České vysoké učení technické
Fakulta Elektrotechnická
Katedra elektrických pohonů a trakce

Program: Elektrotechnika, energetika a management
Obor: Aplikovaná Elektrotechnika

Aplikace Raspberry Pi pro řízení pohonů

BAKALÁRSKA PRÁCA

Autor: Filip Janík
Vedúci: Ing. Jan Bauer, Ph.D.
Odovzdané: Máj 2018

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Janík** Jméno: **Filip** Osobní číslo: **453139**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra elektrických pohonů a trakce**

Studijní program: **Elektrotechnika, energetika a management**

Studijní obor: **Aplikovaná elektrotechnika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Aplikace Rapsberry PI pro řízení pohonů

Název bakalářské práce anglicky:

Pokyny pro vypracování:

1) Prostudujete možnosti aplikace Rapsberry PI v pohonech
2) Popište kroky potřebné pro využití Rapsberry PI jako mikrokontrolér
3) Vytvořte ukázkovou aplikaci řízení pohonu s Rapsberry PI

Seznam doporučené literatury:

[1] HEROUT, Pavel. Učebnice jazyka C. Praha, 2014
[2] Applikační listy RapsberryPi, Arduino dostupné online
[3] Häberle, H. a kol. Průmyslové elektronika a informační technologie, EUROPA - SOBOTÁLES, Praha, 2003

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Jan Bauer, Ph.D., katedra elektrických pohonů a trakce FEL

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

Datum zadání bakalářské práce: **26.04.2018** Termín odevzdání bakalářské práce: 25.5.2018

Platnost zadání bakalářské práce: **30.09.2019**

Ing. Jan Bauer, Ph.D.
podpis vedoucí(ho) práce

podpis vedoucí(ho) ústavu/katedry

prof. Ing. Pavel Ripka, CSc.
podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

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

Datum převzetí zadání

Podpis studenta

© ČVUT v Praze, Design: ČVUT v Praze, VIC

Prohlášení

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

V Praze dne ....

Filip Janík

Název práce:

Aplikace Raspberry Pi pro řízení pohonů

Autor: Filip Janík

Obor: Aplikovaná Elektrotechnika

Druh práce: Bakalářská práce

Vedoucí práce: Ing. Jan Bauer, Ph.D.
Katedra elektrických pohonů a trakce, ČVUT v Praze, FEL

Abstrakt: Požiadavky kladené na moderné systémy riadenia elektrických pohonov nie sú zamerané iba na ich reguláciu. Rozširujú sa napríklad o zber dát o chode, integráciu v rámci väčšieho systému, prístup k ovládaniu prostredníctvom bezdrátových sietí a mnoho ďaľších. Tieto požiadavky často nútia návrhára siahnuť po riešeniach, ktoré spájajú svet typickej silnoprúdovej elektrotechniky a počítačových systémov. Práca sa zaoberá Raspberry Pi a jeho uplatnením v rámci riešení týchto výziev. Cieľom je preskúmať možnosti tejto platformy, ako aj popísať spôsob práce s ňou, jej charakteristiky a poskytnúť komplexný pohľad na ňu. Získané znalosti sú následne aplikované na modelovej aplikácii, ktorá využíva Raspberry Pi a mikrokontroler Arduino pre riadenie pohonu v jednoduchom robotovi.

Klíčová slova: RaspberryPi, Arduino, embedded systems, multithreading, Linux

Title:

Application of Raspberry Pi for electrical drive systems

Author: Filip Janík

Abstract: Requirements upon modern electrical drive systems go beyond their regulation. They are quickly expanding to accommodate collecting data, integration within larger systems, connectivity via wireless networks and more. These requirements often result in need to use solutions which lay somewhere between traditional power engineering and computer systems. The topics explored in this thesis are concerned with application of Raspberry Pi as a solution to these problems. The target is to explore the characteristics and capabilities of the platform and describe processes needed to utilize them, to provide a complex perspective. Acquired skills are applied to a model application, consisting of Raspberry Pi and an Arduino microcontroller, as a means to manipulate movement of a simple robot.

Key words: RaspberryPi, Arduino, embedded systems, multithreading, Linux

Obsah

Kapitola 1

Úvod

Schopnosti a možnosti výpočtovej techniky rastú každým rokom viac a viac. Na trh sa dostáva mnoho kvalitných nízkonákladových riešení obsahujúcich plnohodnotné procesory, oplývajúce dostatočným výkonom pre pokrytie funkcie osobného počítača. Jedným z takýchto riešení je aj Raspberry Pi. Toto integrované zariadenie je s množstvom vyše 19 miliónov predaných kusov najpredávanejším britským počítačom v histórii [1]. Jeho potenciálne využitie zďaleka nekončí pri osobných počítačoch. Vďaka miniaturizácii elektroniky sa posúvajú hranice a rozširujú možnosti, akými možno ovládať pohony, zbierať dáta o ich operácii a automatizovať ekosystémy, v ktorých pracujú. Cieľom tejto práce je preskúmať, ako možno aplikovať Raspberry Pi v rámci tejto paradigmy.

1.1 Výzvy v modernom riadení pohonov

1.1.1 Požiadavky na konektivitu a integráciu

Pri pohľade na požiadavky kladené na riešenia implementované na riadenie pohonov sa čoraz viac vyskytuje možnosť vzdialeného monitorovania stavu riešenia, či už prostredníctvom internetu alebo v rámci lokálnej siete. Takáto požiadavka je náročná, ak nie nemožná pre bežný mikrokontroler, oveľa lepším riešením je medzičlánok medzi mikrokontrolerom a vonkajším svetom, ktorým môže byť práve Raspberry Pi.
Raspberry Pi poskytuje veľký rozsah integrovaných modulov sprostredkúvajúcich takúto konektivitu a možnosť pripojenia mnohých druhov sériových komunikačných liniek, komunikujúcich s podradenými mikrokontrolermi, robí tento prístup naozaj atraktívnym. Takáto abstrakcia od člena, ktorý naozaj sprostredkúva riadenie, takisto uľahčuje zmeny do celého systému, kedy netreba meniť konfigurácie jednotlivých mikrokontrolerov, keďže ich komunikácia je vedená nepriamo. Samozrejme, zaradením každého ďalšieho členu sa

zvyšuje latencia komunikácie, spôsobená napríklad maximálnou rýchlosťou zberníc, dobou vykonania inštrukcií, ktoré sú zodpovedné za túto komunikáciu medzi jednotlivými podradenými členmi, a v prípade, že zariadenie, ktoré túto komunikáciu sprostredkúva, je tak komplexné ako napríklad navrhované Raspberry Pi, réžia operačného systému.

1.1.2 Doba a cena uvedenia konceptu k prototypu

Medzi jeden z hlavných faktorov rozhodujúcich o uskutočniteľnosti návrhu rozhodne patrí aj cena. Raspberry Pi je schopné zredukovať túto cenu tým, že svojou podporou platformovo agnostických implementácií [1] sa značne urýchľuje vývoj, ako aj obecne nízka cena oproti iným riešeniam, poskytujúcim podobné možnosti [2]. Tieto fakty robia z Raspberry Pi ideálneho kanditáta na rýchle prototypovanie a v niektorých prípadoch aj ako konečnú platformu. Taktiež množstvo dostupných open-source knižníc je rozhodne výhodou - s ich pomocou sa ďalej znižuje doba a náklady vývoja.

[1] Pri použití operačného systému.

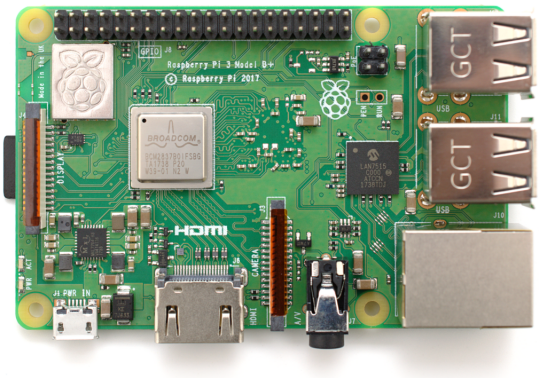
[2] V dobe písania v Českej Republike cena približne 1000 Kč. [7]

Kapitola 2

Charakteristiky Raspberry Pi

2.1 Raspberry Pi ako platforma

Raspberry Pi je platforma založená na SoC (Silicon on Chip) riešení Broadcom, v prípade ďalej v práci spomínaného a využívaného modelu 3B+ konkrétne BCM2837B0, obsahujúceho procesor ARM Cortex-A53. Úložisko je sprostredkované prostredníctvom SD karty. Obsiahnutá je sieťová konektivita cez ethernetový port a Wi-Fi, medzi ďalšie rozhrania nachádzajúce sa na tomto počítači patrí HDMI port, USB porty a GPIO piny, ktoré okrem iného natívne podporujú pripojenie cez SPI a I^2C zbernice [2]. Celé toto vybavenie je dostupné v rámci kompaktnej jednotky, ktorá sa zmestí do dlane.



Obr. 2.1: Raspberry Pi 3 Model B+, prevzaté z [1]

2.2 Architektúra Raspberry Pi

Procesor v Raspberry Pi 3B+ je architektúry ARM (Advanced RISC Machine)[3] z rodiny RISC - Reduced Instuction Set Computing - čitateľovi zrejme najznámejšia ako architektúra, ktorá momentálne drží monopol vo svete smartfónov. Oproti rodine CISC (Complex Instruction Set Computing), ktorej hlavným predstaviteľom je architektúra x86, používaná vo väčšine bežných PC, sa vyznačuje, ako už názov hovorí, redukovanou inštrukčnou sadou.
Hlavným charakterom redukovanej inštrukčnej sady je nízky počet cyklov procesora, potrebných na vykonanie inštrukcie v porovnaní s CISC, každá z inštrukcií je prevedená v rámci jedného cyklu. ARM taktiež nasleduje model load-and-store - každá operácia medzi dvomi objektami v pamäti vyžaduje explicitné načítanie z pamäti do registrov, vykonanie operácie a opätovné uloženie dát späť z registrov. Priamym dôsledkom jednoduchších inštrukcií je typicky zmenšenie počtu tranzistorov pretože sa znižuje komplexnosť hardvérovej logiky, potrebnej pre vykonanie inštrukcie, zmenšených nárokov na napájanie a tým pádom aj menšie množstvo stratového výkonu, ktorý je nutné chladiť [1].

