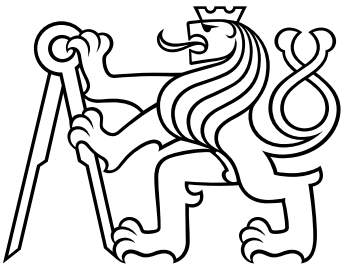
České Vysoké Učení Technické v praze
Fakulta elektrotechnická
Katedra měření



Bakalářská práce

Multifunkční diagnostická logická sonda

Milan Jiříček

jiricmi1@fel.cvut.cz
https://github.com/jiricmi/logic_probe

Vedoucí práce: doc. Ing. Jan Fischer, CSc.

Studijní program: Otevřená informatika
Obor studia:

March 2025

assignment page 1

assignment page 2

Abstract

Abstrakt

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, 25. 03. 2025

Obsah

Kapitola 1

Úvod

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

Tato semestrální práce se zaměřuje počáteční návrh a realizaci logické songy, 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

Teoretické základy

2.1 STM32G030

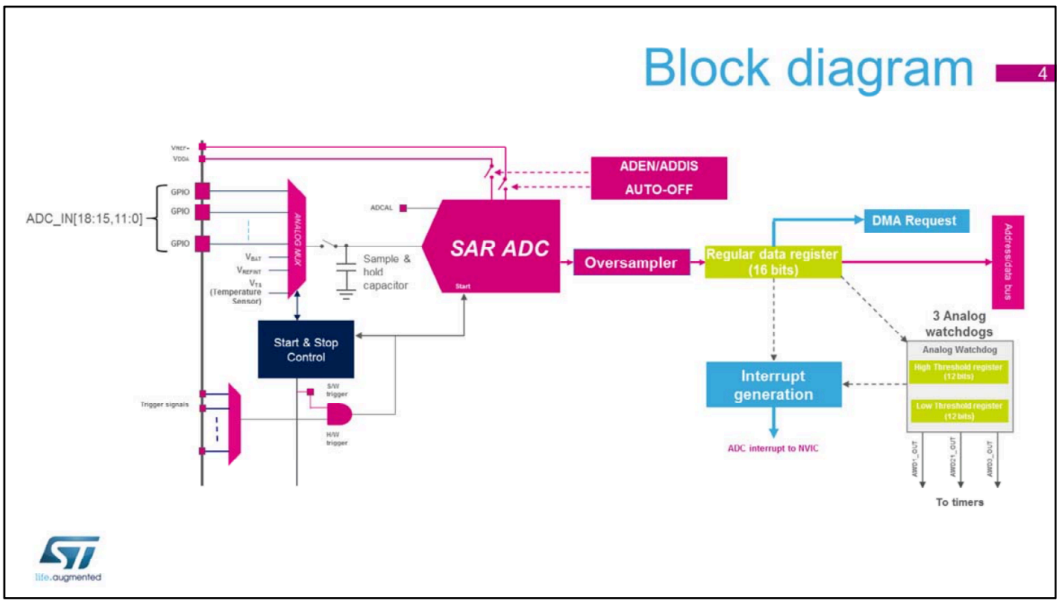
Pro návrh v této semestrální práci byl zvolen mikrořadič **STM32G030** od firmy STMicroelectronics. [1] Tento mikrořadič je vhodný pro aplikace s nízkou spotřebou. Je postavený na 32bitovém jádře ARM Cortex-M0+, které je energeticky efektivní a nabízí dostatečný výkon pro běžné vestavné aplikace. Obsahuje 32 KiB flash paměť a 8 KB SRAM [2].

Pro řadu G030 jsou typické kompaktní rozměry ať už vývojové Nucleo desky, tak typové pouzdra jako například TSSOP20 nebo QFN32, což poskytuje snadnou integraci do kompatního hardwarového návrhu [3].

2.1.1 Analogo-digitální převodník

Mikrokontrolér STM32G030 je vybaven ADC, který obsahuje 8 analogových kanálů o rozlišení 12 bitů. Toto rozlišení poskytuje vysokou přesnost při měření napětí a umožňuje detekci i malých změn v signálu. Maximální vzorkovací frekvence převodníku je 1 MSPS[1].

Při měření kanálů se postupuje sekvenčně, která je určená 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.



