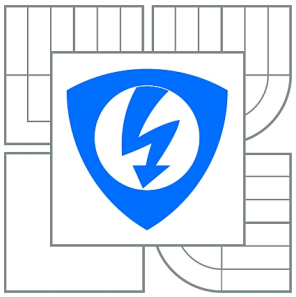
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF CONTROL AND INSTRUMENTATION

FIRMWARE PRO ROBOTICKÉ VOZÍTKO

FIRMWARE FOR THE ROBOTIC VEHICLE

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — Bc. LUKÁŠ OTAVA

AUTHOR

VEDOUCÍ PRÁCE — Ing. PAVEL KUČERA, Ph.D.

SUPERVISOR

BRNO 2013

VYSOKÉ UČENÍ
TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

Ústav automatizace a měřicí techniky

Diplomová práce

magisterský navazující studijní obor
Kybernetika, automatizace a měření

Student: Bc. Lukáš Otava
Ročník: 2

ID: 119558
Akademický rok: 2012/13

NÁZEV TÉMATU:

Firmware pro robotické vozítko

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s HW řešením robotického vozítka a s architekturou Cortex-M3. Pro tuto architekturu vyberte vhodné operační systémy reálného času, tyto otestujte a stanovte jejich parametry. Na základě testů zvolte vhodný operační systém pro robotické vozítko. Pod tímto operačním systémem vytvořte firmware pro jednotlivé HW prvky vozítka a jeho činnost otestujte. Pro firmware vytvořte API a zdokumentujte jej.

DOPORUČENÁ LITERATURA:

Yiu Joseph. The Definitive Guide to the ARM Cortex-M3, Second Edition. Burlington: Newnes, 2009.

Skalický, J., Patočka, M., Feiler, Z.: Elektrické pohony a výkonová elektronika. Brno. Vysoké učení technické v Brně, 2006. s. 1-237. ISBN: 80-214-3286-1.

Termín zadání: 11.2.2013 ***Termín odevzdání:*** 20.5.2013

Vedoucí práce: Ing. Pavel Kučera, Ph.D.
Konzultanti diplomové práce:

doc. Ing. Václav Jirsík, CSc.
předseda oborové rady

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ Fakulta elektrotechniky a komunikačních technologií Ústav automatizace a měřicí techniky

UPOZORNĚNÍ:

ABSTRAKT

Tato práce je zaměřena na vytvoření firmware pro robotické vozítko, založené na architektuře ARM Cortex-M3, který využívá operační systém reálného času. Úvodní část obsahuje informace o dostupných operačních systémech reálného času pro malé vestavné systémy, popis architektury ARM Cortex-M3 a hardwarového řešení řídicího systému. Praktická část se zabývá výběrem a měřením zvolených operačních systémů reálného času. Výsledkem práce je firmware složený z programových modulů, které slouží k ovládání jednotlivých HW prvků. Je vytvořena také ukázková aplikace, která umožňuje na dálku ovládat pohyb robota a ukládat provozní data robota.

KLÍČOVÁ SLOVA

ARM, Cortex M3, LM3S8962, reálný čas, RTOS, FreeRTOS, CooCox, CoOS, uC/OSIII, absolutní časování, přepnutí kontextu, DC motor, akcelerometr, FreeRTOS+TRACE, synchronizace, regulátor, SLIP, diagnostika RTOS

ABSTRACT

This thesis is focused on a firmware for robotic vehicle based on the ARM Cortex-M3 architecture that is running a real-time operating system (RTOS). Theoretical part describes available solutions of embedded RTOS and concrete HW implementation of the robotic vehicle. There is also comparison of the three selected RTOS with their measurements. Result of this thesis is base firmware compounded by a program modules that controls HW parts. There is also a sample PC and firmware application that extends base firmware. This sample application is able to communicate with robotic vehicle, control wheel motion and measure process data.

KEYWORDS

ARM, Cortex M3, real-time, LM3S8962, RTOS, FreeRTOS, CooCox, CoOS, uC/OSIII, absolute timing, context switch, DC motor, accelerometr, FreeRTOS+Trace, synchronization, controller, SLIP, run-time diagnostic

OTAVA, Lukáš. *Firmware pro robotické vozítko*: diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky, 2013. 69 s. Vedoucí práce byl Ing. Pavel Kučera, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Firmware pro robotické vozítko“ jsem vypracoval samostatně pod vedením vedoucího diplomové 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 diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

Brno ....

(podpis autora)

PODĚKOVÁNÍ

Děkuji vedoucímu diplomové práce Ing. Pavlu Kučerovi Ph.D. za odbornou pomoc, kritické připomínky a cenné rady při zpracování mé diplomové práce. Děkuji také mé rodině a blízkým přátelům za podporu během celého studia i psaní této diplomové práce.

Brno ....

(podpis autora)

OBSAH

1 ÚVOD

Společně se stoupajícím výkonem současných mikrokontrolérů a jejich klesající cenou je kladen daleko větší důraz na firmware (vnitřní SW) zařízení. S trendem integrace více subsystémů do jednoho relativně výkonného mikrokontroléru markantně narůstá potřeba správného využití prostředků mikrokontroléru, a to jak v prostoru (paměť), tak zejména v čase. Výkon mikrokontroléru je třeba vhodně rozdělit, aby bylo dosaženo vykonání všech úloh zdánlivě současně. Na vykonávání těchto úloh jsou často kladeny požadavky ohledně jejich včasného provedení. To se týká i reakce na vnější nebo vnitřní události. Splnění těchto časově deterministických požadavků - společně se zachováním přehlednosti celé aplikace - je usnadněno operačními systémy reálného času.

V teoretické části práce je uvedeno, jakým způsobem se liší jednotlivé operační systémy, jak mohou být hodnoceny jejich parametry a vlastnosti. Dále jsou uvedena různá vývojová prostředí, která se zaměřují na architekturu ARM Cortex-M3 a vývoj v jazyce C nebo C++, které jsou v této oblasti standardem. Součástí práce je i stručný pohled na architekturu ARM Cortex-M3 a popis dodané HW platformy, jednotlivých částí a podvozku. Dále jsou uvedeny také teoretické informace z oblasti regulace stejnosměrného komutátorového motoru. Je zde popsán kinematický model podvozku a také model pohonu kol, který je porovnán s identifikovaným modelem z naměřených dat.

Praktická část práce se zabývá prověřením několika RTOS, vývojovým prostředím a následujícími měřeními několika jejich parametrů.

Jádrem práce je vytvoření firmware na bázi RTOS, který bude základním kamenem pro další projekty. Tento firmware bude sloužit k regulaci rychlosti podvozku, ke sběru dat z akcelerometru a ke komunikaci pomocí sběrnice CAN. Vytvořená ukázková aplikace založená na tomto firmware bude realizovat otestování HW platformy. Část práce je také věnována diagnostice běžícího systému.

2 TEORETICKÝ ROZBOR

2.1 Operační systém reálného času

Operační systém (zkráceně OS) je software, který se skládá z jádra a systémových knihoven. Základní funkce jádra (anglicky *kernel*) operačního systému je zprostředkovat programátorovi prostředí pro správu sdílených zdrojů řídicího systému (počítače) a možnost běhu jednotlivých úloh (programů) využívajících tyto zdroje.

Jednou z důležitých vlastností operačního systému je podpora paralelismu, neboli umožnění běhu více úloh najednou, během daného intervalu. Tyto operační systémy jsou také nazývány jako víceúlohové nebo také z angličtiny *multitasking* systémy. U řídicích systémů s pouze jedním procesorem se více běžících úloh jeví tak, jako by běžely skutečně naráz, přestože běží v jeden časový okamžik pouze jedna úloha a úlohy se přepínají.

U jednoprocesorových systémů se používají nejčastěji dvě podoby multitaskingu z hlediska toho, kdy je možné uskutečnit přepnutí úlohy. U kooperativního multitaskingu, je možné úlohu přepnout, pokud dostane jádro řízení od právě běžící úlohy – zhroucení jedné úlohy – způsobí často zhroucení celého systému. U druhé metody, preemptivního multitaskingu, přepnutí úlohy zajišťuje kód jádra, který je spouštěn periodicky a nezávisle na aktuálně běžící úloze. Rozhodování, která úloha bude následovat, zajišťuje plánovač jádra.[7]

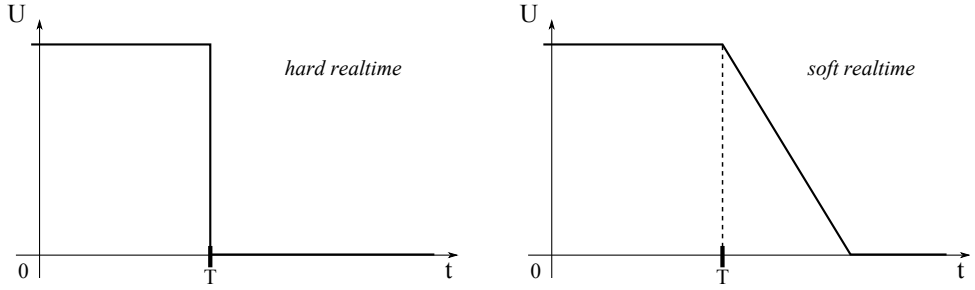
Mezi používané plánovací algoritmy patří prioritní plánování a Round-Robin algoritmus. U prioritního plánování se vykoná dříve ta úloha, která má vyšší prioritu. U algoritmu Round-Robin, označovaného také jako *time-slicing* (krájení času), dochází k pravidelnému přepínání úloh na předem stanovený čas, který je pro všechny úlohy stejný (*time quanta*). U preemptivních algoritmů může dojít k přepnutí v jakékoli části zpracovávané úlohy. Proto je kritickým parametrem preemptivních OS doba přepnutí kontextu (anglicky *context switch*), kdy musí dojít k uložení pracovních registrů a přepnutí zásobníku. S tím také souvisí jejich režie a paměťová náročnost, která je značně větší než u OS s kooperativním multitaskingem.[8]

Rozčlenění řídicí aplikace na více úloh přináší znatelný posun v komfortu tvorby složitějších systémů. Systém je možné přehledně rozčlenit a snadněji ladit části aplikace. Je také možné zajistit, aby byl systém imunní vůči chybě jedné úlohy, kdy nedojde k pádu celé aplikace, ale pouze k pozastavení chybné úlohy a přechodu systému do bezpečného chybového stavu. Tímto mechanismem je možné dosáhnout systému, který je odolný proti poruše (fault - tolerant system) na softwarové úrovni.

Operační systémy reálného času (dále jen RTOS) jsou takové operační systémy, které jsou schopny reagovat na externí a interní události v časově deterministic-

kých úsecích a přitom provádět další řídicí funkce. Maximální časový úsek reakce je často označován anglickým termínem *deadline*, který se běžně používá i v české terminologii. Preeemptivní prioritní multitasking je jednou z nutných podmínek pro umožnění realtime odezvy. V RTOS se proto používá prioritní plánovací algoritmus nebo případně prioritní plánovací algoritmus rozšířený o Round-Robin algoritmus pro úlohy se stejnou prioritou.[7]

Operační systémy reálného času se dále dělí na *soft realtime* a *hard realtime* systémy, podle závažnosti splnění jejich deadlines. To je možné vyjádřit pomocí funkce užitečnosti na obrázku 2.1. Příkladem hard realtime systému může být protiraketový obranný systém. U tohoto systému, zpožděná reakce na letící raketu je bezvýznamná – střelu nemůžeme zneškodnit, pokud systém nezareaguje v daný čas. Příkladem soft-realtime systému je DVD přehrávač. U něho je nutné, aby byl obraz synchronní se zvukem, pokud ale dojde vlivem vnějšího signálu (vyvolání nabídky na obrazovce pomocí dálkového ovladače) k chvilkovému rozsynchronizování nebo zpoždění, divák to zaznamená, ovšem na informační hodnotu tato chyba nemá vliv.[8]



Obr. 2.1: Funkce užitečnosti hard a soft realtime systému

Nedílnou součástí operačních systémů jsou také prostředky pro komunikaci mezi úlohami. Zpravidla se jedná o prostředky pro zabezpečení sdíleného zdroje (Mutual Exclusion (Mutex), Semafor) a prostředky pro předávání dat mezi úlohami (Fronta, Schránka, Příznak).[8]

2.1.1 Klíčové vlastnosti

Mezi základní vlastnosti obecných RTOS patří krátká reakce na přerušení, preemptivní plánování umožňující zaručení realtime odezvy, případně prioritní Round-Robin plánovací algoritmus.

Operační systémy reálného času pro „malé“ *embedded* (vestavěné) systémy mají i další vlastnosti, které jsou odvozeny od použitého HW, na kterém běží (malá paměť RAM ∼10 kB, malá paměť ROM ∼100 kB, frekvence jádra ∼10 MHz), které je omezují. Tyto operační systémy hojně využívají podmíněný překlad zajišťovaný pomocí

maker (direktiv preprocesoru překladače). Dosahují tím různé velikosti kódu jádra přesně podle požadavků, tato vlastnost je označována jako *scalability* (škálovatelnost). Nastavení jsou často shrnuta v hlavičkovém konfiguračním souboru RTOS. Malé RTOS dále umožňují čistě statický překlad, kdy je využívána pouze statická alokace paměti, žádná paměť se nealokuje dynamicky. To může za určitých podmínek zjednodušit návrh systému z hlediska bezpečnosti a ladění chyb. Některé malé RTOS zahrnují také možnost předefinování funkcí, které jsou volány při určitých událostech uvnitř operačního systému, označované jako *hooks* (záchytné funkce). Jedná se např. o funkce, které jsou volány se systémovým časovačem, s přepnutím na úlohu běžící na pozadí, při manipulaci s úlohou, při výpočtu statistik. Tohoto je možné využít k optimalizaci běhu systému, např. pro využití režimů snížené spotřeby, které si uživatel implementuje sám, podle požadavků aplikace.

Důležitým aspektem malých RTOS je také počet architektur mikroprocesorů, na kterých je možné daný RTOS provozovat, neboli jejich portovatelnost. Toho je dosahováno částí nízkoúrovňového kódu, který využívá i assembler, který je specifický pro různé architektury. Při přechodu na jinou architekturu mikroprocesoru se API operačního systému nemění, dojde pouze k modifikaci souborů specifických pro architekturu procesoru.

Některé RTOS obsahují část kódu, pomocí kterého je možné získávat informace o běhu jádra ve formě záznamu událostí, případně ve formě statistik. Tím může být usnadněno ladění celé aplikace.

2.1.2 Dostupná řešení RTOS

Za pomoci internetových vyhledávačů a poznámek výrobců mikrokontrolérů Cortex-M3 byl vypracován malý přehled operačních systému reálného času, který je reprezentován tabulkou 2.1. Tento přehled zahrnuje systémy, jak pouze komerční, tak systémy zbavené poplatků za použití, případně i s otevřeným zdrojovým kódem (opensource). Z licenčního hlediska má pro studijní účely velice zajímavou politiku společnost Micrium – pro studijní účely je jejich Micrium μC/OSIII možné používat s akademickou licencí.

Dokumentace k RTOS jsou dodávány různými způsoby. U menších projektů zdarma je to zpravidla dokumentace dostupná na webových stránkách (FreeRTOS, eCOS, ChibiOS, BeRTOS) nebo formou elektronických manuálů (CoOS, SYS/BIOS). U komerčních produktů je dokumentace často formou tištěné knihy (FreeRTOS, μC/OSIII). Dokumentace by měla obsahovat minimálně popis implementace, referenční manuál, ukázkové příklady použití a možnosti nastavení vlastností RTOS.

Důležitým aspektem výběru RTOS je také forma uživatelské podpory. U placených produktů je podpora samozřejmostí. K bezplatným řešením je často poskyto-

vána prémiová podpora, která je placená (BeRTOS, FreeRTOS, eCOS).

Pokud by se při vývoji mělo hledět na budoucí použití na odlišném HW, je vhodné vybrat systémy s podporou více architektur a podporou pro HAL vrstvu, která obecně popisuje HW.

Zajímavým doplňkem k RTOS jsou aplikace, které umožňují nějakým způsobem nahlížet na běh jádra, získávat statistiky a informace o jednotlivých objektech. Takové možnosti mají FreeRTOS (Trace View) a μC/OSIII (Kernel Probe).

Hlouběji byly otestovány CooOS, FreeRTOS a μC/OSIII. Popis a jejich srovnání je součástí testování jejich parametrů a vlastností v části 3 tohoto textu.

2.1.3 Vývojové nástroje

Kvalita vývojového prostředí se silně projevuje na množství námahy vynaložené na vývoj produktu. Výsledek dále určuje i cena za licenci. Pro mikrokontroléry ARM je na trhu poměrně velké množství vývojových nástrojů, ať už jen toolchainů (sad vývojových nástrojů), nebo kompletních IDE s komerčními licencemi s bezplatnou licencí nebo pod GPL licencí.

Byly uvažovány nástroje pro OS MS Windows. Pro ostatní OS nástroje využívají zpravidla překladače GCC a další nástroje GNU společné s různými IDE, a to i komerčními (CodeBench, CrossWorks).

Přehled dostupných produktů pro vývoj

Všechny tyto produkty obsahují základní toolchain: kompilátor, linker, debugger a standardní knihovny.

- **ARM Workbench IDE** (ARM)
  - licence: komerční
  - IDE založeno na Eclipse
  - součástí je kompilátor přímo od ARM
- **Code Composer Studio** (Texas Instruments)
  - licence: komerční, evaluation 30 dnů
  - ladění HW: TI JTAG adaptéry
  - IDE součástí
  - kompletní relativně složité řešení pro produkty TI
- **CoIDE** (CooCox)
  - licence: bezplatná
  - ladění HW: množství JTAG adaptérů
  - IDE založeno na Eclipse
  - pouze Cortex-M, jednoduše a prakticky uzpůsobené Eclipse IDE, kompilátor Sourcery nebo GCC, online repositář knihoven ostatních uživatelů

název	AVIX-RT	BeRTOS	CooOS	ChibiOS	eCos	FreeRTOS	μC/OSIII	SYS/BIOS
poskytovatel	AVIX-RT	BERTOS	CooCox	ChibiOS/RT	eCosCentric Ltd.	Real Time Engineers Ltd.	Mircium Inc.	Texas Instruments Inc.
licence	komerční	opensource	opensource	opensource	opensource	opensource, komerční SafeRTOS	komerční, pro studijní účely zdarma	zdrama pro součástky TI
platformy	Cortex-M3, PIC	8 bit, ARM7, Cortex-M3	Cortex-M3	8-bit, Cortex-M3	14 architektur	31 architektur	15 architektur	msp430, ARM 9, DSP, Cortex A8, Cortex M3
HAL	ne	ano	formou CoX	ano, bez podpory LM3S	ano, bez podpory LM3S	ne	BSP	ne
dokumentace a podpora	komerční	web, prime support	na webu,	web	web, komerční	Web, komerční formou knihy	komerční	web TI
vývojové prostředí	IAR, KEIL	GCC, code wizzard, BeRTOS SDK	CooIDE, ImageCraft, GCC	GCC, IAR	GCC	GCC, IAR, Rowley, W32 sim	IAR	GCC, Code Composer Studio
ladící nástroje	RTOS Viewer plugin	debug	debug	debug	debug	Trace View	Kernel Probe	debug

Tab. 2.1: Tabulka vybraných dostupných RTOS

- **CrossWorks for ARM** (Rowley)
  - licence: komerční, evaluation 30 dnů
  - ladění HW: množství JTAG adaptérů
  - IDE vlastní, součástí (umožňuje stahování online obsahu)
- **Embedded wokbench for ARM** (IAR Systems)
  - licence: komerční, evaluation 30 dnů, kickstar 32 kB
  - ladění HW: množství JTAG adaptérů
  - IDE vlastní, je součástí produktu
  - podpora pro komerční produkty formou plug-inů (RTOS, knihovny)
- **MDK-ARM Microcontroller Development Kit** (Keil)
  - licence: komerční, lite verze 32 kB
  - ladění HW: množství JTAG adaptérů
  - IDE vlastní, μVision4
- **Sourcery CodeBench** (Mentor Graphics)
  - licence: komerční, Lite GNU/GPL
  - ladění HW: využívá GDB servery JTAG adaptérů, OpenOCD
  - IDE: lite verze jen command-line GCC, komerční Sourcery CodeBench
  - založeno na GCC kompilátoru, upraveno a předkompilováno pro Win
- **WinARM, GNUARM, YAGARTO, devKitPro**
  - licence: GNU/GPL
  - ladění HW: využívá GDB servery JTAG adaptérů, OpenOCD
  - makefile nebo IDE Eclipse IDE+ARM plugin, Code::Blocks, atd.
  - kombinace binutils+GCC+linker+debugger+newlib založeno na MinGW nebo Cygwin

2.2 Jádro ARM Cortex M3

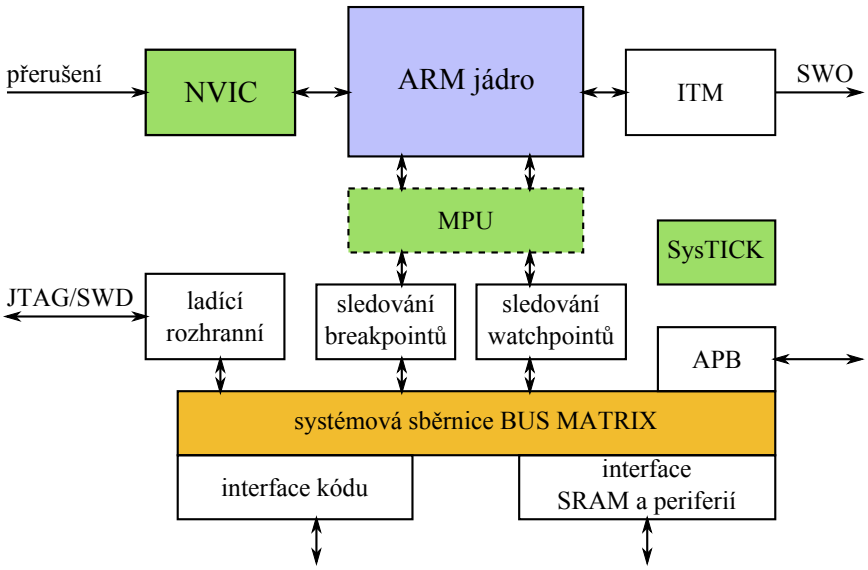
Firma ARM vznikla v Británii jako výrobce kompletních mikroprocesorů v roce 1984 jako **A**dvanced **R**ISC **M**achine Limited. Už první mikroprocesory měly šířku adresní sběrnice 32 bitů a dodržovaly charakteristické rysy RISC. Později ale přešla pouze k vývoji jader mikroprocesorů. Tímto chytrým krokem, kdy schéma jádra mikroprocesoru je vlastnictvím firmy ARM a výrobci platí pouze licenci za použití tohoto jádra, nalezneme mikrokontroléry s logem ARM na pouzdrech čipů mnoha světových výrobců ve velké škále vestavěných a přenosných zařízení[1].[4]

Jádro Cortex-M3 je vystavěno na Harvardské architektuře, která má oddělenou sběrnici pro data a pro instrukce. To umožňuje rychlejší vykonávání programového kódu, kdy je možné paralelně načítat novou instrukci a zároveň ukládat data

[1]Pro přehled, bylo prodáno již 800 licencí, více než 250 společnostem[1].

výsledku do paměti. Toho se využívá při *pipeliningu* (zřetězování instrukcí nebo průtokové zpracování instrukcí[5]), který má 3 fáze - načtení instrukce, dekódování instrukce a vykonání instrukce. Zároveň se také odhaduje, zda může aktuálně vykonávaná instrukce větvení ovlivnit aktuálně načítanou a dekódovanou instrukci. Mechanismus pipeliningu tím umožňuje mnohonásobné zvětšení výkonu vzhledem k hodinovému kmitočtu jádra mikrokontroléru.[3]

Jádro Cortex-M3 podporuje standardní 32 bitovou instrukční sadu ARM, ale také novou instrukční sadu Thumb-2, která vychází z instrukční sady Thumb, ale vylepšuje ji, co do efektivity výsledného kódu. Instrukční sada Thumb-2 umožňuje míchání instrukcí s délkou 16 b a 32 b a tyto instrukce nemusejí být zarovnány na 32 b adresy.



