České vysoké učení technické v Praze
Fakulta elektrotechnická

Software obsluhující periferie a FlexRay na automobilové řídicí jednotce

Diplomová práce

Bc. Michal Horn

Praha, jaro 2013

Prohlášení

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

V Praze, dne ..... Podpis:.....

Bc. Michal Horn

Vedoucí práce: Ing. Pavel Píša Ph.D.

Anotace

Diplomová práce se zabývá vývojem software pro obsluhu periferií a komunikačního rozhraní FlexRay na automobilové řídící jednotce. V rámci práce byl vyvinutý software pro komunikaci mezi prototypem řídící jednotky a počítačem po sériové lince, interpretaci přijatých příkazů a sada příkazů, která umožní otestovat funkčnost jednotlivých periferií. Testovací příkazy umožňují ověřit návrh hardware jednotky, detekovat chyby na jednotlivých vyrobených kusech před jejich dodáním a provádět základní operace se vstupy, výstupy a sběrnicemi, které řídící jednotka obsahuje.

Annotation

The diploma thesis deals with development of an IO and FlexRay firmware for Automotive ECU. The goal of the thesis is to develop a software which allows communication between the unit and a computer using serial line, develop an interpreter of received commands and set of commands for peripherals functionality testing. The set of the developed commands is used to prove hardware design, check for design errors and can be used as base for quality control of the produced boards. The commands can also be used for basic operations with inputs/outputs and busses included on the control unit.

Klíčová slova

Automobilová řídící jednotka, firmware, FlexRay, ARM, Cortex-R4.

Poděkování

Rád bych na tomto místě poděkoval svému vedoucímu Ing. Pavlu Píšovi Ph.D. za neocenitelnou pomoc při řešení všech obtíží spojených nejen s programováním vestavěných systémů, za trpělivost při vysvětlování a důkladné odpovědi na vznesené dotazy. Dále bych chtěl poděkovat panu Ing. Michalovi Sojkovi Ph.D. a panu Ing. Rostislavu Lisovému za pomoc v začátcích vývoje. V neposlední řadě chci poděkovat profesoru Zdeňkovi Hanzálkovi za příležitost podílet se na tomto zajímavém projektu, ve kterém jsem se získal mnoho nových znalostí a zkušeností.

Rejstřík zkratek

Zkratka	Význam
ABS	Protiblokovací systém pro zabránění zablokování kola při brzdění.
ADC	Převodník analogového signálu na digitální.
CAN	Komunikační rozhraní určené původně pro automobilový průmysl a propojení jednotlivých elektronických systémů. Je založen na adresaci zpráv v síti jejich určením a arbitrací podle jejich priorit.
CMDPROC	Název příkazového interpretru používaného v této práci.
CPU	Jádro mikrokontroléru vykonávající instrukce programu a základní aritmetické a logické operace.
CRC	Hašovací funkce pro detekci chyb během přenosu či ukládání dat.
DAC	Převodník digitálního signálu na analogový.
DIN	Digitální vstup.
DMM	Modul mikrokontroléru pro přímou modifikaci dat v paměti MCU vnějšími periferiemi.
ECU	Elektronická řídící jednotka (někdy též motorová řídící jednotka).
EEPROM	Programovatelná napěťově nezávislá (nevolatilní) paměť pro čtení s možností elektronického zápisu/přepisu.
EMC	Elektromagnetická kompatibilita. Schopnost elektrického zařízení pracovat spolehlivě ve společném elektromagnetickém prostředí s dalšími zařízeními bez podstatného rušivého vlivu na toto prostředí a zařízení.
FRAY	Komunikační rozhraní navržené pro automobilový průmysl, založené na opakujících se komunikačních cyklech s deterministickým a nedeterministickým segmentem.
FLASH	Paměť vyvinutá z EEPROM. Přináší možnost mazání po celých jednotlivých stránkách (NAND FLASH) nebo slovech (NOR FLASH) a rychlejší programování než EEPROM.
GPIO	Obecný vstupně výstupní pin (port).
HAL	Vrstva pro abstrakci hardwarových detailů od programátora vyšších vrstev.
HBR	H-můstek pro ovládání motoru.
HOUT	Silový výstup 12 V pro zátěž indukčního charakteru do 10 A.
JRVA	Jednotka pro rychlý vývoj aplikací – pracovní název pro řídící jednotku a zkratka zavedená v rámci této práce.
JTAG	Protokol a rozhraní pro ladění vestavěných systémů.
LIN	Komunikační rozhraní navržené pro automobilový průmysl. Založen na principu master-slave, kde master vždy zahajuje komunikaci.

LOUT	Název pro výstup 12 V pro zátěž do 100 mA.
LSB	Nejméně významná pozice bitu v binární reprezentaci čísla.
MCU	Mikrokontrolér – systém procesorového jádra, vestavěné paměti, sběrnic a periferií, které dohromady tvoří celek sloužící pro řízení elektronického zařízení.
MOUT	Název pro silový výstup 12 V pro zátěž do 2 A.
MPL	Mozilla Public Licence.
NHET	Modul mikrokontroléru sloužící jako programovatelný časovač.
PCB	Deska tištěných spojů.
POC	Podsystém protokolu FlexRay zajišťující reakci vnitřních mechanismů na vnější příkazy a vnitřní stavy.
PWM	Pulzně šířková modulace signálu.
SCI	Modul mikrokontroléru poskytující sériové komunikační rozhraní (UART nebo LIN)
SDRAM	Dynamická paměť ukládající data ve formě elektrických nábojů v kondenzátorech se synchronním přenosem dat. Data je potřeba periodicky obnovovat.
SMD	Elektronické součástky určené pro povrchovou montáž.
SPI	Synchronní sériové rozhraní určené pro komunikaci mezi kontrolérem a periferiemi nebo jiným kontrolérem.
SRAM	Statická polovodičová paměť. Data nepotřebují obnovovat.
TTL	Standart definující napěťové úrovně pro logickou úroveň 0 (0–0.8 V) a logickou úroveň 1 (2–5 V)
UART	Standart pro asynchronní sériový přenos dat.
VBAT	Název pro napájecí okruh jednotky.

Obsah

1 **Úvod** 1
1.1 Struktura práce 1
1.1.1 Textová zpráva 1
1.1.2 Doprovodné CD 2
2 **Cíle diplomové práce** 3
2.1 Shrnutí úkolů 3
2.2 Testování periferií 4
2.3 Další funkce k testování 6
3 **Vývoj projektu** 7
3.1 Vznik projektu 7
3.2 Příprava a vývoj před dodáním prototypů 7
3.3 Pokračování vývoje na prototypech 7
3.4 Současný stav 8
4 **Automobilová řídící jednotka** 9
4.1 Úvod do tématu 9
4.2 Jednotka pro rychlý vývoj aplikací 9
4.3 Požadavky na produkt 10
4.4 Použitý mikrokontrolér 11
4.4.1 Parametry 13
4.4.2 Architektura 13
4.5 Využití jednotky 13
5 **Rozbor** 17
5.1 Vývojové prostředí 17
5.2 FreeRTOS 17
5.3 CMDPROC 18
5.4 Sériová linka 18
5.5 CAN High speed 18
5.6 Analogově-digitální převodník (ADC) 19
5.7 Digitálně-analogový převodník (DAC) 20
5.8 Digitální vstupy (DIN) 20
5.9 FlexRay (FRAY) 20
5.10 H-můstek (HBR) 21
5.11 Výstup do 10 A (HOUT) 21
5.12 Výstup do 2 A (MOUT) 21
5.13 Výstup do 100 mA (LOUT) 22
5.14 LIN 22
5.15 Napájecí okruhy (VBAT) 23

6 **Návrh testovacího software** 24
6.1 Cíle návrhu 24
6.2 Architektura firmware 24
6.3 Jmenné konvence 25
6.3.1 Názvy souborů se zdrojovými kódy 25
6.3.2 Názvy funkcí 27
6.4 Sériová linka a CMDPROC 27
6.5 FlexRay 28
6.6 Obecné rozhraní pro ovládání pinů 29
6.7 Rozhraní pro ovládání PWM 32
6.8 Rozhraní pro přístup k SPI 32
7 **Implementace** 34
7.1 Navázání CMDPROC na sériovou linku 34
7.1.1 CMDPROC - IO interface 34
7.1.2 Buffery pro příjem a odesílání 35
7.1.3 Odesílání dat 35
7.1.4 Přijímání dat 35
7.2 Hardwarová abstrakční vrstva 35
7.2.1 Mapy pinů 35
7.2.2 Mapy portů 37
7.3 Dekódování odpovědi z SPI 38
7.4 FlexRay 40
7.4.1 Konfigurační funkce 40
7.4.2 Kontrolní funkce 41
7.4.3 Pomocné funkce 41
7.5 Příklad implementace high-level driveru 41
8 **Testování** 43
8.1 Použité vybavení 43
8.2 Testování a výsledky 43
8.2.1 Externí paměť SDRAM 43
8.2.2 6x výstup do 10 A s možností PWM (HOUT) 45
8.2.3 6x výstupy do 2 A (MOUT) 45
8.2.4 8x digitální výstupy do 100 mA (LOUT) 46
8.2.5 16x digitální vstup (DIN) 46
8.2.6 2x LIN 46
8.2.7 4x digitálně-analogový převodník 46
8.2.8 12x analogově-digitální převodník 47
8.2.9 3x CAN High speed 47
8.2.10 Ethernet 47
8.2.11 1x H-můstek 48

8.2.12 1x FlexRay pro dva kanály 48
8.2.13 Ovládání napájecích okruhů 49
8.2.14 Přechod do úsporného režimu 49
8.2.15 Probuzení z úsporného režimu 49
8.3 Automatizace testů 50
8.3.1 Testovací skript 50
8.3.2 Konfigurace skriptu 50
8.3.3 Spuštění skriptu a ovládání 50
8.4 Shrnutí výsledků testů 51
9 **Závěr** 53
A **Tvorba příkazu pro CMDPROC** 58
A.1 Definice příkazu 58
A.2 Příklad přidání nového příkazu 59
B **Manuál pro navázání komunikace jednotky s počítačem** 61
C **Dokumentace k příkazům** 62
C.1 Help 62
C.2 Sleep 62
C.3 Readadc 62
C.4 Testcan 63
C.5 Dacpinon 63
C.6 Dacpinval 64
C.7 Dacstat 64
C.8 Dinportst 65
C.9 Dinportpr 65
C.10 Dinportint 66
C.11 Dinreset 66
C.12 Dinswst 67
C.13 Dinportval 67
C.14 Dinstat 67
C.15 Ethernet 67
C.16 Fraystat 68
C.17 TestfrayA 68
C.18 TestfrayB 68
C.19 Hbrwdg 69
C.20 Hbrpwm 69
C.21 Hbrpwmstart 70
C.22 Hbrpwmstop 70
C.23 Hbren 70
C.24 Hbrdir 70
C.25 Houtfail 71

C.26 Readhifbk 71
C.27 Houtpwm 71
C.28 Houtpwmstart 72
C.29 Houtpwmstop 72
C.30 Loutwr 72
C.31 Loutval 73
C.32 Loutstat 73
C.33 Testlin 74
C.34 Pinval 74
C.35 Pindir 74
C.36 Portval 75
C.37 Testsdram 75
C.38 Spitr 76
C.39 Spimst 76
C.40 Poweron 77
C.41 Poweroff 77
D **Tabulka pinů** 78
D.1 Analogově-digitální převodník 78
D.2 Digitální vstupy 79
D.3 Digitálně-analogový výstup 80
D.4 Výstup do 10 A (HOUT) 80
D.5 Výstup do 2 A (MOUT) 81
D.6 Výstup do 100 mA (LOUT) 81
D.7 H-můstek 82
D.8 CAN High speed 82
D.9 FlexRay 83
D.10 LIN 83
D.11 Napájecí okruhy 83

Seznam obrázků

4.1 První automobil s elektronickým vstřikováním paliva. VW411 z roku 1967. [5] *9*
4.2 Blokové schéma JRVA *12*
4.3 Organizace paměti TMS570LS3137. [21, strana 112] *14*
4.4 Schéma architektury TMS570LS3137. [21, strana 100] *15*

5.1 CAN datový rámec [3, strana 10] *19*
5.2 FlexRay komunikační cyklus [7, strana 100] *21*
5.3 LIN rámec [21, strana 1433] *22*

6.1 Architektura firmware *26*
6.2 Připojení CMDPROC k sériové lince *28*
6.3 Schéma FlexRay driveru *30*
6.4 Schéma řešení HAL pro piny *31*
6.5 Schéma řešení HAL pro porty *32*
6.6 SPI dekodér odpovědi, struktura map *33*

7.1 Bitové schéma deskriptoru pinů *36*

8.1 Testovací plocha a dvě jednotky *44*

1 Úvod

Moderní automobily obsahují řadu elektronických řídících jednotek (ECU), sloužících různým účelům. Jedna jednotka slouží pro řízení samotného motoru, další kontroluje automatickou převodovku, jiná protiblokovací systém (ABS) a může existovat mnoho dalších systémů nezbytných pro bezpečný provoz, ale i zvýšení komfortu posádky. ECU musí splňovat velmi vysoké nároky na spolehlivost a přesnost časování úloh, protože v případě chyby by mohlo snadno dojít k ohrožení majetku nebo i zdraví.