Obrázek 1: Blokový diagram AD převodníku

[1] milion vzorků za sekundu

Aby během měření bylo dosaženo maximální přesnosti, převodník podporuje tzn. oversampling[2]. Převodník obsahuje **accumulation data register**, který akumuluje měření a poté pomocí data shifteru vydělí počtem cyklu [4].

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

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 VDDA[3] 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:

$$V_{\text{REF+}} = \frac{V_{\text{REFINT\_CAL}} \times 3000}{V_{\text{REFINT\_ADC\_DATA}}} \tag{2}$$

kde:

- $V_{\text{REFINT\_CAL}}$ je kalibrační hodnota interního referenčního napětí, která je uložená 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 adresy[4][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.

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

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

STM32G0 obsahuje několik časovačů, které se dají využít pro logickou sondu. Mikrořadič má zabudovaných několik základních[5] a jeden advanced timer[6]. Základní timery jsou

[2]Proběhne více měření a následně jsou výsledky např. zprůměrovány aby byla zajištěna větší přesnost.
[3]VDDA je označení pro analogové napájecí napětí v mikrokontrolérech STM32.
[4]Např. u STM32G0 je adresa kalibrační hodnoty: 0x1FFF75AA
[5]Basic Timers

16 bitové a jsou vhodné pro měření doby či generování jednoduchých PWM signálů. Pokročilý časovač je na tomto mikrokontroleru 32bitový a poskytuje více kanálů. Tyto časovače také podporují nejen generování signálů na výstup, ale také zachytávání signálů a měření délky pulzů externího signálu. Pokročilý časovač nabízí řadu nastavení např. nastavování mezi normálním a inverzním výstupem PWM, generovat přerušení při dosažení specifické hodnoty časovače a podobně [6].

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 MHz[7]. Frekvence časovače určuje, jak často časovač inkrementuje svou hodnotu za jednu sekundu [2].

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

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

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

2.1.3 USART

Universal Synchronous Asynchronous Receiver Transmiter je flexibilní periferie, která umožňuje seriovou komunikaci jak v asynchroním tak v synchroním režimu. Pro tuto aplikaci je využíván UART, kde data jsou odesílána bez společného hodinového signálu mezi odesílatelem a přijemcem. Místo toho je podstatný baudrate[9], což určuje počet přenesených bitů za sekundu. UART podporuje nastavení různých protokolů komunikace jako například RS-232 a R-485. UART také umí full duplex komunikaci [7].

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 dat[10].
- **Paritní bity** - Paritní bity slouží k detekci chyby v přenosu. Parita nám dokáže pouze detekovat chybu rámce pouze v případech, kdy nevznikne chyb více[11].
- **Stop bit** - Stop bit[12] signalizuje konec přenosu rámce. Obvykle logická „1“.

[6] Advanced Timers
[7] Frekvenci hodin je možné upravit například pomocí CUBE IDE.
[8] U 16 bitového časovače je maximální perioda 65535.
[9] Rychlost přenosu
[10] Lze používat i 7 bitů nebo 9 bitů dat.
[11] Chyba z dat 1101 na 1000 nelze detekovat, protože lichá parita má paritní bit v obou případech 0.
[12] Stop bitů může být i několik.

Práce využívá periferii USART2, protože je na většině vývojových desek řady STM32, konkrétně STM32G0, připojena k rozhraní STLink, což umožňuje komunikaci přes USB konektor. Díky této konfiguraci je možné jednoduše realizovat sériovou komunikaci prostřednictvím připojení k počítači přes USB pro vývoj [7].== 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.=== Historie Sekvence vnikly jako standardizovaný soubor kódů, který měl za úkol sjednotit různé značky a modely terminálů, které používaly vendor-specific[13] metody, pro tvorbu TUI. Nejznámější terminál, který podporoval ANSI sekvence byl Digital VT100. Jelikož byl velice populární, většina nových terminálů se začaly chovat podle ANSI[14] sekvencí. Následně tyto sekvence byly standardizovány [8].