Obr. 2.2: Blokové schéma jádra Cortex-M3 [18]

Blokové schéma jádra Cortex-M3 je uvedeno na obrázku 2.2. Blok APB představuje most pro sběrnici integrovaných periferií.

Jádro mikrokontroléru umožňuje ladění několika způsoby. Je možné využít ladění pomocí standardního rozhraní JTAG a pomocí SWD JTAG rozhraní se sníženým počtem vyžadovaných komunikačních vodičů. Jednotka ladícího rozhraní využívá bloky pro sledování breakpointů a watchpointů (sledování proměnných). Pomocí jednotky ITM (Instrumentation Trace Macrocell) mohou být po ladícím rozhraní odesílány znakové zprávy. Na tuto jednotku můžeme přesměrovat např. standardní výstup.[18]

Jádro může obsahovat i jednotku pro ochranu paměti označovanou jako MPU. Tato jednotka rozděluje paměť až na 8 regionů, které jsou popsány umístěním, ve-

likostí a právy přístupu k paměti. Jednotlivé regiony se mohou překrývat. Mimo těchto 8 regionů existuje také tzv. *background* region, který je přístupný pouze v privilegovaném režimu. Pokud se program snaží přistupovat do zakázaných oblastí, je vygenerováno *exception* (výjimka) - přerušení Memory Management Fault.[18]

Na druhou stranu, tato jednotka bohužel není příliš často v RTOS využívána, její funkcionalita je řešena programovým kódem. Z uvedených dostupných řešení využívá MPU například port systému FreeRTOS (`ARM_CM3_MPU`). API FreeRTOS obsahuje další funkce specifické pro MPU jednotku a to: `xTaskCreateRestricted()`, `vTaskAllocateMPURegions()` a `portSWITCH_TO_USER_MODE()`.[10]

Jádro mikroprocesoru může pracovat ve dvou režimech privilegií. V neprivilegovaném režimu není možné přistupovat do registrů systémového časovače, jednotky přerušení, ani registrů jádra, pomocí MPU je možné omezit přístup do paměti a periferií. V privilegovaném režimu je možné přistupovat všude.

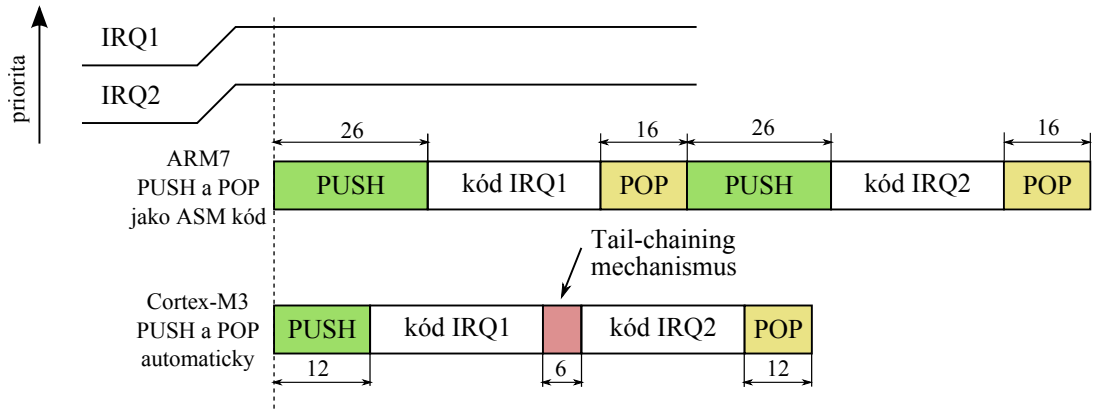
Při vykonávání programu může být jádro ve dvou režimech. V režimu Thread mode, ve kterém je procesor po resetu, je vykonáván kód aplikace. V Handler mode je procesor ve fázi vykonávání obsluhy přerušení (exception). Návratem z přerušení se procesor dostává zpět do Thread mode.[18]

Obsluhu přerušení zabezpečuje jednotka NVIC (Nested Vector Interrupt Controller), která je označovaná jako rychlá a deterministická, díky reakci během 12 cyklů jádra. Jednotka NVIC podporuje 36 přerušení, které mohou být reakcí na úroveň nebo na hranu. Těmto přerušením je možné přiřadit 8 různých priorit (pomocí 3 nejvyšších bitů registru priority), a tím je rozdělit do několika podskupin. Tyto priority je možné měnit za běhu aplikace. Tato jednotka má extrémně krátkou dobu reakce na přerušení, které dopomáhá automatické uložení pracovních registrů do zásobníku (označované jako Hardware Interrupt Handling) a také funkce Tail-chaning (neboli navazování přerušení), jejíž funkce je naznačena na obrázku 2.3.

NVIC obsahuje také systémové registry (obsahující např. stavové slovo) a registry pro použití dvou režimů snížené spotřeby nebo jednoho režimu spánku.

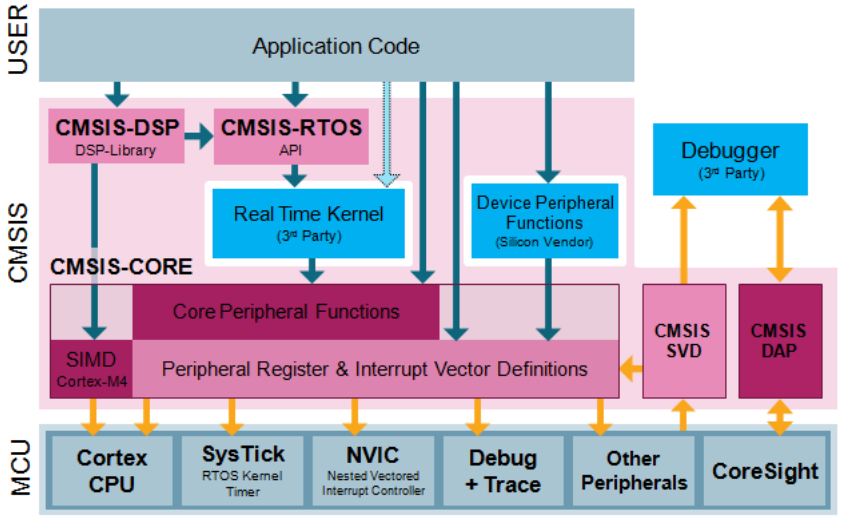
Jádro Cortex-M3 dále obsahuje jednoduchý 24-bitový časovač, označovaný jako systémový (SysTick), s předurčením k časování řídicí aplikace, případně plánovače operačního systému. Tento časovač počítá dolů, generuje přerušení při přetečení a má funkci *autoreload* (automatické přednastavení).

Zajímavou vlastností jádra Cortex-M3 a Thumb2 instrukční sady je podpora atomických operací. Jsou implementovány operace, které jsou běžně realizovány několika instrukcemi, které není možné přerušit tak, že jsou realizovány použitím pouze jedné instrukce, kterou nelze přerušit. Jedná se například o funkcionalitu nazývanou bit-banding – neboli logické operace nad bity pomocí masky během jednoho taktu. Tato funkcionalita je realizována paměťovou oblastí, která tvoří *alias* (zástupná paměťová místa) pro registry GPIO portů.[3]



Obr. 2.3: Porovnání odezvy na přerušení mikrokontroléru s jádrem Cortex M3 s využitím Tail-chaining mechanismu a ARM7 (čas v hodinových cyklech jádra) [3]

Jako softwarovou podporu poskytuje firma ARM knihovnu CMSIS . Hierarchie knihovny verze dostupné v druhé polovině roku 2012 je na obrázku 2.4. Tato verze již obsahuje CMSIS-DSP knihovnu s rychlými algoritmy pro zpracování signálů. CMSIS-RTOS je nově definované API pro výrobce RTOS.[1]

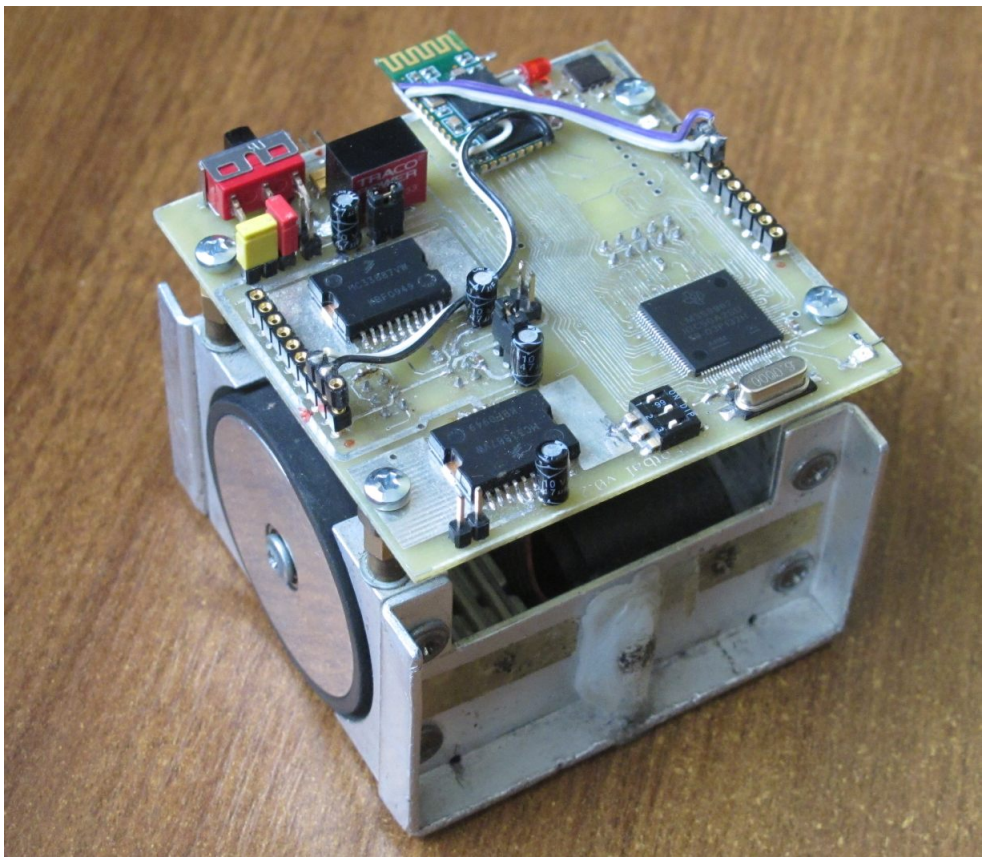


Obr. 2.4: Hierarchie knihovny CMSIS (převzato z [1])

2.3 Deska řídicí jednotky

Dodaný návrh řídicího jednotky robota a již vyrobená neosazená deska plošných spojů (dále jen DPS), byly navrženy před několika lety na Ústavu automatizace

a měřicí techniky VUT v Brně.



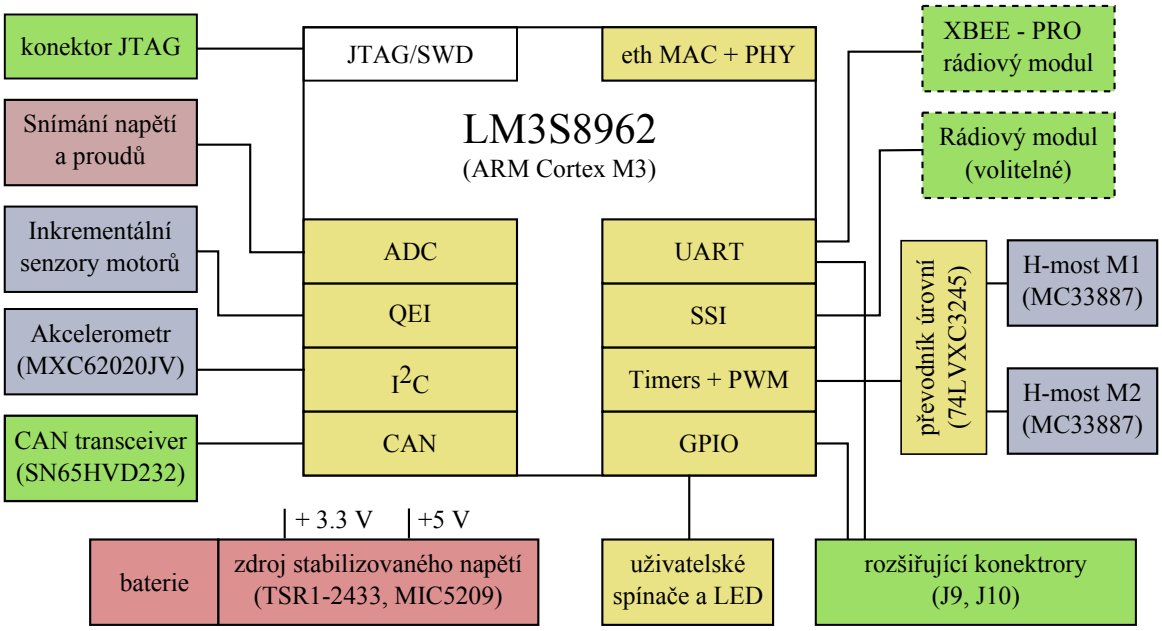
Obr. 2.5: Fotografie podvozku s řídicí deskou a komunikačním modulem

Koncepce řídicí jednotky a její jednotlivé části jsou patrné z blokového schéma zobrazeného na obrázku 2.6. Tyto jednotlivé části jsou zběžně popsány v textu dále.

2.3.1 Mikrokontrolér LM3S8962 Stellaris (Texas Instruments)

Mikrokontroléry řady LM3S původně vyráběla firma Luminary Micro, která byla koupena společností Texas Instruments. Přes nesporné výhody tohoto mikrokontroléru již dnes není výrobcem doporučen do nových návrhů pravděpodobně pro výrobní technologii.

Maximální frekvence jádra mikrokontroléru je 50 MHz. Tuto frekvenci produkuje generátor hodin, který využívá obvodu fázového závěsu (PLL). Jádro mikrokontroléru a interní logika jsou napájeny z vnitřního stabilizátoru 2,5 V s nízkým úbytkem, vstupně-výstupní obvody využívají 3,3 V napájení a logiku. Teplotní rozsah použití mikrokontroléru, pro účel robotického vozítka do vnitřních prostor není příliš důležitý, ale je -40 až 85 °C.



Obr. 2.6: Blokové schéma řídicí desky

Pro naše účely mikrokontrolér obsahuje PWM generátory, jednotku kvadraturních enkodérů, analogově digitální převodník a komunikační kanály (UART, I^2C a CAN).

Zajímavostí mikrokontroléru LM3S je integrace ethernetového PHY přímo na čip. Pro realizaci ethernetového rozhraní stačí připojit magnetikum a konektor. Tato funkcionalita ovšem není využita. Využito je ale integrované rozhraní CAN, vyvedené na rozšiřující konektor.

2.3.2 Obvodové řešení

Napájení celého robota je realizováno pomocí sekundárních článků (akumulátorů) o hodnotě napětí 6 až 9 V. Součástí vstupních obvodů jsou ochranné diody. Proud z baterií je přes hlavní vypínač přiváděn na obvody motorů bez stabilizace a přes stabilizátory na všechny další obvody. Stabilizátor pro napětí 3,3 V je spínaného typu realizovaný hybridním integrovaným obvodem, tím je zajištěna vysoká účinnost této části. Pro napětí 5 V, kde je minimální odběr, byl zvolen běžný lineární stabilizátor.

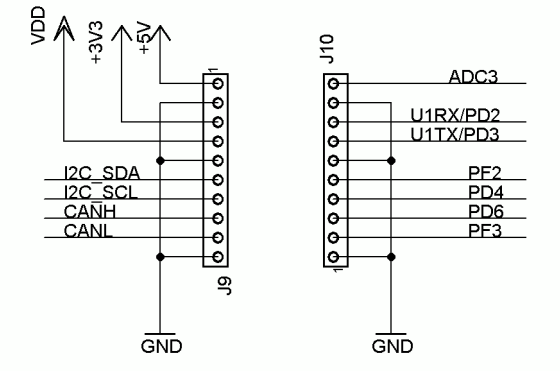
Všechny cesty napájení, kde by mohly vznikat napěťové špičky vlivem impulsního odběru, jsou opatřeny keramickými blokovacími kondenzátory.

Cesty proudů a místa důležitá pro měření napětí (včetně výstupů motorů) jsou vyvedeny na dvoupinové jumper konektory, pro snadné ověřování a testování.

Generátor hodinového kmitočtu mikrokontroléru je taktován krystalem o nominální frekvenci 6 MHz. Kondenzátory okolo krystalu je zajištěna funkce oscilátoru.

Systémové hodiny jsou získány z této základní frekvence pomocí jednotky PLL. Toto řešení umožňuje snadnější realizaci generátoru hodin, vzhledem k problematickému použití krystalů na vyšší frekvence, a zároveň umožňuje volbu taktu systémových hodin pomocí software v rozumném rozsahu frekvencí. Tím je možné měnit výkon i spotřebu jádra za běhu. RESET vývod je přiveden na integrační článek dle katalogového listu a zároveň na resetovací tlačítko.

Deska obsahuje i transceiver rozhraní CAN tvořené obvodem SN65HVD232. Komunikační vodiče jsou vyvedeny na rozšiřující konektor. Rozšiřující konektory dále obsahují vývody I^{2}C sběrnice, UART jednotky, napájení a 4 GPIO vývody.



Obr. 2.7: Zapojení vývodů rozšiřujících konektorů (převzato z [17])

Pro programování a ladění aplikace v mikrokontroléru je deska osazena konektorem JTAG rozhraní.

Volitelné komunikační moduly je možné připojit do připravených konektorů.

Na desce je umístěno několik indikačních LED a 3 DIP spínače připojené na GPIO vývody. Vývody hibernation module ani ethernetu nejsou využity.

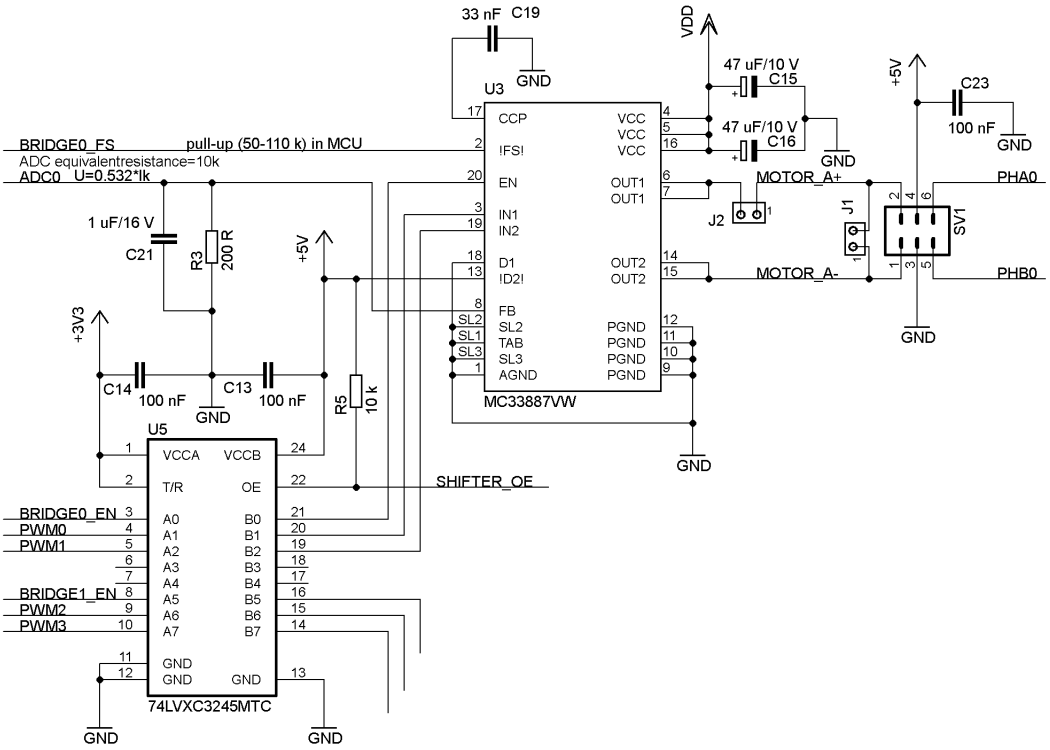
2.3.3 Pohony a navázání na mechanickou část

Pohyb robotu je realizován diferenciálním podvozkem, který je poháněn dvěma malými stejnosměrnými kartáčovými motory s permanentními magnety. Součástí těla motoru je i inkrementální senzor. Použitý motor vyrábí firma FAULHABER. Typové označení je 2224R006SR a mezi jeho parametry patří nominální napětí 6 V, výkon 4,5 W. Motor je s koly spojen převodovkou z ozubených kol. Tímto dochází k přidání chyby v podobě nelinearity v převodech do výstupu soustavy.

