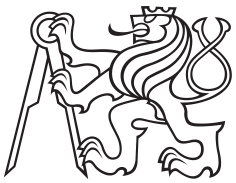
Diplomová práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra měření

Softwarově definovaný osciloskop s mikrořadičem STM32F103

Bc. Jakub Pařez

Vedoucí práce: doc. Ing. Jan Fischer, CSc.
Studijní program: Kybernetika a Robotika
Obor: Kybernetika a Robotika
Květen 2021

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Pařez** Jméno: **Jakub** Osobní číslo: **492966**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra měření**

Studijní program: **Kybernetika a robotika**

Studijní obor: **Kybernetika a robotika**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Softwarově definovaný osciloskop s mikrořadičem STM32F103

Název diplomové práce anglicky:

Software defined oscilloscope based on STM32F103

Pokyny pro vypracování:

Navrhněte a realizujte softwarově definovaný osciloskop s mikrořadičem STM32F103, kde všechny funkce budou realizovány programově na PC, nebo pouze s využitím bloků mikrořadiče. Přístroj bude mít obdobné funkce, jako má standardní osciloskop nebo logický analyzátor, avšak s omezeními danými použitým způsobem realizace. Návrh orientujte tak, aby výsledný přístroj bylo možno používat v hromadné distanční výuce při domácí práci studentů.
Navrhněte vhodný protokol pro komunikaci mezi mikrořadičem a PC, vytvořte potřebný firmware i PC aplikaci. Celý návrh koncipujte tak, aby jej bylo možno snadno upravit i pro jiné mikrořadiče. Správnost návrhu ověřte implementací i pro mikrořadiče STM32F303, STM32L072, STM32L412, případně mikrořadiče řady STM32Gxx.
Prověřte též možnosti implementace dalších funkčních bloků, jako je impulsní generátor, případně čítač.

Seznam doporučené literatury:

[1] Yiu, J.: The Definitive Guide to ARM® Cortex®-M3 and Cortex®-M4 Processors,
[2] STMicroelectronics: RM0008 Reference manual, STM32F101_107
[3] STMicroelectronics: DS5319 - STM32F103 Data

Jméno a pracoviště vedoucí(ho) diplomové práce:

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

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

Datum zadání diplomové práce: **23.01.2021** Termín odevzdání diplomové práce: _____

Platnost zadání diplomové práce:
do konce letního semestru 2021/2022

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

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

. _____
Datum převzetí zadání

_____
Podpis studenta

Poděkování

Rád bych poděkoval svému vedoucímu diplomové práce doc. Ing. Janu Fischerovi, CSc. za profesionální vedení práce, mnoho cenných rad a připomínek, ale především za možnost vytvořit něco, co mi dělalo a bude dělat radost následujících mnoho let. Dále také děkuji Ing. Hladíkovi za inspiraci a skvělou technickou podporu.

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, 20. května 2021 .....

Abstrakt

Diplomová práce se zabývá návrhem a realizací softwarově definovaného osciloskopu, který má být cílen na hromadnou distanční výuku praktické elektroniky. Jako srdce osciloskopu byly zvoleny mikrořadiče STM32. Firmware byl vyvinut v jazyce C univerzálním způsobem, proto uživatel není omezen při výběru hardware. Uživatel může volit mezi řadiči STM32F103C8, STM32F103RE, STM32F303RE nebo STM32L412, kdy většina z nich je dostupná ve vývojové sadě Nucleo nebo BluePill. Veškeré funkce jsou realizovány pomocí interních bloků mikrořadiče. Data se do klientské aplikace v PC zasílají standardním SCPI protokolem. Připojení je volitelné mezí USB nebo UART, záleží na konkrétním typu hardware. Aplikace byla vytvořena v Qt frameworku pro operační systémy Windows, Linux a macOS.

Při vývoji a realizace funkcí osciloskopu byl kladen důraz především na tři body. Prvním je stabilita, implikovaná dodržením správných postupů softwarového inženýrství, jako jsou fáze vývoje, UML digramy, standardy a testování. Druhým je uživatelská vstřícnost, vycházející z vlastního grafického návrhu PC aplikace, dokumentovaného kódu a implementace textového komunikačního protokolu. Třetím je funkčnost, zajištěna inspirací z komerčních produktů a spektrem hardwaru a systémů, ze kterých může uživatel vybírat.

Kromě osciloskopu byl realizován také logický analyzátor a voltmetr. Tato trojice instrumentů je exkluzivního charakteru a představuje primární scénář použití. Dále byl realizován čítač, PWM generátor a signálový generátor. Tyto tři doplňkové přístroje jsou na ostatních nezávislé. Parametry, jako maximální hodnoty a režimy podléhají konkrétnímu typu mikrořadiče, signálový generátor je pak přítomen pouze u řadičů s DAC převodníkem, jako je například STM32F303RE.

Osciloskop dostal jméno EMBO a veškeré jeho zdrojové kódy jsou zveřejněny.[1]

Klíčová slova: EMBO, Embedded, Oscilloscope, vývoj, ST, STM32, STM32F103, STM32F303, STM32L412, BluePill, Nucleo, osciloskop, logický analyzátor, voltmetr, čítač, PWM generátor, signálový generátor, multiplatformní, Qt, C++, UML, github

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

Abstract

The diploma thesis deals with the design and implementation of a software-defined oscilloscope, which should be aimed at mass distance learning of practical electronics. STM32 microcontrollers were chosen as the heart of the oscilloscope. Firmware was developed in the C language in a universal way, so the user is not limited in choosing hardware. User can choosen between STM32F103C8, STM32F103RE, STM32F303RE or STM32L412, most of which are available in the Nucleo or BluePill development kit. All functions are implemented using internal microcontroller blocks. The data is sent to the client application on the PC using the standard SCPI protocol. The connection is selectable between USB or UART and depends on the specific type of hardware. The application was created in the Qt framework for Windows, Linux and macOS operating systems.

During the development and implementation of the oscilloscope functions, emphasis was placed on three points. The first is stability, implied by following good software engineering practices, such as development phases, UML diagrams, standards, and testing. The second is user-friendliness, based on the own graphic design of the PC application, documented code and the implementation of a text communication protocol. The third is functionality, provided by inspiration from commercial products and a range of hardware and systems from which the user can choose.

In addition to the oscilloscope, a logic analyzer and voltmeter were also implemented. These three instruments are exclusive and represents the primary scenario of use. Furthermore, a counter, PWM generator and signal generator were implemented. These three secondary devices are independent of each others. Parameters, such as maximum values and modes, depend on the specific type of microcontroller, the signal generator is available only for controllers with a DAC converter, such as STM32F303RE.

The oscilloscope was named EMBO and is completely open-sourced.[1]

Keywords: EMBO, Embedded, Oscilloscope, development, ST, STM32, STM32F103, STM32F303, STM32L412, BluePill, Nucleo, oscilloscope, logic analyzer, voltmeter, counter, PWM generator, signal generator, multiplatform, Qt, C++, UML, github

Title translation: Software defined oscilloscope based on STM32F103

Obsah

Obrázky

Tabulky

Kapitola 1

Úvod

Ve výuce elektroniky je na rozdíl od humanitních oborů nepostradatelná její praktická část. Motivací této práce je zkvalitnění distanční výuky, kterou nejen dnešní doba kovidová vyžaduje. Dále to je pak zpřístupnění funkcionalit laboratorních přístrojů, s parametry omezenými použitým hardwarem.

Ne každý student si může domů pořídit osciloskop a funkční generátor, přitom tyto nástroje jsou nezbytné jak při studiích tak při následném zaměstnání, například ve vývoji. Ne každý potřebuje vzorkovací rychlost osciloskopu 1 Gsps s hloubkou paměti 1 MB na kanál. V mnoha případech by uživatel uvítal tyto hodnoty 1000 krát menší, za cenu toho, že by mohl měřit ihned a skoro zadarmo. Toho lze docílit více způsoby. Například lze použít vstup ze zvukové karty, to je ale pomalé a nepraktické. Dále lze zvolit třeba LEO, populární projekt od Ing. Hladíka. Nebo se může podobný osciloskop vytvořit z nuly, což je náplní této práce.

Úkolem je vytvořit multifunkční laboratorní přístroj na platformě STM32, jehož jádro poběží na mikrořadičích z řad F1, F3 a L4, případně L0 a G0, kdy veškerá funkcionalita bude realizována pomocí vnitřních bloků čipu. Podobně jako u LEO je tento přístroj cílen také na vývojové desky STM32 Nucleo, na rozdíl od něj však primárně cílí na extrémně levnou desku s STM32F103 jménem BluePill, která lze pořídit za zhruba 100 Kč. Za tuto cenu uživatel získá osciloskop, logický analyzátor, voltmetr, čítač a PWM generátor, a celý přístroj se mu vejde do kapsy. Pochopitelně to je ale bez ošetření úrovní vstupních a výstupních signálů, což je možné řešit pomocí několika odporových děličů. O něco dražší vývojová deska Nucleo s STM32F303 dále nabízí DAC a tím i signálový generátor. Osciloskop musí být multiplatformní, minimálně spustitelný na systémech Windows, Linux a macOS.

Osciloskop dostal jméno EMBO neboli EMBedded Oscilloscope a má ambicí se stát švýcarským nožem každého elektrotechnika. V této práci bude detailně popsána každá část vývoje. Osciloskop bude po důkladném otestování zveřejněn na serveru github včetně zdrojových kódu, firmware a PC aplikace. [1]

Kapitola 2

Analýza zadání

V této kapitole bude shrnuta motivace a požadavky na vývoj softwarově definovaného osciloskopu, dále už jen osciloskopu EMBO (EMBedded Oscilloscope). Budou popsány potřebné metody a technologie, a vždy bude zvolena jedna konkrétní s patřičným odůvodněním.

2.1 Základní požadavky

Hlavním úkolem je vytvořit osciloskop, který bude dostupný pro široké spektrum lidí. Převážně se cílí na studenty elektrotechniky a distanční výuku, středoškolskou i vysokoškolskou. Mezi základní parametry pro dosažení cíle patří zanedbatelná pořizovací cena, jednoduchá instalace a přívětivé uživatelské rozhraní. Aby byl využit potenciál takového zařízení, bude kromě osciloskopu obsahovat další přístroje jako logický analyzátor, voltmetr, čítač, PWM a signálový generátor. Aplikace bude spustitelná na majoritně používaných operačních systémech a firmware zařízení bude dostupný pro více mikrořadičů a vývojových desek.

Osciloskop je pro každého elektrotechnika klíčovým nástrojem. Vznik analogového osciloskopu umožnil pohled do nitra elektrických obvodů. Dnešní digitální osciloskopy jsou vysoce výkonné přístroje, které často kombinují mnoho funkcí dohromady. Tomu také odpovídá jejich cena. Každý student tak jistě ocení multifunkční zařízení s podobnými funkcemi, horšími parametry s cenou okolo 100 Kč. Horší parametry často nemusí znamenat omezení. Pro běžné nízkofrekvenční obvody stačí ve většině případů vzorkovat rychlostí do 1 MSps. Nejlevnější stolní osciloskopy přitom začínají s parametry na hodnotách 500 MSps.

2.2 Vstupní parametry

Osciloskop EMBO bude implementován jako firmware zařízení v mikrořadiči. Pomocí komunikačního rozhraní s ním bude spojena PC aplikace, která bude sloužit jako terminál.

Zařízením je myšlena hardwarová část, která bude zpracovávat vstupní signály, generovat výstupní signály a komunikovat s počítačem. Hlavní důraz je kladen na cenu a dostupnost. V dnešním světe dominují procesory ARM, jak nízkou cenou, tak vysokým výkonem, promyšlenou instrukční sadou a především softwarovou podporou. Z těchto důvodů byly zvoleny mikrořadiče s jádrem ARM Cortex-M od firmy STM32.

Jako primární čip byl vybrán STM32F103 a vývojová deska BluePill. Čip STM32F103 je jeden z prvních od švýcarské firmy ST. Je velmi rozšířený a přístupný. Často se můžeme setkat s jeho čínskými kopiemi. Jeho parametry stačí na postavení základního osciloskopu.

Kromě série F1 bude osciloskop EMBO fungovat na dalších čipech z rodiny STM32. Nesmí chybět podpora řadičů F303, které jsou považovány za analogový high-end od ST. Podporou je myšleno to, že pro každou řadu bude vytvořen konkrétní soubor s firmwarem zařízení. Dále se nabízí řadiče rodin F4, G0, L0 a L4. Větší spektrum podporovaných mikrořadičů klade nároky na promyšlený a univerzální návrh firmware.

Veškerá hlavní funkcionalita bude implementována ve firmware zařízení. Bude se využívat jen vnitřních bloků mikrořadiče, jako je ADC, DAC, DMA nebo časovač. Osciloskop bude tak dobrý, jak mu umožní parametry konkrétního čipu. Mezi kritické parametry patří vzorkovací frekvence ADC, velikost paměti a přítomnost DMA. Další omezující parametry jsou rozlišení ADC, přítomnost DAC, taktovací frekvence procesoru nebo implementace USB řadiče, který může měřený signál rušit.

Osciloskopy lze dělit na 3 typy (viz. obrázek 2.1):

- Analogový osciloskop — měřící přístroj postaven na analogovém zpracování
- Digitální osciloskop — moderní měřící přístroj postaven na FPGA nebo ASIC
- Softwarově definovaný osciloskop — měřící funkce realizovány programem v MCU

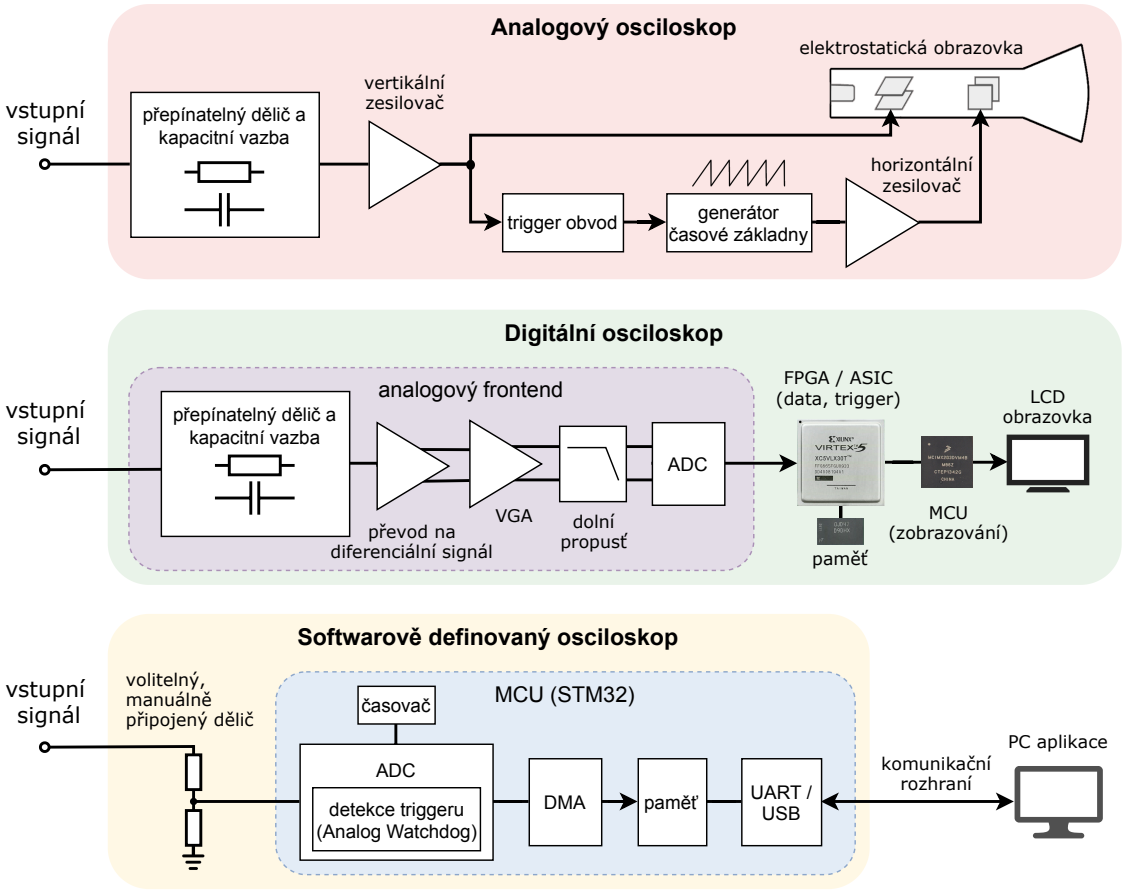
U analogového osciloskopu je vstupní signál přiveden na přepínatelný vstupní dělič a případně na kapacitu. Dále signál pokračuje na vertikální zesilovač, který budí vertikální vychylovací systém obrazovky. Horizontální zesilovač je buzen buď jiným vstupem (XY režim) nebo interní časovou základnou. Ta je tvořena pilovým signálem z integrátoru. Spouštění časové základny je řízeno trigger obvodem. Jeho správné nastavení docílí synchronizaci periodického signálu na obrazovce.

U digitálního osciloskopu je nejdůležitější částí analogový frontend, který určuje jeho základní parametry. Opět je signál přiveden na přepínatelný dělič a vazební kondenzátor. Dále je převeden zesilovačem na signál diferenční a následuje VGA (digitálně ovládaný zesilovač s variabilním zesílením), který je analogií vertikálního zesilovače. Uvnitř VGA bývá také často filtr dolní propusť. Výstupní diferenční signál s úrovní v jednotkách Volt vede přímo do rychlého AD převodníku (paralelní nebo řetězové). Digitalizovaný signál se dále zpracovává buď v obvodech FPGA (Rigol DS2000) nebo ASIC (Keysight Megazoom).[8]

V obou zmíněných případech se signál zpracovává hardwarově, ať už analogovými nebo digitálními FPGA obvody. Softwarově definovaný osciloskop tuto možnost nemá. Je u něj nutné

zařídit funkčnost pomocí programu v mikrořadiči. Vstupní signál samozřejmě je zpracováván pomocí ADC, jsou použity hardwarové časovače, ale vyhodnocení triggeru, konfigurace periferií a komunikace se musí realizovat softwarově. Mikrořadič oproti FPGA má omezenou operační paměť a je pomalejší. Vývojové desky STM32 logicky nemají přepínatelný vstupní dělič, signál pro ADC tak jde přímo na periferii. Z toho všeho plynou omezení pro osciloskop EMBO, kterými se bude lišit od standardních měřících přístrojů.

Největším omezením je pomalá rychlost vzorkování mikrořadiče STM32F103. Dále bude problém komunikace, která v případě použití UART bude úzké hrdlo celého systému. Volitelně použité USB bude dosahovat mnohem větší rychlosti než UART. Rychlost UART komunikace bude omezena na nejrychlejší standard sériového rozhraní 115200 bps (11.52 KB/s). Signál se nejprve zachytí a uloží do paměti. Následuje pomalý přenos do PC. Během této doby se čeká, aby nedošlo k přepsání paměti. Po přenosu se opět zapne ADC. Aby tedy správně fungoval trigger osciloskopu, musí být implementován přímo v zařízení.



obr 2.1: Základní princip funkce analogového, digitálního a softwarově definovaného osciloskopu

Kapitola 3

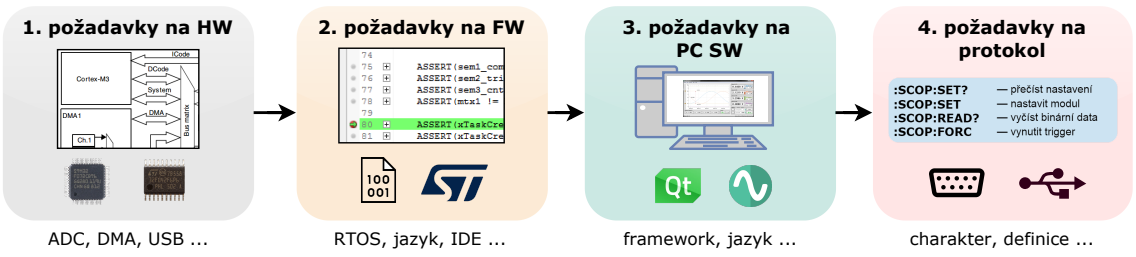
Plán realizace a analýza požadavků

Naplánovat správně projekt před začátkem je důležité, pokud chceme, aby výsledné dílo bylo kvalitní. Software šitý horkou jehlou často není stabilní a bývá špatně rozšiřitelný. Základním požadavkem projektu EMBO je stabilita, proto bude během práce kladen důraz na přehlednou vývojovou dokumentaci a standardní postupy softwarového inženýrství.

3.1 Systémové požadavky

V této kapitole se definují hlavní systémové požadavky. Z nich vyplynou problémy, které mohou při vývoji nastat. Obecné požadavky již byly nastíněny, tato kapitola je proto rozvede a rozdělí na bloky, kterých se požadavky týkají (viz. obrázek 3.1):

1. Požadavky na mikrořadič a jeho periferie
2. Požadavky na vývoj firmware zařízení
3. Požadavky na vývoj PC aplikace
4. Požadavky na komunikaci mezi osciloskopem a PC aplikací



obr 3.1: Přehled hlavních bodů analýzy požadavků pro vývoj osciloskopu EMBO

3.1.1 Požadavky na mikrořadič a jeho periferie

Jako hlavní platforma hardware byla zvolena rodina mikrořadičů STM32 s jádry ARM Cortex-M. Architektura ARM dnes představuje nejrozšířenější instrukční sadu na světě. Kromě instrukční sady také její tvůrci licencují procesorová jádra (IP) včetně grafického koprocesoru (Mali) nebo SIMD koprocesoru (NEON). Čipy ARM nalezneme ve spotřební elektronice, v serverech, v počítačích a ve většině mobilních telefonů. Nejmodernější architektura ARM Cortex lze rozdělit na 3 základní kategorie: [9]

- Cortex-A — Výkonné vícejádrové procesory (telefony, počítače)
- Cortex-R — Výkonné procesory pro aplikace reálného času (roboti, průmysl)
- Cortex-M — Univerzální procesory (embedded a nízkoenergetické zařízení)

Dlouhou dobu platilo, že čipy ARM, v porovnání s x86/64 instrukční sadou, výkonově zaostávaly. Architektura x86 desítky let dominovala na PC a serverech, tedy všude tam, kde je potřeba vysoký výkon. Vysokému výkonu vděčí x86 kombinaci použitých technik. Jsou to superskalární vícejádrové procesory s vícestupňovou pipeline. Mají více úrovní paměti cache a především vykonávají instrukce spekulativně pomocí jednotek ROB (reorder buffer). Počet ROB je ale omezen množstvím dekodérů instrukcí.

Jelikož má architektura x86 variabilní délku instrukcí, je počet dekodérů omezen zhruba na 4 na jádro. Kvůli zachování 30-ti let zpětné kompatibility nelze zvyšovat počet ROB. V roce 2020 tak vytvořila firma Apple vlastní procesor (s instrukční sadou ARM), který obsahuje 8 jednotek ROB. S poloviční taktovací frekvencí a s mnohem nižším příkonem poráží dnešní procesory AMD a Intel v jedno-vláknovém výkonu. Tedy už neplatí to, že ARM zaostává za x86.

Pro projekt EMBO jsou ideální mikrořadiče s architekturou Cortex-M. Švýcarská firma ST nabízí celou řadu čipů s jádry Cortex-M. Projekt bude cílit především na tyto řady STM32:

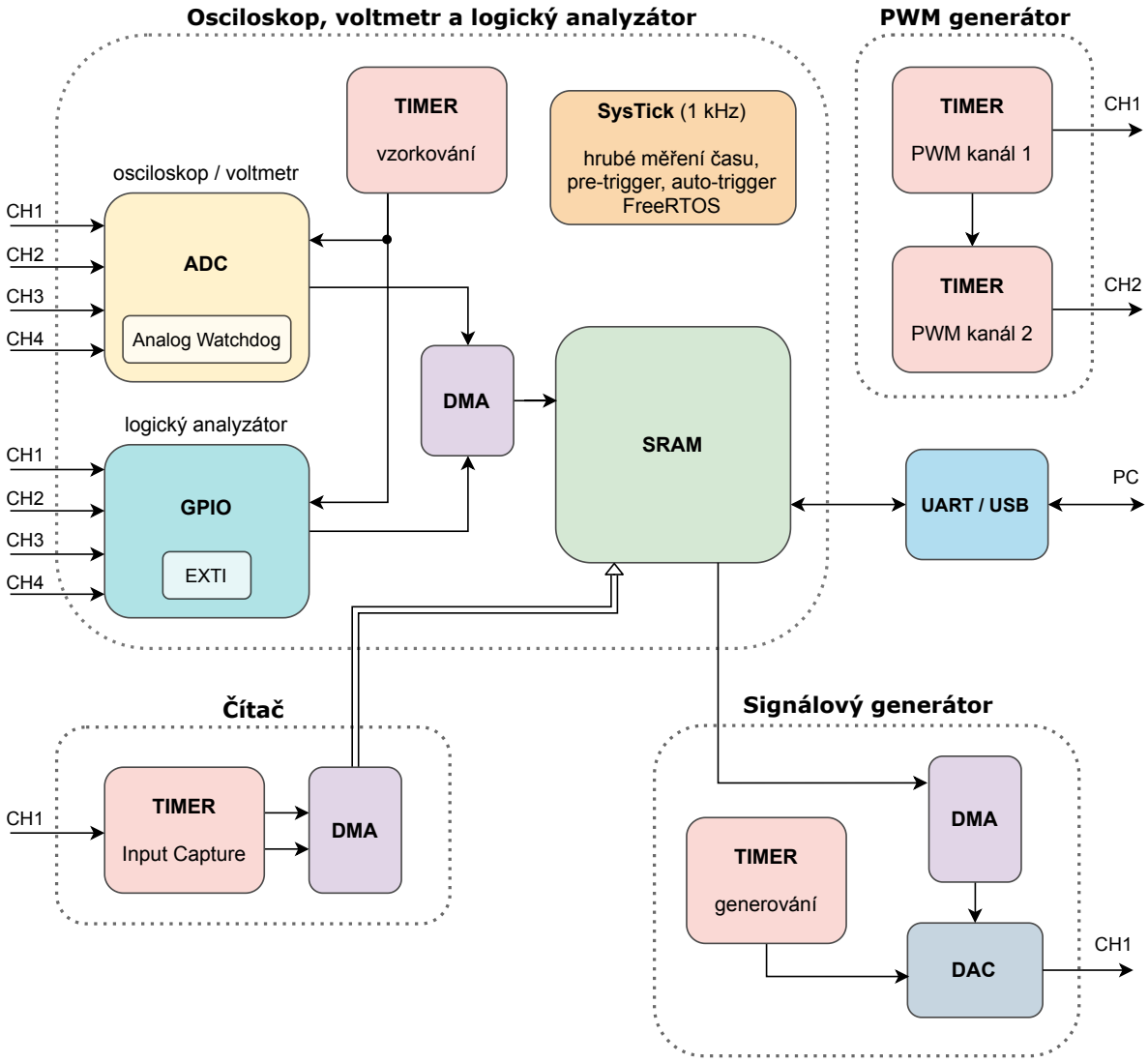
- F0, L0, G0 — Cortex-M0(+) (nízkoenergetické, Von Neumannova architektura)
- F1, F2, L1 — Cortex-M3 (levné, hardwarová dělička, Harvardská architektura)
- F3, F4, G4, L4 — Cortex-M4F (výkonné, FPU, Harvardská architektura)

Mikrořadiče se liší jak cenou tak parametry. Nyní je třeba promyslet, jaké jsou požadované parametry a jaké potenciální problémy mohou nastat.

Jelikož se bude programovat v jazyce C, nebude hrát roli, jestli je čip založen na Von Neumannově nebo Harvardské architektuře. Co však bude hrát významnou roli je frekvence jádra, velikost programové paměti a velikost operační paměti. Taktovací frekvence vybraných čipů začíná na 32 MHz. To je pro základní funkci osciloskopu dostačující. Velikost programové paměti dnes také nebývá problém. Klíčový parametr je tedy velikost operační paměti, který

se u vybraných čipů pohybuje mezi 6-192 KB. Tato hodnota přímo určí velikost paměti osciloskopu, tedy množství vzorků, které lze zachytit.

Kromě CPU a pamětí budou potřeba tyto periferie a vnitřní bloky mikrořadiče: GPIO, DMA, NVIC, ADC, DAC, UART, USB a časovače (viz. obrázek 3.2).



obr 3.2: Blokové schéma osciloskopu EMBO se zaměřením na periferie MCU

Analýza potřebných periferií pro softwarově definovaný měřící přístroj

GPIO.

Vstupní a výstupní logické piny spravuje periferie GPIO. Výstupní režim GPIO lze volit mezi Push-Pull (dva tranzistory, kdy na výstupu je buď log. 1 nebo log. 0) nebo Open-Drain (jeden tranzistor v zapojení s otevřeným kolektorem). Kromě ovládání signalizační LED diody nebude výstupní režim GPIO nutný. [10]

Naopak vstupní režim GPIO bude třeba použít pro logický analyzátor. Napětí přivedené na GPIO by nemělo přesáhnout napájecí napětí čipu. Vstupní signál GPIO je zpracován synchronně, převeden na logickou hodnotu a uložen do vstupního registru. Děje se tak každý cyklus periferních hodin. Z periferie lze číst přímo v hlavním programu, to se ale pro rychlé nebo náhodné signály nepoužívá. Mnohem častější je číst v přerušení nebo použít DMA. Oba dva principy budou použity v logickém analyzátoru, první pro detekci triggeru a druhý uložení dat.

DMA.

Pro přístup k paměti nezávisle na procesorovém jádře se používá řadič DMA. Může pracovat s operační pamětí nebo s pamětí periferie. Každý řadič DMA obsahuje několik kanálů, které můžou zapisovat nebo číst z různých míst. Operace DMA může být odstartována časovačem a délka provedení je většinou několik cyklů periferních hodin. DMA nejčastěji pracuje v režimu cirkulárního bufferu. Nejprve se nastaví adresa čtení, adresa zápisu a množství dat. Po zapnutí řadiče jsou prováděny DMA transakce a po naplnění bufferu se pokračuje od začátku. [11]

Výhodou DMA je vysoká rychlost operací, která je omezena pouze délkou transakce a šířkou pásma sběrnice (paralelní DMA přenosy již naráží na limity). DMA bude nutné použít pro osciloskop, logický analyzátor, voltmetr, čítač a signálový generátor. Z toho lze usoudit, že DMA je stěžejní prvek systému. V mikrořadiči ATmega z Arduina není DMA, proto by podobný projekt na Arduinu měl řádově horší parametry.

NVIC.

Řadič vnořených přerušení v architektuře ARM se jmenuje NVIC. Přerušení je základní princip obsluhy asynchronní události. Přerušení může být vyvoláno náběžnou nebo sestupnou hranou, dále také interní událostí v mikrořadiči. Přerušit program a začít vykonávat jiný (kód obslužné rutiny) trvá na jádrech Cortex-M zhruba 12-16 cyklů. [12] Během této doby se uloží programový čítač a stavový registr na zásobník a provedou další nezbytné akce. Po ukončení obslužné rutiny procesor obnoví registry ze zásobníku a pokračuje tam, kde byl přerušen.

Přerušení bude potřeba pro hlavní systém (hrubé čítání času), čítač (softwarové zvětšení bitů čítače), logický analyzátor (trigger) a UART/USB komunikaci. Latence přerušení bude nejkritičtější u logického analyzátoru, kde může způsobit nestabilitu triggeru u vysokých hodnot vzorkovací frekvence.

DAC.

Analogový signál se bude generovat pomocí periferie DAC. Toho se využije u přístroje signálový generátor. DAC je dostupný pouze u některých řad STM32 mikrořadičů. U čipů bez DAC převodníku se pro stejnou funkci standardně používá PWM výstup, RC filtr a operační zesilovač jako napěťový sledovač. Jelikož filosofií projektu EMBO je nepoužít žádné externí komponenty, bude funkce signálového generátoru volitelná. DAC je nutné použít s DMA. Jelikož je požadován paralelní běh osciloskopu a signálového generátoru, může být u některých čipů s jedním DMA řadičem problém. [13]

Časovač.

Pro měření času, generování systémových událostí a generování PWM se použije periferie časovač (TIM). Jádro časovače tvoří Prescaler (předdělička), čítač a Auto-Reload registr (strop). Se znalostí frekvence taktovacího signálu umožňuje časovač programátorovi pracovat v časové doméně. Časovače disponují digitálním vstupem (režim Input Capture) a digitálním výstupem (režim Output Compare). Tyto režimy mohou sloužit k měření periody vstupního signálu nebo ke generování PWM signálu. V mikrořadičích STM32 jsou k dipozici pokročilejší funkce, jako třeba zřetězení, vstupní filtrace nebo generování události, kterou lze distribuovat do dalších bloků kontroléru. Většina časovačů v čipech STM32 má rozlišení 16 bitů. [14]

Pro EMBO bude potřeba minimálně 5 časovačů: pro generování vzorkovací frekvence, měření pretriggeru, generování 2 synchronních PWM signálů a pro implementaci čítače. Ve všech jádrech ARM je také k dispozici 1 kHz časovač (Systick), který lze použít pro hrubé měření času nebo pro časovou základnu RTOS. [15]

UART a USB.

Ke komunikaci s PC aplikací se použije buď UART nebo nativní USB. UART je přítomný ve všech STM32 čipech, USB jen v některých. V obou případech jde o přenos typu Full-Duplex, tedy zvlášť kanál pro příjem a zvlášť kanál pro odesílání. Aby se návrh co nejvíce zjednodušil, bude použit režim USB CDC a UART bude omezen na nejvyšší standardní rychlost 115200 bps. UART bude mít před USB prioritu. Na straně PC aplikace se bude používat virtuální sériový port. Tímto se docílí maximální kompatibilita mezi řadami čipů STM32 a operačními systémy.

Analýza AD převodníku s ohledem na návrh osciloskopu

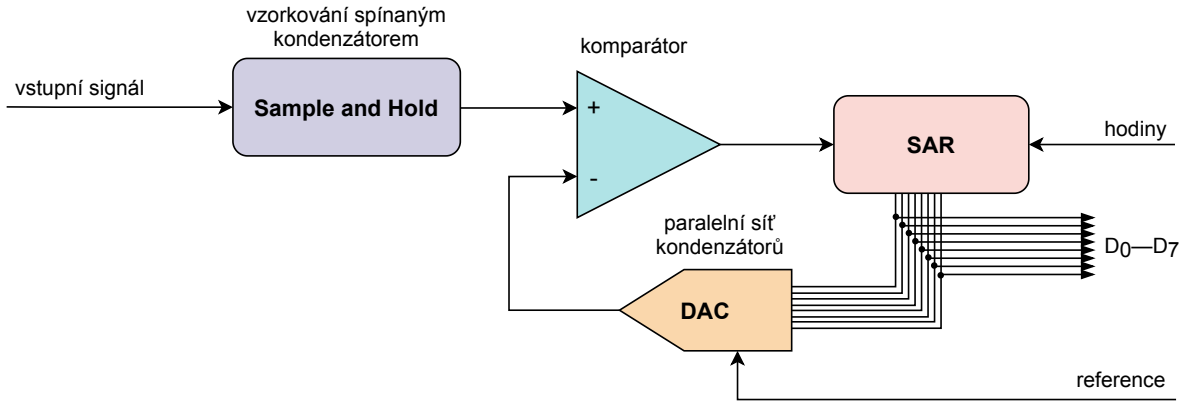
Převodník ADC je nejdůležitější část každého digitálního osciloskopu. AD převod se skládá ze tří hlavních kroků. Vzorkování, kvantování a kódování. Vzorkování je proces určení bodů v časové oblasti. Kvantování je diskretizace, kdy dojde k převodu spojité informace na diskrétní, v pevně stanovených kvantech. Kódování je převod této diskrétní informace do binární podoby.

SAR převodník.

Převodník s postupnou aproximací (SAR) je dnes nejpoužívanější typ ADC. Nejvyšší rozlišení zhruba 18 bitů a vzorkovací frekvence do 10 MSps jsou ideální kompromis. Výhodou

je nízká cena, jednoduchost, linearita a nízký šum. Funguje na principu půlení intervalů, známého z algoritmu binárního vyhledávání. Je majoritně zastoupený v běžně dostupných mikrokontrolérech. V osciloskopu EMBO bude u všech čipu STM32 přítomen pouze převodník SAR.

Základní funkční bloky SAR převodníku jsou komparátor, aproximační registr a DA převodník. Konverze postupuje od MSB k LSB, pro 12 bitový režim je potřeba 12 cyklů. Na počátku konverze se nastaví v aproximačním registru nejvyšší bit na 1 a všechny ostatní na 0. V DA převodníku se vygeneruje napětí odpovídající této hodnotě a měřený signál se následně komparuje vůči tomuto napětí. Pokud je hodnota vyšší, bit zůstává v 1, pokud je menší, změní se na 0. Takto se pokračuje až jsou rozhodnuty všechny bity (viz. obrázek 3.3). [16]



obr 3.3: Blokové schéma AD převodníku s postupnou aproximací (SAR)

Vzorkování.

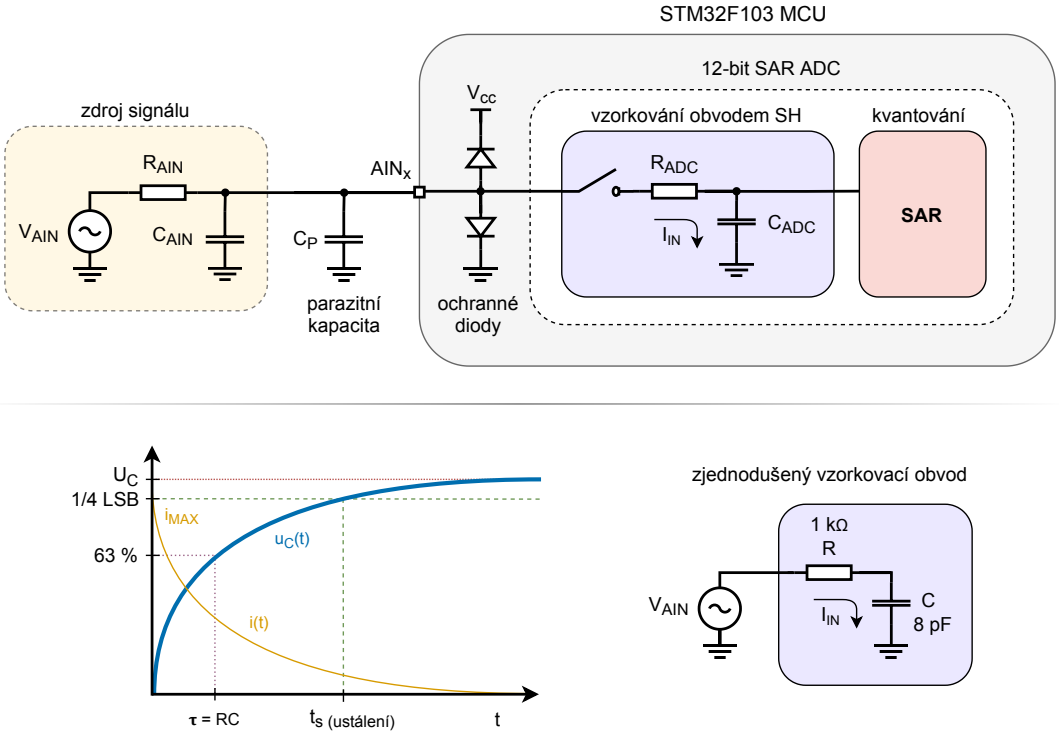
Vzorkování je definováno vzorkovací frekvencí a lze chápat jako pravidelné spínání spojitého signálu spínačem. Základním pravidlem pro určení této frekvence je Nyquistův-Shannonův teorém, který tvrdí, že vzorkovací frekvence musí být alespoň dvakrát vyšší, než je hodnota nejvyšší spektrální složky signálu, kterou chceme měřit. Jinými slovy musíme vzorkovat alespoň dvakrát tak rychleji než je frekvence signálu, abychom ze vzorků rekonstruovali signál o stejné frekvenci.

