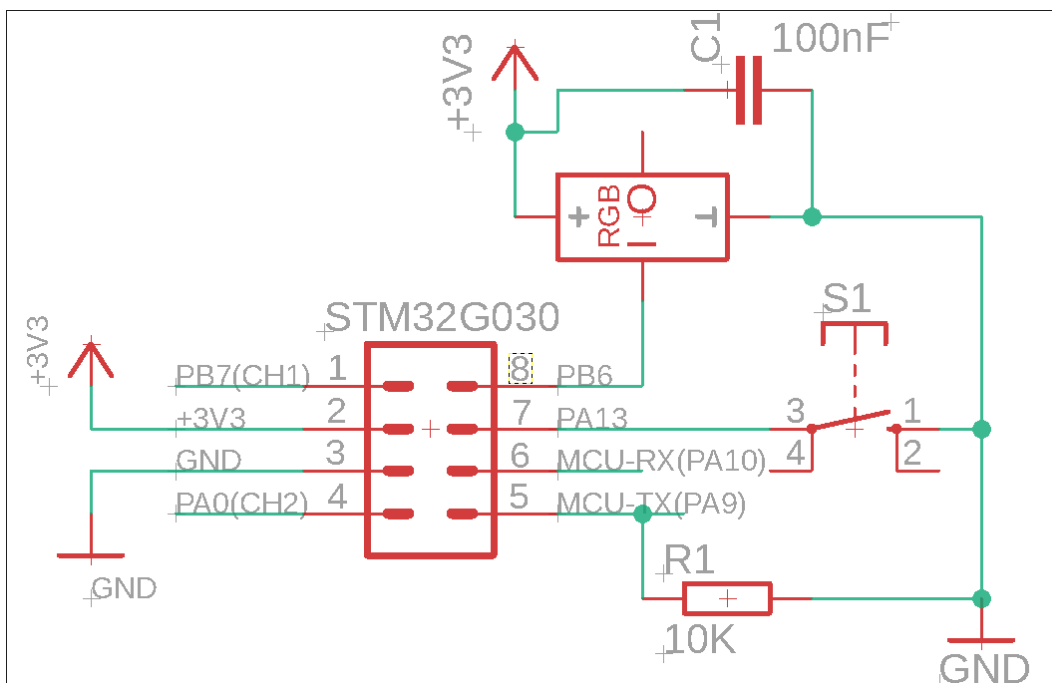
<sup>18</sup>ETR je možnost externího hodinového signálu, který řídí interní časovač.

<sup>19</sup>Schéma zapojení bylo zrealizováno pomocí nástroje *Autodesk Eagle*. [16] Komponenta Neopixel RGB LED byla použita jako externí knihovna. [17]

Address <sup>(1)</sup>	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_OPTCR <i>(3.7.8)</i>	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	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 10: Paměťový prostor Flash option bits [2]

Další problém představuje pin 8, který obsahuje PA14-B00T0. 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-B00T0 ve vysoké logické úrovni, MCU se chová stejně, jako by paměť byla prázdná. [2] Standartně se mikrokontroler nahrává a debuguje pomocí tzn. SWD<sup>20</sup>, 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 11. Pro jednoduchost se firmware nahraje pomocí UART. V tomto případě je ale potřeba řídit, zda má být nahráván firmware nebo spuštěn program. Optional bit nB00T\_SEL určuje, zda má být toto řízeno pomocí bitů nB00T0 a nB00T1 nebo pomocí úrovně PA14-B00T0. V případě sondy, je potřeba druhá možnost, takže je nutné nastavit bit nB00T\_SEL na 0.



Obrázek 11: Schéma zapojení STM32G030 v pouzdře SOP8

<sup>20</sup>Serial Wire Debug slouží pro jednodušší vývoj na mikrokontrolerech, je možné číst FLASH, RAM, nahrávat program, nastavovat option bity apod.

PB7 byl využit jako poslední, protože neobsahuje žádné zásadní periferie pro logickou sondu, kromě kanálu AD převodníku. Obecně hlavní je PA0, který má velké množství funkcí a PB7 je v tomto případě jako sekundární kanál.