2.3 Limity platformy

Ak chceme využiť Raspberry Pi pre priame riadenie pohonu, existujú určité obmedzenia. Ak neberieme ohľad na náročnosť práce s komplexnou architektúrou - hlavným obmedzením je fakt, že mimo sériových zberníc Raspberry Pi neoplýva potrebnými komponentami potrebnými pre reguláciu. Neobsahuje napríklad analógové vstupy kvôli absencii AD prevodníka, má pomerne nízky počet pinov, ktoré možno využiť na pripojenie ostatných zariadení priamo bez použitia zbernice a podobné funkcionality, bežne dostupné v tradičných mikrokontroleroch.
Hoci je samozrejme možné použiť systém ako "bare metal", t.j. bez operačného systému, jeho cieľom, ako už písmeno A (application profile) v názve procesoru nasvädčuje, je flexibilita.
Hlavnou dizajnovou črtou odlišnou od ostatných profilov je Memory Management Unit(MMU), poskytujúci napríklad virtuálnu pamäť , čo je prakticky nevyužiteľné vybavenie pre účely regulácie ale zároveň vitálna súčasť moderných operačných systémov. Na priame použitie sú vhodnejšie profily -R (Real-Time), ktorá poskytuje okresanú verziu MMU s podporou ochrany pamäte, a -M (Microcontroller), ktorý je zameraný na rýchlosť obsluhy prerušení, nízku spotrebu energie a integráciu s FPGA (Field-programmable Gate Array) [5].

[1] Celková spotreba tohto riešenia pri maximálnej záťaži nepresahuje 7 wattov. [4]

To síce neznamená, že Raspberry nemožno využiť pre takéto aplikácie, ale je nutné brať na vedomie jeho limity a use case, s ktorým bol tento systém vyvinutý. Architektúra A nebola vyvinutá so zameraním na garanciu tvrdej odozvy v reálnom čase - táto je v priamom rozpore s maximalizáciou priepustnosti.
Pomerne zložitá štruktúra Cortex -A profilov v porovnaní s mikrokontrolerom rozdeľuje prístupy k využitiu na dva tábory - operáciu bez operačného systému, využitie ARM assembleru a čiastočne programovať v jazyku symbolických inštrukcií a týmto využiť nízkoúrovňové možnosti platformy, alebo použiť operačný systém. Do istej limitovanej miery je možné skombinovať oba tieto prístupy, ale mali by byť vhodne zvolené vzhľadom k riešenému problému.
Nezanedbateľnou veličinou je taktiež energia ktorú zariadenie spotrebúva pre svoj chod, s čím je spojená aj nutnosť chladenia, ak sa procesor rozhodneme zaťažiť v plnom rozsahu jeho možností. Tento faktor nabúda na dôležitosti ak vyvíjame zariadenie, ktoré nemá byť pripojené ku sieti ale bude napájané z batérií.

Kapitola 3

Možnosti aplikácie Raspberry Pi v pohonoch

3.1 Raspberry Pi ako samostatná jednotka

Napriek vyššie spomenutým obmedzeniam je Raspberry Pi použiteľné ako samostatná jednotka (bez mikrokontroleru) na reguláciu pohonu v jednoduchých, nekritických aplikáciách. Pri požiadavkách na vyššiu konektivitu sa nedostatky dajú ošetriť vďaka množstvu lacných, ľahko pripojiteľných a dostupných prídavných zariadení. S ich pomocou je možné rozšíriť konektivitu k pohonom napríklad prostredníctvom PWM/Servo Driver-u alebo AD prevodníku, ktoré sú predpripravené na jednoduché pripojenie pomocou sériových zberníc. Samozrejme, existuje možnosť navrhnúť vlastné riešenie a jednoducho ho pripojiť s ľubovoľným GPIO pinom, vyvedeným z dosky Raspberry Pi. Ak sa rozhodneme použiť operačný systém, pri jeho vhodnom zvolení a konfigurácii sa dá dosiahnuť rešpektovateľná úroveň odozvy v reálnom čase a použiť Raspberry Pi na zastúpenie mikrokontroleru v obmedzenom rozsahu.
Druhou cestou je priame použitie jazyka symbolických inštrukcií v spolupráci s jazykom C. Týmto prístupom sa možno priblížiť k funkcionalite mikrokontroleru ešte bližšie. Otázkou však zostáva, či je použitie takto komplexného systému vhodnou voľbou na nízkoúrovňové riešenia, vzhľadom na náročnosť práce v ňom pre dosiahnutie podobných výsledkov, ako aj spotreby energie, ceny a spoľahlivosti. Taktiež sa zvyšuje časová náročnosť uskutočniteľnosti niektorých úloh z pohľadu času stráveného vývojom - obrovskou výhodou abstrakcie od hardvéru je možnosť platformovo-agnostického programovania, začína sa rozširovať množstvo použiteľných knižníc a portabilita napísaného softvéru.

3.2 Raspberry Pi v spolupráci s mikrokontrolerom

Spolupráca s mikrokontrolerom je prístup, ktorý je vhodný pre komplikované systémy. Jeho výhodou je využitie predností oboch platforiem, jasné rozdelenie zodpovedností a značné zjednodušenie vývoja. Každá platforma je z princípu vhodná na iné účely - mikrokontroler ľahšie poskytuje priamy deterministický výsledok operácií aj za použitia vyšších jazykov, ľahší prístup k časovačom a ich konfiguráciu, jeho konektivita je zameraná na interakciu s analógovým svetom. Oveľa nižšia latencia prerušení umožňuje rýchlu odozvu na vstupy a tým pádom plynulejšiu reguláciu.
Raspberry Pi je potom možné využiť na úlohy ktoré vyžadujú paralelizáciu a súbežnú obsluhu procesov, výpočtovo náročné operácie, ako napríklad obsluha vstupov cez periférie, vykreslovanie grafiky užívatelského rozhrania alebo konektivitu cez bezdrôtové siete bez nutnosti zložitej konfigurácie. Týmto prístupom sa taktiež redukuje možnosť nežiadaného krížového ovplyvňovania jednotlivých úloh, zjednodušenie údržby a pridávania nových funkcionalít.
Nevýhodou je, že časť výkonu oboch jednotiek v systéme je spotrebovaný réžiou komunikácie medzi nimi, výrobná aj prevádzková cena takéhoto riešenia omnoho prevyšuje integrované riešenie a pribúda počet miest v systéme, kde môže prísť k zlyhaniu.

3.3 Prístup k využitiu Raspberry Pi

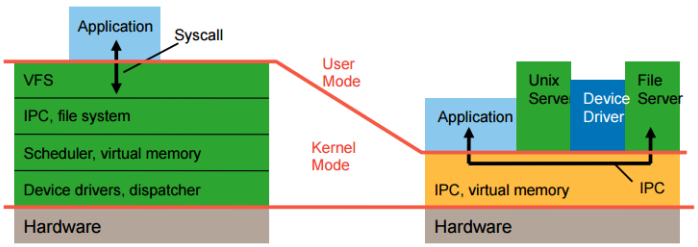
Charakteristiky spomenuté v kapitole 2 poukazujú na fakt, že pre dosiahnutie plných možností, poskytovaných touto platformou je vhodné použiť operačný systém. Existuje mnoho operačných systémov pre Raspberry, či už s natívnou podporou alebo portnuté, ktoré spadajú do rôznych sekcií spektra real-time - napríklad ChibiOS/RT, RISC OS, FreeRTOS a veľká skupina založená na Linuxovom kerneli, napríklad framework Xenomai alebo originálna distribúcia pre Raspberry Pi Raspbian, ktorá s úpravami poskytuje základné možnosti pre real-time.
Bohužiaľ, podpora skutočných real-time OS na RaspberryPi3B+ je práve kvôli použitému procesoru obmedzená a často sa jedná o neoficiálne porty bez podpory. Kritériom, ktoré treba zohľadniť pri výbere OS je komplexita úloh potrebných na zvládnutie riešenia. Samozrejme, ide o inverznú závislosť medzi náročnosťou úloh a schopnosťou riešenia poskytnúť tvrdú real-time odozvu, a tým sa dostávame k najkritickejšej časti návrhu: k rozdeleniu úloh v rámci kapabilít platformy, ktorá je prevedená na konkrétnom príklade v kapitole 4.

