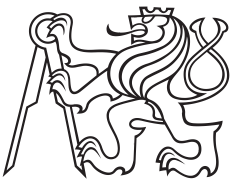
Diplomová práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra řídicí techniky

Diagnostické zařízení pro automatické testování automobilových řídicích jednotek

Jakub Janata
Otevřená informatika – Počítačové inženýrství

Květen 2016
http://www.fel.cvut.cz/
Vedoucí práce: Ing. Michal Sojka Ph.D.

Poděkování / Prohlášení

Chtěl bych poděkovat Ing. Michalu Sojkovi Ph.D. za konstruktivní vedení práce a za jeho neocenitelné rady, které mi v průběhu poskytoval. Dále pak Bc. Joelu Matějkovi za podnětné náměty k návrhu desky plošných spojů. A v neposlední řadě pak rodině, za její dlouholetou podporu během studia.

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 18. 5. 2016

.....

Abstrakt / Abstract

Cílem této práce je navrhout a implementovat hardwarovou a softwarovou podporu pro testování automobilové řídicí jednotky vyvíjené v projektu Rapid Prototyping Platform (RPP).

V práci je popsán návrh schématu zapojení elektronických obvodů a desky plošných spojů, která rozšiřuje vývojovou deskou BeagleBone Black. Dále je zdokumentován softwarový framework RPP-Tester (sloužící pro testování) a postup jeho instalace v kombinaci s nastavením linuxové distribuce Debian. Navrhnutá deska je připojená ke vstupům, výstupům a komunikačním portům RPP a testování probíhá na úrovni těchto rozhraním. Ve výsledné formě je testovací platforma schopná po integraci s verzovacím systémem provádět automatizované regresní testy, případně může sloužit jako nástroj pro agilní vývoj.

Součástí dokumentu jsou i testovací scénáře a výsledky z testování na fyzických deskách RPP.

Klíčová slova: BeagleBone Black, Sitara AM335x, Linux, Python, eCAP

The goal of this work is to design and implement hardware and software support for testing of automotive ECU developed in the Rapid Prototyping Platform (RPP) project.

The thesis describes the design of electronic circuits and printed circuit board which extends the BeagleBone Black board. The RPP-Tester, that performs automated testing, framework as well as the procedure of its installation combined with settings of Debian Linux distribution are described too. The platform has been integrated with a version control system, the testing platform in its final form is able to perform automated regression tests. It may also serve as an instrument for agile development.

Testing scenarios and results of testing on RPP physical boards also constitute a part of the document.

Keywords: BeagleBone Black, Sitara AM335x, Linux, Python, eCAP

Title translation: Diagnostic unit for automated testing of automotive ECU

Obsah /

Tabulky / Obrázky

Kapitola 1
Úvod

1.1 Zadání a motivace

Zadáním mé diplomové práce bylo vytvoření testovací jednotky, která má za úkol ověřovat softwarovou funkčnost hardwaru automobilové řídicí jednotky Rapid Prototyping Platform (RPP), vyvinuté pro Porsche Engineering Services (obr. 1.1). Původní software projektu RPP byl po dokončení hlavního vývoje portován i na další hardware, např. pro firmu Eaton. Jelikož se při úpravách softwaru pro platformu výše zmíněné společnosti často stávalo, že se do funkcionality desky RPP zanesly chyby, vznikl požadavek na možnost přidat ke stávajícím softwarovým testům i ověření funkčnosti na reálném hardwaru. K tomuto účelu jsem navrhl testovací desku postavenou na vývojovém produktu BeagleBone Black, na kterém je spuštěna linuxová distribuce Debian a nasazen mnou vytvořený pythonovský framework RPP-Tester. Tento nástroj ve spojení s testy, které jsem vytvořil a otestoval, zpracovává mnou zadané testovací scénáře a výsledky provedených měření reportuje v čitelné podobě zpět vývojářům. Součástí řešení je i podpora automatického nahrávání zkompilovaných binárních souborů do FLASH paměti RPP v kombinaci s automatizovaným spouštěním aplikace pomocí nástroje pro průběžnou integraci.

Ačkoliv byly všechny RPP jednotky po výrobě podrobeny důkladným testům a fungovaly, tester vyvinutý v této práci odhalil, že po třech letech jejich intenzivního používání se na některých z nich vyskytují i fatální chyby.

1.2 Struktura práce

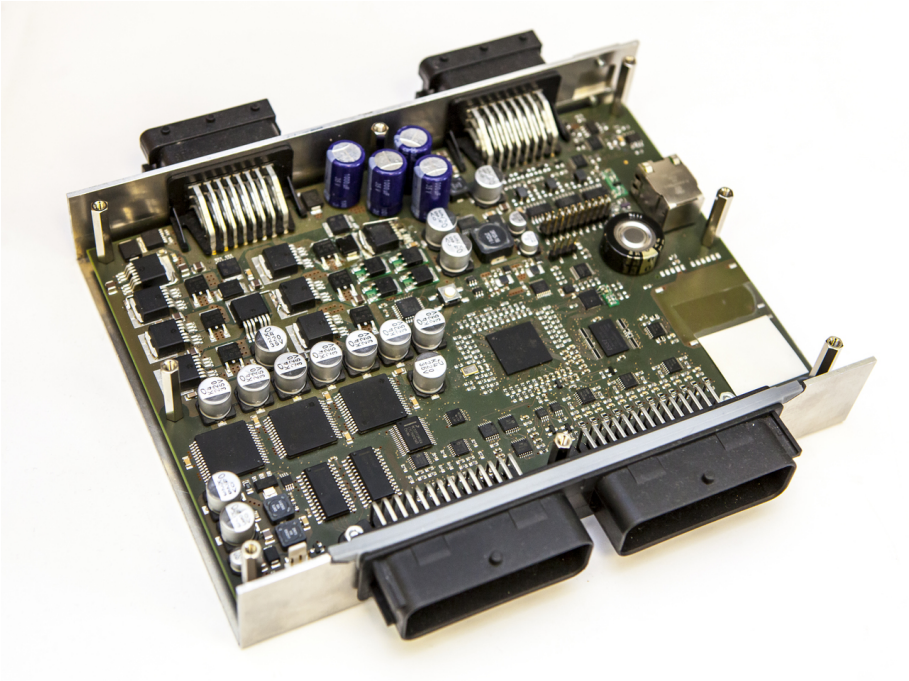
Text této práce má následující strukturu:

- **Textová zpráva**

  Kapitola 2 *Související technologie* obsahuje popis použitých technologií a postupů. V kapitole 3 *Hardware pro testování* je popsán návrh schématu a desky plošných spojů. Jsou zde vysvětleny funkce jednotlivých periferií a jejich případná omezení. Kapitola 4 *Softwarová podpora pro testování* vysvětluje problematiku použití frameworku RPP-Tester a nutné kroky k nastavení linuxového systému. V kapitole 5 *Funkční testy periferií* jsou uvedeny jednotlivé testovací scénáře a jejich popis. Kapitola 6 *Výsledky testů* shrnuje výsledky testování na fyzických deskách používaných katedrou pro projekt PES-RPP.

Přiložené médium

Přiložené médium obsahuje elektronickou verzi toho dokumentu včetně příloh. Dále pak soubory pro program KiCad, které obsahují schéma zapojení, produktové listy součástek a jednotlivé vrstvy desky plošných spojů. V neposlední řadě i zdrojové kódy frameworku RPP-Tester a konfigurační skripty pro nastavení systému. Nicméně všechny tyto informace jsou umístěny i v repozitáři na webu katedry na adrese `ssh://git@rtime.felk.cvut.cz:pes-rpp/rpp-tester`.



Obrázek 1.1. Fotografie jednotky RPP.

Kapitola 2
Související technologie

Tato kapitola popisuje existující technologie, které jsem ve své práci využil. Jsou zde základním způsobem popsány témata a pojmy jako Rapid Prototyping Platform, BeagleBone Black, AM335x Sitara, OpenOCD, Kicad a další.

2.1 Rapid Prototyping Platform

Porsche Engineering Services – Rapid Prototyping Platform (PES-RPP), neboli platforma pro agilní vývoj a testování prototypů, je vývojová deska vyvinutá katedrou řídicí techniky na Českém vysokém učení technickém v Praze (obr. 1.1). Je osazena řídicím mikrokontrolerem TMS570LS3137ZWT od firmy Texas Instruments (`http://rtime.felk.cvut.cz/rpp-tms570`). Blokové schéma jednotky RPP je na obr. 2.1 a popis konektorů na obr. 2.2. Vývoj hardwaru již byl dokončen, nicméně práce na softwarové části stále probíhají.

2.1.1 Hardwarová část

RPP deska obsahuje většinu periferií, které se používají v automobilových řídicích jednotkách:

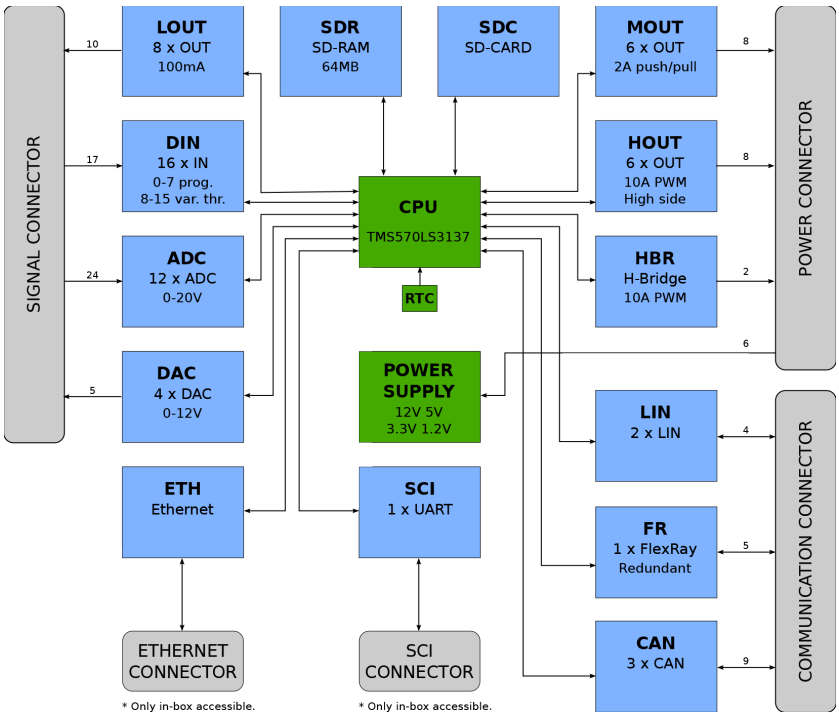
- **Vstupy/výstupy**
  - 6× PWM výstup (10 A)
  - 1× H-můstek (10 A)
  - 6× Digitální výstup (2 A)
  - 8× Digitální výstup (100 mA)
  - 16× Digitální vstup (min. 8× podpora přerušení)
  - 12× ADC (0–20 V)
  - 4× DAC (0–12 V)
- **Komunikační rozhraní**
  - 3× CAN
  - 1× FlexRay (2 kanály)
  - 1× Ethernet
  - 2× LIN

2.1.2 Softwarová část

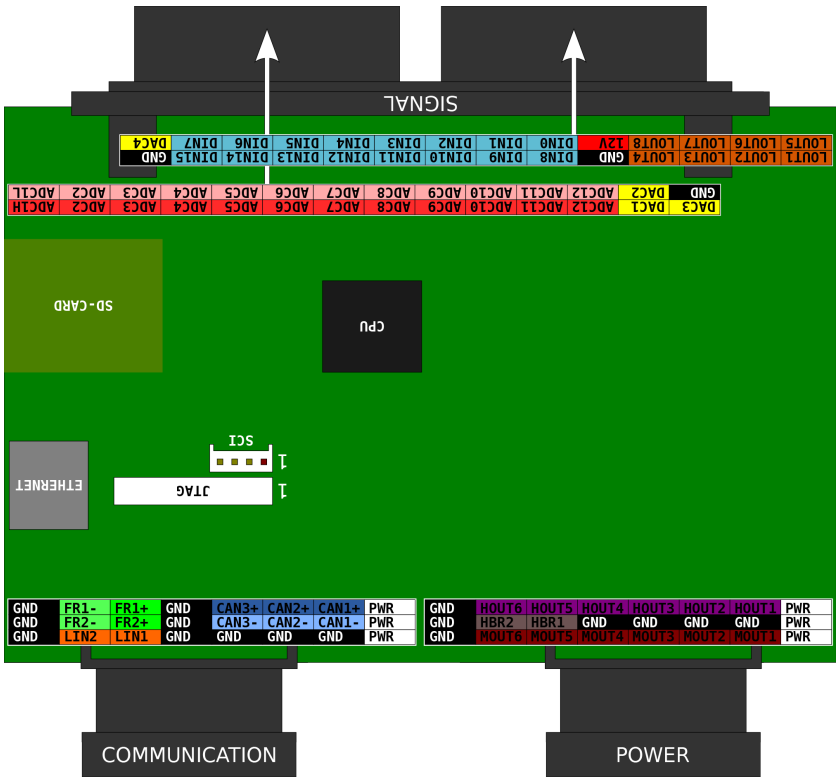
Softwarová část RPP projektu obsahuje hlavní tři softwarové balíčky:

- **rpp-simulink** obsahuje podporu pro automatické generování firmwaru z programů Matlab/Simulink, dema modelů a skripty pro nahrávání firmwaru na desku z Matlabu/Simulinku.
- **rpp-test-sw** obsahuje aplikace pro interaktivní testování a ovládání RPP desky přes sériové rozhraní.

- **rpp-lib** obsahuje zdrojové kódy RPP knihovny napsané v jazyku C. Tato knihovna je využívána jak balíkem rpp-simulink, tak rpp-test-sw.



Obrázek 2.1. Blokové schéma periferií jednotky RPP.



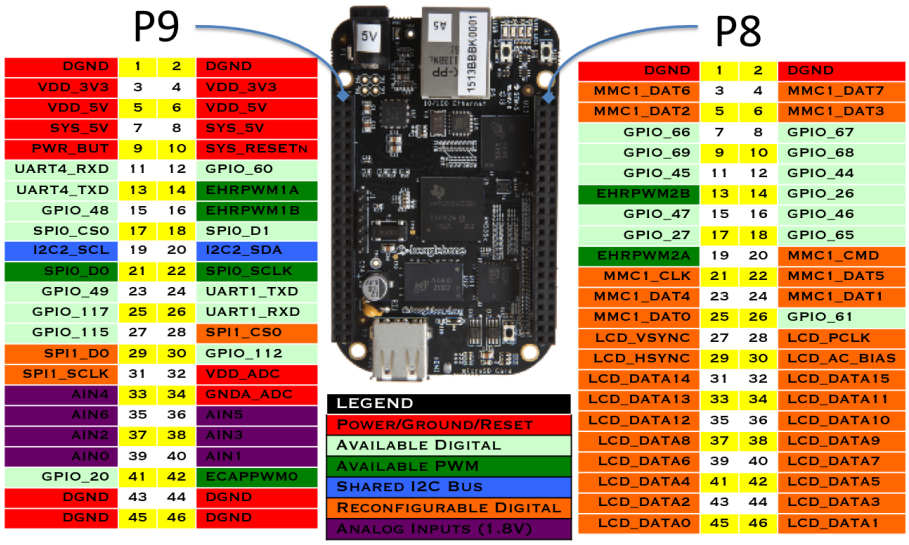
Obrázek 2.2. Popis konektorů RPP.

2.1.3 rpp-test-sw

Aplikace *rpp-test-sw* je součástí RPP softwaru. Její funkcí je umožňovat prostřednictvím sériového rozhraní interakci s deskou. Pomocí zadávání příkazů na konzoli tak může vývojář testovat a nastavovat jednotlivé periferie. Základní nastavení sériové linky je 115200bps, 8bitů, žádná parita a 1 stop bit (použité napětí do 3.3 V). Seznam všech podporovaných příkazů lze vypsat příkazem *help*, který zároveň funguje jako nástroj pro procházení dokumentace.

2.2 BeagleBone Black

BeagleBone Black (BBB) je vývojová deska z řady BeagleBoard [1]. Tato deska tvoří základ mnou navrhovaného testeru. Mezi její přednosti patří nízká cena v poměru k množství IO rozhraní a periferií (obr. 2.3). BBB je řízen výkonným mikrokontrolerem Sitara XAM3359AZCZ100 (série AM335x) s architekturou ARM. Deska je vybavena operační pamětí 256Mb x16 DDR3L 4Gb (512MB) SDRAM, 32KB EEPROM, 4GB MMC (eMMC) Flash pamětí, slotem pro microSD kartu, USB 2.0 porty (host i device), 10/100 Mbps Ethernetovým rozhraním, sériovou konzolí (TTL SCI) a dalšími periferiemi. Mezi hlavní výhody patří široká komunita aktivních vývojářů a s tím spojená podpora softwaru. Na procesor BBB lze nainstalovat operační systém Linux a většinu jeho distribucí. V základní konfiguraci se deska dodává s Debianem 7.9 Wheezy a jádrem 3.8.13 nainstalovaným ve vnitřní eMMC Flash paměti. O zavedení systému se stará zavaděč Das U-Boot [2].



Obrázek 2.3. Vývojová deska BeagleBone Black s přehledem rozšiřujících rozhraní [1].

2.2.1 Das U-Boot

Das U-Boot je zavaděč systému určený především pro vestavěné systémy a operační systém Linux. Je k dispozici nejen pro architekturu ARM. Jedná se o svobodný software naprogramovaný v jazyce C a dostupný pod licencí GNU General Public License. Mezi jeho alternativy patří například GRUB 2. Pro správnou funkčnost testovací jednotky je nutné předat U-Bootu upravený device tree soubor pomocí konfiguračního souboru *uEnv.txt* [3].

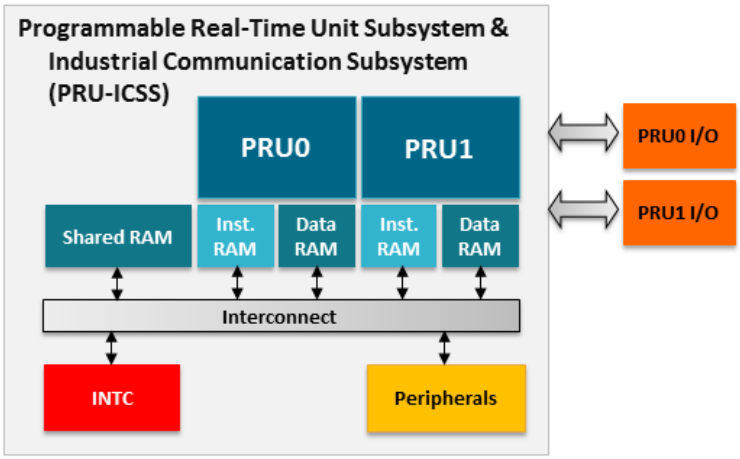
Na procesoru AM335x je samotný proces zavádění systému rozdělen do několika fází. Nejprve vnitřní Boot ROM (read-only) čte konfigurační piny pro boot, které rozhodnou o pořadí externích bootloaderů. Na výběr je možnost zavádět z NAND, UART a SD/MMC karty. Poté je z daného zařízení načten tzv. x-loader (v U-Bootu reprezentován souborem MLO) a ten již inicializuje u-boot.img, který obslouží finální zavedení jádra Linuxu [4–6].

2.3 AM335x Sitara

AM335x Sitara je mikrokontroler navržený firmou Texas Instruments. Je to 32-bitový RISC-ový procesor s jádrem ARM Cortex-A8 (mikroarchitektura ARMv7-A), který může běžet až na frekvenci 1 GHz. Kromě základních GPIO (General-purpose input/output) portů obsahuje i 10/100/1000 Mbit/s EMAC (Ethernet media access control), CAN, USB, UART, PRU-ICSS (Programmable Real-Time Unit Subsystem and Industrial Communication SubSystem) a další periferie.

2.3.1 PRU-ICSS

PRU-ICSS (Programmable Real-Time Unit Subsystem and Industrial Communication SubSystem) je subsystém dvou programovatelných 32-bitových RISC jednotek reálného času (PRU) s datovou, instrukční a sdílenou pamětí (obr. 2.4). Součástí systému je i podpora přerušení a možnost přístupu k dalším periferiím. Jelikož PRU může pracovat téměř nezávisle na hlavním procesoru, nejčastější případy použití jsou aplikace reálného času. Jednotka má i přímý přístup na GPIO porty, což je vlastnost, která je vhodná pro implementaci řadičů sběrnice FlexRay. Tato funkcionalita není součástí této práce, ale je pro ni připravena hardwarová podpora.