Na katedře řídící techniky elektrotechnické fakulty ČVUT probíhá vývoj a testování takovéto jednotky na základě požadavků výrobce automobilů. Cílem diplomové práce bylo implementovat software pro komunikaci této jednotky s počítačem, navrhnout a implementovat funkce pro ovládání periferií a komunikačního rozhraní FlexRay a sadu příkazů k jejich otestování.

1.1 Struktura práce

1.1.1 Textová zpráva

V kapitole 2 CÍLE DIPLOMOVÉ PRÁCE je čtenář seznámen s účelem práce a přesným zadáním jednotlivých úkolů.

V kapitole 3 VÝVOJ PROJEKTU je stručně popsána historie vývoje řídící jednotky od záměru jednotku vyvinout až do současného stavu před odevzdáním diplomové práce.

Text pokračuje kapitolou 4 AUTOMOBILOVÁ ŘÍDÍCÍ JEDNOTKA, zaměřenou na popis cílového hardware. Zde je čtenář uveden do problematiky řídících jednotek a seznámen se zařízením použitým v této práci. Nalezne zde požadavky na produkt, popis použitého mikrokontroléru a blokové schéma jednotky.

Další kapitoly se již věnují přímo řešeným úkolům. Kapitola 5 ROZBOR obeznámí čtenáře se sadou software pro vývoj pro MCU od Texas Instruments, dozví se základní informace o operačním systému FreeRTOS, který je použitý v testovacím software, bude seznámen s principem funkce a vlastnostmi příkazového interpretru a získá přehled o testovaných periferiích, jejich způsobu komunikace s kontrolérem a případně komunikačními protokoly.

V kapitole 6 NÁVRH jsou popsány zohledňované cíle návrhu a samotná architektura firmware. Výklad pokračuje vysvětlením jmenných konvencí zavedených během vývoje a v dalších sekcích následuje popis integrace

příkazového interpretru, rozebrání obecného rozhraní pro ovládání portů a rozhraní pro dekódování odpovědi z SPI zařízení. V této kapitole jsou také popsány poznatky z analýzy kódu pro FlexRay [23], poskytnutého katedrou měření elektrotechnické fakulty ČVUT.

Kapitola 7 IMPLEMENTACE se zaměřuje na popis vlastní implementace. Zde jsou zaznamenány důležité a zajímavé techniky použité pro implementaci vybraných modulů softwaru. Je popsána obsluha sériové linky, navázání CMDPROC na sériovou linku, způsob dekódování odpovědí z SPI periferií na textovou formu přehlednou a čitelnou pro člověka a způsob jednotného ovládání vstupně výstupních pinů bez ohledu na způsob jejich komunikace s kontrolérem.

V předposlední kapitole 8 TESTOVÁNÍ je čtenář obeznámen s postupem testování periferií a dalších funkcí. Je zde popsáno použité vybavení, postup testování s výsledky, odhalené problémy a jejich řešení. Na závěr je uveden způsob automatizace testování.

V kapitole 9 nazvané ZÁVĚR se nachází zhodnocení splnění cílů diplomové práce a navržené další směřování činnosti na projektu.

Na konci dokumentu se nachází přílohy. První příloha A obsahuje návod na začlenění nových příkazů do CMDPROC, v příloze B jsou uvedeny informace potřebné k navázání komunikace jednotky s počítačem. Následuje příloha C s dokumentací příkazů a popisem syntaxe a v poslední příloze D je tabulka všech periferií s přiřazenými piny mikrokontroléru a adresami SPI.

Většina obrázků v dokumentu je v anglickém jazyce z toho důvodu, že se práce zabývá jednotkou vyvíjenou pro zahraničního zadavatele. Předpokládá se tedy využití některých ilustrací i pro účely budoucí prezentace produktu.

1.1.2 Doprovodné CD

Z licenčních důvodů neobsahuje doprovodné CD kompletní zdrojové kódy vytvořeného firmware. Místo toho jsou v kořenovém adresáři k nalezení dvě složky.

Text document Obsahuje pdf soubor s elektronickou verzí tohoto textu.

Test script Obsahuje skript v jazyce Python a jeho konfigurační soubor, použitý pro poloautomatické testování jednotky, popsané v sekci 8.3.

2 Cíle diplomové práce

2.1 Shrnutí úkolů

Hlavním cílem diplomové práce je ověřit návrh hardware a otestovat dosud vyrobené kusy řídící jednotky navržené katedrou řídící techniky pro výrobce automobilů. Tento cíl s sebou nese řadu úkolů.

1. Základním úkolem je *zprovoznění vývojového prostředí* a dalších podpůrných nástrojů. V počátcích práce, se pro vývoj používal vývojový kit se stejným mikrokontrolérem, ale po dodání vyrobených jednotek bylo potřeba nově zprovoznit *také rozhraní JTAG*.

2. Základním diagnostickým prostředkem jednotky je sériové rozhraní integrované na mikrokontroléru. Proto je další povinností *zprovoznit* přenos dat přes *sériovou linku.*

3. Pro větší pohodlí a hlavně flexibilitu řešení následuje *integrace příkazového interpretru* (CMDPROC), který čte příkazy zadané po sériové lince, vykonává jim přiřazené rutiny a poskytuje zpětnou vazbu.

4. Následují již úkoly spojené se samotným testováním. Je potřeba vytvořit takovou sadu příkazů, pomocí které bude možné otestovat správnou funkci jednotlivých periferií a také některé další vlastnosti, které se vnějšími periferiemi přímo nesouvisí.

5. FlexRay je speciální případ pro svou relativní složitost a chybějící podporu ve vývojových nástrojích. Katedrou měření nám byl poskytnutý kód z bakalářské práce pana Bc. Martina Zemana [23], která se tímto rozhraním zabývala. Dalším úkolem je tedy provézt analýzu kódu, porozumět komunikačnímu protokolu a jeho ovládání.

6. V návaznosti na předchozí bod se musí provést úpravy poskytnutého FlexRay kódu a integrovat ho do testovacího firmware tak, aby bylo možné ověřit funkčnost této periferie na jednotce.

7. Na závěr je potřeba zdokumentovat hotový kód a hotové příkazy. Dokumentaci kódu se věnují kapitoly 5, 6 a 7 v tomto dokumentu a příkazy nalezne čtenář v příloze C.

2.2 Testování periferií

Uvedený seznam je kompletním výčtem periferií a jejich funkcí, které byly určené k otestování do odevzdání práce. V současné době se netestují pro provoz na hraničních podmínkách, ale pouze pro běžné provozní podmínky, protože cílem je ověřit, že každá periferie má správně připojené datové a řídící signály k mikrokontroléru a že je plně provozuschopná. Velká část vstupů a výstupů jednotky je obsloužena specializovanými integrovanými obvody, které zaručují normami požadovanou odolnost a diagnostiku. Většina funkcí těchto obvodů je řízena přes synchronní sériové rozhraní (SPI – serial peripheral interface). Ostatní funkce, vstupy nebo výstupy jsou řízené pomocí obecného vstupně-výstupního subsystému GPIO, integrovaného na mikrokontroléru.

- Externí paměť SDRAM:
  - zjištění celkové kapacity osazeného modulu,
  - provedení paměťového testu modulu přes celý adresní rozsah.
- 6x výstup do 10 A s možností PWM (HOUT):
  - zapnutí pinu,
  - vypnutí pinu,
  - čtení analogové hodnoty o protékajícím proudu (IFBK),
  - čtení hodnoty diagnostiky (DIAG),
  - detekce chybového stavu,
  - zapnutí PWM na pin.
- 6x výstupy pro zátěž do 2 A (MOUT):
  - zapnutí pinu,
  - vypnutí pinu,
  - čtení hodnoty diagnostiky (DIAG).
- 8x digitální výstupy pro zátěž do 100 mA (LOUT):
  - zapnutí pinu,
  - vypnutí pinu,
  - čtení stavu budiče po SPI.

- 16x digitální vstup (DIN):
  - čtení stavu pinu přes GPIO,
  - čtení stavu pinu přes SPI,
  - nastavení programovatelných pinů přes SPI.
- 2x LIN:
  - otestování alespoň jednoho LIN pomocí smyčky. Druhý sdílí konektor se sériovým rozhraním, které je otestované samotnou komunikací s počítačem. Podrobné testování má smysl až bude k dispozici podpora pro přístup k diagnostice jednotky přes Ethernet nebo jiné rozhraní.
- 4x digitálně-analogový převodník:
  - nastavení požadovaného napětí na všech kanálech.
- 12x analogově-digitální převodník:
  - čtení hodnot na všech kanálech.
- 3x CAN High speed:
  - odesílání a přijímání zpráv mezi jednotlivými rozhraními sběrnice CAN,
  - zotavení z chybového stavu,
  - detekce chyby na síti.
- 1x Ethernet:
  - odeslání paketu. Kód pro otestování Ethernetu není výsledkem této diplomové práce a není proto rozebírán.
- 1x H-můstek:
  - nastavení směru,
  - zapnutí, vypnutí,
  - komunikace s budičem po SPI,
  - povolení PWM.
- 1x FlexRay pro dva kanály:
  - přijímání a odesílání zpráv mezi dvěma JRVA jednotkami.

2.3 Další funkce k testování

Kromě periferií s externími výstupy je potřeba ověřit ještě další funkce, které sice neslouží přímo k ovládání různých akčních členů nebo snímání údajů, ale jsou přesto nezbytné pro praktické nasazení.

1. Přechod desky do úsporného režimu,
2. probuzení desky z úsporného režimu detekcí externího podnětu,
3. ovládání napájecích okruhů.

3 Vývoj projektu

3.1 Vznik projektu

Projekt vznikl na základě zadání zakázky výrobcem automobilů na prototyp nové řídící jednotky katedře řídící techniky na ČVUT. Požadavek byl vyvinout nové zařízení, které poslouží pro otestování perspektivní a proti poruchám odolné architektury mikrokontroléru a také sady vhodných obvodů realizujících vstupy a výstupy s vysokou úrovní integrované diagnostiky. Byl sestaven šestičlenný tým pracovníků, složený ze dvou hardwarových vývojářů, tří softwarových vývojářů a hlavního koordinátora.

