Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

Inžinierske dielo

Automatické testovanie v prostredí Internetu vecí

Vedúci projektu: doc. Ing. Tibor Krajčovič, PhD.
Product owner: Ing. Lukáš Ondriga
Členovia tímu: Bc. Tomáš Bujna
Bc. Marián Ján Franko
Bc. Rastislav Kováč
Bc. Igor Labát
Bc. Miroslav Sabo
Bc. Filip Starý
Bc. Stanislav Širka
Akademický rok: 2018/2019

Obsah

1. Úvod

Nasledovný dokument predstavuje sprievodný dokument k výslednému systému, ktorý je produktom semestrálnej práce nášho tímu, pričom podrobné informácie o tíme ako aj o členoch tímu sú uvedené v paralelnom sprievodnom dokumente s názvom *Riadenie*. Vo všeobecnosti môžeme dokument definovať ako technickú dokumentáciu výsledného produktu.

Obsahom dokumentu, sú hlavné časti produktu spolu s návodmi, ktoré vznikli pri práci na projekte a sú nevyhnutné pre ďalšie pokračovanie na tomto projekte. V dokumente sú definované globálne ciele, ktoré vyplývali prevažne z požiadaviek Product Ownera. Ďalej je v dokumente uvedený celkový pohľad na systém, v ktorom je uvedená architektúra systému v spojení s využívanými modulmi. Následne sú v dokumente uvedené kapitoly ako analýza, v ktorej sa nachádza podrobná analýza všetkých dôležitých častí systému, ktoré bolo pre správne riešenie potrebné analyzovať. Taktiež implementácia a návrh jednotlivých častí systému, ktoré vyplývajú z celkového pohľadu na systém. Na konci dokumentu, je uvedené testovanie jednotlivých funkčných častí produktu, ako aj výsledné testovanie integrácie všetkých častí. Toto testovanie predstavuje pre systém neoddeliteľnú súčasť, nakoľko z pohľadu Product Ownera predstavuje toto testovanie aj formu akceptačných testov.

Našim cieľom je navrhnúť a implementovať zariadenie na automatické testovanie meracích zariadení. Zariadenie bude simulovať správanie sa fyzikálnych veličín generovaním napätia v rozsahu <-10V, +10V>. Simuláciu bude možné spustiť pomocou digitálnych vstupov. V rámci projektu bude navrhnuté rozhranie REST API, ktoré umožní programovanie rôznych simulácií. Následne bude zariadenie integrované do existujúceho frameworku pre automatické testovanie (Robot framework). V rámci projektu budú vytvorené nástroje, ktoré umožnia simuláciu správania sa rôznych fyzikálnych veličín.

Projekt má trvanie dvoch semestrov a preto je obmedzené množstvo práce času, ktoré mu je venované. Ohraničenia tohto projektu sa vzťahujú na vytvorenie funkčného prototypu zariadenia na automatizované testovanie meracích zariadení. Tento prototyp bude možné rozšíriť aj mimo tohto projektu. Taktiež bude možné vytvárať nové testy pomocou existujúceho frameworku (Robot Framework) s využitím vytvoreného zariadenia. Výsledný

produkt bude majetkom firmy Kistler spolu so všetkými jeho súčasťami ako aj zdrojovými súbormi.

1. Globálne ciele

1.1. Zimný semester

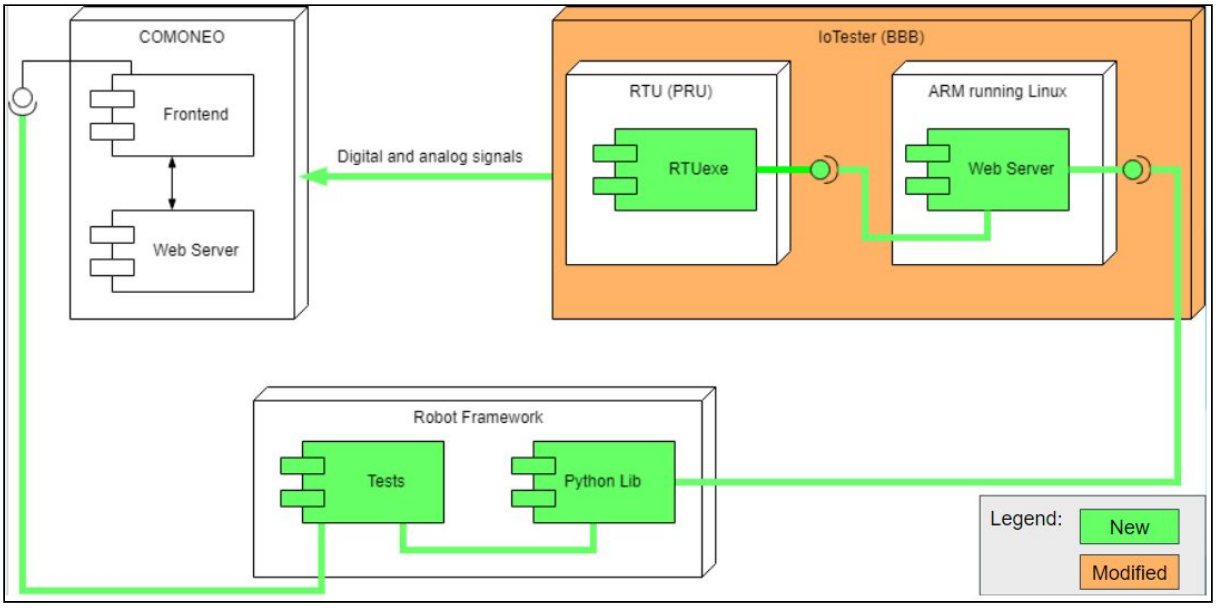
Cieľom zimného semestra je testovanie digitálneho signálu odosielaného z BeagleBonu Black na zariadenie COMONEO od spoločnosti Kistler. Cez COMONEO sa vytvorí test, ktorý odosiela konfiguráciu signálov na BeagleBone Black. Na základe konfigurácii BeagleBone Black vygeneruje signály. Tento test taktiež nastaví COMONEO tak, aby očakával konfigurované signály na vstupe a v prípade ak na vstupe sú očakávané signály, test sa považuje za úspešný.

Prioritou zimného semestra bolo oboznámiť sa s používanými technológiami a vytvoriť základný prototyp, ktorý by obsiahol všetky časti systému.

1.2. Letný semester

Cieľom letného semestra je vytvorenie funkčného prototypu IoTester. Pre vytvorenie takéhoto prototypu je potrebné využiť poznatky nadobudnuté v zimnom semestri. V zimnom semestri sme testovali digitálny signál. Tento digitálny signál je rozšírený o funkcionalitu nastavenia analógových signálov a viacerých digitálnych signalov. IoT tester dokáže generovať požadované digitálne a analógové signály a nastavovať ich na zariadení ComoNeo. Pomocou Robot Framework je vytvorený test, ktorý odošle pomocou REST API hodnoty signálov v čase. Tieto hodnoty sú prijaté serverom, ktorý beží na doske BeagleBone Black. Hodnoty sú na doske zapísané do spoločnej pamäte a PRU je informovaná správou o začatí generovania signálov. PRU generuje signály podľa dát uložených v spoločnej pamäti. Krivky sú zobrazené na grafickom rozhraní zariadenia ComoNeo. Test overí výstupy zariadenia ComoNeo a vyhodnotí výsledky.

2. Celkový pohľad na systém



Obrázok 1: Architektúra systému

Architektúra systému pozostáva z trocha dôležitých častí:

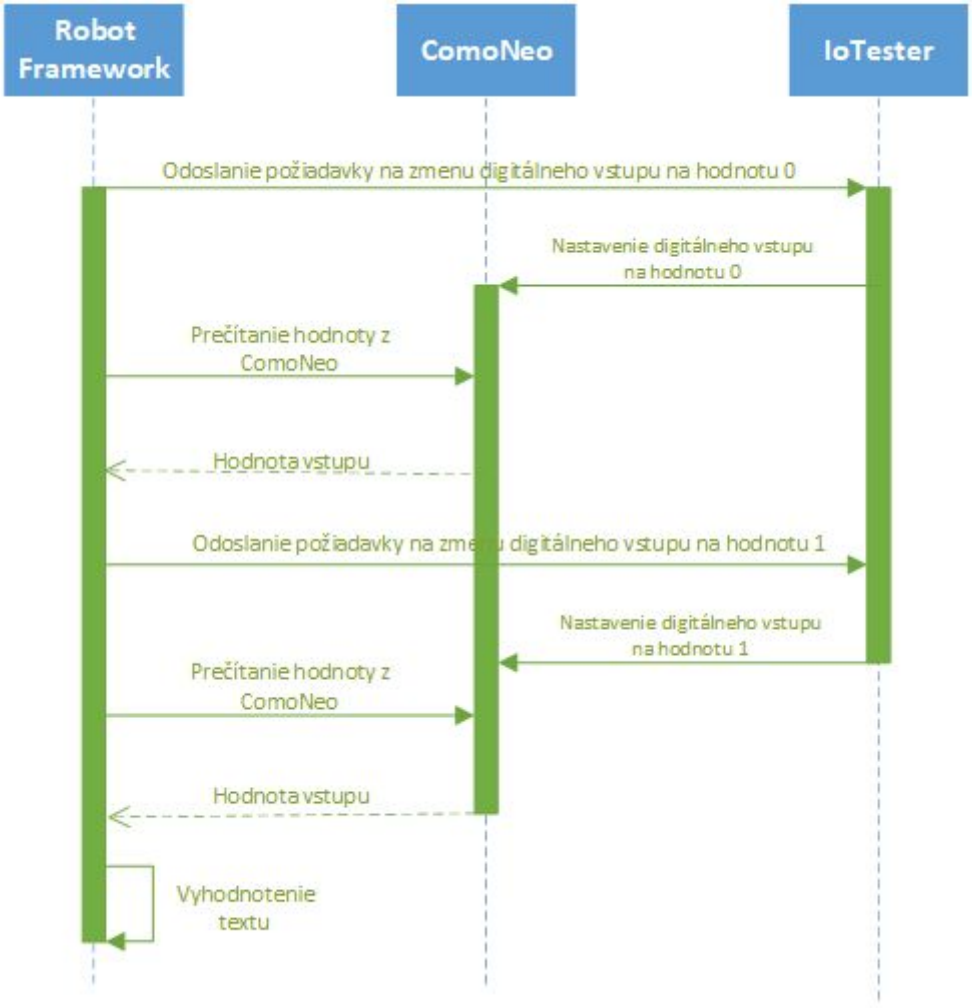
1. COMONEO
2. Robot Framework
3. IoTester (BBB)

COMONEO je zariadenie od spoločnosti Kistler, ktoré umožňuje viacero funkcionalít. Z pohľadu nášho systému, toto zariadenie očakáva digitálne alebo analógové signáli na svojom vstupe od zariadenia IoTester (BBB - Beaglebone Black).

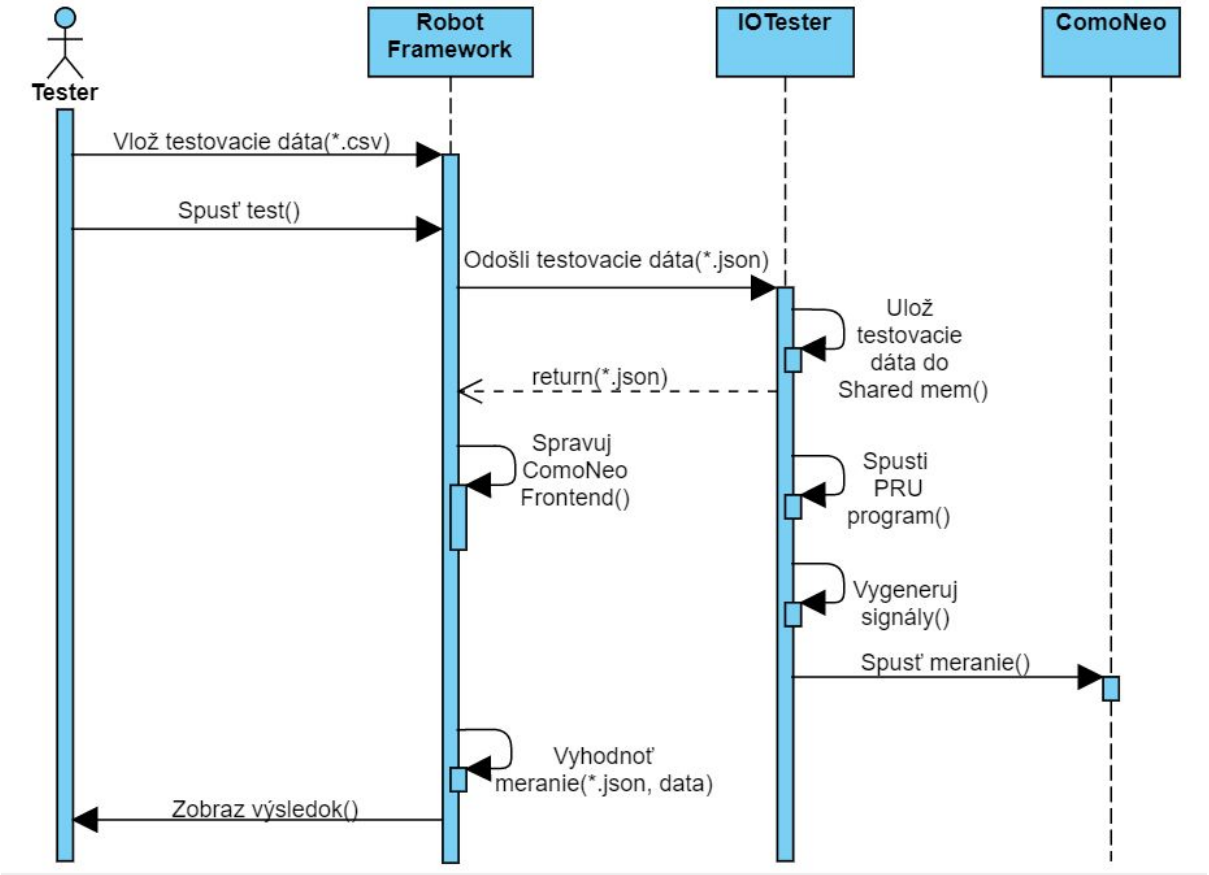
Grafické rozhranie na COMONEO zobrazuje merané vstupné dáta, ako čo sú analógové a digitálne signály. Toto grafické rozhranie je možné ovládať automatický za pomoci Robot Frameworku.

Na základe testov zadefinovaných v Robot Frameworku sa nastavuje grafické rozhranie, z ktorého sa vyčítajú, aké vstupné dáta má COMONEO. Zároveň sa odosielajú požadované signály na IoTester, ktoré má vygenerovať. V prípade ak sa vygeneroval požadovaný signál z IoTestera a bol aj zobrazený v grafickom rozhraní na COMONEO, tak sa považuje test za úspešný, inak sa považuje za neúspešný.

Na IoTester procesorovom čipe ARM je operačný systém Linux, kde je spustení Web Server, ktorý získava konfigurácie signálov na vygenerovanie z Robot Frameworku. Tieto konfigurácie sa prieposielajú na program, ktorý je spustení na mikrokontroléry RTU ("Real Time Unit") alebo známy aj pod názvom PRU ("Programmable Real time Unit"). Tento mikrokontrolér vygeneruje požadované signály v reálnom čase na I/O pinoch Beaglebone Black zariadenia. Vzľadom na hardvérové obmedzenia PRU, všetky dáta je nutné predpripraviť za pomoci procesorového čipu ARM. Komunikácia medzi ARM a PRU je dôležitá, aby sa oboznámilo PRU že dáta sú pripravené a môže vygenerovať požadované signály.



Obrázok 2: Sekvenčný diagram testovania digitálneho vstupu



Obrázok 3: Sekvenčný diagram výsledného fungovania aplikácie

2.1. Zoznam priložených e-dokumentov

Na priloženom elektronickom médiu sa nachádzajú všetky zdrojové súbory pre implementované funkcionality. Nachádzajú sa tu aj všetky dokumentácie vytvorené počas zimného semestra ako napríklad metodiky, inžinierske dielo alebo dokumentácia riadenia projektu.