V praxi je lepší vzorkovat alespoň 5 - 10 krát rychleji, abychom po rekonstrukci získali zpět také správný tvar. Pokud nesplníme Nyquistův teorém, vznikne alias, který se projeví jako falešný signál (s frekvencí rozdílu vzorkovací a reálné frekvence). [17] Toho lze využít, pokud s jistotou víme, že pozorujeme alias. Můžeme tak analyzovat signál o frekvenci vyšší než je Nyquistova. Uvedené platí pouze pro jednoduché projekty tohoto typu, každý kvalitní osciloskop má před AD převodníkem analogový filtr dolní propusť, který by měl z větší části aliasingu zabránit.

Sample and Hold.

Vzorkování je technicky realizováno nejčastěji jako obvod Sample and Hold. Ten je tvořen CMOS spínačem a kondenzátorem o hodnotě zhruba 10 pF. Spínač sepne, kondenzátor se

nabije, pak se obvod rozepne a kondenzátor uchová náboj úměrný vstupnímu napětí. Převodník pokračuje dalšími kroky jako je kvantování a kódování. Existují dvě varianty tohoto obvodu, s bufferem na vstupu a bez něj. Nejpoužívanější je verze bez bufferu, kdy vstupní signál jde po sepnutí přímo na kondenzátor (viz. obrázek 3.4). Druhá varianta s tranzistorem na vstupu je dražší, protože tranzistor musí být dostatečně lineární a nízkošumový. Výhody má ve fixní vstupní impedanci a ve zrychlení přechodových dějů, spínáním kondenzátoru nízkou impedancí.



obr 3.4: Schéma měření signálu pomocí SAR ADC a doba ustálení napětí na vzorkovacím kondenzátoru

Vzorkování s přímo připojeným kondenzátorem je použito u všech STM32 čipů, které budou použity v tomto projektu. Je to levnější a jednodušší, má to ale několik nevýhod. Vstupní parametry závisí na frekvenci a malá část signálu se vrací spolu s vyššími harmonickými zpět do měřeného obvodu. Je důležité si neplést dobu vzorkování a vzorkovací frekvenci.

- Vzorkovací frekvence — jak často sepne spínač signál na vzorkovací kondenzátor
- Doba vzorkování — jak dlouho je spínač sepnutý
- Vstupní impedance — vstupní impedance kanálu osciloskopu
- Maximální přípustná impedance zdroje signálu — závisí na výše zmíněných parametrech

Maximální přípustná impedance zdroje signálu.
Maximální impedance zdroje signálu je důležitý parametr, který může znatelně ovlivnit měření. Měkký zdroj napěťového signálu je u SAR převodníků možné konvertovat špatně, stačí vzorkovat příliš rychle. Proud tekoucí do vzorkovacího kondenzátoru je tak příliš slabý a kondenzátor se nestačí nabít na správnou úroveň. Výsledkem je nižší digitální hodnota, než je reálné analogové napětí. U standardních digitálních osciloskopů se nepoužívají SAR ale paralelní nebo řetězové převodníky, kde se tento problém neřeší.

Obvod Sample and Hold je z principu RC článek, který je definován časovou konstantou τ (3.1). Doba ustálení přechodového jevu na vzorkovacím kondenzátoru je určena hranicí napětí 1/4 LSB. Uvažujme zjednodušený obvod z obrázku 3.4 a zdroj signálu s nekonečně malým vnitřním odporem. V tomto případě platí obecný vzorec pro převodník s N bity (3.2) za předpokladu jednotkového skoku o velikosti referenčního napětí. Pro 12-bitový AD převodník v čipu STM32F103 byla spočtena doba ustálení v ideálním případě na 77.6 ns. [18]

$$\tau = RC \qquad u_C(t) = U_0 \cdot e^{-\frac{t}{RC}} \qquad i(t) = \frac{U_0}{R} \cdot e^{-\frac{t}{RC}} \tag{3.1}$$

$$t_s = -ln\left(\frac{1}{2^N \cdot 4}\right) \cdot \tau \tag{3.2}$$

$$t_{F103} = -ln\left(\frac{1}{4096 \cdot 4}\right) \cdot RC = 9.7 \cdot (1\,k\Omega \cdot 8\,pF) = 77.6\,ns \tag{3.3}$$

V praxi bude ale doba ustálení vždy vyšší. Vnitřní odpor zdroje signálu, parazitní kapacita a odpor vodičů způsobí, že se vzorkovací kondenzátor trvá nabít delší dobu. Proto je důležitá maximální přípustná impedance zdroje signálu, která když se překročí, kondenzátor se nestihne dostatečně nabít. Její hodnota (R_{AIN}) lze vypočítat z časové konstanty SH obvodu (R_{ADC}, C_{ADC}), taktovací frekvence ADC (f_{ADC}), bitového rozlišení (N) a doby vzorkování, vyjádřené v cyklech taktovacích hodin (k). Pro výpočet slouží vzorec (3.4), který počítá s chybou pod 1/4 LSB. V tabulce 3.1 jsou parametry vybraných mikrořadičů STM32, které jsou vstupem pro výpočet.

Parametry STM32 ADC						
MCU	ADC*	max f_s	N (rozlišení)	f_{ADC}	C_{ADC}	R_{ADC}
F103	1	1 Msps	12 bit	12 MHz	8 pF	1 kΩ
F303	4	5 Msps	8 - 12 bit	72 MHz	5 pF	1 kΩ
L412	2	5 Msps	8 - 12 bit	80 MHz	5 pF	1 kΩ

Tabulka 3.1: Parametry AD převodníků v mikrořadičích STM32, vztažené k osciloskopu EMBO (jsou brány v úvahu pouze ADC* s DMA, ostatní vyžadují speciální režim) [2] [3] [4]

V tabulce 3.2 lze vidět nastavitelné hodnoty doby vzorkování pro zmíněné čipy STM32.

$$R_{AIN} \leq \frac{k - 0.5}{f_{ADC} \cdot C_{ADC} \cdot ln\,(2^{N+2})} - R_{ADC} \tag{3.4}$$

Doba vzorkování STM32 ADC dle katalogu	
MCU	k (počet cyklů f_{ADC})
F103	1.5, 7.5, 13.5, 28.5, 41.5, 55.5, 71.5, 239.5
F303	1.5, 2.5, 4.5, 7.5, 19.5, 61.5, 181.5, 601.5
L412	2.5, 6.5, 12.5, 24.5, 47.5, 92.5, 247.5, 640.5

Tabulka 3.2: Nastavitelná doba vzorkování ADC v mikrořadičích STM32 dle katalogu [5] [6] [7]

Pro AD převodník v řadě čipů STM32F103 vychází maximální přípustná impedance zdroje 73.4 Ω pro nejrychlejší vzorkování (600 - 800 kSps) a rozlišení 12 bit. (3.5)

$$STM32F103 \quad ... \quad R_{AINmax} = \frac{1.5 - 0.5}{12\ MHz \cdot 8\ pF \cdot ln\,(2^{12+2})} - 1\,k\Omega = 73.4\,\Omega \tag{3.5}$$

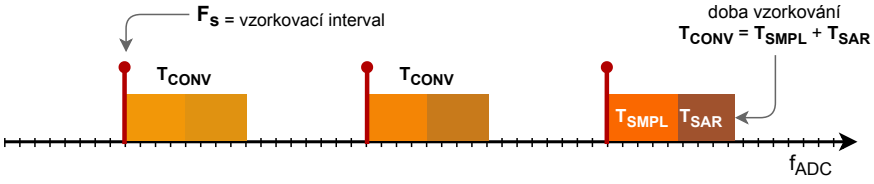
Celková délka AD převodu (T_{CONV}) je tvořena součtem doby vzorkování (T_{SMPL}) a délky konverze (T_{SAR}). Délka konverze vyplývá z podstaty fungování SAR převodníku. Z těchto hodnot lze jednoduše získat přípustný rozsah vzorkovacích frekvencí vzorcem (3.7) a (3.8).

$$T_{CONV} = \frac{T_{SMPL} + T_{SAR}}{f_{ADC}} \tag{3.6}$$

$$f_s(12bit) \leq \frac{f_{ADC}}{k + 12.5} \tag{3.7}$$

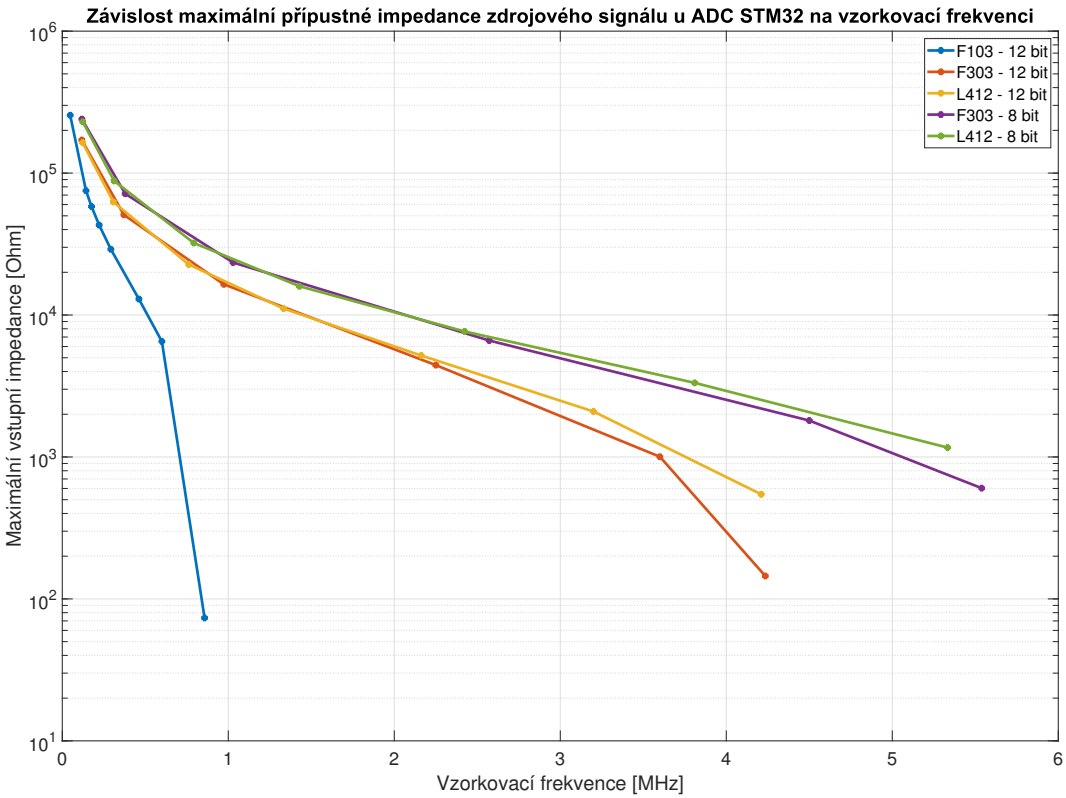
$$f_s(8bit) \leq \frac{f_{ADC}}{k + 8.5} \tag{3.8}$$

Na obrázku 3.6 je zobrazeno porovnání maximální vstupní přípustné impedance zdrojového signálu pro mikrořadiče F103, F303 a L412 v závislosti na vzorkovací frekvenci, resp. době vzorkování, protože čím rychleji se vzorkuje, tím kratší musí být doba vzorkování. Pro pomalé vzorkovaní je tedy vhodné zvolit nejdelší dobu vzorkování. Graficky je doba vzorkování znázorněna na obrázku 3.5. [19]



obr 3.5: Graficky znázorněný rozdíl mezi dobou vzorkování a vzorkovací frekvencí

Doba vzorkování se u AD převodníků řady STM32 nastavuje diskrétně v rozmezí několika hodnot. Maximální přípustná impedance zdrojového signálu je tedy statický parametr, který se mění skokově. Důležité jsou i dynamické parametry. Parazitní a vstupní kapacita (C_p a C_{AIN}) spolu se vstupním odporem (R_{AIN}) ovlivňují velikost přeneseného střídavého napětí.

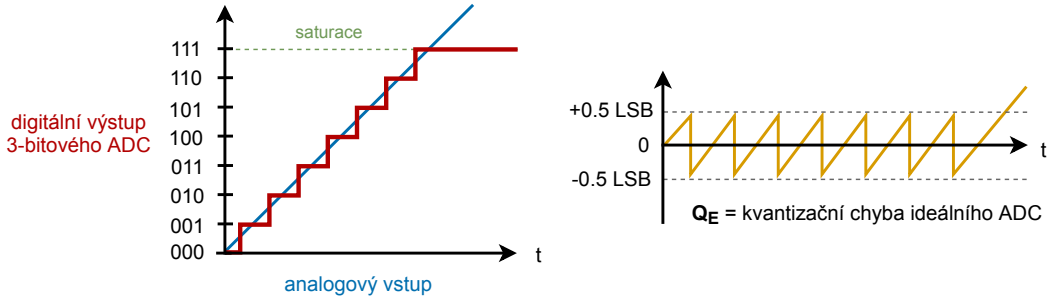


obr 3.6: Porovnání maximální přípustné impedance zdrojového signálu u ADC STM32 pro různou dobu vzorkování, resp. vzorkovací frekvenci

(3.9) Z toho plyne, že při měření je nutné brát ohled nejen na tvrdost zdroje vstupního signálu, ale i na jeho frekvenci (f_{AIN}), a to pro různé nastavení AD převodníku.

$$f_{AIN} \leq \frac{1}{10 \cdot R_{AIN} \cdot (C_{AIN} + C_p)} \tag{3.9}$$

Kvantování.
Po vzorkování následuje kvantování (viz. obrázek 3.7). Je to ztrátový proces převodu spojité informace na diskrétní. Hlavním parametrem kvantování je bitové rozlišení AD převodníku. To je určeno nejmenším možným krokem (kvantem), často uváděným jako LSB (nejméně významný bit). Ten představuje nejmenší rozlišitelnou úroveň převodu, v případě 12-bitového SAR ADC a referenčního napětí 3.3 V to je zhruba 0.8 mV. (3.10). V měřící praxi bývají kvantizační úrovně rozděleny lineárně s rozhodovacími hranicemi v poloviční vzdálenosti. V případě audio signálů, telekomunikací a televizního vysílání se používá nelineární rozdělení, kde u nižších úrovní jsou kvantizační úrovně dál od sebe.



obr 3.7: Graficky znázorněný proces kvantování při AD převodu

Zmíněná ztrátovost procesu pramení z kvantizační chyby (3.11), která periodicky osciluje v průběhu procesu kvantování okolo nuly v rozsahu ± 0.5 LSB (u ideálního ADC). Kvantizační šum je časová posloupnost kvantizačních chyb. Lze pozorovat třeba na 8-bitových GIF obrázcích, kde se části s podobnou barvou slévají do jednolitých čtverců.

$$STM32F103 \quad ... \quad V_{LSB} = \frac{V_{REF}}{2^N - 1} \quad \longrightarrow \quad \frac{3.3\,V}{2^{12} - 1} = \frac{3300\,mV}{4095} \approx 0.8\ mV \tag{3.10}$$

$$STM32F103 \quad ... \quad Q_E = \pm 0.5\ LSB = \pm\frac{V_{LSB}}{2} \quad \longrightarrow \quad \pm\frac{0.8\,V}{2} \approx \pm 0.4\ mV \tag{3.11}$$

Další základní parametry AD převodníků jsou DR (dynamický rozsah) (3.14) a SNR (odstup signálu od šumu) (3.15). Jinými slovy poměr největšího a nejmenšího vstupního signálu (DR) a poměr největšího měřeného signálu vůči velikosti šumu (SNR). Šumem není myšlen stejnosměrný ofset a vyšší harmonické složky. V ideálním případě by se DR rovnal SNR (3.13). V reálném světě ale interní šum ADC, šum vstupního signálu a nestabilita hodinového taktu (Jitter) způsobí, že SNR je nižší než DR. Toto omezení dynamického rozsahu je v celém měřícím řetězci nejvíce patrné na analogovém a digitálním rozhraní. [20]

$$DR = SNR_{ideal} \tag{3.12}$$

$$DR \geq SNR_{real} \tag{3.13}$$

$$DR = 20log_{10}\left(\frac{(2^N - 1) \cdot LSB}{LSB}\right) = (6.02 \cdot N) + 1.76 \qquad (dB) \tag{3.14}$$

$$SNR = 20log_{10}\left(\frac{V_{RMS}signal}{V_{RMS}noise}\right) \qquad (dB) \tag{3.15}$$

Pro měření nás zajímá hodnota SNR, z které lze vypočítat ENOB (efektivní počet bitů) (3.16). ENOB je klíčový parametr AD převodníku, který nám ukazuje reálné bitové rozlišení. To je vždy nižší než deklarované celočíselné bitové rozlišení.

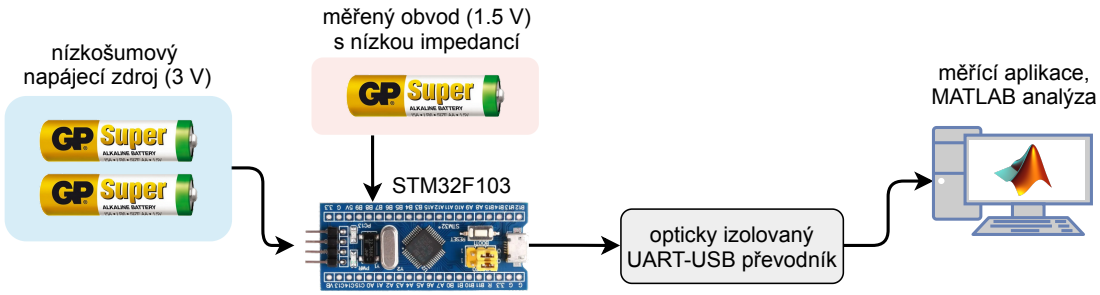
$$ENOB = \frac{SNR - 1.76}{6.02} \qquad (bit) \tag{3.16}$$

Vylepšit SNR a tak i ENOB lze metodu Oversampling (vzorkováním vyšší frekvencí, než je nutné dle Nyquista) nebo zvýšením bitového rozlišení. Pro Oversampling ratio (OSR) 4, tedy

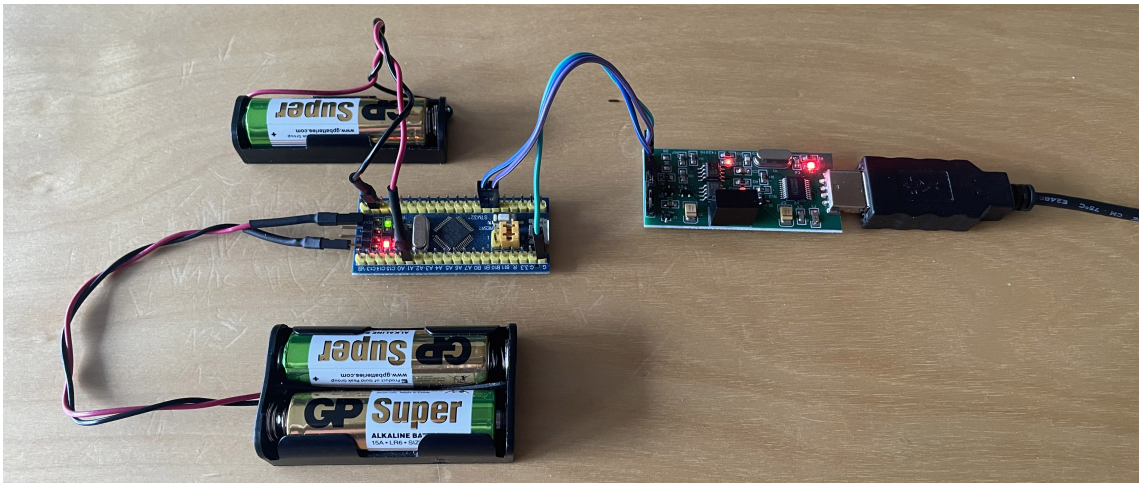
zvýšení vzorkovací frekvence 4 krát, se zvýší SNR o 6 dB. (3.17) Každé zvýšení SNR o 6 dB přidá k ENOB 1 bit. Záleží také na frekvenci a typu ADC. Pro vyšší vzorkovací frekvence roste zkreslení AD převodníku, které se může dostat až nad úroveň SNR a naopak tak zhoršit ENOB.

$$DR_{OSR} = (6.02 \cdot N) + 1.76 + 10log(OSR) \qquad (dB) \tag{3.17}$$

Změření ENOB u STM32F103.
Aby se dala vypočítat hodnota ENOB, musíme znát SNR. Pro výpočet SNR potřebujeme znát efektivní hodnotu šumu vůči efektivní hodnotě signálu. Tyto hodnoty zjistíme měřením. K mikrořadiči STM32F103 připojíme tvrdý zdroj napětí, například baterii (viz. obrázek 3.8). Napíšeme jednoduchý program pro ovládání ADC. Připojíme čip k počítači pomocí izolovaného převodníku. Do PC uložíme 5000 vzorků o vzorkovací frekvenci 600 kHz a 6 kHz. V prostředí MATLAB vypočteme směrodatnou odchylku, která se rovná efektivní hodnotě šumu. Dosazením do vzorce dopočteme SNR a následně ENOB (viz. obrázek 3.10).



obr 3.8: Blokové schéma měření ENOB u AD převodníku STM32F103



obr 3.9: Fotografie měření ENOB u AD převodníku STM32F103

$$STM32F103\ (f_s = 600kHz) \quad ... \quad \sigma = 0.7\ mV \qquad U_{RMS} = 1616\ mV \tag{3.18}$$

$$STM32F103\ (f_s = 6kHz) \quad ... \quad \sigma = 0.6\ mV \qquad U_{RMS} = 1615\ mV \tag{3.19}$$

Pro čipy STM32F103 je dynamický rozsah 12-bitového AD převodníku 74 dB. Na základě měření konstantního napětí nejkratší a nejdelší dobou vzorkování se výsledné SNR liší o zhruba 0.6 dB (67.4 dB a 68.0 dB), a výsledné ENOB se liší o 0.1 bit (10.9 a 11.0 bitů). Kvantizační chyba AD převodníku STM3F103 tedy dosáhne zhruba ± 0.8 mV (viz. obrázek 3.11). To je 2 krát více než u ideálního 12-bitového ADC. Pokud by se nepoužil izolovaný UART převodník, byl výsledek ještě 2 krát horší.

$$STM32F103 \quad ... \quad DR = (6.02 \cdot 12) + 1.76 = 74\ dB \tag{3.20}$$

$$STM32F103\ (f_s = 600kHz) \quad ... \quad SNR = 20log_{10}\left(\frac{1616}{0.7}\right) = 67.68\ dB \tag{3.21}$$

$$STM32F103\ (f_s = 600kHz) \quad ... \quad ENOB = \frac{67.68 - 1.76}{6.02} = 10.9\ bit \tag{3.22}$$

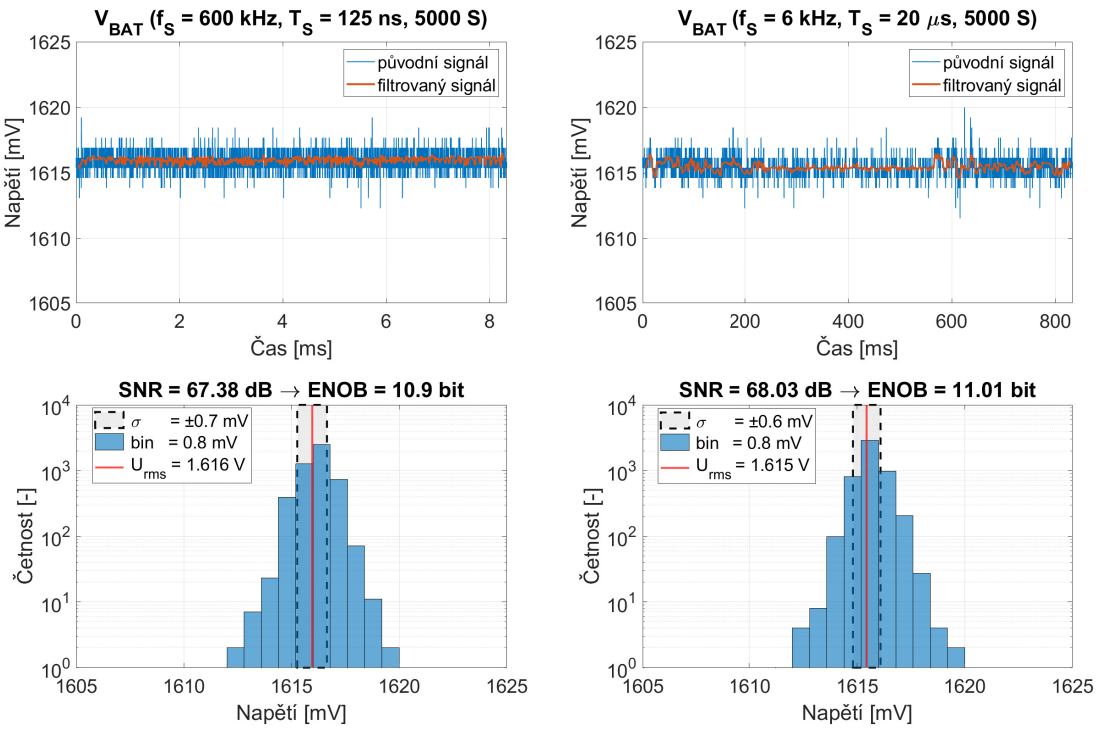
$$STM32F103\ (f_s = 6kHz) \quad ... \quad SNR = 20log_{10}\left(\frac{1615}{0.6}\right) = 68.03\ dB \tag{3.23}$$

$$STM32F103\ (f_s = 6kHz) \quad ... \quad ENOB = \frac{68.03 - 1.76}{6.02} = 11.01\ bit \tag{3.24}$$

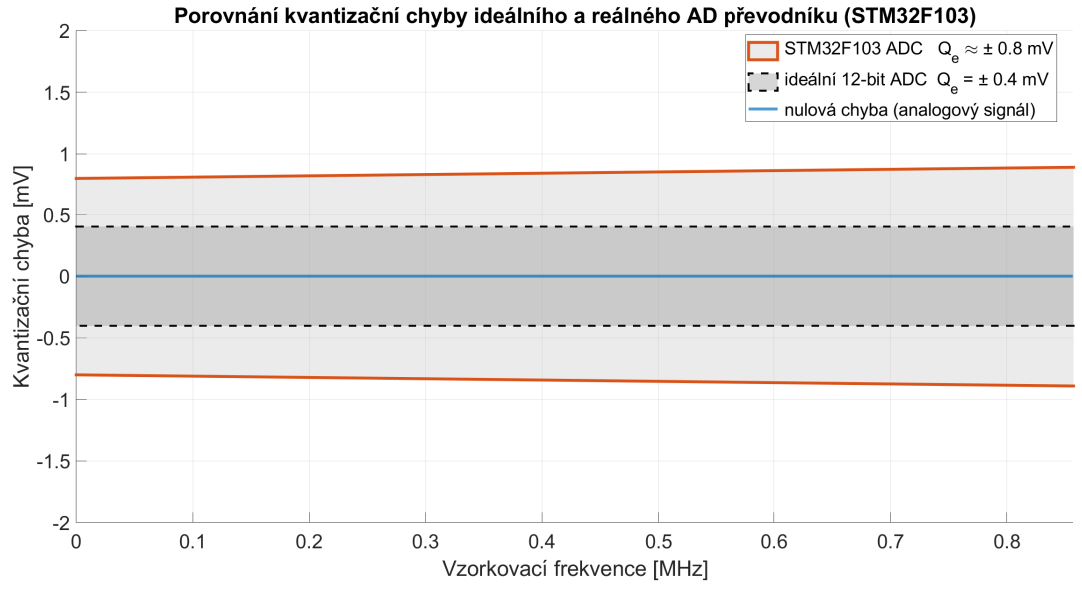
Interní chyby při měření AD převodníků.
AD převodníky vykazují chyby aditivní, multiplikativní (chyby ofsetu a citlivosti) a chyby linearity (integrální a diferenciální). Ofset a citlivost lze kompenzovat kalibrací. Chyby nelinearity úplně potlačit nelze. Kdykoliv signál projde nelineárním systémem, dojde k vytvoření vyšších harmonických frekvencí. Tyto frekvence představují nevratné zkreslení a jejich množství ve výstupu určuje hodnota THD. Můžou být také přítomny chyby v důsledku šumu na referenci nebo chyby v důsledku nesprávného zapojení převodníku do obvodu (blokovací kondenzátory, impedanční přizpůsobení). Důležitá je také stabilita hodinového signálu, který taktuje ADC (Jitter).

Konfigurace ADC v projektu EMBO.
EMBO bude podporovat různé konfigurace a parametry AD převodníků. Ze zběžné analýzy podporovaných mikrořadičů plyne, že bude nejlepší zvolit tyto konfigurace:

- 1 ADC s DMA — 4 kanály
- 2 ADC s DMA — 2 + 2 kanály
- 4 ADC s DMA — 1 + 1 + 1 + 1 kanál



obr 3.10: Porovnání SNR pro nejkratší a nejdelší dobu vzorkování u čipu STM32F103



obr 3.11: Porovnání kvantizační chyby ideálního 12-bitového AD převodníku a reálného čipu STM32F103

Každá konfigurace bude napevno určena pro konkrétní STM32 čip. Navenek se bude osciloskop tvářit buď jako 4-kanálový nebo 2-kanálový (podle dostupných pinů MCU), díky těmto konfiguracím ale nabídne uživateli vždy co nejrychlejší vzorkovací frekvenci. Volitelným parametrem bude bitové rozlišení. U všech čipu STM32 s ADC je možné použít základních 12 bitů. Často je také možné zvolit hodnotu nižší, proto u projektu EMBO bude možné volitelně přepínat mezi 12 a 8 bity, tam kde to půjde.

Kromě klasických režimů nabízí také STM32 ADC speciální režimy vzorkování. U některých čipů, které mají pro DMA přístupné pouze jedno ADC, je možné zapnout režim Dual Mode. Ten má smysl pouze pro sudý počet zapnutých kanálů. Pro dva zapnuté kanály tak zůstává plná vzorkovací rychlost. Další speciální režim se jmenuje Interleaved. Ten funguje pouze pro jeden zapnutý kanál. Vzorkují u něj všechny AD převodníky, které se postupně střídají. Pokud máme tedy 4 ADC, lze tak vzorkovat 4 krát vyšší frekvencí než je frekvence maximální.

- Dual Mode — 1 ADC s DMA + 1 ADC bez DMA (pouze 2 nebo 4 zapnuté kanály)
- Interleaved — 2 nebo 4 ADC s DMA (pouze 1 zapnutý kanál)

Napěťový rozsah analogových vstupů je u většiny mikrořadičů striktně omezen napájecím napětím. Je to kvůli ochranným diodám, které jsou na vstupu ADC. Ty nejsou stavěny na vyšší proud, proto je dobré použít před ADC sériový ochranný rezistor. Tento rezistor ale také sníží proud tekoucí do AD převodníku, sníží se tak maximální přípustná impedance zdroje signálu a tím i maximální vzorkovací frekvence. Místo sériového rezistoru lze rovnou použít napěťový dělič. Překoná se tak omezení vstupního napěťového rozsahu a zároveň se ochrání vstupy.

3.1.2 Požadavky na firmware zařízení

Firmware zařízení je program v mikrokontroléru, který se píše nejčastěji v jazycích C nebo C++, historicky v jazyce symbolických adres (assembler). Poslední dobou se začíná prosazovat i jazyk Rust. Objektově orientovaný přístup k psaní kódu je jistě správná cesta, nicméně nepatřím k lidem, kteří prosazují jazyk C++ ve všech případech. Dnes existuje mnoho C++ knihoven pro ARM mikrořadiče i mnoho lidí, kteří programují firmware v C++. Neříkám, že je to špatně, ale mezi tuto skupinu lidí nepatřím.

Jazyk C je oproti C++ mnohem předvídatelnější, jednodušší a praktičtější. V jazyce C lze také samozřejmě programovat objektově orientovaně. Nevidím tedy žádný důvod proč volit C++. Firmware je jádrem projektu EMBO. Bude v něm implementována veškerá kritická funkcionalita, proto je spolehlivost a přehlednost na prvním místě. Jazyk C++ není přehledný, a když už nepoužíváme šablony, přetěžované operátory a víceúrovňový polymorfismus, tak proč rovnou neprogramovat v C?

Po výběru jazyka přichází na řadu zvolit, jestli bude vhodné použít operační systém reálného času (RTOS). Pro nejjednodušší projekty to často není třeba, protože na zpracování několika asynchronních úloh stačí obsluha přerušení. Pro větší projekty a pokud je potřeba souběhu více úloh, tento princip už není dostačující a jde se cestou plánování. Nejjednodušší je naplánovat úlohy staticky (offline), čemuž se říká cyklická exekutiva.

Pokročilejší a univerzální způsob je použít RTOS, definovat úlohy (tasky) a jejich priority. RTOS pak sám plánuje spouštění úloh na základě plánovacího algoritmu. Synchronizace úloh probíhá pomocí semaforů a událostí. Díky komplexitě projektu EMBO a nárokům na paralelní úlohy byl zvolen operační systém FreeRTOS.

kategorie	✗ zavrženo	✓ zvoleno	důvody
• jazyk	C++	C	přehlednost, jednoduchost
• plánování úloh	žádné, cyklická exekutiva	FreeRTOS 10	lepší možnosti, známé API
• alokace paměti	dynamická (malloc)	statická	spolehlivost, standard
• IDE, psaní kódu	Keil, System Workbench ...	STM32CubeIDE	jednoduchost, vstřícnost
• debugger, ladění	integrovaný v IDE	J-Link, SEGGER Ozone	funkčnost, lepší možnosti
• ovladače periferií	žádné, ST HAL	ST LL	přenositelnost, použitelnost

obr 3.12: Přehled zvolených a zavržených technologií a metod s ohledem na vývoj firmware

Alokace paměti a její korektní využívání je specifický požadavek programování mikrořadičů. Operační paměť SRAM u mikrokontrolérů bývá často malá, proto je potřeba ji používat efektivně. Je zvykem alokovat paměť staticky, tedy již v době kompilace. To je rozdíl oproti PC, kde se často paměť alokuje dynamicky, tedy za běhu programu.

Statická alokace může být v některých případech pracnější, v automotive a aerospace je to ale jediná možná cesta, vynucená například standardem MISRA. Díky statické alokaci paměti lze provést offline analýzu kódu a odhalit potenciální problémy, na které by se jinak přišlo až za běhu programu. Lze tak například předejít přetečení zásobníku. V projektu EMBO bude dynamická alokace nepřípustná (funkce malloc) a vše se bude alokovat staticky.

Volba vývojových nástrojů je další důležitý krok. Jako vývojové prostředí (IDE) jsem zvolil oficiální aplikaci od ST jménem STM32CubeIDE. Výhodu vidím v integrování ovladačů STM32 a nástroje CubeMX, který slouží ke konfiguraci periferií a hodin. Konfigurátor CubeMX je pro mě nepostradatelný pomocník.

Přehledná a rychlá konfigurace periferií je klíčovou výhodou, protože se bude vyvíjet firmware pro více řad STM32. Kdybych měl k dispozici jen referenční manuál, prodloužila by se zbytečně celková doba vývoje. Nevýhodou STM32CubeIDE je to, že je založen na platformě Eclipse, tedy že je napsán v jazyku Java. To znamená, že běží ve virtuálním stroji (JVM), zabírá příliš mnoho paměti RAM, je pomalejší a občas i havaruje.

obr 3.13: Zvolené technologie pro vývoj firmware osciloskopu EMBO

Kromě IDE je důležitý také debugger, nástroj pro ladění programu. Nemám dobré zkušenosti s debuggerem v prostředí Eclipse, protože je pomalý a nespolehlivý. Naopak mám výtečné zkušenosti s debuggerem Ozone od firmy SEGGER, který je napsán ve C++/Qt. Ozone používá rozhraní J-Link, podporuje mnoho ARM mikrořadičů, je rychlý a spolehlivý. Firma ST s firmou SEGGER spolupracuje, takže lze vzít vývojovou desku STM32 Nucleo, přehrát firmware a udělat z ní J-Link programátor, který je pro nekomerční použití zdarma. Vývoj tedy bude probíhat tak, že na jedné PC obrazovce bude STM32CubeIDE s editorem kódu, a na druhé obrazovce bude debugger Ozone.

Programování 32-bitových ARM čipů se liší od programování 8-bitových kontrolérů. S přibývající složitostí hardware roste i složitost software. V případě ARM je nutné i pro minimální program použít abstrakční vrstvu CMSIS. Tuto knihovnu dodává výrobce. Dále už se může klasicky zapisovat a číst z registrů, nebo lze použít další abstrakční vrstvu, ovladač. U čipů STM32 je na výběr mezi dvěma typy ovladačů:

- HAL ovladače — vyšší režie, vyšší paměťové nároky, jednoduchost a přenositelnost
- LL ovladače — nulová nebo zanedbatelná režie a paměťové nároky, vyšší složitost

HAL je softwarový balík, který cílí na jednoduchost a kompatibilitu. Za cenu vyšší režie a větších paměťových nároků nabízí maximální uživatelskou vstřícnost. Je ideální pro jednoduché aplikace nebo prototypy. Naopak LL ovladače jsou navržené formou maker a inline funkcí. To znamená, že po zkompilování není rozdíl mezi aplikací psanou pomocí LL ovladače a aplikací psanou klasicky zápisem do registrů. Mnohonásobně se ale zlepší čitelnost a přenositelnost mezi řadami STM32. Osciloskop EMBO bude postaven výhradně na ovladačích LL.

V některých případech nelze použít LL ovladač, protože buď zatím neexistuje, nebo z principu ani nemůže existovat. To je zrovna případ periferie USB, která je složitá a smysl pro ni dává pouze HAL ovladač. Dále jsem rozhodl, že nebude použita mezivrstva STM32 pro FreeRTOS. Bylo by to zbytečné odklonění od standardního API, se kterým mám už zkušenosti. Navíc se tak ušetří místo v paměti.

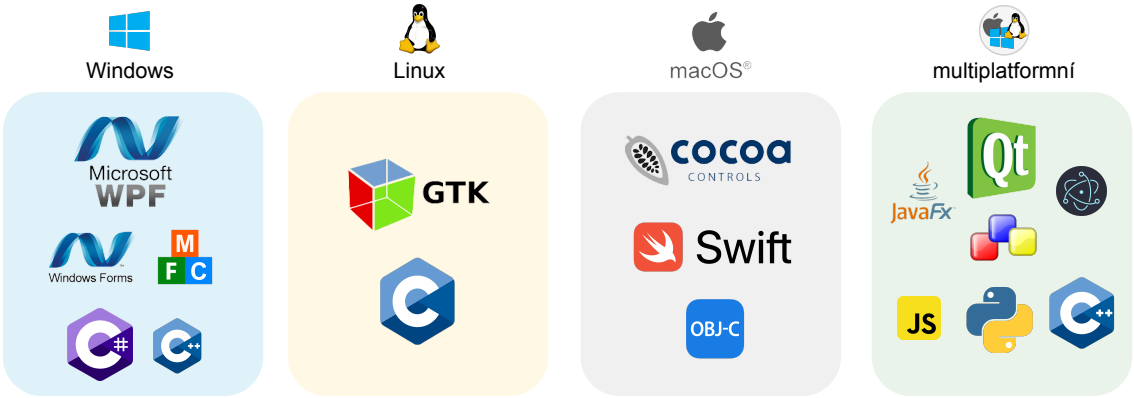
3.1.3 Požadavky na PC aplikaci

Dnes má vývojář široké možnosti vývoje PC GUI aplikací. Pokud je cíleno pouze na platformu Windows, doporučuje se použít .NET framework a technologii WPF. Stále je taky mezi uživateli populární starší technologie Windows Forms. WPF oproti Windows Forms nabízí vykreslování pomocí DirectX, návrh GUI textově pomocí XAML a příjemnou integraci návrhového vzoru MVVM (obdoba MVC) pomocí bindingu. Aplikace běží na virtuálním stroji a potřebuje tedy více systémových prostředků. Naopak těží z dostupných knihoven, rychlé kompilace, kvalitního debuggeru a moderních vlastností programovacího jazyka.