### ■ 3.2 TSSOP20

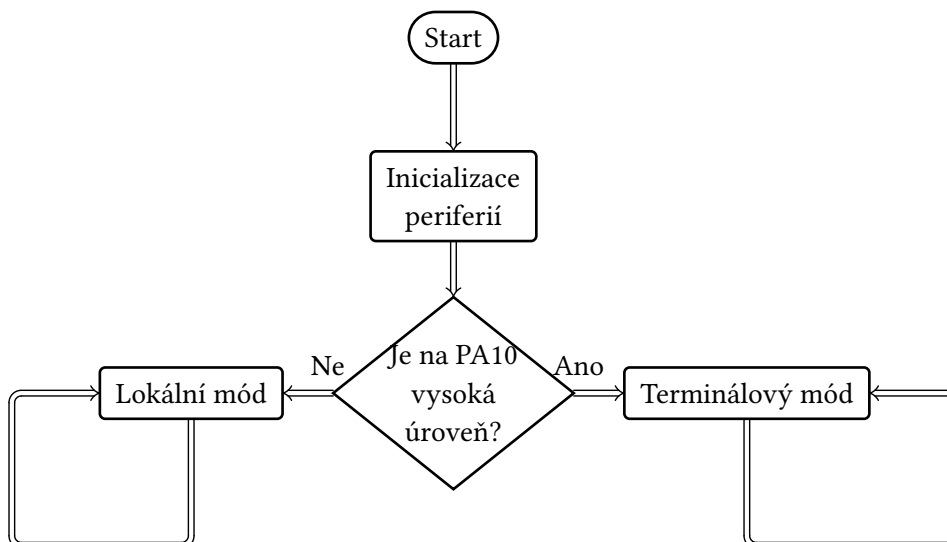
## Kapitola 4

# SW návrh logické sondy STM32

Při zapnutí mikrořadiče, proběhne inicializace všech nutných periférií. Pro STM32 je to Časovače číslo 1,2 a 3, AD převodník a UART1.

### 4.1 Logika nastavení módů

Po inicializaci zařízení zařízení zkontroluje, zda má dále pokračovat v terminál módu, nebo lokálním módu. Mód se aktivuje v závislosti na logické úrovni pinu PA10 na kterém se nachází periferie USART1. Jak bylo zmíněno v Kapitola 2.5.1, pokud je PC propojeno vodičem s mikrořadičem, na vodiči se nachází vysoká úroveň. Takto dokáže kontroler určit, zda je USB převodník připojen či nikoliv. Obrázek 11 a **TODO: Doplnit obrazek tssop20** má v zapojení rezistor o velikosti 10K ohmů na pinu PA9 vůči zemi, který zaručuje, při nezapojeném pinu, nízkou logickou úroveň.



Po načtení módu zařízení reaguje na různé podněty v závislosti, na načteném módu. Aby uživatel mohl měnit jednotlivé módy, tak je zařízení vždy nutné vypnout a zapnout aby došlo ke správné inicializaci. Jednotlivé módy běží v nekonečném cyklu, dokud zařízení není vypnuto.

### 4.2 Lokální mód

Jak Kapitola 2.1 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. Zařízení skrze tlačítko rozpozná tři interakce: krátký stisk slouží k přepínání logických úrovní 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. Stavby logické sondy jsou celkově tři.

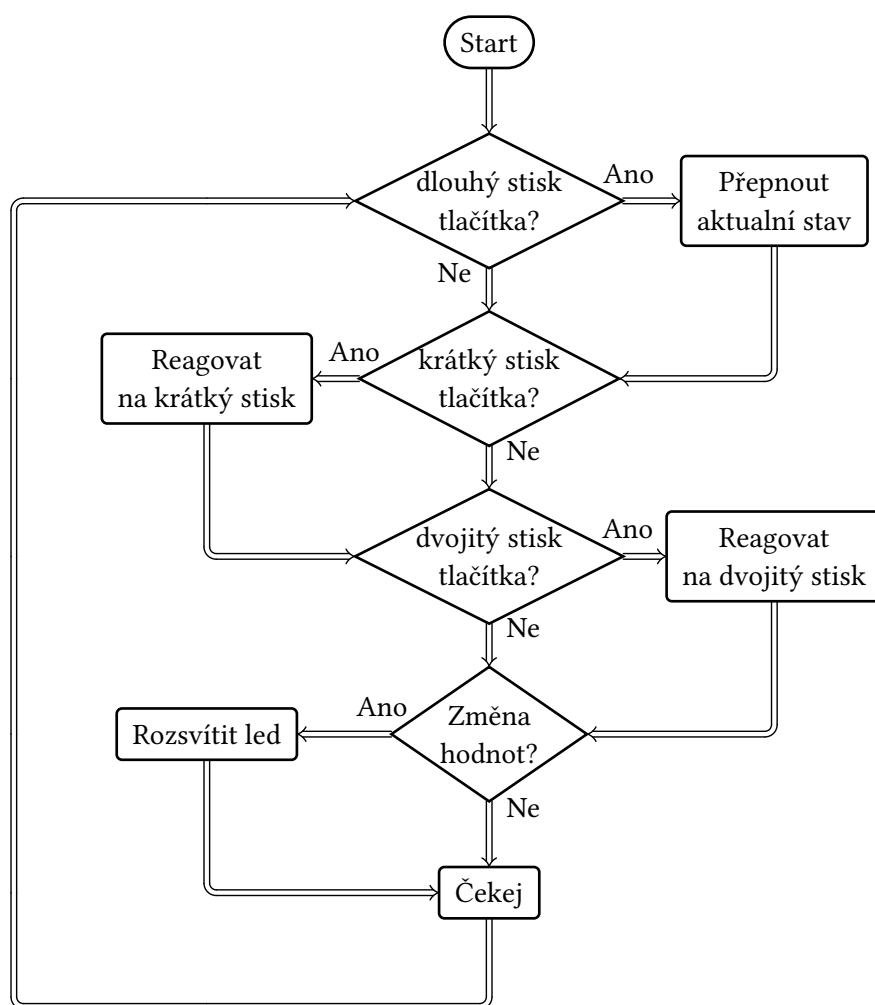
Při zapnutí zařízení se vždy nastaví stav **logické sondy**. Tento stav čte na příslušném kanálu periodicky, jaká logická úroveň je naměřena AD převodníkem. Logickou úroveň je možné číst také jako logickou úroveň na GPIO, nicméně to neumožňuje rozlišit stav, kdy logická úroveň je v neurčité oblasti. Pomocí měření napětí na pinu lze zjistit zda napětí odpovídá TTL logice či nikoliv. Pokud na pinu se nachází vysoká úroveň, LED se rozsvítí zeleně, v případě nízké úrovně se rozsvítí červená a pokud je napětí v neurčité oblasti, LED nesvítí. Tlačítkem poté lze přepínat mezi jednotlivými kanály.

