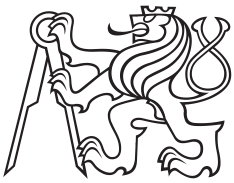
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra měření

Univerzální čítač s STM32G031 pro laboratorní experimety

Ondřej Hloušek

Vedoucí: doc. Ing. Jan Fischer, CSc.
Květen 2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Hloušek** Jméno: **Ondřej** Osobní číslo: **499220**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra měření**

Studijní program: **Kybernetika a robotika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Univerzální čítač s STM32G031 pro laboratorní experimety

Název bakalářské práce anglicky:

Universal counter with STM32G031 for laboratory experiments

Pokyny pro vypracování:

Navrhněte a s mikrořadičem STM32G031 realizujte univerzální čítač, jehož vlastnosti budou orientovány na využití při školních laboratorních experimentech. Čítač umožní přímé i recipročí měření frekvence, měření délky impulsů a jejich periody. Mezi další funkce bude patřit průměrování, měření z více period, dvoukanálové měření, prosté čítání impulsů s hradlováním, zobrazení průběhu měřeného analogového i logického signálu. Pro ovládání přístroje a zobrazení výsledků využijte PC s aplikací Dataplotter. Navrhněte obvody pro potřebnou úpravu vstupních signálů a také obvody pro generaci referenčního hodinového signálu řízeného krystalem. Ověřte funkci výsledného přístroje a posuďte jeho parametry.

Seznam doporučené literatury:

[1] Yiu, J.: The Definitive Guide to ARM Cortex -M0 and Cortex-M0+ processors; 2015
[2] STMicroelectronics: STM32G0x1 RM0444 Reference manual; 2020
[3] Maier, J.: Univerzální GUI pro osciloskopické PC aplikace; ČVUT- FEL, 2021

Jméno a pracoviště vedoucí(ho) bakalářské práce:

doc. Ing. Jan Fischer, CSc. katedra měření FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **14.02.2023** Termín odevzdání bakalářské práce: _____

Platnost zadání bakalářské práce: **22.09.2024**

doc. Ing. Jan Fischer, CSc. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____ _____

Datum převzetí zadání Podpis studenta

Poděkování

Chtěl bych tímto poděkovat vedoucímu mojí bakalářské práce, panu doc. Ing. Janu Fischerovi, CSc., za ochotu a čas, který mi při konzultacích k této práci věnoval. Také za věcné připomínky a poskytnutí materiálů a rad k úspěšnému vyřešení všech technických komplikací s touto prací spojených.

Prohlášení

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

V Praze dne 25 května 2023

.....
Ondřej Hloušek

Abstrakt

Tato práce se zabývá návrhem a realizací softwarově definovaného čítače pro použití ve výukových laboratorních experimentech. Vytvářený měřicí přístroj umožňuje měření a zobrazení analogového signálu v režimu osciloskopu a logického signálu v režimu logického analyzátoru, dále umožňuje přímé i reciproční měření frekvence, periody i střídy signálu, dále pak čas mezi dvěma příchozími událostmi, v podobě hran, z různých zdrojů a záznam časů mezi několika po sobě jdoucími hranami. Přístroj je dále schopen generovat uživatelsky definovaný PWM signál a je možné ho provozovat s vnějším zdrojem hodinového signálu. Měřicí přístroj je realizován na mikroprocesoru STM32G031, který pomocí USB-UART převodníku komunikuje přes rozhraní USB s aplikací Data Plotter v PC. V aplikaci je nejen vytvořeno uživatelské rozhraní, přes které je měřicí přístroj ovládán, ale jsou zde i zobrazovány naměřené výsledky s možností pozdějšího zpracování naměřených dat.

Klíčová slova: čítač, softwarově definovaný přístroj, virtuální přístroj, stm32 mikrokontroler

Vedoucí: doc. Ing. Jan Fischer, CSc.

Abstract

This thesis is focused od design and implementation of software-defined counter, for use in teaching laboratory experiments. The created measuring device enables the measurement and display of an analog signal in the oscilloscope mode and a logic signal in the logic analyzer mode, it also enables direct and reciprocal measurement of the frequency, period and duty cycle of the signal, as well as the time between two incoming events, in the form of edges, from various sources and recording times between several consecutive edges. The device is also able to generate a user-defined PWM signal and it is possible to connect it to an external clock signal source. The measuring device is implemented on an STM32G031 microprocessor, which uses a USB-UART converter to communicate via the USB interface with the Data Plotter application on a PC. The application not only creates a user interface through which the measuring device is controlled, but also displays the measured results with the possibility of postprocessing.

Keywords: counter, software-defined instrument, virtual instrument, stm32 microcontroller

x

Title translation: Universal counter with STM32G031 for laboratory experiments

Obsah

Obrázky

Tabulky

Kapitola 1
Úvod

Námětem pro vznik této bakalářské práce bylo vytvoření univerzálního měřicího přístroje použitelného zejména při výukových experimentech s potřebou přesného měření času. Pro tyto účely se používá čítač, ten ovšem nemusí být součástí výbavy každé školní laboratoře a díky jeho nemalé pořizovací ceně jsou často školy od jeho pořízení odrazeny. Další nevýhodou takového přístroje bývá nutnost mít i další podpůrné vybavení, jako je například osciloskop. Ovšem díky vytvoření čítače a jeho podpůrných měřicích přístrojů ve formě virtuálního přístroje lze nejen dosáhnout nízké ceny při zachování dostatečné přesnosti měření, ale i zaručit kompaktnost a dostupnost přístroje i pro soukromé účely. Virtuálním přístrojem pak v principu rozumíme mikrokontroler, kde jsou jednotlivé měřicí přístroje realizovány software-ově a to za pomocí mnohých periferií, jako jsou čítače, ADC, DMA nebo SPI.

Tato konstrukce bude velice úsporná z hlediska množství použitých součástek a díky poměrně nízké ceně mikrokontrolerů i cenově dostupná. Pro zobrazení naměřených dat bude přístroj používat aplikaci ve stolním PC případně notebooku, který je již dnes běžnou součástí výukových laboratoří a učeben. Výsledný experiment tak bude potřebovat pouze počítač, nepájivé pole se zapojeným mikrokontrolerem a případnými podpůrnými obvody, převodník zprostředkující komunikaci mezi počítačem a mikrokontrolerem, a nakonec už jen senzory převádějící měřenou fyzikální veličinu na napětí. V této práci se pak budeme zabývat i návrhem zapojení různých senzorů pro fyzikální experimenty a několika příklady možných experimentů s tímto měřicím přístrojem. Bude zde kladen důraz na jednoduchou použitelnost, univerzálnost přístroje a názornou ukázku možností využití vytvářeného měřicího přístroje při fyzikálních výukových experimentech.

Kapitola 2
Rozbor

2.1 Motivace

Experimenty ověřující platnost fyzikálních zákonů a matematických vztahů, které je popisují, jsou často náročné na různorodost a přesnost měřicích přístrojů. Tyto přístroje jsou také většinou drahé, těžké, jejich obsluha je komplikovaná a často bývá obtížné zachovat zapojení experimentu při jeho přemisťování. A tyto důvody pak mohou pedagogy a studenty od experimentování odrazovat. Ovšem pro výukové účely by postačoval měřicí přístroj s menší přesností zato, kompaktní, levný, jednoduchý na použití, bez potřeby napájení ze síťového zdroje, s možností přenosného zapojení experimentu na kontaktním nepájivém poli a s jednoduchým zobrazením měřeného experimentu na obrazovce osobního počítače nebo projektoru.

Řada zajímavých "pohyblivých" experimentů pak vyžaduje přesné měření času, například měření gravitačního zrychlení, měření rychlosti zvuku, ověření platnosti matematického popisu kyvadla nebo měření tepové frekvence lidského srdce. Pro tato měření se hodí čítač, který dokáže měřit čas, frekvenci případně periodu a střídu periodických signálů, dále pak je schopen čítat neperiodicky přicházející události a měřit časový interval mezi nimi čehož se dá využít například při měření rychlosti. Čítače ovšem často nebývají standartním vybavením středoškolských fyzikálních laboratoří, ať už kvůli jejich ceně nebo zdánlivě úzkému rozsahu použití a nutnosti pořídit si společně s čítačem i podpůrné měřicí přístroje, například osciloskop nebo logický analyzátor, pro prvotní nastavení zdroje měřeného signálu.

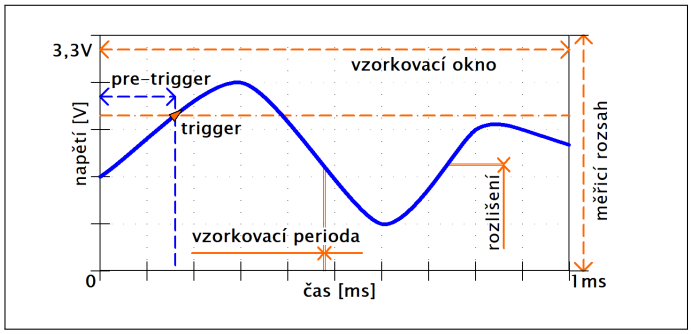
V této bakalářské práci tedy vznikne snadno použitelný, cenově dostupný čítač využitelný zejména ve fyzikálních experimentech s potřebou přesného měření časových veličin. Měřicí přístroj bude založen na mikroprocesoru STM32G031 s jádrem ARM Cortex-M0$^+$, u kterého se zaměříme na realizaci v jeho nejmenším osmi pinovém pouzdře SO8N. Současně vznikne návrh podpůrných obvodů pro zapojení s různými senzory fyzikálních veličin a příklady experimentů, které bude možné s tímto jednoduchým přístrojem uskutečnit. Veškerá zapojení a experimenty budou tvořeny s minimálním nárokem na podpůrné součástky a takovým způsobem, aby je bylo možné použít jako návod k realizaci navržených experimentů i bez větších zkušeností s elektrotechnikou.

2.2 Požadované funkce měřicího přístroje

2.2.1 Osciloskop

Osciloskop využijeme, pro prvotní měření a zobrazení časového průběhu napětí měřeného signálu nebo jako prostý voltmetr, což často potřebujeme pro naladění obvodu případně polohy senzoru měřené

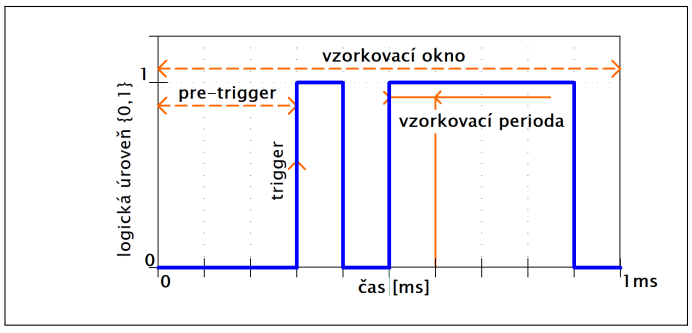
veličiny pro další zpracování čítačem. Funkce osciloskopu spočívá v periodickém měření (vzorkování) napětí měřeného signálu zadanou vzorkovací frekvencí, potažmo vzorkovací periodou. Počet vzorků, které dokáže osciloskop souvisle naměřit a uložit do paměti, pak budeme nazývat vzorkovací okno. Pro započetí vzorkování vzorkovacího okna bude možno použít trigger podmínky, a to překročení nastavené napěťové úrovně měřeného signálu, buď z nižšího napětí na vyšší (náběžná hrana) nebo z vyššího napětí na nižší (spádová hrana). Počet vzorků zaznamenaných před trigger událostí pak budeme nazývat pre-trigger. Dalším parametrem, který nás bude u osciloskopu pro náš měřicí přístroj zajímat, je rozlišení, tedy nejmenší dílek napětí, který ještě dokáže A-D (analogově digitální) převodník na vstupu osciloskopu rozlišit. Důležitou vlastnosti A-D převodníku pak bude ještě rozsah vstupního napětí, které náš osciloskop dokáže měřit.



Obrázek 2.1: Příklad zobrazení naměřeného vzorkovacího okna osciloskopem

2.2.2 Logický analyzátor

Funkce logického analyzátoru je obdobná funkci osciloskopu, ovšem logický analyzátor dokáže rozlišit pouze dvě napěťové úrovně, a to logickou nulu (log.0) a logickou jedničku (log.1). Tento měřicí přístroj využijeme k zobrazení časového průběhu logického signálu, tedy signálu, který budeme později měřit čítačem, který také rozlišuje logické úrovně stejně, jako logický analyzátor. Stejně jako u osciloskopu nás při konstrukci logického analyzátoru bude zajímat velikost vzorkovacího okna, vzorkovací perioda, velikost pre-trigger a trigger podmínky, které jsou zde opět buď náběžná hrana (přechod z log.0 na log.1) nebo sestupná hrana (přechod z log.1 na log.0). Napěťové úrovně rozlišující log.0 od log.1 potom budou dány konkrétně použitým napájecím napětím mikrokontroleru.

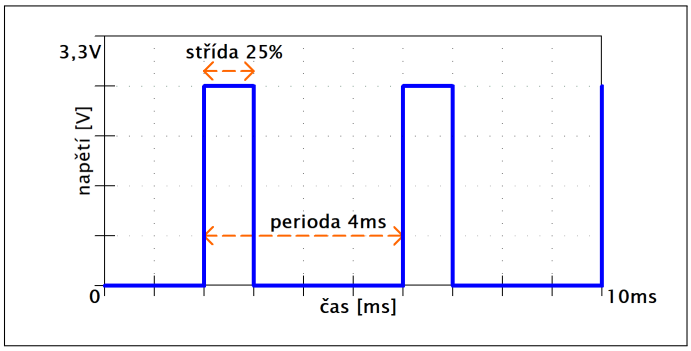


Obrázek 2.2: Příklad zobrazení naměřeného vzorkovacího okna logickým analyzátorem

2.2.3 PWM Generátor

Další částí měřicího přístroje pak bude generátor PWM (pulzně šířková modulace) signálu, který bude zdrojem měřicího signálu, například při měření rychlosti zvuku, a který současně zajistí jednoduché otestování některých zapojení a konkrétních funkcí čítače. Generátor PWM bude generovat logický

signál, tedy signál nabývající pouze dvou napěťových úrovní, a to nula volt pro log.0 a kladného napětí pro log.1 daného napájecím napětím mikrokontroleru typicky 3,3 V. Dalšími parametry PWM signálu jsou jeho frekvence(perioda) a střída, kde střída udává poměr mezi log.1 a log.0 v jedné periodě, často je ale udávána jako procento času z periody, kde signál nabývá log.1. Příklad časového průběhu na výstupu PWM generátoru vidíme na obrázku 2.3.



Obrázek 2.3: Příklad signálu na výstupu PWM generátoru

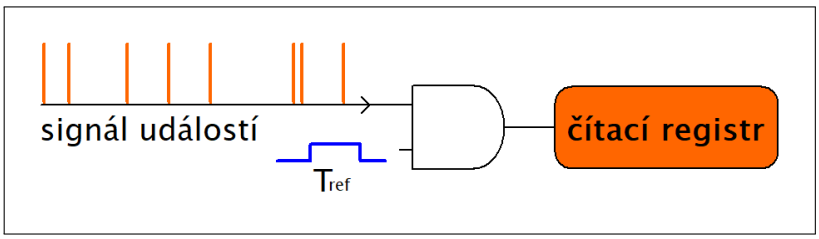
2.2.4 Čítač

Základní princip

Funkce čítače bude pro konstruovaný měřicí přístroj zásadní, proto se na ni také více zaměříme. Čítač je v principu měřicí přístroj určený pro měření časově orientovaných veličin, jako je počet událostí za konkrétní čas nebo frekvence a střída periodického logického signálu. Základní princip všech pracovních režimů je založen na čítacím registru, který zaznamenává každou událost, náběžnou nebo sestupnou hranu logického signálu, a na jejich základě mění hodnotu v registru, typicky k ní přičítá nebo odečítá jedničku.

Pracovní režimy

- **Prosté čítání událostí:**
  V tomto režimu je pomocí hradla na vstup čítacího registru na určitý časový interval T_{ref}[s] přiváděn signál s měřenými událostmi, jak vidíme na obrázku 2.4. Čítač začíná s nulovou hodnotou ve svém čítacím registru a každou příchozí událostí je stav registru inkrementován o jedna. Na konci měřicího okna T_{ref} tak bude v čítacím registru hodnota N odpovídající počtu příchozích událostí za tuto dobu.



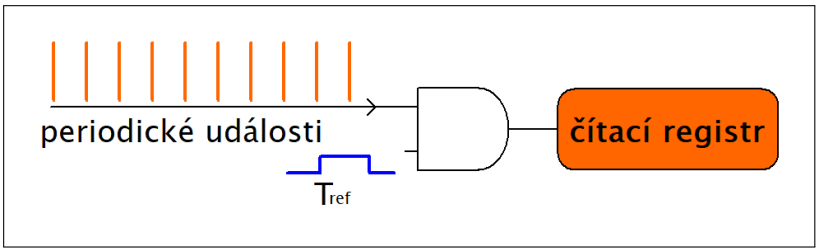
Obrázek 2.4: Schéma čítače v režimu prostého čítání událostí

- **Přímé měření frekvence:**
  Měřit frekvenci událostí v našem případě frekvenci náběžných/sestupných hran periodického

logického signálu lze stejným principem, jako prosté čítání pulzů za určitý čas, jak je naznačeno na obrázku 2.5. Ovšem tentokrát pro přesné měření frekvence musíme zaručit aby:

- měřený signál byl periodický,
- měřená perioda, tedy převrácená hodnota měřené frekvence, byla celočíselným násobkem měřicího okna T_{ref}.

První podmínka nám říká, že bude tato metoda nevhodná pro signály s rychle měnící se frekvencí. Druhou podmínku zaručit u obeckého signálu nemůžeme, ovšem prodloužením měřicího okna T_{ref} dokážeme minimalizovat vliv chyby vnesené nesplněním této podmínky, ovšem musíme zaručit, aby po dobu měření T_{ref} byl signál stále periodický.



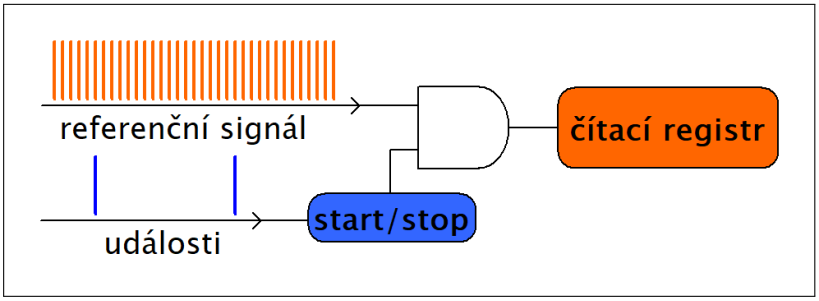
Obrázek 2.5: Schéma čítače v režimu přímého měření frekvence

Změřenou frekvenci pak spočítáme, jako:

$$f = \frac{N}{T_{ref}}, \tag{2.1}$$

kde N bude počet událostí napočtený čítacím registrem za dobu T_{ref}.

- **Měření periody, reciproční měření frekvence:**
  Tato metoda vyžívá časové základny v podobě referenčního signálu o známé frekvenci f_{ref}[Hz] s frekvencí několikanásobně vyšší, než je frekvence měřená. Princip je v podstatě opačný než v předchozí metodě měření frekvence, tedy tentokrát načítáme počet událostí časové základny N za neznámou dobu T mezi dvěma událostmi. Principiální schéma je pak zobrazeno na obrázku 2.6.



Obrázek 2.6: Schéma čítače v režimu měření periody

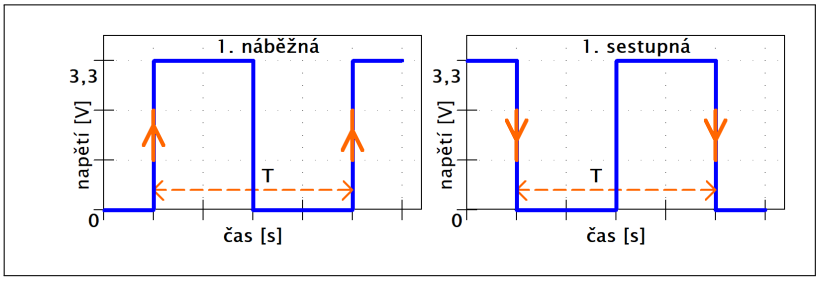
První událost tedy započne čítání referenčního signálu o známé frekvenci f_{ref} a druhá událost toto čítání zastaví. Neznámou dobu T pak spočteme, jako:

$$T = \frac{N}{f_{ref}}. \tag{2.2}$$

Recipročně změřená frekvence $f_{rec} = \frac{1}{T}$ pak odpovídá frekvenci periodicky přicházejících událostí. Pro přesné změření touto metodou bychom museli zaručit, že neznámá perioda T bude celočíselným násobkem referenční periody ($T_{ref} = \frac{1}{f_{ref}}$). To však zaručit nedokážeme, ovšem lze tuto chybu zmenšovat zvyšováním referenční frekvence.

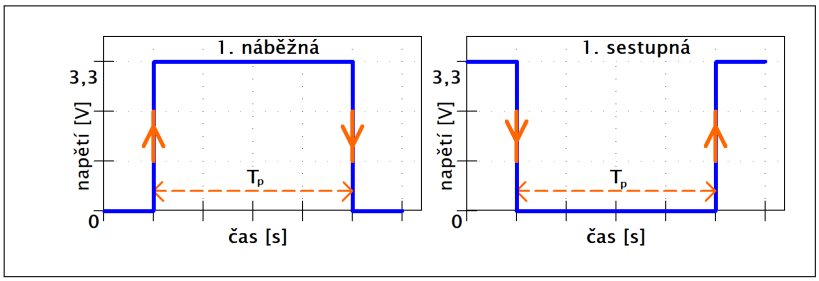
V případě mnoha fyzikálních pokusů však události periodicky přicházet nebudou a bude nás tak zajímat pouze časový odstup prvních dvou, či několika dalších událostí. Hledaný časový odstup pak bude měřen následujícími režimy:

- **Dvě po sobě příchozí události stejného charakteru**
  Tedy hledáme časový odstup mezi první náběžnou [sestupnou] hranou a druhou náběžnou [sestupnou] hranou signálu, jak vidíme na obrázku 2.7, což využijeme v situaci, kdy budeme chtít měřit pouze první příchozí periodu nějakého periodického, odeznívajícího signálu. Například u měření rezonanční frekvence paralelní kombinace L-C (indukčnost, kapacita), kdy je signál v obvodu po jeho vybuzení k rezonanci rychle utlumen a jsme tak schopni měřit jen prvních několik period.



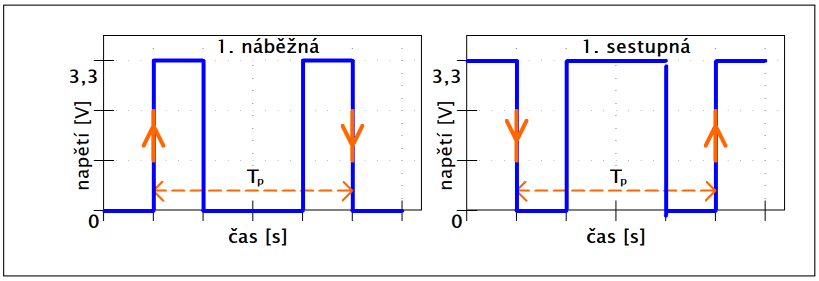
Obrázek 2.7: Časový odstup mezi první náběžnou [sestupnou] hranou a druhou náběžnou [sestupnou] hranou

- **Dvě po sobě jdoucí události rozdílného charakteru**
  Tentokrát hledáme šířku kladného [záporného] pulzu, kdy měříme časový odstup mezi první náběžnou [sestupnou] hranou a první sestupnou [náběžnou] hranou, jak vidíme na obrázku 2.8. Využití tohoto principu pak nalezneme například při měření doby působení nějaké veličiny na senzor, jako je zaclonění optického senzoru pohybujícím se tělesem.



Obrázek 2.8: Šířka kladného [záporného] pulzu

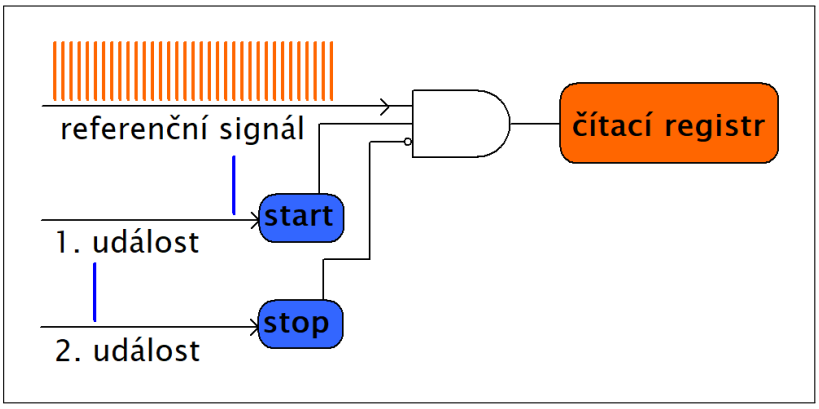
- **Dvě události rozdílného charakteru jdoucích s odstupem jedné události**
  Opět začneme čas měřit příchodem náběžné [sestupné] hrany, ovšem tentokrát končíme měření až po příchodu druhé sestupné [náběžné] hrany, jak je znázorněno na obrázku 2.9. Tento specifický případ nastane, kdy budeme měřit dvěma senzory vhodně spojenými do jednoho výstupu, čas, po který bude měřená veličina působit alespoň na jeden ze senzorů.



Obrázek 2.9: Časový odstup mezi první náběžnou [sestupnou] hranou a druhou sestupnou [náběžnou] hranou

- **Záznam několika po sobě jdoucích událostí**
  Ačkoliv je startovací podmínkou u všech předchozích způsobů měření náběžná [sestupná] hrana, ukončovací podmínka se pro různé aplikace liší a předešlé tři režimy nedokážou všechny specifické způsoby ukončení měření pokrýt. Tedy pro ukončení měření na třetí, čtvrtou, pátou, ... náběžnou [sestupnou] hranu budeme využívat univerzálního režimu, kdy začneme měření náběžnou [sestupnou] hranou a čas příchodu každé události si zaznamenáme, a to až do příchodu N-té události, kde celkový zaznamenaný počet událostí N bude nastavitelný. V případě menších nároků na přesnost a potřeby měření delšího záznamu, pak může být tento režim principiálně nahrazen výstupem logického analyzátoru viz. 2.2.2, ovšem díky omezeným možnostem našeho mikrokontroleru nedokážeme u logického analyzátoru dosáhnout takového rozlišení, jako u čítače v režimu záznamu.

- **Měření časového odstupu dvou událostí z různých zdrojů:**
  Princip této metody je stejný, jako měření periody popsané v minulých odstavcích, ovšem tentokrát je startovací a ukončovací událost každá z jiného zdroje. Tedy jak vidíme na obrázku 2.10 příchod události z prvního zdroje spustí čítání referenčního signálu a událost ze druhého zdroje toto čítání zastaví. Výsledný časový odstup T těchto událostí spočteme pomocí vzorce 2.2.

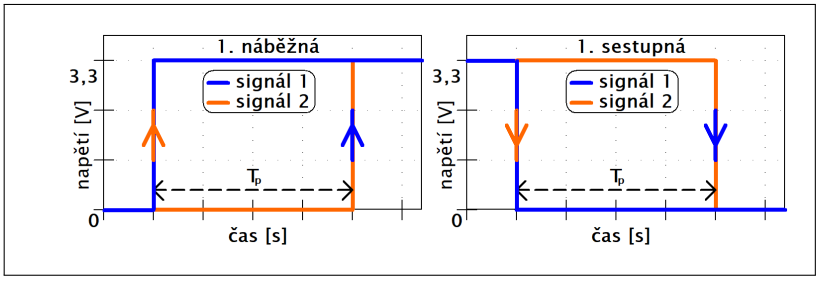


Obrázek 2.10: Schéma čítače v režimu měření časového odstupu dvou událostí z různých zdrojů

Tento režim čítače využijeme typicky při měření rychlosti pohybu tělesa dvěma senzory polohy, kdy zaznamenání měřeného tělesa prvním senzorem způsobí první událost a zaznamenání tělesa druhým senzorem způsobí druhou událost, kdy rychlost tělesa dopočteme ze změřeného časového odstupu těchto dvou událostí a známé vzdálenosti senzorů. Měřicí režimy podle typu událostí pak budou následující:

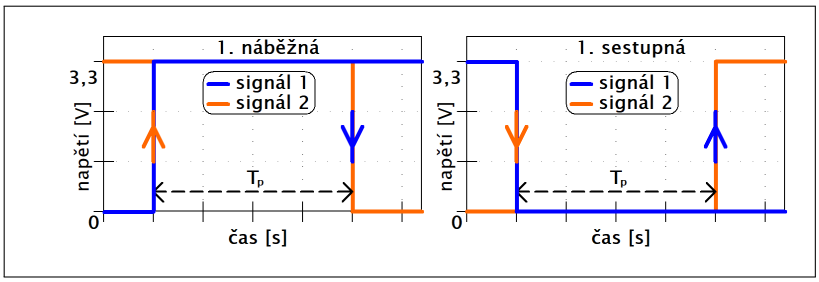
- **Obě události stejného charakteru**
  Měření v tomto režimu začneme na náběžnou [sestupnou] hranu prvního signálu a skončíme

na náběžnou [sestupnou] hranu druhého signálu, jak vidíme na obrázku 2.11. Tento režim bude využit pro měření s oběma senzory stejného charakteru, tedy oba budou reagovat na zaznamenání veličiny náběžnou [sestupnou] hranou.



Obrázek 2.11: Časový odstup mezi náběžnou [sestupnou] hranou 1. signálu a náběžnou [sestupnou] hranou 2. signálu

- **Obě události opačného charakteru**
  Měření opět začíná náběžnou [sestupnou] hranou prvního signálu, ovšem tentokrát skončí na sestupnou [náběžnou] hranu druhého signálu, jak je znázorněno na obrázku 2.12. Tento režim využijeme pro měření se senzory opačného charakteru, tedy jeden bude reagovat na zaznamenání veličiny náběžnou [sestupnou] hranou a druhý opačnou událostí, a to sestupnou [náběžnou] hranou.



Obrázek 2.12: Časový odstup mezi náběžnou [sestupnou] hranou 1. signálu a sestupnou [náběžnou] hranou 2. signálu

Zdroj referenčního signálu

Pro všechna měření čítačem se používá referenční signál, tedy buď pulz o známé šířce nebo periodický signál o známé frekvenci. Od tohoto časového signálu se pak odvozuje naměřená hodnota časové veličiny, je tedy žádoucí, aby byl referenční signál co možná nejpřesnější, jelikož na něm bude záviset i výsledná přesnost měření. Přesnost referenčního signálu pak bude v našem mikrokontroleru závislá na vnitřních RC hodinách, které nevykazují takovou teplotní stálost, jako oscilátory založené na mechanickém kmitání krystalu. Pro náročnější měřicí aplikace tak bude vhodné připojit vnější, přesnější, krystalový oscilátor. Vznikne tedy i návrh jednoduchého zapojení takového oscilátoru.

Průměrování

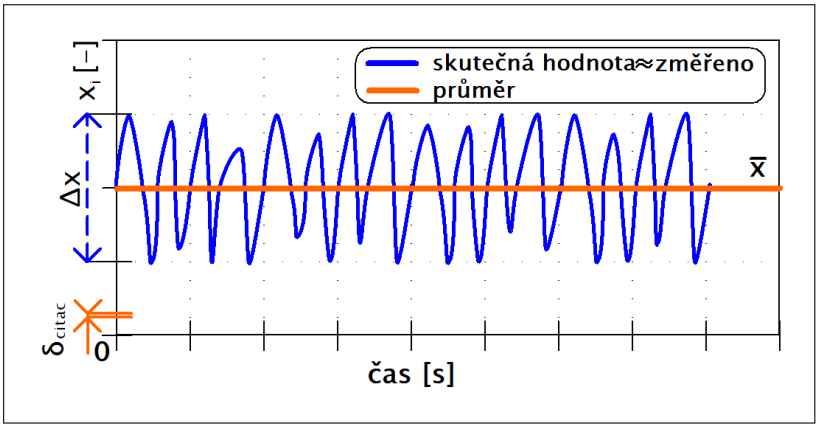
Důležitou funkcionalitou čítače bude průměrovaní z několika po sobě naměřených hodnot, a to především u těch pracovních režimů čítače, kde používáme referenční frekvenci, viz. režimy čítače 2.2.4. U režimů s referenčním časovým oknem můžeme kýženého efektu průměrování dosáhnout prodloužením časového okna což lze dělat téměř libovolně, ovšem u režimů s referenční frekvencí by to znamenalo zvyšovat

referenční frekvenci a to již brzo naráží na konstrukční limity mikrokontroleru. Při průměrování pak budeme rozlišovat tyto případy:

- Kolísání hodnoty měřeného signálu bude mnohem **větší** než rozlišení čítače $\Delta x >> \delta_{citac}$
  Tento případ typicky nastává, když přesným měřicím přístrojem měříme signál zatížený poměrně velkým šumem, jak vidíme na obrázku 2.13. Průměrováním N naměřených hodnot $x_1, \ldots, x_N$ podle vzorce:

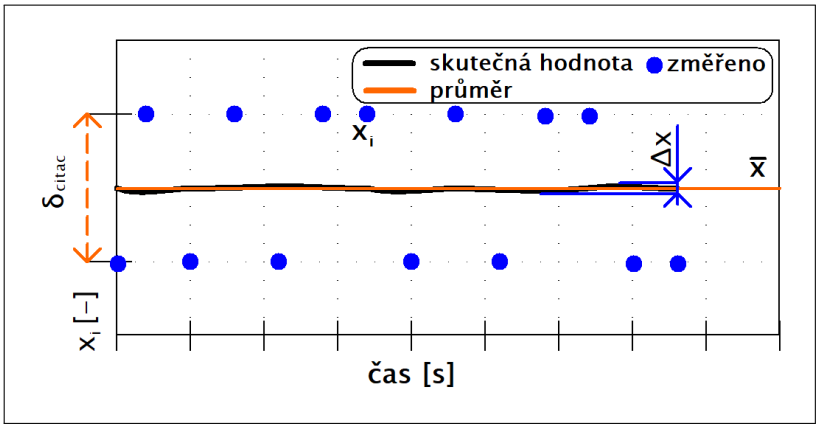
$$\overline{x} = \frac{1}{N} \sum_{i=1}^{N} x_i, \tag{2.3}$$

  pak dostaneme střední hodnotu naměřené veličiny $\overline{x}$, která představuje konkrétní hodnotu, okolo které měřený signál díky šumu kmitá.