Obsah elektronického média:

- Docs – obsahuje všetky dokumenty
- RobotFramework – súbory testovacieho frameworku
- RTU – súbory jednotky reálneho času

3. Moduly systému

3.1. Analýza

Pri riešení projektu bolo potrebné naštudovať si viaceré technológie s ktorými sa stretneme počas návrhu a implementácie zadania. Táto kapitola je venovaná práve týmto technológiám. Vysvetľuje, ktoré technológie sme použili, aké predpoklady museli byť splnené, aby sme s týmito technológiami mohli pracovať a taktiež vysvetľuje ako samotné technológie fungujú a načo sa používajú. V nasledujúcich kapitolách sú podrobne analyzované webové servery, vývojová doska BeagleBone Black, Real Time Unit ako aj samotný prototyp zariadenia, ktoré nám bolo poskytnuté.

3.1.1. Analýza dosky

Ako vstup projektu bola nášmu tímu poskytnutá vývojová doska, pomocou ktorej je možné testovanie zariadenia ComoNeo. Jednou z hlavných úloh nášho tímu je návrh nového kompaknejšieho dizajnu tejto dosky, kedže doska, je momentálne iba prototyp. Prvým krokom pri návrhu novej dosky však bola analýza pôvodnej dosky a to konkrétne z dôvodu analýzi jednotlivých komponentov ich zapojenia a možnosti redukovania niektorých vybraných komponentov. Hlavným cieľom bolo rozloženie evaluačnej dosky prevodníkov digitálnych signálov na analógové.

Prvým krokom pri analýze dosky bolo preloženie a prečítanie si bakalárskej práce, ktorej výsledkom bola práve táto doska analyzovaná doska. Pri analýze dosky sme sa zamerali hlavne na zistenie prepojenia vývojovej dosky Beaglebone Black s ostatnými komponentami dosky akými sú evaluačná doska prevodníka digitálnych na analógové signály ako aj na zistenie, ktoré porty na doske Beaglebone zabezpečujú komunikáciu s touto evaluačnou doskou ako aj digitálnymi konektormi.

3.1.1.1. Zapojenie dosky Beaglebone Black

Doska Beaglebone Black pozostáva z dvoch hlavných sád portov s názvami P9 a P8. Napájanie dosky zabezpečujú v našom návrhu porty 5 a 6 s názvami 5V RAW na sade portov P9, do ktorých je privedené napätie 5V+, ktoré je vstavaným regulátorom automaticky prevedené na pracovné napätie dosky 3V. Porty 1, 2, 43, ,44, 45, 46 taktiež na sade P9 a 1, 2 na sade P8 zabezpečujú naopak uzemnenie dosky a sú napojené na zem.

Sada portov P9 v našom návrhu zabezpečuje pomocou portov 24, 26, 28, 31 , 30 komunikáciu s evaluačnou doskou prevodníka digitálnych na analógové signály(DAC8734EVM) a to konkrétne pomocou SPI zbernice SPI0. Konkrétne zapojenie portov je zobrazené v **tabuľke číslo 1**. Pomocou týchto evaluačných dosiek, z ktorých každá obsahuje 4 analógové výstupy sú ovládané konektory X24.

Tabuľka číslo 1.

Beaglebone Black	**DAC8734EVM**	**Názov signálu**
GPIO0_12 (24)	7	RST
GPIO0_13 (26)	6	LDAC
GPIO0_17 (28)	2	CS
GPIO0_16 (30)	4	SDI
GPIO0_14 (31)	3	SCLK

Sada portov P8 v našom návrhu zabezpečuje naopak komunikáciu s digitálnymi konektormi. Keďže pracovné napätie dosky Beaglebone Black je 3V a zariadenia ComoNeo je 24V je potrebné zapojiť medzi tieto dva komponenty prevodníky napätia, ktorými sú v

našom prípade komponenty IC102 a IC103. Zapojenie týchto komponentov je zobrazené v **tabuľke číslo 2**.

Tabuľka číslo 2.

Beaglebone Black	**IC102**	**IC103**
GPIO1_0 (25)	18	
GPIO1_1 (24)	17	
GPIO1_2 (5)	16	
GPIO1_3 (6)	15	
GPIO1_4 (23)	14	
GPIO1_5 (22)	13	
GPIO1_6 (3)	12	
GPIO1_7 (4)	11	
GPIO1_14 (37)		11
GPIO2_15 (38)		12
GPIO2_16 (36)		13
GPIO2_17 (34)		14

Porty zapojené do prevodníka IC102 obsluhujú konektory X14, X11 a porty zapojené do prevodníka IC103 zase konektory X12, X15.

3.1.1.2. Zapojenie evaluačnej dosky DAC8734EVM

Druhou a obšírnejšou časťou analýzy bola analýza samotnej evaluačnej dosky DAC8734EVM. Keďže výstupom projektu má byť taktiež upravená doska hlavným cieľom je odstrániť z návrhu evaluačné dosky a nahradiť ich iba potrebnými komponentami.

Hlavným komponentom dosky je prevodník DAC8734E. Pri analýze tejto dosky sme zistili nasledovné zapojenie portov (**tabuľka číslo 3**).

Tabuľka číslo 3.

DAC Port	**Názov portu**	**Potrebné zapojenie**
1,36,37,25,24,19,12,42	NC	NIE
2	CS	ÁNO
3	SCLK	ÁNO
4	SDI	ÁNO
5	SDO	NIE
6	LDAC	ÁNO
7	RST	ÁNO
8	GPIO-0	NIE
9	GPIO-1	NIE
10,48	UNI/Bip A/B	NIE
11,27	DGND/AGND	ÁNO

31,32	REF A, REF B	ÁNO
13	IOVDD	NIE
26,35	AVDD	ÁNO
28,33	AVSS	ÁNO
14	DVDD	ÁNO
29,32	REFGND-A/B	ÁNO
15,23,46,36	Vout-0/1/2/3	ÁNO
18,20,43,41	SGND-0/1/2/3	ÁNO
17,16,21,22,44,45,40,39	RFB	NIE

Na evaluačnej doske sa nachádza niekoľko „jumprov“ a prepínačov, ktorými je túto dosku možné nastaviť. Pri analýze sme pre každý komponent zisťovali aký má účel a na akú hodnotu je nastavený. Podrobné informácie o prepínačoch sú dostupné v oficiálnom datasheete dosky. Podrobné zapojenie a analýza DAC dosky je priložená na elektronickom médiu.

3.1.2. Analýza webového servera

Cieľom bolo vybratie webového servera, ktorý by umožňoval komunikáciu medzi ARM a RTU a tiež ARM a Robot Framework-om.

Lighttpd

Výhody:

- Extrémne rýchly pre statický obsah web. serveru
- Schopný zvládnuť tisíce požiadaviek za sekundu
- Minimálna záťaž pamäte/CPU pri behu programu
- Neblokuje I/O operácie lebo beží ako single proces
- Pridávanie funkcionalít cez moduly

Nevýhody:

- Problémy so stabilitou
- Nekompatibilný s niektorými Apache modulmi
- Slabá podpora
- PHP
- Nemá zabudovanú podporu pre WSGI

Nginx

Výhody:

- Dokumentácia a podpora na vysokej úrovni
- Minimálna záťaž pamäte/CPU pri behu programu
- Schopný zvládnuť milióny požiadaviek za sekundu
- Pridávanie funkcionalít cez moduly
- Neblokuje I/O operácie lebo beží ako single proces

Nevýhody:

- Nevhodný pre malé projekty a scenáre s malým počtom požiadaviek
- Oproti Apache menej dostupných modulov pre rozšírenie funkcionalít

Flask

Výhody:

- Dokumentácia a podpora na vysokej úrovni
- Vhodný pre malé projekty a začiatočníkov
- Minimalistický Python framework
- Jednoduchá konfigurácia
- Flexibilný
- Veľa dostupných knižníc
- Nezávisí od ORM, preto je ľahká integrácia s databázami

Nevýhody:

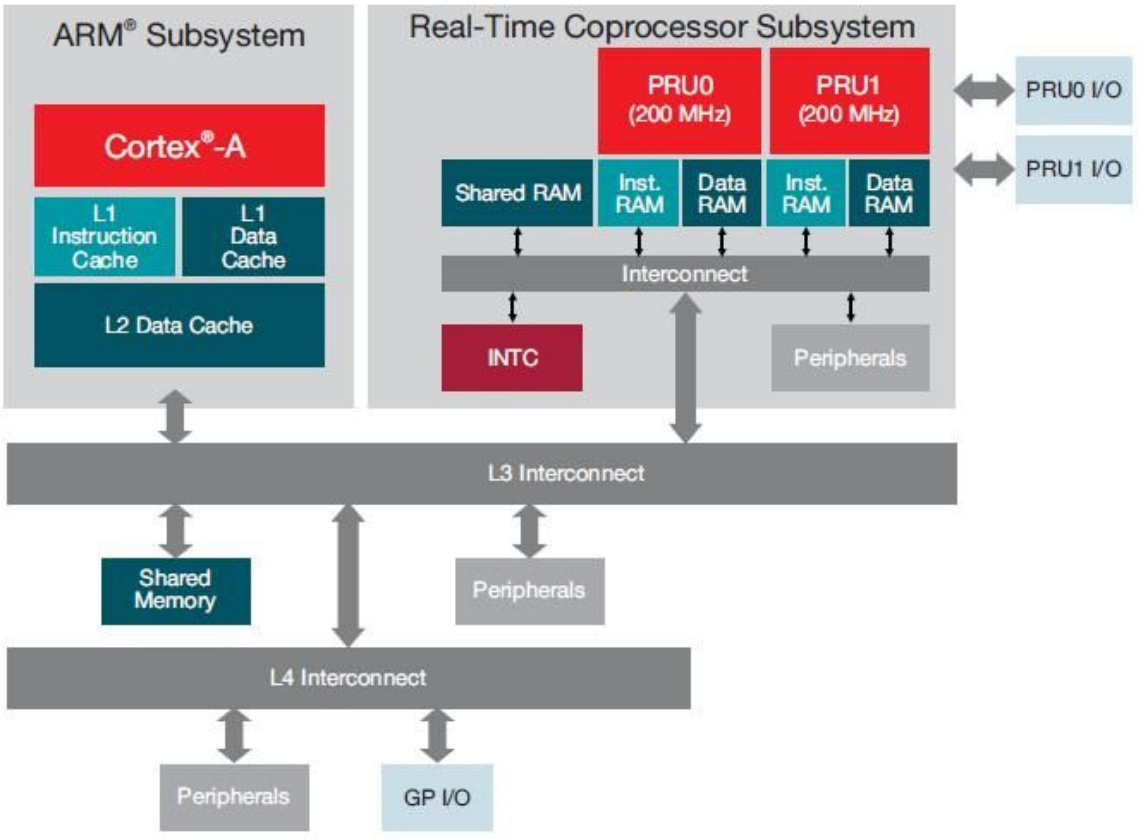
- Ťažšie async. programovanie
- Limitovanie v niektorých funkciách
- Väčšie projekty si vyžadujú dôkladné preštudovanie frameworku

Vybrali sme si web server Flask najmä kvôli flexibilite a vhodnosti pre malé projekty. Taktiež pri vyberaní zavážila aj skutočnosť, že product owner už na začiatku prvého stretnutia nám odporučil Flask.

3.1.3. Analýza BeagleBone Black

BeagleBone Black je lacný počítač s veľkosťou kreditnej karty, ktorý má dva vstavané mikrokontroléry nazývané PRU. PRU poskytuje schopnosť spracovania v reálnom čase, ktoré chýbajú v systéme Linux.

BeagleBone používa Sitara AM3358, je to procesorový čip ARM bežiaci na frekvencii 1 GHz. Na vykonávanie operácií v reálnom čase, procesor ARM BeagleBone nebude fungovať správne, pretože Linux nie je operačný systém v reálnom čase. Čip Sitara však obsahuje dva 32-bitové mikrokontroléry PRU (programmable real time unit). Použitím PRU je možné dosiahnuť rýchle, deterministické riadenie I / O pinov a zariadení v reálnom čase.

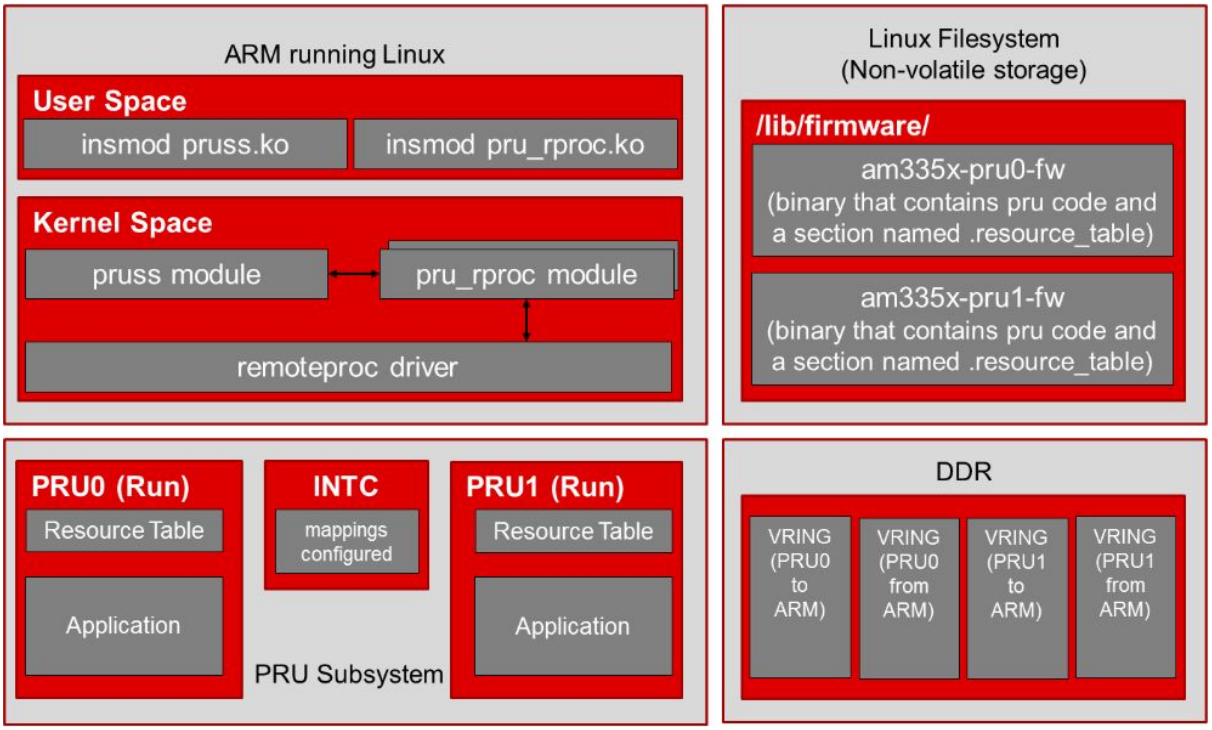


Obrázok 4: Architektúra ARM/PRU

Na používanie PRU je potrebné nainštalovať špeciálny linux. Tento linux musí byť schopný poskytnúť nasledovné služby:

1. Vložiť firmvér do PRU jadra

2. Riadiť vykonávanie programu na PRU (štart, stop, atď.)
3. Správu prostriedkov (pamäť, mapovanie prerušení atď.)
4. Umožniť odosielanie/prijímanie správ



Obrázok 5: Časti ARM/PRU

Na obrázku vyššie sú znázornené štyri bloky:

1. ARM na ktorom beží Linux
2. Linux súborový systém
3. PRU podsystém
4. DDR pamäť

Všetky tieto služby sú poskytované prostredníctvom pru_rproc a rpmsg_pru ovládačov.

V jadre sa nachádza Remoteproc ovládač. Remoteproc je framework, ktorý umožňuje ARM procesoru načítanie firmvéru do jadier PRU, spúšťanie jadier PRU, zastavenie jadier PRU a konfigurovanie prostriedkov (resources), ktoré môžu PRU počas behu potrebovať.

Sysfs rozhranie sa nachádza v používateľskom priestore, pomocou neho vieme spustiť alebo zastaviť PRU jadrá a načítať firmvér.

Binárne firmvér súbory sa nachádzajú v /linux/filesystem/ adresáry.

