VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY
DEPARTMENT OF CONTROL AND INSTRUMENTATION

VYUŽITÍ SYSTÉMU RASPBERRY PI PRO ŘÍZENÍ.
CONTROL SYSTEM WITH RASPBERY PI

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Michal Zgrebňák**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Tomáš Macho, Ph.D.**
SUPERVISOR

BRNO 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Kybernetika, automatizace a měření**
Ústav automatizace a měřicí techniky

Student: Bc. Michal Zgrebňák ***ID:*** 158272
Ročník: 2 ***Akademický rok:*** 2017/18

NÁZEV TÉMATU:

Využití systému Raspberry PI pro řízení.

POKYNY PRO VYPRACOVÁNÍ:

1. Seznamte se se systémem Raspbery PI a principy OS reálného času.
2. Proveďte literární průzkum a popište OS, které lze portovat na Raspbery PI. Diskutujte vhodnost těchto OS pro využití v řídicí technice.
3. Rozeberte možnosti implementace PSD regulátorů s využitím OS na platformě Raspbery PI. Stanovte, jaké nejkratší periody vzorkování lze dosáhnout.
4. V případě potřeby navrhněte přizpůsobovací obvody, které by umožnily připojit alespoň 4 analogové vstupy a 4 analogové výstupy k systému Raspbery PI. Uvažujte vstupní a výstupní napětí v rozsahu 0 - 10 V.
5. Na základě literárního průzkumu z bodu 2 zvolte 2 operační systémy a implementujte do nich PSD regulační algoritmy. Uvažujte 4 PSD regulátory pracující souběžně.
6. Ověřte činnost regulátorů, zjistěte skutečnou dobu běhu regulačních algoritmů, případně její rozptyl, stanovte minimální dobu vzorkování. Diskutujte dosažené výsledky.

DOPORUČENÁ LITERATURA:

Joseph M, Real-time systems: specification, verification and analysis. London. Prentice-Hall, 1996.

Termín zadání: 5. 2. 2018 ***Termín odevzdání:*** 14.5.2018

Vedoucí práce: Ing. Tomáš Macho, Ph.D.
Konzultant:

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

Cieľom diplomovej práce je overiť praktickú využiteľnosť platformy Raspberry Pi v riadiacich aplikáciách. Práca sa skladá z výberu vhodného operačného systému a následnej implementácie diskrétneho PID algoritmu. Dôležitou súčasťou práce bola modifikácia OS Linux a jeho kompilácia. Meraním bola dokázaná využiteľnosť platformy pre riadiace aplikácie. Výsledkom práce je diskrétny PID regulátor realizovaný ako modul jadra OS Linuxu. Riešenie taktiež obsahuje webové rozhranie pre účely komunikácie medzi človekom a strojom.

KĽÚČOVÉ SLOVÁ

Raspberry Pi, PSD regulátor, Linux

ABSTRACT

The goal of this diploma thesis is to verify the practical applicability of the Raspberry Pi platform in control applications. The work consists of choosing a suitable operating system and implementing a discrete PID algorithm. An important part of the work was the Linux OS modification and compilation. Measurement has demonstrated the usability of the platform in control applications. The result of this work is a discreet PID controller implemented as a Linux kernel module. The solution also includes a web interface as a human-machine interface.

KEYWORDS

Raspberry Pi, discrete PID regulator, Linux

ZGREBŇÁK, Michal. *Využití systému Raspberry PI pro řízení*. Brno, 2018, 70 s. Diplomová práca. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedúci práce: Ing. Tomáš Macho, Ph.D.

VYHLÁSENIE

Vyhlasujem, že som svoju diplomovú prácu na tému „Využití systému Raspberry PI pro řízení“ vypracoval samostatne pod vedením vedúceho diplomovej práce, využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej diplomovej práce ďalej vyhlasujem, že v súvislosti s vytvorením tejto diplomovej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomý následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona Českej republiky č. 121/2000 Sb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon), v znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhej, hlavy VI. diel 4 Trestného zákoníka Českej republiky č. 40/2009 Sb.

Brno ....

podpis autora

Obsah

Úvod **9**

1 Analýza RTOS a Raspberry PI **10**
1.1 Multitasking . . . 10
1.2 Latencia . . . 10
1.3 Operačný systém reálneho času . . . 11
1.4 Raspberry Pi . . . 12

2 Operačné systémy portovateľné na Raspberry Pi **14**
2.1 Realtimové vlastnosti nemodifikovaného Linuxového jadra . . . 14
2.2 PREEMPT RT . . . 15
2.3 Xenomai . . . 15
2.4 Raspbian . . . 17
2.5 RISC OS . . . 18
2.6 ChibiOS . . . 18
2.7 FreeRTOS . . . 18
2.8 BitThunder . . . 20
2.9 RTEMS . . . 20

3 Špecifikácia požiadaviek a návrh regulátora **22**
3.1 PSD regulátor . . . 22
3.1.1 Antiwindup . . . 22
3.1.2 Beznárazové prepínanie . . . 23
3.1.3 Filtrácia derivačnej zložky . . . 23
3.2 Rozbor možných riešení . . . 24
3.2.1 Kritériá pre výber prevodníkov . . . 24
3.2.2 Najkratšia perióda vzorkovania . . . 25
3.3 Návrh regulátora . . . 26

4 Hardvérová špecifikácia **28**
4.1 40-pinový konektor . . . 28
4.2 AD prevodník . . . 28
4.3 3-stavový oddelovač zbernice . . . 29
4.4 DA prevodník . . . 29

5 Softvérová implementácia **30**
5.1 Základné funkcie modulu jadra . . . 30
5.2 Komunikácia modulu jadra s užívateľským priestorom . . . 31

5.2.1 Znakové zariadenia . . . 31
5.2.2 Sysfs . . . 33
5.3 HR časovač . . . 34
5.4 Kthread . . . 37
5.5 GPIO rozhranie . . . 39
5.6 AD, DA prevodník . . . 40
5.7 PSD algoritmus . . . 41
5.8 Rozhranie človek-stroj . . . 43

6 Overenie korektnej činnosti regulátora 44
6.1 Softvérové merania časových úsekov . . . 44
6.1.1 Meranie doby sleep a delay . . . 44
6.1.2 Meranie doby PSD algoritmu . . . 45
6.1.3 Meranie doby PSD algoritmu s AD a DA prevodníkmi . . . 46
6.2 Meranie prechodovej funkcie . . . 46

7 Výsledky práce 48
7.1 Hardvérové riešenie . . . 48
7.2 Softvérové riešenie . . . 48
7.3 Výsledky meraní . . . 49

8 Záver 50

Literatúra 51

Zoznam symbolov, veličín a skratiek 56

Zoznam príloh 57

A Inštalácia a kompilácia OS Raspbian 58
A.1 Inštalácia nemodifikovaného OS Raspbian . . . 58
A.2 Kompilácia linuxového jadra . . . 59
A.2.1 Zakázanie FIQ . . . 63
A.3 Testovanie realtimových vlastností na patchovanom OS Linux . . . 63

B Vizualizácie 66

C Obsah priloženého CD 70

Zoznam obrázkov

1.1 Rapsberry Pi 2 model B [32] . . . 13
1.2 bloková schéma procesora Cortex-A7 MPCore [41] . . . 13
2.1 bloková schéma konceptu Xenomai 2 [22] . . . 16
2.2 bloková schéma konceptu Xenomai Cobalt [46] . . . 17
2.3 bloková schéma konceptu Xenomai Mercury [46] . . . 17
2.4 softvérové vrstvy FreeRTOS [42] . . . 19
2.5 architektúra RTEMS [39] . . . 21
3.1 bloková schéma PSD regulátora s filtráciou derivačnej zložky [25] . . . 23
3.2 bloková schéma softvérového konceptu . . . 26
4.1 40-pinový konektor [47] . . . 28
4.2 bloková schéma hardvérového zapojenia . . . 29
5.1 algoritmus funkcie executePID_sleep() . . . 36
5.2 algoritmus funkcie executePID_delay() . . . 38
A.1 Win32DiskImager . . . 58
A.2 povolenie SSH rozhrania menu 1 . . . 59
A.3 povolenie SSH rozhrania menu 2 . . . 59
A.4 SSH enable 2 . . . 62
A.5 SSH enable 2 . . . 62
A.6 výpis programu htop . . . 64
B.1 HMI webové rozhranie . . . 66
B.2 fyzická implementácia regulátora . . . 67
B.3 schéma zapojenia . . . 68
B.4 schéma simulácie v matlabe . . . 69

Zoznam tabuliek

6.1 Meranie doby sleep a delay 44
6.2 Meranie doby PSD algoritmu 45
6.3 Meranie doby PAD algoritmu s AD a DA prevodníkmi 46
A.1 Prehlad meraných latencií pre jednotlivé CPU 64

Úvod

Súčastný trend nástupu internetu vecí a embedded systémov otvára nové možnosti v oblasti výpočtovej techniky. Neustály vývoj stimuluje vytváranie účelového softvéru a podporuje čoraz nižšie ceny hardvéru. Do oblasti softvéru to prináša vývoj nových operačných systémov reálneho času. Toto sa dá využiť pre účely regulácie a riadenia.

Jedným z najznámejších jednodoskových počítačov vhodných pre embedded aplikácie je Raspberry Pi. Jeho hlavná výhoda oproti konvenčne využívaným platformám je nízka obstarávacia cena. Pôvodne bola platforma Raspberry Pi navrhnutá ako desktopový počítač určený pre výukové účely [26], avšak čoskoro bola využívaná aj pre embedded aplikácie.

Pri riešení praktickej časti diplomovej práce bolo potrebné primárne vybrať vhodný operačný systém. Pre Raspberry Pi je portovaných množstvo operačných systémov z ktorých je určitá časť vhodná pre dané účely. Mnoho z týchto portácií je nových, postrádajú dlhoročnú podporu. Ako operačný systém bol zvolený OS Linux a jeho verzia modifikovaná patchom PREEMPT RT. Regulátor bol realizovaný ako modul jadra. Hardvérový koncept práce je založený na externom analógovo-digitálnom a digitálno-analógovom prevodníku. Realtimové vlastnosti regulátora boli overené meraním. Dosiahnuté výsledky sú v závere práce diskutované.

1 Analýza RTOS a Raspberry PI

Kapitola v krátkosti opisuje vlastnosti systémov reálneho času a platformu Raspberry PI. V úvode bolo potrebné definovať základé pojmy súvisiace s danou problematikou.

1.1 Multitasking

Pojem multitasking je spojený s operačným systémom ktorý je schopný vykonávať niekoľko procesov v rovnakom čase bez toho, aby sa tieto procesy navzájom ovplyvňovali. Každý jeden proces je vykonávaný akoby bol jediným procesom na počítači a mal exkluzívny prístup ku systémovým zdrojom operačného systému. Pretože CPU je schopné vykonávať v danom čase iba jeden proces, zdanie paralelizmu je zabezpečené pomocou časového multiplexovania [18].

Kooperatívny multitasking

Podstatou kooperatívneho, alebo aj nepreemptívneho multitaskingu je, že práve bežiaci proces odovzdá systémové zdroje až vtedy keď si to sám určí. Z toho vyplýva pomalá odozva systému na udalosti s vysokou prioritou. Ďalším znakom daného systému je nedeterministickosť z hľadiska časovo kritických udalostí [7]. Tieto vlastnosti sú nezlúčiteľné s požiadavkami na systém reálneho času.

Preemptívny multitasking

Preemptívny multitasking je založený na skutočnosti, že práve vykonávaný proces v CPU môže byť prerušený hardvérovým časovačom. Táto vlastnosť umožňuje prepnutie iného procesu bez toho aby sa predchádzajúci proces dobrovoľne vzdal CPU. Z toho vyplýva rýchla odozva na úlohy s vyššou prioritou a deterministické vlastnosti systému. Dané vlastnosti sú klúčové pre systém reálneho času [18] [7].

1.2 Latencia

Latencia prerušenia, nazývaná aj čas odozvy prerušenia, je časový úsek od momentu vzniku prerušenia po moment reakcie na prerušenie. Hlavnými faktormi ovplyvňujúcimi latenciu sú architektúra procesora, hodinová frekvencia a typ operačného systému [35].

1.3 Operačný systém reálneho času

Operačný systém reálneho času, angl. Real time operation system (RTOS) je preemptívnym multitaskingovým operačným systémom vytvoreným pre účely riadenia systémov reálneho času. RTOS musí byť deterministický. To znamená, že systém garantuje splnenie úloh do konečného termínu. Musí byť splnená podmienka predikcie výstupov pre každý možný stav vstupov. Ďalšou nutnou podmienkou je behaviorálnosť. RTOS musí byť navrhnutý s dôrazom na časovú minimalizáciu latencie prerušenia, angl. Interrupt latency a prepínania medzi jednotlivými procesmi, angl. Context switching [16].

Embedded system

Vnorený systém, angl. embedded system je výpočtové zariadenie, ktoré je súčasťou väčšieho celku [16].

Safety-critical system

Je systémom reálneho času, ktorý má v prípade zlyhania katastrofálne následky [16].

Hard RTOS

Hard RTOS garantuje, že dané úlohy reálneho času musia byť splnené do konečného termínu. Je využívaný v aplikáciách, kde nesplnenie úlohy do konečného termínu môže viesť k závažným škodám alebo k stratám na životoch. Z tohoto dôvodu je potrebné, aby výpočet prebehol za každých okolností [16].

Firm RTOS

Firm RTOS toleruje minimálny výskyt nedokončených výpočtov do konečného termínu. Avšak výskyt väčšieho počtu takýchto zlyhaní môže viesť k závažným škodám [16].

Soft RTOS

Občasná degradácia výpočtového výkonu je tolerovateľná. Výskyt viacerých zlyhaní má za následok zhoršenú kvalitu poskytovanej služby, avšak bez závažných následkov [16].

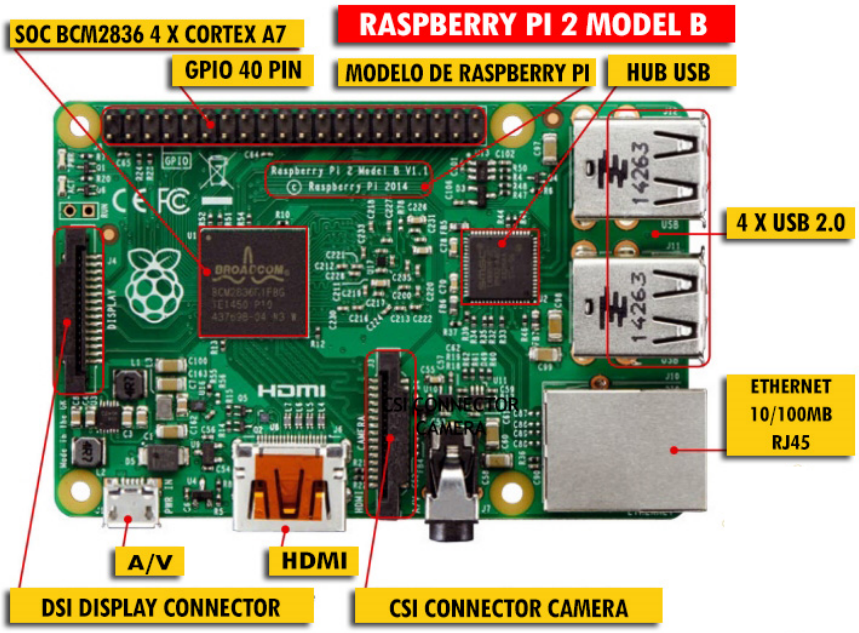
1.4 Raspberry Pi

Raspberry Pi je jednodoskový počítač veľkosti platobnej karty. Má široké uplatnenie v oblasti hobby elektrických projektov a konvenčných funkcií stolného počítača. Taktiež podporuje prehrávanie videa vo vysokom rozlíšení. Primárnym zámerom výrobcu, neziskovej charitatívnej organizácie z Veľkej Británie Raspberry Pi Foundation, je využitie Rapsberry Pi na výučbu programovania a digitálneho spracovania pre mladistvých [26].

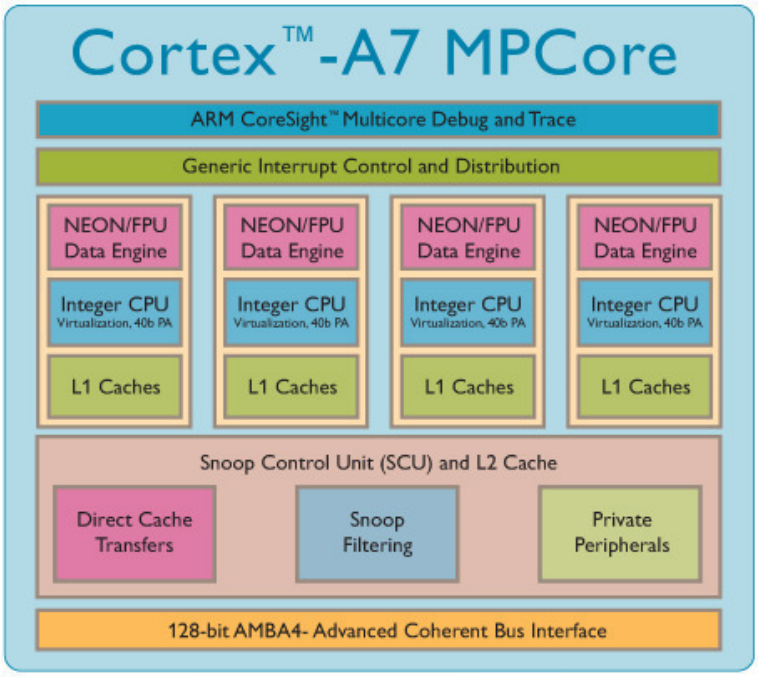
Práca pojednáva konkrétne o modeli Raspberry Pi 2 B, ktorý predstavuje druhú generáciu Raspberry Pi. Model nahradzuje Raspberry Pi 1 Model B+. Základ dosky tvorí integrovaný obvod BMC 2836 od spoločnosti Broadcom. Disponuje 900 Mhz štvorjadrovým procesorom Cortex-A7 MPCore 1.2, 1 GB DDR2 RAM a VideoCore IV 3D grafickým jadrom. Dostupné sú nasledujúce periférie, z ktorých je väčšina dostupných na 40-pinovom konektore [27] [8]:

- časovače
- radič prerušenia
- GPIO
- USB
- PCM / I2S
- DMA radič
- I2C master
- I2C / SPI slave
- SPI0, SPI1, SPI2
- PWM
- UART0, UART1

Doska ďalej disponuje štyrmi USB portami. Interface pre multimédiá zaisťuje Full HDMI port a štvorpólový 3.5mm jack pre stereo zvuk a kompozitné video. Permanentné dátové úložisko je realizované slotom pre Micro SD kartu. Pripojenie na sieť obstaráva 100 Mbs Ethernetový port. Pre ďalšie možnosti embedded aplikácií je na doske vyvedené aj sériové rozhranie pre kameru alebo display (CSI alebo DIS). Napájanie môže byť realizované pomocou micro USB (ktorý neslúži na prenos údajov), alebo pomocou 40-pinového konektora.