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.

Poslední stav je **detekce pulzů**. Detekování pulzů probíhá za pomoci 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 je nutné aby bylo přerušení krátké, tzn. není možné aby se na 1 sekundu rozsvítila led. Další důvod je ten, že takto je zaručeno, že se vždy splní úkony ve správném pořadí. V ... 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.

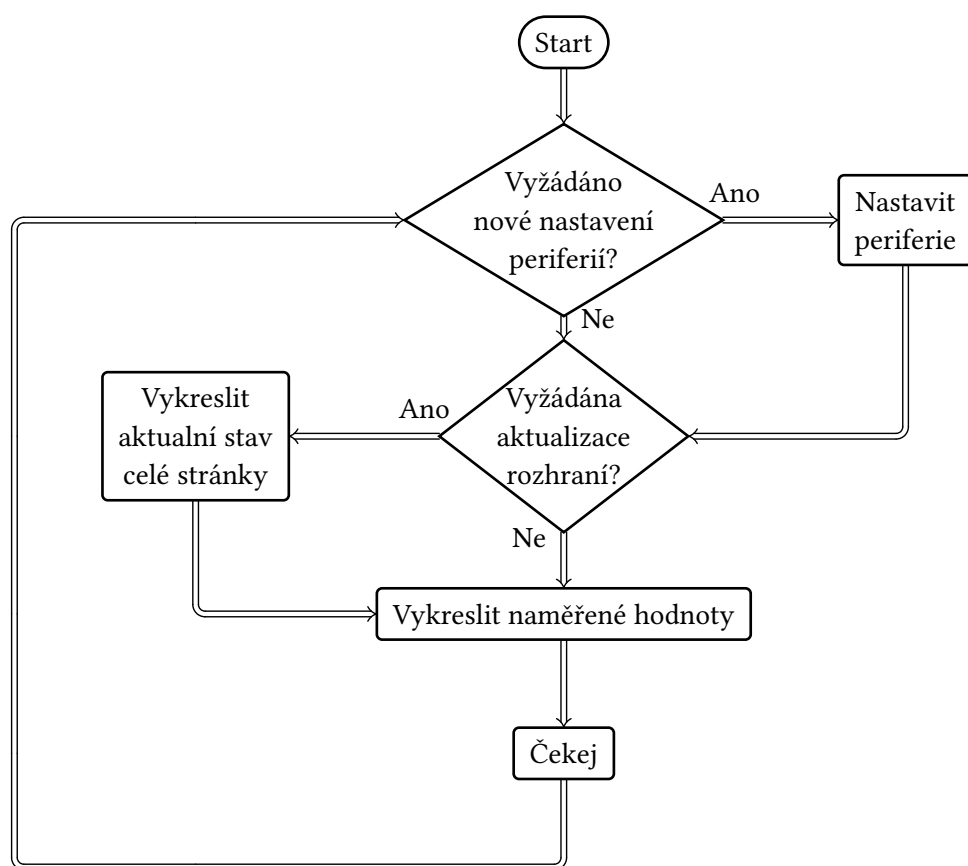




### 4.3 Terminálový mód

Tento mód využívá rozhraní UART, pro seriovou komunikaci s PC. Mód funguje způsobem, kdy periodicky reaguje na změny, které periferie či uživatel vyvolá. Tuto skutečnost ukazuje Obrázek 12. Sonda obsahuje datovou strukturu, ve které uchovává flagy, které značí požadavek na změnu. Tyto flagy jsou ovládány skrze přerušení. Pokud uživatel, stiskne tlačítko a tím pošle znak, UART rozhraní vyvolá přerušení a následně se dle stavu sondy a poslaného znaku provede akce. Přerušení také nastaví flagy, pokud například, je nutné vykreslit jinou stránku, nebo změnit měřicí režim. Smyčka při dalším cyklu na tyto skutečnosti zareaguje a přenastaví potřebné periferie a vykreslí stránku. Pokud stránka zobrazuje měřené hodnoty, jsou aktualizovány v každém cyklu.

Tato metoda oproti okamžité reakci již v přerušení má výhodu v tom, že nemůže dojít k překrytí činnosti hlavní smyčky. 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. Touto metodou zajistíme, že vždy je vykonávána akce ve správném pořadí. **TODO: dokončit, napsat o tom jak funguje nastavování periferií apod**



Obrázek 12: Diagram smyčky terminálového módu

## Kapitola 5

# Realizace logické sondy

**TODO: REVIZE**

### 5.1 Grafické rozhraní

**TODO: REVIZE** Pro snadné pochopení ovládání i jedincem, který se zabývá podobným tématem poprvé, je podstatné, aby logická sonda byla jednoduše ovladatelná, přenositelná a obecně aby zprovoznění sondy nebylo náročné. Pro spoustu začátečníků může být obtížné zjistit, jak mikrokontrolér připojit k PC, jaký program nainstalovat, jaké ovladače nainstalovat a podobně.