Celý .NET framework a jazyk C# prochází poslední roky intenzivním vývojem. Od .NET Framework přes .NET Core po čerstvě vydaný .NET 5 uběhlo již skoro 20 let a na rozdíl od Javy (z které se inspirovali) se z něj stal moderní ekosystém s velkou budoucností. Veškeré větve se spojily do jediné (.NET 5), která míří mimo jiné k multiplatformnímu GUI. Aktuálně je však WPF stále závislé na Windows. Z toho důvodu byl .NET zavržen. Nemožnost psát PC aplikaci v jazyce C# je tedy smutný, ale očekávaný fakt. [21]

Pokud by vývojář cílil pouze na některou linuxovou distribuci, pravděpodobně by sáhl po knihovně GTK. A to logicky z toho důvodu, že je napsána v C, stejně jako celé linuxové jádro. [22] Pokud jazyk C nevyhovuje, je možné použít 4. nejpopulárnější jazyk roku 2020 a to Python. [23] Tento interpretovaný jazyk je multiplatformní, ale nejvíce je používán ve světě open-source. V Pythonu existuje binding na knihovny GTK nebo Qt. Několika řádky a bez kompilace lze jednoduše vytvořit GUI aplikaci. Nevýhoda je nižší rychlost oproti nativnímu kódu a závislost na interpretru.

Pokud bychom chtěli vyvíjet pouze pro systém OS X, neboli nově macOS, zvolili bychom jazyk Swift a framework Cocoa. Moderní Swift je doporučován oproti staršímu Objective-C. Pro vývoj takových aplikací se standardně používá IDE Xcode který běží pouze na operačním systému macOS. Toto omezení lze vyřešit například virtualizací. Je-li potřeba publikovat aplikace do oficiálního App Store, je nutné platit roční poplatek 99 dolarů za vývojářskou licenci Apple Developer Program.



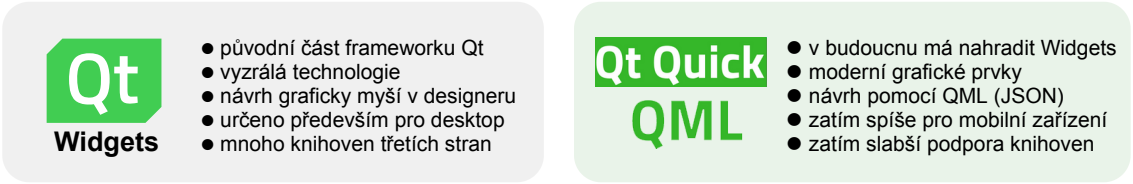
obr 3.14: Přehled nejpoužívanějších frameworků pro tvorbu GUI v PC aplikacích

Multiplatformní aplikace se také tradičně píší v Javě, ale já nejsem jejím příznivcem. Její přehnané paměťové nároky, zastaralá syntaxe a pomalý vývoj mě vždy donutily jít jinou cestou. Další možností je dnes populární Electron a mezi amatéry oblíbený jazyk Javascript. Tudy zcela jistě cesta nevede. Aplikace v Electronu jsou velké, pomalé a nestabilní, světlou výjimkou budiž Visual Studio Code. Všechny zmíněné technologie mají své výhody, ale pro tento projekt se nehodí.

Framework Qt, původně produkt firmy Nokia, je známý ze starých telefonů se systémem Symbian S60. Je napsán v jazyce C++, což může mnoho vývojářů odradit. Na druhou stranu se aplikace v Qt vyznačují vysokou rychlostí a díky kvalitní dokumentaci také širokou podporou systémů a zařízení. Framework Qt je aktivně vyvíjen již přes 22 let. Je to dnes jedna z nejlepších alternativ pro vývoj multiplatformních GUI aplikací. Dosvědčuje to fakt, že klíčové aplikace největších firem světa jsou psány právě v Qt (AMD Radeon Software, RME TotalMix, Tesla Model S infotainment, Ableton Live, Adobe Photoshop, Autodesk Maya nebo CryEngine). [24] Proto byl pro vývoj EMBO zvolen framework Qt.

Pro vývojáře existuje placená nebo volně dostupná verze Qt. Ve verzi zdarma chybí některé moduly, musí se explicitně uvést, že se používá Qt, statické linkování knihoven není možné a samozřejmě taková aplikace nesmí sloužit komerčním účelům. Ani jedna z těchto věcí není pro pro většinu open-source projektů klíčová.

Vývoj v Qt se dělí na dvě hlavní větve. Qt Widgets a Qt Quick. Qt Widgets je původní a primární metoda pro vývoj PC aplikací. Nejčastěji se používá multiplatformní IDE Qt Designer, který obsahuje vše potřebné. Návrh GUI je realizován myší v aplikaci. Modernější přístup je použít deklarativní jazyk QML. Na něm je postavená větev Qt Quick, která cílí na mobilní a responzivní aplikace.



obr 3.15: Hlavní rozdíly mezi frameworkem Qt Widgets a Qt Quick QML

Použít pro návrh GUI textový formát pro výměnu dat není nový nápad. V JavaFX se pro GUI používá jazyk FXML, v .NET WPF je to zase XAML, oproti nim je QML postaven na jazyku JSON. Použití jazyků JSON a XML pro vývoj GUI přináší pouze výhody. Patří mezi ně lepší čitelnost, kompatibilita i rozšiřitelnost.

Pro vývoj výkonných aplikací pro stolní počítače je doporučována větev Qt Widgets. Z tohoto důvodu, a také kvůli rozsáhlejší dokumentaci, lepší podpoře knihoven a celkově vyzrálejší technologii byla pro vývoj EMBO zvolena větev Qt Widgets.

Jelikož již mám zkušenost s vývojem C++ aplikací, zmíním se o tomto tématu jen okrajově. C++ je vnímáno vývojářskou komunitou jako ne příliš oblíbený jazyk. V roce 2020 se umístilo v anketě oblíbených jazyků až na 18. místě, hned za Javou (první skončil Rust a poslední 25.

Perl). Jazyk C++ měl být vylepšenou verzí jazyka C. Přinesl toho ale mnohem víc. Dnes je C++ striktně odmítán od vývojářů linuxového jádra po automotive. [25]

Na jednu stranu je C++ vysoce výkonný a rozšířený jazyk, na druhou stranu je ale příliš překombinovaný a nepřehledný. Jinými slovy, je velice lehké v něm psát nepřehledný a chybný kód. Pokud by se z C++ odstranily výjimky, šablony a vícenásobná dědičnost, byl by daleko předvídatelnější. Toto mimo jiné řeší dialekt Embedded C++ (EC++).

Qt Designer je pouze editor. Překladač se volí mezi MSVC (Windows), GCC (Linux) nebo LLVM/Clang (macOS). Jako generátor konfigurací je pro Qt5 výchozí nástroj QMake. Ten se nastavuje v projektovém souboru s příponou .pro. V budoucí verzi Qt6 dojde ke změně z QMake na populární CMake. Qt designer automatizuje značnou část projektového nastavení.

Příjemnou funkcí Qt Designeru je integrace analyzátoru paměti Valgrind. Je sice přítomný jen ve verzi pro Linux, pro vývoj rozsáhlých C++ aplikací je ale užitečným pomocníkem. C++ neobsahuje automatickou správu paměti jako C# nebo Java (Garbage Collector). Programátor je sám zodpovědný za korektní uvolňování paměti. I když se používají Smart Pointery, je dobré si zkontrolovat, jestli někde nedochází k memory-leakům, protože chyby týkající se alokace paměti jsou jedny z nejčastějších problémů C++ programů.

Na počátku vývoje každé kvalitní OOP aplikace je dobré provést objektovou analýzu. Definují se třídy, objekty a relace mezi nimi. Z těchto informací pak vyplyne třídní diagram. Dále se připraví základní scénáře použití a k nim odpovídající diagramy aktivit. Pro tvorbu těchto analýz se standardně používají nástroje UML. Z těchto grafů většinou vyplynou hlavní rysy kódu.

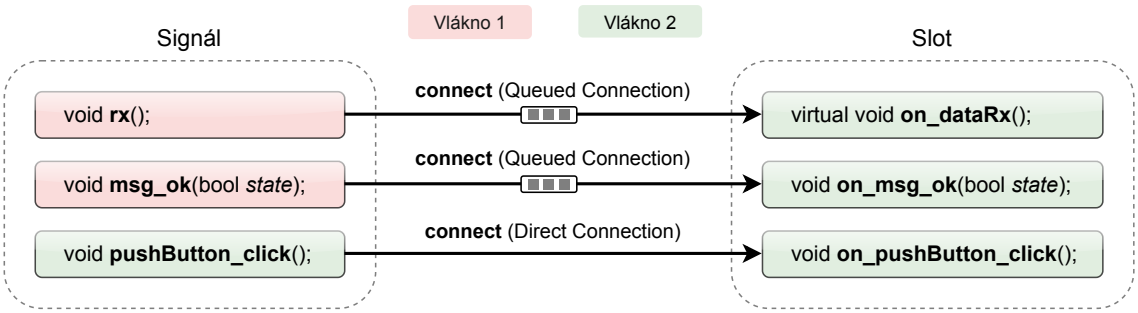
obr 3.16: Zvolené technologie pro vývoj PC aplikace osciloskopu EMBO

Vývojář zjistí, kde by mohly být problémy, a případně změní návrh ještě v počátku. Pokud by to musel udělat později, stálo by to více času, tedy i prostředků. Většinou pozdější zásah do architektury způsobí zvýšení nepřehlednosti a nestability celé aplikace. Podle těchto plánu pak pokračuje vývoj, stane se z nich tedy automaticky dokumentace.

Vývoj Qt aplikací má své implementační specifikum. Jde o princip signál-slot. Je na něm postaveno celé Qt, proto je nutné, aby vývojář tomuto principu porozuměl ještě před objektovým návrhem aplikace. Každá GUI aplikace je závislá na systému zachytávání uživatelských událostí (klik myší) a jejich asynchronním zpracování.

Nejstarší a nejjednodušší způsob, jak toto řešit, je použít ukazatel na funkci jako callback a frontu. Příkladem jsou Win32 aplikace. Ukazatele na funkci jsou sice velmi rychlé, nejsou ale typově bezpečné. V prostředí .NET se používají delegáti a události pro asynchronní výměnu dat mezi objekty. Jde o promyšlený a bezpečný koncept.

Oproti tomu Qt jde jinou cestou. Signál je funkce (s parametrem), kterou volá objekt posílající zprávu. Slot je funkce se stejnou signaturou v jiném objektu, která slouží jako příjemce asynchronní zprávy. Podobně jako v C#, kde se pro aktivaci události musí svázat handler s daným objektem (přetížený operátor +=), tak v Qt se musí zavolat funkce connect, která spojí objekt signálu a objekt slotu. Je tak zaručena typová bezpečnost již během kompilace.



obr 3.17: Princip asynchronního zpracování událostí v Qt pomocí techniky signál-slot. Fronta (Queued Connection) se používá pro komunikaci mezi vlákny.

Signál může být vyslán i bez spojení se slotem, může také být vysláno více signálů do jednoho slotu. Slot může být virtuální, což umožňuje využít OOP polymorfismus. Virtuální a typově bezpečné callbacky v C++ GUI aplikaci dělají z Qt přehledný a výkonný nástroj.

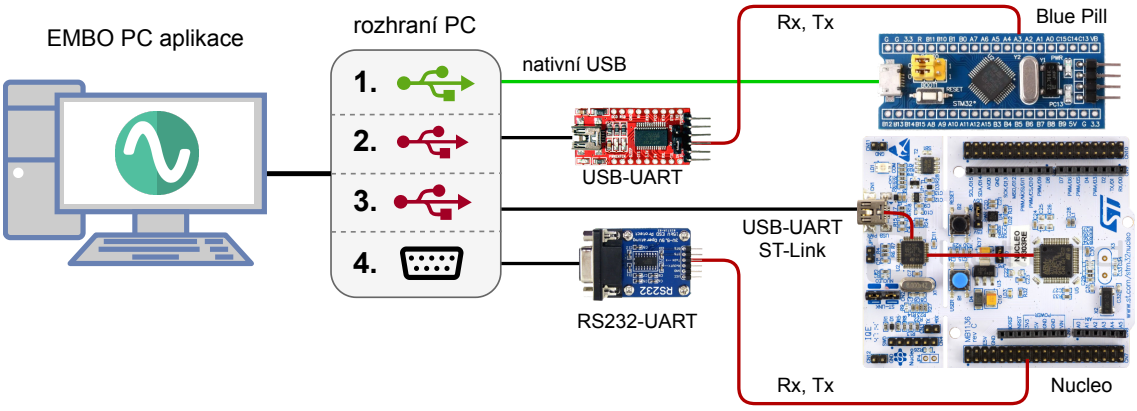
Pro vývoj osciloskopu EMBO bude kromě standardních modulů, dostupných v každém GUI frameworku, potřeba dvou speciálních knihoven. Jedná se o obsluhu sériového portu a grafů. Qt již obsahuje multiplatformní knihovnu QSerialPort, jeden problém je tedy vyřešen. Pro grafy také existuje modul QCharts, jelikož ale neumožňuje pokročilé funkce a není aktivně vyvíjen, rozhodl jsem se jej nepoužít. Z dostupných možností zbývá na výběr Qwt (Qt Widgets for Technical Applications) a nebo QCP (QCustomPlot).

Volím QCP, protože se mi velmi líbí dokumentace, kvalita kódu a dostupné funkce. Výběr má ovšem jednu nevýhodu. QCP disponuje pouze lineární interpolací, bude tedy nutné implementovat spline nebo sinc interpolaci ručně. Díky přehlednému kódu by to ale neměl být problém. Dále ze zběžné analýzy vyplývá, že minimálně bude ještě nutné dopsat knihovnu pro manipulaci s kurzory (včetně prvku Range Slider, který v Qt Widgets chybí). Knihovna QCP je distribuována jako jeden soubor se zdrojovým kódem (.h + .cpp), odpadají tedy problémy s linkováním pro různé platformy. [26]

3.1.4 Požadavky na komunikaci mezi osciloskopem a PC aplikací

Komunikace mezi osciloskopem a PC aplikací je důležitá část analýzy. Jelikož je komunikace úzkým hrdlem celého systému, musí být rozumně navržena. Širší podpora mikrořadičů zúžila výběr na klasickou sériovou komunikaci, protože UART je dostupný téměř ve všech mikrořadičích. Přesněji se bude jednat o tyto 4 režimy:

1. nativní USB v režimu CDC, připojeno přímo do PC ($\gtrsim 20\times$ rychlejší než UART)
2. UART (Rx a Tx) připojený do PC pomocí převodníku UART-USB.
3. UART připojený přes čip ST-Link (desky Nucleo) do PC pomocí USB.
4. UART připojený do PC pomocí externího RS232-UART převodníku



obr 3.18: Přehled možností jak připojit osciloskop EMBO k počítači

Maximální rychlost komunikace pomocí UART je omezena na 115200 bps. Přenos USB je sice v režimu virtuálního COM portu, oproti UART je ale mnohem rychlejší. Kvůli pomalé rychlosti přenosu je nutné realizovat trigger osciloskopu přímo v zařízení. Díky omezené paměti bude obnovovací frekvence obrazovky osciloskopu (FPS) přímo závislá na rychlosti příjmu dat. Pro vyšší hodnoty nastavené paměti pro vzorky bude přenos trvat déle, klesne tedy FPS.

Signál vzorkovaný frekvencí 1 MHz o rozlišení 12 bit zaplní paměť o velikosti 1000 vzorků za 1 ms (2000 bytů). Následující UART přenos do PC trvá zhruba 200 ms, USB přenos trvá zhruba 10 ms. Během přenosu je vzorkování vypnuto, aby byla zachována integrita dat. V případě komunikace pomocí UART tak může osciloskop zaznamenat 1 ms signálu pouze jednou za 200 ms. Protože návrh systému nepočítá s žádnými externími komponentami, musí se s tímto omezením počítat.

Dalším důležitým rozhodnutím je volba mezi textovým a binárním protokolem. Nyní zběžně popíšu jejich hlavní rozdíly:

- Binární protokol — rychlý, bezpečný, nepřehledný, hůře rozšiřitelný
- Textový protokol — pomalejší než binární, uživatelský vstřícný, lehce rozšiřitelný

Textový ASCII protokol je výhodný pro přenos krátkých zpráv. Jeho největším přínosem je čitelnost a srozumitelnost. Pro testování není potřeba žádný speciální software a odpadá také počítání kontrolních součtů. Nevýhoda je vyšší režie. Zprávy jsou nejčastěji definovány pomocí speciálních ukončovacích znaků, může to být například kombinace znaků <CR> a <LF>. Příkladem textového protokolu je například HTTP/1.

Binární protokol nabízí lepší efektivitu přenosu, ale horší čitelnost. Z komerčního hlediska může být špatná čitelnost chtěná vlastnost, v otevřených projektech je to ale naopak. Binární protokol se používá především pro výměnu dat pevně definovanými datovými strukturami. Zprávy se nejčastěji definují pomocí hlavičky a patičky s kontrolním součtem CRC.

Striktně binární protokol není vhodný pro osciloskop EMBO. Komunikace textem je výhodná pro krátké příkazy a nastavení. Výměnu dat z osciloskopu by zase bylo dobré realizovat binárně. Z toho vyplývá, že ideálním řešením by bylo použít textový protokol s možností přepnutí do binárního režimu.

Další úvahy jistě směřují k tomu, jaký protokol použít. Vždy je možné si vytvořit vlastní protokol. Pro triviální přístroj s několika typem zpráv je to rychlé a korektní řešení. EMBO zahrnuje 6 laboratorních přístrojů, každý s různou funkcionalitou. Pouštět se do tvorby vlastního kombinovaného protokolu v takovém případě asi není nejlepší nápad. Zvlášť pokud víme, že již přes 30 let existuje mezinárodní standard, který podporuje většina laboratorních měřících přístrojů.

Protokol SCPI vychází z se standardu IEEE-488.2 a je široce rozšířen. Protokol SCPI je ideálním řešením pro projekt EMBO, protože nabízí jak textový tak binární přenos. Navíc existují volně dostupné knihovny v různých jazycích, které umožňují snadnou integraci SCPI protokolu do libovolné aplikace. Já jsem zvolil populární knihovnu od Jana Breuera z fakulty ČVUT FEL. [27]

3.2 Využití systému pro správu verzí

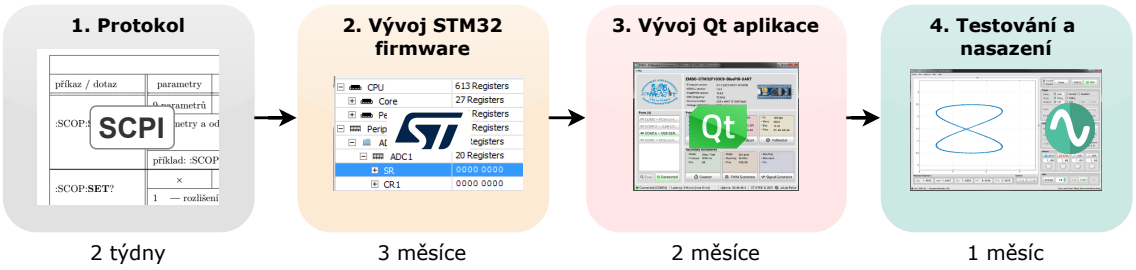
Při vývoji projektu jakéhokoliv rozsahu by se měl používat systém synchronizace zdrojových kódů. Usnadní se tím mnoho věcí. Je zajištěna integrita souborů, projekt je zálohován, je možné členit vývoj na větvě a je velice snadné se vrátit k předešlé verzi a vidět konkrétní změny. Příkladem je například starší systém SVN. Já jsem zvolil moderní systém git, který byl vytvořen pro vývoj linuxového jádra, konkrétně službu GitHub. Nejprve bude vývoj probíhat privátně, po obhájení této práce pak změním repozitář na veřejný.

Kapitola 4

Řešení realizace osciloskopu EMBO

Vývoj osciloskopu EMBO je rozdělen na 4 hlavní bloky. Každý blok představuje specifickou část, která navazuje na tu předchozí. Ladění firmware bez PC aplikace by šlo velmi těžko, proto je vhodné použít jednoduchý Python skript s balíčky pyserial a matplotlib.

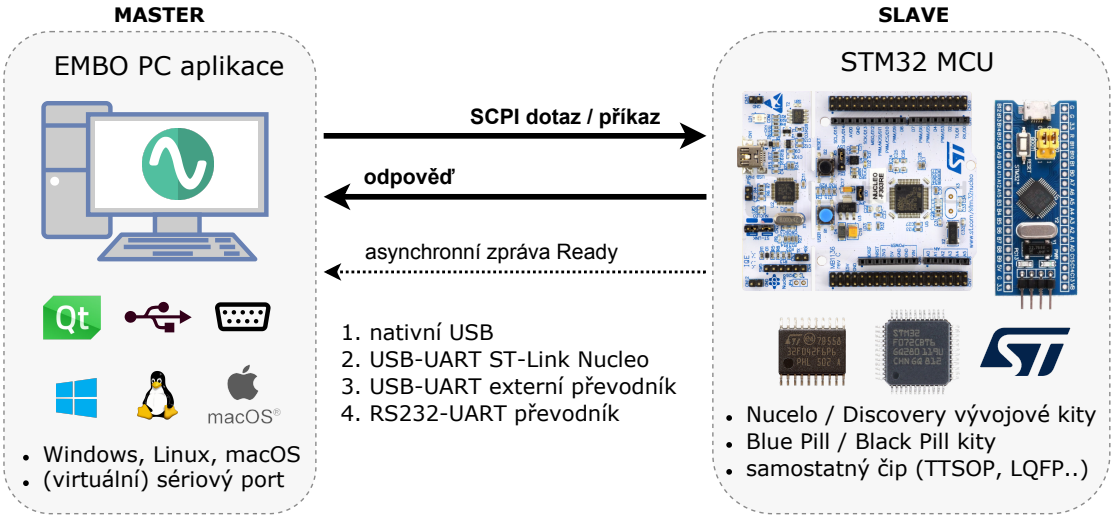
1. Návrh komunikačního rozhraní
2. Vývoj firmware zařízení pro řadiče STM32
3. Vývoj Qt PC aplikace
4. Testování a nasazení



obr 4.1: Graficky znázorněný postup realizace osciloskopu s předpokládanou délkou trvání

4.1 Fáze 1 — Návrh komunikačního rozhraní

Jelikož v plánu projektu bylo rozhodnuto, že protokol bude SCPI kompatibilní, není třeba vymýšlet základní mechanismy, jako je třeba struktura zprávy. PC aplikace je MASTER a osciloskop EMBO je SLAVE. PC aplikace řídí tok zpráv, na každou odeslanou zprávu musí přijít ihned odpověď. Jedinou výjimkou je asynchronní zpráva Ready, která informuje PC aplikaci o tom, že má poslat zprávu pro vyčtení binárních dat. Komunikuje se prioritně pomocí UART (115200 bps),pokud mikrořadič disponuje USB, tak lze použít i to. Podrobně je to popsáno v kapitole 3.1.4.



obr 4.2: Hlavní charakteristika komunikačního rozhraní

4.1.1 Základní popis protokolu

Na straně PC je implementován v jádru aplikace vlastní parser, který rozlišuje textové, binární a asynchronní zprávy. Dále každou zprávu rozdělí na pod-zprávy a díky OOP polymorfismu jednoduše pošle jako signál příslušnému objektu. Na straně osciloskopu EMBO je implementován parser jako lehce modifikovaná knihovna od pana Breuera. ([27]).

Textový standardní protokol umožňuje fungování bez nutnosti použít PC GUI aplikaci. Uživateli stačí terminál, případně může použít tento dokument k návrhu vlastní aplikace. Další námět na rozšíření projektu EMBO je implementace VISA dynamické knihovny, díky které by se dal osciloskop použít v prostředí LabVIEW, CVI nebo .NET.

- Celkový charakter komunikace
  - Na každou zprávu z PC osciloskop IHNED odpoví
  - Kdykoliv může z osciloskopu přijít asynchronní zpráva
  - Na asynchronní zprávu ze osciloskopu PC NEODPOVÍDÁ
  - Textová zpráva je ve formátu ASCII
  - Binární zpráva začíná definovanou hlavičkou (IEEE 488.2): # NXXX<DAB>...
  - Textový řetězec je uzavřen do uvozovek
  - Oddělovač prvního řádu je <CR><LF> (odděluje zprávy)
  - Oddělovač druhého řádu je středník (odděluje pod-zprávy)
  - Oddělovač třetího řádu je čárka (odděluje parametry)
- Typy (pod)zpráv
  - Příkaz (Command) — Odpověď je buď OK nebo ERROR
  - Dotaz (Querry) — Odpověď jsou data oddělená čárkou, dotaz končí otazníkem

Dotaz se liší od příkazu pouze otazníkem na konci. Každá pod-zpráva může obsahovat libovolný počet parametrů. Parametry jsou od příkazu odděleny jednou mezerou, vždy jsou ale až za případným otazníkem. Příkazy se nemusí psát celé, stačí jen tu část s velkými písmy (IEEE 488.2).

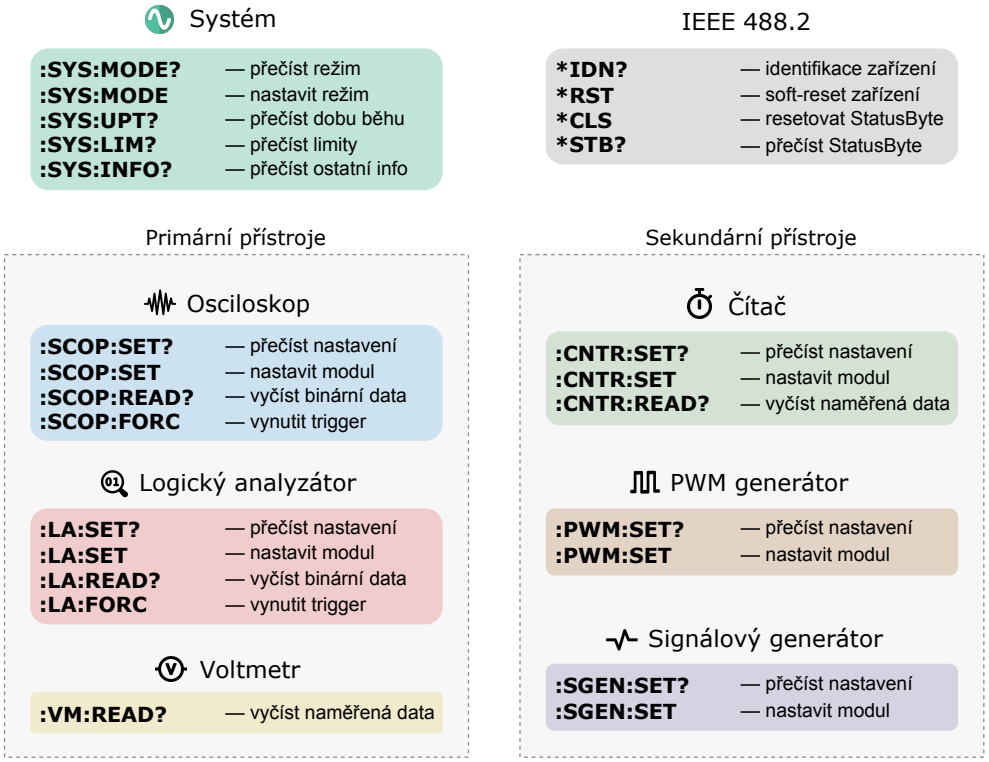
4.1.2 Rozdělení protokolu dle modulů

Firmware EMBO je rozdělen do modulů, které vždy reprezentují jednu funkční skupinu. Podobně jako u ostatních měřících přístrojů s podporou SCPI, ke každému modulu lze přistupovat pomocí dvojtečky (:). Kořenový modul (root) značí první dvojtečka zleva, kterou je nutné použít pro zřetězené příkazy. Následuje jméno modulu, další dvojtečka a příkaz nebo dotaz. SCPI podporuje vnořené moduly, pro jednoduchost ale byla použita jen jednovrstvá topologie . Na následujících příkladech lze vidět příkaz s parametrem, dotaz s dvěma parametry a dva dotazy v jedné zprávě:

- :MODul:PŘÍKaz PARAMetr<CR><LF>
- :MODul:DOTaz? PARAMetr1,PARAMetr2<CR><LF>
- :MODul:DOTaz1;:MODul:DOTaz2?<CR><LF>

Mandatorní příkazy dle IEEE 488.2 jsou ve speciálním modulu definovaným hvězdičkou (*), na obrázku jménem SCPI. Ostatní příkazy patří vždy do konkrétního modulu. Tři měřící přístroje jsou označeny jako primární (VM, SCOPE, LA) a další tři jako sekundární (CNTR, PWM, SGEN). Vždy může být aktivní jen jeden primární přístroj, naopak sekundární přístroj je na ostatních nezávislý. Instrument SGEN je dostupný jen v případě, že STM32 čip má v sobě DAC převodník.

- :**SYS**tem — systém
- :**VM** — voltmetr (primární přístroj)
- :**SCOP**e — osciloskop (primární přístroj)
- :**LA** — logický analyzátor (primární přístroj)
- :**CNTR** — čítač (sekundární přístroj)
- :**PWM** — PWM generátor (sekundární přístroj)
- :**SGEN** — signálový generátor (sekundární volitelný přístroj)



obr 4.3: EMBO SCPI protokol - přehled modulů a jejich příkazů

SCPI — příkazy dle IEEE 488.2 standardu

Tyto příkazy musí být přítomny, aby byl osciloskop SCPI kompatibilní. Jedná se o obecné příkazy, které jsou univerzální a slouží především k identifikaci a správě zařízení. Dle standardu je jich 13, pro EMBO budou použity jen 4. Zbylých 9 není nutné popisovat, jsou implementovány SCPI knihovnou, ale nejsou využity. Definice je v tabulce 4.1.

SYStem — systémové příkazy

Systémové příkazy slouží ke komunikaci nadřazené všem přístrojům. Patří zde především dotazy, kterými lze vyčíst numerické limity osciloskopu EMBO (LIM?) a dostupnou funkcionalitu (INFO?). Další důležitý prvkem je přepínání režimů, které bude podrobněji popsáno v následující kapitole. Pro přepnutí primárního přístroje je vyčleněn příkaz MODE. Zbývá dotaz na délku běhu systému UPT?, který slouží k udržování komunikace. Definice je v tabulkách 4.2, 4.3 a 4.4.

VM — voltmetr

Voltmetr je jednoduchý primární přístroj založený na modulu DAQ, který bude podrobně popsán v následující kapitole. Je dostupný pouze jeden dotaz pro vyčtení hodnot napětí (ze 4 kanálů) a napájecího napětí Vcc jménem READ?. Bez parametrů vypíše poslední aktuální hodnoty napětí. S parametrem 1 vyčítá hodnoty v sekvenčním režimu, kdy je zaručeno, že odstup mezi vzorky je přesně 10 ms. Druhý režim je vhodný pro PC GUI aplikace, které jsou schopny rychlé komunikace, kdy typicky se zašle několik takových dotazů v jedné zprávě. Definice je v tabulce 4.5.

Asynchronní odpovědi

Pro režimy osciloskopu a logického analyzátoru je nutné implementovat asynchronní odpověď. Ta může přijít kdykoliv a nese informace o typu triggeru a numerické pozici začátku dat v kruhovém bufferu. Osciloskop se zapne, nastaví se jeho parametry a pak se čeká na asynchronní zprávu Ready. Ta přijde v čase závislém na typu triggeru a velikosti paměti pro vzorky. Po přijetí zprávy Ready následuje okamžité vyčtení dat a vše se opakuje. Tato metoda je mnohem efektivnější než cyklické dotazování (polling). Definice je v tabulce 4.11.

SCOPe — osciloskop

Osciloskop je stěžejní primární přístroj, postavený na modulu DAQ. První dotaz, který by na tento přístroj měl být zaslán, je přečtení nastavení pomocí dotazu SET?. Dále, když víme jak je modul nastaven, můžeme upravit jeho konfiguraci pomocí příkazu SET. Data lze vyčíst pouze po přijetí asynchronní odpovědi Ready dotazem READ?. Modul se resetuje po každém příkazu SET nebo MODE. Volitelným příkazem je FORC, který vynucuje okamžitý trigger (analogie tlačítka Force Trigger na stolních osciloskopech). Definice je v tabulce 4.6.

LA — logický analyzátor

Logický analyzátor je postaven na modulu DAQ velmi podobně jako osciloskop. Jejich veřejné rozhraní je téměř shodné, i když implementace firmware využívá GPIO místo ADC. Rozdílem oproti osciloskopu je nedostupnost některých parametrů, které logicky nemají význam (bitové rozlišení, vybrané kanály, úroveň triggeru a maximální vstupní impedance). Definice je v tabulce 4.7.

CNTR — čítač

Čítač je jednoduchý sekundární (nezávislý) přístroj pro měření frekvence. Lze používat kdykoliv, například spolu s PWM generátorem a osciloskopem, kdy se příkazy zašlou v jedné zprávě. Čítač se zapíná příkazem SET, lze zvolit mezi režimem Fast pro rychlé signály a režimem Slow pro pomalé. Naměřené hodnoty se čtou dotazem READ?. Pro pomalé signály může trvat jedno změření až 2 sekundy. Pokud čítač nedetekoval náběžnou hranu během 2 sekund, vypíše místo frekvence timeout error. Definice je v tabulce 4.8.

PWM — PWM generátor

PWM generátor je poslední sekundární přístroj. Jde o 2-kanálový synchronní generátor. Pro zachování jednoduchosti systému lze ovládat pouze jedním dotazem a příkazem SET, podobně jako signálový generátor. Příkazem SET se nastavuje frekvence, společná pro oba kanály, střída pro každý kanál zvlášť a offset druhého kanálu od prvního. Dále lze oba kanály zapnout nebo vypnout. Definice je v tabulce 4.9.

SGEN — signálový generátor

Signálový generátor je podobně jako čítač sekundární přístroj, který lze používat nezávisle ke generování základních signálů (sinus, pila,trojúhelník,obdélník,šum a konstanta). K jeho ovládání slouží pouze jeden příkaz a dotaz SET. Definice je v tabulce 4.10.

Definice EMBO protokolu

IEEE 488.2			
příkaz / dotaz	parametry	odpověď	popis
***IDN**?	×	A,B,C,D	identifikace osciloskopu EMBO
	A — jméno autora B — název produktu + řada STM32 C — sériové číslo (vždy 0) D — verze firmware (datum kompilace)		
***RST**	×	OK	soft-reset osciloskopu
***CLS**	×	OK	reset StatusByte
***STB**?	×	StatusByte	vyčtení StatusByte

Tabulka 4.1: Definice protokolu pro standardní příkazy dle IEEE 488.2

:**SYS**tem			
příkaz / dotaz	parametry	odpověď	popis
:SYS:**MODE**?	×	VM | SCOPE | LA	dotaz na hlavní režim
:SYS:**MODE**	VM | SCOPE | LA	OK	nastavení hlavního režimu
	VM — voltmetr SCOPE — osciloskop LA — logický analyzátor		
:SYS:**UPT**ime?	×	HH:MM:SS.Z	dotaz na dobu běhu systému
	hodiny : minuty : sekundy . setiny		

Tabulka 4.2: Definice protokolu pro modul SYStem — systém (1. část)

:**SYS**tem			
příkaz / dotaz	parametry	odpověď	popis
:SYS:**LIM**its?	×	17 parametrů	dotaz na numerické limity
	1 — max. vz. frekvence 12-bit ADC [Sps]		
	2 — max. vz. frekvence 8-bit ADC [Sps]		
	3 — max. paměť pro DAQ [Hz]		
	4 — max. vz. frekvence modulu LA [Sps]		
	5 — max. frekvence modulu PWM [Hz]		
	6 — přítomný 2. kanál PWM (1 | 0)		
	7a — počet DAQ kanálů (2 | 4)		
	7b — počet hlavních ADC (1 | 2 | 4)		
	7c — režim DualMode (znak 'D')		
	7d — režim Interleaved (znak 'I')		
	8 — režim 8-bit ADC (1 | 0)		
	9 — přítomný DAC (1 | 0)		
	10 — vz. frekvence modulu VM [Sps]		
	11 — paměť na vzorky modulu VM		
	12 — timeout modulu CNTR v ms		
	13 — max. frekvence modulu SGEN [Hz]		
	14 — max. paměť modulu SGEN		
	15 — max. frekvence modulu CNTR [Hz]		
	16 — rezerva vzorků v DAQ bufferu		
	17 — pozice kanálů LA v portu		

Tabulka 4.3: Definice protokolu pro modul SYStem — systém (2. část)

:**SYS**tem			
příkaz / dotaz	parametry	odpověď	popis
:SYS:**INFO**?	×	10 parametrů	dotaz na informace o systému
	1 — FreeRTOS verze systému		
	2 — STM LL verze ovladačů		
	3 — dostupné režimy komunikace		
	4 — frekvence CPU [MHz]		
	5 — napájecí napětí Vcc [mV]		
	6 — 4 piny osciloskopu a voltmetru		
	7 — 4 piny logického analyzátoru		
	8 — 1 pin čítače		
	9 — 2 piny PWM generátoru		
	10 — 1 pin signálového generátoru		

Tabulka 4.4: Definice protokolu pro modul SYStem — systém (3. část)

:**VM**			
příkaz / dotaz	parametry	odpověď	popis
:VM:**READ**?	S	U1,U2,U3,U4,Vcc	dotaz na napětí 4 kanálů a Vcc
	S — 1: sekvenční režim | 0		
	U1 — napětí kanálu 1 [V]		
	U2 — napětí kanálu 2 [V]		
	U3 — napětí kanálu 3 [V]		
	U4 — napětí kanálu 4 [V]		
	Vcc — napájecí napětí [V]		
	příklad: :VM:READ? 1 ⇒ 1.723,0.124,0.564,2.721,3.338		

Tabulka 4.5: Definice protokolu pro modul VM — voltmetr (4 kanály)

:**SCOP**e			
příkaz / dotaz	parametry	odpověď	popis
:SCOP:**SET**	9 parametrů	OK,Z,T,F	nastavení všech parametrů osciloskopu
	parametry a odpovědi definovány níže		
	ex: :SCOP:SET 12,100,850000,1000,1,50,R,A,50 ⇒ OK,73.4,1.5,857142		
:SCOP:**SET**?	×	9 parametrů + Z,T,F	vyčtení všech parametrů osciloskopu
	1 — rozlišení ADC [bit] 2 — paměť [vzorky na kanál] 3 — vzorkovací frekvence v Hz [Sps] 4 — volba 4 kanálů (1 | 0) 5 — trigger: aktivní kanál (1 - 4) 6 — trigger: úroveň (0 - 100 %) 7 — trigger: hrana (R | F) R:Rising , F:Falling 8 — trigger: režim (A | N | S | D) A:Auto , N:Normal, S:Single, D:Disabled 9 — trigger: pretrigger (0 - 100 %) Z — maximální impedance zdroje [Ω] T — doba vzorkování [ns] F — reálná vzorkovací frekvence [Sps]		
:SCOP:**READ**?	×	binární data	vyčtení kruhového bufferu osciloskopu
	binární data s hlavičkou dle IEEE 488.2 ve tvaru: $\#NXXX\langle B1\rangle\langle B2\rangle\langle B3\rangle\langle B4\rangle$ ... kde N je počet znaků čísla XXX a číslo XXX udává počet bytů ve zprávě		
:SCOP:**FORC**etrig	×	OK	vynucení okamžitého triggeru

Tabulka 4.6: Definice protokolu pro modul SCOPe — osciloskop (4 kanály)

<table>
<tr><th colspan="4">:LA</th></tr>
<tr><th>příkaz / dotaz</th><th>parametry</th><th>odpověď</th><th>popis</th></tr>
<tr><td rowspan="3">:LA:SET</td><td>6 parametrů</td><td>OK,F</td><td rowspan="2">nastavení všech parametrů logického analyzátoru</td></tr>
<tr><td colspan="2">parametry a odpovědi definovány níže</td></tr>
<tr><td colspan="3">příklad: :LA:SET 100,850000,1,R,A,50 ⇒ OK,857142</td></tr>
<tr><td rowspan="2">:LA:SET?</td><td>×</td><td>6 parametrů + F</td><td rowspan="2">vyčtení všech parametrů logického analyzátoru</td></tr>
<tr><td colspan="2">1 — paměť [vzorky na kanál]
2 — vzorkovací frekvence [Sps]
3 — trigger: aktivní kanál (1 - 4)
4 — trigger: hrana (R | F | B)
R:Rising , F:Falling, B:Both
5 — trigger: režim (A | N | S | D)
A:Auto , N:Normal,
S:Single, D:Disabled
6 — trigger: pretrigger (0 - 100 %)
F — reálná vzorkovací frekvence [Sps]</td></tr>
<tr><td rowspan="2">:LA:READ?</td><td>×</td><td>binární data</td><td rowspan="2">vyčtení kruhového bufferu logického analyzátoru</td></tr>
<tr><td colspan="2">binární data s hlavičkou dle IEEE 488.2
ve tvaru: $\#NXXX\langle B1\rangle\langle B2\rangle\langle B3\rangle\langle B4\rangle$ ...
kde N je počet znaků čísla XXX a číslo
XXX udává počet bytů ve zprávě</td></tr>
<tr><td>:LA:FORCetrig</td><td>×</td><td>OK</td><td>vynucení okamžitého triggeru</td></tr>
</table>

Tabulka 4.7: Definice protokolu pro modul LA — logický analyzátor (4 kanály)

:**CNTR**			
příkaz / dotaz	parametry	odpověď	popis
:CNTR:**SET**?	×	E,M	vyčtení parametrů čítače
	parametry a odpovědi definovány níže		
:CNTR:**SET**	E,M	×	nastavení čítače
	E — zapnutí čítače (1 | 0) M — režim Fast nebo Slow (1 | 0)		
:CNTR:**READ**?	×	F,P	vyčtení naměřených hodnot, případně je vypsán error
	F — frekvence P — perioda		
	příklad: :CNTR:SET 1,0;:CNTR:READ? ⇒ OK;1 kHz,1 ms		

Tabulka 4.8: Definice protokolu pro modul CNTR — čítač (1 kanál)

:**PWM**			
příkaz / dotaz	parametry	odpověď	popis
:PWM:**SET**?	×	F,D1,D2,O,E1,E2,R	vyčtení parametrů PWM generátoru
	parametry a odpovědi definovány níže		
:PWM:**SET**	F,D1,D2,O,E1,E2	OK,R	nastavení PWM generátoru
	F — frekvence obou kanálů [Hz] D1 — střída 1. kanálu (0 - 100 %) D2 — střída 2. kanálu (0 - 100 %) O — offset 2. kanálu (0 - 100 %) E1 — zapnutí 1. kanálu (1 | 0) E2 — zapnutí 2. kanálu (1 | 0) R — reálná frekvence [Hz]		
	příklad: :PWM:SET 1100000,50,50,50,1,1 ⇒ OK,1107692.3		

Tabulka 4.9: Definice protokolu pro modul PWM — PWM generátor (4 kanály)

:**SGEN**			
příkaz / dotaz	parametry	odpověď	popis
:SGEN:**SET**?	×	F,A,O,M,E,R,N	vyčtení parametrů signálového generátoru
	parametry a odpovědi definovány níže		
:SGEN:**SET**	F,A,O,M,E	OK,R,N	nastavení signálového generátoru
	F — požadovaná frekvence [Hz] A — amplituda (0 - 1000 %) O — offset (0 - 100 %) M — režim: 0: CONST (konstanta) 1: SINE (sinus) 2: TRIANGLE (trojúhelník) 3: SAWTOOTH (pila) 4: SQUARE (čtverec) 5: NOISE (náhodný šum) E — zapnutí generátoru (1 | 0) R — reálná frekvence [Hz] N — počet generovaných vzorků		
	příklad: :SGEN:SET 1000,50,0,1,1 ⇒ OK,1000.0,500		

Tabulka 4.10: Definice protokolu pro modul SGEN — signálový generátor (1 kanál)

Asynchronní odpovědi		
odpověď	parametry	popis
ReadyT,pos	T:A — triggeru vypršel časový limit (Auto) T:N — úspěšně zatriggerováno (Normal) T:S — to samé jako N akorát režim Single T:F — trigger byl vynucen uživatelem (Forced) T:D — trigger je vypnutý (Disabled) pos — pozice začátku dat v kruhovém bufferu	data jsou připravena k vyčtení z kruhového bufferu
	příklad: ReadyN,352	

Tabulka 4.11: Definice protokolu pro asynchronní odpovědi osciloskopu EMBO

4.2 Fáze 2 — Vývoj STM32 firmware

Pro vývoj firmware je použito oficiální vývojové prostředí STM32CubeIDE a jazyk C. V naprosté většině případů jsou použity LL ovladače od výrobce, jen v případě periferie USB je nutné použít ovladač HAL. Pro komunikaci je použita volně dostupná SCPI knihovna [27]. Jako operační systém reálného času je vybrán volně dostupný FreeRTOS. Veškerý kód kromě knihoven třetích stran a některých pomocných funkcí je má originální práce.