Obrázek 2.13: Průměrná hodnota signálu zatíženého šumem

- Kolísání hodnoty měřeného signálu bude mnohem **menší** než rozlišení čítače $\Delta x << \delta_{citac}$
  Tento případ typicky nastává, když nějaký časově stálý signál měříme přístrojem s omezeným rozlišením a hodnota naměřené veličiny x_i "skáče" mezi dvěma hodnotovými hladinami, jak vidíme na obrázku 2.14. Průměrováním N naměřených hodnot $x_1, \ldots, x_N$ podle vzorce 2.3 pak dokážeme zvyšovat "rozlišení přístroje", tedy principielně z poměru naměřených hodnot nad a pod skutečnou hodnotou dokážeme průměrem přibližně dopočíst hodnotu skutečnou. Příklad takového výpočtu vidíme na obrázku 2.14, kde jsme naměřily 7 vzorků na hodnotové hladině vyšší, než skutečná hodnota a 7 vzorků na hodnotové hladině nižší, než skutečná hodnota. Po vyhodnocení průměru $\overline{x}$ z těchto vzorků dostáváme hodnotu uprostřed hodnotových hladin, tedy takovou hodnotu, kterou náš přístroj již nedokáže rozlišit a vidíme, že je průměrná hodnota blíže hodnotě skutečné než kterákoli naměřená hodnota x_i.



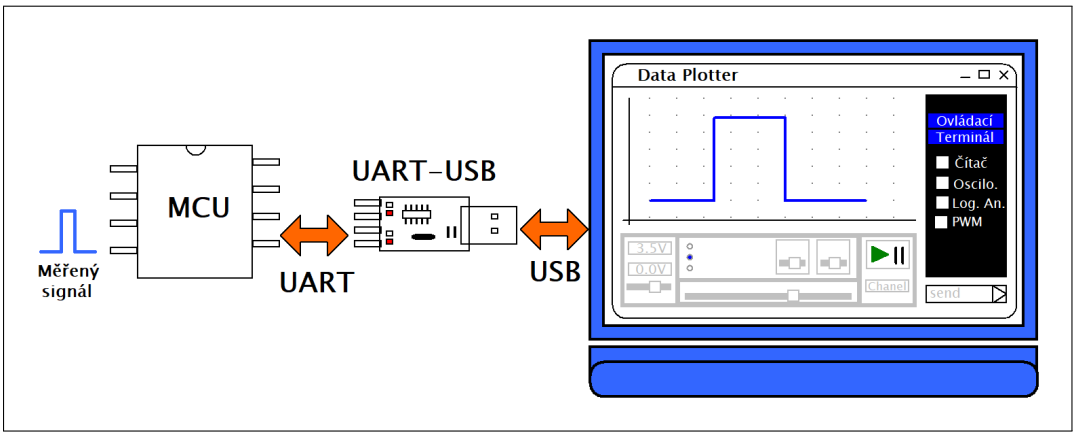
Obrázek 2.14: Průměrná hodnota signálu nezatíženého šumem

- Kolísání hodnoty měřeného signálu bude **srovnatelné** s rozlišením čítače $\Delta x \approx \delta_{citac}$
  V tomto případě bude průměrování naměřených hodnot částečně eliminovat vliv naindukovaného rušivého napětí v signálu na měření a částečně zvyšovat "rozlišení přístroje".

2.3 Ovládání přístroje a zobrazení naměřených dat

V neposlední řadě nás bude zajímat přenos naměřených a zpracovaných hodnot na nějaký zobrazovací prostředek. Bude se tak dít pomocí USART (univerzální sériový asynchronní přijímač a vysílač) a převodníku UART-USB, který nám zajistí konektivitu s jakýmkoli zařízením vybaveným USB. Díky podpoře pro USB „Plug and play" pak budeme moci připojit měřicí přístroj k osobnímu počítači bez nutnosti instalace nadstandartních ovladačů.

Nakonec budeme řešit samotné zobrazení naměřených a zpracovaných dat které zrealizujeme v open-source aplikaci Data Plotter, která umožňuje jednak zobrazit časový průběh měřeného signálu ale i jeho pozdější zpracování například odečítání hodnot pomocí kurzorů, FFT (Fourierova transformace),... Další důležitou funkcionalitou této aplikace je podpora komunikace po UART přes USB-UART převodník a možnost vytvořit si v aplikaci jednoduchý uživatelský terminál, přes který se bude vytvořený měřicí přístroj ovládat. Neposlední výhodou použité aplikace je i podpora Linuxových operačních systémů a možnost aplikaci na daném zařízení provozovat i bez předchozí instalace, je tak možné ji do počítače pouze přetáhnout z flash disku.



Obrázek 2.15: Schéma propojení MCU s ovládací a zobrazovací aplikací Data Plotter

Kapitola 3

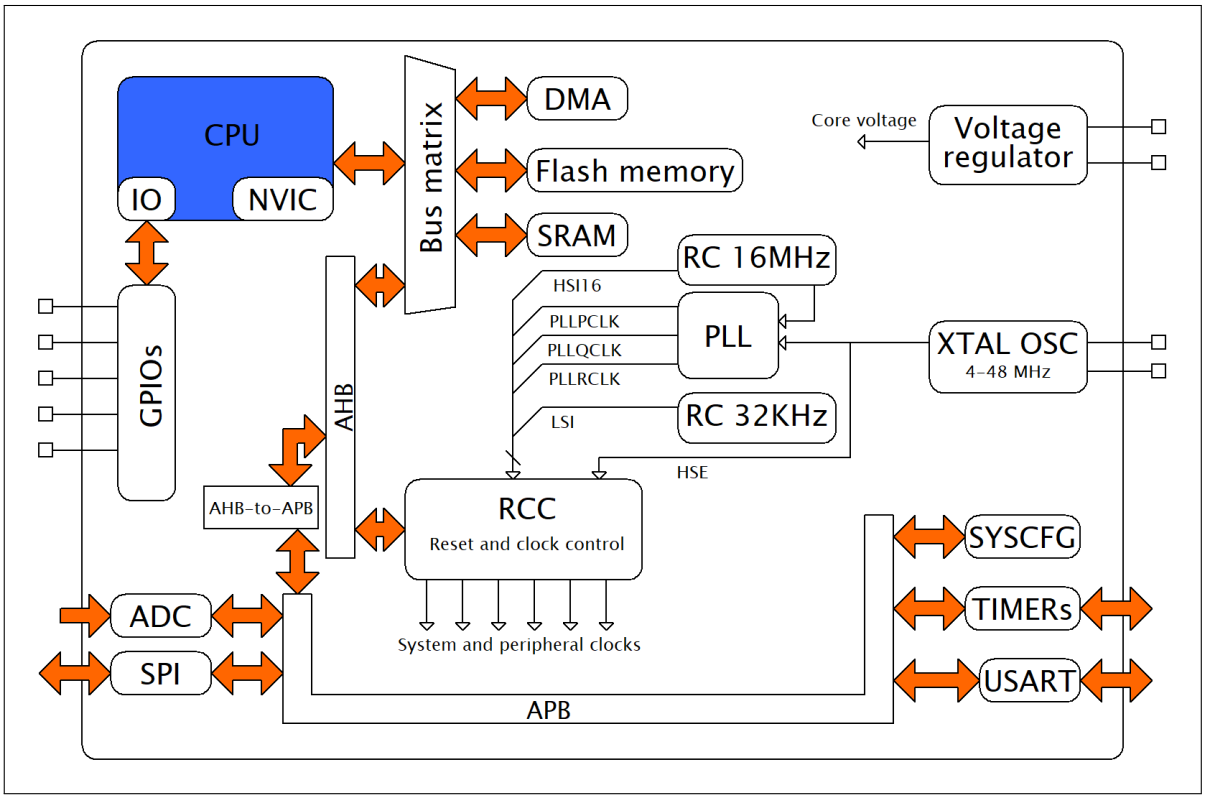
Prostředky pro realizaci

3.1 Mikroproesor STM32G031

Pro realizaci měřicího přístroje byl zvolen mikroprocesor STM32G031 s jádrem ARM Cortex-M0$^+$. Jedná se o jeden z nejméně výkonných 32bitových mikroprocesorů vyráběných firmou ST. Byl vybrán zejména kvůli jeho nízké ceně, dobré dostupnosti, relativně snadnému programování a stále dostatečnému výkonu a vybavení potřebného k realizaci požadovaného měřicího přístroje.

3.1.1 Obecný popis mikroprocesoru (MCU)

Na obrázku 3.1 je znázorněno zjednodušené blokové schéma MCU s bloky, které budou pro realizaci našeho přístroje kritické. Mimo nich se ještě v tomto MCU nachází například hodiny reálného času (RTC), blok pro podporu sběrnice I2C nebo systém window watchdog, kterým lze nezávisle na chodu jádra procesoru kontrolovat správný průběh programu.



Obrázek 3.1: Zjednodušené blokové schéma vnitřního zapojení MCU

- Na schématu z obrázku 3.1 tedy vidíme, blok označený CPU, který představuje jádro MCU a je jakýmsi jeho řídícím členem. V jádru se vykonává samotný program, probíhají zde výpočty a odsud jsou potom řízeny všechny ostatní části MCU. Pro zpracování přerušení je zde pak jednotka NVIC.
- Dále se zde nacházejí dva bloky paměti a to Flash (v našem případě 32kB ovšem vyrábí se i 64kB), ve které je uložen samotný program a SRAM (8kB), která má menší kapacitu a je zde jako paměť dat.
- Blok přímého přístupu do paměti(DMA) dokáže nezávisle na běhu programu v jádru MCU přesouvat data z periferií do paměti, případně pak z jednoho úseku paměti do jiného. Můžeme tak ušetřit "procesorový čas", kterým by program monotoně přesouval velké bloky dat. Použitý mikrokontroler má celkem 5 nezávislých kanálů DMA.
- Brány univerzálních vstupů a výstupů GPIOs slouží k nastavení jednotlivých vstupně-výstupních pinů na vstup/výstup případně jinou alternativní funkci pinu například, jako výstup PWM signálu generovaného čítačem.
- Blok označený jako RCC, jenž zajišťuje nastavení hodinového signálu pro periferie a jádro. Dále se pak stará o nastavení resetu například vlivem nízkého napájecího napětí nebo resetu vyvolaného programem.
- Do bloku RCC jsou napojeny dva interní RC (rezistor-kapacita) oscilátory, které generují referenční signál 16 MHz a 32 kHz. Blok XTAL OSC potom představuje oscilační obvod s nutným připojením vnějšího krystalu. Výstupy ze všech oscilátorů jsou ještě přivedeny na vstup fázového závěsu, jež dokáže generovat na svém výstupu hodinový signál s frekvencí vynásobenou celočíselným zlomkem, vstupní frekvenci tak můžeme podle potřeby zvýšit nebo snížit.
- Analogově digitální převodník (ADC) dokáže převést velikost napětí na konkrétním pinu MCU na digitální hodnotu, díky němu je tak mikroprocesor propojen s okolním světem i analogově.
- Komunikační rozhraní je pro MCU zajištěno jednak pomocí USART (univerzální sériový asynchronní přijímač a vysílač), jenž použijeme ke spojení MCU s PC a SPI (sériové periferní rozhraní), které se typicky používá k připojení periferií například LCD displeje k MCU, ovšem pro náš měřicí přístroj bude SPI důležité pro konstrukci logického analyzátoru.
- Blok Voltage regulator je zodpovědný za zajištění napájení jádra. Pro systémovou konfiguraci je zde blok SYSCFG, u některých fyzických pinů například určuje které piny z GPIOs na ně budou připojeny.
- Stěžejní komponentou MCU pro náš měřicí přístroj budou čítače v bloku TIMERs, který obsahuje několik nezávislých čítačů. Ty mohou pracovat v mnoha režimech a slouží pro přesné měření časových veličin nezávisle na běhu programu v jádru MCU.
- Propojení vnitřních částí MCU je realizováno sběrnicí AHB, jež je připojena na matici sběrnice (Bus matrix) a sběrnicí APB, která slouží k obsluze periferií.

3.1.2 Zdroje hodinového signálu

Jelikož budeme realizovat náš měřicí přístroj na osmi pinovém pouzdře, kde budeme mít k dispozici pouze velmi omezené množství pinů pro připojení vnějšího oscilačního obvodu, bude pro nás výhodné použít oscilátor integrovaný uvnitř mikrokontroleru. V základním režimu bude tedy zdrojem hodinového signálu pro celé MCU vnitřní RC oscilátor, který kmitá na frekvenci 16 MHz. Výhodou vnitřního RC

oscilátoru je kromě mizící potřeby připojení vnějších součástek i relativně rychlý náběh po zapnutí napájení do pracovního kmitočtu oproti oscilátoru keramickému. Ovšem hlavní nevýhodou RC oscilátoru je nepřesnost generované frekvence, která se podle výrobce liší kus od kusu vyrobeného MCU ±1% při teplotě 25 °C[1]. Oscilátor sice lze zkalibrovat pomocí HSITRIM bitů v RCC_ICSCR registru, ovšem díky značné tepelné nestálosti RC oscilátorů bude výhodnější pro přesnější měření použít vnější oscilátor krystalový, který se vykazuje vysokou teplotní stabilitou.

Mikrokontroler má již integrovaný obvod pro přímé, dvou pinové připojení krystalu, ovšem pro námi používaná pouzdra není tento obvod vyveden. Hodinový signál je však možné přivést do MCU jedním pinem, a to z vnějšího krystalového oscilačního obvodu, jehož návrhem se budeme zabývat. Náš mikrokontroler pak dovoluje, aby byl zdroj vnějšího hodinového signálu obdélníkový, sinusový nebo trojúhelníkový se střídou 40-60% a frekvencí nižší jak 48 MHz. V našem případě tedy budeme navrhovat vnější harmonický oscilátor s frekvencí 8 a 16 MHz.

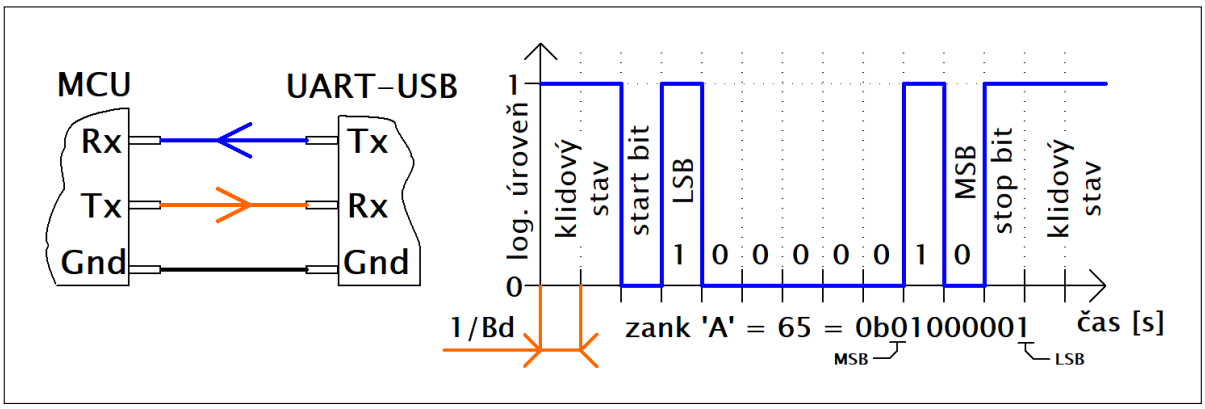
Pro maximální využití potenciálu použitého mikrokontroleru budeme ovšem požadovat vyšší referenční hodinový signál, než dokáže generovat vnitřní RC oscilátor nebo vnější oscilátor krystalový. Pro zvýšení frekvence hodinového signálu tak využijeme bloku fázového závěsu, který dokáže velice přesně z referenčního hodinového signálu f_{ref} na svém vstupu generovat signál o frekvenci f_{out}, jež je reálným násobkem frekvence referenční, tedy:

$$f_{out} = \frac{N}{D} f_{ref}, \tag{3.1}$$

kde N a D jsou celá čísla. V naší aplikaci pak budeme chtít z referenčního hodinového signálu 16 MHz (případně 8 MHz) získat 64 MHz, což je maximální kmitočet pro jádro MCU ale i další periferie, jako většina čítačů, ADC nebo USART.

3.1.3 Komunikační rozhraní UART

Pro komunikaci mezi MCU a PC aplikací na straně mikrokontroleru využijeme jednoduchého komunikačního standardu USART (Univerzální sériový, asynchronní přijímač a vysílač), někdy také označovaný UART. V našem použití bude mít UART dva komunikační vodiče a jeden referenční, tedy nulový vodič. První z nich bude sloužit mikrokontroleru, jako přijímací (Rx) a druhý, jako vysílací (Tx), jak vidíme na schématu z obrázku 3.2.



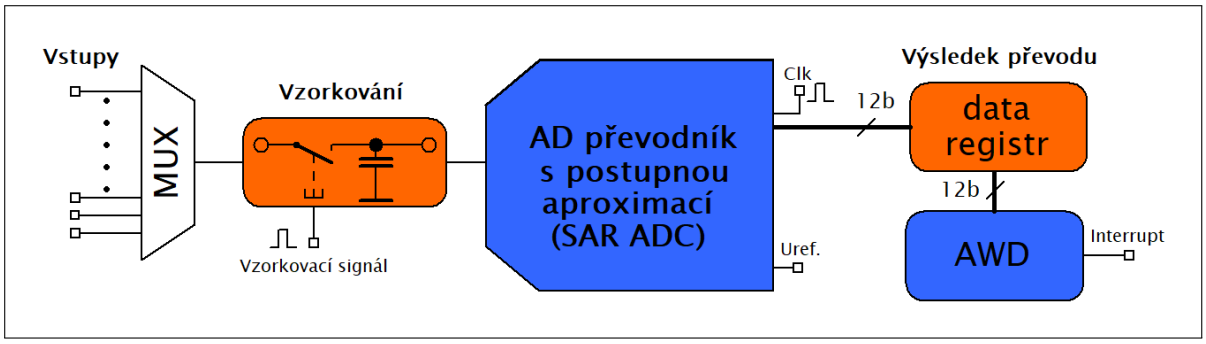
Obrázek 3.2: Schéma zapojení UART a časový průběh při vyslání znaku 'A'

Na obrázku 3.2 je také znázorněn příklad odeslání jednoho bajtu (znaku 'A') bez parity a s jedním stop bitem, kdy je v klidovém stavu na médiu log.1 (zde 3,3 V), následuje jeden start bit v log.0 (zde 0 V) a dále pak od LSB (bit nejnižšího významu) do MSB (bit nejvyššího významu), pokračuje odesílání

jednoho bajtu zakončeného stop bitem v log.1, po něm již může pokračovat další start bit, případně médium setrvává v log.1 až do začátku další komunikace. Start bit a stop bit jsou zde důležité pro synchronizaci přijímacího zařízení. Rychlost komunikace je pak udávána hodnotou Bd[bit/s], která představuje počet bitů odeslaných za jednu sekundu a její převrácená hodnota $\frac{1}{Bd}$[s/bit] časovou délku jednoho bitu.

3.1.4 Analogově digitální převodník (ADC)

Funkční blok ADC bude pro náš měřicí přístroj důležitý zejména při realizaci osciloskopu. V mikrokontroleru STM32G031 je umístěn jeden 12bitový převodník schopný dosáhnou rychlosti převodu až 2,5 Msample/s, což bude pro potřeby našeho přístroje plně dostačovat. Ve schématu analogově digitálního převodníku zobrazeného na obrázku 3.3, vidíme nejprve multiplexor (MUX) pro výběr konkrétního kanálu - jemu odpovídajícímu vstupnímu pinu případně vnitřnímu zdroji napětí, jako je teplotní senzor uvnitř MCU nebo referenční napětí používané při kalibraci ADC. Dále se vybraný signál ovzorkuje, tedy na určitou dobu se připne spínačem měřený signál ke vzorkovacímu kondenzátoru, čímž se kapacita nabije a po dobu převodu zůstává nabitá na ovzorkované napětí signálu. Doba vzorkování lze nastavit od 1,5 do 160,5 cyklu ADC, prodlužování doby vzorkování lze využít při přesném měření signálu s velkým výstupním odporem, který vyžaduje více času na plné nabití kapacity vzorkovacího obvodu.



Obrázek 3.3: Principiální schéma ADC

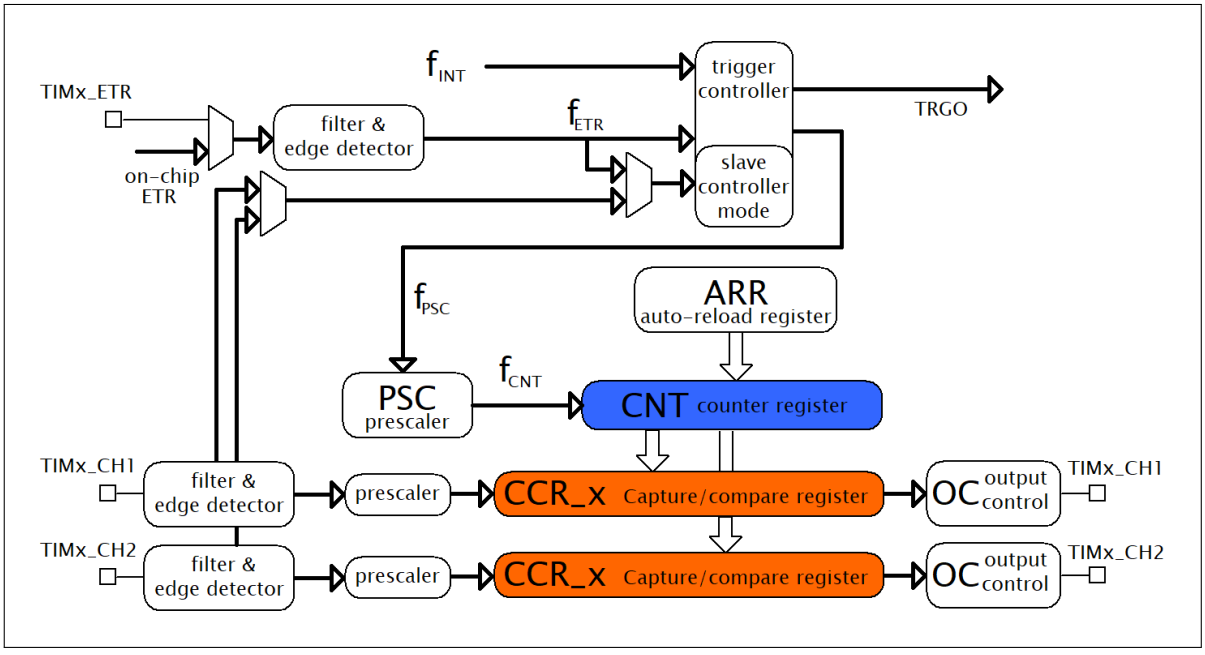
Nejdůležitější částí ADC jednotky je potom AD převodník s postupnou aproximací (SAR), který slouží pro samotný převod již ovzorkovaného analogového signálu na digitální hodnotu. SAR obsahuje 12 kondenzátorů (na každý bit jeden), jejichž kapacity jsou váženy podle bitu, který představují a po provedení vzorkování je každý z nich nabit na hodnotu měřeného napětí. Kapacita pro MSB kondenzátor je potom nejvyšší a uchovává polovinu náboje ze všech kapacit dohromady, další kondenzátor pro méně významný bit uchovává čtvrtinu náboje ze všech kapacit a tak dále až do LSB kondenzátoru. Převodník v principu spíná kapacity do kapacitního děliče od největší váhy po nejmenší, tedy mění bity od MSB po LSB a vytváří tak porovnání měřeného napětí s určitým zlomkem referenčního napětí. Nejprve tedy připnutím pouze MSB kondenzátoru zkusí, zda je měřené napětí větší než $\frac{1}{2}$ referenčního napětí (U_{ref}) a pokud ano ponechá MSB sepnutý, pokud ne MSB kondenzátor opět odpojí. Následně je připnut méně významný kondenzátor a dochází k porovnání měřeného napětí s $\frac{3}{4}U_{ref}$ případně s $\frac{1}{4}U_{ref}$ podle MSB z minulé iterace a výsledek porovnání opět rozhodne, zda tento kondenzátor zůstane sepnutý nebo ne. To se opakuje pro všechny bity až do LSB.

Po ukončení převodu aproximačním převodníkem je pak výsledná hodnota uložena v data registru odkud si již jádro nebo DMA může naměřenou hodnotu přečíst. Posledním prvkem ADC je analog-watchdog (AWD), který umožňuje nezávislou kontrolu naměřené hodnoty v data registru a na jejím základě vyvolávat přerušení. Principem AWD je nastavení kontrolovaného okna, tedy dvou hodnot, mezi kterými dovolujeme, aby se číslo v data registru pohybovalo a AWD pak kontroluje, zda jsou

čísla uvnitř tohoto okna. V případě že se v data registru vyskytne číslo mimo kontrolované okno AWD vyvolá přerušení a jádro MCU na to tak může patřičně zareagovat. Tuto funkcionalitu využijeme pro funkci trigger u osciloskopu, kdy nebudeme muset neustále programově kontrolovat, zda nenastala trigger podmínka, ale pouze si nastavíme AWD a v případě nastání trigger podmínky reprezentované překročením určité napěťové úrovně, opuštění nastaveného kontrolního okna, upozorní AWD probíhající program přerušením.

3.1.5 Čítače (TIMx)

Jednotlivé čítače v MCU budou pro realizaci čítače, jako měřicího přístroje, stěžejními prvky. V mikrokontroleru máme k dispozici celkem osm nezávislých čítačů s poměrně širokým rozsahem možného použití, přitom jejich typů je několik a liší se maximální provozní frekvencí, velikostí čítacího registru nebo počtem CCR_x jednotek (jednotka pro porovnání/záznam hodnoty z čítacího registru). Centrálním prvkem každého čítače je potom čítací registr (CNT), jak vidíme na obrázku 3.4, který příchodem každé náběžné/sestupné hrany inkrementuje/dekrementuje svou hodnou o jedna. Vstupní signál je do čítacího registru veden přes blok PSC fungující, jako před-dělička dělicí vstupní frekvenci f_{PSC}, tedy vstupní frekvence do CNT registru bude $f_{CNT} = \frac{f_{PSC}}{PSC}$, kde PSC je hodnota nastavená v bloku před-děličky. Čítací registr dokáže čítat v režimu nahoru, kdy inkrementuje svůj stav až do hodnoty v „auto-reload“ registru (ARR), dále pak v režimu dolů, kdy dekrementuje svůj obsah od hodnoty ARR do nuly, a nakonec v režimu nahoru/dolů, kdy nejprve inkrementuje od nuly do hodnoty ARR a poté dekrementuje opět do nuly.



Obrázek 3.4: Principiální schéma čítače

Jednotky CCR_x pracují ve dvou základních režimech, a to „Capture“, ve kterém si na základě příchozí události do CCR_x jednotka zapamatuje aktuální hodnotu z CNT, což využijeme zejména při přímém měření periody viz. 2.2.4. Příchozí události jsou pak ještě před vstupem do CCR_x ze vstupního pinu, zpracovány filtrem a před-děličkou. Druhým základním režimem je „Compare“, kdy CCR_x porovnává hodnotu v CNT s hodnotou uloženou uvnitř CCR_x a na základě toho pak dává pokyny k nastavení výstupu přes OC jednotku určenou pro ovládání výstupního pinu přes čítač. Tento režim pro nás bude důležitý zejména při tvorbě PWM generátoru viz.2.2.3.

Zdroj signálu pro čítací registr, tedy signálu f_{PSC}, který po průchodu blokem PSC tvoří signál f_{CNT} pro čítací registr, lze zvolit vnější nebo vnitřní. Vnitřním zdrojem hodin bude potom f_{INT}, což je hodinový signál pocházející buď z vnitřního R-C oscilátoru nebo vnějšího, v našem případě krystalového, oscilátoru. Tento hodinový signál se stejně, jako hodinový signál pro všechny periferie, lze modifikovat v bloku RCC viz. 3.1.1 a získat tak požadovanou frekvenci f_{INT}. Dalším možným zdrojem hodinového signálu pro čítací registr je potom f_{ETR}, představující buď externí vstup do MCU TIMx_ETR, který bude pro realizaci přímého měření frekvence nebo prostého čítání událostí důležitý. Zdroj hodinového signálu lze také přivést z ostatních periferií, typicky z jiného čítače. Díky jejich propojení pak lze druhým čítačem v kaskádě čítat počet přetečení vlivem ARR registru nebo omezené velikosti CNT, a tak se vyhnout použití programového přerušení.

Blok „trigger controller“ se v čítači stará o generování přerušovacích událostí („trigger-output“) pro ostatní periferie, tuto událost pak lze v jiné periferii použít například pro zapnutí konverze v ADC, jak to budeme používat při realizaci osciloskopu. Zdrojů přerušovací události pak existuje několik, například nulování čítacího registru způsobené ARR nebo způsobené jednotkou CCR_x v obou základních režimech. Další důležitou funkcionalitou tohoto bloku je požadovaný výběr hodinového signálu pro čítací registr f_{PSC} a jeho možné blokování v případě „slave controller mode“. Tento mód umožňuje na základě určité události vyvolané bloky „filter & edge detector“ zadržovat případně propouštět hodinový signál do čítacího registru. Lze tak například realizovat měření periody viz. 2.2.4, kdy příchodem události měřeného signálu na vstup TIMx_CHx povolíme průchod referenční frekvence ze vstupu f_{INT} do čítacího registru, příchodem druhé události pak opět tento průchod zablokujeme a měřenou časovou veličinu nakonec už jen dopočteme ze známé referenční frekvence a hodnoty načítané v CNT.

V následující tabulce 3.1 se podíváme na vybavenost jednotlivých čítačů, která bude důležitá pro zvolení konkrétního čítače ke konkrétnímu účelu, ať už pro realizaci samotného měření periody, frekvence a dalších časových veličin nebo pro vytvoření PWM generátoru a osciloskopu.

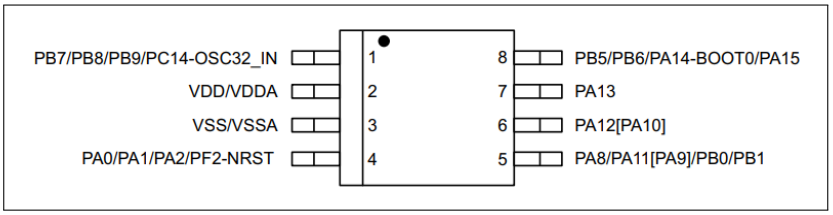
Název čítače	Velikost CNT	Typ čítání	Provozní frekvence	Velikost PSC	Podpora DMA	Počet CCR
TIM1	16-bit	nahoru, dolů, nahoru/dolů	max 128 MHz	2^{16}	Ano	4
TIM2	32-bit	nahoru, dolů, nahoru/dolů	max 64 MHz	2^{16}	Ano	4
TIM3	16-bit	nahoru, dolů, nahoru/dolů	max 64 MHz	2^{16}	Ano	4
TIM14	16-bit	nahoru	max 64 MHz	2^{16}	Ne	1
TIM16	16-bit	nahoru	max 64 MHz	2^{16}	Ano	1
TIM17	16-bit	nahoru	max 64 MHz	2^{16}	Ano	1
LPTIM1	16-bit	nahoru	max 64 MHz	2^{16}	Ne	1
LPTIM2	16-bit	nahoru	max 64 MHz	2^{7}	Ne	-

Tabulka 3.1: Přehled vlastností jednotlivých čítačů viz.[2]

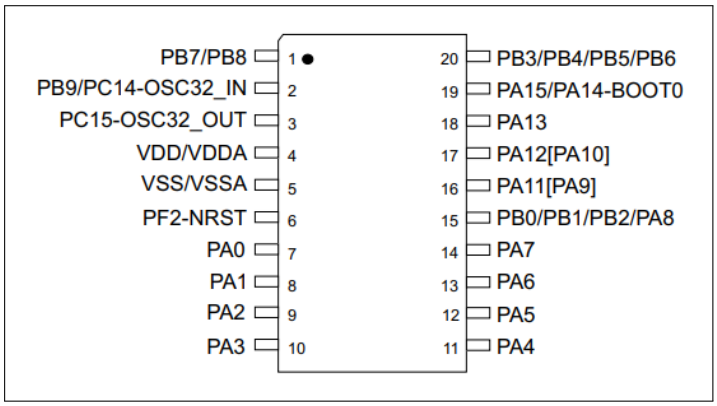
3.1.6 Pouzdra použitého mikrokontroleru STM32G031

Měřicí přístroj budeme realizovat na třech pouzdrech mikrokontroleru STM32G031. Stěžejním pak bude použití nejmenšího osmi pinového pouzdra SO8N, které vidíme na obrázku 3.5. Ovšem toto pouzdro má velmi omezený počet použitelných pinů, tedy po odečtení dvou pinů napájecích, dvou pinů

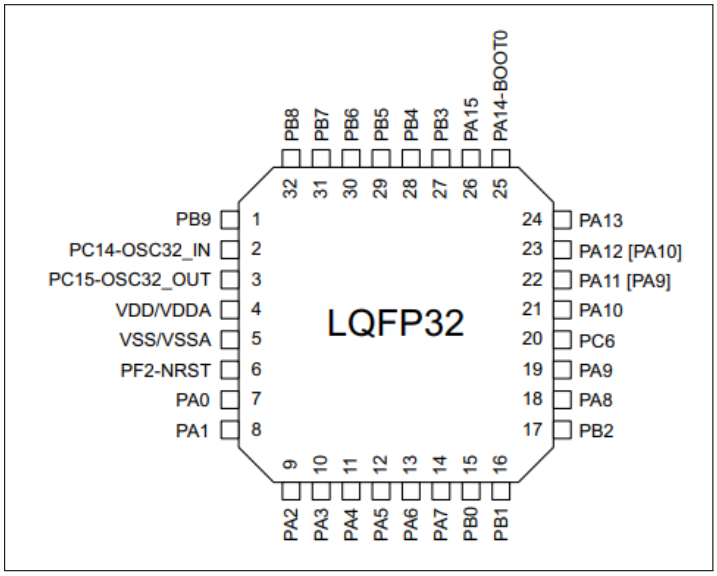
nutných pro komunikaci po UART s ovládací a zobrazovací aplikací a jednoho pinu pro možné připojení vnějšího oscilátoru pro přesná měření nám zbývají pouze tři piny pro realizaci samotného měřicího přístroje čítače, osciloskopu, logického analyzátoru a PWM generátoru, z čehož vidíme, že při potřebě měřit i dvou kanálově, tedy zároveň dva signály z různých zdrojů, bude toto pouzdro nedostačující a některé funkce tak budou muset být omezeny. Pro plnohodnotná měření ve dvou kanálovém režimu tak budeme přístroj realizovat i ve 20 pinovém pouzdře TSSOP20 viz. obrázek 3.6, které již nabízí dostatečné množství použitelných pinů. Posledním pouzdrem na kterém budeme náš měřicí přístroj realizovat pak bude 32 pinové pouzdro LQFP32, které vidíme na obrázku 3.7.



