VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

DEPARTMENT OF CONTROL AND INSTRUMENTATION

ČASOVĚ KRITICKÉ APLIKACE NA STM32MP157

TIME CRITICAL APPLICATIONS ON STM32MP157

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — Bc. Jakub Kouřil

AUTHOR

VEDOUCÍ PRÁCE — Ing. Lukáš Pohl, Ph.D.

SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Kybernetika, automatizace a měření**

Ústav automatizace a měřicí techniky

Student: Bc. Jakub Kouřil ***ID:*** 220992

Ročník: 2 ***Akademický rok:*** 2023/24

NÁZEV TÉMATU:

Časově kritické aplikace na STM32MP157

POKYNY PRO VYPRACOVÁNÍ:

1) Proveďte rešerši o procesorové řadě STM32MP1.
2) Nainstalujte na vývojový jednodeskový počítač os Linux podporovaný výrobcem.
3) Seznamte se s ekosystémem pro vývoj aplikací pro STM32MP157, demonstrujte funkčnost na poskytnuté demo aplikaci.
4) Vytvořte pro jednodeskový počítač řídicí aplikaci pracující s vzorkovací frekvencí >16kHz.
5) Vytvořte pod os Linux nadřazenou aplikaci pracující na řádově menší vzorkovací frekvenci, která si bude s podřízeným jádrem předávat naměřená data a instrukce.
6) Analyzujte latence takto vytvořeného řetězce pomocí data loggeru.

DOPORUČENÁ LITERATURA:

LIBERAL de los RÍOS, Alberto, 2021. Linux Driver Development with Raspberry Pi - Practical Labs. ⁋ Independently published, ISBN 979-8516120688.

Termín zadání: 5.2.2024 ***Termín odevzdání:*** 15.5.2024

Vedoucí práce: Ing. Lukáš Pohl, Ph.D.

doc. Ing. Petr Fiedler, Ph.D.
předseda rady studijního programu

ABSTRAKT

Tato práce se zabývá testováním možnosti využití procesorů řady STM32MP15x pro provoz časově kritických aplikací. Testování je realizováno za použití vývojového modulu STM32MP157F-DK2 pro který byla vyvinuta aplikace pro vektorové řízení motoru vyžadující přesné časování regulačních smyček. Pro analýzu latencí v systému byly v programu ukládány a vypisovány časové značky.

KLÍČOVÁ SLOVA

STM32, STM32MPU1, STM32MPU157, vektorové řízení motoru, časově kritické aplikace, Linux, OpenSTLinux

ABSTRACT

The goal of this thesis is to test the viability of using processors from the STM32MP15x series to run time critical applications. Testing was performed using a STM32MP157F-DK2 discovery kit. An application for field oriented control requiring precise timing was developed for the module to test its properties. Latency in the system was measured by storing and logging timestamps during execution.

KEYWORDS

STM32, STM32MPU1, STM32MPU157, field oriented control, time critical applications, Linux, OpenSTLinux

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Bc. Jakub Kouřil
VUT ID autora:	220992
Typ práce:	Diplomová práce
Akademický rok:	2023/24
Téma závěrečné práce:	Časově kritické aplikace na STM32MP157

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Ing.Lukáši Pohlovi, Ph.D.
především za nekonečnou trpělivost při konzultacích během vypracovávání této práce.

Obsah

Seznam obrázků

Seznam výpisů

Úvod

Výrobce mikročipů STMicroelectronics v minulých letech rozšířil své portfolio o mikroprocesory řady STM32MP1. Oproti ostatním mikrokontrolerům STM32 které tento výrobce poskytoval do této doby s sebou zmíněná nová zařízení přinesla operační systém založený na Linux jádře. Tento operační systém je založený na mnohých open-source komponentách a v případě potřeby je možné jej modifikovat pro konkrétní využití.

Cílem této práce je prozkoumat možnosti které nabízí mikroprocesory série STM32MP15x v rámci provozu časově kritickým aplikací. Tato zařízení obsahují hlavní procesor (Cortex -A7) hostující operační systém a koprocesor (Cortex-M4) který je řízen pomocí hlavního procesoru a může se plně věnovat časově kritickým procesům. Tato možnost dělit aplikace mezi heterogenní jádra může kombinovat výhody operačních systémů společně s výhodami mikrokontrolérů nepoužívajících operační systémy a tudíž nezatížených nedeterministickou režií spojenou s provozem operačních systémů. Architektura tohoto mikroprocesoru tak může přinést zejména lepší vlastnosti pro provoz časově kritických částí aplikací. Procesory této řady je také možné použít na aplikace využívající umělou inteligenci, při kterých s podporou knihoven a nástrojů výrobce lze například hostovat neuronové sítě na koprocesoru a nezatěžovat tak hlavní procesor vyhodnocováním jejich výstupu.

Druhým cílem této práce je vytvořit podklady použitelné při zprovozňování procesorů řady STM32MP15x do provozu a pro vývoj aplikací pro tato zařízení. V práci jsou tak popsány i softwarové nástroje poskytované výrobcem pro vývoj s použitím této řady procesorů.

Během této práce proběhlo uvedení do provozu jak nástrojů poskytovaných výrobcem, tak modulu STM32MP157F-DK2 který využívá mikroprocesor STM32MP157. Byly otestovány demonstrační aplikace od výrobce a také bylo vytvořeno několik programů pro testování vlastností procesoru. Konečným cílem diplomové práce je zhodnocení vlastností této nové řady mikroprocesorů při provozu časově kritických aplikací na vytvořené aplikaci.

První kapitola této práce se věnuje popisu procesorové řady STM32MP1 a vlastnostem jednotlivých kategorií zařízení uvnitř této řady. Druhá kapitola popisuje vývojový ekosystém přichystaný STMicroelectronics pro testování a tvorbu nových aplikací pro mikroprocesor. Třetí kapitola popisuje demonstrační projekty které jsou součástí distribuce. Čtvrtá kapitola se věnuje testování rychlosti výměny dat mezi jednotlivými procesory uvnitř zařízení. Pátá kapitola obsahuje popis návrhu časově kritické aplikace pro zařízení a poslední dvě kapitoly pojednávají o procesu implementace částí této aplikace pro jednotlivé procesory Cortex-M4 a Cortex-A7.

1 Procesorová řada STM32MP1

Mikroprocesory řady STM32MP1 jsou určeny pro obecné použití a dají se využít při řadě aplikací. Pro některá potencionální nasazení tohoto zařízení může být výhodné zaměření těchto procesorů na spolehlivost a zabezpečení dat. Zařízení disponují množstvím periferií pro komunikaci a mohou tak nalézt využití například i v průmyslových aplikacích. V této procesorové řadě výrobce STMicroelectronics nabízí dvě kategorie zařízení: STM32MP13x a STM32MP15x. [1]

1.1 Procesory STM32MP13x

Mikroprocesory z kategorie STM32MP13x obsahují jeden Cortex-A7 procesor s taktem až 1 GHz a nabídku periferií dle konkrétního typu jednotky.[1] Procesory této kategorie jsou dle výrobce určeny například pro nízkoenergetické aplikace, automatizaci nebo vývoj komponent pro inteligentní domy, lze je však využít pro libovolný účel.

Společně s tímto procesorem je možné používat Linux distribuci OpenSTLinux, uživatel však má možnost použít i libovolný RTOS (Real-Time Operating System). Pro procesor také existuje podpora pro hostování neuronových sítí s pomocí nástrojů určených pro Linux distribuci udržovanou výrobcem.

1.2 Procesory STM32MP15x

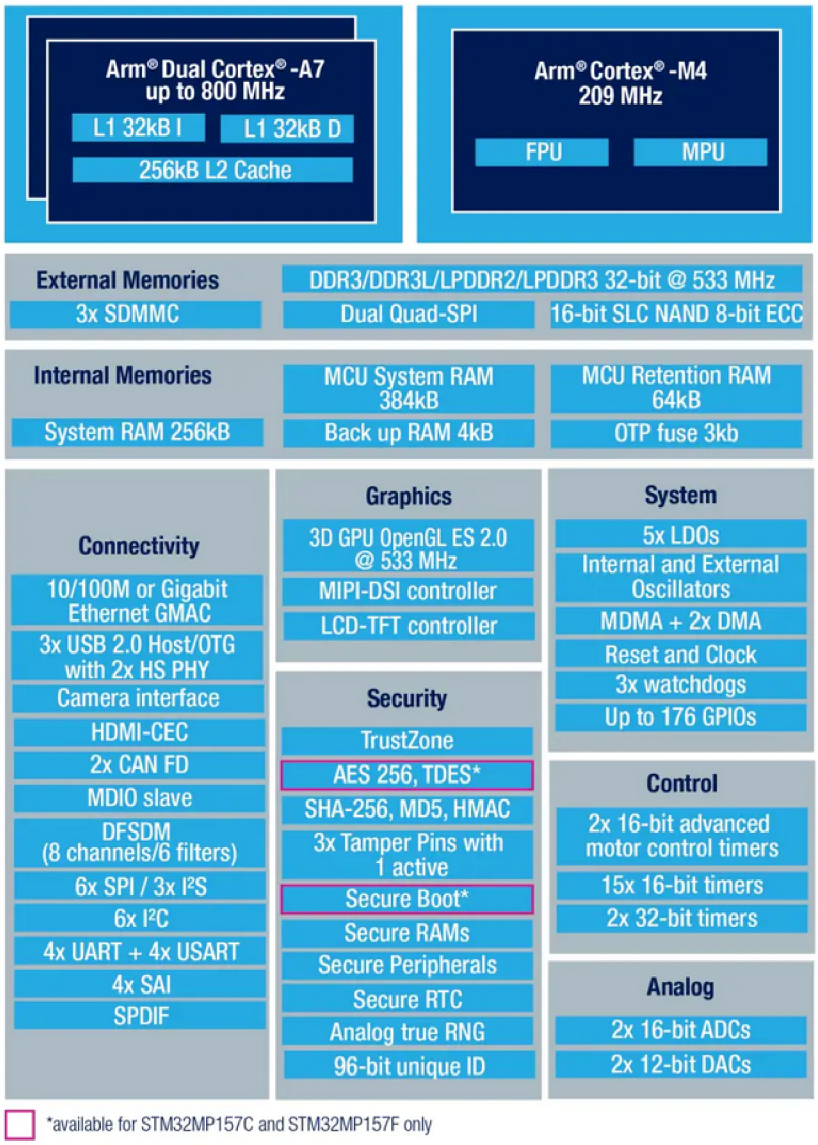
Oproti procesorům kategorie STM32MP13x jsou zařízení z této sekce vybavena dvěma heterogenními procesory: jedno-jádrovým Cortex-M4 a až dvou-jádrovým Cortex-A7.[1]

V porovnání s předchozí kategorií jsou procesory STM32MP15x určeny pro aplikace vyžadující vyšší výkon a zlepšené vlastnosti při provozu časově kritických aplikací díky možnosti přesunout části aplikace do programu běžícího v koprocesoru. Pro programy hostované na procesoru Cortex-M4 je jednodušší dosáhnout precizního časování.

Cortex-A7 v tomto zařízení hraje roli hlavního procesoru, zatímco Cortex-M4 je považován za koprocesor řízený z Cortex-A7 za pomocí periferie IPCC (Inter-Processor Communication Controller), která zajišťuje výměnu dat mezi procesory a bude jí věnována jedna z následujících kapitol. Hlavní procesor řídí nahrávání programu do koprocesoru a také řídí jeho spouštění pomocí řídicích zpráv odesílaných za pomoci této periferie. V porovnání s řadou procesorů STM32MP13x jsou procesory Cortex-A7 taktovány nižším taktem s omezením do 800 MHz , což je

kompenzováno možností zvolit verzi se dvěma jádry a také přidaným koprocesorem Cortex-M4 s taktovací frekvencí 209 MHz.

Cortex-A7 i Cortex-M4 sdílejí připojení k periferiím procesoru jako časovače, ADC převodníky a paměti. Struktura STM32MP157 a dostupné periferie procesoru jsou zobrazeny na obrázku č. 1.1. Při programování tohoto procesoru je potřeba použité periferie přiřadit jednomu z těchto procesorů, jinak by mohlo dojít ke konfliktu v přístupu k nim. Prakticky je tohoto oddělení přístupu k periferiím dosaženo úpravou stromu zařízení pro Linux, což zajistí že hlavní procesor nebude zasahovat do periferií používaných v koprocesoru. Existují však i periferie, které oba procesory sdílejí současně. Příkladem sdílené periferie je IPCC.



Obr. 1.1: ST32MP157 struktura [2]

1.2.1 Výměna dat mezi jádry

Mikroprocesory STM32MP15x používají pro výměnu dat mezi hlavním procesorem a koprocesorem sdílený paměťový prostor společně s dedikovanou periferií IPCC poskytující podpůrné funkce pro předávání dat [3].

Procesory mohou zapisovat do sdílené sekce RAM paměti data a přes IPCC pak signalizovat druhému jádru že jsou data připravena ke zpracování či že jejich zpracování již proběhlo.

V jednotlivých procesorech je komunikace řízena ovladači. Pro Cortex-A7 se jedná o mailbox framework, na straně Cortex-M4 pak jde o jednoduchý HAL (Hardware Access Layer) ovladač [4].

IPCC periferie

IPCC periferie nabízí šest obousměrných kanálů pro výměnu dat vybavených signalizací stavu přenosu pro obě jádra. Periferie samotná nijak nezasahuje do paměti ale pouze vystavuje signály indikující stav odesílání a přijímání dat mezi jádry. jednotlivé kanály obsahují dvě jednosměrné linky pro signalizaci stavu přenosu dat. Ke každé z linek přísluší stavový signál specifikující zda je linka obsazená doposud nevyčtenými daty nebo volná pro další zápis. Manipulací tohoto signálu, kde odesílající jádro jej nastavuje při odeslání zprávy a přijímající jádro tento signál resetuje po jejím zpracování, je řízen přenos dat. [4]

Periferie generuje signál přerušení pro jádra na základě zmíněného signálu. Výsledkem jsou signály přerušení:

- **RXO** - RX channel occupied (signál pro příjemce)
- **TXF** - TX channel free (signál pro odesílatele)

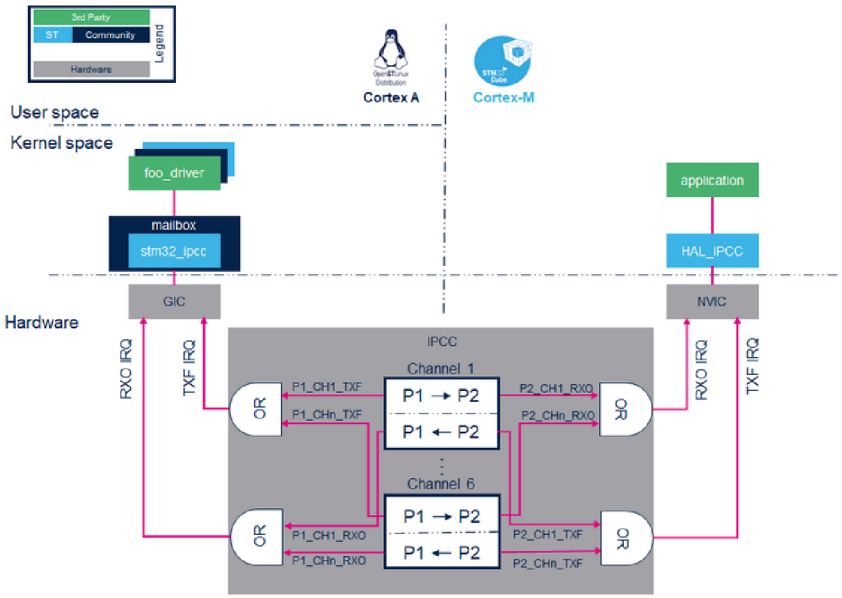
Architektura periferie, signály pro generování přerušení a rozhraní mezi periferií a softwarem jader jsou znázorněny v obrázku 1.2.

Dostupné jsou tři módy operace: Simplex, Half-Duplex a Full-Duplex. V Simplex režimu je využita jen jedna linka z kanálu a komunikace je tak pouze jednosměrná. V Half-Duplex módu je také využita jen jedna z linek, ale data do ní mohou zapisovat obě jádra a komunikace je obousměrná. Full-Duplex režim využívá obou linek v kanálu a jádra tak mezi sebou mohou asynchronně a obousměrně posílat data.

Z 6 kanálů dostupných v periferii IPCC jsou 3 využity pro přednastavené funkce

- **Kanál 1** - RPMsg komunikace z Cortex-M jádra do Cortex-A
- **Kanál 2** - RPMsg komunikace z Cortex-A jádra do Cortex-M
- **Kanál 3** - jednosměrné odesílání příkazů k vypnutí jádra Cortex-M4

Zbylé tři kanály jsou ponechány nevyužité a lze je použít pro uživatelské potřeby.



Obr. 1.2: Architektura a využití IPCC periferie [4]

Linux mailbox framework

Tento systém je součástí jádra Linux systému běžícím na jádru Cortex-A7. Mailbox framework se skládá ze dvou vrstev:

- mailbox controller - ovladač pro IPCC periferii který je specifický pro použitý hardware, ale nabízí standardní API (Application Programming Interface) pro nadřazeného klienta. Pro procesor STM32MP157 se jedná o ovladač stm32-ipcc.
- mailbox client - klient obstarávající odesílání a přijímání dat

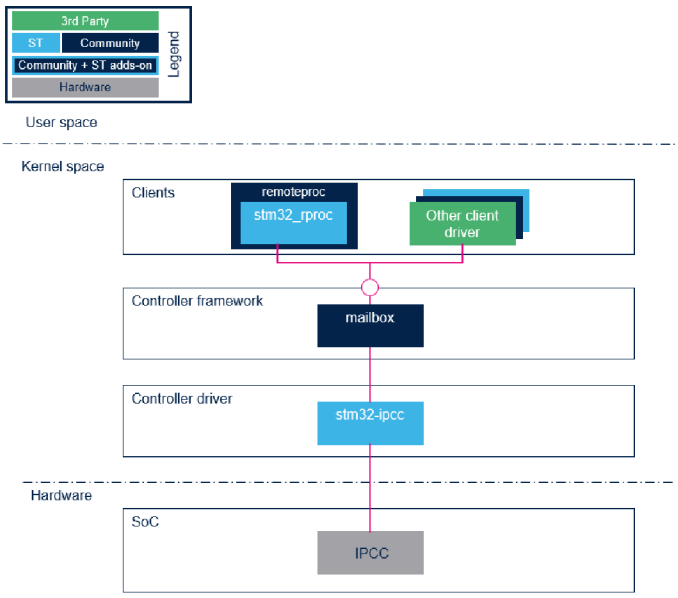
Struktura mailboxu je zobrazena na obrázku 1.3, kde je také uveden klient pro RPMsg protokol, kterým je remoteproc ovladač. Uživatel si v případě potřeby může vytvořit vlastní mailbox client s potřebnou funkcionalitou. [5]

RPMsg framework

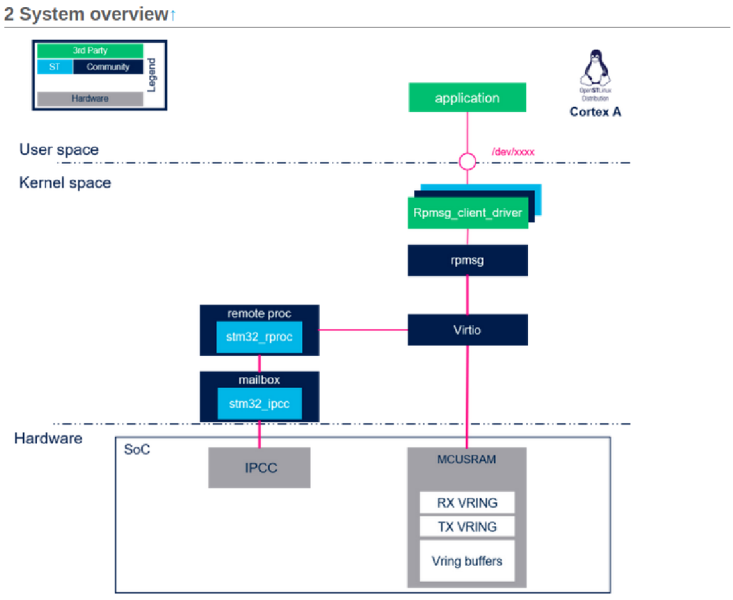
RPMsg (Remote Processor Message) je poslední vrstvou mezi IPCC a uživatelem. RPMsg framework využívá ovladač Virtio který spravuje paměť a buffery pro výměnu dat mezi procesory a zároveň řídí mailbox framework. Zde dochází k propojení skutečného paměťového místa pro předávání zpráv a ovladače řídícího periferii IPCC. [6]

Strukturu celého komunikačního kanálu vytvořeného pomocí RPMsg framework je možné sledovat na obrázku č. 1.4, kde je zobrazena i role Virtio ovladače.

Výsledný komunikační kanál se v operačním systému projevuje jako soubor



Obr. 1.3: Struktura mailbox framework [5]



Obr. 1.4: Struktura RPMsg framework [6]

#/dev/ttyRPMSGx připravený pro čtení/zápis. K tomuto souboru mohou přistupovat jak uživatelé tak aplikace spuštěné v operačním systému a komunikovat tak s koprocesorem.

Důležitou poznámkou pro úspěšné používání tohoto systému pro výměnu dat mezi procesory je fakt že první zprávu musí odeslat hlavní procesor, respektive

zprávy odeslané koprocesorem před momentem kdy hlavní procesor odešle svou první zprávu budou ztraceny.

1.2.2 Discovery kit STM32MP157F-DK2

Tento modul od STMicroelectronics je určený pro prototypování a experimentaci s mikroprocesorem STM32MP157. Modul umožňuje přístup k řadě rozhraní která může procesor využívat, od dostupných pinů pro analogově-digitální převodníky a GPIO, až po komplexnější rozhraní jako USB, Ethernet, HDMI nebo dotykovou obrazovku dodávanou jako součást modulu. Vzhled modulu je možné vidět na obrázku č. 1.5.