2.1.4 Kódy

Escape kódy začínají **ESC**[15] znakem, následovným **[**, který značí začátek sekvence, a poté symboly, které určují efekt a celá sekvence je zakončena písmenem.[16]

```
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: Přeškrtnutý

Sekvence také lze použít pro pohyb kurzoru, což je užitečné pro vizuál aplikace [9].

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

[13] Nestandardatizované kódy, který si každá společnost navrhla sama.
[14] Název ANSI byl vytvořen až později. Odpovídá zkratce American National Standards Institute.
[15] \33
[16] Existují také ESC N nebo ESC \ apod.ale tyto se téměř nepoužívají.

2.2 STM HAL

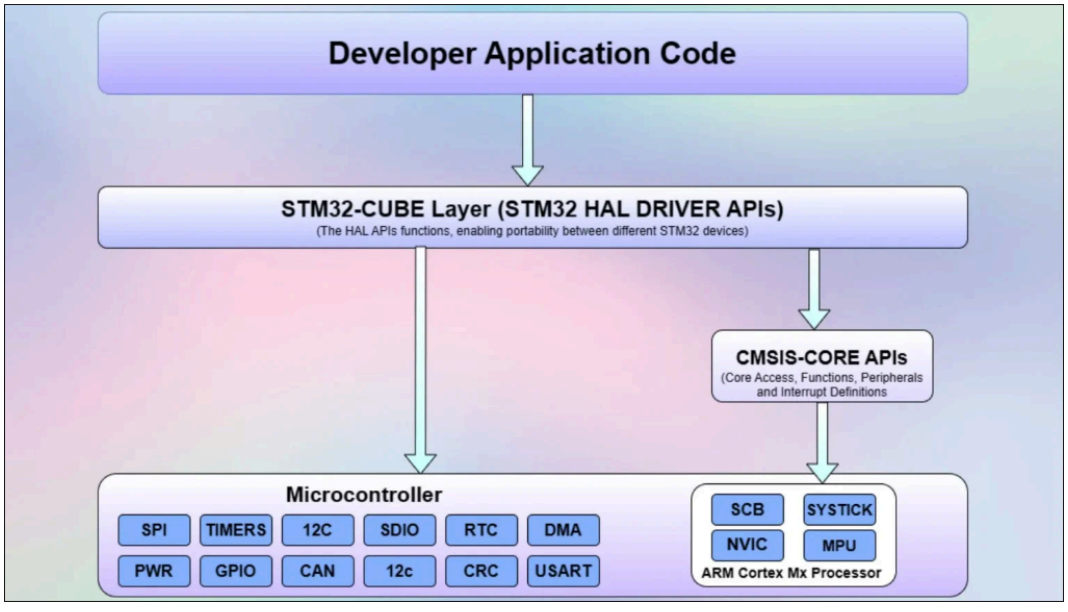
Hardware abstraction layer je knihovna poskytovaná společností STMicroelectronics pro jejich mikrořadiče řady STM32. Tato knihovna tvoří vrstvu abstrakce mezi aplikací a periferiemi mikrokontroléru. Pokytuje funkce na vyšší úrovni, které usnadňují přístup např. k GPIO, USART, SPI, I2C bez nutnosti přímého přístupu k registrům procesoru [10].

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 [11].

Součástí HALu je tzv. CMSIS[17], což je sada standardizovanách rozhraní, které umožňují konfiguraci periferií, správu procesorového jádra, obsluhu přerušení a další [12].

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[18] apod. [12]

Hlavní rozdíl mezi CMSIS a HALu[19] STMicroelectronics je ten, že CMSIS je poskytnuto přímo ARM a slouží pouze na ovládání Cortex M procesorů zatímco část od STMicroelectronics poskytuje abstrakci periferií.



Obrázek 2: STM32CubeMX HAL architektura

[17] Cortex Microcontroller Software Interface Standard
[18] Nested Vectored Interrupt Controller
[19] STMicroelectronics do svého HALu zabaluje i CMSIS od ARM.

Kapitola 3

Realizace

3.1 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 standartní 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**. FOSS[20], 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.

3.1.1 Odesílání zpráv

Před odesláním první zprávy přes seriovou komunikaci je žádoucí inicializovat UART periferii. To je možné přes STM32CubeMX[21], kde vývojář nastaví potřebné parametry a je mu vygenerován základní kód. Pro potřeby projektu bylo zvoleno následující nastavení:

```
static void MX_USART2_UART_Init(void) {
    huart2.Instance = USART2;
    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(&huart2) != HAL_OK) {
        Error_Handler();
```

[20] Free open source software

[21] STM32CubeMX je grafický program, který dává jednodušší možnost úpravy periferií.