4.2.1 Hlavní charakteristika firmware

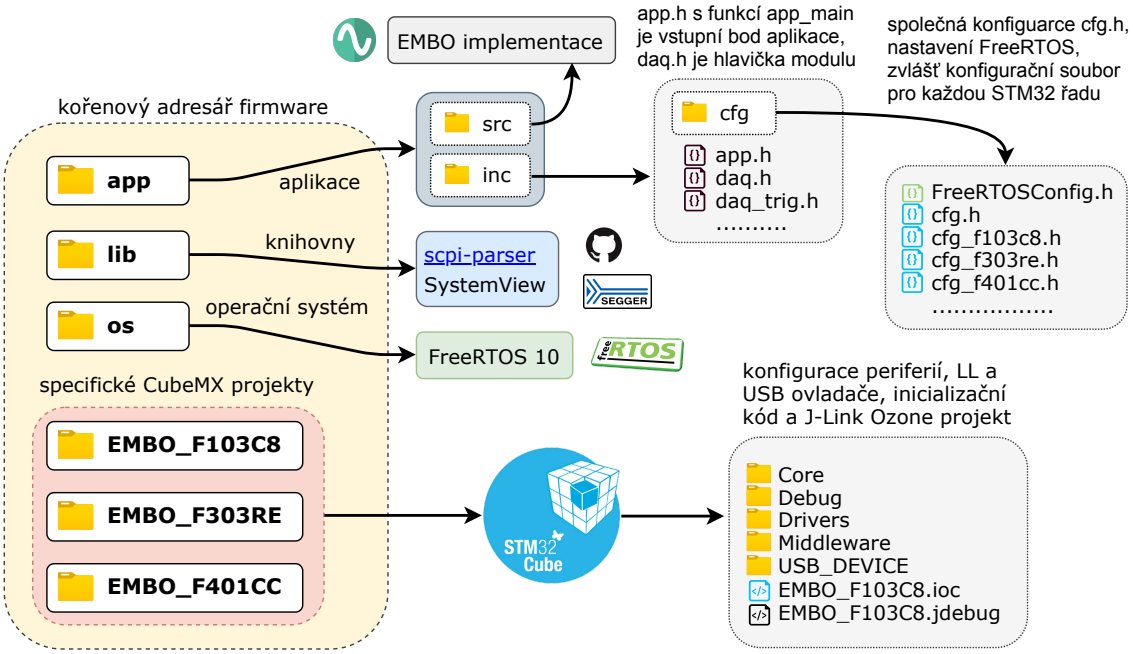
Správa projektu

Projekt je rozdělen do 3 hlavních složek (aplikace, knihovny, operační systém), ve kterých je implementována společná část kódu. Dále existují specifické složky pro každou STM32 řadu. V těchto složkách je vždy založen projekt s CubeMX konfigurací, vytvořena inicializační funkce main.c, na jejímž konci se zavolá společná část kódu. Architektonické odlišnosti jsou řešeny na úrovni kompilace pomocí maker (#ifdef). Pro každou STM32 řadu existuje kromě složky s projektem také konfigurační soubor ve společné části v podsložce cfg. Ten obsahuje specifické nastavení pro každý čip a řeší mírné odlišnosti v LL ovladačích. Přehledně je to znázorněno na obrázku 4.4.

- **app** — aplikační společná část firmware (hlavní EMBO implementace)
- **lib** — cizí společné knihovny
- **os** — operační systém reálného času FreeRTOS
- EMBO_F103C8 — CubeMX projekt a inicializace periferií pro STM32F103
- EMBO_F303RE — CubeMX projekty pro další řady STM32

Alokace paměti

V celém firmware je striktně zakázána dynamická alokace. Důvody jsou popsány v kapitole 3.1.2. Díky použití FreeRTOS lze mít nad alokovanou pamětí kompletní přehled. Nejprve je nutné nastavit linker-skript (ručně nebo pomocí CubeMX), kde se omezí velikost haldy (heap) na 0 a velikost zásobníku (stack) na hodnotu 0x300. Tato hodnota byla zvolena jako kompromis pro inicializaci periferií, před zapnutím plánovače OS, a pro rutiny přerušení. Dále se v konfiguračním souboru FreeRTOSConfig.h povolí pouze statická alokace a soubory heap_X.c se odstraní. Ujistíme se, že kód neobsahuje žádné volání *malloc*.



obr 4.4: Rozdělení firmware do adresářů a funkčních celků včetně popisu

Nyní zbývá správně definovat velikost zásobníku pro jednotlivá vlákna. K tomu může pomoct zapnout detekci hladiny zásobníku (*water mark*) a detekci přetečení zásobníku. Určení velikosti zásobníku vyžaduje vždy kompromis. Je dobré nechat rezervu, aby se předešlo přetečení, ale zase je zbytečné plýtvat pamětí, která je v tomto projektu velmi cenná. Osvědčilo se nechat rezervu minimálně 20 bloků (32-bit integer). Minimální velikost zásobníku pro primitivní tasky a IDLE task je určena na 64 bloků.

Část souboru LD (linker skript):

- _Min_Heap_Size = 0x0 — dynamická alokace je zakázána
- _Min_Stack_Size = 0x300 — zásobník pro inicializaci a obsluhy přerušení

Část souboru FreeRTOSConfig.h (konfigurační soubor FreeRTOS):

- configSUPPORT_DYNAMIC_ALLOCATION 0 — zakázána dynamická alokace
- configSUPPORT_STATIC_ALLOCATION 1 — povolena statická alokace
- configTOTAL_HEAP_SIZE 0 — velikost haldy
- configMINIMAL_STACK_SIZE 64 — minimální velikost zásobníku tasku
- configCHECK_FOR_STACK_OVERFLOW 2 — detekce přetečení zásobníku
- INCLUDE_uxTaskGetStackHighWaterMark 1 — detekce hladiny zásobníku

Základní parametry tasků. Hvězdičkou * je vyznačeno použití mutexu.					
N	jméno	priorita (max:5)	perioda (ms)	zásobník (uint32)	popis
1	**wd**	3	10	40	watchdog, blikání LED
2*	**trig_check**	1	5	65	pre-trigger, Auto trigger, Ready
3*	**trig_post_count**	5	×	55	počítání post-triggeru (kritické!)
4*	**comm_and_init**	2	×	300	komunikace, inicializace
5	**cntr**	4	50	55	čítač - výpočet měřených hodnot

Tabulka 4.12: Přehled FreeRTOS tasků (velikost zásobníku odpovídá STM32F103)

Tasky — přehled

Hlavní důvod integrace FreeRTOS je paralelismus, v případě jedno-jádrových mikrořadičů zdánlivý paralelismus. Paralelní úlohy se ve FreeRTOS nazývají tasky. Každý task má svůj zásobník. Jádro každého OS je plánovač, část kódu, která se cyklicky spouští a plánuje úlohy. Pro čipy ARM je typické použít pro plánovač časovač jádra SysTick s frekvencí 1 kHz. Veškeré časování v RTOS je odvozeno od tiků tohoto časovače, rozlišení plánování je tedy 1 ms.

Pro synchronizaci tasků jsem zvolil mutexy a binární semafory (lze použít i eventy). Semafor je synchronizační primitivum, které je reprezentováno nezáporným číslem. Pro semafor jsou definovány 2 atomické operace *Give* a *Take*. Take se pokusí dekrementovat číslo semaforu. Pokud se mu to podaří, program pokračuje, v opačném případě se task zablokuje a je vyřazen z plánování. Give inkrementuje číslo semaforu a program ihned pokračuje. Binární semafory slouží k synchronizaci. Mutexy jsou speciální binární semafory, u kterých musí funkci Give a Take vykonat vždy stejný task. Používají se tak pro ošetření kritických sekcí, kdy je nutné zajistit, aby daný kód vykonával právě jeden task (MUTual EXclusion).

- Take — dekrementuje semafor. pokud to nelze, vlákno se uspí
- Give — inkrementuje semafor. program okamžitě pokračuje

Firmware EMBO využívá 5 tasků (viz. Tabulka 4.12), 2 tasky jsou aperiodické (dopočítání post-triggeru a komunikace) a 3 tasky jsou periodické (watchdog, kontrola auto-triggeru a čítač). Komunikační task má největší zásobník, proto slouží také jako inicializační task. Ostatní tasky čekají, až je dokončena inicializace, pak se spustí. Aktivace aperiodických tasků probíhá z přerušení za pomocí semaforů. Některé tasky potřebují exkluzivní přístup ke kritické sekci, což je zajištěno pomocí mutexu.

Kromě velikosti zásobníku definuje každý task jeho priorita, a u periodických perioda. Priorita tasků ve FreeRTOS se nastavuje přesně opačně než priorita přerušení v jádrech ARM.

Čím vyšší priorita FreeRTOS tasku, tím vyšší číslo. Při určování priorit se postupuje tak, že nejkratší tasky dostanou nejvyšší prioritu. Vysokou prioritu také potřebují aperiodické (sporadické) tasky, které většinou navazují na přerušení.

Task 1 — wd

Tato úloha slouží jako podpůrný task, který každých 10 ms resetuje watchdog. Watchdog je bezpečnostní prvek v mikrořadičích, založený na čítači, který běží nezávisle na jádře. Pokud přeteče, resetuje celý kontrolér. Dále tato úloha řeší signalizaci LED:

- Krátké bliknutí LED (50 ms) — při každé odpovědi na dotaz nebo příkaz
- Dlouhé bliknutí LED (500 ms) — 3x signalizuje úspěšnou inicializaci

Task 2 — trig_check

Task č. 2 s periodou 5 ms má 3 základní úkoly:

- Počítání pre-triggeru — kontroluje dostatečný počet vzorků pro pre-trigger
- Detekce Auto-triggeru — detekuje, jestli vypršel timeout pro Auto-trigger
- Asynchronní odpověď Ready — data osciloskopu a logického analyzátoru připravena

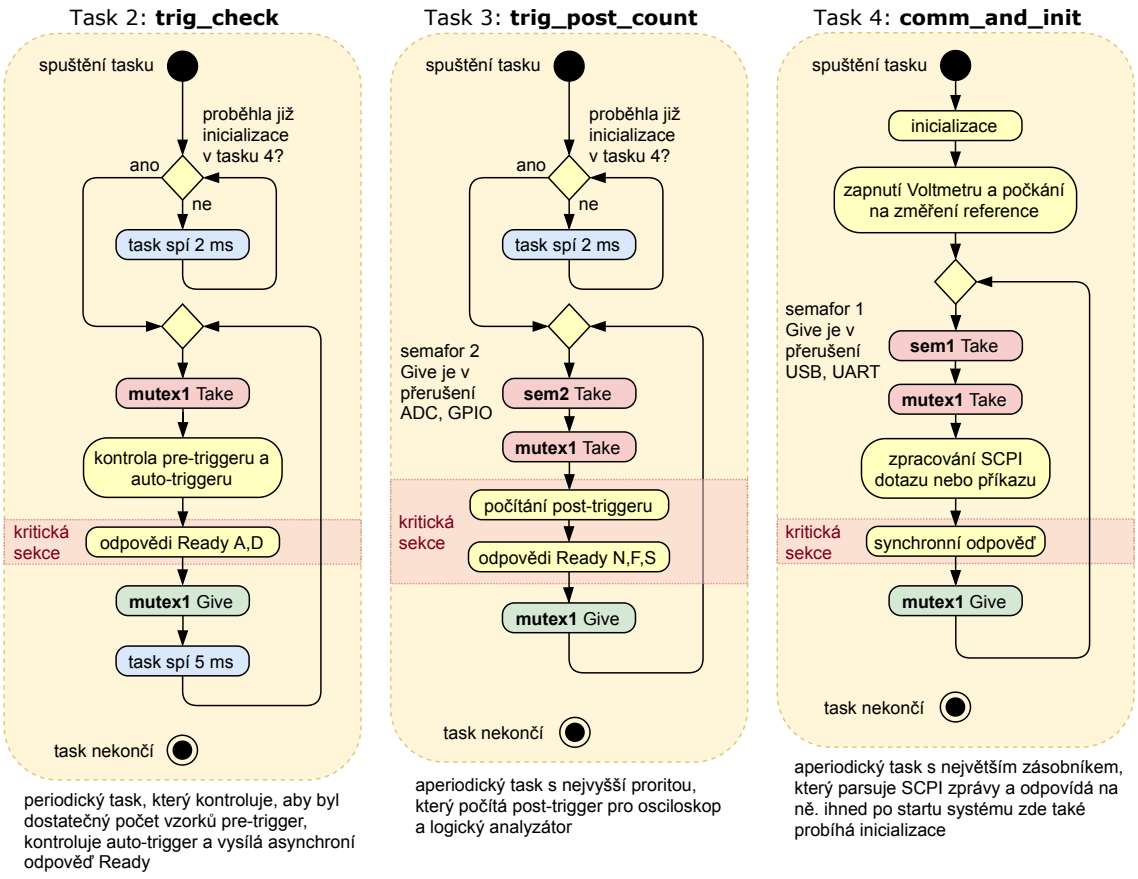
Počítání pre-triggeru zajišťuje, aby když se povolí trigger, byl v paměti již dostatečný počet vzorků. Pre-trigger má smysl jen u osciloskopu a logického analyzátoru. Na rozdíl od post-triggeru, který musí být napočítaný na vzorek přesně, u pre-triggeru toto není třeba. Je nutné pouze zajistit, počet naměřených vzorků pro pre-trigger byl větší nebo rovno konkrétní hodnotě. Toto zjednodušení ušetří jeden časovač a nemá žádná vliv na funkcionalitu.

Další funkcí tohoto tasku je kontrola auto-triggeru. Auto-trigger je základní režim spouštění u téměř všech osciloskopů. Oproti režimu Normal, kdy se čeká na hranu nekonečně dlouhou dobu, v režimu Auto je časovač, který po uplynutí jisté doby vynutí automatický trigger. Uživatel tak na obrazovce vždy vidí signál, i když nebyla splněna spouštěcí podmínka. Pokud tento task zjistí, že je zapnutý režim Auto, a vypršel časový limit pro trigger (zhruba 500 ms), vynutí automatický trigger. Dále také řeší situaci, když je trigger vypnutý.

Task odesílá tyto asynchronní odpovědi:

- ReadyA — připravena data v režimu Auto trigger
- ReadyD — připravena data bez aktivního spouštění

I když jsou tyto odpovědi asynchronní vůči komunikačnímu toku (dotaz → odpověď), musí být synchronní vůči ostatním zprávám. Nesmí se stát, že odesílání jiné zprávy bude uprostřed přerušeno tímto taskem. Tomu by šlo zamezit tak, že komunikační task dostane vyšší prioritu. Jelikož je ale tento task mnohonásobně kratší, potřebuje on vyšší prioritu. Byl proto implementován mutex pro kritickou sekci. Tím se zaručí, že se oba tasky vzájemně nikdy nepřeruší. Kritická sekce je modul comm, ke kterému přistupuje Task 2 a Task 4.



obr 4.5: Diagram aktivit pro tasky 2, 3 a 4

Task 3 — trig_post_count

Task 3 je aperiodický a velmi kritický task s nejvyšší prioritou. Jeho jediná úloha je správně napočítat post-trigger u osciloskopu a logického analyzátoru. Je aktivován z přerušení periferie GPIO nebo z periferie ADC funkcí Analog Watchdog:

- Aktivace semaforem z přerušení GPIO — logický analyzátor
- Aktivace semaforem z přerušení ADC (Analog Watchdog) — osciloskop

Po prvním spuštění se tento task ihned pokusí vzít semafor (Take). Jelikož zatím nebyla splněna spouštěcí podmínka, uspí se. Následuje manipulace s nastavením, spuštění aplikace uživatelem a začátku měření. Pokud přijde přerušení z ADC nebo GPIO a jsou splněny další podmínky, je provedena inkrementace semaforu (Give). Tím je odblokován tento task a protože má nejvyšší prioritu začne ihned počítat post-trigger. Během této doby nemůže být ničím přerušený.

Velice záleží na tom, aby po dopočítání délky bufferu ihned vypnul vzorkování. Paměť je v tomto projektu omezená a není prostor pro větší rezervu. Pokud by se s počítáním opozdil, došlo by k přepsání dat v kruhovém bufferu. Pro vysoké vzorkovací frekvence u logického analyzátoru může dojít ke zpoždění o několik vzorků z důvodu latence přerušení. Tomu se nedá zabránit a musí se to kompenzovat v SW PC aplikace.

Po ukončení je signalizováno předchozímu tasku 2 aby odeslal odpověď Ready. Task 3 se opět pokusí vzít semafor, ale ten už je prázdný. Task se tedy uspí a čeká na další počítání triggeru. Algoritmy detekce a počítání triggeru jsou popsány v kapitole 4.2.2.

Task odesílá tyto asynchronní odpovědi:

- ReadyN — připravena data v režimu Normal trigger
- ReadyS — jako Normal trigger, akorát pouze 1x (Single)
- ReadyF — trigger vynucen uživatelem

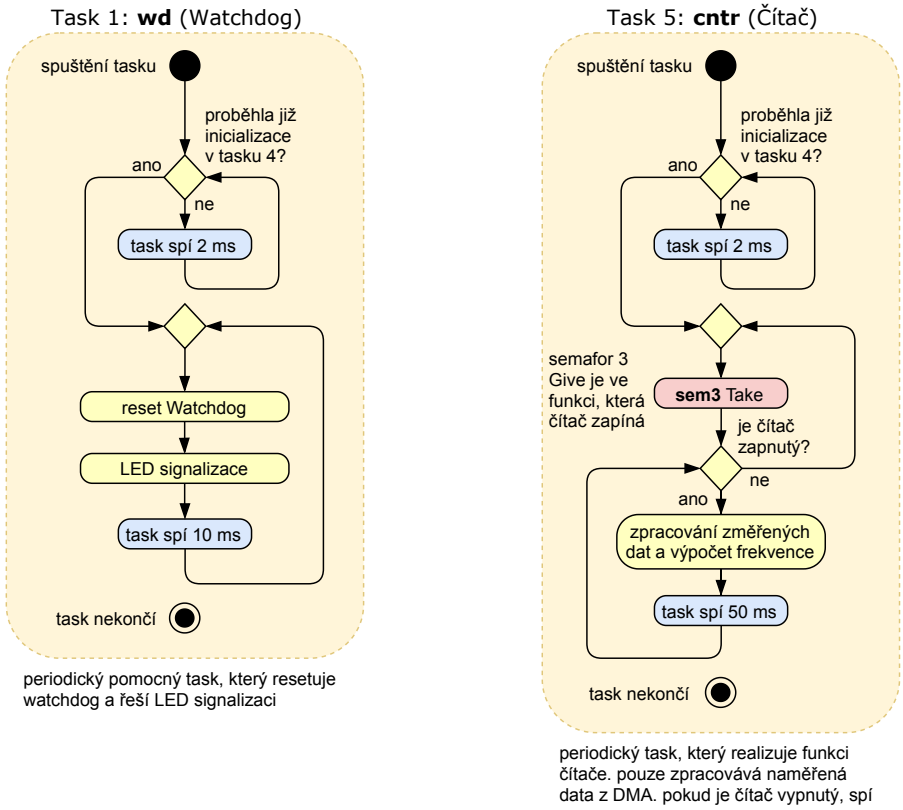
Task 4 — comm_and_init

Task 4 je aperiodický a slouží pro hlavní komunikaci. Jelikož je to task s největší cyklomatickou složitostí, potřebuje největší zásobník. Toho je využito pro inicializaci. Složitost je způsobena jednak prací s textem a jednak tím, že většina firmware je implementována tímto taskem.

Inicializace periferií probíhá v automaticky generovaném kódu z CubeMX s minimálním využití zásobníku. Inicializace aplikace je ale složitější. Možností je vytvořit pro ni jednorázový task, tím by se ale zbytečně plýtvalo pamětí. Proto je zvolena technika, kdy se veškerá inicializace vloží na začátek tasku s největším zásobníkem. Dokud probíhá inicializace, ostatní tasky spí.

Hlavní funkcí tasku je komunikace, tedy zpracování dotazu nebo příkazu a okamžité odeslání odpovědi. Používá se stejný princip aktivace semaforem jako u tasku 3. Periferie UART a USB zapisují příchozí data do bufferu v přerušení. Pokud přijde sekvence znaků <CR><LF>, tak je inkrementován semafor (Give) a task zpracuje zprávu. Poté je buffer vymazán, task se uspí a čeká se na další zprávu. Data z periferie UART mají vždy přednost před USB.

- Aktivace semaforem z přerušení UART
- Aktivace semaforem z přerušení USB



obr 4.6: Diagram aktivit pro tasky 1 a 5

Task 5 — cntr

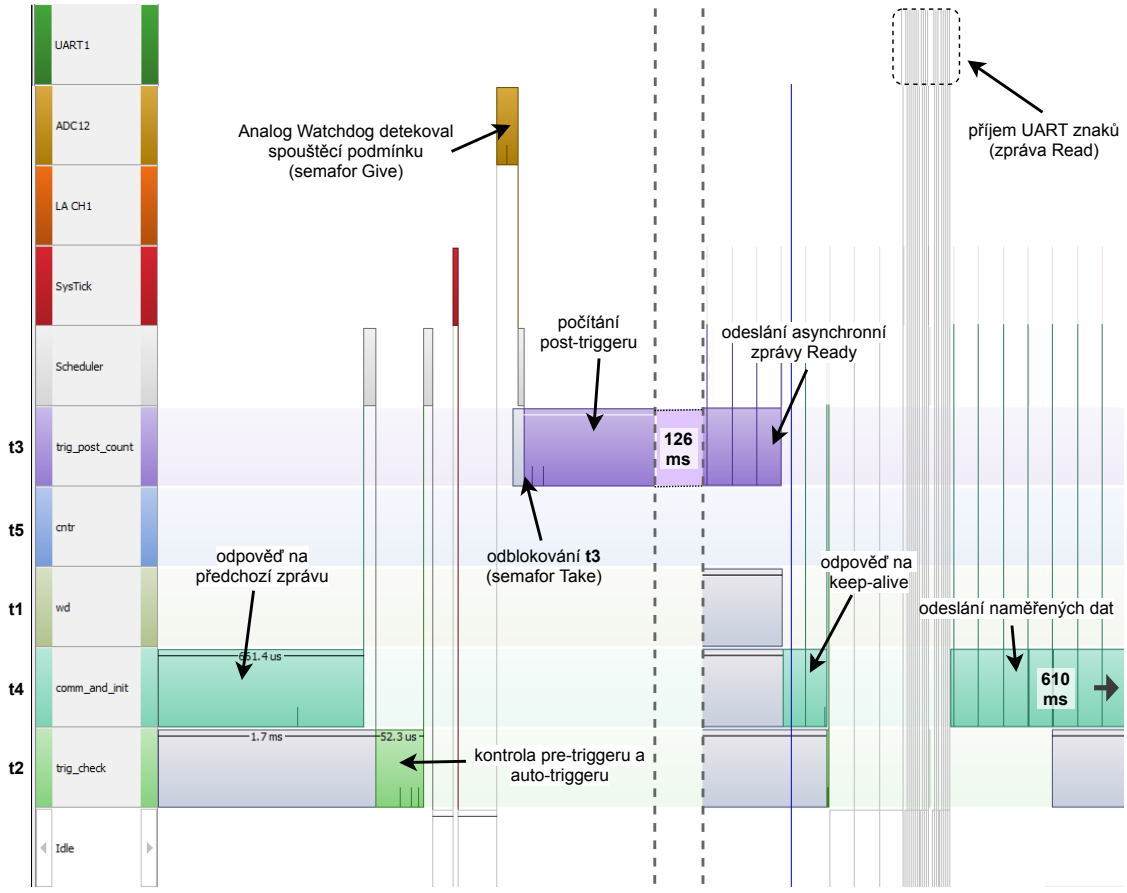
Poslední task je periodický s periodou 50 ms a jeho funkcí je čítač. Specialitou tasku je, že se dokáže uspat, pokud je čítač vypnutý. Tím se uvolní prostředky. Task se může nacházet ve dvou stavech:

- Task je aktivní (s periodou 50 ms)
- Task spí

Přechod mezi aktivním periodickým taskem a spícím taskem je realizován opět stejným principem s použitím semaforu. V tomto případě je semafor inkrementován (Give) z komunikačního tasku 4, jako výsledek uživatelské akce (zapnutí čítače). Následuje periodické zpracování měřených dat, ze kterých se počítá frekvence a perioda. Tyto hodnoty pak uživatel vyčítá komunikačním taskem. Implementace čítače je popsána v kapitole 4.2.2.

Grafické zobrazení tasků a událostí pomocí nástroje SEGGER SystemView

Při vývoji netriviálního firmware se často hodí zachytit a graficky zobrazit události v systému. Událostí je myšleno přerušení, změna kontextu, přepínání tasku nebo operace se semafory. Krokování může odhalit mnoho problémů, ale nevyrovná se grafickému přehledu všech událostí. Jelikož je použit pro ladění SEGGER J-Link (viz. kapitola 3.1.2), může se pro analýzu událostí využít nástroj SEGGER SystemView (viz. obrázek 4.7). Jen je nutné korektně integrovat knihovnu SystemView se zdrojovými kódy FreeRTOS.



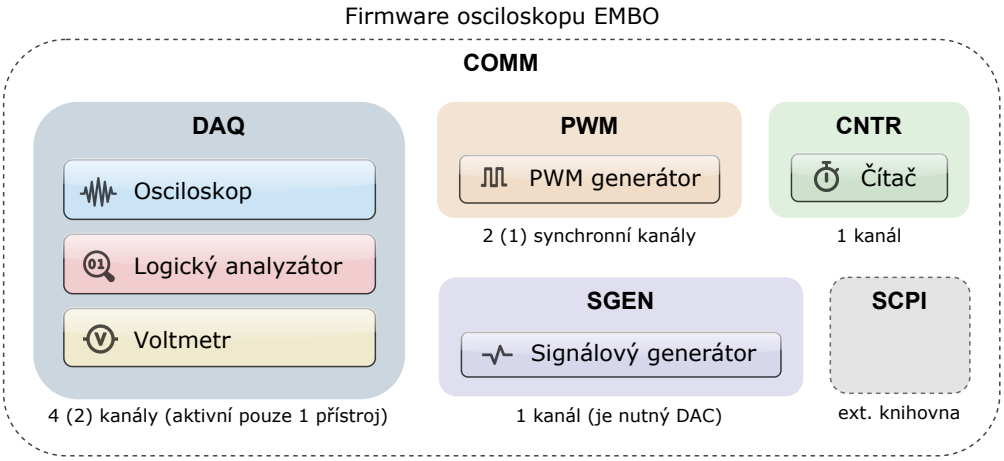
obr 4.7: Grafické zobrazení FreeRTOS tasků a událostí pomocí knihovny SystemView

4.2.2 Implementace firmware modulů

Firmware je rozdělen do několika modulů. Ty spolu nesouvisejí, kromě jediné výjimky, kterou je komunikační modul COMM. Modul se nechová v tomto případě jako *plugin*. Je to kód se společnou funkcionalitou a daty, ideálně nezávislý na ostatních. Modul lze typicky zkopírovat do jiného projektu a jednoduše použít. Pokud by se používalo C++, hovořil bych o třídě. Příslušnost jednotlivých modulů a přístrojů je znázorněna na obrázku 4.8.

Rozdělení na moduly:

- **COMM** — USB-UART komunikace a SCPI zprávy
- **DAQ** — 4(2)-kanálový osciloskop, logický analyzátor a voltmetr
- **PWM** — synchronní 2(1)-kanálový PWM generátor
- **CNTR** — reciproční 1-kanálový čítač
- **SGEN** — 1-kanálový signálový generátor (pouze když je přítomný DAC)



obr 4.8: Blokový popis firmware modulů a jejich propojení

Zapouzdření dat v modulech

I když byl zvolen jazyk C, je při vývoji důležité dodržovat základní OOP principy, protože ty platí univerzálně. Především je využita enkapsulace, tedy zapouzdření. Každý modul, tedy kód se společnou funkcionalitou, má určenou svou datovou strukturu. Každá funkce modulu má jako první argument ukazatel na svou strukturu. Jde o analogii tříd v jazyce C++, které jsou jen *syntaktický cukr*. Kompilátor C++ ve skutečnosti vkládá metodám jako první argument

referenci na jejich datovou strukturu - třídu. Pro polymorfismus ani dědičnost jsem nenašel využití, snažil jsem se o maximální přehlednost kódu. Příklad EMBO modulu je uveden na 4.1.

```
/**** pwm.h ****/

typedef struct /* data modulu - analogie private sekce v C++ class */
{
    uint8_t enabled;
    uint32_t freq;
    uint8_t duty;
}pwm_data_t;

/**** pwm.c ****/

/* konstruktor modulu */
void pwm_init(pwm_data_t* self)
{
    self->enabled = EM_FALSE;
    self->freq = 1000;
    self->duty = 50;
}

 /* metoda modulu */
int pwm_set(pwm_data_t* self, uint32_t freq, uint8_t duty, uint8_t enable)
{
    self->enabled = enable;
    self->freq = freq;
    self->duty = duty;
    /* SET TIMERS ... */
}
```

Ukázky kódu 4.1: Ukázka EMBO firmware modulu PWM se zapouzdřením dat

Implementace modulu COMM — komunikace

Komunikační modul je rozdělen do 3 hlavních bloků:

- comm.c — inicializace, hlavní funkce pro čtení a odpověď
- comm_proto.c — implementace EMBO SCPI protokolu
- comm_irq.c — obsluha přerušení pro USB a UART pro příjem zpráv

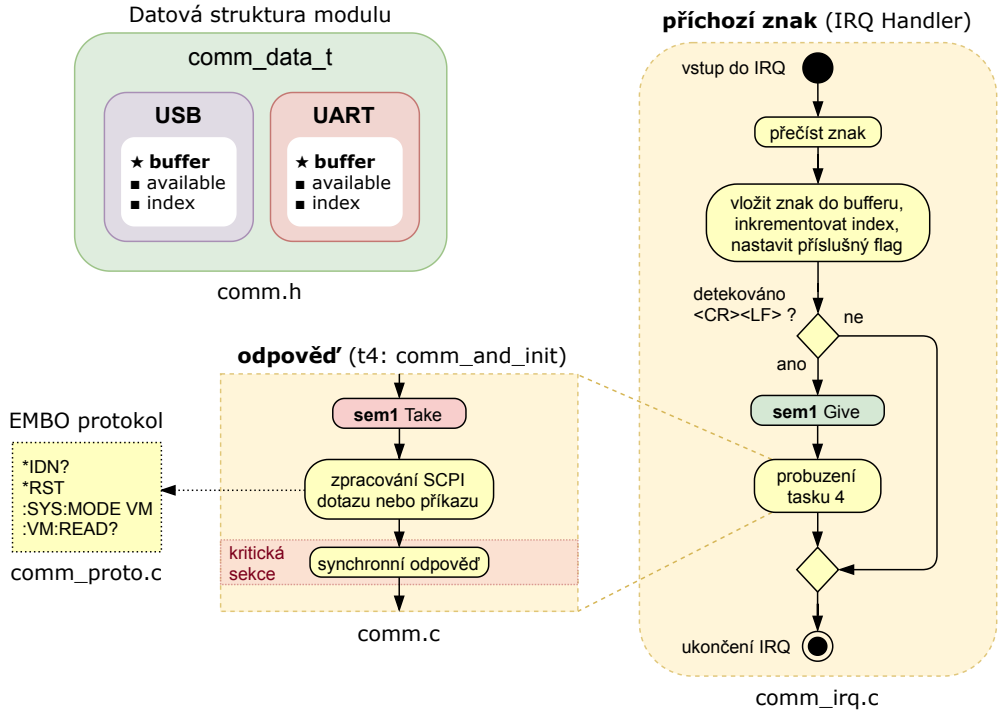
Události začínají nejprve příjmem ASCII znaků na periferii UART nebo USB. Pro příjem i odesílání je zvoleno přímé zpracování dat v přerušení místo DMA, a to z několika důvodů. Pro příjem je potřeba kontrolovat každý znak a detekovat sekvenci <CR><LF>. Navíc rychlost 115200 bps je relativně pomalá, takže se procesor příliš nezatěžuje. Pro odeslání by se mohlo zdát vhodné použít DMA, protože se posílá mnoho dat. Na druhou stranu je to ale zbytečné, protože systém stejně čeká, až se data odešlou. Až pak se může začít měřit znovu.

Navíc je DMA nutné pro modul DAQ, SGEN a CNTR, takže v tomto případě by to bylo kontraproduktivní.

V přerušení UART a USB je použit stejný kód, akorát s jiným bufferem pro příjem. Ten je omezen na 200 ASCII znaků. Buffer pro odesílání není potřeba, používají se datové struktury z jiných modulů.

Každý přijatý znak je přidán do vstupního bufferu a zkontrolován. Pokud je detekována sekvence <CR><LF>, je inkrementován semafor 1 (Give) a plánovač okamžitě odblokuje task 4, který se začne vykonávat. Je zvolena aktivní periferie (UART nebo USB) a příslušný vstupní buffer je předán SCPI knihovně. Po zpracování se buffer resetuje, task 4 se pokusí vzít prázdný semafor 1 (Take) a tím se uspí. Nyní se čeká na další zprávu, resp. sekvenci <CR><LF>.

Zpracování SCPI zprávy probíhá ve dvou fázích. Nejprve se zpracuje celá zpráva, rozloží se na pod-zprávy, které se validují. Pokud je vše v pořádku, SCPI knihovna postupně volá příslušné callbacky jednotlivých funkcí protokolu. Ty jsou umístěny zvlášť v souboru comm_proto.c. Tyto funkce jsou navázány na ostatní moduly a řeší většinu funkcionality celého systému. Proto má task 4 největší zásobník. Implementace komunikačního modulu je shrnuta na obrázku 4.9.



obr 4.9: Implementace modulu COMM (komunikace)

Implementace modulu DAQ — voltmetr, osciloskop a logický analyzátor

DAQ (Data Acquisition) modul je nejdůležitější a nejsložitější součást systému. Skládá se z těchto hlavních bloků:

- daq.c — inicializace a hlavní implementace
- daq_trig.c — část implementace řešící pouze trigger
- daq_irq.c — obsluha přerušení pro ADC (Analog Watchdog) a GPIO EXTI

DAQ kombinuje funkcionalitu osciloskopu, logického analyzátoru a voltmetru. I když vstupní data přichází na jiné periferie (ADC nebo GPIO) a mají jiný charakter (analogové nebo digitální), podstatná část funkcionality se prolíná. Stejný kód je použit pro inicializaci, nastavení (paměť, vzorkovací frekvence) nebo počítání post-triggeru. Ostatní kód je parametrizovaný a obsluhy přerušení jsou specifické.

- Osciloskop - DAQ v režimu ADC, plně nastavitelný
- Voltmetr — DAQ v režimu ADC, s pevnými parametry ($f_s = 100$ Hz)
- Logický analyzátor — DAQ v režimu GPIO

Popis modulu DAQ lze rozdělit do několika hlavních bloků. Nejprve se konfigurují periferie a interní datové bloky v SRAM. Po konfiguraci se modul zapne. Pokud je detekován validní trigger, je spuštěn task s nejvyšší prioritou, který dopočítá post-trigger, modul vypne a odešle asynchronní zprávu Ready. Nakonec se data vyčtou synchronní zprávou Read a modul se opět zapne.

1. Konfigurace → zapnutí modulu
2. Detekce triggeru → dopočítání post-triggeru → vypnutí modulu
3. Odeslání naměřených hodnot → zapnutí modulu → pokračování 2.