Obr. 1.5: ST32MP157C-DK2 modul [7]

Modul ze spodní strany poskytuje konektor shodný s prototypovacími deskami Arduino a umožňuje tak použití množství zařízení určených přímo pro Arduino moduly. Mimo toto rozhraní je na modulu pod obrazovkou osazen GPIO expandér s jehož pomocí lze získat přístup k dalším periferiím mikroprocesoru.

Na desce modulu je osazen debugovací čip ST-Link který zprostředkovává připojení k procesoru v Engineering režimu ve kterém poskytuje možnost ladit program nahraný do procesoru Cortex-M4 v kombinaci s vývojovým prostředím. ST-Link také vytváří spojení pro připojení k terminálu operačního systému spuštěném na hlavním procesoru pokud je modul v Production módu.

Přepínání mezi Enginerring a Production režimy je na Discovery Kitu realizováno pomocí přepínačů **BOOT0** a **BOOT2** na spodní straně prototypovací desky. Modul také obsahuje signál BOOT1, jeho manipulace však není zpřístupněna pro

uživatele. Pokud jsou oba přepínače v poloze ON, při resetu desky se spustí bootloader a bude spuštěna Linux distribuce nahraná na SD kartu připojenou k modulu. V případě, že je BOOT0 v poloze OFF a BOOT2 v poloze ON, bootloader se při resetu nebude spouštět a je možné skrze ST-Link přítomný na desce nahrávat a ladit program pro koprocesor Cortex-M4. Když se oba přepínače nacházejí v poloze OFF, je možné se skrze program CubeProgrammer připojit k modulu a nahrát na SD kartu novou Linux distribuci.

Modul nabízí Ethernet port k připojení do sítě, pro toto zařízení je vhodné vytvořit zcela oddělenou lokální síť nebo jej jiným způsobem zajistit proti vnějšímu přístupu ze zařízení neurčených pro práci s modulem. Platforma není sama o sobě v základní konfiguraci zabezpečena a bez těchto opatření jakýkoli uživatel lokální sítě by získal plný přístup k zařízení.

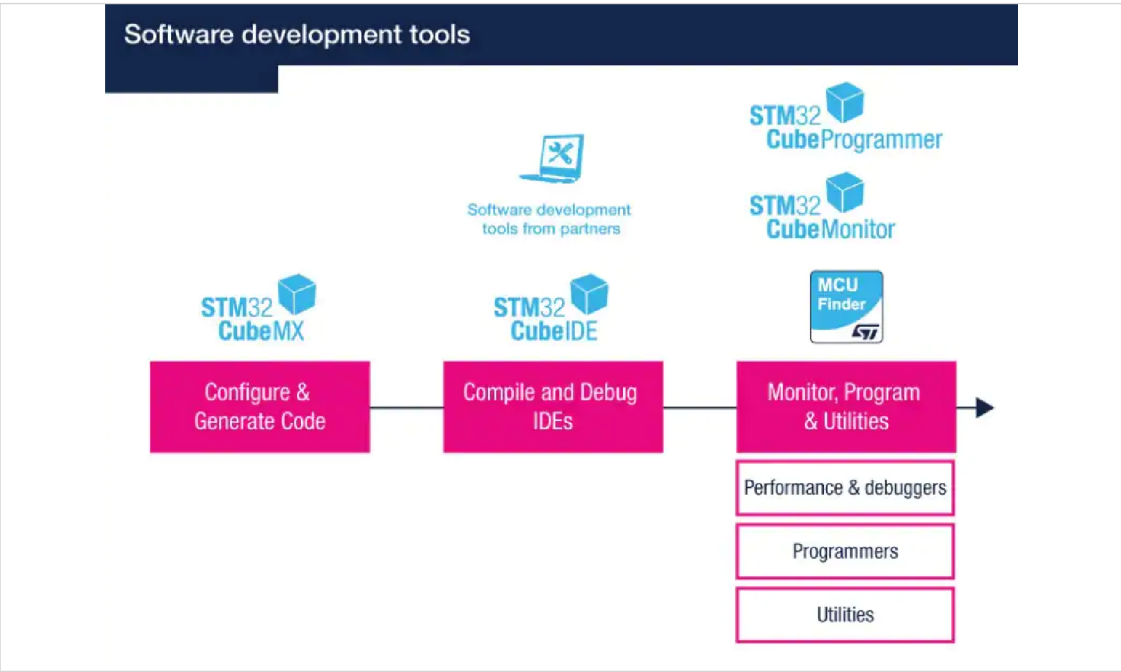
Pro konfiguraci síťového rozhraní na modulu je možné využít příkaz *ifconfig* skrze který lze rozhraní dočasně nastavit. Pokud je potřeba toto rozhraní modifikovat perzistetně, je nutné vytvořit konfiguraci skrze rozhraní **systemd-networkd** vytvořením nového *.network* souboru v adresáři */etc/systemd/network*.

2 Vývojový ekosystém pro STM32MP157

Pro vývoj aplikací pro procesor ST32MP157 je možné využít softwarové nástroje poskytované výrobcem za účelem urychlení vývoje na procesorech jeho výroby. K samotnému vývoji je poté potřeba stáhnout odpovídající části vývojového balíčku, který obsahuje zdrojové kódy, hlavičkové soubory pro Linux a SDK (Software Development Kit) potřebné pro vývoj a cross-kompilaci aplikací určených pro STM32MP157 na PC.

2.1 Softwarové nástroje STMicroelectronics

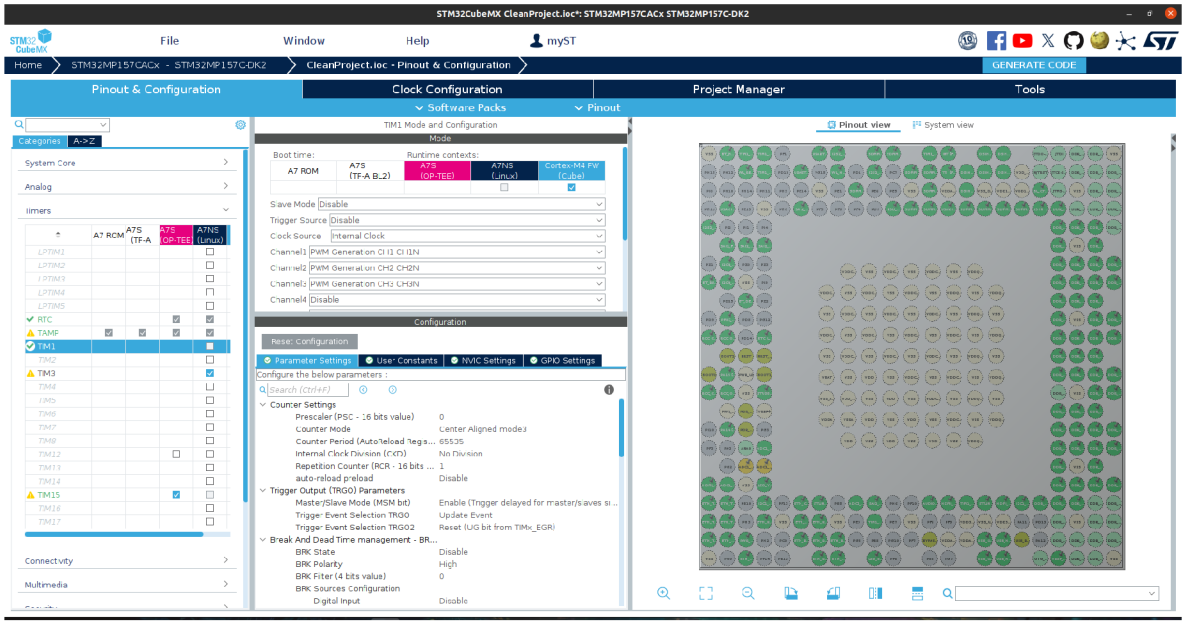
STMicroelectronics nabízí řadu programů pro realizaci a usnadnění vývoje programů pro procesory své výroby.[8] Tyto softwarové nástroje umožňují konfigurovat periferie, generovat kostry projektů pro firmware a poskytují rozhraní pro nahrávání a debugování programů na cílových zařízeních. Tyto nástroje a jejich role během procesu vývoje společně s předpokládaným pořadím jejich použití je možné sledovat na obrázku č. 2.1.



Obr. 2.1: Přehled software nástrojů STM pro vývoj [8]

2.1.1 STM32CubeMX

STM32CubeMX, dále jen CubeMX, je program který umožňuje konfigurovat zařízení skrze jednoduché grafické rozhraní. Uživatel si z výběru periferií uspořádaných do logicky oddělených sekcí zvolí ty které plánuje použít. CubeMX pro jednotlivé periferie otevře podrobnější menu, kde si uživatel může nastavit konkrétní konfiguraci pro danou periferii, jako například použité kanály a režim spouštění převodu u ADC převodníků nebo režimy funkce časovačů. Program zároveň provádí kontroly vzájemné kompatibility zvolených konfigurací pro jednotlivé periferie a nahlašuje případné kolize ve využití pinů. CubeMX taktéž graficky zobrazuje které z pinů zařízení jsou využity pro jaké funkce. Popsané funkce je možné vidět na obrázku č. 2.2.



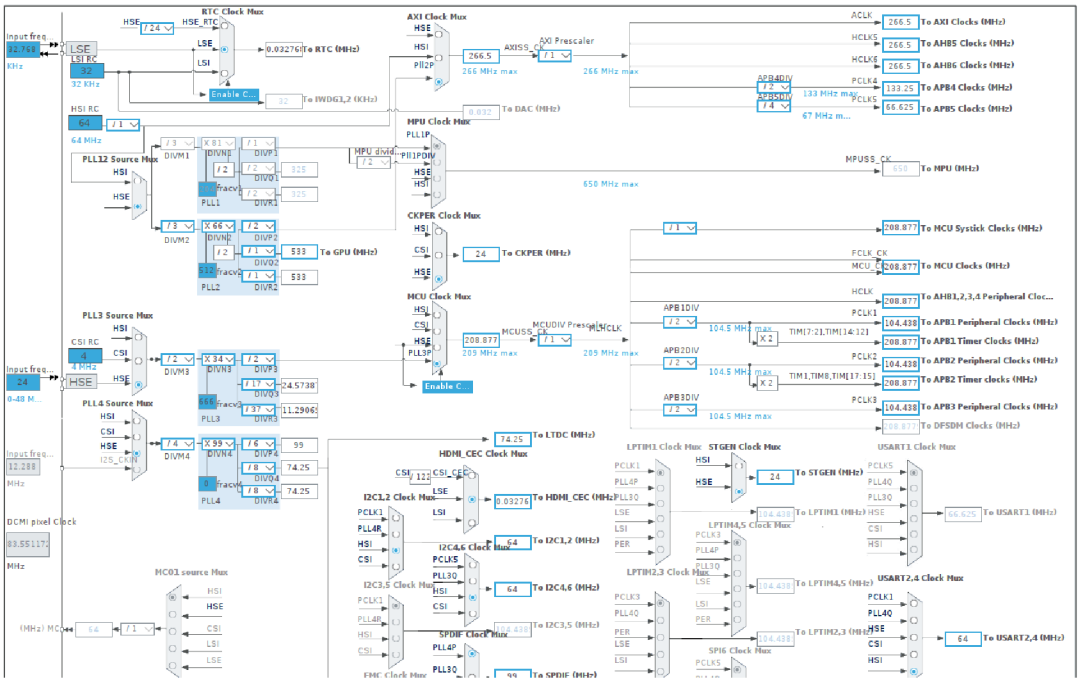
Obr. 2.2: CubeMX - nastavování periferií

Mimo konfiguraci periferií program odděleně zprostředkovává nastavení zdrojů hodinového signálu pro jednotlivé periferie v rámci zařízení. Obvod s hodinovým signálem je zde graficky zobrazen ve formě jednoduchého schématu, příklad nastavení pro STM32MP157 je zobrazen na obrázku č. 2.3.

Konfigurace zařízení může být uložena do souboru formátu **.ioc**, ze kterého je možné vycházet při další konfiguraci zařízení.

Projekt generovaný CubeMX

Poté co si uživatel zvolí konfiguraci periferií a nastavení zdrojů hodinového signálu CubeMX dovoluje vygenerovat kostru projektu ve zvoleném programovacím jazyce



Obr. 2.3: CubeMX - nastavování hodinového signálu

určenou pro vybrané vývojové prostředí. Tato kostra obsahuje relevantní hlavičkové soubory HAL rozhraní pro ovládání potřebných periferií. V kostře projektu jsou zároveň vygenerovány konfigurační funkce které nastaví jednotlivé periferie dle zvolených parametrů v CubeMX. S pomocí tohoto nástroje může být vytvořena i kostra funkce **main()** programu pro zkoumaný modul včetně volání konfiguračních funkcí. Uživatel pak musí již nastavené periferie spustit voláním odpovídající funkce z HAL hlavičkového souboru.

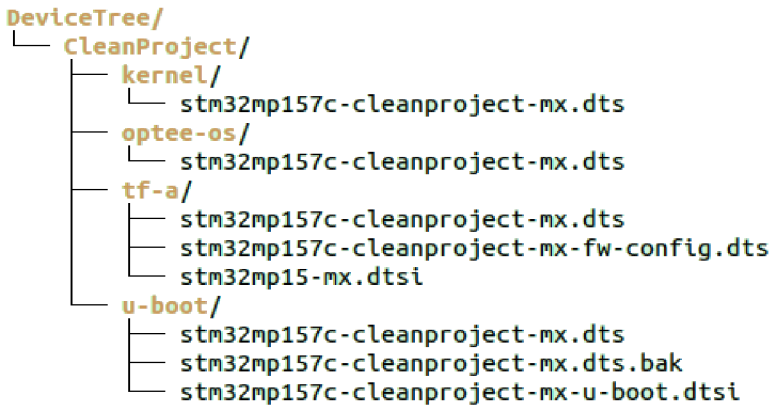
Vygenerovaný projekt je označen pomocí komentářů popisujících jeho funkcionalitu a zároveň je opatřen komentáři označujícími sekce určené pro modifikaci. Je vhodné aby při vývoji firmware uživatelem byla nová funkcionalita implementována uvnitř sekcí k tomu označených dle zápisu zobrazeném ve výpisu č. 2.1.

Výpis 2.1: Styl značení sekcí uživatelského kódu v projektu generovaném pomocí CubeMX

```
/* USER CODE BEGIN <jméno uživatelské sekce> */
  <blok kódu>
/* USER CODE END  <jméno uživatelské sekce> */
```

CubeMX je možné použít na regeneraci již existujícího projektu, například při potřebě změnit konfiguraci periferií nebo přidání doposud nepoužité periferie. V tomto případě budou existující soubory přepsány nově generovanými, kód který se nachází uvnitř sekcí určených pro modifikaci uživatelem však bude zachován.

Společně se zdrojovými soubory pro firmware CubeMX uvnitř projektu pro modul STM32MP157F-DK2 generuje také fragmenty pro úpravu stromu zařízení operačního systému Linux. Příklad struktury vygenerovaných souborů je zobrazen na obrázku č. 2.4. Tyto soubory obsahují specifikaci pro periferie přiřazené koprocesoru. Ty je možné použít pro sestavení upraveného stromu zařízení který zaručí že hlavní procesor nebude zasahovat do periferií příslušejících procesoru Cortex-M4.



Obr. 2.4: CubeMX - generované fragmenty stromu zařízení

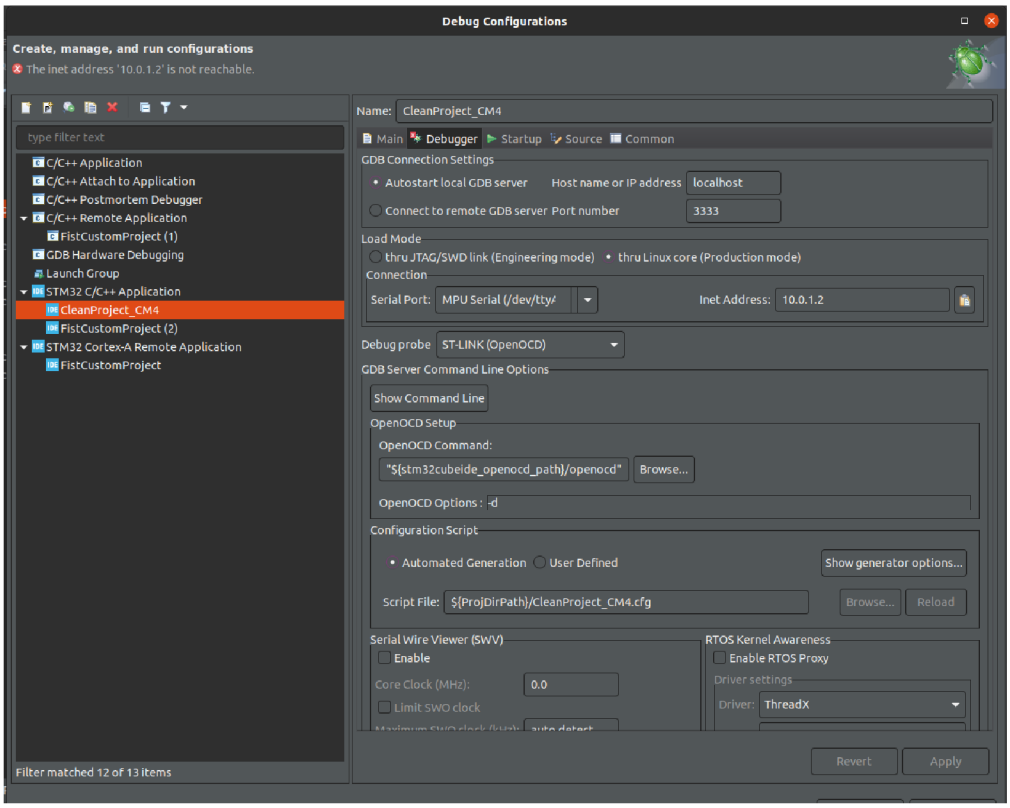
2.1.2 STM32CubeIDE

STM32CubeIDE je vývojové prostředí založené na základech prostředí Eclipse. Toto prostředí poskytuje základní funkce pro psaní programů, jejich sestavování do binárních souborů pro cílová zařízení a ladění. Mimo základní funkce vývojového prostředí tento program nabízí funkcionalitu specifickou pro vývoj na zařízeních od STM, jako napojení na SDK pro OpenSTLinux.

Program nabízí také možnost generování prázdného projektu pro zvolené zařízení, obdobně jako CubeMX. Oproti předvyplněné kostře kterou je možné získat z druhého z těchto programů je projekt pro STM32MP157F-DK2 připravený pouze s pomocí STM32CubeIDE zcela prázdný a ponechává například i vložení HAL hlavičkových souborů na uživateli.

Konfigurace ladění firmware pro Cortex-M4 použitá při vývoji jednoduché aplikace je zobrazena na obrázku č. 2.5. Při debugování a testování ukázkového firmware byla použita statická IP adresa přiřazená modulu STM32MP157F-DK2 na lokální síti a Production režim procesoru vhodný pro zkoumání již funkčního firmware.

STMCubeIDE při ladění dovoluje otevřít konzoli pro Linux terminál připojený na OpenSTLinux běžící na modulu. S pomocí konzole je možné spustit při ladění programu v koprocesoru modulu jako STM32MP157F-DK2 vlastní Linux programy



Obr. 2.5: CubeIDE - použitá debug konfigurace

na hlavním procesoru které mají s firmware interagovat a dále interagovat s operačním systémem. Skrze terminál lze zadávat příkazy pro zápis do RPMsg kanálů pro manuální interakci s koprocesorem.

Engineering vs Production režimy procesoru

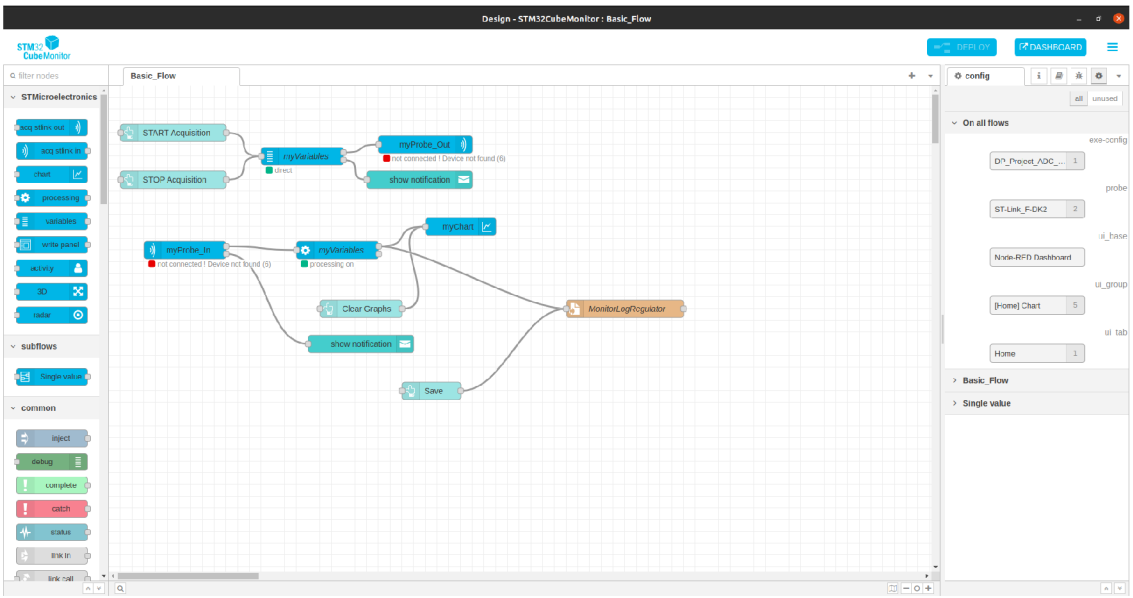
Důležitou poznámkou pro ladění firmware Cortex-M4 je rozdíl mezi **Engineering** a **Production** režimy. V Production režimu je ladění prováděno skrze prostředí OpenOCD (Open On-Chip Debugger) hostované v operačním systému OpenSTLinux na hlavním procesoru. Při tomto režimu je do jádra Cortex-M4 nahrán program ke kterému se následně OpenOCD připojí, v tuto chvíli již však program běží a není jej možné zastavit například na začátku main() funkce. Tento režim je vhodný pro rychlé prototypování jednoduchých aplikací či modifikaci již existujícího a funkčního firmware.