Obr. 1.1: Rapsberry Pi 2 model B [32]



Obr. 1.2: bloková schéma procesora Cortex-A7 MPCore [41]

2 Operačné systémy portovateľné na Raspberry Pi

Kapitola pojednáva o jednotlivých operačných systémoch portovateľných na Raspberri Pi. Dôraz je kladený na operačné systémy relevantné pre aplikácie reálneho času. Úvod kapitoly je zameraný na OS Linux, ktorého nemodifikované angl. vanilla jadro nie je realtimové. Preto sú najskôr opisované jednotlivé vylepšenia jadra ako patch PREEMPT RT a Xenomai. Následne je opisovaný Rapbsbian, ako typický predstaviteľ linuxovej rodiny pre platformu Raspberry Pi. Neskôr sú popisované operačné systémy, ktoré majú natívne realtimové jadro, z čoho vyplývajú aj lepšie vlastnosti v oblasti riadenia. Záporom týchto operačných systémov je, že neponúkajú takú pestrú škálu softvérových možností ako OS Linux.

2.1 Realtimové vlastnosti nemodifikovaného Linuxového jadra

Tradičné nemodifikované jadro dovolí prerušenie práve vykonávaného procesu tzv. preempciu iba za určitých okolností:

- keď je CPU v móde užívateľa
- keď sa kód jadra vráti zo systémového volania alebo z obsluhy prerušenia do módu užívateľa
- keď sa kód jadra zablokuje na mutexe, alebo explicitne prenechá CPU inému procesu (inštrukcia yeld)

Problém nastáva v prípade súčasného výskytu dvoch špecifických udalostí. Výskyt novej požiadavky, ktorá vyžaduje vykonávanie procesu s vysokou prioritou a udalosti keď je práve vykonávaný kód v režime jadra. V tomto prípade musí daný proces čakať, kým sa dokončí kód jadra, alebo sa jadro nevzdá CPU. V najhoršom prípade môže latencia systému dosiahnuť stovky milisekúnd alebo aj viac.

Lepšie latentné vlastnosti v Linuxovom jadre vo verzii 2.6 a vyššej je možné dosiahnuť pomocou nastavenia CONFIG_PREEMPT_VOLUNTARY. Dané nastavenie obmedzuje dlhé latencie tak, že jadro sa v príhodných častiach kódu môže vzdať CPU.

Ďalšie nastavenie, ktoré pomáha zmenšiť latenciu systému je CONFIG_PREEMPT.

Toto nastavenie zabezpečí, že jadro mimo regiónov ošetrených spinlokmi a IRQ je schopné sa vzdať CPU v prospech procesu s vyššou prioritou. S týmto nastavením môže latencia klesnúť rádovo na jednotky milisekúnd. Napriek tomu môže byť latencia značne vyššia[14].

2.2 PREEMPT RT

PREEMPT_RT je Linuxový patch, ktorý má za úlohu modifikovať OS Linux na systém reálneho času. PREEMPT_RT patch modifikuje jadro na plne preemptívne, vlastnosti RTOS sú zabezpečené vďaka[14]:

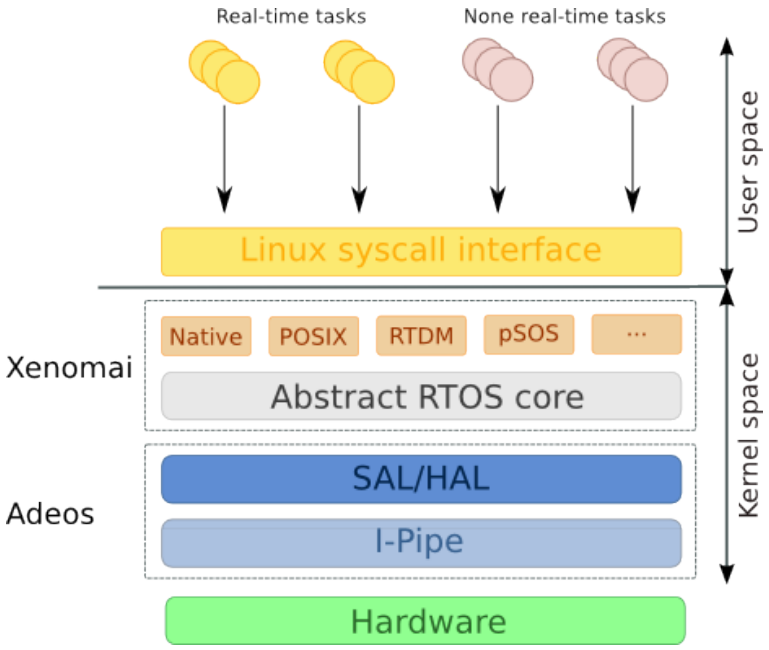
- zmene uzamykacích primitív v jadre, spinloky sú nahradené realtimovými mutexami
- kritické sekcie využívajúce spinlock_t a rwlock_t sú po aplikácii patcha plne preemptívne, vytváranie nepreemptívnych sekcií je stále možné použitím raw_spinlock_t
- implementovanie dedičnej priority pre spinloky a semafóry v jadre
- konvertovanie IRQ na preemptívne vlákna jadra
- konvertovanie starého API časovačov na nové API POSIX časovačov s veľkým rozlíšením

2.3 Xenomai

Xenomai je platforma spolupracujúca s jadrom OS Linux určená pre realtimové aplikácie. Súčasťou daného konceptu je aplikovanie patchu PREEMPT RT. Výhodou konceptu je, že v určitej konfigurácii poskytuje výrazne lepšie realtimové vlastnosti ako pôvodné jadro OS Linux. Ďalšou z výhod je možnosť emulácie POSIX nekompatibilných API ako sú VxWorks či pSOS. Xenomai je opensource platforma pod licenciou GPL v2.

Xenomai 2

Koncept platformy Xenomai je založený na vrstve Adeos. Adeos je virtualizačná vrstva systémových zdrojov dostupná ako linuxový patch. Vrstva Adeos poskytuje dokovanie viacerých entít-domén, ktoré sú zväčša operačnými systémami. Vrstva adeos zaisťuje aby boli úlohy realizované v správnom poradí. V praxi sú všetky požiadavky radené do fronty v závislosti od priority a domény. Toto radenie predstavuje abstrakciu I-pipeline viď. obr. 2.1.



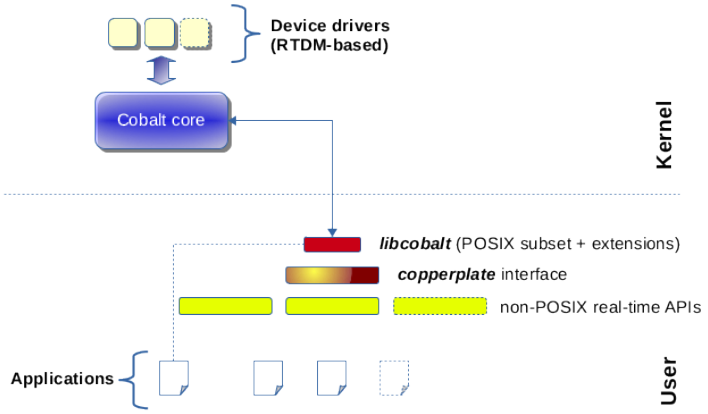
Obr. 2.1: bloková schéma konceptu Xenomai 2 [22]

Xenomai 3 Cobalt

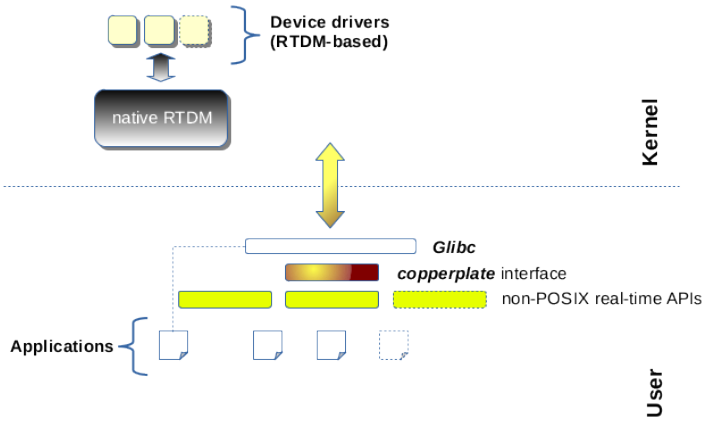
Verzia jadra Cobalt je evolúciou Xenomai verzie 2. Predstavuje koncept dvoch jadier bežiacich vedľa seba na jednom hardvéri. Hlavná zmena oproti verzii 2 spočíva v migrácii emulátorov API z priestoru jadra do užívateľského priestoru.

Xenomai 3 Mercury

Druhým typom jadra verzie Xenomai 3 je Mercury. Jedná sa o jednojadrový koncept, kde je jadro OS Linux vylepšené konceptom Mercury. Koncept Mercury spolieha na vlastnosti reálneho času pôvodného jadra OS Linux. V praxi je toto jadro zväčša vylepšené patchom PREEMPT RT. Hlavná výhoda danej koncepcie spočíva v emulácii POSIX nekompatibilných API reálneho času [45].



Obr. 2.2: bloková schéma konceptu Xenomai Cobalt [46]



Obr. 2.3: bloková schéma konceptu Xenomai Mercury [46]

2.4 Raspbian

Raspbian je voľne dostupný operačný systém založený na linuxovej distribúcii Debian. Jeho tvorcovia si kladú za cieľ, aby bol Raspbian primárnym operačným systémov portovaným na Raspberry Pi.

Raspbian je neoficiálny port Debianu, ktorý má snahu o čo najväčšiu podobnosť s Debianom. Daná filozofia má svoje opodstatnenie vo veľaročnej podpore, podrobnej dokumentácii a množstve rozširujúcich balíčkov. Raspbian podporuje hardvérové výpočty s pohyblivou desatinnou čiarkou. Pôvodný operačný systém oficiálne poskytovaný pre Raspberry Pi bola odľahčená verzia Debianu. Záporom tejto verzie bola chýbajúca podpora hardvérových výpočtov s pohyblivou desatinnou čiarkou.[9] [26].

2.5 RISC OS

RISC OS je plnohodnotný desktopový operačný systém špecializovaný na ARM architektúru. Bol vyvinutý v Cambridge spoločnosťou Acorn Computers tímom ľudí, ktorí navrhli architektúru ARM. [34] Risc OS má malé a rýchle jadro. Je podstatne jednoduchší ako moderné konvenčné operačné systémy, napr. OS Linux. Vyznačuje sa vysokou modularitou. Komunikácia medzi modulmi je podrobne dokumentovaná. Je pozoruhodné, že RISC OS s celým grafickým rozhraním môže mať veľkosť pod 6MB. RISC OS je jednoužívateľský systém, preto nie je vhodný pre aplikácie vyžadujúce vysoký stupeň bezpečnosti. RISC OS využíva kooperatívny multitasking, čo umožňuje ľahko vytvárať aplikácie vyžadujúce kontrolu nad celým operačným systémom. Príkladom môže byť komunikácia s hardvérom vyžadujúcim presné časovanie. Pre Raspberry Pi bola vydaná oficiálna distribúcia RISC OS Pi.[28]

2.6 ChibiOS

ChibiOS je nízkoúrovňový operačný systém určený špeciálne pre aplikácie reálneho času. ChibiOS sa podstatne líši od konvenčných operačných systémov. U konvenčných operačných systémov aplikácia a operačný systém bežia oddelene. Jadro je takto chránené pred možným poškodením od aplikácie. Koncept ChibiOS je založený na tom, že operačný systém a aplikácia tvoria jeden program. Daný koncept vyžaduje kompilovanie operačného systému spolu s aplikáciou. Jadro potrebuje iba cca 6KiB ROM a kompilácia trvá krátko[5].

ChibiOS je voľný software pod licenciou GPL3. Je jednoduchý na portovanie a podporuje preemptívny multitasking. Podporuje 128 úrovní priorít jednotlivých vlákien. Pre viac vlákien s rovnakou prioritou je realizované plánovanie typu Round-robin. Podporované sú programové primitívy ako vlákna, virtuálne časovače, semafory, mutexy, podmienené premenné, príznakové flagy, systémové správy, mailboxy a vstupno-výstupné fronty. Je dodávaný s PC simulátorom pod OS Windows alebo Linux. ChibiOS nepotrebuje pamäťový alokátor, všetky štruktúry jadra sú alokované deklaratívne, staticky. Poskytuje blokované a neblokované I/O fronty s timeoutom a generovaním udalostí[40].

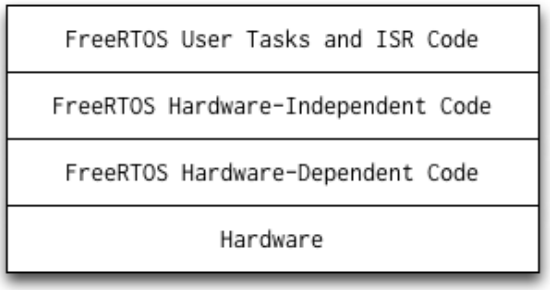
2.7 FreeRTOS

FreeRTOS je jeden z najpoužívanejších systémov reálneho času. Tento fakt podporuje aj odstránenie pravidiel všeobecne platných pre používanie slobodného softvéru

(GPL), čo umožňuje jeho použitie v komerčných aplikáciách bez požiadavky na zverejnenie proprietárneho kódu. Je vyvíjaný profesionálne spoločnosťou Real Time Engineers Ltd.

FreeRTOS je navrhovaný ako široko škálovateľný operačný systém. Môže byť skompilovaný ako úzko špecializovaný RTOS bežiaci na jednom CPU. Iným príkladom môže byť verzia určená pre mnohojadrový systém podporujúci protokol TCP/IP, súborový systém a USB rozhranie. Softvérová podpora obsahuje aj hĺbkový analyzér poskytujúci analýzu dynamického správania programu. Tieto informácie sú vysoko prínosné pri odladovaní, optimalizácii alebo analýzu výkonu kódu. Prednosťou FreeRTOS je jeho jednoduchosť. Jadro obsahuje iba tri zdrojové súbory jazyka C.

Koncept FreeRTOS je možné rozdeliť do troch vrstiev. Prvá vrstva obsluhuje hardvér. Druhá vrstva je nezávislá na hardvéri a tvorí základ operačného systému. Tretia vrstva obsahuje užívateľský program viď. obr. 2.4. FreeRTOS podporuje širokú škálu



Obr. 2.4: softvérové vrstvy FreeRTOS [42]

procesorov od 8 do 64 bitov a 35 rôznych architektúr. Podporuje širokú škálu prekladačov napr. CodeWarrior, GCC, IAR. Je nenáročný na hardvérové zdroje, typicky potrebuje ROM/Flash od 5 do 15KB a cca. 250B RAM. Taktiež podporuje tickless mód pre nízku spotrebu energie. FreeRTOS obsahuje nasledovné funkcionality [33] [12] [42]:

- preemptívne alebo kooperatívne plánovanie s možnosťou voľby
- mutexy s dedením priority
- softvérové časovače
- udalostné a skupinové príznaky
- detekcia pretečenia zásobníka
- semafóry a rekurzívne semafóry

- viac druhov alokácie pamäte

FreeRTOS je vydávaný v ďalších dvoch verziách OpenRTOS a SafeRTOS. OpenRTOS je komerčne licencovaná verzia poskytovaná pod licenciou Real Time Engineers Ltd. SafeRTOS je založený na rovnakom princípe ako pôvodný FreeRTOS, ale je vyvíjaný s ohľadom na rôzne medzinárodné bezpečnostné štandardy napr. SIL 3 [4].

2.8 BitThunder

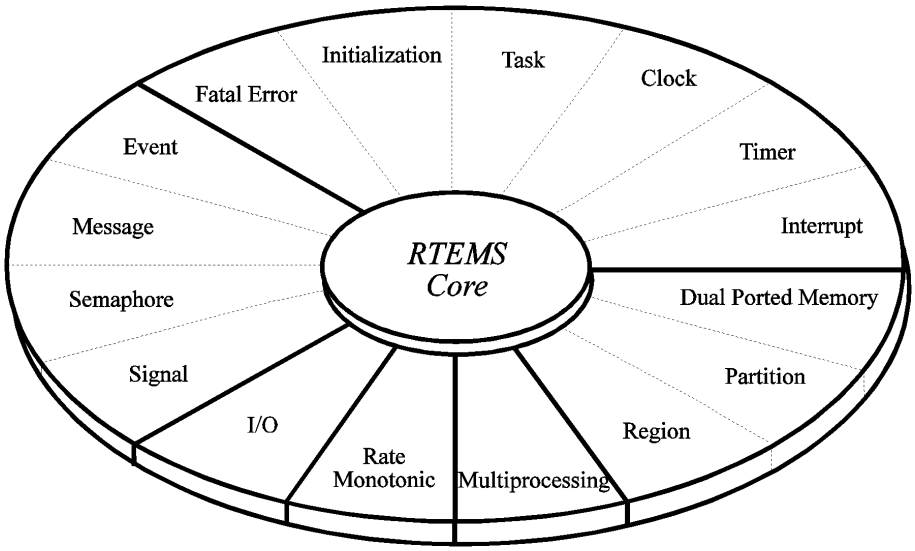
BitThunder je systémom reálneho času založený na plánovači FreeRtos. Je distribuovaný pod licenciou GPLv2. BitThunder pridáva do FreeRTOS viac ucelenú správu procesov a virtuálnej pamäte. Dôležitý význam BitThunderu je náhrada v aplikáciách, v ktorých by bolo pre ich komplexnosť potrebné využiť OS Linux. Náhrada OS Linux a zavádzača U-boot OS BitThunderom skráti čas bootovania a vylepší realtimové vlastnosti systému. Snaha vývojárov je urobiť BitThunder POSIX kompatibilný [44].