1. Konfigurace modulu DAQ.
Základním interním blokem modulu DAQ je časovač vzorkování. Ten se nastaví na nejbližší možnou frekvenci od požadované frekvence pomocí vzorce 4.1. Prescaler nastavuje děličku taktovací frekvence časovače, Reload je horní strop čítání časovačel, f_{osc} je taktovací frekvence a f_{want} je požadovaná frekvence časovače ((4.1)).

$$(Prescaler + 1) \cdot (Reload + 1) = \frac{f_{osc}}{f_{want}} \tag{4.1}$$

Je to rovnice o dvou neznámých, která lze řešit jednoduchým cyklem se složitostí menší než O(N). Funkce, která toto řeší, lze vidět v kódu 4.3 a je součástí EMBO firmware.

```
/*
 * vstupy:  max_reload (65535), f_osc (72000000), f_want (123456)
 * vystupy: prescaler (0), reload (582)
 * return:  realna nejblizsi frekvence, ktera sla nastavit (123499.142)
 */
double get_freq(int* prescaler, int* reload, int max_reload, int f_osc, int f_want)
{
    ASSERT(f_osc >= f_want && f_want > 0 && f_osc > 0 && max_reload > 0);
    *prescaler = 0;

    do
    {
        *reload = (int)((double)freq_osc / (*prescaler + 1) / freq_want) - 1;
        if (*reload > max_reload)
            (*prescaler)++;
    }
    while (*reload > max_reload);

    return freq_osc / ((double)(*prescaler + 1) * (*reload + 1));
}
```

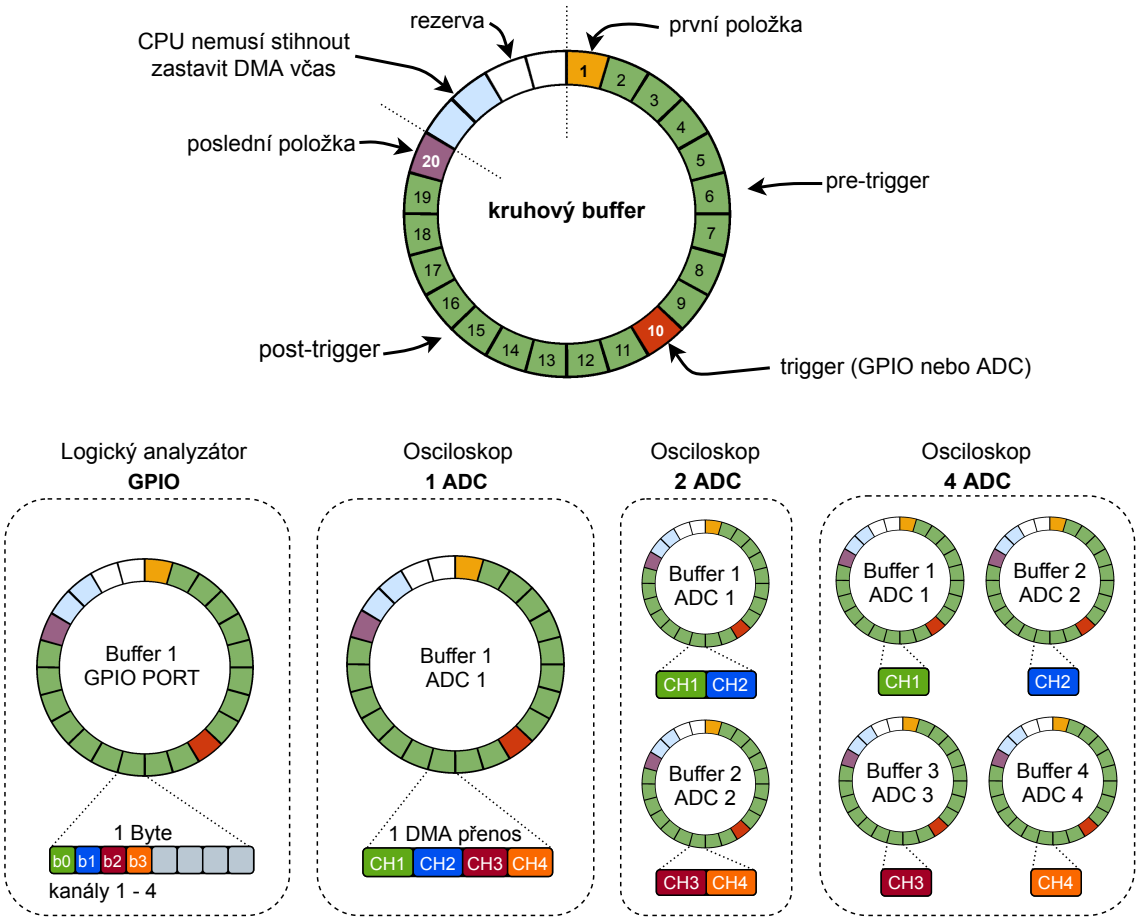
Ukázky kódu 4.2: Pomocná funkce z EMBO firmware pro výpočet Prescaler a Autoreload časovače

Událost ze vzorkovacího časovače je přivedena na příslušnou periferii — ADC nebo GPIO. Na této periferii je zapnutý režim DMA. Pro ADC i GPIO je nakonfigurován DMA kanál v režimu cirkulární buffer. Jeho velikost je o něco větší než hloubka paměti přístroje. Tzv. rezerva je tam kvůli vysokým rychlostem, při nichž by mohlo docházet k přepisu dat na začátku v kruhovém bufferu. Přidělení paměti kruhovým bufferům je podmíněno typem řadiče, resp. počtem ADC s DMA:

- Logický analyzátor – 1 GPIO port → 1 buffer → 4 kanály
- Osciloskop – 1 ADC (STM32F103) → 1 buffer → 4 kanály
- Osciloskop – 2 ADC (STM32L412) → 2 buffery → 2 + 2 kanály
- Osciloskop – 4 ADC (STM32F303) → 4 buffery → 1 + 1 + 1 + 1 kanál

Paměť je přidělená staticky z jednoho velkého bufferu. Jelikož je zakázaná dynamická alokace, je nutné použití speciální funkci *daq_malloc*, která přidělí v závislosti na množství ADC a zapnutých kanálech paměť pro kruhový buffer. Tato funkce musí rozlišovat bitovou velikost jednoho vzorku, která v případě ADC může být 8 nebo 12 bitů. Pokud je zvoleno 12 bitů, jsou alokovány 2 byty. Pokud jsou dostupné 4 ADC, nehraje počet zapnutých kanálů roli. Pokud je dostupné pouze 1 ADC, je maximální velikost bufferu přímo závislá na počtu zapnutých kanálů (Blue Pill).

Kromě vzorkovací frekvence a paměti je pro ADC nutné natavit dobu vzorkování (viz. obrázek 3.5. Ideální by bylo mít dobu vzorkování vždy nejdelší, aby nebylo problém vzorkovat



obr 4.10: Popis kruhového buffer v modulu DAQ v závislosti na režimu a mikrořadiči

měkké zdroje signálu. Se vzrůstající vzorkovací frekvencí se ale zmenšuje dostupný časový úsek pro dobu vzorkování. Nejlepším řešením je dynamicky volit dobu vzorkování tak, aby byla co nejdelší, ale aby se vešla do vzorkovací periody.

Kromě standardních ADC režimů existují také 2 speciální. V režimu Dual Mode je využito druhé ADC, které nepodporuje DMA. Pro 2 zapnuté kanály je tak docíleno stejné vzorkovací frekvence jako kdyby byl zapnutý pouze 1 kanál. Podobně je to u 4 zapnutých kanálu, které běží s rychlostí jako 2 kanály. V režimu Interleaved pracují 2 nebo 4 ADC střídavě a výsledkem je 1 kanál s 2-násobnou, resp. 4-násobnou vzorkovací frekvencí. Nutno zmínit, že tyto režimy jsou zatím pouze v testovací fázi.

Konfigurace se provádí vždy na základě příkazu SET nebo příkazu SYS:MODE. Pokud byla konfigurace úspěšná, uloží se nastavení a modul se zapne. Pokud konfigurace selhala (např. parametry mimo rozsah), obnoví se předešlé nastavení a modul se taky zapne. Filosofie modulu DAQ je taková, že pokud může být modul zapnutý, tak je vždy zapnutý. Uživatel nezapíná voltmetr, osciloskop ani logický analyzátor na úrovni modulu DAQ. Modul DAQ je vypnutý pouze tehdy, pokud jsou v bufferu validní data, která čekají na odeslání do PC.

```
#define EM_FREQ_ADCCLK      72000000  /* ADC clock */
#define N                   8         /* velikost pole SMPLT */
#define TCONV_12BIT         12.8      /* delka SAR konverze  */
#define ADC_1CH_SMPLTM(T,B) ((1.0 / (double)EM_FREQ_ADCCLK) * ((double)T + (B)))
/* makro vypocet doby vzorkovani pro 1 kanal (T = smpl ticks, B = Tconv) */

/*
 * vstupy:  fs - vzorkovaci frekvence, chans - pocet kanalu v 1 ADC
 * return:  nejdelsi mozna doba vzorkovani
 */
double dynamic_smplt(double fs, double chans)
{
    /* katalogove hodnoty doby vzorkovani pro STM32F103 */
    const double SMPLT[N]  = { 1.5, 7.5, 13.5, 28.5, 41.5, 55.5, 71.5, 239.5};

    double T = 1.0 / fs;        /* vzorkovaci perioda vypoctena z fs */
    double result = SMPLT[0];   /* vysledna doba vzorkovani */

    /* iteruj, dokud se doba vzorkovani vleze do vzorkovaci periody */
    for (int i = 0; i < N; i++)
    {
        if ((chans * ADC_1CH_SMPLTM(SMPLT[i] + 0.5, TCONV_12BIT)) < T)
            result = SMPLT[i];
        else
            break;
    }
    return result;
}
```

Ukázky kódu 4.3: Algoritmus pro dynamické nastavení doby vzorkování STM32 ADC

2. Detekce triggeru a dopočítání post-triggeru.
Modul DAQ je nakonfigurovaný a právě se zapnul. Nyní se musí počkat, dokud nebude v paměti dostatečný počet vzorků odpovídající velikosti pre-trigger. K tomu slouží task 2 s periodou 5 ms. Pro zjednodušení je počet vzorků pre-trigger převeden na čas a ten je porovnáván s hrubým časovačem SysTick s periodou 1 ms. Průměrná přesnost 2.5 ms (definována polovinou periody kontroly) je dostačující, protože se kontroluje podmínka větší nebo rovno, a v porovnání s délkou odesílání zprávy jsou jednotky milisekund zanedbatelné.

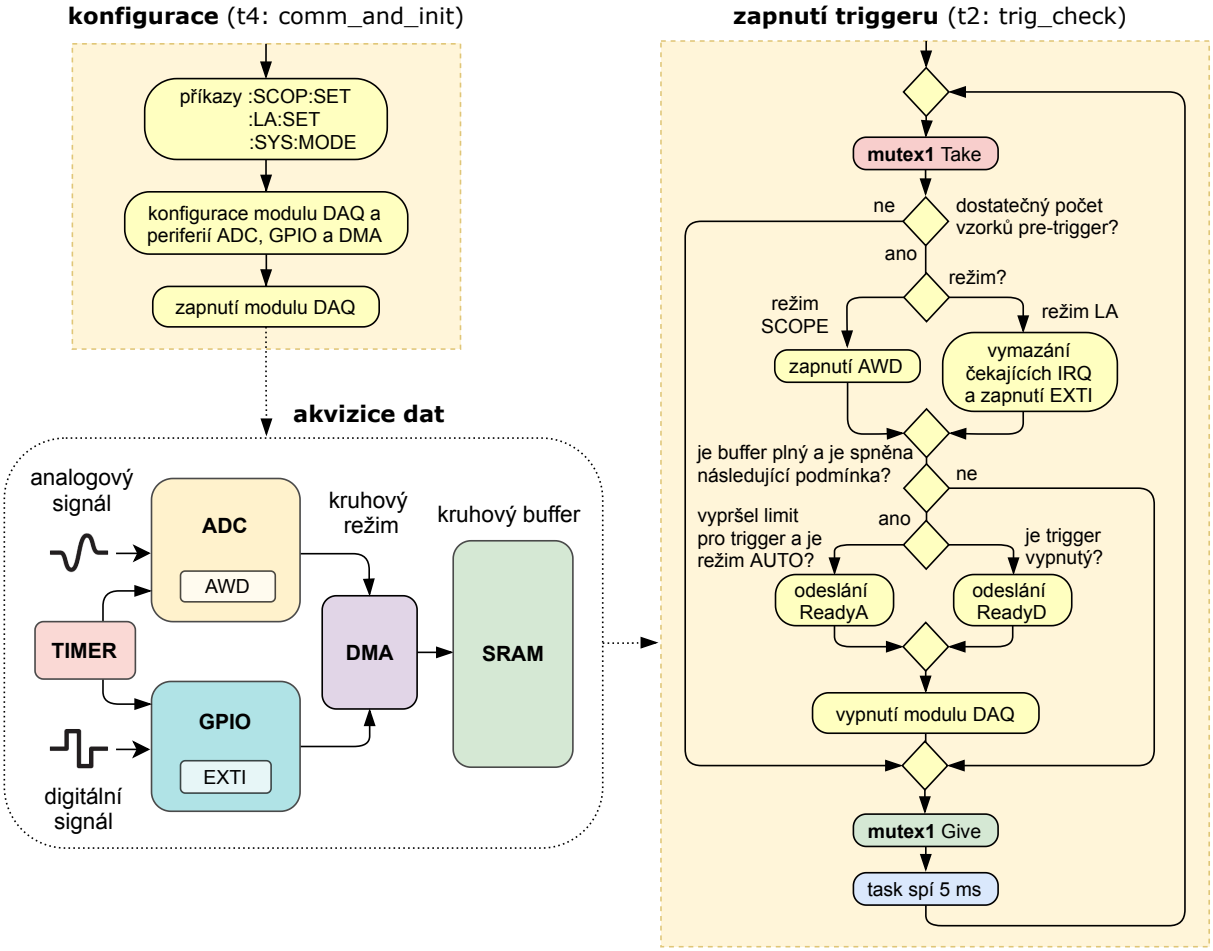
V momentu splnění podmínky dostatečného počtu vzorků pre-trigger je systém připraven na detekci a validaci triggeru. Aktivace se provede v závislosti na aktuálním režimu:

- Logický analyzátor – vymazání čekajících přerušení a aktivace GPIO přerušení
- Osciloskop – zapnutí funkce Analog Watchdog u ADC

Nyní se čeká na trigger. V tasku 2 ale už může celý proces rovnou skončit. To může nastat v těch nejjednodušších případech, kdy trigger vůbec nepřišel, nebo je vypnutý. Podmínkou je samozřejmě dostatečný počet vzorků pre-trigger. Odešle se asynchronní zpráva Ready, modul DAQ se vypne a čeká se na odeslání naměřených dat do PC:

- ReadyA – trigger nebyl detekován v časovém intervalu, tak byl vynucen (režim AUTO)
- ReadyD – trigger je vypnutý (režim DISABLED)

V opačném případě je nutné zpracovat trigger. Detekce začíná vždy v přerušení. V případě logického analyzátoru je to přerušení GPIO EXTI, žádná další validace není nutná a rovnou se probouzí task 3, který dopočítá post-trigger.



obr 4.11: Implementace modulu DAQ (osciloskop, logický analyzátor, voltmetr) — 1. část

Pokud je DAQ v režimu osciloskop, detekce triggeru začíná v přerušení ADC, které vyvolal Analog Watchdog. Analog Watchdog je hardwarová součást ADC, která kontroluje, jestli je měřený signál ve stanoveném rozsahu. Pokud dojde k vybočení z rozsahu, je generováno přerušení. Na rozdíl od GPIO je u ADC nutné provést 2 korekce:

1. Nalezení toho správného indexu vzorku v DMA transakci (více kanálů u 1 ADC)
2. Kontrola spouštěcí podmínky a případné prohození limitů u Analog Watchdog

Tyto 2 korekce jsou velmi důležité. Pokud se neprovedou, bude trigger fungovat nestabilně. Nalezení správného indexu v DMA transakci je nutné proto, že aktuální pozice DMA (která je stěžejní pro nalezení vzorku v bufferu) nemusí odpovídat tomu vzorku, který způsobil přerušení Analog Watchdog. Jinými slovy, díky latenci přerušení a dalším faktorům může při validaci triggeru aktuální vzorek patřit již jinému kanálu. To lze vyřešit jednoduše pomocí operace modulo.

Z podstaty fungování Analog Watchdog je zřejmé, že pokud tuto funkci zapneme v libovolný čas, může nastat situace, kdy bude periodický signál zrovna v zakázaném pásmu. Analog Watchdog tedy bude neustále generovat přerušení, kde při validaci triggeru bude v lepším případě toto rozpoznáno, v horším bude generován falešný trigger. V obou případech by ale byl systém zaplaven spoustou zbytečných přerušení. Řešením je správně tuto situaci detekovat a pokud k ní dojde, prohodit limity Analog Watchdog pro náběžnou a sestupnou hranu. Při dalším přerušení se limity prohodí zpět a další přerušení bude již korektní.

Po detekci a validaci triggeru dochází k probuzení tasku 3 pomocí inkrementování semaforu 2 (Give). Úkol tasku 3 s nejvyšší prioritou je napočítat přesně počet vzorků post-trigger v bufferu a pak ihned vypnout modul DAQ. Tato operace je kritická, protože osciloskop EMBO disponuje omezenou velikostí paměti RAM. Pokud by se ukončila předčasně, chyběly by vzorky na konci signálu. Pokud by se počítání opozdilo, přepsal by se zase začátek signálu.

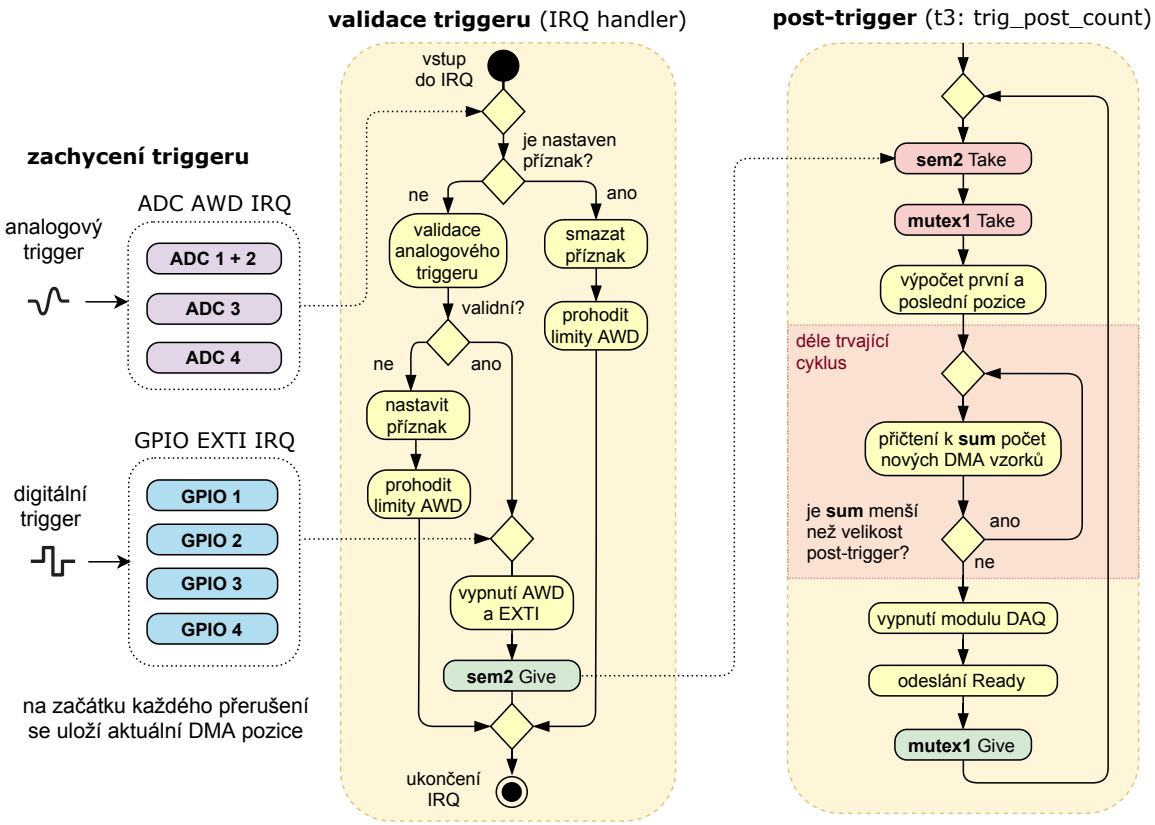
Vstupem do tasku 3 je index vzorku v bufferu, který splnil spouštěcí podmínku, resp. index triggeru. Nejprve se díky znalosti velikosti bufferu a počtu kanálů vypočte index prvního a posledního vzorku. Dále se vypočte přesná velikost vzorků post-trigger. V tomto okamžiku máme vše potřebné pro zahájení počítání ve smyčce *while*.

Vyzkoušel jsem více způsobů počítání post-triggeru a jako nejlepší se ukázalo použít akumulátor. Do proměnné se v každém cyklu přičte počet nově zapsaných vzorků DMA. Tato proměnná se porovnává s velikostí post-trigger. Pokud je větší nebo rovna, modul DAQ se vypne, task 3 se uspí (semafor 2 Take) a odešle se asynchronní zpráva Ready. Systém nyní čeká na zprávu Read pro vyčtení dat.

- ReadyN – trigger byl úspěšně detekován a validován (režim NORMAL nebo AUTO)
- ReadyS – to stejné jako ReadyN, akorát je zapnutý režim SINGLE
- ReadyF – to stejné jako ReadyN, akorát byl trigger vynucen manuálně

Pro vysoké vzorkovací frekvence se může stát, že CPU nestihne kontrolovat přibývající vzorky po jednom. Jelikož se kontroluje post-trigger podmínkou větší nebo rovno, a jelikož je za post-triggerem v bufferu rezerva 10 vzorků na kanál, neměl by to být problém.

Naopak problém může být to, že během počítání post-triggeru systém nemá čas odpovídat na zprávy. Komunikace mezi PC aplikací a osciloskopem je udržována pomocí *keep-alive* zprávy, díky které je detekován timeout (porucha, odpojení). Pro nízké hodnoty vzorkovací frekvence (např. 1 Hz) může trvat napočítat post-trigger velmi dlouho. Systém tak nerozliší, že nastal timeout, nebo že se stále počítá post-trigger. Toto se může vyřešit vhodným omezením vstupních parametrů.



obr 4.12: Implementace modulu DAQ (osciloskop, logický analyzátor, voltmetr) — 2. část

3. Odeslání naměřených hodnot.

Poslední fází je odeslání naměřených dat do PC aplikace. V závislosti na řadě mikrořadiče a počtu ADC se odešle odpovídající množství kruhových bufferů. Zpracování dat se provádí v PC aplikaci, je tak využit řádově vyšší výkon CPU. Po odeslání je modul DAQ opět zapnut a celý proces začíná od začátku.

Existuje výjimka, kdy je potřeba data zpracovat. Při inicializaci modulu DAQ se nejprve na krátkou chvíli zapne režim voltmetr. Počká se na první vzorky a z nich se vypočte hodnota napájecího napětí V_{CC}. Ta představuje referenční napětí pro analogové měření. Slouží pro potřeby ostatních modulů i jako informace pro uživatele. Vypočte se pomocí vzorce níže (4.2).

$$V_{CC} = 3.3 \cdot \frac{VREFINT_{CAL}}{VREFINT_{DATA}} \tag{4.2}$$

$$V_{meas} = V_{CC} \cdot \frac{ADC_{RAW}}{ADC_{MAX}} \tag{4.3}$$

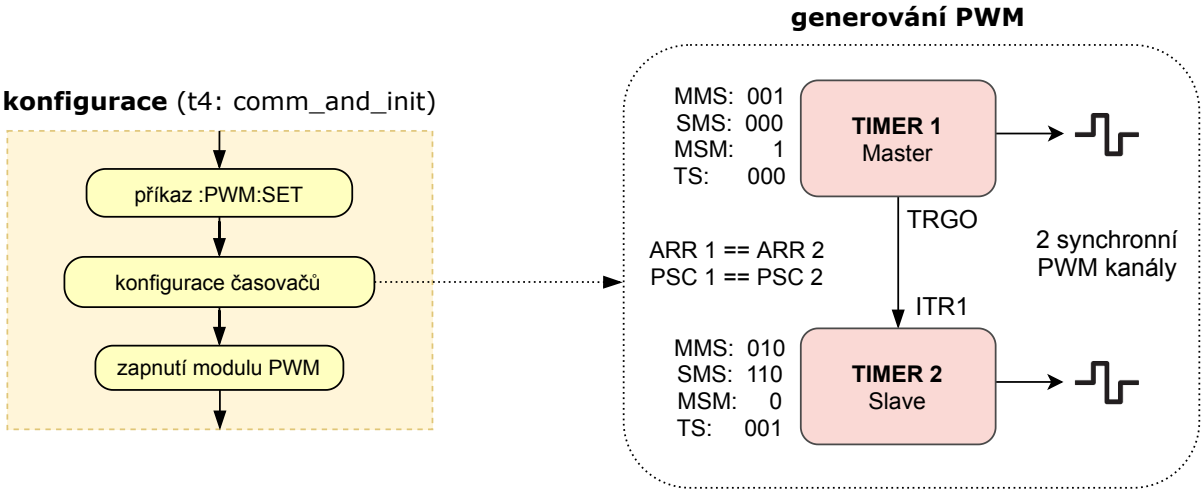
Většina STM32 ADC má interní napěťovou referenci, typicky o hodnotě 1200 mV. Lze číst podobně jako ostatní kanály a její bitová hodnota je VREFINT_DATA. Modernější řady mají továrně kalibrovanou hodnotu této reference uloženou v registru. Starší řady ji postrádají. Její 12-bitová hodnota je zhruba 1490 a ve vzorci je to proměnná VREFINT_CAL.

Implementace modulu PWM — PWM generátor

Synchronní 2-kanálový generátor PWM signálu je jednoduchý modul, který je postaven nad dvěma standardními STM32 časovači. Je u něj možné nastavit společnou frekvenci, střídu na kanál a offset druhého kanálu. Pokud by šlo o jednokanálový generátor, byla by konfigurace triviální. Jelikož ale jde o synchronní generátor, je využita pokročilejší funkce řetězení časovačů.

Princip funkce je takový, že jeden časovač je Master a druhý je Slave. Frekvence je společná, tedy registry PSC a ARR musí být u obou časovačů shodné. Střída (registr CCRx) se nastavuje individuálně. U Master časovače se nastaví trigger výstup (TRGO) na událost zapnutí (CNT_EN). U Slave časovače se zapne Slave režim a nastaví správný trigger vstup (ITR). Oba PWM generátory se zapínají pouze pomocí prvního časovače, druhý se ovládá nastavení kanálu (registr CCER).

Nastavení offsetu nemusí být zrovna intuitivní. Trik spočívá v tom, že se nejprve vypne Master a pak se Slave časovači nastaví čítač (registr CNT) na hodnotu $offset/100 \cdot ARR$. Následně se Master zapne a Slave běží synchronně s definovaným offsetem.



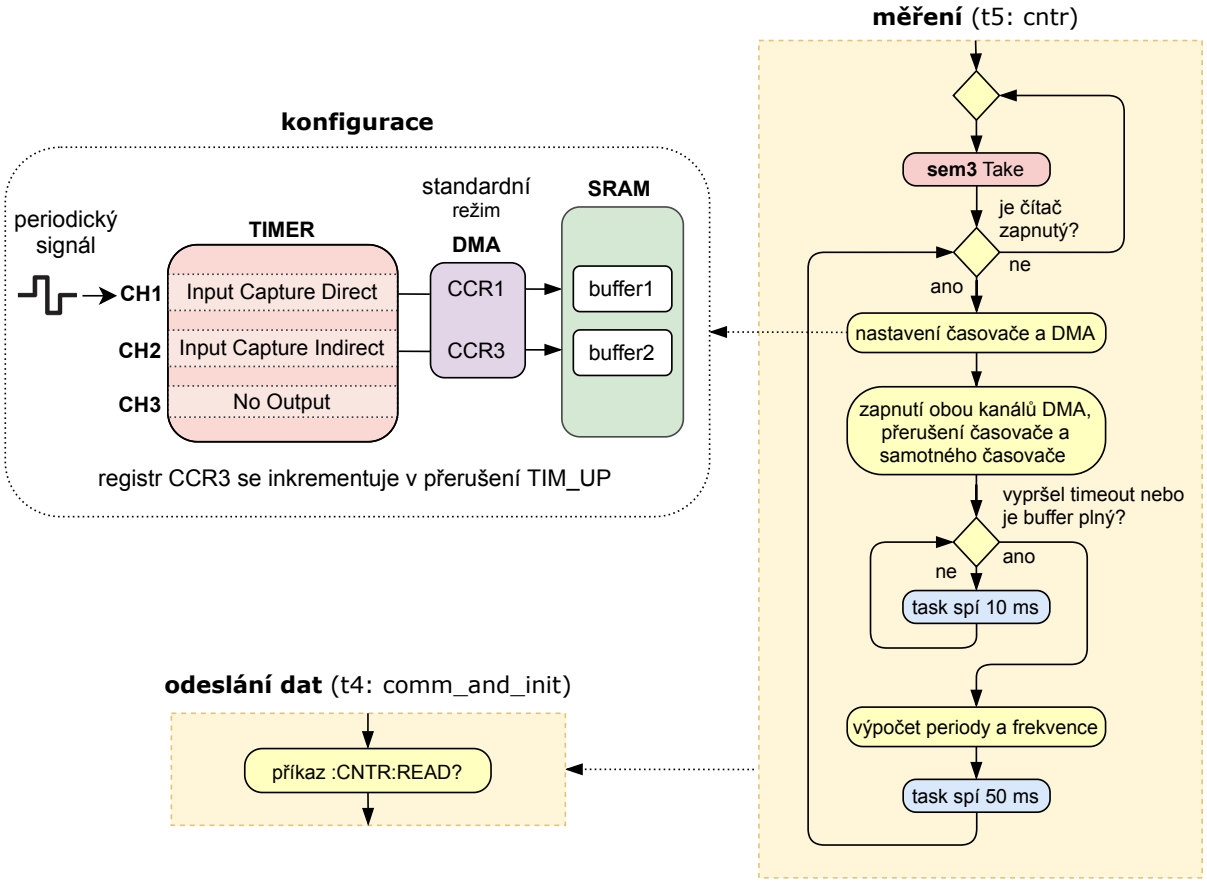
obr 4.13: Implementace modulu PWM (PWM generátor)

Implementace modulu CNTR — čítač

Čítač je jednoduchý modul, který je založen na recipročním principu měření. Princip spočívá v časovači v režimu Input Capture s vysokou frekvencí. Vstupní periodický signál generuje s každou náběžnou hranou DMA transakce, ve kterých se ukládá aktuální hodnota čítače do paměti. Za definovanou dobu nebo po naplnění paměti se čítač vypne. Data se vyhodnotí, zprůměrují a výsledkem je perioda vstupního signálu.

Prvním krokem je konfigurace, která se provádí pomocí SCPI příkazu dle protokolu. Veškerá funkcionalita je realizována jen pomocí jednoho pokročilého STM32 časovače. První kanál je nastaven jako Input Caputre (Direct), druhý jako Input Caputre (Indirect) a třetí pomocný jako Output Compare No Output. U prvního kanálu je zapnut DMA přenos registru CCR1, který představuje aktuální hodnotu čítače. Druhý kanál má nastaven DMA přenos registru CCR3, který představuje softwarově rozšířenou hodnotu pomocí přerušení.

1. Kanál 1. – Direct Input Capture (DMA kanál 1: CCR1 → buffer1)
2. Kanál 2. – Indirect Input Capture (DMA kanál 2: CCR3 → buffer2)
3. Kanál 3. – No Output (slouží jako prostředník pro softwarové rozšíření)



obr 4.14: Implementace modulu CNTR (čítač)

Naprostá většina časovačů STM32 má rozlišení 16 bitů. Pro tento účel by ale byl vhodnější 32 bitový časovač. Problém lze vyřešit pomocí softwarového rozšíření, kdy se nastaví se přerušení na přetečení časovače (TIM_UP). V přerušení se inkrementuje pomocná proměnná, jejíž hodnotu je potřeba nastavit jako zdroj DMA transakce druhého kanálu. Použil jsem pro

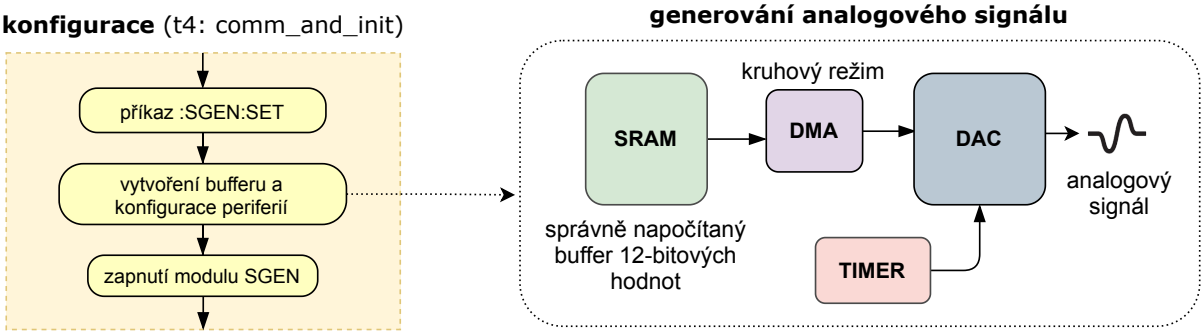
tuto funkci třetí kanál jako prostředníka. V přerušení se proměnná inkrementuje a rovnou zapisuje do CCR3 registru. Druhý DMA kanál má nastaven jako zdroj právě registr CCR3.

Řadič DMA s každou náběžnou hranou vstupního signálu zapisuje do 2 bufferů vždy aktuální hodnotu čítače a aktuální počet přetečení. Po uplynutí timeoutu (2 sekundy) nebo po naplnění bufferů proběhne výpočet a čítač se na 50 ms uspí. Toto se děje kontinuálně ve vlastním tasku, dokud je čítač zapnutý. Odeslání hodnot probíhá nezávisle v komunikačním tasku 4.

Když pro nízkofrekvenční vstupní signál je velikost bufferu x, tak pro vyšší frekvence (MHz) je dobré mít velikost bufferu zhruba $10x$. Dále je dobré pro vyšší frekvence zapnout v časovači na vstupu předděličku, aby DMA řadič stíhal přenášet data. Tyto požadavky jsou zohledněny pomocí režimu Slow a Fast. Režim Slow má velikost bufferu 30 hodnot. Režim Fast má buffer veliký 200 hodnot a navíc má na vstupu děličku s poměrem 1:8. Režim Fast je vhodný pro vstupní signál s frekvencí v řádu MHz.

Implementace modulu SGEN — signálový generátor

Signálový generátor je doplňkový modul, který je přítomný pouze pokud MCU disponuje periferií DAC. Většina DAC převodníků u mikrořadičů STM32 má rozlišení 12 bitů. Konfigurace tedy spočívá ve vygenerování bufferu, ve kterém budou 12-bitové hodnoty odpovídající zvolené funkci, frekvenci a amplitudě. Toto generování probíhá pro jednoduché funkce přímo v MCU. Pro složitější tvary specifikované uživatelem je nejlepší vygenerovat buffer v PC a zaslat ho do generátoru. Tato funkcionalita bude časem doplněna.



obr 4.15: Implementace modulu SGEN (signálový generátor)

Pro generování stačí kromě DAC nastavit časovač a DMA. Časovač definuje frekvenci, s kterou DAC čte a převádí vzorky z paměti na analogový signál. Řadič DMA je v tomto případě nutný, protože signálový generátor je jen doplňková funkcionalita osciloskopu, takže generování musí probíhat nezávisle. Navíc generátor bez DMA je v podstatě nepoužitelný, resp. velice omezený. Na druhou stranu, pro vysoké vzorkovací frekvence osciloskopu a signálového generátoru může docházet k vypadávání vzorků, protože DMA sběrnice je plně vytížena.

4.3 Fáze 3 — Vývoj Qt aplikace

Pro vývoj PC aplikace bylo zvoleno oficiální IDE jménem Qt Creator. Tento nástroj je multiplatformní, pro každou verzi osciloskopu EMBO pro různé systémy tak bude nutné zkompilovat aplikaci na konkrétním operačním systému. Tento postup je nutný ve verzi Qt Open Source. Pokud by se použily placené nástroje, šlo by *cross kompilovat* z jednoho systému, případně by šlo použít službu CI/CD (Continuous Integration / Continuous Delivery).

Hlavní okno aplikace bude sloužit jako rozcestník. Uživatel si v něm zvolí COM port a připojí se k osciloskopu. Podle typu MCU bude mít na výběr ze 3 primárních přístrojů a ze 3 sekundárních přístrojů. Každý přístroj bude ve vlastním okně. Dohromady bude mít aplikace 7 hlavních oken. Podpůrné a doplňkové služby budou řešeny vyskakovacími okny a dialogy.

Architektura PC aplikace osciloskopu EMBO není založena na populárním principu MVC nebo MVVM, protože je to první Qt aplikace, kterou píšu. Pokud bych to psal v .NET a C#, zvolil bych například MVVM framework Caliburn, protože s ním již mám zkušenosti. Odstínila by se tak logika od GUI, a kód by byl přehlednější. V tomto případě bohužel musím postupovat tím nejjednodušším způsobem.

Následující kapitoly budou rozděleny do několika bloků. Nejprve představím třídní diagram, který zběžně ukáže hlavní rysy aplikace. Dále detailně popíšu třídu Core, který slouží jako komunikační jádro a která řídí celou aplikaci. Následně se budu věnovat grafickému návrhu, doplňkovým funkcím a integraci se s systémem a knihovnami.