Pro vývoj zcela nové aplikace může být vhodné přepnout procesor do Engineering režimu (modul STM32MP157C-DK2 k tomuto účelu má dva přepínače na spodní straně desky). V tomto stavu se při zapnutí nespustí bootloader z SD karty v Cortex-A7 a je možné do Cortex-M4 nahrát firmware s pomocí ST-Link programátoru

a debuggeru (pro zkoumaný modul je již přichystaný na desce). Takto je možné vytvořit a následně i odladit firmware pro Cortex-M4 bez potřeby zároveň řešit kompatibilitu a interakci s Cortex-A7, zároveň se tím přístupní možnost zastavit program a krokovat jej již od jeho začátku.

2.1.3 STM32CubeMonitor

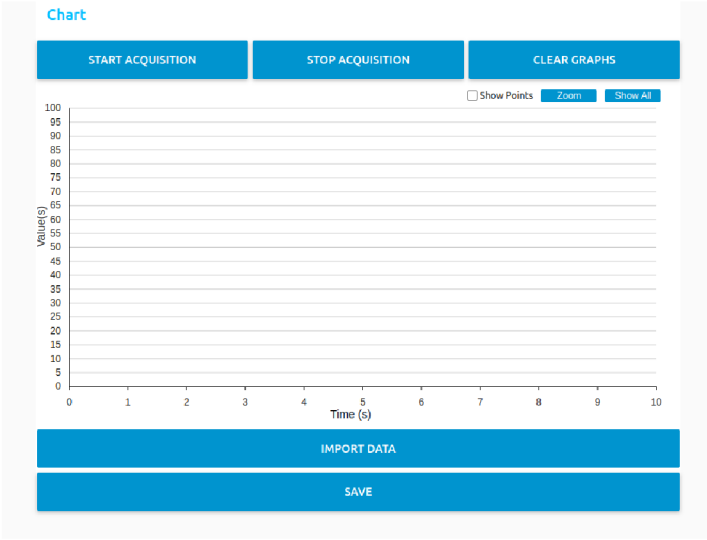
Užitečným nástrojem pro ladění a testování programů může být STM32CubeMonitor. Jedná se o aplikaci ve které lze pomocí grafického programovacího jazyka vytvořit aplikaci pro vyčítání a zobrazování proměnných uvnitř firmware. Pro připojení k procesoru je potřeba ST-Link, který může být provozovaný v režimu přímého nebo také sdíleného spojení. Při sdíleném spojení je možné zároveň připojit například CubeMonitor i CubeIDE. Na obrázku č. 2.6 lze vidět hlavní obrazovku aplikace s grafickým programovacím prostředím.



Obr. 2.6: CubeMonitor - hlavní obrazovka

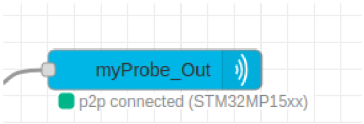
Po kliknutí na tlačítko **Dashboard** se zobrazí graf, který je možné vidět na obrázku 2.7. Po spuštění záznamu tlačítkem **START Acquisition** lze začít zobrazování získaných dat.

Pro používání aplikace CubeMonitor v kombinaci s STM32MP157F-DK2 je nutné dodržovat přesnou sekvenci kroků pro úspěšné připojení a zaznamenání dat. Pomocí CubeIDE je nejprve potřeba nahrát program, spustit jej a odpojit se. ST-Link ve sdíleném režimu pro tento modul bohužel nefunguje a je potřeba se tak opakovaně připojovat a odpojovat od modulu. Druhým krokem je aktualizace bloku



Obr. 2.7: CubeMonitor - Dashboard obrazovka

který vyčítá proměnné dle definic v binárním souboru s programem. V základní konfiguraci programu jde o blok **myVariables**. V případě že tento blok bude neaktuální, hrozí že budou vyčteny data z adres které nebudou obsahovat očekávané proměnné. Po aktualizaci je nutné počkat než CubeMonitor detekuje připojený ST-Link - detekované připojení je možné sledovat na obrázku 2.8. Poté co je modul STM32MP157F-DK2 detekován, je možné spustit záznam dat. Po ukončení záznamu je před dalším připojením CubeMonitor aplikace nutné znovu nahrát program přes CubeIDE. Opětovné pokusy o spojení (i neúspěšné) ze strany této aplikace s pravidelností vedou k potřebě celý proces opakovat od prvního kroku. V některých případech také po odpojení CubeMonitoru není možné skrze ST-Link připojit ani CubeIDE a je potřeba restartovat celý modul a také odpojit USB kabel pro ST-Link před opětovným zapnutím.

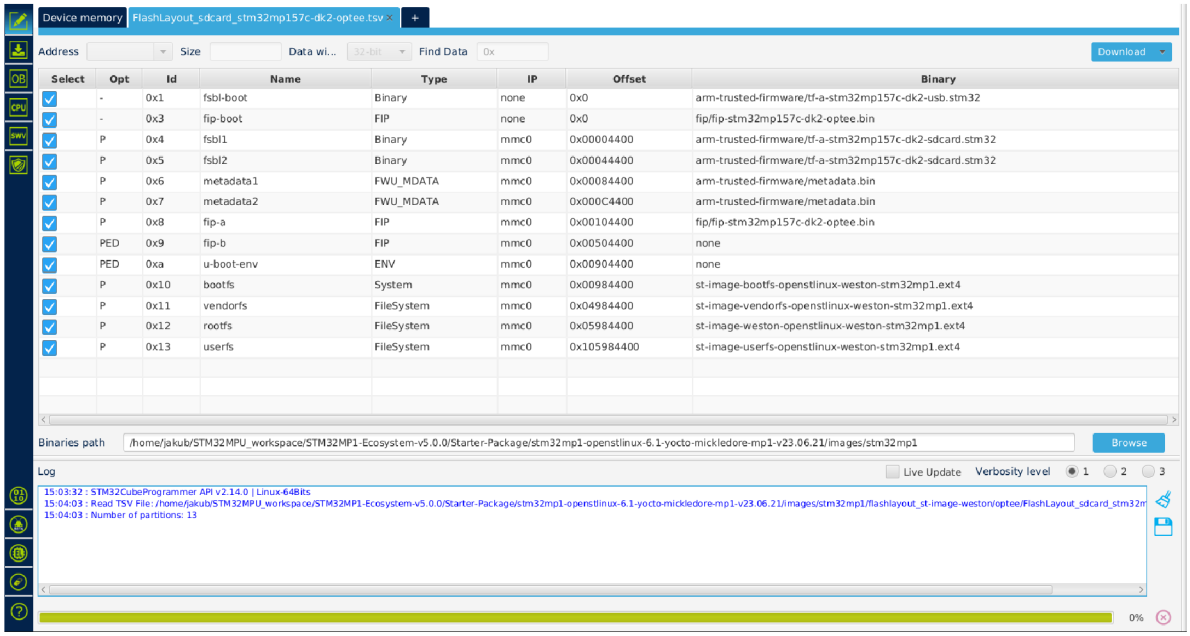


Obr. 2.8: CubeMonitor - Detekce připojeného modulu

2.1.4 STM32CubeProgrammer

STM32CubeProgrammer, dále pouze CubeProgrammer, je nástroj určený pro nahrávání programů do zařízení. V případě práce s modulem STM32MP157F-DK2 je

využitý pouze k nahrávání Linux distribuce na SD kartu ze které probíhá boot sekvence procesoru Cortex-A7. Nahrávání a řízení běhu firmware v procesoru Cortex-M4 je řízeno z hlavního procesoru buď skrze uživatelskou aplikaci nebo s pomocí propojení přes OpenOCD hostovaném na tomto procesoru při ladění v STM32CubeIDE.



Obr. 2.9: CubeProgrammer - volba distribuce k nahrávání

Linux distribuce k nahrání do modulu je popsána souborem s příponou *.tsv*, který specifikuje oddíly pro výslednou distribuci a jaké soubory budou do těchto oddílů nahrány. Po zvolení .tsv souboru v programu CubeProgrammer je v grafickém rozhraní zobrazen jeho obsah. Příklad pro distribuci OpenSTLinux určenou k nahrání do modulu STM32MP157F-DK2 vybaveného SD kartou je možné vidět na obrázku č. 2.9. Skrze toto rozhraní je možné taktéž zvolit pouze některé části distribuce k nahrání v případě potřeby pouze modifikovat již zavedenou instanci OpenSTLinux pomocí souborů s aktualizovanými komponentami.

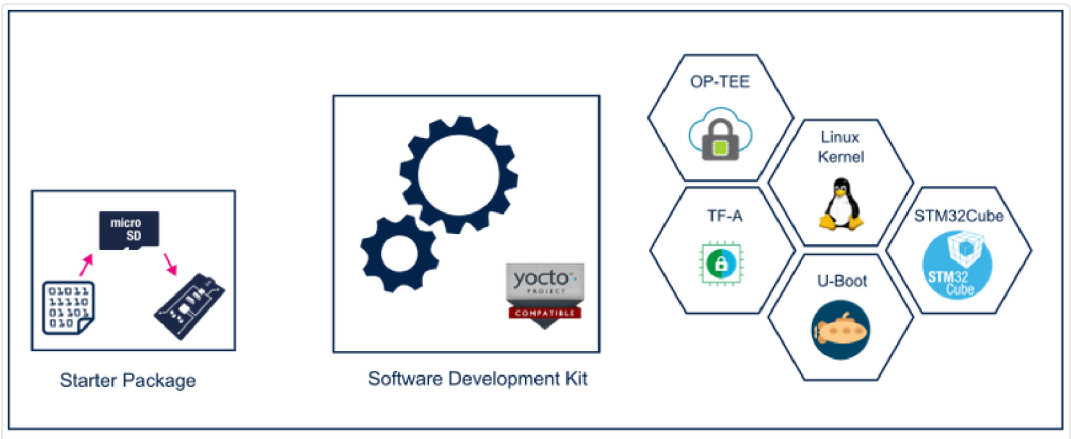
Při nahrávání distribuce je nutné správně nastavit cestu k binárním souborům, .tsv konfigurace totiž specifikuje cesty k souborům relativně vůči této lokaci. V případě změny nahrávané distribuce je potřeba modifikovat tuto cestu i .tsv soubor aby odpovídali nové distribuci, jelikož konfigurace pro distribuce s podobnou strukturou si mohou odpovídat, a v případě že cesta k binárním souborům nebude změněna dojde k nahrání původně zvolené distribuce bez vyvolání chyby při nahrávání do zařízení.

2.2 Vývojové prostředí

Pro modifikaci programů a vývoj nových aplikací pro procesory řady STM32MP1 lze použít vývojářské balíčky poskytované STM. Tyto balíčky obsahují zdrojové kódy pro komponenty Linux distribuce pro procesor, HAL knihovny pro použití ve firmware, kompilátory a ukázkové aplikace. Dle zamýšleného účelu STM poskytuje pro zkoumaný modul dva hlavní balíčky: Developer Package a Distribution Package.

2.2.1 Developer Package

Tento balíček obsahuje všechny komponenty a balíčky potřebné pro vývoj aplikací pro Linux běžící na procesoru Cortex-A7 i pro vývoj firmware spouštěného na jádře Cortex-M4. [9] Samotný balíček se skládá ze tří hlavních komponent zobrazených na obrázku č. 2.10



Obr. 2.10: Developer Package - struktura [9]

Starter Package

Tato komponenta obsahuje pouze základní distribuce OpenSTLinux pro zprovoznění operačního systému na STM32MP157. [9] Distribuce je možné nahrát pomocí programu STM32CubeProgrammer. Balíček obsahuje množství konfigurací pro různé moduly a je důležité zvolit správný z nich. Je vhodné tento balíček zachovat i po prvotním nahrání distribuce do modulu pro případ potřeba obnovy při poškození existující distribuce či výměně SD karty v zařízení.

Software Development Kit

Tento balíček obsahuje komponenty (např. Linux hlavičkové soubory) a skript pro nastavení cross-kompilačního prostředí k vývoji aplikací určených pro běh v operačním systému OpenSTLinux hostovaném na cílovém zařízení. Konfigurační skript je potřeba spustit v terminálu před kompilací programu pro STM32MP157, samotný vývoj je ale možné realizovat v libovolném textovém editoru nebo prostředí se schopností sestavovat makefile programy.

Board Support Package

V tomto balíčku jsou obsaženy zdrojové kódy vybraných částí Linux distribuce které uživatel může modifikovat, následně sám sestavit a nahradit jejich originální verze v distribuci na modulu či s jejich pomocí upravit existující distribuci k nahrání na další moduly. Tento balíček neobsahuje prostředky k sestavení celé distribuce. [9] Komponenty které jsou součástí tohoto balíčku lze sledovat na obrázku č. 2.10 společně se Starter Package, Software Development Kit a STM32Cube komponentami.

Komponenty pro STM32Cube

Součástí Developer Package jsou také nástroje a zdrojové soubory určené pro Cortex-M4. [9] S pomocí tohoto balíčku je možné vyvíjet a modifikovat Firmware pro jádro Cortex-M4. Zde jsou obsaženy HAL knihovny, ukázkové příklady Firmware předvádějící obsluhu jednotlivých periferií modulu a také příklady jednoduchých aplikací pro demonstraci výměny dat mezi Cortex-A7 a Cortex-M4.

Tyto ukázkové aplikace a demonstrační programy lze využít jako výchozí projekt pro vlastní firmware v případě že jsou potřeba pouze malé modifikace existující implementace. Pokud je potřeba změnit konfiguraci projektu, např. předefinováním dostupných periferií pro novou aplikaci, může být snazší alternativou vygenerovat nový projekt s pomocí CubeMX (dle sekce 2.1.1) a požadovanou funkcionalitu z ukázkového programu přenést do nového projektu. K těmto ukázkovým aplikacím není v rámci tohoto balíčku poskytován soubor .ioc který by bylo v CubeMX možné načíst a modifikovat.

2.2.2 Distribution Package

Tento balíček slouží pro sestavování vlastních distribucí vycházejících z OpenSTLinux a obsahuje všechny součásti této distribuce které lze dále modifikovat. [10] Mimo sestavování distribuce je s pomocí tohoto balíčku možné sestavit také další komponenty potřebné pro vývoj s použitím modifikovaných distribucí, jako jsou Board Support Package a Software Development Kit. Strukturu tohoto balíčku je možné

pozorovat na obrázku č. 2.11. Samotné sestavování distribucí a dalších komponent které tento balíček generuje je prováděno pomocí nástroje bitbake.

Obr. 2.11: Distribution Package - struktura [10]

Před zahájením sestavování je potřeba spustit skript pro nastavení vývojového prostředí z balíčku Software Development Kit. Tento skript nastaví systémové proměnné jako je typ zařízení pro které bude distribuce sestavována. Následně je možné zavolat příkaz **bitbake st-image-weston** který sestaví základní distribuci pro modul STM32MP157. Samotný proces sestavování se řídí vrstvami které lze měnit, přidávat a odebírat. V těchto vrstvách jsou definovány změny výsledné distribuce a jsou aplikovány dle priority nastavené pro jednotlivé vrstvy.

Pro sestavování distribucí s pomocí tohoto balíčku je vhodné mít dostatek místa na disku kde bude balíček distribuce sestavovat. Ačkoli výsledná velikost je pro OpenSTLinux se pohybuje v řádu jednotek Gb, při sestavování může být zapotřebí přes 100-150 Gb na uložení dočasných souborů. Během procesu sestavování je kontrolováno zbývající místo na disku a pokud dochází volný prostor, tak bude sestavování přerušeno a ukončeno. V tomto případě je sice možné dočasné soubory vymazat, uvolnit tak místo a navázat na přerušené sestavování, zvyšuje se tak ale šance že při sestavování dojde k chybě nebo bude dočasně potřebný soubor stahován opakovaně. Po dokončení sestavování komponenty balíček sám odstraní zbývající dočasné soubory.

Distribuční balíček je vhodné vždy udržovat aktuální. Verze balíčku (dostupná po dobu experimentování s balíčkem) ze stránek STM se bez aktualizace na novější verzi balíčku odkazuje při sestavování OpenSTLinux distribuce na git repozitář dílčí komponenty, jehož větev byla přejmenována z *master* na *main* bez zachování větve s původním pojmenováním, což vede k selhání sestavování.

Výslednou distribuci lze upravit opětovným sestavením celé distribuce, je však také možné modifikovat pouze vybrané komponenty. Oddíl ve kterém se komponenta nalézá lze připojit pomocí příkazu **mount** a následně přepsat odpovídající soubory uvnitř oddílu pomocí nových souborů.

Do distribuce lze zavést i vlastní uživatelské aplikace které mají být součástí výsledného zařízení.

X-Linux-RT

Společně s výchozí distribucí OpenSTLinux STMicroelectronics nabízí také možnost tuto distribuci upravit rozšířením X-Linux-RT. [11] Tato distribuce s sebou přináší zlepšení real-time vlastností skrze minimalizaci jitteru přítomného v systému. Rozšíření však do systému zavede omezení: neumožňuje změnu taktovací frekvence pro procesor na kterém bude distribuce provozována což znemožňuje provoz v úsporném režimu s dynamickou taktovací frekvencí. Pro sestavení této distribuce je potřeba do distribučního balíčku zavést rozšíření vložením nové vrstvy pro bitbake, alternativně lze distribuci také modifikovat skrze úpravu Linux kernelu a OP-TEE které jsou přítomné v Board Support Package.

2.3 Dokumentace STMicroelectronics

STMicroelectronics poskytuje dokumentaci v řadě forem, zde jsou zmíněny některé z nich které se při vývoji aplikací a obecně při práci s modulem ukázaly jako užitečné.

Datasheety

STM v datasheetech pro svá zařízení uvádí především informace o vlastnostech a schopnostech výrobku. Jsou zde uvedeny dostupné periferie včetně jejich funkčního popisu a elektrické vlastnosti zařízení. Dle datasheetu tak lze především vybrat z katalogu zařízení konkrétní model vhodný pro plánovanou implementaci.

Referenční manuály

V referenčních manuálech je zařízení popsáno z hlediska implementace funkcionality. Jsou popsány principy operace periferií, registry které periferiím náleží a jsou naznačeny předpokládané sledy signálů v časové doméně. Tyto manuály obsahují užitečné informace pro vývoj firmware a programů interagujících s periferiemi na nízké úrovni.

Referenční manuál také může být vhodnou pomůckou při nastavování zařízení skrze CubeMX kde poskytne informace o režimech provozu periferií a jejich možnostech. Dále může sloužit při vývoji Firmware který obchází použití HAL ovladačů.

wiki stránky

STM hostuje vlastní webové stránky s články pro svůj ekosystém MPU zařízení. Obsah těchto článků se částečně překrývá s informacemi poskytovanými v referenčním manuálu a dalších dokumentech, ale jsou zde uvedeny i informace o softwarových nástrojích a vývojových prostředích. Mnohé z článků mají charakter tutoriálu provázejícího uživatele procesem nastavení software komponent spravovaných STM nebo nabízí postupy pro práci s moduly či procesory. Webové stránky také obsahují články detailně popisující jednotlivé komponenty software a hardware pro STM zařízení.

Zde jsou také hostovány Quick Start návody určené pro rychlé zprovoznění a otestování modulů jako Discovery Kit, kterým se tato práce zabývá. Pro velkou část vývoje aplikace pro Linux běžící na STM modulech je možné vycházet z informací uvedených na těchto stránkách.

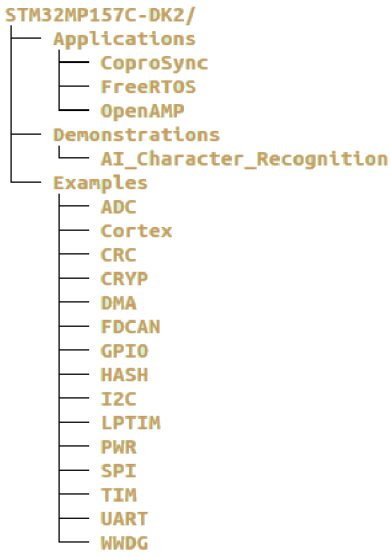
dokumentace uvnitř balíčků a komponent

Uvnitř jednotlivých vývojářských balíčků a jejich dílčích komponent STM umisťuje *readme* soubory popisující práci s balíčky. Na některé z těchto souborů se odkazují i návody na wiki stránkách.

Příkladem mohou sloužit readme soubory uvnitř balíčků se zdrojovými soubory pro komponenty Linux distribuce dostupnými jako součást Board Support Package. Tyto soubory obsahují sekvenci příkazů potřebných pro sestavení/modifikaci těchto komponent které lze jednoduše implementovat jako skript pro sestavení.

3 Ukázkové projekty pro STM32MP157

STM jako součást balíčku s komponentami pro STM32Cube poskytuje ukázkové projekty pro zkoumaný modul. Všechny tyto ukázkové projekty jsou součástí základní distribuce OpenSTLinux a jsou tak na testovacím modulu přítomné a spustitelné již po prvním nahrání distribuce. Cesta k projektům v souborovém systému OpenSTLinux je *#/usr/local/Cube-M4-examples/STM32MP157C-DK2/*. Zde se nachází jednotlivé projekty roztřízené do složek. Tuto souborovou strukturu je možné vidět na obrázku č. 3.1. Adresář s projekty uvnitř balíčku pro STMCubeIDE má identickou strukturu jako adresář s projekty nacházející se uvnitř distribuce na samotném zařízení.



Obr. 3.1: Struktura adresáře s ukázkovými projekty

Ve složce Examples se nacházejí ukázkové aplikace demonstrují práci s jednotlivými periferiemi zařízení a nabízí tak výchozí bod pro vývoj vlastního firmware který tyto periferie bude používat.