3.3.1 Bez operačného systému

Procesor Cortex A-53 je 64-bitový a podporuje viacero inštrukčných sád - 32-bitové A32 a mix 16- a 32-bitových nazvané T32 - "Thumb", čo sú pôvodné inštrukčné sady zdieľané s predošlými generáciami a novú inštrukčnú sadu A64. Táto umožňuje pristupovať k oveľa väčšej pamäti, avšak jej prínos pre naše potreby je takmer nulový. [6]
Ak chceme využiť Raspberry Pi takýmto spôsobom, nie je nutné byť plne zoznámený so všetkými špecifikáciami procesora - toolchain kompilátora sa vo väčšine prípadov o to postará za nás. Ak však chceme pristupovať napríklad k časovačom alebo meniť stavy výstupov s maximálnou optimálnosťou a predvídateľnosťou, je vhodné poznať spôsob fungovania na najnižšej úrovni. Veľká zbierka príkladov poskladaná komunitou je dostupná napríklad v repozitároch na github.com. [8]

3.3.2 S operačným systémom

Operačné systémy umožňujú využiť opačnú časť spektra funkcionalít poskytovaných touto platformou. Spotredkúvajú abstrakciu od hardvéru, spravujú alokáciu pamäti a procesy a obsahujú plánovač, ktorý nám umožnuje jednoducho využívať vlákna a zaručuje optimálne využitie procesoru.
Hlavnými predstaviteľmi, ktorí podporujú Raspberry Pi, je Windows IoT Core [9] a Raspbian [10], oficiálny operačný systém pre Raspberry Pi. S ich použitím sa Raspberry Pi stáva plnohodnotným PC. Tieto systémy v defaultnej konfigurácii žiadnym spôsobom nezaručujú odozvu v reálnom čase pre užívateľské programy - bolo by to v priamom konflikte s ich účelom, pretože garantovanie záruk znižuje priepustnosť systému. Existuje však možnosť ich modifikácie, ktorou tieto charakteristické znaky možno pozmeniť.
Operačné systémy možno rozdeliť do dvoch kategórií podľa implementácie kernelu, na OS založené na mikrokerneli a tie založené na monolitickom kerneli.



Obr. 3.1: Rozdiely v kernelových návrhoch, prevzaté z [11]

Rozdiely sa dajú zhrnúť takto - operácie sprostredkúvajúce funkcionalitu kernelu sú

v prípade mikrokernelu rozdelené na viacero procesov, zvaných servery, ktoré sa nachádzajú v oddelených adresných priestoroch a sú na sebe nezávislé. Monolitický kernel vystupuje ako jeden proces, v jednom adresnom priestore. Dá sa povedať, že monolitický kernel je rýchlejší - zdieľaný adresný priestor redukuje čas vykonania procedúry, keďže každá procedúra má priamy prístup ku všetkým ostatným. Výhodou mikrokernelu je nezávislosť serverov - zlyhanie jedného z nich nezapríční zrútenie celého kernelu. Úplný popis týchto dvoch dizajnových prístupov a ich implikácií presahujú rozsah tejto práce, preto sa nimi ďalej nebudeme zaoberať.

3.3.3 S operačným systémom pracujúcim v reálnom čase

Kompromisom medzi dvomi predošlými prístupmi je použitie upraveného operačného systému. Tieto modifikované verzie taktiež možno rozdeliť podľa kernelu - predstaviteľom mikrokernelového prístupu sú napríklad operačné systémy ChibiOS/RT [12] a FreeRTOS [13].
Hlavný operačný systém, ktorého ďalej skúmame, Raspbian, je založený na monolitickom Linux kerneli.
Špeciálnym prípadom zmiešaného prístupu je Xenomai [14]. Tento systém obsahuje dve konfigurácie. Prvá konfigurácia je zložená z dvoch súbežne bežiacich kernelov. Jeden z nich tvorí Xenomai RTOS jadro, ktoré spolupracuje s tradičným Linux kernelom, pomocou rozhrania, ktoré tento framework poskytuje. Druhá konfigurácia je nadstavbou RT_PREEMPT, čo je dlhoročný projekt, ktorý založil Ingo Molnár a Thomas Gleixner ako rozšírenie tradičného Linux kernelu[1]. Hoci Xenomai v dobe písania práce ešte nie je plne funkčný na Raspberry Pi 3B+, v prípade úspešného portu môže byť práve on ideálnou voľbou naprieč spektrom aplikácií popísaných v kapitole 3.

[1] Tento projekt a jeho aplikácia na Raspbian je ďalej popísaná v tejto práci.

Kapitola 4

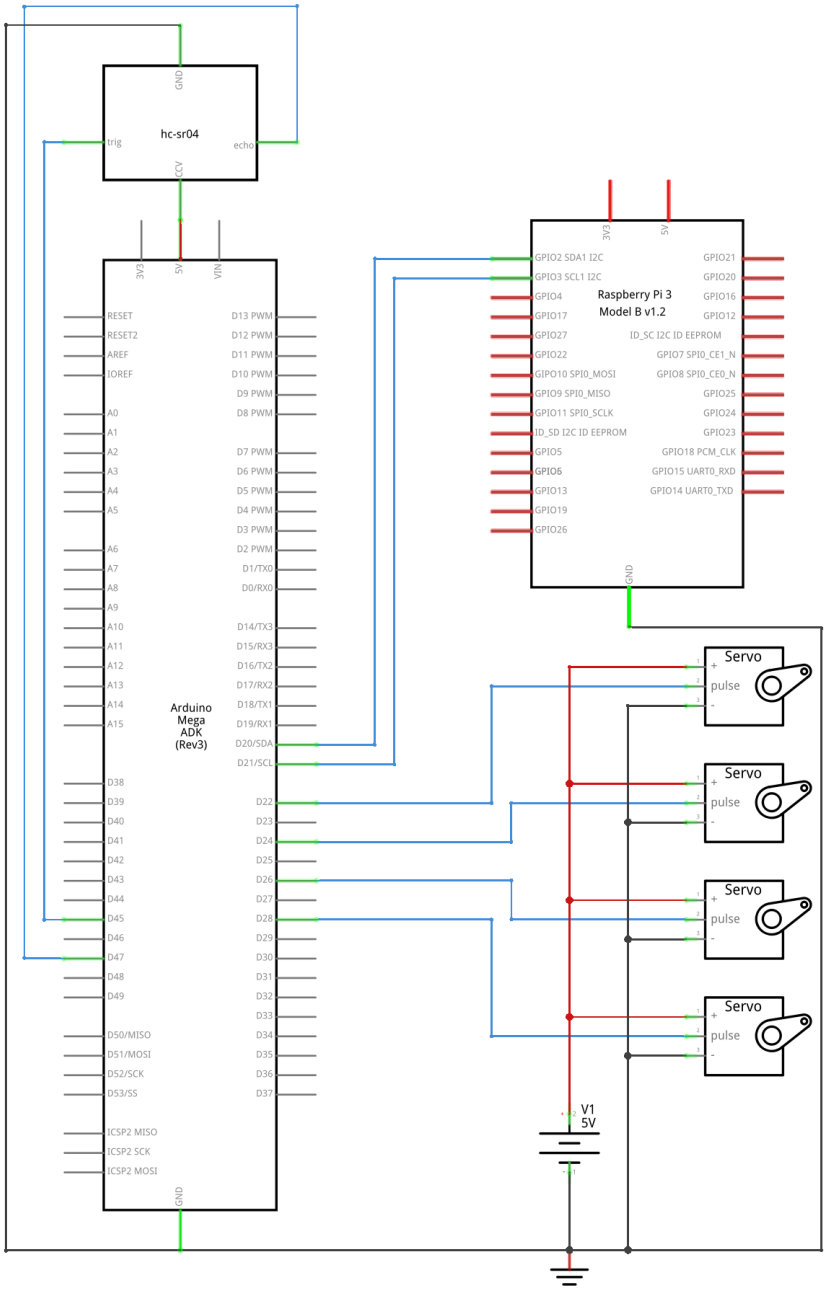
Obecný prehľad návrhu ukážkového riešenia

Vstupom do tejto časti bol robot založený na jednoduchej platforme s dvomi stupňami voľnosti s mikrontrolerom Arduino, ktorý bol výstupom jednej z mojich semestrálnych prác. Celý systém je držaný na tejto platforme pomocou boxu vytlačeného na 3D tlačiarni. Cieľom bolo zoznámiť sa s vývojom viac-vláknových aplikácií pre Linux a riešenie problémov spojených s nimi. Aplikácia by mala umožňovať zadávanie príkazov, zobrazovanie výsledkov meraní senzoru a umožniť pohyb robota.

4.1 Použité komponenty

- Raspberry Pi model 3B+, procesor ARM Cortex-A53
- Arduino MEGA 2560 REV3, procesor ATmega2560
- Ultrazvukový senzor HC-SR04
- 4x servomotor Hitec HS-422

4.2 Schéma



Obr. 4.1: Schéma zapojenia

4.3 Rozdelenie zodpovedností

Pre modelovú aplikáciu sme sa rozhodli pre riešenia s nasledovnými charakteristikami: Raspberry Pi bude fungovať ako komunikačná a koordinačná jednotka spojená s mikrokontrolerom prostredníctvom zbernice I^2C, sprostredkúvajúca vstupný bod interakcie s užívateľom. Táto jednotka bude zodpovedná za:

- Poskytnutie možnosti pripojenia ako prístupový bod Wi-Fi na bezdrátové diaľkové ovládanie.
- Zobrazovanie grafického rozhrania.
- Komunikáciu s mikrokontrolerom.

Druhou polovicou riešenia je mikrokontroler – v našom prípade Arduino, ktorý bude vykonávať nasledovné činnosti:

- Skutočné riadenie v reálnom čase – ovládanie 4 servo motorov.
- Obsluhovať ultrazvukový senzor, ktorý posiela zbernicou naspäť dáta o vzdialenosti od prekážok.
- Spätnú komunikáciu s Raspberry Pi.

4.4 Konfigurácia Raspberry Pi

4.4.1 Operačný systém

Ako operačný systém sme zvolili Raspbian Stretch Lite, okresanú minimalistickú verziu oficiálneho operačného systému pre Raspberry Pi, založeného na kerneli 4.14.y. K realizácii cieľu čo najnižšej latencie sme aplikovali **RT_PREEMPT patch** v konfigurácii **RT_PREEMPT_FULL**, ktorého funkcia je popísaná ďalej v tejto kapitole. Vzhľadom na presunutie zodpovedností, pre ktorých správny chod je nutné operovať v tvrdom reálnom čase na mikrokontroler, bola zvolená práve táto konfigurácia ako kompromis medzi požiadavkou na rýchlosť operácie, prívetivosti, schopnosti použiť štandartné programy pre poskytnunie funkcionalít popísaných v sekcii 5.2 a nabratiu skúseností s funkcionalitou, konfiguráciou a kompiláciou kernelu.