```
    }
}
```

Po inicializaci je možné poslat zprávu pomocí například následovně:

```
void ansi_send_string(const char* str) {
    HAL_UART_Transmit(&huart2, (uint8_t*)str, strlen(str), HAL_MAX_DELAY);
}
```

kde reference na `huart2` je inicializovaná struktura. Pro jednodušší generování rozhraní byla vytvořena abstrakce zvaná `ansi abstraction layer` (dále jen AAL). Tato abstrakce umí potřebné ansi senkvence 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);
}
```

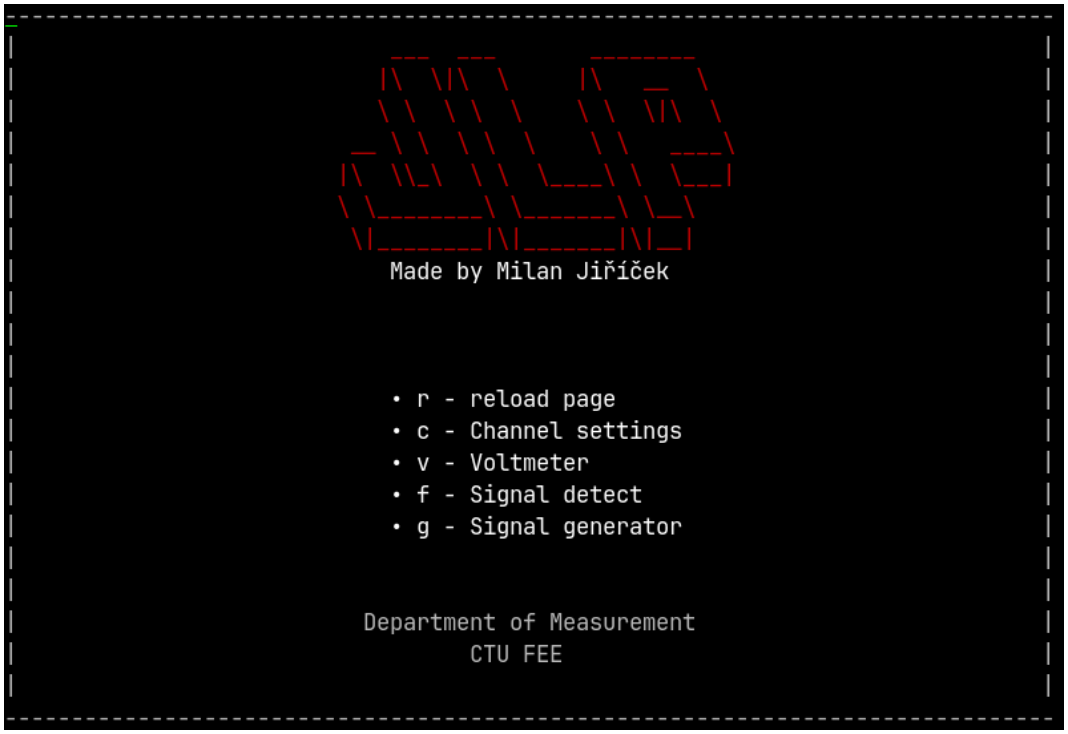
Další funkce, které zjednodušují ovládání jsou k nalezení v souboru `ansi_abstraction_layer.c`. Pro další kontext budou postačovat hlavně tyto dvě funkce.

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 odchytávání signálu:

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

Tyto větší celky ulehčili tvorbu tzv. `ASCII ART`[22], ohraničení nebo tvorby menu, což vylepší vizuál stránky. Obrázek 3 ukazuje, jak vypadá hlavní stránka logické sondy realizované skrze AAL.

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



Obrázek 3: TUI hlavní stránky logické sondy

3.1.2 Přijimání 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 == USART2) {
        HAL_UART_Receive_IT(&huart2, &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 interupt 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.

3.2 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 realné referenční napětí.