K napájení motorů a regulaci jejich výkonu slouží H-most[2] MC33887 vyráběný firmou Freescale. Mezi jeho klíčové vlastnosti patří malá velikost pouzdra, malý

[2]H-most je označení pro výstupní obvod složený z dvou dvojic tranzistorů umožňujících reverzaci výstupního napětí.

odpor výstupních MOSFET tranzistorů v sepnutém stavu, maximální frekvence spínání 10 kHz. Obvod má vyveden výstup úměrný protékajícímu proudu a také výstup ochranného obvodu indikující chybový stav (vlivem zkratu nebo vysoké teploty pouzdra). Jeho vstupy, kterými je možné řídit směr a přivádět řídicí PWM signál, jsou přizpůsobeny na 5 V TTL logiku.[27]



Obr. 2.8: Schéma zapojení připojení H-mostů k mikrokontroléru (převzato z [17])

Pro připojení na 3,3 V logiku slouží oboustranný transceiver 74LVXC3245. Vývodem pro nastavení směru je trvale zvolen směr z mikrokontroléru do H-mostu. Vývod, který přepíná výstup obvodu do stavu vysoké impedance, je vyveden na GPIO vývod procesoru a je také opatřen zdvihacím pull-up rezistorem. Tímto je zajištěno, že mikrokontrolér nemůže ovládat H-most, pokud není ovládací pin korektně inicializován. Je možné také odpojit jedním vývodem ovládání obou H-mostů. PWM výstupy jsou odebírány z mikrokontroléru z jednotky Pulse Width Modulator přes piny PWM0–PWM3. Ostatní řídicí a indikační signály jsou přivedeny na GPIO vývody procesoru.

2.3.4 Senzory

Mezi senzory pro regulaci pohybu robota můžeme zařadit inkrementální senzory, které jsou součástí motorů a dvouosý snímač zrychlení - akcelerometr.

Inkrementální senzory jsou dvoufázové kvadraturní enkodéry pracující na magnetickém principu. Tyto senzory generují digitální signál s 512-ti impulsy na otáčku. Znaménko fáze, o kterou jsou výstupní signály posunuty, odpovídá směru otáčení. Maximální výstupní frekvence impulsů je 160 kHz. Senzory jsou napájeny 5 V a mají odběr typicky 6 mA. Vyhodnocení polohy a rychlosti otáčení kola zajišťuje jednotka mikrokontroléru označená jako QEI, kterou jsou snímače připojeny vývody PHA a PHB.

Akcelerometr typu MXC6202 je levný integrovaný obvod vyráběný firmou MEMSIC, který slouží k měření zrychlení ve dvou kolmých osách v rozsahu $\pm 2g$, určený pro použití ve spotřební elektronice.[19] Tento obvod je vyroben technologií CMOS a neobsahuje žádné pohyblivé části. Snímání zrychlení je založeno na detekci tepelného proudění pomocí skupiny termočlánků. Tyto termočlánky jsou umístěny do čtyř stran od regulovaného zdroje tepla. Tato technologie přináší oproti běžně používané MEMS technologii velkou odolnost oproti extrémním zrychlením. Výhodou tohoto obvodu je přímá integrace všech komponent na čip a digitální výstup formou připojení na dvouvodičovou I^2C sběrnici. Akcelerometr dále obsahuje teplotní senzor, který slouží pro kompenzaci teplotní závislosti snímače, ale jeho výstup můžeme využít v naší aplikaci. K mikrokontroléru je připojen na jednotku I^2C pomocí vývodů I2C0SDA a I2C0SCL.

Měřené analogové veličiny jsou přiváděny na vstupy ACD0–ADC3 jednotky analogově číslicového převodníku ADC mikrokontroléru. Snímání napětí baterie je realizováno odporovým děličem napětí. Maximální vstupní napětí je tedy vzhledem k referenčnímu napětí ADC rovno:

$$U_{MAX} = U_{ref} * \frac{R10 + R11}{R11} = 3,0 * \frac{68 + 150}{68} = 9,617V$$

Proud protékající motory je převáděn na napětí v H-mostech dle vztahu:

$$U_{ADC0} = I_{FB} * R_3 = \frac{1}{375} * R_3 * I_M = \frac{200}{375} * I_M = 0,5\overline{3} * I_M$$

Čtvrtý vstup je vyveden na rozšiřující konektor.

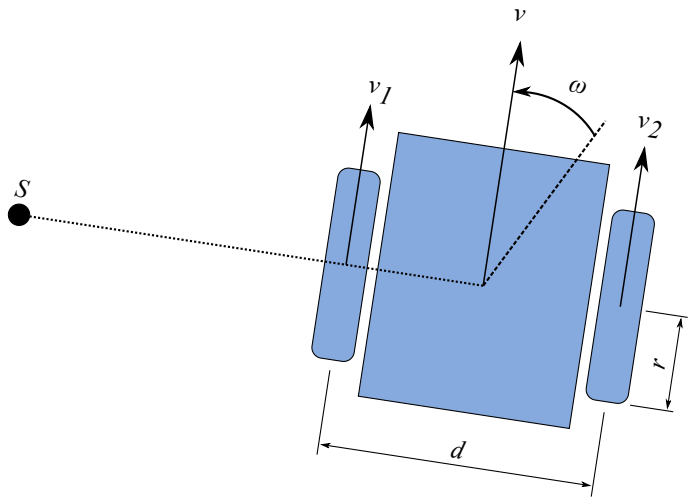
ADC obsahuje i teplotní senzor, kterým je možné kontrolovat teplotu čipu mikrokontroléru a odhalit tím případné chybové stavy.

2.4 Regulace rychlosti diferenciálního podvozku

2.4.1 Diferenciální podvozek

Diferenciální podvozek obsahuje dvě zvlášť hnaná kola. Pokud mají kola stejnou rychlost, robot se pohybuje přímo vpřed. Pokud je rychlost kol opačná, robot se otáčí kolem své osy. Pokud jsou rychlosti kol různé stejného znaménka, robot se pohybuje po kružnici.

Podvozek je konstruovaný pro jízdu na rovném povrchu, bez nerovností.



Obr. 2.9: Nákres pro odvození kinematického modelu diferenciálního podvozku

V robotice se zpravidla používají pro řízení podvozků dvě skalární rychlosti, které tvoří výsledný vektor rychlosti. Jedná se o rychlost dopředu, označovanou jako v, a rychlost kolem osy otáčení, označovanou ω.

Podle nákresu 2.9 můžeme napsat tyto rovnice: [21]

$$v = R\omega \qquad v_1 = (R - \frac{d}{2})\omega \qquad v_2 = (R + \frac{d}{2})\omega \tag{2.1}$$

Kde R je poloměr otáčení.

Vyřešením soustavy rovnic dostaneme:

$$v = \frac{v_1 + v_2}{2} \qquad \omega = \frac{v_1 - v_2}{d} \tag{2.2}$$

Pokud známe změny natočení kol, můžeme určit změnu polohy a natočení robota pomocí následujících diferenciálních rovnic, uvedených v [21]:

$$\begin{aligned} \dot{x} &= v cos\varphi \\ \dot{x} &= v sin\varphi \\ \dot{\varphi} &= \omega \end{aligned} \tag{2.3}$$

Z hlediska geometrických rozměrů byly určeny parametry pro přepočet signálů, se kterými počítá mikrokontrolér na signály v inženýrských jednotkách. Vzdálenost kol byla změřena 65 mm. Obvod kol byl změřen a průměrován s vypočtenou hodnotou z poloměru kola na hodnotu 147,65 mm.

Pokud budeme uvažovat převodovku s převodovým poměrem $n = \frac{100}{12}$, inkrementální senzor s $N = 4 \cdot 512$ pulzy na otáčku kola s obvodem $o = 147,65\,mm$, potom ujetá vzdálenost s_P na jeden pulz enkodéru bude:

$$s_P = \frac{o}{n * N} = \frac{147,65}{\frac{100}{12} \cdot 4 \cdot 512} = 8,7\mu m \tag{2.4}$$

2.4.2 Model a identifikace pohonu podvozku

Náhradní schéma statoru stejnosměrného motoru s permanentními magnety tvoří tři sériově zapojené prvky. Jedná se indukčnost vinutí L_a $[H]$, odpor vinutí R_a $[\Omega]$ a o zdroj zpětně indukovaného napětí. Tento obvod můžeme popsat rovnicí:

$$u_a = R_a i_a + L_a \frac{di_a}{dt} + C\Phi\omega \tag{2.5}$$

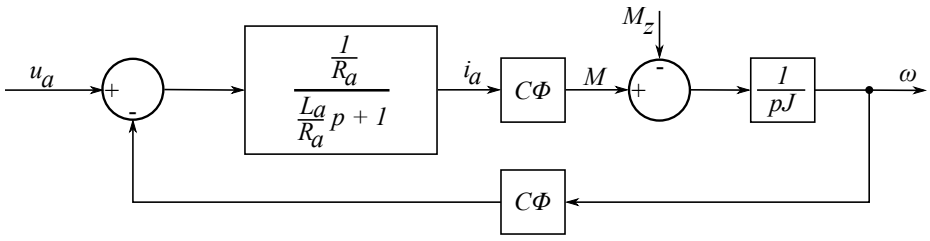
Kde u_i je napětí statoru $[V]$, i_a je proud statorem $[A]$, $C\Phi$ je konstanta motoru $[V.s]$ a ω vyjadřuje otáčky $[rad.s^{-1}]$. Tuto rovnici můžeme pomocí Laplaceovy transformace upravit do tvaru, kdy pro obraz proudu statorem platí:

$$I_a(p) = \frac{\frac{1}{R_a}}{1 + \frac{La}{Ra}p}[U_a(p) - C\Phi\Omega(p)] \tag{2.6}$$

Pro moment na hřídeli motoru platí rovnice:

$$M = C\Phi i_a = J\frac{d\omega}{dt} + M_z \tag{2.7}$$

Kde J je moment na hřídeli motoru $[kg.m^2]$ a M_z je moment zátěže [N.m]. Pomocí těchto rovnic můžeme vytvořit simulační schéma stejnosměrného motoru, které je na obrázku 2.10. [22]



Obr. 2.10: Model stejnosměrného motoru s permanentními magnety

Abychom mohli vytvořit přenos regulované soustavy motoru a kol s podvozkem, který bude následně využit pro nastavení parametrů regulátoru rychlosti, potřebujeme znát celkový moment setrvačnosti na hřídeli motoru. Mezi koly a motorem je převodovka dvou rotačních pohybů. Mezi koly a povrchem (setrvačnost vlastního robota) je převodovka rotačního na translační pohyb. Vzhledem ke konstrukci robota, bude pro celkový moment setrvačnosti platit:

$$J = J_m + J_{Rkolo} + J_{Rpodvozek} \tag{2.8}$$

Kde J_m je moment setrvačnosti motoru, který je uvedený v katalogovém listu motoru [28]. J_{Rkolo} je redukovaný moment setrvačnosti kola, který v sobě zahrnuje i moment setrvačnosti převodovky. Pomocí zákona zachování energie bude redukovaný moment setrvačnosti kola:

$$J_{Rkolo} = \frac{1}{n^2} \cdot 0,5 \cdot m_k \cdot r_k^2 \tag{2.9}$$

Kde n převodový poměr převodovky, m_k je hmotnost kola a r_k je poloměr kola.

$J_{Rpodvozek}$ je redukovaný moment setrvačnosti, který je tvořen hmotností těla celého robota. Ten můžeme získat z rovnice:

$$J_{Rpodvozek} = \frac{1}{n^2} \cdot m_t \cdot \frac{(2 \cdot r_k)^2}{4} \tag{2.10}$$

Kde m_t je polovina hmotnosti těla celého robota.

K modelu motoru je pro návrh regulátoru nutné připojit náhradu za tvarovač (dopravní zpoždění o hodnotě půl periody vzorkování). Model měniče (dopravní zpoždění, o velikosti půl periody PWM signálu) je možné zanedbat.

Po dosazení hodnot získaných z měření rozměrů a hmotností a z katalogových hodnot dostaneme výsledný přenos s obsaženým zesílením měniče a senzoru v Laplaceově transformaci:

$$F_S(p) = \frac{0,934}{(1 + \frac{1}{5}p)(1 + \frac{1}{100}p)(1 + \frac{1}{8,75 \cdot 10^4}p)}$$

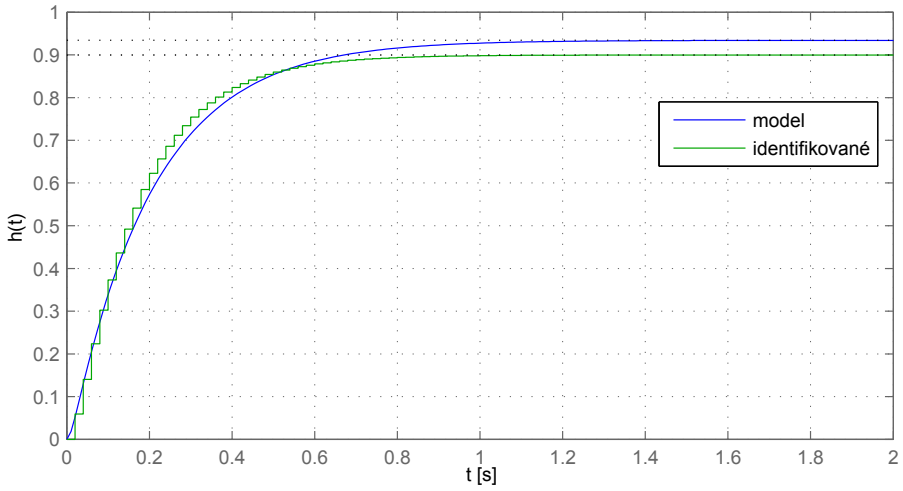
Výpočty dílčích částí přenosu jsou součástí skriptu `drivemodel.m`, který je součástí přílohy CD.

Přenos motoru byl také identifikován z naměřených dat pomocí nástroje Identification Toolbox ze softwarového balíku Matlab, který využívá odhad parametrů modelu pomocí metody nejmenších čtverců. Naměřená data byla vyexportována ukázkovou C# aplikací uvedenou v praktické části této práce. Pro identifikaci byl napsán jednoduchý skript `driveident.m`, který provede identifikaci ARX modelu, vypíše odhadnutý přenos a vykreslí přechodovou charakteristiku. Vzhledem k fyzikálnímu principu systému byl zvolen diskrétní model 2. řádu se zpožděním jeden krok. Vzorkovací frekvence byla 20 ms.

Identifikovaný diskrétní přenos motoru s připojeným měničem a senzoru:

$$F_S(z) = \frac{0.0594z^{-1} + 0.0043z^{-2}}{1 - 1.294z^{-1} + 0.365z^{-2}}$$

Porovnání odezev obou přenosů je v grafu 2.11. Z grafu je patrné, že oba systémy se liší ve statickém zesílení a lehce také v dynamice.



Obr. 2.11: Přechodové charakteristiky modelovaného a identifikovaného systému

2.4.3 Modifikovaný diskrétní PID regulátor a jeho nastavení

Standardním řešením regulace lineárních soustav je použití PID regulátoru, nebo případně jeho zjednodušených verzí. Oproti ostatním typům regulátorů poskytuje standardní PID regulátor výhodu v průhlednosti jeho nastavení. Oddělení jednotlivých složek usnadňuje jeho modifikace pro praktické použití.

Obraz přenosové funkce modifikovaného PID regulátoru v $\mathcal{Z}$ transformaci je následující [24]:

$$U(z) = K_R \left[\beta W(z) - Y(z) + \frac{T_S z^{-1}}{T_I(1 - z^{-1})}(W(z) - Y(z)) - N\frac{1 - z^{-1}}{1 - e^{-\frac{T_S N}{T_D}} z^{-1}} Y(z) \right] \tag{2.11}$$

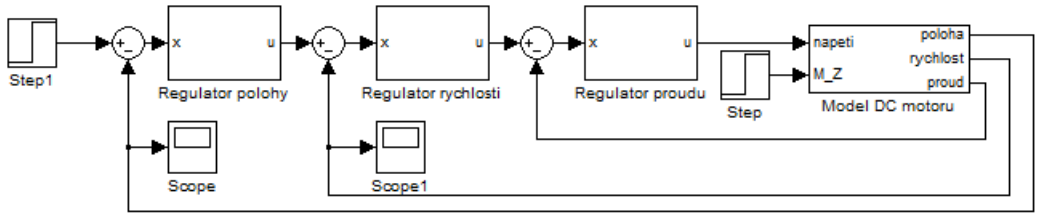
Kde T_s je perioda vzorkování, $U(z)$ je obraz akčního zásahu, $W(z)$ je obraz žádané hodnoty a $Y(z)$ je obraz měřené veličiny.

Filtrace derivační složky je realizována filtrem, který využívá stavovou veličinu derivační složky. Použití filtrace derivační složky je u diskrétního typu regulátoru takřka nutností, abychom se vyhnuli kmitavému průběhu akční veličiny. Konstanta N regulátoru určuje mezní frekvenci filtru vzhledem k časové konstantě derivační složky T_D. V praxi se N volí v intervalu <3;10> podle množství šumu v procesu.[24]

Jak je z přenosové funkce regulátoru patrné, do derivační složky vstupuje pouze měřená veličina a do proporciální složky vstupuje rozdíl měřené veličiny váhovaný koeficientem β a žádané hodnoty. Tímto koeficientem, který se volí v rozmezí <0;1> můžeme omezit překmit v odezvě na skokovou žádanou hodnotu. [25]

Dle charakteru soustavy je přirozené použití pouze PI regulátoru. Parametry pro tento regulátor byly nastaveny modifikací parametrů, které vycházejí hodnot metody Zieglera a Nicholse. Protože se v regulačním pochodu objevovaly zákmity vlivem nelinearit v převodovce, bylo sníženo proporciální zesílení regulátoru na hodnotu $K_R = 0,5$. Integrační časová konstanta byla nastavena dle výpočtu metodou Z-N na hodnotu $T_S/T_I = 0,2$.

Kaskádní regulace pohonů



Obr. 2.12: Simulační schéma kaskádní regulace polohy stejnosměrného motoru

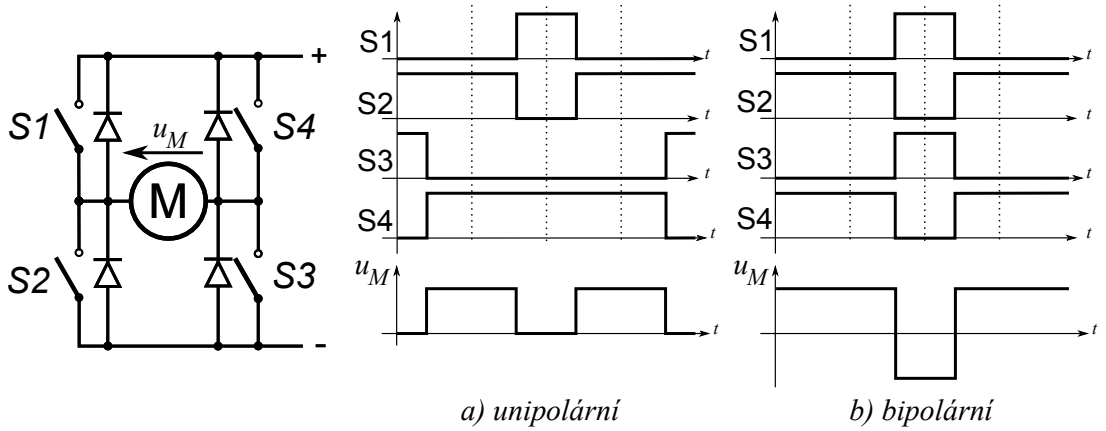
V běžné praxi regulace velkých servopohonů je používána kaskádní regulace, kdy jsou regulovány veličiny: natočení, rychlost a proud. Žádané hodnoty podřízených regulátorů zadávají nadřazené regulátory, jak je uvedeno na schématu 2.12. [22]

Pro regulaci rychlosti by tedy bylo vhodné použít dva regulátory, jeden proudu a jeden rychlosti. Pokud se ale pokusíme navrhnout regulátor proudu, narazíme hned na několik nepříjemností. Hlavním problémem je rozpor elektrické časové konstanty motoru a mnohem nižší rychlosti převodu A/D převodníku. Při vzorkovací periodě 500 μs je nezanedbatelné i vytížení procesoru. Z těchto důvodů není vhodné použít proudový regulátor u takto malého motoru.

2.4.4 Ovládání H-mostu PWM signálem

Jak již bylo uvedeno v úvodní části, motory budou ovládány PWM signálem, jehož střída bude úměrná žádanému momentu. Abychom mohli ovládat směr otáčení motoru pomocí logického signálu, při použití nesouměrného napájecího napětí, je vhodné použít obvod, jehož spínací tranzistory na výstupu tvoří dvojitý pár nazývaný jako H-most. Toto zapojení realizuje čtyř-kvadrantový měnič. Tento měnič

umožňuje oba směry proudu i napětí. Umožňuje tedy vracet energii při brzdění zpět do zdroje. Existuje několik možností jak tyto tranzistory ovládat, jak je uvedeno na obrázku 2.13.



Obr. 2.13: Možnosti ovládání H-mostu PWM signálem

Nákres *a)* ukazuje unipolární řízení, v tomto režimu je během jedné periody řízení na zátěži pouze jedna polarita napětí nebo žádné napětí. Nákresy *b)* je označován jako bipolární řízení, kdy je střídavým přepínáním obou dvojic na zátěž během jedné periody připojováno napětí obou polarit. Tím je dosaženo dvojnásobné změny napětí na motoru. Zvlnění proudu zátěží je dvojnásobné oproti unipolárnímu řízení. Použití unipolárního řízení je výhodné z hlediska nižšího akustického hluku vlivem dvojnásobné frekvence proudu a napětí zátěže.[22]

V konkrétním zapojení mikrokontroléru a H-mostu je možné použít obě možnosti. Z hlediska bateriového napájení bylo užito unipolární řízení, kdy je ovládána vždy jen polovina H-mostu. Podle výběru poloviny H-mostu, která je ovládána, je zvolen směr otáčení motoru.

3 VÝBĚR RTOS A VÝVOJOVÉHO PROSTŘEDÍ

Z tabulky 2.1 byly vybrány tři RTOS pro hlubší rozbor, dle několika kritérií. Prvním kritériem byla dostupnost RTOS. Aby byly systémy porovnatelné, bylo zvoleno kritérium prioritního preemptivního multitaskingu s možností Round-Robin algoritmu pro úlohy stejné priority. Posledním kritériem výběru byla podpora objektů pro mezi-procesní komunikaci.

3.1 CoOS

Prvním testovaným systémem byl CoOS, byl zvolen pro jeho jednoduchost a vlastnost, že je na míru pro jádra Cortex-M. Jako vývojové prostředí bylo zvoleno CoIDE (kap. 2.1.3), které bez problémů pracuje s ladícím JTAG rozhraním CoLinkEX, které bylo dodáno společně s HW. Jako kompilátor byl vyzkoušen CodeSourcery(v 4.5.2) i GNU ARM (v 4.6.0) a oba pracovaly bez problému.