4.4.2 RT_PREEMPT patch

Tento patch poskytuje zníženie latencie prepínania procesov vďaka niekoľkým mechanizmom, ktoré sú zmenené jeho aplikáciou oproti štandartnému kernelu. Nižšie použité slovenské termíny sú vysvetlené v kapitole „Synchronizácia vlákien, možné problémy a ich riešenia“. Mechanizmy implementované v **RT_ PREEMPT patch**, ktoré sú postupne prevádzané aj do štandartného kernelu, umožňujú deterministické plánovanie procesov („scheduling“, modul zodpovedný za plánovanie procesov sa nazýva „scheduler“, teda plánovač) a garanciu termínov prevedenia, záruk („deadlines“), za cenu zníženej priechodnosti systému („throughput“) pomocou nasledovných mechanizmov [15]:

- Plná preempcia systémových procesov až na priame výnimky.
- Konverzia vláknovo-synchronizačných primitívov typu **spin_ lock** na **mutex**.
- Ako ochrana pred inverziou priorít („priority inversion“), ktorá môže spôsobiť uviaznutie („deadlock“) je implementované dedenie priorít systémových procesov („priority inheritance“)
- Konverzia prerušení na vlákna, za ktoré je zodpovedný plánovač, a obsluha prerušení môže vyžiadať jeho zaradenie do fronty.

Dôležitým konceptom pre deterministiku plánovania v Linuxe je pochopenie princípov, na ktorých plánovače operujú. O poradí vykonania vlákien žiadajúcich o zdroje rozhoduje plánovač poďla priority, jednoducho povedané, proces s najvyššou prioritou vždy vyhrá. Akonáhle sa vo fronte nachádza viacero vlákien s rovnakou prioritou, začína sa na ne aplikovať plánovacia stratégia („scheduling policy“). Každé plánovanie má nasledovnú vlastnosť - je preemptívne, t.j. akonáhle je vlákno s vyššou prioritou pripravené na chod, prichádza k preempcii, teda prerušeniu vykonávania sa súčasného vlákna a zdroje sa uvoľnia.

Plánovače pracujú poďľa piatich základných plánovacích stratégií **SCHED_FAIR**, **SCHED_BATCH**, **SCHED_OTHER** a pre nás podstatných **SCHED_FIFO** a **SCHED_RR**, ktoré sú schopné zaručit prácu v reálnom čase [16]. Prvé tri spomenuté stratégie sú pre operáciu v reálnom čase pomerne nezaujímavé, majú totiž prioritu vždy 0 a operujú s iným kritériom, zvaným prívetivosť („nice level“). Posledné dve poskytujú už ozajstnú záruku a ich princíp je nasledovný: **SCHED_FIFO** operuje, ako už názov napovedá, systémom „prvý dnu, prvý von“ („First In-First Out“), vždy vykoná preempciu vlákien s nižšou prioritou a beží až do zavolania funkcie **sched_yield()**, ktorou sa

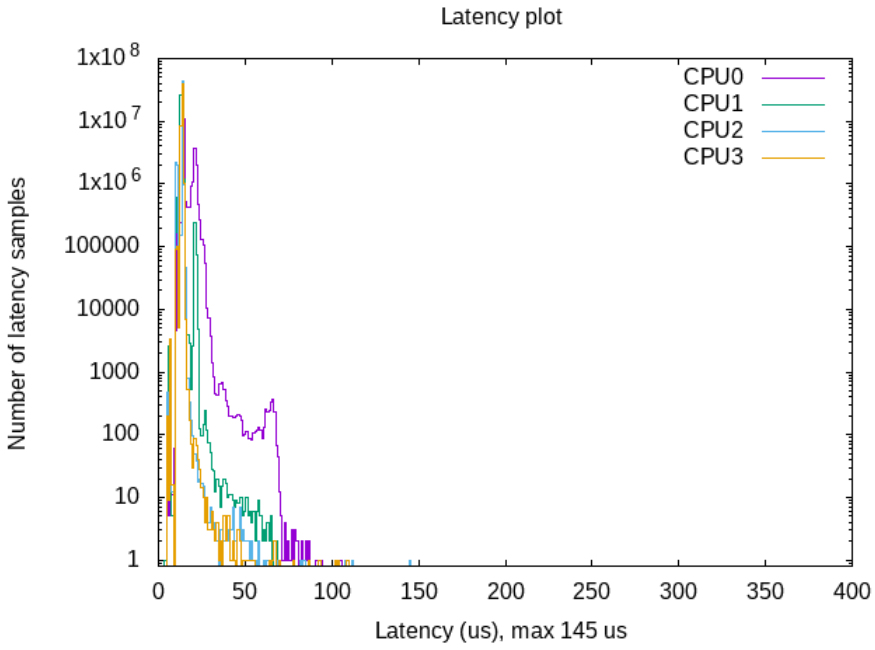
vzdáva zdrojov. **SCHED_RR** (Round-Robin) je rozšírením predošlej stratégie o maximálny čas, koľko môže toto vlákno bežať. Pre garanciu záruk je nutné použiť plánovaciu stratégiu v reálnom čase, ale systém fungujúci pod svojou maximálnou kapacitou zdrojov je schopný dostatočne fungovať aj bez nej - napríklad na našom vzorovom programe nie je citeľný rozdieľ medzi jednotlivými plánovačmi, ba nevhodne zvolená stratégia a priorita môže byť až na škodu. Napríklad ak čakáme na vstup od užívateľa cez periférie, všetky vlákna sprostredkúvajúce pripojenia musia mať vyššiu prioritu, aby bolo možné zaručiť ich vykonanie, keďže ďaľšie výsledky závisia na nich. Preto by malo byť ich využitie na mysli už pri plánovaní výsledného riešenia.

4.4.3 Kompilácia kernelu

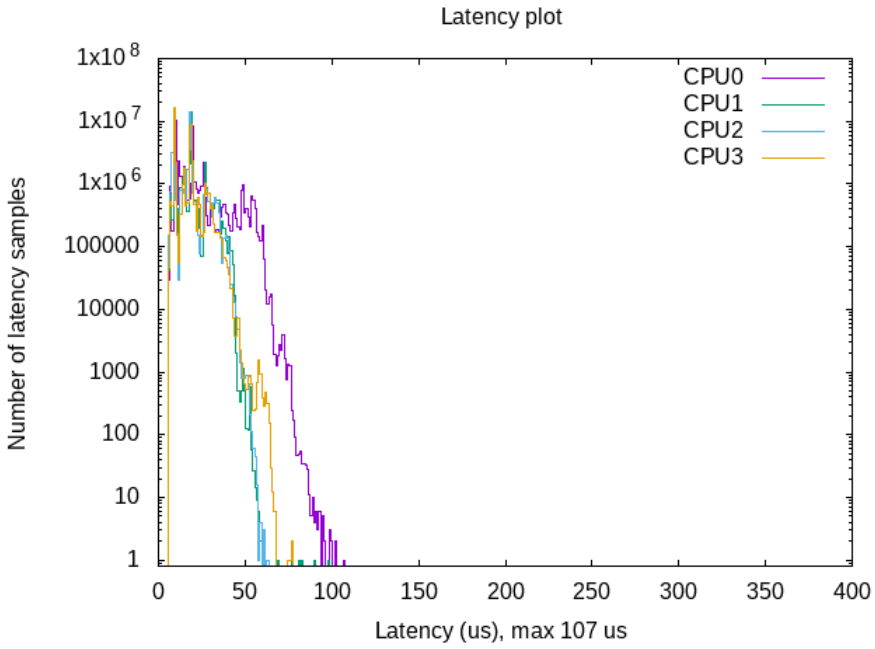
Po inštalácii operačného systému je možné aplikovať vyššie uvedené zmeny na kernel. Pre kompiláciu kernelu bol nasledovaný návod, ktorý Ing. Mauro Riva publikoval na svojej osobnej stránke. [17]. V návode je podrobne popísaný postup, potrebné nástroje a dodatočné konfigurácie pre zlepšenie odozvy.

4.4.4 Test latencie RT_PREEMPT

Pomocou utility Cyclictest sme otestovali dobu, ktorá je rozdielom plánovaného rozbehu vlákna a jeho skutočného štartu vlákna. Test je založený na periodickom zobúdzaní meracích vlákien, ktoré pracujú s plánovacou stratégiou **SCHED_FIFO** a informácie o rozdiele časov vracajú hlavnému vláknu, ktoré zaznamenáva maximálnu, minimálnu a priemernú dobu latencie [18].
Tieto dáta boli následne spracované v GNUPlot pomocou skriptu od Open Source Automation Development Lab [19]. Test bol spúšťaný pre 50 miliónov iteracií s dobou trvania približne 2,5 hodiny. Keďže pri takejto záťaži prichádza k prudkému zahrievaniu, čo je nasledované znížením taktu procesora, platformu sme celý čas chladili okrem pasívneho chladiča aj ventilátorom.
Výsledky testov potvrdzujú teoretické predpoklady - maximálna latencia systému s kernelom 4.14.34-rt27-v7+ je znížená tým, že keď je vlákno zobudené, nie je blokované systémovými procesami, ktoré sa snažia dobehnúť. Ďaľším zaujímavým pozorovaním je vyššia hodnota priemernej latencie, kde sa dá pozorovať fakt, že systém v reálnom čase nezaručuje najrýchlejšiu možnú odozvu, iba znižuje a zaručuje maximálnu možnú dobu, ktorá bude potrebná na vykonanie zmeny kontextu.