Během návrhu hardware bylo nutné projít několika schvalovacími koly, během kterých se kontrolovala správnost elektronického návrhu s ohledem na rozmístění součástek, spojů a integritu přenášených signálů (požadavky na elektromagnetické vyzařování a odolnost - EMC). Po několika iteracích návrhu byly s určitým zpožděním proti původnímu plánu vyrobené první kusy jednotek.

3.2 Příprava a vývoj před dodáním prototypů

Vývoj firmware začal ještě před tím, než byl dokončen vývoj a výroba prototypů. První seznamování s mikrokontrolérem a některými periferiemi, u kterých to bylo možné, probíhalo na vývojovém kitu od Texas Instruments. Jednalo se o TMS570LS31x Hercules Development Kit [22].

Během léta 2012 probíhalo studium kitu a operačního systému FreeRTOS. V první polovině léta se podařilo zprovoznit sériovou linku a integrovat interpretr příkazů CMDPROC. Ve druhé polovině léta probíhaly experimenty se sběrnicí CAN, během kterých byl připraven kód pro posílání a přijímání zpráv, byl zprovozněn analogově-digitální převodník a vytvořeny první verze funkcí pro ovládání periferií mimo SPI.

3.3 Pokračování vývoje na prototypech

Na přelomu srpna a září byla dokončena výroba prvních 10 kusů jednotek a po nalezení způsobu jak použít JTAG bylo možné pokračovat ve vývoji již na cílovém hardware. Během zimy 2012 vznikla hlavní část příkazů pro ovládání většiny periferií, ovládání napájení a bylo odhaleno několik nedostatků návrhu hardware. Tyto nedostatky byly opraveny nejprve dočasně s využitím drátkových propojek a provizorním umístěním SMD součástek

na potřebná místa. Pro účely prezentací u zadavatele bylo však nutné desky poslat zpět dodavatelské firmě, která dle vytvořených podkladů provedla profesionální opravy.

3.4 Současný stav

V rámci diplomové práce byly dokončeny příkazy pro testování zbytku periferií, doplněno ovládání PWM, byla vytvořena dokumentace na wiki stránkách projektu, zprovozněna sběrnice FlexRay a byly otestovány všechny kompletní a dostupné jednotky. Detailní popis je k nalezení v dalších kapitolách tohoto dokumentu.

4 Automobilová řídící jednotka

4.1 Úvod do tématu

Automobilová elektronická řídící jednotka (ECU) je zařízení, které pomocí dat získaných z čidel motoru, událostí vyvolaných řidičem a dalšími periferiemi, kterých může být v moderním automobilu značné množství, zajišťuje bezpečný, spolehlivý a ekonomický chod spalovacího motoru, ABS nebo jiných vzájemně propojených systémů. Je nezbytné, aby řídící jednotka pracovala v reálném čase, byla naprosto spolehlivá, uměla detekovat selhání některé části stroje a zamezit v tom případě vzniku životu ohrožující situace. První motor s elektronickým vstřikování paliva byl použit v automobilu VW 411 (viz obrázek 4.1) představeném v roce 1967, ale až v 80. letech 20. století se tento koncept začal masivně uplatňovat.

Obrázek 4.1: První automobil s elektronickým vstřikováním paliva. VW411 z roku 1967. [5]

4.2 Jednotka pro rychlý vývoj aplikací

Cílový hardware, na kterém probíhá veškerá činnost v této diplomové práci, je elektronika navržená jako prototyp na zakázku pro zahraničního výrobce automobilů. Její pracovní název pro účely této práce je *Jednotka pro rychlý vývoj aplikací* (JRVA).

4.3 Požadavky na produkt

Specifikace zakázky obsahuje několik základních požadavků.

- Výsledné zařízení musí být rozšiřitelné ve smyslu dostatečného výpočetního výkonu, úložné kapacity a vybaveno konektory pro připojení různých akčních členů.
- Musí být vyroben kompletní vlastní prototyp, jako stavební prvek nesmí být použit hotový komerční produkt (např. MicroAutoBox).
- Je nutné dodržet požadavky definované pro použití v řízení automobilu a VW legals (Norma 8000).
- Návrh schématu a rozmístění součástek a subsystémů na desce musí být modulární, ale výsledný hardware musí být monolitický.

Další technologické požadavky nesouvisí přímo s diplomovou prací, ale pro úplnost jsou zde přesto uvedeny.

- Elektronika bude umístěna v odolném krytu (automotive box), s čímž souvisí výběr konektorů.
- Všechny základní funkce budou umístěny na jednom PCB o maximálních rozměrech 200x160 mm.
- Použití automobilových konektorů, IO 56 pinů, 2x24 pinů pro silové signály, Ethernet konektor.
- Návrh na maximálně osmivrstvé desce plošných spojů (PCB).
- Mikrokontrolér s podporou Matlabu a Simulinku.

Z pohledu diplomové práce jsou zajímavější konkrétní požadavky na hardware, protože od těchto požadavků se odvíjí její zadání.

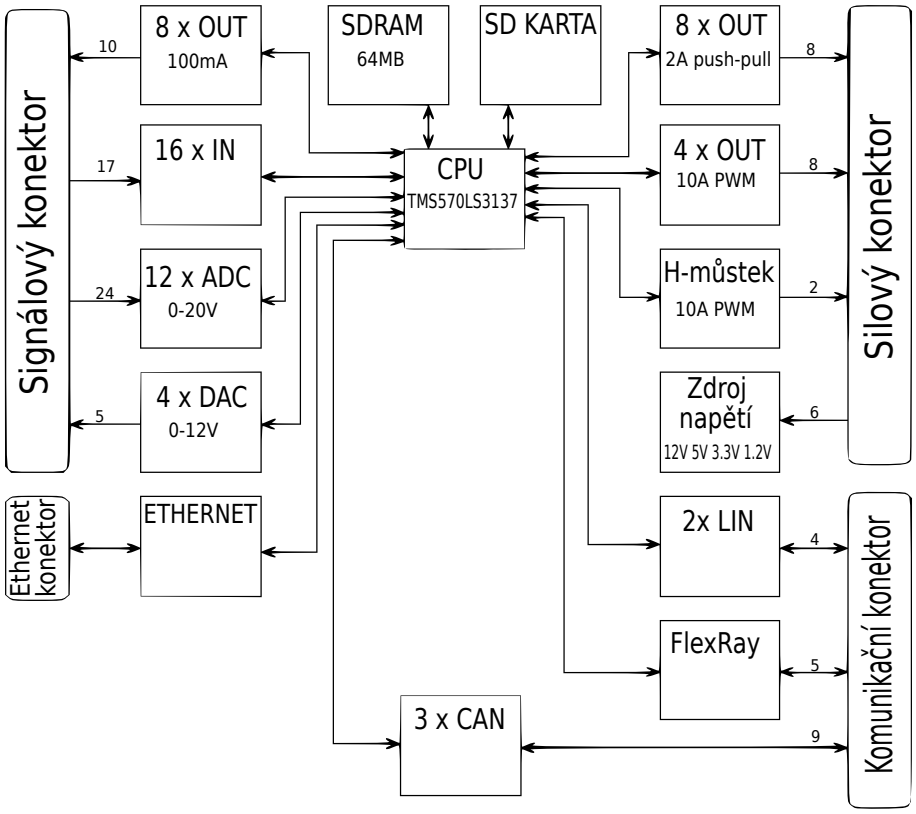
- Mikrokontrolér s dostatečným výkonem.
- Externí paměť SDRAM s maximální kapacitou 256 MB, rychlost není kritická.
- 6x výstup 10 A s PWM s proudovou ochranou a monitoringem.
- 6x výstup 2 A push-pull s diagnostikou a ochranou.
- 8x výstup 100 mA spínající 0/12 VDC.

- 16x vstup s přepínací úrovní 6 VDC.
- 2x LIN s funkcí wakeup.
- 4 kanálový DA převodník (0–12 V).
- 12 kanálový AD převodník (0–20 V).
- 3x CAN High speed s funkcí wakeup.
- 1x rozhraní Ethernet.
- 1x H-můstek s přepěťovou ochranou a diagnostikou.
- 1xFlexRay.
- 1x slot pro SD kartu.

Blokové schéma jednotky je uvedeno na obrázku 4.2

4.4 Použitý mikrokontrolér

JRVA je osazena mikrokontrolérem (MCU) TMS570LS3137 od firmy Texas Instruments postaveném na procesorovém jádru ARM® Cortex™-R4F. Jedná se o dvoujádrový lockstep systém, kdy obě jádra vykonávají paralelně stejnou instrukci, jejich výstup je na konci každého hodinového cyklu porovnán a v případě neshody je vyvoláno chybové přerušení s nejvyšší prioritou nebo restart systému. Pro odstranění důsledků vnějších vlivů jsou paralelně vykonávané instrukce oproti sobě posunuté o 1.5 taktu. Další bezpečnostní vlastností jsou měřící obvody přímo na čipu, které sledují distribuci hodin, napětí, signálu resetu a rozdíly teplot jednotlivých oblastí. MCU obsahuje hardwarový self-test pro CPU i SRAM, technologii pro opravu chyb v SRAM a FLASH pamětích přímo spojených s CPU a detektor selhání externího oscilátoru kdy dojde k zapojení záložního interního RC oscilátoru. Veškerý kód a data uložená v paměti FLASH i RAM jsou chráněna kontrolním součtem nebo kódem pro opravu chyb. Díky těmto a dalším bezpečnostním opatřením je možné používat TMS570LS3137 v aplikacích s vysokými požadavky na spolehlivost a bezpečnost, mezi které spadá i požadované použití pro řídicí jednotky v automobilech.



Obrázek 4.2: Blokové schéma JRVA

4.4.1 Parametry

- Maximální frekvence procesorového jádra: 180 MHz
- Flash: 3 MB
- RAM: 256 KB
- CPU napětí: 1.2 V
- IO napětí: 3.3 V
- 32-bit Big-endian

4.4.2 Architektura

MCU podporuje až 7 zdrojů hodin. Jeden nízkofrekvenční interní oscilátor určený pro generování periodických přerušení budících procesor, druhý vysokofrekvenční interní oscilátor používaný k monitorování hodin, dva externí zdroje hodin, dvě jednotky fázového závěsu (Phase Lock Loop) a jeden hlavní oscilátor. Hodiny jsou poté rozvedeny do 10 domén s konfigurovatelným kmitočtem.

Cortex-R4F používá 32 bitovou adresní sběrnici a dokáže tedy adresovat 4 GB velký adresní prostor, jehož rozdělení do oblastí je znázorněné na obrázku 4.3. Hlavní instrukční paměť začíná na adrese 0x00000000, kde se nachází ukazatel na první vykonanou instrukci (reset vector) a ve výchozím stavu začíná prostor pro datovou RAM paměť na adrese 0x08000000.