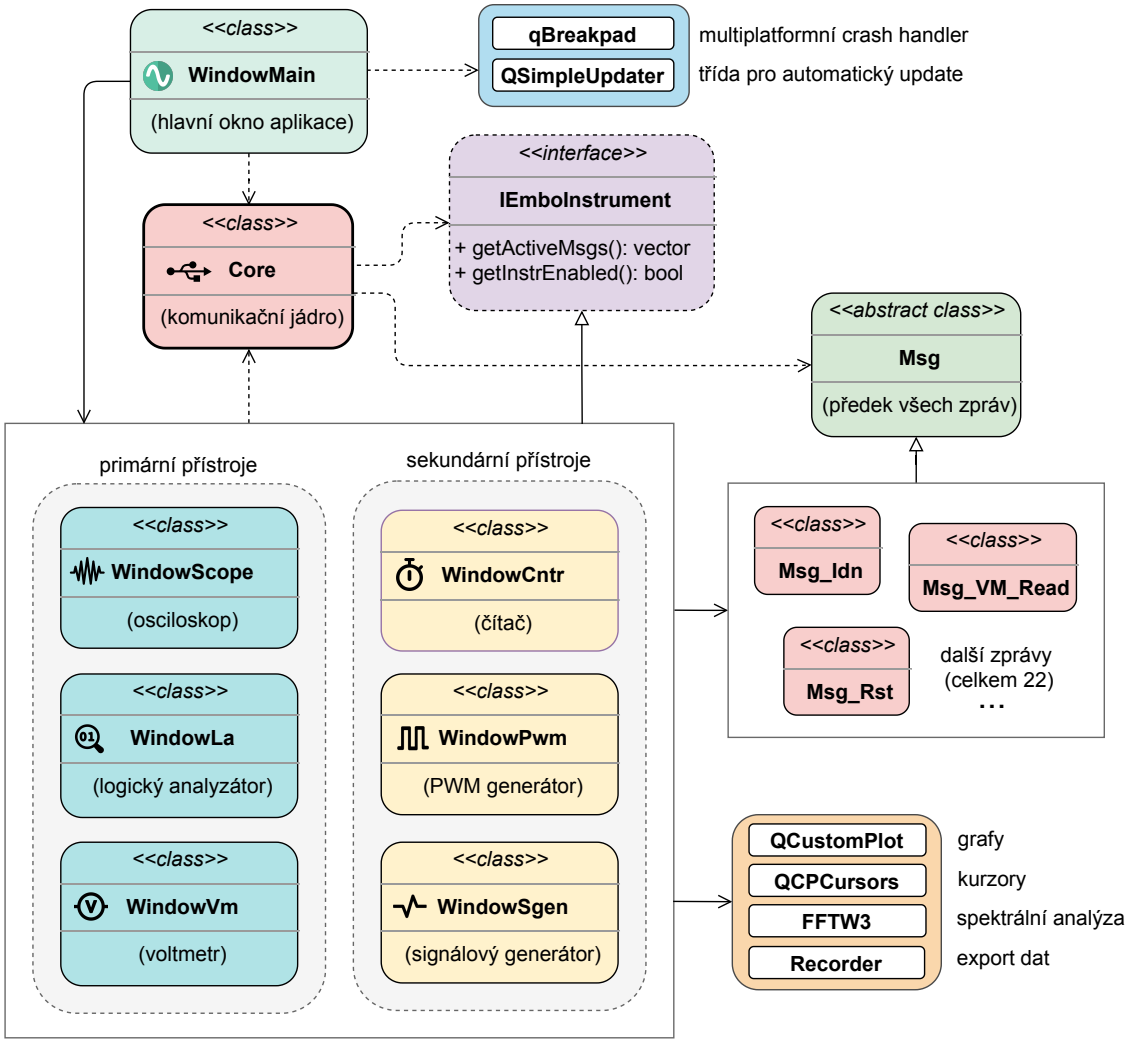
- Třídní diagram – obecný popis implementace
- Třída Core – komunikační jádro aplikace
- Grafický návrh – vlastní CSS design, logo, fonty a ikony
- Interpolace – důvody použití Spline interpolace oproti lineární
- FFT – spektrální analýza pomocí knihovny FFTW3
- Doplňkové funkce – kurzory, průměrování, XY režim
- Integrace s knihovnami – QCP, updater, crash handler, FFTW3
- Instalátor – multiplatformní specifika a instalace souborů

Třídní diagram aplikace

Na obrázku 4.16 je vidět, že hlavní okno aplikace se jmenuje WindowMain. Toto okno je hlavní bod celé aplikace. Závisí na externí statické knihovně qBreakPad, který slouží k tvorbě logů, v případě pádu aplikace a na knihovně QSimpleUpdater, která slouží k automatické aktualizaci. Třída WindowMain také drží reference na ostatní okna, resp. primární a sekundární přístroje.

Primární a sekundární přístroje představují 6 oken, které budu dále nazývat jako Přístroj. V těchto třídách leží hlavní implementace. Závisí na podpůrných třídách pro grafy, kurzory, FFT a export. Každý Přístroj implementuje několik zpráv, které dědí z abstraktní třídy Msg.

Všechna okna závisí na třídě Core, která představuje jádro aplikace. Je v ní implementován stavový stroj, který řídí komunikaci, posílání zpráv a rozesílání odpovědí do příslušných tříd.

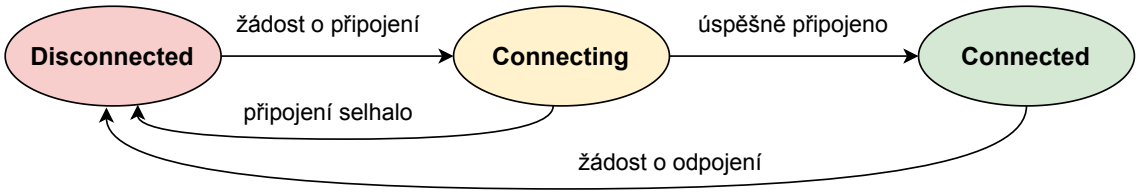


obr 4.16: Třídní diagram PC aplikace EMBO osciloskopu

Implementace komunikačního jádra

Třída Core je stavový stroj, který ve zjednodušené verzi může existovat v právě 3 stavech: Disconnected, Connecting a Connected. V první stavu systém spí. V druhém stavu se navazuje komunikace tak, že se odešle zpráva *IDN?;:SYS:LIM?;:SYS:INFO? a čeká se na odpověď. Pokud odpověď přijde do časového limitu a je korektní, systém přejde do třetího stavu. Ve stavu Connected systém přijímá odpovědi, zpracovává je a v definovaném intervalu periodicky odesílá zprávy od všech přístrojů.

1. Disconnected – systém spí
2. Connecting – navazování spojení pomocí zprávy s timeoutem
3. Connected – je aktivní příjem znaků včetně časovače, který udává interval odesílání zpráv

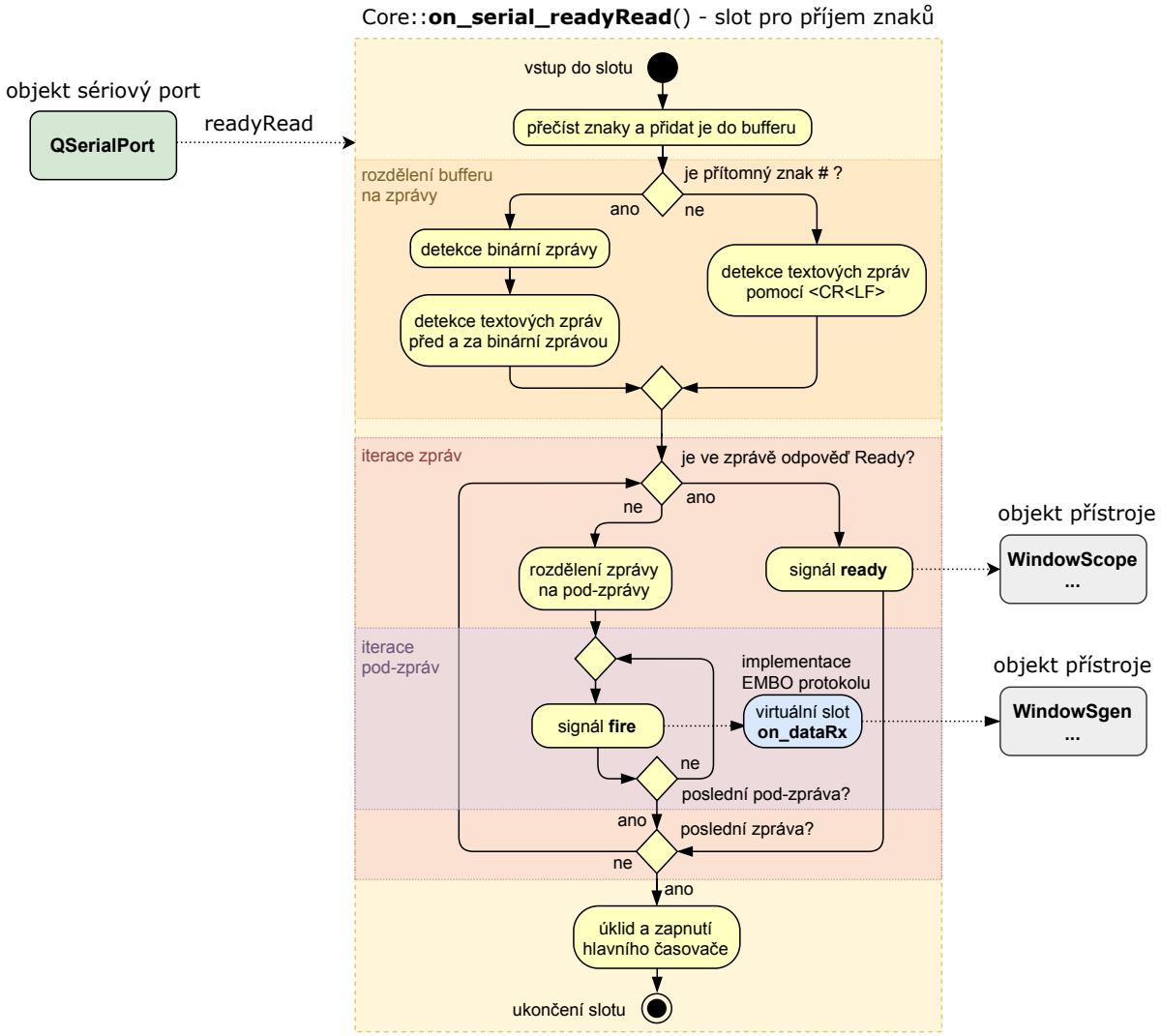


obr 4.17: Implementace komunikačního jádra PC aplikace – stavový stroj Core

Díky známému bugu v knihovně QSerialPort musí třída Core běžet ve vlastní vlákně, protože jinak by docházelo k blokování příjmu zpráv při manipulaci s oknem aplikace. Tento bug existuje od počátku, vývojáři Qt to vědí, ale nic s tím hodlat nedělají. Pokud by neexistoval, mohla by celá aplikace běžet v jednom vlákně. I když je použití více vláken komfortní, spotřebovávají systémové prostředky a mohou vést k chybám souběhu (race condition). V aplikacích obsluhující pomalý sériový port není potřeba vícevláknového zpracování, protože výkonu je dostatek.

Každá zpráva se může skládat z více pod-zpráv. Hlavní zprávy jsou odděleny dvojicí znaků <CR><LF>. Pod-zprávy jsou odděleny středníkem (;). Pokud je v pod-zprávě více parametrů, ty jsou odděleny čárkou (,). Toto platí pro čistě textové zprávy. Každá zpráva ale může obsahovat jednu a více textových pod-zpráv a jednu binární pod-zprávu. V takovém případě se musí nejprve zpracovat binární zpráva, až pak lze zpracovávat textové zprávy.

Jádro je postaveno na 2 časovačích, bufferu příchozích znaků a frontě odchozích zpráv. Nejprve se plní buffer přijatými znaky. S každým přijatým znakem je prohledán celý buffer pomocí algoritmu popsaného na obrázku 4.18. Pokud do 3 sekund nepřijde žádná zpráva, je pomocí časovače detekován timeout a systém přejde do stavu Disconnected.

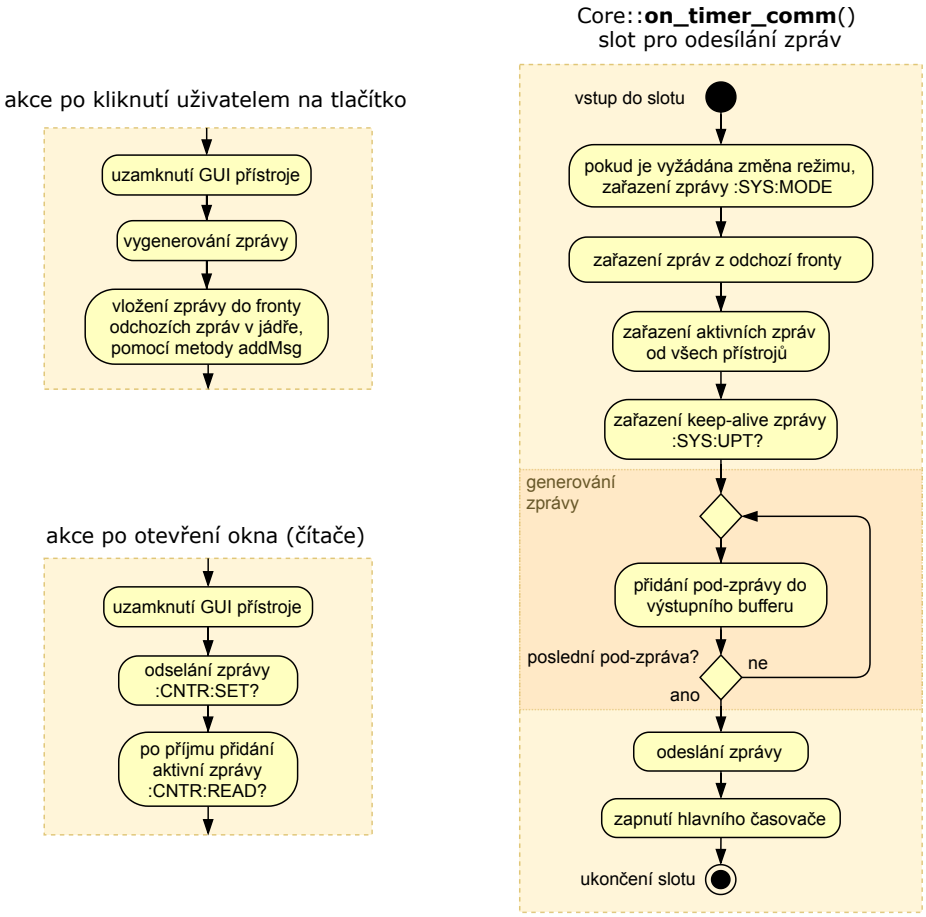


obr 4.18: Implementace komunikačního jádra PC aplikace – algoritmus zpracování příchozích zpráv

Pro každou přijatou zprávu musí vždy existovat korespondující objekt v odchozí frontě. Výjimkou je asynchronní zpráva Ready. Uživatel např. klikne na 2 tlačítka ve 2 přístrojích a do fronty se zařadí 2 zprávy. Hlavní časovač odesílá zprávy v odchozí frontě v stejném pořadí v jakém očekává příchozí odpovědi. Časovač se zapne až po té, co jsou přijaty všechny odpovědi. Perioda časovače se může dynamicky měnit v závislosti na délce zpráv.

Rozesílání přijatých dat do jednotlivých tříd je řešeno pomocí mechanismus signál-slot. Základním komunikačním objektem je třída Msg. Třída Msg obsahuje přetíženou metodu fire, signál rx a virtuální slot on_dataRx. Parametr signálu jsou přijatá nezpracovaná data. V konstruktoru se spojí signál s virtuálním slotem. Dále existuje pro každou SCPI zprávu zvlášť třída, která dědí z třídy Msg a implementuje virtuální slot on_dataRx. V tomto slotu se zpracují přijatá data, který se dále rozesílají jako signály ostatním třídám.

Metoda fire vysílá signál rx, efektivně tak volá virtuální překrytý slot z potomka. Tímto mechanismem se řeší mnoho věcí. Především je přehledně oddělena část jádra, část příjmu zpráv a část zpracování zpráv. Dále je při kompilaci kontrolováno, že každá koncová třída má implementováno zpracování zprávy, tedy překrývá čistě virtuální slot on_dataRx.



obr 4.19: Implementace komunikačního jádra PC aplikace – algoritmus rozesílání odpovědí k příjemcům

Vytvoření nových zpráv je jednoduché a další rozšíření protokolu je tedy bez problémů. Stačí vytvořit novou třídu, podědit z třídy Msg, překrýt slot on_dataRx implementací konkrétní zprávy a nakonec vyslat signál pro objekt příjemce. Můžou se použít obecné signály ok a err, nebo se přidá vlastní signál např. result.

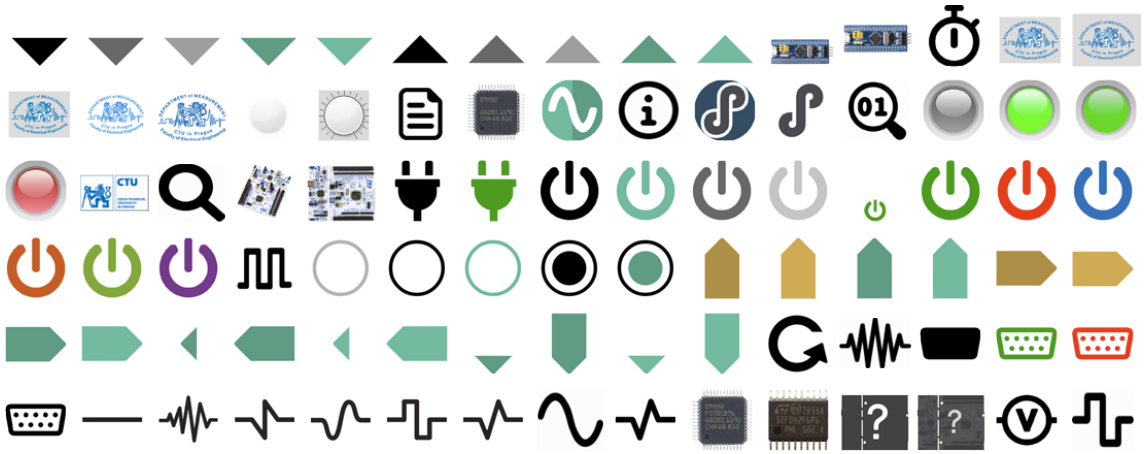
Třída Msg neslouží jen pro příjem a zpracování dat. Představuje kompletní komunikační objekt pro jednu zprávu z EMBO protokolu. Příkaz se definuje v konstruktoru. Pak lze libovolně přepínat mezi dotazem a příkazem, nebo měnit parametry. Pro odeslání zprávy se zavolá metoda addMsg, která objekt zařadí do fronty odchozích zpráv. Od této chvíle se programátor už nemusí o nic starat. Je zaručeno, že zpracovaná odpověď přijde té správné třídě ve správný čas.

Kromě explicitního odesílání zpráv po uživatelské akci existuje také mechanismus pro automatické odesílání. To se hodí např. pro přístroj voltmetr nebo čítač, kdy je nutné periodicky vyčítat nová data. Za tímto účelem existuje rozhraní IEmboInstrument. Každý přístroj má pole aktivních zpráv, které je normálně prázdné. Pokud chce aktivně zpřístupnit zprávu (např. :CNTR:READ?), tak ji přidá do tohoto pole. Jádro pak pomocí metody z rozhraní getActiveMsgs tuto zprávu periodicky odesílá.

Pro udržování komunikace (*keep-alive*) se posílá zhruba každých 10 ms zpráva :SYS:UPT? pro vyčtení doby běhu od spuštění. Pokud není otevřen žádný přístroj, posílá se pouze tato zpráva, v opačném případě se zařadí jako pod-zpráva nakonec. Interval 10 ms je pouze optimistický, protože odesílání většího bufferu dat pro osciloskop může trvat mnohem déle. Pro krátké zprávy se systém snaží regulovat celkovou latenci na 10 ms. Naopak u déle trvajícího přenosu se zkrátí interval na 1 ms, aby celková latence byla co nejnižší.

Grafický návrh PC aplikace

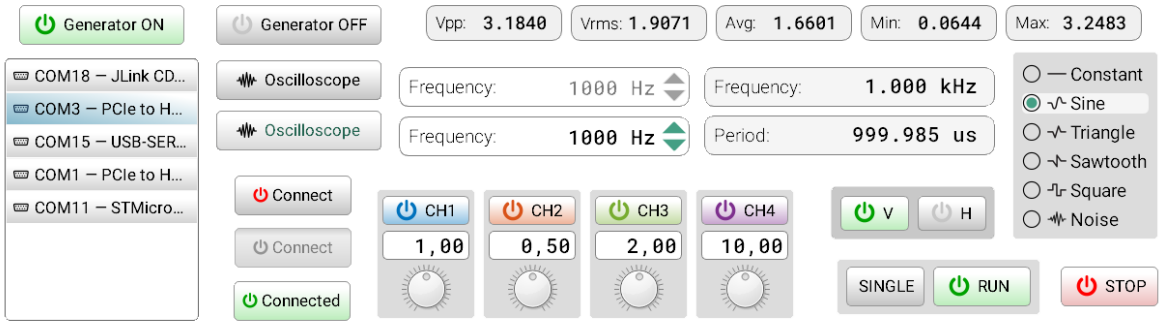
Dle mého názoru často nebývá kladen na grafický návrh aplikací dostatečný důraz. Při vývoji PC aplikace osciloskopu EMBO je investováno do grafiky mnoho času. To z toho důvodu, že pro koncového uživatele je vzhled GUI a kvalita interakce s ovládacími prvky na prvním místě. Zjednodušeně řečeno, pokud se uživateli nelíbí GUI, často to nezachrání ani skvělý backend.



obr 4.20: Obrázky a ikony použité v PC aplikaci

Pro každý ovládací prvek je existuje unikátní styl definovaný textově pomocí kaskádových stylů (CSS). Dohromady všechny CSS styly ladí a vytváří tak jednotné grafické prostředí napříč okny aplikace. Každý ovládací prvek má pod-styl *hover*, který definuje změnu stylu po přejetí myší. Pro výplň větších prostor, jako je pozadí nebo tlačítka, jsou zvoleny odstíny šedé. Jako hlavní tématická barva osciloskopu EMBO je zvolena azurovo-zelená (#449f84). Tato barva je použita pro malé a výrazné plošky, jako je logo, šipky nebo kurzory.

Kromě CSS stylů je v aplikaci také použito přes 80 obrázků a ikon. Azurové logo s bílou vlnovkou, které jste již mohli často zahlédnout v této práci, jsem vytvořil přímo pro tuto aplikaci. Jako hlavní zdroj ikon je použita volně dostupná databáze FontAwesome. Ikony jsou často modifikované ad-hoc, například pro různé barevné kombinace a odstíny.



obr 4.21: Ukázka ovládacích prvků s vlastním CSS designem

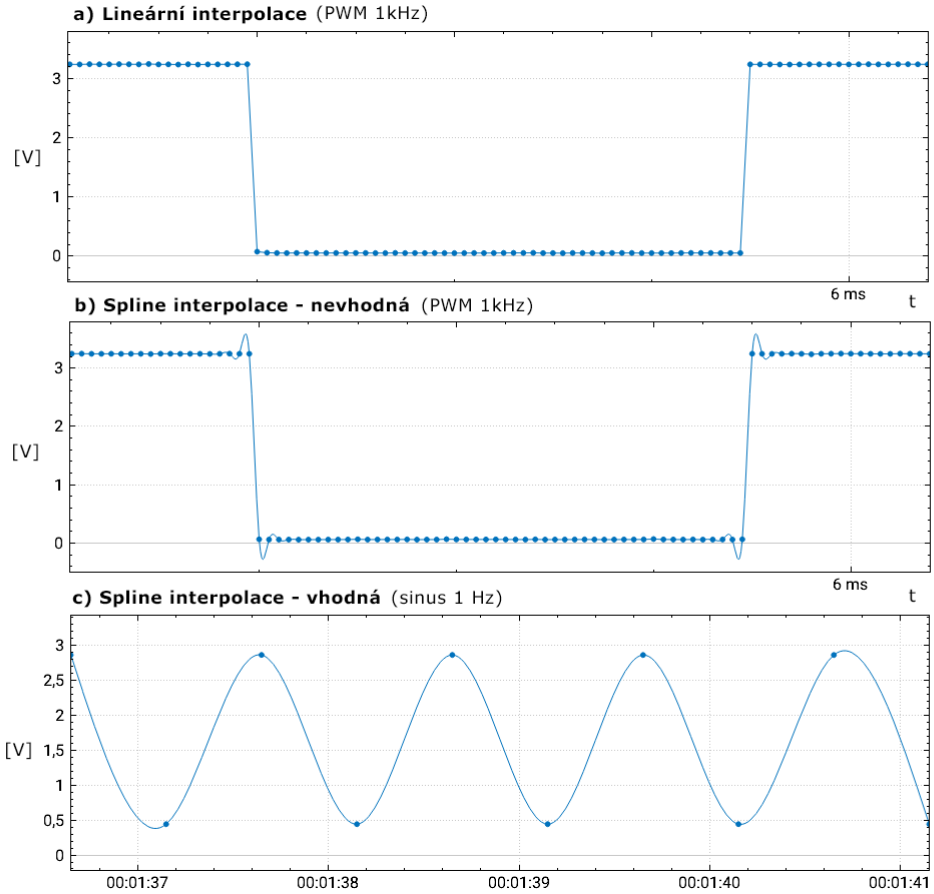
Hlavní a jediné písmo (font) použité v aplikaci je Roboto od Google v různých variantách (Monospace, Medium, Thin). Písmo Roboto je volně dostupná verze nejpopulárnějšího písma všech dob jménem Helvetica, resp. Helvetica Neue. Písmo Helvetica je perfektní, bohužel ale není zdarma. Tak samo písmo San Francisco od Apple (nástupce Helveticy), které je často označováno jako to nejlepší písmo, není zdarma. Pro grafický návrh aplikací je nutné zvolit font a distribuovat ho jako *resource* spolu s aplikací, protože jinak se může stát, že GUI bude na každém systému vypadat jinak.

Lineární a kubická interpolace

Interpolace bodů znamená jejich propojení pomocí definované funkce. Je to mocný nástroj, který ale někdy dokáže být kontraproduktivní. Proto je dobré vědět princip funkce a kterou metodu kdy zvolit.

Základní a nejjednodušší je interpolace lineární, tedy propojení přímou čarou. Knihovna QCustomPlot podporuje pro vykreslování grafů pouze tuto metodu. Lineární interpolace je univerzální a hodí se ve většině případů. Díky spojení nejbližších bodů nejkratší čarou je maximalizována pravděpodobnost, že analogový signál opravdu leží na spojnici bodů. Výpočetní výkon je zanedbatelný.

Pokud víme, že analogový signál má sinusový charakter, může nám kubická interpolace výrazně pomoct, zvlášť pokud máme málo vzorků na periodu. Kubická (Spline) interpolace dokáže i s 2 vzorků na periodu vytvořit krásný sinus. Problém ale nastává, pokud signál nemá sinusový charakter. Pak dochází k artefaktům (překmitům) na signálu tam, kde ve skutečnosti nejsou. Osciloskopy standardně používají $sin(x)/x$ neboli *sinc* interpolaci, která je podobná kubické interpolaci.



obr 4.22: Rozdíl mezi lineární a kubickou (spline) interpolací (v prvním případě je vidět PWM o frekvenci 1 kHz, v druhém je vidět falešný překmit a v třetím je sinus s 2 vzorky na periodu)

Pro osciloskop EMBO je použita kubická interpolace ve verzi Spline. Matematicky to znamená, že první a druhé derivace navazujících polynomů, definujících jednu křivku, musí být vždy spojité. Dále se jejich druhá derivace musí rovnat 0.

$$p_1'(x_0), \quad p_n'(x_n) \qquad \ldots \quad spojitá \tag{4.4}$$

$$p_1''(x_0) = p_n''(x_n) = 0 \qquad \ldots \quad spojitá \tag{4.5}$$

Knihovna QCustomPlot bohužel nepodporuje kubickou interpolaci. Lze ale využít vestavěné funkce frameworku Qt *cubicTo* a *quadTo*, které s minimálním vytížení CPU kreslí Bézierovy křivky pomocí 4 bodů. Pro každou křivku je třeba počátek, konec a pak dva pomocné body, které určují tvar. Nejprve se projde pole všech interpolovaných bodů a vypočítají se pomocné body. V druhé iteraci se pak postupně kreslí křivky.

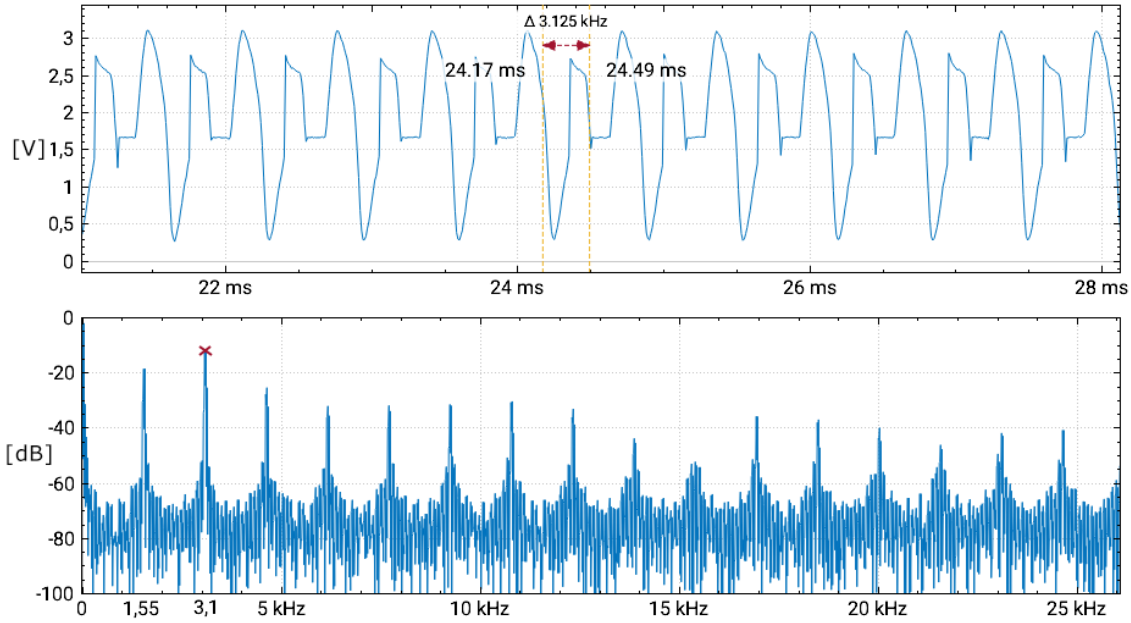
- Lineární interpolace – vhodná pro zobrazování PWM a neperiodických signálů
- Kubická (spline) interpolace – vhodná pro zobrazování sinusového periodického signálu

FFT spektrální analýza

Součástí osciloskopu EMBO je také FFT spektrální analýza. Jako knihovna realizující algoritmus analýzy je zvolena volně dostupná FFTW3 (The Fastest Fourier Transform in the West), která představuje to nejlepší v dané sekci. Knihovna je distribuována s aplikací staticky, odpadají tak potíže s kompatibilitou dynamických knihoven. Nicméně je nutné pro každý systém předkompilovat tuto statickou knihovnu v určité verzi.

1. Zkopírování vstupních dat do pole o velikosti 2^N a vyplnění 0 (zero padding)
2. Aplikace okenní funkce (Hanning)
3. Výpočet FFT (fftw_execute)
4. Normalizace amplitudy a převod do oboru reálných čísel na decibely

Aby byl výpočet FFT rychlý, musí být vstupní data velikosti mocniny čísla 2. Je dobré zvolit vyšší číslo a vstupní data doplnit 0 (padding), tím se sice rozlišení nezvýší, ale výsledná křivka se vyhladí. Výsledek je nutné korektně převést do oboru reálných čísel a normalizovat. Dále je nutné aplikovat okenní funkci pro utlumení nespojitostí v periodické funkci. Je zvolena okenní funkce Hanning.

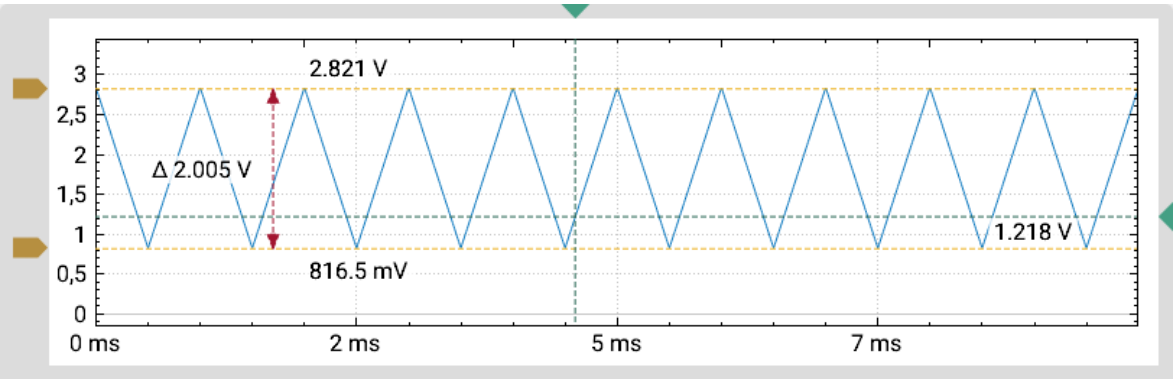


obr 4.23: Ukázka výsledku spektrální analýzy pomocí knihovny FFTW3 v osciloskopu EMBO (frekvence modulovaného signálu: 3.1 kHz, vzorkovací frekvence: 100 kSps)

Další doplňkové funkce

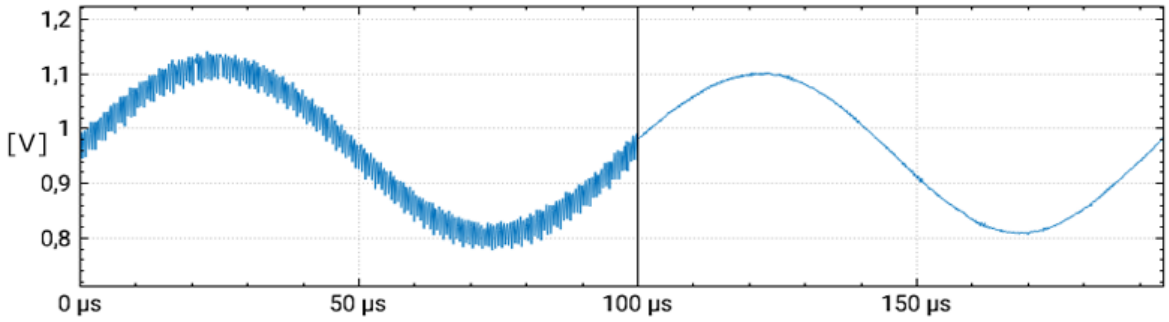
Kurzory – vlastní knihovna QCPCursors.
Kurzory jsou důležitý nástroj každého osciloskopu. Díky nim lze snadno měřit signál v obou osách. Knihovna QCustomPlot nenabízí žádný ucelený nástroj, který by se dal použít jako kurzor. Je proto nutné implementovat vlastní knihovnu pro tento účel.

V PC aplikaci jsou kurzory použity v přístroji osciloskop, logický analyzátor a voltmetr. Kurzory jsou implementovány v třídě QCPCursors a mohou být použity i v jiných projektech. Je nutné použít speciální ovládací prvek RangeSlider, který v Qt Widgets neexistuje (v Qt Quick už je přítomen). Prvek RangeSlider dědí z widgetu QSlider a přidává funkci dvou posuvných táhel. Volně dostupná knihovna ctkRangeSlider řeší tento problém.



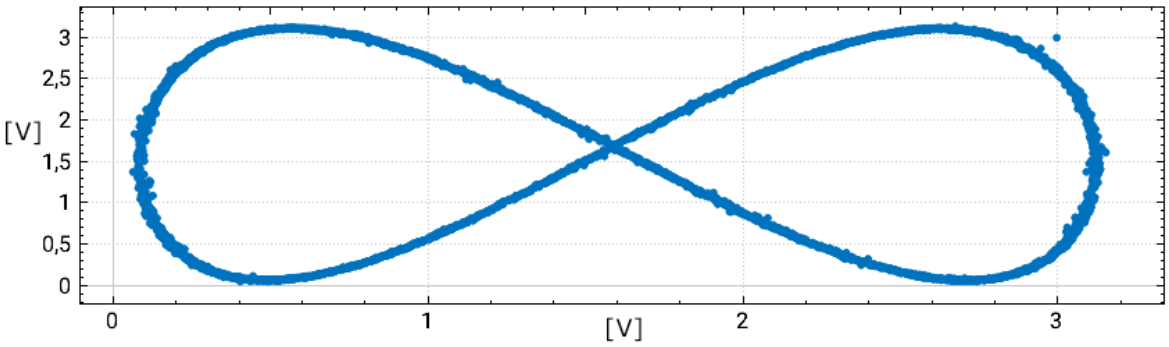
obr 4.24: Kurzory v osciloskopu EMBO implementované pomocí vlastní knihovny

Režim průměrování v osciloskopu.
Režim průměrování lze zapnout u mnoha stolních osciloskopů a je vhodný použít pro filtrování zašumělého signálu. Tento režim je také implementován v osciloskopu EMBO. Jediné nastavení je počet iterací průměrování. Implementace je řešena pomocí 2D kruhového bufferu.



obr 4.25: Na levé straně je vidět zašumělý sinusový signál (f=10 kHz) a na pravé straně je výsledek po zapnutém průměrování (50 iterací) – vzorkovací frekvence: 5 MSps

Režim Math a režim XY v osciloskopu.
Režim Math slouží zobrazení rozdílu dvou kanálů. V osciloskopu EMBO je implementován pouze rozdíl kanálů 2-1 a rozdíl 4-3. Režim XY je vhodný pro zobrazení VA charakteristik nebo Lissajousových obrazců. V tomto režimu je místo osy času X použit jiný kanál a na ose Y je zobrazen druhý kanál. Zapnout tyto režimy lze v horním menu.



obr 4.26: Režim XY a Lissajousův obrazec (poměr frekvencí 1:2)

Integrace s knihovnami

Ideální je použít externí knihovny jako zdrojové soubory, které budou zkompilovány spolu s aplikací. To je vhodné pro knihovny napsané ve stejném jazyce a frameworku. Druhou možností je staticky linkovat, kdy knihovny budou připojeny již při kompilaci. Tato možnost je vhodná pro rozsáhlejší knihovny psané v jiném jazyce. Dále je možné použít dynamické knihovny, kdy se knihovna připojí při startu aplikace. Tato možnost není vhodná pro Linux, protože roztříštěnost verzí distribucí způsobuje výrazné problémy s binární kompatibilitou.

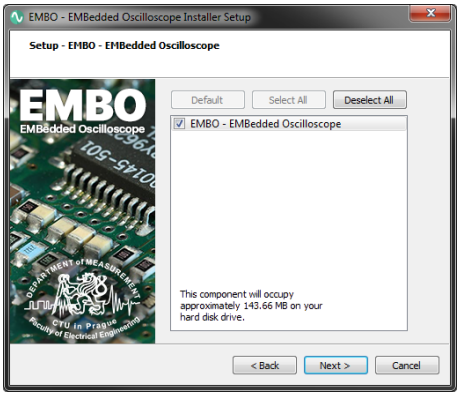
Jsou použity 4 externí knihovny pro rozšíření funkcionality EMBO osciloskopu. Pro grafy je použita knihovna QCustomPlot a pro automatickou aktualizaci knihovna QSimpleUpdater, obě jako zdrojový soubor. Dále je použita knihovna Google Breakpad v Qt verzi jménem qBreakPad pro ukládání hlášení o pádu (crash handler). Pokud aplikace havaruje (run-time chyba, *segfault* ..) je automaticky uložen výpis paměti (dump) do složky *crashes*, který slouží vývojáři pro nalezení chyby. Tato knihovna je linkována staticky. Poslední použitou knihovnou je FFTW3, která je linkována staticky, pouze v případě Windows dynamicky.

- QCustomPlot (grafy) – zdrojový kód
- QSimpleUpdater (automatická aktualizace přes internet) – zdrojový kod
- qBreakPad (crash handler) – statická knihovna a zdrojový kód
- FFTW3 (spektrální analýza) – statická knihovna (dynamická pro Windows)

Instalátor

Formát výsledného spustitelného soubor osciloskopu EMBO se liší v závislosti na použitém operačním systému. Pro běžného uživatele je dobré tento fakt odstínit, např. pomocí instalátoru, který jednoduše integruje aplikaci do systému. Uživateli pak stačí kliknout na zástupce na ploše nebo panelu.

V případě systému Windows je aplikace distribuována jako soubor EXE spolu s dynamickými knihovnami (Qt framework a ostatní). Knihovny nejdou v tomto případě linkovat staticky kvůli licenci firmy Qt. Je tedy nutné stáhnout archív se zhruba 60 soubory, vytvořit složku na disku a soubory nakopírovat. Tento proces právě zjednodušuje instalátor.



obr 4.27: Instalátor osciloskopu EMBO ve verzi pro systém Windows 7

Systémy Linux, např. Ubuntu, obecně velmi trpí na binární kompatibilitu (ABI) dynamických knihoven mezi jednotlivými verzemi. Díky tomu se musí pro každou verzi každé distribuce kompilovat aplikace zvlášť a i tak může dojít k problémům s kompatibilitou. Proto vývojáři neradi kompilují GUI aplikace pro Linux. Firma Qt si je toho vědoma a licence pro Linux umožňuje linkovat staticky. Výsledná aplikace je tak ve formě jednoho spustitelného souboru.

I když je systém macOS (OS X) *UNIX-like* podobně jako Linux, problémy s ABI kompatibilitou má obecně menší, protože většina uživatelů aktualizuje a verzí OS je podstatně méně. Díky tomu stačí kompilovat pro poslední 2 verze macOS a většina uživatelů bude spokojena. Aktuální téma je Apple Silicon a přechod na ARM. Díky dynamickému binárnímu překladači Rosetta 2 by měl být přechod snadný. Aplikace pro macOS Catalina (Intel) běží na macOS Big Sur (ARM) většinou bez problémů, protože Rosetta 2 provede dynamický překlad strojového kódu při startu aplikace.