Proto v případě logické sondy bylo zvoleno řešení, kde uživatel pouze připojí logickou sondu k PC a nainstaluje si známý terminál pro seriovou komunikaci, a může sondu používat. Je to z důvodu, že logická sonda využívá ANSI escape sekvence pro generování terminálového uživatelského rozhraní. Tento přístup nevyžaduje instalaci ovladačů pro specifický software a hlavně není závislý na operačním systému. Tzn. podpora této logické sondy je na všechny standardní operační systémy. V Kapitola 2.5.1 bylo zmíněno, že UART umí full duplex komunikaci, díky tomu počítač může posílat zprávy i do mikrokontroleru. Tuto vlastnost logická sonda použije pro ovládání rozhraní uživatelem.

V rámci této práce byl využíván software **PuTTY**. FOSS<sup>21</sup>, který má podporu různých komunikačních protokolů, jako je SSH, Telnet, SCP a další. PuTTY podporuje také ANSI escape sekvence a je možné upravit velké množství nastavení.

#### 5.1.1 Odesílání zpráv

Před odesláním první zprávy přes seriovou komunikaci je žádoucí inicializovat UART periférii. To je možné přes STM32CubeMX<sup>22</sup>, kde vývojář nastaví potřebné parametry a je mu vygenerován základní kód. Pro potřeby projektu bylo zvoleno následující nastavení:

```
static void MX_USART1_UART_Init(void) {
    huart2.Instance = USART1;
    huart2.Init.BaudRate = 115200;
    huart2.Init.WordLength = UART_WORDLENGTH_8B; // velikost dat
    huart2.Init.StopBits = UART_STOPBITS_1; // počet stop bitů
    huart2.Init.Parity = UART_PARITY_NONE;
    huart2.Init.Mode = UART_MODE_TX_RX; // Zapnut full duplex
    huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
    huart2.Init.OverSampling = UART_OVERSAMPLING_16;
    huart2.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
    huart2.Init.ClockPrescaler = UART_PRESCALER_DIV1;
    huart2.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
    if (HAL_UART_Init(&huart1) != HAL_OK) {
        Error_Handler();
    }
}
```

<sup>21</sup>Free open source software

<sup>22</sup>STM32CubeMX je grafický program, který dává jednodušší možnost úpravy periférií.

```

    }
}

```

Po inicializaci je možné poslat zprávu pomocí například následovně:

```

void ansi_send_string(const char* str) {
    HAL_UART_Transmit(&huart1, (uint8_t*)str, strlen(str), HAL_MAX_DELAY);
}