2.9 RTEMS

RTEMS angl. Real-Time Executive for Multiprocessor Systems je open-source operačný systém vyvíjaný spoločnosťou OAR. Je využívaný v letectve, armáde, medicíne, vesmírnych programoch a iných embedded aplikáciách. Je portovateľný na širokú škálu architektúr napr. ARM, PowerPC, Intel, Blackfin, MIPS, Microblaze a iné [36]. Keďže je POSIX kompatibilný, môže využívať širokú škálu GNU softvérového vybavenia, ako je napr. GDB, či podpora pre rôzne programovacie jazyky. Podporuje viacero štandardov z BSD vrátane TCP/IP a BSD soketov. Primárnymi programovacími jazykmi sú C, C++ a ADA95. RTEMS je široko škálovateľný. Maximálne odľahčená verzia môže mať veľkosť pod 20KB. Iným príkladom môže byť koncept s modulmi pre protokol TCP/IP alebo pre POSIX súborový systém. RTEMS obsahuje nasledovné funkcionality [37]:

- preemptívne plánovanie založené na priorite a udalostiach
- voliteľné RMS plánovanie
- mutexy s dedením priority
- dynamickú alokáciu pamäte
- vlákna kompatibilné s POSIX 1003.1b
- vnútroprocesovú komunikáciu a synchronizáciu

RTEMS je zložený z viacerých projektov, každá jeho časť spadá pod inú licenciu. Jeho primárna licencia je GPL 2, kde spadá väčšina systému [38].



Obr. 2.5: architektúra RTEMS [39]

3 Špecifikácia požiadaviek a návrh regulátora

Kapitola v úvode pojednáva o požiadavkách na PSD regulátor a na výber AD a DA prevodníkov. Následne je diskutovaná teoretická najkratšia možná doba vzorkovania. V závere je spomenutá konkrétna voľba implementácie regulátora.

3.1 PSD regulátor

PSD regulátor má predpis (3.1). Daný predpis vychádza z diskretizácie PID regulátora použitím náhrady integrálu obdĺžnikmi zľava. Takéto vyjadrenie PSD regulátora sa nazýva aj polohový algoritmus [25].

$$u(k) = K\left\{e(k) + \frac{T}{T_I}\sum_{i=1}^{k} e(k) + \frac{T_d}{T}[e(k) - e(k-1)]\right\} \tag{3.1}$$

k - index kroku
e(k) - regulačná odchýlka
u(k) - výstupá hodnota z regulátora - akčný zásah
K - zosilnenie regulátora
T - perióda vzorkovania
T_I - integračná konštanta
T_D - derivačná konštanta

Pri realizácii PSD regulátora je potrebné implementovať nasledovné:

- ošetrenie windup efektu
- beznárazové prepínanie
- filtráciu derivačnej zložky

3.1.1 Antiwindup

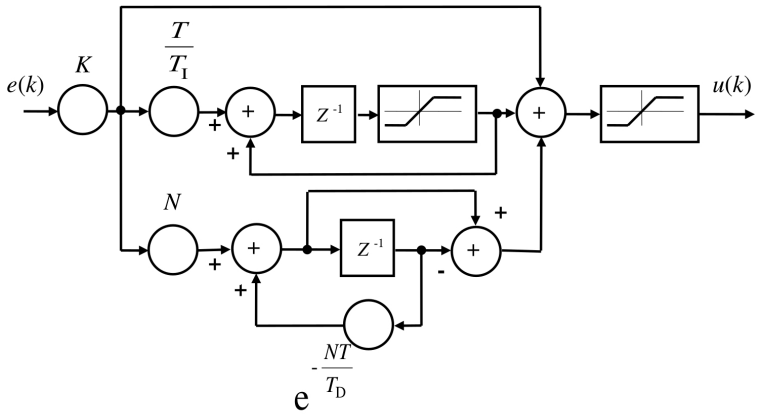
Windup je pomenovanie pre jav, keď sumačná zložka dosahuje hodnotu väčšiu, ako je maximálna hodnota akčného zásahu. Tento jav má za následok nepriaznivé predlžovanie prechodového deja. Z tohoto dôvodu je potrebné implementovať ochranu proti prebudeniu regulátora angl. antiwindup [25].

3.1.2 Beznárazové prepínanie

Pri prepínaní z manuálneho do automatického režimu môže nastať nežiaduci skok na priebehu akčného zásahu. Túto zmenu je potrebné obmedziť na technologicky prijateľnú hodnotu. Dosiahne sa to prepínaním výhradne v stave keď je regulačná odchýlka nulová [25].

3.1.3 Filtrácia derivačnej zložky

Nefiltrovaný vstup na PSD regulátore môže spôsobovať kmitavý priebeh akčného zásahu. Pre dobrú dynamickú odozvu a rýchle vyregulovanie poruchových signálov sa odporúča vzorkovacia frekvencia 6 až 20-krát väčšia ako je frekvencia najvyššej harmonickej na vstupe. Dodržanie danej podmienky pre brum a šum by vyžadovalo nereálne vysoké vzorkovanie. Najviac je ovplyvnená derivačná časť regulátora substituovaná nefiltrovanou prvou diferenciou. Jednoduchý a účinný filter derivačnej zložky je možné realizovať podľa obr. 3.1 [25].



Obr. 3.1: bloková schéma PSD regulátora s filtráciou derivačnej zložky [25]

3.2 Rozbor možných riešení

Pri riešení zadania práce je potrebné zohľadniť softvérovú aj hardvérovú časť implementácie.

Hardvérová časť pozostáva z vhodného výpočtového zariadenia, AD a DA prevodníkov a poprípade ďalších prispôsobovacích zariadení. Zadanie práce udáva ako výpočtovú platformu Raspberry Pi. Problém výberu AD a DA prevodníkov spočíva najmä v rýchlosti prevodu, použitej zbernici a rozlišovacej schopnosti, viac je popísané v kapitole 3.2.1. Otázka dodatočných prispôsobovacích obvodov je úzko spojená s výberom prevodníkov. Tieto obvody sú potrebné v prípade že Raspberry Pi a prevodníky pracujú s rôznymi napätiami na logických termináloch. Príkladom je AD prevodník pracujúci s logickou úrovňou 5v a Raspberry pi pracujúce s 3.3V úrovňou. Druhým prípadom potreby prispôsobovacích obvodov je prípad, keď AD a DA prevodníky nepodporujú na vstupe, resp. na výstupe zadané vstupné, resp. výstupné napätie, 0-10V.

Hlavná časť problematiky implementácie softvérovej časti spočíva vo výbere vhodného operačného systému. Operačný systém by mal byť systémom reálneho času, viď. kapitola 2, alebo by rýchlosť odozvy a deterministickosť mala byť zabezpečená iným nekonvenčným spôsobom. U jednoúčelových systémov reálneho času môže byť problematická implementácia dodatočných funkcionalít, napr. užívateľského rozhrania. U zložitejších operačných systémov je potrebné riešiť otázku implementácie softvéru v užívateľskom priestore alebo v module jadra. Výhodou implementácie v užívateľskom priestore je jednoduchosť. Kladom implementácie v module jadra je predovšetkým vyššia rýchlosť podporená priamym pristupovaním k systémovým zdrojom, napr. priamy prístup ku zbernici. Nezanedbateľná je aj dostupnosť, resp. podpora daných operačných systémov pre konkrétnu hardvérovú platformu.

3.2.1 Kritériá pre výber prevodníkov

Pri výbere prevodníka treba zohľadniť nasledovné kritériá:

- typ a rýchlosť zbernice
- počet rozlišovacích úrovní
- počet vstupov alebo výstupov
- rýchlosť prevodníka
- maximálne vstupné alebo výstupné napätie
- cenu a dostupnosť prevodníka

Zbernica môže byť paralelná, SPI, alebo I2C. Najrýchlejšia zbernica je paralelná. Pre zbernicu SPI sú na 40-pinovom konektore dva chip select piny. Zbernica I2C využíva adresovanie po zbernici so sedem bitovou adresou [29].

Počet požadovaných rozlišovacích úrovní je závislý na konkrétnej aplikácii. Prevodníky s väčším počtom rozlišovacích úrovní ako je šesnásť, či dvadsať bitov spravidla nie sú potrebné, pretože najmenej významné bity sú zväčša utopené v šume.

Otázka počtu vstupov resp. výstupov je jednoduchá. Pre zadanie práce je potrebné, aby prevodník obsahoval najmenej štyri vstupy resp. výstupy. Zadanie by bolo riešiteľné aj pre konfiguráciu viacerých prevodníkov s menej terminálmi. Toto by malo za následok komplikácie spojené s rýchlosťou a problémom komplikovanejšieho adresovania po zbernici.

Rýchlosť prevodníka závisí od konkrétneho typu prevodníka a od vyššie uvedených parametrov.

Maximálne vstupné alebo výstupné napätie je dôležité z hľadiska zadania napätí 0 - 10V. Ideálne by bolo, keby prevodník pracoval s danými napäťovými rozsahmi. V prípade že rozsahy napätí sú menšie, je možné prevodníky prispôsobiť pre dané parametre. Vstupy možno rozšíriť o odporový napäťový delič a výstupy o operačný zosilňovač pracujúci v neinvertujúcom zapojení.

Otázka ceny a dostupnosti je očividná. Z hľadiska praktickej využiteľnosti nie je vhodné, aby obstarávacia cena prevodníka bola porovnateľná alebo vyššia s obstarávajúcou cenou zvyšného hardvéru. Preto je vhodnejšie uprednostniť konvenčný "nedokonalejší" prevodník pred "dokonalejším", ktorý je cenovo nedostupný.

3.2.2 Najkratšia perióda vzorkovania

Najkratšia možná perióda vzorkovania je daná rýchlosťou snímania vstupov, rýchlosťou spracovania v operačnom systéme a rýchlosťou zápisu na výstup. Je zrejmé, že presnejšie informácie o najkratšej perióde vzorkovania je možné zistiť iba pri praktickej realizácii na hardvéri.

Popisované sú iba latencie pre systém OS Linux patchovaný patchom PREEMPT RT a XENOMAI. Daný stav je zapríčinený neexistujúcou dokumentáciou pre ostatné operačné systémy portovateľné na platformu Raspberry Pi. Nasledujúce časové údaje

sú čerpané z oficiálnej dokumentácie alebo z empírie rôznych užívateľov. Dané časové údaje sú len orientačné a nezahŕňajú použitie analógových prevodníkov.

Preempt RT

Latencia upraveného Linuxu patchom PREEMPT RT je približne 100-200 μs [11] [10]. Daná latencia systému teoreticky umožňuje realizovať regulačnú slučku s periódou rádovo v jednotkách až desiatkách milisekúnd.

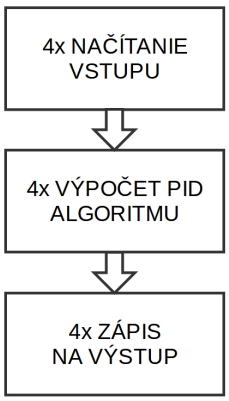
Xenomai

Latencia OS Linux na platforme Xenomai je rádovo v desiatkach mikrosekúnd [6]. Daná latencia systému teoreticky umožňuje realizovať regulačnú slučku s periódou rádovo v jednotkách milisekúnd.

3.3 Návrh regulátora

Ako operačný systém pre realizáciu regulárora som si zvolil OS Linux vo verzii vanilla a vo verzii modifikovanej patchom PREEMPT RT. Daný operačný systém som si zvolil najmä kvôli jeho univerzálnosti a dostupnosti pre platformu Raspberry Pi.

V oblasti softvérovom návrhu som sa rozhodol pre sériové spracovanie jednotlivých algoritmov regulátorov viď. obr. 3.2. Prvým krokom je načítanie všetkých vstupov, nasleduje výpočet PSD algoritmu a hromadné zapísanie hodnôt na výstupy.



Obr. 3.2: bloková schéma softvérového konceptu

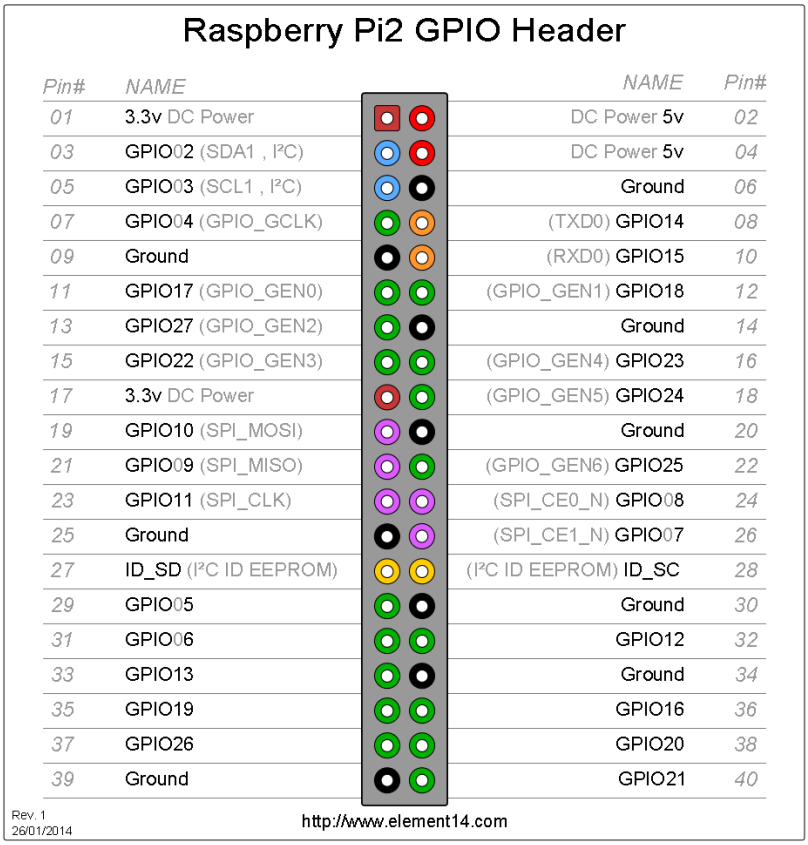
Ako periférne zariadenia som zvolil 8-bitové AD a DA prevodníky s komunikáciou po paralelnej zbernici. Daný koncept využíva priame pristupovanie ku GPIO pinom. Výhoda daného konceptu je vysoká rýchlosť komunikácie s perifériami.

4 Hardvérová špecifikácia

Kapitola stručne popisuje použitý hardvér. Bloková schéma zapojenia je na obr. 4.2.

4.1 40-pinový konektor

Platforma Paspberri Pi verzie 2 disponuje 40-pinovým konektorom viď. obr. 4.1.



Obr. 4.1: 40-pinový konektor [47]

4.2 AD prevodník

MAX114 je štvorkanálový 8-bitový analógovo-digitálny prevodník s paralelnou zbernicou. Vzorkovacia frekvencia dosahuje hodnoty okolo 1 MHz. Limitné hodnoty vstupného napätia sú 0-5V. Prepínanie medzi meracími kanálni je zabezpečené pomocou zabudovaného analógového multiplexora. Prevodník pracuje s 5V logikou. Vyčítanie dát z paralelnej zbernice je realizované pomocou trojstavového buffera.

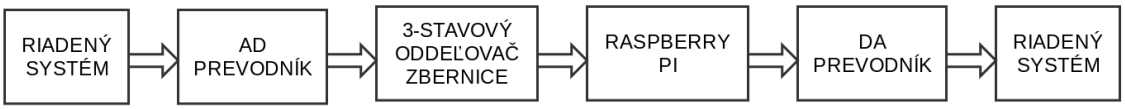
Prevodník pracuje v dvoch základných módoch: read mode a write-read mode. Dané módy ovláda pin MODE. Práca využíva prevodník v móde write-read mode (pin nastavený na vysokú logickú úroveň) viď. schéma B.3 [19].

4.3 3-stavový oddelovač zbernice

Platforma Raspberry Pi pracuje s logickou úrovňou 3.3V, horeuvedený AD prevodník pracuje s logickou úrovňou 5V. Preto je potrebné medzi tieto prvky zaradiť dodatočný prispôsobovací obvod. 74LVC245A je 8-bitový 3-stavový oddeľovač zbernice, ktorý vie operovať s obomi logickými úrovňami. Aby bola zabezpečená kompatibilita s platformou Raspberry Pi je obvod napájaný napätím 3.3V. Smer toku údajov je ovládaný pinom DIR. Pin NOT OE ovláda preklopenie vstupov na výstupy, pre účely práce je permanentne uzemnený viď. schéma B.3 [21].

4.4 DA prevodník

TLC7226 je štvorkanálový 8-bitový digitálno-analógový prevodník s paralelnou zbernicou. Prevodník vie pracovať v móde bipolárneho a unipolárneho výstupu. Práca využíva prevodník v unipolárnom móde. Pre daný mód sú limitné hodnoty výstupného napätia 0 - 12,5V. Prevodník obsahuje štiry samostatné analógové prevodníky ovládané pinmi A0, A1 a NOT WR. Piny A0, A1 určujú, ktorý výstup je aktívny pre zapísanie údajov. Pin NOT WR ovláda zapisovanie údajov na výstup. Prevodník pracuje s logickou úrovňou 5V, ktorá toleruje aj 3,3V vstupy. Z tohoto dôvodu nebolo potrebné pridávať medzi platformu Raspberry Pi a prevodník dodatočné prispôsobovacie obvody viď. schéma B.3 [43].



Obr. 4.2: bloková schéma hardvérového zapojenia

5 Softvérová implementácia

Kapitola je zameraná na opis softvérovej stránky regulátora. V úvode sú popisované základné funkcie modulu jadra. Následne je popísané API konceptu znakových zariadení, API časovača a API vlákien jadra. Komunikácia s periférnymi zariadeniami je popísaná v podkapitolách GPIO rozhrenie a AD, DA prevodník. Kardinálnou je podkapitola opisujúca algoritmus PSD regulátora. V závere je popisovaná funkcionalita webového HMI rozhrania.

Globálne premenné a funkcie modulu jadra je vhodné definovať ako statické. Klúčové slovo `static` zabezpečí platnosť daných inštancií iba v module jadra. V opačnom prípade majú dané inštancie platnosť v celom priestore jadra. Ak je nevyhnutné použiť globálne premenné, odporúča sa deklarácia s unikátnym prefixom [20].

5.1 Základné funkcie modulu jadra

Modul jadra môže byť načítaný pomocou príkazu `sudo insmod` a odstránený pomocou príkazu `sudo rmmod`. Nasledujúce funkcie sú deklarované v linux/module.h, linux/kernel.h a linux/init.h.