Kromě základní funkcionality periferií jsou zde také příklady komplexnějších programů nastiňujících možnosti spolupráce procesorů Cortex-A7 a Cortex-M4. V adresáři *Applications/OpenAMP* je připraveno několik způsobů obsluhy RPMsg kanálů ze strany Cortex-M4. Ve složce *Applications/FreeRTOS* je uveden příklad vytváření vláken pro uživatele kteří budou chtít na koprocesoru využívat tento operační systém. Ve složce *Applications/CoproSync* je jednoduchý příklad použití IPCC k předávání příkazu pro vypnutí procesoru Cortex-M4.

V souboru *Demonstrations* je umístěna ukázková aplikace pro rozpoznávání znaků. Tato aplikace představuje příklad konečného použití zařízení a využívá spolupráce

obou procesorů. Cortex-A7 řídí uživatelské rozhraní kde uživatel může pomocí dotykové obrazovky kreslit znaky. Informace o nakresleném znaku jsou předány neuronové síti hostované na koprocesoru za účelem snížení zátěže hlavního procesoru, do kterého je navrácena pouze informace o klasifikaci nakresleného znaku.

Každý z těchto projektů má v STMCube balíčku odpovídající projekt pro CubeIDE obsahující zdrojové soubory pro program v Cortex-M4. Tyto projekty lze využít pro debugování s pomocí OpenOCD. Konfigurace pro připojení je popsána v sekci 2.1.2.

Mimo projekty dostupné přímo v distribuci a jako součást předpřipravených projektů pro STMCubeIDE je poksytovaný také ukázkový projekt implementující funkcionalitu logického analyzátoru.[12] Tento projekt je zaměřený na demonstrování způsobu předávání vyšších objemů dat mezi procesory, při kterém by použití existujícího systému RPMsg bylo nedostačující. Jedná se také o další příklad komplexní aplikace rozdělující zátěž mezi hlavní procesor a koprocesor. Hlavní procesor obsluhuje grafické rozhraní na displeji pro uživatele který může tlačítky řídit běh analyzátoru a také zobrazuje naměřená data. Cortex-M4 obstarává vzorkování dat na měřených rozhraních a odesílá naměřená data do hlavního procesoru od kterého také přijímá příkazy řídící běh programu.

3.1 Ukázkový projekt OpenAMP_TTY_echo

Tento projekt implementuje jednoduchý loopback na dvou RPMsg kanálech. Procesor Cortex-M4 nastaví tyto dva kanály a následně ve smyčce kontroluje zda skrze nějaký z nich přišla zpráva z hlavního procesoru, kterou pak odešle na stejném kanále zpět.

V Linux systému běžícím na procesoru Cortex-A7 se tyto dva kanály objevují jako soubory *#/dev/ttyRPMSGx*, kde x označuje číslo kanálu. Do těchto souborů lze zapisovat a číst z nich. Při zápisu jsou data odeslána koprocesoru a při čtení jsou vyčítána data odeslaná koprocesorem. Obsah souboru je po čtení vyprázdněn.

Projekt byl zvolen pro demonstrování funkčnosti modulu a pro validaci nahrané distribuce. S pomocí STMCubeIDE byl tento projekt sestaven a následně nahrán na modul, takto byla otestována konfigurace pro ladění kódu v STMCubeIDE. Samotný kód byl upraven aby oproti původní implementaci, která pouze vracela přijatou zprávu, společně s původní zprávou odesílal i informaci o kanále skrze který procesor zprávu obdržel.

Na obrázku č. 3.2 je zobrazen proces testování funkčnosti aplikace skrze terminál otevřený prostřednictvím STMCubeIDE dle instrukcí v návodu uvnitř ukázkové aplikace. První příkaz nastavuje RPMsg kanál, druhý z příkazů spustí vypisování

obsahu souboru, do kterého budou předávány zprávy odeslané z koprocesoru. Následující příkaz odešle zprávu *"Hello Virtual UART0"* přes RPMsg kanál 0. Poslední zobrazený řádek pak obsahuje obdrženou zprávu z koprocesoru obsahující také informaci o kanále přes který zpráva dorazila.

```
root@stm32mp15-rt:/# stty -onlcr -echo -F /dev/ttyRPMSG0
root@stm32mp15-rt:/#
root@stm32mp15-rt:/#
root@stm32mp15-rt:/# cat /dev/ttyRPMSG0 &
root@stm32mp15-rt:/#
root@stm32mp15-rt:/# echo "Hello Virtual UART0" >/dev/ttyRPMSG0
root@stm32mp15-rt:/# Channel RPMSG0: Hello Virtual UART0
```

Obr. 3.2: Terminál STM32MP157C-DK2 při testování ukázkového projektu

Jak je uvedeno v předchozí sekci, tento projekt je již přítomný v samotné distribuci nahrané na modulu a je možné jej spustit a otestovat také výhradně lokálně bez propojení k STMCubeIDE. Při debugování s pomocí STMCubeIDE však dojde k nahrání sestaveného projektu znovu do *#/usr/local/projects/<jméno projektu>* a do koprocesoru je nahrán nově sestavený program.

4 Měření rychlosti předávání dat mezi procesory

Pro testování real-time vlastností modulu byla vytvořena jednoduchá aplikace pro hlavní procesor. Výsledná aplikace ve formě zdrojových kódů, samotného binárního souboru a makefile souboru pro sestavení je součástí elektronické přílohy. K sestavení aplikace je potřeba nejprve spustit konfigurační skript pro Software Development Kit k modulu STM32MP157F-DK2.

Tato aplikace funguje v kombinaci s ukázkovým firmware pro Cortex-M4 popsaným v kapitole č. 3 a jejím cílem je měřit limit přesnosti časování spouštění procesů na hlavním procesoru společně s měřením rychlostí předávání zpráv mezi koprocesorem a procesorem Cortex-A7.

Samotný program obsahuje jednoduchou definici vlákna pomocí knihovny pthread jejíž použití je dokumentováno na stránkách Linux Foundation. [13] [14] Uvnitř vlákna které vznikne se v nekonečném cyklu zapisuje a čte ze souboru */dev/ttyRPMSG0*, přičemž se ukládají časové známky odesílání zpráv a příjmu odezvy odeslané z procesoru Cortex-M4. Rozdíl těchto dvou časových známek produkuje interval za který proběhlo předání zprávy do koprocesoru a zpět. Společně s těmito časovými známkami je také ukládán čas začátku nové iterace smyčky vlákna s jehož pomocí je určován interval se kterým je vlákno spouštěno operačním systémem.

Program do výstupní konzole zapisuje informace o intervalu jeho spouštění a také časový údaj o rychlosti obdržení odezvy na odeslání zprávy koprocesoru. Příklad výstupu aplikace v terminálu je zobrazen na obrázku č. 4.1.

```
RTT [us]: 134
Iteration time: 45
Iteration time: 60
Iteration time: 41
Iteration time: 40
RTT [us]: 135
Iteration time: 45
Iteration time: 61
Iteration time: 41
RTT [us]: 142
Iteration time: 96
Iteration time: 62
Iteration time: 42
Iteration time: 41
RTT [us]: 136
```

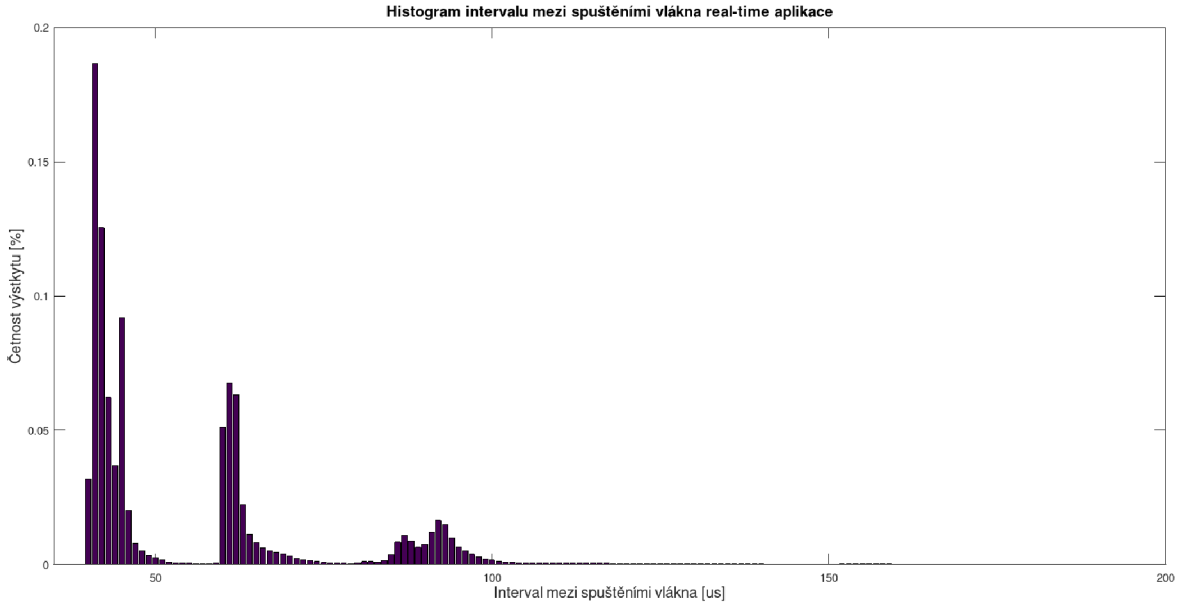
Obr. 4.1: Ukázka zaznamenávání intervalů řídicí aplikací

V aplikaci je použita funkce **clock_nanosleep()**, která má vlákno pozastavit na dobu předanou jako argument, v programu je jí však předávána hodnota kterou není systém schopen dosáhnout. Tímto způsobem je zajištěno že procesu bude

věnován veškerý výpočetní výkon který není dedikován ostatním procesům a bude zjištěna limitní frekvence systému se kterou je schopen spouštět cyklické procesy. Za účelem zvýšení této frekvence byla použita modifikovaná distribuce X-Linux-RT popsaná v sekci 2.2.2.

4.1 Naměřené hodnoty frekvence spouštění procesu

Graf s naměřenými intervaly mezi jednotlivými spuštěními smyčky vlákna je vyobrazen na obrázku č. 4.2. Zde je možné vidět tři samostatné shluky časových hodnot. Kratší dva shluky intervalů přináleží standardnímu běhu aplikace. Nejkratší z těchto shluků představuje cykly ve kterých smyčka programu pouze kontroluje zda byla obdržena zpráva z koprocesoru. Prostřední shluk intervalů náleží cyklům aplikace ve kterých se odesílá nová zpráva pro koprocesor, čímž je hlavní procesor více zatížen. Poslední shluk je zapříčiněn dlouho trvajícími přerušeními mezi ukládáním časových značek při běhu programu. Tato přerušení nemusí mít definovanou dobu trvání, což se podepisuje na tvaru tohoto shluku časových údajů.



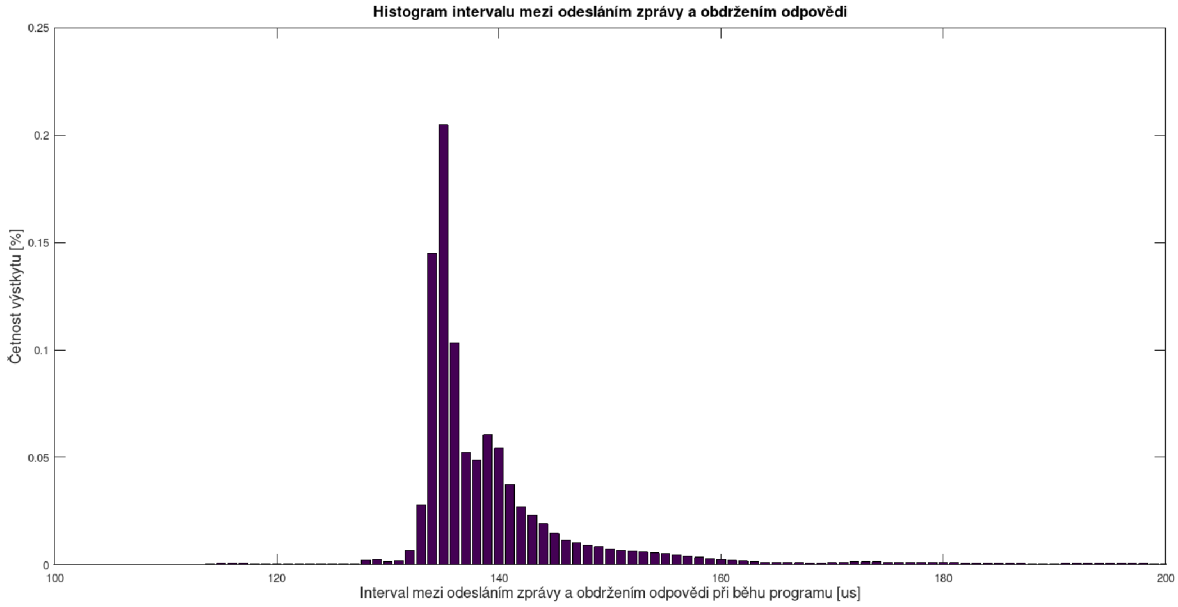
Obr. 4.2: Graf naměřených intervalů mezi spuštěními vlákna

4.2 Měření rychlosti předávání zpráv mezi procesory

Druhým výsledkem měření je graf intervalů mezi odesláním zprávy z hlavního procesoru a přijetím odezvy generované koprocesorem. Tento graf je možní vidět na

obrázku č. 4.3. Z tohoto grafu je možné odhadovat latence komunikace mezi procesory v zařízení.

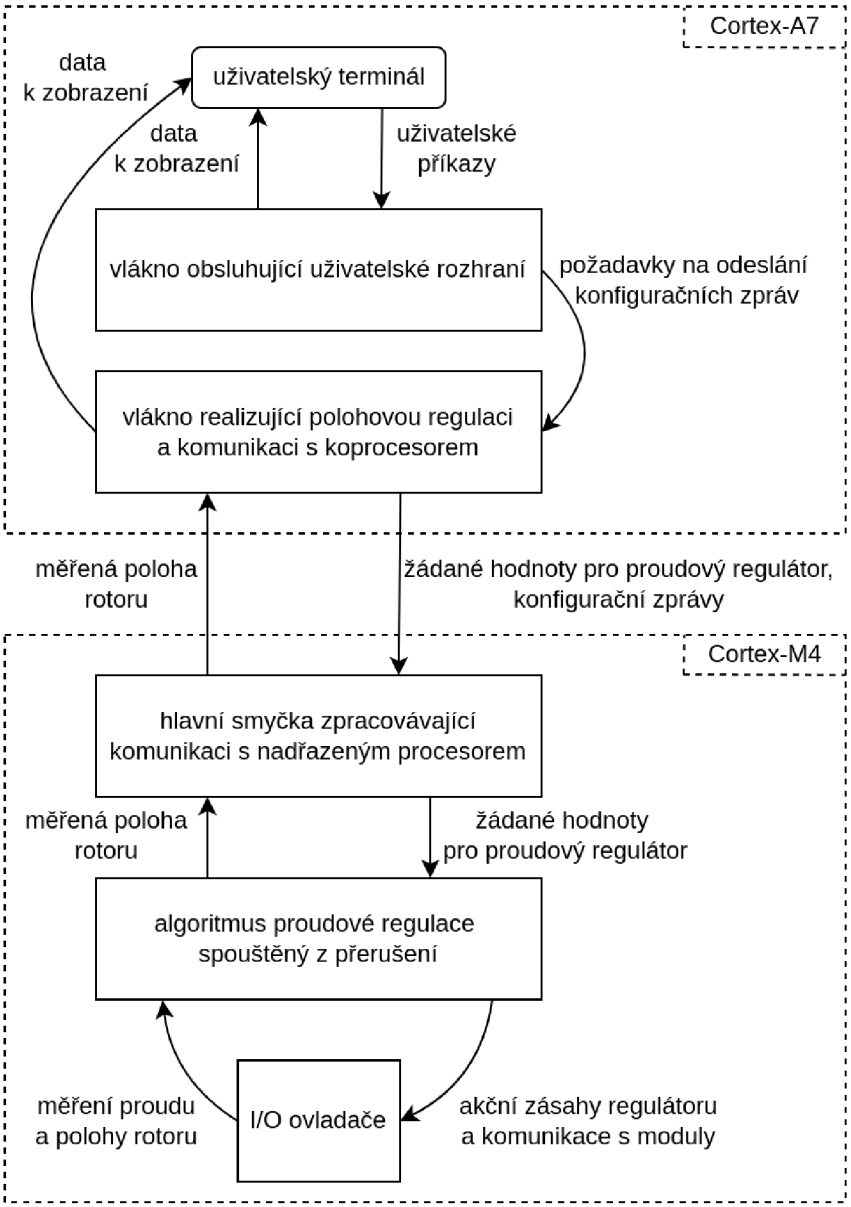
Tento časový údaj je v měření zatížen chybou způsobenou zpožděními při příjmu a odesílání zpráv vytvořených vlivem operačním systémem běžícím na hlavním procesoru. Není jisté zda je odesílání a přijímání zpráv operačním systémem stejně časově náročné a nelze tak prohlásit latenci jednosměrného přenosu za přesně poloviční vůči těmto naměřeným hodnotám, měření ale poskytuje orientační představu o prodlevách v komunikai. Do naměřené hodnoty také zasahuje prodleva se kterou zprávu zpracuje Cortex-M4 a odešle ji zpět. Koprocesor při tomto měření pouze cyklicky kontroluje kanály a odesílá přijaté zprávy zpět, tato prodleva by tedy vůči prodlevám způsobeným operačním systémem na hlavním procesoru měla být zanedbatelná.



Obr. 4.3: Graf naměřených intervalů mezi vysláním a přijetím zpráv

5 Návrh časově kritické aplikace pro zařízení

Pro demonstraci a ověření vlastností procesoru STM32MP157 při nasazení do časově kritických procesů byla zvolena aplikace vektorového řízení motoru v kaskádní smyčce. Tato aplikace vyžaduje přesné časování regulačních smyček a je tak na ní možné zkoumat vlastnosti tohoto procesoru.



Obr. 5.1: Schéma návrhu výsledné aplikace

Řídicí aplikace byla při návrhu rozdělena do dvou částí: proudový regulátor a správa hardware rozhraní bude obsluhována programem v koprocesoru Cortex-M4,

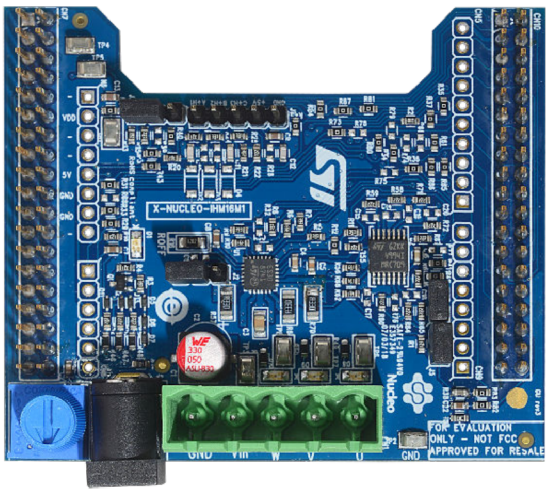
zatímco nadřazený poziční regulátor společně s obsluhou rozhraní pro uživatele budou řešeny z aplikace v hlavním procesoru Cortex-A7. Toto rozdělení umožní přesně časovat 16kHz smyčku proudového regulátoru v koprocesoru zatímco na Cortex-A7 budou kladeny menší nároky na přesné spouštění polohové regulační smyčky. Základní koncept a rozvržení aplikace je znázorněno na obrázku č. 5.1, kde jsou také zaznačeny datové toky mezi jednotlivými bloky aplikace.

5.1 Hardware prostředky

Jako hardware prvky pro tuto aplikaci byly zvoleny modul **IHM16M1** společně s BLDC motorem **GBM2804H-100T**. Tato zařízení jsou společně s vývojovou deskou **NUCLEO-G431RB** součástí balíčku pro testování řízení motorů **P-NUCLEO-IHM03** nabízeného STM [15]. Pro měření úhlu natočení rotoru byl zvolen modul **AS5147P-TS_EK_AB** poskytující PWM rozhraní indikující absolutní natočení rotoru.

5.1.1 IHM16M1

IHM16M1 je modul pro řízení motorů v šestikrokovém režimu a pro vektorové řízení motorů. Modul je zobrazen na obrázku č. 5.2.



Obr. 5.2: Modul IHM16M1 [16]

Toto zařízení je určeno pro provoz v kombinaci s vývojovou deskou **NUCLEO-G431RB**, která obsahuje procesor **Nucleo STM32G431RB**. Jádrem IHM16M1 je řídicí čip pro motory **STSPIN830** který je schopný ovládat třífázové motory s operačním napětím 7 - 45 V. Modul STSPIN830 nabízí dva režimy operace,

jeden pro řízení motoru třemi PWM signály řídícími proud cívkami motoru, a druhý za použití 6 PWM signálů, z nichž 3 jsou komplementární vůči zbylým třem.

Modul IHM16M1 také umožňuje volbu z řady konfigurací pro různé typy řízení motoru. Tyto režimy lze nastavit úpravou propojek na modulu dle tabulky na obrázku č. 5.3. [16].

6-step(1)				FOC (3-shunt)(2)				FOC (single shunt)			
Single shunt	Current sensing	Current limiter enabled (3)	Adjustable current limiter threshold (3)	Three-shunt	Current sensing	Current limiter disabled	Fixed current limiter threshold	Single shunt	Current sensing	Current limiter enabled (3)	Adjustable current limiter threshold (3)
Close JP4 and JP7, solder bridge at the bottom	Open J5 and J6 jumpers	J2 closed to 1-2 position	J3 closed to 2-3 position	Open JP4 and JP7, solder bridge at the bottom	Close J5 and J6 jumpers	J2 closed to 2-3 position	J3 closed to 1-2 position	Close JP4 and JP7, solder bridge at the bottom	Close J5 and J6 jumpers	J2 closed to 1-2 position	J3 closed to 2-3 position

1. *Voltage or current mode.*
2. *Default configuration*
3. *Optional.*

Obr. 5.3: Režimy provozu IHM16M1 dle použitého algoritmu ovládání motoru [16]

Motor ve výsledné aplikaci bude řízen za pomoci vektorového řízení (dle tabulky FOC 3-shunt). Modul je tak ponechán ve své tovární konfiguraci ve které byl dodán. V dokumentaci je také uvedena citlivost a klidová hodnota signálů proudových zpětných vazeb dle tabulky na obrázku č. 5.4.