Postup načítavania firmvéru a spustenia programu:

- Pru_rproc modul musí predtým, ako čo načíta firmvér do PRU zistiť, či sa firmvérové binárne súbory nachádzajú v /lib/firmware/ .Modul pru_rproc tiež analyzuje binárne súbory firmvéru a hľadá sekciu s názvom .resource_table. Sekcia .resource_table firmvéru špecifikuje systémové prostriedky, ktoré PRU budú potrebovať počas vykonávania programu.
- Modul pru_rproc konfiguruje všetky zdroje, ktoré firmvér potrebuje.V tomto prípade to zahŕňa vytvorenie vrstiev v pamäti DDR na komunikáciu, ako aj nastavenie mapovania prerušenia v module INTC PRU subsystému.
- Modul pru_rproc potom načíta binárne súbory do inštrukčnej pamäti PRU a taktiež skopíruje resource table do dátovej pamäti PRU.
- Keď je všetko nakonfigurované a program sa nachádza v pamäti, modul pru_rproc spustí vykonanie programu na PRU.

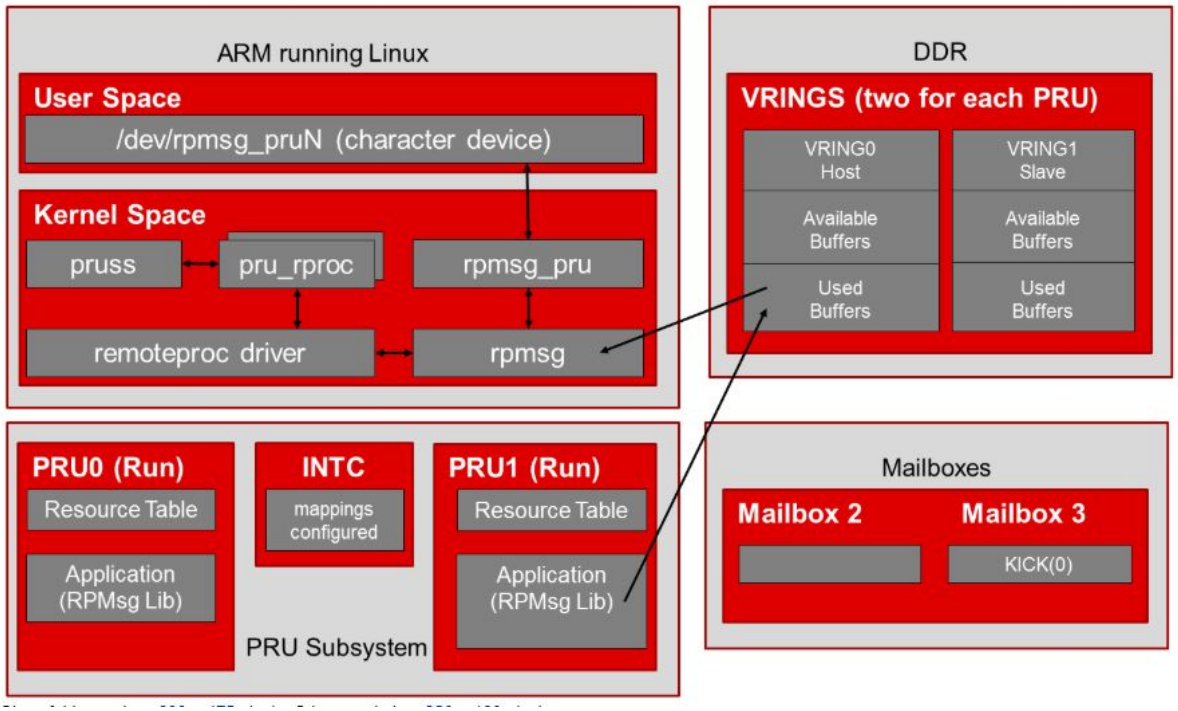
Príklad načítania firmvéru do PRU a spustenia programu:

- echo 'am335x-pru0-fw' > /sys/class/remoteproc/remoteproc1/firmware
- echo 'start' > /sys/class/remoteproc/remoteproc1/state

RPMsg je mechanizmus posielania správ, ktorý požaduje prostriedky prostredníctvom remoteproc a beží nad virtio frameworkom. Zdieľané vyrovnávacie pamäte (buffer) sú požadované prostredníctvom resource_table a poskytované remoteproc modulom počas načítania firmvéru do PRU. Zdieľané vyrovnávacie pamäte sa nachádzajú vnútri vring

dátovej štruktúry v pamäti DDR. Každé PRU jadro má k dispozícii dve vring, jedna sa používa pre správy prenesené na ARM a druhá sa používa pre správy prijaté z ARM. Systémové mailboxy sa používajú na oznamovanie jadrom (ARM alebo PRU), keď nové správy čakajú v zdieľaných vyrovnávacích pamätiach.

K dispozícii sú dve softvérové implementácie RPMsg. Na strane ARM Linuxu je komunikácia RPMsg prijatá v priestore jadra. Je poskytnutý modul rozhrania (rpmsg_pru), v používateľskom priestore, takže používatelia môžu zapisovať / čítať do / zo znakového zariadenia na odosielanie / prijímanie správ do / z PRU. Na strane PRU je k dispozícii knižnica RPMsg v PRU softvérovom balíku podpory (Software support package), ktorej cieľom je umožniť prepojenie, kde používateľský kód môže jednoducho volať funkcie pru_rpmsg_receive a pru_rpmsg_send, aby komunikoval s jadrom ARM.



Obrázok 6: Časti ARM/PRU správy

ARM - PRU správy

Na obrázku nižšie je znázornený proces posielania správ z ARM na PRU.

ARM:

1a. Alokuj vyrovnávaciu pamäť

alebo

1b. Získaj použitú vyrovnávaciu pamäť zo slave Vring

2. Skopíruj dáta do vyrovnávacej pamäte

3. Pridaj naplnenú vyrovnávaciu pamäť do zoznamu dostupných v slave Vring

4. Naštartuj slave Vring zapísaním jeho indexu (1) a odoslaním správy do Mailbox 2

PRU:

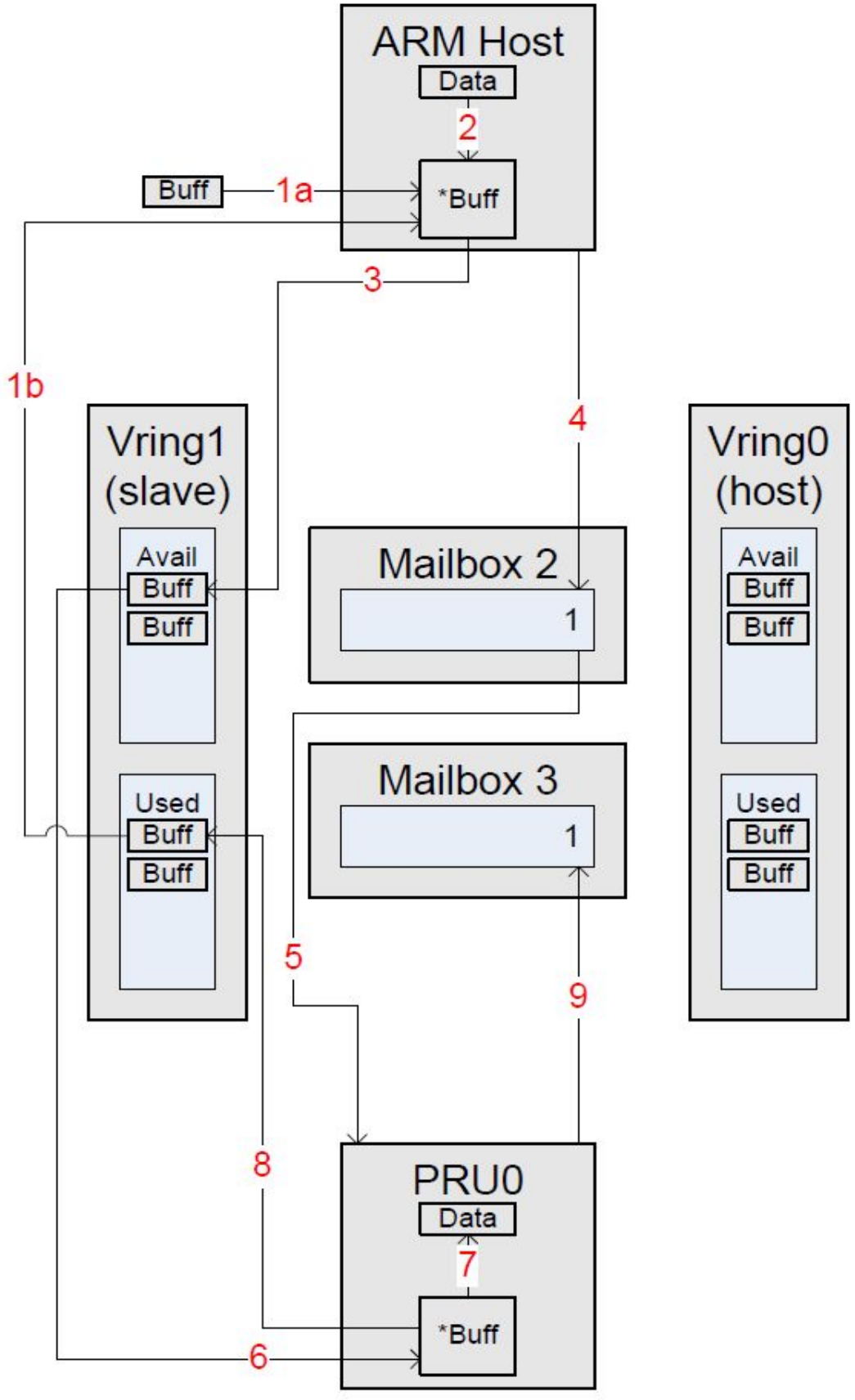
5. V Mailbox 2 sa nájde správa s indexom Vring-u (1), čo znamená že dáta sú pripravené na príjem.

6. Získaj vyrovnávaciu pamäť z slave Vring

7. Skopiruj dáta z vyrovnávacej pamäte z kroku 2.

8. Pridaj vyrovnávaciu pamäť do zoznamu použitých v slave Vring.

9. Naštartuj slave Vring zapísaním jeho indexu (1) do správy v Mailbox 3.



Obrázok 7: ARM -> PRU komunikácia

PRU - ARM správy

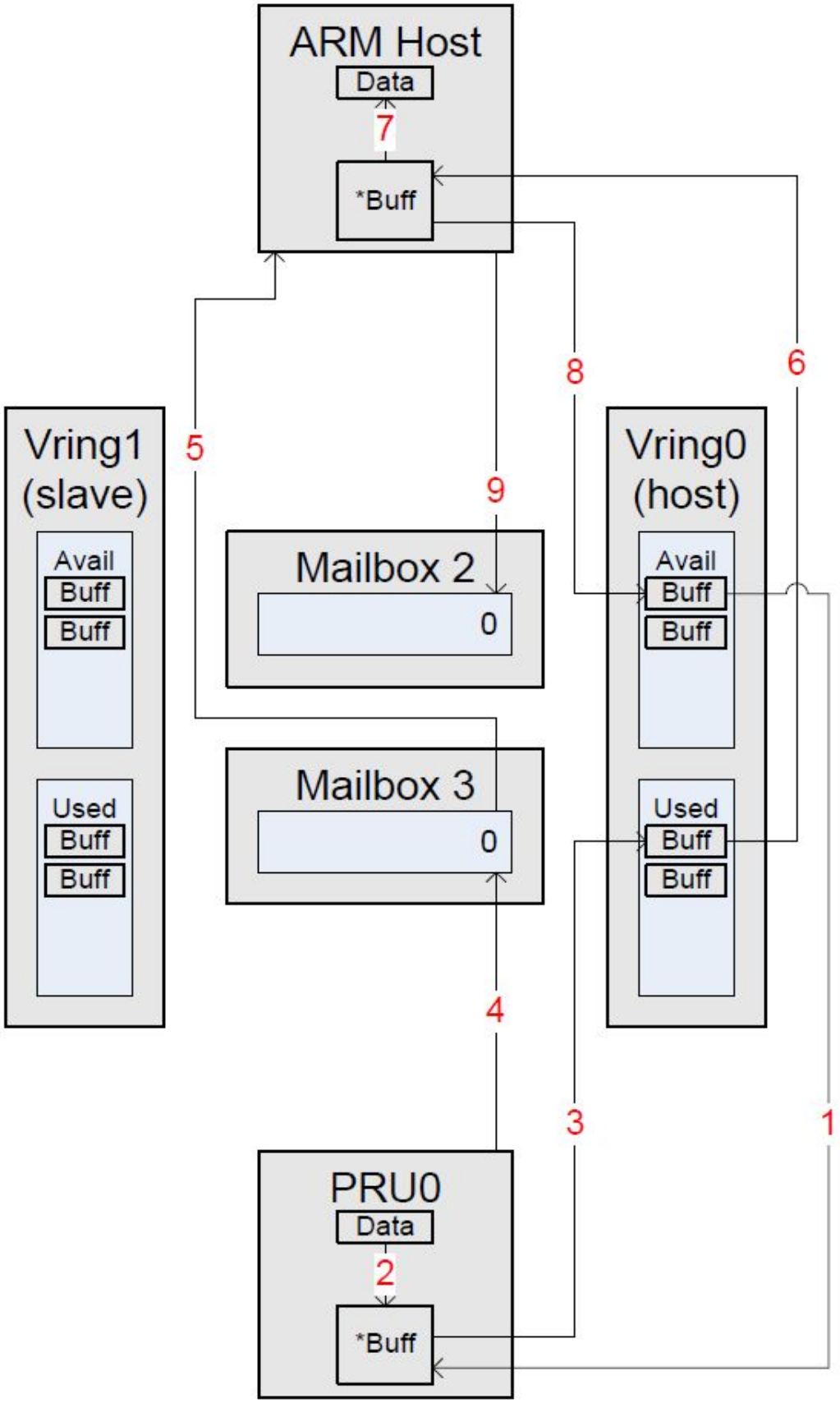
Na obrázku nižšie je znázornený proces posielania správ z PRU do ARM.

PRU:

1. Získaj voľnú vyrovnávaciu pamäť z host Vring
2. Skopiruj dáta na prenos do vyrovnávacej pamäti
3. Pridaj naplnenú vyrovnávaciu pamäť do zoznamu použitých v host Vring
4. Naštartuj host Vring zapísaním jeho indexu (0) do správy v Mailbox 3

ARM:

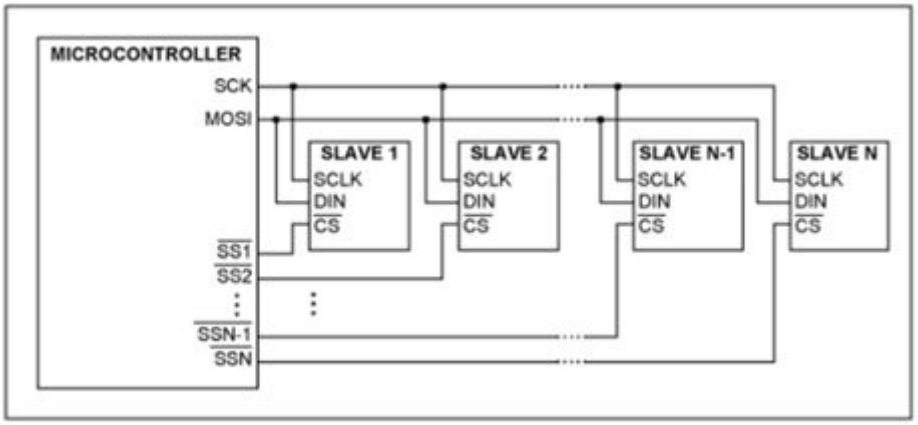
1. Prerušenie indikuje, že Mailbox 3 má správu od Vring (0). To oznamuje ARM procesoru že má dostupné dáta na príjem
2. Získaj použitú vyrovnávaciu pamäť z host Vring
3. Skopiruj dáta na príjem z vyrovnávacej pamäti z kroku 2.
4. Pridaj prázdnu vyrovnávaciu pamäť do zoznamu dostupných v host Vring
5. Naštartuj host Vring zapísaním jeho indexu (0) do správy v Mailbox 2.