Obrázek 2.4. Blokové schéma PRU-ICSS. [7]

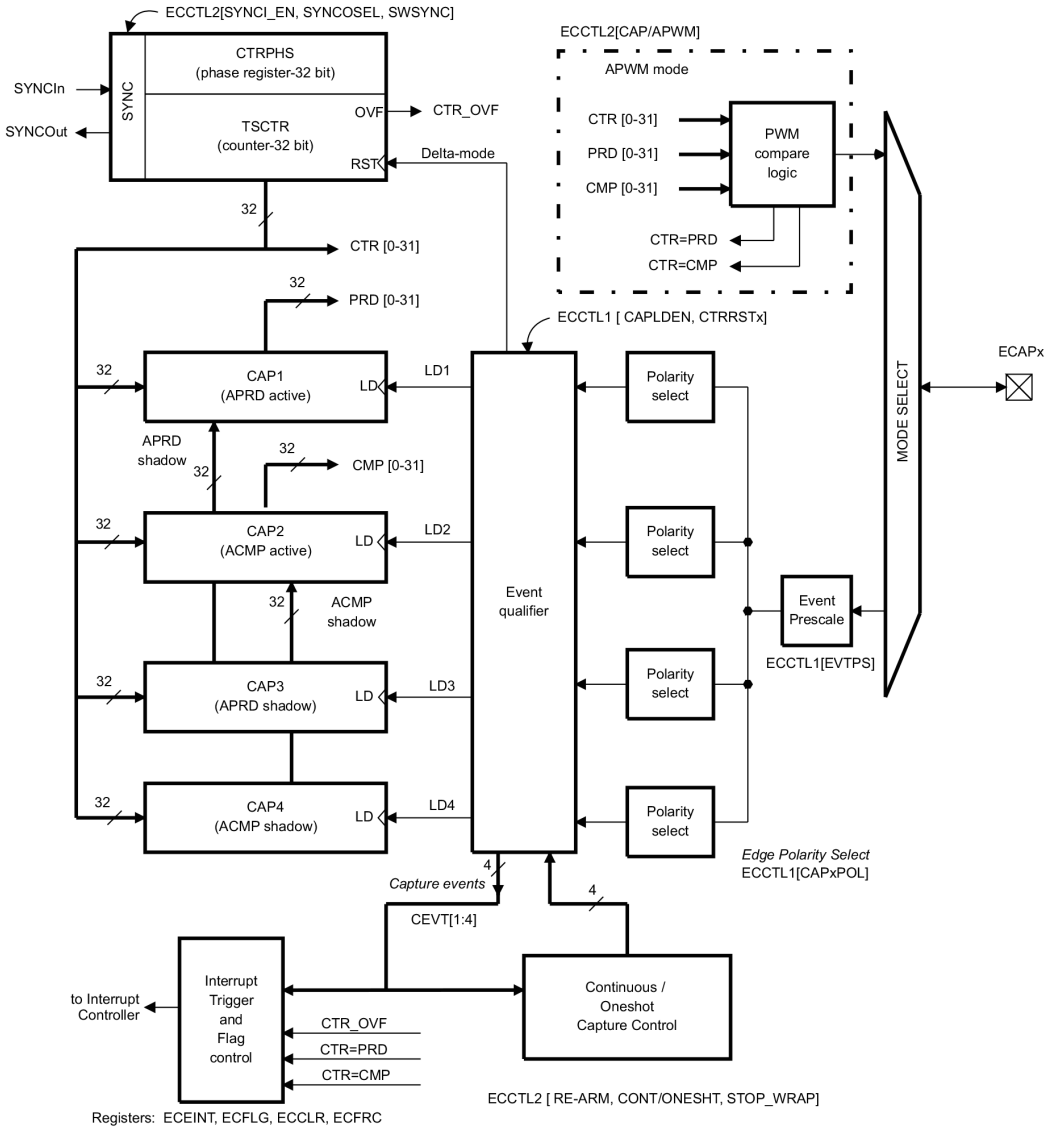
2.3.2 Enhanced capture

Enhanced capture (eCAP) modul je periferie AM335x procesoru, která má za úkol zaznamenávání signálových pulsů (např. měření audio vstupů, PWM nebo rychlosti otáček motorů). K dispozici je čtveřice záchytných 32-bitových registrů, které podle nastavení zaznamenávají aktuální hodnotu čítače (obr. 2.5). Periferii lze využít i pro generování pulsně-šířkové modulace (PWM). AM335x obsahuje 3 tyto submoduly, které na sobě mohou být zcela nezávislé. BBB má ovšem vyvedeny pouze 2 (ECAP0 a ECAP2) [8].

Pro účely testování v mém testeru se využívá eCAP k zaznamenávání výstupních signálů RPP desky z HOUT a HBR periferií.

V subsystému eCAP lze použít několik režimů záznamu podle konfigurace:

- záznam náběžné/sestupné hrany (případně obě varianty)
- jednoběhový nebo kontinuální záznam
- inkrementální (absolutní) nebo *delta* mód čítače



Obrázek 2.5. Blokové schéma eCAP [8].

2.4 OpenOCD

Open On-Chip Debugger (OpenOCD) je vývojový nástroj pro programování a ladění embedded zařízení přes rozhraní JTAG. Projekt vznikl v roce 2005 jako součást diplomové práce Dominika Ratha (University of Applied Sciences Augsburg). (http://www.hs-augsburg.de) a od svého počátku je to open-source s podporou komunity softwarových a hardwarových vývojářů z celého světa. Program podporuje nahrávání do flash NAND a NOR pamětí [9]. K tomu je také využit v rámci této práce s využitím FTDI FT2232 převodníku.

- **FTDI FT2232**
  Future Technology Devices International (FTDI) je rodina produktů, které umožňují použití UART, JTAG a dalších rozhraní přes standardní USB port.

2.5 Device Tree

Device tree (volně přeloženo jako strom zařízení) je stromová datová struktura sloužící k popisu systémového hardwaru. Struktura by neměla sloužit ke konfiguraci, ale pouze popisovat svým specifickým jazykem HW (jednotlivé periferie procesoru a desky). Hlavním důvodem existence tohoto řešení je odstranění nutnosti popisovat HW přímo do zdrojových kódů jádra operačního systému. Principem je vytvářet soubory v hierarchických stromech, které jsou nezávislé na operačním systému a zvýšit možnost jejich opětovného použití. Při startu systému boot loader načte device tree do uživatelské paměti a na základě toho operační systém zpřístupní ukazatele periferií (uvedené v binárce) uživateli.

- **Datová struktura**
  Základ tvoří uzly, které popisují jednotlivá zařízení v systému. Každý uzel má jeden nebo více párů proměnná/hodnota, které slouží ke charakteristice periferie. Podle specifikace by každý uzel měl mít právě jednoho předka s výjimkou root uzlu, který nic takového mít ani nemůže.

- **Hierarchie souborů**
  Soubory je možno do sebe vkládat a případně tím přepisovat a doplňovat informace v uzlech. Pro kompilaci *dts* a *dtsi* souborů se používá program *dtc*. Taktéž lze využít C kompilátor pro využití maker (např. #include a další). Výstupem je *dtb* soubor v binární podobě. Seznam souborů:
  - **.dts** soubory obsahují textový popis na úrovni desky (např. BBB), případně doplňují nastavení pinctrl pro pin-muxing. Rovněž se používají k přepsání souborů nižších vrstev (overlaying).
  - **.dtsi** soubory popisují typicky HW na úrovni SoC (System on Chip). Písmeno *i* značí, že jsou typicky „includovány“ do .dts souborů.
  - **.dtb** je soubor binárního formátu, který vzniká kompilací dts a dtsi zdrojových kódů. Formát dat souboru je podobný jako Flattened Device Tree (FDT). Linux je schopen z těchto informací nalézt a zaregistrovat hardware do systému během bootu.

- **Pinctrl**
  Subsystém pinctrl umožňuje přiřazovat k jednotlivým periferiím jejich piny (např. RX a TX pro UART nebo různé CS pro SPI) [10–11].

- **Pinmux**
  Pinmux (Pin Multiplexor) je název pro konfiguraci IO pinu, který může mít podle nastavení různou funkčnost (např. GPIO, UART atd.) v závistosti na zvoleném módu.

Příklad použití device tree pro popsání a případné zaregistrování sériové linky číslo 4 na pinech P9.11 a P9.13, který je použit v RPP testeru.

```
/** formát dat **/
/dts-v1/;

/** připojené soubory **/
#include "am33xx.dtsi"
#include "am33xx-overlay-edma-fix.dtsi"

/** doplnění popisu periferie pinmuxu - struktura am33xx_pinmux **/
/** se již nachází v souboru am335x-bone-common.dtsi, zde je    **/
/** pouze přidání dalšího uzlu                                  **/
&am33xx_pinmux {

    /** nastavení pinů pro UART 4 **/
    uart4_pins: pinmux_uart4_pins {
        pinctrl-single,pins = <
            AM33XX_IOPAD(0x870, PIN_INPUT_PULLUP | MUX_MODE6)
            /* p9_11.uart4_rxd */
            AM33XX_IOPAD(0x874, PIN_OUTPUT_PULLDOWN | MUX_MODE6)
            /* p9_13.uart4_txd */
        >;
    };
};

/** zaregistrování periferie UART 4 s odpovídajícími piny **/
/** (skupinou pinů)                                       **/
&uart4 {
    pinctrl-names = "default";
    pinctrl-0 = <&uart4_pins>;
    status = "okay";
};
```

2.6 Buildbot

Buildbot je open-source framework napsaný v jazyce Python. Slouží jako nástroj k průběžné integraci. Lze jej nakonfigurovat např. tak, aby automaticky vykonával posloupnosti příkazů jako např. kompilace softwaru, testování, zveřejňování aplikací atd. Podporuje integraci s verzovacími systémy. Příkladem jeho praktického využití v praxi je např. automatické spuštění řetězu úloh po přidání nového commitu do gitu. Z repozitáře je stažena poslední verze, která projde kompilací a otestováním unit a integračními testy. Mezi jeho přednosti patří podpora paralelního vykonávání podúloh, reportování průběhu akcí a využití více platforem. Buildbot je již od počátku využíván projektem RPP a můj tester byl pouze zaintegrován do již existujících procedur [12].

2.7 KiCad

KiCad je otevřený software pro tvorbu schémat elektronických obvodů a návrh plošných spojů. Je vyvíjen od roku 1992 a od r. 2013 se jeho patronem stala organizace CERN. O vývoj se stará komunita (v dubnu 2016 byl počet přispěvatelů do projektu 97) [13]. V této práci byl KiCad využit pro návrh hardwaru testovací desky.

Samotný projekt KiCad je rozdělen do několika částí (programů):

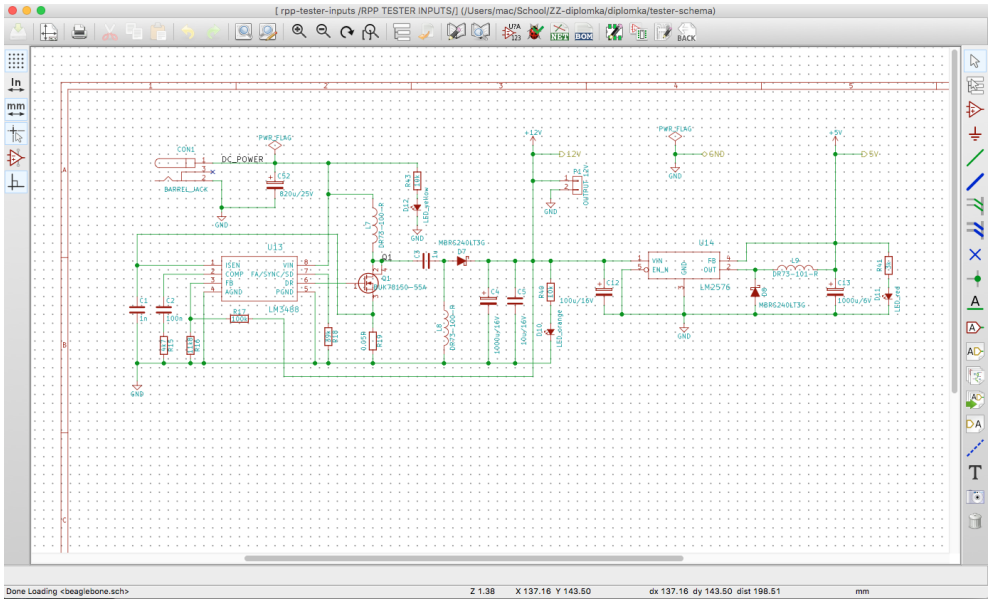
- KiCad – projektový manažer
- Eeschema – editor schémat el. obvodů
- Pcbnew – editor pro návrh desek plošných spojů (DPS – angl. Printed Circuit Board)
- CvPcb – program sloužící k propojení součástek ve schématu (včetně jejich fyzických pouzder) s návrhem DPS
- GerbView – prohlížeč Gerber souborů (Gerber je ASCII vektorový formát - standard pro výrobu DPS)

2.7.1 Eeschema

Eeschema slouží k návrhu schémat. Podporuje vytváření hierarchických listů a modulů, které zlepšují přehlednost návrhu obvodů. Princip tvorby spočívá ve vkládání komponent (pasivních i aktivních elekt. prvků) a jejich spojování pomocí vodičů. Vzniklé bloky je možné nechat programově zkontrolovat na elektrické chyby (např. nezapojené piny, propojení dvou výstupů atd.) (obr. 2.6).

- **CvPcb**

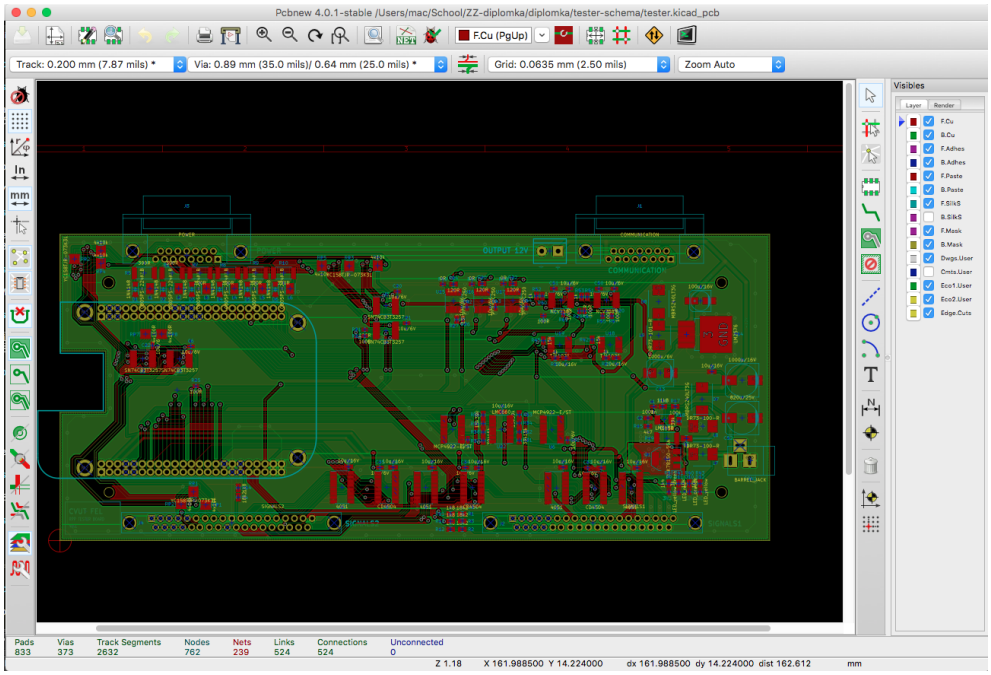
  Program CvPcb se používá po dokončení návrhu schémat. Jakmile jsou uživatelem (nebo automaticky) doplněny anotace komponent (každá jedinečným identifikátorem), lze ke každé blokové součástce připojit footprint (otisk součástky na DPS). Tento krok je nutný pro propojení schématu s layoutem.



Obrázek 2.6. Náhled uživatelského rozhraní programu Eeschema.

2.7.2 Pcbnew

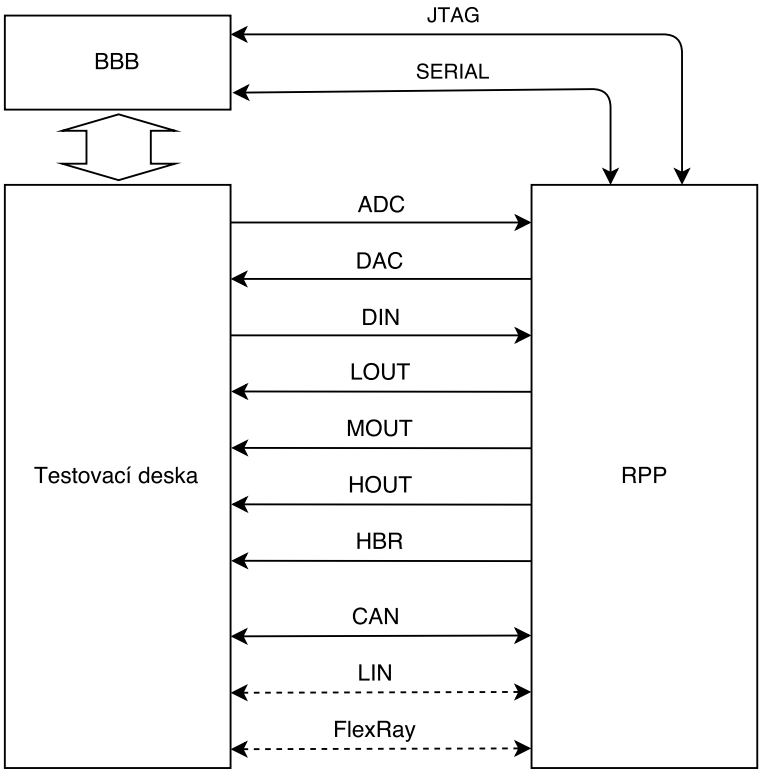
Pcbnew je program pro tvorbu DPS, který podporuje až 32 vodivých a 14 technických vrstev plošného spoje. Obsahuje kontrolu layoutu na vzdálenosti jednotlivých vodičů a jejich neúmyslnému propojení. Také umožňuje přecházení mezi layoutem a schématem, což v praxi velice usnadňuje a urychluje celý návrh (obr. 2.7).



Obrázek 2.7. Náhled uživatelského rozhraní programu Pcbnew.

Kapitola 3
Hardware pro testování

Tato kapitola popisuje návrh hardwarových částí práce. Na základě požadavků od vývojářů PES-RPP byla vypracována schémata elektr. obvodů a podle nich vyrobena a osazena deska plošných spojů (obr. 3.7). Ta slouží k testování PES-RPP desky (obr. 3.8). Je postavena na vývojovém produktu BeagleBone Black (BBB) s procesorem AM335x. Pomocí akčních vstupů/výstupů je schopna posílat a přijímat signály z PES-RPP. Blokové schéma všech prvků je zobrazeno na obr. 3.1.



Obrázek 3.1. Blokové schéma propojení RPP, testovací desky a BBB.

3.1 Požadavky na testovací desku a specifikace

V této sekci jsou uvedeny požadavky na testovací desku, které vyplynuly ze zadání a potřeb softwarových vývojářů.

Testovací deska bude sloužit k regresnímu testování a pro agilní vývoj prototypu univerzální řídicí jednotky (ECU) pro automobilový průmysl. Požadavkem je, aby deska byla programovatelná pomocí C/C++ (v ideálním případě na ní poběží OS Linux, a bude k němu plný přístup).

Testování výstupů ECU bude probíhat tak, že se prostřednictvím sériové linky budou do ECU posílat příkazy a pomocí testovací desky se ověří, že výstupy ECU jsou nastaveny dle parametrů. Vstupy ECU budou testovány obdobně. Pomocí testovací desky

se nastaví vstupy ECU do požadovaného stavu a pomocí příkazů sériové linky bude ověřeno, že ECU čte vstupy správně. V ideálním případě se budou testovat i komunikační rozhraní CAN, LIN, FlexRay a Ethernet. Následující sekce detailně specifikují jednotlivá rozhraní testovací desky.

3.1.1 Signály

- **Analogový výstup**
  - 12× DA převodník
  - Optimálně se symetrickým výstupem.
  - Napětí 0–12 V
- **Analogový vstup**
  - 4× AD převodník
  - Napětí 0–12 V
- **Digitální výstup**
  - 16× digitální výstup s volitelnou napěťovou úrovní log. 1 (prakticky DA)
  - Napětí 0–12 V
  - Rozhodovací úroveň ideálně 6 V
- **Digitální vstup**
  - 8+6× digitálních vstupů
  - Napětí 0–12 V
  - Rozhodovací úroveň ideálně 6 V

3.1.2 Vstupy pro měření výkonových výstupů

- **Vstup pro měření časových parametrů PWM**
  - 6× input capture
  - Napětí 0–12 V
  - Nutná indukční zátěž !!!
  - Lze použít multiplex
- **Vstup pro H-můstek vstup**
  - 2× input capture pro každý vstup
  - Napětí 0–12 V

3.1.3 Komunikace

- **CAN**
  - 3× CAN bus
  - Včetně budičů
- **LIN**
  - 2× LIN bus
  - Včetně budičů

- **FlexRay**
  - 2× FlexRay
  - Včetně budičů
- **SCI**
  - 1× UART
  - 3.3 V logika

3.1.4 Poptávka

Na základě výše uvedené specifikace byl poptán hardware u jedné nejmenované komerční firmy specializující se na hardware pro testováni. Nabízená cena byla ovšem na možnosti katedry příliš vysoká. Proto bylo přistoupeno k návrhu vlastního hardwaru, což je popsáno v následujících sekcích.

3.2 Navržený hardware a jeho rozhraní

Deska plošných spojů byla navržena pomocí programů z projektu KiCad. K návrhu byly použity standardní knihovny programy KiCad, open-source rozšíření FlyingBone (`https://github.com/piranha32/FlyingBone`) a vlastní knihovna obsahující bloková schémata a otisky komponent na DPS, které nebyly veřejně dostupné. Jednotlivá bloková schémata, vytvořená v programu *Eeschema*, jsou uspořádána do hierarchických listů (schéma E.4):

- **RPP** obsahuje konektory testovací desky, které jsou přímo připojeny k RPP (schéma E.5).
- **RPP TESTER OUTPUTS** obsahuje bloky výstupů (RPP ADC, RPP DIN) (schéma E.7).
- **RPP TESTER INPUTS** obsahuje vstupní periferie (RPP DAC, RPP LOUT, RPP HOUT, RPP MOUT, RPP HBR) (schéma E.6).
- **RPP TESTER COMMUNICATION** obsahuje komponenty pro komunikaci (CAN, FlexRay, LIN) (schéma E.9).
- **RPP TESTER POWER** obsahuje schémata napájecích spínaných zdrojů (12 V, 5 V) (schéma E.8).
- **BEABLEBONE** obsahuje piny desky BeagleBone Black (schéma E.10).

3.2.1 Návrh a výroba desky plošných spojů

Výsledná deska plošných spojů byla vytvořena exportem dat ze schématických listů vytvořených v programu Eeschema do aplikace *Pcbnew*. K propojení souborů slouží nástroje *CvPcb*. Pomocí Pcbnew byly součástky ručně rozmístěny po desce a následně propojeny vodiči a propojkami.