Opamp	Sensed current	Gain	Out offset	J5	J6	Remarks
1	None (grounded)	1	0 V			Unused
2	Phase V(1)	1.53	1.56 V	Closed	Closed	FOC
		3	0 V	Open	Open	6STEP
3	Phase W(1)	1.53	1.56 V			FOC
4	Phase U(1)	1.53	1.56 V			FOC

1. *In single shunt topology, all the operational amplifiers sense the same current.*

Obr. 5.4: Základní nastavení proudové zpětné vazby modulu IHM16M1 [16]

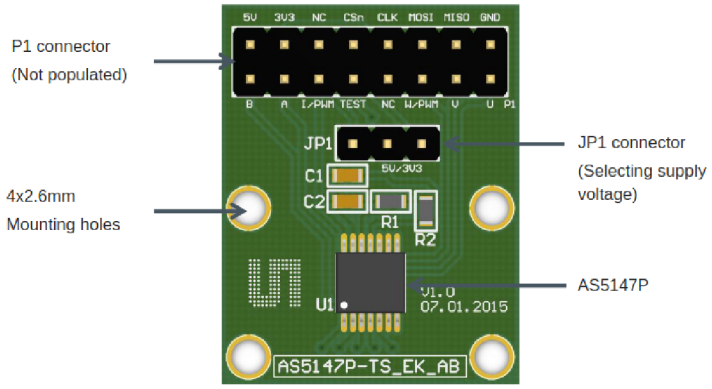
STM nabízí knihovny a demonstrační aplikace pro použití společně s vývojovou deskou Nucleo přiloženou v balíčku k řízení motorů. V kombinaci s vývojovou deskou STM32MP157F-DK2 však bylo potřeba vycházet ze schémat v dokumentaci pro modul řízení motorů i pro samotný čip STSPIN830.

Důležitou poznámkou k modulu je také fakt, že ačkoli je modul vybaven prokovy sloužícími pro osazení standardního konektoru Arduino desek a modulů viditelných na obrázku č. 5.2, některé signály na tomto konektoru nejsou kompatibilní se signály

přítomnými na Arduino konektoru STM32MP157F-DK2 pro zvolenou konfiguraci a řídicí algoritmus. V FOC 3-shunt režimu je jeden ze tří analogových signálů zpětné proudové vazby veden mimo sekci Arduino konektoru typicky určenou pro analogové piny modulů. Část potřebných enable signálů povolujících připojení napájení na jednotlivé větve vinutí motoru také není na tyto konektory vyvedena. Modul tak nelze jednoduše připojit k STM32MP157 testovacímu modulu a je nutné propojit jednotlivé signály.

5.1.2 AS5147P-TS_EK_AB

Tento modul je založený na čipu **AS5147P** určeném pro měření natočení rotorů. Obrázek č. 5.5 zobrazuje tento modul.



Obr. 5.5: Modul pro měření polohy AS5147P-TS_EK_AB [17]

Poloha je měřena pomocí sledování natočení magnetického pole vyvolávaného permanentním magnetem magnetovaným kolmo k ose otáčení. Magnetické pole je měřeno Hallovými sondami uvnitř čipu a je citlivé na vzdálenost magnetu od čipu a jeho síle. Společně s modulem byl dodán neodymový magnet vhodný k použití v kombinaci s modulem. Dle dokumentace je vhodné magnet spojit s rotorem a umístit jej 0,5 - 3 mm nad čip osazený na modulu [17].

Modul nabízí několik způsobů měření polohy: enkodérové rozhraní ABI, rozhraní UVW pro řízení motoru v šestikrokovém režimu, vyčítání polohy skrze SPI sběrnici a také PWM signál modulovaný dle absolutní úhlové polohy. [18]

Modul je možné provozovat buď v 5V nebo 3.3V režimu volbou napájecího napětí. 5V režim je pro tento modul výchozí variantou a pro přepnutí do módu 3.3V je potřeba dle dokumentace upravit pozici propojky JP1 a pře-osadit propojovací odpor z polohy R1 do pozice R2 [18].

Čip AS5147P lze konfigurovat pomocí zápisu do registrů skrze SPI rozhraní. Zde jsou nabízeny dva režimy konfigurace: zápis do volatilní paměti, při kterém se nastavení resetuje po odpojení zařízení od napájení, a jednorázové programování které nastaví výchozí hodnoty ze kterých se zařízení bude při zapnutí obnovovat. Jednorázové programování není možné pro ukončení modifikovat a spouští se zápisem do speciálního registru. Po ukončení programování je nutné dalším zápisem do tohoto registru naprogramované hodnoty potvrdit.

Zpracování PWM signálu je dále popsáno v sekci 6.2.3.

5.1.3 GBM2804H-100T

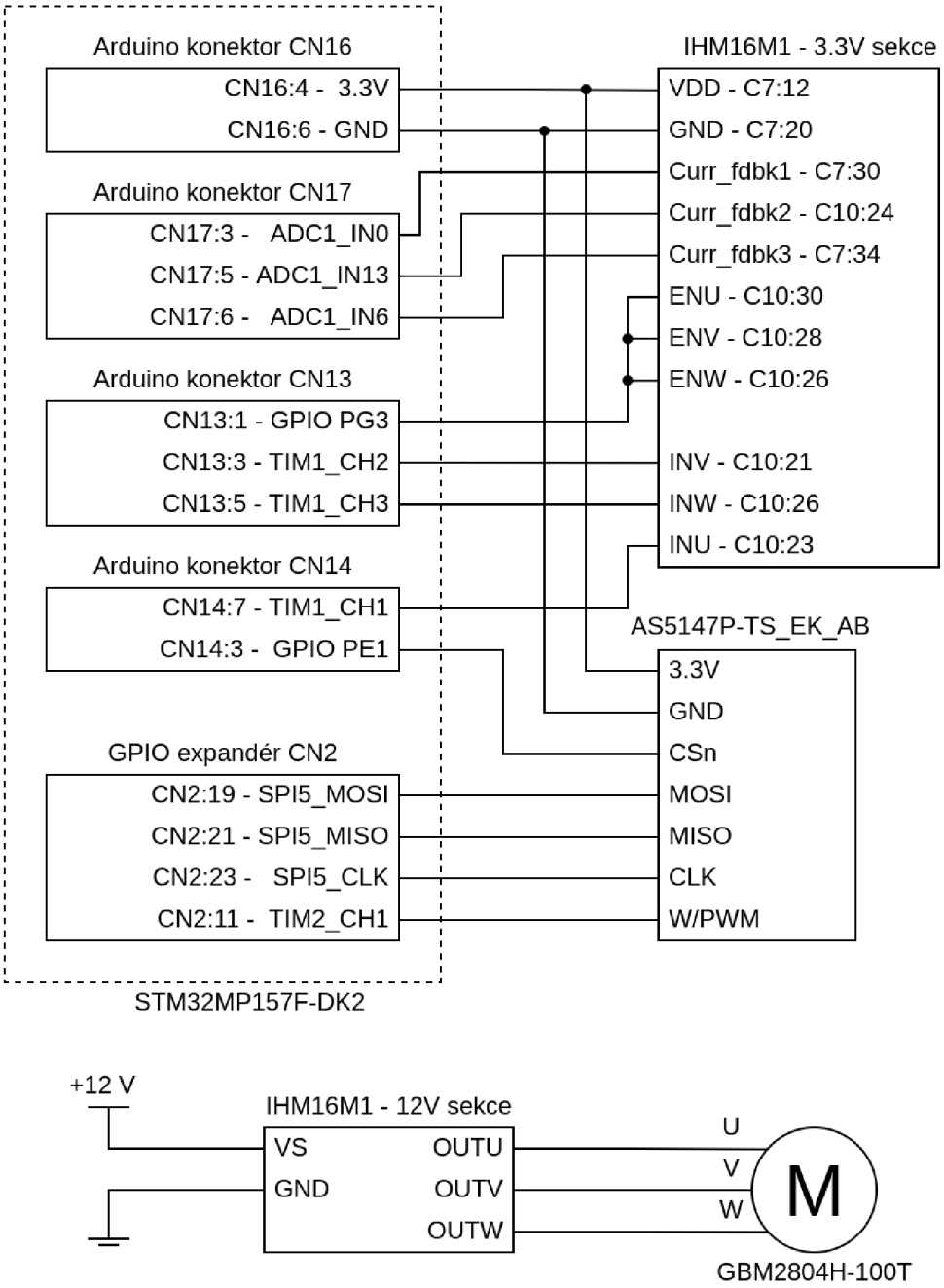
GBM2804H-100T je BLDC třífázový motor určený například pro manipulaci s kamerami během dronových letů. Motor je zobrazen na obrázku č. 5.6. Tento motor má 7 pólových párů permanentního magnetu [15]. Plášť motoru se sestává ze dvou krytů, z nichž je jeden spojen se statorem a druhý s rotorem. Oba pláště jsou vybaveny závity pro připevnění motoru. Vnitřní část klece ložiska umístěného v ose motoru je spojena s rotorem. kabely jsou vyvedeny ze statorové části pláště.

Obr. 5.6: Motor GBM2804H-100T [15]

5.1.4 Testovací přípravek

Elektrické schéma výsledného zapojení je zakresleno na obrázku č. 5.7.

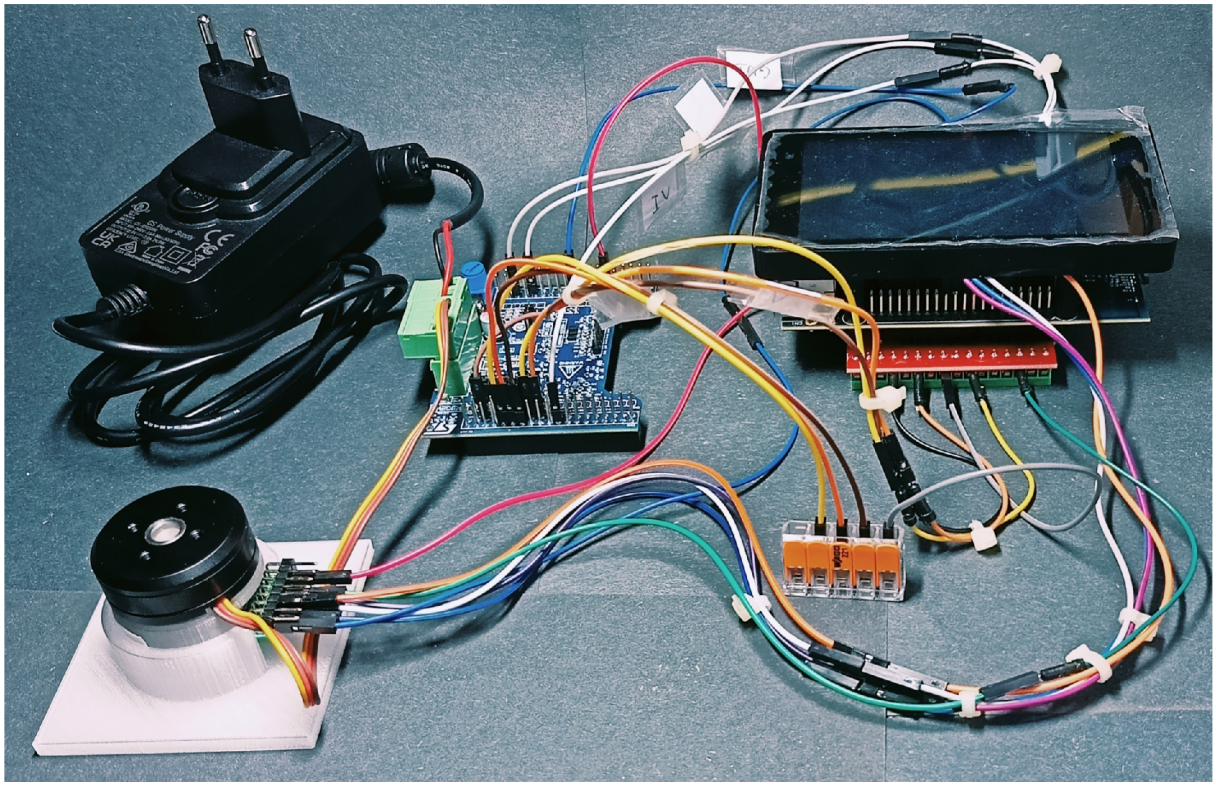
Pro fyzickou realizaci přípravku byly k propojení jednotlivých modulů využity běžné propojovací kabely. Pro vytvoření uzlů ve spojeních kde tomu bylo potřeba, například povolovací signály pro modul IHM16M1, jsou využity WAGO svorky.



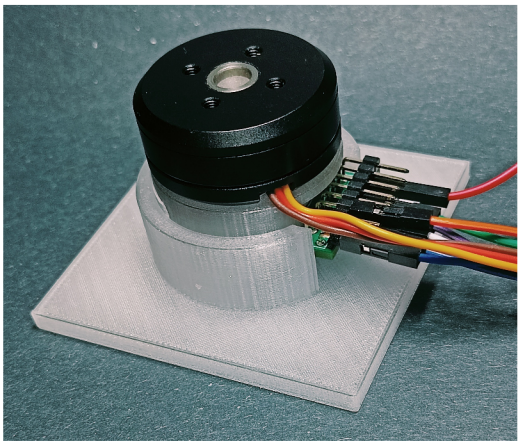
Obr. 5.7: Elektrické schéma testovacího přípravku

Na obrázku č. 5.8 je fotografie výsledného testovacího přípravku. Pro zjednodušení přístupu k signálům na Arduino konektorech na spodní straně vývojové desky byl zakomponován modul který tyto konektory propojí na svorkovnice.

Pro mechanické spojení modulu AS5147P-TS_EK_AB s motorem a připevnění magnetu k rotoru tohoto motoru byla použita armatura vytvořená pomocí 3D tisku viditelná na obrázku č. 5.9.



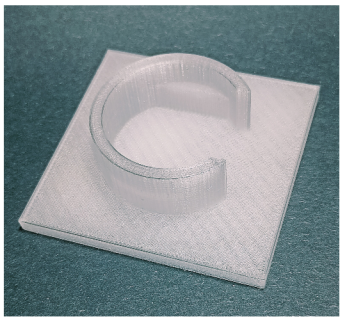
Obr. 5.8: Realizovaný testovací přípravek



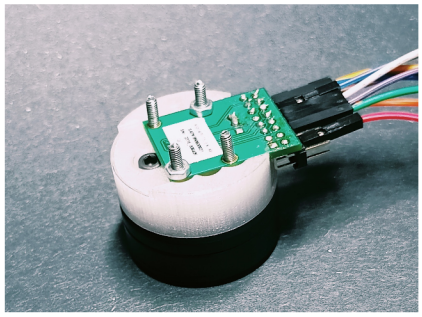
Obr. 5.9: Armatura pro motor

Tato armatura se skládá ze tří dílů: podstavce, centrálního bloku a pouzdra pro magnet. Centrální blok je pomocí šroubů připevněn k plášti statoru motoru i k modulu měřícímu polohu a je umístěn do podstavce. Pouzdro na magnet bylo vytvořeno aby jej bylo možné zasunout do ložiska v ose rotoru a vhodně tak magnet umístit vůči čipu AS5147P na modulu pod pouzdrem. Na obrázku č. 5.10 je fotografie sa-

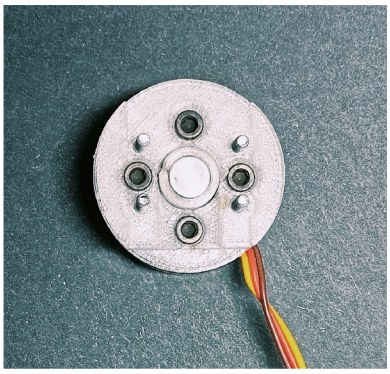
motného podstavce armatury. Centrálním blok s připevněným motorem a modulem AS5147P-TS_EK_AB je na fotografii v obrázku č. 5.11. Fotografie na obrázku č. 5.12 znázorňuje umístění magnetu uvnitř pouzdra zasunutého do středu motoru.



Obr. 5.10: Armatura pro motor - podstavec



Obr. 5.11: Armatura pro motor - sestavený centrální blok vyjmutý z podstavce



Obr. 5.12: Armatura pro motor - pouzdro s magnetem pod vyjmutým modulem AS5147P-TS_EK_AB

6 Tvorba řídicí aplikace pro jádro Cortex-M4

Dle návrhu popsaného v obr. 5.1 je potřeba aby se program koprocesoru zabýval obsluhou hardware součástí a aby realizoval proudovou smyčku kaskádní regulace. Program také musí komunikovat s vyšší vrstvou.

Pro ovládání motoru je nutné aby zařízení generovalo 3 PWM signály vyžadované modulem IHM16M1. Dále je také nutné nakonfigurovat 3 kanály pro ADC převody zpětné proudové vazby z tohoto modulu a je vyžadován jeden GPIO výstup pro ovládání enable signálu pro jednotlivé větve vinutí. Aby bylo možné měřit polohu z PWM signálu poskytovaného čipem AS5147P, bude nutné měřit střídu tohoto signálu. Posledním požadavkem na konfiguraci je dostupnost SPI rozhraní pro konfiguraci AS5147P-TS_EK_AB vývojového modulu.

Pro časování regulační smyčky byla navržena struktura aplikace řízená časovačem generujícím PWM signály pro ovládání motoru. Tento časovač bude nastaven do režimu generování PWM a bude generovat událost při podtečení interního čítače. Na tuto událost bude navázáno spouštění převodu sekvence ADC kanálů po jejímž dokončení je zavolána callback funkce. Algoritmus pro výpočet akčních zásahů bude prováděn v tomto callbacku který bude mít přesně danou frekvenci díky odvození od události časovače.

6.1 konfigurace v STM32CubeMX

Pro vytvoření kostry programu a konfiguraci periferií procesoru je vhodné založit projekt v STM32CubeMX. Zde je možné jednoduše a rychle nadefinovat chování jednotlivých periferií potřebných pro projekt. Společně se základem programu jsou také generovány fragmenty pro sestavení stromu zařízení při sestavování modifikované distribuce Linux systému pro hlavní jádro. Jako výchozí bod pro nastavení projektu byl zvolen přímo modul STM32MP157F-DK2.

Během konfigurace nebylo pro periferie potřeba měnit zdroje hodinových signálů, jejich nastavení tak bylo v STMCubeMX ponecháno v konfiguraci výchozí pro použitý vývojový modul.

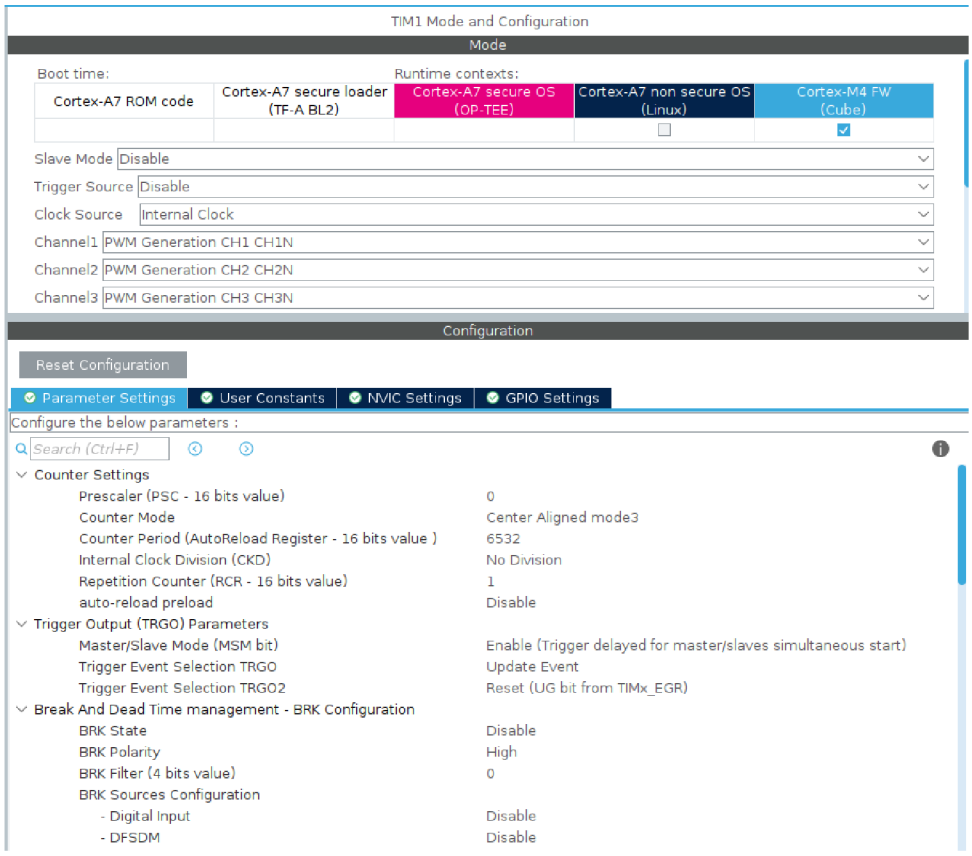
Před generováním projektu bylo také nutné změnit nastavení vývojového prostředí pro které bude generován konfigurační soubor jelikož program STM32CubeIDE použitý při tomto projektu nebyl pro tuto položku výchozí volbou.

6.1.1 Ovládání motoru - Advanced Control Timer

STM ve svých mikrokontrolérech nabízí periferii **Advanced Control Timer**. Tato periferie umožňuje oproti časovačům pro obecné použití vyšší míru konfigurova-

telnosti. Advanced Control Timer časovače také obsahují speciální funkce, jako je například nastavení Dead Time intervalu pro zabránění otevření obou tranzistorů v push-pull konfiguraci. Tento typ časovače je vhodný pro řízení motoru a je použitý v projektu, STM32MP157 nabízí dva časovače tohoto typu: **TIM1** a **TIM8**. Časovač **TIM1** má signály pro své kanály příhodně dostupné na Arduino konektoru vývojové desky a byl zvolen pro použití v projektu.