Obr. 4.2: Latencia pre štandartný kernel 4.14.34



Obr. 4.3: Latencia pre kernel s patchom 4.14.34-rt27-v7+

Kapitola 5

Softvérová implementácia

5.1 Použité nástroje

5.1.1 Qt Creator

Na vývoj programu sme použili Qt Framework v Qt Creator [20], ktorý bežal na host PC s operačným systémom Kubuntu 18.04.0 LTS. Qt Creator podporuje krížové prekladanie („cross compiling“) a debugovanie programu bežiaceho na Raspberry Pi z inej platformy než na akej beží hostujúci počítač. Pre umožnenie takéhoto fungovania je nutné stiahnuť zdrojové kódy knižníc, adekvátny toolchain pre ARM architektúry a vytvoriť konexie medzi adresármi, ktoré budú systémy zdieľať. Podrobný návod vytvorený komunitou je dostupný v nemeckom jazyku na fórach raspberrypi.org [22].
Poznámky k návodu:
Ako cielové zariadenie pre kompiláciu pre Raspberry Pi 3B+ je treba uviesť parameter " -device linux-rasp-pi3-g++". Taktiež pre správne fungovanie GDB ako vzdialeného debuggeru musí mať hostovací počítač nainštalovaný balík "gdb-multiarch-a na synchronizáciu adresárov na koreňovej úrovni musí mať adresár usr nastavené práva na čítanie-/zapisovanie pomocou shell príkazu "chmod -777 usr".

5.1.2 Arduino IDE

Na kompiláciu a nahrávanie programu do mikrokontroleru Arduino sme použili oficiálne vývojové prostredie Arduino IDE [21].

5.1.3 Qt Framework

Qt Framework je multiplatformový aplikačný framework primárne používaný pre grafické aplikácie. Je nadstavbou jazyka C++ a zjednodušuje vývoj grafických aplikácií. Práve Qt Framework je často využívaný pre vytvorenie desktopového prostredia, používa ho napríklad KDE Plasma v Linuxovej distribúcii Kubuntu, v ktorom bola táto práca napísaná. Náš program bol napísaný pre verziu Qt 5.9.5.
Jeho prednosťou je multiplatformovosť pri zachovaní natívnosti - kód je možno použiť či už na Windows platforme alebo POSIX systémoch, pričom každá platforma má vlastnú implementáciu obsiahnutú v Qt knižniciach a je ju potrebné špecifikovať až pri kompilácii. Objekty popísané týmto frameworkom teda možno pri písaní programu používať bez špecifikácie platformy, a teda ich možno ľahko preniesť medzi systémami za pomoci malých alebo žiadnych úprav. Qt Framework implementuje wrappery k natívnym rozhraniam systémov, značne uľahčujúcu napríklad prácu s vláknami alebo synchronizačnými primitívami. Tento framework implementuje pre každé vlákno svoj vlastný systém spracovania udalostí – Event Loop, čo je tradičný návrhový vzor pre aplikácie, ktorých funkcionalita závisí na vstupe od užívateľa. Ako dodatok je na komunikáciu medzi vláknami sprostredkovaná cez frontu udalostí („Event Queue“), ktorá zabraňuje súbehu („race condition“) medzi dvomi vláknami. Tento systém je abstrahovaný od typického spôsobu volania funkcií „callback“, pomocou mechanizmu „Slot“ a „Signal“. Signál je pri jeho vyvolaní kľúčovým slovom **emit** zaradený do Event Queue vlákna, v ktorom sa nachádza slot, ktorý obslúži príslušnú udalosť.
Hoci Qt neobsahuje priamy prístup k nastaveniam vlastností vlákien, ako napríklad plánovača alebo plného rozsahu priorít[1], vďaka open-source licencii je možné modifikovať zdrojové kódy. Trieda **QThread**, handler v ktorej vlákno beží, začína volaním metódy **start()**, ktorej argumentom je priorita vlákna. Ak by sme sa rozhodli, že naša aplikácia potrebuje možnosť meniť vlastnosti vlákien, je to možné dosiahnuť pozmenením práve tejto metódy. Ako ideálny prístup, ktorý zachováva princíp enkapsulácie je pridanie ďalšieho výčtového typu argumentu metódy. V ňom môžeme následne pozmeniť stratégiu plánovača.

[1]Stratégia plánovača je dedená pre všetky vlákna v aplikácii podľa toho, aký sa zvolí pri spustení aplikácie podľa parametra.

5.2 Naplnenie rozdelenia úloh

Naplnenie úloh, ktoré má vykonávať Raspberry Pi je nasledovné:

- Poskytovanie pripojenia je vyriešené pomocou programu dnsmasq, ktorý sprostredkuje DHCP pre Wi-Fi sieť ktorá je vytvorená cez hostapd, zodpovednosťou ktorého je vytvorenie samotného prístupového bodu a autentifikačného serveru.
- Vytvorenie grafického užívateľského rozhrania je zodpovednosťou trojice programov – Robot, LightDM ako Display Manager a Openbox ako Window Manager. V spolupráci tieto tri programy následne umožňujú vykreslenie užívateľského rozhrania a pomocou x11vnc serveru je možné toto užívateľské rozhranie zobraziť na akomkoľvek počítači pripojenom na Wi-Fi, ktoré je poskytované podľa prvého bodu.
- Komunikácia s mikrokontrolerom je sprostredkovaná nezávislým vláknom v programe Robot, ktoré pomocou knižnice Pi2c komunikuje cez zbernicu I2C s Arduinom. Toto vlákno je počas chodu zviazané pomocou zdieľaných zdrojov so synchronizačnými mechanizmami s hlavným vláknom, na ktorom beží grafické rozhranie.

5.3 Komunikácia

Ako už bolo niekoľko krát spomenuté, pre zaistenie komunikácie medzi časťami systému je použitá zbernica I^2C. Táto zbernica sa vyznačuje veľmi jednoduchým pripojením pomocou 2 vodičov a typickou frekvenciou približne 100-400kHz.
Ďaľšou výhodou je samotné prevedenie elektrického rozhrania, ktoré je zapojené ako otvorený kolektor. Dôsledkom toho je možnosť priameho prepojenia Raspberry Pi a Arduino, ak Raspberry Pi vystupuje ako Master, hoci tieto dve zariadenia primárne pracujú na odlišných napäťových hladinách (3.3V pre RaspberryPi a 5V pre Arduino). Toto je možné vďaka tomu, že Arduino nemá žiadne pull up odpory na výstupoch spojených s I^2C rozhraním, a Raspberry Pi má integrované 1.8kΩ odpory na príslušných výstupoch. Špecifikácia rozhrania hovorí, že logická jednotka je pre túto zbernicu napätie "pulled-down-k nule. Pretože 3.3V je v rozsahu logickej nuly pre Arduino, toto zapojenie je funkčné. [23]

5.3.1 Riešenie problému súbehu

Situácia, kedy sa dve a viac súbežne bežiace vlákna snažia manipulovať so zdielanými zdrojami, je bez zabezpečenia mimoriadne nebezpečná. Môže sa totiž stať, že jedno

vlákno zmení tento zdroj, zatiaľ čo druhé sa s ním snaží pracovať. Výsledkom je nepredvídateľné správanie systému, čo je v priamom v konflikte s požadovaným determinizmom operácií[2]. Na zamedzenie výskytu takéhoto stavu vo viac-vláknových systémoch existuje niekoľko dizajnových návrhov, ktoré sa delia na dva prístupy - zamedzenie zdieľaných stavov a synchronizácia. Keďže nami implementovaný návrh priamo vyžaduje zdieľanie zdrojov, možné riešenia pre zamedzenie súbehu sú ***vzájomné vylúčenie („mutual exclusion“)*** a ***atomické operácie***.

Atomické operácie sú operácie, ktoré sa z pohľadu zvyšku systému stanú okamžite. Pri redukcii na úplný základ je atomická operácia jedna strojová inštrukcia ktorá je vykonaná za jeden cyklus procesoru. Tento koncept možno rozšíriť na blok kódu, ak vieme zaručiť, že jeho chod nemôže byť prerušený. Táto funkcia je sprostredkovaná v jazyku C++ napríklad knižnicou std::atomic, ktorá tento koncept aplikuje na prácu s premennými.

Vzájomné vylúčenie sprostredkúva sériový prístup k dátam. Základný objekt, ktorý implementuje tento prístup sa nazýva ***lock(mutex)***. Takýto objekt implementuje funkcie lock() a unlock(), ktoré menia stav tohto objektu na „zamknutý“, resp. „odomknutý“, a premennú, ktorá v sebe drží aktuálny stav. Obsluha kritickej sekcie začína pokusom o získanie zámku nad takýmto objektom. V prípade, že je mutex už zamknutý a zlyhá pokus o zamknutie, vlákno, v ktorom sa vykonáva prístup, sa vzdáva procesorových zdrojov a čaká na povolenie na prístup. Nutnosťou je odomknutie po vykonaní obsluhy, aby iné vlákna mohli nasledovne získať chránené zdroje. Existuje niekoľko nadstávb nad týmto konceptom, z ktorých najpodstatnejšie sú ***spinlock*** a ***semafór***.

Spinlock (spomínaný v časti 4.4.2) je implementácia mutex, ktorá sa nevzdá zdrojov procesoru pri neúspešnom pokuse o získanie zámku, naopak, testuje či je možné zámok získať, až pokiaľ sa mu to nepodarí. Presne z tohto dôvodu má konverzia typov spinlock na mutex v prípade konfigurácie RT_ PREEMPT_ FULL za výsledok zníženú latenciu zmeny kontextu.

Semafór je nadstavbou typu ***mutex***. Semafór chráni určitý počet zdrojov tým, že okrem funkcií mutexu drží aj počet dostupných zdrojov. Každé zamknutie znižuje počet počet dostupných zdrojov až do nuly a každé odomknutie inkrementuje tento počet o jedna. Mutex je v princípe binárny semafór.