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

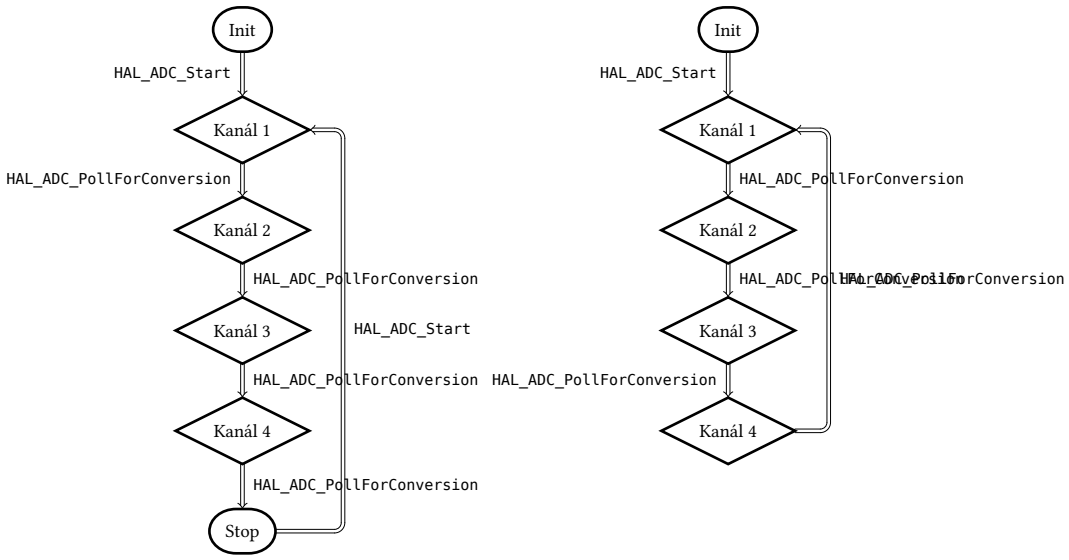
```
static void MX_ADC1_Init(void) {
    ADC_ChannelConfTypeDef sConfig = {0};

    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_SAMPLINGTIME_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[23]. 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`.

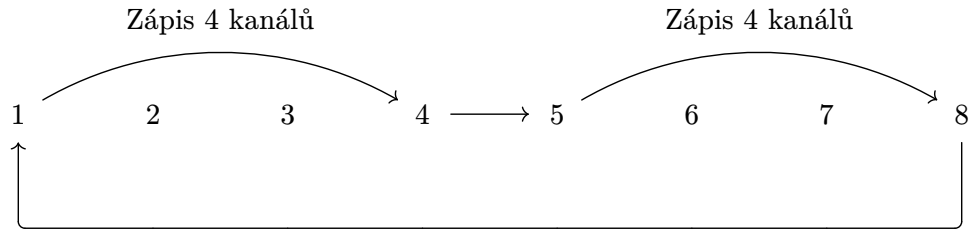
[23]Tato hodnota je v STM32G0 nejvyšší a zaručuje nám tu největší možnou přesnost. Některé mikrořadiče nabízí i vyšší počet odběrů.

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

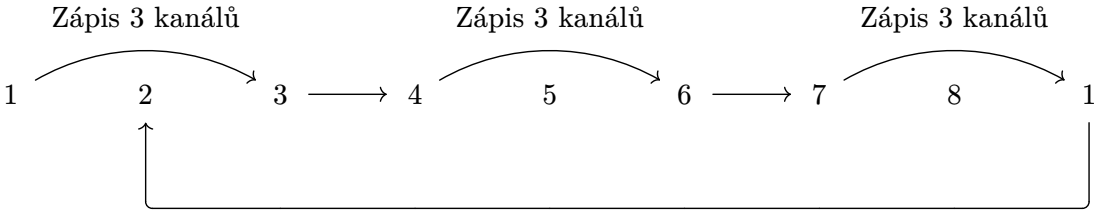
Během měření kanálů je nutné, aby nebyl zbytečně zatěžován procesor. Procesor v hlavní smyčce pracuje na vykreslování dat do terminálu a pokud by prováděl měření, mohlo by to ovlivnit rychlost měření napětí a nemuselo by být dosaženo dostatečnému sebrání vzorků pro plovoucí průměr. Toto je vyřešeno pomocí DMA[24]. HAL aktivuje DMA pro AD převodník pomocí `hadc1.Init.DMAContinuousRequests` nastavené na `ENABLE`. Parametry DMA lze nastavit pomocí STM32CubeMX. DMA je nastaveno jako cirkularní buffer, do kterého se zapisují hodnoty z AD převodníku. Procesor k těmto datům přistoupí, až bude potřebovat.

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

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



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



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

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

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

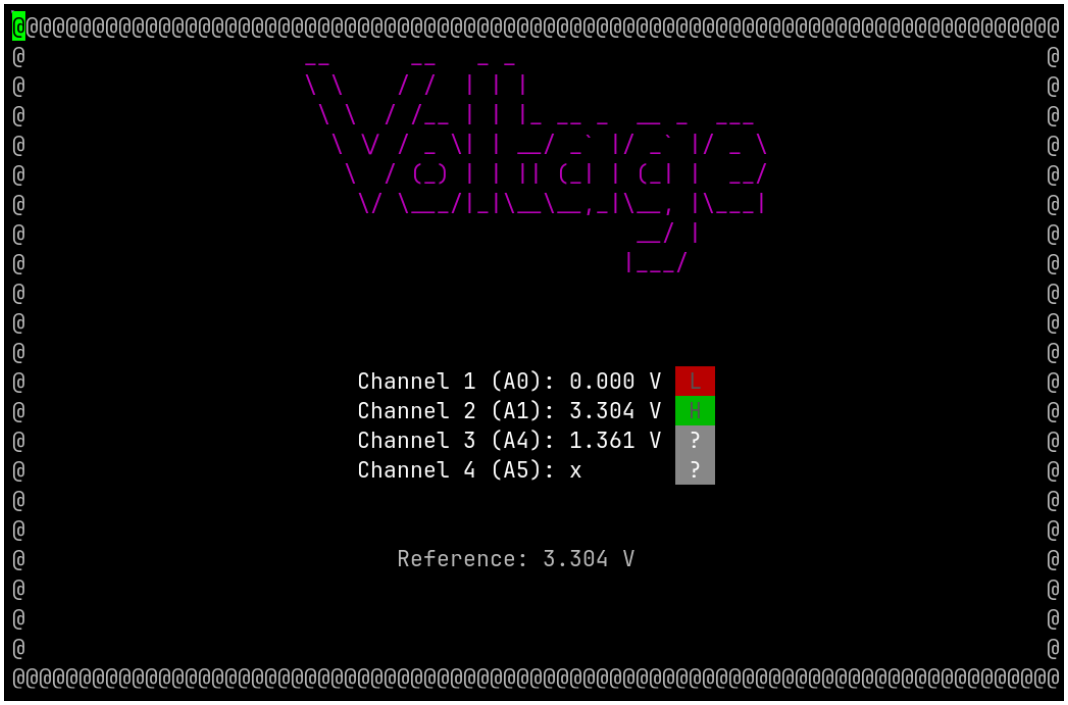
Po určitém časovém úseku, procesor zpracuje data z DMA a zprůměruje hodnoty z AD převodníku, následně hodnoty převede dle metodity v Kapitola 2.1.1 a vykreslí na seriovou linku pomocí ANSI sekvencí zmíněné v Kapitola 3.1.1.

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 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 zkratek 1 až 4 volí jaké kanaly aktivovat, s tím, že po zvolení kanálů je nutné nastavení uložit stisknutím klávesy S.

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

