

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
March 2025**

assignment page 1

assignment page 2

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.

Abstrakt

Výuka základů elektroniky vyžaduje nástroje, které studentům umožní experimentovat s reálnými 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.

Poděkování

Rád bych tímto poděkoval panu doc. Ing. Janu Fischerovi, CSc., za jeho cenné rady, odbornou pomoc a ochotu sdílet své znalosti. Děkuji mu také za jeho čas a podnětné připomínky.

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, 28. 03. 2025

2 TODOs remaining

Obsah

1 Úvod	1
2 Použité principy a technologie	2
2.1 STM32G030	2
2.1.1 Analogo-digitální převodník	2
2.1.2 Časovače	3
2.1.3 USART periferie	4
2.2 STM HAL	4
2.3 Raspberry Pi Pico	5
2.4 Logická sonda	5
2.5 Sériová komunikační rozhraní	5
2.5.1 UART	5
2.5.2 I2C	6
2.5.3 SPI	6
2.5.4 Neopixel	6
2.6 Ansi sekvence	6
2.6.1 Historie Sekvence vnikly	6
2.6.2 Kódy	6
2.7 Metody měření	7
3 Návrh logické sondy	8
3.1 Požadavky	8
3.2 HW návrh	8
3.2.1 STM32G030 SOP8	8
3.2.2 STM32G030 TSSOP20	8
3.2.3 Raspberry Pi Pico	8
3.3 SW návrh	8
3.3.1 Logika nastavení módů	8
3.3.2 Logika lokálního módu	9
4 Realizace logické sondy	11
4.1 Grafické rozhraní	11
4.1.1 Odesílání zpráv	11
4.1.2 Přijímání zpráv	15
4.2 Měření napětí a zjišťování logické úrovně	16
4.3 Odchytání pulzů a frekvence	20
4.4 Generování pulzů	23
5 Závěr a zhodnocení	25
Citace	26

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 semestrální práce se zaměřuje počáteční 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, odchyťování 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

Použité principy a technologie

2.1 STM32G030

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].

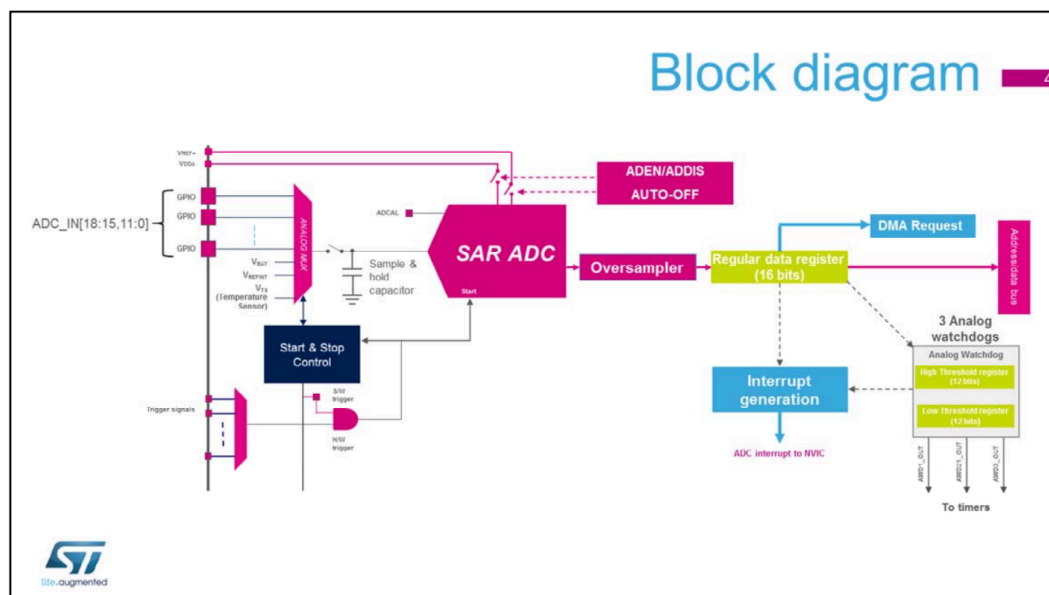
Pro řadu G030 jsou typické kompaktní rozměry ať už vývojové Nucleo desky, tak typové pouzdra jako například **TSSOP20** nebo **SOP8**, což poskytuje snadnou integraci do kompatního hardwarového návrhu [3]. Obě zmíněné pouzdra jsou použity pro implementaci logické sondy, o které se více zmíníme v ...