Konfigurace časovače vyžaduje nastavení 3 kanálů časovače pro generování PWM signálů, zde je možné zvolit režimy **PWM Generation CHx** nebo **PWM Generation CHx CHxN**. Režim s CHxN pouze umožní generovat i komplementární signál pro případ že by bylo potřeba přenastavit režim STSPIN830 do módu řízení 6ti PWM signály. Konfigurace časovače je zobrazena na obrázku č. 6.1.



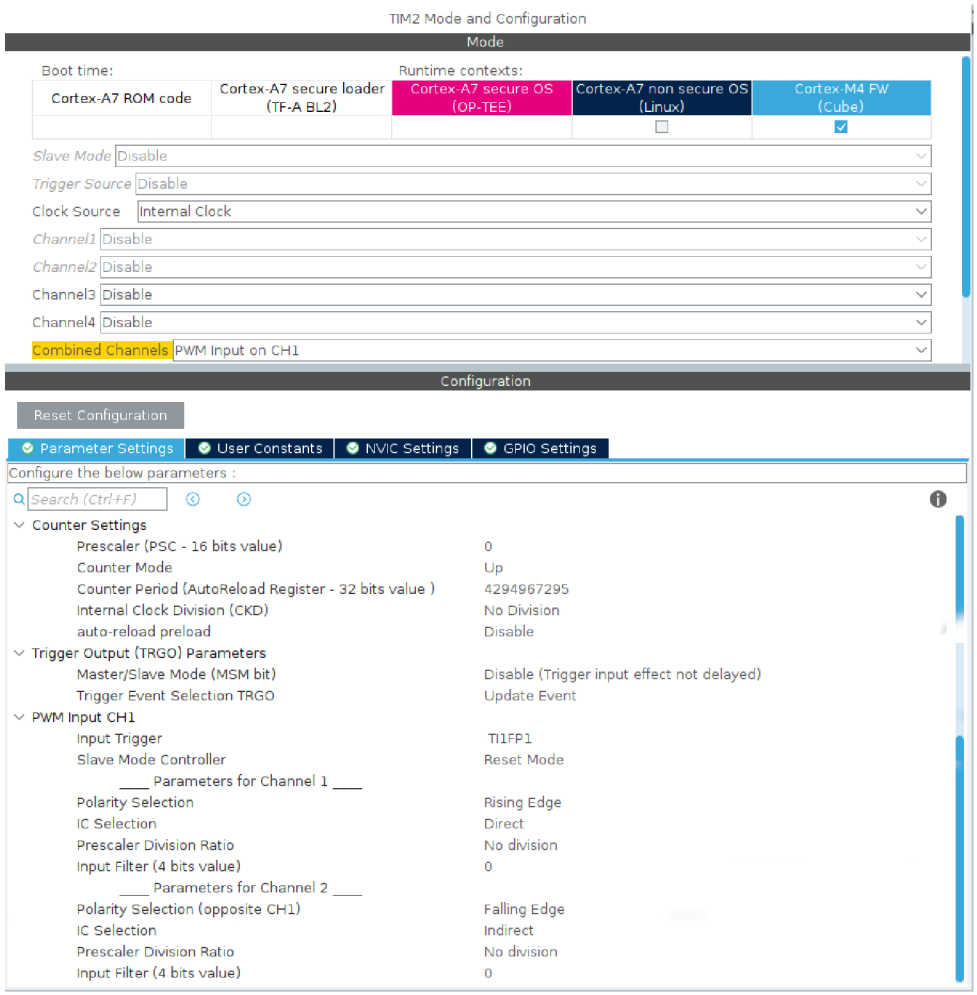
Obr. 6.1: Advanced Control Timer konfigurace

Aby časovač mohl spouštět převody ADC periferie, je nutné generovat správnou událost pomocí nastavení **Trigger Event Selection** na **Update Event**. Zároveň je potřeba nastavit **Repetition Counter** na hodnotu 1, aby se tato událost spouštěla pouze při každém druhém přetečení/podtečení čítače. Toto nastavení společně s výběrem režimu **PWM Mode 2** pro jednotlivé kanály zajistí že tato událost bude generována pouze při podtečení tohoto čítače.

Zdrojové hodiny jsou zvoleny na systémové hodiny s frekvencí 209 MHz. Aby bylo docíleno frekvence signálu 16 kHz je potřeba změnit **Counter Period** na hodnotu 6532. Pro **Center Aligned** režim generování PWM signálu je tato hodnota poloviční oproti ostatním režimům, jelikož časovač během jedné periody čítá od 0 do této hodnoty a pak zpět od této hodnoty do nuly.

6.1.2 Měření polohy - General Purpose Timer

Pro měření střídy PWM signálu lze využít režim **PWM input** časovačů od STM. Pro použitou vývojovou desku je dostupný pouze jeden časovač s touto funkcionalitou jehož kanály lze vyvést na dostupné piny vývojového modulu: **TIM2**.



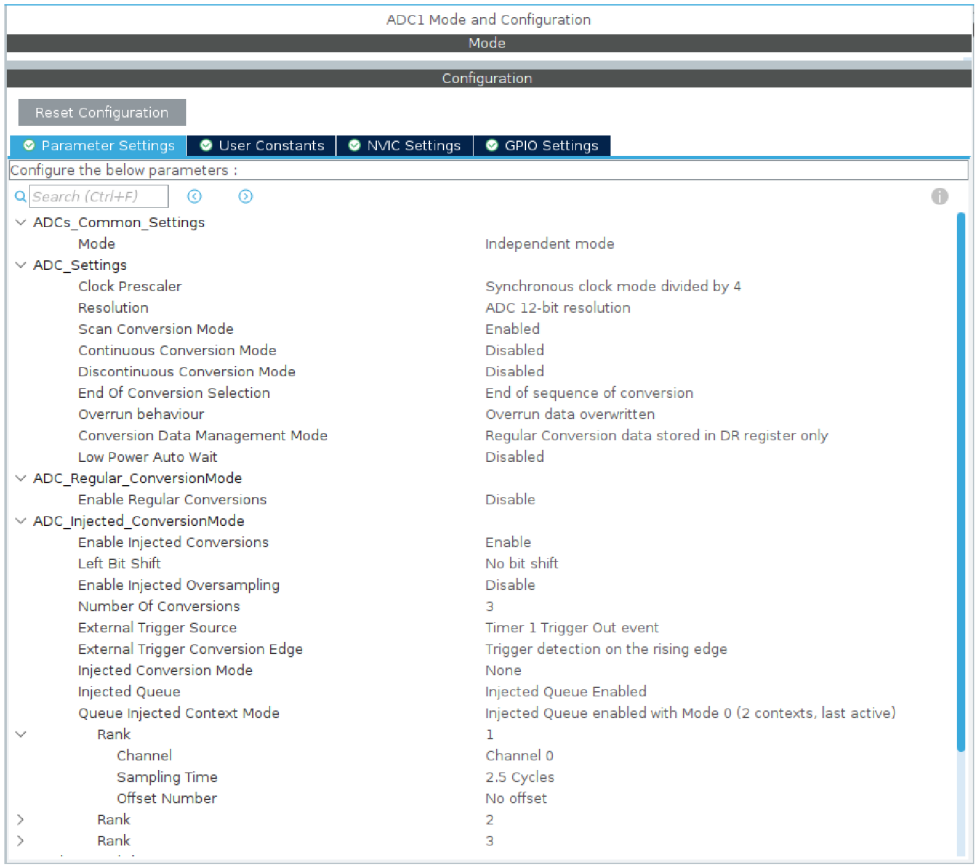
Obr. 6.2: konfigurace časovače pro měření střídy

Po zvolení režimu časovače a nastavení kanálů tak, že první kanál reaguje na náběžné hrany sledovaného signálu a druhý kanál naopak na hrany sestupné je potřeba nastavit periodu časovače dostatečně vysokou aby při měření nedošlo k jejímu

překročení a přetečení čítače. Další zpracování hodnot načtených tímto čítačem bude prováděno až v programu samotném během přerušení. Nastavená konfigurace časovače TIM2 je zobrazena na obrázku č. 6.2

6.1.3 Měření proudu - ADC převodník

STM32MP157 nabízí k dispozici dva ADC převodníky, ty jsou však vzájemně spřažené a nelze tak přidělit každému z procesorů uvnitř zařízení jeden z převodníků. Ačkoli je tedy v projektu využitý pouze převodník ADC1, bylo nutné procesoru Cortex-M4 přiřadit i převodník ADC2. Pro účely tohoto projektu je potřeba zvolit tři kanály z nabídky dostupných. Při měření byly nejprve použity kanály 0, 1 a 6, měření na prvním kanálu číslo 1 ale bylo zatížené výrazným rušením ve kterém se měřený signál efektivně ztrácel. Proto byl eventuálně zvolen kanál 13 pro náhradu tohoto kanálu.



Obr. 6.3: konfigurace ADC převodníku v Injected Queue režimu spouštěném z časovače TIM1

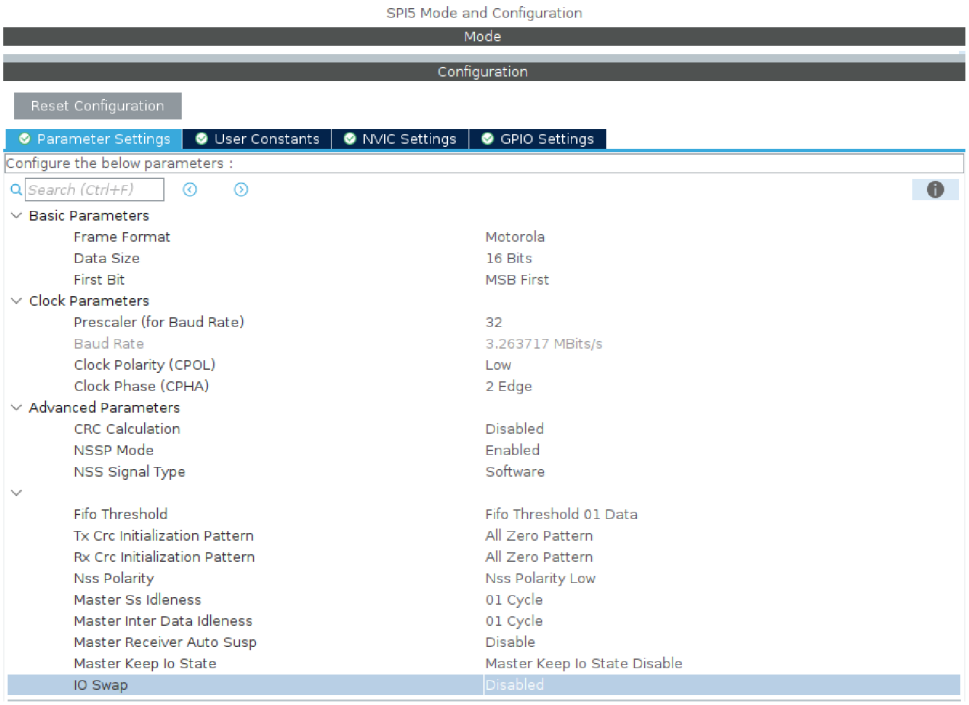
Vytvořená konfigurace přo převodník je vyvedena na obrázku č. 6.3. Pro nastavení spouštění převodů událostí z časovače TIM1 je potřeba nastavit hodnoty

External Trigger Source a **External Trigger Source Edge** aby odpovídaly nastavené události v časovači TIM1. Dále je nutné povolit **Injected Queue** a přenastavit **Number Of Conversions** na počet prováděných převodů. Tato konfigurace způsobí že při detekci události kterou je převod spuštěn bude provedena celá sekvence definovaných převodů, které lze definovat v sekcích konfigurace pod záložkou **Rank x**.

Je vhodné kontrolovat pořadí kanálů v této sekvenci, v programu se totiž vyčítají dle tohoto pořadí pomocí makra **ADC_INJECTED_RANK_x** a ne podle čísla kanálů.

6.1.4 komunikace s AS5147P-TS_EK_AB - SPI rozhraní

Pro konfiguraci modulu AS5147P-TS_EK_AB je potřeba nastavit SPI rozhraní. Pro používaný vývojový kit je vyžito rozhraní **SPI5** kvůli své dostupnosti na GPIO expandéru pod displejem. AS5147P využívá 16bitové rámce pro komunikaci a modul posílá nejvyšší bit zprávy jako první. Modul pracuje s SPI v konfiguraci CPOL = 0, CPHA = 1, data jsou tedy zpracovávána na sestupné hraně hodinového signálu.



Obr. 6.4: Konfigurace SPI rozhraní pro komunikaci s modulem AS5147P-TS_EK_AB

Tyto parametry je potřeba nakonfigurovat v nastavení periferie. Předděličku hodinového signálu bylo nutné nastavit tak aby rozhraní komunikovalo na frekvenci 3,2 Mbit/s. Vyšší rychlosti nebylo při zachování spolehlivosti komunikace možné

dosáhnout ačkoli datasheet modulu uvádí, že AS5147P podporuje rychlosti až do 10 Mbit/s. S kratší délkou kabelu či lepším spojením by komunikační rychlost bylo možné zvýšit, pro účely tohoto projektu však komunikační rychlost není omezující. Výslednou konfiguraci je možné vidět na obrázku č. 6.4.

Pro zprovoznění SPI rozhraní je také potřeba dodatečně použít jeden GPIO pin plnící funkci Chip Select. Modul zpracovává příkazy na náběžné hraně signálu CSn a není tedy možné jej permanentně připojit k zemi i když se jedná o jediné zařízení na SPI sběrnici.

6.1.5 SPI Chip Select a IHM16M1 Enable signály - GPIO

Pro použití v projektu je nakonec potřeba zvolit dva GPIO piny pro použití jako výstupy. Zde CubeMX nenabízí celou konfigurační stránku jako je tomu u ostatních zmíněných periferií, ale je potřeba pouze tyto piny aktivovat v **Pinout View** kartě. Pro funkci programu nemá vliv ani toto nastavení, konfiguraci je potřeba provést manuálně uvnitř vytvářeného programu i v případě že jsou piny nastaveny jako výstup v CubeMX. Nastavení GPIO pinů v CubeMX má tedy za následek pouze vyblokování použitých pinů pro ostatní funkce, čímž se zvýší přehlednost a piny budou také nadefinovány ve fragmentech pro stromy zařízení na straně Linux procesoru v modulu.

Pro Chip Select signál byl zvolen pin **PE1** (port E, pin 1) a pro Enable signál modulu IHM16M1 byl vybrán pin **PG3** (port G, pin 3). Oba tyto piny jsou dostupné na Arduino konektoru na spodní straně vývojové desky. Konfigurace těchto pinů je provedena v samotném programu pro Cortex-M4.

6.2 Tvorba programu v prostředí STM32CubeIDE

Po vygenerování kostry projektu je možné jej přímo otevřít ve vývojovém prostředí. STMCubeMX používá pro generované programy strukturu popsanou v sekci 2.1.1. Úpravy které budou prováděny v existujících souborech jsou tedy drženy uvnitř označených sekcí dle stylu popsaného na výpisu č. 2.1.

Výpis 6.1: Ilustrace stylu dokumentace nově vytvářených funkcí

```
/**
   * @brief Low-level SPI read-write
   * @param data 16bit frame to write to the bus
   * @retval 16bit data frame read from the bus
   * @note driver has to be initialized prior
   */
uint16_t as_spi_read_write(uint16_t data);
```

Nově tvořený kód je dokumentován pomocí komentářů a nově vytvářené hlavičkové soubory obsahují dokumentaci funkcí v Doxygen formátu v souladu se zbytkem projektu generovaného CubeMX. Příklad takového zápisu je uveden na výpise č. 6.1.

6.2.1 Ovládání motoru

Pro řízení motoru byla implementována jednoduchá funkce nastavující střídu PWM signálu pro jednotlivé kanály která zapíše požadovanou hodnotu do compare registru odpovídajícího kanálu. Byla také vytvořena komplementární funkce která zapíše invertovanou hodnotu střídy pro přímé řízení komplementárního PWM signálu. Kvůli použití PWM režimu 2 jsou komplementární a hlavní PWM signály v periferii zaměněny a výstup na hlavním PWM signálu daného kanálu je tak řízen skrze tuto komplementární funkci.

6.2.2 Implementace a validace funkce taktované řídicí smyčky

Dle návrhu software bude jádrem programu v procesoru Cortex-M4 blok kódu spouštěný periodicky z časovače TIM1. Aby v bloku kódu bylo možné pracovat s hodnotami naměřenými ADC převodníky, je tento blok navázán na callback funkci která je spouštěna po dokončení převodu sekvence Injected Queue převodů nadefinované v CubeMX. Callback funkce jsou v STM projektech nadefinovány pomocí **weak** klíčového slova a na jejich využití tedy postačí vytvořit novou funkci nesoucí stejné jméno jako callback funkce kterou chceme využít. Pro tento případ se jedná o funkci **HAL_ADCEx_InjectedConvCpltCallback(ADC_HandleTypeDef *hadc)**.

V této funkci je ve finální verzi programu větvící se **switch** blok který rozhoduje o tom jaký blok kódu se bude provádět na základě proměnné ukládající stav programu. Stavy aplikace jsou definované pomocí **enum** výčtu ve výpise č. 6.2.

Výpis 6.2: Definované stavy aplikace

```
//Application state
enum ApplicationState
{
  None = 0,
  Setup,
  PreparingCalibrationADC,
  CalibratingADC,
  AngleSensingSetup,
  AngleOffsetSetup,
  MotorSetup,
  Idle,
  Running,
  TestSineOutput,
  Error,
  PositionRegulator,
  RegulatorDesign,
} app_state = None;
```

Stavy programu se dělí na dvě části: stavy definované mezi **Setup** - **Idle** jsou stavy kterými program projde samovolně po spuštění. Během setrvání v těchto stavech dochází k inicializaci jednotlivých komponent, kalibraci měření a nastavování aplikace. Stavy následující v definici jsou provozní stavy do kterých je možné program přepnout a modifikovat tak chování celého programu při běhu. Dle stavové proměnné se v hlavním přerušení vykonává například samotná regulace, kalibrace měření proudu, zarovnání polohy nebo je generován testovací sinusový signál pro rychlý test funkce motoru.

Pro složitější aplikace by mohlo dojít k situaci kde je použito více ADC periferií a použitá callback funkce tak bude volána pro každou z nich. V takovém případě je možné využít argument ***hadc**, což je ukazatel na datovou strukturu popisující periferii ze které byla callback funkce volána. Při tvorbě projektu pomocí CubeMX jsou pro konfiguraci a následnou správu periferií vytvořeny datové struktury, pro konfiguraci tohoto projektu se jedná o struktury obsažené ve výpise č. 6.3. Porovnáním ukazatele s adresou datové struktury pro správu konkrétní periferie lze zjistit zda byla callback funkce zavolána právě z této periferie.

Výpis 6.3: Datové struktury pro správu a konfiguraci periferií generované v kostře projektu

```
/* Private variables ---------------------------------------------------------*/
ADC_HandleTypeDef hadc1;

IPCC_HandleTypeDef hipcc;

SPI_HandleTypeDef hspi5;

TIM_HandleTypeDef htim1;
TIM_HandleTypeDef htim2;
```

Inkrementací proměnné uvnitř této callback funkce byla nejprve ověřena perioda jejího volání při měření na časových úsecích různé délky. Takto bylo potvrzeno že se tento blok kódu spouští s frekvencí 16 kHz.

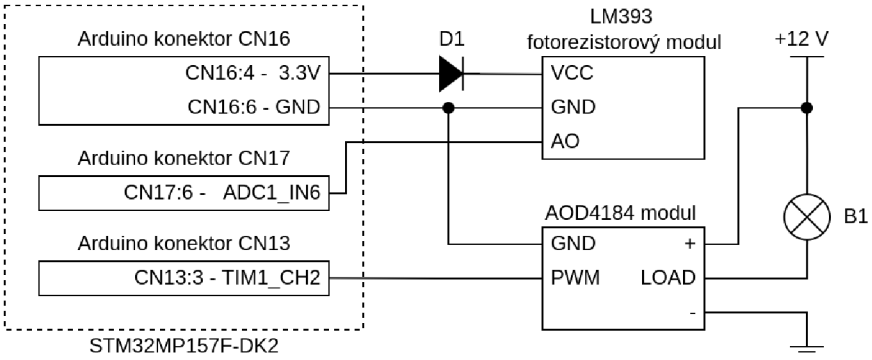
Pro algoritmus výpočtu regulačního zásahu je využita PID implemetace [1] dostupná pod MIT licencí, která byla zachována v souborech které byly převzaty. Použitá implementace umožňuje nastavit limity pro akční zásah i integrační složku. Je možné také nastavit filtr pro derivační složku, ta v projektu ale zůstává nepoužita. Kód byl převzat s cílem jej upravit v případě že by bylo potřeba zrychlit výpočet akčního zásahu. I s frekvencí přerušení 16 kHz však nedocházelo k problémům spojeným s příliš velkým zatížením procesoru, který i s hotovou aplikací zvládá všechny

[1] PID implemetace dostupná z: https://github.com/pms67/PID/tree/b871f94ebbeba8879304af916bfd63b07630f270, z repozitáře byly vyjmuty a použity pouze soubory PID.h a PID.c, do obou ze zdrojových souborů byla vložena MIT licence z repozitáře společně s odkazem na repozitář samotný

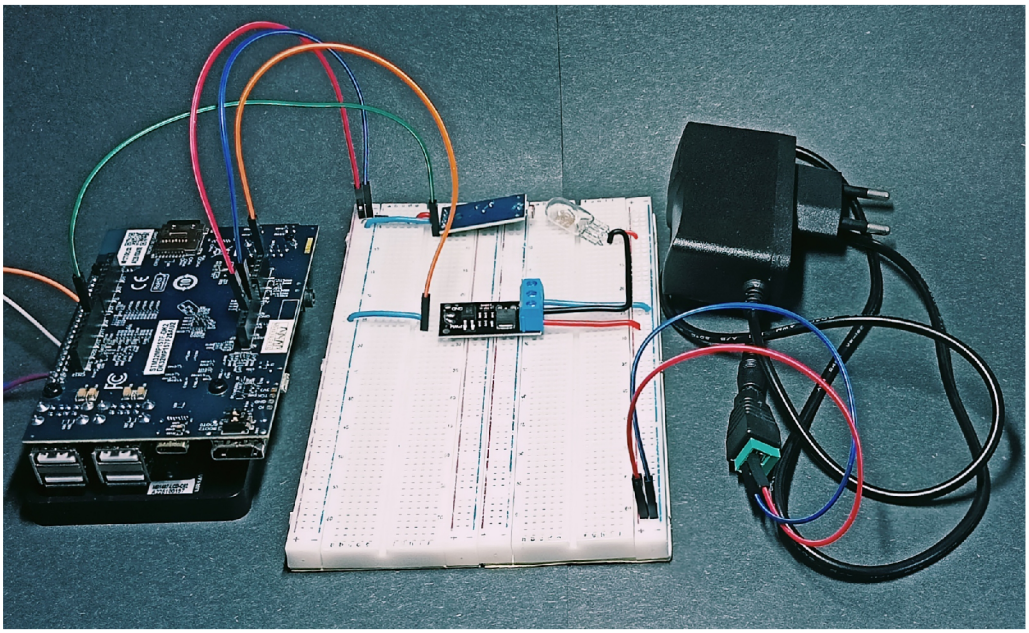
výpočty provést a vrátit se z přerušení do hlavní smyčky pro obsluhu komunikace s hlavním procesorem v zařízení. PID implementace byla tedy ponechána ve svém původním stavu.