Obrázok 8: PRU -> ARM komunikácia

4.1.3 Daisy-chain koncept

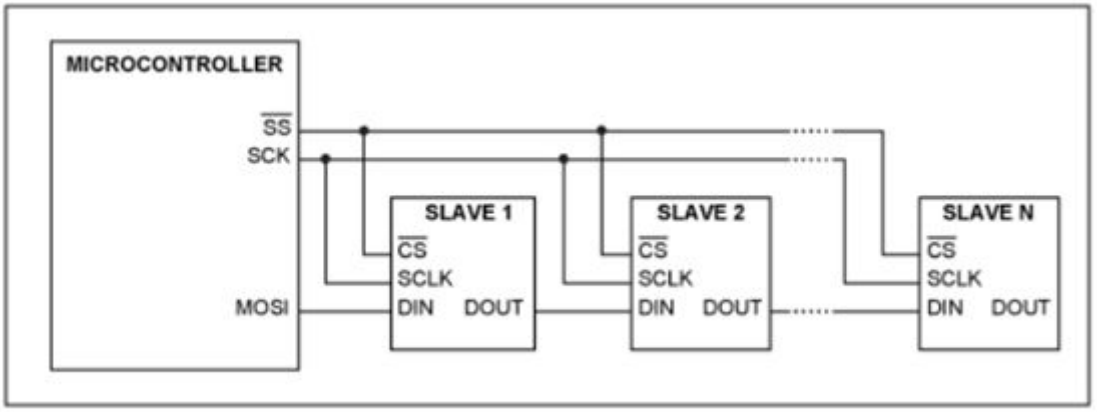
Štandardné SPI komunikuje so SLAVE zariadeniami pomocou 3 alebo 4-linkového (wire) seriového rozhrania. Typické rozhranie má chip-select signál (CS), serial clock (SCLK), data input signal (DIN) a môže obsahovať aj data output signal (DOUT).
Veľkú väčšinu SPI zariadení nemožno individuálne adresovať. Komunikácia medzi týmito zariadeniami a jedným zariadením na zbernici si vyžaduje dodatočnú hardvérovu alebo aj softvérovú podporu.



Obrázok 5: Štandardné SPI

Na hore uvedenej fotke, mikroprocesor používa jeden serial clock output (SCK) a jeden master-out slave-in (MOSI) na ovládanie všetkých SLAVE zariadení. Na individuálne adresovanie jednotlivých SLAVE zariadení, mikroprocesor používa slave-select (SSx) signál. Keďže všetky SLAVE zariadenia zdielajú SCLK aj DIN linky, len SLAVE zariadenia ktoré majú nastavené CS vstupy na nízku úroveň budú adresované.
Tento systém je jednoduchý na implementáciu keď nemáme veľký počet SLAVE zariadení. So zvyšovaním počtu SLAVE zariadení, zvyšuje sa aj počet SSx výstupov na mikropočítači.

Alternatívnou metódou pre sériové rozhrania je daisy-chain, ktoré šíri príkazy prostredníctvom sériovo zapojených zariadení.

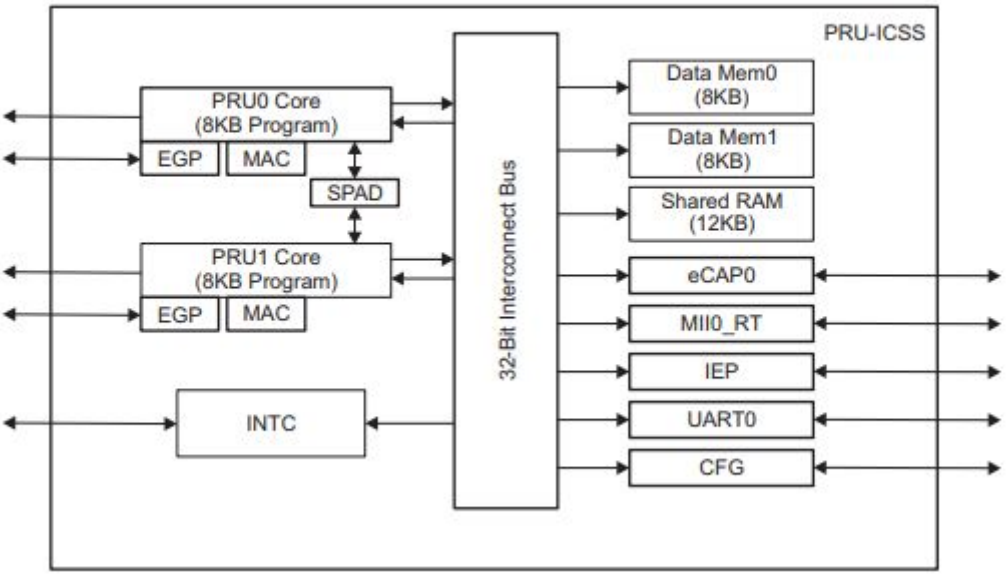


Obrázok. 6: Daisy-chain koncept

Jeden SS signál sa používa na ovládanie všetkých CS vstupov SLAVE zariadení. Všetky SLAVE zariadenia dostávajú ten istý CLOCK signal. Len prvé SLAVE zariadenie dostáva vstupné dáta priamo od mikropočítača. Všetky ostatné dostanú data od predchádzajúceho zariadenie v sérii. V jednom clock cykle SLAVE prijme na DIN vstupné dáta alebo príkazy a v nasledovnom musí poslať cez DOUT tie isté dáta alebo príkaz. SLAVE môže vykonať príkaz ktorý dostal na vstupe len počas nábežnej hrany CS signálu (počas tohto sa dáta neposielajú ďalej na DOUT). Pokiaľ CS signál nie je aktívny, SLAVE ignoruje príkaz a pošle ho ďalej. Mikropočítač potrebuje len tri signály na ovládanie SLAVE zariadení: SS, SCK, MOSI.

3.1.4. Analýza spoločnej pamäte medzi PRU a ARM

Programovateľná jednotka v reálnom čase a komunikačný podsystém (PRU-ICSS) pozostáva z dvoch 32-bitových RISC procesorových jadier (Programmable Real-Time Units alebo PRUs), pričom tieto jednotky sú prepojené s pamäťovým podsystémom pomocou 32-bitovej zbernice.



Obrázok. 7: Spoločná pamäť Blok diagram

Pamäťový podsystém PRU obsahuje:

· Dve PRU s:

o 8KB programová pamäť

o 8KB dátová pamäť

· Jedna 12 KB všeobecná zdieľaná pamäť pre obe PRU

Start Address	PRU0	PRU1
0x0000_0000	Data 8KB RAM 0[1]	Data 8KB RAM 1[1]
0x0000_2000	Data 8KB RAM 1[1]	Data 8KB RAM 0[1]
0x0001_0000	Data 12KB RAM2 (Shared)	Data 12KB RAM2 (Shared)
0x0002_0000	INTC	INTC
0x0002_2000	PRU0 Control Registers	PRU0 Control Registers
0x0002_2400	Reserved	Reserved
0x0002_4000	PRU1 Control	PRU1 Control
0x0002_4400	Reserved	Reserved
0x0002_6000	CFG	CFG
0x0002_8000	UART 0	UART 0
0x0002_A000	Reserved	Reserved
0x0002_C000	Reserved	Reserved
0x0002_E000	IEP	IEP
0x0003_0000	eCAP 0	eCAP 0
0x0003_2000	MII_RT_CFG	MII_RT_CFG
0x0003_2400	MII_MDIO	MII_MDIO
0x0003_4000	Reserved	Reserved

Obrázok. 8: Spoločná pamäť, mapa lokálnej pamäte

Keď PRU0 pristupuje k Data RAM0 na adrese 0x00000000, PRU1 tiež pristupuje k Data RAM1 na adrese 0x00000000.Dátová RAM0 má byť primárna dátová pamäť pre PRU0 a Data RAM1 pre PRU1. Avšak na odovzdávanie informácií medzi PRU môže každá PRU pristupovať k údajovému ramenu ostatných PRU na adrese 0x0001_0000. Štartovacia adresa zdieľanej pamäte PRU je 0x4a312000 .

3.1.5. Analýza Robot Framework

V rámci testovania Robot Framework, bola spočiatku nevyhnutná analýza viacerých už existujúcich testov spoločnosti Kistler, pričom bola taktiež nutná analýza práce so samotnou technológiou Robot Framework a taktiež externou knižnicou Selenium, k čomu bola využívaná prevažne oficiálna dokumentácia k produktu Robot Framework (http://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html) spolu s rôznymi video návodmi, ktoré slúžili primárne na zozname sa s technológiu a jej osvojenie.

3.2. Návrh

Pred samotnou implementáciou bolo potrebné navrhnúť jednotlivé časti systému. Táto kapitola pozostáva z viacerých častí a popisuje návrh jednotlivých častí.

3.2.1. Návrh novej dosky

Pri návrhu novej dosky sme sa rozhodli zachovať zapojenie napájania z pôvodného návrhu dosky. Hlavným cieľom bolo rozdelenie evaluačnej dosky DAC na jednotlivé komponenty ako aj vyriešenie chýb, ktoré sa nachádzali v pôvodnom návrhu.

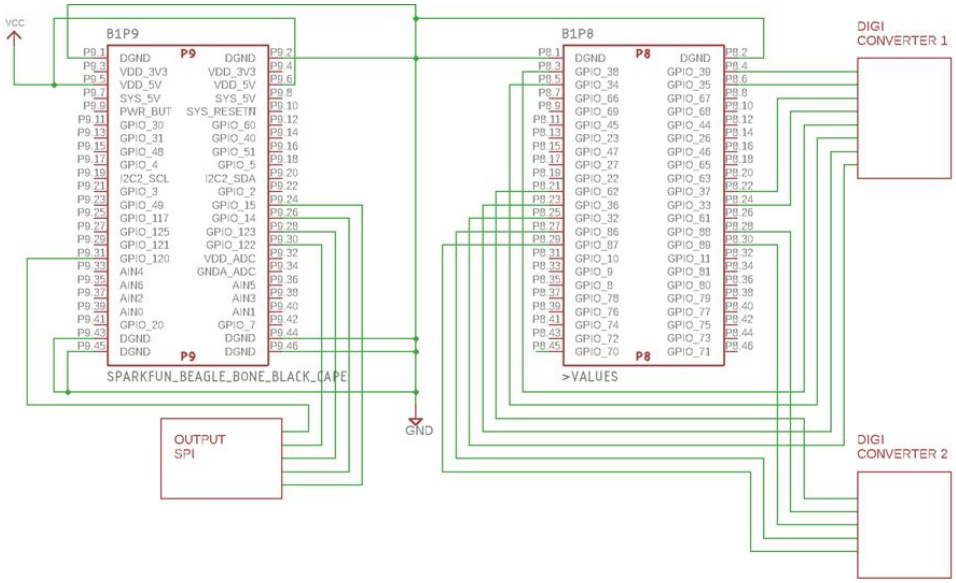
Prvou chybou, ktorú sme vyriešili bolo obsadenie bootovacích pinov v pôvodnom návrhu dosky. Konkrétne sa jedná o piny zodpovedné za zápis digitálnych hodnôt pre digitálne konektory. V tabuľke číslo 4 môžeme vidieť pôvodné rozloženie pinov spolu s novým návrhom rozdelenia týchto pinov.

Tabuľka číslo 4.

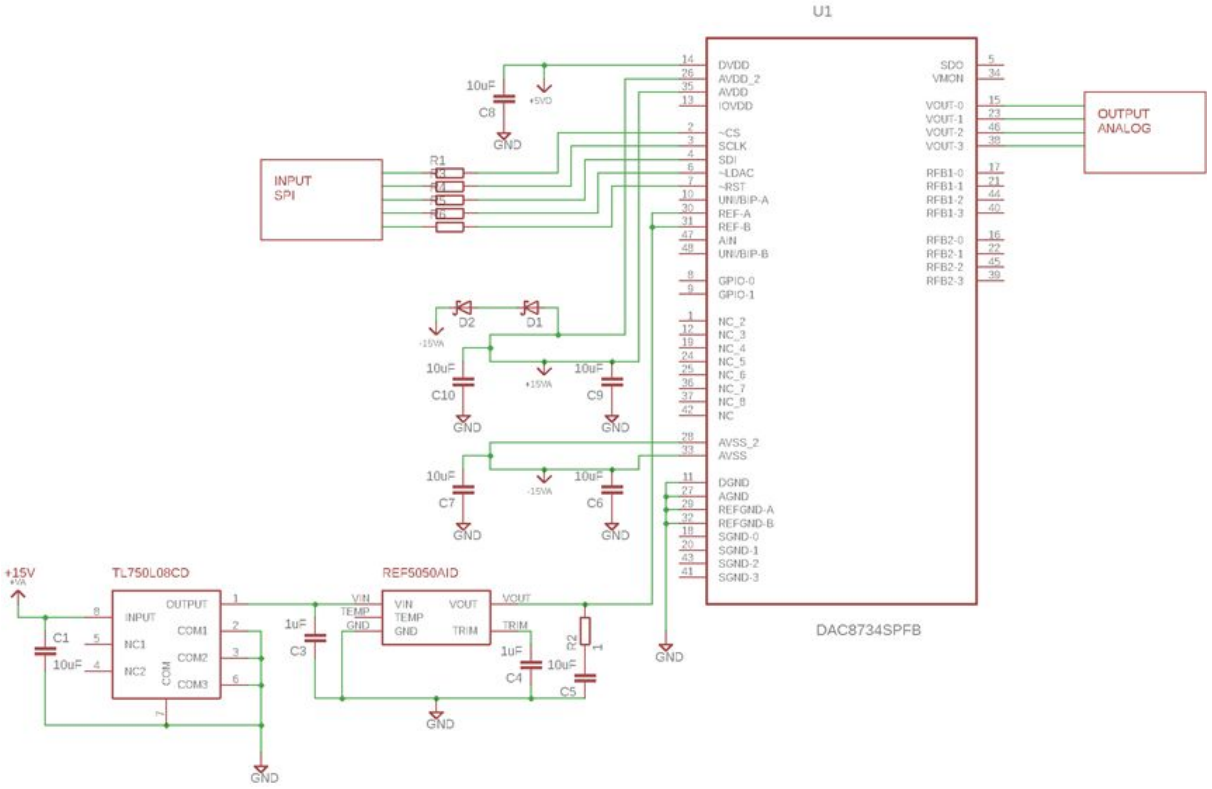
Starý pin	Nový pin
GPI01_14 (37)	GPI02_22 (27)

GPI02_15 (38)	GPI02_23 (29)
GPI02_16 (36)	GPI02_24 (28)
GPI02_17 (34)	GPI02_25 (30)

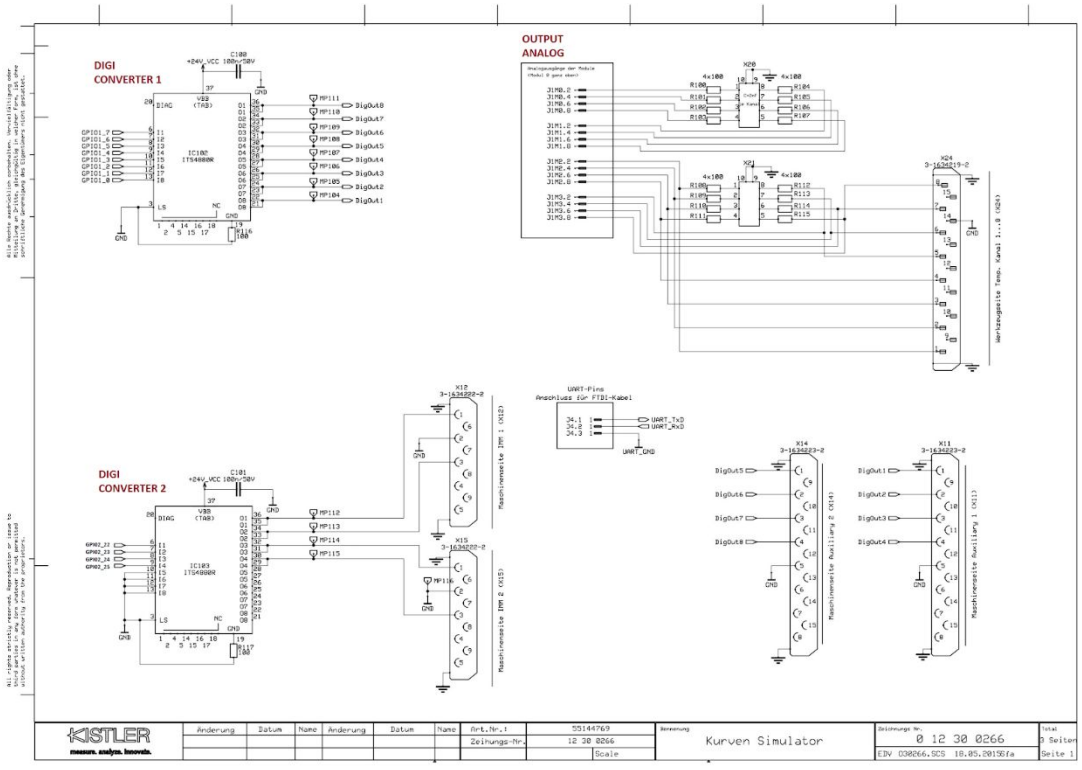
Návrh novej dosky sa následne vykonával pomocou kreslenia zapojení jednotlivých komponentov. Tieto zapojenia sú zobrazené na nasledujúcich obrázkoch.