K exportu do standardního formátu pro výrobu DPS byl použit nástroj *GerbView*, který layout rozdělil do výrobních vrstev:

- **top** vrchní vrstva mědi
- **bot** spodní vrstva mědi
- **smt** vrchní nevodivá maska
- **smb** spodní nevodivá maska
- **plt** vrchní potisk

- **pth** vrstva prokovového vrtání
- **mill** obrys desky pro nařezání

- **Parametry DPS**
  - rozměry: 220x95 mm
  - vrstvy: 2x
  - materiál: Isola DE104, 1.55 mm
  - tloušťka Cu folie: 35 μm
  - zelená nepájivá maska: 2×
  - servisní potisk: 1×

- **Chyby návrhu DPS**

  Během návrhu DPS jsem se bohužel nevyvaroval drobných chyb. Největší komplikací je použití špatného pouzdra pro diody (D7, D8) u spínaných zdrojů (layout D.3). V nákresu DPS jsem navrhl pouzdro DO-214AB, ale následně objednal součástku s typem DO-214AA, což způsobilo menší komplikace při osazování, jelikož jednotlivé rozměry neodpovídaly. Problém byl vyřešen kusem vodiče a izolační podložkou.

3.2.2 Napájení a napěťové domény

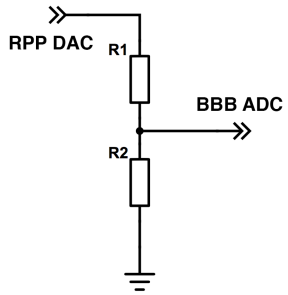
Testovací deska je napájena stejnosměrným napětím v rozsahu 4–20 V. O transformaci napětí na 12 V se stará snižující/zvyšující spínaný zdroj LM3488. Z takto generovaného napětí je napájena i testovaná RPP deska. V průběhu vývoje však vyšlo najevo, že spínané zdroje RPP mají při startu velký proudový odběr a mohou skončit v oscilujícím stavu, proto byl dodatečně k testeru přidán externí 10000 μF kondenzátor. Z 12 V je pomocí spínaného zdroje LM2576 vytvořeno stabilizované napětí 5 V sloužící pro napájení desky BBB. BeagleBone obsahuje obvody TPS65217C a TL5209 pro vytvoření 3.3 V. Komunikace s čipem Sitara AM335x je vždy provozována maximálně do této úrovně napětí (3.3 V).

3.2.3 Nízkovýkonové signály

V této sekci jsou popsána rozhraní testeru pro nízkovýkonové signály.

- **Analogový vstup (pro RPP DAC)**

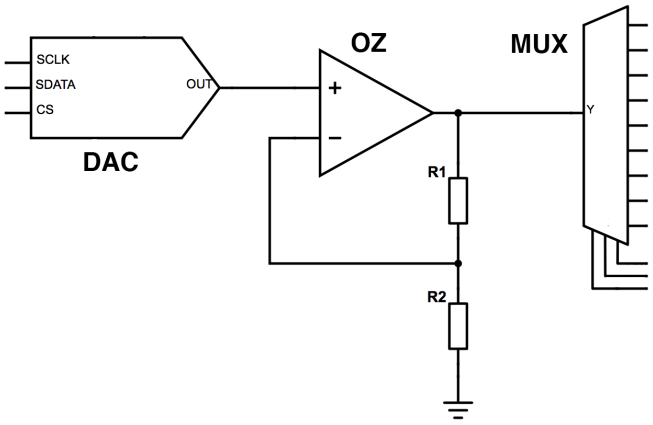
  Analogové vstupy testeru se skládají z děličů napětí, které převádí vstupní signál (max. 20 V) na nižší úroveň, a interních AD převodníků v procesoru AM335x (max. 1.8 V). Zjednošené blokové schéma periferie je zobrazeno na obr. 3.2.



Obrázek 3.2. Zjednodušené blok. schéma analogového vstupu.

Analogový výstup (pro RPP ADC)

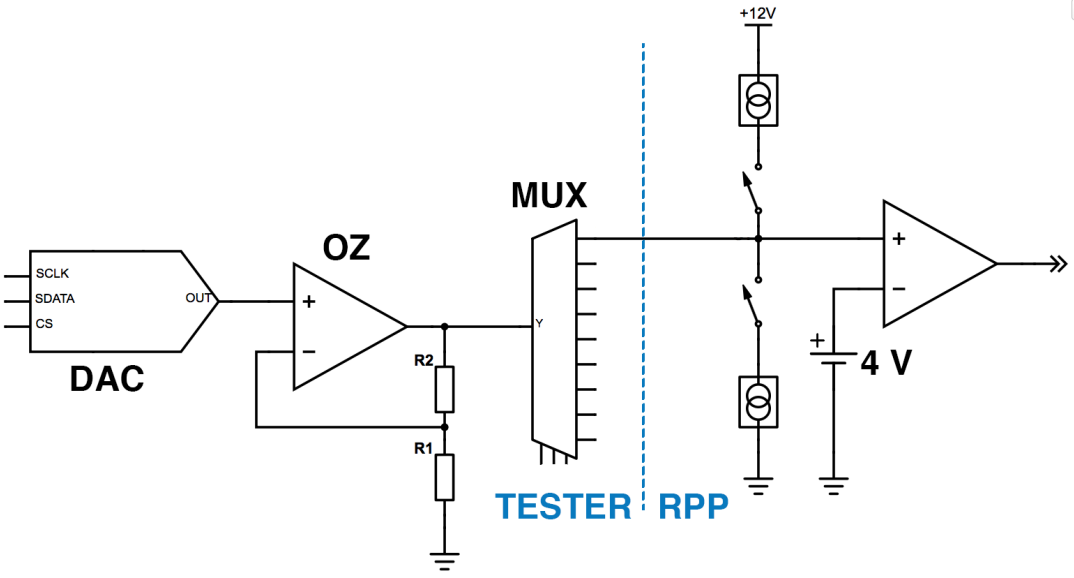
Analogové výstupy testeru jsou vyřešeny pomocí 16bitového DA převodníku MCP4922 (U6), který je ovládán přes SPI (schéma E.7). Ten generuje napětí v rozmezí 0–3.3 V, které je pomocí neinvertujícího zesilovače (OZ LMC660 rail-to-rail) (U22) zesíleno na hodnoty 0–12 V. Tento signál je následně přiveden přes analogový multiplexor CD4051 (U9, U10) na odpovídající výstupní konektor. Multiplexor má pro ovládání řídicích vstupů rozhodovací úroveň 6 V (polovina Ucc) a z tohoto důvodu je pro jeho řízení přidán obvod CD4504 (level shifter) (U23, U24), který v TTL módu provádí posun logické úrovně z 3.3 V na 12 V. Zjednošené blokové schéma periferie je zobrazeno na obr. 3.3.



Obrázek 3.3. Zjednodušené blok. schéma analogového výstupu.

Digitální výstup (pro RPP DIN)

Digitální výstupy testeru jsou navrženy shodně jako výstupy analogové. V základní konfiguraci se používají s nastavením DA převodníku (U7) v režimech 0 nebo 4095 lsb. Zároveň ale umožňují používat i celý interval hodnot od 0–12 V, což je vhodné pro budoucí testování rozhodovací úrovně DIN vstupu RPP desky. Zjednošené blokové schéma periferie je zobrazeno na obr. 3.4.

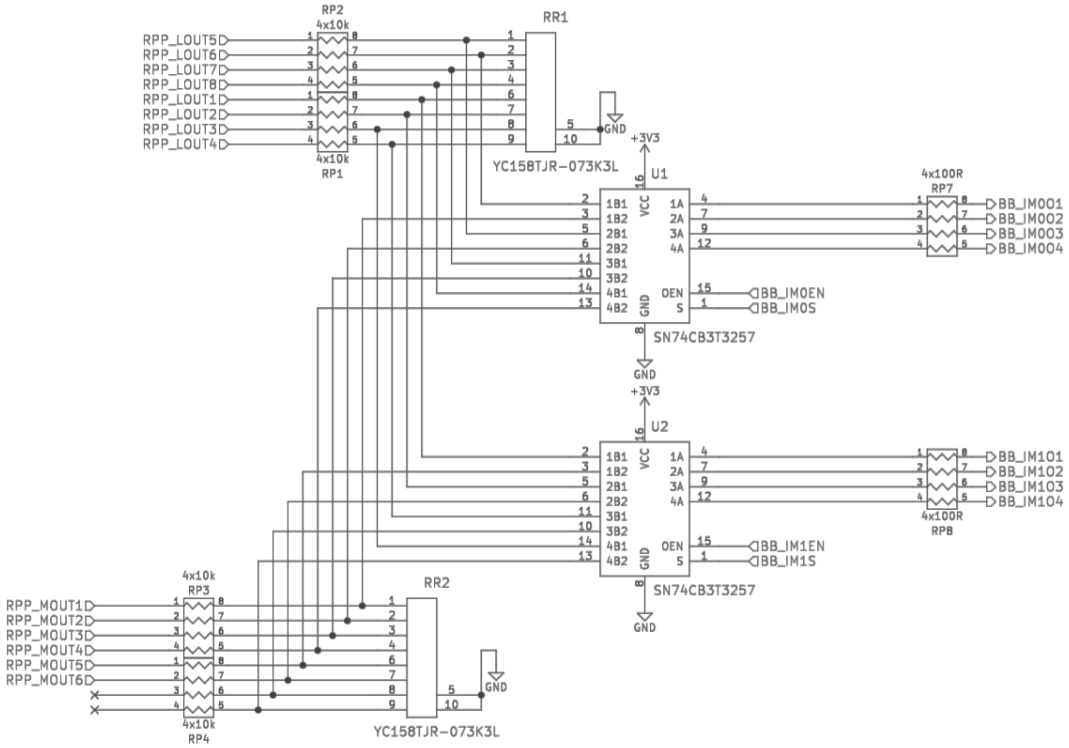


Obrázek 3.4. Zjednodušené blok. schéma digitálního výstupu.

V návhnu je bohužel chyba, jelikož nebyla brána v potaz přítomnost proudových zdrojů DIN RPP, které zde tvoří pull-up nebo pull-down nastavení. V případě zapnutí DIN RPP pull-up se aktivuje 16 mA proudový zdroj, který se spíná do země přes OZ (a jeho zpětnou vazbu). Jednak má multiplexor 4051 mezní hodnotu dovoleného proudu 10 mA a zároveň úbytky napětí na všech prvcích tvoří hodnotu větší než je rozhodovací úroveň RPP komparátoru (4 V). Při testování je tedy nutné vyvarovat se zapnutí pull-up/down módu na DIN vstupech.

- **Digitální vstup (pro RPP LOUT)**

  Digitální vstupy testeru jsou tvořeny dvojicí digitálních multiplexorů obvodu SN74CB3T3257 (U1, U2). Ten obsahuje 4 dvojice 1:2 a lze jej ovládat 3.3 V logikou. Zároveň poskytuje ochranu vstupů, na něž lze připojit napětí až do výše 7V. Tato úroveň je převedena na max. 3.3 V. Jelikož výstup z RPP ve stavu logické jedničky může být až 12 V, před každým vstupem je umístěn dělič napětí. Zapojení je navrženo tak, že při jednom nastavení multiplexorů lze přečíst všechny výstupní hodnoty RPP LOUT (8 pinů). Blokové schéma je zobrazeno na obr. 3.5.



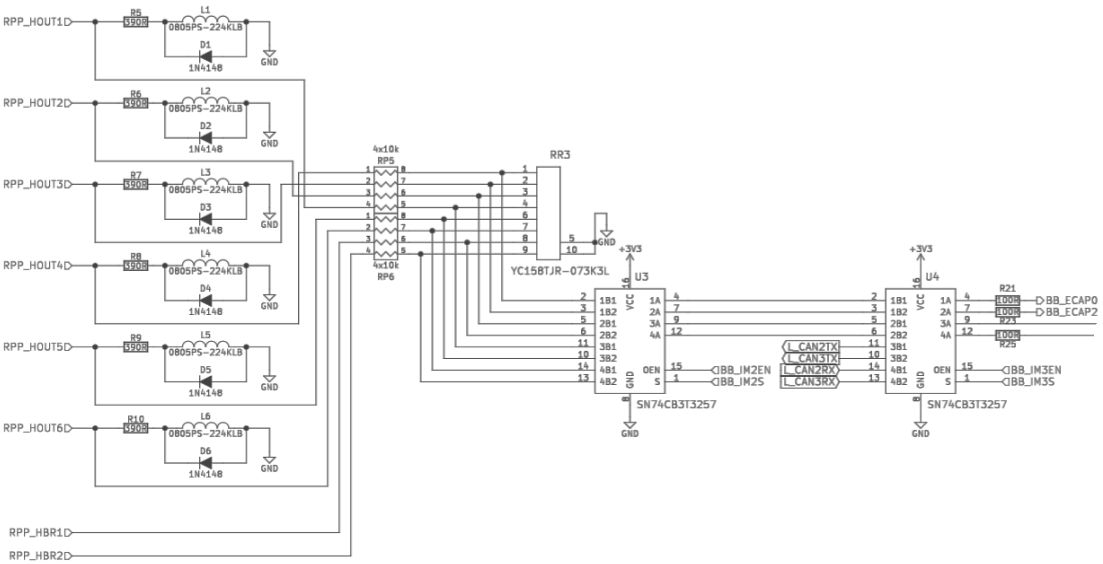
Obrázek 3.5. Blok. schéma logických vstupů.

3.2.4 Vstupy pro měření výkonových výstupů

V této sekci jsou popsána rozhraní testeru pro měření výkonových vstupů.

- **Vstup pro měření časových parametrů PWM (pro RPP HOUT)**

  Obvod RPP HOUT je tvořen budiči AUIR33401S, které vyžadují indukční zátěž, v opačném případě nahlásí chybu a přestanou generovat PWM. Z tohoto důvodu Tester RPP obsahuje kombinaci odporu (390 Ω) a cívky (220 μH) navrženou na protékající proud cca 30 mA. Paralelně k cívkám je přidána rekuperační dioda 1N4148 jako ochrana před napěťovými pulsy, které by indukčnost mohla způsobovat. Signály jsou pomocí multiplexorů SN74CB3T3257 (U3, U4) přivedeny na eCAP periferii BBB. Napětí je ještě před vstupem do multiplexoru zmenšeno z 12 V na cca 3.6 V. Blokové schéma je zobrazeno na obr. 3.6.



Obrázek 3.6. Blokové schéma výkonových vstupů.

- **H-můstek vstup (pro RPP HBR)**

  Pro měření PWM generované obvodem L99H01 (H-můstek) RPP desky je využit opět BBB eCAP. Struktura multiplexorů (U3, U4) je sdílena s HOUT. V jeden okamžik lze souběžně zaznamenávat oba RPP HBR výstupy a tím detekovat případnou chybu. Blokové schéma je zobrazeno na obr. 3.6.

- **Výkonový digitální vstup (pro RPP MOUT)**

  Výkonové digitální vstupy testeru (MOUT) jsou vyřešeny zcela totožným způsobem jako LOUT. Blokové schéma je zobrazeno na obr. 3.5.

3.2.5 Komunikační rozhraní

V této sekci jsou popsána komunikační rozhraní testeru.

- **CAN**

  BBB obsahuje 2 CAN řadiče, oproti tomu RPP má 3 řadiče. Z tohoto důvodu je BBB CAN0 napojen přímo na budič sběrnice (PCA82C250 – U15) a následně propojen s RPP CAN1. BBB CAN1 lze přepínat pomocí multiplexoru (U4) mezi Tester budiči (U16, U17). Ty jsou zapojeny na RPP CAN2 a CAN3. Sběrnice jsou oddělené a každá je zakončena terminátorem o hodnotě 120 Ω.

- **LIN**

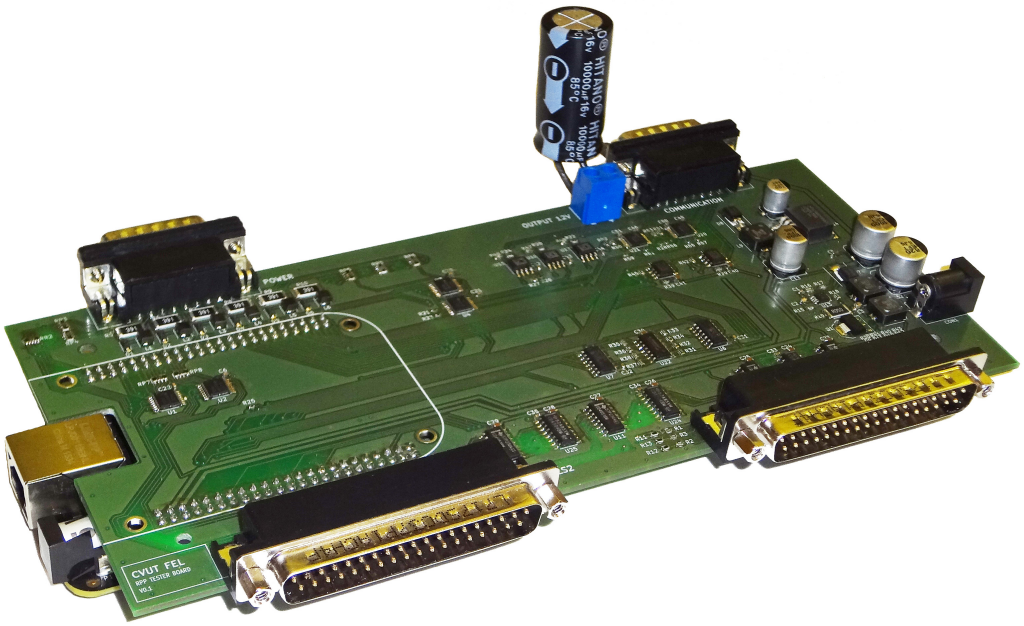
  LIN komunikace je řešena pomocí budiče sběrnice (TJA1021 – U18, U19), který je propojen s UART obvody BBB. Testování LIN periferie není součástí této práce, ale je zajištěna HW podpora pro možnost další implementace.

- **FlexRay**

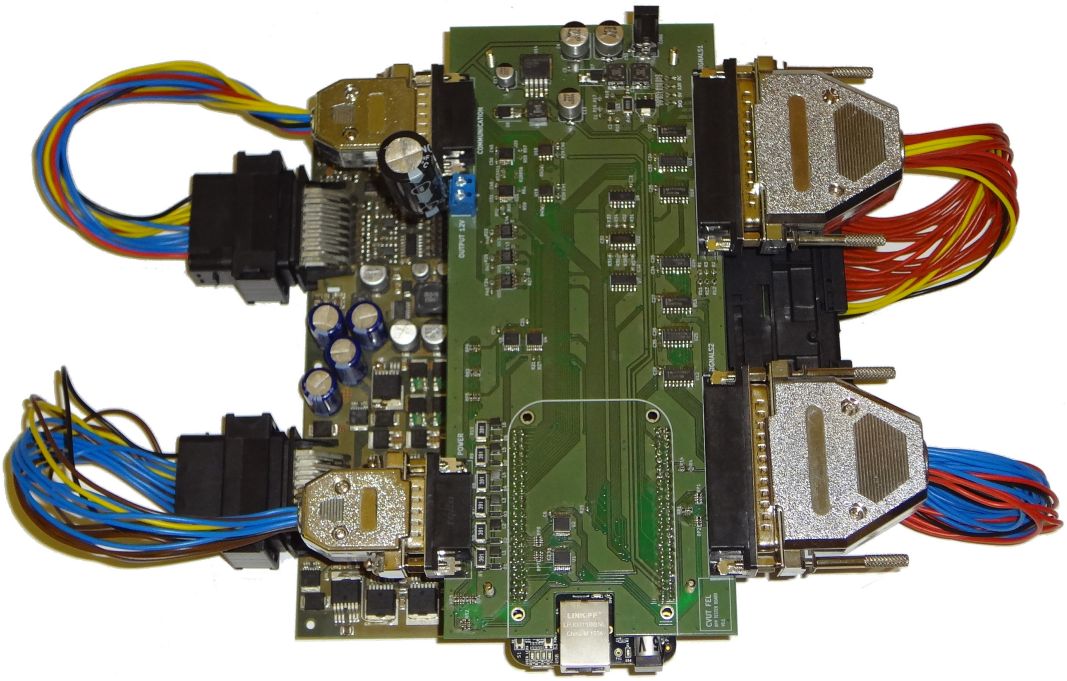
  BBB neobsahuje FlexRay řadič. Z tohoto důvodu jsou vstupy a výstupy z FR budičů (NCV7383 – U20, U21) přivedeny do Sitara procesoru na vstupně/výstupní piny, které lze propojit s vnitřními PRU-ICSS. Implementace těchto řadičů není rovněž součástí této práce. Nicméně hardwarová podpora je stejně jako u LIN zachována.

- **SCI**

  K zadávání příkazů RPP desce po sériové lince slouží USB-Serial TTL převodník. Ten je zapojen do USB portu BBB a pracuje v logice do 3.3 V.



Obrázek 3.7. Fotografie testovací desky.



Obrázek 3.8. Fotografie testovací desky spojené s RPP deskou.

Kapitola 4
Softwarová podpora pro testování

Tato kapitola popisuje funkcionalitu pythonovského frameworku RPP-Tester. Zároveň je zde detailně probrán postup instalace a nastavení OS Debian na desku BeagleBone Black. Jsou zde objasněny použité technologie a způsoby jejich použití – např. OpenOCD, Device Tree, Buildbot a eCAP driver.

4.1 Linux na BeagleBone Black

Jedním z důvodů, proč jsem zvolil Linux jako operační systém bylo, že celá práce si již od začátku kladla za cíl používat výhradně open-source řešení. Navíc je dle mého skromného názoru nejlepším řešením pro větší embedded projekty. Další nespornou výhodou je silná podpora komunity, která speciálně pro procesor AM335x zanáší nové změny a vylepšení přímo do hlavní linie vývoje linuxového jádra.