Testování regulační smyčky

Pro ověření funkce regulační smyčky byl vytvořen testovací přípravek určený k regulaci proudu 12V žárovkou do automobilu. Jako zpětná vazba byl použit modul s fotorezistorem poskytující analogový výstup. Modul s MOSFET tranzistorem AOD4184 a optočlenem pro oddělení 3.3V a 12V větví slouží pro řízení proudu žárovkou. Pomocí tohoto testovacího přípravku byla ověřena funkce měření ADC převodníkem a také regulační smyčka která je později modifikována pro regulaci proudu motorem.



Obr. 6.5: Schéma přípravku pro testování regulační smyčky

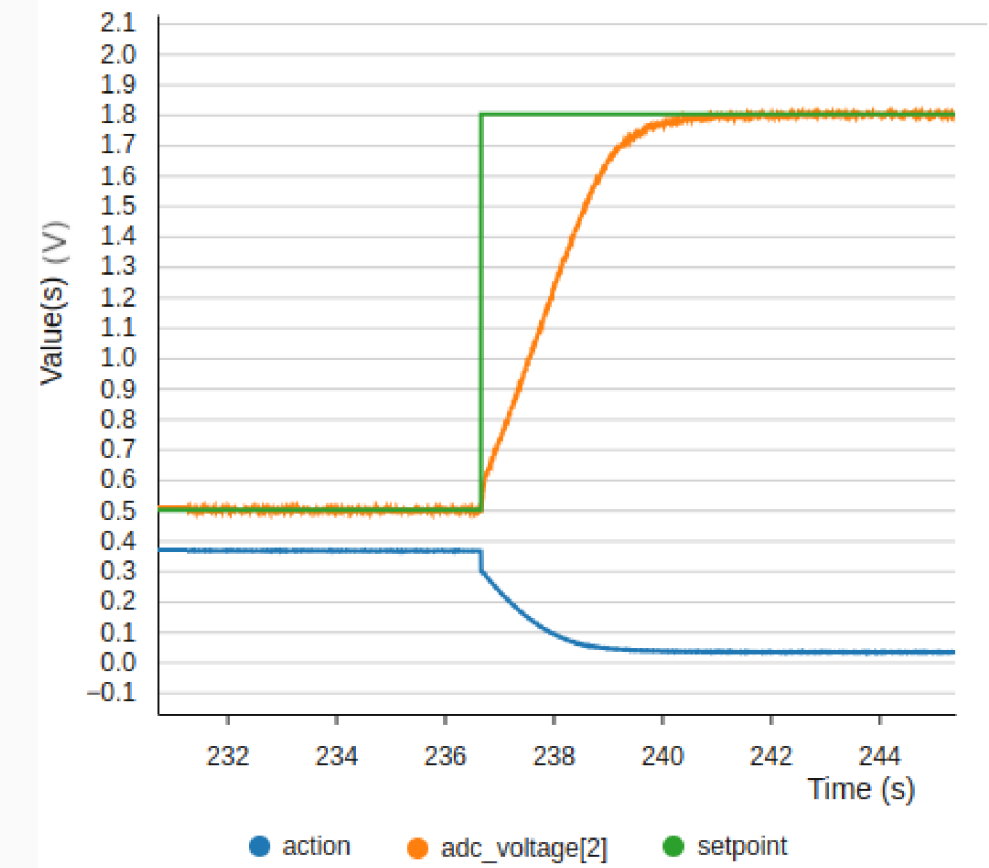


Obr. 6.6: Přípravek pro testování regulační smyčky

Schéma navrženého přípravku je zobrazeno na obrázku č. 6.5, obrázek č. 6.6 obsahuje fotografii realizovaného přípravku. K propojení přípravku k modulu STM32MP157F-DK2 jsou použity propojovací kabely které nejsou pevně připevněny, do přípravku proto byla přidána dioda D1 pro ochranu proti záměně napájecích kabelů a přepólování napájecího napětí modulu s komparátorem LM393.

Přechodovou charakteristiku regulovaného systému je možné vidět na obrázku č. 6.7. Tato charakteristika byla naměřena pomocí programu STM32CubeMonitor.

Program pro tento testovací přípravek je součástí eletronické přílohy. Pro tento test byl využit pouze procesor Cortex-M4 v engineering režimu.



Obr. 6.7: Přechodová charakteristika regulovaného systému

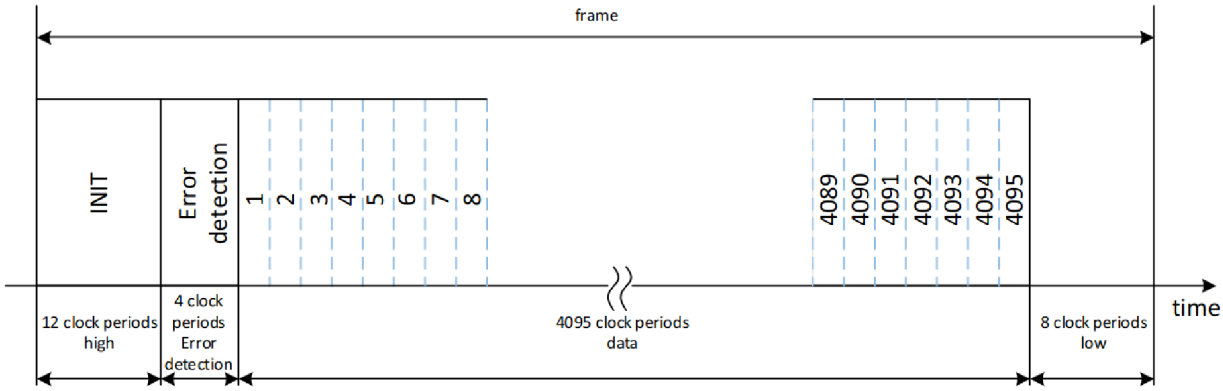
6.2.3 Měření polohy

Pro měření polohy byl v CubeMX nakonfigurován časovač TIM2 do režimu PWM Input. V tomto režimu kanál 1 čítá pulzy hodinového signálu mezi náběžnými hranami zkoumaného signálu a kanál 2 čítá pulzy zatímco je signál ve stavu logická 1. Výpočet střídy obecného signálu pak spočívá v implementaci callback funkce

pro přerušení **HAL_TIM_IC_CaptureCallback**, kde je pouze potřeba podělit hodnoty vyčtené z čítačů obou kanálů.

PWM signál generovaný AS5147P má formát zobrazený na obrázku č. 6.8. Při výpočtu střídy ze které je následně vypočten úhel natočení rotoru je tedy potřeba aplikovat korekci pro části rámce které neobsahují informaci o poloze dle vzorců 6.1, kde *pulzy_int* je počet pulzů zdrojového hodinového signálu časovače načtených kanálem měřícím šířku pulzu a *pulzy_per* je počet pulzů z kanálu, který měří periodu signálu. *Korekce_int* je určena jako počet hodinových pulzů časovače který bude načten za interval 16ti pulzů hodin generátoru PWM signálu. Obdobně je určena i hodnota *korekce_per*, kde je pouze vypočten počet pulzů pro 24 period hodin generátoru PWM. PWM signál je generován se zdrojem o periodě 444 ns, zatímco zdroj časovače je nastaven na systémové hodiny o frekvenci 209 MHz, výsledné hodnoty korekcí jsou nastaveny jako 2227 pro *korekce_per* a 1485 pro *korekce_int*.

$$Střída = (pulzy\_int - korekce\_int)/(pulzy\_per - korekce\_per) \tag{6.1}$$



Obr. 6.8: Formát PWM signálu generovaného AS5147P [18]

Měřená střída je ukládána ve formátu v rozmezí 0.0 - 100.0 % pro zobrazování v STM32CubeMonitor. Ze střídy je následně vypočten mechanický úhel a vynásobením počtem pólových dvojic motoru i elektrický úhel potřebný pro transformace vektorového řízení.

Po startu programu je také provedeno zarovnání polohy rotoru s první fází na nejbližší elektrické otáčce. Do jedné fáze motoru je spuštěna PWM střída 55% zatímco do zbylých dvou pouze 45% což způsobí natočení rotoru souhlasně s první fází. Motor je v této poloze ponechán po dobu kalibrace při které se průměruje měřená poloha, která se při výpočtu polohy během regulace odečítá od polohy měřené. Tato

kalibrace zajistí že měřená poloha je zarovnaná vůči první fázi pro transformace vektorového řízení.

SPI komunikace s modulem AS5147P

PWM rozhraní je po zapnutí zařízení vypnuté a je potřeba jej povolit zápisem do konfiguračního registru, vytvořená aplikace tedy obsahuje i ovladač vytvořený pro komunikační protokol využívaný modulem na rozhraní SPI. Tento protokol se sestává z 16bitových rámců které jsou odesílány ve dvojicích: řídicí a datový rámec. Dvojice rámců pak definují jednotlivé transakce protokolu.

Formát rámců je uveden v tabulce č. 6.1. První rámec v transakci je řídicí a definuje adresu registru pro nadcházející operaci a zda bude probíhat zápis nebo čtení. Rámce protokolu odeslané modulem také obsahují bit indikující zda došlo při komunikaci k chybě. V druhém rámci transakce jsou obsažena data k zápisu. Při operaci čtení je pro implementovaný proces řízení komunikace obsah druhého rámce irelevantní a pouze se vyčítají data o která bylo zažádáno v řídicím rámci. Pro zvýšení efektivity a rychlosti komunikace by při čtení dat z registru bylo možné odesílat již první rámec další transakce. Toto řešení by mohlo být efektivní v případě že je například potřeba opakovaně číst registr obsahující informaci o poloze kdyby bylo zvoleno toto rozhraní i pro měření polohy. Rámce protokolu dále obsahují paritní bit pro detekci chyb přenosu zpráv, zde je použita sudá parita. [18]

BIT	Obsah
15	paritní bit
14	bit pro zápis/čtení, v odpovědi indikuje chybu
13:0	adresa registru/data

Tab. 6.1: Formát SPI rámců AS5147P

Vytvořený ovladač obsahuje funkce pro provoz SPI periferie na nízké úrovni nad kterými implementuje vyšší vrstvu využívající datové struktury definované podle tabulky č. 6.1, struktura také využívá podvojný popis paměťového místa pomocí klíčového slova **union**. Tento způsob definice datové struktury umožnůje přistupovat k jednotlivým bitovým polím uvnitř zprávy a zároveň povoluje celou zprávu zpracovávat jako jedinou 16bitovou hodnotu během odesílání/přijímání skrze poskytnuté HAL funkce pro ovládání SPI rozhraní. Jedna z datových struktur je uvedena k nahlédnutí na výpise č. 6.4.

Výpis 6.4: Datové struktury definující formát zpráv SPI protokolu využívaného modulem AS5147P

```
//structure of the protocol command frame
union AS_command_frame
{
  //raw value
  uint16_t val;
  //bit fields
  struct
  {
    uint16_t address : 14;  //address to write to
    uint16_t rw      : 1;  //read/write flag - 1:read
    uint16_t parc    : 1;  //parity check bit
  }fields;
};
```

Zapnutí PWM výstupu spočívá v nastavení bitu **PWMON** v registru **SETTINGS1**. K provedení konfigurace tedy stačí jediná transakce. V aplikaci je konfigurace implementována jako cyklus opětovně provádějící zápis dokud při vyčtení registru bit PWMON není nastaven korektně. Tímto řešením je zvýšena robustnost v případě že by první pokus o konfiguraci neuspěl.

6.2.4 Měření proudů cívkami motoru

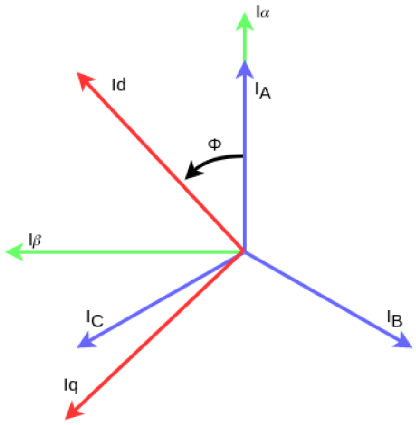
Při výpočtu měřených proudů ze signálů zpětných vazeb poskytovaných modulem IHM16M1 bylo zjištěno že katalogová hodnota klidového napětí na jednotlivých zpětných vazbách není identická pro všechny tři signály. Byla tedy implementována kalibrace po zapnutí programu která tuto hodnotu zkalibruje poté co je motor zastaven a ponechán v klidovém stavu po dobu kalibrace.

Kalibrace je prováděna v přerušení a je realizována iteračním výpočtem průměru z měřených hodnot. Délku kalibrace je tedy možné jednoduše upravit změnou délky intervalu po který je aplikace ponechána v tomto stavu.

6.2.5 Algoritmus vektorového řízení

Pro vektorové řízení motoru se používá dvojice transformací která nejprve transformuje proudy i_A, i_B a i_C (či jiné obvodové veličiny) ze tří vektorů do dvou vektorů v souřadnicového systému zarovnaného se statorem, zde vznikají proudy I_α a I_β. Druhou transformací jsou následně vektory transformovány do druhého souřadnicového systému zarovnaného s rotorem a vznikají tak proudy I_d a I_q. [19]

Vzájemné vztahy těchto vektorů jsou znázorněny na obrázku č. 6.9. Pro výpočet těchto transformací je potřeba znát aktuální elektrický úhel Φ rotoru zarovnaný s vektorem proudu I_A. Pro zpětnou transformaci lze použít inverzní transformace, nejprve opět z proudů I_d a I_q do I_α a I_β a následně do proudů I_A, I_B a I_C.



Obr. 6.9: Vektory proudu při transformacích vektorového řízení motoru

V programu jsou tyto transformace implementovány dle rovnic 6.2 a 6.3. Zpětné transformace jsou zpracovány dle rovnic 6.4 a 6.5.

$$I_\alpha = i_a \qquad I_\beta = \frac{2}{\sqrt{3}}(0,5i_a + i_b) \qquad [19] \quad (6.2)$$

$$I_d = I_\alpha cos(\phi) + I_\beta sin(\phi) \qquad I_q = -I_\alpha sin(\phi) + I_\beta cos(\phi) \qquad [19] \quad (6.3)$$

$$I_\alpha = I_d cos(\phi) - I_q sin(\phi) \qquad I_\beta = I_d sin(\phi) + I_q cos(\phi) \qquad [19] \quad (6.4)$$

$$i_A = I_\alpha \qquad i_B = \frac{-1}{2}I_\alpha + \frac{\sqrt{3}}{2}i_\beta \qquad i_C = \frac{-1}{2}I_\alpha - \frac{\sqrt{3}}{2}i_\beta \qquad [19] \quad (6.5)$$

Samotný proces vektorového řízení je založen na měření proudu v jednotlivých větvích motoru: i_A, i_B a i_C a jejich transformaci na proudové vektory I_d a I_q. Velikosti těchto vektorů jsou následně regulovány nezávislými regulátory, proud I_d je regulován do nulové hodnoty jelikož se nepodílí na tvorbě momentu, zatímco proud I_q je regulován na požadovanou hodnotu a vytváří moment na rotoru. Hodnoty generované regulátory jsou již interpretované jako napětí a pomocí zpětných transformací jsou převedeny na napětí pro aplikaci do jednotlivých větví vinutí u_A, u_B a u_C. Takto vypočtená napětí jsou následně použita k řízení motoru aktualizací střídy PWM signálů pro modul IHM16M1.

Pro zrychlení výpočtu jsou rovnice upraveny vyčíslením zlomků a koeficientů násobení. Příklad implementace rovnic 6.2 je uveden na výpisu č. 6.5.

Výpis 6.5: Implementace Clarkovy transformace dle rovnice 6.2

```
*i_alp = i_a;
*i_bet = 0.5773502692f * i_a + 1.1547005f * i_b;
```

Pro řízení proudové smyčky byl v této vrstvě implementován jednoduchý P regulátor.

6.2.6 Komunikace s nadřazeným procesorem

Pro předávání informací mezi procesory uvnitř STM32MP157 jsou využity předdefinované RPMsg kanály číslo 0 a 1. Pro komunikaci byl vytvořen jednoduchý protokol založený na textových zprávách. Kanálem 0 jsou předávány zprávy směrem z hlavního procesoru do Cortex-M4, na druhém kanále pak koprocesor odesílá zprávy zpět.

Každá zpráva obsahuje hlavičku o délce 4 znaky která definuje typ zprávy. Hlavička o fixní délce umožňuje využít rychlé porovnávání řetězců pomocí funkce **memcmp()**. Za hlavičkou následuje datový blok libovolné délky ve kterém je možné předat data. Pro ovládání programu v procesoru Cortex-M4 byly vytvořeny 4 příkazy: **MOD:**, **SET:**, **RST:** a **PID:**.

Příkaz **MOD:** je následován třemi znaky identifikujícími stav do kterého se má program přepnout, pomocí tohot příkazu je tak možné zapnout a zastavit regulaci nebo zapnout generování testovacího sinusového signálu pro motor příkazem **MOD:SIN**.

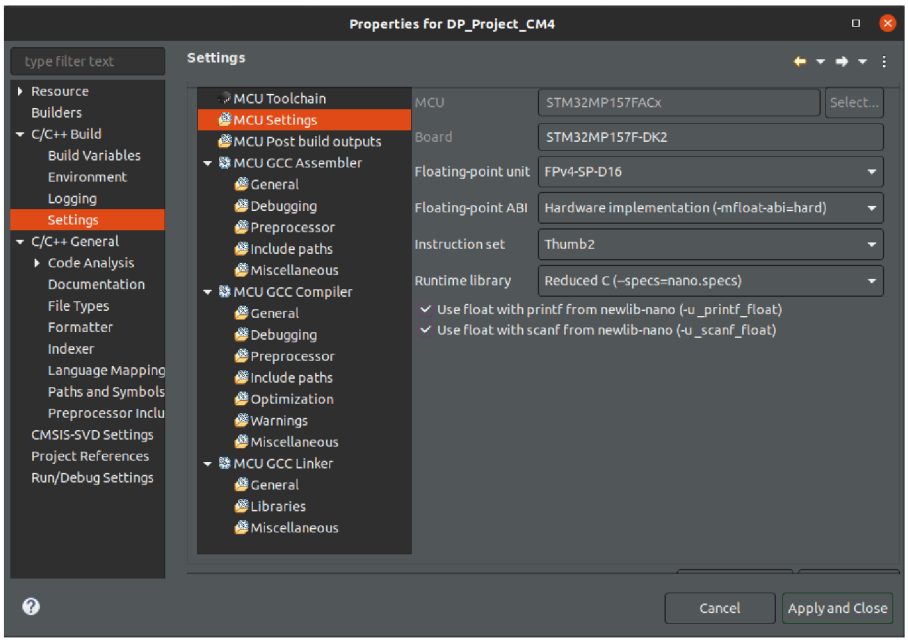
Pro aktualizaci žádané hodnoty proudu z vyšší vrstvy kaskádní regulace je zaveden příkaz **SET:**. Tuto hlavičku následuje desetinné číslo zapsané jako řetězec znaků. Pro zapnutí podpory formátování desetinných čísel do textových řetězců je pro Cortex-M4 potřeba upravit nastavení projektu dle obrázku č. 6.10

Příkaz **RST:** byl vytvořen jako nouzová možnost restartovat program v procesoru Cortex-M4. Příkaz je implementován skrze instrukci **goto**, která program vyvede z hlavní smyčky zpět na inicializaci.

Pomocí příkazu **PID:** lze přenastavit parametry regulátoru uvnitř proudové smyčky, hlavičku následují dvě desetinná čísla oddělená čárkou která definují zesílení složky P a I, příkaz má tedy formát **PID:Kp,Ki**. Tento příkaz nebyl při dalším vývoji implementován do programu pro hlavní procesor, ladění regulátoru v koprocesoru proběhlo v Engineering režimu procesoru.

Kromě těchto čtyř příkazů je definována také zpráva **POS:**, kterou program nepřijímá ale pouze odesílá na RPMsg kanále 1. Tato zpráva je určena pro program v Cortex-A7 a obsahuje informaci o měřeném mechanickém úhlu.

V hlavní smyčce programu se nachází blok kódu který se spouští jednou za každou smyčku nadřazeného regulátoru (zde každých 10 ms), v tomto bloku je odeslána



Obr. 6.10: Nastavení podpory formátování desetinných čísel v CubeIDE projektu pro Cortex-M4

zpráva **POS:** a je aktualizována žádaná hodnota proudu (pokud za uplynulých 10 ms dorazila zpráva **SET:**) pro proudový regulátor aby byla lépe dodržována konstantní perioda její aktualizace.