Obrázok 9: Zapojenie pinov P9 a P8 dosky Beaglebone Black



Obrázok 10: Refractoring DAC dosky



Obrázok 11: Zapojenie konektorov ku doske Beaglebone Black

3.2.2. Návrh webového servera

Z pohľadu výsledného produktu ide o programové rozhranie, ktoré zabezpečuje komunikáciu centrálnej testovacej jednotky Robot framework so vzdialeným zariadením BeagleBone, ktoré pomocou príslušného modulu spoločnosti Kistler, komunikuje s monitorovacím zariadením Como Neo, taktiež od spoločnosti Kistler.

Na základe aktuálneho priebehu je žiaduce, aby REST API umožňovalo za pomoci POST REQUESTOV odosielanie jednoduchého digitálneho signálu(0/1). Preto môžeme hovoriť o prototype. V budúcnosti bude potreba toto rozhranie rozšíriť aj o zasielanie zložitejších signálov, čo sa podarilo načrtnúť už v doterajšom priebehu. Aktuálne REST API umožňuje zasielanie aj zložitejších signálov a to za pomoci dát zasielaných vo formáte JSON, čo môžeme považovať za výhodu a do budúcna.

3.2.3. Návrh programu pre komunikáciu PRU a CPU

Pre overenie analýzy a otestovanie funkčnosti PRU jednotky bolo potrebné navrhnúť jednoduchý program, vďaka ktorej vie systém pracovať v reálnom čase.

Požiadavkou tejto úlohy bolo vytvoriť program, ktorý bude čítať neustále zadávané premenné od používateľa(posielané z CPU na PRU) a umožní používateľovi zobraziť zadané premenné (preposlané späť z PRU na CPU).

Aby bolo možné zabezpečiť takúto funkcionalitu je potrebné vytvoriť komunikačný kanál RPMsg, v ktorom bude prebiehať komunikácia od používateľa k PRU obojstranne a dve metódy na zachytávanie a výpis správ.

3.2.4. Návrh programu pre spínanie digitálnych pinov z PRU

Pre samotné ovládanie digitálnych pinov pomocou PRU bolo potrebné navrhnúť program, ktorý reaguje na prichádzajúce správy z CPU. Správy sú v tvare “1” alebo “0”. Správy iných formátov sú programom ignorované. Program funguje na PRU v nekonečnom cykle, v ktorom počúva na prichádzajúce správy od CPU. Po prijatí správy “1” program

nastaví hodnoty požadovaných digitálnych výstupov na logickú jednotku. Po prijatí správy "0" program nastaví hodnoty požadovaných digitálnych výstupov na logickú nulu.

Tento program je založený na jednoduchom programe ktorý je navrhnutý vyššie.

3.2.5. Návrh finálneho programu pre nastavenie analógových a digitálnych výstupov z PRU

Pre nastavenie analógových a digitálnych výstupov bol navrhnutý program, ktorý využíval komunikáciu PRU a ARM procesora. Program reaguje na správu „spi" v smere z ARM procesora na PRU a po zachytení takejto správy začne vyčítavať hodnoty zo spoločnej pamäte. Je nevyhnutné aby tieto hodnoty boli v spoločnej pamäti uložené ešte pred odoslaním správy „spi". Hodnoty sú uložené v rámci procesora ARM. Pri čítaní hodnôt program najskôr vyčíta informáciu o počte bodov, ktorý je rovnaký pre každú krivku, ktoré je potrebné nastaviť na analógovom výstupe a ďalej počet SPI kanálov využitých pri konkrétnej krivke. Následne program vynuluje hodnoty na SPI zbernici nastavením hodnoty 0 na všetky kanály. Program spustí digitálny pin, ktorým oznamuje zariadeniu ComoNeo aby spustilo meranie. Program v tomto momente vchádza do slučky, ktorú opakuje pre zistený počet bodov. Pre každý bod program vyčíta hodnotu pre každý využívaný kanál a nastaví túto hodnotu na SPI zbernicu. Rovnako vyčíta s pamäti aj hodnotu digitálneho výstupu, ktorá predstavuje v binárnej forme reprezentáciu digitálnych pinov. Podľa tejto hodnoty nastaví hodnoty digitálnych pinov. Program taktiež vyčíta hodnotu oneskorenia, ktorá predstavuje množstvo času, ktoré musí program udržať nastavené digitálne a analógové hodnoty. Program čaká po dobu vyčítanú z pamäti. Uvedený cyklus opakuje pre každý bod v čase čoho následkom je vykreslenie kriviek s definovanými hodnotami v každom bode. Počet kriviek je rovnaký ako počet kanálov.

Zjednodušený postup vykonávania:

- Dáta sú uložené do spoločnej pamäti.
- Vyčítanie správ „spi"
- Vyčítanie hodnôt z pamäte.
- Spustenie merania pomocou digitálneho pinu.
- Nastavenie SPI a digitálnych hodnôt v cykle.
- Čakanie programu po dobu vyčítanú z pamäti pre konkrétny bod.
- Nastavenie všetkých hodnôt.

3.2.6. Návrh prvého prototypu pre REST API na BBB

Návrh prvého protypu REST API mal za úlohu vytvoriť aplikačné rozhranie medzi testovacím zariadením Robot Framework, ktorý predstavuje zároveň aj technológiu iniciujú jednotlivé testy so vzdialeným zariadením Beagle Bone Black, ktoré je priamo pripojené pomocou navrhnutej dosty priamo na testovacie zariadenie ComoNeo.

Požiadavkou na toto rozhranie bolo vytvoriť digitálne spojenie, ktoré bude umožňovať odosielanie jednoduchých digitálnych vstupov (0/1), k čomu sa využila predošlá analýza poskytovaných aplikačných rozhraní uvedená vyššie.

Aplikačné rozhranie je navrhnuté na platforme Flask (Python), ktorá umožňuje jednoduché zasielanie POST-ov na vzdialené zariadenie v tvare

(*http://IP_adreda_BBB:5000/digital_input?digIn=**1***), pričom číslo **1** uvedené na konci POST-u predstavuje zasielaný digitálny vstup.

3.2.7. Návrh JSON pre odosielanie analógového aj digitálneho signálov

Ďalej bolo nevyhnutné rozšírenie vyššie uvedeného prototypu aplikačného rozhrania o možnosť zasielania štruktúrovaných dát, ktoré predstavujú nevyhnutnú podmienku pre zasielanie digitálnych aj analógových signálov.

Štruktúra pre dáta, ktoré je potrebné zasielať na vzdialené zariadenie, pomocou JSON súborov:

```
struct Sample
{
   std::uint32_t sampleIndex; ///< Index of sample in the cycle.
   std::uint32_t digitalIn; ///< Status of digital inputs.
   std::int32_t channel[cChannelsInSample];
}
```

Pričom dáta budú zasielané, ako pole vo formáte JSON vyššie uvedených štruktúr:

{

"sampleIndex": "0",

"digitalIn": "256",

"channel": ["0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "-2751", "-1566", "-1190", "-2775", "-24", "-88", "-588", "477", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0"]

}

Príklad jedného prvku poľa JSON štruktúry.

Jednou zo základných podmienkou bolo taktiež prispôsobenie tohoto rozhrania tak, aby zasielané dáta boli parsované už na vyššej vrstve rozhrania a to teda na Rest API Beagle Bone Black, keďže parsovanie týchto dát na PRU, by bolo časovo a aj implementačne náročnejšie.

3.2.8. Návrh uloženia dát v spoločnej pamäti

Dáta opisujúce analógové a digitálne signály sú medzi PRU a ARM zdieľané prostredníctvom spoločnej pamäte nazývanej “shared memory”. Dáta sú tu uložené štruktúrovane. Na prvej 32 bitovej adrese sa nachádza počet bodov, ktorý je rovnaký pre každú krivku. Jeden bod znamená jednu zmenu hodnoty na krivke v čase. Na nasledovných 32 bitoch je uložený počet kanálov, ktorý sa bude v nasledujúcej generácii kriviek využívať. Počet kanálov je rovný počtu vygenerovaných kriviek. Ďalej nasleduje v pamäti časť ktorá sa opakuje pre každý bod a teda táto časť sa tam nachádza toľkokrát aký je počet bodov. Jedna takáto časť presne opisuje hodnoty jednotlivých SPI kanálov ako aj hodnoty digitálnych signálov. Veľkosť tejto časti závisí od počtu práve využívaných kanálov. Časť obsahuje časový interval, počas ktorého majú dané hodnoty byť nastavené na výstupoch. Tento interval je uložený ako 32 bitová hodnota a predstavuje počet 62,5 us časových úsekov (vzorková frekvencia zariadenia ComoNeo je 62,5 us). Ďalej nasleduje 32 bitová hodnota predstavujúca hodnotu digitálnych signálov. Jednotlivé bity predstavujú zapnutý alebo vypnutý digitálny pin. Následne sú v časti uložené 16 bitové hodnoty pre každý kanál. Časť sa opakuje pre každý bod na krivkách.

Adresa	Posun adresy	Údaj
0x4a312000	(+4)	Počet bodov na krivke
0x4a312004	(+4)	Počet kanálov
0x4a312008	(+4)	Časova dĺžka zápisu (Delay)
0x4a31200c	(+4)	Digital Input
0x4a31200e	(+2)	Hodnota kanálu 1
0x4a312010	(+2)	Hodnota kanálu 2
...	...	...
0x4a312014	(+4)	Časova dĺžka zápisu (Delay)
0x4a312018	(+4)	Digital Input
0x4a31201a	(+2)	Hodnota kanálu 1
0x4a31201c	(+2)	Hodnota kanálu 2
...	...	...
	Hlavička (iba na začiatku)	
	hodnoty vstupu v čase x	
	hodnoty vstupu v čase x+1	

Obrázok 12: Návrh uloženia dát v spoločná pamäti

3.2.9. Návrh Robot Framework

Pri návrhu testov pre Robot Framework sme sa snažili, aby testy boli ľahko modifikovateľné (nie napevno naprogramované), a aby vykonanie zmeny vyžadovalo čo najmenší zásah v kóde. Napríklad, aby sme vedeli odoslať digitálne hodnoty 0 aj 1 pomocou jedného testu, pričom v kóde sa bude meniť len posledné číslo.

Ako ďalšie sme navrhli, že testy sa budú skladať z troch častí:

- Tests - Obsahuje jednotlivé testy ktoré sa budú vykonávať. Tieto testy sú písané pomocou Keywords, ktoré sú uložené v ďalších dvoch častiach. Testy sú písané v jazyku Robot Framework.
- Resources - Obsahuje podrobnejšie kroky samotných testov. Testy sú písané v jazyku Robot Framework.
- ExternalKeywords - Obsahuje kroky testov, ktoré nie je možné vykonať pomocou knižnice SeleniumLibrary. Tieto kroky testov sú písané v jazyku python.

3.2.10. Návrh siete

Ako akceptačné kritérium úspešnej implementácie nášho návrhu bolo potrebné na zariadení ComoNeo zasvietit a zhasnúť požadované LED diódy predstavujúce digitalne konektory.

Zariadenie ComoNeo má pridelenú statickú IP adresu 192.198.197.151. Pre overenie správneho fungovania nášho prototypu bolo potrebné okrem zariadenia ComoNeo zapojiť do siete taktiež samotnú vývojovú dosku a jeden počítač, na ktorom bude bežať automatizovaný test, ktorý sa postará o zasvietenie a vypnutie LED diódy ako aj o overenie, či sa naozaj tak stalo. Pre prepojenie jednotlivých komponentov sme sa rozhodli vytvoriť si vlastnú sieť s maskou /24 a adresou siete 192.168.197.0. Zariadeniu ComoNeo prislúchala v našej sieti IP adresa 192.168.197.151, vývojovej doske IP adresa 192.168.197.154 a adresa počítača bola nastavená na 192.168.197.155. Na prepojenie jednotlivých komponentov sme použili sieťový prepínač.

3.3. Implementácia

3.3.1. Implementácia programu pre komunikáciu PRU a CPU

Pre správne pochopenie fungovania PRU a najmä komunikácie PRU a CPU bol použitý jednoduchý program. Program je voľne dostupný pre úpravu a použitie vo forme zdrojového kúdu ako aj v binárnej forme pod podmienkou dodržania distribučných pravidiel. Program beží na PRU. Tento program počúva v nekonečnej slučke na prichádzajúce správy od CPU. Po prijatí správy program odpovedá na prijatú správu preposlaním tejto správy späť. Komunikácia prebieha vo forme súboru do ktorého dokáže CPU zapísať správu a následne si z rovnakého súboru dokáže vyčítať odpoveď od PRU. Nižšie je zobrazená ukážka zo zdrojového kódu programu. Ukážka zobrazuje proces komunikácie v cykle po incializovaní všetkých potrebných premenných. Celý zdrojový kód je priložený na elektronickom médiu.

```
while (1) {
    /* Zistenie ci je spojenie s CPU aktivne */
```

```
    if (__R31 & HOST_INT) {
      /* Vycistenie statusu */
      CT_INTC.SICR_bit.STS_CLR_IDX = FROM_ARM_HOST;
      /*Prijatie vsetkych sprav od CPU */
      while (pru_rpmsg_receive(&transport, &src, &dst, payload, &len) ==
            PRU_RPMSG_SUCCESS) {
        /* Preposlanie spravy spat */
        pru_rpmsg_send(&transport, dst, src, payload, len);
      }
    }
}
```

3.3.2. Implementácia programu pre spínanie digitálnych pinov z PRU

Po otestovaní správneho fungovania PRU jednoduchým programom bol tento jednoduchý program rozšírený o funkcionalitu spínania digitálnych pinov dosky BeagleBone Black. Pre možnosť ovládania digitálnych pinov boli použité jednoduché správy. Obsah správ bol "1" alebo "0". Správa "1" spínala na digitálnych pinoch logickú jednotku, zatiaľ čo správa "0" spínala na digitálnych pinoch logickú nulu. Pred samotným počúvaním na správy bolo potrebné zavolať niekoľko inštrukcií. Inštrukcie boli zavolané pomocou inline assemblera. Tieto inštrukcie majú za úlohu aktiváciu OPC-master a tým uvoľnenie komunikácie s digitálnymi pinmi pre zapisovanie hodnôt.

```
__asm__ __volatile__
(
" LBCO &r0, C4, 4, 4 \n"
" CLR r0, r0, 4 \n"
" SBCO &r0, C4, 4, 4 \n"
);
```

Ďalej bolo potrebné zadefinovať požadovaným digitálnym pinom smer komunikácie a teda, že budú na tieto piny zapisované hodnoty a nebudú z nich hodnoty čítané.

```
GPIODirModeSet(SOC_GPIO_2_REGS, 14, GPIO_DIR_OUTPUT);
GPIODirModeSet(SOC_GPIO_2_REGS, 15, GPIO_DIR_OUTPUT);
GPIODirModeSet(SOC_GPIO_2_REGS, 16, GPIO_DIR_OUTPUT);
GPIODirModeSet(SOC_GPIO_2_REGS, 17, GPIO_DIR_OUTPUT);
```

Napokon bolo možné na tieto piny zapisovať požadované logické hodnoty. Tento zápis bol ovplyvňovaný prichádzajúcimi správami od CPU. Po zachytení správy obsahujúcej "1" boli všetky požadované digitálne piny nastavené na logickú hodnotu jedna. Po zachytení správy "0" boli všetky požadované piny nastavené na logickú hodnotu nula.

```
while (pru_rpmsg_receive(&transport, &src, &dst, payload, &len) ==
PRU_RPMSG_SUCCESS) {
        int i;
        for (i = 0; i < len; i++) {
        if (payload[i] == '1') {
                GPIOPinWrite(SOC_GPIO_2_REGS, 14, GPIO_PIN_HIGH);
                GPIOPinWrite(SOC_GPIO_2_REGS, 15, GPIO_PIN_HIGH);
                GPIOPinWrite(SOC_GPIO_2_REGS, 16, GPIO_PIN_HIGH);
                GPIOPinWrite(SOC_GPIO_2_REGS, 17, GPIO_PIN_HIGH);
        }
        else if (payload[i] == '0') {
                GPIOPinWrite(SOC_GPIO_2_REGS, 14, GPIO_PIN_LOW);
                GPIOPinWrite(SOC_GPIO_2_REGS, 15, GPIO_PIN_LOW);
                GPIOPinWrite(SOC_GPIO_2_REGS, 16, GPIO_PIN_LOW);
        GPIOPinWrite(SOC_GPIO_2_REGS, 17, GPIO_PIN_LOW);
        }
        }
}
```

Celý zdrojový kód tohto programu je dostupný na priloženom elektronickom médiu.