2.1.1 Analogo-digitální převodník

Mikrokontrolér STM32G030 je vybaven ADC, který obsahuje 8 analogových kanálů o rozlišení 12 bitů. Maximální vzorkovací frekvence převodníku je 1 MSPS¹.

Při měření kanálů se postupuje sekvenčně, která je určena pomocí tzv. ranků. 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. oversampling². Převodník obsahuje **accumulation data register**, který akumuluje měření a poté pomocí data shifteru vydělí počtem cyklu [4].



Obrázek 1: Blokový diagram AD převodníku

¹milion vzorků za sekundu

²Proběhne více měření a následně jsou výsledky např. zprůměrovány aby byla zajištěna větší přesnost.

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

AD převodník, po dokončení měření vzorků, vrací hodnotu, která není napětí. Pro převedení hodnoty převodníku na napětí je nutné znát referenční napětí systému ($V_{\text{REF}+}$). Referenční napětí může být proměnlivé, hlavně pokud systém využívá VDDA³ jako referenci, která může kolísat vlivem napájení a nebo zatížení.

Pro výpočet $V_{\text{REF}+}$ se používá interní referenční napětí V_{REFINT} kalibrační data uložená během výroby mikrořadiče a naměřené hodnoty z ADC [2].

Vztah pro výpočet je následující:

$$V_{\text{REF}+} = \frac{V_{\text{REFINT_CAL}} \times 3000}{V_{\text{REFINT_ADC_DATA}}} \quad (2)$$

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 3.0 V. Hodnota se získává čtením z pevné adresy⁴[2], [5].
- 3000 je konstanta odpovídající referenčnímu napětí při kalibraci vyjádřená v milivoltech.
- $V_{\text{REFINT_ADC_DATA}}$ je aktuální naměřená hodnota na AD převodníku.

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.

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

kde:

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

2.1.2 Časovače

STM32G030 obsahuje několik časovačů, které se dají využít pro logickou sondu. Mikrořadič má zabudovaných několik základních⁵ a jeden advanced timer⁶. 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 a podobně [6].

³VDDA je označení pro analogové napájecí napětí v mikrokontrolérech STM32.

⁴Např. u STM32G0 je adresa kalibrační hodnoty: 0x1FFF75AA

⁵Basic Timers

⁶Advanced Timers

Časovače jsou obecně velice komplexní téma. Tato práce se bude soustředit pouze na potřebnou část.

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 MHz⁷. Frekvence časovače určuje, jak často časovač inkrementuje svou hodnotu za jednu sekundu [2].

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

Velikost čítače časovače, zda je 16bitový nebo 32bitový⁸, 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.

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

■ 2.1.3 USART periferie

Universal Synchronous Asynchronous Receiver Transmitter je flexibilní periferie, která umožňuje seriovou komunikaci jak v asynchroním tak v synchroním režimu.

Práce využívá periferii USART1, pro komunikaci s PC během terminálového módu.

TODO: nevím, tohle asi lépe

■ 2.2 STM HAL

Hardware abstraction layer je knihovna poskytovaná společností STMicroelectronics pro jejich mikrořadiče řady STM32. Tato knihovna tvoří vrstvu abstrakce mezi aplikací 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. ti přímého přístupu k registrům procesoru. Na Obrázek 2 je znázorněn diagram, který znázorňuje architekturu HAL [8].

Součástí HALu je tzv. CMSIS⁹, což je sada standardizovaných rozhraní, které umožňují konfiguraci periférií, správu procesorového jádra, obsluhu přerušování 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 perifériím procesoru, obsahuje definice registrů, přístup k NVIC¹⁰ apod. [9]

Hlavní rozdíl mezi CMSIS a HALu¹¹ STMicroelectronics je ten, že CMSIS je poskytnuto přímo ARM a slouží pouze na ovládání Cortex M procesorů zatímco část od STMicroelectronics poskytuje abstrakci periférií.