[2]Tento problém nie je exkluzívny iba pre viac vláknové systémy, rovnaký problém môže nastať v jednovláknovom systéme, ktorý umožňuje prerušenia. V takom prípade existuje jednoduché riešenie - zakázanie prerušení pri vykonávaní obsluhy kritickej sekcie kódu, čím sa zabráni preempcii procesu.

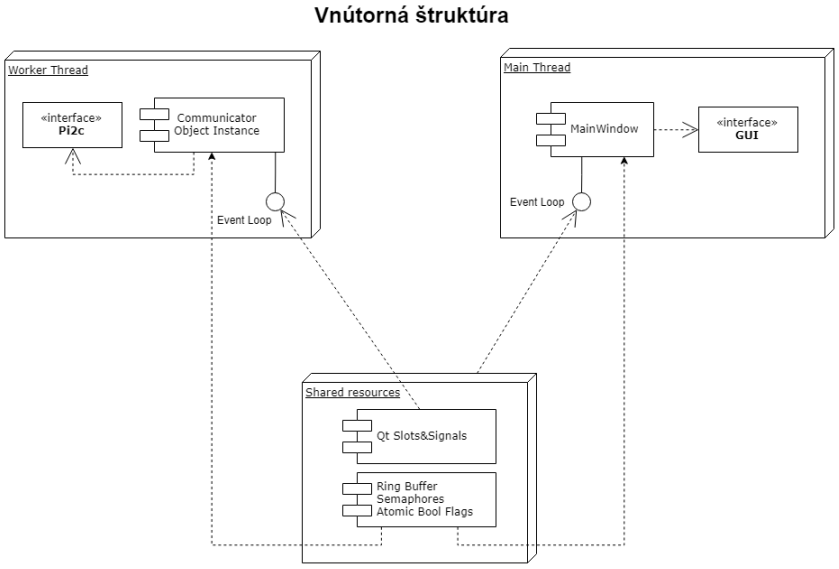
5.4 Program Robot pre Raspberry Pi

5.4.1 Dependencie pre správne fungovanie

Nasledovné balíky boli využité pre sprostredkovanie požadovaných funkcií:

1. Xorg, pre sprostredkovanie grafického výstupu [24].
2. Openbox, ako WindowManager [25].
3. x11vnc, ako server poskytujúci vzdialené pripojenie priamo ku grafickému výstupu [26].
4. i2c-tools, ako štandardná Linux knižnica zodpovedná za I^2C komunikáciu.
5. Pi2c, open-source knižnica implementujúca rozhranie sprostredkované i2c-tools pre komunikáciu s Arduinom [27].

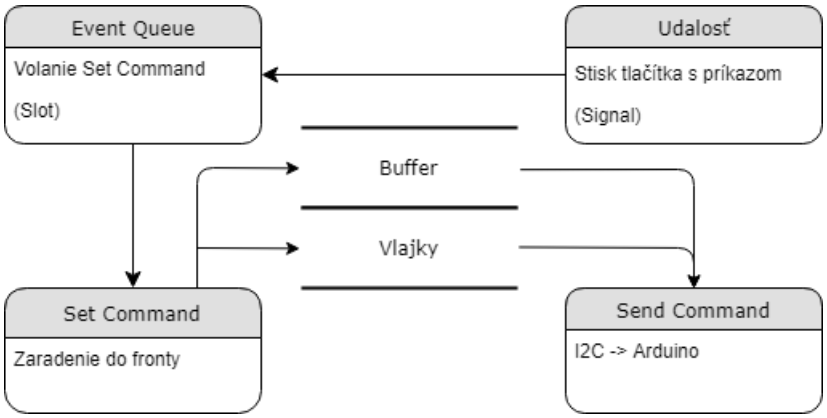
5.4.2 Popis programu



Obr. 5.1: Schéma vnútornej štruktúry programu.

Ako podklad pre tento program sme sa rozhodli použiť C++ framework Qt5, kvôli jeho schopnosti jednoduchého poskytnutia a obsluhy užívateľského rozhrania a dostupnosti zdrojových kódov v rámi open-source licencie. Vďaka tomu si bolo možno overiť

presnú implementáciu POSIX vlákien triedou *QThread* a taktiež nadstavby nad synchronizačnými primitívami.
Koncept návrhu je pomerne jednoduchý – program obsahuje dve vlákna, jedno obsluhujúce užívateľské rozhranie a druhé zodpovedné za komunikačnú logiku implementovanú triedou Communicator. Dôvod návrhu aplikácie ako viac-vláknovej vychádza zo základnej premisy návrhu aplikácií, ktorých výstup závisí na udalosti vyvolanej užívateľom. Rozdelenie na dve vlákna v našom prípade zaručuje, že odozva užívateľského rozhrania bude čo najrýchlejšia - čas medzi udalosťou, vyvolaním funkcie obsluhujúcej túto udalosť a návrat z nej nie je predlžovaný logikou, spracovaním a odoslaním príkazu do mikrokontroleru. Tieto výpočty sú odbavené v druhom vlákne, v nezávisle bežiacej nekonečnej slučke, ktorá pravidelne sleduje zdielané zdroje a v prípade, že nastala v nich zmena, ich okamžite obslúži.



Obr. 5.2: Diagram znázorňujúci spracovanie vstupu užívateľa

Previazanie medzi vláknami je sprostredkované pomocou viacerých mechanizmov - kruhového bufferu chráneného dvomi inštanciami triedy QSemaphore, vďaka ktorej je možný asynchrónny prenos príkazov k pohybu medzi oboma vláknami bez súbehu nad zdrojmi, a vlajkami primitívov typu atomic pre prioritné posielanie príkazov „Stop“ a „Kill“.

Využitie kruhového bufferu je implementované nasledovne:

Listing 5.1: Kruhový buffer v hlavnom vlákne

```
1  void MainWindow::setCommand(int command){
2      freeBytes.acquire();
3      buffer[bufferCounter%BufferSize]=command;
4      if (bufferCounter == BufferSize)
5      { bufferCounter = 0;}
6      else {bufferCounter++;}
7      usedBytes.release();
8      QString message = ...//správa pre UI
9      ui->label_cmd_sent->setText(message);
10 }
```

Vstupom je príkaz daný výčtovým typom. Nasleduje pokus o získanie zdroja, v tomto prípade semafóru **freeBytes**. Počet zamknutých jednotiek v tomto semafóri udáva počet príkazov v bufferi. Funkcie v jednotlivých vláknach pristupujúce k bufferu majú svoj vlastný čítač zvaný **bufferCounter**, ktorý sleduje na ktorom indexe sa nachádza ukazovateľ. Hlavné vlákno po zápise príkazu do bufferu uvoľní zdroje, ktoré komunikačné vlákno označilo za použité v semafóre **usedBytes**.

Listing 5.2: Kruhový buffer v komunikačnom vlákne

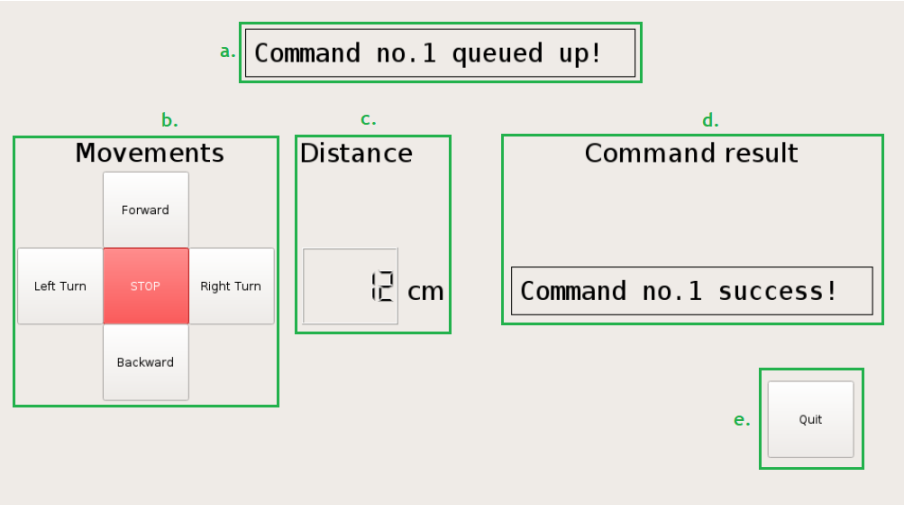
```
1  void Communicator::retrieveCommand(){
2      if (freeBytes.available() < BufferSize){
3          usedBytes.acquire();
4          retrievedCommand = buffer[this->bufferCounter_%BufferSize];
5          if (bufferCounter_ == BufferSize)
6          {bufferCounter_ = 0;}
7          else {bufferCounter_++;}
8          this->sendCommand(retrievedCommand);
9          emit signal_CommandResult(this->commandResultSuccess());
10         freeBytes.release();
11     }
12 }
```

Spracovanie príkazu na strane komunikačného vlákna funkciou **retrieveCommand** je periodicky volané v hlavnej slučke. Ak sa v bufferi nachádza príkaz, príde k jeho vyzdvihnutiu a odoslaniu do mikrokontroleru. Taktiež je oznámený výsledok prijatia príkazu

Arduinom. **commandResultSuccess** je jednoduchý wrapper nad chybovými kódami knižnice **Pi2C**, vracia boolean hodnotu. Následne je uvoľnený semafór **freeBytes**.
Takýto dizajnový návrh je tradičným spôsobom riešenia problému producent-konzument [28], aby sa predišlo situáciam, kedy sa snaží viacero vlákien konzumovať z (a zapisovať do) rovnakého bufferu.
V limitovanom rozsahu je pre tieto účely využitý aj Qt Event systém. Keďže ale réžia riešenia pripojení mechanizmov „Slot“ a „Signal“ nie je zanedbateľná a nedá sa zaručiť ich okamžité nezávislé obslúženie, tento systém vo vlákne zodopovednom za komunikáciu používame len na vysielanie signálov pre aktualizovanie užívateľského rozhrania a upratanie objektov pri konci aplikácie.