4.1.1 Instalace

Tato sekce podrobně popisuje jednotlivé kroky nutné k instalaci všech potřebných softwarových komponent (OS, programy atd.). Návod je psán pro typ instalace na SD kartu, ale lze jej využít i pro interní úložiště.

Nejprve je nutné stáhnout aktuální verzi linuxové distribuce Debian z oficiálních stránek `https://beagleboard.org/latest-images`. Doporučená verze je Debian 8.4. Po stažení souboru nahrajte binárku do microSD karty příkazem uvedeným níže (platí pro Linux). Parametr *if* (input file) nahraďte názvem staženého souboru a do argumentu *of* (output file) doplňte název karty, která je připojena do vašeho systému.

```
$ dd bs=4M if=jessie.img of=/dev/sdd
```

Pro práci s BBB je potřeba získat přístup k příkazové řádce. Lze využít ethernetové rozhraní, které získá svou adresu pomocí protokolu DHCP a následně je možné připojení přes SSH. Druhou možností je použití sériové konzole, která je k dispozici přes USB port. Jako první krok doporučuji odebrat nepotřebné balíčky, které pro projekt nemají význam a zbytečně zabírají kapacity úložiště.

```
$ apt-get purge "apache*"
$ apt-get purge "desktop*"
$ apt-get purge "node*"
$ apt-get purge "xorg*"
$ apt-get purge "x11-*"
$ apt-get autoremove
```

Pro desku RPP tester jsem musel upravit device tree a tato modifikovaná verze vyžaduje nové jádro 4.4. Upgrade distribuce na aktuální verzi není povinný, ale doporučuji jej.

```
$ apt-get update
$ apt-get install linux-image-4.4.6-bone6
$ apt-get dist-upgrade
$ reboot
```

Dalším krokem je instalace zdrojových kódu testeru. Všechny soubory lze nalézt v rtime git repozitáři. Přístup není veřejný a je nutné zažádat o povolení. Aktuální verze je přiložena k této práci. Celý repozitář je nutné naklonovat do složky */opt/rpp-tester* z důvodu použití absolutních cest.

```
$ cd /opt
$ git clone git@rtime.felk.cvut.cz:pes-rpp/rpp-tester
```

K nastavení device tree slouží Makefile v adresáři */opt/rpp-tester/software/dts*. Ten vygeneruje novou device tree strukturu přesně pro požadavky RPP Testeru (*rpp-tester.dtb*). K jejímu načtení při startu čipu pomocí u-bootu je nutné přidat do konfiguračního souboru *uEnv.txt* název dtb souboru do parametru dtb.

```
$ cd /opt/rpp-tester/software/dts
$ make
$ echo "dtb=rpp-tester.dtb" >> /boot/uEnv.txt
```

Device tree popíše a přiřadí periferie systému (včetně pinmuxů), ale samotné nastavení GPIO portů zajišťuje inicializační skript, který je třeba zaregistrovat, aby se po bootu spustil. Má za úkol rovněž inicializovat sběrnice CAN.

```
$ ln -s /opt/rpp-tester/software/init/init /etc/init.d/rpp-tester
$ chmod 755 /etc/init.d/rpp-tester
$ update-rc.d rpp-tester defaults
```

Samotný testovací framework RPP-Tester lze jednoduše nainstalovat pomocí programu *pip*.

```
$ apt-get install python3
$ apt-get install python3-pip
$ pip3 install pytest
$ cd /opt/rpp-tester/software/python
$ make
```

Framework využívá ke komunikaci po sériové lince symbolický odkaz */dev/usb_uart*. Původní přímé navázání na /dev/ttyUSB0 bylo odebráno z důvodu možné záměny v případě připojení více USB zařízení (např. JTAG konvertor pro OpenOCD). Změna jména souboru zařízení se nastavuje v konfiguraci udevu. K tomu je potřeba znát identifikátor daného USB zařízení. K zjištění identifikátorů je doporučeno používat následující příkaz, kterým lze získat hodnoty $ATTRS\{idVendor\}$ a $ATTRS\{idProduct\}$.

```
$ udevadm info -a -p  $(udevadm info -q path -n /dev/ttyUSB0)
```

Pro správné přiřazení symlinků je nutné editovat (případně vytvořit) soubor */etc/udev/rules.d/10-local.rules*. Ukázka níže odpovídá využití *FT230X Basic UART* jako *usb_uart* a *Texas Instruments Inc.XDS100 Ver 2.0* jako *usb_jtag*. Bez nastavení usb_uart nemůže framework komunikovat s RPP deskou.

```
ACTION=="add", ATTRS{idVendor}=="0403", \
  ATTRS{idProduct}=="6015", SYMLINK+="usb_uart"
ACTION=="add", ATTRS{idVendor}=="0403", \
  ATTRS{idProduct}=="a6d0", SYMLINK+="usb_jtag"
```

Pro potřeby získání dokumentace přímo ze zdrojových kódů je určen program *pdoc*. Generuje přehlednou dokumentaci do formátu html. Tento krok není nutný, ale doporučuje se, protože následně se lze lépe orientovat v dostupných funkcích. Tento krok lze provést i na vlastním PC, ale je nutné používat OS Linux kvůli závislostem na hlavičkové soubory SPI periferie.

```
$ pip3 install pdoc
$ cd /opt/rpp-tester/software/python
$ make create-doc
```

Pokud chce vývojář prohlížet dokumentaci a výsledky testů přes webový prohlížeč, doporučuji použít webový server *nginx*. Při zadání přednastavené konfigurace lze následně po připojení na BBB port 80 vše prohlížet přes webový prohlížeč. Jelikož je port 80 po základní instalaci systému obsazen jinými službami, je nutné je zakázat.

```
$ systemctl disable cloud9.service
$ systemctl disable gateone.service
$ systemctl disable bonescript.service
$ systemctl disable bonescript.socket
$ systemctl disable bonescript-autorun.service
$ systemctl disable avahi-daemon.service
$ systemctl disable gdm.service
$ systemctl disable mpd.service
$ apt-get install nginx
$ cd /etc/nginx/sites-enabled
$ cd rm default
$ ln -s /opt/rpp-tester/software/server/nginx.conf default
$ service nginx restart
```

K nahrávání nových binárek přímo do Flash paměti RPP desky je nutné nainstalovat OpenOCD z přiloženého archivu. Instalační skript je upraven pro procesory TMS570 a obsahuje doporučenou konfiguraci.

```
$ apt-get install zip libftdi-dev libhidapi-dev libusb-1.0.0
$ cd /tmp
$ cp /opt/rpp-tester/software/openocd/openocd-tms570-f021-wip.zip .
$ unzip openocd-tms570-f021-wip.zip
$ cd openocd-tms570-f021-wip
$ bash openocd.configure-smirnov
$ make
$ make install
$ cd /usr/local/bin
$ ln -s /opt/openocd-tms570/bin/openocd .
```

Jelikož systém běžící na SD kartě (nebo vnitřním eMMC) by po několika měsících mohl způsobit poškození média, je vhodné zakázat zápis na tato úložiště. Jakmile budete mít všechny nástroje a nastavení připraveny, doporučuji upravit nastavení funkce *mount* v souboru */etc/fstab* na přibližně následující obsah:

```
UUID=xxx  /  ext4 ro,noatime,errors=remount-ro  0  1
debugfs  /sys/kernel/debug  debugfs  defaults  0  0
tmpfs /tmp tmpfs defaults
tmpfs /var/log tmpfs defaults
```

U výše uvedené konfigurace v prvním řádku výrazem *ro* nastavujeme *read-only* režim a řádky s řetězcem *tmpfs* značí připojení ramdisku do daného adresáře.

V případě vzniku požadavku na zápis do vnitřního úložiště tak lze učinit příkazem níže. Pro přechod zpět do read-only režimu stačí parametr *rw* nahradit za *ro* (nebo restarovat systém).

```
$ mount -o remount,rw /
```

4.1.2 Device Tree

Pro správnou funkčnost testovací desky bylo nutné upravit device tree. V přiloženém souboru *rpp-tester.dts* jsou přidány rozšiřující uzly, které nastavují pinmuxy IO pinů a přiřazují je k jednotlivým periferiím:

- UART4 a UART5
- SPI0
- ECAP0 a ECAP2
- CAN0 a CAN1
- GPIO porty

4.1.3 eCAP driver

Periferie eCAP je použita pro měření RPP HOUT a HBR PWM pulsů. RPP-Tester obsahuje pythonovské rozšíření implementované v jazyku C, které přistupuje přes funkci *mmap* přímo na registry modulu eCAP. Linuxové jádro totiž obsahuje verzi ovladače eCAP, která je schopná pouze generování signálů, nikoliv jejich měření.

Registry bylo třeba nastavit pro zachycení vzestupné i sestupné hrany s použitím absolutních hodnot čítače. Rozšíření ve smyčce čte hodnoty z eCAP registrů a zapisuje údaj o hodnotě a typu hrany do pole. Po ukončení cyklu, které je vyvoláno buď dosažením přednastaveného počtu měření nebo přesáhnutím dovoleného času měření, dojde k výpočtu parametrů změřených pulsů (minimální/maximální/průměrná hodnota periody/střídy). Stěžejní část kódu nastavujícího eCAP periferie je uvedena níže.

Jelikož obsloužení přerušení není z uživatelského prostoru možné, výčet hodnot ze záchytných registrů probíhá ve while cyklu. Operační systém může v této době přeplánovat běh procesů a vzniklé zpoždění překročí mez, kdy dojde k vynechání přečtení jednoho (nebo více) cyklu. To samozřejmě nepříznivě ovlivní výsledek celého měření. Změřená data jsou nicméně vrácena uživateli a k nim je přidán i údaj o maximálním časovém rozdílu mezi dvěma měřeními. Následně je na uživateli, jestli výsledky považuje za validní.

```
int c_ecap_initRegister(uint8_t ecapId, ecapRegister ** ecapX)
{
    // základní adresa pro submodul nastavení podle čísla periferie
    off_t baseAddress = ecapId == ECAP0_ID ?
       ECAP0_BASE_ADDRESS : ECAP2_BASE_ADDRESS;

    // zjištění velikosti stránky systému
    long pageSize = sysconf(_SC_PAGE_SIZE);

    // získání offsetu, který zarovná ukazatel podle velikosti stránky
    off_t offset = baseAddress;
    offset = (offset / pageSize) * pageSize;

    off_t addrOffset = baseAddress - offset;

    // ukazatel do virtuální paměti, která ukazuje na fyz. adr. registrů
    void * ecapPointer = mmap(0, ECAP_MEM_SIZE + addrOffset
     PROT_READ | PROT_WRITE, MAP_SHARED, fd, offset);

    // funkce mmap proběhla úspěšně
    if (ecapPointer != MAP_FAILED)
```

```
    {
     // přetypování a přidání offsetu
        *ecapX = (void *)((unsigned char *)ecapPointer + addrOffset);
    } else {
        fprintf(stderr, "open nmap error : %s\n", strerror(errno));
        return 5;
    }

    (*ecapX)->ECCTL1 =
    (EC_RISING << ECCTL1_CAP1POL) |   // CAP1 zachytí vzest. hranu
    (EC_FALLING << ECCTL1_CAP2POL) |  // CAP2 zachytí sest. hranu
    (EC_RISING << ECCTL1_CAP3POL) |   // CAP3 zachytí vzest. hranu
    (EC_FALLING << ECCTL1_CAP4POL) |  // CAP4 zachytí sest. hranu
    (EC_ABS_MODE << ECCTL1_CTRRST1) | // CAP1 bere abs. hodnotu
    (EC_ABS_MODE << ECCTL1_CTRRST2) | // CAP2 bere abs. hodnotu
    (EC_ABS_MODE << ECCTL1_CTRRST3) | // CAP3 bere abs. hodnotu
    (EC_ABS_MODE << ECCTL1_CTRRST4) | // CAP4 bere abs. hodnotu
    (EC_ENABLE << ECCTL1_CAPLDEN) |   // povolení zachytávání
    (EC_DIV1 << ECCTL1_PRESCAL);      // dělička impulsů = 1

    (*ecapX)->ECCTL2 =
    (EC_CAP_MODE << ECCTL2_CAPAPWM) |   // capture mód
    (EC_CONTINUOUS << ECCTL2_CONTONE) | // kontinuální záznam
    (EC_SYNCO_DIS << ECCTL2_SYNCOSEL) | // zakáz. synchron.
    (EC_DISABLE << ECCTL2_SYNCIEN) |    // zakáz. synchron.
    (EC_EVENT4 << ECCTL2_STOPWRP)|      // záznam do všech 4 registrů
    (EC_RUN << ECCTL2_TSCSTOP);         // abs. hodn. čítače

    return 0;
}
```

4.2 RPP-Tester

RPP-Tester je framework napsaný v jazyce Python (verze python3). Slouží pro potřeby testování fyzických periferií RPP desky. Umožňuje s root oprávněním přímý přístup k hardwaru Tester desky a přes sériovou konzoli i k periferiím RPP desky. V následujících sekcích jsou popsány jednotlivé moduly a třídy frameworku.

4.2.1 RPP

RPP je modul RPP-Tester frameworku, který přes sériovou konzoli komunikuje s programem rpp-test-sw běžícím na RPP desce. Pomocí příkazů nastavuje a čte hodnoty jednotlivých periferií. V zásadě se jedná o jednoduchou pythonovskou nadstavbu nad příkazy rpp-test-sw.

- **Console**

  Pomocí submodulu Console lze zadávat příkazy sériové konzoli a číst návratové hodnoty. Komunikace probíhá s využitím python modulu *pyserial*, který posílá a čte data na externí USB-SCI konvertor.

- **ADC**
  Submodul ADC umožňuje číst hodnoty změřené RPP analog-digital převodníky. Data lze získat v hodnotě lsb bitů přímo z procesoru nebo jako přepočtený údaj v milivoltech.

- **CAN**
  Submodul CAN obsahuje základní funkce pro příjem a vysílání CAN zpráv na sběrnici. Zároveň lze nastavit baud rate a časování řadičů.

- **DAC**
  Pomocí submodulu DAC lze inicializovat a změnit výstup RPP digital-analog převodníků. Vstupní proměnou může být údaj v lsb bitech nebo hodnota v milivoltech.

- **DIN**
  Submodul DIN nabízí možnost nastavení jednotlivých digitálních vstupů RPP a čtení aktuálních hodnot na konkrétních pinech. Nastavit lze použití pull-up/pull-down proudových zdrojů, případně jejich celkové vypnutí a převedení pinu do stavu vysoké impedance (tri-state).

- **HBR**
  Submodul HRB slouží ke konfiguraci a spuštění/vypnutí RPP H-můstku. Lze zadat hodnotu rychlosti a šířky pulsu.

- **HOUT**
  Submodul HOUT umožňuje nastavovat parametry PWM (periodu a střídu) a číst aktuální stav jednotlivých periferií. V praxi se často stává, že při nevhodné konfiguraci končí některé bloky ve stavu FAIL a je nutné je před dalším testováním vyresetovat.

- **LOUT**
  Submodul LOUT umožňuje nastavit binární hodnoty logického výstupu.

- **MOUT**
  Submodul MOUT je v principu kopií submodulu LOUT, hlavním rozdílem je však využívání výkonových výstupů, které ale umožňují opět pouze binární výstup.

4.2.2 Tester

Tester je modul RPP-Tester frameworku, který umožňuje přímý přístup k HW periferiím testovací desky. Názvy submodulů jsou úmyslně voleny podle periferií RPP desky, pro usnadnění používání celého frameworku, zvláště pak při psaní testů. Např. *rpp_tester.tester.adc* pak v praxi nastavuje hodnoty DA převodníků na testovací desce, které jsou připojeny k ADC vstupům na RPP. Jelikož Tester modul v sobě obsahuje sekce kódu, které využívají přímé mapování paměti, je nutné jej používat (a v základu i celý framework) s právy uživatele root.

- **ADC**
  Submodul ADC umožňuje nastavovat hodnoty DA převodníků. Lze zadat parametry v milivoltech nebo jako hrubé číslo pro DAC. Vnitřní logika funkcí zajistí, aby se argumenty převedly na odpovídající hodnoty pro HW. Tato vlastnost je přidána pro pohodlnější psaní testů bez nutnosti přepočítávat parametry v samotných testech.

CAN

Submodul CAN zajišťuje možnost posílat a zachycovat zprávy na CAN sběrnici. Zachycování probíhá ve dvou cyklech, kdy nejprve je nutné spustit odposlouchávač sběrnice, dále poslat z RPP data a až nakonec metodou stop_dump přečíst zprávy. To samé platí i pro opačný směr komunikace. Pro účely diagnostiky je do submodulu přidána metoda pro výpis systémových informací o CAN řadičích.

DAC

Submodul DAC může číst hodnoty na pinech ADC převodníků BBB. Přečtená hodnota je vnitřně upravena, aby odpovídala lsb 4095 pro 12 V, případně 12000 při čtení miliVoltů. S ovladačem ADC je možné komunikovat pomocí souborů ve složce */sys/bus/iio/devices/iio:device0/*, kde framework pouze vyčítá data ze souborů.

DIN

Submodul DIN disponuje funkcí pro nastavení binární hodnoty pinu 0 nebo 1. Zároveň je možné přes metodu *set_high_impedance* uvést všechny výstupy do stavu vysoké impedance.

HBR

Submodul HBR využívá C rozšíření eCAP, kterým lze měřit průběh PWM pulsů na pinech HBR1 a HBR2. Výstupem měření je pole o dvou strukturách („ecap0“ „ecap2“), které obsahují následující páry klíč/hodnota:

- **periodMin** je minimální naměřená hodnota periody
- **periodMax** je maximální naměřená hodnota periody
- **periodAvg** je průměrná naměřená hodnota periody
- **dutyPosMin** je minimální naměřená hodnota délky pulsu v logické 1
- **dutyPosMax** je maximální naměřená hodnota délky pulsu v logické 1
- **dutyPosAvg** je průměrná naměřená hodnota délky pulsu v logické 1
- **dutyNegMin** je minimální naměřená hodnota délky pulsu v logické 0
- **dutyNegMax** je maximální naměřená hodnota délky pulsu v logické 0
- **dutyNegAvg** je průměrná naměřená hodnota délky pulsu v logické 0
- **measuresCount** je počet úspěšných měření (počet vzestupných a sestupných hran)
- **maxMeasureDiff** je maximální rozdíl mezi dvěma měřeními, slouží pro kontrolu uživatele, zda mohly být vynechány pulsy (nutno řešit přímo v testu)

HOUT

Submodul HOUT podobně jako HBR využívá eCAP rozšíření. Hlavní rozdílem je, že kde funkce *measure_pwm* vrací místo pole strukturu samotnou (popsanou v HBR výše).

LOUT

Submodul LOUT umožňuje číst binární hodnotu výstupu RPP LOUT. Je možné číst pouze jeden port nebo zavolat funkci, která vrátí v poli aktuální stav všech portů.

MOUT

Submodul MOUT má stejnou funkčnost jako LOUT, jen pracuje s jinými piny.

- **Internal**
  Interní submodul, který v praxi vývojář přímo neobsluhuje, a který je využit v Tester submodulech např. ke komunikaci po SPI atd.

  - **eCAP**
    Interní submodul eCAP je naimplementovaný v jazyce C a přímo využívá eCAP driver. Jedinou metodou je *measure_pwm*, která vrací list dvou struktur popsaný v sekci Tester HBR. Případně vyhazuje výjimku, pokud dojde k chybě.

  - **GPIO**
    Interní submodul GPIO nastavuje a čte binární hodnoty pinů BBB. K práci s touto periferií se používá čtení a zápis ze souborů ve složce */sys/class/gpio*. Uživateli je dovoleno číst i zapisovat do výstupních pinů, u vstupních je samozřejmě umožněno pouze čtení. K identifikaci pinu pro metody *set_value* a *get_value* slouží název funkce gpio. Seznam těchto klíčů je uveden ve vnitřní proměnné submodulu – *_gpios*.

  - **SPI**
    Interní submodul SPI zajišťuje zápis dat přes SPI rozhraní. Momentálně je využit pouze pro komunikaci s DA převodníky s použitím knihovny *spidev*. Čtení dat prozatím není podporováno.

  - **Mux**
    Interní submodul Mux v sobě obsahuje další submoduly, které zprostředkovávají nastavení multiplexorů *4051* a *SN74CB3T3257*. Nastavení se provádí metodou *set_mode* s parametrem čísla elementu, případně názvu periferie (např. „hbr“, „can“ nebo „hout“).

4.2.3 Asserts

Asserts je modul RPP-Tester frameworku, který usnadňuje vytváření jednotlivých testů HW komponent. Jelikož použitý nástroj py.test je jako většina ostatních testovacích programů zaměřen na ověření funkčnosti a nalezení problému ve zdrojovém kódu, modul Assert se snaží cílit na popis chyb samotného HW. K tomu obsahuje několik tříd, které vykonávají funkci testu a výpisu chyby zároveň. Pro jejich shromažďování a kontrolu slouží třída AssertsCollector.

- **EqualAssertRppInput**
  Slouží k přesnému porovnání hodnoty přečtené RPP deskou proti hodnotě nastavené Tester deskou. Použití např. pro logické vstupy RPP.

- **EqualAssertRppOutput**
  Podobně jako EqualAssertRppInput porovnává hodnoty. Konkrétně to je ale výstup z RPP proti hodnotě přečtené Testerem. Použití např. pro logické výstupy RPP.

- **RangeAssertRppInput**
  Zjišťuje, jestli je vstup RPP v zadaném rozmezí, které vyplývá z výstupní Tester hodnoty. Použití např. pro RPP ADC.

- **RangeAssertRppOutput**
  Podobně jako RangeAssertRppInput testuje, zda hodnota přečtená Testerem leží v zadaném intervalu, který se odvíjí od nastavené výstupní hodnoty RPP. Použití např. pro RPP DAC.