Založení projektu v CoIDE je snadné a sestává z několika kroků: volba adresáře, volba výrobce a typu mikrokontroléru a poslední volba SW modulů, které se mají k projektu přidat. Pro nastavení kompilace slouží karta *Configuration*. V ní můžeme nastavovat jednotlivé parametry pomocí editačních polí na kartě Overview, nebo můžeme přímo editovat skripty s nastavením ve formě souborů.

Zaškrtnutím CoOS na kartě Repository - Components dojde k přidání CoOS do projektu, k přidání složek se zdrojovými a hlavičkovými soubory dojde automaticky, obsluhy přerušení pro RTOS (SysTick_Handler a PendSV_Handler) jsou nastaveny automaticky. Nastavení CoOS RTOS probíhá v hlavičkovém souboru `OSConfig.h`. Zde je nutné uzpůsobit frekvenci hodin mikrokontroléru a žádanou frekvenci systémového časovače SysTick. Je zde možné pomocí maker povolit nebo zakázat jednotlivé části RTOS. Inicializace CoOS, před kterou je vhodné nastavit korektně zdroj hodin pro mikrokontrolér, probíhá voláním funkce `CoInitOS()`. Spuštění RTOS probíhá pomocí funkce `CoStartOS()`.

Ladění probíhá snadno připojením ladícího rozhraní CoLinkEx, umístěním breakpointů, nahráním aplikace, startem ladění (tlačítko Debug), až po krokování programu. Při programování testovacích aplikací nebyly zaznamenány žádné závažné problémy, až na kolizi jmen souborů RTOS a knihoven mikrokontroléru.

K CoOS je dodávána bezplatná dokumentace, ve formě referenčního manuálu s příklady, na webu CooCox. Dokumentace je sice strohá, ale přehledná a srozumitelná.

3.2 FreeRTOS

FreeRTOS byl zvolen pro jeho popularitu, cenu, množství architektur a také možnost případného navýšení na komerční verzi. Pro testování FreeRTOS bylo zvoleno také prostředí CoIDE (kap. 2.1.3), pro jeho bezproblémovost, snadné ladění a cenu. Založení projektu počíná dialogem jako v případě CoOS.

Začlenění FreeRTOS se skládá z několika kroků. Nejprve je nutné do projektu přidat zdrojové soubory jádra FreeRTOS a k nim příslušné include (složky s hlavičkovými soubory), poté je nutné přidat kód a hlavičkové soubory modulu správy paměti (byl zvolen modul `heap_2.c`) a nakonec modul obsahující kód závislý na platformě mikrokontroléru. Jako poslední krok je nutné zvolit správná jména funkcí obsluhy přerušení (SysTick_Handler, SVC_Handler a PendSV_Handler) například pomocí maker v konfiguračním souboru. Nastavení FreeRTOS probíhá v konfiguračním hlavičkovém souboru `FreeRTOSConfig.h`, který je nutné umístit do složky s projektem. Podrobnějšímu pohledu na FreeRTOS je vyčleněna samostatná kapitola v další části práce.

Dokumentace k systému FreeRTOS je dostupná na webových stránkách, podrobný popis API funkcí je také součástí zdrojových kódů. Pro vyšší komfort je možné zakoupit oficiální tištěnou literaturu k tomuto systému.

3.3 μC/OSIII

μC/OSIII byl zvolen pro jeho popularitu, výhodnou licenční politiku pro studijní účely (profesionální produkt s akademickou licencí) a také kvalitní dokumentaci.

Nejprve byla upravena ukázková aplikaci v prostředí IAR Embedded Workbench (kap. 2.1.3). Protože ladění přes CoLinkEx ladícího rozhraní nebylo možné, byl použit port μC/OSIII pro procesory od STM a překladač GCC, potom mohl být μC/OSIII přeložen i v prostředí CoIDE.

Založení projektu probíhá jako v předchozích OS. Začlenění μC/OSIII do projektu spočívá v přidání zdrojových a hlavičkových souborů částí μC/OSIII, μC/LIB a μC/CPU do projektu, a to i s platformě závislým kódem. Všechny tyto části jsou nutné k běhu systému. Množství knihoven se také podílí na velikosti výsledného kódu. Nakonec je nutné správně nastavit funkce volané při přerušení. Poměrně komplikované nastavení μC/OSIII probíhá v souborech `os_cfg.h` a `app_cfg.h` .

Dokumentace k μC/OSIII je dostupná jako kniha v tištěné nebo elektronické podobě. Kniha v první části podrobně popisuje jednotlivé mechanismy použité v tomto RTOS. Poté popisuje jejich implementaci a využití. Součástí je také přehled API funkcí. Druhá část knihy je tvořena popisem kódu specifického pro danou architekturu mikroprocesoru.

4 TESTOVÁNÍ RTOS

Pro porovnání kvality implementace jádra jednotlivých RTOS byla provedena měření doby přepnutí kontextu a měření periody SW časovače společné s analýzou jejich rozptylu. Tato měření jsou společně s výsledky popsána v následujících podkapitolách.

Měření bylo provedeno pro CoOS, FreeRTOS, μC/OSIII. Byl použit stejný kompilátor (GCC 4.5.2) s nastavenou optimalizací O2. Instrukční soubor byl Thumb-2. U všech RTOS byla nastavena konfigurace jádra na použití *Preemptive Priority Round-Robin* plánovacího algoritmu a neuplatněné části RTOS byly zakázány definicemi maker.

4.1 Měření, přenášení a zpracování hodnot

Doba byla měřena pomocí časovače TIMER0 ve funkci čítače hran vstupního obdélníkového signálu na GPIO pinu z generátoru Agilent 33120. Maximální frekvence vstupního signálu pro detekci hran periferie TIMER0 je čtvrtinová oproti frekvenci jádra. Při frekvenci jádra 40 MHz byla použita frekvence maximálně 9 MHz. Funkce pro měření časového intervalu byly sdruženy do knihovny tvořené soubory `test.c` a `test.h`.

Data byla přenášena přes UART1 jako ASCII text, přes sériový terminál byla uložena a poté naimportována do MATLABU, kde byly vypočítány statistické údaje a vykresleny histogramy. Knihovna pro výpis přes UART1 je tvořena soubory `serial.c` a `serial.h` a je pojata velmi minimalisticky, obsahuje pouze funkce pro přenos řetězců a bez znaménkového 16b integeru. Pro měření pomocí AD převodníku jsou využity funkce z knihovny periferií *Stellaris DriverLib*, je uplatněno dotazování na dokončení převodu.

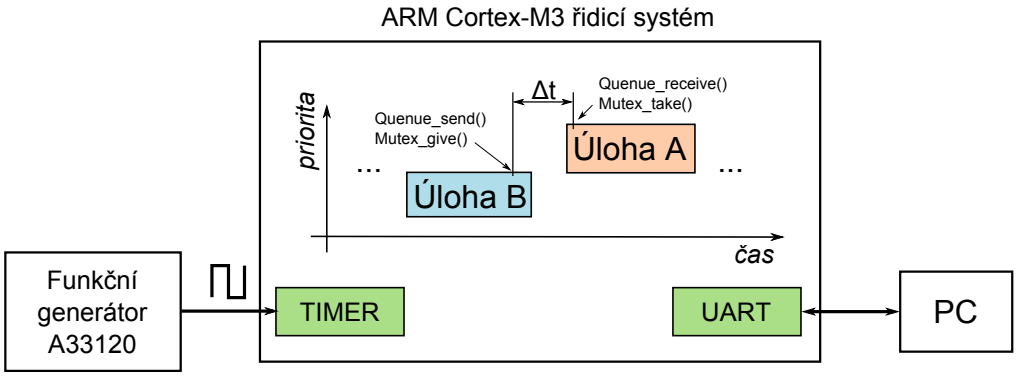
Pro kontrolu správnosti údajů měření časovače byly události indikovány na GPIO pinu. Abychom se vyvarovali hrubých chyb při měření pomocí časovače, signál z GPIO pinu byl zobrazen na osciloskopu, kde byly odečteny hodnoty považované za správné.

4.2 Měření doby přepnutí kontextu s čekáním na synchronizační objekt

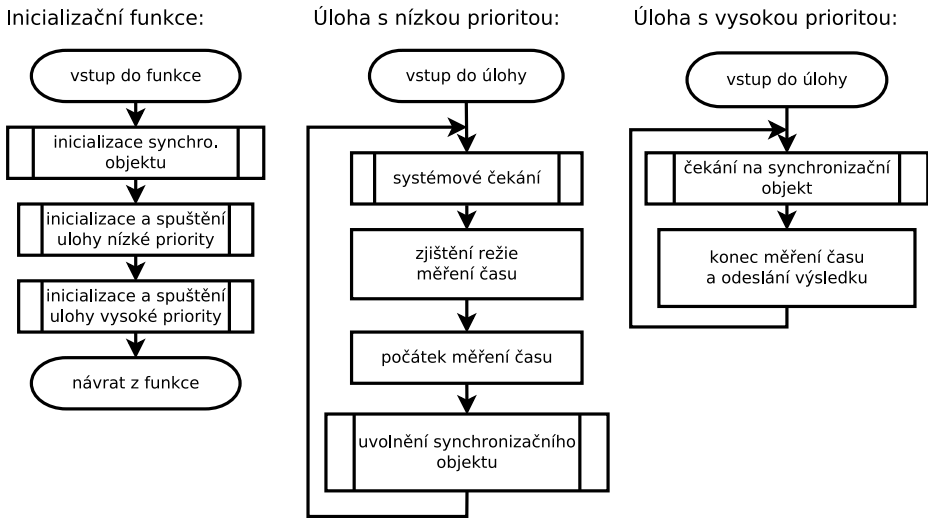
Přepnutí kontextu je společně se systémovým voláním pro mezi-procesní komunikaci nejčastěji prováděná operace jádra OS. Doba této operace proto reprezentuje nejpodstatnější část režie RTOS z hlediska času. Měření doby přepnutí kontextu

s čekáním na synchronizační objekt v sobě zahrnuje režii volání API funkcí, a proto je z hlediska porovnání výkonu objektivnější, než pouze často uváděná doba přepnutí kontextu.

V systému byly vytvořeny dvě úlohy, jedna s vysokou a jedna s nízkou prioritou. Čas, kdy úloha s vysokou prioritou čeká na úlohu s nízkou prioritou (pomocí synchronizačního objektu typu mutex a fronta), je měřena pomocí HW časovače. Uspořádání experimentu je patrné z obrázku 4.1. Program mikrokontroléru implementovaný do jednotlivých RTOS se řídí diagramem 4.2.



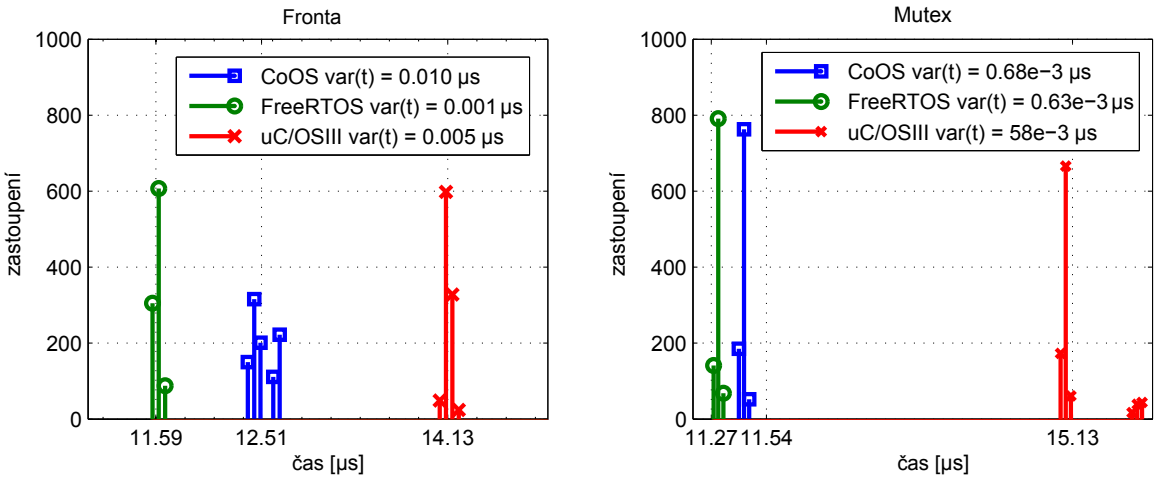
Obr. 4.1: Uspořádání HW a SW při měření doby přepnutí kontextu s čekáním na synchronizační objekt



Obr. 4.2: Diagram měření doby přepnutí kontextu s čekáním na synchronizační objekt

Pro toto měření byl vstupní obdélníkový signál pro časovač zvolen o frekvenci 9 MHz s amplitudou $2,8\,V_{P-P}$ a offsetem $1,5\,V$.

Výsledné časy, které vycházejí z 1000 opakovaných měření, jsou uvedeny společně se statistickými údaji v tabulce 4.1. Histogramy doby přepnutí kontextu a čekání na synchronizační objekty typu mutex a fronta jsou uvedeny v grafu 4.3.



Obr. 4.3: Histogram měření doby přepnutí kontextu s čekáním na synchronizační objekt

Tab. 4.1: Tabulka doby přepnutí kontextu s čekáním na synchronizační objekt

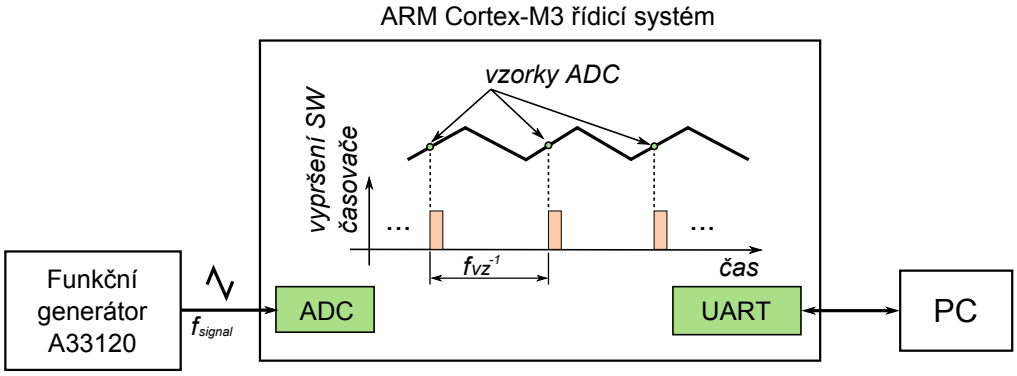
	CoOS	FreeRTOS	μC/OSIII
čekání na frontu (průměr)	12,51 μs	11,59 μs	14,13 μs
čekání na frontu (medián)	12,50 μs	11,61 μs	14,11 μs
čekání na frontu (rozptyl)	0,010 μs	0,001 μs	0,005 μs
čekání na mutex (průměr)	11,54 μs	11,27 μs	15,13 μs
čekání na mutex (medián)	11,55 μs	11,28 μs	15,07 μs
čekání na mutex (rozptyl)	0,68e-3 μs	0,63e-3 μs	58e-3 μs

Měření bylo nejvíce ovlivněno chybou odečítání času pomocí časovače ($\Delta_T = 55,\overline{5}\,ns$) a chybou frekvence generátoru ($\Delta_f = 0,002\ ns$)[16]. Rozšířená nejistota doby přepnutí kontextu s čekáním na synchronizační objekt pro pravděpodobnost 95 % potom bude:

$$u_{c-tSW} = k_r \cdot \sqrt{\frac{\Delta_O}{\sqrt{3}}^2 + \frac{\Delta_F}{\sqrt{3}}^2} = 2 \cdot \sqrt{\frac{55.\overline{5}}{\sqrt{3}}^2 + \frac{0.002}{\sqrt{3}}^2} = 0,0642\ \mu s.$$

Z výsledků měření je patrné, že jednotlivé implementace jádra vykazují různé výsledky pro daný testovací kód. U obou synchronizačních prostředků uspěl nejlépe FreeRTOS s nejkratším časem i rozptylem. U měření fronty vykazoval největší rozptyl CoOS, u mutexu μC/OSIII.

4.3 Měření absolutního časování softwarového časovače



Obr. 4.4: Uspořádání HW a SW při měření absolutní odchylky periody časovače

Abychom zjistili přesnost absolutního časování, neboli přesnost frekvence časovače, oproti skutečnému času, můžeme využít aliasing efektu. Ten vznikne při nedodržení Shannon-Kotělnikovova vzorkovacího teorému, kdy musí být frekvence vzorkovaného signálu poloviční oproti frekvenci vzorkováním, aby nedocházelo k překrytí spekter. Toho dosáhneme, pokud zvolíme frekvenci vzorkování stejnou jako frekvenci vzorkovaného signálu. Potom bude platit vztah [15]:

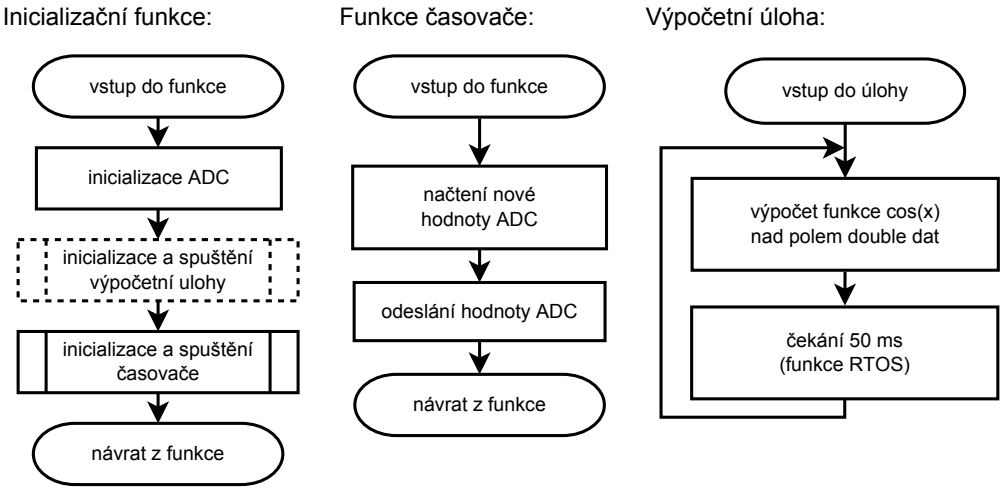
$$f_A = f_{VZ} - f_{signal} \quad \Longrightarrow \quad f_{VZ} = f_A + f_{signal} \quad [Hz]$$

Kde f_A je frekvence signálu ovlivněného aliasing efektem, f_{VZ} je frekvence vzorkování resp. frekvence SW časovače, f_{signal} je frekvence vzorkovaného signálu.

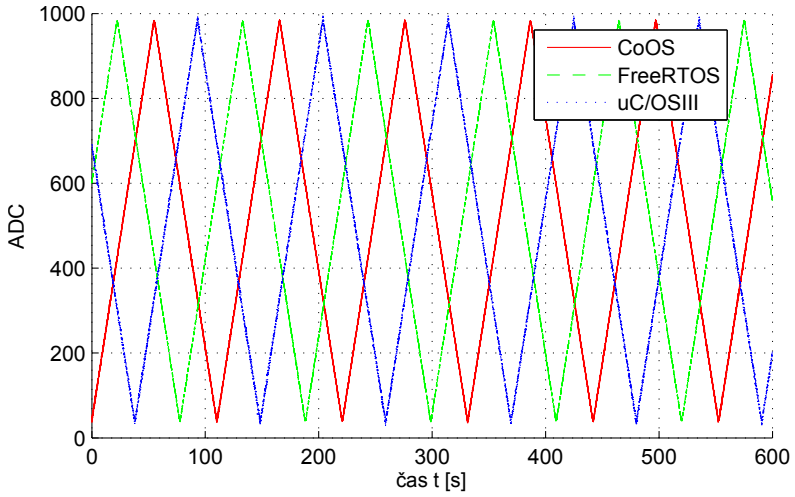
Testovací program odpovídá diagramu 4.5. Pro měření byla použita frekvence vzorkování $f_{VZ} = 100\,Hz$ a frekvence vzorkovaného trojúhelníkového signálu také $f_{signal} = 100\,Hz$, výstupní napětí generátoru bylo $2,8\,V_{P-P}$ s offsetem $1,5\,V$ pro maximální využití rozsahu AD převodníku. Grafy zobrazující signály ovlivněné aliasing efektem jsou na obrázku 4.6. Pro měření bylo získáno 60000 vzorků signálu z AD převodníku. Výsledky jsou shrnuty v tabulce 4.2.

	CoOS	FreeRTOS	μC/OSIII
f_A	90,42 mHz	90,50 mHz	90,25 mHz
f_{VZ}	100,09 Hz	100,09 Hz	100,09 Hz

Tab. 4.2: Frekvence signálu vzniklého aliasing efektem při měření absolutního časování se vzorkovací frekvencí 100 Hz.



Obr. 4.5: Diagram měření absolutního časování softwarového časovače



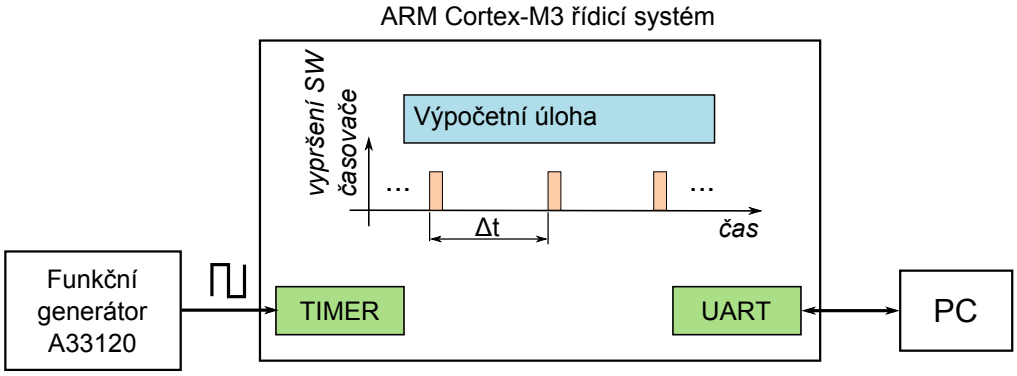
Obr. 4.6: Průběh signálů ovlivněného aliasing efektem

Měření bylo nejvíce ovlivněno odečítáním periody signálu vzniklého aliasing efektem vlivem šumu ($\Delta_O = \pm 0,9827\, mHz$) a chybou hodnoty frekvence generátoru ($\Delta_F = \pm 2\, mH$)[16]. Do nejistoty měření je zahrnuta nejistota typu B, která se skládá z těchto dvou složek. Rozšířená nejistota odchylky frekvence softwarového časovače pro pravděpodobnost 95 % potom bude:

$$u_{c-fA} = k_r \cdot \sqrt{\frac{\Delta_O}{\sqrt{3}}^2 + \frac{\Delta_F}{\sqrt{3}}^2} = 2 \cdot \sqrt{\frac{0,9827^2}{\sqrt{3}} + \frac{2^2}{\sqrt{3}}} = 2,573\ mHz.$$

Z výsledků měření absolutního časování je patrné, že všechny testované RTOS se chovají stejně a odchylka periody časovače je způsobená nepřesností zdroje hodinového kmitočtu mikrokontroléru.