5.4.3 Užívateľské rozhranie

Užívateľské rozhranie sa skladá z piatich sekcií. V sekcii **a.** je zobrazený výsledok odoslania príkazu druhému vláknu cez príkaz vyvolaný prostredníctvom tlačidiel v skupine **b.** Výsledok odoslania príkazu do mikrokontroleru je užívateľovi oznámený v políčku **d.** Sekcia **c.** zobrazuje vzdialenosť poskytovanú ultrazvukovým senzorom. Tlačidlo **e.** zavrie komunikačný kanál, ukončí vykonávanie vlákien, zastaví robota a zavrie program.



Obr. 5.3: Užívateľské rozhranie.

5.5 Program Robot pre Arduino

5.5.1 Dependencie pre správne fungovanie

Pri implementácii na platforme Arduino sme využili nasledovné knižnice:

1. NewPing, ktorý poskytuje meranie vzdialenosti pomocou senzoru HC-SR04 [29].
2. Wire, umožňujúci komunikáciu cez rozhranie I^2C [30].

5.5.2 Popis

Program bežiaci na Arduino je napísaný v jazyku C++ a skompilovaný pomocou toolchainu v Arduino IDE. Program je logicky členený na tri triedy:

- Srv_mv, ktorá poskytuje funkcie zodpovedné za pohyby.
- Logic, ktorá spracuje vstupné príkazy a premieňa ich na príkazy k pohybu a obsluhuje cez knižnicu NewPing ultrazvukový senzor.
- Simple_Queue, ako fronta v ktorej čakajú príkazy z Raspberry Pi pred ich vykonaním.

Na vytvorenie pulzne-šírkovej modulácie na riadenie servo motorov je použitá obsluha prerušení časovača TIMER3, ktorá generuje štvorcový signál o štandardnej frekvencii 50 Hz. Na nastavenie činiteľa plnenia, ktorý určuje uhol natočenia, sú využité funkcie v triede Srv_mv. Jednotlivé funkcie v nej sú implementované tak, aby bol poskytnutý krútiaci moment na viacerých servách súčasne, podľa typu pohybu.
Pre zaistenie okamžitého zastavenia je použitá globálna vlajka Stop, ktorej stav sa sleduje aj v jednotlivých častiach vykonávania pohybu. Na stave tejto premennej zároveň závisí maska, ktorou sa vypína generovanie pulzov pre servo motory pri úplnom zastavení.
Príjmanie a čítanie zo zbernice medzi Raspberry Pi a Arduinom je vďaka knižnici **Wire** značne zjednodušené. Knižnica za nás rieši špecifikáciu adresy, začiatok a koniec komunikácie, jedinou našou úlohou je spracovať dáta prichádzajúce po zbernici. Spracovanie prebieha vyvolaním prerušenia či už pri čítaní alebo zápisu do slave jednotky. Tomuto prislúchajú dve handler funkcie knižnice Wire - **onRequest** a **onReceive**.

Ich argumentom sú nasledovné nami vytvorené funkcie:

Listing 5.3: Funkcie pre komunikáciu

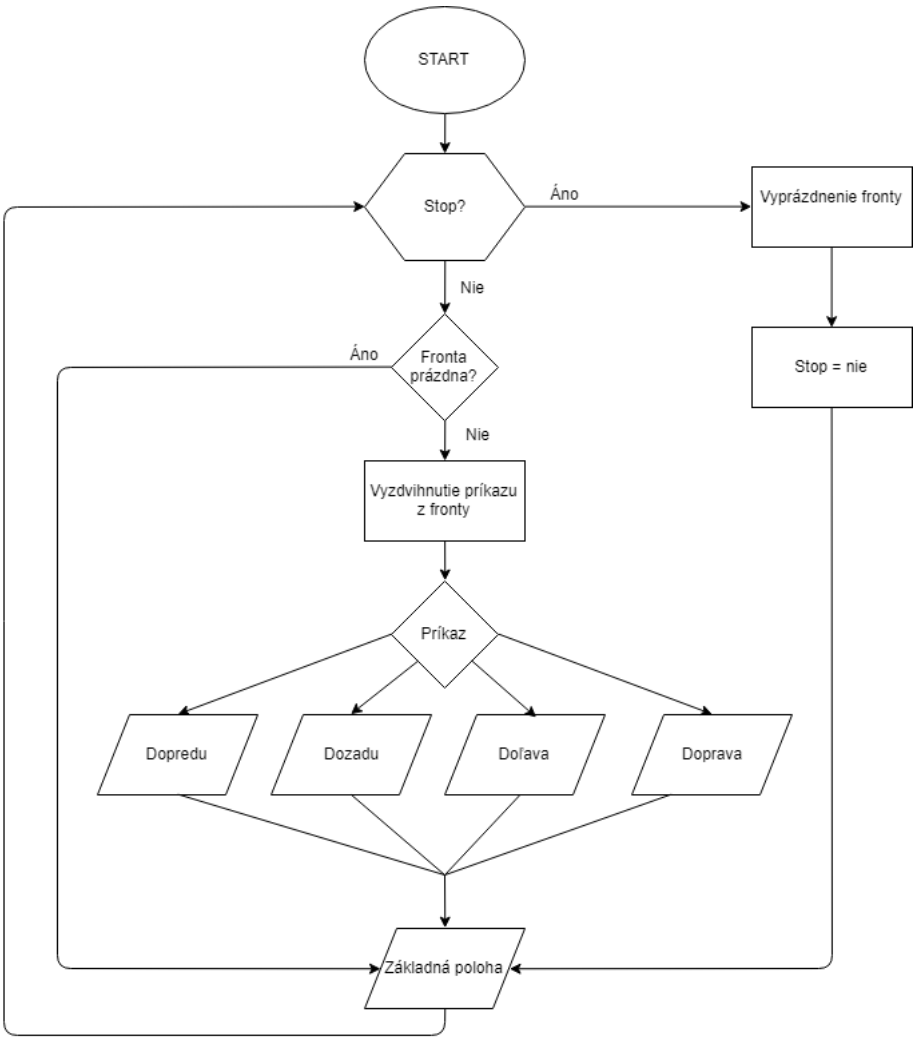
```
1  static void send_distance_onRequest(){
2    byte sender_buffer[2];
3    sender_buffer[0] = (distance_output_i2c >> 8) & 0xFF;
4    sender_buffer[1] = distance_output_i2c & 0xFF;
5    Wire.write(sender_buffer,2);
6  }
7
8  static void receive_command(int bytes_read){
9    int receive_int = 0;
10   int byte_count = 0;
11   char c;
12   while (Wire.available()){
13     c = Wire.read();
14     receive_int = c << (8 * byte_count) | receive_int;
15     byte_count++;
16   }
17   if (receive_int > 0)
18   {Simple_Queue::push_to_queue(receive_int);}
19   else
20   {stop_command_i2c = STOP;}
21 }
```

Keďže typ integer je v Arduino dvojbytový a komunikačný blok zbernice je jeden byte, posielame každé číslo rozkúskovane pomocou prekrývania maskou. Podobne funguje príjem príkazu - objekt knižnice Wire pomocou funkcie **available()** dáva prístup k počtu bytov v komunikačnom kánali a následne ich za pomoci masky transformujeme na typ integer. Podmienka v poslednej časti roztriedi príkazy - ak prišiel pohybový príkaz, zaradí sa do fronty. Ak tento príkaz nie je väčší od nuly[3], môže to byť spôsobené dvomi príčinami - buď sme dostali príkaz STOP, alebo prišlo pri komunikácii k chybe. V oboch prípadoch je žiadané zastaviť chod programu a pohyb robota.

[3]Nula vo výčtovom type pre príkazy odpovedá príkazu STOP.

5.5.3 Hlavný algoritmus

Hlavná slučka logiky pohybu funguje nasledovne:



Obr. 5.4: Algoritmus hlavnej slučky.

Okrem príkazov k pohybu sa v nej taktiež aktualizuje premenná držiaca vzdialenosť z čítača. Pri každom opakovaní sa tiež aktualizuje stav masky, ktorý v prípade príkazu STOP vypne generáciu signálov PWM.

5.6 Diskusia k implementácii

Nami implementované riešenie naplnilo požiadavky a úlohy – umožnuje ovládanie robota prostredníctvom grafického rozhrania a jeho rýchlu reakciu na príkazy.
Pri overovaní funkčnosti sme prišli k viacerým záverom. Jedným z nich je zložitosť, ak nie nemožnosť, zaručenia rýchlej odozvy u programu, ktorej výstup závisí na vstupe od užívateľa. Predpokladáme, že rýchlosť odozvy je v našom prípade spôsobená skôr vysokou priepustnosťou systému ako akýmkoľvek iným faktorom – keďže čas spotrebovaný na vykonanie réžie odoslania príkazu je vďaka výkonu platformy omnoho nižší ako rýchlosť komunikačnej zbernice a perióda signálu, ktorý očakáva servo motor. Takisto nie je možné v takomto prípade skúmať vplyv plánovača – vlákna vytvorené naším programom totiž kvôli tomu, že vstupom je udalosť vyvolaná užívateľom, musia mať nutne prioritu nižšiu ako všetky systémové procesy, ktoré sprostredkúvajú jej vyvolanie a obsluhu.
Pri spätnej analýze sme dospeli k záveru, že návrh by pravdepodobne bol uskutočniteľný bez využitia mikrokontroleru – pri sledovaní záťaže mikroprocesoru počas chodu programu sme nedosiahli špičkové vyťaženie väčšie ako 20% naprieč jadrami. Možným prístupom by bolo vytvorenie ešte jedného vlákna, ktoré by bolo zodpovedné čisto za generovanie signálu, a nechať ho pod plánovacou stratégiou FIFO bežať na jednom jadre bez vzdávania sa zdrojov.
Výhodou nášho riešenia je jeho modularita a rozšíriteľnosť – podobný program je bez nutnosti prepisovania vrstvy v Raspberry Pi možno použiť pre ovládanie inej robotickej platformy. Použitá verzia mikrokontroleru by bola taktiež schopná ovládať oveľa komplikovanejšiu sústavu – napríklad rozšírenú o viac stupňov voľnosti pohybu, prípadne vybavenú viacerými senzormi pre monitorovanie okolia.