- **InfoError**
  Z důvodu kompatibility s třídou AssertsCollector InfoError implementuje interface IAssert. Slouží pro předání informace o chybném průběhu testu, který přímo nesouvisí s naměřenými hodnotami, ale popisuje vzniklou situaci. Použití např. v testech HOUT, kde může docházet ke špatné inicializaci RPP periferie.

- **AssertsCollector**
  AssertsCollector je třída, která sbírá všechny asserty, na konci testu je vyhodnotí a vypíše nalezené chyby. Jelikož nelze použít standardní assert jazyka python, jenž by vyvolal výjimku okamžitě a přerušil např. probíhající cyklus testů, AssertsCollector tak učiní až zcela na závěr a díky tomu lze otestovat všechny jednotlivé komponenty periferie v cyklu.

4.3 Integrace s verzovacím systémem

Pro integraci s verzovacím systémem byl využit nástroj *buildbot*. Přes SSH spouští připravené skripty, které využívají OpenOCD a framework RPP-Tester.

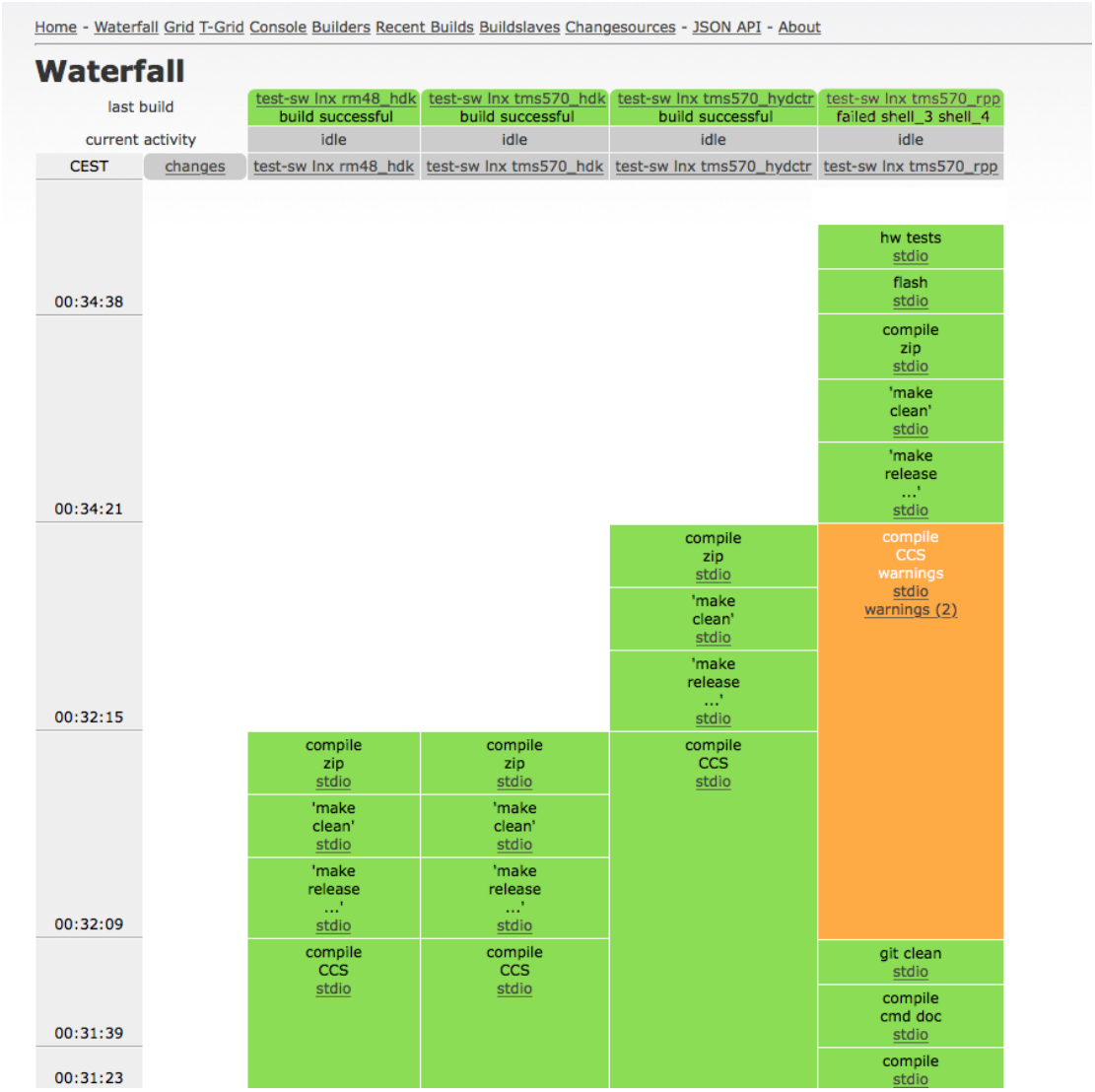
4.3.1 Buildbot

Do konfiguračního souboru buildbotu byl přidán skript uvedený níže, který má za úkol zkompilovanou binárku pro RPP přes SSH nahrát do adresáře */tmp*. Následně si ji přebere program *run-flash*, který se pokusí pomocí openOCD nahrát data do vnitřní FLASH paměti RPP desky. Pokud tato procedura skončí úspěšně, je zavolán skript *run-tests*, který spustí integrační testy. V buildbotu lze po skončení testů (případně i v průběhu) prohlížet stdout a stderr výstup (obr. 4.1)). Výsledek každé úlohy spuštěné buildbotem se ověřuje podle návratové hodnoty (0 značí bezchybný průběh a cokoliv jiného chybu).

```
# spuštění pouze pro desku PES-RPP
if target == 'tms570_rpp':
 # rozšíření kroků pro linuxový target
 steps_linux += [

  # zkopírování binárky do /tmp adresáře BBB
  # +spuštění skriptu, který vykoná nahraní do FLASH RPP
  ShellCommand(command="
   scp $(make print-release-basename)/rpp-test-sw/Debug/rpp-test-sw.out
   debian@rpp_bbb_tester:/tmp &&
   ssh root@rpp_bbb_tester /opt/rpp-tester/software/buildbot/run-flash
   /tmp/rpp-test-sw.out",
   description="flash"), #, haltOnFailure=True),

  # spuštění testů v RPP-Testeru
  ShellCommand(command="
   ssh root@rpp_bbb_tester /opt/rpp-tester/software/buildbot/run-tests",
   description='hw tests')
  ]
```



Obrázek 4.1. Výsledky úloh provedené Buildbotem.

4.3.2 OpenOCD

BeagleBone Black je s RPP deskou propojen kromě sériové linky i JTAG rozhraním. To umožňuje přes program OpenOCD nahrávat binárky do FLASH paměti RPP. Níže je konfigurační soubor upravený pro desku TMS570.

```
# openOCD konfigurační soubor config.cfg
adapter_khz 1500
source [find interface/ftdi/xds100v2.cfg]
transport select jtag
source [find target/ti_tms570.cfg]
reset_config trst_only
init; ftdi_set_signal PWR_RST 1; jtag arp_init

reset halt
wait_halt
bp 0x00000000 4 hw
reset halt
wait_halt
```

K nahrání nové verze binárky slouží skript *start-flash* v adresáři */opt/RPP-Tester/software/openocd*. Jeho zkrácená ukázka je uvedena níže. Kód spouští program openocd s konfiguračním skriptem *config.cfg* a v parametru *-c* jsou příkazy, které se mají vykonat.

```
# skript start-flash
cmd="debug_level 3 # režim výpisu událostí
flash probe 0      # validace parametrů FLASH
program $1         # nahrání binárky ze souboru uvedeném v argumentu
rbp 0x00000000     # odebrání breakpointu z adresy 0
reset run"         # spuštění programu

openocd -f config.cfg -c "$cmd"
```

V průběhu testování ovšem docházelo k chybám při zápisu do RPP FLASH paměti. Problém se bohužel zatím nepodařilo vyřešit. Náhradou za OpenOCD je nástroj Code Composer Studio `http://www.ti.com/tool/ccstudio` od firmy Texas Instruments. Dočasným řešením tedy může být použití dalšího PC, které nahraje novou verzi binárky přes CCS.

Do budoucna se plánuje používání Ethernetového bootloaderu, který je momentálně ve vývoji.

Kapitola 5
Funkční testy periferií

Součástí zadání práce bylo i vypracování testů, které mají za úkol prověřit funkčnost jednotlivých periferií. Mnou napsané testy testují základní parametry a dají se v budoucnu snadno rozšířit. Konvence tvorby testů je následující:

- Název souboru, ve kterém je testovací scénář umístěn, obsahuje název testované RPP periferie.
- Každý soubor obsahuje testovací scénář, kterým je třída s názvem začínajícím na *Test* a následně jméno periferie (např. TestAdc).
- Jednotlivé metody testovacího scénáře začínají názvem *test_*.
- Pro testování periferií s více komponentami se v cyklu využívá *AssertsCollector*.

Následující sekce popisují jednotlivé testy.

5.1 ADC

Test pro RPP analog-digital převodník vždy nejprve nastaví na výstupní pin DAC Testeru požadované napětí/hodnotu a následně čte napětí/hodnotu zaznamenanou RPP deskou. Porovnání probíhá v intervalu, který je nastaven na cca 3% možnost odchylky.

```
class TestAdc:
 def test_value_raw(self):
  asserts_collector = AssertsCollector()

  tester_value = 1965
  allowed_diff = 50
  for adc_id in range(1, 13):
   tester_adc.set_value_raw(adc_id, tester_value)
   rpp_value = rpp_adc.get_value_raw(adc_id)
   asserts_collector.add(RangeAssertRppInput(
    'ADC {:2}'.format(adc_id),
    rpp_value,
    tester_value - allowed_diff,
    tester_value + allowed_diff
   ))

  asserts_collector.check()

 def test_value_voltage(self):
  asserts_collector = AssertsCollector()

  tester_value = 12000
  allowed_diff = 300
  for adc_id in range(1, 13):
```

```
    tester_adc.set_value_voltage(adc_id, tester_value)
    rpp_value = rpp_adc.get_value_voltage(adc_id)
    asserts_collector.add(RangeAssertRppInput(
     'ADC {:2}'.format(adc_id),
     rpp_value,
     tester_value - allowed_diff,
     tester_value + allowed_diff
    ))

   asserts_collector.check()
```

5.2 DAC

Test RPP digital-analog převodníku spočívá v inicializaci a nastavení RPP DAC periferie a následném přečtení napětí na ADC Testeru. Opět je dovolena tolerance v zadaném intervalu.

```
class TestDac:
 def test_value_raw(self):
  asserts_collector = AssertsCollector()

  rpp_value = 1000
  allowed_diff = 10
  for dac_id in range(1, 5):
   rpp_dac.pin_enable(dac_id, 1)
   rpp_dac.set_value_raw(dac_id, rpp_value)
   tester_value = tester_dac.get_value_raw(dac_id)
   asserts_collector.add(RangeAssertRppOutput(
    'DAC {:2}'.format(dac_id),
    tester_value,
    rpp_value - allowed_diff,
    rpp_value + allowed_diff
   ))
  asserts_collector.check()

 def test_value_voltage(self):
  asserts_collector = AssertsCollector()

  rpp_value = 10000
  allowed_diff = 200
  for dac_id in range(1, 5):
   rpp_dac.pin_enable(dac_id, 1)
   rpp_dac.set_value_voltage(dac_id, rpp_value)
   tester_value = tester_dac.get_value_voltage(dac_id)
   asserts_collector.add(RangeAssertRppOutput(
    'DAC {:2}'.format(dac_id),
    tester_value,
    rpp_value - allowed_diff,
    rpp_value + allowed_diff
   ))
  asserts_collector.check()
```

5.3 DIN

Testovací scénář RPP DIN periferie prověřuje pouze část statických parametrů. Např. generování přerušení není zahrnuto. Stejně tak test s použitím aktivního pull-up/down, jelikož k tomu tester neobsahuje požadovanou hardwarovou podporu (viz. obr. 3.4). Měření je tedy omezeno na RPP vstupy (případně Tester výstupy) ve stavu vysoké impedance.

```
class TestDin:
 def test_din0_7_programmable(self):
  asserts_collector = AssertsCollector()

  # Nastavit pull-up, active, zkontrolovat, že ctu 0
  tester_din.set_high_impedance()
  expected_value = 0
  for din_id in range(0, 8):
   rpp_din.setup(din_id, rpp_din.PULL_UP, rpp_din.ACTIVE,
   rpp_din.IRQ_DISABLED)
   rpp_value = rpp_din.get_value(din_id)
   asserts_collector.add(EqualAssertRppInput(
    'DIN {} pull-up, active (high impedance)'.format(din_id),
    rpp_value,
    expected_value
   ))

  # Nastavit pull-down, active, zkontrolovat, že ctu 0
  tester_din.set_high_impedance()
  expected_value = 0
  for din_id in range(0, 8):
   rpp_din.setup(din_id, rpp_din.PULL_DOWN, rpp_din.ACTIVE,
     rpp_din.IRQ_DISABLED)
   rpp_value = rpp_din.get_value(din_id)
   asserts_collector.add(EqualAssertRppInput(
    'DIN {} pull-down, active (high impedance)'.format(din_id),
    rpp_value,
    expected_value
   ))

  # Nastavit pull-down, tri-state, otestovat, že výstup z Testeru se
  # objeví na DIN vstupu
  # test DIN to GND
  tester_value = 0
  for din_id in range(0, 8):
   rpp_din.setup(din_id, rpp_din.PULL_DOWN, rpp_din.TRI_STATE,
     rpp_din.IRQ_DISABLED)
   tester_din.set_value(din_id, tester_value)
   rpp_value = rpp_din.get_value(din_id)
   asserts_collector.add(EqualAssertRppInput(
    'DIN {} pull-down, tri-state (GND input)'.format(din_id),
    rpp_value,
    tester_value
   ))
```

```
    # test DIN to 12V
    tester_value = 1
    for din_id in range(0, 8):
     rpp_din.setup(din_id, rpp_din.PULL_DOWN, rpp_din.TRI_STATE,
       rpp_din.IRQ_DISABLED)
     tester_din.set_value(din_id, tester_value)
     rpp_value = rpp_din.get_value(din_id)
     asserts_collector.add(EqualAssertRppInput(
      'DIN {} pull-down, tri-state (12V input)'.format(din_id),
      rpp_value,
      tester_value
     ))

    asserts_collector.check()

   def test_din8_15_gnd(self):
    asserts_collector = AssertsCollector()

    # Nastavit pull-up, active, zkontrolovat, že ctu 0
    tester_din.set_high_impedance()
    expected_value = 0
    for din_id in range(8, 16):
     rpp_din.setup(din_id, rpp_din.PULL_UP, rpp_din.ACTIVE,
       rpp_din.IRQ_DISABLED)
     rpp_value = rpp_din.get_value(din_id)
     asserts_collector.add(EqualAssertRppInput(
      'DIN {} pull-up, active (high impedance)'.format(din_id),
      rpp_value,
      expected_value
     ))

    # Nastavit pull-up, tri-state, zkontrolovat, že ctu 1
    tester_din.set_high_impedance()
    expected_value = 1
    for din_id in range(8, 16):
     rpp_din.setup(din_id, rpp_din.PULL_UP, rpp_din.TRI_STATE,
       rpp_din.IRQ_DISABLED)
     rpp_value = rpp_din.get_value(din_id)
     asserts_collector.add(EqualAssertRppInput(
      'DIN {} pull-up, tri-state (high impedance)'.format(din_id),
      rpp_value,
      expected_value
     ))

    asserts_collector.check()
```

5.4 CAN

Test řadičů RPP CAN obsahuje metody ověřující správné odesílání a příjem zpráv po CAN sběrnici. Obsah i identifikátor zprávy jsou generovány a porovnává se vždy přesná shoda všech položek (*arbitration_id*, *length*, *message*).

```
class TestCan:
 def test_send(self):
  asserts_collector = AssertsCollector()

  rpp_messages_count = 10

  rpp_can.init()

  for can_id in (1, 2, 3):
   rpp_messages =
   [[random.randint(0, 255) for i in range(random.randint(0, 8))]
       for x in range(rpp_messages_count)]
   rpp_arbitration_ids =
   [random.randint(20, 255) for x in range(rpp_messages_count)]

   tester_can.start_dump(can_id)
   for i in range(rpp_messages_count):
    rpp_can.send(can_id, rpp_arbitration_ids[i], rpp_messages[i])
   tester_messages = tester_can.stop_dump(can_id)

   if asserts_collector.add(EqualAssertRppOutput(
     'CAN {} count of messages'.format(can_id),
     len(tester_messages),
     rpp_messages_count
   )):
    continue

   for i in range(rpp_messages_count):
    asserts_collector.add(EqualAssertRppOutput(
     'CAN {} message data'.format(can_id),
     tester_messages[i]['data'],
     rpp_messages[i]
    ))

    asserts_collector.add(EqualAssertRppOutput(
     'CAN {} message arbitration_id'.format(can_id),
     tester_messages[i]['arbitration_id'],
     rpp_arbitration_ids[i]
    ))

  asserts_collector.check()

 def test_dump(self):
  asserts_collector = AssertsCollector()

  tester_messages_count = 10
```

```
rpp_can.init()

for can_id in (1, 2, 3):
 tester_messages =
  [[random.randint(0, 255) for i in range(random.randint(0, 8))]
        for x in range(tester_messages_count)]
 tester_arbitration_ids =
  [random.randint(20, 255) for x in range(tester_messages_count)]

 rpp_can.start_dump(can_id)
 for i in range(tester_messages_count):
  tester_can.
   send(can_id, tester_arbitration_ids[i], tester_messages[i])
 rpp_messages = rpp_can.stop_dump()

 if asserts_collector.add(EqualAssertRppInput(
   'CAN {} count of messages'.format(can_id),
   len(rpp_messages),
   tester_messages_count
 )):
  continue

 for i in range(tester_messages_count):
  asserts_collector.add(EqualAssertRppInput(
   'CAN {} message data'.format(can_id),
   rpp_messages[i]['data'],
   tester_messages[i]
  ))

  asserts_collector.add(EqualAssertRppInput(
   'CAN {} message arbitration_id'.format(can_id),
   rpp_messages[i]['arbitration_id'],
   tester_arbitration_ids[i]
  ))

asserts_collector.check()
```

5.5 HBR

Test RPP HBR prověřuje, jestli jsou PWM signály generovány pouze na jednom z vodičů a zároveň jejich parametry odpovídají konfiguraci. Jsou použity tři délky periody (300, 1000 a 10000 μs) a celý rozsah střídy (po 10%). Pokud změřená průměrná hodnota periody a pozitivní/negativní střídy překročí o 10% svůj nastavený základ, končí test chybou.

```
class TestHbr:
 def measure_pwm(self, rpp_period, rpp_speed):
  asserts_collector = AssertsCollector()

  if rpp_speed > 0:
   active_ecap = 'ecap2'
   pasive_ecap = 'ecap0'
  elif rpp_speed < 0:
   active_ecap = 'ecap0'
   pasive_ecap = 'ecap2'
  else:
   raise ValueError('rpp_speed cant be zero')

  rpp_pos_duty = rpp_period * (abs(rpp_speed) / 100)
  rpp_neg_duty = rpp_period * ((100 - abs(rpp_speed)) / 100)

  try:
   rpp_hbr.disable()
  except Exception:
   pass

  rpp_hbr.enable(rpp_period)
  rpp_hbr.control(rpp_speed)

  results = None
  # 10 pokusu pro planovac, snad to aspon jednou vyjde
  for i in range(10):
   try:
    results = tester_hbr.measure_pwm()
    if results['ecap0']['maxMeasureDiff'] < rpp_period and
     results['ecap2']['maxMeasureDiff'] < rpp_period:
     break
   except ecap.Error as e:
    raise InfoError('HBR', e.__str__())

  if results is None:
   raise InfoError('HBR', 'Can\'t get valid results')

  rpp_hbr.disable()

  asserts_collector.add(EqualAssertRppOutput(
   'HBR other signal pulses',
   results[pasive_ecap]['measuresCount'],
   0
  ))
```

```
 asserts_collector.add(RangeAssertRppOutput(
  'HBR period',
  results[active_ecap]['periodAvg'],
  rpp_period * 0.9,
  rpp_period * 1.1
 ))

 asserts_collector.add(RangeAssertRppOutput(
  'HBR positive duty',
  results[active_ecap]['dutyPosAvg'],
  rpp_pos_duty * 0.9,
  rpp_pos_duty * 1.1
 ))

 asserts_collector.add(RangeAssertRppOutput(
  'HBR negative duty',
  results[active_ecap]['dutyNegAvg'],
  rpp_neg_duty * 0.9,
  rpp_neg_duty * 1.1
 ))

 asserts_collector.check()

def test_pwm_plus(self):
 for rpp_period in (300, 1000, 10000):
  for rpp_speed in range(10, 99, 10):
   self.measure_pwm(rpp_period, rpp_speed)

def test_pwm_minus(self):
 for rpp_period in (300, 1000, 10000):
  for rpp_speed in range(10, 99, 10):
   self.measure_pwm(rpp_period, -rpp_speed)

def test_pwm_zero(self):
 rpp_period = 1000
 rpp_speed = 0

 rpp_hbr.enable(rpp_period)
 rpp_hbr.control(rpp_speed)

 with pytest.raises(ecap.Error) as e:
  tester_hbr.measure_pwm()

 if 'MeasureError: Zero results' != str(e.value):
  raise InfoError('HBR', 'Some pulses were detected')
```