4.4 Měření relativního časování softwarového časovače



Obr. 4.7: Uspořádání HW a SW při měření relativního časování

Relativní časování vyjadřuje, jak se liší jednotlivé periody softwarového časovače. Tento rozptyl frekvence je běžně označován pojmem *jitter*. Minimální rozptyl periody časovače je důležitým parametrem při použití časovačů pro řídicí algoritmy a také pro synchronizaci komunikačních protokolů. Uspořádání systému pro měření je znázorněno na obrázku 4.7. Testovací program odpovídá diagramu 4.8.

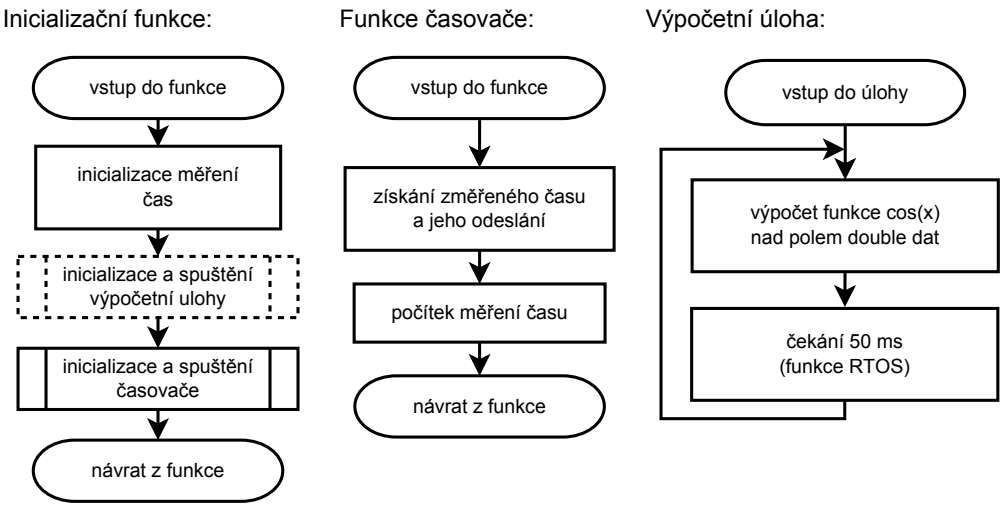
Pro toto měření byl vstupní obdélníkový signál pro čítač zvolen o frekvenci 3 MHz s amplitudou $2,8\,V_{P-P}$ a offsetem $1,5\,V$. Výsledné statistické údaje, které vycházejí z 5000 opakovaných měření, jsou uvedeny v tabulce 4.3. Rozložení period je patrné z histogramu 4.9.

Tab. 4.3: Statistické hodnoty periody časovače

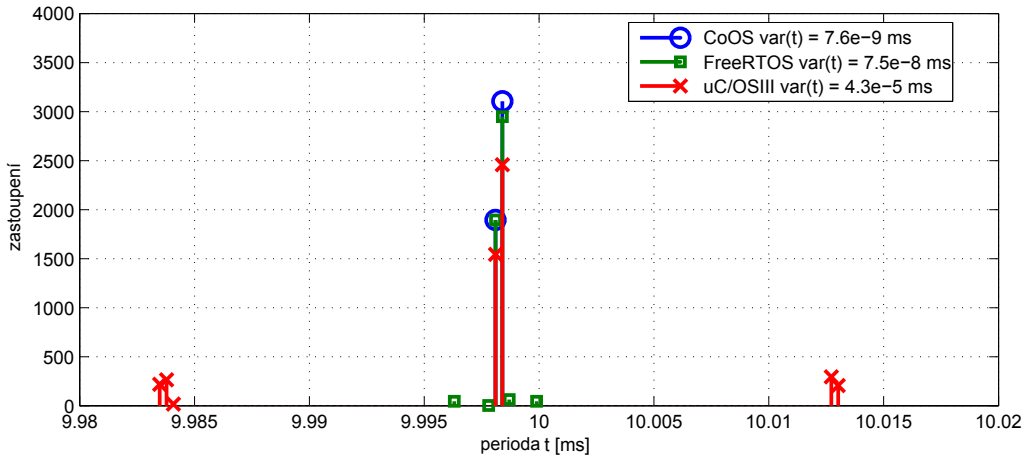
	CoOS	FreeRTOS	μC/OSIII
perioda soft. časovače (průměr)	0,01 s	0,01 s	0,01 s
perioda soft. časovače (medián)	0,01 s	0,01 s	0,01 s
perioda soft. časovače (rozptyl)	7,6e-9 ms	7,5e-8 ms	4,3e-5 ms

Měření bylo nejvíce ovlivněno chybou odečítání času pomocí časovače ($\Delta_T = 0,16\overline{6}\,\mu s$). Chyba frekvence generátoru je zanedbatelná[16]. Rozšířená nejistota periody softwarového časovače pro pravděpodobnost 95 % potom bude:

$$u_{c-tR} = k_r \cdot \sqrt{\frac{{\Delta_O}^2}{\sqrt{3}}} = 2 \cdot \sqrt{\frac{0,16\overline{6}^2}{\sqrt{3}}} = 0,192\ \mu s.$$



Obr. 4.8: Diagram měření relativního časování softwarového časovače



Obr. 4.9: Histogram měření relativního časování softwarového časovače

Výsledky měření relativního časování ukazují stejnou střední hodnotu u všech RTOS. Nejmenší rozptyl periody časovače byl zaznamenán u CoOS. FreeRTOS vykazoval mírně horší rozptyl než CoOS. Oproti tomu μC/OSIII vykazoval horší rozptyl než oba zmíněné.

Závěrem je uvedena velikost kódu jednotlivých testovacích programů, tyto údaje jsou shrnuty v tabulce 4.4.

Tab. 4.4: Velikost kódu jednotlivých testů

velikost kódu [B]	CoOS	FreeRTOS	μC/OSIII
čekání na frontu	6860	6792	10736
čekání na mutex	7236	7004	10560
abs. časování	15144	16776	13204
rel. časování	14936	16565	12984

5 FREERTOS

Dle výsledků předchozí kapitoly byl FreeRTOS zvolen pro implementaci firmwaru robotického vozítka.

Jak integrovat FreeRTOS do projektu bylo popsáno v kapitole o testování RTOS. Důležitou částí procesu vývoje *embedded* systému na bázi RTOS je konfigurace RTOS a základní pochopení jeho činnosti a kritických vlastností.

Nastavení FreeRTOS je koncentrováno do souboru `FreeRTOSConfig.h`. Při vytváření nového projektu je vhodné vycházet z konfiguračního souboru DEMO projektu, který je alespoň jeden pro daný port. Z hlediska využití CPU v čase jsou důležité parametry `configCPU_CLOCK_HZ` a `configTICK_RATE_HZ`, které korespondují s taktovací frekvencí. Z hlediska využití CPU v prostoru jsou důležité parametry `configTOTAL_HEAP_SIZE` a `configMINIMAL_STACK_SIZE`. Ohledně nastavení úloh obsahuje konfigurační soubor makra `configMAX_TASK_NAME_LEN` a `configMAX_PRIORITIES`. Pro škálování velikosti kódu FreeRTOS je nutné povolit jednotlivé skupiny funkčních volání pomocí maker, např. `INCLUDE_vTaskDelay`. V neposlední řadě obsahuje konfigurační soubor také makra související s portem a přerušeními, jak je uvedeno dále. Podrobný popis všech nastavení je možné nalézt na webových stránkách projektu.

Priorita plánovače FreeRTOS v rámci přerušení je nastavena prioritou přerušení systémového časovače, ta je definována pomocí makra `configKERNEL_INTERRUPT_PRIORITY`. Přerušení, které nevyužívají API funkce FreeRTOS, mohou používat libovolnou prioritu. Hraniční priorita, kdy je možné používat API funkce FreeRTOS je definována makrem `configMAX_SYSCALL_INTERRUPT_PRIORITY`. Tento mechanismus nastavení je nutné zavést pro korektní chování systému při vhnízďování přerušení. Názvy API funkcí FreeRTOS, které je možné používat v přerušeních končí `FromISR`. Konkrétní hodnoty nastavení pro Cortex-M3 mohou být `configKERNEL_INTERRUPT_PRIORITY=255` a `configMAX_SYSCALL_INTERRUPT_PRIORITY= 191`. Způsob, jakým je možné volit priority přerušením, je ukázán v tabulce 5.1. Vyšší hodnota NVIC priority označuje nižší prioritu přerušení.

Pro práci s úlohami v takto malém systému využijeme zpravidla pouze funkci `xTaskCreate()` a spuštění plánovače `vTaskStartScheduler()`. Funkce pro odstranění úlohy zpravidla nejsou potřeba. Mechanismus inverze priority je implementován přímo v jádře systému, funkce pro změnu priority úlohy proto nebyla využita. Důležitou diagnostickou funkcí je `vTaskList()`, která vrací výpis úloh společně s aktuálním využitím zásobníku.

Pro mezi-procesní komunikaci je základním prostředkem fronta. Od ní jsou odvozeny i další synchronizační prostředky. Kód, který může být vykonáván pouze jednou úlohou, můžeme chránit pomocí kritických sekcí, pokud se jedná o krátký

Tab. 5.1: Příklad nastavení priorit přerušení a chování FreeRTOS [10]

Priorita NVIC	
0b001xxxxx 0b010xxxxx 0b011xxxxx 0b100xxxxx	Přerušení které nemají nic společného s FreeRTOS, nebudou nikdy zpožděny plánovačem
0b101xxxxx 0b110xxxxx 0b111xxxxx	Přerušení s prioritou, které mohou volat „FromISR“ FreeRTOS API

přístup k HW nebo použít mutex. Mutex je koncipován tak, že úloha, která mutex dostane, musí mutex také vrátit. Pokud požadujeme synchronizaci typu čekání na událost mezi úlohami, použijeme binární semafor.

Základní literaturou pro používání systému FreeRTOS jsou webové stránky výrobce. Zde nalezneme obecné informace o různých částech systému. Stránky obsahují jisté množství reklamy odkazující na sesterské projekty. Orientace v menu není příliš intuitivní, snazší je použití mapy stránek, jednotlivé části systému jsou ovšem dobře popsány. Pro získání konkrétních informací je vhodné studovat volně dostupné zdrojové kódy, které jsou bohatě komentovány dle syntaxe pro *Doxygen* systém generování dokumentace. Na internetu jsou dostupné také dokumenty, které výpisem jednotlivých API funkčních volání systému s jejich popisem a příklady tvoří referenční API manuál. Důležitou literaturou ohledně FreeRTOS jsou také ukázkové aplikace. Kódy aplikací jsou komentované a vhodný jednoduchý program vysvětlí použití prostředků často nejlépe. V ukázkových aplikacích jsou také popsána místa kódu, kterým je třeba věnovat zvláštní pozornost. Při řešení případných problémů se systémem je vhodné využít diskusní fóra s rozsáhlou komunitou uživatelů FreeRTOS nebo využít placenou uživatelskou podporu. Je možné také získat oficiální placenou literaturu *Richard Barry: Using the FreeRTOS™ Real Time Kernel - A Practical Guide* od vedoucího FreeRTOS v tištěné nebo elektronické podobě.

Webové stránky FreeRTOS obsahují také databázi neoficiálních částí systému tvořených komunitou pod položkou *FreeRTOS Interactive!*

FreeRTOS je šířen pod modifikovanou GPL licencí, kdy je nutné dále pod GPL licencí poskytovat pouze kódy jádra FreeRTOS s jejich případnými úpravami. Tím je zachována možnost používat FreeRTOS i v komerčních aplikacích, bez dalších poplatků, bez požadavku zveřejňovat vlastní know-how, které je součástí kódu.

Společně s velkou popularitou [1] a množstvím podporovaných architektur je FreeRTOS vhodným kandidátem pro začlenění i do nových projektů.

[1] 77500 stažení za rok [10]

5.1 Diagnostika běžícího RTOS

Pro zajištění stabilní funkce řídicího systému obecně je potřeba znát využití jeho prostředků, a to jak v prostoru (paměť), tak i v čase (využití CPU, čekání na vstupně/výstupní operace).

Paměťové nároky zahrnují požadavky na paměť programu a paměť dat. Využití paměti programu je známé po kompilaci, po získání výsledného strojového kódu. Pokud používáme paměť dat pouze staticky, je její využití známé také v době kompilace. Pokud paměť dat obsahuje statické i dynamické proměnné, může být využití paměti dat proměnné v čase a známe je až za běhu systému. Operační systém FreeRTOS využívá dynamickou paměť k alokaci struktur jednotlivých úloh a k alokaci jednotlivých objektů mezi-procesní komunikace. Dominantní velikost paměti využívají jednotlivé uložené zásobníky úloh a položky front.

Maximální velikost oblasti dat pro dynamicky alokované položky je volena pomocí makra v konfiguračním souboru FreeRTOS. Volí se také typ alokačního algoritmu, v závislosti na našich požadavcích, přiložením vhodného zdrojového souboru do projektu. Velikost dynamicky alokované paměti volíme s ohledem na počet a přibližnou velikost jednotlivých objektů OS. Obecná možnost pro zjištění aktuálního využití dynamicky alokované paměti je „ladění za běhu“ přímo v aplikaci. FreeRTOS poskytuje nástroj pro zobrazení informací o vytvořených úlohách ve formě funkce `vTaskList()` [10]. Tato funkce podává aktuální informace o stavu jednotlivých úloh a také o využití jejich dynamicky alokované paměti pro zásobník. Analýzou této hodnoty můžeme vhodně optimalizovat výchozí hodnotu nastavení maximální velikosti zásobníku úlohy. Pro detekci přetečení zásobníku úlohy je možné použít funkci `vApplicationStackOverflowHook()` a vhodně na tuto nepříjemnou situaci reagovat. Pokud máme málo paměti, je vhodnou taktikou provádět všechny operace volající alokaci dynamické části paměti ještě před spuštěním plánovače úloh. Zároveň také ve spuštěných úlohách nepoužívat příliš rozmanitá volání funkcí rekurzivně. Potom může být maximální velikost zásobníku nastavena relativně těsně. Alokace všech komunikačních objektů a synchronizačních objektů před startem plánovače nám také usnadní psaní kódu kontrolujícího závislosti, které vznikají až za běhu systému.

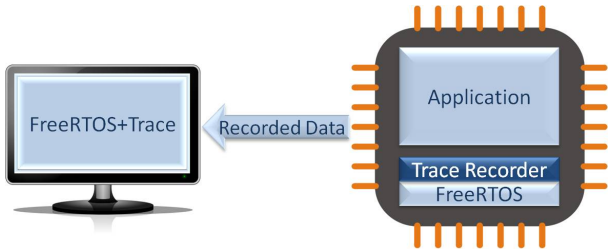
FreeRTOS obsahuje tzv. trasovací makra, která je možné povolit pomocí nastavení `configUSE_TRACE_FACILITY` na 1. Ve volání API funkcí FreeRTOS jsou umístěna na vhodných místech makra, kterými je možné reagovat na události, při kterých jsou volána. Tato makra jsou ve výchozím stavu prázdná, negenerují žádný kód. Použitím vhodných maker můžeme například ovládat analogové nebo digitální výstupy při přepnutí kontextu. Je možné také použít sofistikovanější knihovny zajišťující kompletní trasování běžícího systému.

Další možností sledování využití CPU je použití funkce Run Time Stats realizovanou funkcí `vTaskGetRunTimeStats()`, která provede výpis běžících úloh společně s jejich absolutním a relativním časováním. Pro umožnění tohoto mechanismu je nutné implementovat funkce, které zaručí přístup k HW časovači.

Tyto nástroje pracují s nenulovou režií paměti a výkonu CPU, ovšem k jejich diagnostickým možnostem a komfortu ladění za běhu, je ztráta několika procent výkonu na Cortex-M3 zanedbatelná.

5.1.1 Nástroj pro run-time diagnostiku Trace

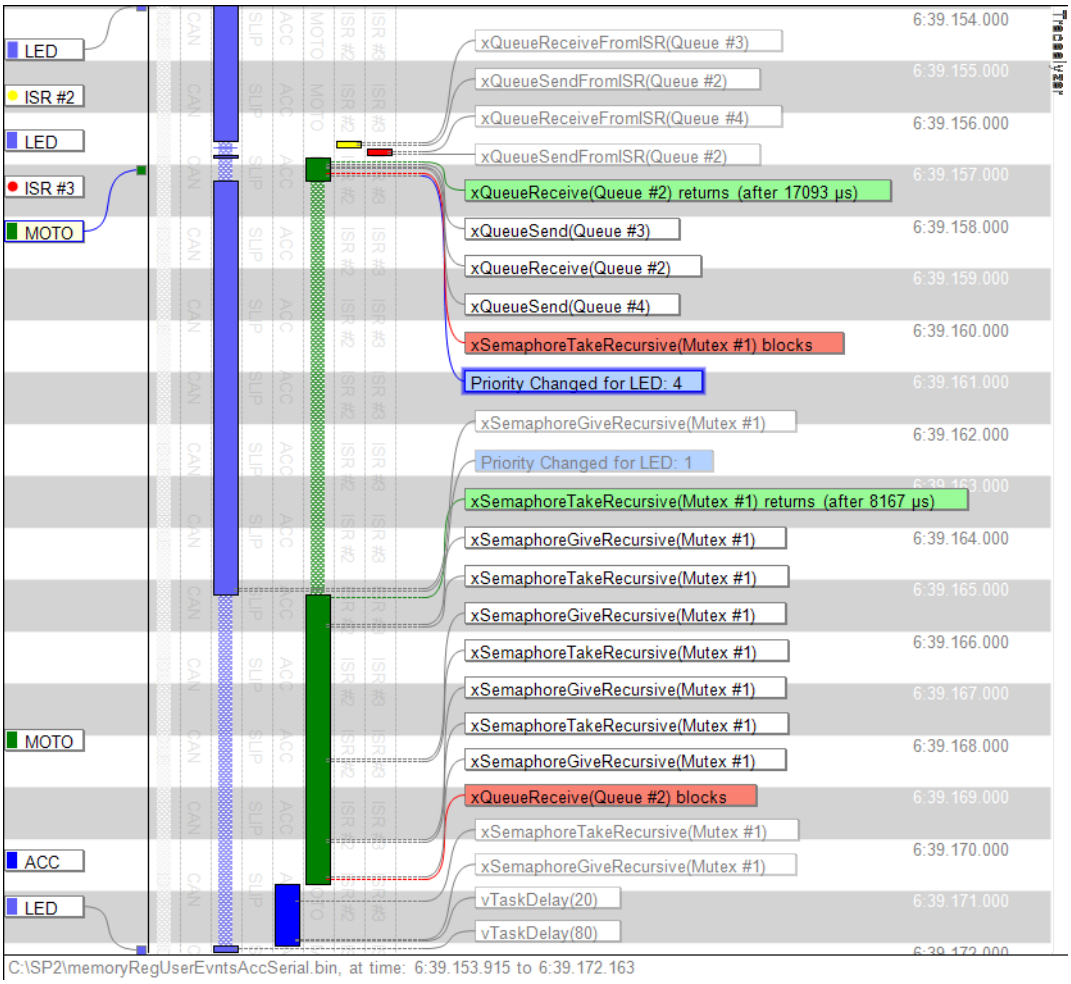
FreeRTOS+Trace je produkt firmy Percepio, který slouží ke sledování využití prostředků systému, ve kterém běží FreeRTOS. Skládá se ze dvou částí, z knihovny pro zaznamenávání aktivit jádra OS, která je navázaná na trasovací makra ve FreeRTOS a z aplikace pro PC (Tracealyzer), která slouží k dekódování těchto záznamů a jejich filtrování a reprezentaci do přehledné podoby. Tyto záznamy společně s tabulkou objektů jsou uloženy v bloku paměti dat.



Obr. 5.1: Nákres principu FreeRTOS+Trace [11]

Knihovna je, počínaje verzí 7.3.0, součástí balíku FreeRTOS. Do projektu je ji možné zařadit vložením zdrojových souborů ze složky `FreeRTOS-Plus/FreeRTOS-Plus-Trace` společně s přiložením hlavičových souborů. Dále je nutné do projektu přiložit dva konfigurační soubory. Soubor `trcPort.h` obsahuje portovatelné funkce pro HW časovač podle architektury CPU (je nutné upravit makro `SELECTED_PORT`). Soubor `trcConfig.h` obsahuje nastavení délky kruhového bufferu (ve výchozím stavu 1000 položek) pro záznam jednotlivých akcí a dále volby, jaký typ akcí zaznamenávat. Do souboru `FreeRTOSConfig.h` nadefinujeme `configUSE_TRACE_ FACILITY 1` a přiložíme trasovací makra ze souboru `trcHooks.h`. Do zdrojových souborů přidáme `trcUser.h`. [10]

Zaznamenaná data jsou při běhu programu uložena v paměti dat. Nejsnadnějším způsobem, jak je získat, je použití JTAG rozhraní a načtení celé paměti RAM do souboru během ladění ve vývojovém prostředí. V CoIDE je to možné provést na



Obr. 5.2: Zobrazené TraceView

kartě *Memory*. Při zastaveném programu načteme paměť pomocí tlačítka *Export memory* od adresy `0x20000000` o velikosti 65536 b do **.bin* souboru.

Aplikace FreeRTOS+Trace (Tracealyzer) umožňuje načíst binární data, ve kterých nalezne podle značky start záznamu. Ze záznamu vytvoří graf sledu událostí, kterým lze snadno procházet a filtrovat jednotlivé operace OS. Ukázka tohoto grafu je na obrázku 5.2.