Výsledek kompilace pro systém macOS je ve formátu APP, což je v podstatě adresář, který systém nativně prezentuje jako spustitelnou aplikaci. Zde není třeba instalátor, protože uživateli stačí přesunout *soubor* do složky Aplikace.

4.4 Fáze 4 — Testování a nasazení

Testování je důležitá část vývoje každého software, která by měla probíhat kontinuálně a s důrazem na různé principy (unit testy, integrační testy, funkcionální testy). Při vývoji osciloskopu EMBO je nutné důsledně testovat, protože více podporovaných operačních systémů a více podporovaných řad mikrořadičů mnohonásobně zvyšuje pravděpodobnost vzniku chyby.

Firmware osciloskopu EMBO je postavený na jednotném zdrojovém kódu pro různé řady čipů STM32. Tím je docílena nulová redundance a unifikované chování. Může ale nastat situace, že se změní implementace pro vyřešení problému u jedné řady MCU. Tím ale může dojít k vzniku problému u jiné řady MCU. Tomuto problému se lze vyvarovat dostatečnou abstrakcí, promyšleným kódem a hlavně dostatečným testováním.

Ideální by bylo veškerý proces testování automatizovat. Trend automatického testování a nasazení je již roky viditelný v jiných segmentech (např. webové aplikace) a jmenuje se CD/CI. Nástroj typu Kubernetes v intervalech automaticky stáhne zdrojový kód z gitu a zkompiluje a provede všechny testy. Pokud jsou testy v pořádku, automaticky nahraje binární aplikaci na server, pokud došlo k chybě, zašle upozornění vývojáři.

Jelikož je vývoj firmware svázán se specifickým hardwarem a jsou třeba specifické nástroje, musím testovat a nasazovat převážně ručně. Testování ve finální fázi probíhá tak, že po kompletním odladění firmware pro jednu mikroprocesorovou řadu uložím firmware ve formátu EMBO_F123_X.Y.Z.HEX do složky *build*. Verze firmware je definována pomocí X.Y.Z a řada mikrořadiče pomocí F123. Pokud dojde k větší změně, např. je upraven protokol nebo je přidána nová funkce, je nutné vytvořit nový HEX soubor pro všechny řady a hlavně jej zvlášť znovu otestovat.

Jako věc k zamyšlení může být vytvořit automatický systém, který by fungoval podobně jako Kubernetes, ale pro *embedded* vývoj. Každý den by se kontrolovalo, jestli je v gitu nový commit. Pokud ano, stáhnul by celý git strom, zkompiloval by se firmware pro všechny řady STM32 a nahrál by se do připojených vývojových desek. Následně by se spustila testovací aplikace (např. v MATLAB nebo C#), která by dle definovaných parametrů provedla různé testy a vyhodnotila výsledek na základě komunikace se zařízeními.

Kapitola 5

Vlastnosti a parametry výsledné realizace softwarově definovaného osciloskopu

obr 5.1: Logo osciloskopu EMBO

V této kapitole jsou prezentovány parametry osciloskopu EMBO a výsledky měření především pomocí mikrořadičů STM32F103 a STM32F303. Následující přehled má sloužit jako zběžný katalogový list, který je určen hlavně pro koncového uživatele. Popis funkcionality přístrojů přeložený do angličtiny (viz. kapitola 5.2) je distribuován spolu s aplikací ve formátu CHM (Windows) a PDF (UNIX).

Nejprve jsou uvedeny numerické limity jednotlivých přístrojů, které byly určeny na základě měření. Dále jsou prezentovány snímky obrazovky přístrojů z operačních systémů Windows 7 a Xubuntu 18. Kapitola je zakončena ověřením šířky pásma ADC u desky BluePill pomocí stroboskopického vzorkování (vzorkování v ekvivalentním čase).

1. Základní parametry a numerické limity jednotlivých přístrojů
2. Představení PC aplikace a jednotlivých přístrojů
3. Ověření šířky pásma ADC u STM32F103 pomocí stroboskopického vzorkování

5.1 Parametry osciloskopu

Firmware osciloskopu EMBO byl zkompilován a otestován pro mikrořadiče STM32F103C8, STM32F103RE, STM32303RE a STM32L412KB. U STM32F103 a STM32L412 lze volit komunikaci mezi UART a USB. U STM32F303RE založeném na desce Nucleo-64 je použit výhradně UART.

Pro mikrořadiče STM32G031 a STM32L072KZ existuje firmware v experimentální verzi. Tyto MCU jsou postavené na jádru Cortex-M0+, které nedisponuje hardwarovou děličkou. Při jisté situaci při uvolnění semaforu dojde k přepsání návratové adresy v zásobníku a tím ke korupci datových struktur FreeRTOS. Je třeba tento problém detailně analyzovat a opravit. Po té bude možné rozšířit tabulku podporovaných čipů i o tyto MCU.

Základní parametry přístrojů osciloskopu EMBO								
STM32	produkt	USB	SCOPE	LA	VM	CNTR	PWM	SGEN
F103C8	BluePill	✓	4 ch / 1 ADC	4 ch	4 ch	1 ch	2 ch	×
F103RE	LQFP64	✓	4 ch / 2 ADC	4 ch	4 ch	1 ch	2 ch	1 ch
F303RE	Nucleo64	STLINK	4 ch / 4 ADC	4 ch	4 ch	1 ch	2 ch	1 ch
L412KB	Nucleo32	✓	4 ch / 2 ADC	4 ch	4 ch	1 ch	1 ch	×

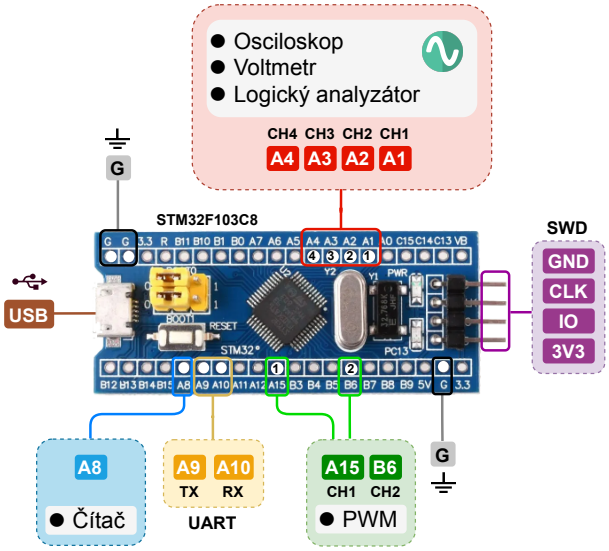
Tabulka 5.1: Parametry přístrojů osciloskopu EMBO – SCOPE: osciloskop, LA: logický analyzátor, VM: voltmetr, CNTR: čítač, PWM: PWM generátor, SGEN: signálový generátor

Limity přístrojů osciloskopu EMBO							
STM32	produkt	DAQ	SCOPE	LA	CNTR	PWM	SGEN
F103C8	BluePill	10 kB	0.800 MSps	5.142 MSps	33 MHz	14.4 MHz	×
F103RE	LQFP64	50 kB	0.800 MSps	5.142 MSps	33 MHz	14.4 MHz	4.5 Msps
F303RE	Nucleo64	50 kB	4.965 MSps	14.400 MSps	57 MHz	20.6 MHz	4.5 Msps
L412KB	Nucleo32	28 kB	5.000 MSps	13.333 MSps	37 MHz	16.0 MHz	×

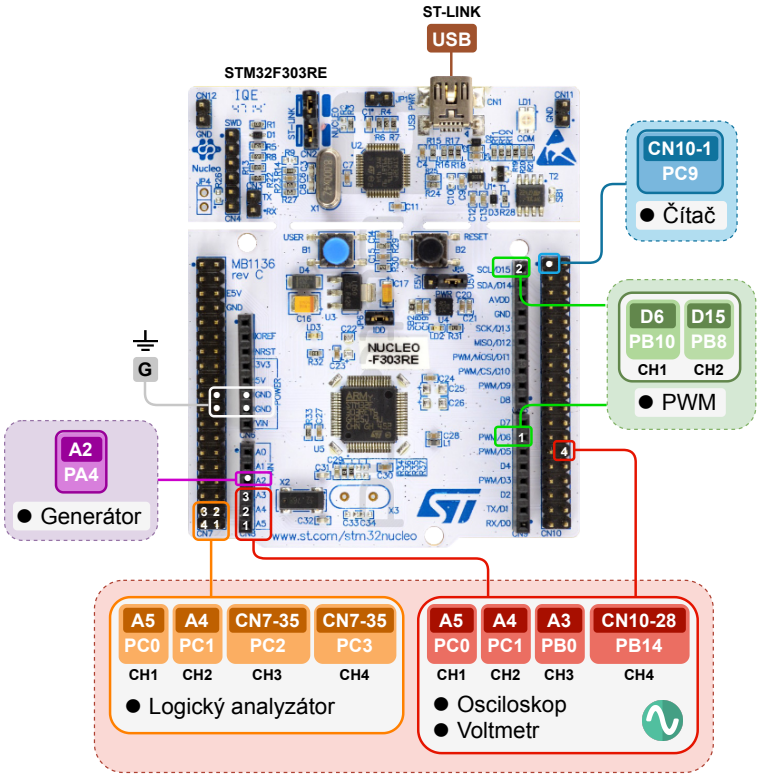
Tabulka 5.2: Limity přístrojů EMBO (DAQ je velikost paměti přístrojů SCOPE a LA)

V tabulce 5.2 jsou stanoveny numerické limity osciloskopu EMBO. Hodnota ve sloupci DAQ je celková velikost paměti pro osciloskop nebo logický analyzátor. Sloupce SCOPE a LA definují maximální vzorkovací frekvenci na jeden kanál. Ve sloupci CNTR je maximální frekvence obdélníkového signálu, u něhož lze korektně změřit frekvenci čítačem. Ve sloupci PWM je maximální frekvence PWM signálu. Sloupec SGEN určuje maximální frekvenci generování vzorků pomocí periferie DAC, kdy platí, že pro maximální hodnoty se dynamicky zmenšuje velikost výstupního bufferu.

5.1.1 Rozložení přístrojových pinů pro STM32F103C8 a STM32F303RE



obr 5.2: Rozložení přístrojových pinů – STM32F103C8 (BluePill)



obr 5.3: Rozložení přístrojových pinů – STM32F303RE (Nucleo64) – LEO kompatibilní

5.2 Představení PC aplikace osciloskopu

5.2.1 Hlavní okno

Po spuštění PC aplikace se uživatel dostane do hlavního okna s výběrem jednotlivých přístrojů. Po levé straně je seznam dostupných portů (v případě OS Windows jsou to *COM* porty, v případě *UNIX-like* systému jsou to *tty* porty). Uživatel vybere port a připojí se k osciloskopu. Zobrazí se tři panely na pravé straně. V horním panelu jsou informace o mikrořadiči a firmware. V prostředním panelu jsou primární přístroje a jejich parametry a ve spodním panelu jsou dostupné sekundární přístroje.

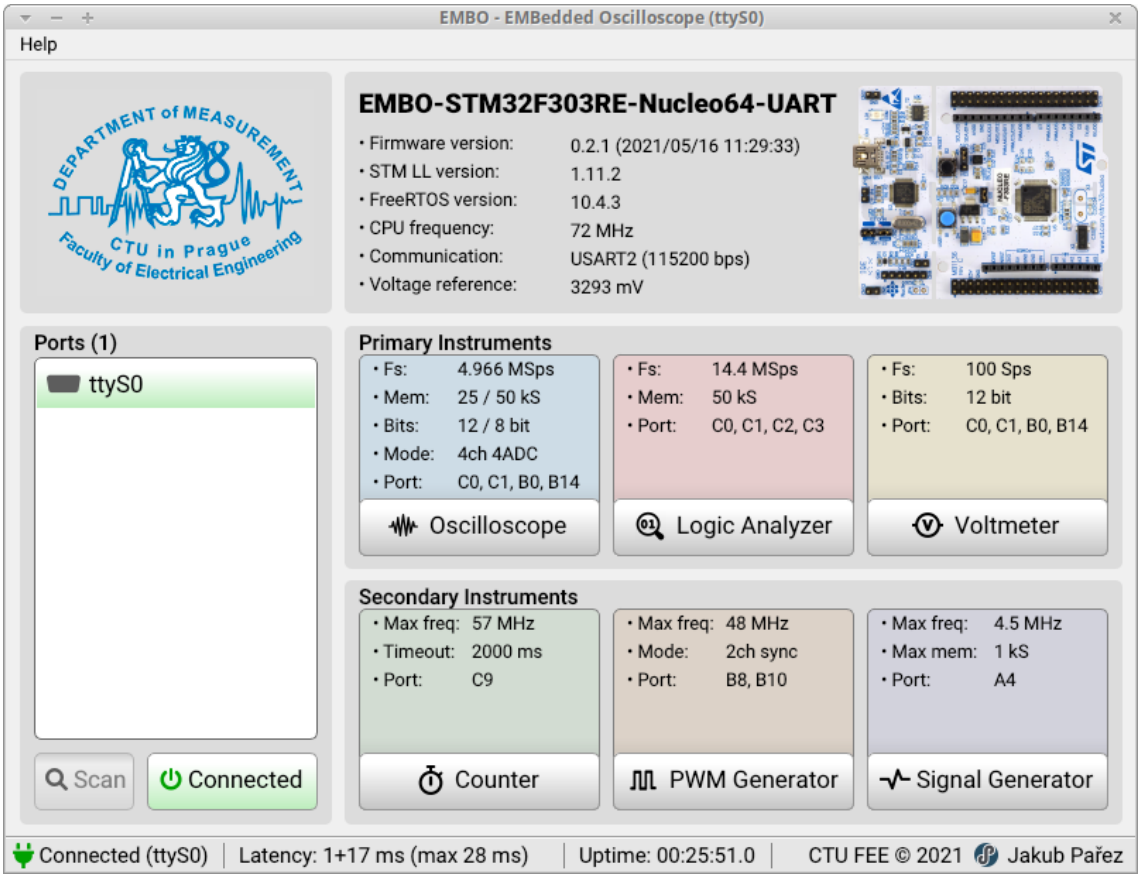
obr 5.4: Hlavní okno osciloskopu EMBO v systému Windows 7 (F103C8)

V dolní liště (stavovém panelu) jsou informace o aktuální relaci, které se v ideálním případě aktualizují s periodou 10 ms. Latence udává dobu odezvy osciloskopu a Uptime udává dobu od zapnutí. Pokud je komunikace vytížena (osciloskop odesílá mnoho dat) může se latence zvýšit až na stovky ms v případě použití UART (ST-LINK).

V horním menu lze otevřít uživatelskou nápovědu nebo lze spustit Updater, který automaticky aktualizuje aplikaci na nejnovější verzi. Tato funkcionalita bude plně funkční až po vytvoření webové stránky osciloskopu na stránkách katedry (léto 2021).

Popis parametrů přístrojů:

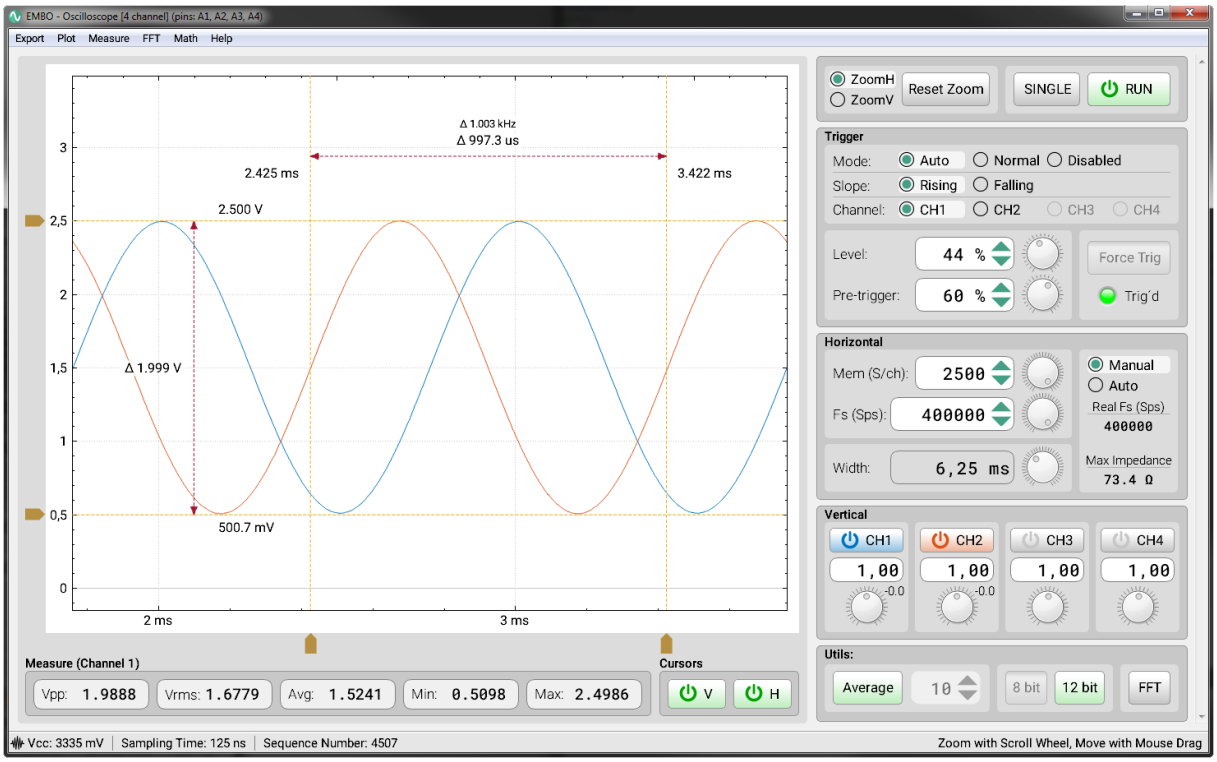
- Fs – vzorkovací frekvence ve vzorcích za sekundu
- Mem – hloubka paměti ve vzorcích
- Bits – rozlišení v bitech
- Mode – režim přístroje (4ch 4ADC → 4 kanály na 4 ADC)
- Port – přiřazení portů mikrořadiče přístrojovým kanálům (C0 → PC0)
- Max freq – maximální frekvence vstupního nebo výstupního signálu



obr 5.5: Hlavní okno osciloskopu EMBO v systému Xubuntu 18 (F303RE)

5.2.2 Osciloskop

Osciloskop je stěžejní přístroj celého systému. Okno osciloskopu je rozděleno do dvou hlavních bloků. V levém bloku je interaktivní graf, pod kterým leží měřící prvky a ovladače kurzorů. V pravém bloků jsou soustředěny ty nejdůležitější ovládací prvky. Pomocné a doplňkové funkce se ovládají z horního menu. Spodní část obrazovky je vyčleněna pro stavový panel.



obr 5.6: Přístroj osciloskop (F103C8) – 2× sinus, f=1 kHz, φ=120°, f_s=400 kSps (průměrováno)

Popis ovládacích panelů:

- Hlavní panel – leží vpravo nahoře, základní režimy (Run, Stop, Single)
- Trigger – nastavení spouštění
- Horizontal – nastavení časové základny, resp. vzorkovací frekvence a paměti
- Vertical – nastavení kanálů a jejich zisk a posun
- Utils – doplňkové funkce (průměrování, rozlišení, FFT)
- Cursors – ovládání kurzorů
- Measure – indikátory měřených parametrů

Hlavní panel.
Podobně jako u většiny osciloskopů i EMBO disponuje 3 hlavními režimy: Run, Stop a Single. Implementace těchto režimů je inspirována stolními digitálními osciloskopy (Agilent, Tektronix). Režim Run je primární režim s kontinuální akvizicí dat. Režimu Stop slouží k zastavení akvizice a k analýze signálu. Speciální režim Single slouží pro zachycení jednorázového děje. Dále v tomto panelu má uživatel možnost resetovat zoom nebo celý přístroj.

Trigger.
Nastavení spuštění je také inspirováno stolními přístroji. Výchozí je vždy režim Auto, který kombinuje funkcionalitu režimu Normal s časovým limitem, který umožňuje vidět signál ve všech případech. Dále je nastavitelná hrana spouštění (Slope), kanál (Channel), spouštěcí úroveň (Level) a pre-trigger. V režimu Normal a Single má uživatel možnost vynutit trigger, což slouží jako rychlý náhled. Indikační LED jménem Trig´d svítí zeleně, pokud byl trigger úspěšný (bliká s každou novou zprávou).

Horizontal.
Nastavovat horizontální parametry lze ve 2 režimech. V manuálním režimu uživatel nastavuje vzorkovací frekvenci (Fs) a hloubku paměti na kanál (Mem). V automatickém režimu jsou tyto 2 parametry optimalizovány na základě celkové délky zobrazovaného signálu (Width). Tento přístup byl zvolen proto, že klasická mřížka a nastavení (sekunda/dílek) zde nedává smysl, protože graf je interaktivní a uživatel s ním může libovolně zoomovat a hýbat.

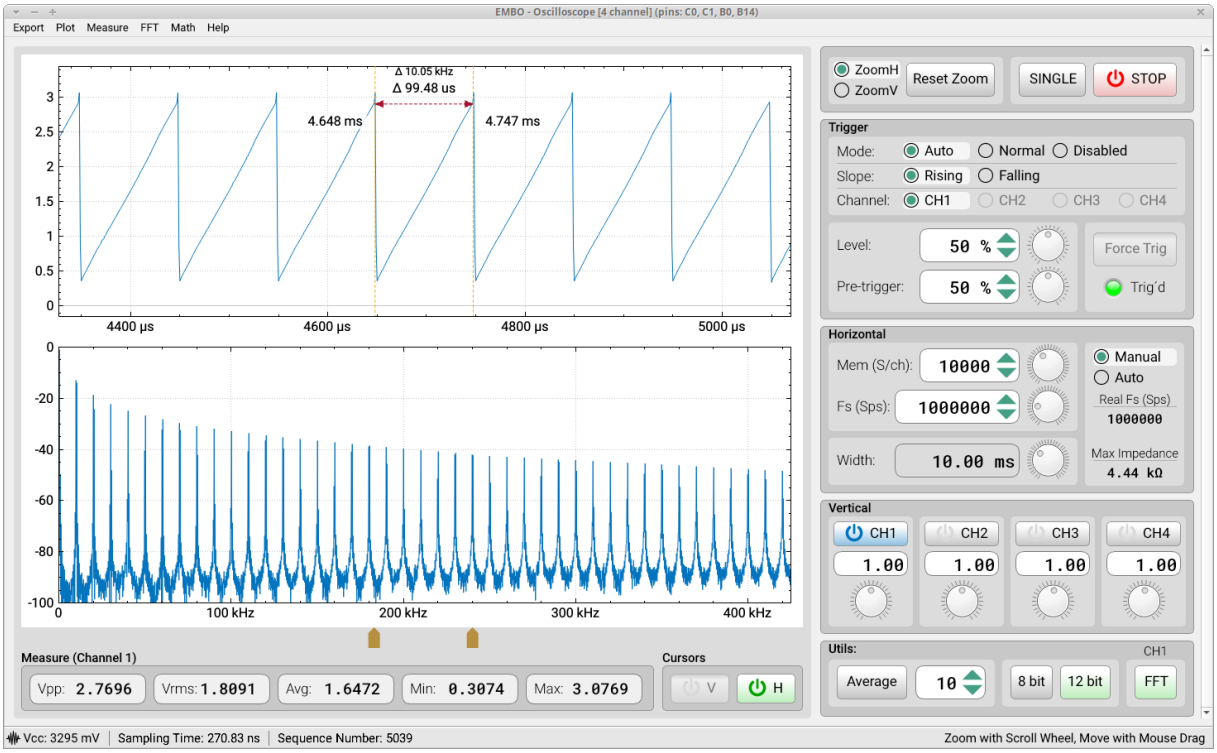
Dále jsou zobrazeny 2 důležité informace. Reálná vzorkovací frekvence (Real Fs) s přesností na 6 řádů je potřebná při vzorkování v ekvivalentním čase a následnému přepočtu frekvence. Maximální přípustná impedance zdrojového signálu je do detailu popsána v kapitole 3.1.1.

Vertical.
Vertikální nastavení umožňuje zapínat jednotlivé kanály, nastavovat jejich zisk a posun.

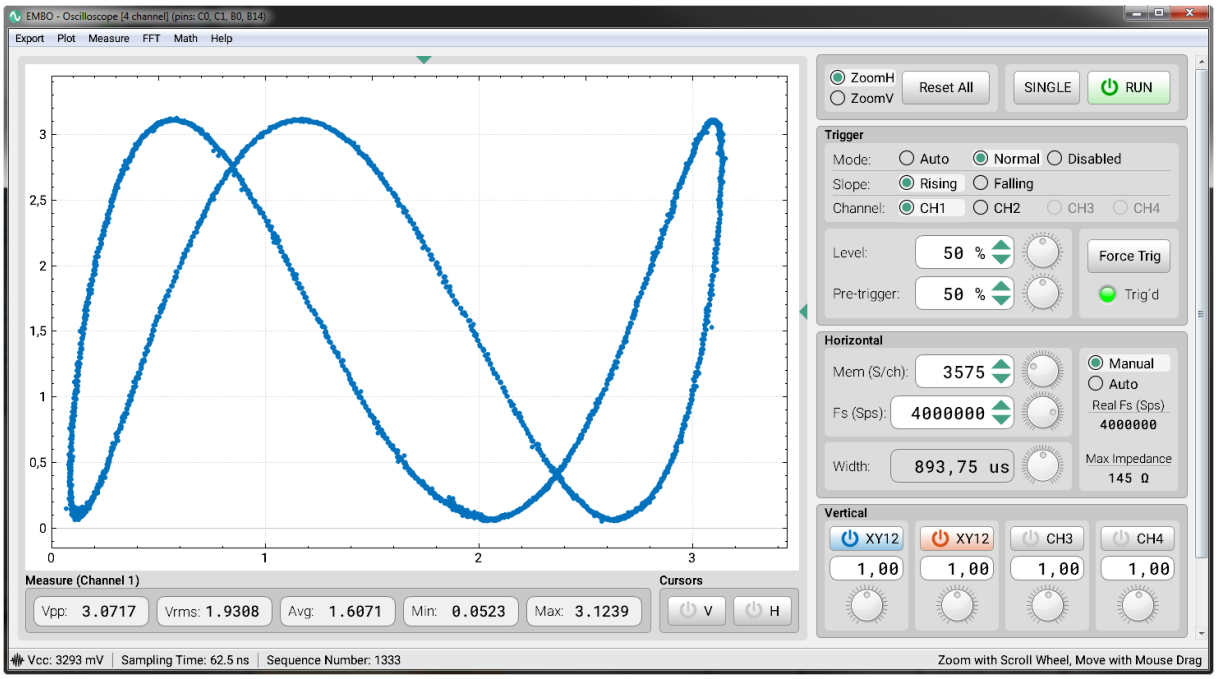
Utils, Cursors a Measure.
Průměrování (Average) je vhodné pro vyhlazení zarušeného signálu. Případný 8-bitový režim urychlí přenos dat na úkor snížení rozlišení a FFT zapíná režim spektrální analýzy. Kurzory (Cursors) slouží pro měření signálu v obou osách a měřené parametry (Measure) představují základní informaci o signálu.

Popis horního menu:

- Export – export dat ve formátu CSV, TXT nebo jako screenshot
- Plot – nastavení zobrazení grafu a interpolace
- Measure – nastavení měřených parametrů
- FFT – natavení FFT spektrální analýzy
- Math – ovládání speciálních režimů (rozdíl, XY)



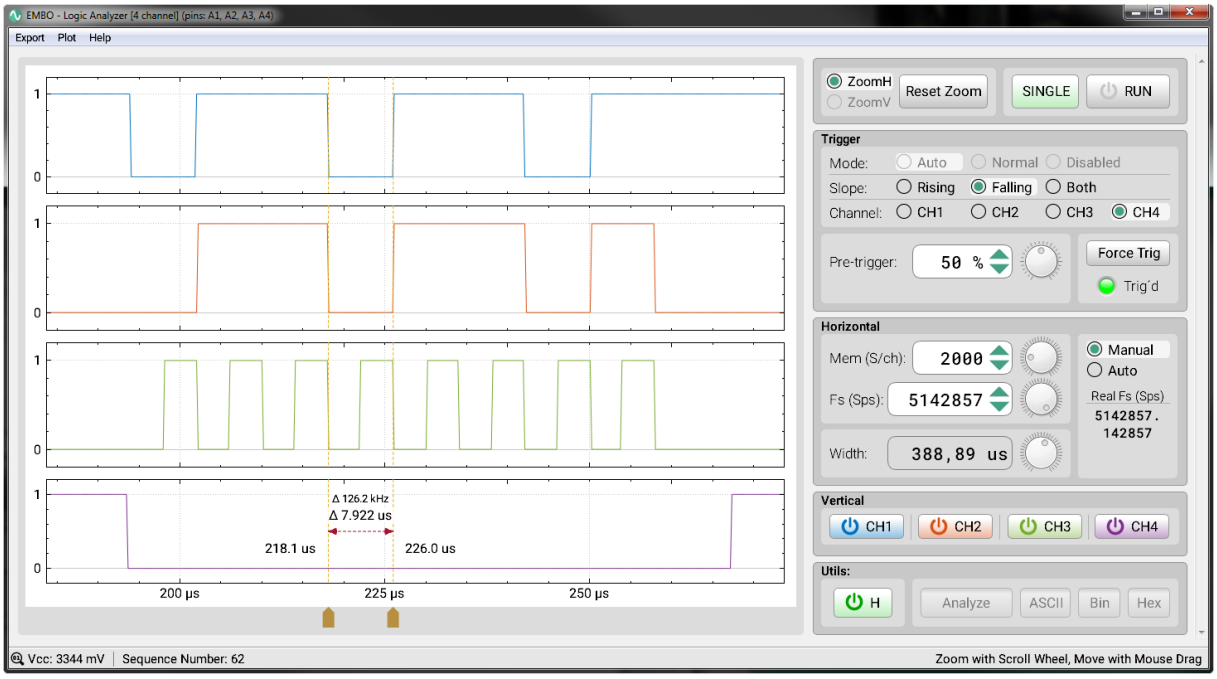
obr 5.7: Přístroj osciloskop (F303RE) – pila, f=10 kHz, režim FFT, f_s=1 MSps



obr 5.8: Přístroj osciloskop (F303RE) – 2× sinus, f_1=5 kHz, f_2=15 kHz, φ=15°, režim XY (Lissajousův obrazec s poměrem frekvencí 1:3), f_s=4 MSps

5.2.3 Logický analyzátor

Logický analyzátor zprostředkovává doplňkovou funkcionalitu osciloskopu pro sledování rychlých digitálních signálů. Podobně jako osciloskop je přístroj rozdělen do dvou hlavních bloků. V levém bloku je interaktivní graf, který dynamicky zobrazuje příslušný počet kanálů, na rozdíl od osciloskopu ale pod sebou. V pravém bloků jsou hlavní ovládací prvky vycházející z osciloskopu. Doplňkové funkce se opět ovládají z horního menu. Spodní část obrazovky je vyčleněna pro stavový panel.

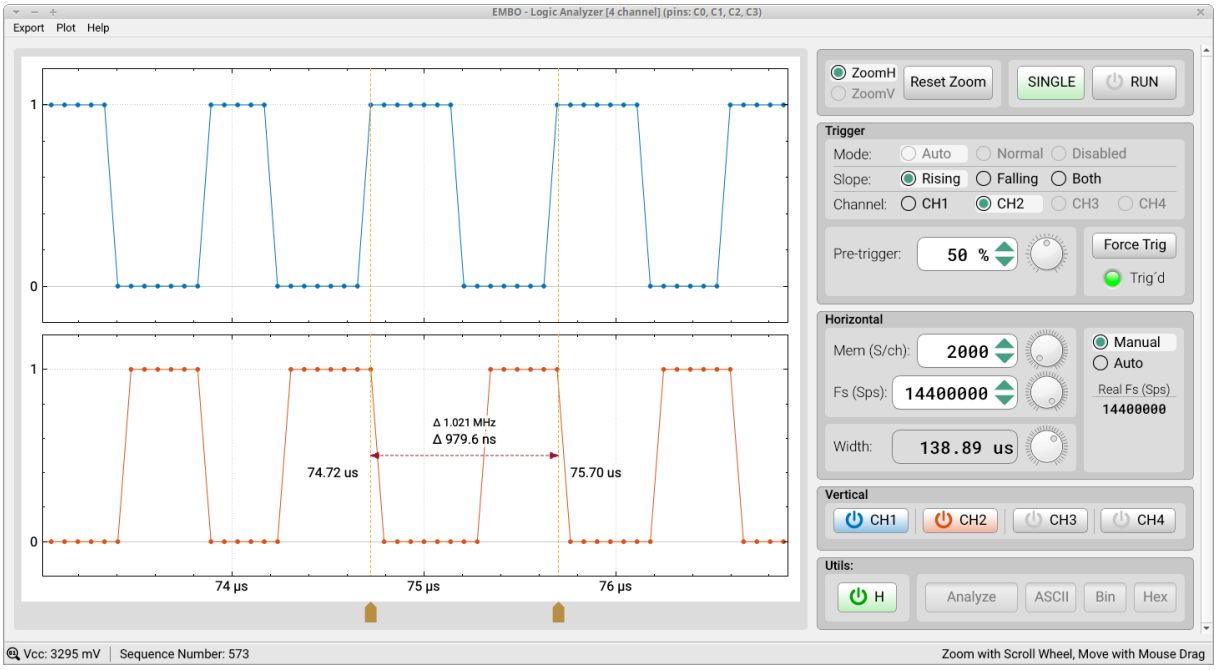


obr 5.9: Přístroj logický analyzátor (F103C8) – SPI komunikace, f_{SCK}=125 kHz, f_s=5.142 MSps

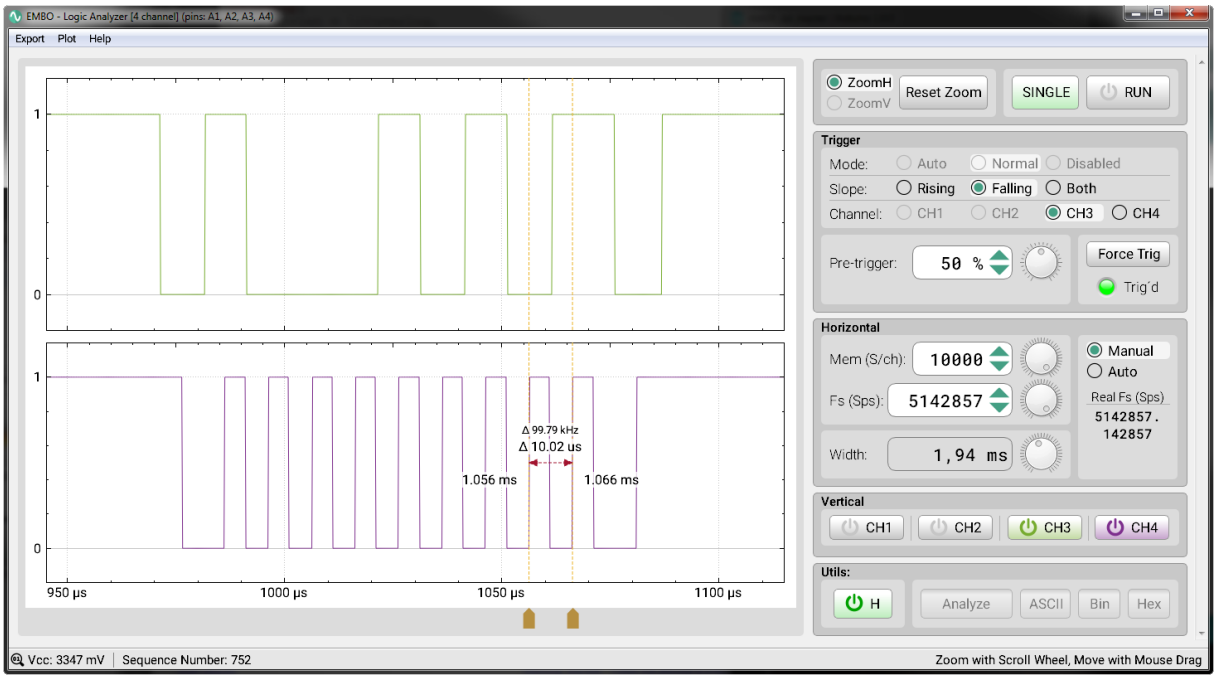
Většina ovládacích prvků sdílí stejnou funkčnost i design s osciloskopem. Rozdíl je u triggeru, kde chybí spouštěcí úroveň, naopak přibyla možnost triggerovat na obě hrany (Both). Dale chybí vertikální zesílení a offset, protože pro logický analyzátor nemají smysl. Podobně také chybí režim průměrování, bitové rozlišení a FFT.

Popis ovládacích panelů:

- Hlavní panel – leží vpravo nahoře, základní režimy (Run, Stop, Single)
- Trigger – nastavení spouštění
- Horizontal – nastavení časové základny, resp. vzorkovací frekvence a paměti
- Vertical – výběr zobrazených kanálů
- Utils – ovládání kurzorů, analýza dat (bude doplněna později)



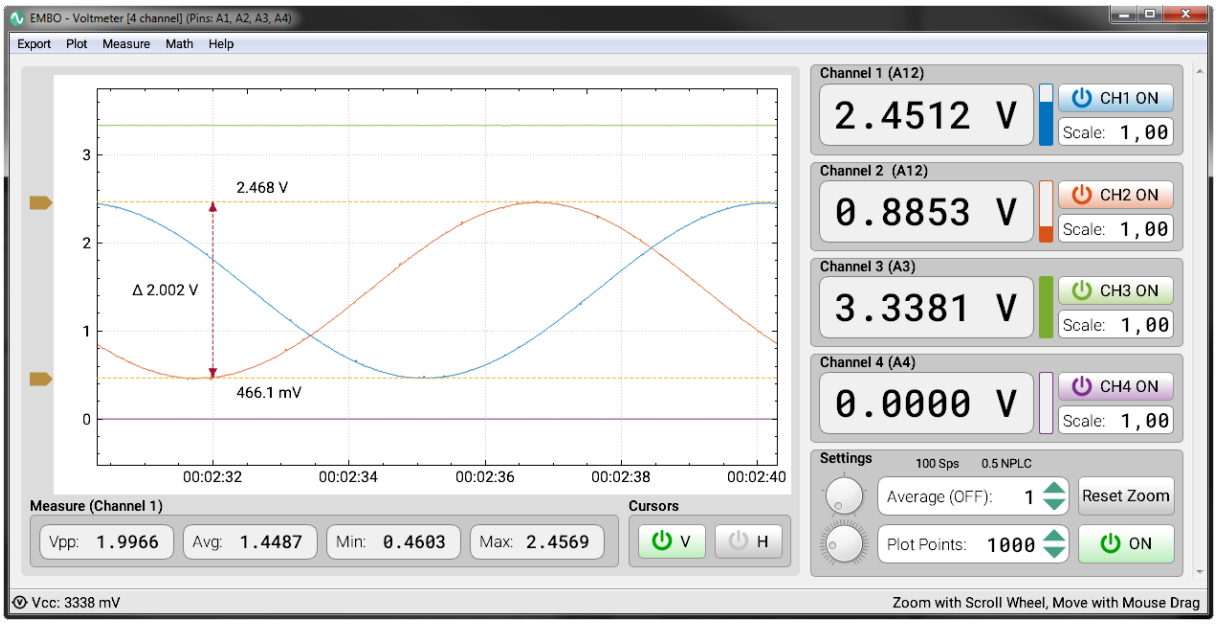
obr 5.10: Přístroj logický analyzátor (F303RE) – 2× synchronní PWM, f=1 MHz, φ=180°, f_s=14.4 MSps



obr 5.11: Přístroj logický analyzátor (F103C8) – I2C komunikace (senzor SHT31), f_{SCL}=100 kHz, f_s=5.142 MSps

5.2.4 Voltmetr

Voltmetr je poslední primární přístroj osciloskopu EMBO. Jeho okno je opět rozděleno na dva hlavní bloky. V levém bloku je interaktivní graf, pod kterým jsou zobrazeny měřené parametry a ovladač kurzorů. V pravém bloku jsou pod sebou 4 panely kanálů a pod nimi je hlavní ovládací panel (Settings).