Obrázek 3.5: Pouzdro SO8N, mikrokontroler STM32G031Jx



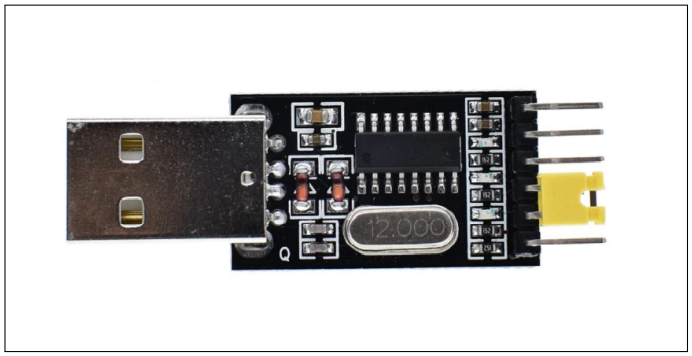
Obrázek 3.6: Pouzdro TSSOP20, mikrokontroler STM32G031Fx



Obrázek 3.7: Pouzdro LQFP32, mikrokontroler STM32G031KxT

3.2 Převodník USB-UART

Pro potřebu komunikace námi vytvořeného měřicího přístroje s aplikací v PC, která nám zajistí zobrazení naměřených dat a ovládání přístroje, budeme používat ze strany osobního počítače sběrnici USB. To je výhodné zejména díky skutečnosti, že nějaký typ této sběrnice nalezneme téměř u všech běžných stolních počítačů a laptopů. Důležitou vlastností USB je potom i možnost napájení našeho měřicího přístroje, což by například při komunikaci pomocí Bluetooth nebo Wi-Fi možné nebylo a museli bychom tak napájení měřicího přístroje řešit zvlášť. Ovšem mikrokontroler, na kterém budeme měřicí přístroj realizovat, USB nepodporuje, a tak na straně MCU pro komunikaci s PC použijeme rozhraní UART. Pro to abychom mohli tyto standarty zkombinovat využijeme převodník USB-UART s obvodem CH340G viz. obrázek 3.8, který bude jednak zajišťovat převod z jednoho použitého komunikačního rozhraní na druhé a naopak, ale i přívod napájecího napětí z PC do našeho měřicího přístroje.



Obrázek 3.8: Převodník USB-UART s obvodem CH340G

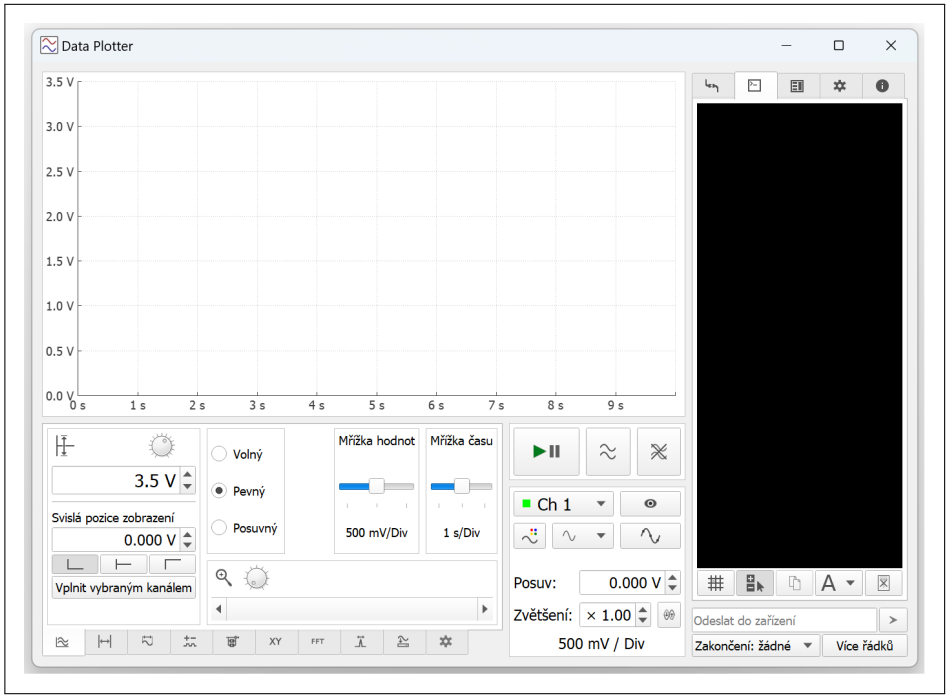
3.3 Zobrazovací aplikace Data Plotter

Data Plotter je univerzální aplikace vytvořená primárně pro zobrazení a zpracování analogového signálu přicházejícího v reálném čase z nějakého vnějšího zařízení, tedy v našem případě z vytvářeného měřicího přístroje. Vzhledem k volnému, bezplatnému stažení aplikace a možnosti ji provozovat bez předchozí instalace, bude tato platforma pro použití pro výukové účely, vhodná a její náhled pak vidíme na obrázku 3.9. Tuto aplikaci budeme používat především k zobrazení časového průběhu napětí v režimu osciloskopu, časového průběhu logických úrovní v režimu logického analyzátoru nebo časový vývoj naměřené frekvence a periody v režimu čítače. Specifickým požadavkům pro tato různá zobrazení lze vyhovět díky možnému nastavení popisků os a jejich měřítka nebo zobrazení více měřených kanálů do jednoho grafu. Pro podporu zpracování již zobrazeného signálu zde pak jsou:

- **Kurzory**
  Kterými dokážeme odečítat hodnoty konkrétních vzorků nebo rozdíly mezi nimi.
- **Výpočet parametrů periodického signálu**
  Z naměřeného střídavého periodického signálu odečítá frekvenci, periodu, maximum, minimum, stejnosměrnou (DC) složku signálu, průměrnou délku vzestupných a sestupných hran a efektivní hodnotu střídavé složky (RMS).
- **Průměrování**
  Z posledních N příchozích naměřených hodnot počítá průměr, kde N je nastavitelné.
- **Matematické operace**
  Dokáže provádět sčítání, odčítání, násobení a dělení signálů ze 2 až 3 různých kanálů, případně

lze místo konkrétního kanálu zvolit konstantu. Vytvoří tedy signál nový, který bude výsledkem konkrétní matematické operace s vybranými kanály nebo kanálem a nastavenou konstantou.

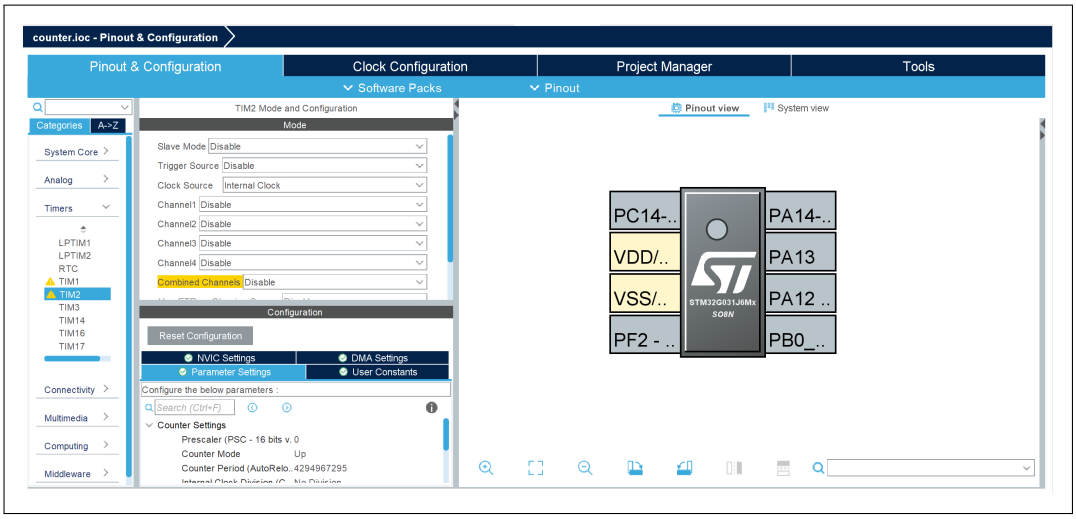
- **Fourierova transformace (FFT)**
  Z naměřeného signálu provede Fourierovu transformaci, kdy po nastavení žádané první harmonické frekvence rozloží tato metoda signál na jednotlivé součtové složky vyšších harmonických a prozradí nám tak frekvenční parametry změřeného signálu.



Obrázek 3.9: Náhled na okno aplikace Data Plotter

3.4 Vývojové prostření použité při vývoji programu pro miktokontroler

Pro vývoj samotného programu realizujícího náš měřicí přístroj lze využít řadu vývojových prostředí, které mikrokontroler STM32GO31 podporují. Jedním z nich je potom STM32CubeIDE, které mimo požadované podpory námi vybraného mikrokontroleru a variant jeho zapouzdření, umožňuje nastavení většiny funkcí MCU, ať už nastavení hodinového signálu, režimů čítačů nebo ADC, v grafickém prostředí, z něhož následně generuje patřičný kód v jazyce C. Tato část kódu je potom tvořena pomocí HAL nebo LL knihoven. Pro náš mikrokontroler s velmi omezenou pamětí však budou HAL knihovny, vzhledem jejich paměťové náročnosti, nepoužitelné. Knihovny LL jsou potom méně náročné na paměť, a tak je budeme při vývoji našeho programu využívat. Po odladění konkrétních částí programu s LL knihovnami se však budeme snažit i funkce, které je používají nahradit přímými přístupy do registrů, a tak zajistit co nejefektivnější využití paměti. Náhled na grafické prostředí STM32CubeIDE vidíme na obrázku 3.10.



Obrázek 3.10: Náhled na okno grafického nastavení MCU vývojového prostředí STM32CubeIDE

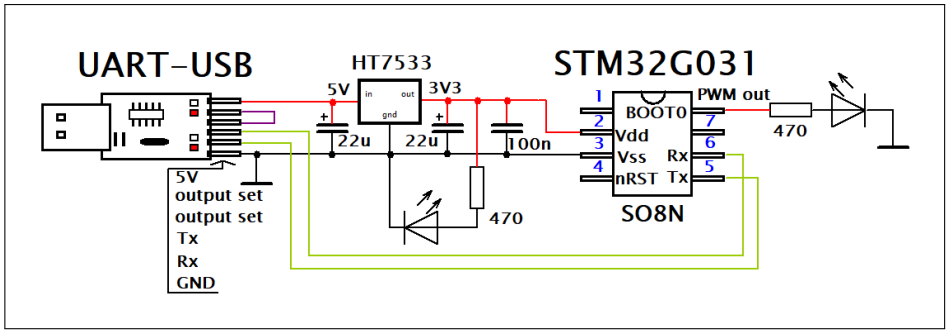
Kapitola 4

Realizace měřicího přístroje

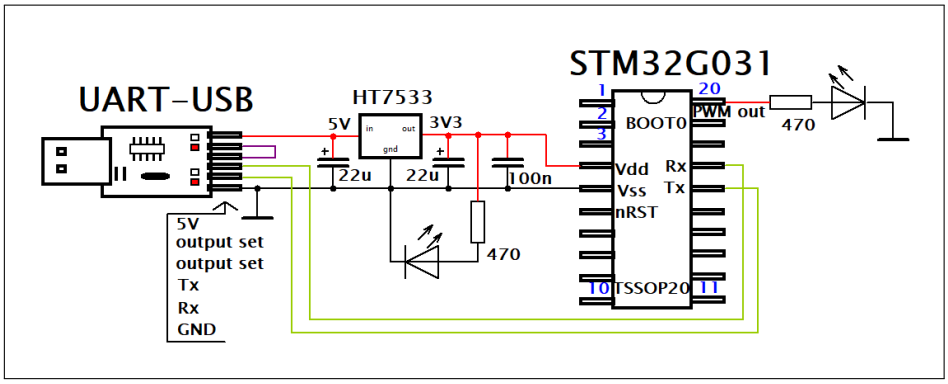
4.1 Základní nastavení a zprovoznění mikrokontroleru

4.1.1 Zapojení na nepájivém poli

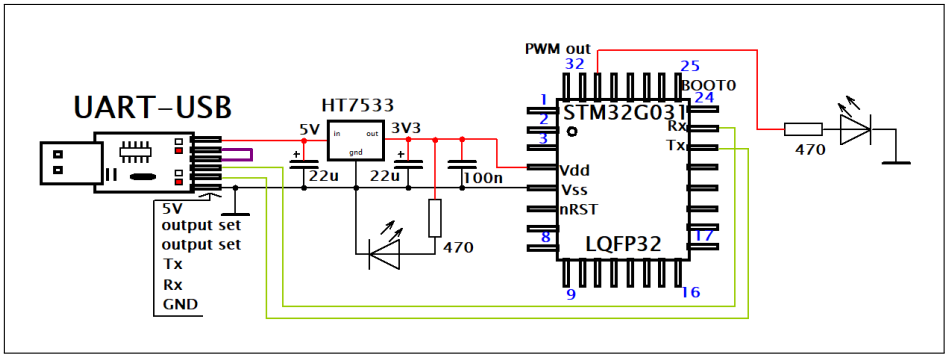
Ještě před započetím samotného programování MCU musíme zajistit připojení napájecího napětí a vodičů Rx a Tx k bráně UART, který bude sloužit nejen jako komunikační rozhraní mezi mikrokontrolerem a aplikací v PC, ale i jako prostředek pro nahrání programu (firmware-u) do MCU. Pro samotné napájení pak použijeme napětí z USB zprostředkované USB-UART převodníkem, který ovšem poskytuje jen napětí 5 V, což však našemu mikrokontroleru nevyhovuje. Budeme tedy používat lineární stabilizátor z 5 V na 3,3 V, které jsou již pro naše MCU přípustné. Zapojení pro všechna tři pouzdra vidíme na obrázcích 4.1, 4.2, 4.3. Na všech schématech pak vidíme kromě již zmíněného stabilizátoru napětí i kondenzátor na jeho vstupu zabraňující jeho rozkmitání, dále pak kondenzátory keramický a elektrolytický na vstupu napájení mikrokontroleru, které zde jsou pro pokrytí odběrových špiček MCU, keramický kondenzátor kompenzuje krátké, málo výkonové špičkové odběry a ten elektrolytický pak delší, více výkonové špičkové odběry. Nakonec je zde LED připojená přes rezistor na napájecí napětí sloužící k indikaci napájení mikrokontroleru a LED přes rezistor připojená k pinu MCU s výstupem PWM Generátoru, která slouží jednak k testování samotného PWM Generátoru, ale i k indikaci začátku běhu programu po úspěšném nahrání vytvořeného firmware-u do mikrokontroleru.



Obrázek 4.1: Zapojení mikrokontroleru v pouzdře SO8N



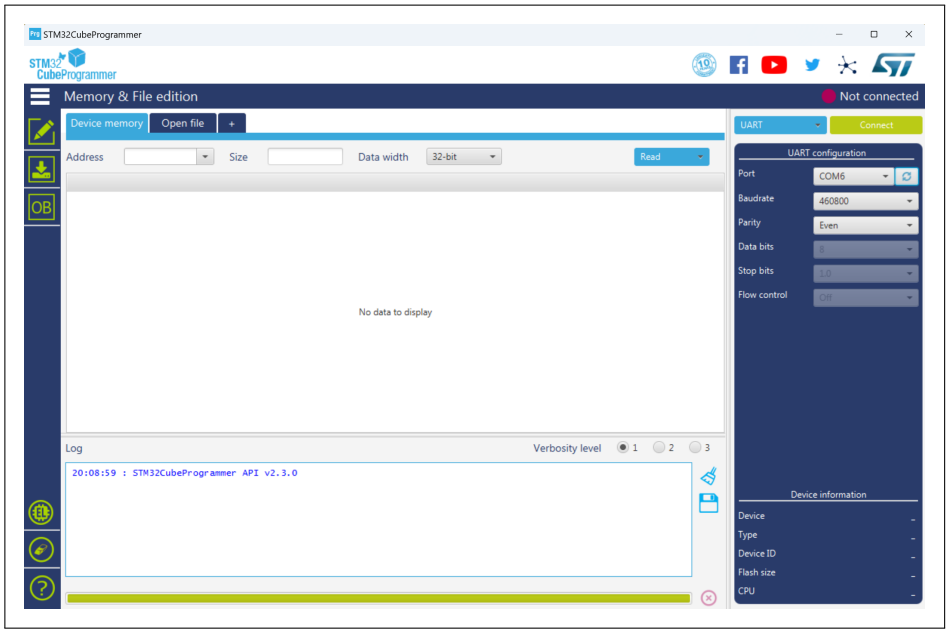
Obrázek 4.2: Zapojení mikrokontroleru v pouzdře TSSOP20



Obrázek 4.3: Zapojení mikrokontroleru v pouzdře LQFP32

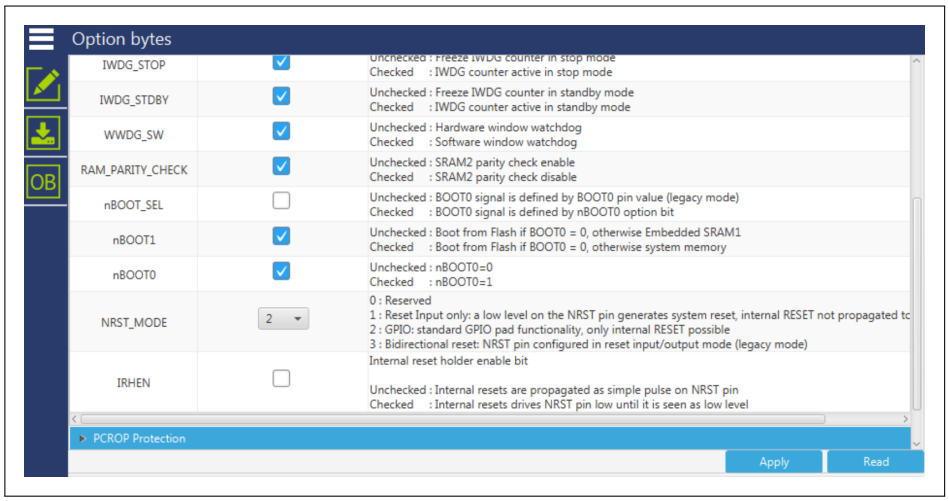
4.1.2 Nahrání firmware-u do MCU a nastavení reset pinu

Pro nahrání firmware-u do mikrokontroleru budeme používat aplikaci STM32CubeProgrammer. Konkrétně použijeme jeho verzi 2.3.0, jak vidíme v náhledu aplikace na obrázku 4.4. Novější verze tohoto programu již některá nastavení neumožňují, proto s nimi nebude možné níže uvedený postup zopakovat. Po řádném zapojení mikrokontroleru, viz. 4.1.1, a připojení převodníku USB-UART do vhodného portu počítače musíme MCU přepnout do BOOT režimu, kdy do něj lze nahrát program. To provedeme připojením BOOT0 pinu na log.1 (3,3 V) přes rezistor cca 470 Ω a zároveň resetu mikrokontroleru přivedením na pin nRST log.1 (3,3 V) přes rezistor 470Ω. Dále pak nastavíme v pravé části okna aplikace komunikaci po UART, konkrétní port, rychlost komunikace „Baudrate“ na 460800Bd/s, paritu sudou „even“ a připojíme se se k MCU tlačítkem „Connect“.



Obrázek 4.4: Náhled aplikace STM32CubeProgrammer

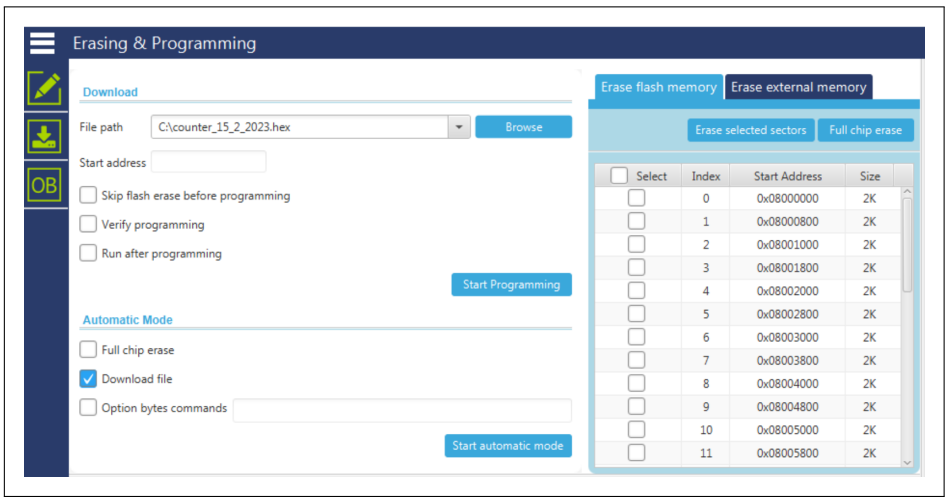
Po úspěšném propojení aplikace a mikrokontroleru v levé horní části okna aplikace otevřeme položku „Option bytes“ a v ní rozbalíme lištu „User Configuration“, kde úplně dole odstraníme označení nejprve z položky nBOOT_SEL, dále pak nastavíme NRST_MODE na číslo 2 čímž si uvolníme pin s vývodem resetu, tedy v případě připojení napájení se MCU podívá zda je na reset pinu log.1 a dokud zde log.1 je mikrokontroler bude držen v resetu, po odpojení zdroje log.1 pak MCU reset stav opustí a začne vykonávat program, ovšem v tomto módu již nezáleží na logickém stavu reset pinu a k resetu dochází opět až při odpojení napájení. Tento mód nám tedy umožní využívat periferie vyvedené na stejný fyzický pin, jako reset, což využijeme především u nejmenšího osmi pinového pouzdra, ovšem u větších pouzder již tuto úpravu díky dostatečnému množství dělat nemusíme. Po nastavení resetu ještě odstraníme označení u položky IHREN a měli bychom tak dosáhnout nastavení, jako na obrázku 4.5, kde jsou všechny ostatní položky zaškrtnuté. Teď tedy zbývá pouze do mikrokontroleru uložit naše nastavení tlačítkem „Apply“.



Obrázek 4.5: Nastavení reset pinu v aplikaci STM32CubeProgrammer

Pro samotné nahrání programu do mikrokontroleru se přesuneme do části „Erasing & programming“

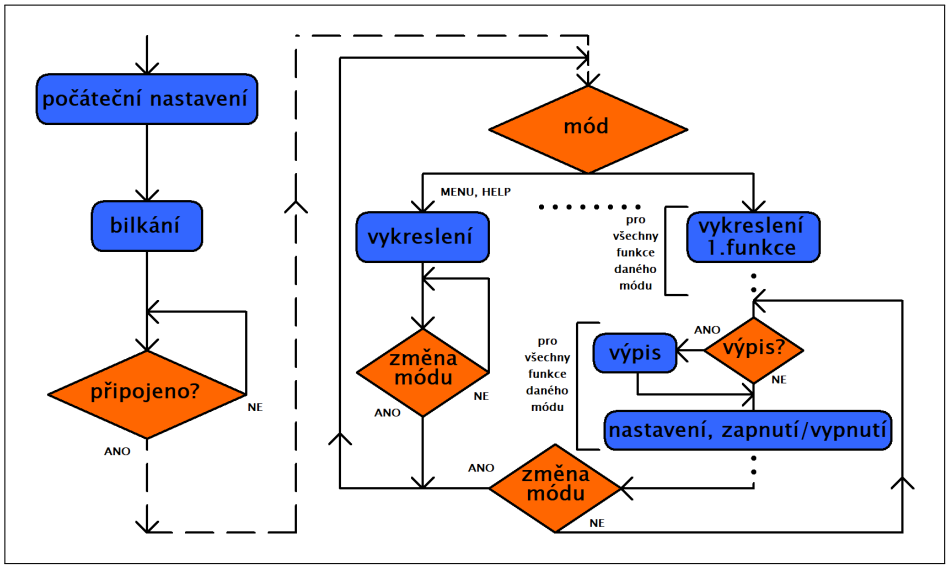
nacházející se opět v levé horní části, jeho náhled pak vidíme na obrázku 4.6. Zde pouze vybereme požadovaný firmware, typicky soubory s příponou .bin, .elf nebo .hex, a nahrajeme jej do MCU tlačítkem „Start Programming“. Po úspěšném nahrání programu do mikrokontroleru ho musíme opět dostat z BOOT režimu, což docílíme odstraněním všeho připojeného na tlačítko BOOT0 a reset pin, tedy přivedením log.0 na BOOT0 pin a log.1 na reset pin a následným resetem MCU, což lze uskutečnit buď odpojením a opětovným připojením napájecího napětí, nebo pokud jsme u větších pouzder nezablokovali funkci reset pinu, pak i přivedením log.1 na reset a následným odejmutím log.1 z reset pinu. Při každém dalším spouštění mikrokontroleru musí zůstat piny BOOT0 a reset volné, případně připojené na log.0, aby nedocházelo k nechtěnému přechodu do BOOT režimu. Správné rozběhnutí programu si pak lze ověřit krátkým zablikáním LED na pinu s výstupem PWM Generátoru při každém startu programu po resetu.



Obrázek 4.6: Nahrání firmware-u do MCU pomocí aplikace STM32CubeProgrammer

4.2 Struktura programu

Program našeho měřicího přístroje bude muset řídit mnoho periferií, přenosů a zpracování dat a přitom interaktivně reagovat na požadavky uživatele v terminálu, a to vše zdánlivě paralelně. Proto je důležité si vhodně zvolit strukturu programu tak, aby byla přehledná a zajistila všem probíhajícím měřicím procesům dostatek procesorového času a zároveň umožnila rychlou odezvu na požadavky uživatele. Námi použitá struktura je zobrazena na vývojovém diagramu 4.7. Odtud vidíme, že program začíná jednak základním nastavením, jako je hodinový signál, ADC nebo UART, dále pak následuje zablikání, které slouží jako signalizace začátku běhu programu, a nakonec čeká se na připojení Data Plotter-u. Po jeho připojení již program přechází do nekonečné smyčky struktury fungující na principu stavového automatu, tedy každý mód bude představovat jeden stav, který lze opustit pouze přechodem do jiného stavu, typicky z módu MENU budeme přecházet do jednotlivých měřicích stavů a z nich opět do módu MENU.



Obrázek 4.7: Vývojový diagram programu

Prvním typem módů budou MENU a HELP, kde se po vstupu do tohoto módu nejprve vykreslí náležité terminálové okno a pak už jen mikrokontroler čeká na příchod znaku z terminálu, který přepne na jiný mód. Dalším typem pak budou jednotlivé měřicí módy, které budou mít více nezávislých funkcí, například Čítač, PWM Generátor, Osciloskop, běžících na MCU zdánlivě paralelně. Prvním úkonem tedy bude vykreslení části terminálu pro každou z funkcí. Následně bude ve smyčce probíhat části kódu jednotlivých funkcí a zde jednak kontrolovat, zda jsou již připravena naměřená data k výpisu a případně je vypíše do Data Plotter-u a dále pak kontrolovat, zda nepřišel z PC nějaký znak pro nastavení či zapnutí/vypnutí měření a případně na tyto požadavky reaguje. Nakonec po projití všech funkcí daného módu se mikrokontroler podívá, zda nepřišel znak pro přepnutí módu, pokud ano MCU se přepíná do konkrétního módu a pokud ne, program opět začne procházet části kódu jednotlivých funkcí, kde vypisuje naměřená data na do Data Plotter-u a reaguje na požadavky uživatele.

4.3 Nastavení zdroje hodinového signálu

V našem měřicím přístroji budeme využívat primárně vnitřní zdroj hodinového signálu, a to konkrétně RC oscilátor o referenční frekvenci 16 MHz (HSI). Frekvenci tohoto zdroje potom budeme chtít upravit v bloku fázového závěsu tak, abychom pro všechny periferie zajistili signál 64 MHz, což je u většiny interních periferií maximální pracovní frekvence a umožní nám to tak plně využít jejich potenciál. Mimo vnitřního RC oscilátoru budeme ještě využívat možnost vstupu vnějšího hodinového signálu (HSE) například z přesnějšího krystalového oscilátoru. Tento vstup je u pouzdra SO8N umístěn na fyzickém pinu č.1 a budeme si tak tento pin pro tuto funkci rezervovat. Jelikož by bylo poměrně nepraktické přepínat zdroj hodinového signálu v běhu uživatelského programu, budeme toto přepínání realizovat pouze v inicializační části programu po restartu mikrokontroleru, s tím že zrealizujeme detekci a rozpoznání frekvence připojení vnějšího oscilátoru a podle detekovaného signálu pak nastavíme blok fázového závěsu, aby byla výstupní frekvence do jádra a periferií 64 MHz.

Nejprve musíme při konfiguraci hodin nakonfigurovat paměť flash, kdy ve smyčce čekáme na dokončení konfigurace:

```
    FLASH->ACR |= LL_FLASH_LATENCY_2;
    while(LL_FLASH_GetLatency() != LL_FLASH_LATENCY_2);
```

Dále budeme potřebovat zapnout vnitřní RC oscilátor, nakonfigurovat blok fázového závěsu (PLL) do režimu pro vstup z vnitřního oscilátoru, který bude násobit 16ti a dělit 2ma čímž dostaneme oněch 64 MHz, které pak do zbytku mikrokontroleru pustíme povolením PLL:

```
RCC->CR |= RCC_CR_HSION;
while(LL_RCC_HSI_IsReady() != 1);

LL_RCC_PLL_ConfigDomain_SYS(LL_RCC_PLLSOURCE_HSI,
 LL_RCC_PLLM_DIV_1, 16, LL_RCC_PLLR_DIV_4);
RCC->CR |= RCC_CR_PLLON;
RCC->PLLCFGR |= RCC_PLLCFGR_PLLREN;
while(LL_RCC_PLL_IsReady() != 1);

RCC->CFGR |= LL_RCC_SYS_CLKSOURCE_PLL;
while((RCC->CFGR & RCC_CFGR_SWS) != LL_RCC_SYS_CLKSOURCE_STATUS_PLL);
```

Nakonec ještě musíme nastavit dvě systémové proměnné, které definují frekvenci hodinového signálu na požadovaných 64 MHz:

```
LL_InitTick(64000000, 1000U);
SystemCoreClock =  64000000;
```

Dokončením těchto kroků jsme nakonfigurovali MCU pro běh na vnitřní RC oscilátor a mikrokontroler již začal na tyto hodiny běžet.

Pro detekci a případné měření frekvence vnějšího zdroje hodinového signálu použijeme čítač TIM16, jehož CC1 jednotka je připojena na stejný fyzický pin pouzdra SO8N, jako pin RCC_OSC_IN sloužící pro vstup vnějších hodin. Čítač TIM16 budeme nastavovat do „capture“ módu, kdy v případě zaznamenání náběžné hrany CC1 zkopíruje do CCR1 stav čítače CNT. Nastavení začneme s konfigurací vstupního pinu, pro který nejprve povolíme hodiny příslušné brány, přidělíme mu alternativní funkci odpovídající CC1 jednotce čítače TIM16 a nastavíme zde pull-down rezistor, abychom v případě nezapojeného pinu neměřili frekvenci rušivého indukovaného napětí:

```
RCC->IOPENR |= LL_IOP_GRP1_PERIPH_GPIOB;

LL_GPIO_InitTypeDef GPIO_InitStruct = { 0 };
GPIO_InitStruct.Pin = LL_GPIO_PIN_8;
GPIO_InitStruct.Mode = LL_GPIO_MODE_ALTERNATE;
GPIO_InitStruct.Pull = LL_GPIO_PULL_DOWN;
GPIO_InitStruct.Alternate = LL_GPIO_AF_2;
LL_GPIO_Init(GPIOB, &GPIO_InitStruct);

RCC->APBENR2 |= LL_APB2_GRP1_PERIPH_TIM16;
```

Kde jsme ještě posledním řádkem povolili hodiny do periferie čítače. Následně nastavíme do ARR registru maximální hodnotu 65535, připojíme CC1 kanál pro „capture“ vstup a nastavíme předděličku měřeného signálu na 8, tedy později bude naměřená hodnota frekvence osm krát menší než ve skutečnosti:

```
TIM16->ARR = 65535;
TIM16->CCMR1 |= TIM_CCMR1_CC1S_0;
TIM16->CCMR1 |= TIM_CCMR1_IC1PSC;
LL_mDelay(1500);
```

Nakonec jsme ještě před započetím samotného měření čekali 1,5s na ustálení vnějšího zdroje hodinového signálu.

Měření započneme nulováním čítače, nulováním před-děličky CC1 kanálu, povolením vstupu CC1 a zapnutím čítání:

```
TIM16->CNT = 0;
TIM16->EGR |= TIM_EGR_UG;
TIM16->CCER |= LL_TIM_CHANNEL_CH1;
TIM16->CR1 |= TIM_CR1_CEN;
```

Tímto se již rozběhlo čítání v registru CNT, a to s frekvencí 64 MHz. Teď budeme ve smyčce čekat na první odměr, který si po odměření uložíme do paměti, následně na druhý odměr, který si uložíme do paměti, až na čtvrtý odměr, který si opět ukládáme a měření tím skončí. Pokud ovšem nebyl učiněn žádný odměr, než čítač napočítá cca 1ms smyčka skončí:

```
while(TIM16->CCR1 == 0 && TIM16->CNT < 60000);
cap1 = TIM16->CCR1;
while(TIM16->CCR1 == cap1 && TIM16->CNT < 60000);
cap2 = TIM16->CCR1;
while(TIM16->CCR1 == cap2 && TIM16->CNT < 60000);
cap1 = TIM16->CCR1;
while(TIM16->CCR1 == cap1 && TIM16->CNT < 60000);
cap2 = TIM16->CCR1;
```

Teď již máme v proměnných cap1 a cap2 dva po sobě jdoucí odměry, ze kterých bude možno dopočítat frekvenci, kde proměnné cap1 a cap2 jsou globální uint16_t a byli v předchozím kódu odměřeny čtyřikrát kvůli ustálení měřené veličiny.

Následně se přesvědčíme, že byli odměry uskutečněny správně, tedy že nebyla načítána cca 1ms. Pokud byli odměry správné přepneme na vnější oscilátor a té následujícím způsobem. Nejprve vypneme a odpojíme vnitřní RC oscilátor a PLL (první tři řádky pod if funkcí), následně zapneme vstup vnějšího hodinového signálu (další tři řádky pod if funkcí):

```
if(TIM16->CNT < 60000){
    RCC->CFGR = 0;
    RCC->PLLCFGR = 0;
    RCC->CR = 0;

    RCC->CR |= RCC_CR_HSEBYP;
    RCC->CR |= RCC_CR_HSEON;
    while(LL_RCC_HSE_IsReady() != 1);
```

Následně musíme rozhodnout o frekvenci vnějších hodin, která bude v našem případě 8 MHz nebo 16 MHz, kdy naměřenému rozdílu mezi dvěma sousedními odměry 64 taktů čítače, bude náležet frekvence 8 MHz a rozdílu 32 taktů čítače pak frekvence 16 MHz. Pro zdroj 8 MHz nastavíme PLL, pro získání výstupu 64 MHz, na násobení 32 a dělení čtyřmi a volbu zdroje na HSE, tedy vnější oscilátor. Pokud bude vnější zdroj 16 MHz bude nastavení PLL stejné, jako u vnitřního oscilátoru, pouze volba zdroje bude tentokrát HSE:

```
if((cap2-cap1) > 48)
    LL_RCC_PLL_ConfigDomain_SYS(LL_RCC_PLLSOURCE_HSE,
            LL_RCC_PLLM_DIV_1, 32, LL_RCC_PLLR_DIV_4);
```

```
    else
        LL_RCC_PLL_ConfigDomain_SYS(LL_RCC_PLLSOURCE_HSE,
                LL_RCC_PLLM_DIV_1, 16, LL_RCC_PLLR_DIV_4);
```

Nakonec ještě zapneme PLL a povolíme jeho výstup, jak jsme to dělali u vnitřního oscilátoru:

```
    RCC->CR |= RCC_CR_PLLON;
    RCC->PLLCFGR |= RCC_PLLCFGR_PLLREN;
    while(LL_RCC_PLL_IsReady() != 1);

    RCC->CFGR |= LL_RCC_SYS_CLKSOURCE_PLL;
    while((RCC->CFGR & RCC_CFGR_SWS) != LL_RCC_SYS_CLKSOURCE_STATUS_PLL);
}
```

Teď již mikrokontroler běží na příslušném hodinovém signálu a zbývá jen vypnout čítač, resetovat jeho nastavení a deaktivovat používaný pin přiřazením nedefinované alternativní funkce:

```
    TIM16->CCER = 0;
    TIM16->CCMR1 = 0;
    TIM16->CR1 = 0;
    GPIO_InitStruct.Alternate = LL_GPIO_AF_3;
    LL_GPIO_Init(GPIOB, &GPIO_InitStruct);
```

Po načtení uživatelského menu pak bude informace o konkrétním používaném zdroji hodinového signálu vypsána, jak vidíme na obrázku 4.8, kde vlevo je výpis pro vnitřní RC oscilátor, vprostřed pak pro vnější 8 MHz oscilátor a vpravo pro oscilátor 16 MHz.