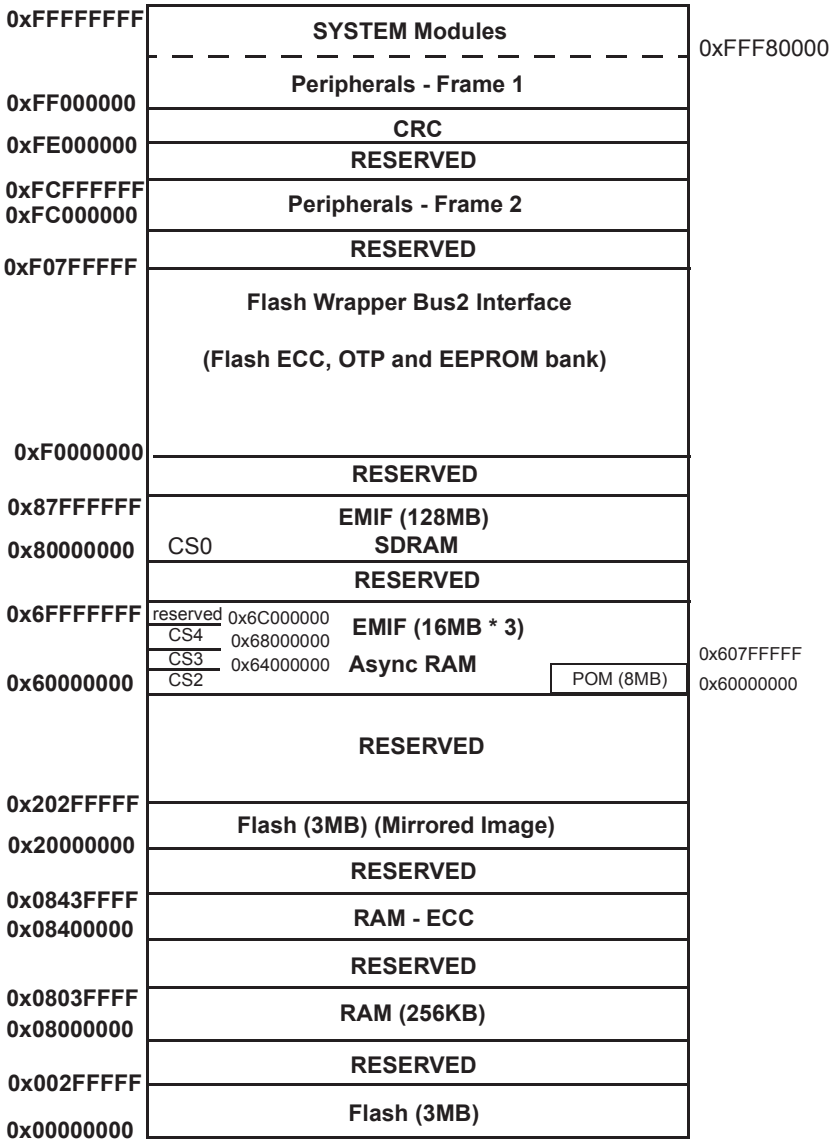
FLASH paměť je rozdělená na dvě banky o velikosti 1.5 MB. Banky se dále dělí na sektory o velikosti 128 KB, což je nejmenší oblast v bance, která může být samostatně vymazána. První čtyři sektory mají velikost 32 KB, banka určená pro emulaci EEPROM má sektory o velikosti 16 KB.

MCU poskytuje možnost provést hardwarovou inicializaci většiny pamětí na systémové a periferní sběrnici. Inicializace zabezpečí uvedení obsahu pamětí a systému pro opravu chyb do definovaného nechybového stavu.

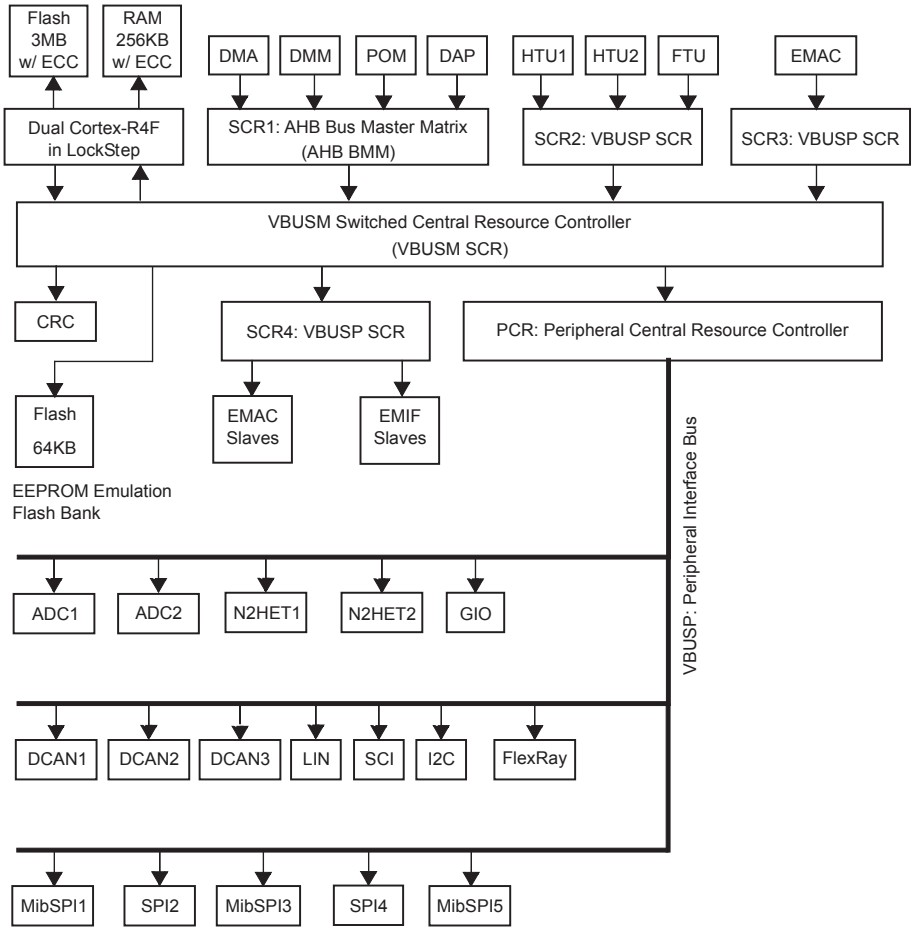
Celková architektura kontroléru včetně znázorněných periferií je zobrazena na obrázku 4.4.

4.5 Využití jednotky

Konkrétní účel a aplikace pro navrženou jednotku nebyl zadavatelem specifikovaný. Jednotka je však vybavena třemi druhy automobilových sběr-



Obrázek 4.3: Organizace paměti TMS570LS3137. [21, strana 112]



Obrázek 4.4: Schéma architektury TMS570LS3137. [21, strana 100]

nic, převodníky, výkonovými výstupy, budiči motorů a digitálními i analogovými vstupy, takže umožní použití v celé škále aplikací. Je vhodná pro účely testování jednotlivých automobilových dílů i celých soustav, může sloužit jako vývojová deska pro software do jiných jednotek podobného designu a v případě dostatečně robustního software splňujícího přísná bezpečnostní kritéria může sloužit i přímo jako jedna z řídících jednotek v automobilu.

5 Rozbor

5.1 Vývojové prostředí

Texas Instruments poskytuje pro své mikrokontroléry integrované vývojové prostředí pod názvem Code Composer Studio, aktuálně ve verzi 5. Jedná se o vývojové nástroje integrované do otevřeného vývojového prostředí Eclipse, což přináší výhodu v možnostech použití dalších doplňku, využití stability, rychlosti a pokročilých funkcí prostředí.

Code Composer Studio obsahuje především editor kódu s automatickým doplňováním kódu, kompilátory pro různé rodiny MCU od Texas Instruments a debugger. Velkou výhodou je také podpora operačního systému Linux i na straně hostitelského (vývojového) systému. Více informací, odkaz ke stažení a návod k použití naleznete ve zdroji [19].

Dalším nástrojem, který Texas Instruments poskytuje, je software HAL Code Generator. Jedná se o nástroj s grafickým rozhraním pro nastavení řídících registrů periferií, přerušení, hodin, konfigurace operačního systému a dalších parametrů. Po nastavení konfigurace umí nástroj vygenerovat inicializační a ovládací kód, který lze importovat do projektu Code Composer Studia.

HAL Code Generator je velice nápomocný při počátečním seznamování s jednotlivými periferiemi, ale pro profesionální nasazení je vhodné napsat vlastní drivery, protože vygenerovaný kód není zcela prostý chyb. Další informace a odkaz ke stažení je k nalezení ve zdroji [20].

Vývoj probíhá primárně na počítači s operačním systémem Linux a ke komunikaci s jednotkou přes sériovou linku byl použitý program GTKTerm. Jako systém pro správu verzí byl zvolen Git.

5.2 FreeRTOS

FreeRTOS je volně dostupný real-time operační systém. Jeho licence povoluje bezplatné použití i v komerčních projektech bez potřeby zveřejnění zdrojových kódů. Pro diplomovou práci je podstatné, že poskytuje fronty zpráv, semafory a mutexy (i s možností dědění priority) pro synchronizaci mezi úlohami (tasky) i mezi úlohou a obsluhou přerušení.

FreeRTOS byl zvolen pro vývoj z důvodů snadné instalace přímo z prostředí HAL Code Generator, který také umožňuje provádět příslušná nastavení.

5.3 CMDPROC

CMDPROC je příkazový interpretr naprogramovaný pro laboratorní a lékařské přístroje vyvíjené ve firmě PiKRON. Je uvolněný pod licencí MPL umožňující zahrnutí i do komerčního firmware třetích stran. Obsahuje rozhraní pro definici hardwarově specifických funkcí komunikace po sériové lince a rozhraní pro definici příkazů.

Rozhraní pro sériovou linku obsahuje ukazatele na funkce putc, getc, write a read pro odeslání znaků, přijmutí znaku, odeslání řetězce a přijmutí řetězce. Je volitelně dostupná také funkce echo, proto nejsou potřeba žádné vedlejší zásahy do datového proudu na sériové lince.

Rozhraní pro definici příkazů je tvořené seznamem deskriptorů příkazů, zakončeným prázdnou položkou (NULL). Seznam může také obsahovat vnořené podseznamy. Deskriptor příkazu obsahuje název příkazu, ukazatel na funkci s jeho rutinou, krátkou nápovědu a nastavení parametrů příkazu. Parametry mohou být pro čtení a/nebo zápis a mohou být od příkazu odděleny operačním znakem : (nastavení hodnoty) nebo ? (přečtení hodnoty). Funkce s rutinou příkazu musí být deklarovaná podle vzoru

```
int cmd_do_commandname(cmd_io_t *cmd_io, const struct cmd_des *des
    , char *param[])
```

Spolu s kódem zajišťujícím hlavní funkci CMDPROC, tj. dekódování a spouštění příkazů, přichází také sada funkcí pro zpracování parametrů příkazů a převodu čísla na text a zpět.

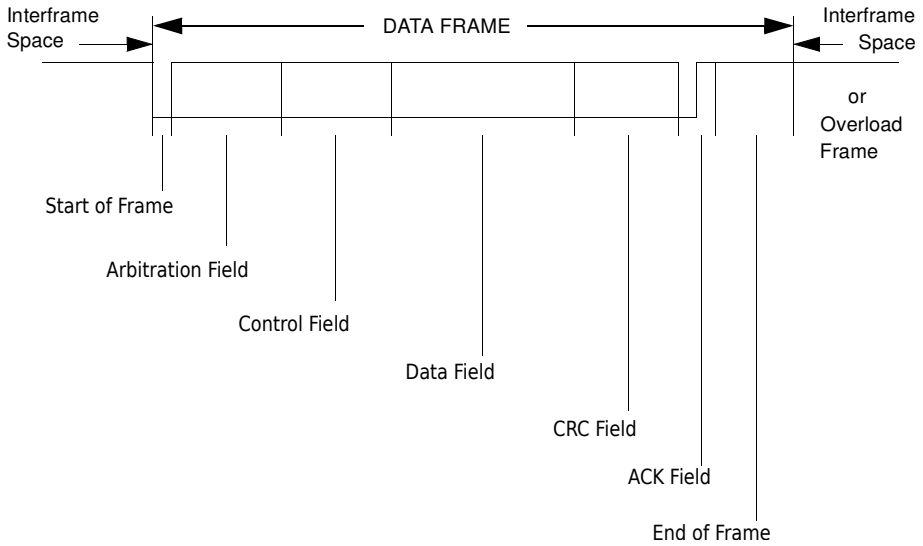
5.4 Sériová linka

Sériová linka je na JRVA zajišťována modulem SCI v mikrokontroléru. Jedná se o klasický komunikační standart UART s napěťovými úrovněmi LV-TTL. Pro komunikaci s počítačem, který využívá standart RS-232 s logickými úrovněmi ±12 V, je nutné použít převodník.