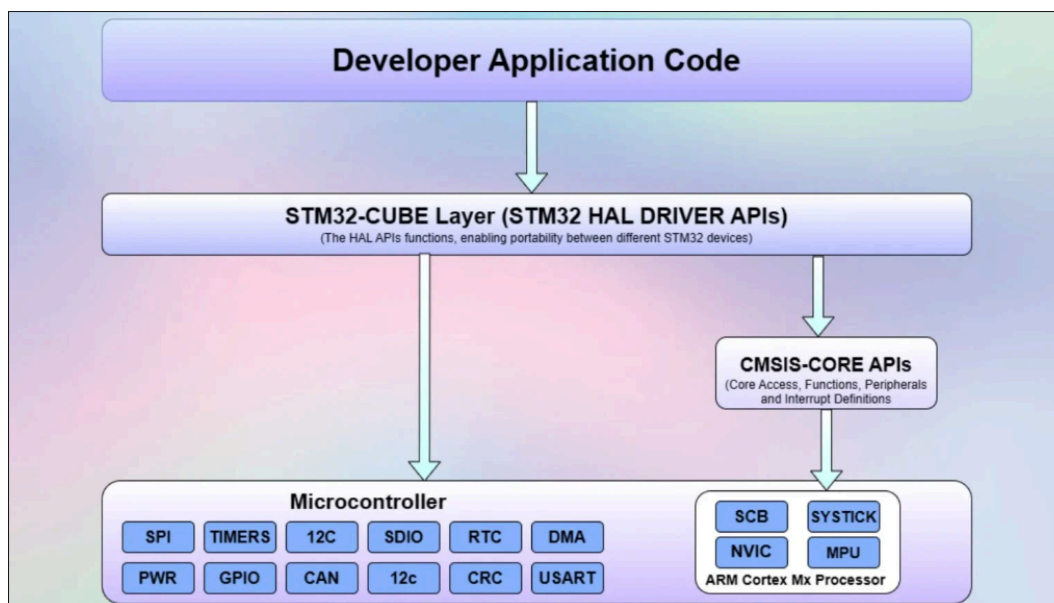
⁷Frekvenci hodin je možné upravit například pomocí CUBE IDE.

⁸U 16 bitového časovače je maximální perioda 65535.

⁹Cortex Microcontroller Software Interface Standard

¹⁰Nested Vectored Interrupt Controller

¹¹STMicroelectronics do svého HALu zabaluje i CMSIS od ARM.



Obrázek 2: STM32CubeMX HAL architektura

■ 2.3 Raspberry Pi Pico

■ 2.4 Logická sonda

■ 2.5 Sériová komunikační rozhraní

■ 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 baudrate¹², 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].

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

- **Start bit** - Každý rámec začíná start bitem, který určuje začátek rámce. Bit je vždy „0“.
- **Slovo dat** - Poté následuje 8 bitů dat¹³.
- **Paritní bity** - Paritní bity slouží k detekci chyby v přenosu. Parita nám dokáže pouze detekovat chybu rámce pouze v případech, kdy nevznikne chyb více¹⁴.
- **Stop bit** - Stop bit¹⁵ 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. Více o periférii, která pracuje s UART rozhraním mluví Kapitola 2.1.3.

¹²Rychlost přenosu

¹³Lze používat i 7 bitů nebo 9 bitů dat.

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

¹⁵Stop bitů může být i několik.

■ 2.5.2 I2C

■ 2.5.3 SPI

■ 2.5.4 Neopixel

■ 2.6 Ansi sekvence

Ansi escape codes jsou speciální kódy používané pro formátování textu v terminálech, které podporují ANSI standard. ANSI kódy poskytují změnu vzhledu textu, jako je barva pozadí, písma, pozicování a další. Největší využití mají ve vývoji terminálových rozhraní zvaná TUI.

■ 2.6.1 Historie Sekvence vznikly

jako standardizovaný soubor kódů, který měl za úkol sjednotit různé značky a modely terminálů, které používaly vendor-specific¹⁶ metody, pro tvorbu TUI. Nejznámější terminál, který podporoval ANSI sekvence byl Digital VT100. Jelikož byl velice populární, většina nových terminálů se začaly chovat podle ANSI¹⁷ sekvencí. Následně tyto sekvence byly standardizovány [11].