Funkcia pid_init() je volaná pri načítaní. Slúži na prvotnú inicializáciu systémových zdrojov. Nahradzuje funkciu init_module(), ktorá je v module jadra povinná. Substitúcia je realizovaná pomocou makier ___init a module_init().

```
static int __init pid_init(void)
```

Funkcia pid_exit() je volaná pred odstránením modulu jadra. Slúži na uvoľnenie alokovaných systémových zdrojov. Nahradzuje funkciu cleanup_module, ktorá je v module jadra povinná. Substitúcia je realizovaná pomocou makier ___exit a module_exit().

```
static void __exit pid_exit(void)
```

Funkcia printk() je v priestore jadra ekvivalentná funkcii printf(). Správy sú zapisované do systémového logu. Systémový log je možné zobraziť pomocou príkazu `dmesg`, alebo vyčítať z virtuálneho súboru `/var/log/syslog`. Vstupným parametrom funkcie je správa, ktorá môže obsahovať makro určujúce jej dôležitosť. V práci sú použité makrá KERN_INFO pre informačné výpisy a KERN_ALERT pre chybové výpisy.

```
int printk(const char *fmt, ...)
```

Informácie o module jadra zobrazuje príkaz `modinfo`. Dané informácie sú do modulu pridané pomocou nasledujúcich makier.

Makro MODULE_LICENSE() poskytuje informáciu o licencovaní. Linuxové jadro je licencované pod licenciou GPL, z tohoto dôvodu je vhodné takto licencovať aj moduly jadra pokiaľ to situácia nevyžaduje inak.

```
MODULE_LICENSE("GPL")
```

Makro MODULE_AUTHOR() poskytuje informáciu o autorovi.
Makro MODULE_DESCRIPTION() popisuje modul jadra.
Makro MODULE_VERSION() určuje verziu modulu jadra.

5.2 Komunikácia modulu jadra s užívateľským priestorom

Komunikácia modulu jadra s užívateľským priestorom môže byť realizovaná pomocou konceptu znakových zariadení. Tieto zariadenia sa nachádzajú v adresári `/dev/`. Práca využíva daný koncept pre účely vyčítania nameraných časových údajov.
Druhou možnosťou komunikácie je koncept systémového súborového systému sysfs nachádzajúceho sa v adresári `/sys/`. Práca využíva daný koncept pre účely nastavovania parametrov regulátora.

5.2.1 Znakové zariadenia

Dôležitým identifikátorom znakových zariadení je hlavné a vedľajšie číslo zariadenia. Hlavné číslo určuje o aký typ zariadenia sa jedná. Vedľajšie číslo je jedinečné pre každé zariadenie. Príkladom môžu byť dve zariadenia sériového portu `/dev/ttyS0` a `/dev/ttyS1` ktoré majú rovnaké hlavné, ale rozdielne vedľajšie číslo. API znakových zariadení je deklarované v linux/fs.h.

Štruktúra fops typu file_operations obsahuje ukazovatele na funkcie. Dané funkcie definujú ako sa má zariadenie správať pri operáciách so súborom. Príkladom je implementovaná funkcia dev_read(), ktorá definuje ako sa má zariadenie správať v prípade vyčítania údajov.

```
static struct file_operations fops = {.read = dev_read}
```

Funkcia dev_read() slúži ako obslužná funkcia. Realizuje vyčítanie údajov zo znakového zariadenia. Dôležitým vstupným parametrom je ukazovateľ char *buffer. Daný

ukazovateľ následne slúži ako vstupný parameter pre funkciu copy_to_user(). Táto funkcia zaisťuje kopírovanie dát z priestoru jadra do užívateľského priestoru.

```
static ssize_t dev_read(struct file *filep, char *buffer,\
size_t len, loff_t *offset)
```

Funkcia register_chrdev() má vstupné argumenty hlavné číslo (v prípade 0 dynamicky alokované), názov zariadenia a ukazovateľ na štruktúru typu file_operations. Návratová hodnota je typu int, v prípade úspechu vracia kladné hlavné číslo zariadenia. V prípade neúspechu je návratová hodnota záporné číslo.

```
int register_chrdev(unsigned int major, const char *name,\
const struct file_operations *fops)
```

Funkcia class_create() vytvára triedu potrebnú ako vstupný parameter vo funkcii device_create().

```
struct class * class_create(struct module *owner,\
const char *name)
```

Funkcia device_create() registruje znakové zariadenie. Dôležité vstupné argumenty sú ukazovateľ na vyššie opisovanú triedu class, identifikačné číslo zariadenia získané z makra MKDEV() a názov zariadenia.

```
struct device *device_create(struct class *class,\
struct device *parent, dev_t devt, const char *fmt, ...)
```

Funkcia device_destroy() odstráni znakové zariadenie vytvorené funkciou device_create().

```
void device_destroy(struct class *class, dev_t devt)
```

Funkcia class_unregister() odregistruje danú triedu.

```
void class_unregister(struct class *class)
```

Funkcia class_destroy() dealokuje danú triedu.

```
void class_destroy(struct class *cls)
```

Funkcia unregister_chrdev() odregistruje znakové zariadenie.

```
static inline void unregister_chrdev(unsigned int major,\
const char *name)
```

5.2.2 Sysfs

API sysfs je deklarované v linux/kobject.h.

Funkcia mode_show0() je obslužná funkcia, slúži na vyčítanie režimu v ktorom pracuje regulátor číslo 0. Vstupný parameter struct kobject *kobj je referencia na objekt zariadenia, ktorý sa zobrazí v štruktúre sysfs. Vstupný parameter char *buf je referencia na textový reťazec, ktorý má byť vypísaný v užívateľskom priestore. Návratová hodnota je počet vyčítaných znakov.

```
static ssize_t mode_show0(struct kobject *kobj,\
struct kobj_attribute *attr, char *buf)
```

Funkcia mode_store0() je podobná ako mode_show0(). Slúži na načítanie režimu v ktorom má pracovať regulátor číslo 0. Oproti vyššie uvedenej funkcii má naviac vstupný parameter size_t count, ktorý určuje počet znakov v buffri. Návratová hodnota je počet znakov vyčítaných z buffra.

```
static ssize_t mode_store0(struct kobject *kobj,\
struct kobj_attribute *attr, const char *buf, size_t count)
```

Štruktúra mode_attr0 typu kobj_attribute agreguje funkciu načítania a vyčítania, prístupové práva a meno virtuálneho súboru v sysfs. Inicializáciu zjednodušuje makro ___ATTR().

```
static struct kobj_attribute mode_attr0 = __ATTR(mode, 0664,\
mode_show0, mode_store0)
```

Pole ukazovateľov *pid0_attrs[] na štruktúru typu attribute agreguje viacero atribútov medzi inými aj vyššie spomenutú štruktúru mode_attr0. V konkrétnej aplikácii to znamená, že v určitom priečinku naviazanom na práve jeden regulátor, bude viacero virtuálnych súborov, parametrov regulátora.

```
static struct attribute *pid0_attrs[] = {
      &P_attr0.attr,
      &I_attr0.attr,
      &D_attr0.attr,
      &N_attr0.attr,
      &setPoint_attr0.attr,
      &mode_attr0.attr,
      &input_attr0.attr,
      &output_attr0.attr,
      NULL,};
```

Štruktúra pid0 typu attribute_group agreguje štruktúru pid0_attrs a názov priečinku v ktorom sú lokalizované parametre jedného regulátora.

```
static struct attribute_group pid0 = {
      .name  = "pid0",
      .attrs = pid0_attrs,
};
```

Funkcia kobject_create_and_add() vvtvorí priečinok v sysfs. Vstupný argument name určuje názov priečinku. Vstupný parameter parent určuje, kde má byť priečinok vytvorený. Návratová hodnota je ukazovateľ na novovzniknutý kobject.

```
struct kobject * kobject_create_and_add(const char *name,\
struct kobject *parent)
```

Funkcia sysfs_create_group() priradí kobjektu štruktúru typu attribute_group. V konkrétnej aplikácii to znamená, že priečinky s atribútmi pre každý regulátor budú v rodičovskom priečinku. Návratová hodnota funkcie je číslo. V prípade chyby je číslo nenulové.

```
int sysfs_create_group(struct kobject *kobj,\
const struct attribute_group *grp)
```

Funkcia kobject_put() dekrementuje referenčné číslo na objekt. Ak referenčné číslo klesne na 0, zavolá sa funkcia kobject_cleanup().

```
void kobject_put(struct kobject *kobj)
```

5.3 HR časovač

API časovačov s vysokým rozlíšením je deklarované v linux/hrtimer.h.

Štruktúra hrtimer obsahuje informácie o časovači. Dôležitou premennou v štruktúre je ukazovateľ na funkciu ktorá sa má vykonávať v prípade že časovač expiruje.

```
struct hrtimer {
      struct rb_node node;
      ktime_t expires;
      int (* function) (struct hrtimer *);
      struct hrtimer_base * base;
}
```

Funkcia hrtimer_init() zaisťuje prvotnú inicializáciu časovača. Prvý argument je ukazovateľ na štruktúru časovača. Druhý vstupný argument udáva typ časovača. Časovač typu CLOCK_MONOTONIC sa riadi časom, ktorý sa vzťahuje k nulovému času pri bootovaní systému. Časovač typu CLOCK_REALTIME sa riadi časom tak, ako ho vnímajú ľudia. Tento časovač zohľadňuje napríklad aj posun času. Pre účely práce je vhodný časovač typu CLOCK_MONOTONIC. Tretí argument nastavuje mód časovača. HRTIMER_MODE_ABS nastavuje absolútny čas, HRTIMER_MODE_REL nastavuje relatívny čas k aktuálnemu okamihu [15].

```
void hrtimer_init(struct hrtimer *timer,\
clockid_t which_clock, enum hrtimer_mode mode)
```

Funkcia hrtimer_start() spustí časovač. Vstupné argumenty sú ukazovateľ na štruktúru časovača, premenná typu ktime_t obsahujúca hodnotu času v nanosekundách a premenná nastavujúca mód časovača.

```
static inline void hrtimer_start(struct hrtimer *timer,\
ktime_t tim, const enum hrtimer_mode mode)
```

Funkcia hrtimer_forward_now() nastavuje časovač na dobu, kedy má expirovať. Prvý vstupný argument je ukazovateľ na štruktúru časovača. Druhý vstupný argument je časový interval, v ktorom má časovač expirovať. Tento časový interval sa vzťahuje k časovému okamihu kedy je volaná funkcia.

```
u64 hrtimer_forward_now (struct hrtimer * timer,\
ktime_t interval)
```

Funkcia ktime_set() slúži na nastavovanie času. Vstupné argumenty sú počet sekúnd a počet nanosekúnd. Návratová hodnota je typu ktime_t.

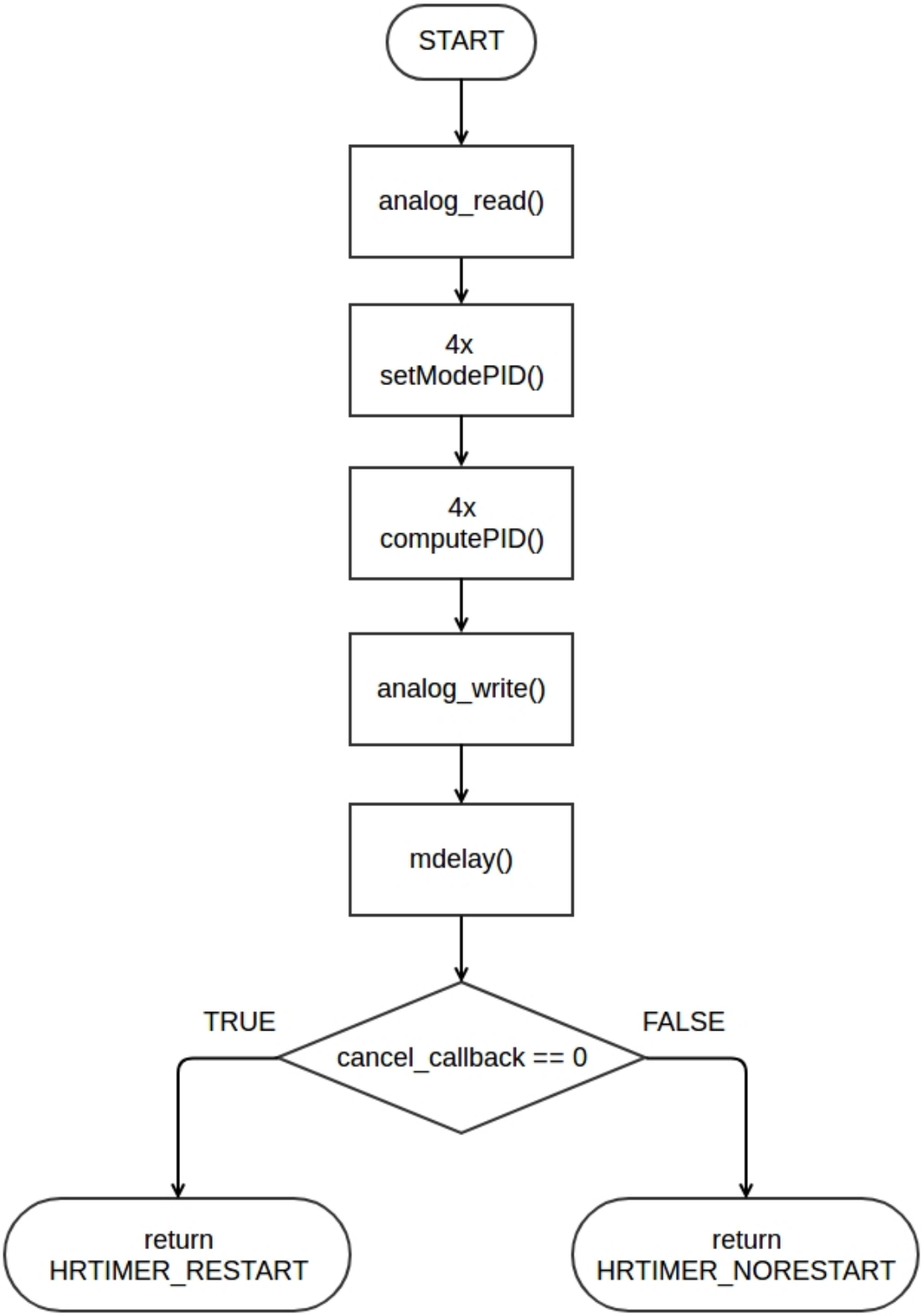
```
ktime_t ktime_set (const long secs, const unsigned long nsecs)
```

Funkcia hrtimer_cancel() deinicializuje časovač. Vstupný argument je ukazovateľ na štruktúru časovača. V prípade že časovač bol aktívny, návratová hodnota je 1. V opačnom prípade je návratová hodnota 0.

```
int hrtimer_cancel(struct hrtimer *timer)
```

Funkcia executePID_sleep() je definovaná pre účely spúšťania pid algoritmu v presných časových intervaloch. Vstupným parametrom je ukazateľ na štruktúru časovača. Funkcia je volaná vždy keď časovač expiruje. Návratová hodnota je typu enum hrtimer_restart a určuje či sa má časovač v nasedujúcom cykle obnoviť.

```
enum hrtimer\_restart executePID\_sleep(struct hrtimer *timer)
```



Obr. 5.1: algoritmus funkcie executePID_sleep()

5.4 Kthread

API vlákien v priestore jadra kthread je deklarované v linux/kthread.h.

Funkcia kthread_run() vytvorí a spustí vlákno. Prvým vstupným argumentom je ukazovateľ na funkciu, ktorá sa má vykonávať. Druhý vstupný argument je ukazateľ na dáta, ktoré sa majú predať do funkcie vlákna. Tretí argument je meno vlákna. Návratová hodnota je ukazateľ na štruktúru task_struct ktorá obsahuje informácie o vlákne.

```
task_struct *kthread_run(int (*threadfn)(void *data),\
void *data, const char *namefmt)
```

Funkcia kthread_stop ukončí vlákno. Vstupný parameter je ukazovateľ na štruktúru vlákna.

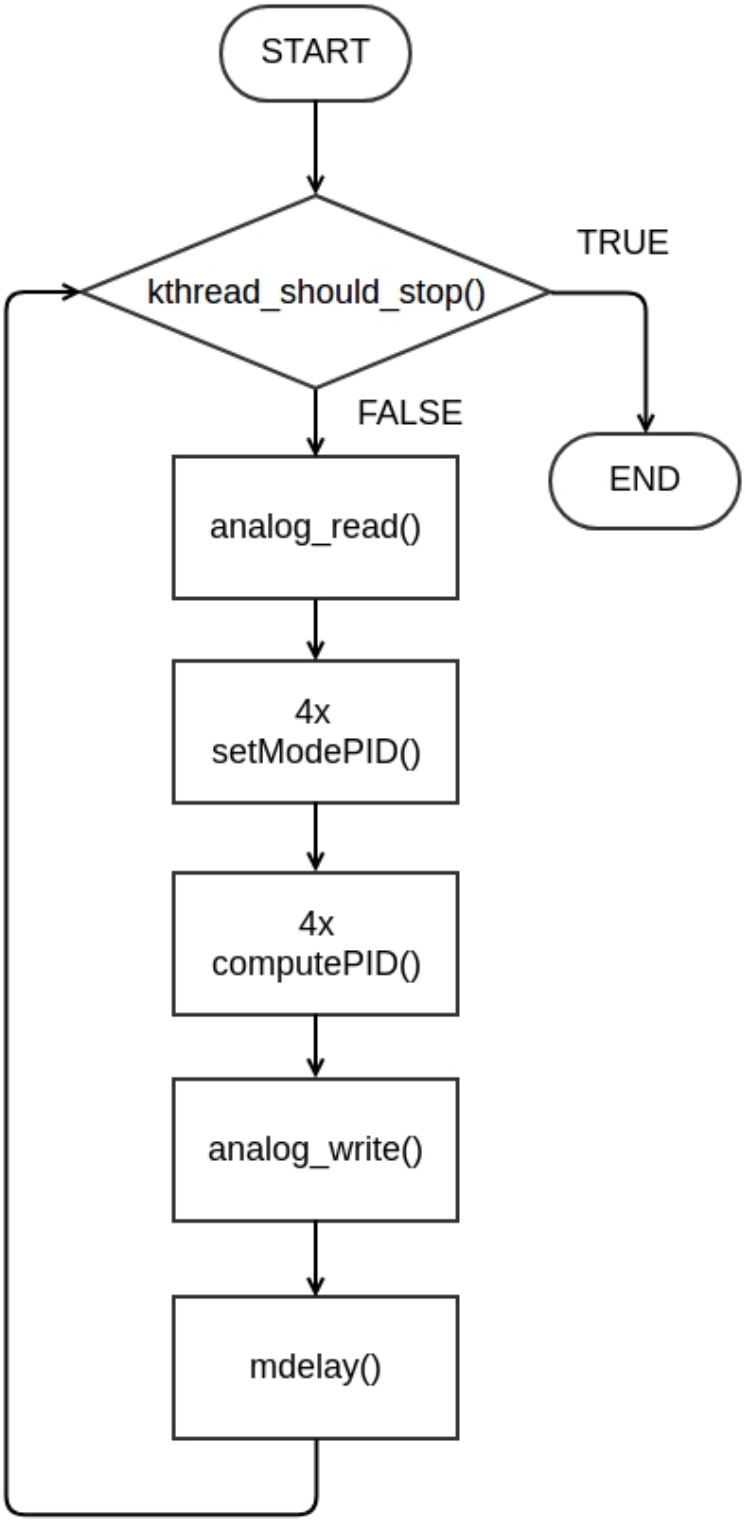
```
int kthread_stop(struct task_struct *k)
```

Funkcia kthread_should_stop() sa volá z funkcie vlákna. V prípade že bola zavolaná funkcia kthread_stop() návratová hodnota je TRUE.

```
int kthread_should_stop(void)
```

Funkcia executePID_delay() je definovaná pre účely spúšťania psd algoritmu v presných časových intervaloch. Periodicita spúšťania pid algoritmu ja zabezpečená pomocou cyklu while, kde sa ako podmienka ukončenia volá funkcia kthread_should_stop(). Časovanie je zabezpečené funkciou mdelay().

```
static int executePID_delay(void *arg)
```



Obr. 5.2: algoritmus funkcie executePID_delay()

5.5 GPIO rozhranie

Prístup ku gpio pinom je realizovaný priamo, pomocou zapisovania hodnôt do registrov. Makrá pre zaobchádzanie s registrami definuje knižnica pin_tag.h.