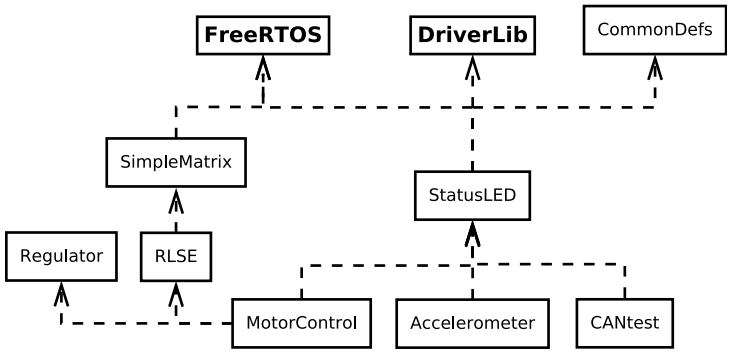
Je možné zobrazovat také chování během přerušení (v kódu zaznamenané makrem `vTraceStoreISRBegin(n)`). Uživatelské hodnoty, v kódu zaznamenané pod názvem funkcí `vTracePrintF()` je možné zobrazovat v časových grafech.

FreeRTOS+Trace je komerční produkt, který v omezené bezplatné verzi poskytuje pouze zobrazení sledu událostí systému. Bezplatná verze je značně ořezána, ovšem oproti tradičnímu ladění kódu umožňuje nacházet daleko záludnější chyby v jednotlivých úlohách.

6 FIRMWARE PRO ROBOTICKÉ VOZÍTKO

6.1 Struktura firmware a jeho API

Pro zachování přehlednosti a modularity je vhodné každý rozsáhlejší projekt vhodně členit na programové moduly. U rozsáhlých systémů má toto členění důvod nejen v přehlednosti, ale je zde také důvod překladu, kdy je možné jednotlivé části překládat separátně, nebo vytvářet knihovny. Firmware je členěn z jednoho hlediska podle kooperujících HW částí a z druhého hlediska podle nezávislých knihoven. Programové moduly a jejich závislosti jsou shrnuty na obrázku 6.1.



Obr. 6.1: Hierarchie jednotlivých modulů firmware

Operační systém nám umožňuje rozčlenit řídicí software na úlohy, které jsou zaměřeny na konkrétní činnosti. Programové moduly `StatusLED`, `MotorControl`, `Accelerometer` a `CANtest` vykonávají svou činnost každá ve vlastní FreeRTOS úloze. Každý z těchto modulů vytváří tuto úlohu v inicializační funkci. Úloze je nastaveno jméno, pevná velikost zásobníku a priorita, která je zadávána jako parametr inicializační funkce. U inicializačních funkcí je priorita předána funkci prvním parametrem `xxxInit( priority,...)`. Inicializační funkce vracejí hodnotu `pdPass`, pokud inicializace proběhla v pořádku. Důležitou roli v systému hraje také obsluha přerušení a jejich priority. V programových modulech je tato priorita nakonfigurována pevně tak, aby souhlasila s nastavením FreeRTOS (viz tabulka 5.1).

Všechny programové moduly firmware mají závislost na souboru `commonDefs.h`. Tento soubor obsahuje obecně použitelná makra související s HW a také definice vývodů a portů mikrokontroléru, které jsou využívány v jednotlivých programových modulech. To umožňuje snadné přemapování vývodů při změně zapojení.

Statická nastavení, která souvisejí s konkrétními vlastnostmi programového modulu jsou ve formě maker umístěna na začátku hlavičkového souboru každého programového modulu zvlášť.

V následujícím popisu jednotlivých programových modulů jsou uvedeny pouze základní informace o implementovaných funkcích, kompletní podrobný popis je součástí přílohy na CD. Aplikační programové rozhraní jednotlivých modulů je zdokumentováno v kódu pomocí komentářů speciálního formátu. Tento formát odpovídá *Doxygen* syntaxi pro automatické generování API a dokumentace softwarových projektů stejnojmenným programem. Tento postup se v praxi běžně používá a jistou automatizací zefektivňuje práci na dokumentaci projektu. Výsledná dokumentace může být ve formě PDF souboru, souboru Windows nápovědy `*.chm` nebo formou webových stránek. Pro generování dokumentace byl s volně dostupným programem Doxygen použit také program Graphviz, pro vytváření diagramů, a program HTML Help Workshop pro vytvoření `*.chm` souboru. Volby vytváření dokumentace jsou v souboru `Doxyfile`, který je součástí zdrojových souborů firmware.

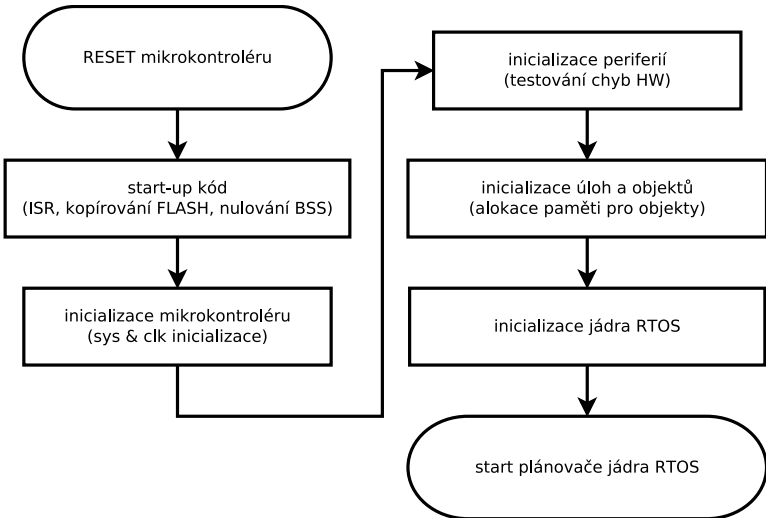
Pro zdrojový kód firmware byl využit verzovací systém `git`, hostovaný na serveru *github.com*. Tento systém umožňuje kromě zálohy zdrojových kódů také jejich snadnou aktualizaci s možností vrátit se k předchozí verzi, případně vytvoření další odkloněné verze. Součástí serveru *github.com* jsou také funkce na hlášení chyb a různé komunikační funkce podobné sociálním sítím. Veřejně přístupná adresa repositáře se zdrojovým kódem firmware je na adrese https://github.com/lukaso25/robotic-vehicle-firmware. Zde je možné stáhnout aktuální verzi firmware ve formě `*.zip` souboru. Zdrojový kód včetně dokumentace a API je součástí přílohy CD.

6.2 Start systému

Před startem plánovače úloh FreeRTOS probíhá řada operací, které souvisí s nastavením HW i operačního systému samotného. Sled událostí je zobrazen v diagramu na obrázku 6.2. Inicializace periferií a objektů operačního systému je součástí inicializačních funkcí programových modulů.

6.3 Ovladače periferií mikrokontroléru

Výrobce mikrokontroléru poskytuje knihovny pro ovládání vnitřních periferií v balíku pod jménem *Stellaris® Peripheral Driver Library*[13]. Knihovna je distribuována formou zdrojových kódů. Je napsaná v jazyce C a je možné ji použít s většinou kompilátorů včetně GCC. Moduly tvořené soubory `*.c` a `*.h` odpovídají jednotlivým periferiím. Pro přístup k periferiím je možné použít buď přímý přístup k registrům (v knihovně jsou definovány symbolické názvy jednotlivých řídicích bitů), nebo použít volání funkcí specifických pro danou periferii (náročnější kód, který více odpovídá logickému uspořádání a je snadnější pochopení a použití).



Obr. 6.2: Diagram inicializace řídicího software

6.4 Modul indikace chybových stavů

Tento programový modul je realizován v souborech `StatusLED.c` a `StatusLED.h`. Je tvořen inicializační funkcí `StatusLEDInit()` a funkcí úlohy `StatusLED_task()`.

Nejprve byl vytvořen jednoduchý kód, který blikal LED a indikoval běh jádra FreeRTOS. Aby bylo možné zobrazovat větší množství chybových stavů najednou, byla úloha upravena tak, aby místo konstantního blikání umožňovala „vyblikat“ chybové kódy. Určitý počet bliknutí, odpovídající chybovému kódu, je následován prodlevou pro oddělení chyb. Jedním bliknutím je indikován správný běh systému. Pro zaznamenání chyby byla vytvořena funkce `SetError()`. Tato funkce je využívána ostatními moduly pro indikaci chyb vzniklých za běhu. Tento modul je tedy možné snadno rozšířit na další reakce vzniklých chyb (akusticky, informováním nadřazeného systému). Definované chyby jsou shrnuty v tabulce 6.1.

6.5 Modul matematických maticových operací

Tento modul slouží pro práci s maticemi, které jsou použity v modulu identifikace metodou nejmenších čtverců. Je tvořen soubory `SimpleMatrix.c` a `SimpleMatrix.h`

Matice je reprezentovaná pomocí struktury `MatrixType`. Tato struktura obsahuje ukazatel na jednorozměrné pole prvků matice a proměnné reprezentující rozměry matice. Pro prvky matice byl definován nový typ `matrixValType`.

Inicializace matic využívá dynamické alokace pomocí správy paměti FreeRTOS. K alokaci matice slouží funkce `matAlloc()` a k uvolnění funkce `MatFree()`.

Tab. 6.1: Chyby indikované LED diodou

počet záblesků LED	popis chyby
1	systém v pořádku
2	kritická hodnota napětí baterie
3	chyba komunikace s akcelerometrem
4	chyba v řízení motorů
5	vypršení časovače komunikace s nadřazeným systémem
6	chyba komunikace CAN
7	chybná přijatá data

K prvkům matice je možné přistupovat pomocí makra `CELL(matrix,i,j)`. Byly implementovány funkce pro inicializaci prvků matice na stejnou hodnotu `matFill()` a pro hodnoty na hlavní diagonále `matEye()`.

V knihovně je možné pomocí makra `MATRIX_CHECK_DIMENSIONS` nastavit, aby byly při matematických operacích kontrolovány rozměry matic. Pokud kontrola zjistí rozpor, daná funkce vrací hodnotu -1.

Popis jednotlivých funkcí pro sčítání, odčítání, násobení, transponování, přičtení konstanty, škálování matic jsou uvedeny v dokumentaci API.

Tato knihovna by v současné době mohla být nahrazena funkcemi z knihovny CMSIS-DPS, ovšem v době programování těchto knihoven ještě nebyla dostupná.

6.6 Modul identifikace RLS

Tento programový modul realizuje identifikaci rekurzivní metodou nejmenších čtverců a je tvořen soubory `RLS.c` a `RLS.h`. Tento modul používá modul matematických maticových operací. Obsahuje deklaraci struktury, která obsahuje matice pro výpočet identifikačního algoritmu.

K inicializaci této struktury slouží funkce `RLS_init()`, která provede alokaci jednotlivých matic algoritmu a jejich inicializaci na výchozí hodnoty.

Identifikovaná soustava byla určena 2. řádu. Parametry identifikované soustavy jsou v tomto modulu reprezentovány modelem ARX, který je ve vektorovém zápisu uveden dle následujícího vztahu:

$$y_{(k)} = \varphi^T_{(k)}\theta_{(k)} + e_{s(k)} \tag{6.1}$$

kde

$$\varphi^T_{(k)} = \begin{pmatrix} -y_{(k-1)} & -y_{(k-2)} & u_{(k-1)} & u_{(k-2)}) \end{pmatrix} \tag{6.2}$$

je vektor pozorování

$$\theta_{(k)}^T = \begin{pmatrix} a_1 & a_2 & b_1 & b_2 \end{pmatrix} \tag{6.3}$$

je vektor parametrů, e_s je chyba predikce, y je měřená hodnota na výstupu a u je akční zásah vstupující do soustavy. Koeficienty a_n a b_n jsou koeficienty polynomů identifikovaného přenosu.

K vlastní identifikaci slouží funkce `RLS_update()`, která má jako vstupní parametry měřenou hodnotu výstupu soustavy a akční zásah do soustavy. Třetím parametrem funkce je určeno, zda bude probíhat identifikace, nebo pouze uložení měřených hodnot. Vhodnou volbou podmínky můžeme tímto parametrem určit chování, že identifikace bude probíhat pouze při změnách v systému. Tato funkce musí být volána v každé periodě vzorkování.

Identifikace přírůstkovou metodou nejmenších čtverců je prováděna dle následujících rovnic[23]:

$$\begin{aligned} K_{(k+1)} &= P_{(k)}\varphi_{(k+1)}\left[\lambda_e + \varphi_{(k+1)}^T P_{(k)}\varphi_{(k+1)}\right]^{-1} \\ \theta_{(k+1)} &= \theta_{(k)} + K_{(k+1)}(y_{(k+1)} - \varphi_{(k+1)}^T\theta_{(k)}) \\ P_{(k+1)} &= \left(P_{(k)} - K_{(k+1)}\varphi_{(k+1)}^T P_{(k)}\right)\frac{1}{\lambda_e} \end{aligned} \tag{6.4}$$

kde P je kovarianční matice, matice K určuje změnu parametrů v daném kroku.

Výsledné identifikované parametry jsou součástí struktury identifikace `rlsType` v maticí označené `th`.

6.7 Modul regulátoru

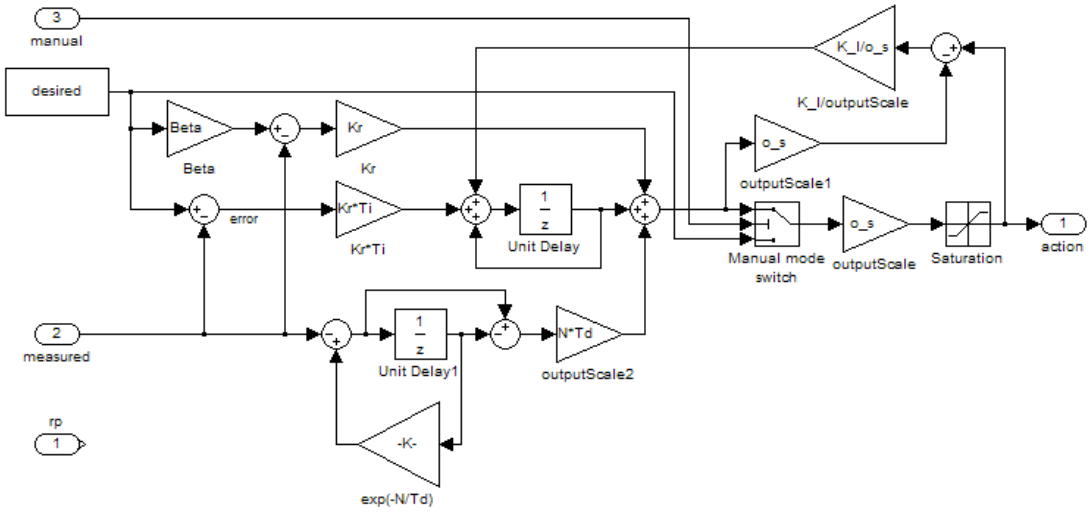
Programový modul `Regulator.c` a `Regulator.h` je implementaci diskrétního PID regulátoru, který je uzpůsoben pro praktické použití modifikacemi uvedenými v teoretické části.

Pomocí makra `REGULATOR_PI_VERSION_ONLY` je možné zvolit kód, který realizuje pouze PI regulátor.

Modul obsahuje deklaraci struktury `RegulatorParams`, která udržuje stavové proměnné a nastavené hodnoty regulátoru.

Pro aktualizaci akčního zásahu slouží funkce `RegulatorAction()`. Tato funkce počítá akční zásah dle diagramu 6.3. Pro nastavování parametrů regulátoru slouží funkce `RegulatorSetDesired()`, `RegulatorSetPID()`, `RegulatorSetParams()` a `RegulatorSetScaleLimit()`.

Pro inicializaci a speciální situace (snížení vlivu pásma necitlivosti soustavy) je možné pomocí funkce `RegulatorResetStates()` nastavit stavové proměnné na výchozí nulové hodnoty.



Obr. 6.3: Diagram realizovaného diskrétního regulátoru

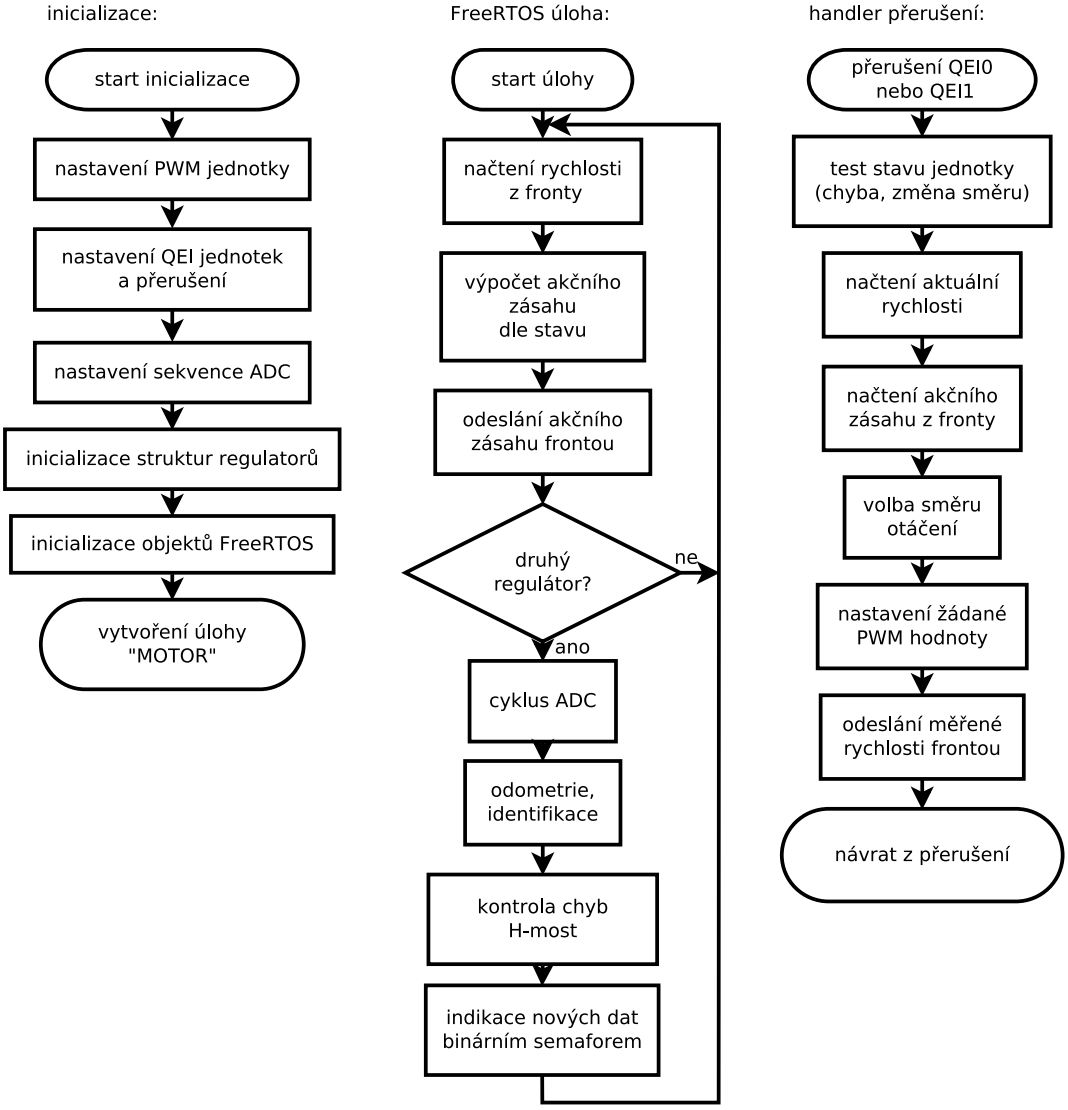
6.8 Modul řízení podvozku

Modul regulace rychlosti je tvořen soubory `MotorControl.c` a `MotorControl.h`. Využívá PWM jednotku, jednotky kvadraturních dekodérů QEI a analogově-číslicový převodník ADC.

Nejprve byl implementován mechanismus různých režimů regulace pohonu, aby mohlo být reagováno na chybové stavy. Je zde chybový režim, režim STOP a režim SHUTDOWN, který vypne H-mosty, a tím dojde k odbrzdění. Je zde také manuální režim, který dovoluje přímé řízení PWM s vyřazenými regulátory. Tento režim může být vhodný pro identifikaci nebo implementaci regulátoru v nadřazeném systému.

Frekvence PWM byla zvolena s ohledem na maximální frekvencí H-mostů a rozlišení PWM generátoru. Podle těchto parametrů byla výchozí zvolena na 10 kHz. Je ji možné změnit makrem v hlavičkovém souboru modulu. Při této frekvenci PWM a frekvence jádra 20 MHz je rozlišení jednotky 2000 úrovní. Tato frekvence je slyšitelná lidským uchem. Nizké frekvence nejsou vhodné z hlediska opotřebení motoru, vyšší frekvence nejsou vhodné z hlediska ztrát na H-mostu a nižšímu rozlišení PWM, které ovšem není kritické, vzhledem k nelinearitám převodovky a prokluzům kol. Perioda vzorkování byla zvolena vzhledem k dynamice pohonů a rozlišení žádané rychlosti z inkrementálních senzorů na 20 ms. Regulátor reguluje v jednotce množství pulzů/periodu vzorkování.

Inicializační funkce obsahuje inicializaci HW jednotek a inicializaci objektů FreeRTOS. Sled operací je patrný z diagramu na obrázku 6.4. Nejprve je nastavena PWM jednotka, její požadované kanály a frekvence. Následuje nastavení QEI jednotek,



Obr. 6.4: Diagram úlohy regulace rychlosti motoru

kde je nastaveno čítání obou kvadraturních signálů pro dosažení vyššího rozlišení. Jsou zde nastaveny časovače měření rychlosti s periodou 20 ms, které slouží jako generátory vzorkovací frekvence pro regulátory. Jsou povolena přerušení a jsou jim nastaveny různé priority. Pro nastavení AD převodníku jsou použity sekvence. Sekvence je nakonfigurována pro načtení 3 kanálů převodníku, iniciátorem převodu je SW. Protože se měřené signály rychle mění v čase, byla také aktivována průměrovací jednotka převodníku. Výsledná hodnota je průměrem 64 vzorků. Pro komunikaci mezi přerušeními jsou vytvořeny 3 fronty. První fronta slouží k přenosu naměřené rychlosti. K hodnotě rychlosti je přiřazen identifikátor QEI jednotky pro korektní synchronizaci. Další dvě fronty slouží k odesílání akčního zásahu, každá fronta slouží zvlášť pro jeden PWM kanál motoru.

Pro každou jednotku QEI je definován jeden vektor přerušení. V obsluze přerušení je nejprve uložena hodnota rychlosti. Dále dochází k vyzvednutí akčního zásahu pro motor a jeho předání PWM jednotce. Jsou zde ošetřeny nestandardní hodnoty a jednotka je nastavena pro správný směr otáčení. Následně je odeslána hodnota rychlosti pomocí fronty řídicí úloze.