```

kde reference na `huart1` je inicializovaná struktura. Pro jednodušší generování rozhraní byla vytvořena abstrakce zvaná `ansi abstraction layer` (dále jen AAL). Tato abstrakce umí potřebné ANSI sekvence generovat. Zde je příklad funkce, která odesílá text, který již má speciální efekty:

```

// Odesílání textu s efekty
void ansi_send_text(const char* str,
                   const char* color,
                   const char* bg_color,
                   const _Bool bold) {
    if (bg_color != NULL && strlen(bg_color) != 0) {
        ansi_send_string(bg_color);
    }
    if (color != NULL && strlen(color) != 0) {
        ansi_send_string(color);
    }
    if (bold) {
        ansi_send_string("\033[1m"); // Tučné písmo
    }
    ansi_send_string(str);
    ansi_send_string("\033[0m"); // Reset formátování
}

```

Nebo například funkce, která nastavuje kurzor na specifickou pozici:

```

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

    char result[BUFF_SIZE];

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

    if (ret >= sizeof(result) || ret < 0) {
        Error_Handler(); // kontrola zda se text vešel do bufferu
    }
}

```

```

    }

    ansi_send_string(result);
}

```

Další funkce, které zjednodušují ovládání jsou k nalezení v souboru `ansi_abstraction_layer.c`. Pro další kontext budou postačovat hlavně tyto dvě funkce.

Pomocí AAL je možné vykreslovat efektivně větší celky. Zde je příklad z TUI, kdy jedna funkce pomocí instrukcí sestaví větší celek, v tomto případě nápovědu pro odchyťávání signálu:

```

void ansi_frequency_reader_generate_hint(void) {
    ansi_set_cursor(TERMIAL_HEIGHT - 2, 4);
    ansi_send_text("m - change mode ", RED_TEXT, "", false);
    ansi_set_cursor(TERMIAL_HEIGHT - 2, 21);
    ansi_send_text("t - change gate time ", BLUE_TEXT, "", false);
    ansi_set_cursor(TERMIAL_HEIGHT - 2, 45);
    ansi_send_text("d - delete flag ", GREEN_TEXT, "", false);
}

```

Tyto větší celky ulehčili tvorbu tzv. ASCII ART<sup>23</sup>, ohraničení nebo tvorby menu, což vylepší vizuál stránky. Obrázek 13 ukazuje, jak vypadá hlavní stránka logické sondy realizované skrze AAL.

---

<sup>23</sup>ASCII ART je termín pro obrázek, který je vytvořen pomocí symbolů ASCII.



Obrázek 13: TUI hlavní stránky logické sondy

### ■ 5.1.2 Přijímání zpráv

Pro zajištění komunikace mezi sondou a zařízením je nutné také zpracovávat vstupy od uživatele, které jsou přijímány na rozhraní UART. K tomuto účelu sonda využívá přerušení, které je vyvolané při přijetí znaku na UART. Pro implementaci je použit callback `HAL_UART_RxCpltCallback`, který je již deklarován v knihovně HAL, tzn. stačí ho definovat pro správnou funkcionalitu.

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef* huart) {
    if (huart->Instance == USART1) {
        HAL_UART_Receive_IT(&huart1, &received_char, 1);
        if (received_char == 'r' || received_char == 'R') { // reload
            ansi_clear_terminal();
            render_current_page();
        } else {
            get_current_control();
        }
    }
}
```

Callback se skládá ze dvou částí, jedna je samotný callback, který zjistí, jaký UART přijmul znak a zkontroluje, zda to není znak, který je obecný pro všechny okna TUI. Pokud ne, zavolá `get_current_control`. Kde se nachází switch, který podle stránky, na které se uživatel nachází, zvolí ovládací funkci.

```
void control_main_page(void) {
    switch (received_char) {
        case 'v':
        case 'V':
            ansi_clear_terminal();
            ansi_voltage_page();
            break;
        case 'c':
        case 'C':
            ansi_clear_terminal();
            ansi_channel_set_page();
            break;
        case 'f':
        case 'F':
            ansi_clear_terminal();
            ansi_frequency_reader_page();
            break;
        case 'g':
        case 'G':
            ansi_clear_terminal();
            ansi_impulse_generator_page();
    }
}
```

Zde je možné například vidět funkci, která slouží pro ovládání hlavní stránky. Jak je vidět, funkce podporuje velké i malé písmo. Po zjištění znaku spustí určitou akci. Například při stisku V, vyčistí terminalové okno a přenastaví stránku na voltmetr.

Zde je podstatné, že samotný interrupt nenastavuje okno, ale pouze přenastaví flag a ke změně dojde v hlavní smyčce. Pokud by to tak nebylo, mohlo by dojít ke kolizi akcí. Například hlavní smyčka by byla v průběhu překreslování a v ten moment, by bylo vyvoláno přerušení UARTem. Přerušení by překreslilo stránku a program by po přerušení pokračoval ve stejném bodě. Zde by se mohlo stát, že by se polovina stránky překreslila a polovina ne.

## ■ 5.2 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 reálné referenční napětí.

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