Makro BCM2836_PERI_BASE definuje fyzickú adresu, kde začínajú registre periférií. Makro GPIO_BASE definuje začiatočnú adresu pre gpio piny. Jeho hodnota je definovaná ako BCM2836_PERI_BASE + 0x200000.

Štruktúra bcm2836_peripheral obsahuje premenné potrebné ku prístupu k registrom. Dôležitou premennou je adresa registra definovaná ako ukazovateľ na typ int.

```
volatile unsigned int *addr
```

Premenná gpio je definovaná nasledovne.

```
struct bcm2835_peripheral gpio = {GPIO_BASE};
```

Makro INP_GPIO(g) alebo makro OUT_GPIO(g) nastavuje daný gpio pin ako vstupný alebo výstupný. Funkciaonalita GPIO pinov sa nastavuje v GPIO Function Select Registers skrátene GPFSEL. 32-bitový register obsahuje nastavenie pre 10 pinov. Funkcionalita každého pinu sa nastavuje tromi bitmi. Prístup k správnemu registru zaistí nasledovný kód.

```
*gpio.addr + ((g))/10
```

Následne je potrebné nastaviť tri bity v danom registri. Pre funkciu vstupu treba nastaviť príslušné bity na hodnotu `000`.

```
#define INP_GPIO(g) *(gpio.addr+((g)/10)) &=~(7<<(((g)%10)*3))
```

Pre funkciu výstupu treba nastaviť príslušné bity na hodnotu `001`.

```
#define OUT_GPIO(g) *(gpio.addr+((g)/10)) |=(1<<(((g)%10)*3))
```

Horeuvedené makro využíva na nastavovanie bitov logický or. Z tohoto dôvodu je nevyhnutné pred každým použitím makra OUT_GPIO() vynulovať dané bity makrom INP_GPIO().

Makro GPIO_SET sprostredkúva nastavenie logickej 1 na výstupnom pine. Výstupné hodnoty pinov sa nastavujú v registri GPSET. Tento register sa nachádza na adrese o 7 väčšej ako GPIO_BASE.

```
#define GPIO_SET  *(gpio.addr + 7)
```

Piny ktoré majú príslušiace bity registra na logickej 1 sú nastavené na úroveň HIGH. Bity s logickou 0 sú ignorované. Príkladom môže byť nasledovné nastavenie pinu číslo 4 na logickú úroveň HIGH.

```
GPIO_SET = 1 << 4;
```

Makro GPIO_CLR sprostredkúva nastavenie logickej 1 na výstupnom pine. Register GPCLR sa nachádza na adrese o 10 väčšej ako GPIO_BASE.

```
#define GPIO_CLR  *(gpio.addr + 10)
```

Makro GPIO_READ(g) vyčítava hodnotu na danom pine. Register GPLEV sa nachádza na adrese o 13 väčšej ako GPIO_BASE. Na vyčítanie hodnoty daného pinu sa využíva logický and s príslušnou maskou [24].

```
GPIO_READ(g)  *(gpio.addr + 13) &= (1<<(g))
```

Funkcia void pin_initialize() inicializuje GPIO piny. GPIO piny sú nastavené ako vstupné alebo výstupné v závislosti od špecifikácie pre AD, alebo DA prevodník.

```
void pin_initialize(void)
```

5.6 AD, DA prevodník

Funkcia analog_read() vyčítava hodnoty jednotlivých vstupov z analógovo-digitálneho prevodníka. Vstupné argumenty sú ukazovatele na vstupné hodnoty pre pid algoritmus.

```
inline void analog_read(int *num0, int *num1, int *num2,\
int *num3)
```

Funkcia pracuje s prevodníkom v móde write-read. Postup pre vyčítanie údajov pre jeden vstup prevodníka je nasledovný [19].

1. zapísanie LOW úrovne na pin NOT WR, začiatok konverzie v AD prevodníku
2. čakanie po dobu Twr
3. zapísanie HIGH úrovne na pin NOT WR
4. čakanie kým hodnota na pine NOT INT nie je LOW, indikátor konca konverzie
5. prepnutie multiplexora na nasledujúci vstup

6. zapísanie LOW úrovne na pin NOT RD
7. čakanie po dobu Tacc
8. vyčítanie analógovej hodnoty zo zbernice
9. zapísanie HIGH úrovne na pin NOT RD
10. čakanie po dobu Tacq, potrebné aby mohlo začať vyčítanie z ďalšieho vstupu

Funkcia analog_write() zapisuje jednotlivé hodnoty do digitálno-analógového prevodníka. Vstupné argumenty sú ukazatele na výstupné hodnoty z pid algoritmu.

```
inline void analog_write(int *num0, int *num1, int *num2,\
int *num3)
```

Postup pre načítanie údajov pre jeden vstup prevodníka je nasledovný [43].

1. nastavenie multiplexora
2. zapísanie LOW úrovne na pin NOT WR
3. zapísanie analógovej hodnoty na zbernicu
4. zapísanie HIGH úrovne na pin NOT WR

5.7 PSD algoritmus

PSD regulátor bol realizovaný podľa schémy 3.1. Schéma neznázorňuje prepínanie z automatického do manuálneho režimu a s tým spojené beznárazové prepínanie.

Jadro výpočtu PSD algoritmu vykonáva funkcia computePID().

```
inline void computePID( PID *pid )
```

Na začiatku algoritmu sa kontroluje, či je regulátor v manuálnom režime. Ak je táto podmienka vyhodnotená kladne, celý algoritmus sa vynechá.

```
if( pid->mode == MAN )  return;
```

Nasleduje výpočet regulačnej odchýlky.

```
//ERROR COMPUTING
pid->oldErr = pid->err;
pid->err = pid->setPoint - pid->input;
```

Nasleduje výpočet derivačnej zložky s filtrom. Násobením a delením hodnotou NORMALIZE sa kompenzuje neprítomnosť premennej s pohyblivou desatinnou čiarkou.

```
//D ERROR COMPUTING
pid->dOldErr = pid->dErr;
pid->dErr = (pid->dOldErr * pid->N + (pid->err * NORMALIZE))\
/ NORMALIZE;
```

V ďalšom kroku sa vypočíta sumačná zložka. Zároveň je implementovaná ochrana proti windup efektu. Ochrana je realizovaná obmedzeniami. Obmedzenia odpovedajú medzným hodnotám DA prevodníka prenásobenými hodnotou NORMALIZE.

```
//ERROR*INTEGRAL_GAIN SUM COMPUTING + ANTI WINDUP
pid->sum += pid->I * pid->err;
if( pid->sum > OUT_MAX )    pid->sum = OUT_MAX;
else if( pid->sum < OUT_MIN )   pid->sum = OUT_MIN;
```

Nakoniec sa realizuje sčítanie všetkých troch zložiek PID regulátora. Po sčítaní je realizované ďalšie obmedzenie tak, aby boli vypočítané hodnoty v intervale medzných hodnôt DA prevodníka. Finálne je výstup predelený hodnotou NORMALIZE. Z toho vyplýva, že všetky zosilnenia regulátora majú nižšiu váhu. Váha zosilnení je akoby podelená hodnotou NORMALIZE.

```
//OUTPUT COMPUTING
pid->output = pid->P*pid->err + pid->sum \
+ pid->D*( pid->dErr - pid->dOldErr );
if( pid->output > OUT_MAX )   pid->output = OUT_MAX;
else if( pid->output < OUT_MIN )  pid->output = OUT_MIN;
pid->output /= NORMALIZE;
```

Funkcia setModePID() nastavuje mód v ktorom pracuje regulátor.Funkcia musí byť volaná pred funkciou computePID(). V opačnom prípade by nebolo možné zaistiť beznárazové prepínanie.

```
void setModePID( enum modes mode, PID *pid )
```

Na začiatku sa kontroluje či regulátor prechádza zo stavu MAN do stavu AUTO.

```
if( mode == AUTO && pid->mode == MAN )
```

Ak je táto podmienka vyhodnotená kladne, nasledujú kroky zabezpečujúce beznárazové prepínanie. Sumačná zložka je nastavená na hodnotu rovnú výstupu. Je vykonaný nový výpočet regulačnej odchýlky a hodnoty pre výpočet derivačnej zložky s filtrom. Tento krok má za následok že derivačná zložka vo výpočte PID algoritmu bude nulová. Algoritmus sa proporciálnou zložkou nezaoberá pretože túto zložku neovplyvňujú predchádzajúce hodnoty regulačnej odchýlky. Z tohoto dôvodu je pre beznárazové prepínanie nerelevantné.

```
pid->sum = pid->output;
pid->err = pid->setPoint - pid->input;
pid->dErr = (pid->dOldErr * pid->N + (pid->err * NORMALIZE))\
/ NORMALIZE;
```

Vo finálnom kroku algoritmu je zabezpečené nastavenie módu regulátora.

```
pid->mode = mode;
```

5.8 Rozhranie človek-stroj

Rozhranie určené na komunikáciu medzi človekom a strojom angl. human machine interface (HMI) je dôležitou súčasťou softvérovej implementácie riadenia. Natívna komunikácia medzi modulom jadra a užívateľským priestorom je zabezpečená pomocou virtuálneho súborového systému. Jednotlivé súbory sa nachádzajú v adresári `/sys/kernel/PID`. Obsah adresára je nasledovný.

```
period  pid0  pid1  pid2  pid3  readout
```

Pričom cez súbory `period` a `readout` sa dá nastaviť a vyčítať perióda cyklu regulátorov resp. vyčítať všetky parametre regulátorov vo forme json, neskôr využívané pri komunikácii s webovým rozhraním. Súbory `pid0-pid3` predstavujú adresáre. Každý z nich obsahuje parametre vlastné jednému regulátoru.

```
/......................................................./sys/kernel/PID
├── D
├── I
├── N
├── input
├── mode
├── output
├── P
└── setPoint
```

Z dôvodou lepšej vizualizácie údajov bolo vytvorené grafické webové rozhranie. HMI rozhranie funguje na platforme linuxového servera Appache, konkrétne na verzii 2.4. Frontendová časť webu je tvorená html a css súbormi `index.html a style.css`. Načítavanie a nastavovanie údajov zaobstaráva javascriptový súbor `script.js`. Daný skript zabezpečuje periodické načítanie aktuálnych údajov každú sekundu pomocou technológií AJAX a CGI. Backendová časť webu je tvorená bashovými skriptami `readout.sh a set.sh`, ktoré sa nachádzajú v štandardnom adresári pre CGI skripty `/usr/lib/cgi-bin`. Grafické prevedenie HMI rozhrania je na obr. B.1.

6 Overenie korektnej činnosti regulátora

Kapitola obsahuje popis jednotlivých scenárov merania tak, ako boli realizované spolu s vizualizáciou nameraných hodnôt.

6.1 Softvérové merania časových úsekov

Merania boli vykonané pre rôzne verzie regulátora. Podmienenú kompiláciu zdrojového kódu pre rôzne verzie meraní zabezpečujú príznaky v súbore Makefile. Počet vzoriek pre každý variant meraní bol 10 000. Výsledky meraní boli následne spracované ako aritmetický priemer hodnôt so smerodajnou odchýlkou typu A. Pre účely merania bolo potrebné využiť nasledujúce funkcie z knižnice linux/time.h.

Funkcia getnstimeofday() uloží časovú vzorku v okamihu volania. Vstupným argumentom je ukazovateľ na štruktúru do ktorej sa uloží časová vzorka.

```
void getnstimeofday(struct timespec *ts)
```

Funkcia timespec_to_ns() zabezpečuje konverziu časovej vzorky na milisekundy. Vstupným argumentom je ukazovateľ na štruktúru časovej vzorky. Návratová hodnota je počet nanosekúnd.

```
s64 timespec_to_ns(const struct timespec *ts)
```

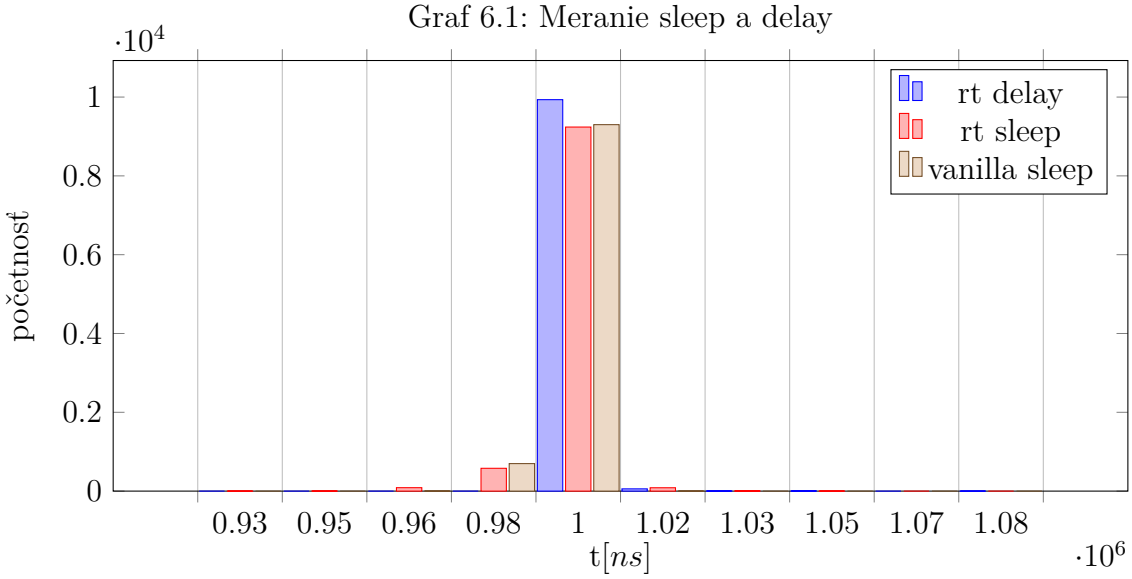
6.1.1 Meranie doby sleep a delay

Merania boli vykonané s periódou regulátora nastavenou na 1ms. Prvé meranie overovalo presnosť funkcie mdelay(). Toto meranie bolo realizované iba pre patchované jadro linuxu. V nepatchovanom linuxovom jadre funkcia mdelay() spôsobovala "zamrznutie" operačného systému. Presnosť spania pomocou časovača s vysokým rozlíšením bola meraná na obidvoch softvérových platformách.

Tab. 6.1: Meranie doby sleep a delay

	min	max	aritmetický priemer	smerodajná odchýlka
`rt delay[ns]`	1 000 271	1 094 283	1 000 639	22,2
`rt sleep[ns]`	938 122	1 058 121	999 999	38,5
`vanilla sleep[ns]`	971 182	1 030 608	999 998	19,8

Graf 6.1: Meranie sleep a delay



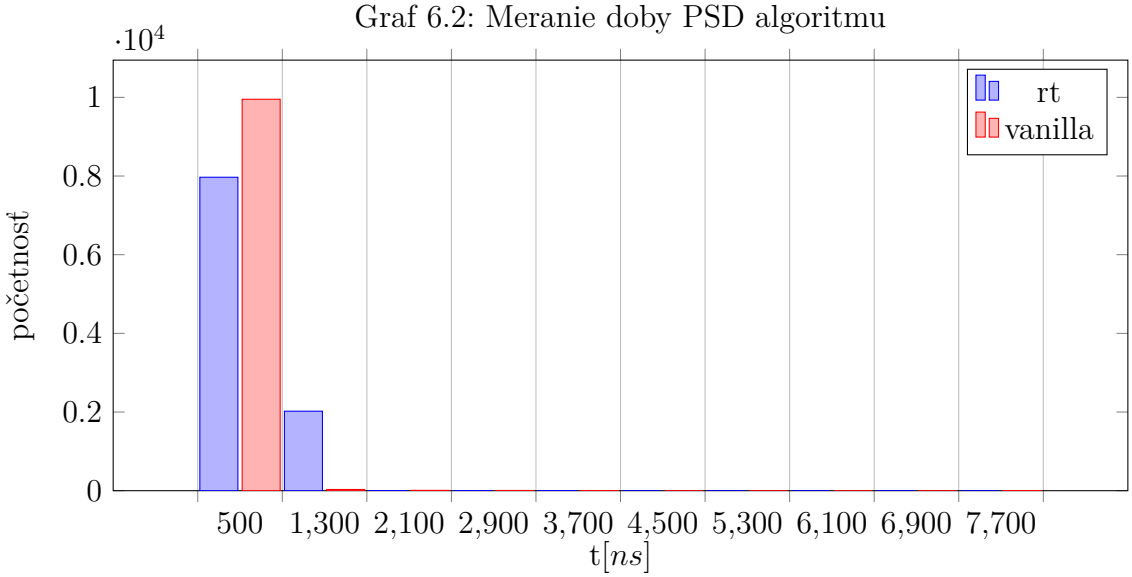
6.1.2 Meranie doby PSD algoritmu

Meranie zahrnuje čas potrebný na volanie štyroch inštancií funkcií setModePID() a computePID(). Merania boli vykonané na patchovanom aj vanilla linuxovom jadre.