Obrázek 4.8: Zdroj hodinového signálu vypsaný na terminálu

4.4 Komunikace po UART

Jak již bylo mnohokrát zmíněno, pro komunikaci s Data Plotter-em bude mikrokontroler používat UART. Ten budeme chtít nastavit na rychlost přenosu na 460800Bd/s, která by měla plně dostačovat i pro přenosy větších objemů dat například u výpis naměřených hodnot v režimu osciloskopu. Ostatní nastavení pak bude stačit základní, tedy datových bitů osm, paritu žádnou, stop bit jeden a bez kontroly toku dat. Na straně Data Plotter-u bude nastavení poměrně snadné, jelikož jsou zde již defaultně všechny hodnoty nastaveny a stačí pouze změnit přenosovou rychlost na námi požadovanou. Abychom nemuseli vždy po spuštění aplikace tento parametr zadávat, můžeme využít uložení veškerého nastavení aplikace do souboru. To provedeme v sekci nastavení, kterou rozklikneme v pravém horním rohu nad terminálem a soubor s nastavením pak uložíme tlačítkem vpravo dole „Save settings“, kde za jméno souboru pro jednoduchost zvolíme „degault.cfg“, čímž ale přepíšeme defaultní nastavení, proto je vhodné si ho nejprve uložit do jiné složky případně pod jiným jménem.

Mikroprocesor pak budeme nastavovat v programu v několika krocích. Nejprve bude nutné přemapovat piny PA11 na PA9 a PA12 na PA10 pro aktivaci jejich možného použití pro UART a to příkazy:

```
RCC->APBENR2 |= LL_APB2_GRP1_PERIPH_SYSCFG;
SYSCFG->CFGR1 |= LL_SYSCFG_PIN_RMP_PA11;
SYSCFG->CFGR1 |= LL_SYSCFG_PIN_RMP_PA12;
```

Pro nastavení těchto dvou pinů využijeme automatické generování kódu, kdy po zadání požadované funkce UART1_Tx na pinu PA9 a UART1_Rx na pinu PA10 získáme kód definující mód těchto pinů, dále jejich alternativní funkci, rychlost, typ výstupu a volbu použití pull-up/pull-down rezistoru. Dále pak musíme pustit hodinový signál do periferie UART a GPIO brány GPIOA a povolit přerušení od UART, aby bylo možné vyčítat přijaté znaky v přerušení:

```
RCC->CCIPR |= (LL_RCC_USART1_CLKSOURCE_PCLK1 & 0x0000FFFFU);
RCC->APBENR2 |= LL_APB2_GRP1_PERIPH_USART1;
RCC->IOPENR |= LL_IOP_GRP1_PERIPH_GPIOA;
NVIC_SetPriority(USART1_IRQn, 0);
NVIC_EnableIRQ(USART1_IRQn);
```

Nakonec už jen zbývá nastavit rychlost komunikace, její mód a povolení funkce této periferie. Do BRR „Baud rate register“, dopočítáme hodnotu ze známé velikosti „Baud rate“ Bd a frekvence hodinového signálu této periferie f, jako:

$$BRR = \frac{f}{Bd} + 1 = \frac{64000000}{460800} + 1 = 140. \tag{4.1}$$

Mód komunikace pak bude nastaven na příjem i vysílání:

```
USART1->BRR = 140;
USART1->CR1 |= LL_USART_DIRECTION_TX_RX;
USART1->CR1 |= USART_CR1_UE;
```

Příjem dat budeme řešit v přerušení, a tedy ve funkci USART1_IRQHandler(), která se volá při každém přijetí znaku, kdy vyčteme přijatý znaku do k tomu určené proměnné, jako:

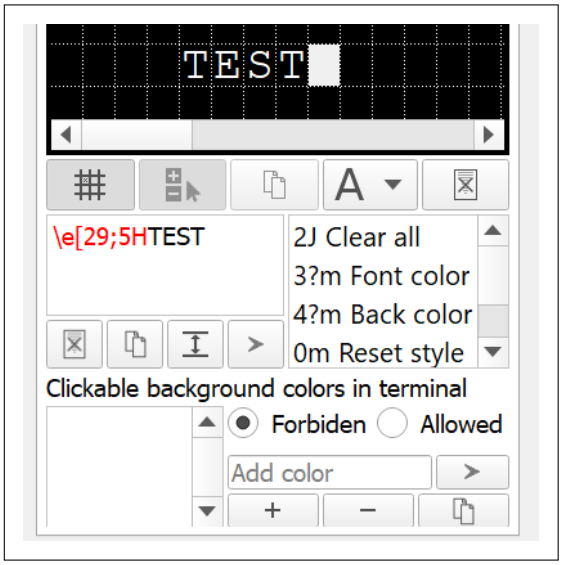
```
rec_char = (UART->RDR & USART_RDR_RDR & 0xFFU);
```

Pro odeslání znaků pak pro jednoduchost použijeme knihovnu stdio.h a funkci printf(), ovšem tato funkce bude při každém jejím volání odesílat znaky přes funkci _write(), kterou musíme teprve definovat. Funkce má tři parametry, a to jméno souboru kam se má zapisovat, ovšem to nás zajímat nebude, dále pak pole znaků, tedy znaky, které chceme odeslat, a nakonec počet znaků, které chceme odeslat. Budeme tedy procházet všechny odesílané znaky a jeden po druhém je vkládat do odesílacího registru UART->TDR, po každém vložení pak budeme čekat, dokud nebude možné do tohoto registru vložit další znak. Po projití všech požadovaných znaků už jen počkáme, než bude dokončen přenos po UART a všechny znaky tak budou korektně odeslány. Vytvořená funkce pak vypadá následovně:

```
int _write(int file, char *buf, int nbytes) {
    uint8_t num_of_byte = 0;
    while (num_of_byte < nbytes) {
        while ((UART->ISR & USART_ISR_TXE_TXFNF) != USART_ISR_TXE_TXFNF);
        UART->TDR = buf[num_of_byte++];
    }
    while ((UART->ISR & USART_ISR_TC) != USART_ISR_TC);
return nbytes;
}
```

4.5 Uživatelský terminál

Uživatelský terminál budeme v aplikaci Data Plotter tvořit pomocí k tomu určených příkazů umožňujících psát text, volit si jeho barvu a barvu jeho pozadí, odřádkovat, přecházet na libovolné políčko, psát tučně či podtrženě, případně mazat řádky nebo celý obsah terminálu. Pro rozlišení zápisu do terminálu bude sloužit $$T na začátku zprávy Data Plotter-u, po této značce pak již bude následovat vypisovaný text a příkazy pro jeho úpravu. Terminál může být přepnut do dvou velikostí jeho šířky a to 14 znaků na šířku nebo 21 znaků na šířku, přičemž my budeme pro lepší přehlednost používat pro většinu režimů šířku 14 znaků, jedinou výjimkou pak bude režim HELP, kdy budeme potřebovat do terminálu vypsat poměrně hodně textu a 21 znakový terminál tak bude pro tento účel vhodnější. Sekvenci příkazů a textu pro vytvoření požadovaného vzhledu a funkce terminálu pak budeme moci tvořit přímo v Data Plotter-u v k tomu určené části. Tu nalezneme pod terminálem s názvem „Debug mode“ a zde si již velice intuitivně vybíráme požadované příkazy z nabídky případně kliknutím na políčko v terminálu vygenerujeme příkaz pro přesun kurzoru na toto políčko. Vytvořenou sekvenci příkazů si pak lze zkopírovat, vložit do programu MCU a z mikrokontroleru tuto sekvenci, při každém požadavku na vykreslení terminálu, s předponou $$T, odeslat. Náhled „Debug mode“ v Data Plotter-u vidíme na obrázku 4.9.

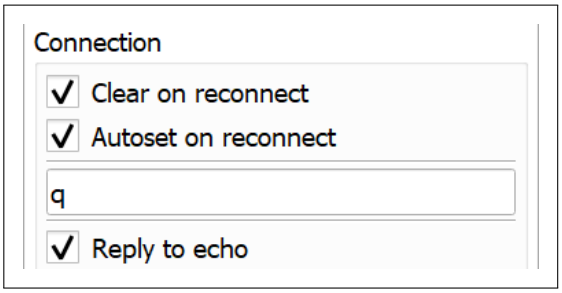


Obrázek 4.9: Tvorba terminálu v Data Plotter

Před samotným odesíláním příkazů pro vykreslení konkrétní obrazovky terminálu musíme provést základní nastavení, a to šířky terminálu a zvolit si barvu pozadí, která nám bude sloužit jako tlačítka, tedy každý znak s tímto pozadím bude odeslán zpět do mikrokontroleru. Pro nastavovací příkazy slouží předpona $$S a po ní již může následovat libovolný počet nastavovacích příkazů. Nastavení šířky terminálu provádíme sekvencí sidepanel:small; pro 14 znakový terminál a sidepanel:wide; pro 21 znakový terminál. Nastavení odesílacího pozadí budeme chtít provést na čistě bílou a to příkazem clickclr:47.1;. Tyto příkazy budeme používat ve dvou případech, a to při vstupu do režimu MENU, kdy nastavujeme odesílací pozadí a velikost terminálu na 14 znaků odesláním příkazu $$Sclickclr:47.1; sidepanel:small;. Dalším použitím pak bude přechod do režimu HELP, kdy nastavujeme šířku terminálu na 21 znaků, a to odesláním příkazu $$Ssidepanel:wide;.

Po zapnutí samotného Data Plotter-u nebo mikrokontroleru budeme chtít obě aplikace dostat do jejich výchozích nastavení. Pro to abychom toho docílili však nejprve musíme zajistit, aby se o sobě obě strany po připojení o sobě dozvěděli. Na straně Data Plotter-u tedy povolíme v režimu

nastavení vymazání po připojení, „autoset“ po připojení, odpověď na echo a jako znak odeslaný do mikrokontroleru po připojení nastavíme na ‘q‘, který pokaždé resetuje nastavení MCU a vrátí jej do režimu MENU. Toto nastavení zobrazené na obrázku 4.10 lze opět uložit do defaultního nastavení Data Plotter-u stejně jako jsme to udělali po nastavení UART viz. 4.4. S tímto nastavením tedy Data Plotter po připojení k UART vymaže všechna data přijatá do přijímací konzole, uživatelského terminálu nebo vykreslená do grafu, dále pak vyšle mikrokontroleru znak ‘q‘ jež uvede MCU do výchozího nastavení a bude čekat na příchod echo zprávy, na kterou odpoví a dá tak najevo svou připravenost k příjmu dat.



Obrázek 4.10: Nastavení potvrzení připojení v Data Plotter

Na straně mikrokontroleru budeme požadovat, aby při každém přijetí znaku ‘q‘ vypnul všechna měření a přešel do režimu MENU, což bude díky struktuře naší aplikace v MCU a přijímání znaků v přerušení automaticky zajištěno. Ovšem musíme určit chování mikrokontroleru po jeho restartu, kdy se musí pokusit spojit s PC aplikací a vyčkat na potvrzení spojení. V programu to tedy budeme řešit periodickým odesíláním zpráv do doby něž přijde nějaký znak jako odpověď. Po inicializaci všech potřebných periferií tedy začneme vykonávat cyklus, kde nejprve odešleme do PC aplikace echo zprávu se znakem ‘q‘, na kterou očekává mikrokontroler odpověď opět znakem ‘q‘, čímž by dal Data Plotter potvrzení, že je připojen. Ovšem pokud by zde byla odpověď na echo zprávy zakázána a ani by uživatel nenastavil odesílání znaku při připojení, mikrokontroler by nedostal žádné potvrzení o funkčním spojení a nevykreslil by uživatelský terminál. Proto ve smyčce budeme současně vykreslovat terminál MENU uživatel tak potvrdí připojení mikrokontroleru kliknutím na některé z ovládacích tlačítek. Kód realizující tuto funkci pak vidíme zde:

```
while (rec_char == '␣') {
    printf("$$Eq");
    fflush(stdout);
    print_menu();
    LL_mDelay(500);
}
```

Kde znak mezery uložený v proměnné rec_char indikuje, že zatím nebyl přijat žádný znak, dále pak funkce fflush(stdout); nám zajistí neprodlené odeslání všech znaků zadaných k odeslání a nakonec je tu funkce čekání 500ms a to abychom zbytečně nezahltili Data Plotter v případě že by čekal na potvrzení připojení ze strany uživatele.

4.6 PWM Generátor

Pro realizaci PWM Generátoru musíme nejprve vybrat vhodný čítač, který by umožňoval výstup PWM. Při náhledu do technického listu [2] zjišťujeme, že na 8 pinovém pouzdře máme možnost využít buď fyzický pin č.4 nebo fyzický pin č.8, jelikož na ostatních pinech buď vyvedený čítač s PWM výstupem nenalezneme, případně je zde již jiná funkce, jako napájení, UART nebo vstup referenčního hodinového signálu OSC32_IN, který použijeme pro zpřesnění měření. Na fyzickém pinu 4 je vyveden pouze čítač

TIM2 ovšem tento čítač je zřejmě nejkomplexnějším z našeho mikrokontroleru a bylo by tedy vhodné ušetřit si ho pro realizaci složitějších funkcí. Na fyzickém pinu 8 je potom možnost generovat PWM signál pomocí TIM1, TIM3 nebo TIM16. Vzhledem k tomu, že TIM1 a TIM3 jsou opět poměrně komplexní a umožňují mnohé nadstandartní funkce oproti TIM16, které bychom později mohli využít jinde, bude vhodné pro PWM Generátor vybrat čítač TIM16, který bude v následujících kódech značen také jako TIM_GEN.

4.6.1 Nastavení pro PWM generátor

Pro nastavení čítače do režimu generátoru PWM nejprve vhodně nastavíme výstupní pin PB6, tedy jeho mód, alternativní funkci odpovídající výstupu TIM16, rychlost, typ výstupu a volbu použití pull-up/pull-down rezistoru, k čemuž opět využijeme automaticky generovaný kód vývojovou aplikací. Dále pak do periferie čítače pustíme hodinový signál, povolíme „pre-load“, aby čítač generoval PWM signál cyklicky, nastavíme ho do režimu MODE_PWM1 a povolíme „fast mode“, kvůli dosažení co nejostřejších hran generovaného signálu, a to následujícím kódem:

```
RCC->APBENR2 |= LL_APB2_GRP1_PERIPH_TIM16;

TIM16->CCMR1 |= TIM_CCMR1_OC1PE;
TIM16->CCMR1 |= LL_TIM_OCMODE_PWM1;
TIM16->CCMR1 |= TIM_CCMR1_OC1FE;
```

Potom ještě zablokujeme funkci bloku „Break and dead time“, který za určitých podmínek, nezávisle na běhu mikrokontroleru, blokuje výstup PWM, čehož se využívá například pro odpojení řízení stroje v případě jeho přetížení. A nakonec povolíme výstup PWM na kanál CH1N, kde se pro TIM16 nachází požadovaný výstup na pin PB6. Zmíněná nastavení jsme provedli následujícími příkazy:

```
TIM16->BDTR |= TIM_BDTR_MOE;
TIM16->CCER |= LL_TIM_CHANNEL_CH1N;
```

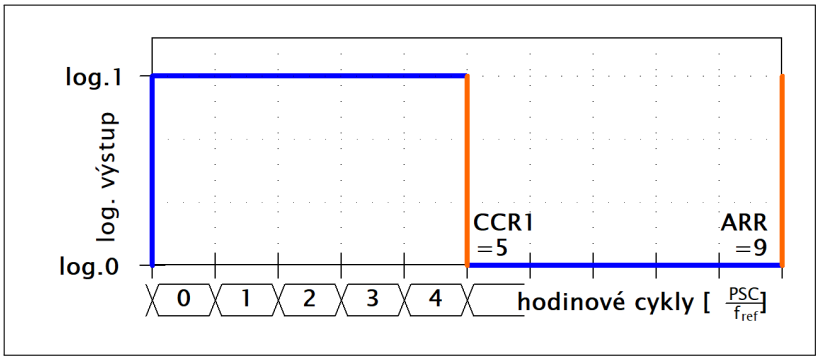
Po provedení základního nastavení již můžeme zadat do příslušných registrů čítače parametry požadovaného PWM signálu a začít signál generovat. Jakým způsobem to budeme dělat, vyčteme z principu čítače v režimu PWM Generátoru, který je znázorněn na obrázku 4.11, kde vidíme že čítač po vynulování generuje log.1 až do načítání hodnoty uložené v jednotce CCR1, kdy začne generovat na svém výstupu log.0, tentokrát do chvíle než čítač dosáhne hodnoty uložené v ARR registru, kdy se opět nuluje. Samotná perioda T je pak určena z hodnoty ARR registru, referenční frekvence f_{ref} a velikosti před-děličky PSC, jako:

$$T = \frac{(PSC + 1)(ARR + 1)}{f_{ref}}, \tag{4.2}$$

kde přičítání jedničky k hodnotám registrů kompenzuje jejich číslování od nuly, tedy například pokud bude v před-děličce PSC hodnota nula bude se tento blok chovat, jako by dělil referenční frekvenci jedničkou. Střída signálu T_s je potom určena právě hodnotou CCR1 a opět referenční frekvencí a před-děličkou, jako:

$$T_s = \frac{(PSC + 1)CCR1}{f_{ref}}, \tag{4.3}$$

kde již od CCR1 jedničku neodčítáme.



Obrázek 4.11: Princip čítače v režimu PWM

4.6.2 Řídící program PWM generátoru

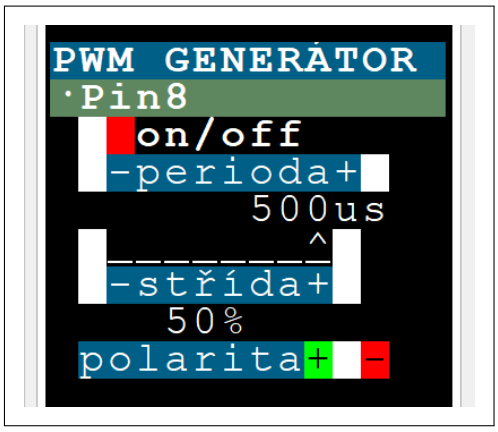
Nyní již můžeme přenést výše uvedená nastavení periody a střídy do řídícího programu tak, aby šlo oba parametry interaktivně měnit. Frekvenci lze teoreticky měnit od 32MHz, kdy je PSC rovné nule, ARR a CCR1 rovné jedné a výstupní signál tak bude mít střídu 50%, až do frekvence, kdy bude PSC i ARR rovno jejich maximální hodnotě a dohromady tak budou dělit 64 MHz vstupního signálu číslem 2^{32} což představuje periodu výstupního signálu cca 67s. Pro výukové účely ovšem nebude potřeba generovat vysoké frekvence, a tedy maximální frekvenci omezíme na 1MHz, pro jejíž generování nastavíme PSC na dělení 32 a ARR na 1 čímž bude omezena volba střídy pouze po 50%, ovšem pro tyto vyšší frekvence nám to vadit nebude. Zvyšováním hodnoty ARR pak budeme moci dosáhnout periody signálu až 65ms, což ovšem bude potřeba zvýšit, a tak pro dosažení vyšší periody zvýšíme PSC. Zvyšováním PSC však přicházíme o možnost jemného nastavení konkrétní délky periody, proto jej budeme měnit vždy jen na desetinásobek předchozího. Jakým způsobem budeme nastavovat PSC pro jednotlivé délky period a jaké tomu budou odpovídat rozlišení ladění délky periody je zaznamenáno v následující tabulce 4.1, ze které také vidíme, že maximální generovanou periodou bude něco přes 32s.

Perioda $[ms]$ **horní mez**	**PSC**	**Nejmenší laditelný dílek** $[\mu s]$
32,767	32 - 1	1
327,67	320 - 1	5
3 276,7	3 200 - 1	50
32 767	32 000 - 1	500

Tabulka 4.1: Přehled použitého nastavení před-děličky pro konkrétní periody

Ještě, než začneme tvořit samotné funkce realizující změnu periody a střídy, vytvoříme si pro PWM Generátor uživatelské rozhraní v terminálu Data Plotter-u, jak vidíme na obrázku 4.12. První ovládací tlačítko on/off bude sloužit pro samotné zapnutí/vypnutí generátoru. Dále zde bude nastavení periody, u které chceme mít možnost měnit každou cifru zvlášť, tak aby bylo možné nastavit požadovanou velikost periody. Z toho důvodu zde zrealizujeme přepínatelný ukazatel na ovládanou cifry a tlačítka pro zvyšování hodnoty cifry a snižování hodnoty této cifry. U střídy pak nebude důležité její jemné nastavování, a tak si vystačíme se dvěma tlačítky, kde jedno bude střídu o 5% zvyšovat a druhé o 5% snižovat. Nakonec zde ještě přidáme možnost volby polarity generovaného signálu, a to na kladnou nebo zápornou, které způsobí invertování výstupního signálu, a to i při vypnutém generování PWM, tedy pokud bude polarita kladná a generování PWM vypnuté bude na výstupním pinu log.0 a pokud

bude polarita záporná (invertovaná) a generování PWM vypnuté bude na výstupním pinu log.1. V každém tlačítku terminálu se pak skrývá jeden ovládací znak a to které znaky byli na konkrétní funkce použity vidíme v tabulce 4.2.



Obrázek 4.12: Náhled na uživatelský terminál PWM Generátoru

Znak	Funkce
'g'	zapnutí/vypnutí generování PWM
'y'	perioda -
'x'	perioda +
'f'	cifra periody -
'j'	cifra periody +
'c'	střída -
'v'	střída +
'B'	změna polarity
'q'	návrat do MENU

Tabulka 4.2: Přehled použitých ovládacích znaků PWM Generátoru

Jelikož máme již cestu z čítače na výstupní pin nastavenou, bude zapnutí generování PWM signálu realizováno pouze povolením čítání čítače:

```
GEN_TIM->CR1 |= TIM_CR1_CEN;
```

vypínání pak bude obdobné, čítání zakážeme, ovšem tentokrát ještě stav čítače nastavíme na nejvyšší možný a to, aby na výstupu zůstala defaultní logická hodnota, tedy pro kladnou polaritu log.0 a pro tu inverzní log.1. Druhým důvodem tohoto nastavení je, že čítač se po zapnutí první hranou vynuluje a začne tak generovat periodu PWM od začátku:

```
GEN_TIM->CR1 &= ~TIM_CR1_CEN;
GEN_TIM->CNT = 0xFFFF;
```

Dalším nastavitelným prvkem bude délka periody, u které nejprve nastavíme požadovanou cifru, kterou si program uchovává v proměnné peri_ptr a následně po příchodu požadavku na zvýšení/snížení periody

v dané cifře dopočte příslušný přírůstek disc, který musí být z principu nastavení PSC dvojnásobkem požadované změny v μs, jak vidíme v následujícím kódu:

```
uint32_t disc = 2;
for (uint8_t i = 0; i < peri_ptr; i++) disc *= 10;
```

přírůstek disc se potom podle požadavku přičte či odečte od aktuální délky periody uložené opět, jako její dvojnásobek, v proměnné peri. Posledním krokem nastavení délky periody bude její zapsání do registru ARR, což pro vede následovně:

```
GEN_TIM->ARR = peri_arr-1;
```

kde $peri\_arr = \frac{peri}{x}$ kdy x bude buď 1, 10, 100 nebo 1000 a to podle velikosti periody, viz. tabulka 4.1, kde současně podle velikosti periody nastavíme i PSC:

```
GEN_TIM->PSC = prsc;
GEN_TIM->EGR |= TIM_EGR_UG;
```

kde prcs bude rovné konkrétní hodnotě před-děličky pro danou velikost periody a druhý příkaz pak vygeneruje „update event“, tedy vynutí si u čítače, aby aktualizoval hodnoty PSC a ARR registrů, jinak by to udělal až za jednu periodu, což může být v našem případě až za 32s.
Dalším realizovaným nastavením bude velikost střídy, jež bude udávána v procentech a její hodnota bude uložena a uživatelsky upravována v proměnné perc. Samotná střída pak bude určena poměrem CCR1 a ARR, kde ARR je číslované od nuly a tak střídu v procentech spočítáme, jako:

$$perc[\%] = \frac{CCR1}{ARR+1}100. \tag{4.4}$$

Odtud tedy odvodíme výpočet složky CCR1 a v programu ji zapíšeme následujícím kódem:

```
GEN_TIM->CCR1 = ((GEN_TIM->ARR+1)*perc)/100;
```

Posledním uživatelsky ovládaným parametrem PWM Generátoru bude jeho polarita, která dokáže invertovat výstup generátoru a periodu tak můžeme začít stavem v log.0 na místo obvyklého začátku v log.1. Změna polarity také umožňuje při vypnutém generování PWM na výstup měnit logický stav na výstupním pinu. Tím lze nejen přivést na výstupní pin log. 1 nebo log.0, ale i při změně polarity generovat vzestupné a sestupné hrany, což využijeme například pro odstartování nějakého měření, kde potřebujeme ostrou hranu bez zákmitů, kterou například při sepnutí mechanického tlačítka zaručit nedokážeme. Nastavení polarity pak bude velice jednoduché, jelikož je řízeno pouze jedním bitem v CCER registru, jak vidíme v následujícím kódu, kde první řádek slouží pro nastavení inverzního výstupu a druhý vrací polaritu do původního nastavení:

```
GEN_TIM->CCER |= TIM_CCER_CC1NP;
GEN_TIM->CCER &= ~TIM_CCER_CC1NP;
```

Skrytou funkcionalitou pak bude ještě zablikání s LED připojenou na výstup PWM Generátoru po každém restartu mikrokontroleru, což provedeme velice jednoduše pomocí změn polarity a funkcí čekání z LL knihovny:

```
GEN_TIM->CCER |= TIM_CCER_CC1NP;
LL_mDelay(1000);
for (uint8_t i = 0; i < 5; i++) {
    GEN_TIM->CCER |= TIM_CCER_CC1NP;
    LL_mDelay(50);
    GEN_TIM->CCER &= ~TIM_CCER_CC1NP;
    LL_mDelay(50);
}
```

4.7 Osciloskop

Ideální volbou pro pin osciloskopu bude na nejmenším osmi pinovém pouzdře pin č.7, který zatím není obsazen žádnou funkcionalitou a je umožněno jeho použití jako vstupu do AD převodníku, jak se můžeme přesvědčit v technickém listu [2]. Vyvedení vstupu AD převodníku nalezneme i na dalších pinech, ovšem na tento pin žádná jiná periferie, kterou budeme používat pro realizaci čítače a dalších částí přístroje, vyvedena není, a tak tento pin dokážeme využít pouze jako vstup kanálu osciloskopu. Dále si musíme zvolit vhodný čítač který nám zajistí časovou základnu osciloskopu, tedy čítač, který bude generovat impulzy pro spouštění ADC, a to v přesných časových intervalech vzorkovací periody. Pro tuto volbu tedy budeme požadovat čítač, jehož výstup je připojen na spouštěcí vstup ADC. Tento požadavek splňují hned tři čítače a to TIM1, TIM2 a TIM3, jelikož TIM2 má jako jediný velikost CNT 32 bitů ušetříme si ho pro použití v čítači, TIM1 má potom více CC jednotek, a tak bude nejvhodnější použít čítač TIM3, který bude v následujících kódech značen i jako OSC_TIM.

4.7.1 Nastavení pro osciloskop

Jako první se budeme zaobírat nastavením vstupního prvku osciloskopu, kterým je analogově digitální převodník. Jak již bylo zmíněno budeme pro vstup do ADC využívat pin č.7 na pouzdře SO8N, a tedy používat pin PA13 kam je vyveden kanál 17 ADC. Nastavení pinu PA13 opět necháme na automaticky generovaném kódu, který nastaví mód pinu na analogový. Dále budeme pokračovat zapnutím hodin do periferie ADC a nastavení před-děličky tohoto signálu na 2 což je nejnižší možná hodnota pro mód, ve kterém budeme ADC provozovat:

```
RCC->APBENR2 |= LL_APB2_GRP1_PERIPH_ADC;
ADC1->CFGR2 |= LL_ADC_CLOCK_SYNC_PCLK_DIV2;
```

Budeme pokračovat nastavením výsledku konverze na 8 bitů, což sice plně nevyužívá možnosti převodníku, ale za to umožní šetřit paměť při stále pro nás dostatečném rozlišení. Povolením přepisu dat pak umožníme, aby v případě, že si jádro od ADC výsledek konverze nevyzvedne, přepsalo výsledek konverze předchozí na nový. Pokud bychom toto neudělali při nevyzvednutí jediného vzorku by se přestal výsledek nové konverze do výstupního registru řádně ukládat. V následujícím kódu pak ještě nastavujeme, že bude každá konverze spouštěna náběžnou hranou signálu od TIM3:

```
ADC1->CFGR1 |= LL_ADC_RESOLUTION_8B;
ADC1->CFGR1 |= LL_ADC_REG_OVR_DATA_OVERWRITTEN;
ADC1->CFGR1 |= LL_ADC_REG_TRIG_EXT_TIM3_TRGO;
```

Dále nastavíme zdroj referenčního napětí pro ADC, který budeme používat interní a vybereme za vstup kanál 17, na kterém se nachází pin PA13:

```
ADC1->CR |= ADC_CR_ADVREGEN;
ADC1->CHSELR |= (LL_ADC_CHANNEL_17 & ADC_CHANNEL_ID_BITFIELD_MASK);
```

Pro realizaci funkce „trigger“ u osciloskopu budeme používat AWD „Analog WatchDog“, který bude sledovat výslednou hodnotu konverze a na jejím základě vyvolávat přerušení. Nastavíme tedy nejprve kanál 1 AWD na kanál 17 vstupu ADC a následovně povolíme přerušení od celého převodníku:

```
ADC1->CFGR1 |= (1<<23) | (17<<26);
NVIC_SetPriority(ADC1_IRQn, 0);
NVIC_EnableIRQ(ADC1_IRQn);
```

kde přerušení od AWD bude vykonáváno jako přerušení příchozí od ADC se značkou definující původce přerušení jako AWD.

Nutné pak bude ještě nastavení DMA, které bude řídit přenos naměřených dat z ADC do paměti. Nejprve tedy pustíme hodinový signál do DMA:

```
RCC->AHBENR |= LL_AHB1_GRP1_PERIPH_DMA1;
```

Nastavení pak bude pokračovat výběrem kanálu DMA, který si vybereme hned první nepoužitý, tedy kanál 1. Defaultně je zde nastaven přenos po jednom bajtu, který požadujeme a nastavíme tedy jen inkrementování v paměti, tedy DMA bude plnit bajt za bajtem námi určené datové pole a kruhový mód, který umožňuje, aby DMA po naplnění tohoto pole začalo plnit pole opět od začátku a přepisovat tak stará data. Tento režim se musí nastavit i v ADC, a to povolením neomezeného přenosu DMA:

```
DMA1_Channel1->CCR |= DMA_CCR_MINC;
DMA1_Channel1->CCR |= DMA_CCR_CIRC;
ADC1->CFGR1 |= LL_ADC_REG_DMA_TRANSFER_UNLIMITED;
```

Po nastavení samotného DMA musíme nastavit i spojení mezi ním a ADC, což uděláme pomocí DMA přepínače (DMAMUX), jehož kanály jsou číslovány od nuly, a tak pro spojení používaného kanálu 1 u DMA musíme nastavit kanál 0 u DMAMUX:

```
DMAMUX1_Channel0->CCR |= LL_DMAMUX_REQ_ADC1;
```

Teď už jen potřebujeme nastavit TIM3, který se bude starat o začátek každé konverze. To bude poměrně snadné, jelikož pouze povolíme hodinový signál, vybereme jako zdroj referenční frekvence pro čítač, interní zdroj hodinového signálu a na konec nastavíme generování výstupního spouštěcího signálu při každém nulování čítače:

```
RCC->APBENR1 |= LL_APB1_GRP1_PERIPH_TIM3;
TIM3->SMCR |= LL_TIM_CLOCKSOURCE_INTERNAL;
TIM3->CR2 |= LL_TIM_TRGO_UPDATE;
```

Po kompletním nastavení ADC je pak ještě nutné jej zkalibrovat. Tedy prvním příkazem za následujícího kódu zapneme kalibraci a budeme v cyklu čekat na její ukončení:

```
ADC1->CR |= ADC_CR_ADCAL;
while ((ADC1->CR & ADC_CR_ADCAL) == (ADC_CR_ADCAL));

LL_mDelay(20);
ADC1->CR |= ADC_CR_ADEN;
```

kde po krátké pauze po dokončení kalibrace ještě povolíme samotnou funkci ADC.

4.7.2 Řídící program osciloskopu

Nyní si budeme potřebovat rozvrhnout všechny funkce osciloskopu tak, abychom mohli vytvořit terminálové rozhraní v Data Plotter-u. Základní funkcionalitou bude tlačítko pro vypnutí a zapnutí měření, které stejně, jako u PWM Generátoru označíme on/off, jak vidíme na obrázku 4.13. Další terminálové tlačítko bude mít na starost přepínání mezi třemi volitelnými režimy. Pro režimy NORMAL a SINGLE pak budeme potřebovat nastavení „trigger“ napětí a „trigger“ hrany. Jemnost nastavení „trigger“ napětí bude pro naše potřeby postačovat malá a budeme tedy toto napětí měnit dvěma tlačítky, přičemž jedním budeme napětí zmenšovat o 0,2 V, a to maximálně na 0,2V a druhým tlačítkem pak napětí zvětšovat o 0,2V, a to až na možné 3 V. Hranu, na kterou bude „trigger“ probíhat pak bude možné přepínat jedním tlačítkem, a to mezi náběžnou a sestupnou. Důležitým parametrem, který budeme nastavovat, je potom vzorkovací perioda, tedy perioda, se kterou bude probíhat odměr jednotlivých vzorků z ADC. Vzorkovací periodu budeme volit dvěma tlačítky, kdy jedním budeme

přidávat, a to až na 200μs a druhým pak ubírat až na 0,5μs, což odpovídá vzorkovací rychlosti 2 Msample/s, kde jsme tuto hodnotu zvolili jako hodnotu menší než maximální vzorkovací rychlost, kterou ADC zvládá 2,5Msampl/s. Ovládací znaky, které byly použity při tvorbě tlačítek v terminálu pak nalezneme v tabulce 4.3.