Sériová linka sdílí TX pin s modulem LIN2. Není proto možné používat obě komunikační rozhraní najednou.

5.5 CAN High speed

CAN (Controller Area Network) je komunikační protokol navržený pro vzájemnou komunikaci mezi mikrokontroléry v jednotlivých automobilových zařízeních. První verze specifikace byla vydána v roce 1986. Jedná se o sdílenou sběrnici s arbitrací přístupu k médiu založenou na prioritách zpráv.



Obrázek 5.1: CAN datový rámec [3, strana 10]

To znamená, že všechny uzly mohou začít vysílat na volnou sběrnici tím, že se odešlou nejprve bity identifikátoru (ID) zprávy. Uzel, jehož numerická hodnota identifikátoru je nejnižší, smí pokračovat ve vysílání. Takový uzel totiž na každé pozici, kde se vysílané bity identifikátorů liší, vysílá dominantní bit, který převáží nad vysíláním recesivního bitu jiné jednotky. Ta od dalšího vysílání odstoupí a počká na další příležitost. Z tohoto mechanismu vyplývá, že ID zprávy musí být v celém systému unikátní, jinak by mohlo dojít ke kolizím. Zprávy jsou na sběrnici adresovány svým určením (ID), nikoliv adresou cílového uzlu. Datový rámec je znázorněný na obrázku 5.1.

JRVA obsahuje tři vysokorychlostní (High speed) CAN linky, což odpovídá počtu modulů v mikrokontroléru. Fyzická vrstva je tvořena budiči NXP TJA1041AT [14], jenž jsou navržené pro použití v automobilech. S mikrokontrolérem jsou tyto budiče propojeny přímo datovými vodiči na piny MCU vyhrazené pro CAN a řídícími vodiči na GPIO piny.

5.6 Analogově-digitální převodník (ADC)

Mikrokontrolér obsahuje dva moduly ADC s architekturou postupné aproximace o volitelném rozlišení 10–12 bitů. Na vstupy procesoru AD1IN0–AD1IN14 jsou připojeny výstupy zesilovačů pro úpravu rozsahu a filtrování AD kanálů 1–15. Tyto pomocné obvody mají za cíl zvětšit rozpětí měřitelných napětí na 0–20 V.

5.7 Digitálně-analogový převodník (DAC)

DAC je řešený pomocí externích obvodů Microchip MCP4922-E/ST [12], připojených k MCU pomocí rozhraní SPI3. Jedná se o 12-bitový digitálně analogový převodník s nastavitelným rozsahem až na dvojnásobek referenčního napětí. Na výstupy převodníků jsou připojeny zesilovací obvody pro zvětšení výstupního rozsahu na 0–12 V.

5.8 Digitální vstupy (DIN)

JVRA je vybavena celkem 16 digitálními vstupy. Všechny jsou připojené k obvodu Freescale MC33972 [9], který komunikuje s MCU pomocí rozhraní SPI1. Stav na skupině vstupů DIN8–DIN15 je navíc možné sledovat přímo na tomu vyhrazených GPIO pinech MCU. Vstupy ve výchozím nastavení detekují sepnutí vůči 12 V, ale volitelně lze nastavit prvním 8 vstupům detekci vůči zemi.

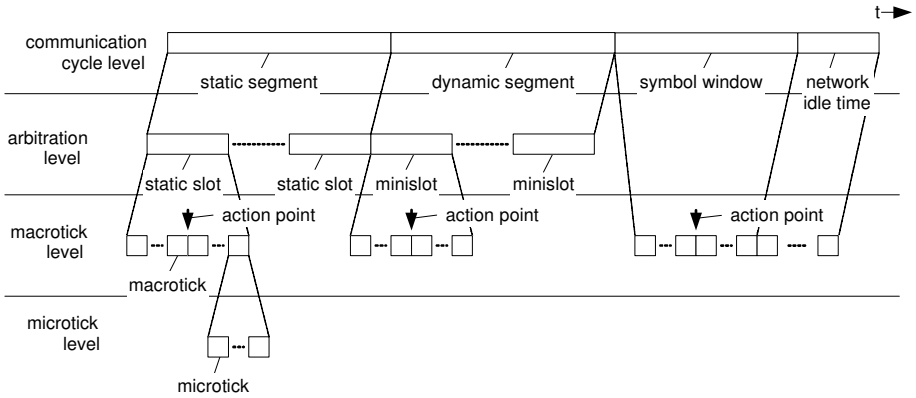
5.9 FlexRay (FRAY)

FlexRay je komunikační protokol, vyvinutý FlexRay konsorciem v roce 1999, určený především pro použití v automobilovém průmyslu. Oproti sběrnici CAN je rychlejší, spolehlivější a také dražší. Pro vyšší spolehlivost umožňuje sběrnice propojení uzlů dvěma fyzickými kanály. Komunikace po síti FlexRay probíhá v opakujících se komunikačních cyklech, které obsahují statický a dynamický segment. Struktura komunikačního cyklu je na obrázku 5.2 na následující straně.

Statický segment se dělí na statické sloty, kdy má každý uzel v síti přiděleny své vlastní sloty, ve kterých posílá data. Díky tomu, že se periodicky opakují v přesně definovaném časovém rozmezí, je tato část komunikačního cyklu deterministická a vhodná pro posílání kritických dat.

Dynamický segment se dělí na dynamické sloty a nemá předem určený čas odeslání rámce – může dojít k odložení vysílání, když se již nevejde do komunikačního cyklu. Tento segment se tudíž hodí na přenos běžných dat, u kterých záleží méně na čase odvysílání.

Implementace FlexRay na jednotce umožňuje využívat oba kanály A i B. Fyzická vrstva je tvořena dvěma budiči NXP TJA1082 [15], které jsou propojeny s mikrokontrolérem přímo datovými vodiči a navíc umožňují číst stav pomocí rozhraní SPI4. Žádnou konfiguraci budič fyzické vrstvy po SPI neumožňuje.



Obrázek 5.2: FlexRay komunikační cyklus [7, strana 100]

5.10 H-můstek (HBR)

Jako kontrolér H-můstku tvořeného tranzistory MOSFET byl zvolen obvod STM L99H01 [17]. Tento obvod má připojeny signály pro řízení PWM, směru a povolovací signál přímo k mikrokontroléru na piny GPIO. Konfigurace můstku a čtení stavu je možné přes rozhraní SPI4.

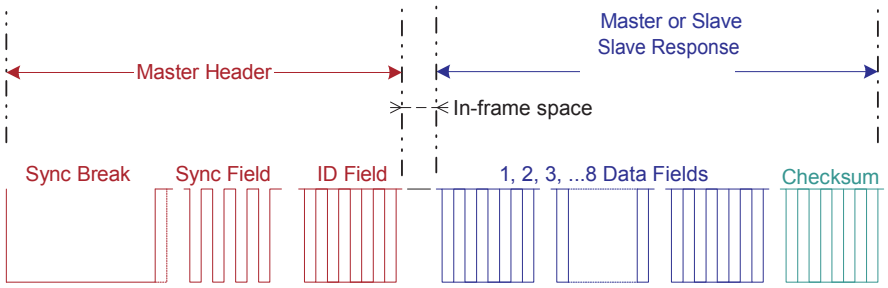
Pro správnou funkčnost je potřeba opakovaně s periodou maximálně 60 ms posílat obvodu příkaz pro reset watchdogu a před zapnutím nastavit teplotní senzor tak, aby nehlásil chybu za běžné provozní teploty.

5.11 Výstup do 10 A (HOUT)

Jednotka pro rychlý vývoj aplikací má 6 výkonových výstupů 0–12 V pro spínání až 10 A zátěže, nejlépe indukčního charakteru. Pro tyto účely byly zvoleny obvody International Rectifier AUIR33401S [10]. S MCU je každý z nich propojený vodiči na dva GPIO piny a jeden kanál AD převodníků pro zpětnou vazbu. Jeden GPIO pin je použit jako vstupní signál pro obvod a určuje, zda je výstup aktivní nebo neaktivní. Druhý GPIO pin slouží pro zjištění stavu výstupu. V případě bezchybného provozu kopíruje stavový pin signál na vstupním pinu. Pokud obvod detekuje chybu, přepne se do chybového stavu a začne periodicky stahovat vstupní signál k zemi.

5.12 Výstup do 2 A (MOUT)

Dalších 6 výstupů 0–12 V pro maximálně 2 A je realizováno pomocí tří obvodů STM VNH5019A [18]. Každý z nich je propojen s MCU přímo na



Obrázek 5.3: LIN rámec [21, strana 1433]

GPIO piny. První dvojice pinů aktivují nebo deaktivují výstup, druhá dvojice slouží pro ověření stavu nebo nouzové vypnutí výstupu.

5.13 Výstup do 100 mA (LOUT)

Výstupy 0–12V pro 100 mA jsou tvořeny dvojicí obvodů ON Semiconductor NCV7608DWRG2 [16], připojených k MCU pomocí SPI1 pro konfiguraci i čtení stavu. Obvody jsou na SPI zřetězeny za sebou.

5.14 LIN

LIN (Local Interconnect Network) je protokol vyvinutý pro komunikaci mezi automobilovými komponentami a má spolu s rozhraním CAN, FlexRay a dalšími robustními protokoly tvořit hierarchii a propojovat nekritické komponenty na spodních úrovních hierarchie. Na síti není potřeba žádná arbitrace, protože veškerou komunikaci vždy zahajuje uzel master. Jeden rámec se skládá z hlavičky, kterou vždy posílá master, a z odpovědi, kterou může posílat slave i master sám sobě. Detailní struktura rámce je patrná z diagramu na obrázku 5.3.

Na jednotce jsou dostupné linky LIN1 a LIN2. Pro obě je fyzická vrstva tvořena obvody NXP TJA1021T [13], které jsou propojeny datovými vodiči a kontrolním vodičem přímo s mikrokontrolérem.

LIN2 sdílí jeden datový vodič (TX) se sériovou linkou použitou pro komunikaci s počítačem a není proto možné ji v současné chvíli použít.

5.15 Napájecí okruhy (VBAT)

Některé periferie na jednotce potřebují stabilní napájecí napětí, přičemž jejich odběr může kolísat podle připojené zátěže a jejího spínání. Pro vyrovnávání kolísání napájecího napětí a potlačení proudových rázů existují na desce tři napájecí okruhy, každý se svými vlastními kapacitory, sloužícími jako filtr pro daný okruh. Napěťové okruhy je možné z důvodu úspory elektrické energie zapínat a vypínat pomocí signálů VBATx_EN a jedním globálním signálem VBAT_EN. Všechny tyto signály vycházejí přímo z GPIO pinů mikrokontroléru a ovládají stejné budící obvody, jaké jsou použity pro výstupy HOUT, popsané v sekci 5.11.

6 Návrh testovacího software

6.1 Cíle návrhu

Během návrhu byl kladen důraz na znovupoužitelnost kódu, modularitu a oddělení programátora vyšších vrstev, například příkazů, od implementačních detailů jednotlivých periferií.