3.3.3. Implementácia finálneho programu pre nastavenie analógových a digitálnych výstupov z PRU

Navrhnutý program bol implementovaný ako rozšírenie pôvodného programu na spínanie digitálnych pinov. Postup fungovania programu je vysvetlený v časti návrhu. Konkrétne postupy implementácie sú uvedené nižšie.

Nastavenie analógových hodnôt

Významnou časťou programu je nastavovanie analógových hodnôt na SPI zbernici. Pre správne fungovanie SPI zbernice bolo potrebné v programe inicializovať SPI. Bola využitá knižnica „mcspi.h“. Inicializácia je vykonaná funkciou „InitSPI“, ktorá je obsiahnutá v zdrojových súboroch. Po inicializácii je SPI zbernica pripravená na nastavovanie požadovaných hodnôt. Hodnoty sú nastavené pomocou funkcie, ktorá taktiež využíva knižnicu „mcspi.h“. Táto funkcia postupne prechádza hodnoty jednotlivých kanálov a nastavuje tieto hodnoty na SPI zbernicu v cykle. Na nastavenie hodnôt je využitý princíp Daisy Chain, kde hodnoty sa postupne postupne zapisujú do hardvérových prevodníkov. Zápis funguje podobne ako fronta a prvá hodnota je postupne odsúvaná medzi prevodníkmi. Prevodníky sú využité štyri. Po zapísaní všetkých hodnôt pre každý kanál je zavolaná funkcia „UpdateDac“ ktorá resetuje digitálny pin LDAC a tým vystaví hodnoty na výstupy.

Zapisovanie digitálnych hodnôt

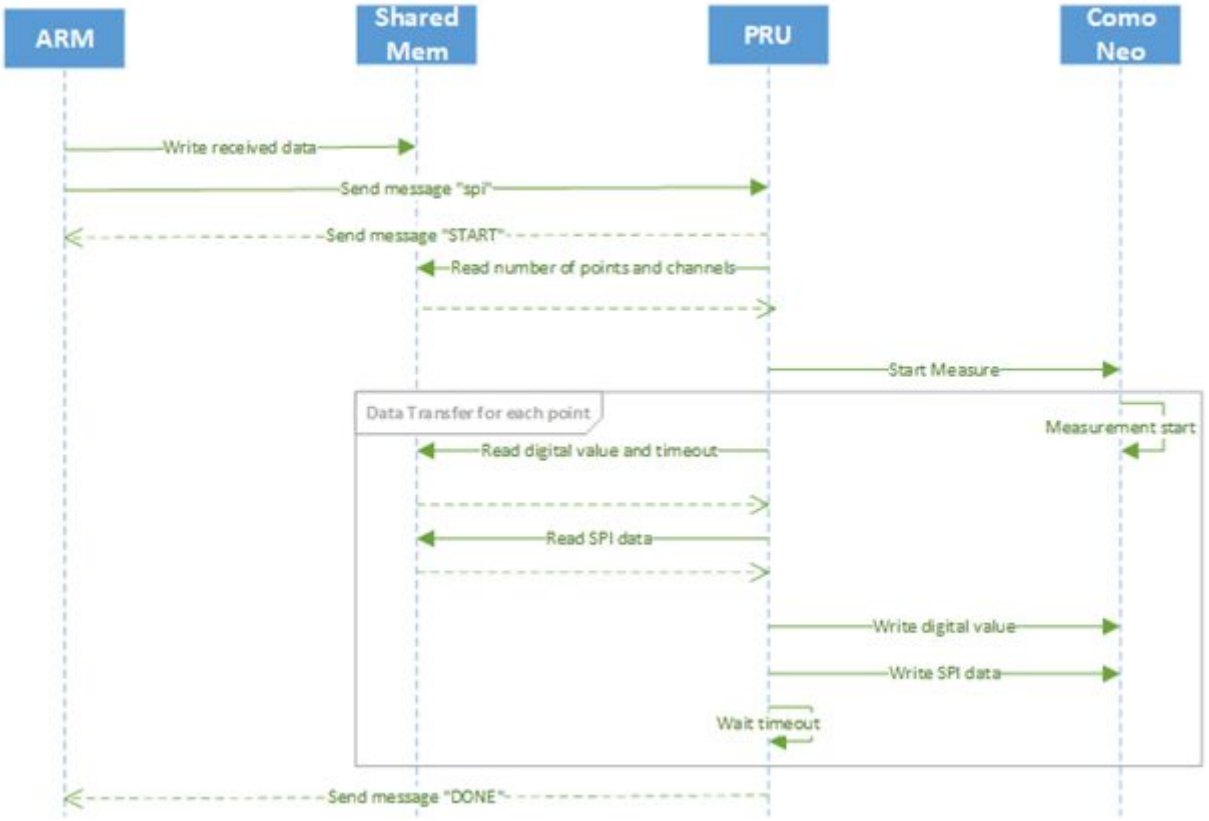
Digitálne hodnoty sú zapisované pomocou funkcie „GPIOWriteData“, kde vstupom 16 bitová hodnota. Spodných 12 bitov predstavuje zapnutý alebo vypnutý konkrétny pin. Táto hodnota je funkcii pretransformovaná na dve nové hodnoty pomocou bitových operácií. Bity týchto nových dvoch hodnôt zodpovedajú usporiadaniu pinov dosky BB a teda je možné vďaka nim zapnúť alebo vypnúť požadované piny pomocou funkcií, poskytovaných knižnicou „hw_gpio_v2.h“.

Čakanie programu presne stanovený čas

V programe bolo potrebné zabezpečiť čo najpresnejšie časovanie nastavovaných hodnôt. Zariadenie ComoNeo má vzorkovaciu frekvenciu 62,5 us, čo znamená že vyčítava hodnoty na vstupoch každých 62,5 us. Najmenším časovým úsekom teda bolo 62,5 us. Časové úseky medzi jednotlivými bodmi na krivkách boli násobkami hodnoty 62,5 us. Z

tohto dôvodu bola implementovaná funkcia „Delay_62us“, ktorá pozastavila vykonávanie programu na požadovaný počet 62,5 us intervalov. Funkcia využívala knižničnú funkciu „__delay_cycles“, ktorej argumentom bol staticky zadaný počet cyklov. Pre využitie tejto funkcie na pozastavenie vykonávania programu na 62,5 us bolo potrebné vypočítať množstvo cyklov za jednu us. Frekvencia PRU je 200MHz a teda na PRU sa vykoná 200 cyklov za jednu us. Následne je možné vynásobiť túto hodnotu hodnotou 62,5 a získame množstvo cyklov potrebných na pozastavenie programu presne na 62,5 ms. Funkcia „__delay_cycles“ s vypočítaným počtom cyklov je v cykle volaná vo funkcii „Delay_62us“ požadovaný počet krát.

Pre zachovanie čo najpresnejšieho časovania je potrebné vziať do úvahy aj čas potrebný na vyčítanie hodnôt z pamäti a nastavenie analógových a digitálnych hodnôt. Tento čas je rádovo menší ako čas jedného 62,5 us intervalu. Vďaka tomu je možné využiť prvý 62,5 us interval na zapísanie hodnôt a následne čakať pomocou funkcii „Delay_62us“ o jeden cyklus menej. Najmenšia možná hodnota čakania medzi dvoma bodmi krivky je 62,5 us a teda v takomto prípade sa vo funkcii „Delay_62us“ nečaká vôbec. Čas potrebný na čítanie z pamäti a nastavenie hodnôt sa môže meniť a je potrebné tento čas doplniť o čakanie do konca prvého 62,5 us intervalu. Na zabezpečenie tejto podmienky je využitý register IEP, ktorý obsahuje počet vykonaných cyklov na úrovni procesora od posledného resetovania registra. Register je dostupný pomocou knižnice „pru_iep.h“. Register je možné resetovať pred začatím čítania a nastavovania hodnôt. Následne je možné vyčítať počet uplynutých cyklov a tým zistiť koľko cyklov procesora bolo spotrebovaných z intervalu 62,5us a koľko je ešte potrebné čakať.



Obrázok 13: Sekvenčný diagram vykonávania PRU programu pre analógové a digitálne signály

3.3.4. Implementácia REST API na BBB

Vo všeobecnosti ide o program napísaný v jazyku Python, ktorý pomocou knižnice Flask a knižnice JSON, umožňuje prijímanie a zapisovanie dát. Pre funkčnosť tohto rozhrania boli následne vytvorené akceptačné testy využitím Robot frameworku, kde Robot odošle za pomoci rozhrania testovacie dáta, pričom z hľadiska testov je požiadavka aby rozhranie zaslané dáta v nezmenenom formáte vrátilo. Ak rozhranie tieto dáta vráti ako odpoveď, vieme že rozhranie je aktívne.

Nižšie je uvedené funkcia REST API (POST), ktorá je aktívna po spustení implementovaného aplikačného rozhrania, a zabezpečuje prijímanie dát, či už cez parameter alebo taktiež vo formáte JSON.

```
def digital_input():
    if request.method == 'POST':
```

```
        if not request.json:
            input =  request.args.get('digIn')
            print(input)
            return input
        print(request.json)
        json_array = json.loads(request.json)
        for item in json_array:
            print()
            print("sampleIndex: "+item['sampleIndex'])
            print("digitalIn: "+item['digitalIn'])
            arr = []
            arr = item['channel']
            res_arr = []
            for x in arr:
                res_arr.append(x)
            print("Channel: ")
            print(res_arr)

        return json.dumps(request.json)
```

Následne uvádzam finálnu verziu tohto REST API, ktorá umožňuje prijímanie ako digitálnych, tak aj analógových vstupov, pričom táto už aj priamo mapuje prijaté dáta do pamäte PRU z ktorej sú následne čítané a vykonávané. Pre mapovanie pamäťového priestoru sme využili štandardnú funkciu operačného systému pre zápis do pamäte *devmem2*.

```
def digital_input():
   if request.method == 'POST':
      json_array = json.loads(request.json)
      StartAdd = 0x4a312000 #zaciatocna adresa PRU shared memory
      CountChannelsDef = True #zapisanie hlavicky na zaciatku kazdeho POST volania
      i = 0
      for item in json_array:
          delay = 0 #obsahuje casovy deadline pre generovnaie krivky na PRU
```

```
if i != len(json_array)-1: #pokracujeme na vypocitanie delay
        timestamp1 = int(json_array[i+1]['sampleIndex'])
        timestamp2 = int(json_array[i]['sampleIndex'])
        delay = timestamp1 - timestamp2
        i += 1
else:
        delay = 0 #delay pre poslednu krivku
arr = []
arr = item['channel']
if CountChannelsDef: #ak True, zapisanie hlavicky
        bits = split_number(str(len(json_array)))
        os.system('devmem2 '+ hex(StartAdd)+ ' h '+ bits[0])
        StartAdd += 0x00000002
        os.system('devmem2 '+ hex(StartAdd)+ ' h '+ bits[1])
        StartAdd += 0x00000002
        bits = split_number(str(len(arr)))
        os.system('devmem2 '+ hex(StartAdd)+ ' h ' + bits[0])
        StartAdd += 0x00000002
        os.system('devmem2 '+ hex(StartAdd)+ ' h ' + bits[1])
        StartAdd += 0x00000002
        CountChannelsDef = False #nebudeme viac zapisovat hlavicku
 bits = split_number(str(delay))
 os.system('devmem2 '+hex(StartAdd)+' h ' + bits[0])
 StartAdd += 0x00000002
 os.system('devmem2 '+hex(StartAdd)+' h ' + bits[1])
 StartAdd += 0x00000002

 bits = split_number(item['digitalIn'])
 os.system('devmem2 '+hex(StartAdd)+' h ' + bits[0])
 StartAdd += 0x00000002
 os.system('devmem2 '+hex(StartAdd)+' h ' + bits[1])
 StartAdd += 0x00000002

for x in arr:
```

```
            os.system('devmem2 '+hex(StartAdd)+' h '+ str(int(x)))
            StartAdd += 0x00000002

    return json.dumps(request.json)
```

3.3.5. Implementácia testu Robot Framework

Počas samotnej implementácie sme využívali knižnicu SeleniumLibrary. Taktiež sme implementovali vlastné funkcie pomocou jazyka python. Tieto funkcie slúžili hlavne na komunikáciu cez REST API. Najprv sme implementovali menšie časti ako napríklad: dostať sa na testovacie piny na web stránke, odoslanie hodnoty pomocou REST API.

Na konci implementácie po prvom semestri sme nakoniec naprogramovali samotný integračný test, ktorý obsahoval celkovú funkcionalitu. Samotný test sa skladal z troch častí, ktoré vznikli počas návrhu.

Prvá časť - integračný test, ktorý sa nachádza v zložke Tests/DigitalInput.robot

```
*** Settings ***
Library SeleniumLibrary
Resource ../Resources/Resources.robot
*** Test Cases ***
Integration test
    Send dig in 1
    Open testing page
    Go to connector //*[@id="main-content"]/div/div/ul/div[1]/spanxpath://*[@id="socketX12"]
${value} = Read connector
    //*[@id="main-content"]/div/div/div/div[1]/div/ng-include/div/div[2]/generic-table/div/div/table/tbody/tr[1]/td[3]/
    div
    Test connector ${value} led turned-on
    Sleep 5
    Send dig in 0
${value} = Read connector
    //*[@id="main-content"]/div/div/div/div[1]/div/ng-include/div/div[2]/generic-table/div/div/table/tbody/tr[1]/td[3]/
    div
    Test connector ${value} led turned-off
    Sleep 5
    Close testing page
```

Test najprv odošle digitálny vstup hodnoty 1 pomocou REST API na webový server. Následne otvorí testovaciu webovú aplikáciu a prejde na konektor, ktorý má byť testovaný. Prečíta jeho hodnotu a skontroluje, či sa rovná hodnote, ktorá bolo odoslaná. Toto isté sa následne opakuje, len odoslaná hodnota je 0 a na konci sa ešte zatvorí webový prehliadač. Druhá časť - podrobnejšie kroky integračného testu, ktoré sa nachádzajú v zložke Resources/Resources.robot

```
*** Settings ***
Library SeleniumLibrary
Library ../ExternalKeywords/ExternalKeywords.py
*** Variables ***
${Browser} chrome ${URL_COMO} http://192.168.197.151/app/home
${URL_REST} http://192.168.197.153/digital_input

*** Keywords ***
Open testing page
    Open Browser ${URL_COMO} ${Browser}
    Maximize Browser Window
    Wait Until Page Contains Element //*[@id="main-view"]/nav/div[6]/a/div
    Click Element //*[@id="main-view"]/nav/div[6]/a/div
    Wait Until Page Contains Element //*[@id="main-view"]/nav/div[6]/div/a[2]/div
    Click Element //*[@id="main-view"]/nav/div[6]/div/a[2]/div

Close testing page
    Close Browser

Go to connector
    [Arguments] ${menu} ${connector}
    Wait Until Page Contains Element ${menu}
    Click Element ${menu}
    Wait Until Page Contains Element ${connector}
    Click Element ${connector}

Read connector
    [Arguments] ${connector}
    ${status} = Get Element Attribute ${connector} class
    [return] ${status}

Test connector
    [Arguments] ${status} ${value}
    Should Be Equal As Strings ${value} ${status}
```

```
Send dig in
[Arguments] ${input}
send_digital_input ${URL_REST} ${input}
```

Táto zložka obsahuje Keywords, ktoré sú volané zo samotného testu. Obsahuje tri globálne premenné, ktoré sa volajú pri vykonávaní niektorých Keywords.

Tretia časť - funcie, ktoré boli implementované v jazyku python ExternalKeywords/ ExternalKeywords.py

```
import requests
import json
def send_digital_input(url, input):
    payload = {'digIn': input}
    requests.post(url, params=payload)
```

Tieto funkcie slúžia na komunikáciu s využitím REST API. Sú volané v samotnom teste.