■ 2.6.2 Kódy

Escape kódy začínají **ESC**¹⁸ znakem, následovným [, který značí začátek sekvence, a poté symboly, které určují efekt a celá sekvence je zakončena písmenem.¹⁹

```
ESC [ <parametry> <akce>
```

Pro změnu barvy a obecně textu je použito písmeno **m** jako akce. Nejčastější kódy jsou následující:

- Změna barvy textu
 - 30 až 37: Základní barvy
 - 90 až 97: Světlé verze barev
 - 40 až 47: Základní barvy pozadí
 - 100 až 107: Světlé verze barev
- Textové efekty
 - 0: Reset předchozích efektů
 - 1: Tučný text
 - 4: Podtržení
 - 7: Inverzní
 - 9: Přeskrtnutý

Sekvence také lze použít pro pohyb kurzoru, což je užitečné pro vizuál aplikace [12].

¹⁶Nestandardizované kódy, který si každá společnost navrhla sama.

¹⁷Název ANSI byl vytvořen až později. Odpovídá zkratce American National Standards Institute.

¹⁸\33

¹⁹Existují také ESC N nebo ESC \ apod. ale tyto se téměř nepoužívají.

```
ESC[<row>;<col>H // Pohyb na konkrétní pozici  
ESC[<posun><směr> // Posune o danou pozici
```

■ 2.7 Metody měření

Kapitola 3

Návrh logické sondy

3.1 Požadavky

3.2 HW návrh

3.2.1 STM32G030 SOP8

3.2.2 STM32G030 TSSOP20

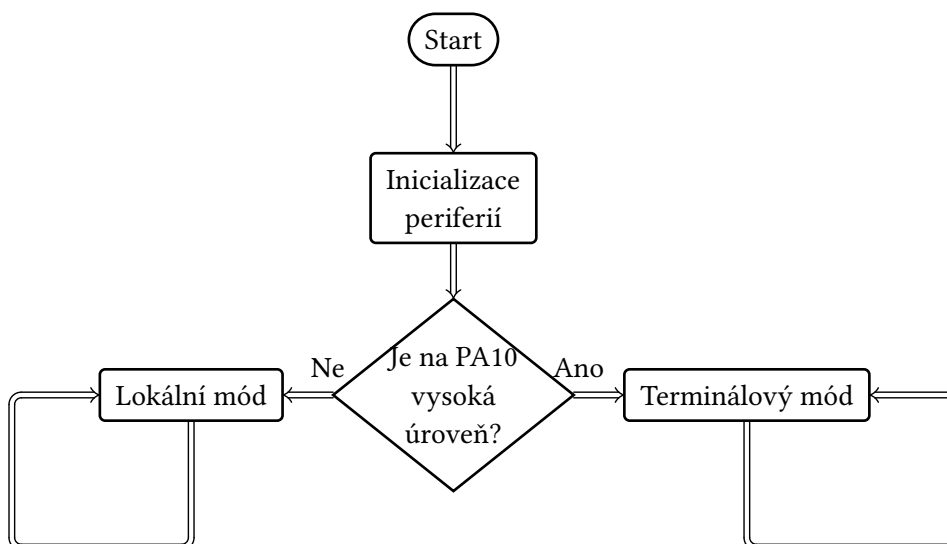
3.2.3 Raspberry Pi Pico

3.3 SW návrh

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

3.3.1 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.



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.

■ 3.3.2 Logika lokálního módu

Lokální mód je stav, kdy zařízení nevyžaduje interakci s počítačem. Tento mód je potřebný v momentě, kdy uživatel, potřebuje rychlou analýzu obvodu bez nutnosti zjišťování podrobností. Celá interakce s uživatelem je prováděna skrze tlačítko a RGB LED. Tlačítko poskytuje 3 druhy interakce:

- **Krátký stisk**
 - slouží pro přepínání úrovní
- **Dvojitý stisk**
 - slouží pro přepínání kanálů
- **Dlouhý stisk**
 - slouží pro přepínání stavů