```
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();
    }
    sConfig.Channel = ADC_CHANNEL_1;
    sConfig.Rank = ADC_REGULAR_RANK_1;
    sConfig.SamplingTime = ADC_SAMPLETIME_COMMON_1;
}
```



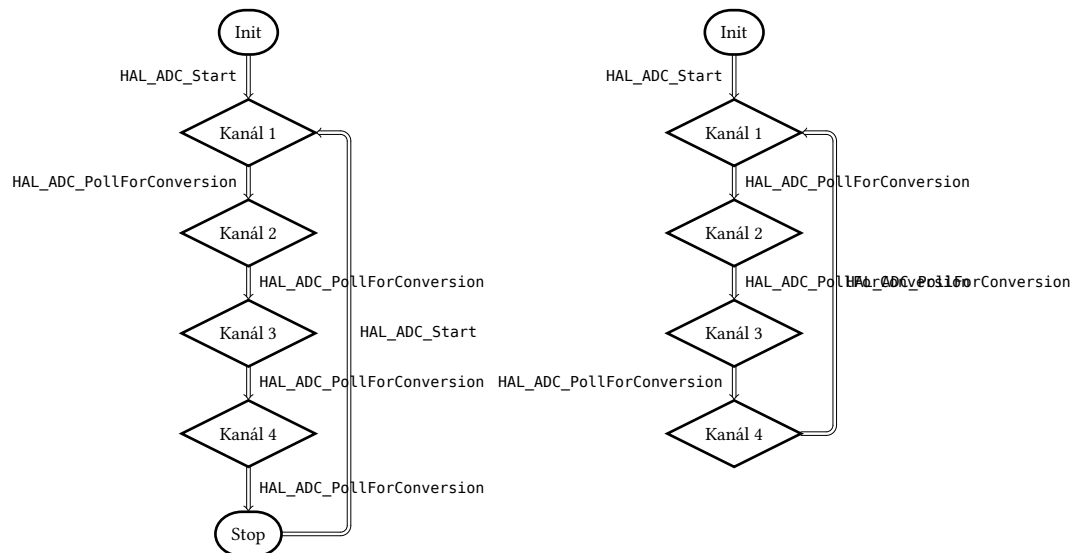
```

if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK) {
    Error_Handler();
}
adc1_ch = create_adc_channels(&hadc1);
realloc_v_measures(adc1_ch, &v_measures);
setup_adc_channels(&hadc1, adc1_ch, true);
}

```

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<sup>24</sup>. 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.

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`.

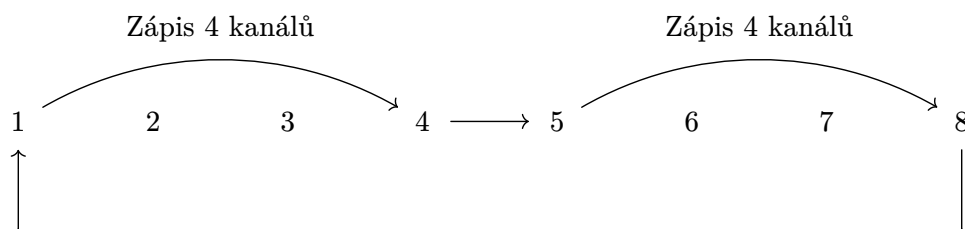
Logická sonda pracuje ve smyčce, kdy jednou za určitý krátký úsek zastaví klasické měření kanálů a změří referenční napětí. Tento způsob zajišťuje neustále validní referenci.

<sup>24</sup>Tato hodnota je v STM32G0 nejvyšší a zaručuje nám tu největší možnou přesnost. Některé mikrořadiče nabízí i vyšší počet odběrů.

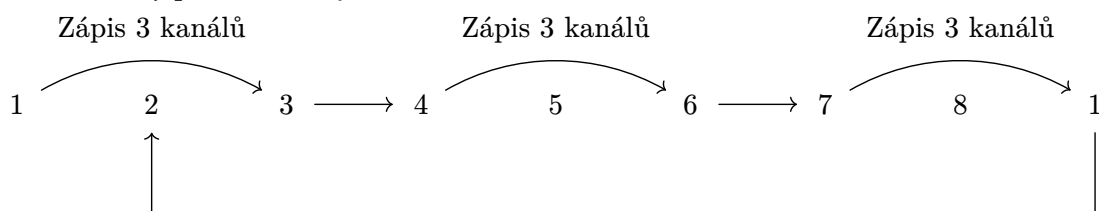
Během měření kanálů je nutné, aby nebyl zbytečně zatěžován procesor. Procesor v hlavní smyčce pracuje na vykreslování dat do terminálu a pokud by prováděl měření, mohlo by to ovlivnit rychlost měření napětí a nemuselo by být dosaženo dostatečného sebrání vzorků pro plovoucí průměr. Toto je vyřešeno pomocí DMA<sup>25</sup>. HAL aktivuje DMA pro AD převodník pomocí `hadc1.Init.DMAContinuousRequests` nastavené na `ENABLE`. Parametry DMA lze nastavit pomocí `STM32CubeMX`. DMA je nastaveno jako cirkulární 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 paměť 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 jednoduše 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.

Po určitém časovém úseku, procesor zpracuje data z DMA a zprůměruje hodnoty z AD převodníku, následně hodnoty převede dle metody v Kapitola 2.3.1.1 a vykreslí na seriovou linku pomocí ANSI sekvencí zmíněné v Kapitola 5.1.1.

TUI vykresluje hodnoty na každém kanálu a poté vykresluje, zda je logická úroveň vysoká, nízká a nebo je nejasná. Obrázek 14 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

<sup>25</sup>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.



Obrázek 14: Stránka pro měření napětí a logických úrovní

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. Obrázek 15 ukazuje vzhled této stránky. Uživatel pomocí klávesových zkratk 1 až 4 volí jaké kanály 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.