Modularita kódu je zajištěna třemi způsoby.

1. Oddělením nízkoúrovňových (low-level) driverů mikrokontroléru od rozhraní pro ovládání konkrétních JRVA periferií.
2. Separací příkazového interpretru od příkazů pro ovládání periferií.
3. Izolováním jednotlivých sad příkazů periferií.

V zájmu znovupoužitelnosti v budoucích projektech využívajících Matlab/Simulink a oddělení programátora příkazů od hardwarových detailů je pro ovládání JRVA periferií navrženo rozhraní, tvořící abstrakční vrstvu (HAL). Tato vrstva zajistí jednotné ovládání vstupně výstupních pinů nezávisle na tom, ke které periferii MCU (NHET, GPIO, DMM) nebo expandéru SPI jsou připojeny.

6.2 Architektura firmware

Architektura software z pohledu jednotlivých vrstev je znázorněna na obrázku 6.1. Nejspodnější vrstva je tvořena samotným hardwarem mikrokontroléru. Hardware je přímo ovládán low-level drivery, které poskytují rozhraní pro základní inicializaci vstupně výstupních pinů, sběrnic a komunikačních modulů. Poskytují také funkce pro základní ovládání periferie a obsluhu přerušení. Přímé ovládání jednotlivých JRVA periferií pomocí těchto nízkoúrovňových funkcí se liší a někdy je i značně složité, protože některé z nich jsou připojeny na SPI rozhraní MCU, některé pomocí GPIO a některé periferie využívají kombinaci několika různých GPIO portů a SPI zároveň.

Pro pohodlnější ovládání byla navržena abstrakční vrstva (HAL). V ní jsou namapovány piny periferií jednotky JRVA na piny jednotlivých portů mikrokontroléru. Jsou zde definovány skupiny společných vstupně-výstupních pinů tvořících port a k nim jsou přiřazeny ovládací funkce podle způsobu připojení ke kontroléru. Tato vrstva je základním stavebním kamenem oddělujícím programátora od hardwaru a důkladnější popis naleznete v sekci 6.6.

Protože různé porty jednotky poskytují různé funkce, existuje vrstva vysokoúrovňových ovladačů (high-level drivers). V ní se nalézají sady metod pokrývající možnosti jednotlivých periferií. Například HOUT umožňuje diagnostiku, řízení výstupu pomocí pulsní modulace a čtení analogové hodnoty o protékajícím proudu. Proto obsahuje jeho high-level driver metody pro obsluhu právě jen těchto funkcí. Oproti tomu DAC dovoluje nastavovat každému svému pinu výstupní napětí, zapínat je, vypínat a číst stav budiče. Proto jeho high-level driver obsahuje úplně jinou sadu metod, poskytující rozhraní pro tyto funkce.

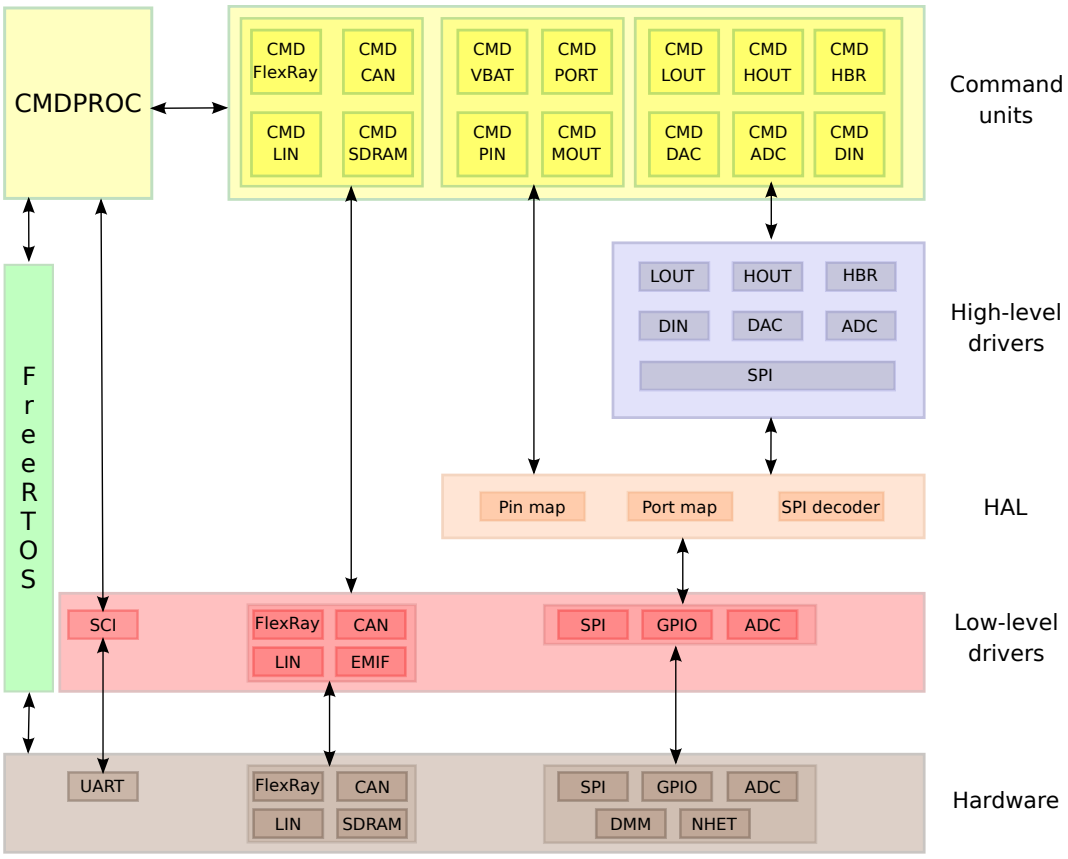
Každá JRVA periferie má svou vlastní sadu příkazů pro interpretr (Command units), využívajících příslušný high-level driver. Jednotlivé sady příkazů jsou na sobě nezávislé a lze je tedy přidávat a odebírat dle aktuálních potřeb. Všechny příkazy jsou sjednoceny v kořenovém seznamu příkazů a zastřešuje je samotný interpretr příkazů CMDPROC.

6.3 Jmenné konvence

6.3.1 Názvy souborů se zdrojovými kódy

Pro rozlišení příslušnosti zdrojových kódů k jednotlivým vrstvám architektury jsou zavedeny konvence pro prefixy názvů souborů s kódy.

- Low-level drivery mají prefix *ti_drv_*, například driver pro ovládání GPIO je v souboru ti_drv_gio.c.
- Soubory pro hardwarovou abstrakční vrstvu mají prefix *hal_*, například definice JRVA portů se nachází v souboru hal_port_def.c.
- High-level drivery mají prefix *drv_*, například driver pro ovládání periferie HOUT je v souboru drv_hout.c.
- Sada příkazů pro ovládání jedné periferie se nachází v souboru s prefixem *cmd_*. Například opět periferie HOUT má všechny své příkazy v souboru s názvem cmd_hout.c.
- Zdrojové kódy pro CMDPROC používají prefix *cmdproc_*.
- Soubory operačního systému FreeRTOS jsou odlišeny od ostatních prefixem *os_*.
- Soubory s konfigurací mikrokontroléru, prováděné vždy po resetu, se nachází v souborech s prefixem *sys_*



Obrázek 6.1: Architektura firmware

Některé soubory se vymykají výše zmíněným konvencím. Tyto obsahují kód pomocných funkcí a nebo poskytují funkcionalitu nezapadající do schématu. Zdrojové kódy jsou však bohatě komentované, takže by neměl být problém se v nich orientovat.

6.3.2 Názvy funkcí

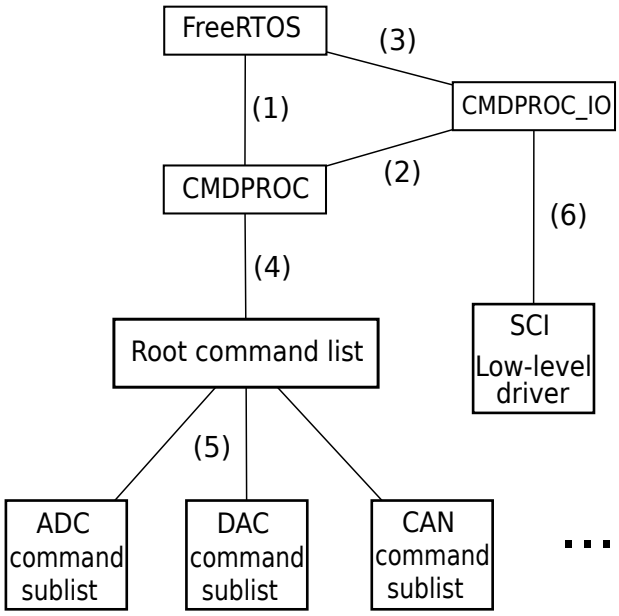
- Funkce s rutinami pro příkazy začínají vždy prefixem *cmd_do_*.
- Funkce high-level driverů mají vždy na začátku prefix odpovídající jménu ovládané periferie, následovaný prováděnou akcí. Například funkce pro přečtení hodnot na analogově digitálním převodníku se jmenuje adc_get_port_val, funkce pro nastavení výstupního napětí na pinu digitálně analogového převodníku se jmenuje dac_set_pin_voltage.
- Hardwarová abstrakční vrstva obsahuje dvě sekce. První mapuje piny procesoru na piny periferií jednotky, druhá sjednocuje piny periferií do celků, tvořících porty.
  - Funkce pro správu a ovládání namapovaných pinů používají prefix *hal_gpio_pin*.
  - Funkce pro ovládání vytvořených portů používají prefix *hal_gio_port* nebo *hal_spi_port* podle způsobu připojení k MCU.
- Funkce low-level driverů začínají vždy názvem periferie, kterou ovládají.

6.4 Sériová linka a CMDPROC

Provázání knihovny CMDPROC s operačním systémem FreeRTOS a modulem mikrokontroléru SCI je ilustrováno obrázkem 6.2 a k tomuto obrázku se také vztahuje další výklad v této sekci.

Blok CMDPROC obsahuje hlavní rutinu, která zpracovává přijaté příkazy a spouští k nim přiřazené funkce. Vzhledem k tomu, že je použit operační systém, běží tato rutina ve své vlastní úloze (task), spravované FreeRTOSem, což naznačuje spojnice (1).

Při vývoji interpretru se počítalo s nasazením na různých mikrokontrolérech a proto je připraveno obecné rozhraní pro vstupně výstupní funkce. Toto rozhraní je znázorněno na obrázku spojnicí (2) mezi moduly CMDPROC a CMDPROC_IO.



Obrázek 6.2: Připojení CMDPROC k sériové lince

V modulu CMDPROC_IO jsou definovány buffery a funkce pro příjem a odesílání znaků. Ty využívají služeb operačního systému (3), konkrétně semaforů a front zpráv, které je možné používat pro synchronizaci mezi úlohami, ale i mezi úlohou a obsluhou přerušení.

Pro definici příkazů je modulem CMDPROC specifikováno další rozhraní. Dle něho je vytvořen kořenový seznam příkazů (4), do kterého jsou formou podseznamů vnořeny jednotlivé sady příkazů konkrétních periferií (5).

Vstupně výstupní modul CMDPROC_IO již komunikuje přímo s low-level driverem SCI (6) a detailní informace o implementaci jsou k dispozici v sekci 7.1.

6.5 FlexRay

Pro otestování funkčnosti sítě FlexRay je potřeba umožnit příjem a odesílání testovacích dat. Tento zdánlivě jednoduchý úkol sebou nese mnoho práce, protože je potřeba síť správně nakonfigurovat a nastartovat. K tomu slouží low-level driver, obsahující několik hlavních funkcí. Obrázek 6.3 znázorňuje propojení základních potřebných bloků, které jsou tvořeny funkcemi driveru.