Tab. 6.2: Meranie doby PSD algoritmu

	min	max	aritmetický priemer	smerodajná odchýlka
`rt[ns]`	781	4 166	1 220	1,2
`vanilla[ns]`	937	8 229	1 039	1,2

Graf 6.2: Meranie doby PSD algoritmu



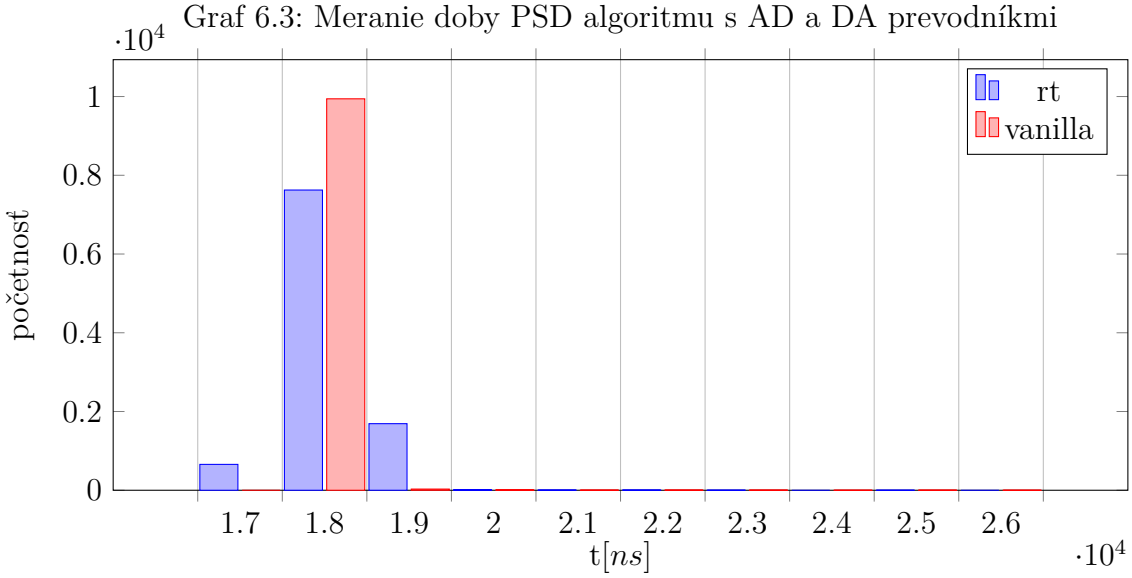
6.1.3 Meranie doby PSD algoritmu s AD a DA prevodníkmi

Meranie zahrnuje čas potrebný na volanie štyroch inštancií funkcií setModePID() a computePID() a funkcie analog_read() a analog_write(). Merania boli vykonané na patchovanom aj vanilla linuxovom jadre.

Tab. 6.3: Meranie doby PSD algoritmu s AD a DA prevodníkmi

	min	max	aritmetický priemer	smerodajná odchýlka
`rt[ns]`	17 135	25 833	18 763	1,4
`vanilla[ns]`	18 229	26 041	18 646	2,4

Graf 6.3: Meranie doby PSD algoritmu s AD a DA prevodníkmi



6.2 Meranie prechodovej funkcie

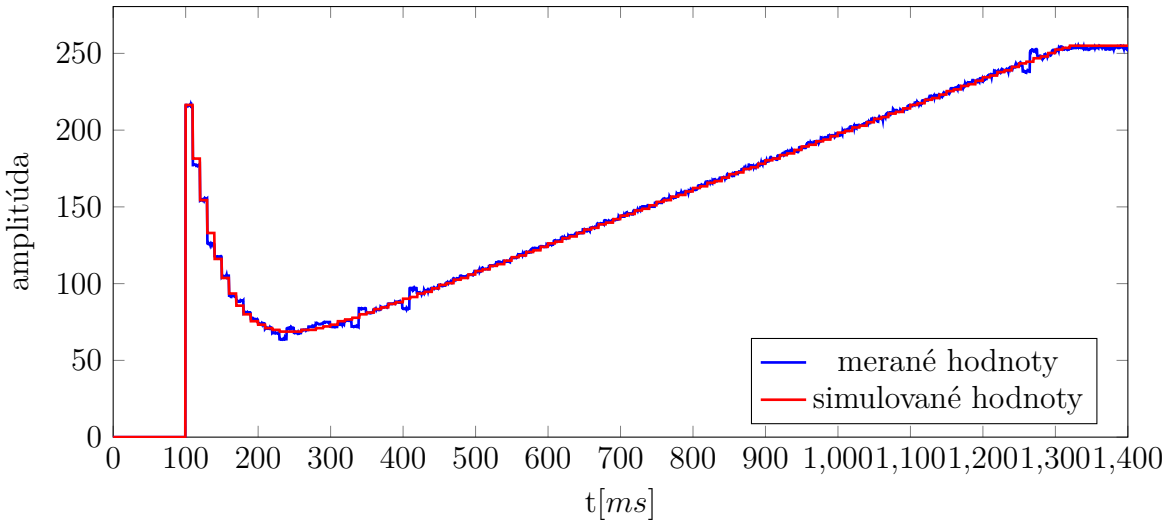
Prechodová funkcia regulátora bola meraná pomocou externého zariadenia arduino vo verzii Uno. Jednotkový skor aj meranie výstupu regulátora boli realizované platformou arduino. Vzorkovacia frekvencia analógovo-digitálnych prevodníkov platformy arduino je približne 10kHz [2]. Periodické vzorkovanie bolo zaistené časovačom s periódou 0.5ms. Perióda pid regulátora bola nastavená na 10ms. Meranie bolo realizované na patchovanom linuxovom jadre vo verzii s časovačom s vysokým rozlíšením. Pre účely tohoto merania bolo potrebné do funkcie executePID_sleep() pridať nasledovný kód.

```
pid_param[0].input = -pid_param[0].input;
```

Ďalšou nevyhnutnou podmienkou bolo nastavenie požadovanej hodnoty na nulu setPoint=0. Parametre regulátora boli nastavené nasledovne. P=100, I=5, D=500,

N=80. V grafe 6.2 sú vynesené hodnoty z merania a zo simulácie. Simulácia bola realizovaná v prostredí matlab podľa schémy B.4.

Graf 6.4: Meranie prechodovej funkcie



7 Výsledky práce

Kapitola pojednáva o dosiahnutých výsledkoch v rámci praktickej implementácie. Diskutované výsledky sú členené do jednotlivých podkapitol podľa kategórií.

7.1 Hardvérové riešenie

V rámci hardvérového riešenia práce bolo potrebné využiť externé periférne zariadenia. Konkrétne AD a DA prevodníky a 3-stavový oddeľovač zbernice. Daný koncept bol realizovaný na nepájivom kontaktnom poli viď. obr. B.2. Potreba 3-stavového oddeľovača zbeznice bola daná rôznymi logickými úrovňami AD prevodníka a platformy Raspberry Pi. Oba prevodníky sú 4-kanálové a pre komunikáciu s platformou Raspberry Pi využívajú paralelnú zbernicu, ktorá je pripojená priamo na GPIO piny. Dané riešenie sa ukázalo pre účely riadenia ako dostatočne rýchle. Vzorkovacia frekvencia jedného vstupu AD prevodníka je približne 1 Mhz. Problémom v niektorých aplikáciách by mohol byť malý počet vzorkovacích úrovní (256). Vybraný DA prevodník splňuje zadané požiadavky pre výstupné napätie v rozsahu 0 - 10 V. AD prevodník má vstupné napätie v rozsahu 0 - 5 V, preto je potrebné zaradiť pred analógové vstupy napäťový delič.

7.2 Softvérové riešenie

Ako operačný systém bol vybraný OS Linux vo verzii vanilla a vo verzii s patchom PREEMPT RT. Regulátor bol implementovaný ako modul jadra. Pre obe verzie OS Linux boli realizované regulátory s dvomi typmi časovania. Prvý typ časovania využíva vlákno jadra a funkciu mdelay(), jedná sa o blokujúce časovanie. Druhý typ časovania využíva časovače s vysokým rozlíšením, jedná sa o neblokujúce časovanie. Ukázalo sa že vanilla systém vo verzii využívajúcej vlákno a funkcionalitu delay po krátkom čase “zamrzne“. Z tohoto dôvodu neboli na systéme s danou konfiguráciou vykonávané dodatočné merania. Zároveň sa daná konfigurácia vyčlenila z možností riešení práce. Algoritmus regulátora bol riešený sériovo s hromadným vyčítaním AD prevodníka a hromadným nastavením výstupov DA prevodníka. Nemožnosť použitia matematických operácií s pohyblivou desatinnou čiarkou bola kompenzovaná dodatočným prepočtom s definovanou konštantou.

7.3 Výsledky meraní

Ako prvé bolo realizované meranie časovacích mechanizmov viď tab. Z nameraných maximálnych a minimálnych hodnôt je možné určiť najmenšiu periódu vzorkovania na 1ms viď tab. 6.1. Je zrejmé, že oba typy časovacích mechanizmov dosahujú približne rovnako dobré vlastnosti v oblasti časovania. Nevýhodou konceptu využívajúceho delay je blokujúce čakanie a nemožnosť použitia vo verzii s vanilla OS Linux. Z vyššie uvedených dôvodov vyplýva, že koncept s využitím časovača s vysokým rozlíšením má univerzálnejšie využitie.

Doba potrebná na výpočet štyroch PSD algoritmov dosahuje hodnoty rádovo jednotky mikrosekúnd viď. tab. 6.2. Ako posledné bolo vykonané meranie PID algoritmu s AD a DA prevodníkmi. Namerané časové hodnoty sú niekoľko rádov nižšie ako je latencia systému viď. tab. 6.3. Z tohoto dôvodu nie je doba PID algoritmu s AD a DA prevodníkmi limitujúcim faktorom pri určovaní najkratšej možnej periódy vzorkovania.

Je badateľné, že regulátor s vanilla OS Linux dosahoval zľahka lepšie vlastnosti v oblasti časovania. Toto môže byť zapríčinené rozdielnou verziou jadra OS Linux. Vanilla jadro je vo verzii 4.9.59 a patchované jadro je vo verzii 4.9.47.

Posledné meranie zahŕňalo meranie prechodovej funkcie PSD regulátora. Následným porovnaním so simuláciou bola overená správnosť algoritmu.

8 Záver

Diplomová práca mala za úlohu overiť využiteľnosť platformy Raspberry Pi pre účely riadenia. Kľúčovou časťou práce bola problematika operačných systémov reálneho času. V úvode práce boli preto prvotne zadefinované pojmy súvisiace s touto problematikou. Následne boli opísané rôzne operačné systémy portovateľné na Raspberry Pi. Na základe tohoto prieskumu boli vybrané vyhovujúce operačné systémy. S ohľadom na možnosti práce bol počet typov operačných systémov zúžený na dva. Ako vhodný operačný systém bol vybraný OS Linux a to vo verzii vanilla a vo verzii s patchom PREEMPT RT.

Následne sa práca zaoberala problémom softvérového návrhu. Ako vhodný sa ukázal koncept realizovaný ako modul jadra využívajúci API časovačov s vysokým rozlíšením. Ako regulačný algoritmus bol zvolený algoritmus PSD. Do algoritmu bolo implementované antiwindup, beznárazové prepínanie a filtrácia derivačnej zložky.

S problémom softvérovej implementácie súvisel výber AD a DA prevodníkov. V práci boli použité prevodníky s paralelnou zbernicou napojené priamo na GPIO piny. Daná hardvérová konfigurácia periférií sa ukázala ako vhodná najmä pre svoju vysokú vzorkovaciu frekvenciu.

V závere práce bola činnosť regulátora overená meraním. Z nameraných hodnôt časových intrevalov bola stanovená minimálna vzorkovacia perióda na jednu milisekundu. Porovnaním nameraných hodnôt prechodovej charakteristiky so simuláciou bol overený algoritmus regulátora.

Namerané hodnoty potvrdili využiteľnosť regulátora pre účely riadenia. Výsledkom práce je fungujúci softvérovo-hardvérový koncept PSD regulátora. Naviac bolo implementované webové HMI rozhranie umožňujúce jednoduchšiu obsluhu regulátorov.

Literatúra

[1] ANALOG DEVICES. *Analog Devices Ahea of what is possible: Data Sheet AD5724/AD5734/AD5754* [online]. Analog Devices, © 2008–2016 [cit. 2018-05-11]. Dostupné z URL: </http://www.analog.com/media/en/technical-documentation/data-sheets/AD5724_5734_5754.pdf>

[2] ARDUINO. Analog read. *Arduino.cc* [online]. © 2018 [cit. 2018-05-12]. Dostupné z URL: </https://www.arduino.cc/reference/en/language/functions/analog-io/analogread/>

[3] BALCI, Mete. Latency of Raspberry Pi 3 on Standard and Real-Time Linux 4.9 Kernel. In: *Medium* [online]. Oct 3, 2017 [cit. 2018-05-11]. Dostupné z URL: </https://medium.com/@metebalci/latency-of-raspberry-pi-3-on-standard-and-real-time-linux-4-9-kernel-2d9c20704495>

[4] BARRY, Richard. Mastering the Free RTOS Real Time Kernel. In: *Free RTOS: A hands-on tutorial guide* [online]. Free RTOS, © 2016 [cit. 2018-05-11]. Dostupné z URL: </http://www.freertos.org/Documentation/161204_Mastering_the_FreeRTOS_Real_Time_Kernel-A_Hands-On_Tutorial_Guide.pdf>

[5] BATE, Stephen. ChibiOS/RT on the Raspberry Pi. In: *Stevebate* [online]. [cit. 2018-05-10]. Dostupné z URL: </http://www.stevebate.net/chibios-rpi/GettingStarted.html>

[6] BENEŠ, Jiří. *Diplomová práce: Komparativní analýza multitaskingových operačních systémů pro embedded aplikace* [online]. [cit. 2018-05-10]. Dostupné z URL: </https://otik.uk.zcu.cz/bitstream/11025/12531/1/A12N0026P_DP.pdf>

[7] BRADÁČ, Zdeněk a Petr FIEDLER. *Embedded aplikace pracující v reálném čase*. 2014, Fakulta elektrotechniky a komunikačních technologií Vysoké učení technické v Brně

[8] BROADCOM CORPORATION. Broadcom Corporation: BCM2835 ARM Peripherals. In: *Raspberry Pi : Raspberry Pi Hardware* online]. Broadcom Corporation, © 2012 [cit. 2018-05- 10]. Dostupné z URL: </https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2835/BCM2835-ARM-Peripherals.pdf>

[9] DEBIAN. RaspberryPi Can I put Debian on my Raspberry Pi? In: *Wiki Debian* [online]. Nov 24, 2016, 9 am [cit. 2018-05-10]. Dostupné z URL: </https://wiki.debian.org/RaspberryPi>

[10] DURR, Frank. Raspberry Pi Going Realtime with RT Preempt. In: *Frank.durr* [online]. Oct 25, 2015 [cit. 2018-05-11]. Dostupné z URL: </http://www.frank-durr.de/?p=203>

[11] EMILD. Raspberry Pi real-time kernel. In: *Emlid* [online]. May 19, 2014 [cit. 2018-05-11]. Dostupné z URL: </https://emlid.com/raspberry-pi-real-time-kernel/>

[12] FREE RTOS. The Free RTOS Kernel. *FreeRTOS.org* [online]. © 2018 [cit. 2018-05-11]. Dostupné z URL: </http://www.freertos.org/index.html>

[13] GOOGLE. *Google Summer of Code Archive: Raspberry PI USB and Ethernet Support* [online].Google, © 2016 [cit. 2018-05-11]. Dostupné z URL: </https://summerofcode.withgoogle.com/archive/2016/projects/5992398500921344/>

[14] JASHWINDER, Singh et al. Frequently Asked Questions. In: *Real-Time Linux Wiki* [online]. Sep 13, 2012, 11 am [cit. 2018-05-10]. Dostupné z URL: </https://rt.wiki.kernel.org/index.php/Frequently_Asked_Questions>

[15] JONES, Tim. Timers and lists in the 2.6 kernel. In: *IBM* [online]. Mar 30, 2010 [cit. 2018-05-12]. Dostupné z URL: </https://www.ibm.com/developerworks/library/l-timers-list/>

[16] KOWALSKI, Richard. Real-Time Operating systems (RTOS) 101. In: *NASA* [online]. [cit. 2018-05-11]. Dostupné z URL: </https://www.nasa.gov/sites/default/files/482489main_4100_-_RTOS_101.pdf>

[17] KRAMER, Martin. Compiling a kernel module for the raspberry pi 2. In: *Lost in details* [online]. Feb 25, 2015 [cit. 2018-05-11]. Dostupné z URL: </http://lostindetails.com/blog/post/Compiling-a-kernel-module-for-the-raspberry-pi-2>

[18] The Linux information project: Multitasking definition *LINUX.* [online]. Linux, 2004 [cit. 2018-05-12] Dostupné z URL: </http://www.linfo.org/multitasking.html>

[19] MAXIM. *MAX 114/MAX118: +5V, 1Msps, 4 & 8-Channel.* Maxim Intergrated Products © 1996

[20] MOLLOY, Derek. Writing a Linux Kernel Module-Part 1: Introduction. In: *DerekMolloy* [online]. Apr 14, 2015 [cit. 2018-05-12]. Dostupné z URL: </http://derekmolloy.ie/writing-a-linux-kernel-module-part-1-introduction/>

[21] NEXPERIA. *74LVC245A,74LVCH245A: Octal bus transciever: Product data sheet.* Nexperia © 2017

[22] Operačné systémy reálneho času. In: *Posterus.sk* [online]. [cit. 2018-05-10] Dostupné z URL: </http://www.posterus.sk/wp-content/uploads/p17008_02_obr02.png>

[23] PHILIPPHE. Life with Adeos . In: *Xenomai* [online]. Jun 2, 2014 [cit. 2018-05-10]. Dostupné z URL: </https://xenomai.org//2014/06/life-with-adeos/>

[24] PIETER, Jan. Low level programming of the Raspberry Pi in C. In: *Pieter-Jan* [online]. May 24, 2013 [cit. 2018-05-12]. Dostupné z URL: </http://www.pieter-jan.com/node/15>

[25] PIVOŇKA, Petr. *Číslicová řídicí technika.* Scriptum 2012, Fakulta elektrotechniky a komunikačních technologií Vysoké učení technické v Brně