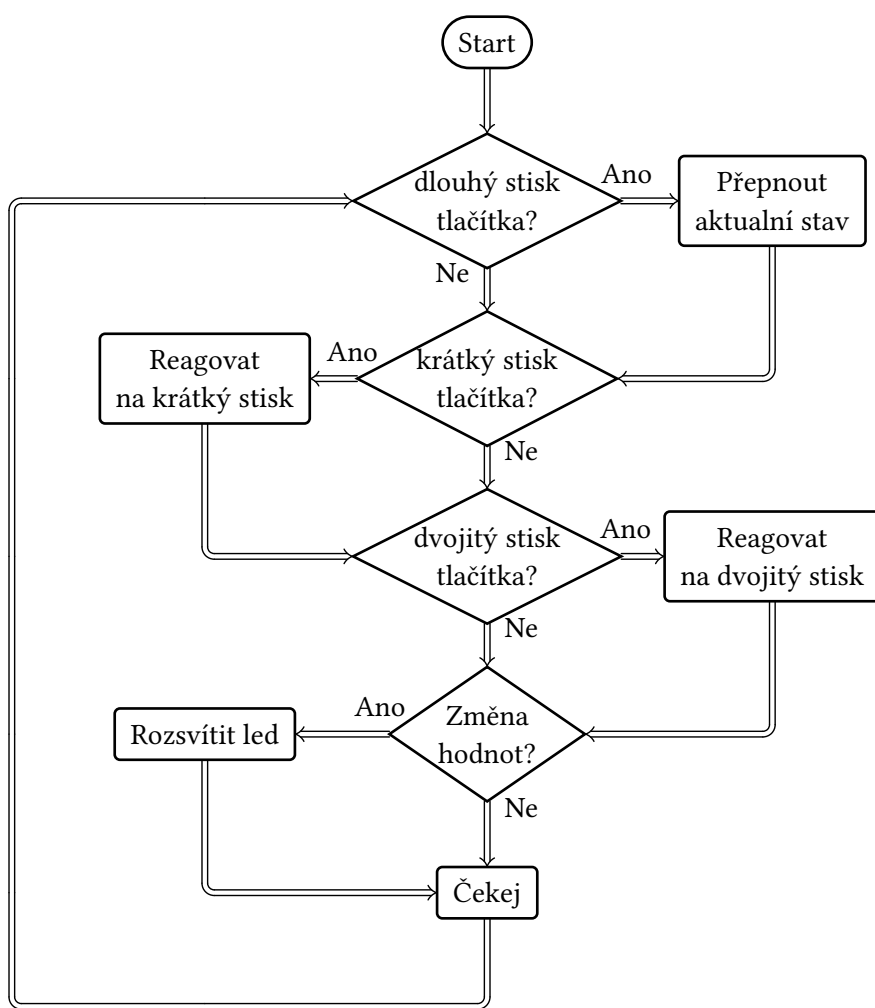
3.3.2.1 Stavy

Sonda ve stavu lokálním, má celkově 3 režimy, které se cyklicky mění dlouhým stiskem tlačítka, jednotlivé módy jsou rozlišeny barvou, která se rozsvítí na LED po dobu jedné sekundy. Stavy jsou následující:

- **Stav logické sondy**
 - Tento stav

3.3.2.2 Hlavní smyčka

Lokální mód běží ve smyčce, kde se periodicky kontrolují změny a uživatelské vstupy. Při začátku každého cyklu proběhne kontrola, zda uživatel dlouze podržel tlačítko. Pokud ano, přepne se stav.



Kapitola 4

Realizace logické sondy

4.1 Grafické rozhraní

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 2.1.3 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**. FOSS²⁰, 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.