5.6 HOUT

Scénář pro RPP HOUT je ovlivněn vlastnostmi obvodu RPP desky. Jelikož při většině testů končily pokusy o zapnutí PWM chybou čipu, je nastavena pouze jedna pevná perioda 10000 μs se střídou 40:60. Ukázalo se, že v tomto nastavení generuje obvod nejméně chyb a je možné ho alespoň částečně otestovat. Toto řešení není ideální, ale pouze dočasně řeší problém. Spuštění každého obvodu je v případě neúspěchu zkoušeno opakovaně, dokud obvod nevrací status *OK*, maximálně však desetkrát. Při úspěšné inicializaci test porovnává stejné hodnoty jako u testu HBR.

```
class TestHout:
 def test_pwm(self):
  asserts_collector = AssertsCollector()

  # stop all
  for hout_id in range(1, 7):
   rpp_hout.stop_pwm(hout_id)

  time.sleep(1)

  rpp_period = 10000
  rpp_duty = 40
  rpp_pos_duty = rpp_period * (rpp_duty / 100)
  rpp_neg_duty = rpp_period * ((100 - rpp_duty) / 100)

  allowed_diff = 100

  for hout_id in range(1, 7):
   rpp_hout.set_pwm(hout_id, rpp_period, rpp_duty)
   time.sleep(1.0 / 100)

   # 10 pokusu o nahozeni pwm
   status = 'UNKNOWN'
   for i in range(10):
    rpp_hout.start_pwm(hout_id)
    time.sleep(1.0 / 100)
    status = rpp_hout.get_fail(hout_id)
    if status == 'OK':
     break
    else:
     rpp_hout.stop_pwm(hout_id)
     time.sleep(1.0 / 20)

   if status != 'OK':
    asserts_collector.add(EqualAssertRppOutput(
     'HOUT {} status'.format(hout_id),
     status,
     'OK'
    ))
    continue

   result = None
```

```
    # 10 pokusu pro planovac, snad to aspon jednou vyjde
    for i in range(10):
     try:
      result = tester_hout.measure_pwm(hout_id)
      if result['maxMeasureDiff'] < rpp_period:
       break
     except ecap.Error as e:
      asserts_collector.add(InfoError(hout_id, e.__str__()))

    if result is None:
     asserts_collector.add(InfoError(hout_id, 'Can\'t get valid result'))
     continue

    # period
    asserts_collector.add(RangeAssertRppOutput(
     'HOUT {} period'.format(hout_id),
     result['periodAvg'],
     rpp_period - allowed_diff,
     rpp_period + allowed_diff
    ))

    # positive duty
    asserts_collector.add(RangeAssertRppOutput(
     'HOUT {} positive duty'.format(hout_id),
     result['dutyPosAvg'],
     rpp_pos_duty - (allowed_diff / 2),
     rpp_pos_duty + (allowed_diff / 2)
    ))

    # negative duty
    asserts_collector.add(RangeAssertRppOutput(
     'HOUT {} negative duty'.format(hout_id),
     result['dutyNegAvg'],
     rpp_neg_duty - (allowed_diff / 2),
     rpp_neg_duty + (allowed_diff / 2)
    ))

    # turn off pwm
    rpp_hout.stop_pwm(hout_id)

   asserts_collector.check()
```

5.7 LOUT

Test RPP LOUT ověřuje shodu nastavené hodnoty Testeru proti údaji přečteným RPP deskou. Jedná se pouze o binární výsledky (0 nebo 1).

```
class TestLout:
 def test_value(self):
  asserts_collector = AssertsCollector()

  rpp_value = 1
  for lout_id in range(1, 9):
   rpp_lout.set_value(lout_id, rpp_value)
   tester_value = tester_lout.get_value(lout_id)
   asserts_collector.add(EqualAssertRppOutput(
    'LOUT {:2}'.format(lout_id),
    tester_value,
    rpp_value
   ))

  rpp_value = 0
  for lout_id in range(1, 9):
   rpp_lout.set_value(lout_id, rpp_value)
   tester_value = tester_lout.get_value(lout_id)
   asserts_collector.add(EqualAssertRppOutput(
    'LOUT {:2}'.format(lout_id),
    tester_value,
    rpp_value
   ))

  asserts_collector.check()
```

5.8 MOUT

Jelikož princip RPP MOUT je zcela identický s LOUT periferií, jsou podobné i testy.

```
class TestMout:
 def test_value(self):
  asserts_collector = AssertsCollector()

  rpp_value = 1
  for mout_id in range(1, 7):
   rpp_mout.set_value(mout_id, rpp_value)
   tester_value = tester_mout.get_value(mout_id)
   asserts_collector.add(EqualAssertRppOutput(
    'MOUT {:2}'.format(mout_id),
    tester_value,
    rpp_value
   ))

  rpp_value = 0
  for mout_id in range(1, 7):
   rpp_mout.set_value(mout_id, rpp_value)
   tester_value = tester_mout.get_value(mout_id)
   asserts_collector.add(EqualAssertRppOutput(
    'MOUT {:2}'.format(mout_id),
    tester_value,
    rpp_value
   ))

  asserts_collector.check()
```

Kapitola 6
Výsledky testů

V této kapitole je uveden ukázkový report generovaný RPP-Testerem s použitím nástroje *py.test* a souhrnné výsledky proběhlých testů na všech aktuálně dostupných RPP deskách, které katedra řídicí techniky vlastní.

6.1 Ukázkový report

Tato sekce podrobně popisuje na ukázkovém reportu, jak je uživatel zpraven o výsledku testů. Výsledek každého testování je uložen do textového souboru, který v názvu obsahuje datum a čas provedení testů.

6.1.1 Přehled a verze

Ve výpisu je nejprve uveden přehled testovaných scénářů s informací o výsledku testu (Znak „.“ znamená bezchybný průběh a znak „F“ označuje nález chyby) a nahrané verzi RPP firmwaru. Z těchto informací lze snadno rychle vyvodit zda je deska v pořádku. V případě nalezení chyb se informace o nich objeví v sekci *FAILURES*.

```
============================= test session starts ====================
platform linux -- Python 3.4.2, pytest-2.9.1, py-1.4.31, pluggy-0.3.1
rootdir: /opt/rpp-tester/software/python, inifile:
collected 15 items

tests/integration/test_adc.py FF
tests/integration/test_can.py ..
tests/integration/test_dac.py FF
tests/integration/test_din.py FF
tests/integration/test_hbr.py FFF
tests/integration/test_hout.py F
tests/integration/test_lout.py F
tests/integration/test_mout.py F
tests/integration/test_x_version.py

================================ RPP VERSION ============================
version=eaton-0.7-17-gee9f11e
=========================================================================

================================== FAILURES =============================
...
```

6.1.2 ADC

Konvence formátu výpisu všech chyb je taková, že je vždy uveden název testovacího scénáře a za tečkou pak název samotného testu. Následuje výpis chyb, které test zaznamenal, v uživatelsky čitelné podobě. Na ukázce níže je výpis testu *test_value_raw* ze scénáře *TestAdc*. RPP deska zde četla na AD převodnících 1, 2, 3, 4, 5 a 10 „surové“ hodnoty, které neležely v povoleném rozsahu 1915–2015.

```
_____________________________ TestAdc.test_value_raw ____________________
tests/integration/test_adc.py:22: in test_value_raw
    asserts_collector.check()
    raise self
E   rpp_tester.asserts.AssertsCollector:
E   RangeAssertError: ADC  1 - read by RPP: 91,
expected in range: 1915-2015
E   RangeAssertError: ADC  2 - read by RPP: 89,
expected in range: 1915-2015
E   RangeAssertError: ADC  3 - read by RPP: 89,
expected in range: 1915-2015
E   RangeAssertError: ADC  4 - read by RPP: 89,
expected in range: 1915-2015
E   RangeAssertError: ADC  5 - read by RPP: 89,
expected in range: 1915-2015
E   RangeAssertError: ADC 10 - read by RPP: 4095,
expected in range: 1915-2015
```

Podobné chyby lze vidět i u testu *test_value_voltage*. Jedná se o stejné submoduly RPP ADC a přečtená hodnota opět neleží v zadaném intervalu.

```
___________________________ TestAdc.test_value_voltage ___________________
tests/integration/test_adc.py:39: in test_value_voltage
    asserts_collector.check()
    raise self
E   rpp_tester.asserts.AssertsCollector:
E   RangeAssertError: ADC  1 - read by RPP: 560,
expected in range: 11700-12300
E   RangeAssertError: ADC  2 - read by RPP: 540,
expected in range: 11700-12300
E   RangeAssertError: ADC  3 - read by RPP: 530,
expected in range: 11700-12300
E   RangeAssertError: ADC  4 - read by RPP: 560,
expected in range: 11700-12300
E   RangeAssertError: ADC  5 - read by RPP: 540,
expected in range: 11700-12300
E   RangeAssertError: ADC 10 - read by RPP: 25000,
expected in range: 11700-12300
```

6.1.3 DAC

Výpis testu RPP DAC se velmi podobá RPP ADC s tím rozdílem, že zde ve výpisu chyby není uvedeno *read by RPP*, ale naopak *read by Tester*, jelikož DAC je na rozdíl od ADC výstupní periferie.

```
_____________________________ TestDac.test_value_raw ____________________
tests/integration/test_dac.py:23: in test_value_raw
    asserts_collector.check()
    raise self
E   rpp_tester.asserts.AssertsCollector:
E   RangeAssertError: DAC  1 - read by Tester: 0,
expected in range: 990-1010
E   RangeAssertError: DAC  2 - read by Tester: 0,
expected in range: 990-1010
E   RangeAssertError: DAC  3 - read by Tester: 0,
expected in range: 990-1010
E   RangeAssertError: DAC  4 - read by Tester: 0,
expected in range: 990-1010

____________________________ TestDac.test_value_voltage ___________________
tests/integration/test_dac.py:41: in test_value_voltage
    asserts_collector.check()
    raise self
E   rpp_tester.asserts.AssertsCollector:
E   RangeAssertError: DAC  1 - read by Tester: 0,
expected in range: 9800-10200
E   RangeAssertError: DAC  2 - read by Tester: 0,
expected in range: 9800-10200
E   RangeAssertError: DAC  3 - read by Tester: 0,
expected in range: 9800-10200
E   RangeAssertError: DAC  4 - read by Tester: 0,
expected in range: 9800-10200
```

6.1.4 DIN

Hodnoty lze testovat kromě intervalu i pouze binárně. Zde je uveden test RPP DIN při konfiguraci pinů *spínání baterie (pull-down)*, *vysokoimpedanční vstup(tri-state)* a 12 V vstupním napětím na pinu. Očekávaný výsledek přečtený RPP deskou byl 1, nicméně z důvodu poruchy byla přečtena hodnota 0.

```
_____________________ TestDin.test_din0_7_programmable _______________
tests/integration/test_din.py:59: in test_din0_7_programmable
    asserts_collector.check()
    raise self
E   rpp_tester.asserts.AssertsCollector:
E   EqualAssertError: DIN 0 pull-down, tri-state (12V input) -
read by RPP: 0, expected: 1
E   EqualAssertError: DIN 1 pull-down, tri-state (12V input) -
read by RPP: 0, expected: 1
E   EqualAssertError: DIN 2 pull-down, tri-state (12V input) -
read by RPP: 0, expected: 1
E   EqualAssertError: DIN 3 pull-down, tri-state (12V input) -
read by RPP: 0, expected: 1
E   EqualAssertError: DIN 4 pull-down, tri-state (12V input) -
read by RPP: 0, expected: 1
E   EqualAssertError: DIN 5 pull-down, tri-state (12V input) -
read by RPP: 0, expected: 1
E   EqualAssertError: DIN 6 pull-down, tri-state (12V input) -
read by RPP: 0, expected: 1
```

Zde je obdobný test jako výše, pouze je nahrazeno vstupní napětí 12 V za stav vysoké impedance.

```
_____________________________ TestDin.test_din8_15_gnd ______________________
tests/integration/test_din.py:88: in test_din8_15_gnd
    asserts_collector.check()
    raise self
E   rpp_tester.asserts.AssertsCollector:
E   EqualAssertError: DIN 8 pull-up, tri-state (high impedance) -
read by RPP: 0, expected: 1
E   EqualAssertError: DIN 9 pull-up, tri-state (high impedance) -
read by RPP: 0, expected: 1
E   EqualAssertError: DIN 10 pull-up, tri-state (high impedance) -
read by RPP: 0, expected: 1
E   EqualAssertError: DIN 11 pull-up, tri-state (high impedance) -
read by RPP: 0, expected: 1
E   EqualAssertError: DIN 12 pull-up, tri-state (high impedance) -
read by RPP: 0, expected: 1
E   EqualAssertError: DIN 13 pull-up, tri-state (high impedance) -
read by RPP: 0, expected: 1
E   EqualAssertError: DIN 14 pull-up, tri-state (high impedance) -
read by RPP: 0, expected: 1
E   EqualAssertError: DIN 15 pull-up, tri-state (high impedance) -
read by RPP: 0, expected: 1
```

6.1.5 HBR

Test H-můstku je oproti ostatním unikátní tím, že periferie je na RPP jediná. Tzn. že se neprovádí cyklus přes více subsystémů, ale pouze přes jeden. Níže je příklad výpisu chyby, kdy nebyl zaznamenán žádný impuls generovaný RPP deskou z důvodu nefunkčnosti obvodu. Ten vrátil informativní upozornění *Enable procedure failed.*

```
_____________________________ TestHbr.test_pwm_plus ____________________
tests/integration/test_hbr.py:79: in test_pwm_plus
    self.measure_pwm(rpp_period, rpp_speed)
tests/integration/test_hbr.py:40: in measure_pwm
    raise InfoError('HBR', e.__str__())
E   rpp_tester.asserts.InfoError: ('HBR', 'MeasureError: Zero results')
_____________________________ TestHbr.test_pwm_minus ___________________
tests/integration/test_hbr.py:84: in test_pwm_minus
    self.measure_pwm(rpp_period, -rpp_speed)
tests/integration/test_hbr.py:40: in measure_pwm
    raise InfoError('HBR', e.__str__())
E   rpp_tester.asserts.InfoError: ('HBR', 'MeasureError: Zero results')
_____________________________ TestHbr.test_pwm_zero ____________________
tests/integration/test_hbr.py:90: in test_pwm_zero
    rpp_hbr.enable(rpp_period)
    raise RuntimeError('Invalid console result:
     {:s}'.format(repr(cmd_result)))
E   RuntimeError: Invalid console result: 'Enable procedure failed.'
```

6.1.6 HOUT

Jak už bylo v předchozích kapitolách zmíněno, RPP HOUT při spouštění často generuje chybu při inicializaci. Zde je ukázka, kde obvody 1, 2 a 6 nebyly schopny přejít do validního stavu a i na 10 pokusů vždy vrátily status *FAIL*.

```
_____________________________ TestHout.test_pwm _____________________
tests/integration/test_hout.py:90: in test_pwm
    asserts_collector.check()
    raise self
E   rpp_tester.asserts.AssertsCollector:
E   EqualAssertError: HOUT 1 status - read by Tester: FAIL,
expected: OK
E   EqualAssertError: HOUT 2 status - read by Tester: FAIL,
expected: OK
E   EqualAssertError: HOUT 6 status - read by Tester: FAIL,
expected: OK
```

6.1.7 LOUT

Častou chybou RPP LOUT je čtení logické 0 za všech nastavení. Zde je ukázka, kdy všechny piny LOUT čtou nízkou úroveň napětí. Může to být způsobeno např. špatnými kontakty.

```
_____________________________ TestLout.test_value ____________________
tests/integration/test_lout.py:30: in test_value
    asserts_collector.check()
    raise self
E   rpp_tester.asserts.AssertsCollector:
E   EqualAssertError: LOUT  1 - read by Tester: 0, expected: 1
E   EqualAssertError: LOUT  2 - read by Tester: 0, expected: 1
E   EqualAssertError: LOUT  3 - read by Tester: 0, expected: 1
E   EqualAssertError: LOUT  4 - read by Tester: 0, expected: 1
E   EqualAssertError: LOUT  5 - read by Tester: 0, expected: 1
E   EqualAssertError: LOUT  6 - read by Tester: 0, expected: 1
E   EqualAssertError: LOUT  7 - read by Tester: 0, expected: 1
E   EqualAssertError: LOUT  8 - read by Tester: 0, expected: 1
```

6.1.8 MOUT

Níže je výsledek testu RPP MOUT, který dopadl obdobně jako LOUT. Všechny porty opět čtou logickou 0.

```
_____________________________ TestMout.test_value ____________________
tests/integration/test_mout.py:30: in test_value
    asserts_collector.check()
    raise self
E   rpp_tester.asserts.AssertsCollector:
E   EqualAssertError: MOUT  1 - read by Tester: 0, expected: 1
E   EqualAssertError: MOUT  2 - read by Tester: 0, expected: 1
E   EqualAssertError: MOUT  3 - read by Tester: 0, expected: 1
E   EqualAssertError: MOUT  4 - read by Tester: 0, expected: 1
E   EqualAssertError: MOUT  5 - read by Tester: 0, expected: 1
E   EqualAssertError: MOUT  6 - read by Tester: 0, expected: 1
```

6.1.9 Shrnutí

Na konci reportu je vždy uveden souhrn, který informuje o:

- počet testů, které skončily chybou (zde 12)
- počet testů, které skončily bez chyby (zde 3)
- celkový čas běhu všech testů (zde 19 sekund)

```
==================== 12 failed, 3 passed in 19.51 seconds ============
```

6.2 Tabulka RPP desek a výsledků

Měření každé desky byla prováděna pouze jednou v případě, že tester nezaznamenal žádnou chybu. V opačném případě jsem testy spouštěl opakovaně, abych vyloučil chybu testovacího frameworku nebo testovací desky. K tomuto kroku jsem se rozhodl i přesto, že jsem všechny vlastnosti a chování testovacího nástoje během vývoje opakovaně ověřoval i ručním měřením a simulacemi možných situací.

RPP periferie	výsledek	info
ADC	OK	
DAC	OK	
DIN	OK	
CAN	OK	
HBR	OK	
HOUT	OK	
LOUT	OK	
MOUT	OK	

Tabulka 6.1. Výsledky testů desky č. 5767

RPP periferie	výsledek	info
ADC	OK	
DAC	FAIL	DAC 3 generuje vždy 0 V.
DIN	OK	
CAN	OK	
HBR	OK	
HOUT	FAIL	HOUT 1, 2, 6 chyba inicializace.
LOUT	FAIL	LOUT 4 generuje vždy 0 V.
MOUT	OK	

Tabulka 6.2. Výsledky testů desky č. 5768

RPP periferie	výsledek	info
FLASH	FAIL	Chyba při zápisu.

Tabulka 6.4. Výsledky testů desky č. 5771

RPP periferie	výsledek	info
ADC	OK	
DAC	OK	
DIN	OK	
CAN	OK	
HBR	OK	
HOUT	FAIL	OUT 1, 2, 3, 5 chyba inicializace.
LOUT	OK	
MOUT	OK	

Tabulka 6.3. Výsledky testů desky č. 5769

RPP periferie	výsledek	info
ADC	FAIL	ADC 1–5 čte cca 500 mV místo 12000 mV. ADC 10 čte 25000 mV místo 12000 mV.
DAC	FAIL	Všechny DAC generují vždy 0 V.
DIN	FAIL	Všechny DIN čtou vždy 0 V.
CAN	OK	
HBR	FAIL	Chyba inicializace.
HOUT	FAIL	HOUT 1, 2, 6 chyba inicializace.
LOUT	FAIL	Všechny LOUT generují vždy 0 V.
MOUT	FAIL	Všechny MOUT generují vždy 0 V.

Tabulka 6.5. Výsledky testů desky č. 5772

RPP periferie	výsledek	info
FLASH	FAIL	Chyba při zápisu.

Tabulka 6.6. Výsledky testů desky č. 5773

Kapitola 7
Závěr

V této práci jsem se zabýval návrhem a implementací desky pro testování automobilových řídicích jednotek z projektu PES-RPP. Navrhl jsem elektronická schémata a desku plošných spojů, která funguje jako rozšíření pro vývojový nástroj BeagleBone Black (procesor AM335x – ARM Cortex-A8). Na něm je spuštěn operační systém Debian 8.4 Jessie s linuxovým jádrem 4.4.6-bone6. K využití všech periferií byl vytvořen speciální device tree. Pro potřeby testování jsem vytvořil v jazyce Python framework s názvem RPP-Tester, který umožňuje ovládání HW komponent testovací desky a zároveň je schopný komunikovat prostřednictvím sériové linky s RPP deskou.

Na základě zadání a přídavných požadavků jsem začlenil podporu pro propojení s automatizovanou průběžnou integrací (buildbot). Díky této vlastnosti je tester schopný přijímat zkompilované binární soubory, které nahraje do FLASH paměti RPP desky (s využitím OpenOCD). Na této nové verzi programu následně spustí připravené testovací scénáře.

V rámci této práce nebyla v souladu se zadáním implementována softwarová podpora pro LIN a FlexRay periferie. Nicméně hardware je pro tyto funkce připraven. Pro LIN je to napojení budičů na interní UART subsystémy a v případě FlexRay jsou piny z budičů přivedeny na vnitřní vstupně-výstupní porty PRU-ICSS.

Na závěr byl tester začleněn do testovacího buildbotu projektu a může tak poskytovat vývojářům kontrolu funkčnosti jejich změn kódu přímo na hardwaru. Během testování ovšem vyšlo najevo, že nahrávání binárního souboru do FLASH paměti desky programem OpenOCD končí v některých případech chybou. Tento problém se v rámci této práce nepodařilo vyřešit. Tato funkcionalita ovšem nebyla součástí zadání a existuje náhradní řešení v podobě použití nástoje od firmy Texas Instruments. Značnou nevýhodou tohoto postupu je ale nutná přítomnost dalšího elementu (PC), který se podílí na testovací úloze.

Literatura

[1] BeagleBoard. *BeagleBone: open-hardware expandable computer*. 2016.
http://beagleboard.org/support/bone101.

[2] element14. *BeagleBone Black – System Reference Manual*. 2014.
https://www.element14.com/community/servlet/JiveServlet/downloadBody/54165-102-6-291579/e14%20BBB_SRM_rev%200.9.pdf.