Obrázek 4.13: Náhled na uživatelský terminál Osciloskopu

Znak	Funkce
's'	zapnutí/vypnutí osciloskopu
'w'	volba režimu
'e'	trigger napětí -
't'	trigger napětí +
'z'	volba trigger hrany
'k'	vzorkovací perioda -
'l'	vzorkovací perioda +
'q'	návrat do MENU

Tabulka 4.3: Přehled použitých ovládacích znaků Osciloskopu

Před implementací samotného spouštění měření v různých režimech osciloskopu bude nutné nastavit adresy, odkud a kam má DMA přenášet data, což provedeme tím, že zapíšeme do adresy periferie adresu registru převodníku, do kterého se ukládají výsledky převodu. Do adresy paměti, kam se mají výsledky AD převodů ukládat pak uložíme adresu pro výsledky alokovaného pole adc_buf[]. DMA pak ještě potřebuje vědět velikost tohoto pole, kterou jsme předvolili na 3kB:

```
#define SAMPLE_NUM 4096
DMA1_Channel1->CPAR = (uint32_t)(&ADC1->DR);
DMA1_Channel1->CMAR = (uint32_t)adc_buf;
DMA1_Channel1->CNDTR = SAMPLE_NUM;
```

Dále potřebujeme nastavit čítač spouštějící AD konverzi. U něj budeme nastavovat jednak před-děličku, která bude trvale nastavena na dělení 16:

```
OSC_TIM->PSC = 16-1;
```

a dále ARR registr, který bude určovat šířku samotné periody. Díky nastavení před-děličky budeme do ARR registru ukládat vždy čtyř násobek požadované periody v μs. Velikost vzorkovací periody bude určovat ukazatel sampl_per_ptr, který bude uživatelsky přičítán a odčítán. Po každé úpravě tohoto ukazatele z terminálu pak bude nová vzorkovací perioda nastavena, jako:

```
const uint16_t sampl_per_arr[] = {1,2,4,10,20,40,100,200,400};
OSC_TIM->ARR = (2*sampl_per_arr[sampl_per_ptr])-1;
```

Jak vidíme z předešlého kódu, proměnná sampl_per_arr[] obsahuje dvojnásobky nastavitelné vzorkovací periody v μs.

Režim AUTO

V tomto režimu bude mikrokontroler cyklicky spouštět vzorkování vzorkovacího okna, čekat do naplnění celé paměti a následně vypisovat naměřená data na terminál. Po zapnutí osciloskopu v tomto režimu tedy nejprve povolíme přenos pomocí DMA, vynulujeme čítač vzorkovací periody a spustíme ho. Nakonec ještě spustíme ADC konverzi:

```
DMA1_Channel1->CCR |= DMA_CCR_EN;
OSC_TIM->CNT = 0;
OSC_TIM->CR1 |= TIM_CR1_CEN;
MODIFY_REG(ADC1->CR,ADC_CR_BITS_PROPERTY_RS,ADC_CR_ADSTART);
```

Po spuštění konverze již pouze budeme čekat, než DMA naplní celé 3kB vzorkovacího okna. To provedeme v cyklu, kde budeme sledovat hodnotu DMA1_Channel1->CNDTR udávající počet bajtů, která mají být přeneseny k zaplnění požadovaného datového bloku, tedy jeho hodnota bude klesat od SAMPLE_NUM k nule. Po tom co se objeví v DMA1_Channel1->CNDTR nula, ukončíme měření vypnutím ADC, zakázáním DMA a přerušením čítání čítače OSC_TIM:

```
MODIFY_REG(ADC1->CR,ADC_CR_BITS_PROPERTY_RS,ADC_CR_ADSTP);
DMA1_Channel1->CCR &= ~DMA_CCR_EN;
OSC_TIM->CR1 &= ~TIM_CR1_CEN;
```

Tímto tedy máme naměřené vzorkovací okno režimu AUTO v proměnné adc_buf[] odkud následně vypíšeme data do Data Plotter-u a to totožným způsobem, jako v následujících režimech.

Režim NORMAL a SINGLE

Pro tyto dva režimy bude důležitá funkce „trigger“, která spočívá v přerušeních vyvolávaných jednotkou AWD na základě změřené hodnoty napětí. Princip této jednotky je následující, do AWD se zadají dvě hodnoty, které představují hlídané pásmo a pokud výsledek AD konverze bude ležet mimo toto pásmo AWD vyvolá přerušení. V našem případě budeme tímto principem detekovat náběžné a sestupné hrany. To pro náběžnou hranu provedeme nejprve nastavením hlídaného pásma od „trigger“ napětí, uloženého a uživatelsky upravovaného v proměnné trig_volt, až po maximum rozsahu a povolíme přerušení od AWD:

```
ADC1->AWD1TR = ((4095<<16) | ((trig_volt<<7)-50));
ADC1->IER |= LL_ADC_IT_AWD1;
```

Kde odečtení 50 v prvním příkazu je pro to, aby se pásma nad a pod „trigger“ napětím překrývaly a zabránilo se tak vyvolání několika přerušení za sebou vlivem rušivého zakmitání okolo této hodnoty. Po přechodu měřeného signálu pod hranici „trigger“ napětí vyvolá AWD přerušení, a to nastaví hlídané pásmo tentokrát od nuly do „trigger“ napětí a, jako ve všech rutinách přerušení se zde vymaže příznak daného přerušení:

```
ADC1->AWD1TR = ((trig_volt<<7)+50)<<16;
ADC1->ISR = LL_ADC_FLAG_AWD1;
```

Přechod signálu opět nad „trigger“ napětí pak bude ona hledaná náběžná hrana a AWD opět vyvolá přerušení, ve kterém již bude čekat na doměření vzorkovacího okna a následně zastaví měření a výsledky vypíše. Při požadavku na začátek měření na sestupnou hranu pak budeme do AWD nejprve zadávat hlídané okno pod „trigger“ napětí a po příchodu prvního přerušení okno nad „trigger“ napětím.

Při spouštění měření v tomto režimu tedy postupujeme nejprve zakázáním přerušení ADW, které mohlo zůstat povolené z předchozího měření, dále pak nastavíme konkrétní hlídané okno pomocí funkce osc_set_trig_volt(uint8_t up_down), která zajistí volbu správného okna pro aktuálně zvolenou „trigger“ hranu uloženou a uživatelsky ovládanou v proměnné osc_edge:

```
ADC1->IER &= ~LL_ADC_IT_AWD1;
osc_set_trig_volt(!osc_edge);
```

Dále pak zapneme samotnou konverzi povolením DMA, vynulováním a povolením čítače OSC_TIM a začátku ADC konverze:

```
DMA1_Channel1->CCR |= DMA_CCR_EN;
OSC_TIM->CNT = 0;
OSC_TIM->CR1 |= TIM_CR1_CEN;
MODIFY_REG(ADC1->CR,ADC_CR_BITS_PROPERTY_RS,ADC_CR_ADSTART);
```

Po zapnutí konverze musí program ještě nějakou dobu čekat, než stihne ADC naměřit počet vzorků o velikosti „pre-trigger“, což odpovídá cca 100ms bezpečně pro všechny vzorkovací periody. Pokud bychom toto čekání vypustili a zaznamenali bychom „trigger“ hranu okamžitě po zapnutí konverze, všechny „pre-trigger“ vzorky by byly stará neplatná naměřená data, která se v paměti nestihla přepsat novými. Po pauze již můžeme povolit přerušení, samozřejmě s předchozím vymazáním příznaku přerušení, a pokračovat v další programové rutině, dokud nepřijde přerušení od AWD:

```
LL_mDelay(100);
ADC1->ISR = LL_ADC_FLAG_AWD1;
ADC1->IER |= LL_ADC_IT_AWD1;
```

Po prvním příchodu očekávaného přerušení od AWD se potom přehodí hlídané okno a vymaže příznak přerušení:

```
osc_set_trig_volt(osc_edge);
ADC1->ISR = LL_ADC_FLAG_AWD1;
```

Po příchodu druhého přerušení pak již přerušení zakážeme a odečteme aktuální pozici DMA přenosu v datovém poli, kam jsou jednotlivé vzorky, jako do kruhového buffer-u, ukládány. Nakonec pak ještě za pomoci aktuální pozice ukazatele, velikosti „pre-trigger“ a celkového počtu vzorků dopočteme konečnou pozici osc_end_pos na kterou budeme chtít, aby se ukazatel do konce měření dostal:

```
#define PRETRIGGER 400
ADC1->IER &= ~LL_ADC_IT_AWD1;
uint16_t start_pos = SAMPLE_NUM-DMA1_Channel1->CNDTR;
osc_end_pos = ((start_pos+SAMPLE_NUM)-PRETRIGGER)%SAMPLE_NUM;
```

Takto získanou počáteční a koncovou pozici v buffer-u využijeme v následujícím cyklu, kde budeme čekat, dokud DMA nenaplní buffer od počáteční do koncové pozice:

```
uint16_t x = SAMPLE_NUM-DMA1_Channel1->CNDTR;
if(osc_end_pos < start_pos){
    while(x < osc_end_pos || x >= start_pos)
            x = SAMPLE_NUM-DMA1_Channel1->CNDTR;
}else{
    while(!(x > osc_end_pos || x < start_pos))
            x = SAMPLE_NUM-DMA1_Channel1->CNDTR;
}
```

Po žádaném naplnění buffer-u pak pouze vypneme konverzi, DMA přenos i čítač:

```
MODIFY_REG(ADC1->CR,ADC_CR_BITS_PROPERTY_RS,ADC_CR_ADSTP);
DMA1_Channel1->CCR &= ~DMA_CCR_EN;
OSC_TIM->CR1 &= ~TIM_CR1_CEN;
```

Teď již naměřená data vypíšeme, jak bude popsáno níže, a v případě že je aktivní režim NORMAL opět spustíme další měření, povolíme přerušení AWD a vykonáváme ostatní program až do příchodu přerušení. V případě aktivního režimu SINGLE potom osciloskop na terminálu označíme jako vypnutý a nové měření začneme až po opětovném zapnutí osciloskopu uživatelem.

Výpis naměřených dat do Data Plotter-u

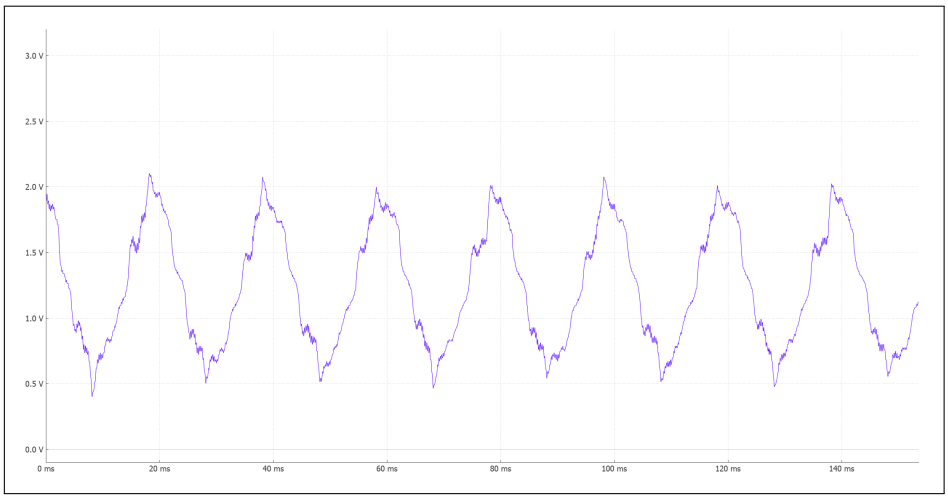
Po kompletním naměření vzorkovacího okna nám naměřená data zůstanou v datovém poli adc_buf[], kde se první vzorek nachází na adrese osc_end_pos+1, kdy DMA právě po dříve vypočtenou adresu osc_end_pos ukládalo jednotlivá měření. Dále jsou v datovém poli uloženy vzorky „pre-trigger“ a nakonec pak zbytek naměřených odměrů vzorkovacího okna. Jako první program pro výpis těchto hodnot upraví proměnnou osc_end_pos tak, aby ukazovala na první vzorek. Dále začne odesílat hlavičku vzorkovacího okna do Data Plotter-u, kdy nejprve zadá příznak, že jsou posílána data pro kanál jedna, následně vypočte a vypíše vzorkovací periodu, tedy čas mezi jednotlivými vzorky. Další částí hlavičky bude celkový počet vzorků, jejich velikost, tedy osm bajtů, význam maximální hodnoty vzorku, což bude pro nás napětí 3,3 V a formát ve kterém je budeme posílat, který jsme určili jako „unsigned integer“ jedno-bajtový (U1). Kód tohoto nastavení pak vidíme na následujících řádcích:

```
osc_end_pos++;
printf("$$C1,");
uint32_t sampl_float = sampl_per_arr[sampl_per_ptr]*5;
print_float(sampl_float, 7);
printf(",%d,8,3.3;U1",SAMPLE_NUM);
```

Teď již Data Plotter očekává příchod jednotlivých vzorků, každý reprezentovaný jedním bajtem. Budeme tedy v cyklu odesílat jednotlivé bajty naměřených dat z pole v němž jsou uloženy. Výpočet ukazatele na aktuálně odesílaný bajt pak musíme normalizovat velikostí buffer-u, jak vidíme v následujícím kódu:

```
for(uint16_t i=0; i<SAMPLE_NUM;i++){
    putchar(adc_buf[(osc_end_pos+i)%SAMPLE_NUM]);
    fflush(stdout);
}
printf(";");
fflush(stdout);
```

Kde jsme ještě po odeslání všech vzorků ukončili přenos středníkem. Náhled konkrétního naměřeného vzorkovacího okna pak vidíme na obrázku 4.14.



Obrázek 4.14: Náhled na naměřené vzorkovací okno v Data Plotter-u

4.8 Logický analyzátor

Logický analyzátor bude v principu velice rychle vzorkovat logickou úroveň na měřicím pinu. Stejným způsobem fungují i mnohé periferie MCU pro datovou komunikaci, jedno takovou periferií je SPI, kterou lze nakonfigurovat pro funkci v režimu I2S (sběrnice pro přenos audio signálu), jež bude vhodná pro realizaci logického analyzátoru nejen pro jednoduché vyčítání hodnot z periferie, ale i pro dostatečnou vzorkovací frekvenci, která lze nastavit až na 16 MHz. V tomto režimu bude SPI nastavenou vzorkovací frekvencí neustále vzorkovat logickou hodnotu na vstupním pinu a pomocí posuvného registru z těchto vzorků skládat 16bitová slova, která lze pak z periferie vyčíst, uložit do paměti a reprezentovat je, jako záznam logického signálu na vstupním pinu mikrokontroleru. Při volbě vhodného fyzického pinu pro umístění datového vstupu analyzátoru na pouzdře S08N máme dvě možnosti, a to pin č.4 nebo č.8, kde vhodnějším by byl pin č.4, na kterém je již umístěn vstup čítače, ovšem kvůli nutnosti vyvedení hodinového signálu až na výstupní bránu musíme na pin č.4 vyvést hodinový signál a datový signál tak bude vyveden na fyzický pin č.8. Z této volby však vidíme, že nebude možné současně s logickým analyzátorem používat čítač a kvůli omezené velikosti SRAM ani osciloskop, jehož vzorkovací buffer budeme využívat pro uložení záznamu z analyzátoru.

4.8.1 Nastavení pro logický analyzátor

Nastavení začneme povolením hodin do periferie SPI, nastavením datového vstupu I2S na pinu PB5 a hodinového signálu na pinu PA1:

```
RCC->CCIPR |= LL_RCC_I2S1_CLKSOURCE_SYSCLK;
RCC->APBENR2 |= LL_APB2_GRP1_PERIPH_SPI1;

LL_GPIO_InitTypeDef GPIO_InitStruct = { 0 };
GPIO_InitStruct.Pin = LL_GPIO_PIN_5;
GPIO_InitStruct.Mode = LL_GPIO_MODE_ALTERNATE;
GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_VERY_HIGH;
GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
GPIO_InitStruct.Alternate = LL_GPIO_AF_0;
```

```
    LL_GPIO_Init(GPIOB, &GPIO_InitStruct);
```

```
    GPIO_InitStruct.Pin = LL_GPIO_PIN_1;
    GPIO_InitStruct.Mode = LL_GPIO_MODE_ALTERNATE;
    GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_HIGH;
    GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
    GPIO_InitStruct.Alternate = LL_GPIO_AF_0;
    LL_GPIO_Init(GPIOA, &GPIO_InitStruct);

    RCC->IOPENR |= LL_IOP_GRP1_PERIPH_GPIOB;
    RCC->IOPENR |= LL_IOP_GRP1_PERIPH_GPIOA;
```

V posledním řádku kódu jsme pak ještě povolili vstup hodinového signálu do obou používaných GPIO bran.

Budeme pokračovat přepnutím SPI do I2S módu v režimu „Master receive“, kdy bude periferie pouze data přijímat a nadefinujeme standard, kde očekáváme seřazení bitů v přijímaných datech od toho nejvíce významného MSB až po ten 16. nejméně významný bit:

```
    SPI1->I2SCFGR |= SPI_I2SCFGR_I2SMOD;
    SPI1->I2SCFGR |= LL_I2S_MODE_MASTER_RX;
    SPI1->I2SCFGR |= LL_I2S_STANDARD_MSB;
```

Přenos načtených dat bude probíhat vždy po načtení 16 bitů, tedy jednoho datového slova. Zajišťovat ho bude DMA, které vždy po naplnění 16bitového bufferu jeho obsah přenese do konkrétního místa v paměti. K tomu využijeme kanál 4 a navolíme u něho inkrementaci v paměti a kruhový mód, který zajistí, že bude přenos dat probíhat kontinuálně a program si později vybere konkrétní naměřená data vhodným ukončením tohoto přenosu. Následně si zvolíme 16bitovou šířku na straně periferie i paměti a posledním řádkem následujícího kódu už jen nastavíme cestu k datům v periferii na přijímací registr v SPI1, kde tuto cestu musíme pro kanál 4 DMA nastavit v DMAMUX kanálu 3:

```
    DMA1_Channel4->CCR |= DMA_CCR_MINC | DMA_CCR_CIRC;
    DMA1_Channel4->CCR |= DMA_CCR_MSIZE_0 | DMA_CCR_PSIZE_0;
    DMAMUX1_Channel3->CCR |= LL_DMAMUX_REQ_SPI1_RX;
```

Pro požadovanou funkci DMA musíme ještě nastavit adresu datového registru v periferii a adresu bufferu, do kterého budeme DMA data přenášet:

```
    #define LOG_SAMPLE_NUM 2048

    DMA1_Channel4->CPAR = (uint32_t)(&SPI1->DR);
    DMA1_Channel4->CMAR = (uint32_t)adc_buf;
    DMA1_Channel4->CNDTR = LOG_SAMPLE_NUM;
```

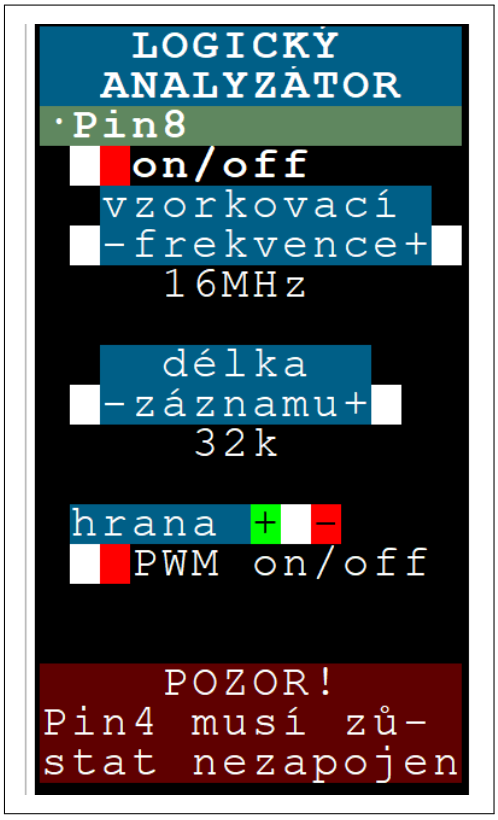
Na posledním řádku kódu jsme ještě nastavili velikost tohoto bufferu, která odpovídá velikosti adc_buf[]v datových slovech. Také vidíme, že byl pro ukládání dat z logického analyzátoru použit stejný buffer, jako tomu bylo u osciloskopu.

Logický analyzátor bude pracovat v režimu podobném režimu „single“ u osciloskopu, kdy po příchodu přerušení ovzorkuje jedno měřicí okno a pak měření ukončí. Přerušení začínající vzorkování zde bude realizované náběžnou/sestupnou hranou na GPIO. Budeme tedy chtít jeden z pinů na fyzickém pinu č.8, kde je datový vstup I2S, nastavit pro generování přerušení. Vybereme si tedy nepoužívaný pin PA14 a nastavíme ho pro generování přerušení a povolíme ho u jednotky spravující přerušení:

```
    LL_EXTI_SetEXTISource(LL_EXTI_CONFIG_PORTA, LL_EXTI_CONFIG_LINE14);
    NVIC_SetPriority(EXTI4_15_IRQn, 0);
    NVIC_EnableIRQ(EXTI4_15_IRQn);
```

4.8.2 Řídící program logického analyzátoru

Opět si nejprve vytvoříme uživatelský terminál se všemi potřebnými funkcemi pro nastavení parametrů měření logickým analyzátorem. První tlačítko označené on/off zajišťuje zapnutí a vypnutí měření, jak vidíme na obrázku 4.15. Pod ním je nastavení vzorkovací frekvence realizované tlačítkem pro zvýšení a tlačítkem pro snížení, kde bude možné tuto frekvenci volit od 125 kHz do 16 MHz. Délku záznamu bude možné zvolit od jednoho tisíce až po 32tisíc vzorků a bude opět nastavována dvěma tlačítky. Hranu, na kterou měření začne pak budeme volit mezi náběžnou a sestupnou pomocí jednoho tlačítka. Pro testovací účely bude možné příslušným tlačítkem ještě povolit výstup PWM generátoru, jehož výstup je přímo spojen se vstupem analyzátoru. Nakonec je ještě v terminálu umístěno upozornění, že pro umožnění správné funkce logického analyzátoru musí zůstat fyzický pin č.4 na pouzdře SO8N nezapojený, jelikož je na něj vyveden hodinový signál z SPI. Použité znaky pro realizaci ovládacích tlačítek jsou potom vypsány v tabulce 4.4.



Obrázek 4.15: Náhled na uživatelský terminál logického analyzátoru

Znak	Funkce
'A'	zapnutí/vypnutí analyzátoru
'G'	vzorkovací frekvence -
'H'	vzorkovací frekvence +
'K'	délka záznamu -
'L'	délka záznamu +
'J'	volba hrany
'I'	povolení PWM generátoru
'q'	návrat do MENU

Tabulka 4.4: Přehled použitých ovládacích znaků logického analyzátoru

Ještě před započetím měření proběhne nastavení vzorkovací frekvence, které lze provést pomocí hodnoty v 8bitové před-děličce hodinového signálu I2S. Nejprve tedy vynulujeme hodnotu před-děličky a následně do ní zapíšeme novou hodnotu, a to podle uživatelsky nastavené proměnné log_prescaler_ptr, která volí hodnoty od 2 do 128 případně 255. Pro dělení dvěma bude vzorkovací frekvence 16 MHz, pro dělení čtyřmi 8 MHz a tak dále:

```
SPI1->I2SPR &= ~0xff;
SPI1->I2SPR |= log_prescaler[log_prescaler_ptr];
if(log_prescaler_ptr == 7) SPI1->I2SPR |= SPI_I2SPR_ODD;
else SPI1->I2SPR &= ~SPI_I2SPR_ODD;
```

Pro maximální možné dělení 256 a tak vzniklé vzorkovací frekvenci 125kHz však musíme nastavit před-děličku na maximální možnou hodnotu 255 a zároveň nastavit bit ODD do jedničky, jak vidíme na posledních dvou řádcích kódu.

Měření započneme spuštěním DMA přenosu na straně periferie i DMA a povolením funkce samotné periferie I2S. Dále vidíme v kódu čekání, jež zajistí ovzorkování „pre-trigger“ vzorkovacího okna, který tu používáme stejně, jako u osciloskopu ovšem zde je velikost „pre-trigger“ nastavena pouze na 160 binárních vzorků:

```
SPI1->CR2 |= SPI_CR2_RXDMAEN;
DMA1_Channel4->CCR |= DMA_CCR_EN;
SPI1->I2SCFGR |= SPI_I2SCFGR_I2SE;
LL_mDelay(2);
```

Po čekání již nastavujeme hranu přerušení, kdy nejprve resetujeme zdroj přerušení z náběžné i sestupné hrany a následně podle uživatelsky zvolené hrany nastavíme příslušný registr. Posledním řádkem kódu pak povolíme samotné přerušení:

```
EXTI->RTSR1 &= ~LL_EXTI_LINE_14;
EXTI->FTSR1 &= ~LL_EXTI_LINE_14;
if(log_edge == 0) EXTI->RTSR1 |= LL_EXTI_LINE_14;
else EXTI->FTSR1 |= LL_EXTI_LINE_14;
EXTI->IMR1 |= LL_EXTI_LINE_14;
```

Teď je již vstupní pin vzorkován a naměřená data jsou cyklicky přenášena do bufferu. Po příchodu přerušení nejprve toto přerušení zakážeme a vynulujeme příznaky přerušení ať už od náběžné nebo sestupné hrany:

```
EXTI->IMR1 &= ~LL_EXTI_LINE_14;
EXTI->RPR1 = LL_EXTI_LINE_14;
EXTI->FPR1 = LL_EXTI_LINE_14
```

Následně si do proměnné log_start_pos načteme aktuální pozici v bufferu a vypočteme koncovou pozici v bufferu log_end_pos po odečtení „pre-trigger“. V následujících „while“ cyklech pak program čeká, až DMA přenese zbytek vzorkovacího okna, tedy než ukazatel na pozici v bufferu dojde až k hodnotě log_end_pos.

```
uint16_t log_start_pos = DMA1_Channel4->CNDTR;
log_start_pos = LOG_SAMPLE_NUM - log_start_pos;
log_end_pos = ((log_start_pos + LOG_SAMPLE_NUM) - LOG_PRETRIGGER)
% LOG_SAMPLE_NUM;

uint16_t x = LOG_SAMPLE_NUM - DMA1_Channel4->CNDTR;
if (log_end_pos < log_start_pos) {
    while (x < log_end_pos || x >= log_start_pos)
        x = LOG_SAMPLE_NUM - DMA1_Channel4->CNDTR;
} else {
    while (!(x >= log_end_pos || x < log_start_pos))
        x = LOG_SAMPLE_NUM - DMA1_Channel4->CNDTR;
}
DMA1_Channel4->CCR &= ~DMA_CCR_EN;
SPI1->I2SCFGR &= ~SPI_I2SCFGR_I2SE;
```

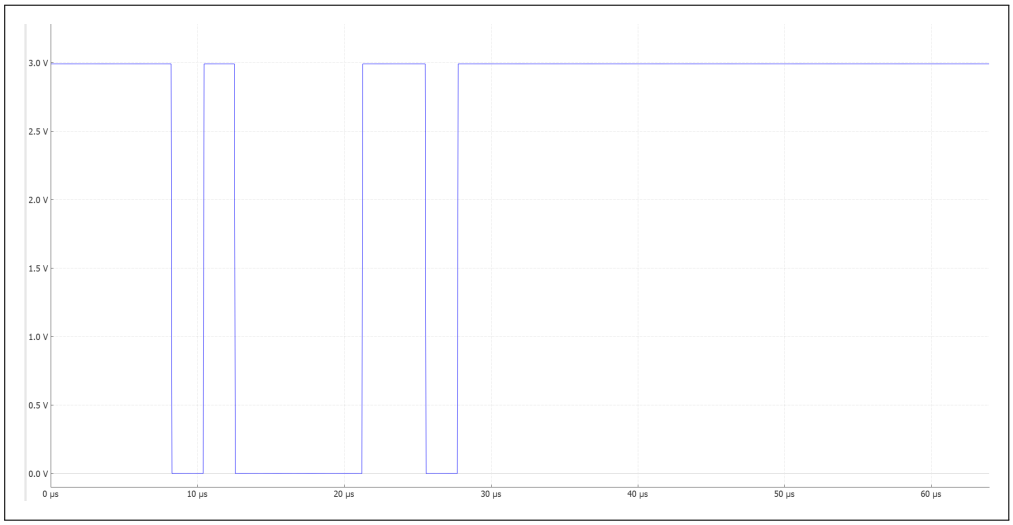
Měření jsme pak ukončili zakázáním DMA přenosu a činnosti I2S sběrnice, jak vidíme v posledních dvou řádcích kódu.

Naměřená data pak vykreslíme do Data Plotter-u obdobně jako u osciloskopu s tím, že každý bit budeme reprezentován buď 0 V pro log.0 nebo 3 V pro log.1. Nejprve tedy vypíšeme hlavičku odesílaných dat, kde je číslo kanálu, vypočtený čas mezi jednotlivými vzorky ze vzorkovací frekvence, vypočtený počet vzorků z délky bufferu a délky záznamu, kolikabitový je jeden vzorek, jaké napětí je přiřazeno nejvyšší hodnotě vzorku a typ proměnné vzorku „unsigned integer“. Následný „for“ cyklus prochází buffer po 16bitech, kde je nejprve do proměnné buf_idx spočtena hodnota ukazatele na konkrétní slovo v bufferu a pomocí něho je do proměnné act_word načtena část naměřené sekvence. Zde ovšem musíme prohodit dva po sobě jdoucí bajty, jelikož DMA ukládalo do bufferu datová slova ve formátu „little endian“, tedy nejprve méně významný bajt a následně nejvíce významný bajt. Nakonec tedy máme v této proměnné srovnanou sekvenci bitů, kdy MSB byl změřen první a LSB jako poslední a již je ve vnořeném „for“ cyklu odmaskováváme a pokud je daný bit log.0 vypisujeme nulu a pokud je log.1 vypisujeme nejvyšší hodnotu bajtu, tedy 255. Posledními dvěma řádky kódu pak jen ukončujeme výpis do datového kanálu a čekáme na odeslání všech znaků periferií USART:

```
printf("$$C1,");
uint32_t sampl_float = log_prescaler[log_prescaler_ptr]*10000;
sampl_float /= 32;
print_float(sampl_float, 10);
printf(",%d,8,3;U1",((SAMPLE_NUM*8*log_rec_length)/32));
```

```
    for (uint16_t i = 0; i < ((LOG_SAMPLE_NUM*log_rec_length)/32); i++) {
        uint16_t buf_idx = ((log_end_pos + i)%LOG_SAMPLE_NUM) * 2;
        uint16_t act_word = (adc_buf[buf_idx+1]<<8) | adc_buf[buf_idx];

        for (uint8_t p = 0; p < 16; p++) {
            uint16_t act_bit = (0x8000 >> p);
            if((act_word & act_bit) == 0) putchar(0);
            else putchar(255);
        }
    }
    printf(";");
    print_flush();
```

Příklad naměřeného vzorkovacího okna pak vidíme na obrázku 4.16, kde bylo zaznamenáno vyslání znaku ‘a‘ přes UART, bez parity a jedním stop bitem.



Obrázek 4.16: Příklad záznamu logického analyzátoru vykresleného v Data Plotter-u

4.9 Čítač

Ideální volbou pro umístění čítače na osmi pinovém pouzdře bude fyzický pin č.4, tedy pin, který jsme si pro tuto funkci vyšetřili. Na tomto pinu nalezneme tři důležité vstupy čítače TIM2, jak se můžeme přesvědčit v technickém listu mikrokontroleru 2, který jak už bylo zmíněno disponuje 32 bitovým čítacím registrem a dokáže tak při své maximální pracovní frekvenci na vstupu 64 MHz dočítat až do 67 sekund. Bude tedy možné měřit i takto dlouhé časové úseky bez potřeby ošetření jeho přetečení, jako by to bylo u 16 bitových čítačů a stále při zachování rozlišení $\frac{1}{6MHz}$. Pro měření důležitými vstupy na vybraném pinu jsou potom dva CC kanály, pro naši aplikaci zásadní především pro měření periody a střídy periodického signálu a vstup externího hodinového signálu pro čítač TIM2, který využijeme pro čítání pulzů a na jeho principu založeném měření vysokých frekvencí. V tomto režimu měření bude čítač TIM2 v takzvaném režimu „hradlování“, což spočívá v povolení vstupu čítací frekvence po konkrétní časové okno. Toto časové okno pak bude určeno některým z čítačů a jelikož jediný neobsazený čítač umožňující tuto funkci je TIM1 zvolíme si pro tuto potřebu právě něj. V následujících kódech pak budeme používat pro TIM2 i název CNT_TIM a pro TIM1 název WIN_TIM.

4.9.1 Nastavení pro čítač

Jako první se budeme zaobírat obecným nastavením TIM2, platným pro všechny režimy měření, a s tím spojeným nastavení DMA a čítače TIM1. Nastavení čítače TIM2 pro jeho konkrétní funkce si ukážeme až při realizaci jednotlivých měřicích režimů. Nejprve tedy pustíme hodinový signál do periferie čítače a periferie brány vstupů a výstupů GPIOA, kterou budeme pro všechny funkce používat:

```
RCC->APBENR1 |= LL_APB1_GRP1_PERIPH_TIM2;
RCC->IOPENR |= LL_IOP_GRP1_PERIPH_GPIOA;
TIM2->ARR = 4294967295;
```

Kde jsme ještě nastavili ARR registr čítače na nejvyšší možnou hodnotu, a to proto abychom mohli naplno využít jeho měřicí rozsah. Další nastavení tohoto čítače pak už bude pro jednotlivé režimy různé a budeme se tedy již zabývat nastavením DMA, které budeme chtít nastavit pro přenos z CC1 a CC2 jednotky čítače do paměti. Tento přenos budou zajišťovat kanály 2 a 3 DMA, které oba nastavíme na inkrementaci v paměti a velikost dat 32bitů v periferii i paměti:

```
DMA1_Channel2->CCR |= DMA_CCR_MINC;
DMA1_Channel2->CCR |= DMA_CCR_PSIZE_1;
DMA1_Channel2->CCR |= DMA_CCR_MSIZE_1;

DMA1_Channel3->CCR |= DMA_CCR_MINC;
DMA1_Channel3->CCR |= DMA_CCR_PSIZE_1;
DMA1_Channel3->CCR |= DMA_CCR_MSIZE_1;

DMAMUX1_Channel1->CCR |= LL_DMAMUX_REQ_TIM2_CH2;
DMAMUX1_Channel2->CCR |= LL_DMAMUX_REQ_TIM2_CH1;
```

Kde jsme ještě propojili kanál 2 DMA (kanál 1 DMAMUX) s CC2 jednotkou a kanál 3 DMA (kanál 2 DMAMUX) s CC1 jednotkou čítače TIM_2.