[26] RASPBERRY PI. *Raspberry Pi: FAQS* [online]. [cit. 2018-05-10]. Dostupné z URL: </https://www.raspberrypi.org/help/faqs/>

[27] RASPBERRY PI. *Raspberry Pi: Raspberry Pi 2 Model B* [online]. [cit. 2018-05-10]. Dostupné z URL: </https://www.raspberrypi.org/products/raspberry-pi-2-model-b/>

[28] RASPBERRY PI New to RISC OS? Read this! In: *Raspberry pi* [online]. Nov 5, 2012, 12 am [cit. 2018-05-10]. Dostupné z URL: </https://www.raspberrypi.org/forums/viewtopic.php?f=55&t=22093>

[29] RASPBERRY PI. *Raspberry Pi Documentation: SPI* [online]. [cit. 2018-05-11]. Dostupné z URL: </https://www.raspberrypi.org/documentation/hardware/raspberrypi/spi/README.md>

[30] RASPBERRY PI. Freezing with RT-patch (Pi 3). In: *Raspberry Pi* [online]. Sep 6, 2016, 7 pm [cit. 2018-05-11]. Dostupné z URL: </https://www.raspberrypi.org/forums/viewtopic.php?f=29&t=159170>

[31] RASPBERRY PI. Raspberry Pi: *Kernel Building* [online]. [cit. 2018-05-11]. Dostupné z URL: </https://www.raspberrypi.org/documentation/linux/kernel/building.md>

[32] Raspberry Pi 2 Model B. In: *Humblepi.org* [online]. Jan 3 2017 13:14 [cit. 2018-05-10]. Dostupné z URL: </http://humblepi.org/images/3/3b/Raspberry-pi-2.jpg>

[33] REAL TIME ENGINEERS. Real Time Engineers: Free RTOS. In: *Real Time Engineers: corporate presentation* [online]. Real Time Engineers, © 2016 [cit. 2018-05-11]. Dostupné z URL: </http://www.realtimeengineers.com/ReferencedDownloads/Real_Time_Engineers_Ltd_FreeRTOS_Overview.pdf>

[34] RISC OS OPEN. Welcome. *RISC OS Open* [online]. RISC OS Open, © 2011 [cit. 2018-05-10]. Dostupné z URL: </https://www.riscosopen.org/content/>

[35] ROUSE, Margaret. Definition of Interrupt latency. In: Whatis [online]. Sep 2012 [cit. 2018-05-11]. Dostupné z URL: </http://whatis.techtarget.com/definition/interrupt-latency>

[36] RTEMS. Real Time Operating System. *RTEMS: Real time operating system* [online]. RTEMS © 2014 [cit. 2018-05-11]. Dostupné z URL: </https://www.rtems.org/>

[37] RTEMS. Open Source RTOS. *Osrtos.com* [online]. © 2014 [cit. 2018-05-11]. Dostupné z URL: </http://www.osrtos.com/rtos/rtems>

[38] RTEMS. RTEMS License Information. *Rtems.org* [online].© 2014 [cit. 2018-05-11]. Dostupné z URL: </https://www.rtems.org/license>

[39] RTEMS. *RTEMS 4.0.0 On-Line Library: 1.5: RTEMS Internal Architecture* [online]. © 1988-1998 [cit. 2018-05-11]. Dostupné z URL: </https://docs.rtems.org/releases/4.0.0/doc/c_user/a00007.html>

[40] SIRIO, Giovanni. ChibiOS/RT. In: *Chibios* [online]. [cit. 2018-05-10]. Dostupné z URL:
</http://chibios.sourceforge.net/html/>

[41] STOKES, John. ARM's new Cortex A7 is tailor-made for Android superphones. In: *Arstechnica* [online]. Oct 20, 2011, 6 pm [cit. 2018-05-10]. Dostupné z URL:
</http://arstechnica.com/gadgets/2011/10/arms-new-cortex-a7-is-tailor-made-for-android-superphones/>

[42] SVEC, Christopher. Free RTOS. In: *Aosabook* [online]. [cit. 2018-05-11]. Dostupné z URL:
</http://www.aosabook.org/en/freertos.html>

[43] TEXAS INSTRUMENTS. *TLC7226C, TLC7226I, TLC7226M: QUADRUPLE 8-BIT DIGITAL-TO-ANALOG CONVERTERS.* Texas Instruments Incorporated © 2016

[44] WALMSLEY, James et al. Embedded, robust, real- time, operating system. *Bit Thunder* [online]. © 2018 [cit. 2018-05-11]. Dostupné z URL:
</http://bitthunder.org/>

[45] XENOMAI. *Xenomai: Introducing Xenomai 3* [online]. [cit. 2018-05-10]. Dostupné z URL:
</https://xenomai.org//introducing-xenomai-3/>

[46] XENOMAI. *Xenomai: Start Here* [online]. [cit. 2018-05-10]. Dostupné z URL:
</https://xenomai.org/start-here/>

[47] Dostupné z URL:
</https://s-media-cache-ak0.pinimg.com/originals/63/1c/74/631c7441767813c645a250cebc9bbd23.png>

Zoznam symbolov, veličín a skratiek

API	application programming interface
ARM	Acorn RISC Machine
CGI	common gateway interface
CLI	command line interface
CPU	central processing unit
CSI	camera serial interface
FIQ	fast interrupt handling
GCC	GNU Compiler Collection
GDB	GNU Debugger
GNU	GNU's Not Unix!
GPIO	general purpose input/output
GPL	GNU General Public License
HMI	human machine interface
IRQ	interrupt request
I-pipe	interrupt pipe
I2C	inter integrated circuit
OS	operating system
POSIX	Portable Operating System Interface
RAM	random access memory
RMS	rate monotonic scheduling
RTOS	real time system
SIL	Safety integrity level
TCP/IP	Transmission Control Protocol/Internet Protocol
UART	universal asynchronous receiver and transmitter
USB	universal serial bus

Zoznam príloh

A Inštalácia a kompilácia OS Raspbian **58**
A.1 Inštalácia nemodifikovaného OS Raspbian 58
A.2 Kompilácia linuxového jadra 59
A.2.1 Zakázanie FIQ 63
A.3 Testovanie realtimových vlastností na patchovanom OS Linux 63

B Vizualizácie **66**

C Obsah priloženého CD **70**

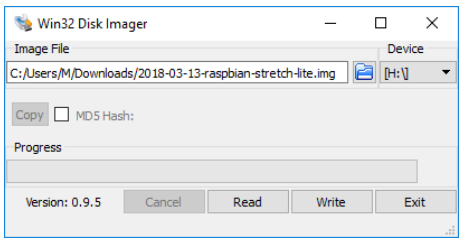
A Inštalácia a kompilácia OS Raspbian

Kapitola pojednáva o inštalácii vanilla a modifikovaného OS Raspbian na platformu Raspberry Pi. Prvá podkapitola opisuje prvotnú inštaláciu vanilla OS Raspbian. Z tejto podkapitoly vychádza aj inštalácia modifikovaného OS Raspbian. Postup inštalácie modifikovaného OS Raspbian využíva štandardný diskový obraz v ktorom sú neskôr prepísané niektoré súbory súbormi novovzniknutými pri kompilácii. Ďalšie podkapitoly sú relevantné iba pre modifikovanú verziu. Posledná podkapitola opisuje odstránenie chyby vzniknutej aplikovaním patchu RT PREEMPT. Jednotlivé kroky sú popísané pre konkrétnu verziu Linuxového jadra Raspbianu použitého pri implementácii. Konkrétne sa jedná o verziu jadra 4.9.47. Jadro Linuxu bolo kompilované pod OS Ubuntu verzie 16.04. Daná téma je esenciálna pre korektnú funkcionalitu práce, preto sú jednotlivé kroky detailne popísané.

A.1 Inštalácia nemodifikovaného OS Raspbian

Pre inštaláciu OS Raspbian na microSD kartu je potrebné najskôr stiahnuť súbor vo formáte .img. Pre embedded aplikácie je vhodnejšia verzia lite, ktorá postráda grafické rozhranie. Daný súbor je možné stiahnuť z oficiálnej webovej stránky nadácie Raspberry Pi Foundation.
`/https://www.raspberrypi.org/downloads/raspbian/`

Následne je potrebné daný obraz disku nainštalovať na microSD kartu. To je možné vykonať komocou programu Win32DiskImager (platí pre OS Windows), alebo pomocou CLI programu dd (platí pre OS Linux). Grafické rozhranie programu Win32DiskImager má intuitívne ovládanie a preto nie je ďalej popisované. CLI program dd má viacero parametrov. Parameter `bs=BYTES` nastavuje počet naraz načítaných a zapísaných bytov, parameter `if=FILE` určuje cestu k obrazu disku, parameter `of=FILE` určuje cestu k microSD karte. Ostatnými parametrami sa nastavujú informačné výpisy o práve prebiehajúcej inštalácii.

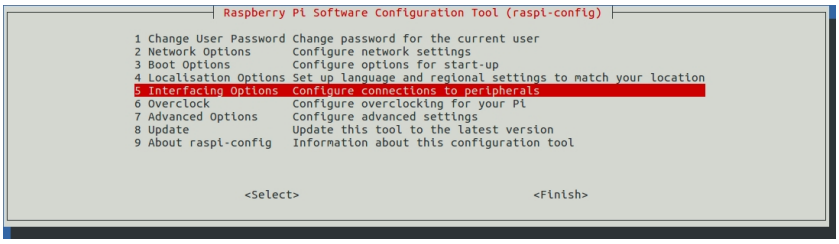


Obr. A.1: Win32DiskImager

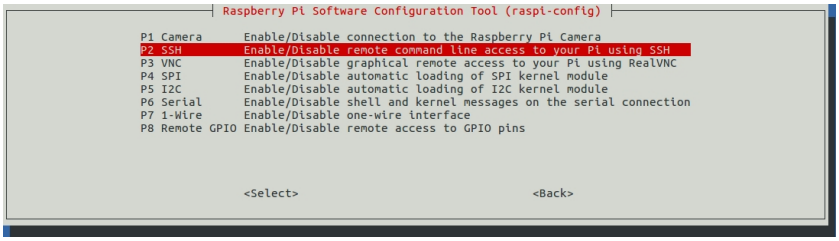
```
$ dd bs=4M if=2018-03-13-raspbian-stretch.img of=/dev/sdX\
status=progress conv=fsync
```

Následne je potrebné vložiť microSD kartu do Raspberry Pi a pripojiť externý monitor. Pri prvotnom bootovaní sa expanduje súborový systém na plnú kapacitu microSD karty. Po dokončení expanzie sa celý systém reštartuje. Východzí užívateľ pre distribúciu je `pi` a heslo je `raspberry`. Po prihlásení je nutné povoliť komunikáciu pomocou ssh rozhrania, ktoré je vo východzom nastavení zakázané. Na to slúži nasledujúci príkaz, ktorý otvorí kontextové menu viď. obr. A.2A.3.

```
sudo raspi-config
```



Obr. A.2: povolenie SSH rozhrania menu 1



Obr. A.3: povolenie SSH rozhrania menu 2

Aplikovaním daného postupu je celý systém pripravený na ďalšie použitie a komunikáciu pomocou ssh rozhrania.

A.2 Kompilácia linuxového jadra

Linuxové jadro môže byť skompilované dvoma spôsobmi, lokálne na platforme Raspberry Pi, alebo tzv. cross-compile pod inou linuxovou distribúciou resp. pod iným hardvérom. Z hľadiska väčšej časovej úspory pri kompilácii bola zvolená druhá možnosť. V prvom rade je potrebné stiahnuť nástroj pre cross-compile. Tento nástroj tzv.

toolchain je možné stiahnuť zo vzdialeného repozitára pomocou gitu. Daný príkaz naklonuje toolchain do domovského adresára.

```
$ git clone https://github.com/raspberrypi/tools ~/tools
```

Následne je potrebné stiahnuť samotné zdrojové kódy linuxového jadra modifikovaného pre platformu RPi. Parameterom `-depth=1` naklonujeme vzdialený adresár iba pre posledný commit. Toto opatrenie je potrebné z hľadiska časovej a diskovej úspory.

```
$ git clone --depth=1 https://github.com/raspberrypi/linux
```

Presunieme sa do linuxového adresára a skontrolujeme verziu linuxového jadra.

```
$ cd linux
$ head -3 Makefile
```

Ak nám daná verzia nesúhlasí pridáme ďalšiu vetvu angl. branch. V tomto prípade už volíme parameter `-depth=1000` väčší, aby bolo možné vybrať konkrétny commit pre patch RT PREEMPT. V ďalšom kroku skontrolujeme naklonované vetvy a prepneme sa do tej, ktorú sme si zvolili.

```
git fetch --depth=1000 git://github.com/raspberrypi/linux.git\
rpi-4.9.y:refs/remotes/origin/rpi-4.9.y
git branch -a
git checkout origin/rpi-4.9.y
```

RT-patche sú vytvorené iba pre určité verzie linuxového jadra. Z tohoto dôvodu je nutné pri výbere verzie najskôr zohľadniť dostupnosť patchu na nasledujúcej webovej stránke.
`/https://mirrors.edge.kernel.org/pub/linux/kernel/projects/rt/`

Nie každý commit danej verzie Linuxového jadra môže byť skompilovaný pre každú verziu platformy Raspberry Pi. Toto je zapríčinené chýbajúcou podporou v niektorých commitoch pre danú architektúru platformy Raspberry Pi. Z tohoto dôvodu je potrebné získať hash firmvérového commitu pre Raspbian. Tieto hashe sú zhodné s hashmi pre oficiálne vydávané distribúcie Raspbianu. Využívajú sa najmä pri upgrade resp. downgrade verzie jadra Raspbianu bez potreby vlastnej kompilácie. Daný hash commitu pre konkrétnu verziu jadra je možné získať z nasledujúcej webovej stránky.
`/https://github.com/raspberrypi/firmware/commits/master`

Z vedomosti hasha firmware commitu získame hash pre konkrétny commit jadra.

Daný hash commitu jadra aplikujeme priamo na platforme na ktorej kompilujeme jadro. Následne sa opäť presvedčíme, či sme sa presunuli do správnej verzie jadra.

```
# Get the git hash for this kernel
$ KERNEL_HASH=$(wget https://raw.github.com/raspberrypi\
/firmware/$FIRMWARE_HASH/extra/git_hash -O -)
# Checkout the files on ubuntu
$ git checkout $KERNEL_HASH
$ head -3 Makefile
```

Alternatívou k danému postupu je získanie správneho firmvérového hasha priamo z nainštalovaného Raspbianu. Nasledujúcim príkazom získame firmwérový hash. Získanie hashu commitu jadra je potom rovnaké, ako je spomínané v postupe uvedenom vyššie [17].

```
# Get the Firmware Hash from the Raspberry Pi
$ FIRMWARE_HASH=$(zgrep "* firmware as of" /usr/share/doc\
/raspberrypi-bootloader/changelog.Debian.gz | head -1 |\
awk '{ print $5 }')
```

Potom čo sa uistíme, že máme správnu verziu zdrojových kódov jadra Linuxu, pokračujeme stiahnutím patchu RT PREEMPT priamo do hlavného adresára.

```
$ wget https://www.kernel.org/pub/linux/kernel/projects/rt\
/4.9/older/patch-4.9.47-rt37.patch.gz
```

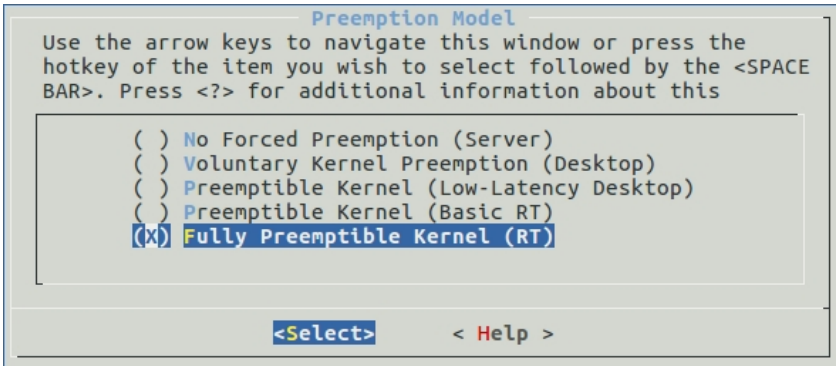
Aplikujeme patch RT PREEMPT nanečisto, aby sme sa presvedčili, že pri aplikácii nedôjde ku chybám.

```
zcat patch-4.9.47-rt37.patch.gz | patch -p1 --dry-run
```

Pre účely zjednodušenia príprav pred kompiláciou bol vytvorený skript `_prepare.sh`. Premenné použité na začiatku skriptu určujú architektúru procesora a cestu k toolchainu. Nasleduje aplikácia RT-patchu a konfigurácia pred kompiláciou. Príkaz `make menuconfig` otvorí kontextové menu.

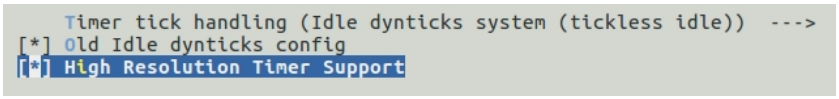
```
#!/bin/bash

export KERNEL=kernel7
export ARCH=arm
export CROSS_COMPILE=/home/$USER/tools/arm-bcm2708\
/gcc-linaro-arm-linux-gnueabihf-raspbian-x64/bin\
/arm-linux-gnueabihf-
zcat patch-*.patch.gz | patch -p1
make bcm2709_defconfig
make menuconfig
```

Pre dosiahnutie vlastností reálneho času je potrebné nastaviť v kontextovom menu dva parametre. Prvý parameter nastavuje plne preemptívne linuxové jadro. V kontextovom menu vyberieme nasledovné: `Kernel Features -> Preemption Model (Fully Preemptible Kernel (RT)) -> Fully Preemptible Kernel (RT)`.



Obr. A.4: SSH enable 2

Druhý parameter povoľuje časovače s vysokou presnosťou. Vo východzom nastavení by táto vlastnosť mala byť povolená. V kontextovom menu vyberieme nasledovné: `General setup -> Timers subsystem -> High Resolution Timer Support` [10].