[3] Wikipedia. *Das U-Boot*. 2014.
https://cs.wikipedia.org/wiki/Das_U-Boot.

[4] Beyond Logic. *BeagleBoneBlack Boot Process*. 2016.
http://wiki.beyondlogic.org/index.php/BeagleBoneBlack_Boot_Process.

[5] Texas Instruments. *AM335x U-Boot User's Guide*. 2016.
http://processors.wiki.ti.com/index.php/AM335x_U-Boot_User's_Guide.

[6] Texas Instruments. *Boot Sequence*. 2012.
http://processors.wiki.ti.com/index.php/Boot_Sequence.

[7] Texas Instruments. *PRU-ICSS*. 2016.
http://processors.wiki.ti.com/index.php/PRU-ICSS.

[8] Texas Instruments. *AM335x Sitara Processorss*. 2016.
http://www.ti.com/lit/ds/sprs717j/sprs717j.pdf.

[9] OpenOCD. *OpenOCD – About*. 2016.
http://openocd.org/doc/html/About.html.

[10] Thomas Petazzoni. *Device Tree for dummies*. 2013.
https://events.linuxfoundation.org/sites/events/files/slides/petazzoni-device-tree-dummies.pdf.

[11] FDTWiki – Grant Likely. *Device Tree – Main Page*. 2014.
http://www.devicetree.org/Main_Page.

[12] Buildbot. *Buildbot – The Continuous Integration Framework*. 2016.
http://buildbot.net/.

[13] Kicad. *Kicad Documentation*. 2016.
http://kicad-pcb.org/help/documentation.

Příloha A
Zadání

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra řídicí techniky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: **Bc. Jakub Janata**

Studijní program: Otevřená informatika
Obor: Počítačové inženýrství

Název tématu: **Diagnostické zařízení pro automatické testování automobilových řídicích jednotek**

Pokyny pro vypracování:

1. Seznamte se automobilovou řídicí jednotkou RPP vyvinutou na katedře.
2. Navrhněte jakým způsobem automatizovaně testovat většinu rozhraní (GPIO, ADC, DAC, CAN, LIN, ...) jednotky RPP.
3. Navrhněte a realizujte hardware pro následné testování.
4. Implementujte softwarovou podporu pro automatické provádění testů na vyvinutém hardwaru.
5. Vytvořte funkční testy několika nejdůležitějších periferií jednotky RPP a navažte jejich automatické spouštění na verzovací systém používaný pro vývoj firmwaru RPP.
6. Vše pečlivě otestujte a zdokumentujte.

Seznam odborné literatury:

[1] BeagleBone Black documentation
http://elinux.org/Beagleboard:BeagleBoneBlack
[2] Michal Horn, "Software obsluhující periferie a FlexRay na automobilové řídicí jednotce", DP ČVUT v Praze
https://support.dce.felk.cvut.cz/mediawiki/images/0/0b/Dp_2013_horn_michal.pdf
[3] RPP wiki https://rtime.felk.cvut.cz/rpp/

Vedoucí: Ing. Michal Sojka, Ph.D.

Platnost zadání: do konce letního semestru 2016/2017

L.S.

prof. Ing. Michael Šebek, DrSc. vedoucí katedry		prof. Ing. Pavel Ripka, CSc. děkan

V Praze dne 11. 2. 2016

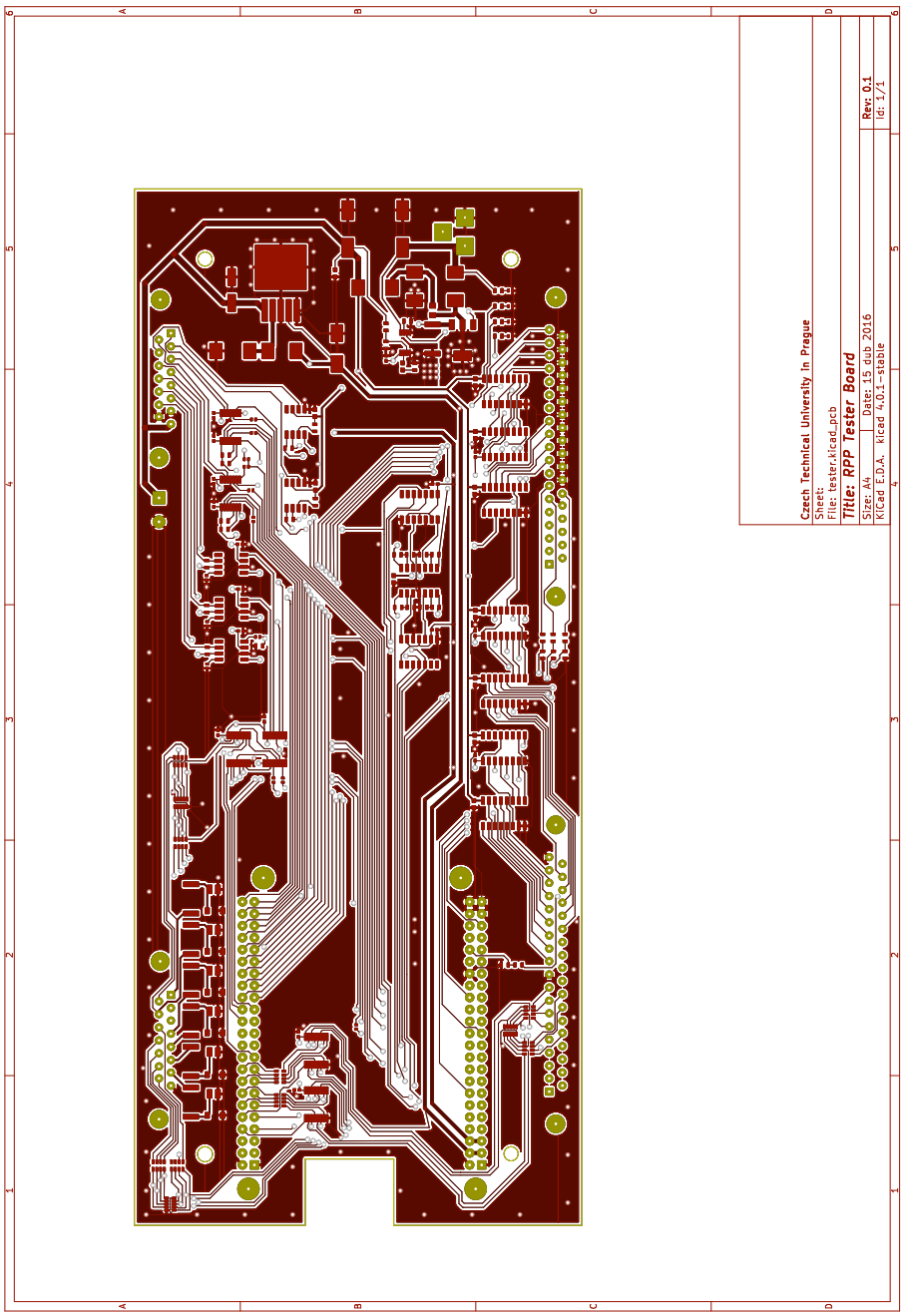
Příloha B
Zkratky

ADC	Analog/Digital Convertor
CAN	Controller Area Network
DAC	Digital/Analog Convertor
DIN	Digital Input
ECAP	Enhanced Capture
HBR	H-Bridge
HOUT	PWM OUTput
JTAG	Joint Test Action Group
LIN	Local Interconnect Network
LOUT	Logical OUTput
MOUT	Power OUTput
PES-RPP	Porsche Engineering Services – Rapid Prototyping Platform
SCI	Serial Communication Interface
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver and Transmitter
USB	Universal Serial Bus

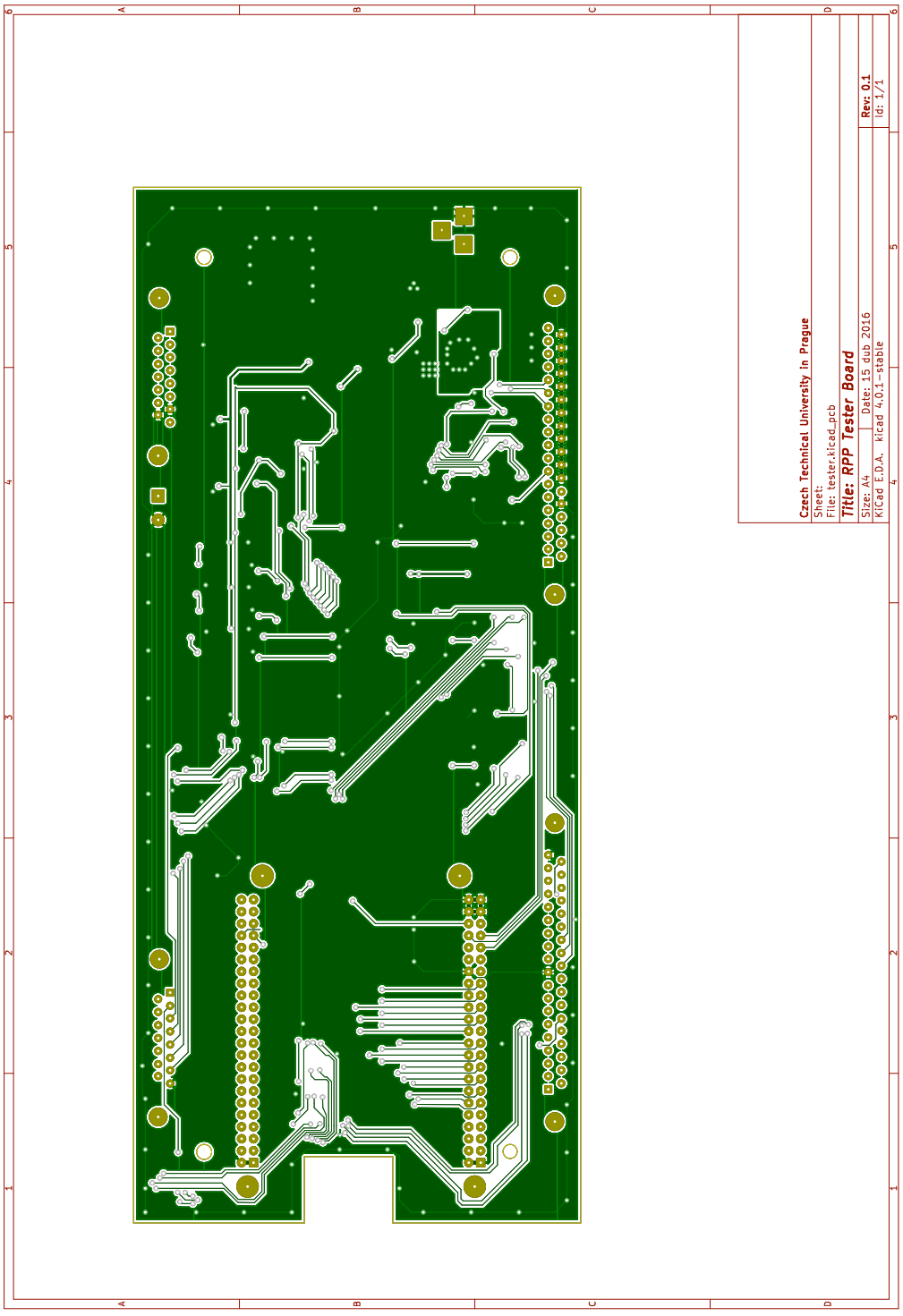
Příloha C
Struktura repozitáře

```
|- dokument
|  |- # zdrojové kódy tohoto dokumentu ve fotmátu .tex
|- rpp-datasheets
|  |- # datasheety elektr. obvodů RPP desky
|- software
|  |- buildbot
|  |  |- # skripty spouštěné nástrojem buildbot
|  |- dts
|  |  |- # device tree struktura pro tester
|  |- init
|  |  |- inicializační skripty
|  |- openocd
|  |  |- openocd flashovací skript + instalační archiv
|  |- python
|  |  |- doc
|  |  |  |- # dokumentace RPP-Testeru
|  |  |- ext
|  |  |  |- # zdroj. kódy rozšíření pro framework v C
|  |  |- results
|  |  |  |- # výsledky testů
|  |  |- rpp_tester
|  |  |  |- # zdroj. kódy RPP-Tester frameworku
|  |  |- tests
|  |     |- integration
|  |     |  |- # testy pro testování RPP desky
|  |     |- unit
|  |        |- # unit testy RPP-Tester frameworku
|  |- server
|     |- # konfigurace nginx a index stránka serveru
|- specifikace
|  |- # specifikace pro National Instruments Corporation
|- tester-schema
   |- datasheets
   |  |- # datasheety elektr. obvodů RPP-Tester desky
   |- gerber
   |  |- # export z KiCadu do souborů formátu .gerber
   |- lib
   |  |- # knihovny pro KiCad - BBB + vlastní rozšíření
   |- # schémata, layouty, seznam součástek a exporty do .pdf
```

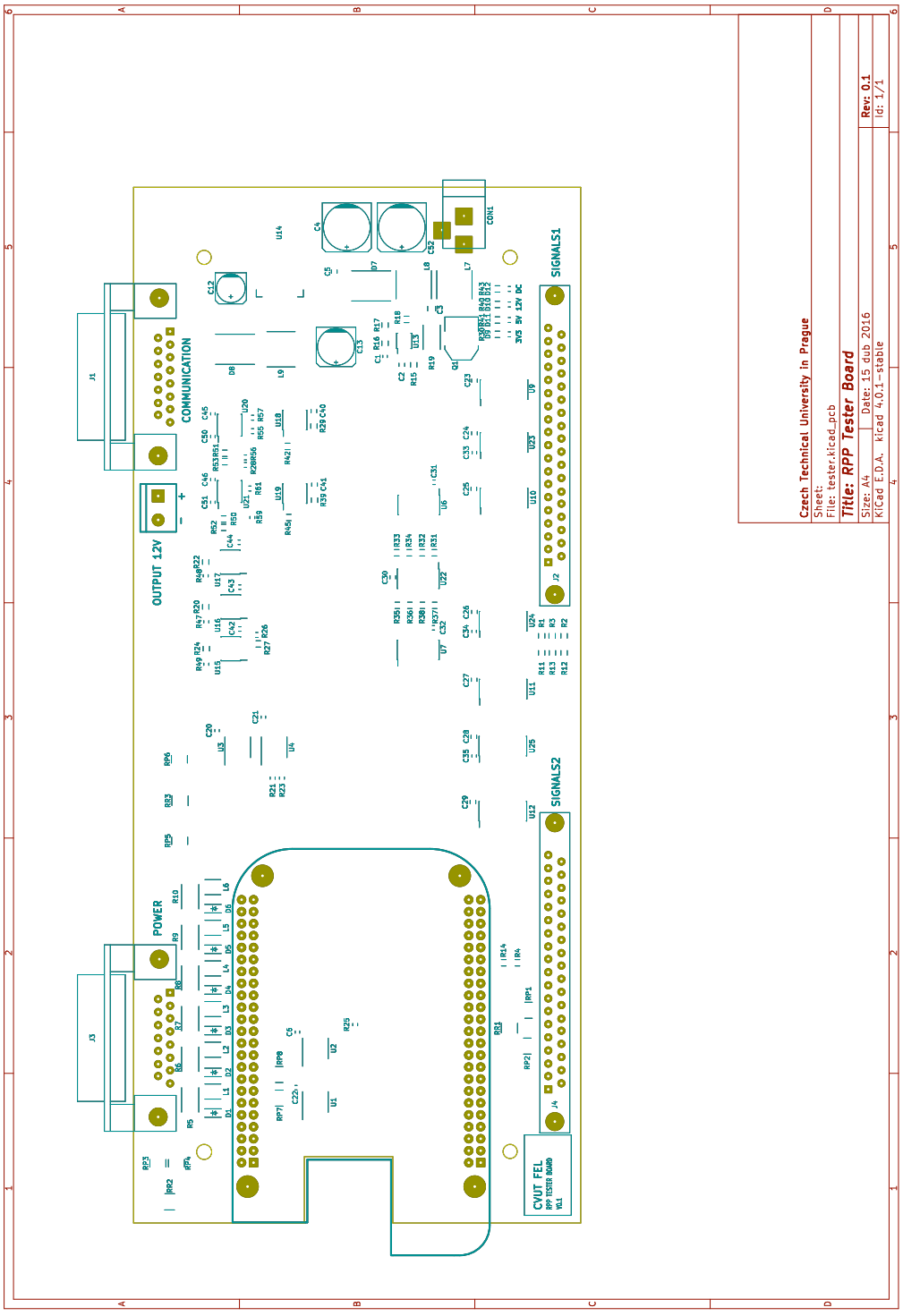
Příloha D
Plošný spoj



Obrázek D.1. Plošný spoj vrchní strana (měď)

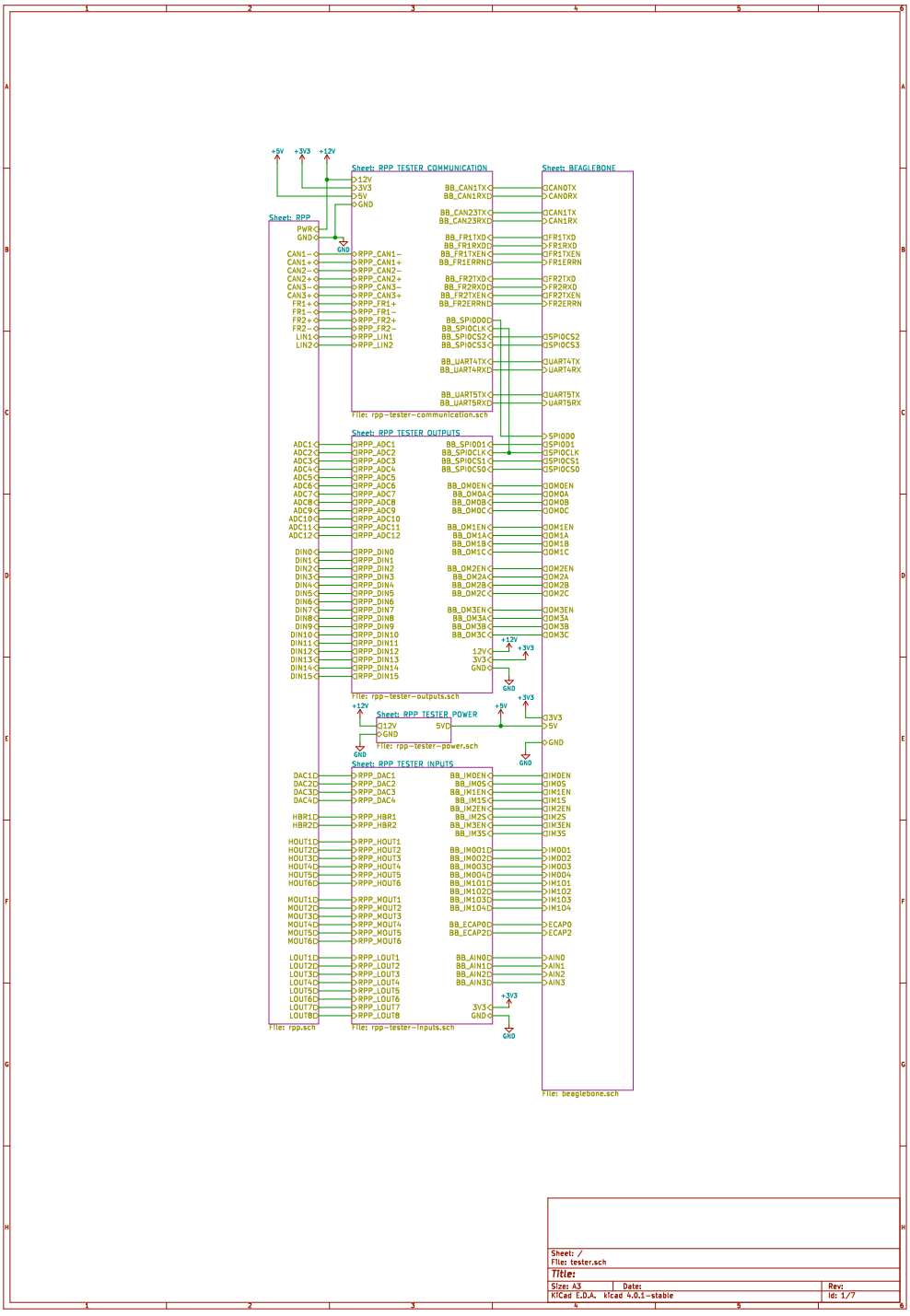


Obrázek D.2. Plošný spoj spodní strana (měď)