Dále budeme nastavovat čítač TIM1, pro „hradlování“ čítače TIM2. Po spuštění hodin do periferie a výběru vnitřního zdroje hodinového signálu pro čítač, nastavíme jeho kanál 3 do režimu PWM, povolíme cyklické generování PWM a jeho rychlý mód:

```
RCC->APBENR2 |= LL_APB2_GRP1_PERIPH_TIM1;
RCC->CCIPR |= (LL_RCC_TIM1_CLKSOURCE_PCLK1 << 16);
TIM1->CCMR2 |= LL_TIM_OCMODE_PWM1;
TIM1->CCMR2 |= TIM_CCMR1_OC1PE | TIM_CCMR1_OC1FE;
TIM1->CR2 |= LL_TIM_TRGO_OC3REF;
```

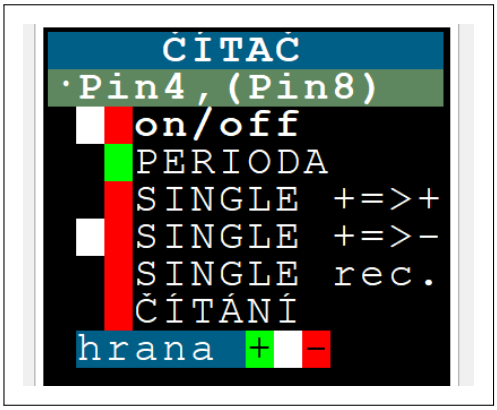
Kde jsme ještě posledním řádkem kódu propojili výstup PWM signálu z TIM1 do TIM2, kde druhý čítač bude vždy povolen pro kladnou část střídy výstupu prvního čítače, tedy dobu „hradlování“ budeme řídit pouze frekvencí signálu při nastavené konstantní střídě. Nastavíme si tak hodnotu ARR na maximální možnou a CCR3, která určuje šířku střídy na 64000, tedy při volbě nulového PSC bude šířka „hradlování“ 1ms, při PSC = 10-1 pak tento čas bude 10ms a tak dále, čímž budeme moci v programu tento čas určovat pouze hodnotou PSC:

```
TIM1->ARR = 0xFFFF;
TIM1->CCR3 = 64000;

NVIC_SetPriority(TIM1_CC_IRQn, 1);
NVIC_EnableIRQ(TIM1_CC_IRQn);
```

Kde jsme ještě povolili přerušení, které bude sloužit pro ukončení měření po skončení „hradlování“.

4.9.2 Řídící program čítače

Tentokrát tvorbu uživatelského terminálu rozdělíme na dvě části, z nichž jedna bude pro všechny režimy společná a bude zajišťovat výběr režimu, zapínání/vypínání měření a volbu hrany. Druhá část terminálu bude potom pro každý režim odlišná a bude v ní mimo nastavení parametrů měření i zobrazení výsledků měření. Této částí se pak budeme věnovat u popisu režimu jednotlivých měření. Společnou část terminálu pak můžeme vidět na obrázku 4.17, kde tlačítko označené on/off zajišťuje zapínání a vypínání měření. Dále zde vidíme tlačítko cyklicky přepínající jednotlivých pět režimů, a nakonec poslední tlačítko této části terminálu sloužící pro volbu měřené hrany. Řídící znaky použité ve společné části terminálu pak vidíme v tabulce 4.5.



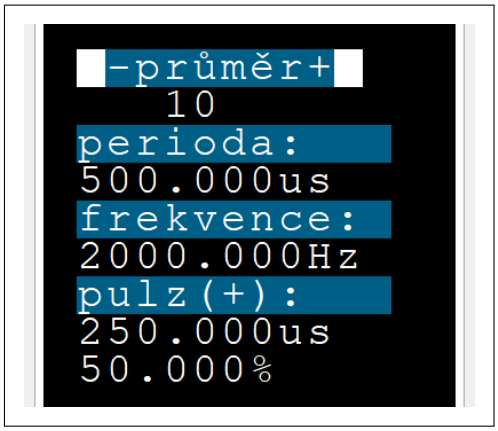
Obrázek 4.17: Náhled na společnou část uživatelského terminálu Čítače

Znak	Funkce
’o’	zapnutí/vypnutí měření
’d’	přepínání režimů
’r’	přepínání hrany
’q’	návrat do MENU

Tabulka 4.5: Přehled použitých ovládacích znaků Čítače

Režim měření periody

V tomto režimu budeme měřit periodu a střídu periodického signálu principem, který byl popsán v rozboru této práce viz. 2.2.4. V uživatelském terminálu tedy budeme chtít zobrazit naměřenou periodu v μs a tomu odpovídající frekvenci v kHz, dále pak střídu v % a jí odpovídající délku pulzu v μs, jak vidíme z obrázku 4.18. Nachází se zde ještě volba průměrování s příslušnými dvěma tlačítky pro jeho nastavení. Průměrování je zde zejména pro zlepšení přehlednosti měření pro vysoké frekvence, ale i pro zpřesnění měření. Znaky použité jako řídící v teminálu pak vidíme v tabulce4.6.

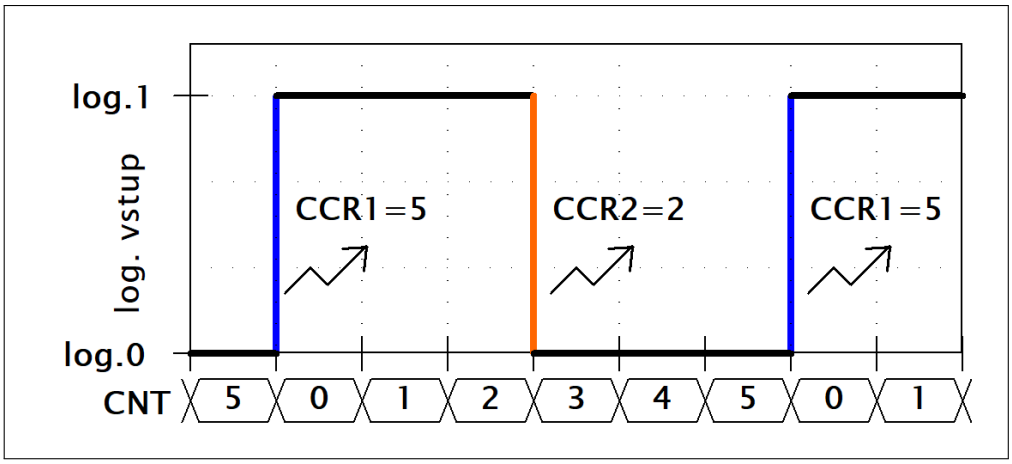


Obrázek 4.18: Náhled na část uživatelského terminálu Čítače v režimu měření periody

Znak	Funkce
'n'	- počet průměrovyných měření
'm'	+ počet průměrovyných měření

Tabulka 4.6: Přehled použitých ovládacích znaků Čítače v režimu měření periody

Pro měření periody a střídy signálu budeme požadovat aby čítač pracoval v režimu zobrazeném na obrázku 4.19. Zde bude čítač spuštěn a bude načítat impulzy o známé frekvenci 64 MHz, po příchodu náběžné hrany pak CC1 jednotka uloží do CCR1 aktuální hodnotu čítače a zároveň tato hrana způsobí nulování obsahu registru CNT. Po příchodu sestupné hrany pak zase CC2 jednotka uloží do CCR2 aktuální hodnotu čítače. Z obrázku 4.19 tedy vidíme, že pokud bude měřený signál periodický bude se CCR1 ukládat vždy hodnota odpovídající periodě signálu a do CCR2 hodnota odpovídající délce kladného pulzu z čehož lze snadno pomocí periody vypočíst střídu signálu. Pokud bude v uživatelském terminálu zvolena sestupná hrana, jako počátek měření bude princip stejný, ovšem tentokrát bude CNT registr nulován na spádovou hranu, stejně jako bude na tuto hranu ukládána hodnota periody do CCR1, náběžná hrana pak bude zajišťovat měření délky záporného pulzu ukládané do CCR2 a jemu odpovídající střídě.



Obrázek 4.19: Princip funkce čítače v režimu měření periody

Nastavení začneme u vstupní brány, kdy budeme chtít nastavit pin PA0 na alternativní funkci odpovídající prvnímu CC kanálu TIM2. To provedeme pomocí LL knihovny a příslušné její funkce, kterou jsme převzali z automaticky generovaného kódu vývojovým prostředím:

```
LL_GPIO_InitTypeDef GPIO_InitStruct = { 0 };
GPIO_InitStruct.Pin = LL_GPIO_PIN_0;
GPIO_InitStruct.Mode = LL_GPIO_MODE_ALTERNATE;
GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_HIGH;
GPIO_InitStruct.Alternate = LL_GPIO_AF_2;
LL_GPIO_Init(GPIOA, &GPIO_InitStruct);
```

Budeme pokračovat nastavením nulování čítače při příchodu události na CC1 kanál, což docílíme následujícími řádky kódu:

```
CNT_TIM->SMCR |= LL_TIM_SLAVEMODE_RESET;
CNT_TIM->SMCR |= LL_TIM_TS_TI1FP1;
```

Jelikož budeme používat pouze vstup kanálu jedna čítače TIM_2 musíme nastavit CC1 i CC2 jednotku pro propojení s prvním kanálem:

```
CNT_TIM->CCMR1 |= TIM_CCMR1_CC1S_0;
CNT_TIM->CCMR1 |= TIM_CCMR1_CC2S_1;
```

Zbývá ještě správné nastavení polarity hran u jednotlivých CC jednotek. Ty budou nastaveny buď na kladnou první hranu, jak vidíme na prvních dvou řádcích kódu, nebo na zápornou první hranu, jak to představují druhé dva řádky kódu:

```
CNT_TIM->CCER &= ~TIM_CCER_CC1P;
CNT_TIM->CCER |= TIM_CCER_CC2P;

CNT_TIM->CCER |= TIM_CCER_CC1P;
CNT_TIM->CCER &= ~TIM_CCER_CC2P;
```

Posledním nutným nastavením pak bude povolení vstupu do obou CC jednotek a správné nastavení DMA, které bude zajišťovat přenos dat v kruhovém módu z CC jednotek do příslušných polí v paměti, jejichž adresu musíme do DMA zadat, stejně tak jako délku kruhového bufferu, která bude odpovídat aktuálně vybranému počtu průměrovaných vzorků:

```
CNT_TIM->CCER |= LL_TIM_CHANNEL_CH2 | LL_TIM_CHANNEL_CH1;

DMA1_Channel2->CCR |= DMA_CCR_CIRC;
DMA1_Channel3->CCR |= DMA_CCR_CIRC;
DMA1_Channel2->CPAR = (uint32_t) (&CNT_TIM->CCR2);
DMA1_Channel2->CMAR = (uint32_t) (meas_duty);
DMA1_Channel3->CPAR = (uint32_t) (&CNT_TIM->CCR1);
DMA1_Channel3->CMAR = (uint32_t) (meas_period);
DMA1_Channel2->CNDTR = mean[mean_ptr];
DMA1_Channel3->CNDTR = mean[mean_ptr];
```

Kde v proměnné mean[] jsou uloženy všechny možné počty průměrování a jsou vybírána z terminálu ovládanou proměnnou mean_ptr.

Spouštění měření v tomto režimu začneme nulováním čítače TIM2 a povolením jeho funkce, následně povolíme oba používané DMA kanály a přenos pomocí DMA na straně obou CC jednotek čítače:

```
CNT_TIM->CNT = 0;
CNT_TIM->CR1 |= TIM_CR1_CEN;
DMA1_Channel2->CCR |= DMA_CCR_EN;
DMA1_Channel3->CCR |= DMA_CCR_EN;
CNT_TIM->DIER |= TIM_DIER_CC2DE | TIM_DIER_CC1DE;
```

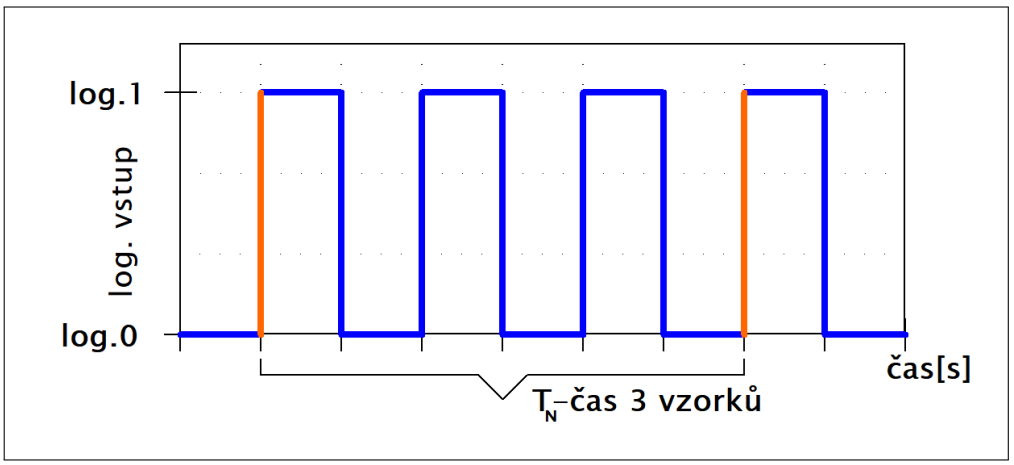
Teď se již nastavení nachází ve fázi, kdy DMA přenáší údaje o periodě do připraveného kruhového bufferu meas_period[] s délkou určenou počtem průměrovaných vzorků a stejně tak DMA přenáší údaje o střídě signálu do kruhového bufferu meas_duty[] s délkou opět určenou počtem průměrovaných vzorků. V principu pak budeme ze vzorků periody, jejich sečtením, počítat dobu trvání N period T_N, jak vidíme na obrázku 4.20, kde N je počet průměrovaných vzorků. Kde průměrnou hodnotu periody T_{prum} získáme pomocí aritmetického průměru, jako:

$$T_{prum} = \frac{T_N}{N}. \tag{4.5}$$

Tímto principem tak dostaneme nejen průměrnou hodnotu, ale docílíme tím i lepšího rozlišení měření, jak bylo popsáno v kapitole 2.2.4. Rozlišení $\Delta t[s]$ bude pro měření bez průměrování (průměr z jednoho vzorku), rovno převrácené hodnotě referenční čítací frekvence čítače, tedy $\Delta t = \frac{1}{64MHz} \approx 15,6ns$. Při průměrování však bude toto číslo ještě děleno počtem průměrování N a tomu odpovídající rozlišení lze pak dopočíst podle vzorce:

$$\Delta t_N = \frac{1}{N}\frac{1}{64MHz} \approx \frac{15,6ns}{N}. \tag{4.6}$$

Pro výpočet průměru a rozlišení z naměřených hodnot střídy pak budeme postupovat podle stejných vzorců, jako tomu bylo výše u periody.



Obrázek 4.20: Princip funkce čítače v režimu měření periody s průměrováním

Při měření periody a střídy v tomto režimu budeme dostávat výsledky v μs, ovšem budeme chtít pro lepší uživatelský komfort tyto hodnoty přepočíst i na frekvenci a střídu v %. Pro tyto účely budeme potřebovat využít aritmetiku „float“ s desetinnou tečkou, tak abychom byli schopni při dělení z výsledku odečíst i desetinná čísla. Tato již v knihovnách implementovaná aritmetika je však poměrně paměťově náročná, a proto ji nahradíme jednodušším formátem, kdy si počítané 32bitové číslo uložíme do 64bitové proměnné a vynásobíme jej 10000cem a tyto čtyři vzniklé nejméně významné dekády nám budou představovat číslo za desetinnou čárkou. Čtyři cifry za desetinnou čárkou jsme zvolili tak, abychom dokázali zobrazit především v μs udávanou periodu a střídu s plným využitím rozlišení měření, a to i pro měření s průměrováním. Na samotné vypsání tohoto formátu čísla jsme pak vytvořili

funkci, která nejprve vypíše standartním způsobem číslo před desetinnou čárkou, pak vhodně umístí desetinnou čárku a následně vypisuje jednotlivé číslice za desetinnou čárkou.

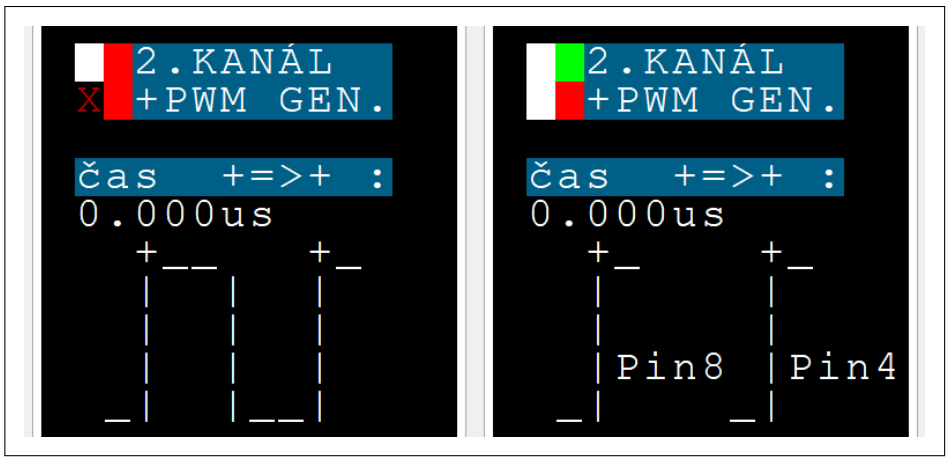
Tento režim své měření ukončuje až po ukončení měření uživatelem v terminálu, kdy nejprve zastaví funkci čítače, a to i povolení přenosu obou CC jednotek pomocí DMA. Dále je pak ukončen přenos dat i na straně samotného DMA:

```
CNT_TIM->CR1 &= ~TIM_CR1_CEN;
CNT_TIM->DIER &= ~(TIM_DIER_CC2DE | TIM_DIER_CC1DE);

DMA1_Channel2->CCR &= ~DMA_CCR_EN;
DMA1_Channel3->CCR &= ~DMA_CCR_EN;
```

Režim single, stejná polarita hran

Tento režim bude vycházet z již v rozboru popsaného měření časového odstupu dvou událostí stejného charakteru viz. 2.2.4, kde obě události pocházejí ze stejného zdroje, tedy bude se jednat o signál připojený na jeden pin MCU. V případě, že budeme chtít měřit totéž ovšem první a druhá událost budou obě z různých zdrojů, tedy připojení k mikrokontroleru bude dvou pinové, se bude jednat o princip popsaný v rozboru jako měření časového odstupu dvou událostí stejného charakteru pocházejících z různých zdrojů viz. 2.2.4. Režim tedy bude mít dvě lehce odlišné části, kde první část bude pro měření jednokanálové, tedy obě události z jednoho zdroje a druhá část bude pro měření dvoukanálové, kdy bude každý zdroj události připojen na jiný pin.

Uživatelský terminál je v tomto režimu tvořen volbou jednokanálového nebo dvoukanálového měření a jemu příslušnému spřažení s PWM generátorem. Terminál pro obě volby pak vidíme na obrázku 4.21, kde jednokanálové měření je na levém výřezu z terminálu a dvoukanálové měření pak na pravém výřezu. Dále je zde realizován výpis naměřené hodnoty v μs a graficky znázorněná měřená veličina pro lepší uživatelskou přehlednost navolených parametrů měření vycházející z obrázků 2.7 a 2.11. Použité řídící znaky v terminálu tohoto režimu pak vidíme v tabulce 4.7.



Obrázek 4.21: Náhled na část uživatelského terminálu Čítače v režimu single, stejná polarita hran

Znak	Funkce
'E'	jednokanálové/dvoukanálové měření
'F'	spřažení s PWM generátorem

Tabulka 4.7: Přehled použitých ovládacích znaků Čítače v režimu single, stejná polarita hran

- **Jednokanálové měření**
  Pro jednokanálové měření bude nastavení prakticky totožné s nastavením pro režim měření periody popsaném výše viz.4.9.2. Budeme opět používat pin PA0 a nastavovat stejným způsobem CC kanály i samotný TIM2. Jediná změna bude v nastavení DMA, kdy namísto kruhového módu nastavíme mód limitovaného přenosu:

```
DMA1_Channel2->CCR &= ~DMA_CCR_CIRC;
DMA1_Channel3->CCR &= ~DMA_CCR_CIRC;

DMA1_Channel3->CNDTR = 2;
DMA1_Channel2->CNDTR = 0;
```

  Kde jsme ještě nastavili délku záznamu, jako dva vzorky CC1 kanálu (periody), kdy jeden měření začne a druhý ukončí a nula vzorků CC2 kanálu (střídy), jelikož tu při měření nepotřebujeme.

  Spuštění měření bude opět obdobné jako u měření periody, ovšem tentokrát ještě před nulováním a zapnutím čítače a následným povolením DMA přenosu, nastavíme příznaky dokončení DMA přenosu na defaultní hodnotu, v tomto případě na samé jedničky:

```
DMA1->IFCR = ~0x0;

CNT_TIM->CNT = 0;
CNT_TIM->CR1 |= TIM_CR1_CEN;
DMA1_Channel2->CCR |= DMA_CCR_EN;
DMA1_Channel3->CCR |= DMA_CCR_EN;
CNT_TIM->DIER |= TIM_DIER_CC2DE | TIM_DIER_CC1DE;
```

  Příznak dokončení přenosu pak použijeme k ukončení měření, které buď nastane na žádost z terminálu, nebo po dokončení přenosu DMA, které je právě konkrétním příznakem detekováno:

```
(DMA1->ISR & DMA_ISR_TCIF3) == DMA_ISR_TCIF3;
```

  Po detekci dokončení přenosu předchozím výrazem v kódu, nebo po vyžádání z terminálu bude vypsán na terminál naměřený čas a měření bude zastaveno stejným způsobem, jako u měření periody viz. 4.9.2.

- **Dvoukanálové měření**
  Dvoukanálové měření se oproti tomu jednokanálovému bude lišit zejména v nastavení vstupních pinů, kdy budeme používat pin PA15, jako vstup do CC1 jednotky čítače TIM2 a pin PA1, jako vstup CC2 jednotky tohoto čítače. Kde pin PA15 je umístěn na fyzickém pinu č.8 pouzdra SO8N a pin PA1 je situován na fyzickém pinu č.4 stejného pouzdra. Vidíme tedy, že pin PA15 je fyzicky propojen s výstupem PWM generátoru, a proto bude používání generátoru až na výjimku blokováno. Nastavení pinů PA1 a PA15 provedeme přiřazením příslušné alternativní funkce odpovídající požadovaným CC kanálům čítače TIM2, a momentálně nepoužívaný pin PA0 nastavíme na alternativní funkci tři, která u tohoto pinu nemá žádnou funkcionalitu:

```
LL_GPIO_InitTypeDef GPIO_InitStruct = { 0 };

GPIO_InitStruct.Pin = LL_GPIO_PIN_0;
GPIO_InitStruct.Alternate = LL_GPIO_AF_3;
LL_GPIO_Init(GPIOA, &GPIO_InitStruct);

GPIO_InitStruct.Pin = LL_GPIO_PIN_1;
GPIO_InitStruct.Mode = LL_GPIO_MODE_ALTERNATE;
GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_HIGH;
GPIO_InitStruct.Alternate = LL_GPIO_AF_2;
LL_GPIO_Init(GPIOA, &GPIO_InitStruct);

GPIO_InitStruct.Pin = LL_GPIO_PIN_15;
GPIO_InitStruct.Mode = LL_GPIO_MODE_ALTERNATE;
GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_HIGH;
GPIO_InitStruct.Alternate = LL_GPIO_AF_2;
LL_GPIO_Init(GPIOA, &GPIO_InitStruct);
```

Dále pak nastavíme, aby byl obsah čítače nulován při příchodu první události, a to z kanálu jedna, stejně jako tomu bylo u minulých měřicích režimů:

```
CNT_TIM->SMCR |= LL_TIM_TS_TI1FP1 | LL_TIM_SLAVEMODE_RESET;
```

Rozdílným nastavením oproti minulým režimům pak bude přiřazení zdroje signálu CC jednotkám, kdy tentokrát CC1 bude mít za zdroj první kanál a CC2 druhý kanál:

```
CNT_TIM->CCMR1 |= TIM_CCMR1_CC1S_0;
CNT_TIM->CCMR1 |= TIM_CCMR1_CC2S_0;
```

Přenos pomocí DMA bude stejný, jako u jednokanálového měření tohoto režimu, ovšem tentokrát budeme chtít, aby byl z každé CC jednotky přenesen právě jeden vzorek:

```
DMA1_Channel2->CNDTR = 1;
DMA1_Channel3->CNDTR = 1;
```

Dalším rozdílem bude i nastavení hran obou CC jednotek a blokování PWM generátoru. Tentokrát totiž budeme nastavovat hranu u obou CC jednotek stejnou, tedy pokud bude zvolena kladná reakční hrana bude nastavení podle prvních dvou řádků následujícího kódu a pokud bude tato hrana zvolena, jako záporná budeme nastavovat čítač druhými dvěma řádky:

```
CNT_TIM->CCER &= ~TIM_CCER_CC1P;
CNT_TIM->CCER &= ~TIM_CCER_CC2P;

CNT_TIM->CCER |= TIM_CCER_CC1P;
CNT_TIM->CCER |= TIM_CCER_CC2P;

TIM16->CCER |= LL_TIM_CHANNEL_CH1N;
```

Kde jsme pak ještě posledním řádkem odpojili PWM generátor od jeho výstupního pinu a zajistili jsme tím, že nebude měření tímto signálem narušeno.

Spouštění a ukončení měření bude s vypnutým spřažením s PWM generátorem stejné, jako tomu bylo u jednokanálového měření tohoto režimu, ovšem s rozdílnou ukončovací podmínkou, kdy tentokrát čekáme na ukončení DMA přenosu z CC2 jednotky:

```
(DMA1->ISR & DMA_ISR_TCIF2) == DMA_ISR_TCIF2;
```

V případě zvoleného spřažení s PWM generátorem bude software-ově zapnut generátor a následně software-ově nulován čítač, čímž se zahájí měření a ukončení měření pak bude příchodem druhé události. V principu tak zapnutí PWM generátoru iniciuje start nějakého měření, například vyslání zvukového signálu a událost na druhém kanálu pak měření ukončuje, například přijetím zvukového signálu. Při spouštění takto definovaného měření tedy nejprve zakážeme nulování při příchodu události na CC1:

```
CNT_TIM->SMCR &= ~LL_TIM_SLAVEMODE_RESET;
```

Následně spustíme měření stejným způsobem, jako u měření bez generátoru, a nakonec zapneme PWM generátorem a nulujeme čítač:

```
DMA1->IFCR = ~0x0;
CNT_TIM->CNT = 0;
CNT_TIM->CR1 |= TIM_CR1_CEN;
DMA1_Channel2->CCR |= DMA_CCR_EN;
DMA1_Channel3->CCR |= DMA_CCR_EN;
CNT_TIM->DIER |= TIM_DIER_CC2DE | TIM_DIER_CC1DE;

GEN_TIM->CCER |= LL_TIM_CHANNEL_CH1N;
GEN_TIM->CR1 |= TIM_CR1_CEN;
CNT_TIM->CNT = 0;
```

Měření se opět ukončí uživatelsky z terminálu, nebo podmínkou:

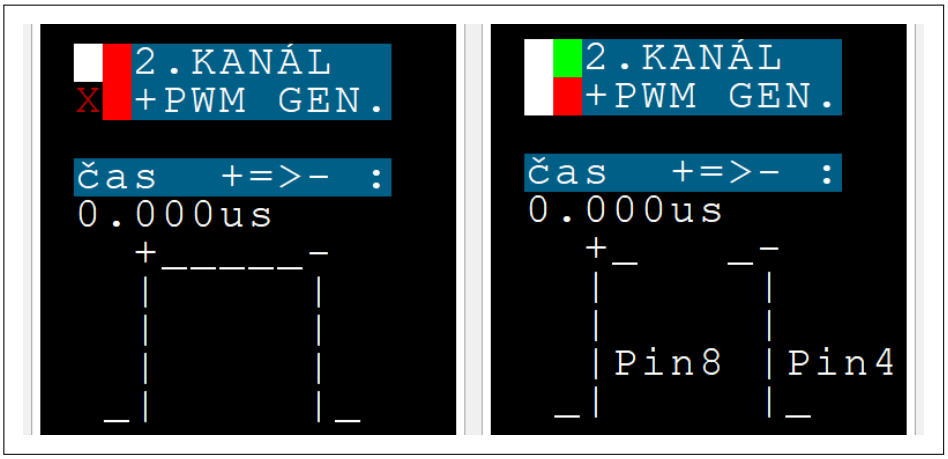
```
(DMA1->ISR & DMA_ISR_TCIF2) == DMA_ISR_TCIF2;
```

Kdy kromě vypnutí DMA přenosu a čítače, jako tomu bylo v předchozím režimu, i vypínáme PWM generátor a vracíme nastavení nulování po příchodu události na CC1 kanál do původního stavu:

```
GEN_TIM->CCER &= ~LL_TIM_CHANNEL_CH1N;
GEN_TIM->CR1 &= ~TIM_CR1_CEN;
CNT_TIM->SMCR |= LL_TIM_SLAVEMODE_RESET;
```

Režim single, opačná polarita hran

Tento režim je principiálně téměř stejný, jako předchozí režim se stejnou polaritou hran, a tak i všechny jeho funkce a ovládání budou s předchozím režimem totožné. Jediná změna nastává v nastavení CC2 jednotky, kdy bude tentokrát volena vždy hrana opačné polarity než ta u CC1 jednotky. Princip měření tohoto režimu je pak pro jednokanálový vstup popsán v rozboru viz. 2.2.4 a pro dvoukanáloví vstup viz. 2.2.4. Tento režim je opět dělen na jednokanálové a dvoukanálové měření, u kterých se ovšem budeme, kvůli velké podobnosti, zabývat pouze odlišnostmi v principu funkce od režimu se stejnou polaritou. Uživatelský terminál, který vidíme na obrázku 4.22 má pak odlišnost od předchozího režimu pouze v grafickém zobrazení měřené veličiny, která zde vychází z obrázků 2.8 a 2.12.



Obrázek 4.22: Náhled na část uživatelského terminálu Čítače v režimu single, opačná polarita hran

- **Jednokanálové měření**
  Nastavení se od předchozího režimu v jednokanálovém měření liší u konfigurace DMA, kdy tentokrát budeme chtít, aby DMA přeneslo z každé CC jednotky jeden odměr:

```
DMA1_Channel2->CNDTR = 1;
DMA1_Channel3->CNDTR = 1;
```

  Spouštění a ukončení měření pak probíhá také stejně ovšem tentokrát bude ukončovací podmínkou ukončení přenosu z CC2 jednotky:

```
(DMA1->ISR & DMA_ISR_TCIF2) == DMA_ISR_TCIF2;
```

- **Dvoukanálové měření**
  V nastavení opět budeme kopírovat dvoukanálové měření předchozího režimu, ovšem tentokrát při nastavení hrany zvolíme vždy pro jednotlivé CC jednotky hrany opačné:

```
CNT_TIM->CCER &= ~TIM_CCER_CC1P;
CNT_TIM->CCER |= TIM_CCER_CC2P;

CNT_TIM->CCER |= TIM_CCER_CC1P;
CNT_TIM->CCER &= ~TIM_CCER_CC2P;
```

  Kde první dna řádky kódu nastavují čítač pro kladnou první hranu a druhé dva řádky pak pro zápornou první hranu. Spouštění a ukončení měření je potom s dvoukanálovým měřením v předchozím režimu totožné, a to včetně ukončovací podmínky. Pro připnutí PWM generátoru pak bude nastavení, spouštění i ukončení s předchozím režimem totožné.

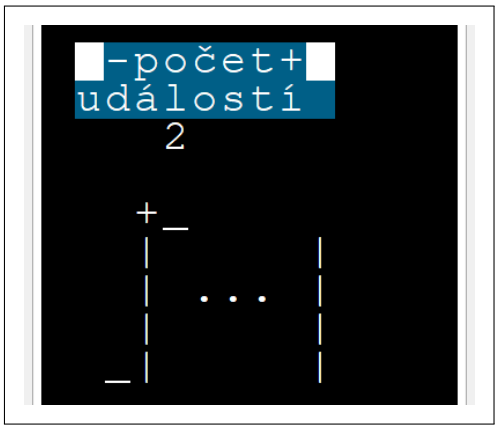
Režim single, záznam

Tento režim je založen na principu odměření $N-1$ časových okamžiků mezi N hranami, jak je popsáno v rozboru viz. 2.2.4. Budeme tedy požadovat po čítači stejnou funkci, jako u režimu měření periody, ovšem tentokrát nebude DMA plnit buffer cyklicky ovšem naplní ho pouze jednou a všechny prvky z něj program vykreslí vhodným způsobem do Data Plotter-u. Nastavení pak provedeme stejně, jako u režimu měření periody viz. 4.9.2, ovšem s tím rozdílem, že DMA nastavíme na limitovaný přenos a počet přenášených odměrů jednotlivých CC jednotek podle následujícího kódu:

```
    DMA1_Channel2->CCR &= ~DMA_CCR_CIRC;
    DMA1_Channel3->CCR &= ~DMA_CCR_CIRC;

    DMA1_Channel2->CNDTR = cnt_events/2;
    DMA1_Channel3->CNDTR = (cnt_events/2)+cnt_events%2;
```

Kde uživatelsky ovládaná proměnná cnt_events uchovává požadovaný počet událostí k zaznamenání. V případě, že je požadovaný počet událostí sudý budeme chtít přenést stejný počet odměrů z obou CC jednotek. Pokud ovšem bude tento počat lichý bude CC1 přenášet o jeden odměr navíc oproti CC2.

Uživatelský terminál pro tento režim bude mít pouze dva prvky, jak vidíme na obrázku 4.23. Nejprve je zde dvoutlačítkové nastavení počtu požadovaných událostí k zaznamenání spolu se zobrazením aktuálně nastaveného počtu. Druhým prvkem terminálu je potom grafické znázornění měřené veličiny stejně jako tomu bylo u minulých „single“ režimů. Ovládací znaky použité u této části uživatelského terminálu vidíme v tabulce 4.8.