V úloze modulu, která je reprezentována funkcí `MotorControl_task()`, je nekonečná smyčka, ve které je realizován řídicí algoritmus regulátorů motoru. Nejprve se čeká na příchod naměřené rychlosti od jedné z QEI jednotek. Tato fronta je plněna s konstantní periodou 20 ms, danou časovačem měření rychlosti. Následuje test režimu řízení motorů. Pokud je modul v režimu RUNNING, jsou vypočítány akční zásahy pro motory, které jsou odeslány frontou. Zde je využita funkce `RegulatorAction()`. Tento zásah je přijat v následujícím kroku časovače. Tato část kódu je stejná pro oba motory - obě měření QEI jednotek.

Všechna data a parametry modulu `MotorControl` jsou ukládána do globální struktury `myDrive`.

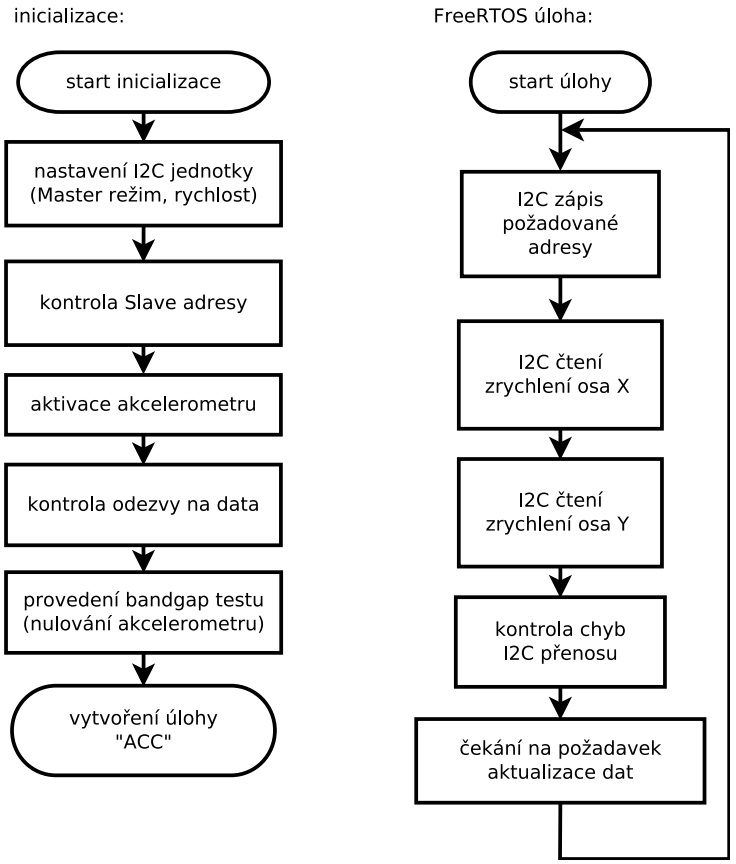
Po každém cyklu obou regulátorů následuje společný kód pro oba regulátory. V této společné části jsou také načteny hodnoty AD převodníků a jsou testovány limitní stavy H-mostů, společně s kontrolou stavu baterií. Z průměrné hodnoty napětí je upraveno zesílení regulátoru, tím je kompenzován aktuální stav nabití akumulátorů. Dále společná část obsahuje kód který ze změny natočení kol vypočítá pomocí odometrie změnu polohy robota. Tato funkce je volitelná je je možné ji zakázat nebo povolit v hlavičkovém souboru pomocí makra. Následuje volitelné volání funkce pro identifikaci soustavy.Závěrem je nastaven synchronizační objekt, který indikuje přítomnost dat z nové periody. Novou periodu je možné detekovat funkcí `MotorControlWaitData()`.

Pro nastavování rychlostí podvozku slouží funkce `MotorControlSetSpeed()`, které předáváme rychlost dopředu v mm/s a rotační rychlost v rad/s. Parametry podvozku a výchozí nastavení regulátorů je pro korektní přepočty uvedeno v hlavičkovém souboru.

Odezva na jednotkový skok je zobrazena v grafu C.1. Průběh akční veličiny je zobrazen v grafu C.2. Z přechodové charakteristiky je patrné, že dojde k vyregulování žádané rychlosti s nulovou ustálenou odchylkou bez překmitu. Z průběhu akční veličiny je patrné, že soustava obsahuje nelinearitu typu pásmo necitlivosti. Integrátor v tomto případě způsobuje, že i po odeznění regulační odchylky je akční zásah nenulový.

6.9 Modul akcelerometru

Modul akcelerometru je realizován soubory `Accelerometr.c` a `Accelerometr.h`. Využívá I^{2}C jednotku mikrokontroléru.



Obr. 6.5: Diagram úlohy čtení hodnot akcelerometru

Průběh inicializační funkce `AccelerometrInit()` je znázorněn v diagramu na obrázku 6.5.

V úloze reprezentované funkcí `Accelerometr_task()` se v nekonečné smyčce čeká na synchronizační objekt typu binární semafor, který značí požadavek na aktualizaci dat z akcelerometru.

Funkce `AccelerometerRequestData()` slouží pro vyžádání nových dat z akcelerometru. Funkce `AccelerometerGetX()` a `AccelerometerGetY()` slouží přečtení hodnoty zrychlení převedené do jednotky $m.s^{-2}$.

6.10 Modul CAN komunikace

Tento programový modul slouží k otestování CAN rozhraní. je tvořen soubory `CANTest.c` a `CANTest.h`, obsahuje funkci pro vytvoření úlohy `CANtestInit()` a funkci FreeRTOS úlohy `CANTest_task()`.

Inicializace HW probíhá povolením hodinového kmitočtu pro periferii, konfigurací GPIO pinů a poté nastavením komunikační rychlosti na 1 Mb/s. Sběrnice CAN implementuje také mechanizmus re-synchronizace stanic připojených na sběrnici. Aby tento mechanismus korektně plnil svoji funkci, je potřeba nastavit správné rozložení časových kvant v rámci jednoho bitu zprávy.[26] V ukázkovém kódu je toto provedeno automaticky voláním funkce `CANBitRateSet()` pro nastavení přenosové rychlosti. Pro alternativní nastavení je možné použít funkci `CANBitTimingSet()`.

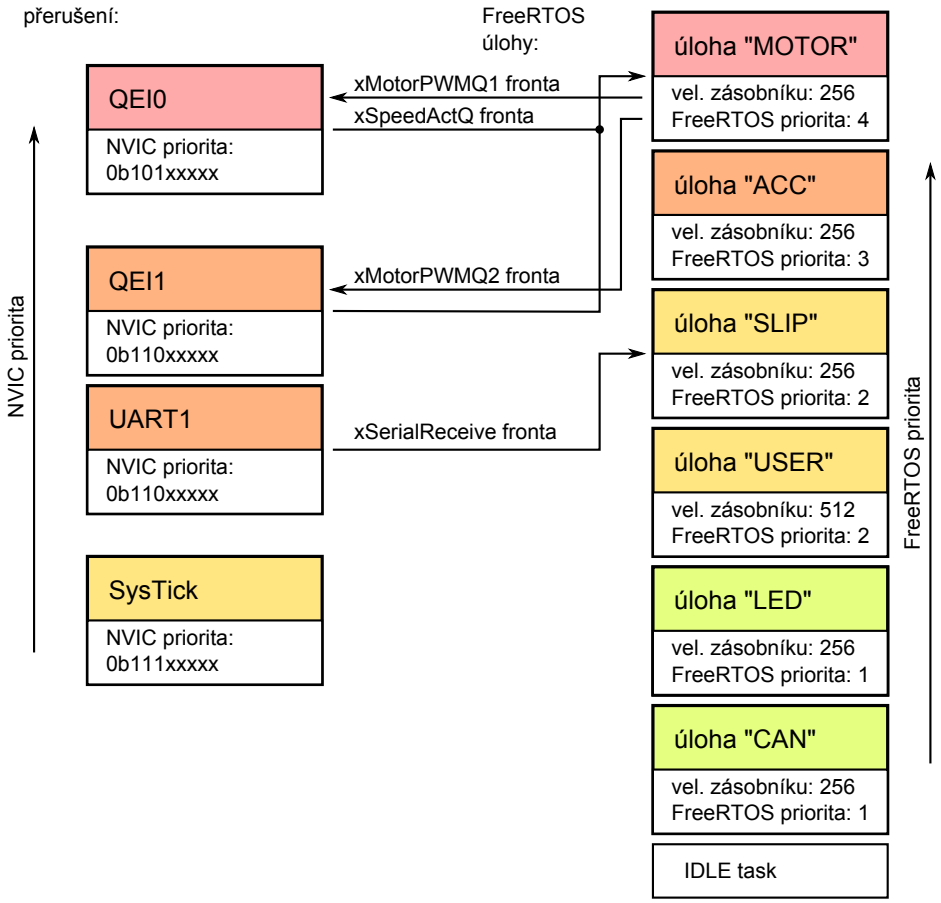
Důležitou částí inicializace je také konfigurace objektů zpráv *Message objects* a jejich bufferů. Byly vytvořeny tři tyto objekty. První z nich slouží pro příjem CAN zpráv s identifikátorem 0x04. Druhý slouží k odpovědi na vzdálené vyžádání s identifikátorem 0x03. Třetí slouží k odesílání zprávy s identifikátorem 0x02.

V nekonečné smyčce úlohy je vložena čekací smyčka na příchozí objekt. Po příchodu je modifikován buffer odesílané zprávy a ta je následně odeslána. Odpověď na vzdálené vyžádání je prováděna automaticky na pozadí v HW - procesor do ní nijak nezasahuje. Rozhraní bylo otestováno připojeným převodníkem CAN/USB.

7 UKÁZKOVÁ APLIKACE

Pro účely testování a pro účely demonstrace použití API programových modulů základního firmware byla vytvořena ukázková aplikace. Tato aplikace sestává z kódu na straně firmware robota a z kódu C# aplikace pro PC. Firmware je rozšířen uživatelskou FreeRTOS úlohu a o programový modul umožňující snadnou komunikaci prostředním sériové linky. Sériové spojení mezi robotem a nadřazeným systémem (PC) bylo realizováno pomocí *Bluetooth* bezdrátové sériové linky. Toto spojení sice nevyniká rychlou a deterministickou odezvou, ovšem svou spolehlivostí přenosu pro účely sběru dat postačuje.

Rozčlenění úloh ukázkové aplikace, jejich názvy a přidělená priorita jsou zobrazeny na obrázku 7.1. Jsou zde také znázorněna přerušení společně s prioritami a tokem dat pomocí front.



Obr. 7.1: Rozčlenění řídicího software na jednotlivé úlohy s naznačenou komunikací v přerušeních

7.1 Modul komunikace sériovou linkou

Programový modul pro komunikaci po sériové lince je tvořen soubory `SlipSerial.c` a `SlipSerial.h`. Tento modul ke komunikaci používá jednotku UART1 mikrokontroleru.

Pro usnadnění komunikace prostřednictvím sériové linky byl zvolen protokol SLIP dle RFC 1055 [20], který rozděluje proud znaků na pakety, byl určen původně pro přenos IP paketů.

Jednotlivé pakety jsou oddělovány znakem SLIP_END. Pokud proud dat obsahuje SLIP_END, je tento znak nahrazen posloupností SLIP_ESC a SLIP_END. Pokud proud dat obsahuje SLIP_ESC, je tento znak nahrazen posloupností SLIP_ESC a SLIP_ESC. Tímto způsobem je zajištěno korektní oddělení paketů s minimální režií. Paket je tvořen tak, že první bajt obsahuje identifikátor dat předem známé délky.

Inicializace programového modulu je realizována ve funkci `SlipSerialInit()`. V této funkci je inicializována UART1 jednotka pro korektní rychlost a formát přenosu dat. Je zde také povoleno přerušení od přijmu znaku s nastavenou NVIC prioritou 6. Dále jsou vytvořeny objekty FreeRTOS, a to mutex pro odesílání paketů a fronta pro příjem znaků z přerušení.

Tab. 7.1: Tvorba paketu protokolu SLIP, ukázka náhrady řídicího znak SLIP_END a SLIP_ESC

data (n bytů)							SLIP_END
data	SLIP_ESC	ESC_END	data	SLIP_ESC	ESC_ESC	data	SLIP_END

Vysílání je realizováno funkčním voláním `SlipSend()`. Aby nemohl být vysílaný paket přerušen jiným, je funkce opatřena mutexem. Vlastní kódování SLIP vychází z kódu uvedeného v [20].

Přijaté znaky ze sériové linky jsou v přerušení předávány frontě příchozích znaků. Dekódování jednotlivých paketů a jejich zpracování je realizováno stavovým automatem v nekonečné smyčce úlohy. Tato úloha je reprezentována funkcí `SlipSerial_task()`.

Na příchozí SLIP pakety reaguje externí funkce `SlipSerialProcessPacket()`, kde jsou data zpracovávána. Na vypršení časovače komunikace reaguje externí funkce `SlipSerialReceiveTimeout()`.

V hlavičkovém souboru byly definovány identifikátory paketů, jak je ukázáno v tabulce 7.2.

Tab. 7.2: Identifikátory paketů

identifikátor	délka [B]	obsah
ID_ACC_STRUCT = 0x10	5	data z akcelerometru
ID_REG = 0x11	13	hodnoty regulátoru
ID_TASKLIST = 0x12	257	řetězec s výpise úloh
ID_ADC = 0x13	7	měření napětí a proudů
ID_MOTOR_MODE = 0x15	2	režim regulátoru
ID_TIME_STAMP = 0x16	2	časová značka každou periodu vzorkování
ID_REG_PARAMS = 0x17	9	parametry regulátoru

7.2 Úloha komunikace s nadřazeným systémem

Základní firmware popsaný v předchozí kapitole byl rozšířen také o FreeRTOS úlohu, která představuje uživatelskou aplikaci. Tato aplikace využívá API volání jednotlivých programových modulů a následně odesílá provozní data pomocí modulu `SlipSerial` nadřazenému systému. Pomocí tohoto modulu jsou zpracovávány i požadavky nadřazeného systému.

V uživatelské úloze se cyklicky čeká na novou periodu regulátoru. S novou periodou regulátoru jsou vytvářeny pakety hodnoty jednotlivých jednotlivých HW částí.

Kód uživatelské aplikace i externí funkce SLIP protokolu jsou umístěny v souboru `main.c`

7.3 Testovací aplikace pro PC

Pro komunikaci s ukázkovým firmware robota byla vytvořena jednoduchá grafická aplikace v jazyce C#. Její funkce odpovídají možnostem komunikačních paketů. Tato aplikace umožňuje:

- ukládání a vizualizace dat měřených robotem
- zobrazení a nastavení režimu řízení motorů
- manuální ovládání rychlosti motorů
- nastavování parametrů regulátoru
- zobrazení trasy vypočítané pomocí odometrie kol
- export dat jako script pro Matlab
- zobrazení výpisu úloh FreeRTOS
- zobrazení identifikovaných parametrů
- automatický režim jízdy po dráze tvaru číslice 8

Většina kódu je součástí třídy hlavního okna. Obrazovka hlavního okna je zobrazena na obrázku B.1. Pro větší přehlednost byly vytvořeny další třídy. Třída

`SlipSerial.cs` zajišťuje sériovou komunikaci společně s kódováním a dekódováním paketů. Třída `Robot.cs` slouží k ukládání naměřených dat. Třída `Charts.cs` vytváří okno pro zobrazování grafů pomocí volně dostupné komponenty `ZendGraph`. Snímek obrazovky okna pro zobrazení grafů je na obrázku B.2. Třída `System1st.cs` realizuje soustavu prvního řádu pro ovládání pohybu robota pomocí šipek. Automatický režim jízdy po dráze tvaru číslice 8 je realizován proporciálním regulátorem úhlu k cílovému bodu. V tomto režimu je rychlost dopředu konstantní.

7.4 Analýza testovací aplikace

Pomocí funkce `vTaskList()` byl získán přehled běžících úloh v systému společně s hodnotou aktuálně využitého zásobníku každé úlohy. Následuje exportovaný výpis úloh:

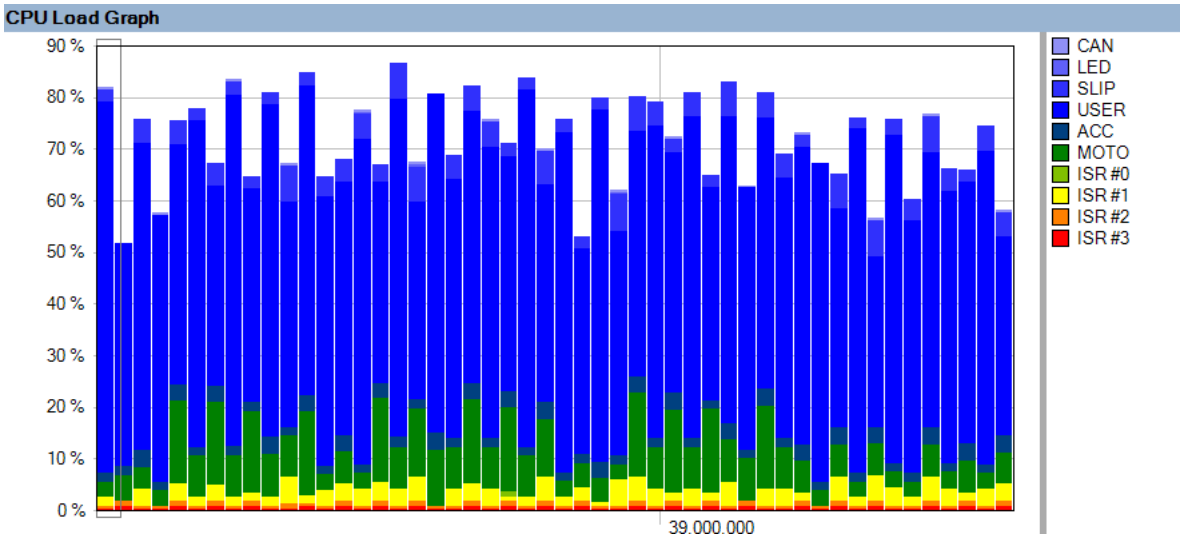
```
Název       Stav    Priorita    Zásobník    Identifikátor
USER        R       2           279         4
IDLE        R       0           105         6
MOTO        B       4           209         2
ACC         B       3           209         3
SLIP        B       2           207         1
LED         B       1           217         0
CAN         B       1           199         5
```

Pomocí nástroje FreeRTOS+Trace byly získány údaje o využití času procesoru vzhledem k frekvenci. Z výsledků uvedených v tabulce 7.3 vyplývá, že velké procento času procesor čeká na vstupně/výstupní operace.

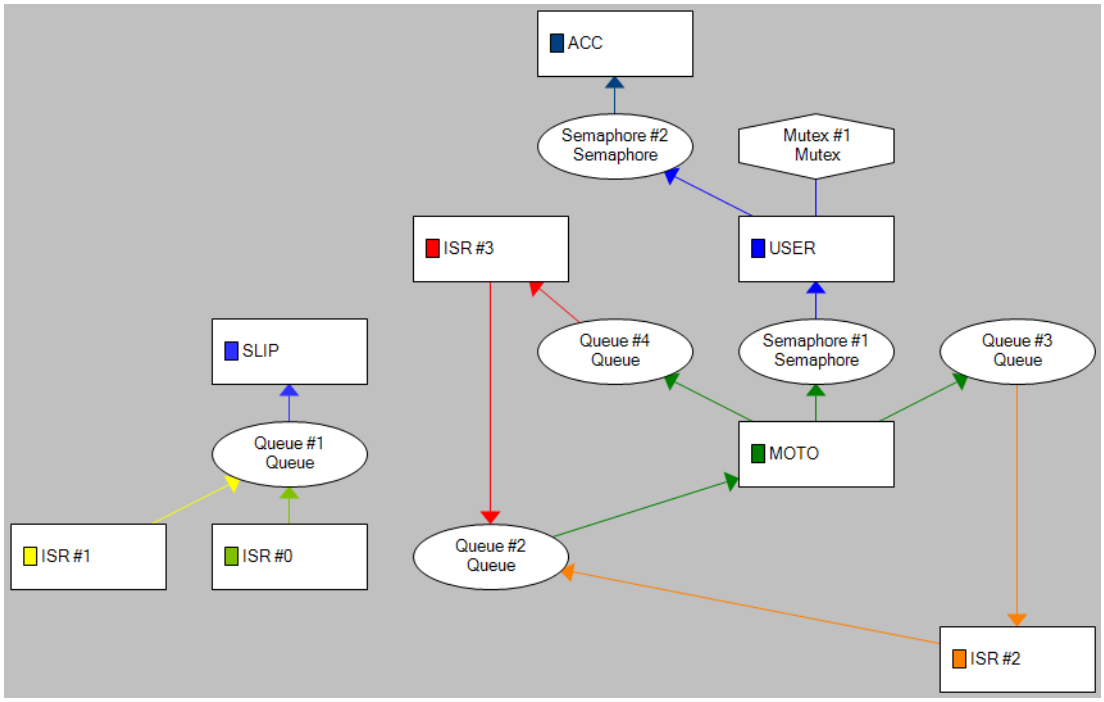
Tab. 7.3: Využití CPU z závislosti na frekvenci CPU

frekvence CPU	využití CPU
10 MHz	92%
20 MHz	72%
40 MHz	64%
50 MHz	63%

Využití procesoru jednotlivými úlohami je patrné z obrázku 7.2. Tok dat mezi komunikačními objekty FreeRTOS je na obrázku 7.3. Soubory `*.bin` s obrazy paměti RAM pro různé verze kódu je součástí přílohy CD v adresáři freertos+trace. V tomto adresáři je také umístěna statistika všech úloh, generovaná programem FreeRTOS+Trace.



Obr. 7.2: Zobrazení grafu využití procesorového času pro 20 MHz



Obr. 7.3: Zobrazení toku dat mezi komunikačními objekty FreeRTOS

8 ZÁVĚR

V teoretické části byly shrnuty důležité vlastnosti architektury ARM Cortex-M3 a byly zde uvedeny informace o operačních systémech reálného času, na které navázal přehled dostupných řešení na trhu. Byl proveden přehled dostupných vývojových nástrojů. Byly zmíněny také teoretické informace o podvozku, jeho vlastnostech a regulaci rychlosti pohybu. V části popisující použitou HW platformu byla nastíněna funkce jednotlivých částí řídicí desky. V tomto popisu jsou uvedena také omezení, která vycházejí z návrhu.

Na počátku práce byla osazena dodaná deska plošných spojů. Následně byla oživena a byla otestována komunikace s mikrokontrolérem pomocí JTAG CoLinkEx rozhraní.