Každý uzel sítě musí mít shodně nastavené globální parametry – časování sítě, poměr statických slotů vůči dynamickým atp. Jednotlivé uzly se potom liší tím, ve kterém slotu posílají své rámce a ve kterých rámcích očekávají data, která chtějí přijímat. Mikrokontrolér tuto záležitost řeší zavedením TX a RX bufferů, sdílejících jednu message RAM. Vzhledem k tomu, že na FlexRay bude kladen důraz v budoucích pracích na projektu JRVA, je součástí návrhu sada obecných funkcí pro konfiguraci těchto bufferů.

Zadávání dat k odeslání se provádí jejich zápisem do části v message RAM, vyhrazené pro TX buffer během konfigurace. Naopak přečtení přijatých dat se provádí z části message RAM vyhrazené pro RX buffer. Obě činnosti budou potřeba i později, proto návrh driveru počítá s příslušnými obecnými funkcemi.

Při spuštění FlexRay zařízení je potřeba buď vytvořit novou FlexRay síť a nebo se zapojit do již existující a běžící sítě. Postup pro iniciaci sítě nebo integraci do sítě je podrobně popsán ve specifikaci protokolu FlexRay [7] a kód pro JRVA projekt je převzat z bakalářské práce pana Zemana [23].

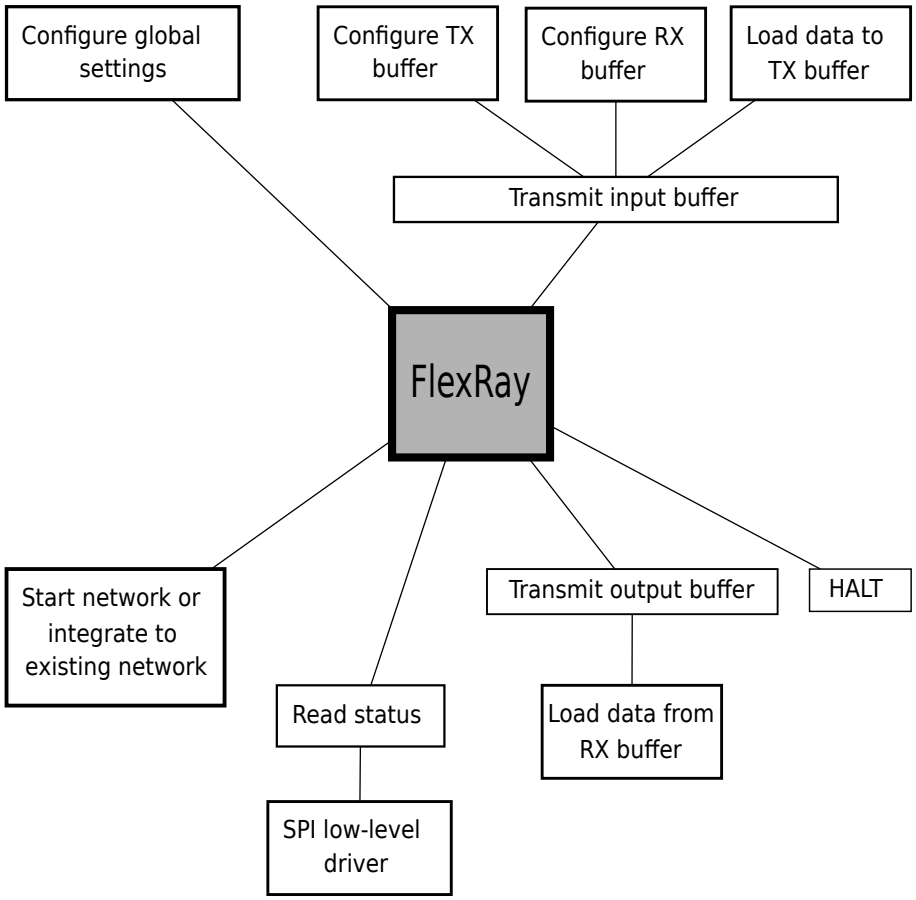
Po skončení testu je potřeba běh sítě zastavit, čehož lze docílit odesláním příkazu HALT FlexRay modulu Protocol Operation Control (POC).

Budič fyzické vrstvy je schopný poskytovat informace prostřednictvím rozhraní SPI a proto je součástí driveru i funkce pro její získání.

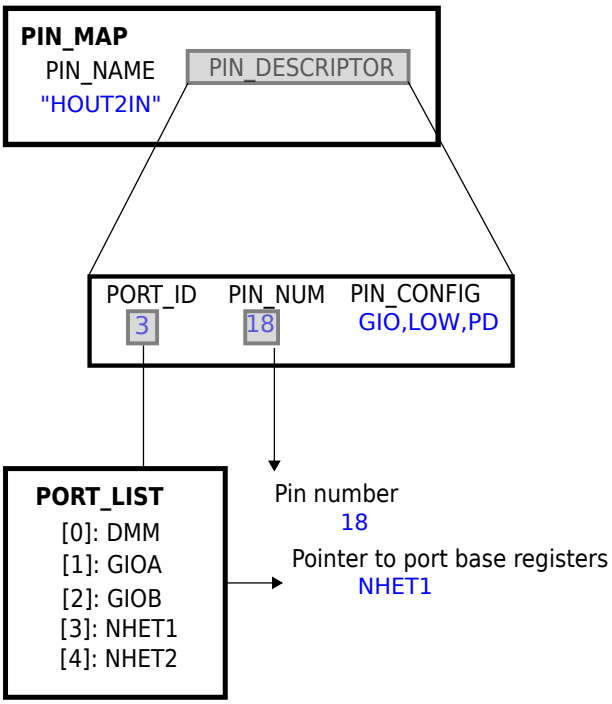
6.6 Obecné rozhraní pro ovládání pinů

Vzhledem k tomu, že jednotka obsahuje velké množství vstupních a výstupních pinů ovládaných pomocí GPIO, je nutné umožnit přístup i pomocí tohoto režimu k většině pinům z portu DMM a NHET. Bez zavedení jednotné identifikace pinů by byl kód nepřehledný. Nepřehlednost kódu a složitost ovládání dále roste se začleněním kanálů AD převodníku a pinů ovládaných pomocí SPI.

Prvním úkolem tedy je vytvořit mapu pinů JRVA periferií na čísla pinů a porty mikrokontroléru. Schéma řešení s modře vyznačeným příkladem pro jeden pin je na obrázku 6.4. Piny v seznamu PIN_MAP jsou definovány názvem pinu JRVA periferie PIN_NAME a deskriptorem PIN_DESCRIPTOR. Deskriptor obsahuje index portu v tabulce portů PORT_LIST, číslo pinu v portu a konfiguraci pinu (vstup, výstup, push-pull, open-drain atd.). Tabulka PORT_LIST obsahuje ukazatele na bázové adresy registrů jednotlivých portů. Díky tomuto schématu lze získat dotazem na jméno pinu ukazatel na příslušný port a pozici pinu v rámci portu. Navíc je možné definovat funkce pro konfiguraci i ovládání pinu podle jeho jména na jednotce



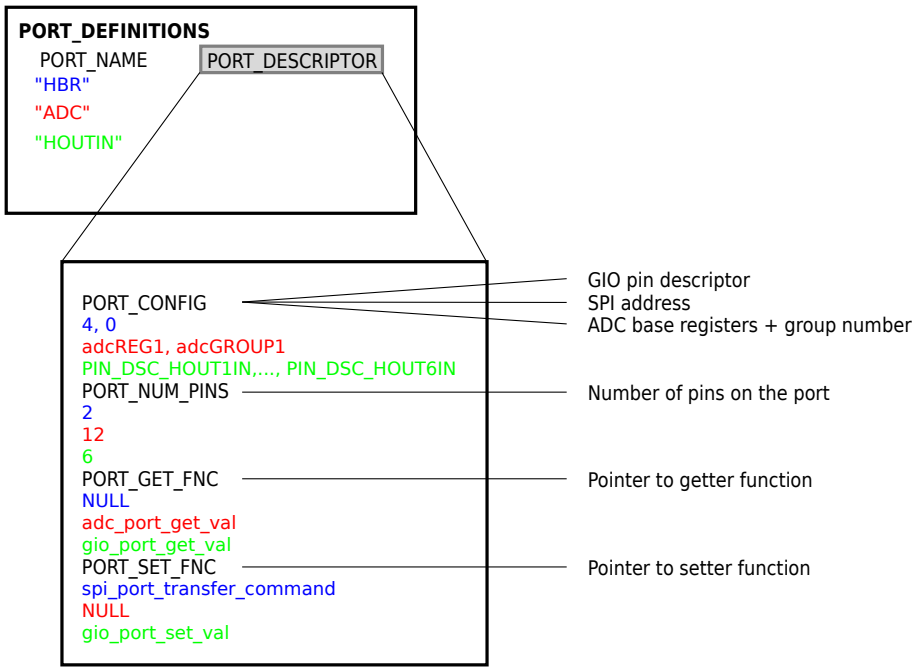
Obrázek 6.3: Schéma FlexRay driveru



Obrázek 6.4: Schéma řešení HAL pro piny

a mít tak veškerou logiku pro GPIO piny v kódu na jednom místě.

Druhým úkolem je skupinu pinů sloučit do celého portu s využitím mapy pinů z předchozího odstavce. Proto jsou navrženy metody pro ovládání pinů pomocí SPI a podobným způsobem i ovládání kanálů AD převodníku. Tím je docíleno logického oddělení od vnitřní struktury hardware. Obrázek 6.5 obsahuje schéma řešení portů. Každý port v seznamu PORT_DEFINITIONS je definován svým jménem a deskriptorem. První položka deskriptoru portu je PORT_CONFIG, určující způsob přístupu k pinům portu (seznam deskriptorů GPIO pinů, SPI adresa, bázová adresa a číslo skupiny AD převodníku). Druhá položka PORT_NUM_PINS znamená počet vstupních nebo výstupních pinů na portu. Třetí a čtvrtá položka jsou ukazatele na funkce pro čtení a zápis dat na port. Na obrázku 6.5 jsou barevně vyznačené tři porty. Modrý pro přístup k SPI periferii, červený pro AD převodník a zelený pro port připojený na GPIO.



Obrázek 6.5: Schéma řešení HAL pro porty

6.7 Rozhraní pro ovládání PWM

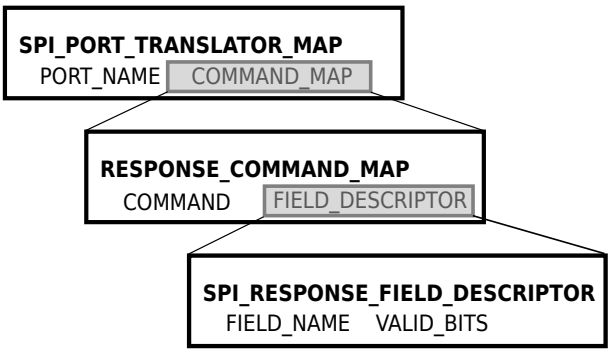
Periferie HBR a HOUT mají umožňovat regulaci výstupu pomocí pulsní modulace. Mikrokontrolér nabízí moduly pro snadnou implementaci této techniky. Je potřeba jen správně naprogramovat paměť pro NHET.

Návrh driverů vyšší úrovně (high-level) obsahuje metody pro nastavení periody a šířky aktivní části pulzů, funkci pro spuštění a zastavení pulzů. Funkce v této vrstvě využívají low-level driver pro NHET, vygenerovaný programem HAL Code Generator.