```
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;
```



Obrázek 15: Stránka pro nastavení jednotlivých kanálů

### ■ 5.3 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 [6].

Pro přesné zjištění 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 5.2.

K zjištění frekvence je použita tzv. metoda hradlování. Metoda hradlování využívá periodu vzorkování, která je využita pro spočítání finální frekvence. Rovnice 6 uvádí způsob, jak spočítat frekvenci pomocí metody hradlování. Frekvence lze spočítat jako polovinu napočítaných pulzů za časový úsek<sup>26</sup>. K tomuto účelu je použit 32 bitový časovač, aby bylo možné měřit, co největší frekvence.

$$F = \frac{N}{\frac{2}{T}} \quad (6)$$

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.

<sup>26</sup>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.

```

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 6 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.

```

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.

Kromě frekvence, umí sonda odchyťávat 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 odchyťí 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.



## ■ 5.4 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 jednoduše 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 úroveň, které uživatel nemusí nutně používat jako generování pulzů, ale jako přepínání úrovní dle potřeby. **TODO: ještě se na to kouknout znovu a upravit to neaktuální**

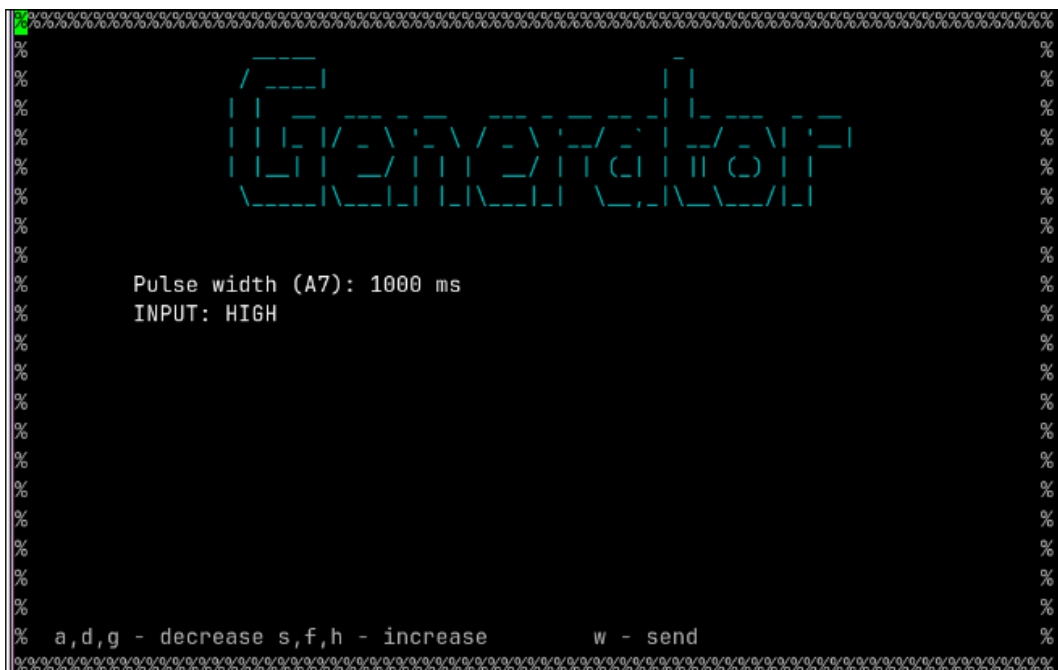
```
static void MX_TIM16_Init(void) {
    htim16.Instance = TIM16;
    htim16.Init.Prescaler = 63999; // předdělička
    htim16.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim16.Init.Period = 999; // výchozí hodnota periody
    htim16.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    htim16.Init.RepetitionCounter = 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 18 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.

```
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);

    if (con && generator->con) {
        HAL_TIM_Base_Stop_IT(&htim16);
        generator->con = false;
    } else {
        generator->con = con;
    }
}
```