4.1.1 Odesílání zpráv

Před odesláním první zprávy přes seriovou komunikaci je žádoucí inicializovat UART periférii. To je možné přes STM32CubeMX²¹, 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();
    }
}
```

²⁰Free open source software

²¹STM32CubeMX 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 ART²², ohraničení nebo tvorby menu, což vylepší vizuál stránky. Obrázek 3 ukazuje, jak vypadá hlavní stránka logické sondy realizované skrze AAL.

²²ASCII ART je termín pro obrázek, který je vytvořen pomocí symbolů ASCII.



Obrázek 3: TUI hlavní stránky logické sondy

■ 4.1.2 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.

■ 4.2 Měření napětí a zjišťování logické úrovně

Pro měření napětí je využíván AD převodník. Jak již bylo uvedeno v Kapitola 2.1.1, 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;
}
```

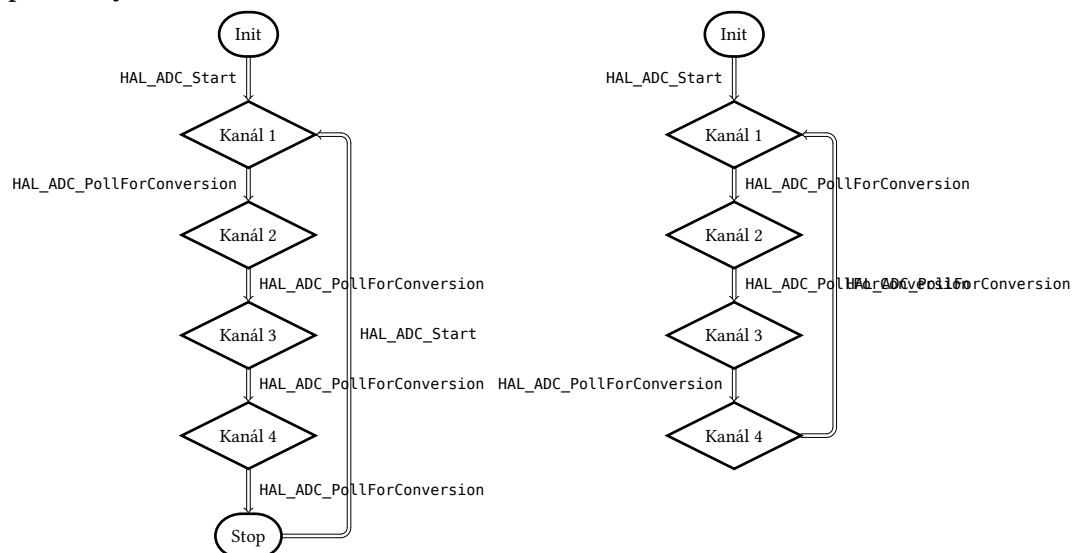
```

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 cyklů²³. 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`.

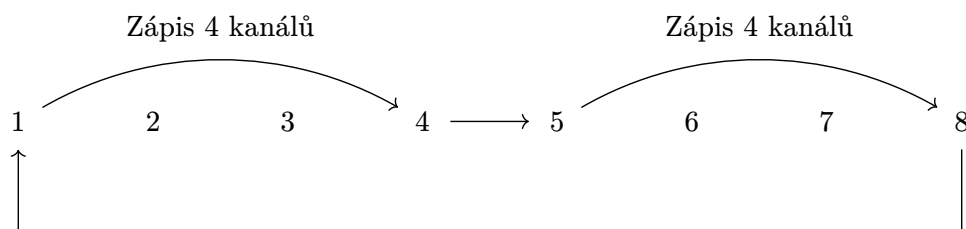
Logická sonda pracuje ve smyčce, kdy jednou za určitý krátký úsek zastaví klasické měření kanálů a změří referenční napětí. Tento způsob zajišťuje neustále validní referenci.

²³Tato hodnota je v STM32G0 nejvyšší a zaručuje nám tu největší možnou přesnost. Některé mikrořadiče nabízejí i vyšší počet odběrů.

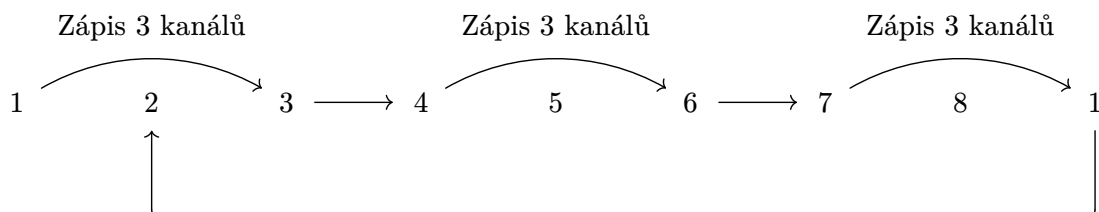
Během měření kanálů je nutné, aby nebyl zbytečně zatěžován procesor. Procesor v hlavní smyčce pracuje na vykreslování dat do terminálu a pokud by prováděl měření, mohlo by to ovlivnit rychlost měření napětí a nemuselo by být dosaženo dostatečného sebrání vzorků pro plovoucí průměr. Toto je vyřešeno pomocí DMA²⁴. 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 metodity v Kapitola 2.1.1 a vykreslí na seriovou linku pomocí ANSI sekvencí zmíněné v Kapitola 4.1.1.