Kapitola 6

Záver

Pokroky v elektronike menia spôsob, akým interagujeme so svetom, rozširujú možnosti tejto interakcie a prepojenosti s ostatnými platformami. Tento trend je nutné nasledovať aj pri riadení pohonov.
Raspberry Pi je výsledkom kulminácie týchto pokrokov – a jeho aplikácie zapadajú práve do smeru, ktorým sa požiadavky uberajú. Hoci nie je vybavené pre priame industriálne využitie ako mikrokontroler, vieme si predstaviť, že môže nájsť svoje miesto aj v takomto prostredí – ako napríklad na sprostredkovanie prístupovej konzoly, na monitorovanie a odosielanie nameraných dát prostredníctvom internetu alebo ako koordinačná jednotka pre väčšie ekosystémy.
Pri využití Raspberry Pi by prvým krokom malo byť určenie jeho pozície v riešení a podľa neho zvoliť prístup k využitiu z hľadiska operačného systému. Od tejto voľby sa ďalej odvíja rozsah zodpovedností, ktoré môžeme naň presunúť.
V našej modelovej implementácii sme sa zoznámili s vývojom aplikácii s grafickým užívateľským rozhraním, ktorej priama funkcionalita na strane pohonu závisela na vstupnej udalosti vyvolanej používateľom. Narazili sme pri nej na problém spojený s týmto dizajnovým návrhom, konkrétne zložitosť zaručenia tvrdej odozvy v reálnom čase. Dosiahnutie tejto záruky je dokonca priamo nemožné ak ide o vzdialený prístup, ako je v našom prípade zdieľanie užívateľského rozhrania cez Wi-Fi na počítač, kde beží obyčajný operačný systém. Tvrdú odozvu vieme garantovať až po vstupe informácie do nášho programu a ak nevieme kontrolovať komunikačný kanál a jeho odozvu, potom nemôžeme o aplikácii prehlásiť, že operuje v reálnom čase.
Konkrétny príklad aplikácie, ktorá by mohla využívať Raspberry Pi a chceli by sme ju v budúcnosti preskúmať, je 3D tlačiareň s bezdrôtovým prístupom. Vstup dát by bol vyriešený cez poslanie súboru s modelom pre vytlačenie, čím by bolo zaručené, že náš

program má prístup k operáciám, ktoré musí vykonať v ich celkovej forme. Potom môžeme konštatovať, že ich vstup do aplikácie nie je zaťažený inými faktormi ako tými, ktoré vieme priamo sledovať a kontrolovať v rámci nášho systému.
Ďalším príkladom odvetvia, kde by Raspberry Pi mohlo nájsť svoje miesto, je v rámci fotovoltaiky. Okrem monitorovania by sa mohlo podielať v prípade použitia panelov na otočných pivotoch v kombinácii so senzormi osvetlenia na zaistenie maximálnej výťažnosti. Možnosťou je aj využitie API zberajúceho informácie o predpovedi počasia na získanie prognóz ohľadom výkonu, ktorý bude možné poskytnúť za časový úsek, a pomocou neho napríklad optimalizovať cykly batérie v nezávislých systémoch.
Fenoménom poslednej doby je Internet vecí (Internet of Things), ktorý prenikol do všetkých sfér - od každodenných predmetov ako hodinky, cez inteligentné budovy až po priemysel. Priemysel 4.0 je za dverami - a kľúčom k nemu sú práve kompaktné, univerzálne, výkonné počítače s nízkou spotrebou, medzi ktoré sa dá zaradiť aj Raspberry Pi.

Kapitola 7

Použité zdroje

[1] Eben Upton *Raspberry Pi 3 Model 3B+ Introduction* [Online]. [cit 20. apríla 2018] https://www.raspberrypi.org/blog/raspberry-pi-3-model-bplus-sale-now-35/

[2] eLinux.org *Aplikačné listy Raspberry Pi 3B+* [Online]. [cit 20. apríla 2018] https://elinux.org/RPi_Hardware

[3] ARM *Aplikačné listy ARM Cortex-A53 MPCore Processor* [Online]. [cit 21. apríla 2018] https://developer.arm.com/docs/ddi0500/latest/preface

[4] Rasperry Pi Foundation *FAQS* [Online]. [cit 20. apríla 2018] https://www.raspberrypi.org/help/faqs/#powerReqs

[5] ARM *ARM Architecture Profiles* [Online]. [cit 22. apríla 2018] http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0471i/BCFDFFGA.html

[6] ARM *The ARMv8-A instruction sets* [Online]. [cit 22. mája 2018] https://developer.arm.com/products/architecture/a-profile/docs/100878/latest/the-armv8-a-instruction-sets

[7] Alza.cz *Raspberry Pi 3 Model B+* [Online]. [cit 17. mája 2018] https://www.alza.cz/raspberry-pi-3-model-b-d5284636.htm

[8] David Welch *Raspberry Pi ARM based bare metal examples* [Online]. [cit 17. mája 2018] https://github.com/dwelch67/raspberrypi

[9] Microsoft *Windows IoT Core Documentation - SoCs and Custom Boards* [Online]. [15. mája 2018]
https://docs.microsoft.com/en-us/windows/iot-core/learn-about-hardware/suggestedboards

[10] Raspberry Pi Foundation *Raspbian* [Online]. [cit 15. mája 2018]
https://www.raspberrypi.org/downloads/raspbian/

[11] Prabhaker Mateti *seL4 Micro Kernel* [Online]. [cit 17. mája 2018]
http://cecs.wright.edu/~pmateti/Courses/FormalMethods/Lectures/SEL4/sel4-whatis.html

[12] ChibiOS *ChibiOS/RT free embedded RTOS* [Online]. [cit 10. mája 2018]
https://sourceforge.net/projects/chibios/

[13] James Walmsley *A port of FreeRTOS to the raspberry pi.* [Online]. [cit 10. mája 2018]
https://github.com/jameswalmsley/RaspberryPi-FreeRTOS

[14] Xenomai *Xenomai* [Online]. [cit 15. mája 2018]
http://xenomai.org/

[15] Steven Rostedt *Prednáška, Embedded Linux Conference 2013* [Online]. [cit 15. mája 2018]
https://events.static.linuxfound.org/images/stories/slides/elc2013_rostedt.pdf

[16] Michael Kerrisk *Linux Programmer's Manual* [Online]. [cit 15. mája 2018]
http://man7.org/linux/man-pages/man7/sched.7.html

[17] Mauro Riva *# Raspberry Pi: Preempt-RT Patching Tutorial for Kernel 4.14.y* [Online]. [cit 29. apríla 2018]
https://lemariva.com/blog/2018/02/raspberry-pi-rt-preempt-tutorial-for-kernel-4-14-

[18] The Linux Foundation *Cyclictest* [Online]. [cit 2. mája 2018]
https://wiki.linuxfoundation.org/realtime/documentation/howto/tools/cyclictest

[19] Open Source Automation Development Lab (OSADL) eG *Create a latency plot from cyclictest histogram data* [Online]. [cit 2. mája 2018]
http://www.osadl.org/Create-a-latency-plot-from-cyclictest-hi.bash-script-for-latency-plot.0.html

[20] Qt Company *Get Qt* [Online]. [cit 15. mája 2018]
`https://www.qt.io/download`

[21] Arduino Foundation *Download the Arduino IDE* [Online]. [cit 15. mája 2018]
`https://www.arduino.cc/en/Main/Software`

[22] Komunita RaspberryPi *Qt Crosscompile (Qt5.10.1)* [Online]. [cit 15. mája 2018]
`https://www.raspberrypi.org/forums/viewtopic.php?f=75&t=204778`

[23] Oscar Liang *Raspberry Pi and Arduino connected using I^2C* [Online]. [cit 10. mája 2018]
`https://oscarliang.com/raspberry-pi-arduino-connected-i2c/`

[24] X.Org Foundation *The X.Org Project* [Online]. [cit 16. mája 2018]
`https://oscarliang.com/raspberry-pi-arduino-connected-i2c/`

[25] Openbox.org [Online]. [cit 16. mája 2018]
`http://openbox.org/wiki/Openbox:Download`

[26] Karl Runge *x11vnc: a VNC server for real X displays* [Online]. [cit 16. mája 2018]
`http://www.karlrunge.com/x11vnc/`

[27] JohnnySheppard *Raspberry Pi C++ library for easy I2C communication to and from an Arduino* [Online]. [cit 16. mája 2018]
`https://github.com/JohnnySheppard/Pi2c`

[28] Arpaci-Dusseau, Remzi H.; Arpaci-Dusseau, Andrea C. (2014), *Operating Systems: Three Easy Pieces [Kapitola: Condition Variables]*

[29] Tim Eckel *NewPing Library for Arduino* [Online]. [cit 10. mája 2018]
`https://playground.arduino.cc/Code/NewPing`

[30] Arduino Foundation *Wire Library* [Online]. [cit 10. mája 2018]
`https://www.arduino.cc/en/Reference/Wire`

Dodatok A

Zdrojové kódy

Zdrojové kódy pre program Robot pre Raspberry Pi a Arduino, vyprodukované ako súčasť tejto práce, sú nahrané na CD a priložené k práci.