```c
typedef struct {
    _Bool channel[NUM_CHANNELS]; // aktivované kanály
    _Bool channel_unapplied[NUM_CHANNELS]; // kanály neaktivované
    _Bool applied; // bylo nastavení uživatele aplikováno?
    uint32_t avg_last_measure[NUM_CHANNELS]; // poslední průměr hodnot
```



Obrázek 4: Stránka pro měření napětí a logických úrovní

Obrázek 5: Stránka pro nastavení jednotlivých kanálů

```
    unsigned int pin[NUM_CHANNELS]; // čísla pinů
    unsigned int count_active; // počet aktivních pinů
    ADC_HandleTypeDef* hadc; // instance adc pro danou strukturu
} adc_channels;
```

3.3 Odchytání pulzů a frekvence

Pro měření frekvence hraje stěžejní roli časovač. Jak bylo zmíněno v Kapitola 2.1.2, 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 zjistění frekvence musí být kladen důraz na režii. Při odběru dat by nesměl procesor provádět jakoukoliv. Toto se dá realizovat pomocí DMA podobně jako v Kapitola 3.2.

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[25]. K tomuto účelu je použit 32 bitový časovač, aby bylo možné měřit, co největší frekvence.

[25] 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.

$$F = \frac{\frac{N}{2}}{T} \tag{6}$$

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.