TUI vykresluje hodnoty na každém kanálu a poté vykresluje, zda je logická úroveň vysoká, nízká a nebo je nejasná. Obrázek 4 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

²⁴DMA 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 4: 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 5 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;
```



```

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.

■ 4.4 Generování pulzů

Při generování pulzu má uživatel možnost nastavit konkrétní délku pulzu 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.

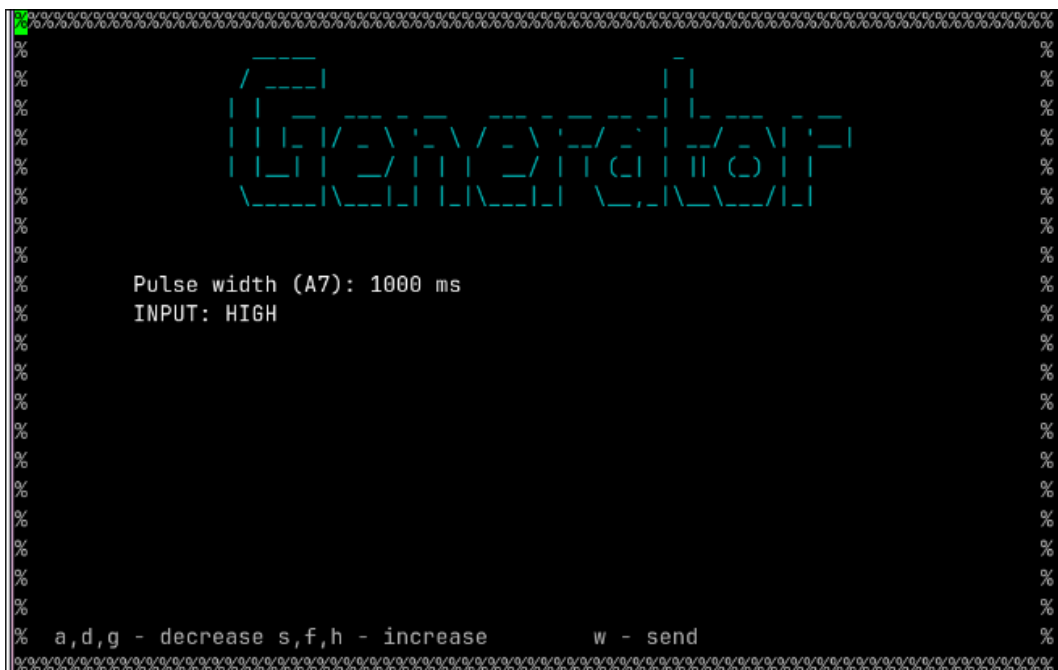
```
static void MX_TIM16_Init(void) {
    htim16.Instance = TIM16;
    htim16.Init.Prescaler = 63999; // předdělička
    htim16.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim16.Init.Period = 999; // výchozí hodnota periody
    htim16.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    htim16.Init.RepetitionCounter = 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 8 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;
        HAL_TIM_Base_Start_IT(&htim16);
    }
}
```



Obrázek 8: Stránka pro generování signálu

```
}
}
```

Kapitola 5

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
- [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
- [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
- [7] STMicroelectronics, „STM32CubeMX”. [Online]. Dostupné z: 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] T. F. E. Wikipedia, „ANSI escape code”. [Online]. Dostupné z: https://en.wikipedia.org/wiki/ANSI_escape_code
- [12] fnky, „ANSI Escape Sequences”. [Online]. Dostupné z: <https://gist.github.com/fnky/458719343aabd01cfb17a3a4f7296797>