Obrázek 4.23: Náhled na část uživatelského terminálu Čítače v režimu single, záznam

Znak	Funkce
'D'	+ počet měřených událostí
'C'	- počet měřených událostí

Tabulka 4.8: Přehled použitých ovládacích znaků Čítače v režimu single, záznam

Spouštění je podobné s režimem měření periody, kdy nulujeme a spouštíme čítač TIM2 a povolujeme přenosy DMA z CC jednotek. Rozdílem je pouze nastavení příznaků ukončení DMA přenosu do defaultního nastavení:

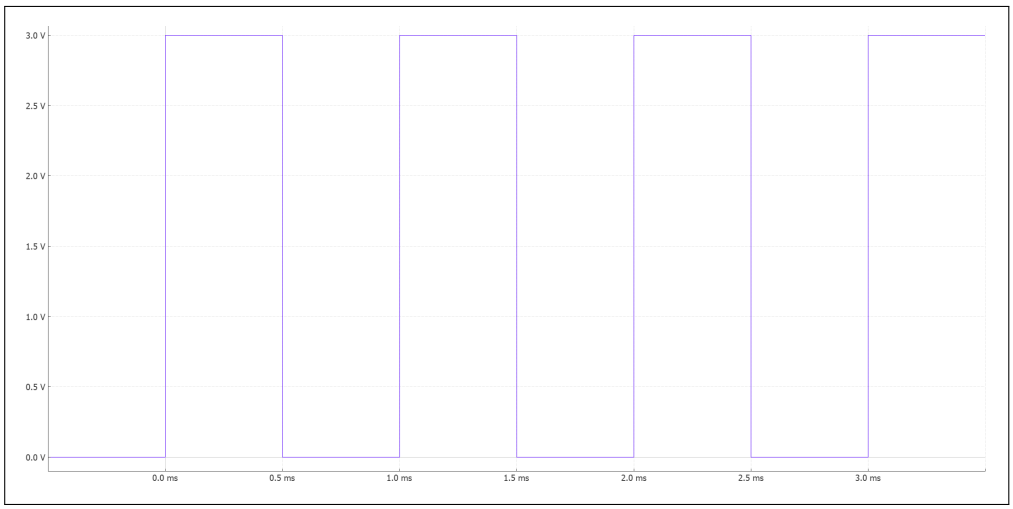
```
    DMA1->IFCR = ~0x0;

    CNT_TIM->CNT = 0;
    CNT_TIM->CR1 |= TIM_CR1_CEN;
    DMA1_Channel2->CCR |= DMA_CCR_EN;
    DMA1_Channel3->CCR |= DMA_CCR_EN;
    CNT_TIM->DIER |= TIM_DIER_CC2DE | TIM_DIER_CC1DE;
```

Ukončení měření je pak vyžádáno buď z uživatelského terminálu, nebo po splnění podmínky, kdy jsou dokončeny přenosy z obou CC jednotek:

```
((DMA1->ISR & DMA_ISR_TCIF2) == DMA_ISR_TCIF2 )
&& ((DMA1->ISR & DMA_ISR_TCIF3) == DMA_ISR_TCIF3);
```

Kde ukončení je pak realizováno stejně jako u režimu měření periody viz. 4.9.2.

Zobrazení naměřených dat budeme realizovat v Data Plotter-u, kdy zobrazíme vždy zachycené hrany a příslušné logické úrovně mezi nimi o patřičných časových délkách. Vykreslování do grafu Data Plotter-u pak bude následující, jak vidíme na příkladu zobrazeného záznamu 4.24. Nejprve vykreslíme pomocí dvou bodů přímku odpovídající logické úrovně před první hranou umístěnou v čase nula a následně vykreslujeme pomocí dvou bodů přímky mezi dalšími dvěma hranami opět o patřičné logické úrovni odvozené od povahy konkrétních hran, tedy pokud je hrana sestupná vykreslujeme log.0 a pokud je vzestupná vykreslíme log.1. První bod, pomocí kterého danou přímku vykreslujeme, bude umístěn ve výchozí hraně a druhý pak v té cílové, kde vzdálenost mezi nimi bude odečtena z patřičného naměřeného časového úseku čítačem. Díky tomuto vykreslování pak bude poměrně jednoduché odečítat z vykresleného grafu, jelikož vždy N-tá hrana se bude nacházet na 2N-tém vykresleném bodě.



Obrázek 4.24: Příklad zobrazení single záznamu v Data Plotter-u

Režim čítání

U tohoto režimu budeme vycházet z principu prostého čítání pulzů a na něj navazujícího přímého měření frekvence rozebrané viz. 2.2.4. Nastavení čítače TIM1, který nám bude „hradlovat“ čítač TIM2 počítající samotné impulzy, jsme si již do požadované funkce nastavili a zbývá tak pouze nastavit druhý používaný čítač TIM2 a jeho patřičné vstupy. Jako vstup měřeného signálu budeme požívat pin PA0, který nastavíme pomocí funkce LL knihovny do alternativního módu pro TIM2_ETR, což je stejná alternativní funkce jako u předchozích režimů s jedním vstupem, tedy:

```
    GPIO_InitStruct.Pin = LL_GPIO_PIN_0;
    GPIO_InitStruct.Alternate = LL_GPIO_AF_2;
    LL_GPIO_Init(GPIOA, &GPIO_InitStruct);
```

Dále nastavíme čítač TIM2 do režimu „hradlování“, vybereme příslušný zdroj vstupního signálu odpovídající pinu PA0 a povolíme vstup těchto vnějších hodin:

```
    CNT_TIM->SMCR |= LL_TIM_SLAVEMODE_GATED;
```

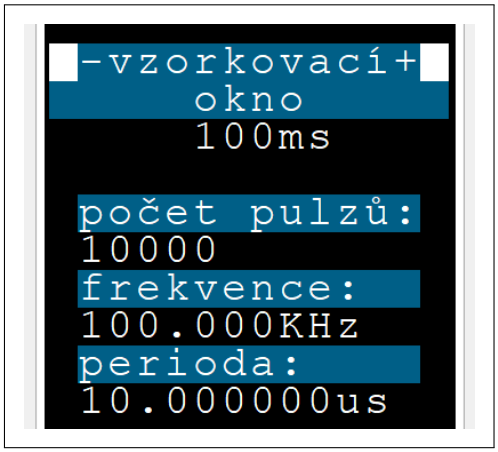
```
    CNT_TIM->SMCR |= LL_TIM_CLOCKSOURCE_EXT_MODE2;
    CNT_TIM->SMCR |= TIM_SMCR_ECE;
```

Konfiguraci zdroje „hradlovacího“ signálu na výstup TIM1 pak dělat nemusíme, jelikož je nastaveno defaultně. V uživatelském terminálu bude mimo nastavení doby „hradlování“ skrze PSC čítače TIM_1, ještě možné nastavit na které hrany bude čítač reagovat. To provedeme následujícími příkazy:

```
    CNT_TIM->SMCR &= ~TIM_SMCR_ETP;
    CNT_TIM->SMCR |= TIM_SMCR_ETP;
```

Kde prvním řádkem nastavujeme defaultní citlivost čítače na náběžné hrany a druhým řádkem kódu pak citlivost na hrany sestupné.

Uživatelský terminál pro tento režim bude obsahovat jednak nastavení čítacího okna, realizované dvěma tlačítky, jedním pro přičítání a druhým pro odčítání, jak vidíme na obrázku 4.25. Dále terminál bude zobrazovat počet naměřených pulzů, a to i v průběhu měření což umožní uživateli interaktivně měření sledovat. Z naměřeného počtu pulzů a délky časového okna ještě dopočteme a zobrazíme odpovídající periodu a frekvenci vstupního, z předpokladu periodického, signálu. Pro režim nekonečně dlouhého časového okna pak budou tyto dvě položky proškrtnuty. Znaky použité pro realizaci tlačítek v tomto režimu vidíme v tabulce 4.9.



Obrázek 4.25: Náhled na část uživatelského terminálu Čítače v režimu čítání

Znak	Funkce
’p’	+ vzorkovací okno
’u’	- vzorkovací okno

Tabulka 4.9: Přehled použitých ovládacích znaků Čítače v režimu čítání

Zapínání měření bude následující. Nejprve vynulujeme čítací registr čítače TIM2, ve kterém bude uložen výsledek měření a povolíme jeho čítání, dále pak vynulujeme čítač časového okna TIM1 a pokud není zvolen nekonečný čítací čas, v programu window_ptr = 6, opět povolíme jeho čítání. Pokud je však nekonečný čítací čas zvolen pouze změníme polaritu výstupu čímž nebude čítání TIM1 povoleno, ale naopak čítání TIM2 tím povoleno bude a to, dokud nebude opět změněna polarita nazpátek při vypnutí čítání vyžádaném uživatelem:

```
    CNT_TIM->CNT = 0;
```

```
CNT_TIM->CR1 |= TIM_CR1_CEN;
WIN_TIM->CNT = 0;
if(window_ptr==6) WIN_TIM->CCER |= TIM_CCER_CC3P;
else WIN_TIM->CR1 |= TIM_CR1_CEN;
WIN_TIM->DIER |= TIM_DIER_CC3IE;
```

Kde jsme ještě povolili přerušení čítače TIM1 po skončení čítacího okna, které zaručí vypnutí měření.

Vypnutí měření pak bude vykonáno v přerušení po skončení čítacího okna, kde program nejprve zakáže čítání TIM1, jeho přerušení, následně vymaže příznak tohoto přerušení, a nakonec povolí programu funkce čítače, aby resetoval zbylá nastavení spojené tímto měřením.

```
TIM1->CR1 &= ~TIM_CR1_CEN;
TIM1->DIER &= ~TIM_DIER_CC3IE;
TIM1->SR &= ~TIM_SR_CC3IF;
cnt_off();
```

Program funkce čítače pak provede následují úkony právě při vyžádání od přerušení, nebo po vyžádání ukončení měření z uživatelského terminálu. Nejprve tedy zakáže čítání dalších měřených pulzů a čítání času měřicího okna:

```
CNT_TIM->CR1 &= ~TIM_CR1_CEN;
WIN_TIM->CR1 &= ~TIM_CR1_CEN;
WIN_TIM->CCER &= ~TIM_CCER_CC3P;
```

Kde poslední řádek kódu ještě nastavuje defaultní kladnou polaritu výstupu, která byla změněna při volbě nekonečného čítacího okna.

Kapitola 5

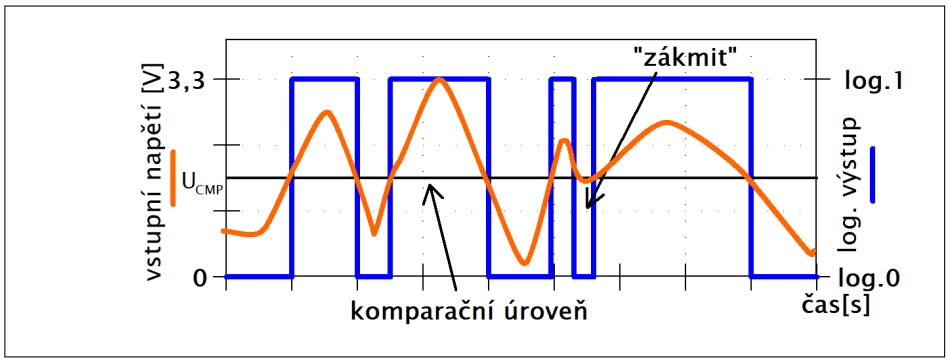
Návrh experimentů a podpůrných obvodů

5.1 Obvody pro zpracování signálu ze senzoru

Před započetím samotného měření s vytvořeným přístrojem musíme experiment nejen opatřit vhodným senzorem, ale je-li to nutné i správným způsobem předzpracovat výstup tohoto senzoru pro použití s naším měřicím přístrojem. Čítač dokáže rozlišit pouze log.0 a log.1 podle napěťových úrovní definovanými napájecím napětím $U_{nap} = 3,3V$, jako:

$$U_{0min} = 0V, \quad U_{0max} = 0,3U_{nap} = 0,99V, \quad U_{1min} = 0,7U_{nap} = 2,31V, \quad U_{1max} = U_{nap}, \tag{5.1}$$

kde oblast mezi U_{0max} a U_{1min} není definována. A my tedy musíme čítači zajistit, že bude na jeho vstupu buď napětí odpovídající log.0 nebo log.1 definované výše. To docílíme pomocí komparátoru, který nám podle definované napěťové úrovně signál rozdělí na log.0 a log.1, tedy na jeho výstupu bude pouze napětí odpovídající některé logické úrovni, případně velmi rychlý přechod mezi nimi, jak vidíme na příkladu z obrázku 5.1.



Obrázek 5.1: Příklad zpracování signálu komparátorem

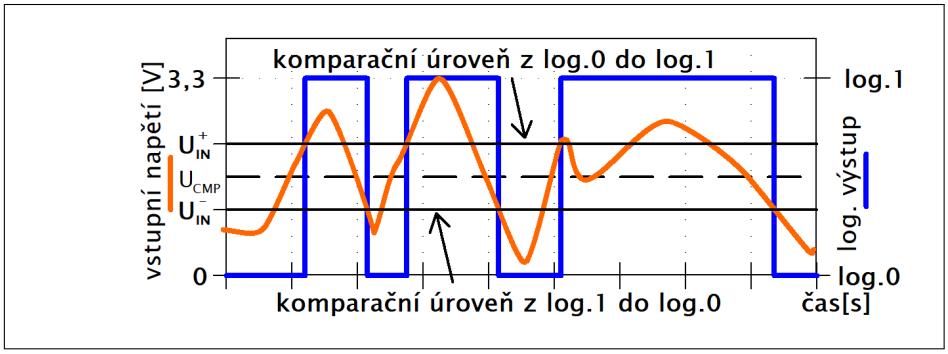
Z obrázku 5.1 také vidíme, že při pomalém přechodu mezi dvěma logickými úrovněmi může dojít k nechtěnému „zákmitu“, který by mohl znesnadňovat některá měření. Lze jej však odstranit použitím komparátoru s vhodnou hysterezí, který má v principu dvě komparační úrovně, kdy jedna slouží pro překlopení do log.0 a druhá pro překlopení do log.1, jak vidíme na obrázku 5.2. Tento obvod lze snadno sestavit pomocí operačního zesilovače (OZ) a dvou rezistorů, jak vidíme na schématu 5.3, kde je zapojení neinvertující a invertující. Komparační napěťové úrovně pak pro oba typy budou, pro neinvertující zapojení:

$$U_{IN}^{+} = U_{CMP} - U_{nap}^{-}\frac{R_1}{R_2}, \qquad U_{IN}^{-} = U_{CMP} - U_{nap}^{+}\frac{R_1}{R_2} \tag{5.2}$$

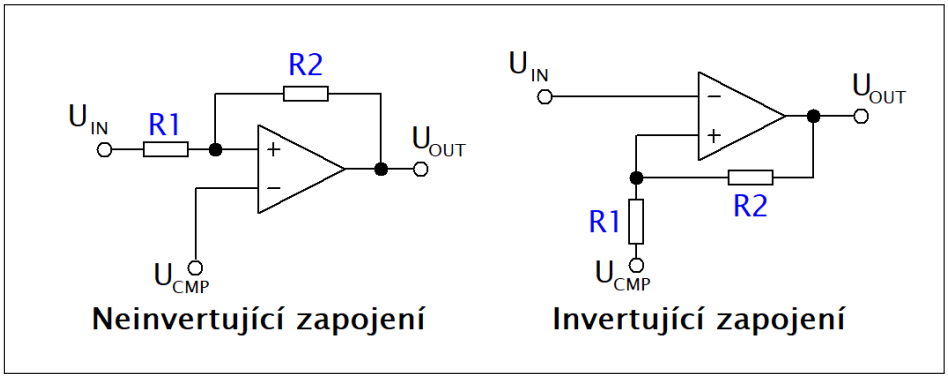
a pro invertující zapojení:

$$U_{IN}^{+} = U_{CMP} + U_{nap}^{+}\frac{R_1}{R_1 + R_2}, \qquad U_{IN}^{-} = U_{CMP} + U_{nap}^{-}\frac{R_1}{R_1 + R_2}, \tag{5.3}$$

kde U_{nap}^{+} bude kladné napájecí napětí OZ oproti nule, U_{nap}^{-} záporné napájecí napětí OZ oproti nule, U_{CMP} je střední komparační napětí, U_{IN}^{+} napětí překlápějící do log.1 a U_{IN}^{-} napětí překlápějící do log.0.



Obrázek 5.2: Příklad zpracování signálu komparátorem s hysterezí



Obrázek 5.3: Schema zapojení komparátoru s OZ

Ovšem komparátor tvořený pomocí operačního zesilovače bude limitovaný takzvanou rychlostí přeběhu OZ, která udává, jak rychle dokáže OZ měnit napětí na svém výstupu. Rychlost přeběhu se u typických operačních zesilovačů, například TL084 viz.7, pohybuje okolo $16V\mu s$, což bude pro některá měření rychle se měnícího signálu nedostačující. Pro tento případ je možné použít buď přímo komparátory, které mají zapojení stejné jako OZ, ale jejich vnitřní struktura je pro komparační účely vhodněji přizpůsobena nebo použít „Schmittův“ klopný obvod, jehož funkce je stejná jako u komparátoru s hysterezí, ale jeho rychlost přeběhu je typicky mnohem vyšší, než u OZ a komparační úrovně jsou zvoleny pevně z výroby. „Schmittův“ klopný obvod je umístěn i na vstupním pinu mikrokontroleru a pokud nám budou vyhovovat jeho „šířka“ hystereze 200mV viz.2, tedy rozdíl mezi komparačním napětím do log.0 a do log.1, nemusíme vnější komparátor pro úpravu vstupního signálu použít.

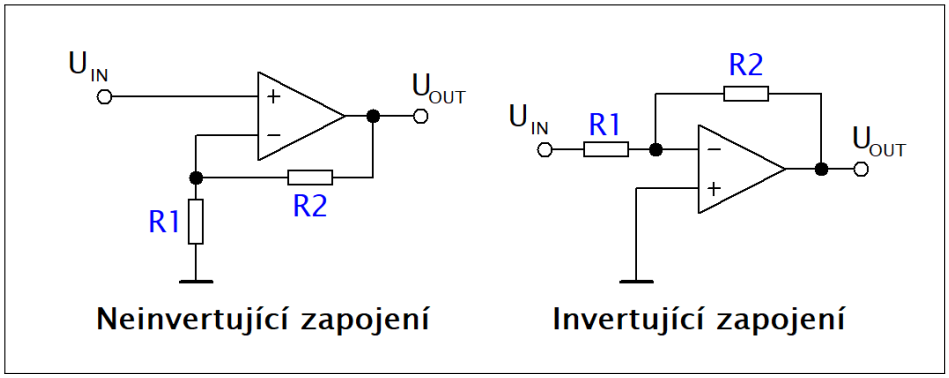
Pro případ, že bude zdroj měřeného signálu slabý a bude pro jeho velikost obtížné nastavit komparátor můžeme tento napěťový signál nejprve zesílit. Přičemž lze opět využít jednoduchého zapojení s operačním zesilovačem. Dvě taková zapojení pak vidíme na obrázku 5.4, kde neinvertující zapojení má výhodu velmi velkého vstupního odporu a je tak vhodné i pro měření napětí z velmi nízkoenergetických zdrojů.

Zesílení neinvertujícího zapojení A_{neinv} bude:

$$A_{neinv} = 1 + \frac{R_2}{R_1} \tag{5.4}$$

a zesílení invertujícího zapojení A_{inv} bude:

$$A_{inv} = -\frac{R_2}{R_1}. \tag{5.5}$$

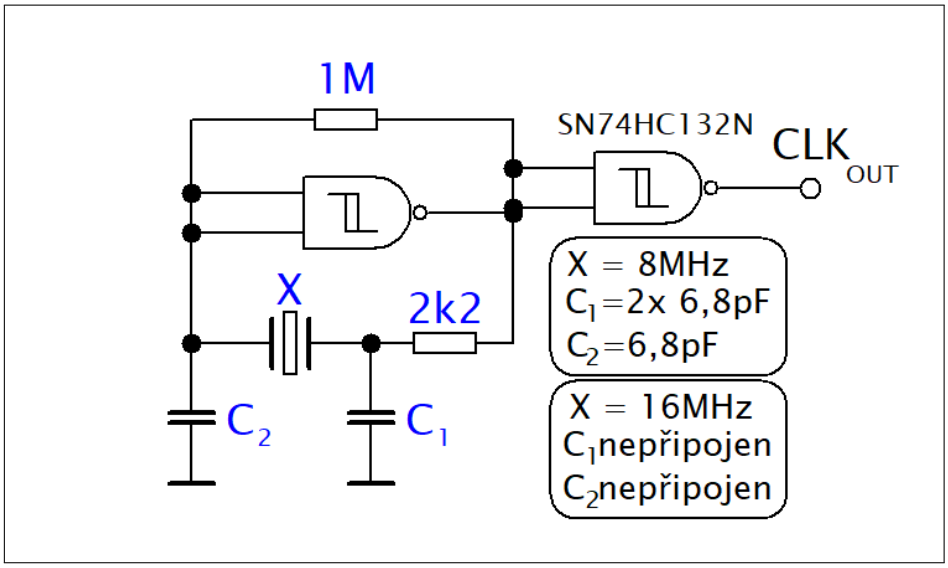


Obrázek 5.4: Schema zapojení zesilovače s OZ

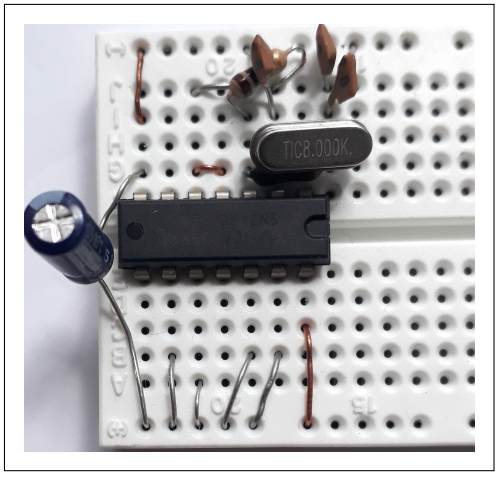
5.2 Obvod vnějšího krystalového oscilátoru

Jak již bylo mnohokrát zmíněno, pro experimenty s většími nároky na přesnost měření budeme u mikrokontroleru používat vnější zdroj hodinového signálu. Zapojení s přesným krystalovým oscilátorem je velmi mnoho a lze od výrobců zakoupit i hotové obvody generující hodinový signál pomocí krystalu. Proto při výběru ukázkového zapojení upřednostníme zapojení s minimem součástek a integrovaným obvodem využitelným nejen při konstrukci generátoru, ale i pro možné předzpracování měřeného signálu. Jak bylo zmíněno výše v této kapitole pro převedení měřeného signálu do definovaných logických napěťových úrovní se hodí využít „Schmittův“ klopný obvod, proto se pokusíme sestavit oscilátor právě s jedním obvodem tohoto typu a to konkrétně s čtyřnásobným logickým NAND integrovaným obvodem viz.8.

Na schématu z obrázku 5.5 je zobrazen příklad zapojení oscilátoru, kde je zapojeno hradlo NAND se spojenými vstupními svorkami fungující jako invertor a ve zpětné vazbě je samotný krystal buzený přes rezistor 2,2kΩ. Dále jsou zde pro krystal 8 MHz dva kondenzátory podporující jeho funkci a rezistor 1MΩ podporující kmitání obvodu. Výstupní signál je následně veden přes hradlo v zapojení invertoru, a to kvůli co nejmenšímu zatížení oscilačního obvodu. Hodnoty rezistorů a kondenzátorů byly voleny a testovány tak, aby zajišťovali co možná nejstabilnější frekvenci generovaného signálu, a to i při zapojení obvodu na nepájivém poli, a přitom byla frekvence signálu co nejblíže požadované. Jak vidíme ze schématu, byla použita pouze dvě ze čtyř možných hradel integrovaných v pouzdře, tedy ostatní budou moci být buď použita k sestavení dalších podpůrných obvodů měření nebo budou jejich vstupy připojeny na nulový potenciál nepájivého pole tak, aby se hradla samovolně nerozkmitala a nerušila tak funkci zapojeného oscilátoru. Na obrázku 5.6 pak vidíme zapojení oscilátoru 8 MHz na nepájivém poli.



Obrázek 5.5: Schema zapojení krystalového oscilátoru



Obrázek 5.6: Zapojení krystalového oscilátoru 8 MHz na nepájivém poli

V následujících dvou tabulkách vidíme změřenou frekvenci, včetně procentuální odchylky od žádané hodnoty, generovanou vytvořeným oscilátorem podle navrženého schématu, a to pro krystal 8 MHz v tabulce 5.1 a pro krystal 16 MHz v tabulce 5.2. Měření jsme provedli s čítačem HP 53131 pro několik různých krystalů a je patrné, že na krystalový oscilátor jsme dostali poměrně velkou odchylku od žádané hodnoty. To je zapříčiněno použitím „Schmittova“ klopného obvodu, který pro tvorbu oscilátoru není ideální, ovšem stále jsme dosáhli o něco větší přesnosti než vnitřní RC oscilátor a oproti RC oscilátoru je ten krystalový mnohem teplotně a časově stálejší, kdy během měření frekvence, u všech typů krystalu, kolísala řádově v desetinách hertzu. Oproti tomu u vnitřního RC (16 MHz) oscilátoru při měření frekvence kolísala v řádu kilohertzů, a proto muselo být při tomto měření použito průměrování. Vnitřní RC oscilátorem jsme měřili stejným přístrojem na několika mikrokontrolerech se spuštěným čítačem generujícím na základě HSI hodinového signálu, 16 MHz PWM signál se střídou 50%. Výsledek měření pak vidíme v tabulce 5.3, odkud je patrné, že všechny MCU bezpečně splnily jednoprocentní odchylku od požadované frekvence udanou výrobcem.

Č. měření	f_{HSE8} [MHz]	δ_{HSE8} [%]
1	8,000576	7,20E-03
2	8,000551	6,89E-03
3	8,000549	6,86E-03

Tabulka 5.1: Měření frekvence HSE(8 MHz)

Č. měření	f_{HSE16} [MHz]	δ_{HSE16} [%]
1	16,003505	2,19E-02
2	16,003583	2,24E-02

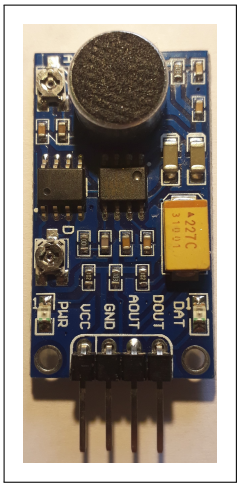
Tabulka 5.2: Měření frekvence HSE(16 MHz)

Č. měření	f_{HSI} [MHz]	δ_{HSI16} [%]
1	16,003198	2,00E-02
2	16,009293	5,81E-02
3	16,005831	3,64E-02
4	15,996410	2,24E-02
5	15,986769	8,27E-02

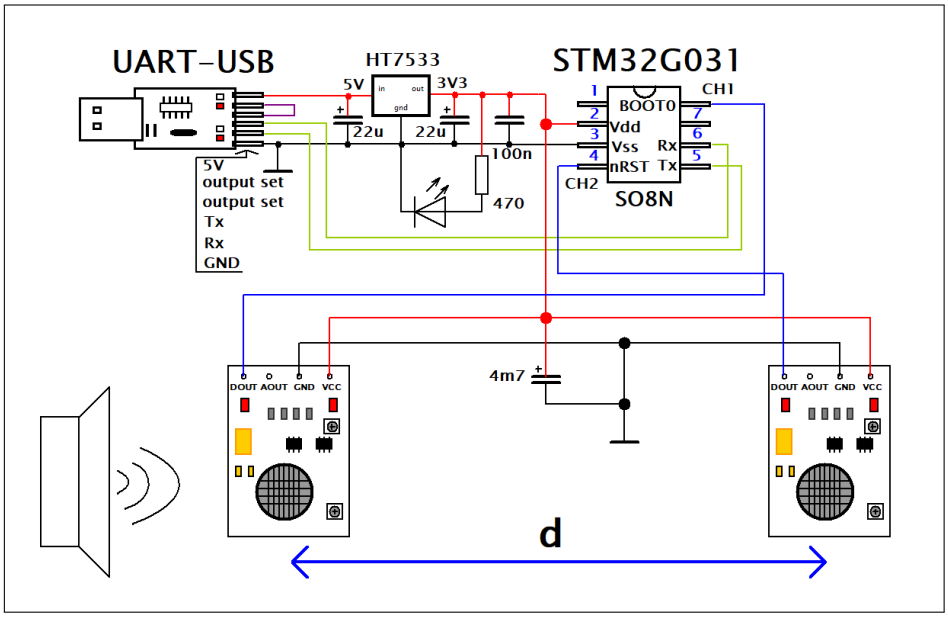
Tabulka 5.3: Měření frekvence HSI(16 MHz)

5.3 Měření rychlosti zvuku

Pro měření rychlosti zvuku využijeme dvou mikrofonů již zapojených s operačním zesilovačem a komparátorem viz. obrázek 5.7. Tento předpřipravený senzor na svorce DOUT indikuje, zda úroveň hluku zaznamenaného mikrofonem nepřekročila určitou mez nastavenou trimerem na senzoru. Tedy experiment bude probíhat v zapojení podle schématu 5.8, kdy oba senzory umístíme do určité změřené vzdálenosti d od sebe a nastavíme indikovanou úroveň hluku tak, aby senzor nezaznamenával šum okolí, ale zaznamenal až námi vygenerovaný měřený zvukový signál. Následně nastavíme čítač do režimu SINGLE se stejnou polaritou hran, zvolíme dvou kanálové měření a zápornou polaritu hran, jelikož používané senzory mají v neaktivním stavu na výstupu log.1 a v aktivním stavu log.0. V tomto případě již můžeme zapnout měření a vytvořit rázový zvukový signál například pomocí nějakého online generátoru vPC, kdy zdroj zvuku musí být v jedné přímce s oběma senzory a senzor připojený na první kanál čítače (fyzický pin č.8 pouzdra SO8N) musí přijmout zvukový signál jako první.



Obrázek 5.7: Zvukový senzor

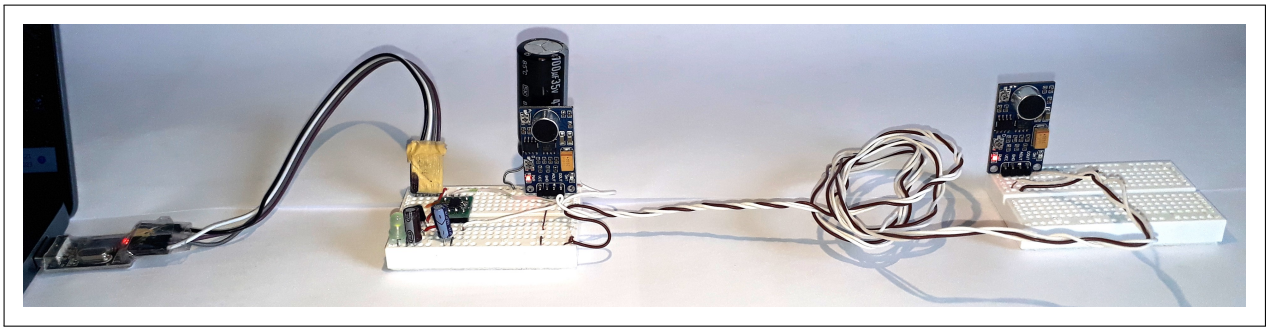


Obrázek 5.8: Schéma zapojení měření rychlosti zvuku se dvěma mikrofony

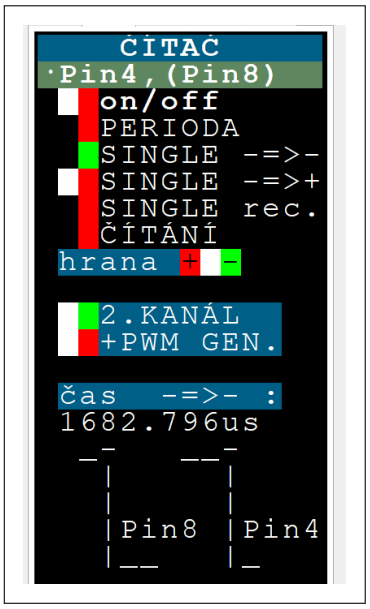
Jak vidíme ve schématu výše, před vstup napájení do senzorů byl ještě přidán kondenzátor velké kapacity, aby kompenzoval jejich nárazové odběry. Měření jsme provedli ve vzdálenosti senzorů $d = 60cm$ a zdroj zvukového signálu jsme použili notebook s online generátorem signálu, kde jsme navolili frekvenci 250 Hz a pilovitý průběh. Frekvenci jsme volili tak, aby byla jí příslušná perioda delší než předpokládaná změřená doba šíření zvuku na zvolenou vzdálenost. Z principu tohoto režimu měření totiž v případě, že přijde hrana na kanál jedna se čítač nuluje a musí sem tedy přijít pouze jedna hrana, jinak měření neproběhne správně. Měření jsme provedli při teplotě 25 °C a vrátilo nám přibližně hodnotu $T_d \approx 1683\mu s$, viz. obrázek 5.10, tento čas pak přepočteme na rychlost zvuku, jako:

$$v = \frac{d}{T_d} \approx 356,5ms^{-1}, \tag{5.6}$$

odkud vidíme, že se tato hodnota velmi blíží obecně udávané rychlosti zvuku ve vzduchu při teplotě 20°C $v_{20} = 334ms^{-1}$. Fotografie zapojení experimentu je potom ke shlédnutí na obrázku 5.9.