Obr. A.5: SSH enable 2

Nasleduje samotná kompilácia linuxového jadra. Daným príkazom skompilujeme linuxové jadro, moduly jadra a súbory device tree. Za zmienku stojí aj parameter `-j6` ktorý umožňuje paralelizáciu pri kompilácii. Odporúčaná hodnota je 1.5 * počet jadier procesora na ktorom sa kompiluje jadro linuxu. V našom prípade to je 4 * 1.5, teda 6 [31].

```
make ARCH=arm CROSS_COMPILE=/home/$USER/tools/arm-bcm2708\
/gcc-linaro-arm-linux-gnueabihf-raspbian-x64/bin\
/arm-linux-gnueabihf- zImage modules dtb -j6
```

Po skompilovaní jadra je potrebné vytvorené súbory skopírovať na microSD kartu, to zaobstará skript s názvom `_upload.sh`. Tento skript je potrebné nakopírovať do hlavného adresára zdrojových kódov. Správnu verziu jadra skontrolujeme na platforme Raspberry Pi pomocou nasledujúceho príkazu s očakávanou odpoveďou.

```
uname -a

Linux raspberrypi 4.9.47-rt37-v7+ #1 SMP PREEMPT RT
Fri Feb 23 13:13:56 CET 2018 armv7l GNU/Linu
```

A.2.1 Zakázanie FIQ

Pri testovaní vlastností reálneho času pomocou nástroja cyclictest som zistil, že operačný systém po niekoľkých minútach testovania "zamrzne". Táto chyba sa vyskytuje v kombinácii FIQ, angl. fast interrupt handling, využívaného pre USB zbernicu, a patchu RT PREEMPT. Preto som FIQ zakázal v súbore `/boot/cmdline.txt` pridaním nasledujúcich parametrov [30].

```
dwc_otg.fiq_fsm_enable=0 dwc_otg.fiq_enable=0
dwc_otg.nak_holdoff=0
```

A.3 Testovanie realtimových vlastností na patchovanom OS Linux

Pri testovaní je potrebné čo najviac zaťažiť jednotlivé CPU procesora, aby sme dostali čo najrelevantnejšie výsledky. To dosiahneme preposielaním údajov z virtuálneho súboru `/dev/zero` do druhého virtuálneho súboru `/dev/null`. Tento príkaz bol spustený ako záťaž v štyroch inštanciách. Ako dodatočná záťaž bol použitý príkaz `ping`, ktorý bol spustený z externého počítača. Parametrom `-i 0.01` sme nastavili interval medzi jednotlivými pingami na 0.01s.

```
#pustené 4x na RPi
$ cat /dev/zero > /dev/null

#pustené z externého pc
$sudo ping -i 0.01 10.42.0.68
```

Po vykonaní horeuvedených príkazov sa môžeme príkazom `htop` presvedčiť, že všetky štyri CPU procesora sú plne vyťažené.

Obr. A.6: výpis programu htop

Následne môžeme pristúpiť k samotnému meraniu latencie pomocou programu `cyclictest`. Daný program je potrebné spustiť ako superužívateľ. Programom `time` odmeráme trvanie testu. Výsledky merania sú po dokončení zapísané do súboru `out.txt`. Nasleduje popis jednotlivých parametrov [3].

- `-l10000000` nastavenie počtu iterácii merania na 10 miliónov
- `-m` zakázanie stránkovania pamäte
- `-S` ekvivalentné s parametrami `-tan`, použitie clock_nanosleep a vytvorenie tolko vlákien koľko je jadier procesora a zaistenie, aby bolo každé vlákno spustené na inom jadre
- `-p90` nastavenie priority prvého vlákna, ostatné vlákna majú zostupné hodnoty priorít (90, 89, 88, 87)
- `-i200` časový interval v us
- `-h400` vytvorenie hystogramu so 400 hodnotami
- `-q` potlačenie výpisu počas prebiehajúceho merania

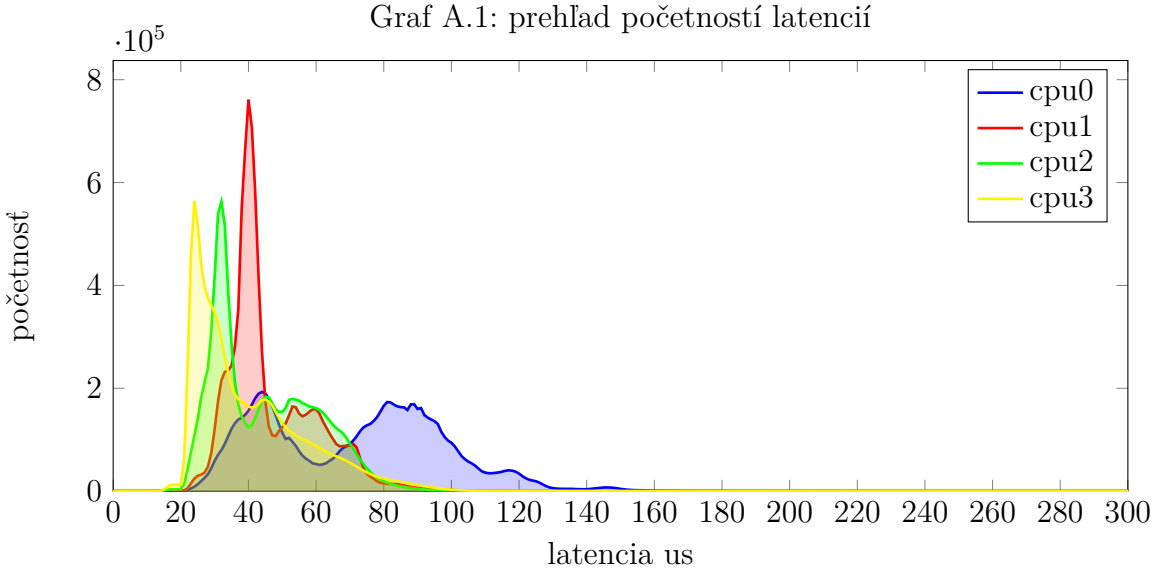
```
$ time sudo ./cyclictest -l10000000 -m -S -p90 -i200 -h400 -q\
> out.txt
```

Dĺžka trvania testu bola približne 33 minút. Výsledky testu je vidieť v tab. A.1, ktorá udáva minimálne maximálne a priemerné hodnoty. Grafické znázornenie výsledkov poskytuje graf A.3.

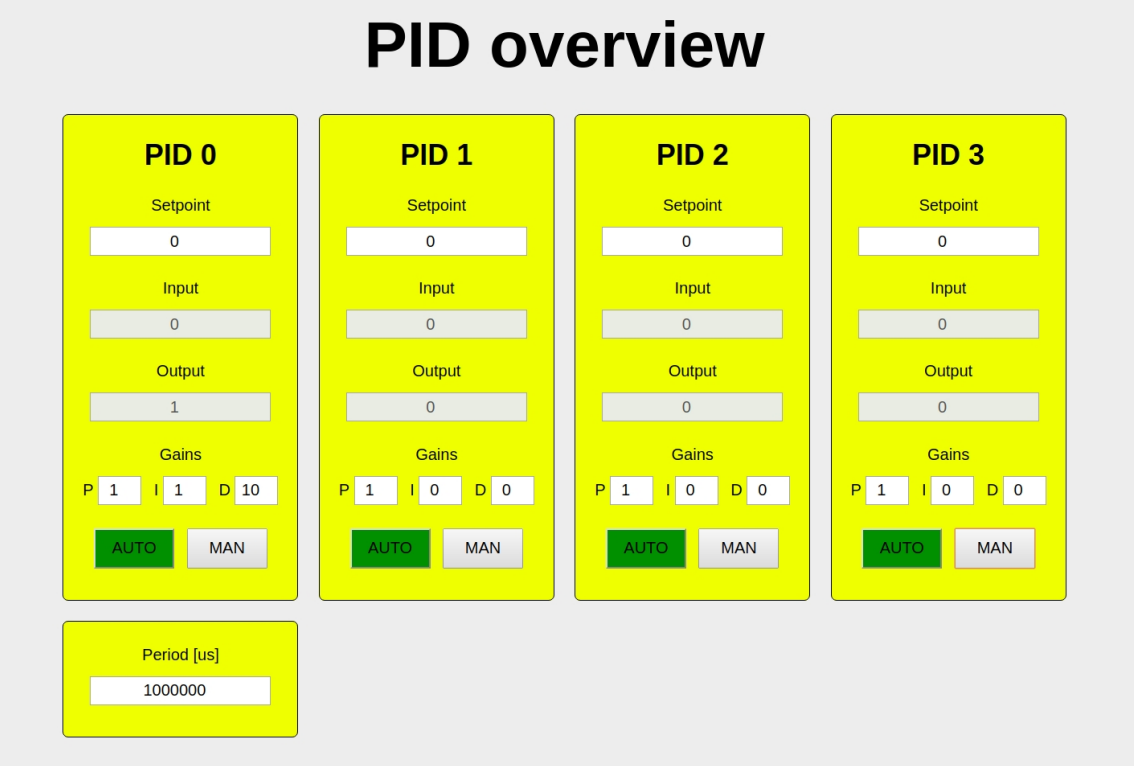
Tab. A.1: Prehlad meraných latencií pre jednotlivé CPU

`latencia`	CPU 0	CPU 1	CPU 2	CPU 3
`t_min[us]`	10	14	13	13
`t_avg[us]`	71	46	44	40
`t_max[us]`	286	227	173	208

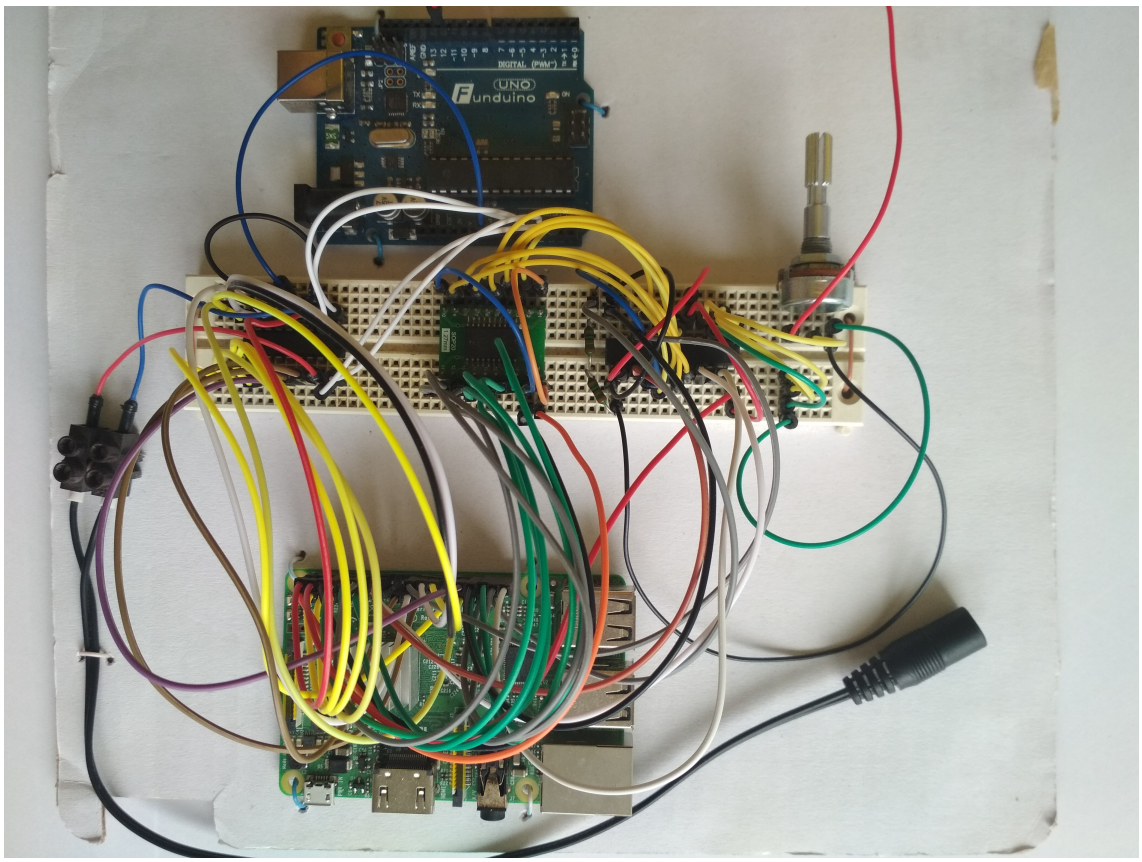
Graf A.1: prehľad početností latencií



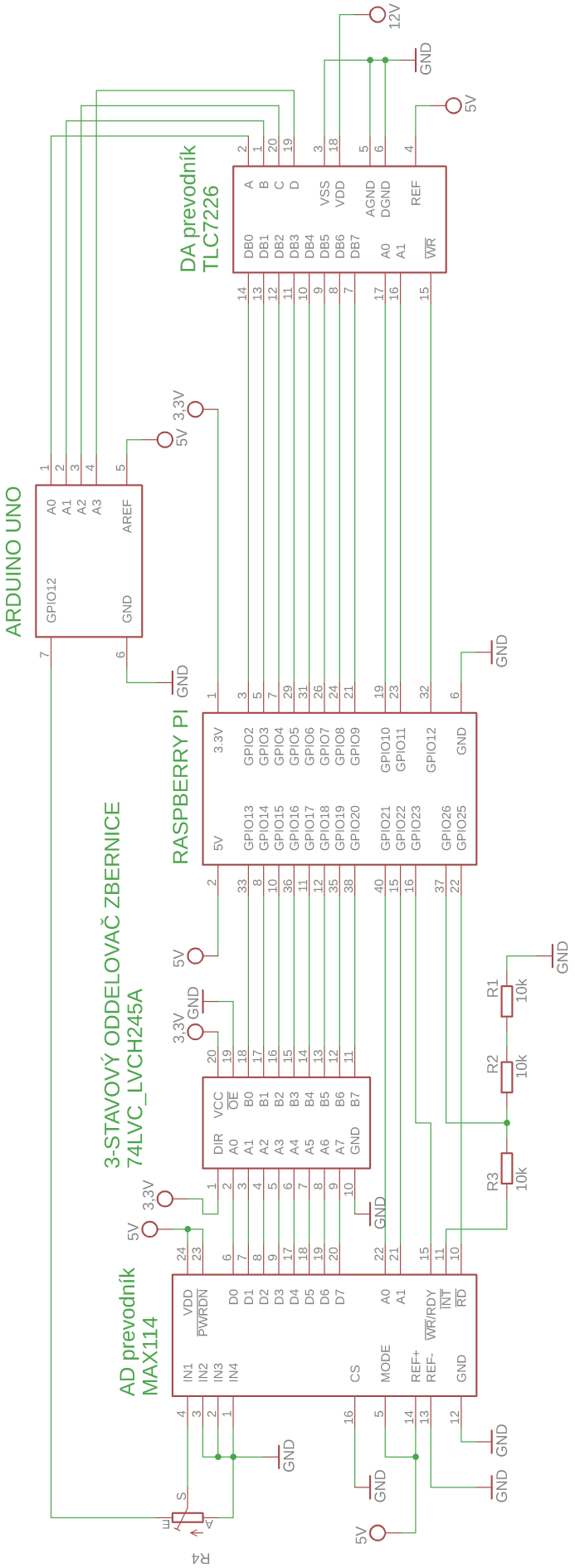
B Vizualizácie



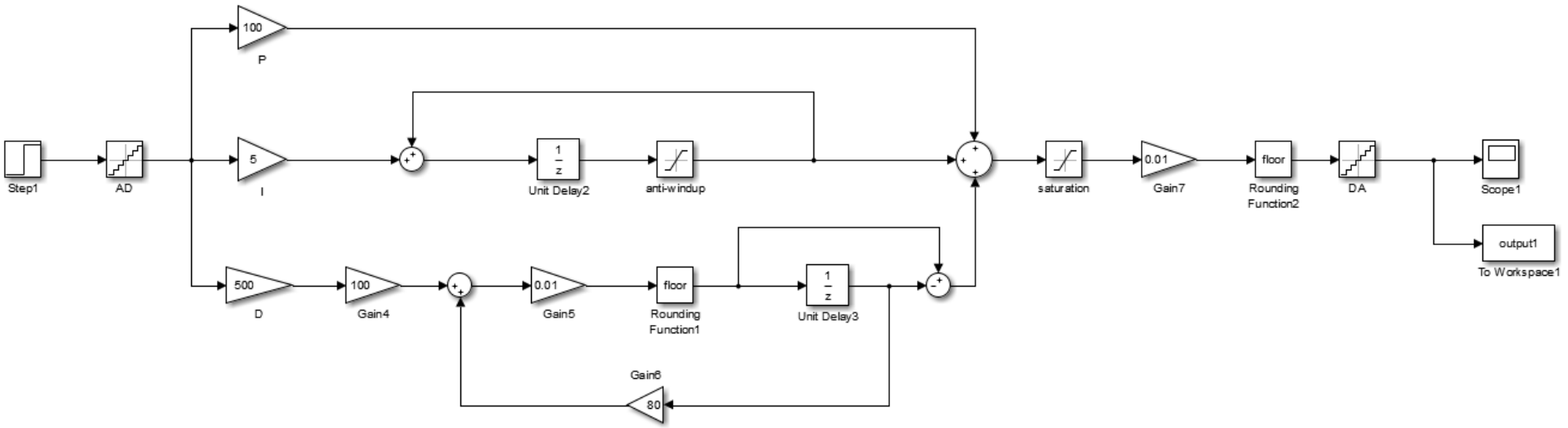
Obr. B.1: HMI webové rozhranie



Obr. B.2: fyzická implementácia regulátora



Obr. B.3: schéma zapojenia



Obr. B.4: schéma simulácie v matlabe

C Obsah priloženého CD

- / ..... koreňový adresár priloženého CD
  - kompilacia_jadra ..... pomocné súbory pri kompilácii jadra
    - _prepare.sh
    - _upload.sh
  - meranie_casov ..... meranie časových úsekov regulátora
    - vycitanie_merani ..... vyčítanie údajov zo znakového zariadenia
      - Makefile
      - readout_time_stamps.c
    - vysledky_merani.ods
  - meranie_prechodovej_funkcie ..... zdrojový súbor pre arduino
    - meranie_prechodovej_funkcie.ino
  - modul_jadra ..... zdrojové súbory modulu jadra
    - Kbuild
    - Makefile
    - src
      - io.c
      - io.h
      - main.c
      - pid.c
      - pid.h
      - pin_tag.c
      - pin_tag.h
  - simulacia_matlab ..... simulácia prechodovej funkcie
    - R2015b
      - pid_model.slx
  - web ..... súbory webového rozhrania
    - cgi
      - readout.sh
      - set.sh
    - html
      - css
        - style.css
      - js
        - script.js
      - index.html
  - text ..... zdrojové súbory latexu