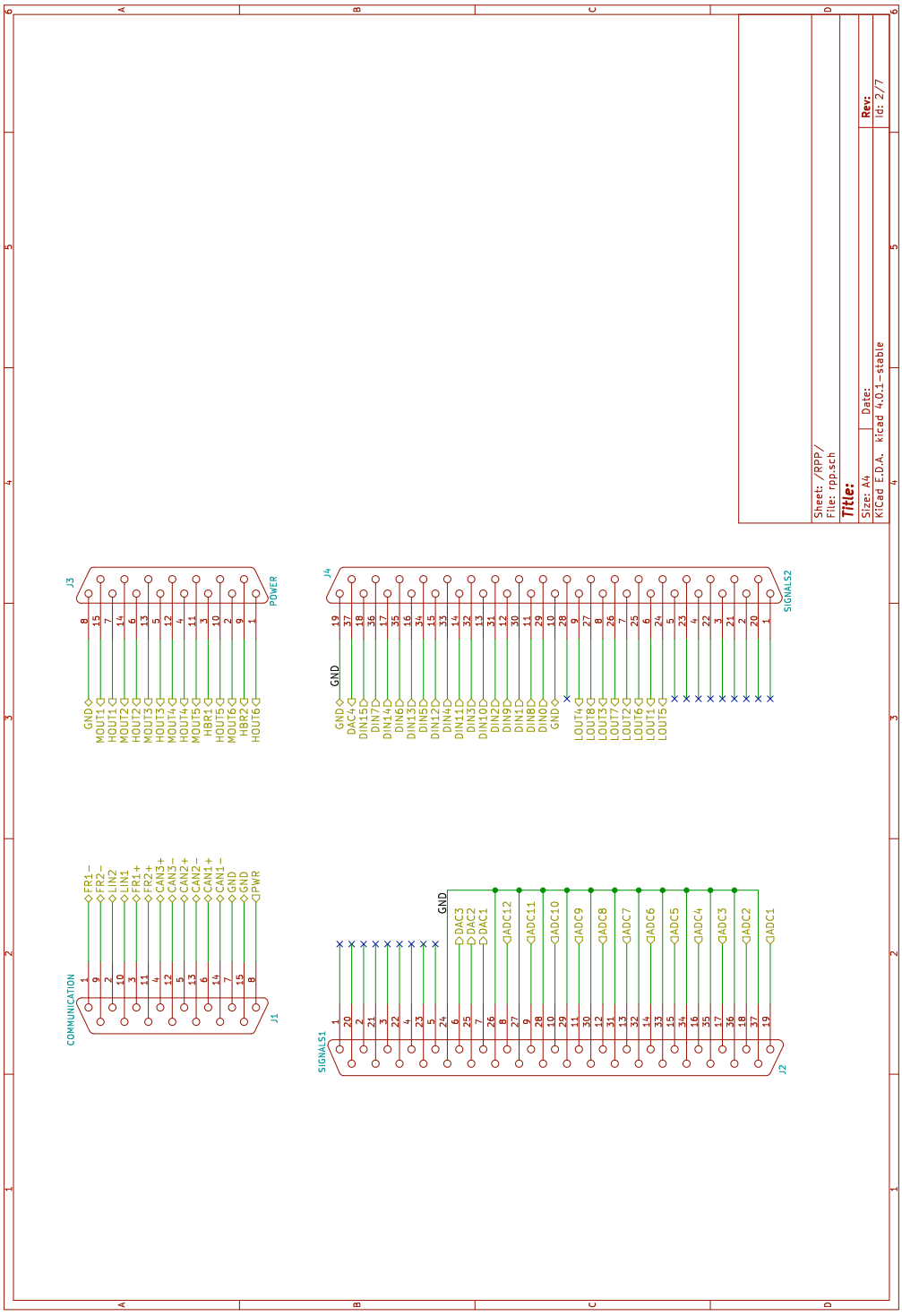


Obrázek D.3. Plošný spoj vrchní strana (potisk)

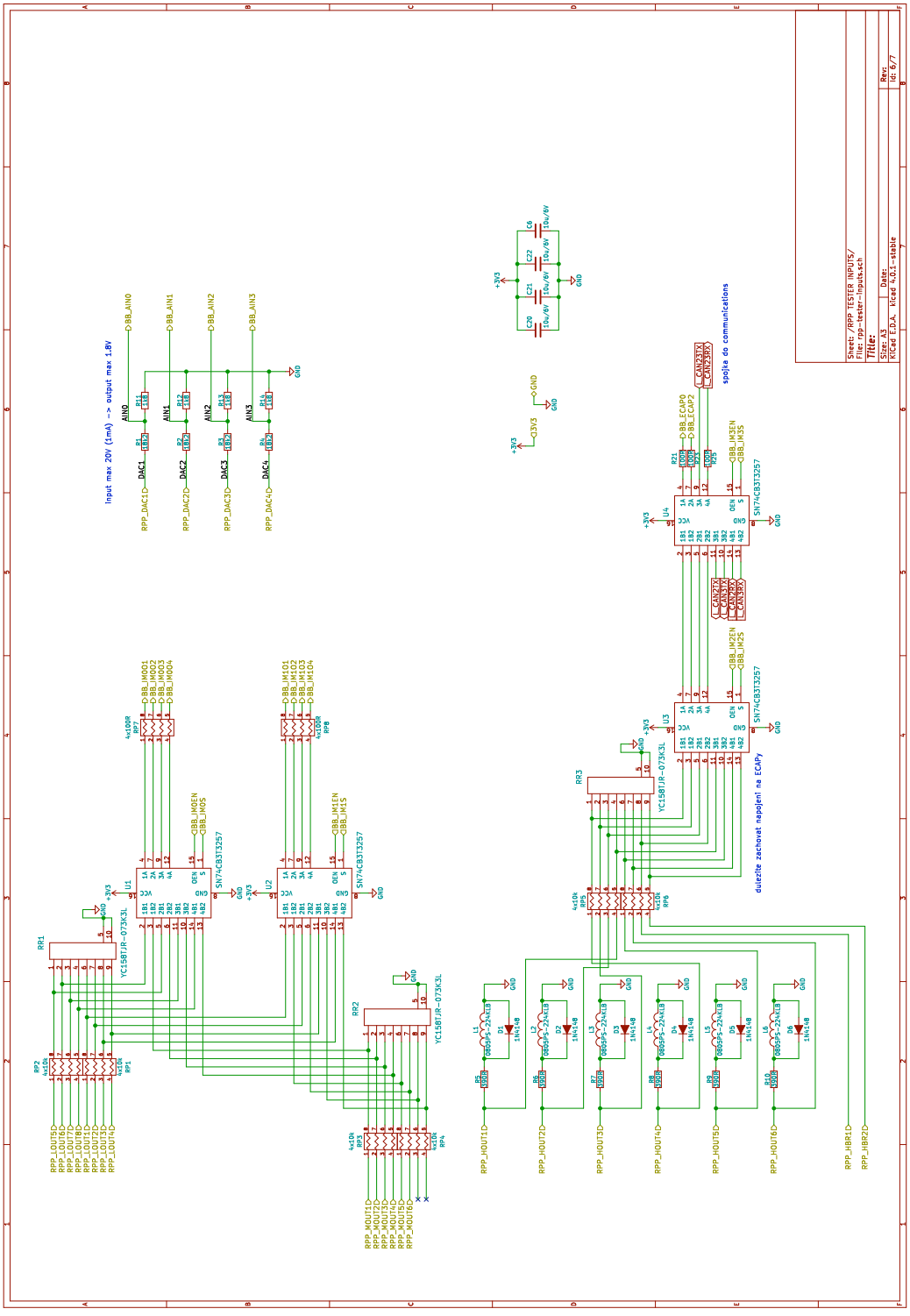
Příloha E
Schéma zapojení



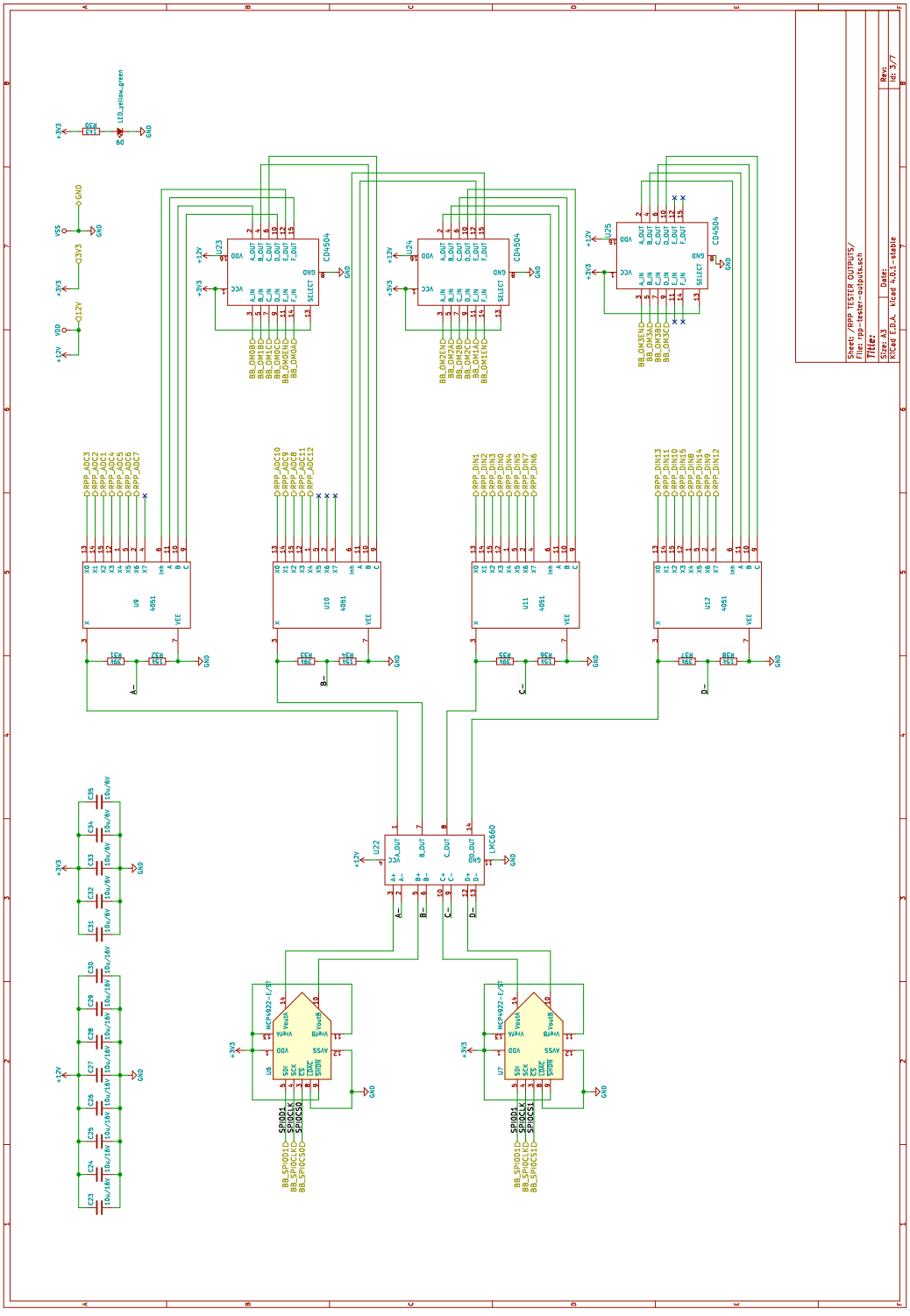
Obrázek E.4. Schéma zapojení (blokové schéma)



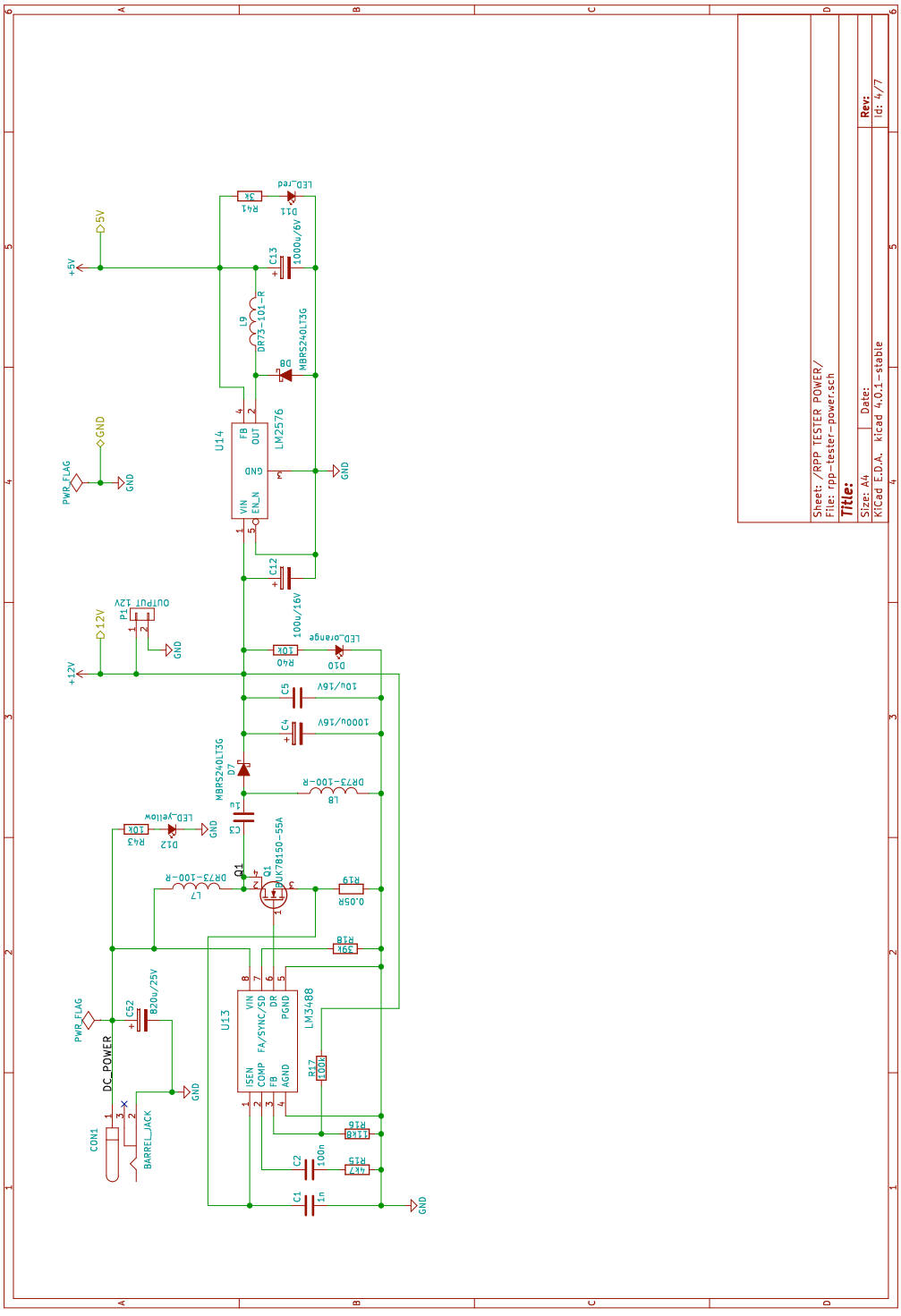
Obrázek E.5. Schéma zapojení (konektory k RPP)



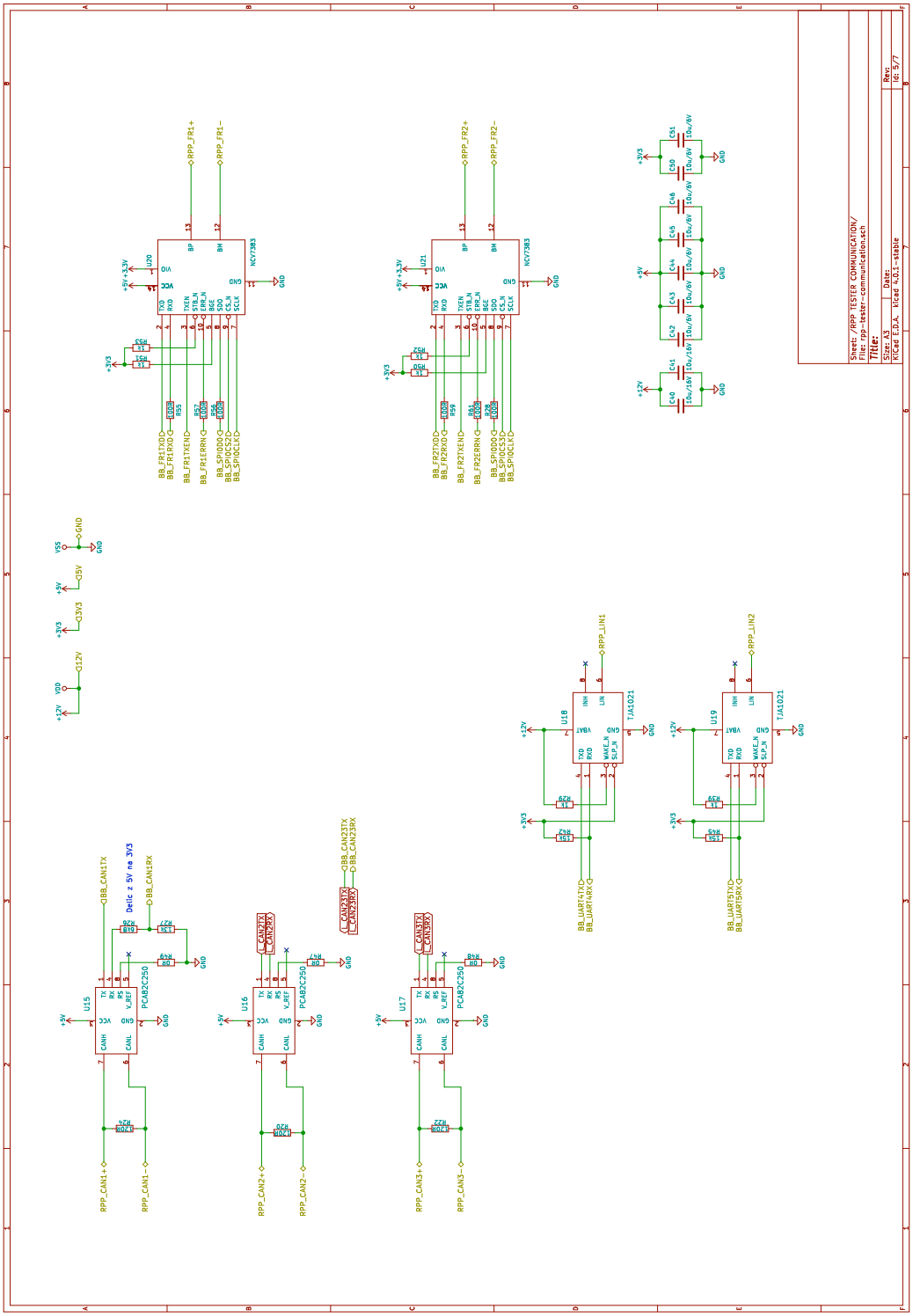
Obrázek E.6. Schéma zapojení (vstupy)



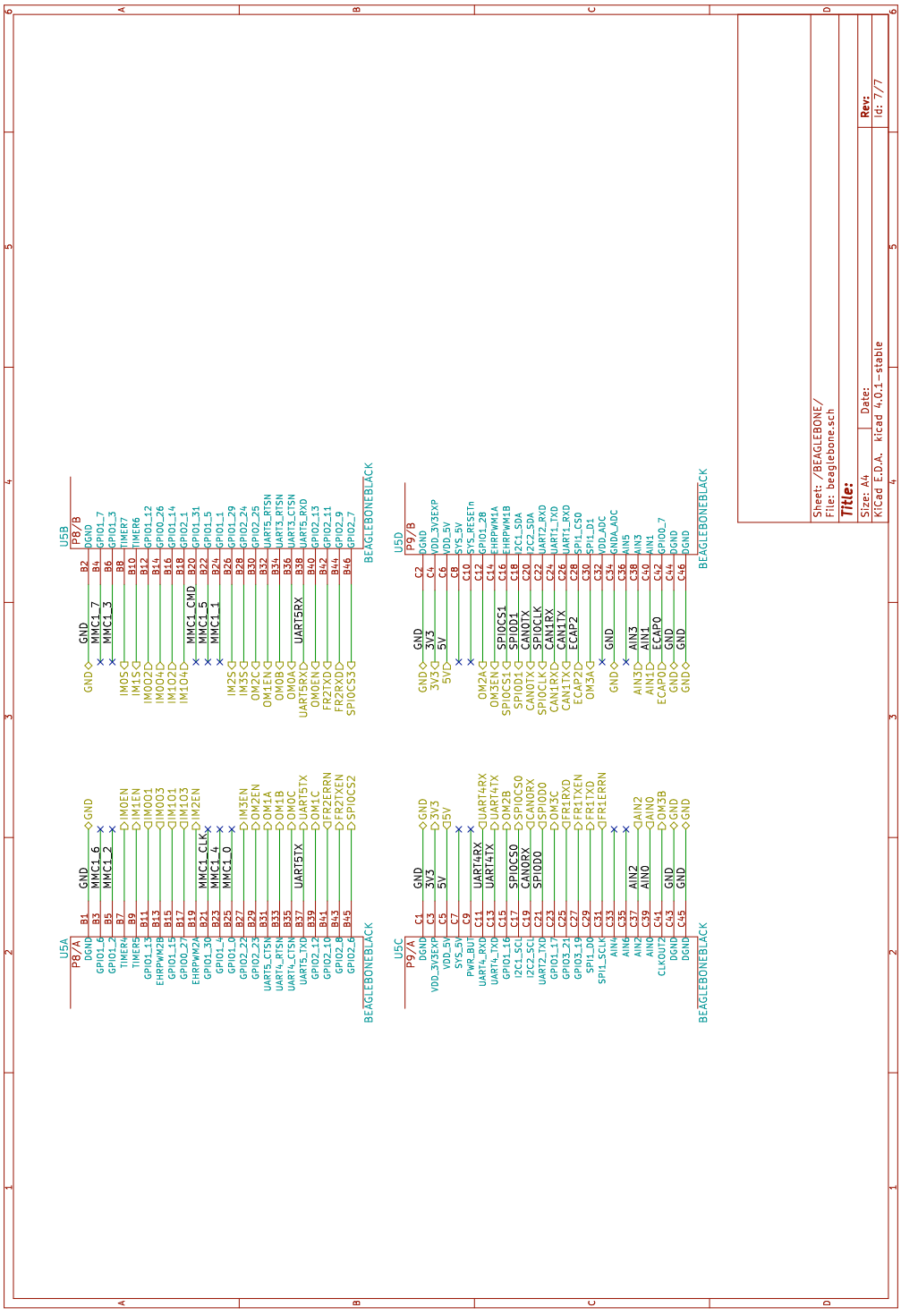
Obrázek E.7. Schéma zapojení (výstupy)



Obrázek E.8. Schéma zapojení (napájení)



Obrázek E.9. Schéma zapojení (komunikace)



Obrázek E.10. Schéma zapojení (konektory k BBB)