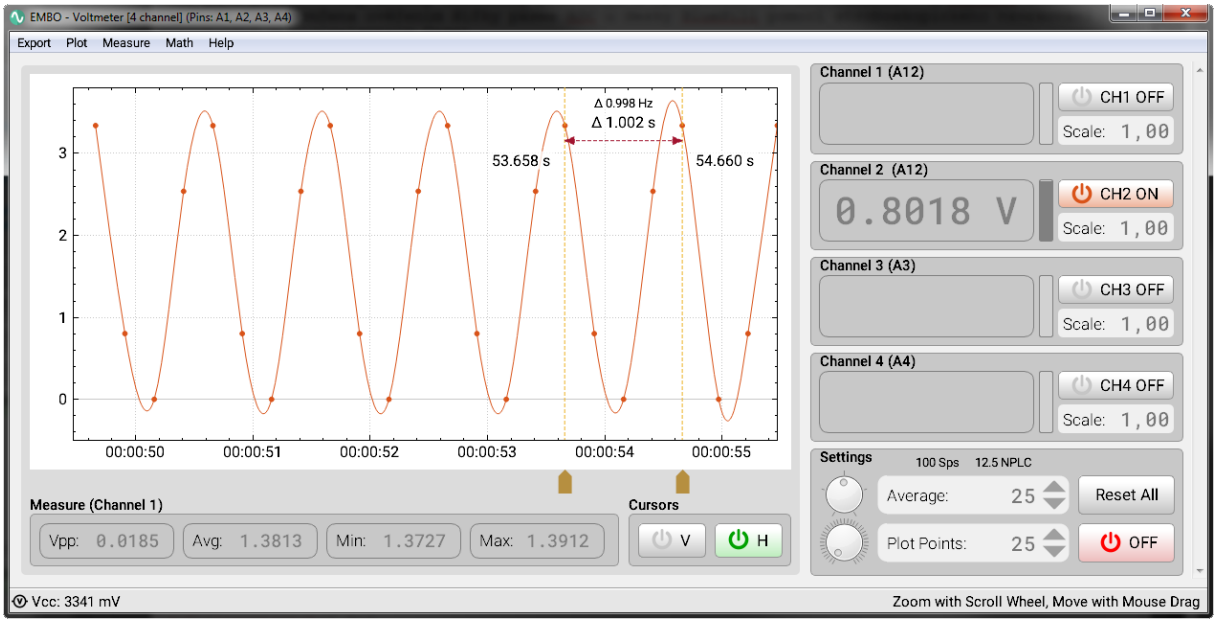
obr 5.12: Přístroj voltmetr (F103C8) – 2× sinus, f=0.1 Hz, φ=120°, f_s: 100 Sps

Popis nastavitelných parametrů:

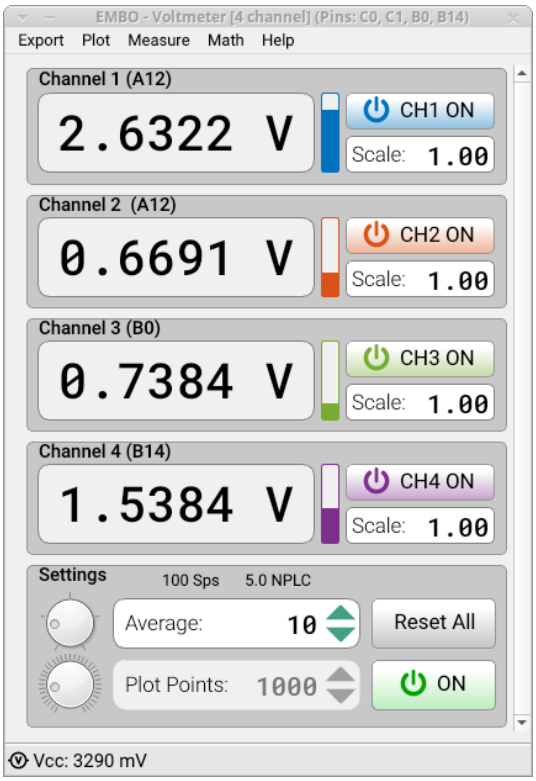
- Scale – konstanta, kterou je na násobeno napětí kanálu
- Average – počet průměrovaných hodnot (1 znamená vypnuto)
- Plot Points – počet zobrazených bodů v grafu

Vzorkovací frekvence je fixně nastavena na 100 Sps. Je to dvojnásobek frekvence napětí z veřejné sítě. Pokud je vypnut režim průměrování, hodnota NPLC (Number of Power Line Cycles) se rovná 0.5. Pokud budeme tuto hodnotu zvyšovat (zvyšováním hodnoty Average), výrazně potlačíme indukovaný 50 Hz šum. Čím vyšší hodnota Average, resp. NPLC, tím pomaleji voltmetr měří a tím více je potlačen šum. Režim Average tedy zvyšuje přesnost při měření konstantního signálu.

Kromě hlavního nastavení lze využít doplňkové možnosti z horního menu, které je podobné jako u osciloskopu. Vedle standardního nastavení, jako je interpolace nebo nastavení měřených parametrů, lze např. zapnout kontinuální nahrávání do souboru (CSV nebo TXT). Zaznamenaný signál bude mít vzorkovací frekvenci 100 Sps.



obr 5.13: Přístroj voltmetr (F303RE) – sinus, f=1 Hz, f_s: 100 Sps, režim průměrování 25 vzorků (4 vzorky na periodu)



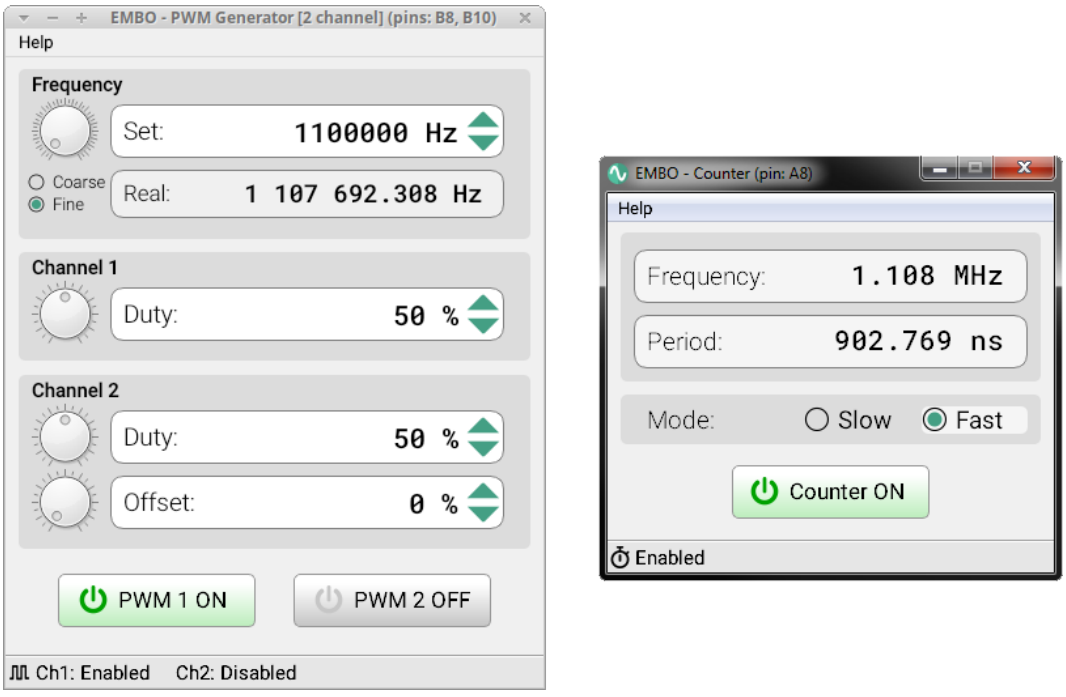
obr 5.14: Přístroj voltmetr (F303RE) – vypnutý graf

5.2.5 PWM generátor

Generátor PWM signálu existuje buď s 1 nebo 2 kanály. V případě 2 kanálů jde o synchronní generátor, který může být použit pro ovládání motorů nebo pro simulaci signálů z kvadraturního enkodéru. Frekvence se nastavuje shodná pro oba kanály. Dále lze nastavit střídu zvlášť pro kanál a offset druhého kanálu. Opět je třeba zdůraznit, že při vysokých hodnotách vzorkovací frekvence osciloskopu může docházet k nežádoucímu ovlivnění měřeného signálu.

5.2.6 Čítač

Čítač je jednoduchý sekundární přístroj, který lze paralelně provozovat s ostatními přístroji. Lze nastavit režim Slow (pro signály do cca 1 MHz) a režim Fast (pro signály cca od 1 MHz). Pro pomalé signály trvá měření delší dobu. V případě že nebyl detekován signál během 2 sekund nastane timeout. Při použití s osciloskopem nebo logickým analyzátorem při maximální vzorkovací frekvenci může docházet k přetížení DMA a vypadávání vzorků.



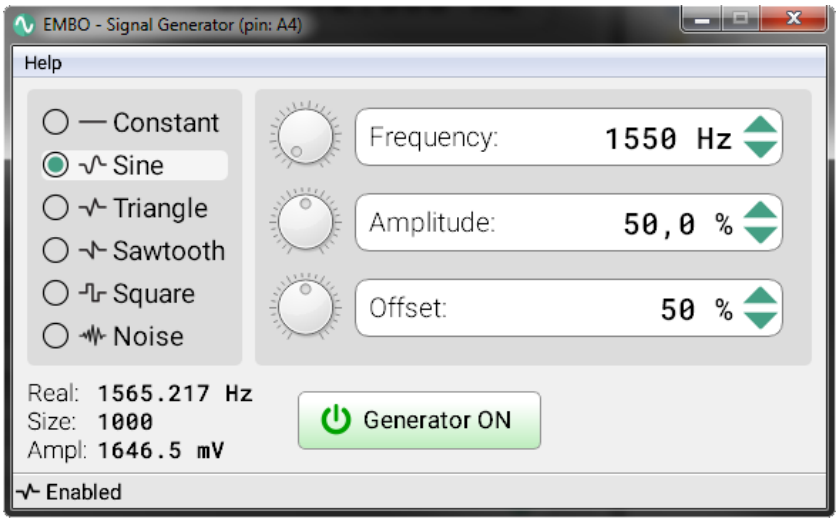
obr 5.15: Přístroje PWM generátor a čítač (F103C8)

5.2.7 Signálový generátor

Signálový generátor je sekundární volitelný přístroj, který využívá 12-bitové DAC. Je dostupný pouze u těch mikrořadičů, které disponují DAC, jako je např. F303RE. Uživatel může nastavit frekvenci, amplitudu a ofset. U vysokých frekvencích je dynamicky snížena velikost výstupního bufferu, tedy vzorků na periodu (Size), aby DAC stíhal generovat. Dále je zobrazena reálná frekvence generovaného signálu a velikost amplitudy v milivoltech. V režimu Constant se chová generátor jako zdroj napětí.

Režimy generování:

- Constant – zdroj napětí
- Sine – sinus
- Triangle – trojúhelník
- Sawtooth – pila
- Square – obdélník
- Noise – pseudonáhodný šum



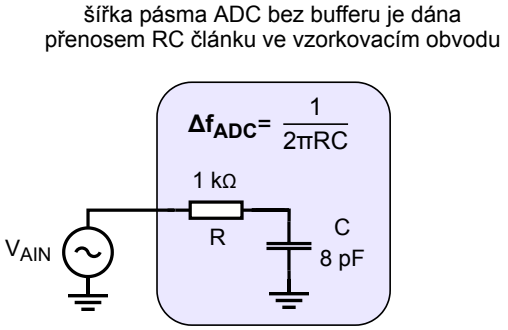
obr 5.16: Přístroj signálový generátor (F303RE)

5.3 Ověření šířky pásma ADC u STM32F103 pomocí stroboskopického vzorkování

Na základě vzorkovacího teorému můžeme vzorkovat signál o 2× nižší frekvenci než vzorkujeme. V případě STM32F103 (BluePill), který má maximální vzorkovací frekvenci 800 kSps, tak můžeme vzorkovat nejvýše signál o frekvenci 400 kHz. Vzorkovací teorém můžeme ale také interpretovat tak, že stačí mít vzorkovací frekvenci 2× vyšší než je analogová šířka pásma signálu. Jinak řečeno, pokud je šířka pásma signálu Δf, stačí vzorkovat frekvencí $2f$.

$$f_0 \leq 2f_s \qquad \Longleftrightarrow \qquad \Delta f \leq 2f_s \tag{5.1}$$

Této interpretace vzorkovacího teorému se využívá při stroboskopickém vzorkování, neboli vzorkování v ekvivalentním čase. V takovém případě nás tedy nezajímá Nyquistova frekvence. Šířka pásma signálu nás také nemusí zajímat, protože bývá řádově nižší než maximální vzorkovací frekvence. Klíčovou roli tady hraje analogová šířka pásma ADC, která je u ADC bez bufferu určena přenosem RC článku ve vzorkovacím obvodu, resp. sériovým odporem a vzorkovacím kondenzátorem.



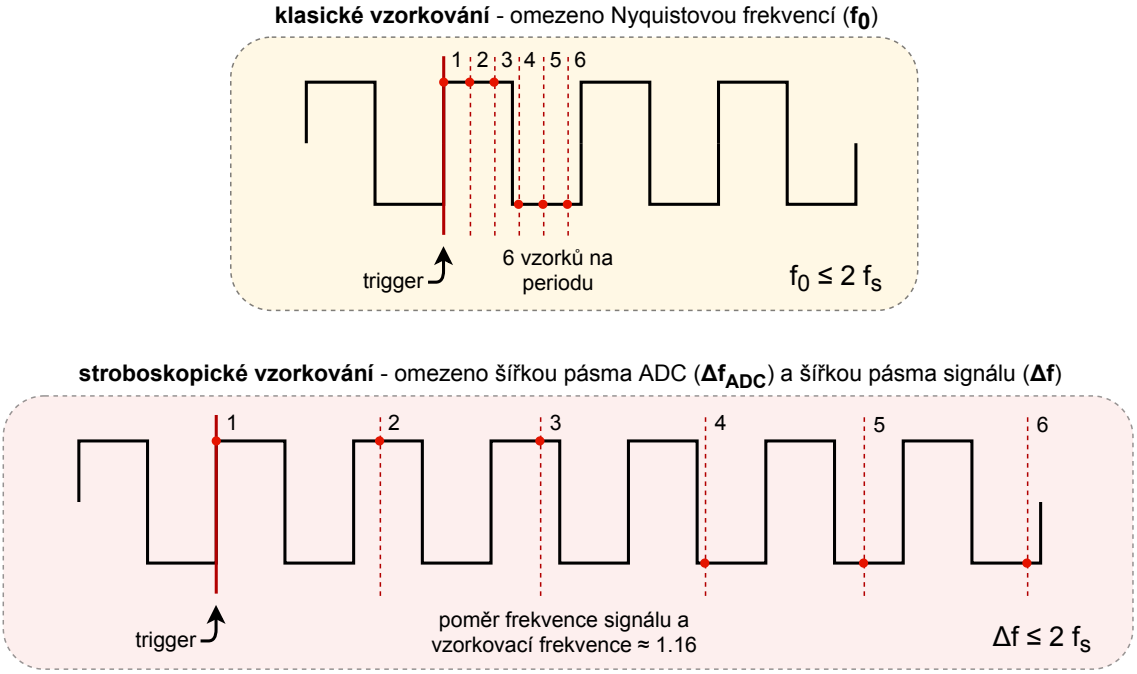
obr 5.17: Šířka pásma ADC bez bufferu je dána přenosem RC článku ve vzorkovacím obvodu

Analogová šířka pásma u STM32F103 je dána 3 dB poklesem na RC článku a pro hodnoty $R = 1\,k\Omega$ a $C = 8\,pF$ vychází zhruba 19.9 MHz. Lze tedy vzorkovat signál do 19.9 MHz, musí se ale použít jiný princip, např. stroboskopické vzorkování. Díky této metodě lze použít relativně pomalé ADC STM32F103 pro analýzu vysokofrekvenčních signálů, jako je např. doba přeběhu operačního zesilovače.

$$\Delta f_{ADC} = \frac{1}{2\pi RC} = \frac{1}{2\pi \cdot 1\,k\Omega \cdot 8\,pF} \approx 19.904\,MHz \tag{5.2}$$

Princip stroboskopického vzorkování, neboli vzorkování v ekvivalentním čase, je založen na kombinaci vzorků signálu z více period. Nutnou podmínkou je periodický a stabilní signál. Metod pro realizaci existuje více. U digitálních osciloskopů se pro tuto funkci používá zpožděný nebo náhodný trigger. Po každém triggerování se akvizice zpozdí o lineárně rostoucí zpoždění nebo o náhodný časový okamžik. Stroboskopicky získané vzorky se pak poskládají do správného pořadí a výsledkem je mnohem vyšší ekvivalentní vzorkovací frekvence.

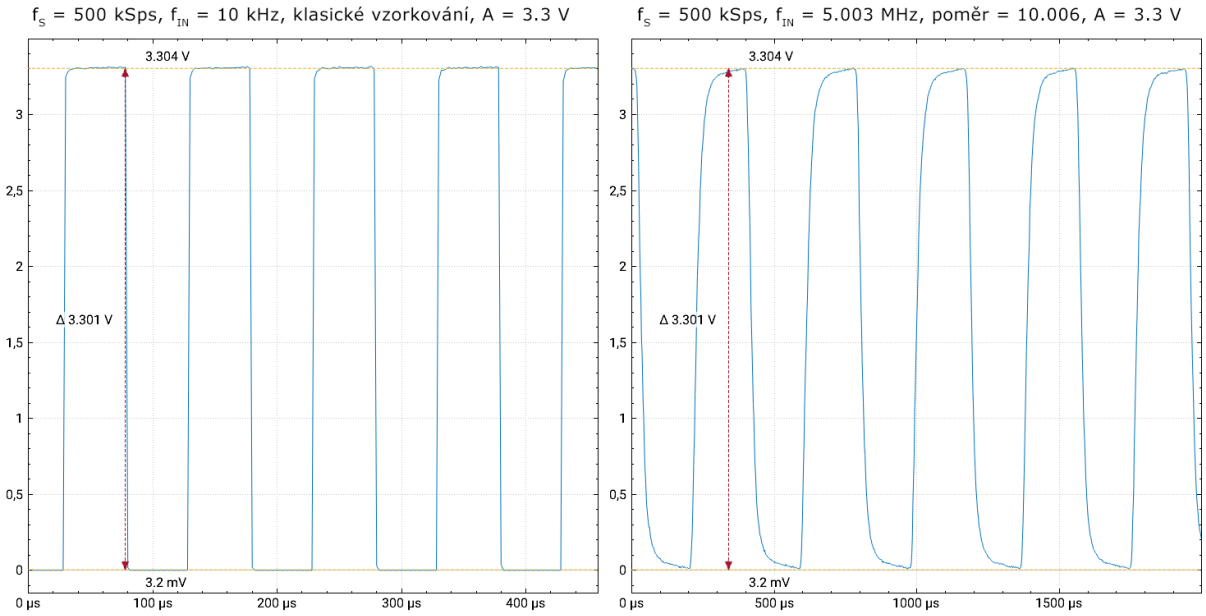
Jednodušší metoda je zvolit správný poměr vzorkovací a vstupní frekvence. Tato metoda nevyžaduje žádné externí komponenty a proto může být realizována přímo osciloskopem EMBO. Nejprve se zvolí frekvence vstupního signálu tak, aby byla celočíselným násobkem vzorkovací frekvence. Pak se vstupní frekvence nepatrně zvýší. Tímto se docílí ten samý efekt jako když se použije trigger s rostoucím zpožděním. Pokud např. bude poměr frekvencí 10.01, zpozdí se vzorkování o 10 celých period a o $0.01T$ oproti předchozímu vzorku. Pokud je to signál periodický a stabilní, 10 period nehraje roli, ale díky posunu o $0.01T$ se postupně navzorkuje celá perioda.



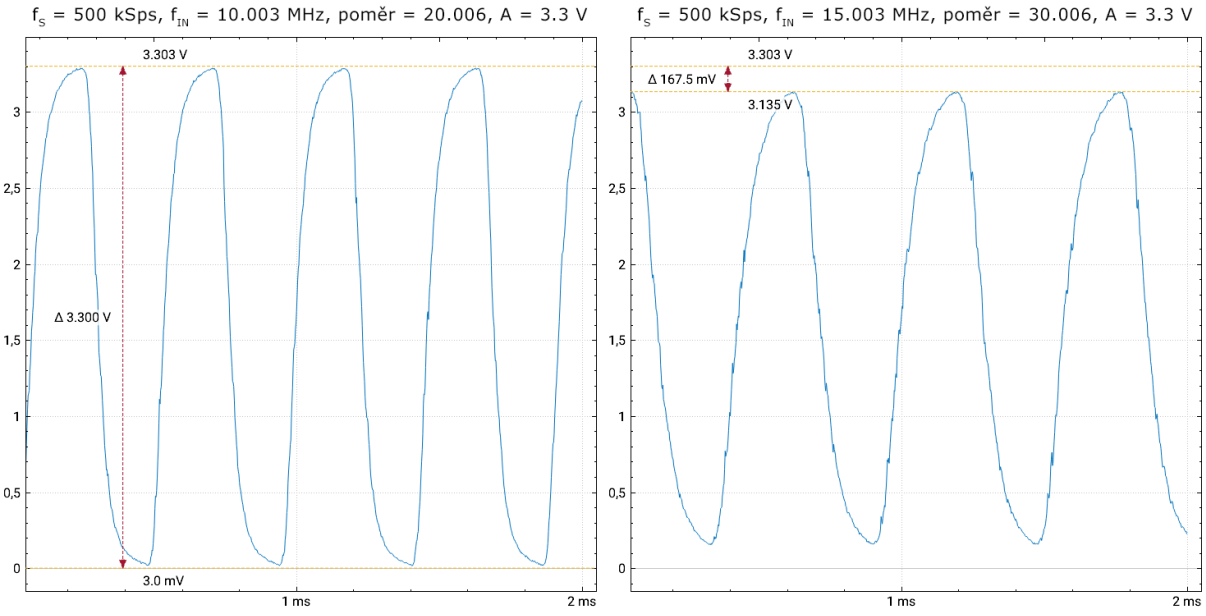
obr 5.18: Princip stroboskopické vzorkování (vzorkování v ekvivalentním čase) realizované určitým poměrem vstupní a vzorkovací frekvence

Na vstup ADC STM32F103 byl přiveden obdélníkový signál o frekvenci 10.003 MHz (periodě 99.97 ns) a amplitudě 3.3 V. Vzorkovací frekvence byla nastavena na 500 kSps, jejich poměr tedy činil 20.006. Doba vzorkování ADC byla nastavena na 625 ns. To znamená, že se odebírá vzorek z každé 20. periody. Vzorkování je při tom režimu Track and Hold, protože během doby, kdy je připojen vzorkovací kondenzátor, se změní zhruba 6 period signálu.

Pro vstupní frekvenci 10.003 MHz byla amplituda stále 3.3 V. Se zvyšující se frekvencí ale amplituda začala mírně klesat a při 15.003 MHz byl již pokles cca 160 mV. Jelikož je šířka pásma dle vzorce cca 19.9 MHz, lze tento pokles vysvětlit nekvalitním generátorem signálu a nedostatečným impedančním přizpůsobením. Pro ověření šířky pásma ADC by bylo nutné použít kvalitní vysokofrekvenční generátor a správně signál impedančně přizpůsobit vstupu.



obr 5.19: Srovnání klasického vzorkování (vlevo) a stroboskopického vzorkování (vpravo) obdélníkového signálu o amplitudě 3.3 V



obr 5.20: Pokles amplitudy při stroboskopickém vzorkování obdélníkového signálu už při frekvenci 15.003 MHz kvůli nedostatečnému impedančnímu přizpůsobení

Kapitola 6

Zhodnocení výsledků práce

Úkolem bylo vytvořit softwarově definovaný osciloskop na platformě STM32, který má cílit na hromadnou distanční výuku praktické elektroniky. Kromě osciloskopu měly být vytvořeny také další přístroje jako logický analyzátor, voltmetr, čítač, PWM a signálový generátor. Primárně měl být osciloskop použitelný na mikrořadiči STM32F103 a vývojové desce BluePill. Dále měla být ověřena funkčnost na STM32 řadách F3, L0, L4 nebo G0.

- Analýza zadání (2)
- Plán realizace a analýza požadavků (3)
  - Požadavky na periferie (3.1.1)
  - Analýza ADC (3.1.1)
  - Požadavky na firmware (3.1.2)
  - Požadavky na PC aplikaci (3.1.3)
  - Analýza komunikačního rozhraní (3.1.4)
- Řešení realizace (4)
  - Fáze 1 – návrh komunikačního rozhraní (4.1)
  - Fáze 2 – vývoj firmware (4.2)
  - Fáze 3 – vývoj PC aplikace (4.3)
  - Fáze 4 – testování a nasazení (4.4)
- Dosažené výsledky a naměřené parametry (5)
  - Parametry a pinout přístrojů (5.1)
  - Představení PC aplikace (5.2)
  - Ověření šířky pásma ADC pomocí stroboskopického vzorkování (5.3)

Nejprve jsem analyzoval zadání a získal tak obecný přehled o tom, na co se zaměřit a čeho se vyvarovat. Následně jsem zpracoval analýzu požadavků, kterou standardně začíná každý projekt. V tomto dokumentu jsem si s vedoucím práce ujasnil požadované parametry a funkcionalitu jednotlivých přístrojů. Zpracoval jsem také přehled mikrořadičů STM32 s jejich parametry, na které bude osciloskop EMBO portován.

Dále jsem pokračoval hlubší analýzou jednotlivých segmentů. Začal jsem požadavky na periferie MCU, který vycházel z blokového schématu celého osciloskopu (viz. obrázek 3.2). Následovala detailní analýza SAR AD převodníku STM32F103. Definoval jsem parametr maximální přípustní impedance zdrojového signálu a stanovil jsem hodnotu ENOB pomocí měření SNR. Oproti ideálnímu 12-bit ADC s kvantizační chybou ± 0.4 mV jsem naměřil pro F103 přibližně kvantizační chybu ± 0.8 mV.

- maximální přípustná impedance zdrojového signálu ADC F103 → 73.4 Ω pro max f_s
- ENOB ideálního 12-bit ADC = 12 → pro F103 změřeno ≈ 11
- kvantizační chyba ideálního 12-bit ADC = ± 0.4 mV → pro F103 změřeno ≈ ± 0.8 mV

Následovala analýza vývoje firmware, PC aplikace a komunikačního rozhraní. Pro vývoj firmware jsem zvolil programovací jazyk C, STM32CubeIDE, debugger SEGGER Ozone, ovladače ST LL, operační systém FreeRTOS a další specifické parametry, které je nutné definovat před zahájením vývoje. Pro vývoj PC aplikace jsem zvolil multiplatformní framework Qt, jazyk C++ a zobrazovací knihovnu QCustomPlot. Komunikační rozhraní jsem se rozhodl postavit na otevřeném standardu SCPI a textovém protokolu s možností přenosu binárních dat. Připojení bude realizováno pomocí nativního USB (BluePill) nebo pomocí UART a ST-LINK (Nucleo).

- firmware – jazyk C, STM32CubeIDE, SEGGER Ozone, ovladače ST LL, FreeRTOS
- PC aplikace – jazyk C++, multiplatformní framework Qt, knihovna QCustomPlot
- komunikace – SCPI, USB a UART, textový protokol s možností přenosu binárních dat

Po analýze jsem začal pracovat na samotné realizaci osciloskopu. Nejprve jsem definoval SCPI protokol, tedy příkazy, dotazy a odpovědi pro jednotlivé přístroje. Určil jsem také hlavní charakter komunikace. Zavedl jsem asynchronní zprávu Ready a rozhodl se použít volně dostupnou SCPI knihovnu od pana Breuera. [27]

- systém a obecné příkazy – :SYS, *IDN?, *RST, *STB?, *CLS
- primární přístroje – :SCOP, :VM, :LA (osciloskop, logický analyzátor, voltmetr)
- sekundární přístroje – :CNTR, :PWM, :SGEN (čítač, PWM a signálový generátor)

Dále jsem několik měsíců pokračoval vývojem firmware. Nejdůležitější bylo správně navrhnout architekturu celého systému. Strávil jsem tedy mnoho času kreslením vývojových diagramů, což se nakonec ukázalo jako klíč úspěchu. Navrhl jsem systém, kde veškerou funkcionalitu řeší 5 FreeRTOS tasků a 5 modulů.

Vyvinul jsem firmware, který je snadné portovat na ostatní mikrořadiče STM32 pomocí konfiguračních souborů, které jsou tvořeny převážně makry. Pomocí maker a CubeMX konfigurátoru se dá rychle přizpůsobit danému MCU. Firmware jsem odladil a plně zprovoznil pro MCU STM32F103C8, STM32F103RE, STM32F303RE a STM32L412KB. Pro MCU STM32G031 a STM32L072KZ, které jsou postavené na jádru Cortex-M0+, je ještě nutné optimalizovat některé funkce a opravit kritickou chybu při operaci s FreeRTOS semafory.

- 5 FreeRTOS tasků
- 5 firmware modulů (DAQ, PWM, SGEN, CNTR, COMM)
- každý modul a task je zdokumentovaný pomocí UML diagramů
- portování na ostatní řady STM32 pomocí jednoho konfiguračního souboru s makry
- plná podpora: STM32F103C8, STM32F103RE, STM32F303RE, STM32L412KB
- experimentální podpora: STM32G031K8, STM32G031J6, STM32L072KZ

Poslední hlavní část této práce bylo vyvinout PC aplikaci. Podobně jako při vývoji firmware jsem zpočátku věnoval několik týdnů návrhu architektury a kreslení UML diagramů. Největší důraz jsem kladl na komunikační jádro jménem Core, které má za úkol příjem a odesílání zpráv. Byl navržen OOP systém tříd, který řeší SCPI komunikaci pomocí virtuálních slotů.

Po odladění jádra, kdy jsem věděl, že je komunikace stabilní, jsem začal s grafickým návrhem. Vytvořil jsem vlastní CSS design celé aplikace, který je postaven na azurově-zelené barvě a odstínech šedé. Dále jsem vytvořil vlastní logo a použil desítky ikon a obrázků, většinu z nich na míru upravených.

Posledním krokem bylo vytvořit GUI pro hlavní okno aplikace a 6 přístrojů. Inspirací byl osciloskop Tektronix TBS řady 2000. Ovládací prvky byly zvoleny a rozmístěny tak, aby se uživateli pracovalo komfortně. Kromě hlavní funkcionality byly implementovány taky doplňkové funkce jako FFT spektrální analýza, průměrování, interpolace, export dat, XY režim, automatická aktualizace přes internet nebo instalátor pro Windows.

- PC aplikace je zkompilována pro Windows 7, Xubuntu 18 a macOS Catalina
- vlastní grafický design
- kritická funkcionalita je zdokumentována pomocí UML diagramů
- většina parametrů přístrojů je plně konfigurovatelná
- doplňkové funkce: FFT, průměrování, interpolace, export, XY režim ...

Výsledek mé práce je osciloskop EMBO, který tvoří 3 primární přístroje (osciloskop, logický analyzátor, voltmetr) a 3 sekundární přístroje (čítač, PWM a signálový generátor). Multiplatformní PC aplikace je spustitelná na operačních systémech Windows, Linux a macOS. Odladěný firmware je dostupný pro STM32F103C8, STM32F103RE, STMF32F303RE a STM32L412KB ve formátu HEX, další MCU (STM32G031 a STM32L072KZ) jsou zatím podporovány experimentálně. Osciloskop EMBO má sloužit jako levný a přístupný nástroj pro studenty při hromadné distanční výuce praktické elektroniky.

Firmware zkompilován a otestován pro:

- STM32F103C8
- STM32F103RE
- STM32F303RE
- STM32L412KB

Maximální naměřené parametry jednotlivých přístrojů u řad STM32F103C8 a STM32F303RE:

- osciloskop
  - F103C8 – 4 kanály na 1 ADC, 0.800 MSps, 10 kB
  - F303RE – 4 kanály na 4 ADC, 4.965 Msps, 50 kB
- logický analyzátor
  - F103C8 – 4 kanály, 5.142 MSps, 10 kB
  - F303RE – 4 kanály, 14.400 Msps, 50 kB
- voltmetr
  - F103C8 – 4 kanály, 100 Sps (fixní)
  - F303RE – 4 kanály, 100 Sps (fixní)
- čítač
  - F103C8 – 1 kanál, 33 MHz
  - F303RE – 1 kanál, 57 MHz
- PWM generátor
  - F103C8 – 2 kanály, 14.4 MHz
  - F303RE – 2 kanály, 20.6 MHz
- signálový generátor
  - F103C8 – nedostupný
  - F303RE – 1 kanál, 4.5 MSps (generování po bodech)

Kapitola 7

Závěr

Cílem této diplomové práce bylo vytvořit softwarově definovaný osciloskop pro STM32F103 BluePill, který bude možno použít při distanční výuce praktické elektroniky. Správnost návrhu jsem měl ověřit implementací pro další řady STM32. Kromě osciloskopu bylo úkolem také prověřit možnost implementace čítače nebo signálového generátoru.

Všechny zadané úkoly byly splněny. Vytvořil jsem osciloskop jménem EMBO, který zahrnuje 3 primární přístroje (4-kanálový osciloskop, logický analyzátor, voltmetr) a 3 sekundární přístroje (čítač, PWM a signálový generátor). Vyvinul jsem firmware pro MCU STM32F103C8, STM32F103RE, STM32F303RE a STM32L412KB a multiplatformní Qt aplikaci, kterou jsem zkompiloval pro operační systémy Windows 7, Xubuntu 18 a macOS Catalina. Osciloskop je připraven na použití při hromadné distanční výuce.

Pro levnou desku BluePill s mikrořadičem STM32F103C8 dosahuje EMBO vzorkovací frekvence 800 kSps a disponuje celkovou pamětí 10 kB. Pro dražší vývojovou desku Nucleo s mikrořadičem STM32F303RE jsou tyto hodnoty mnohem vyšší, vzorkovací frekvence je 4.965 MSps na kanál a celková paměť činí 50 kB.

Literatura

[1] J. Pařez, "EMBO veřejný repozitář." Dostupné z http://github.com/parezj/EMBO.

[2] S. Microelectronics, "Datasheet for STM32F103C8." Dostupné z https://www.st.com/resource/en/datasheet/stm32f103c8.pdf.

[3] S. Microelectronics, "Datasheet for STM32F303RE." Dostupné z https://www.st.com/resource/en/datasheet/stm32f303re.pdf.

[4] S. Microelectronics, "Datasheet for STM32L412KB." Dostupné z https://www.st.com/resource/en/datasheet/stm32l412kb.pdf.

[5] S. Microelectronics, "Reference manual for STM32F101xx, STM32F102xx, STM32F103xx, STM32F105xx andSTM32F107xx." Dostupné z https://www.st.com/resource/en/reference_manual/cd00171190-stm32f101xx-stm32f102xx-stm32f103xx-stm32f105xx-and-stm32f107xx-advanced-armbased-32bit-mcus-stmicroelectronics.pdf.

[6] S. Microelectronics, "Reference manual for STM32F303xB/C/D/E, STM32F303x6/8, STM32F328x8,STM32F358xC, STM32F398xE." Dostupné z https://www.st.com/resource/en/reference_manual/dm00043574-stm32f303xbcde-stm32f303x68-stm32f328x8-stm32f358xc-stm32f398xe-advanced-armbased-mcus-stmicroelectronics.pdf.

[7] S. Microelectronics, "Reference manual for STM32L41xxx, 42xxx, 43xxx, 44xxx, 45xxx, 46xxx." Dostupné z https://www.st.com/resource/en/reference_manual/dm00151940-stm32l41xxx42xxx43xxx44xxx45xxx46xxx-advanced-armbased-32bit-mcus-stmicroelectronics.pdf.

[8] TI, "50-Ohm 2-GHz oscilloscope front-end reference design." Dostupné z https://www.ti.com/lit/ug/tiduba4/tiduba4.pdf.

[9] J. Yiu, *The definitive guide to the ARM Cortex-M3 Joseph Yiu.* Amsterdam ; Boston: Newnes, 2010.

[10] S. Microelectronics, "Reference manual for ultra-low-power STM32L0x2." Dostupné z https://www.st.com/resource/en/reference_manual/dm00108281-ultralowpower-stm32l0x2-advanced-armbased-32bit-mcus-stmicroelectronics.pdf.

[11] C. Noviello, *Mastering STM32 A step-by-step guide to the most complete ARM Cortex-M platform, using a free and powerful development environment based on Eclipse and GCC.* British Columbia ; Canada: Leanpub, 2018.

[12] J. Yiu, "A beginner's guide on interrupt latency." Dostupné z https://community.arm.com/developer/ip-products/processors/b/processors-ip-blog/posts/beginner-guide-on-interrupt-latency-and-interrupt-latency-of-the-arm-cortex-m-processors.

[13] S. Microelectronics, "Reference manual for STM32G0x1." Dostupné z https://www.st.com/resource/en/reference_manual/dm00371828-stm32g0x1-advanced-armbased-32bit-mcus-stmicroelectronics.pdf.

[14] S. Microelectronics, "General-purpose timer cookbook for STM32 microcontrollers." Dostupné z https://www.st.com/resource/en/application_note/dm00236305-generalpurpose-timer-cookbook-for-stm32-microcontrollers-stmicroelectronics.pdf.

[15] S. Microelectronics, "STM32 cross-series timer overview." Dostupné z https://www.st.com/resource/en/application_note/dm00042534-stm32-crossseries-timer-overview-stmicroelectronics.pdf.

[16] Maxim, "Understanding SAR ADCs." Dostupné z https://www.maximintegrated.com/en/design/technical-documents/tutorials/1/1080.html.

[17] A. Devices, "What the nyquist criterion means to your sampled data system design." Dostupné z https://www.analog.com/media/en/training-seminars/tutorials/mt-002.pdf.

[18] S. Labs, "Calculating settling time forswitched capacitor ADCs." Dostupné z https://www.silabs.com/documents/public/application-notes/AN119.pdf.

[19] S. Microelectronics, "How to get the best ADC accuracy in STM32 microcontrollers." Dostupné z https://www.st.com/resource/en/application_note/cd00211314-how-to-get-the-best-adc-accuracy-in-stm32-microcontrollers-stmicroelectronics.pdf.

[20] A. Devices, "Seven steps to successful analog-to-digital signal conversion (noise calculation for proper signal conditioning)." Dostupné z https://www.analog.com/en/technical-articles/seven-steps-to-successful-analog-to-digital-signal-conversion.html.

[21] Microsoft, "What's new in .NET 5." Dostupné z https://docs.microsoft.com/cs-cz/dotnet/core/dotnet-five.

[22] L. Torvalds, "Linux kernel github." Dostupné z https://github.com/torvalds/linux.

[23] S. O. Insights, "2020 developer survey." Dostupné z https://insights.stackoverflow.com/survey/2020.

[24] T. Q. Company, "Qt for beginners." Dostupné z https://wiki.qt.io/Qt_for_Beginners.

[25] L. Torvalds, "Linus torvalds on C++." Dostupné z http://harmful.cat-v.org/software/c++/linus.

[26] E. Eichhammer, "QCustomPlot 2.1.0 documentation." Dostupné z https://www.qcustomplot.com/documentation/index.html.

[27] J. Breuer, "SCPI parser library v2." Dostupné z https://github.com/j123b567/scpi-parser.

Příloha A

Obsah přiloženého CD

- Diplomová práce ve formátu PDF
- Zdrojový kód EMBO firmware
- Zdrojový kód EMBO PC aplikace
- Uživatelská dokumentace ve formátech CHM a PDF
- Zkompilovaná PC aplikace pro Windows 7, Xubuntu 18 a macOS Catalina
- Firmware ve formátu HEX pro STM32F103, STM32F303 a STM32L412