3.3.6. Implementácia finálneho testu Robot Framework

Test mal niekoľko krokov, ktoré sa vykonali aby otestovali odoslané dáta na BeagleBone Black pomocou REST-API, ktoré boli následne zobrazené v aplikácii ComoNeo. Test je implementovaný nasledovne:

```
Test cycle
    Open browser and maximize
    Login as administrator
    ${channels} =  Get number channels
    Create mould  Testing_mould  ${channels}  #arguments name channels
    Set start MeasuringStart
    Send values
    Go to analysis cycle page
    Check sended values  ${channels}  #argument channels
    Close running browser
```

Test ako prvé otvorí webový prehliadač kde prejde na stránku aplikácie ComoNeo a maximalizuje okno. Prihlási sa v aplikácii ako administrátor, pretože bude potrebné vykonanie krokov, ktoré vyžadujú admin práva. Následne si zistí, koľko je vo vloženom csv súbore kanálov (kriviek), aby na základe toho pripravil mold. Test potom odosiela dáta zo samotného súboru vo formáte json cez REST-API na BeagleBone Black. Keď sú dáta odoslané prejde na stránku v aplikácii, kde budú krivky zobrazené a skontroluje zobrazené hodnoty.

V Pyhton bola naprogramovaná funkcia, ktorá vedela odoslať dáta zo súboru vo formáte json. Jej implementácia bola nasledovná:

```
def send_csv_data(url):
    dataAll = {}
    file = open("C:/Users/win7/Desktop/iotester/RobotFramework/Testiot/Data/Input.csv", "r")
    array = []
    for line in file:
        line = line.split(";")
        data = {}
        channel = []
        data['sampleIndex'] = line[0]
        data['digitalIn'] = line[1]
        data['channel'] = channel
        i = 2  # jump to Channel values
        while i < len(line): # make array from input data
            channel.append(line[i].rstrip("\n\r"))
            i += 1
            data['channel'] = channel
        array.append(data)
    dataAll = array
    output = json.dumps(dataAll)
    print(output)
    #POST
    headers = {'content-type': 'application/json'}
    requests.post(url, data=json.dumps(output), headers=headers)
    file.close()
```

Funkcia otvorila súbor, každý riadok si rozdelila na základe znaku (;) a naplnila json samotnými dátami. Po prečítaní všetkých dát dáta odoslalo pomocou GET metódy na samotný BeagleBone Black. Nakoniec už iba zatvorila súbor.

Toto boli hlavné časti, čo sa týka Robot Framework. Ďalšie Pyhton funkcie slúžili ako pomocné na dostávanie hodnôt zo súboru pri kontrolovaní dát zobrazených v aplikácii ComoNeo a podobne.

3.3.7. Nastavenie pinov na správny mód

Aby bolo možné zapínať na pinoch logickú 0 a 1 je potrebné tieto piny nastaviť do správnych módov.

Pin	Offset	Mód
P8_25 GPIO1_0	0x0	7
P8_24 GPIO1_1	0x4	7
P8_05 GPIO1_2	0x8	7
P8_06 GPIO1_3	0xC	7
P8_23 GPIO1_4	0x10	7
P8_22 GPIO1_5	0x14	7
P8_03 GPIO1_6	0x18	7
P8_04 GPIO1_7	0x1C	7
P9_31 SPI1_SCLK	0x190	3
P9_29 SPI1_D0	0x194	3
P9_30 SPI1_D1	0x198	3
P9_28 SPI1_CS0	0x19C	3

Ako tieto piny nastaviť v device tree súbore je v príručkách.

3.4. Testovanie

3.4.1. Testovanie webového servera na BeagleBone Black

Implementované aplikačné rozhranie bolo spočiatku testované využitím nástroja *Postman,* pričom na zariadenie Beagle Bone Black, boli zaslané základné testovacie vstupy (0/1) pomocou parametrov alebo zasielaním jednoduchého JSON dokumentu :

{"digIn": '1', "digIn2": '10'})

pričom test bol úspešný, ak aplikačné rozhranie vypísalo zasielané hodnoty na obrazovku a taktiež odoslalo spätnú odpoveď (response) v presnom tvare požiadavky (request).

Neskôr boli pre túto úlohu vytvorené samostatné akceptačné testy v prostredí *Robot Framework*, ktorých funkcionalita bola totožná z vyššie uvedenou funkcionalitou testov pomocou nástroja *Postman* ale s tým rozdielom, že tieto testy vykonávali toto testovanie automatizovanie a taktiež ho aj automaticky vyhodnocovali. Vstupom testu boli príslušné dáta a výstupom testu bola hodnota (true - úspešný / false - neúspešný).

3.4.2. Testovanie programu pre komunikáciu PRU a CPU

Predpokladáme, že program na prijímanie a preposielanie správ už je skompilovaný, nahraný na BeagleBone Black a je aj v PRU. Testovací scenár je zadanie 5 rôznych premenných od používateľa a následné overenie spätnej komunikácie.

Zadanie testovacích vstupov:

- **echo "iot" > /dev/rpmsg_pru30**
- **echo "Tester" > /dev/rpmsg_pru30**
- **echo "tim15" > /dev/rpmsg_pru30**
- **echo "tp" > /dev/rpmsg_pru30**
- **echo "abcdef" > /dev/rpmsg_pru30**

Overenie odpovede:

- **cat /dev/rpmsg_pru30**

```
/$ cat /dev/rpmsg_pru30
iot
Tester
tim15
tp
abcdef
```

3.4.3. Testovanie programu pre spínanie digitálnych pinov z PRU

Program pre spínanie digitálnych pinov z PRU bol otestovaný. Najskôr bolo potrebné tento program nahrať do PRU a následne spustiť. Nahrávanie a spúšťanie programu je priblížené v príručke uvedenej nižšie. Po spustení programu boli pomocou logickej sondy kontrolované digitálne piny, ktoré mali byť ovládané programom. Postupne boli odoslané z CPU na PRU správy obsahujúce “1” alebo “0”.

Posielanie správ:

- **echo "1" > /dev/rpmsg_pru30**
- **echo "0" > /dev/rpmsg_pru30**
- **echo "1" > /dev/rpmsg_pru30**
- **echo "0" > /dev/rpmsg_pru30**
- **echo "test" > /dev/rpmsg_pru30**

Overenie výsledku:

Logickou sondou boli overené logické hodnoty na požadovaných pinoch. Piny reagovali na správy korektne. V prípade správy “test” nenastala žiadna zmena hodnôt na digitálnych pinoch, čo bol očakávaný výstup. Taktiež boli logickou sondou overené aj iné digitálne piny, pre zaistenie že program mení hodnoty len požadovaných pinov. Po otestovaní bol program prehlásený za funkčný.

3.4.4. Testovanie systému ako celku pre digitálny signál

Predpokladáme, že BBB aj ComoNeo sú pripojené a všetky technické parametre (IP adresy, ...) nastavené. Na BeagleBone Black je spustený program obsahujúci webový server a ovládanie pinov na doske prostredníctvom PRU. Následne je možné spustiť simulačný test v robot frameworku, ktorý najprv odošle na dosku BeagleBone Black cez rozhranie REST API požiadavku na nastavenie digitálnych pinov na hodnotu logickej jednotky. Potom test otvorí prehliadač s IP adresou zariadenia ComoNeo, kde je možné vidieť nastavenú hodnotu. Túto hodnotu test overí. Po overení test odošle REST API požiadavku na nastavenie digitálnych pinov na hodnotu logickej nuly. Test následne vyhodnotí aj tento scenár. Po úspešnej validácii sa prehliadač zavrie a testovanie je vyhodnotené ako úspešné.

3.4.5. Testovanie finálneho programu pre nastavenie analógových a digitálnych výstupov z PRU

Pre testovanie programu pre nastavenie analógových a digitálnych výstupov z PRU bolo potrebné naplniť spoločnú pamäť hodnotami v správnom tvare a poradí. Na naplnenie pamäti bol využitý pregram devmem2. Bol zostrojený skript, ktorý na požadované adresy v pamäti uložil hodnoty, ktoré zodpovedali hodnotám kriviek. Po vykonaní tohto skriptu bola pamäť naplnená dátami a bola odoslaná správa na PRU v tvare “spi”.

- **echo "spi" > /dev/rpmsg_pru30**

Po prijatí tejto správy sa začalo vykonávanie programu na PRU. Okamžite PRU odpovedala správou “START”. Po uplynutí pár sekúnd PRU odpovedala správou “DONE”. Výsledky boli zaznamenané na grafickom rozhraní zariadenia ComoNeo v prehliadači. V prehliadači bolo možné vidieť všetky krivky uložené v pamati aj so správnymi časovými oneskoreniami.

3.4.6. Testovanie prototypu IoTester

Pri testovaní prototypu IoTester bol zostavený komplet celý systém. Test v Robot Framework otvoril automaticky prehliadač zariadenia ComoNeo a následne odoslal špecifické hodnoty signálov pre zobrazenie požadovaných kriviek. Tieto hodnoty boli prijate

serverom bežiacim na doske BBB. Po prijatí týchto dát boli dáta zapísané do pamäte odkiaľ boli po odoslaní správy "spi" na PRU čítané programom v PRU. Program najskôr nastavil špecifický digitálny signál a tým spustil meranie na zariadení ComoNeo. Toto spustenie merania preveroval test Robot Framework. Program následne generoval digitálne a analógové signály podľa uložených hodnôt a tie sa zobrazovali na grafickom rozhraní zariadenia ComoNeo. Tieto signály vytvorili požadované krivky, ktorých správnosť overil test. Po úspešnom zobrazení kriviek sme mohli pozorovať správy prijaté od PRU. Boli to správy "START" a "DONE", ktoré signalizovali úspešné generovanie všetkých bodov kriviek. Taktiež test Robot Framework bol označený ako úspešný. Krivky sme pozorovali aj osobne na grafickom rozhraní zariadenia ComoNeo.

3.4.7. Testovanie Robot Framework

Testovanie časti Robot Framework prebiehalo kontinuálnym implementovaním nových častí a ich následné testovanie. Každá nová funkcionalita bola hneď otestovaná spustením testu. Takto sa testovalo počas celého vývoju testov.

4. Príručky

4.1. Inštalácia flask balíka na BBB

Linux bežiaci na doske BeagleBone Black, je zostavený pomocou projektu Yocto. Tento Linux teda neobsahuje balíčky klasického operačného systému Linux. Medzi tieto balíčky patrí aj Python server Flask. Medzi tieto balíčky patrí aj Python server Flask. Pre inštaláciu balíčku Flask bolo potrebné tento balíček zostaviť a vložiť ho na dosku BeagleBone Black.

Prvý krok je pripravenie prostredia, na ktorom budeme balíček vytvárať. Pri vytváraní sú potrebné viaceré balíčky. Inštaláciu všetkých potrebných balíčkov spustíme príkazom:

- **sudo apt-get install git build-essential python diffstat texinfo gawk chrpath dos2unix wget unzip socat doxygen libc6:i386 libncurses5:i386 libstdc++6:i386 libz1:i386**

Dôležité je aj nastaviť bash ako predvolený shell:

- **sudo dpkg-reconfigure dash**

Ďalším krokom je stiahnutie a inštalácia Linaro Toolchain, ktorý obsahuje veľa knižníc a program ako napr. GCC. Následne je potrebné vytvoriť sadu vývojových nástrojov (anglicky software development kit, SDK). Kroky potrebné na vytvorenie SDK sa nachádzajú v kapitole Príručky 5.2 Vytvorenie SDK. Vytvorením SDK sa súčasne vytvorili aj všetky potrebné balíčky pre operačný systém Linux. Tieto balíčky boli prekopírované na SD kartu BeagleBone Black.

Následne bolo možné balíček Python-Flask na doske nainštalovať príkazom:

- **opkg install python-flask.ipk**

Inštalácia balíčka Flask však obsahovala závislosti na iné balíčky. Konkrétne to boli balíčky:

- Werkzeug - poskytuje Python rozhranie medzi aplikáciami a servermi.
- Jinja - poskytuje vzor pre zobrazovanie stránok, ktoré aplikácie ponúkajú.
- MarkupSafe - bráni v útokoch injekcie (injection attack)
- ItsDangerous - bezpečne podpisuje dáta pre zaistenie integrity dát.
- Click - framework pre písanie aplikácií príkazového riadku.

Všetky tieto balíčky boli vytvorené pri vytváraní SDK a preto boli taktiež prekopírované na SD kartu dosky BeagleBone Black.Všetky potrebné balíčky boli postupne nainštalované. Po nainštalovaní závislostí bolo možné nainštalovať balíček Flask a vďaka tomu vytvoriť jednoduchý JSON server, ktorý počúva na prichádzajúce JSON požiadavky na vybranom porte. V našom prípade to bol port 5000.

4.2. Vytvorenie SDK

Postupnosť príkazov, ktoré je potrebné zadať, aby bolo SDK správné vytvorený:

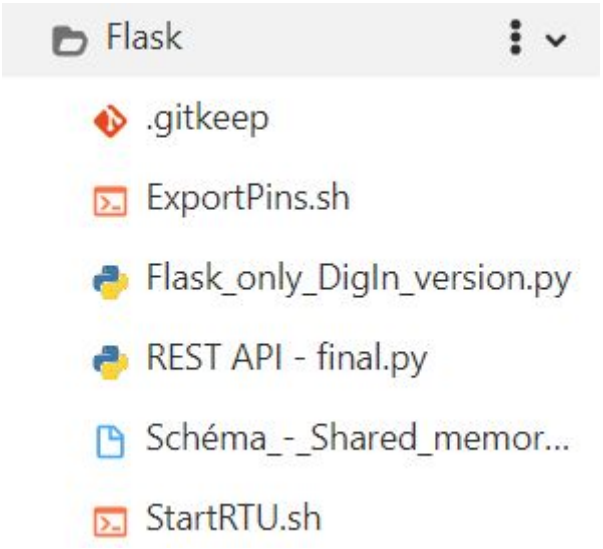
- **git clone git://arago-project.org/git/projects/oe-layersetup.git tisdk**
- **cd tisdk**

- **./oe-layertool-setup.sh -f configs/processor-sdk/processor-sdk-04.02.00.09-config.txt**
- **cd build.**
- **conf/setenv**
- **export PATH=$HOME/gcc-linaro-6.2.1-2016.11-x86_64_arm-linux-gnueabihf/bin:$PATH**
- **MACHINE=am355x-evm bitbake python-flask**

4.3. Spustenie webového servera

Samotné aplikačné rozhranie je nutné aktuálne spúšťať ručne pomocou priameho pripojenia sa na zariadenie BeagleBone Black. Pre jeho spustenie je nutné, aby na zariadení bola nainštalovaná (knižnica) Python a taktiež knižničná funkcia Flask, ktorá je podporovaná práve programovacím jazykom Python. Následne už stačí len umiestniť kód tohto rozhrania na samotné zariadenie BeagleBone Black a to ako štandardný súbor *xxx.py*. Následne už stačí toto rozhranie len pustiť pomocou príkazu *python xxx.py*. Štandardne je aplikačnému rozhraniu pridelená domovská IP adresa zariadenia BeagleBone Black s portom 5000. IP adresa BeagleBone Black je zadaná staticky alebo získaná DHCP serverom.

Aktuálna verzia webového servera, ktorá sa nachádza na linke https://git.kistler.com/-/ide/project/FIIT/iotester/tree/master/-/RTU/Flask/ obsahuje tieto súbory:



Dôležité súbory pre spustenie finálnej verzie, teda verzie, ktorá podporuje digitálny a analógový vstup:

- *REST API - final.py* -> Program REST_API, ktorý je potreba spustiť.
- *ExportPins.sh* -> Skript, ktorý vždy prebehne pri spustení webového servera (*REST API - final.py*). Tento skript aktivuje piny, ktoré využíva RTU.
- *StartRTU.sh* -> Skript, ktorý taktiež prebehne pri spustení webového servera (*REST API - final.py*). Tento skript zavedie a spustí kód pre RTU.

Ďalšie súbory:

- *Flask_Only_DigIn_version.py* -> Protná verzia podporujúca len jeden digitálny vstup.
- *Schéma_-_Shared_memory.PNG* -> Predstavuje obrazovú schému mapovania pamäte medzi webovým serverom (ARM časťou) a pamäťou RTU (PRU časťou)

Neskôr bude toto rozhranie spúšťané automaticky pri štarte zariadenia Beagle Bone Black. Pričom toto zariadenie po štarte odošle svoje konfiguračné údaje (IP adresu) na vopred definovaný aplikačný server, ktorý bude slúžiť na správu týchto zariadení.