Příkazy na které program reaguje jsou uloženy v tabulce jako dvojice [hlavička, adresa funkce pro provedení příkazu]. Tuto tabulku lze jednoduše modifikovat a rozšířit tak protokol o nové příkazy. Definice této tabulky je uvedena na výpisu č. 6.6.

Výpis 6.6: Definice tabulky zpracovávaných příkazů z hlavičkového souboru v projektu

```
typedef bool (*handler) (char* argument, size_t length);

//actual occupied vector table size
extern size_t act_command_vector_table_size;

//command vector structure storing command and handler pointer
struct _cmd_vector
{
  char  command [COMMAND_LENGTH];
  handler handler_pointer;
};

//vector table
extern struct _cmd_vector command_vector_table [MAX_TABLE_SIZE];
```

7 Program v procesoru Cortex-A7

Před tvorbou aplikace pro hlavní procesor byla vytvořena upravené distribuce OpenSTLinux s pomocí souborů vytvořených programem STM32CubeMX. Změny a kompilace distribuce byly provedeny dle návodu poskytnutého výrobcem. [0]. Výsledná distribuce byla následně nahrána na zkoumaný modul pomocí aplikace STM32CubeProgrammer.

Program v hlavním procesoru je méně komplexní než program v koprocesoru a sestává se pouze z hlavního souboru main.c a identické PID implementace také použité v programu pro Cortex-M4.

Program vychází z kostry vytvořené pro měření rychlosti výměny dat mezi procesory v zařízení vytvořené ve čtvrté kapitole. Hlavní vlákno programu nejprve vytvoří vedlejší vlákno s vysokou prioritou které bude cyklicky spouštěno s předem definovanou periodou a které realizuje polohovou regulaci a komunikaci s koprocesorem. Po vytvoření tohoto vlákna vstupuje hlavní větev programu do smyčky ve které čte a zpracovává příkazy obdržené ze vstupního terminálu. V případě potřeby pak hlavní vlákno signalizuje periodickému procesu potřebu odeslat zprávu do koprocesoru.

Komunikace je provozována v jediném vlákně aby nebylo nutné se zabývat správou otevírání a přístupu k souborům v aplikaci s více vlákny. V případě že hlavní vlákno obdrží od uživatele příkaz vyžadující komunikaci s koprocesorem, je tento fakt signalizován druhému vláknu pomocí proměnných které obě vlákna sdílí. Před vstupem do periodické smyčky je programem do obou RMPsg kanálů odeslána zpráva, která zprovozní komunikaci s koprocesorem na tomto kanálu.

Pro komunikaci s uživatelem je využit stejný protokol jako pro výměnu dat mezi samotnými procesory, příkazem **SET:1.42** tak lze například nastavit požadovanou polohu pro polohový regulátor. Implementované příkazy jsou: **RST:** který resetuje program v koprocesoru, **SET:**, který aktualizuje žádanou polohu, příkaz **MOD:** který přepíná režim programu v Cortex-M4 a **EXT:**, který pouze ukončí program běžící v hlavním procesoru.

Hlavní část periodicky spouštěného cyklu vytvořeného vlákna vysoké priority spočívá ve vyčtení zprávy o poloze odeslané z koprocesoru do vlastního datového bufferu a jejím zpracování, následného vypočtení akčního zásahu polohového regulátoru a odeslání zprávy aktualizující žádanou hodnotu pro proudový regulátor v Cortex-M4. Kód pro tento blok programu je uveden ve výpise 7.1. Z vložené části programu byla pro svou velikost a malou informační hodnotu odstraněna sekce kódu která zprávu SET:v bufferu pro odesílání zpráv přepíše zprávou MOD: pokud hlavní vlákno programu žádá o změnu stavu v koprocesoru.

Výpis 7.1: Periodický blok kódu spouštěný aplikací v hlavním procesoru STM32MP157

```
//read message from coprocessor
err = read(fd_ch1,rx_buffer,sizeof(rx_buffer));

//process message if one is received
if(err != -1)
{
    printf("Processing message: %s\n", rx_buffer);
    process_message(rx_buffer,strlen(rx_buffer));
}

//only if position regulator is running
if(regulating)
{
    //update setpoint for nested controller
    current_setpoint = PIDController_Update(&pid_pos,
    angle_setpoint, angle_measured);

    //assemble setpoint update message
    sprintf(tx_buffer,"SET:%f",current_setpoint);

    send_flag = true;
}

...

//send a message to the coprocessor
...
if(send_flag)
{
    printf("sending command: %s\n", tx_buffer);

    err = write(fd_ch0,tx_buffer, sizeof(tx_buffer));
}
```

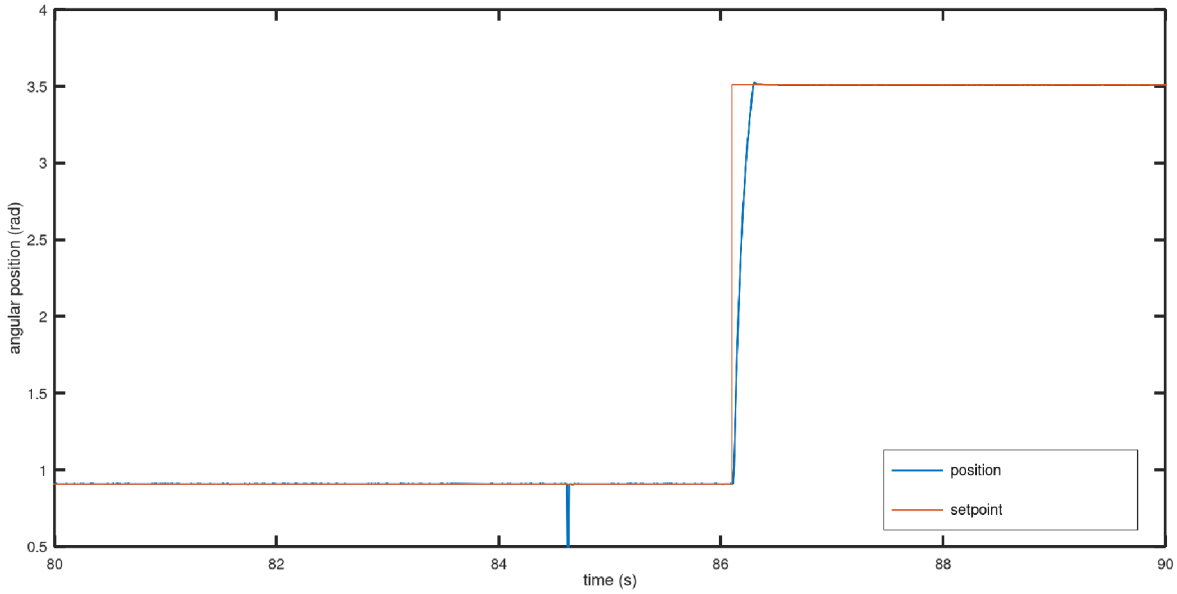
```
Processing message: POS:3.505853
updated measured angle to: 3.505853
sending command: SET:0.005854
Iteration time: 10028
Processing message: POS:3.507495
updated measured angle to: 3.507495
sending command: SET:0.005821
Iteration time: 10030
Processing message: POS:3.507558
updated measured angle to: 3.507558
sending command: SET:0.005820
Iteration time: 10028
sending command: SET:0.005820
Iteration time: 10026
Processing message: POS:3.507517
updated measured angle to: 3.507517
sending command: SET:0.005821
Iteration time: 10029
```

Obr. 7.1: Výstup výsledné aplikace

Program zároveň s prováděním regulace zapisuje do terminálu obdržené zprávy z koprocesoru, intervaly mezi spouštěním periodického vlákna a nastavovaný žádaný proud který bude odeslán do koprocesoru. Tyto hodnoty lze zapisovat do souboru pro pozdější zpracování například pomocí Linux příkazu **script**. Ukázka výstupu programu je zobrazena na obrázku 7.1.

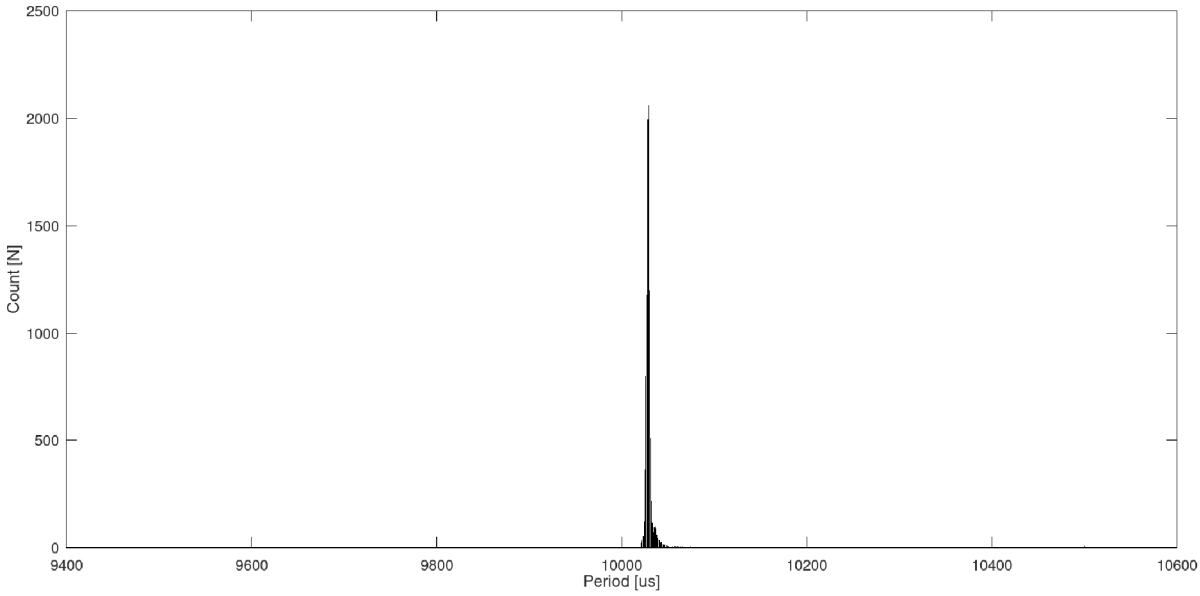
7.1 Měření regulačního děje a časování hlavní smyčky

Pro polohovou regulaci byl navržen PI regulátor pracující ve vlákně s periodou 10 ms. Přechodová charakteristika polohové regulace je zobrazena na obrázku č. 7.2. Ztráta informace o poloze mezi časy 84-85 sekund je pravděpodobně způsoben přerušením v operačním systému kvůli kterému je momentálně nedodrženo časování spouštění smyčky. Intervaly a přesnost spouštění regulační smyčky v Linux distribuci je možné vidět na obrázku č. 7.3.



Obr. 7.2: Přechodová charakteristika polohové regulace

Při pokusech o snížení periody cyklu polohové regulace na řádově jednotky ms se začíná projevovat jitter v systému a komunikace mezi procesory v zařízení začíná být nespolehlivá. V bufferech pro předávání dat se při nižších periodách hromadí zprávy a regulace přestává být efektivní. Pro vyšší spolehlivost časově kritických bloků kódu vyžadujících komunikaci mezi procesory tedy není vhodné tyto sekce časovat s periodou nižší než cca zvolených 10 ms. Při testování však nebyl zcela



Obr. 7.3: Intervaly mezi spuštěními periodického vlákna v Cortex-A7

vytížen procesor Cortex-M4 a bylo by možné do něj přesunout další časově kritické sekce, což by ponechalo Cortex-A7 méně vytíženým.

Závěr

V rámci této práce byla prozkoumána procesorová řada STM32MP1 a její schopnosti při provozování časově kritických procesů. Po úvodním zkoumání této řady se zbytek této práce zabývá užší částí procesorové řady: STM32MP15x, které kromě hlavního jádra s Linux systémem provozují koprocesory. Možnost přesunout části kódu do správy koprocesoru pro snížení zatížení hlavního procesoru a také vylepšení vlastností programu s časově kritickými částmi. Tyto části programu mohou benefitovat z provozu na procesoru nezatíženém operačním systémem.

Při zkoumání vlastností procesoru byl využit modul STM32MP157F-DK2. Procesorová řada i modul samotný jsou popsány v první kapitole této práce.

V rámci práce byly studovány a v kapitolách popsány dostupné materiály a podklady poskytované výrobcem zkoumaného modulu: STMicroelectronics. Vyšší pozornost byla také věnována ekosystému softwarových nástrojů, vývojářských balíčků a zdrojů relevantní dokumentace v kapitole 2. Dílčím cílem této práce je poskytnout čtenáři souhrn těchto nástrojů a popis procesů potřebných pro zprovoznění a započetí vývoje programů pro zkoumanou novou řadu procesorů. V práci je také popsána práce distribučním balíčkem pro OpenSTLinux, distribuci operačního systému určenou pro provoz na procesorech řady STM32MP1.

Pro modul STM32MP157F-DK2 bylo vytvořeno několik aplikací pro testování. Byla ověřena je funkce na demonstračním programu, tento proces je popsán kapitolou 3. Dále byly měřeny latence při výměnně dat mezi hlavním procesorem a koprocesorem. Tyto latence byly měřeny pomocí vlastní aplikace a výsledky měření je možné sledovat na grafech v obrázcích č. 4.2 v kapitole 4.

Druhou testovací aplikací je kaskádní regulace motoru s polohovou smyčkou provozovanou na hlavním procesoru a proudovou smyčkou běžící v koprocesoru. Tato aplikace vyžaduje provoz časově kritického programu v koprocesoru za udržení přesného časování smyčky regulátoru v hlavním procesoru. Oba procesory si také během běhu programu musí stihnout vyměňovat data. Kapitola 5 pojednává o procesu návrhu této aplikace včetně návrhu a realizace testovacího přípravku pro provoz výsledné aplikace. Kapitola 6 pojednává o vývoji části programu pro koprocesor zatímco kapitola 7 se zabývá vývojem zbylé části aplikace provozované v hlavním procesoru. Změřené latence při časování hlavní smyčky je možné sledovat na obrázku 7.3.

Zkoumaný procesor byl úspěšně otestován při provozu časově kritické aplikace. Při potřebě přesně časovat spouštění periodických úloh v programu v hlavním procesoru zařízení je stále možné pozorovat nedeterministické latence. Tyto latence je možné omezit použitím modifikované distribuce. Pro realizovanou aplikaci byla nejnižší stabilní perioda určena jako 10ms. Při této periodě časovaného vlákna v hlav-

ním procesoru stále dochází ke spolehlivé výměnně dat mezi procesory a jitter kterým je zatížené časování spouštění vlákna není problematický pro zvolenou aplikaci.

Další potencionální zlepšení vlastností zařízení provozující hlavní procesor může přinést nově oznámená řada procesorů STM32MP2, která má oproti procesorům zkoumaným v této práci poskytovat větší výkon a využívat 64bitovou architekturu místo 32bitové.

Literatura

[1] STMICROELECTRONICS. *STM32MP1 Series*. Online. C 2023. Dostupné z: https://www.st.com/en/microcontrollers-microprocessors/stm32mp1-series.html. [cit. 2024-01-03].

[2] STMICROELECTRONICS. *STM32MP157C Product overview*. Online. C 2023. Dostupné z: https://www.st.com/en/microcontrollers-microprocessors/stm32mp157c.html. [cit. 2023-12-17].

[3] STMICROELECTRONICS. *RM0436 Reference manual: STM32MP157 advanced Arm ® -based 32-bit MPUs*. Online. 2022, 29. dubna 2022. Dostupné z: https://www.st.com/resource/en/reference_manual/rm0436-stm32mp157-advanced-armbased-32bit-mpus-stmicroelectronics.pdf. [cit. 2023-10-11].

[4] STMICROELECTRONICS. *IPCC internal peripheral*. Online. STMICROELECTRONICS. STM32MPU ecosystem user guide. ©2023. Dostupné z: https://wiki.st.com/stm32mpu/wiki/IPCC_internal_peripheral. [cit. 2023-10-11].

[5] STMICROELECTRONICS. *Linux Mailbox framework overview*. Online. © 2023. Dostupné z: https://wiki.st.com/stm32mpu/wiki/Linux_Mailbox_framework_overview. [cit. 2023-10-11].

[6] STMICROELECTRONICS. *Linux RPMsg framework overview*. Online. 2019, 3.5.2022. Dostupné z: https://wiki.st.com/stm32mpu/wiki/Linux_RPMsg_framework_overview. [cit. 2024-01-02].

[7] STMICROELECTRONICS. *STM32MP157C-DK2 Product overview*. Online. C 2023. Dostupné z: https://www.st.com/en/evaluation-tools/stm32mp157c-dk2.html. [cit. 2023-12-17].

[8] STMICROELECTRONICS. *STM32 Software Development Tools*. Online. C 2023. Dostupné z: https://www.st.com/en/development-tools/stm32-software-development-tools.html. [cit. 2024-01-02].

[9] STMICROELECTRONICS. *STM32MP1 Developer Package*. Online. C 2023, 30.6.2023. Dostupné z: https://wiki.st.com/stm32mpu/wiki/STM32MP1_Developer_Package. [cit. 2023-12-30].

[10] STMICROELECTRONICS. *STM32MP1 Distribution Package*. Online. C 2023, 13.6.2023. Dostupné z: https://wiki.st.com/stm32mpu/wiki/STM32MP1_Distribution_Package. [cit. 2023-12-30].

[11] STMICROELECTRONICS. *X-LINUX-RT expansion package*. Online. 2023, 25.10.2023. Dostupné z: https://wiki.st.com/stm32mpu/wiki/X-LINUX-RT_expansion_package. [cit. 2024-01-02].

[12] STMICROELECTRONICS. *How to exchange data buffers with the coprocessor*. Online. C 2023, 17.11.2023. Dostupné z: https://wiki.st.com/stm32mpu/wiki/How_to_exchange_data_buffers_with_the_coprocessor. [cit. 2024-01-01].

[13] LINUX FOUNDATION. *HOWTO build a simple RT application*. Online. 2017, 29.8.2023. Dostupné z: https://wiki.linuxfoundation.org/realtime/documentation/howto/applications/application_base. [cit. 2024-01-02].

[14] LINUX FOUNDATION. *HOWTO build a basic cyclic application*. Online. 2017, 20.6.2017. Dostupné z: https://wiki.linuxfoundation.org/realtime/documentation/howto/applications/cyclic. [cit. 2024-01-02].

[15] STMICROELECTRONICS. *P-NUCLEO-IHM03 Data brief*. Online. C 2019, 19.4.2019. Dostupné z: https://www.st.com/resource/en/data_brief/p-nucleo-ihm03.pdf. [cit. 2024-03-28].

[16] STMICROELECTRONICS. *E X-NUCLEO-IHM16M1 User manual*. Online. 2018, 21.2.2024. Dostupné z: https://www.st.com/resource/en/user_manual/um2415-getting-started-with-the-xnucleoihm16m1-threephase-brushless-motor-dr pdf. [cit. 2024-03-28].

[17] AMS OSRAM. *AS5147P Adapter Board Eval Kit Manual*. Online. 2014, 14.1.2014. Dostupné z: https://look.ams-osram.com/m/93b89d16503327e0/original/AS5147P_AB_UG000375_1-00.pdf. [cit. 2024-03-28].

[18] AMS OSRAM. *AS5147P Datasheet*. Online. [cca 2015], 27.4.2016. Dostupné z: https://look.ams-osram.com/m/292a62bf288ae88a/original/AS5147P-DS000328.pdf. [cit. 2024-03-28].

[19] NEBORÁK, Ivo. *Modelování a simulace elektrických regulovaných pohonů*. Ostrava: Vysoká škola báňská - Technická univerzita, 2002. ISBN 8024800837.

STMICROELECTRONICS. *How to compile the device tree with the Distribution Package*. Online. 2019, 15.3.2023. Dostupné z: https://wiki.stmicroelectronics.cn/stm32mpu/wiki/How_to_compile_the_device_tree_with_the_Distribution_Package. [cit. 2024-03-28].

A Obsah elektronické přílohy

Elektronická příloha této práce obsahuje tři hlavní soubory s jednotlivými aplikacemi vytvořenými během tvorby diplomové práce.

V adresáři **Timing_STM32MP1** se nachází testovací aplikaci popsaná v kapitole 4. Uvnitř se nachází samotný spustitelný binární soubor **timing** společně se soubory potřebnými pro sestavení aplikace a krátkým popisem uvnitř souboru **README.md**. Pro sestavení aplikace pomocí kompilátoru získaného z SDK pro modul STM32MP157F-DK2 je potřebný makefile soubor společně se zdrojovým kódem obsaženým v souboru main.c umístěným ve složce sources.

Soubor **Regulator_testing** je místem uložení projektu použitého při testování regulační smyčky. Kapitola 6.2.2 popisuje zhotovený přípravek použitý společně s tímto programem. Pro otevření projektu je potřeba v prostředí CubeIDE naimportovat existující projekt a zvolit adresář **Regulator_testing** jako zdrojovou složku. Zde je při importování projektu do vývojového prostředí nutné otevřít i vnořené projekty **CA7** a **CM4**.

Poslední soubor je **Final_application**, uvnitř kterého se nachází demonstrační aplikace pro časově kritické řízení motoru. Tato složka se dále dělí na adresáře **Cortex-A7** a **Cortex-M4**. V adresáři Cortex-A7 se nachází makefile projekt obdobný tomu ve složce **Timing_STM32MP1** a liší se pouze přidanými zdrojovými soubory pro PID regulátor a jménem binárního souboru který je výsledkem kompilace, zde jde o soubor **application**. Soubor Cortex-M4 obdobně jako složka **Regulator_testing** uchovává projekt pro prostředí CubeIDE. Tyto dva projekty lze otevřít stejným způsobem. Ve složce Final_application se nachází také soubor **Distribution_package_layer** s vloženou vrstvou pro distribuční balíček. S pomocí zmíněné vrstvy byla upravena distribuce OpenSTLinux.