```
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.

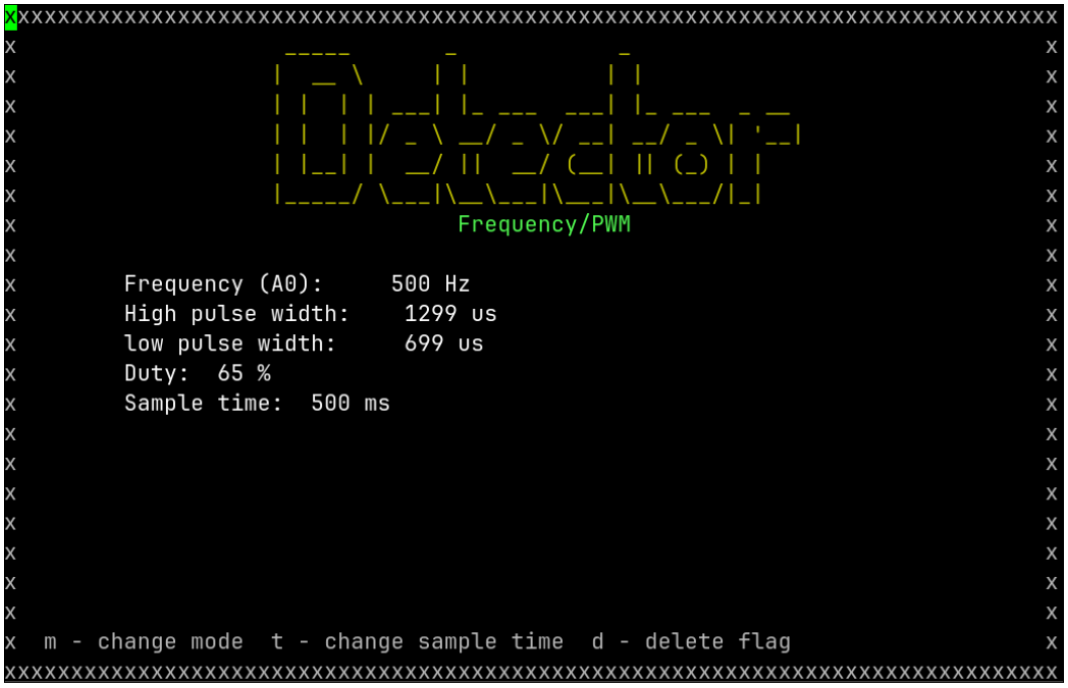
Logická sonda dle Rovnice 6 vypočítá frekvenci. Důvod, proč logická sonda počítá oba pulzy a ne pouze nástupnou nebo sestupnou hranu je ten, že při detekci obou hran dokáž sonda spočítat, jak široké pulzy jsou. Tuto skutečnost je možné využít například pro počítání Duty u PWM signálů.