4.4. Spustenie programu na PRU

Aby bolo možné spustiť program na PRU je ho potrebné najprv skompilovať v programe Code Composer Studio od firmy Texas Instruments a ako cieľovú platformu vybrať

BeagleBone Black. Po úspešnom skompilovaný vznikne súbor s koncovkou out, ktorý je potrebný uložiť na sd kartu v BeagleBone Black.

Nový program sa do PRU nahrá iba keď je PRU stopnutá a na nahratie programu zadáme príkaz:

- **echo 'program.out' > /sys/class/remoteproc/remoteproc1/firmware**

Posledný krok je už len spustenie programu pomocou príkazu:

- **echo 'start' > /sys/class/remoteproc/remoteproc1/state**

Tieto kroky sú zahrnuté a vykonané v programe pre spustenie Python Flask servera pre zjednodušenie spúšťania programu na PRU.

Program je možné zaszaviť príkazom:

- **echo 'stop' > /sys/class/remoteproc/remoteproc1/state**

4.5. Úprava Device Tree

Dekompiláciu súboru devicetree.dtb spustíme príkazom:

- **dtc -I dtb -O dts -f devicetree_file_name.dtb -o devicetree_file_name.dts**

Následne si v dekompilovanom súbore nájdeme požadovaný pin a zmeníme hodnotu na mód, ktorý chceme nastaviť. Chceme nastaviť pinu P8_04 mód 7:

Head_pin	$PINS	ADDR/OFFSET	GPIO NO.	Name	Mode7
P8_01				DGND	
P8_02				DGND	
P8_03	6	0x818/018	38	GPIO1_6	gpio1[6]
P8_04	7	0x81c/01c	39	GPIO1_7	gpio1[7]

```
pinmux_emmc_pins {
        pinctrl-single,pins = <0x80 0x7 0x84 0x7 0x0 0x7 0x4 0x7 0x8 0x7 0xc 0x7 0x10 0x7 0x14 0x7 0x18 0x7 0x1c 0x7>;
        phandle = <0x3c>;
};
```

V device tree sú piny označované podľa offsetu a druhá hodnota je v hexa tvare, v našom prípade mód 7 je 0x7.

Prekompilovanie znova na dtb zabezpečíme príkazom:

- dtc -I dts -O dtb -f devicetree_file_name.dts -o devicetree_file_name.dtb

Po prekompilovaní stačí nahrať vytvorený súbor na sd kartu do správneho priečinka.

4.6. Nahratie nového súboru „Device Tree“

Pre sprístupnenie pinov na doske BeagleBone Black pre PRU je potrebné nahrať na dosku súbor, ktorý definuje okrem iného aj rozloženie pinov dosky a mód každého pinu. Tento súbor sa nazýva „device tree“ a v binárnej forme má koncovku „.dtb“. Súbor je priložený na elektronickom médiu.

Postup nahrania DTB:

- Skopírovanie DTB súboru na SD kartu BeagleBone Black
- Zmena ukazovateľa z pôvodného DTB na nový DTB príkazom:

  sudo ln -f -s am335x-boneblack_PRU.dtb am335x-boneblack.dtb

- Synchronizácia príkazom: **sync**

4.7. Nahratie image na sd kartu

Nemusí to byť zrovna mmcblk0, preto najprv použiť príkaz **fdisk**!!

- **sudo fdisk -l**
- **sudo umount /dev/mmcblk0**
- **sudo dd if=~/sd-card-copy.img of=/dev/mmcblk0**

Operácia kopírovania môže trvať aj **viac ako 30 minút**!!

4.8. Pripojenie na BBB

Musíte držať stlačené tlačidlo S2 (tlačidlo pri SD karte) a vtedy zapojiť do USB, keď začnú blikať ledky, pustiť tlačidlo. Toto iba v prípade ak nie je nastavené default boot z SD karty. Ak je stačí zapojiť napájanie.
V termináli napísať príkaz, nemusí to automaticky byť ttyACM0, preto treba predtým skontrolovať názov portu!!

- **sudo picocom -b 115200 /dev/ttyACM0**

K doske sa dá taktiež pripojiť pomocou SSH príkazom:

- **ssh root@<IP adresa BBB>**

4.9. Build programu

Potrebný software:

1. Linux Processor SDK (http://www.ti.com/tool/PROCESSOR-SDK-AM335X)
2. PRU software balík (https://git.ti.com/pru-software-support-package/pru-software-support-package/trees/master)
3. Code Composer Studio
4. PRU Code Generation Tools (http://software-dl.ti.com/codegen/non-esd/downloads/download.htm#PRU)

Build programu (LAB 6):

5. Zapnúť Code Composer
6. Importovať projekt PRU_RPMsg_LED0 (/labs/lab_6/solution)
7. Zahrnúť závislosti (Project Explorer/Properties/Build/PruCompiler/IncludeOptions
   a. **"../../../../../include"**
   b. **"../../../../../include/am335x"**
8. Pridanie rpmsg_lib (Project Explorer/Properties/Build/PruLinker/FileSearchPath)
   a. **"../../../../../lib/rpmsg_lib.lib"**
9. Build projektu nemôže obsahovať chýbové hlášky
10. Otvoriť terminál a zadať príkazy
    a. **export PRU_CGT=<path_to_Linux_proc_sdk>/linux-devkit/sysroots/x86_64-arago-linux/usr/share/ti/cgt-pru**
    b. **cd <PRU_SW_PATH>/labs/lab_6/solution/PRU_RPMsg_LED0**
    c. **make clean**
    d. **make**

    vytvorí sa program v zložke **gen/PRU_RPMsg_LED0.out**
11. Vytvorený program treba prekopírovať na BBB do zložky **/lib/firmware**
12. Zadáme príkazy (už v BBB termináli)

a. **echo 'PRU_RPMsg_LED0.out' > /sys/class/remoteproc/remoteproc1/firmware**
b. **echo 'start' > /sys/class/remoteproc/remoteproc1/state**

13. Skontrolujeme RPMsg PRU character device
    a. **ls /dev/ | grep pru**
14. Odoslanie správy
    a. **echo "fiit" > /dev/rpmsg_pru30**
15. Zobrazenie správ/y
    a. **cat /dev/rpmsg_pru30**

4.10. Vytvorenie programu pre PRU pomocou CSS

Celý postup je potrebné realizovať na operačnom systéme Linux. PRU pracuje so špecifickými súbormi typu „.out“. Tieto súbory sú upravené tak aby fungovali na procesore reálneho času. Takýto súbor je možný získať kompiláciou programu pre PRU. Tento program bolo donedávna potrebné programovať v jazyku Asembler. Neskôr boli vytvorené knižnice, ktoré umožnili programovať PRU programy v jazyku C. Využili sme jazyk C. Program sme vytvárali pomocou integrovaného vývojového prostredia (IDE) Code Composer Studio (CCS) verzia 8. Ďalej bolo potrebné stiahnuť správne SDK k PRU. PRU pozostáva z procesora typu AM335x a teda bolo potrebné stiahnuť SDK rovnakého typu. Po stiahnutí bola potrebná inštalácia. Toto SDK okrem iného obsahuje knižnice pre prácu s PRU ako aj knižnice pre zasielanie správ RP_MSG. Taktiež obsahuje príklady programov pre PRU. Tento príklad sme importovali do CSS kde bolo možné ho ďalej upraviť. Po upravení programu bolo možné skompilovať program, čím sa vytvoril požadovaný súbor „.out“. Tento súbor je možné nájsť v priečinku Debug, ktorý sa nachádza v rovnakom priečinku ako program samotný. Následne je možné tento súbor kopírovať na dosku BeagleBone Black. Súbory „.out“ je potrebné ukladať do priečinka „rootfs/lib/firmware“. V našom prípade využívame systém Linux, ktorý je uložený na SD karte a BBB sa bootuje priamo z tejto karty. Je teda možné prekopírovať požadovaný súbor priamo na SD kartu pomocou čítačky kariet. Mierne zložitejší spôsob je vytvoriť lokálnu sieť medzi počítačom a BBB. Nami použitý Linux na BBB okrem SSH prístupu v rámci siete štandardne poskytuje jednoduchý

server pre prenos súborov. Vďaka tomu je možné skopírovať súbor pomocou príkazu „scp <názov súboru> root@<IP adresa BBB>“:/lib/firmware“.

Postup vytvorenia programu

1. Inštalácia Code Composer Studio (CCS).
2. Stiahnutie SDK pre AM335x.
3. Importovanie príkladu z SDK do CSS.
4. Úprava programu.
5. Kompilácia programu.
6. Vyhľadanie .out súboru v priečinku Debug.
7. Kopírovanie .out súboru do priečinka /rootfs/lib/firmware na doske BBB.
    a. Priamo na SD kartu.
    b. Vytvorenie lokálnej siete a použitie príkazu:
    scp <názov súboru> root@<IP BBB>:/lib/firmware

Pre detailnejší postup je vhodné riadiť sa návodom na:

http://processors.wiki.ti.com/index.php/RPMsg_Quick_Start_Guide

http://processors.wiki.ti.com/index.php/PRU_Training:_Hands-on_Labs

4.11. Bootovanie z SD karty

Pustiť cez **TLAČIDLO S2** prvýkrát.

Treba si skontrolovať či je SD karta mmcblk0 alebo mmcblk1.

V prípade, že SD karta je mmcblk0 treba zadať príkaz:

dd if=/dev/zero of=/dev/mmcblk1 bs=1M count=1

Po reboot BBB by už mal nabehnúť systém z SD karty bez nutnosti držania tlačidla.

4.12. Púšťanie testov Robot Framework

Pre spustenie testov je potrebné nainštalovať a nastaviť nasledovné časti:

- Inštalácia Pyhton (https://www.python.org/downloads)

- Pridanie cesty do systémovej premennej Path (..\PythonXX;..\PythonXX\Scripts)
- Inštalácia robotframework pomocou cmd (pip install robotframework)
- Inštalácia SeleniumLibrary pomocou cmd (pip install robotframework-seleniumlibrary)
- Stiahnuttie chromedriver (http://chromedriver.chromium.org/downloads)
- Vloženie chromedriver do priečinka (..\PythonXX\Scripts)
- Inštalácia PyCharm (https://www.jetbrains.com/pycharm/download)
- Inštalácia pluginu IntelliBot (v nastaveniach PyCharm)
- Inštalácia requests pre python kvôli využitiu REST API (pip install requests)

Po nainštalovaní všetkých závislostí je potrebné nahrať csv súbor do adresára: Data/ a taktiež v adresáre Resources/Resources.robot je potrebné nastaviť IP adresy aplikácie ComoNeo a taktiež REST-API bežiaceho na BeagleBone Black. Po nahratí súboru a nastavení IP adries je potrebné otvoriť terminál, dostať sa do repozitára, kde sa nachádza súbor Tests/DigitalInput.robot a vykonať príkaz: robot DigitalInput.robot. Následne sa spustí samotný test a vykoná sa.

4.13. Používateľská príručka

V nasledujúcej kapitole sa nachádza príručka určená pre používateľov nášho prototypu, ktorá obsahuje jednotlivé kroky potrebné pre jeho správne spustenie a fungovanie.

Finálny produkt nášho tímového projektu sa skladá zo štyroch hlavných častí:

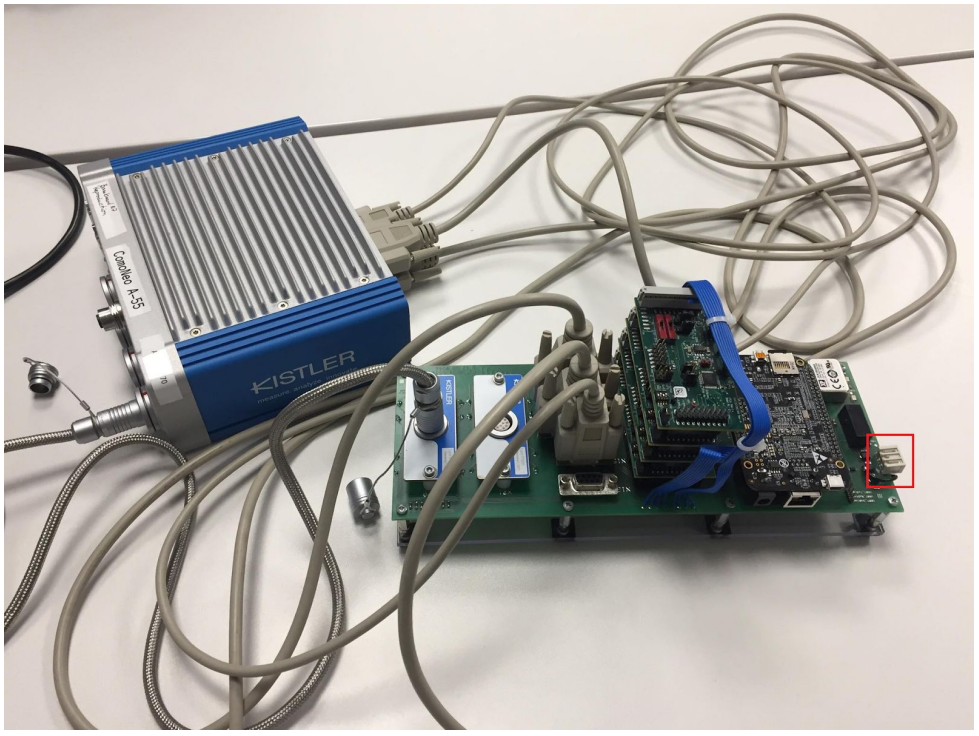
1. Hardvérová doska dodaná firmou Kistler
2. Vývojová doska Beaglebone Black
3. Robot Framework
4. Zariadenie ComoNeo

Prvým krokom pri spustení nášho prototypu je pripojenie vývojovej dosky Beaglebone Black ku hardvérovej doske dodanej firmou Kistler. Pri zapájaní dosky Beaglebone Black ku hardvérovej doske je potrebné dbať na usporiadanie pinov a orientáciu

dosky. Najspoľahlivejší spôsob je riadiť sa podľa šiestich pinov, ktoré sa nachádza pri hlavičke pinov P9, keďže sa tieto piny nachádzajú iba na jednej strane vývojovej dosky a na hardvérovej doske sa k nim taktiež nachádzajú príslušné piny iba na jednej strane.

Po úspešnom zapojení vývojovej dosky ku hardvérovej doske je potrebné zapojiť jednotlivé analógové a digitálne konektory ku zariadeniu ComoNeo. Prepojenie jednotlivých konektorov je veľmi intuitívne a to najmä vďaka tomu, že sa na oboch zariadeniach, ako na hardvérovej doske tak aj na zariadení ComoNeo, nachádzajú čísla jednotlivých konektorov X11, X12, X14, X15 a X24.

Pre zapnutie a aktiváciu hardvérovej dosky dodanej firmou Kistler je potrebné pripojiť ju ku zdroju napätia pomocou príslušných káblov ku konektoru, ktorý je vyznačený červeným štvrocom na nasledujúcom obrázku. Rovnaký je možné nájsť aj na zariadení ComoNeo, pričom je jeho zapnutie je taktiež potrebné tento konektor zapojit ku zdroju napätia.



Obrázok 14: Finálny produkt

Pri súčasnej verzii prototypu je následne potrebné pripojiť vývojovú dosku BeagleBone Black ku počítaču a pripojiť sa ku nej pomocou konzolových príkazov uvedených v kapitole 5.8 Pripojenie na BBB. Následne je potrebné spustenie skriptu s

názvom *REST API - final.py*, ktorý spustí všetky potrebné programy na vývojovej doske BeagleBone black a to konkrétne program bežiaci na PRU a program na zachytávanie a prácu s REST volaním odoslaným z Robot Framework.

Predposledným krokom pri práci s prototypom je pripojenie všetkých zariadení do rovnakej siete, pričom zariadenie ComoNeo má statickú IP adresu 192.168.197.151.

Posledným krokom pri práci s prototypom je spustenie testov pomocou Robot Framework podľa návodu v kapitole 5.12. Spúšťanie testov Robot Framework.

4.14. Inštalačná príručka

Všetky kroky potrebné na inštalovanie jednotlivých balíkov prípadne programov sú uvedené už v jednotlivých príručkách. Ak používateľ chce najaktuálnejšiu verziu súborov na BeagleBone Black, najjednoduchším krokom je nahratie image súboru na sd kartu. Postup je uvedený v sekcii 4.7 Nahratie image na sd kartu.