Obrázek 5.9: Zapojení experimentu měření rychlosti zvuku se dvěma mikrofony

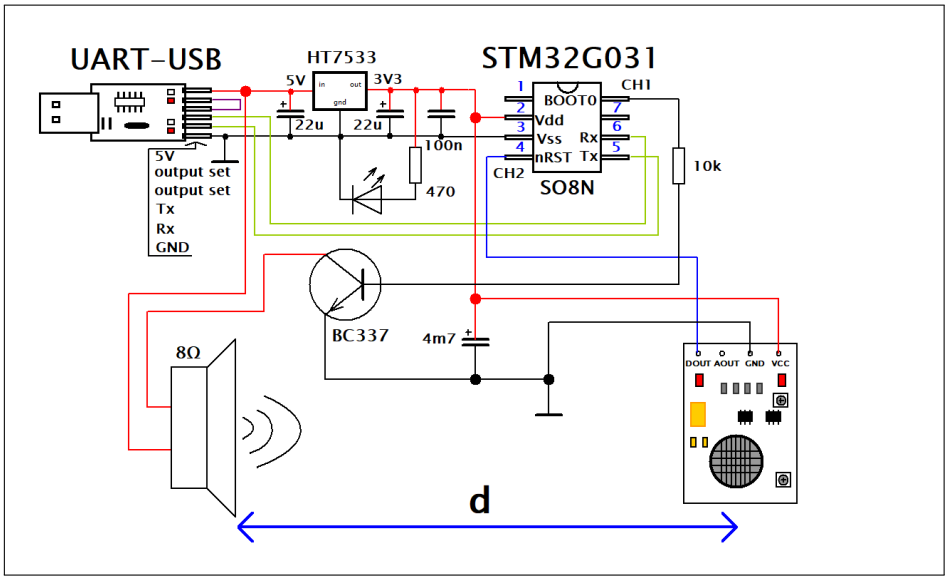


Obrázek 5.10: Terminál experimentu měření rychlosti zvuku se dvěma mikrofony

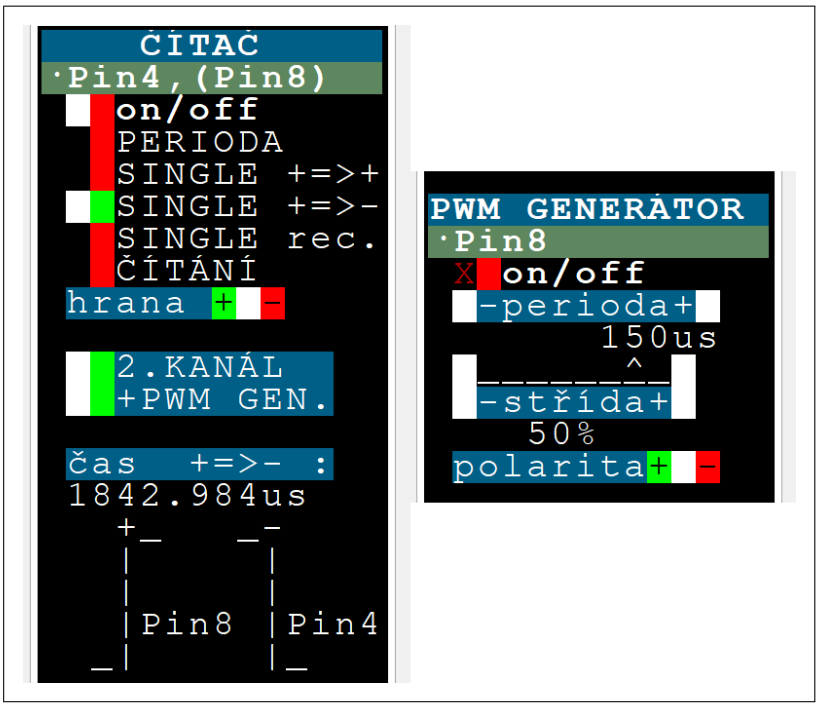
Druhý způsob použitelný pro měření rychlosti zvuku je v zapojení podle schématu na obrázku 5.11, kde bude mikrokontroler generovat měřený zvukový signál pomocí PWM generátoru a k měření tak budeme potřebovat jen jeden senzor zvuku a dodatečně reproduktor s budícím tranzistorem. V terminálu pro tento experiment nastavíme režim SINGLE čítače s opačnou polaritou hran a první náběžnou hranu (tedy druhou spádovou) a následně zvolíme dvou kanálové měření a připnutí PWM generátoru, jak vidíme na obrázku 5.12. Na terminálu generátoru ještě zvolíme libovolnou periodu, například 150 μs. Teď už jen umístíme reproduktor a senzor do vzdálenosti $d = 60cm$ a zapneme měření. V tomto případě jsme naměřili $T_d \approx 1843\mu s$, z čehož dopočteme rychlost, jako:

$$v = \frac{d}{T_d} \approx 325,6ms^{-1}, \tag{5.7}$$

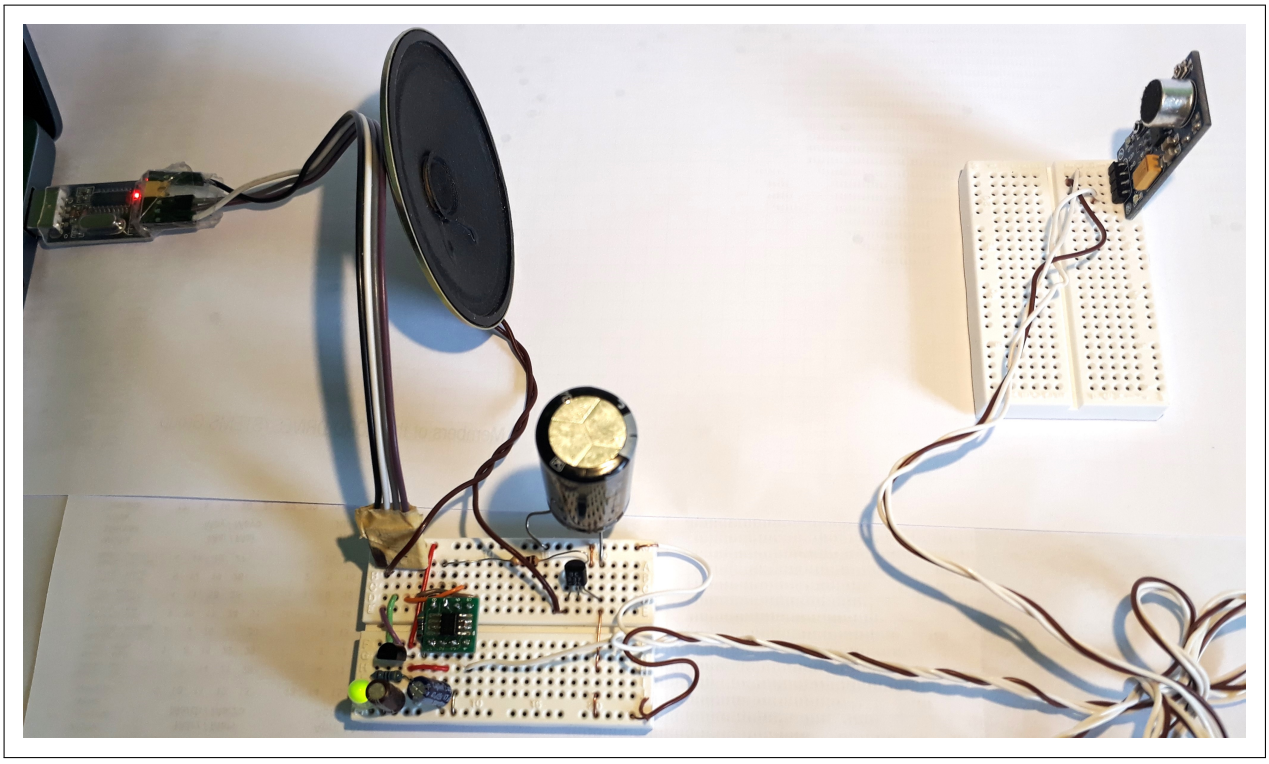
což je opět velmi blízko očekávané hodnotě, ovšem tentokrát je rychlost o něco nižší, což je způsobeno setrvačností membrány použitého reproduktoru. Fotografie zapojení experimentu je potom ke shlédnutí na obrázku 5.13.



Obrázek 5.11: Schéma zapojení měření rychlosti zvuku s reproduktorem



Obrázek 5.12: Terminál experimentu měření rychlosti zvuku s reproduktorem



Obrázek 5.13: Zapojení experimentu měření rychlosti zvuku s reproduktorem

5.4 Měření indukčnosti

Vezměme si příkladovou úlohu s elektrotechniky kdy máme vzduchovou cívku neznámé indukčnosti viz. obrázek 5.14, která byla navinuta z kabelu o délce $d = 100m$ na buben o poloměru $r = 3cm$ a vznikl vnější průměr $r_v = 6,5cm$, lze tedy dopočíst přibližný počet závitů N, jako:

$$N = \frac{d}{2\pi\frac{r_v+r}{2}} = 335. \tag{5.8}$$

Dále lze spočítat indukčnost cívky L pomocí magnetické vodivosti λ, permeabilitě μ, průřezu cívky $S = \pi r^2$ a délky cívky $l = 5,5cm$, jako:

$$L = \lambda N^2 = \mu\frac{S}{l}N^2 = 4\pi 10^{-7}\frac{\pi r^2}{l}N^2 = 7,25mH. \tag{5.9}$$

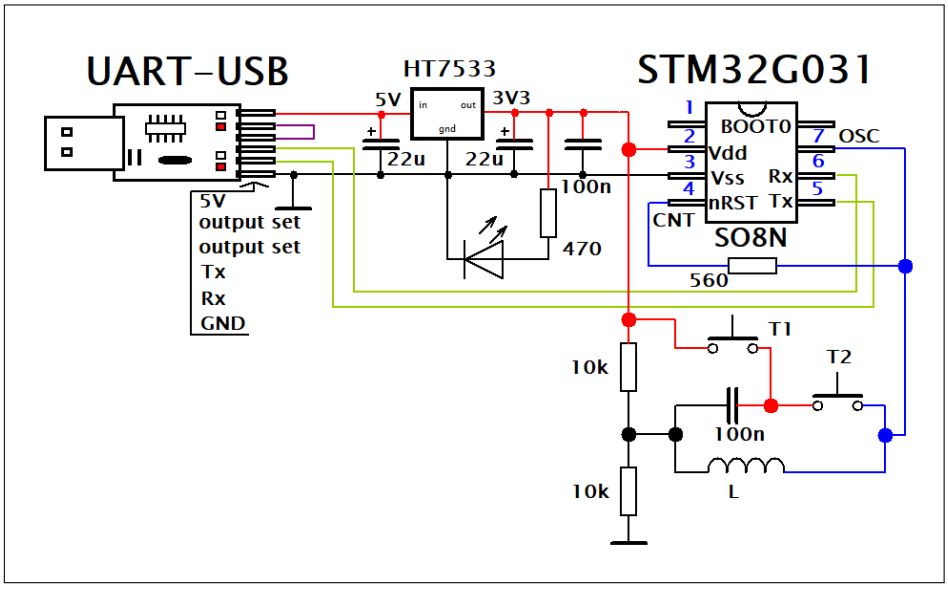
Výslednou hodnotu pak lze experimentálně ověřit pomocí jednoduchého zapojení v paralelní rezonanci vytvořené cívky a známé kapacity C, kdy hodnotu indukčnosti bude možné dopočíst pomocí vzorce pro LC rezonanci:

$$f_0 = \frac{1}{2\pi\sqrt{LC}} \rightarrow T_0 = 2\pi\sqrt{LC} \rightarrow L = \frac{T_0}{2\pi}\frac{1}{C}, \tag{5.10}$$

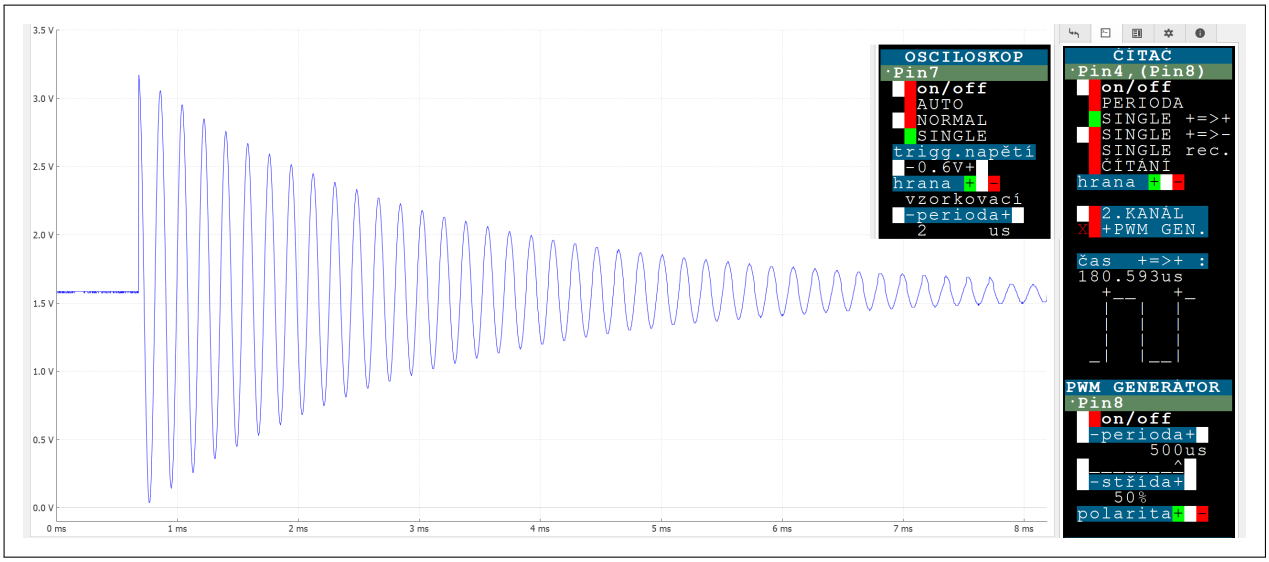
kde f_0 je rezonanční frekvence a T_0 jí odpovídající perioda.

Obrázek 5.14: Měřená vzduchová cívka

Pro ověření výše uvedených vzorců budeme měřit periodu odpovídající rezonanční frekvenci a to podle zapojení z obrázku 5.15, kde jsme známou kapacitu zvolili, jako $C = 100nF$. Rezonanční obvod bude generovat střídavý signál, a tak si jednu jeho svorku připojíme na virtuální střed napájecího napětí vytvořeného odporovým děličem, aby bylo možné signál pozorovat v celém jeho rozsahu. Sepnutým tlačítkem T1 a rozepnutým tlačítkem T2 pak budeme nabíjet kondenzátor C a následně sepnutým T2 a rozepnutým T1 budeme vytvářet měřený rezonanční obvod. Budeme měřit současně osciloskopem, který nastavíme do režimu single s „trigger-em“ v 0,8 V na kladnou hranu při vzorkovací periodě $5\mu s$ a současně čítačem, který nastavíme na SINGLE režim se stejnou polaritou hran a zvolíme vzestupné hrany, jak vidíme na obrázku terminálu 5.16.



Obrázek 5.15: Schéma zapojení experimentu měření indukčnosti



Obrázek 5.16: Terminál experimentu měření indukčnosti

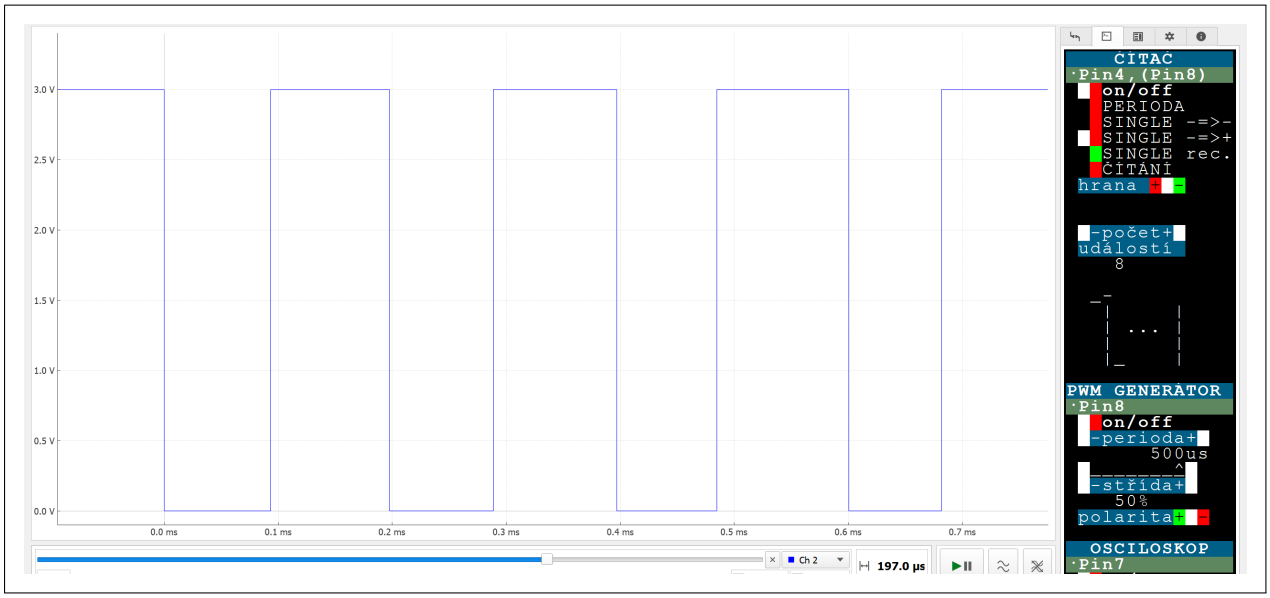
Na obrázku 5.16 je i zobrazen výsledek měření po tom co jsme nejprve zapnuli čítač i osciloskop a následně tlačítkem T1 kondenzátor nabili a tlačítkem T2 experiment spustili. Osciloskopem tedy byl změřen časový průběh exponenciálně utlumované oscilace rezonančního obvodu a čítačem pak její perioda, jako $T_0 \approx 180,6\mu s$. Z tohoto výsledku můžeme dopočíst změřenou indukčnost, jako:

$$L = \frac{T_0}{2\pi}\frac{1}{C} = 8,3mH, \tag{5.11}$$

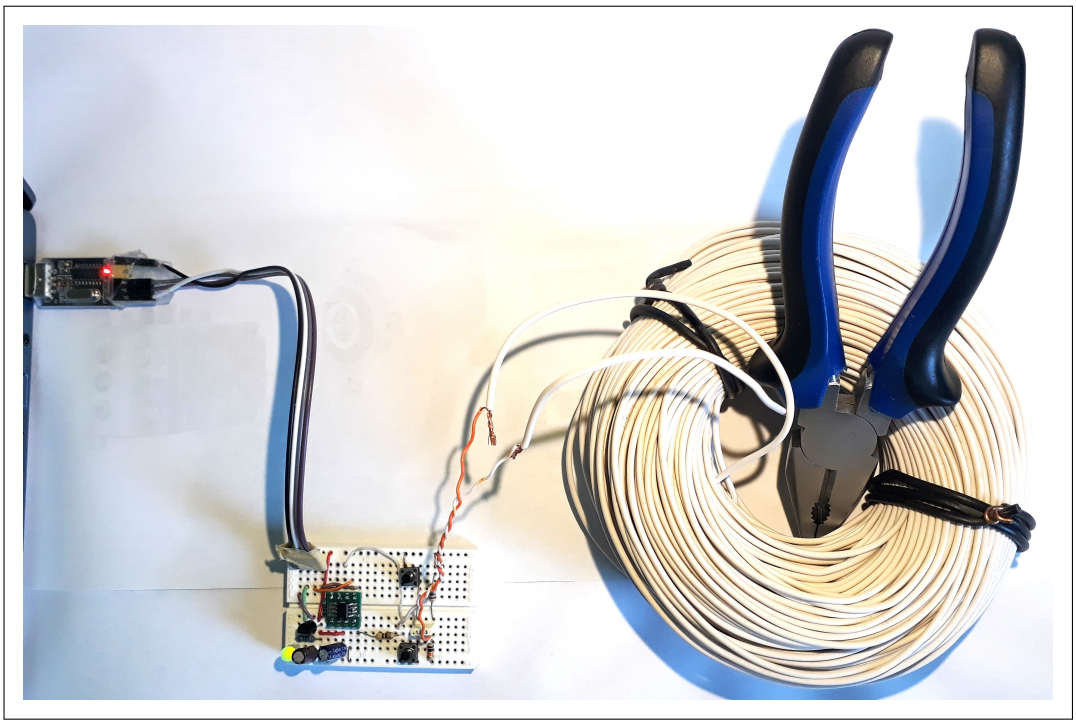
což se téměř přesně shoduje s vypočtenou indukčností a lze si tím ověřit správnost používaných vztahů. Tímto experimentem lze také ověřit změnu indukčnosti pro různá jádra cívky. Vložíme tedy feromagnetický předmět do jádra cívky a provedeme měření znovu. Tentokrát ovšem signál zachytíme pouze čítačem v režimu SINGLE záznam, kde nastavíme osm zaznamenaných událostí a zápornou počáteční hranu. Výsledek pak vidíme na obrázku 5.17 odkud jsme si ověřili stálost frekvence generované obvodem a odečetli odtud $T_{0fe} \approx 197\mu s$, výsledná indukčnost se tedy zvětšila na:

$$L = \frac{T_{0fe}}{2\pi}\frac{1}{C} = 9,8mH. \tag{5.12}$$

Na fotografii 5.18 pak lze shlédnout zapojení experimentu včetně vkládaného předmětu do jádra.



Obrázek 5.17: Terminál experimentu měření indukčnosti s feromagnetickým jádrem



Obrázek 5.18: Zapojení experimentu měření indukčnosti

5.5 Měření gravitačního zrychlení

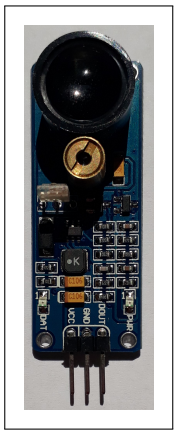
Tentokrát se budeme snažit jednoduchým pokusem přibližně změřit velikost gravitačního zrychlení. Budeme vycházet ze vzorce pro dráhu volného pádu h, kterou těleso uletí za čas t, je přitom zrychlováno gravitačním zrychlením g a v čase nula mělo nulovou počáteční rychlost:

$$h = \frac{1}{2}gt^2. \tag{5.13}$$

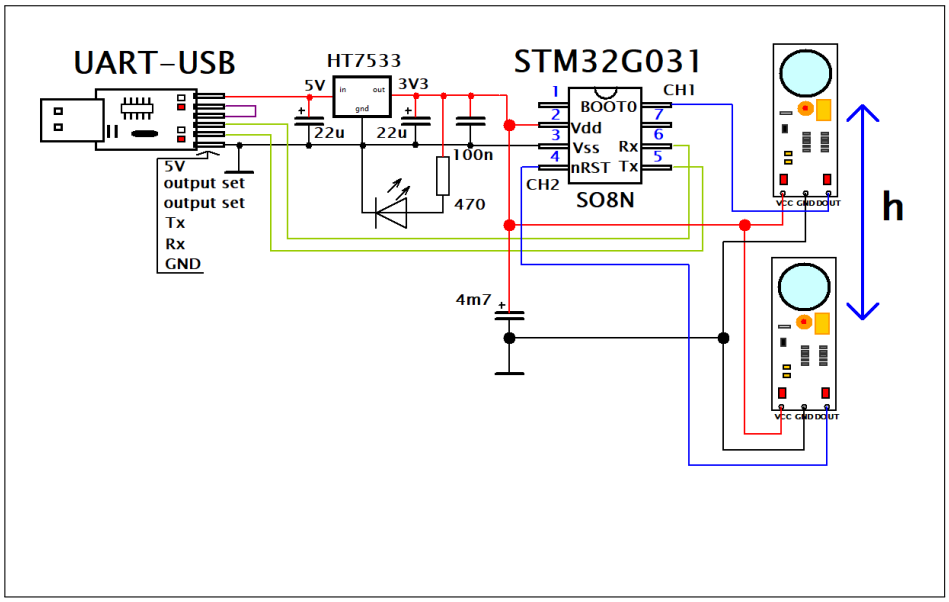
Z tohoto vzorce si pak vyjádříme měřené gravitační zrychlení, jako:

$$g = \frac{2h}{t^2}. \tag{5.14}$$

Čítačem tedy budeme chtít měřit čas t, za který námi zvolené těleso volným pádem spadne z výšky h při nulové počáteční rychlosti. K tomuto účelu použijeme dva odrazové, laserové senzory polohy již v předpřipraveném zapojení, jak vidíme na obrázku 5.19, které umístíme do vzdálenosti $h = 60cm$, kdy první bude ve výšce 70 cm nad zemí a druhý ve výšce 10 cm nad zemí a zapojíme jejich výstupy na první a druhý kanál čítače, jak vidíme na schématu 5.20.



Obrázek 5.19: Laserový odrazový senzor polohy



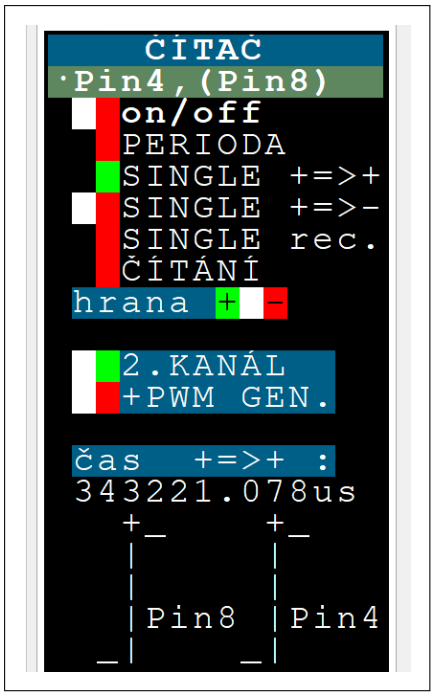
Obrázek 5.20: Schéma zapojení měření gravitačního zrychlení

Po připravení experimentu již jen nastavíme čítač na SINGLE měření se dvěma kanály a stejnou polaritou hran. I když jsou senzory aktivní na log.0, tedy při zaznamenání tělesa vytvoří sestupnou hranu, ponecháme zde kladné reakční hrany, jelikož budeme měřit čas od opuštění dosahu prvního do opuštění dosahu druhého senzoru, jelikož bude snadnější naladit přesnou polohu, kdy těleso opouští první senzor. K měření tedy použijeme míček, který rukou nastavíme do takové polohy, kdy jej ještě první senzor zaznamenává, následně spustíme měření a vypustíme míček. Výsledný změřený čas spolu s

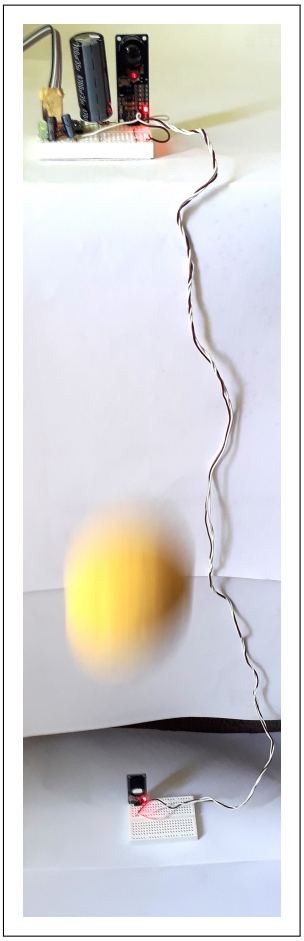
nastavením terminálu vidíme na obrázku 5.21 a fotografii experimentu na obrázku 5.22. Ze změřeného času $t = 0,343s$ a výšky h pak dopočteme gravitační zrychlení, jako:

$$g = \frac{2h}{t^2} \approx 10,19ms^{-2}, \tag{5.15}$$

odkud vidíme, že se hodnota poměrně dobře přibližuje obecně udávané velikosti gravitačního pole 9,81 ms^{-2}.



Obrázek 5.21: Terminál měření gravitačního zrychlení



Obrázek 5.22: Zapojení experimentu měření gravitačního zrychlení

Dalším možným výpočtem gravitačního zrychlení je z rychlosti tělesa v při dopadu z výšky h, kdy mělo těleso v této výšce nulovou rychlost, kde vyjdeme ze vzorců:

$$h = \frac{1}{2}gt^2, \qquad v = gt \tag{5.16}$$

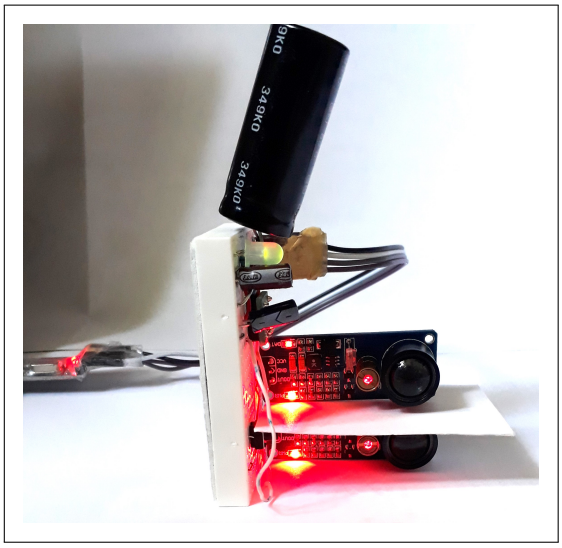
a po jejich sloučení pro gravitační zrychlení dostáváme:

$$g = \frac{v^2}{2h}, \tag{5.17}$$

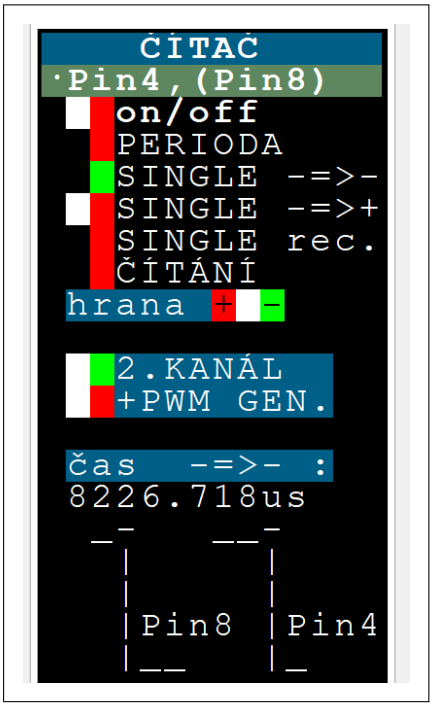
tedy budeme měřit rychlost v při dopadu tělesa z výšky $h = 71cm$ s nulovou počáteční rychlostí. K měření rychlosti opět využijeme dvou laserových senzorů polohy ve stejném zapojení, ovšem tentokrát budou umístěny ve známé vzdálenosti $d = 3cm$ a to u země v místě dopadu, jak vidíme na obrázku experimentu 5.23. Budeme tedy měřit čas t za který těleso uletí onu dráhu d a z toho pak vypočteme rychlost v. Nastavení čítače bude v režimu SINGLE se stejnou polaritou hran, dvěma kanály a zápornými reakčními hranami, jak vidíme na obrázku 5.24, kde vidíme i výsledek měření $t \approx 8227\mu s$ z čehož dopočteme:

$$g = \frac{(\frac{d}{t})^2}{2h} \approx 9,36ms^{-2}, \tag{5.18}$$

což je opět velice blízko obecně udávané hodnotě gravitačního zrychlení.



Obrázek 5.23: Zapojení experimentu měření rychlosti



Obrázek 5.24: Terminál měření rychlosti

Kapitola 6

Zhodnocení dosažených výsledků

V této práci byl realizován samotný čítač s požadovanými režimy měření, osciloskop pro zobrazení analogového signálu, logický analyzátor pro zobrazení logického signálu a PWM generátor pro testovací účely měřicího přístroje nebo jako zdroj spouštěcího signálu v experimentech. V této realizaci bylo dosaženo následujících parametrů jednotlivých měřicích režimů:

- **Měření periody a střídy, reciproční měření frekvence**
  - Rozsah periody: 125 ns až 67,1 s
  - Rozlišení: $\frac{1}{N}15,625$ ns
  - Průměrování z N vzorků, kde N je: 1 až 250
- **Měření časového odstupu dvou po sobě jdoucích hran**
  - Rozlišení: 15,625 ns
  - Obě hany ze stejného zdroje/každá hrana z jiného zdroje
  - 1.hrana náběžná/sestupná
  - 2.hrana náběžná/sestupná
  - Pro dvoukanálové měření možnost začít měřit zapnutím PWM generátoru
- **Záznam časového odstupu N po sobě jdoucích hran**
  - Rozlišení: 15,625 ns
  - Maximální délka záznamu: 67,1 s
  - Počet zaznamenatelných hran: 2 až 500
  - Začátek na náběžnou/sestupnou hranu
- **Měření frekvence**
  - Rozsah: 0 MHz až 64 MHz
  - Rozlišení: $\frac{1}{T_G}$ [Hz]
  - Doba „hradlování“ T_G: 100 ms až 20 s
  - Možnost zvolit prosté čítání hran
  - Možnost čítání vzestupných/sestupných hran
- **Osciloskop**
  - Rozlišení vzorků: 8 bitů

- Vzorkovací rychlost: 5 ksample/s až 2 Msample/s
- Délka záznamu: 4096 vzorků
- „Trigger“ na vzestupnou/sestupnou hranu
- „Pre-trigger“: 400 vzorků
- „Trigger-ovací“ úroveň napětí: $0,2$ V až $3,2$ V po $0,2$ V
- Možnost „single“ záznamu

- **Logický analyzátor**
  - Délka záznamu: 1 k až 32 k vzorků
  - Vzorkovací rychlost: 125 ksample/s až 16 Msample/s
  - „Trigger“ na vzestupnou/sestupnou hranu

- **PWM Generátor**
  - Rozsah periody: 1 μs až $32,767$ s
  - Rozsah střídy: 0% až 100% po 5%
  - Možnost změny polarity (invertování) výstupního signálu

Kapitola 7

Závěr

V této práci bylo za úkol realizovat softwarově definovaný univerzální čítač na mikrokontroleru STM32G031 využitelný zejména pro výukové experimenty. Čítač měl umožňovat přímé i recipročí měření frekvence včetně měření délky pulzů, průměrování výsledků a dvoukanálového měření. Dále měl být měřicí přístroj schopen zobrazit signál v analogové i logické doméně, a to pomocí aplikace Data Plotter, ve které mělo být realizováno i ovládání přístroje. Součástí práce byl i návrh zapojení krystalového oscilátoru pro provoz mikrokontroleru s vnějším hodinovým signálem a několika výukových experimentů realizovatelných s vytvořeným měřicím přístrojem.

Z výsledných parametrů přístroje v kapitole 6 je vidět, že i když byl využit potenciál použitého mikrokontroleru poměrně dobře, v některých parametrech takových výsledků, jako komerční přístroje, nedosahuje. Ovšem jeho přesnost a možnosti stále plně vyhovují většině běžných výukových experimentů a díky jeho univerzálnosti, dostupnosti, nízké ceně a jednoduchému ovládání je pro demonstrační experimenty ve výuce vhodným měřicím přístrojem. V celé práci byl kladen důraz na srozumitelné vysvětlení problematiky čítačů a jejich nastavení v programu, aby bylo možné tuto práci použít jako studijní materiál. Díky dosažení všech zmíněných cílů tedy považuji zadání této práce za naplněné v celé jeho šíři. Čítač by pak mohl být v případném dalším vývoji doplněn o některá specifická měření, která by se mohla objevit během vymýšlení dalších experimentů s tímto přístrojem. Dále by mohlo být navrženo více experimentů s dalšími senzory fyzikálních veličin a výhodnější zapojení vnějšího krystalového oscilátoru.

Literatura

[1] STMicroelectronics. *STM32G0x1 RM0444 Reference manual*[online]. URL: `https://www.st.com/resource/en/reference_manual/rm0444-stm32g0x1-advanced-armbased-32bit-mcus-stmicroelectronics.pdf`

[2] STMicroelectronics: *STM32G031J6 Datasheet*[online]. URL:`https://www.st.com/resource/en/datasheet/stm32g031j6.pdf`

[3] Bittman, Jan: *Mikrořadič jako náhrada obvodu typu Smart Circuit*. ČVUT-FEL, 2020.

[4] Cejp, Martin: *Virtuální přístroj s mikrořadičem pro analýzu signálu v modulační doméně*. ČVUT-FEL, 2017.

[5] Dujava Jozef: *Software-defined oscilloscopes with terminal interface*. ČVUT-FEL, 2022.

[6] Maier Jiří: *Univerzální GUI pro osciloskopické PC aplikace*. ČVUT-FEL, 2021.

[7] STMicroelectronics: *TL084 Datasheet*[online]. URL: `https://www.st.com/resource/en/datasheet/cd00000493.pdf`

[8] *SN74HC132N Datasheet*[online]. URL: `https://www.ti.com/lit/ds/symlink/sn54hc132.pdf`

Příloha A

Vysvětlivka

Zkratka	Význam
MCU	Mikrokontroler
PC	Osobní počítač
CPU	"Central Processing Unit"jádro MCU
USB	Univerzální sériová sběrnice
USART	Univerzální sériový asynchronní přijímač a vysílač
UART	Univerzální asynchronní přijímač a vysílač (stejný jako USART)
DMA	Blok přímého přístupu do paměti
ADC	Analogově digitální převodník
SAR	AD převodník s postupnou aproximací
AWD	"Analog Watchdog"
GPIO	Brána univerzálních vstupů a výstupů
TIM	Čítač
PSC	Registr předděličky čítače
ARR	"Auto-Reload"registr čítače
CCx	"Capture/Compare"jednotka čítače, kanál x
TRGO	Zdroj "trigger"událostí od čítače
HSI	Vnitřní oscilátor v MCU (16 MHz)
HSE	Vnější oscilátor MCU
OZ	Operační zesilovač

Tabulka A.1: Přehled použitých zkratek