Obrázek 18: Stránka pro generování signálu

```

    HAL_TIM_Base_Start_IT(&htim16);
}
}

```



**Kapitola 6**

## **Závěr a zhodnocení**

# Citace

- [1] STMicroelectronics, „STM32G0 Series”. [Online]. Dostupné z: <https://www.st.com/en/microcontrollers-microprocessors/stm32g0-series.html>
- [2] STMicroelectronics, „STM32G0x1 Reference manual”. [Online]. Dostupné z: [https://www.st.com/resource/en/reference\\_manual/rm0444-stm32g0x1-advanced-armbased-32bit-mcus-stmicroelectronics.pdf](https://www.st.com/resource/en/reference_manual/rm0444-stm32g0x1-advanced-armbased-32bit-mcus-stmicroelectronics.pdf)
- [3] STMicroelectronics, „STM32G030x6/x8”. [Online]. Dostupné z: <https://www.farnell.com/datasheets/2882477.pdf>
- [4] STMicroelectronics, „STM32G0-ADC: Product training”. [Online]. Dostupné z: [https://www.st.com/resource/en/product\\_training/STM32G0-Analog-ADC-ADC.pdf](https://www.st.com/resource/en/product_training/STM32G0-Analog-ADC-ADC.pdf)
- [5] iter, „Calibrating STM32 ADC (VREFINT)”. [Online]. Dostupné z: <https://stackoverflow.com/questions/58328342/calibrating-stm32-adc-vrefint>
- [6] M. Dudka, „STM32 Timery”. [Online]. Dostupné z: [http://www.elektromys.eu/clanky/stm\\_timer1/clanek.html](http://www.elektromys.eu/clanky/stm_timer1/clanek.html)
- [7] STMicroelectronics, „STM32CubeMX”. [Online]. Dostupné z: [https://www.st.com/content/st\\_com/en/stm32cubemx.html](https://www.st.com/content/st_com/en/stm32cubemx.html)
- [8] M. Hasan, „Understanding STM32 HAL Library Fundamentals”. [Online]. Dostupné z: <https://embeddedthere.com/understanding-stm32-hal-library-fundamentals/>
- [9] A. Limited, „What is CMSIS?”. [Online]. Dostupné z: <https://www.keil.arm.com/cmsis>
- [10] G. Wright, „Definition USART (universal synchronous/asynchronous receiver/transmitter)”. [Online]. Dostupné z: <https://www.techtarget.com/whatis/definition/USART-Universal-Synchronous-Asynchronous-Receiver-Transmitter>
- [11] Wikipedia contributors, „Universal asynchronous receiver-transmitter — Wikipedia, The Free Encyclopedia”. [Online]. Dostupné z: [https://en.wikipedia.org/w/index.php?title=Universal\\_asynchronous\\_receiver-transmitter&oldid=1281297858](https://en.wikipedia.org/w/index.php?title=Universal_asynchronous_receiver-transmitter&oldid=1281297858)
- [12] V. H. Adams, „Universal Asynchronous Receiver Transmitter”. [Online]. Dostupné z: <https://vanhunteradams.com/Protocols/UART/UART.html>
- [13] Joseph Wu - TI, „A Basic Guide to I2C”. [Online]. Dostupné z: <https://www.ti.com/lit/an/sbaa565/sbaa565.pdf>
- [14] Worldsemi, „WS2812D-F5-1261 Intelligent control LED integrated lightsource”. [Online]. Dostupné z: <https://www.tme.eu/Document/bd6b355a8705d46515a8ada0d153187b/WS2812D-F5-L.pdf>
- [15] fnky, „ANSI Escape Sequences”. [Online]. Dostupné z: <https://gist.github.com/fnky/458719343aabd01cfb17a3a4f7296797>
- [16] Autodesk, „Eagle”. [Online]. Dostupné z: <https://www.autodesk.com/products/eagle/overview>
- [17] Albert van Dalen, „addressable LED rings NeoPixel WS2812B-2020”. [Online]. Dostupné z: <https://github.com/avandalen/Eagle-library-addressable-LED-rings>