Významná část práce byla věnována výběru, měření a porovnávání 3 operačních systémů reálného času. Časově náročná byla fáze založení projektů s testovanými operačními systémy a realizace vlastních testů. Dle výsledku porovnání doby přepnutí kontextu s čekáním na synchronizační objekt a dle výsledku porovnání SW časovačů byl pro další práci zvolen FreeRTOS. Operační systémy a testovací programy byly nastaveny na stejnou konfiguraci a byly přeloženy stejným překladačem v prostředí CoIDE. Pro měření času byl využit externí referenční generátor. Tím byla snížena nepřesnost měření na minimum.

Hlavní část praktické práce byla věnována vytvoření firmware, který tvoří pomyslný základní stavební kámen pro různé konkrétní použití robotického podvozku v budoucích projektech. Základní firmware je tvořen oddělenými programovými moduly. Tento firmware byl vystavěn na knihovnách pro přístup k HW mikrokontroléru *Stellaris Driver Library* a operačním systému reálného času FreeRTOS. Klíčovým modulem je modul pro regulaci rychlosti pohybu podvozku. Regulace rychlosti kol je řešena diskrétním PI regulátorem modifikovaným pro praktické použití.

API rozhraní souboru základních programových modulů bylo zdokumentováno pomocí systému Doxygen pro generování dokumentace, jak je běžné v technické praxi. Kompletní popis API je součástí přílohy na CD.

Všechny programové moduly byly otestovány pomocí ukázkové aplikace. Tato aplikace se sestává z úlohy ve firmware robotu a C# aplikace pro PC. Pro firmware byl implementován protokol pro komunikaci sériovou linkou pomocí bezdrátového Bluetooth modulu. C# aplikace byla koncipována tak, aby umožňovala komfortně manuálně ovládat pohyb robota a umožňovala zobrazovat provozní data v podobě grafů a nastavovat parametry robota. Data je také možné exportovat pro další zpracování v programu Matlab. Součástí ukázkové aplikace je také jednoduchý automatický režim, ve kterém podvozek robota sleduje dráhu tvaru číslice 8.

Cílem diplomové práce bylo vytvořit firmware, který bude připraven pro použití

v dalších pracích využívající tuto robotickou platformu. Tento cíl byl dosažen s využitím operačního systému reálného času FreeRTOS, který byl vybrán na základě měření. API firmwaru bylo zdokumentováno. Byla dokonce vytvořena i ukázková aplikace, která slouží k ovládání a získávání dat z podvozku. Další vývoj by se mohl zabývat modifikací regulačního algoritmu pohonů. Výsledky DP lze v praxi využít pro řídicí a vestavné systémy podobné koncepce.

LITERATURA

[1] ARM Ltd. *Cortex-M3 Series Infromation* [online]. 2012, [cit. 12. 4. 2012]. <http://www.arm.com/products/processors/cortex-m/index.php>.

[2] ARM Ltd. *Cortex-M3 Technical Reference Manual*. 2010.

[3] ST MCD Application Team. *Cortex-M3 Training*. Místo: ST Microelectronics. červen 2008.

[4] Levy M. *The History of The ARM Architecture*: From Inception to IPO. Convergence Promotions: 2004.

[5] Yiu J. *The Definitive Guide to the ARM Cortex-M3, Second Edition*. Burlington: Newnes, 2009.

[6] uC Simply. *Cortex-M3 z pohledu programátora* [online]. 2012, [cit. 12. 3. 2012]. <http://www.ucsimply.cz/cm3/>.

[7] Kučera P. *MRTS:* Přednášky [online]. 2011, [cit. 12. 3. 2012]. Dostupné z URL: <http://taceo.eu/mrts.php>.

[8] Li Q., Yao C. *Real-time Concepts for Embedded Systems*. San Francisco: CMP Books, 2003.

[9] CooCox. *CoOS User's Guide*. 2009.

[10] Barry R. *FreeRTOS API Reference* [online]. 2010, [cit. 24.4.2012]. Dostupné z URL: <http://www.freertos.org/a00106.html>.

[11] Percepio AB. *FreeRTOS+Trace Documentation - Recorder Library* <http://percepio.com/docs/manual/Recorder.html>.

[12] Labrosse J. *uC/OS-III*: The Real-Time Kernel. Weston: Micrium Press, 2010.

[13] Texas Instruments Incorporated. *Stellaris® Peripheral Driver Library*. SW-DRL-UG-8555. 2012.

[14] Tišnovský P. *Co se děje v počítači:* Instrukční sada Thumb-2 u mikroprocesorů ARM [online]. 2012, poslední aktualizace 10. 4. 2012 [cit. 24.4.2012] Dostupné z URL: <http://www.root.cz/clanky/instrukcni-sada-thumb-2-u-mikroprocesoru-arm/>.

[15] Native Instruments. *Aliasing and Sampling at Frequencies Above the Nyquist Frequency* [online]. 2006, [cit. 12. 3. 2012]. Dostupné z URL: <http://zone.ni.com/devzone/cda/tut/p/id/3000>.

[16] Agilent Technologies. *Agilent 33120A 15 MHz Function / Arbitrary Waveform Generator:* User's Guide. 2002. Publication Number 33120-90006.

[17] Kučera P. *Dokumetace HW robotického vozítka*. 2010. Brno.

[18] Texas Instruments Incorporated. *Stellaris® LM3S8962 Microcontroller:* Datasheet. 2011.

[19] Memsic. *MXC6202x:* Ultra Low Cost, $\pm 2.0g$ Dual Axis Accelerometer With I^2C Interface. 2007.

[20] Romkey J. *A NONSTANDARD FOR TRANSMISSION OF IP DATAGRAMS OVER SERIAL LINES*. Network Working Group. červen 1988.

[21] Šolc F., Hrabec J., GreplR. *Modelling of Fast Differentially Driven Mobile Robot*. Vysoké učení technické v Brně, 2010.

[22] Skalický J., Patočka M., Feiler Z. *Elektrické pohony a výkonová elektronika*. Brno. Vysoké učení technické v Brně, 2006. ISBN: 80-214-3286-1.

[23] Blaha P. *Modelování a identifikace:* Přednáškové slide. Vysoké učení technické v Brně, 2012.

[24] Pivoňka P. *Číslicová řídicí technika*. Vysoké učení technické v Brně, 2003.

[25] Bobál V., Böhm J, Prokop R., Fessl J. *Praktické aspekty samočinně se nastavujících regulátorů*. Brno. Vysoké učení technické v Brně, 1999. ISBN: 80-214-1299-2.

[26] Cach P., Fiedler P. *Průmyslové komunikační sítě a sítě v automobilovém průmyslu*. Fiedler S.I. s.r.o., Brno, 2001.

[27] Freescale Semiconductor. *MC33887:* 5.0 A H-Bridge with Load Current Feedback. 2011.

[28] FAULHABER. *DC-Motors:* Precious Metal Commutation. 2008.

[29] Chamberlain F., Red Hat Support. *The Red Hat newlib C Library manual*. Red Hat Inc.: prosinec 2010

[30] CooCox. *CooIDE User manual* [online]. 2011, [cit. 12. 3. 2012]. Dostupné z URL: <http://www.coocox.org/CooCox_CoIDE.htm>.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

ADC	Analog-Digital Converter - analogově-číslicový převodník
APB	Advanced Peripheral Bus - sběrnice pokročilých periferií
ASCII	American Standard Code for Information Interchange - americký standardní kód pro výměnu informací
BSP	Board Supporting Package - balík podpory pro vývojovou nebo aplikační desku
CAN	Controller Area Network - sběrnice pro komunikaci lokálních řídicích jednotek
CMSIS	Cortex Microcontroller Software Interface Standard - Standardní softwarové rozhraní pro mikrokontroléry ARM Cortex-M
DC	Direc Current - stejnosměrný proud
DPS	Deska Plošných Spojů
GCC	GNU Compiler Collection - sada kompilátorů projektu GNU
GPL	General Public Licence - obecná veřejná licence
GND	Ground - uzemnění
GPIO	General Purpose Input/Output - vstupně/výstupní pro všeobecné použití
HAL	Hardware Abstaction Layer - vrstva abstraktně popisující hardware
HW	HardWare - pevné hmatatelné části
IDE	Integred Development Enviroment - integrované vývojové prostředí
I^2C	Inter-Integred Cirtuit bus - vnitřní sběrnice mezi integrovanými obvody
ITM	Instrumentation Trace Microcell - jednotka pokročilého trasování
IO	InputOutput - vstupně výstupní
JTAG	Joint Test Action Group - rozhranní pro testování integravaných obvodů
LSB	Least Significant Bit - nejméně výnamný bit

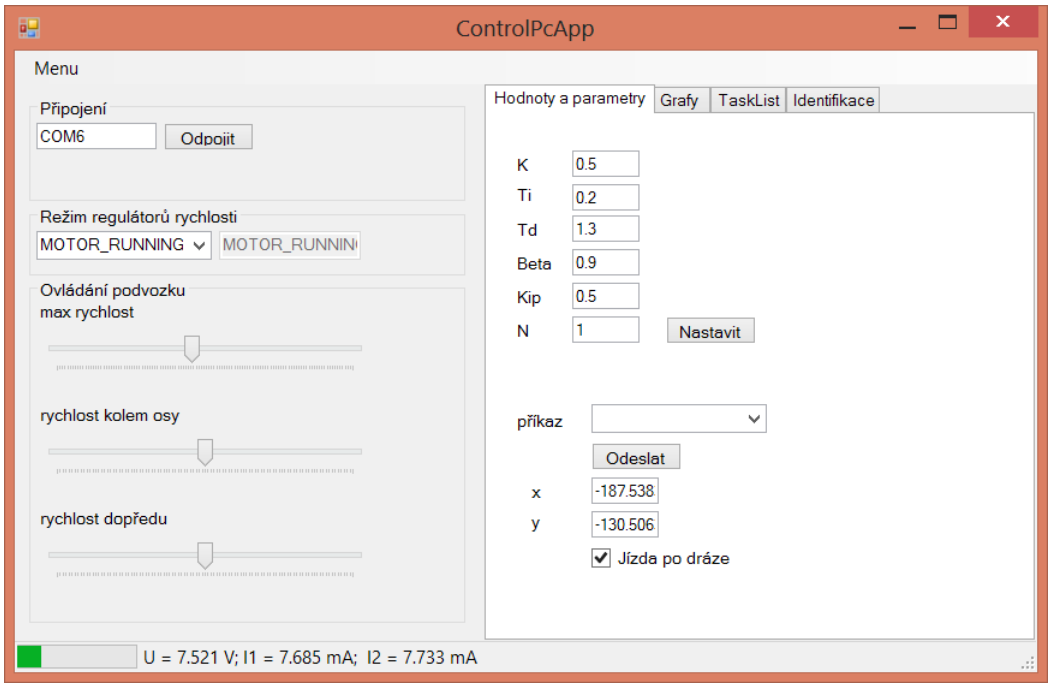
MEMS	Micro-Electro-Mechanical Systems - elektromechanický systém s rozměry v řádu mikrometrů
MOSFET	Metal Oxide Semiconductor Field Effect Transistor - tranzistor řízený polem s vrstvou kov-oxid-polovodič
MPU	Memory protection unit - jednotka pro ochranu paměti
MSB	Most Significant Bit - nejvíce výnamný bit
NVIC	Nested Vector Interrupt Controller - Jednotka vhnízděného přerušení
OS	Operating system - Operační systém
PLL	Phase Locked Loop - fázový závěs
PWM	Pulse Width Modulation - pulsně šířková modulace
QEI	Quadrature Encoder Interface - rozhraní inkrementálního kvadraturního enkodéru
RISC	Reduced instruction set computer - počítač s redukovanou sadou instrukcí
RTOS	Real-time Operating system - operační systém reálného času
SLIP	Serial Line Internet Protokol
SPI	Serial Peripheral Interface - rozhraní sériových přídavných zařízení
SSI	Synchronous Serial Interface - synchronní sériové rozhraní
SRAM	Static Random Access Memory - statická paměť s libovolným přístupem
SW	SoftWare nehmatatelné části - programové vybavení
SWD	Serial Wire Debug - sériové ladící rozhraní s minimem potřebných vodičů
TTL	Transistor-Transistor Logic - tranzistorově-tranzistorová logika
UART	Universal Asynchronous Receiver/Transmitter - univerzální asynchronní přijímač/vysílač
f	frekvence $[Hz]$
i	elektrický proud $[A]$

J	moment setrvačnosti $[kg.m^{-2}]$
m	hmotnost $[kg]$
u	elektrické napětí $[V]$
t	čas $[s]$
v	rychlost $[m.s^{-1}]$
ω	úhlová rychlost $[rad.s^{-1}]$

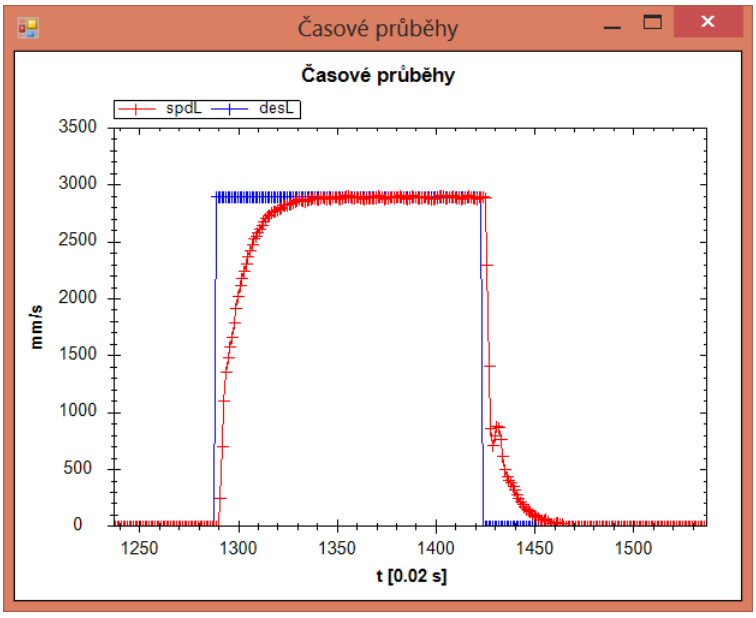
A ADRESÁŘOVÁ STRUKTURA PŘILOŽENÝCH SOUBORŮ

```
+---code-ControlPcApp
+---code-robotic-vehicle-firmware
+---code-RTOSTests
|   +---contextSW&Mutex
|   |   +---CoOS
|   |   +---FreeRTOS
|   |   +---uCOSIII
|   +---contextSW&Queue
|   |   +---CoOS
|   |   +---FreeRTOS
|   |   +---uCOSIII
|   +---timingAbs
|   |   +---CoOS
|   |   +---FreeRTOS
|   |   +---uCOSIII
|   +---timingRel
|       +---CoOS
|       +---FreeRTOS
|       +---uCOSIII
+---documentation
+---freertos+trace
+---identification
+---measured-data
|   +---contextSW&Mutex
|   +---contextSW&Queue
|   +---timingAbs
|   +---timingRel
+---simulation
```

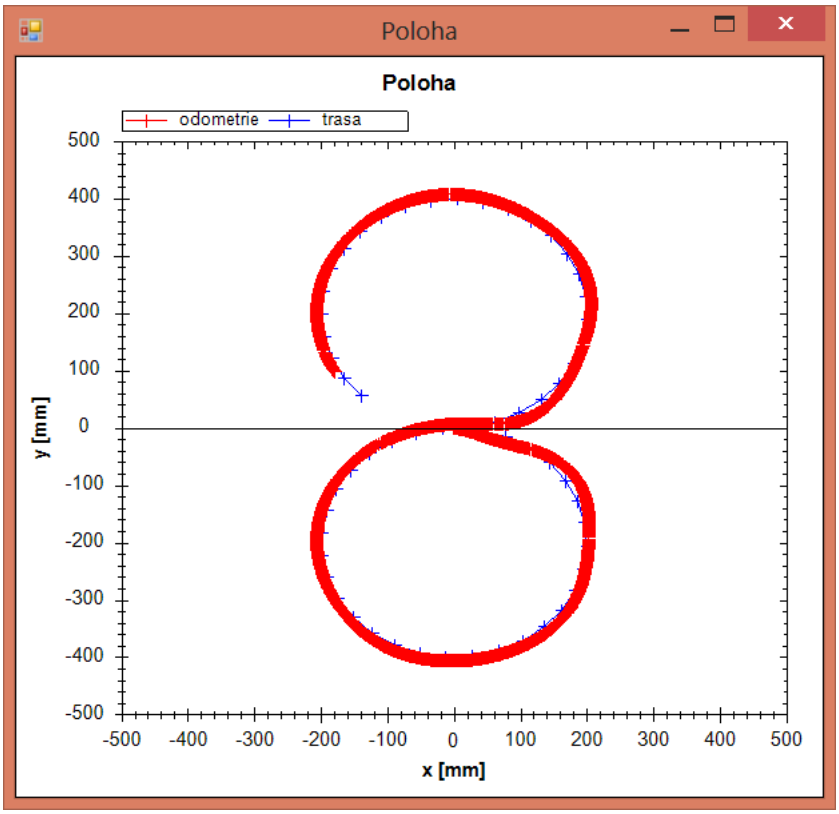
B SNÍMKY OBRAZOVKY C# APLIKACE



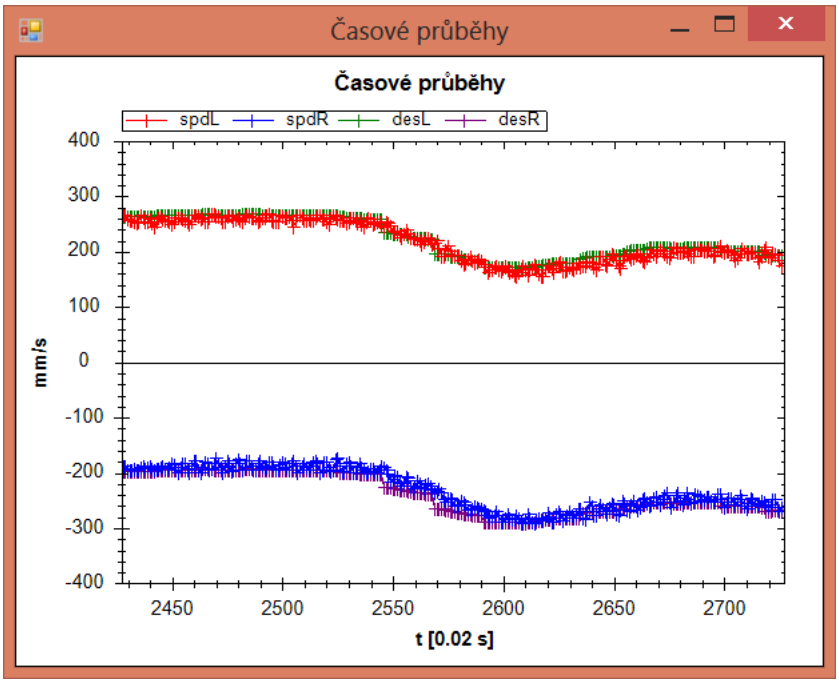
Obr. B.1: Hlavní okno testovací aplikace pro PC



Obr. B.2: Zobrazení průběhu rychlosti v okně grafů

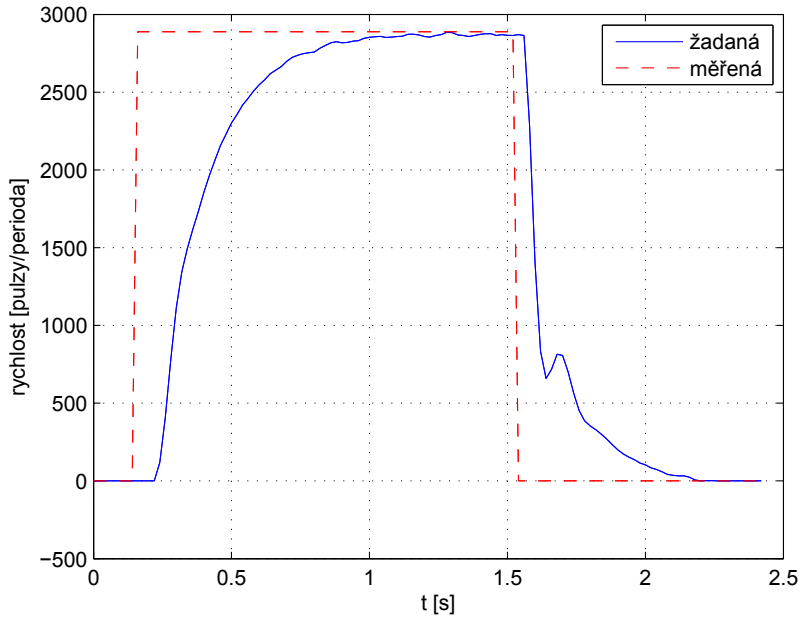


Obr. B.3: Zobrazení průběhu polohy v při jízdě po dráze tvaru 8

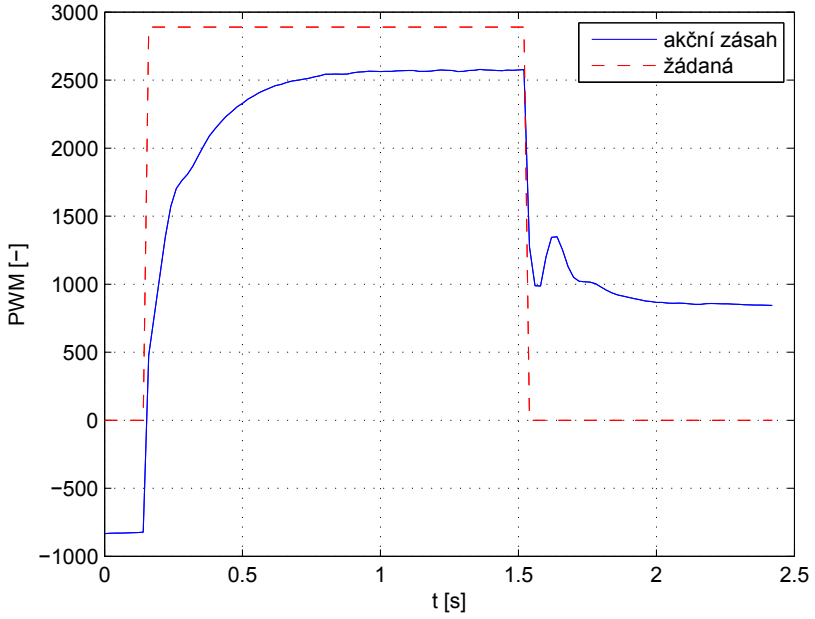


Obr. B.4: Zobrazení průběhu rychlostí v při jízdě dráze tvaru 8

C PRŮBĚHY REGULAČNÍHO POCHODU



Obr. C.1: Odezva rychlosti na jednotkový skok (kola volně)



Obr. C.2: Akční zásah pro jednotkový skok rychlosti (kola volně)