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;
```

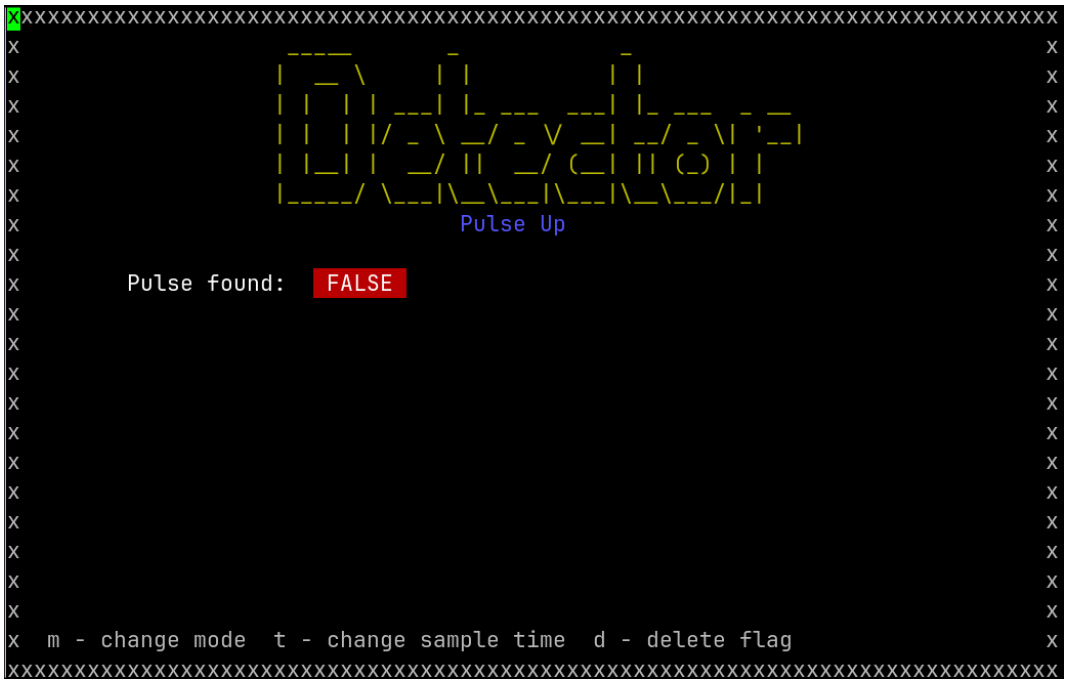
Duty time je poté spočítaný jako poměr času vysokého signálu a nízkého signálu uvedený v procentech.

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



Obrázek 6: Stránka pro měření frekvence a PWM

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



Obrázek 7: Stránka pro hledání pulzů

3.4 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 jednodušše počítal čas v periodě. Poté, co je nastavený časovač, spustí se. Po přetečení časovače se vyvolá přerušení, které značí, že časovač odměřil příslušnou dobu. Přerušení následně časovač zastaví.

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

Díky těmto režimům je možné i nastavit úrovně, které uživatel nemusí nutně používat jako generování pulzů, ale jako přepínání úrovní dle potřeby.

```
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 4

Závěr a zhodnocení

Většina témat této práce, jsou pro mě nová. Na počátku této práce mé znalosti ohledně mikrořadičů byly velice povrchní a pouze na takové úrovni, které mi umožňovaly pracovat s mikrokontrolérem pomocí knihoven a vysoké míry abstrakce.

Tento projekt, vyžadoval, abych se naučil spoustu nových věcí a principy fungování mikrořadiče nevnímal pouze jako „funkci, která mi vrátí výsledek“. Nyní vnímám, že mikrořadič není pouze černá skříňka, ale pod povrchem je to chytře navržený stroj, který dává neskutečné možnosti, jak s ním pracovat. Má cesta v tomto odvětví je stále na začátku, ale jsem zapálen pokračovat a učit se novým věcem.

Tato práce je počátek logické sondy pro střední a vysoké školy, které mají za úkol ulehčit výuku a diagnostiku. Hlavní část této práce pro mě byla zanalyzovat problematiku a najít vhodné řešení. Logická sonda již dnes lze použít na jednoduchou diagnostiku. Logická sonda má funkční TUI, které je realizované pomocí ANSI sekvencí, umí měřit napětí na 4 kanálech, umí určovat frekvenci na vstupu, umí zachytávat krátké pulzy ať jsou nízko úrovňové nebo vysoko úrovňové a dokáže i generovat krátké pulzy. Proto hodnotím výsledek práce jako úspěch.

Tímto ale práce na projektu neskončila. Na projektu plánuji pracovat v rámci bakalářské práce a rozvinout její potenciál a nakonec i naportovat na další mikrořadiče.

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

[8] T. F. E. Wikipedia, „ANSI escape code". [Online]. Dostupné z: https://en.wikipedia.org/wiki/ANSI_escape_code

[9] fnky, „ANSI Escape Sequences". [Online]. Dostupné z: https://gist.github.com/fnky/458719343aabd01cfb17a3a4f7296797

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

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

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