6.8 Rozhraní pro přístup k SPI

Některé periferie připojené k mikrokontroléru rozhraním SPI je možné konfigurovat zasíláním příkazů, z některých je možné číst stavové registry a některé umožňují obojí. Pro posílání dat a přijímání odpovědí je implementován low-level driver, jehož autorem je Ing. Rostislav Lisový.

V zájmu pohodlného ovládání periferie high-level ovladač obsahuje ke každé periferii sadu funkcí pro sestavení příkazu a funkci pro start pře-



Obrázek 6.6: SPI dekodér odpovědi, struktura map

nosu. Například DA převodník umožňuje nastavit jedním příkazem napětí na všech pinech a zapnout nebo vypnout piny. Jeho high-level driver tedy obsahuje analogické funkce, jejichž úkolem je pouze sestavit příkaz tak, aby při odeslání provedl pouze požadované změny. Podobný vzor sledují i ostatní SPI periferie.

V navazujících projektech bude pravděpodobně rozšířen low-level driver tak, aby přenos dat probíhal na pozadí. Toho se docílí využitím multibufferového SPI, který je součástí mikrokontroléru. Při návrhu byl tento požadavek brán v potaz a přenos dat je implementován pomocí dvojice proměnných command a command_sh, jejichž obsah se před zahájením přenosu prohazuje.

Pro zvýšení komfortu při čtení stavových registrů je implementován dekodér přijaté odpovědi do tabulky s textovými popisy. Je tvořen mapou názvů polí a čísel k nim přiřazených bitů v kódu přijaté odpovědi. Mapa umožňuje vyhledat k danému jménu portu z abstrakční vrstvy příslušnou periferii. Situaci mírně komplikuje fakt, že některé periferie poskytují různé odpovědi na různé příkazy. Proto je nutné zavést ještě prostředníka, který přiřadí mapu popisků a bitů k různým třídám příkazů dané periferie. Tento návrh ilustruje obrázek 6.6.

7 Implementace

Vlastní podpora velkého množství rozličných periferií představuje značné množství kódu. Proto budou v dalších sekcích popsané jen principy a vzorové příklady jednotlivých částí.

7.1 Navázání CMDPROC na sériovou linku

7.1.1 CMDPROC - IO interface

V souboru *cmdproc.h* se nachází definice struktury *cmd_io,* která tvoří rozhraní pro připojení interpretru příkazů k různým vstupně výstupním zařízením.

```
typedef struct cmd_io{
    int (*putc)(struct cmd_io *cmd_io,int ch);
    int (*getc)(struct cmd_io *cmd_io);
    int (*write)(struct cmd_io *cmd_io,const void *buf,int count);
    int (*read)(struct cmd_io *cmd_io,void *buf,int count);
    union {
        struct {
            ed_line_buf_t *in;
            ed_line_buf_t *out;
            struct cmd_io *io_stack;
        } ed_line;
        struct {
            long pos;
            void *ptr;
        } device;
    } priv;
} cmd_io_t;
```

Základem rozhraní jsou ukazatele na funkce pro vstup a výstup znaků a celých řetězců. Doplňkem potom ukazatele na vstupní a výstupní řádkové buffery, které CMDPROC používá interně pro zpracování příkazů.

V souboru *cmdproc_io_tisci.c* jsou do této struktury přiřazeny vstupně výstupní funkce pro SCI modul, tedy sériovou linku standardu UART. Jedná se o funkce:

```
int tisci_putc(cmd_io_t *cmd_io, int ch);
int tisci_getc(cmd_io_t *cmd_io);
int tisci_write(cmd_io_t *cmd_io, const void *buf, int count);
int tisci_read(cmd_io_t *cmd_io, void *buf, int count);
```

Tyto funkce vnitřně používají volání obecných funkcí print a read.

```
portBASE_TYPE print(const uint8_t * string);
portBASE_TYPE read(uint8_t * ch);
```

Funkci print je možné používat také uvnitř příkazových rutin pro výpis údajů. Všechny tyto funkce jsou definované v tomto souboru.

7.1.2 Buffery pro příjem a odesílání

Funkce print a read používají pro správu dat buffery. Využívají se zde vlastnosti front zpráv operačního systému FreeRTOS, kdy jedna fronta slouží jako vstupní buffer a druhá fronta jako výstupní buffer. Výhodou této implementace je, že v případě čekání na příjem nebo odeslání znaku je možné zablokovat jednoduše úlohu ve které běží interpretr a uvolní se tím prostor případným jiným úlohám, které mohou v budoucnosti vzniknout.

7.1.3 Odesílání dat

Po zavolání funkce print se nejprve zkontroluje, zda ještě probíhá nějaké odesílání na pozadí (v obsluze přerušení). Pokud ano, začnou se nové znaky jednoduše přidávat do výstupní fronty. V případě plné fronty dojde k zablokování úlohy, dokud se neuvolní místo odesláním prvního znaku ve frontě.

Pokud žádný přenos neprobíhá, musí se nejprve iniciovat vyvoláním nového TX přerušení odesláním prvního znaku. Od této chvíle zbytek odesílání probíhá v jeho obslužné rutině, kde se vybírají a odesílají znaky z fronty. Pokud v ní již není žádný znak k odeslání, přerušení se deaktivuje a příštím voláním funkce print se musí znovu iniciovat.

7.1.4 Přijímání dat

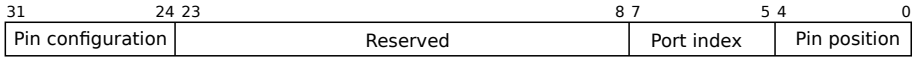
Zavoláním funkce read dojde k vyzvednutí znaku ze vstupní fronty, pokud nějaký existuje. Funkce je blokující, takže čeká, dokud nějaký znak nedorazí.

Samotné přijímání dat ze sériové linky probíhá v obsluze RX přerušení, které je vyvolané vždy při přijmutí nového znaku. Kód obsluhy vloží nově přijatý znak do vstupní fronty. V případě, že je fronta plná dojde k zahození znaku a inkrementaci čítače chyb.

7.2 Hardwarová abstrakční vrstva

7.2.1 Mapy pinů

V souboru *hal_gpio_tms570_def.h* se nachází definice maker pro jména pinů a pro jejich deskriptory. Deskriptor pinu je 32 bitová hodnota, rozdělená do oblastí znázorněných obrázkem 7.1.



Obrázek 7.1: Bitové schéma deskriptoru pinů

Pin position Určuje pozici pinu (kontrolního bitu) v portu,

Port index určuje index ukazatele na bázovou adresu portu v seznamu portů,

Pin configuration definuje nastavení pinu.

Definice deskriptoru přímo hodnotou by byla pro programátory matoucí, proto existuje makro PORT_PIN definované:

```
#define PORT_PIN(p,n,conf)  (((p)<<5) | (n) | (conf))
```

V tom samém hlavičkovém souboru je také deklarace struktury *pin_map_element* pro jeden záznam mapy pinů. Obsahuje dvě položky.

pin_name Ukazatel na řetězec s názvem pinu, podle kterého je možné později pin najít,

pin_desc deskriptor pinu.

V souboru *hal_gpio_tms570_def.c* je možné nalézt definici seznamu portů, tvořeného polem ukazatelů na bázovou adresu registrů portu.

```
gioPORT_t* port_id_map[MAX_PORT_CNT] = {
    (gioPORT_t*) dmmPORT,
    (gioPORT_t*) gioPORTA,
    (gioPORT_t*) gioPORTB,
    (gioPORT_t*) hetPORT1,
    (gioPORT_t*) hetPORT2
};
```

Také je zde definice mapy pinů MCU na piny jednotky. Mapa pinů je klasické pole struktur *pin_map_element*.

```
pin_map_element_t pin_map[MAX_PIN_CNT] = {
    /* DMM pins */
    { .pin_name = PIN_NAME_FANCTRL, .pin_desc = PIN_DSC_FANCTRL },
    { .pin_name = PIN_NAME_ETHRST,  .pin_desc = PIN_DSC_ETHRST },
    { .pin_name = PIN_NAME_VBAT1EN, .pin_desc = PIN_DSC_VBAT1EN },
    ...
    { .pin_name = PIN_NAME_UNUSED,  .pin_desc = PIN_DSC_HET2_UNUSED(30) },
    { .pin_name = PIN_NAME_UNUSED,  .pin_desc = PIN_DSC_HET2_UNUSED(31) }
};
```

Funkce pro ovládání pinů se nalézají v souborech *hal_gpio_tms570.h* a *hal_gpio_tms570.c*. Základem vyhledávání pinu podle jména je funkce

hal_gpio_pin_get_dsc. Funkce očekává jako argument ukazatel na řetězec se jménem vyhledávaného pinu a délku řetězce, pokud není ukončen znakem \0. Při nalezení pinu vrací ukazatel na deskriptor, pokud pin daného jména neexistuje, vrací NULL. Funkce jednoduše prochází lineárně polem, dokud nenajde požadované jméno nebo nedojde na konec.

Další velmi důležité funkce jsou *hal_gpio_pin_set_value* pro nastavení a *hal_gpio_pin_get_value* pro čtení hodnot z pinů daného deskriptorem. Obě adresují pin už pomocí deskriptoru předaného jako parametr.

Funkce pro konfiguraci pinu jsou již také připraveny, přestože nejsou v projektu zatím využívány a konfigurace je svěřena low-level driveru portu. U pinu je možné konfigurovat směr (vstup nebo výstup), zakončení signálu rezistorem (pull-up, pull-down, enabled, disabled), kolektor, funkci a počáteční hodnotu. Pro každý z těchto parametrů je definována speciální funkce a všechny parametry jsou zastřešeny jednou funkcí *hal_gpio_pin_conf_set*, která dokáže nastavit parametry najednou.

7.2.2 Mapy portů

Definice struktur potřebných pro vytvoření seznamu portů se nachází v souboru *hal_port_def.h*. Pro vytvoření deskriptoru portu je zde deklarovaná struktura *port_desc_st* s položkami:

config ukazatel na seznam pinů GPIO portu, adresu SPI nebo adresu a číslo kanálu ADC,

numValues počet pinů na portu, velikost SPI dat v bajtech nebo počet kanálů ADC,

port_getfnc_ptr ukazatel na funkci pro čtení hodnot z portu,

port_setfnc_ptr ukazatel na funkci pro nastavení hodnot na port.

Pro větší přehlednost jsou pro naplnění deskriptoru každého portu připravena makra. Pro každý GPIO port existují makra s polem deskriptorů pinů, počtem pinů a ukazateli na *hal_gpio_set* a *hal_gpio_get* funkce.

```
#define PORT_CFG_HOUTIN { PIN_DSC_HOUT1IN, PIN_DSC_HOUT2IN, PIN_DSC_HOUT3IN
    , PIN_DSC_HOUT4IN, PIN_DSC_HOUT5IN, PIN_DSC_HOUT6IN }
#define PORT_NV_HOUTIN  6
#define PORT_GFC_HOUTIN &hal_gio_port_get_val
#define PORT_SFC_HOUTIN &hal_gio_port_set_val
```

Pro SPI existuje *config* makro jako pole, jehož první položka je číslo SPI a druhá položka číslo obvodu. Další makro určuje počet bajtů příkazu a odpovědi pro periferii, ukazatel pro get funkci je vždy NULL, protože odpověď z periferie se získává vždy hned při odesílání příkazu, tedy voláním

funkce *hal_spi_port_transfer_command* ze souboru *hal_port_spi.c*, na kterou se nastavuje ukazatel set funkce ve struktuře deskriptoru. Příklad maker pro SPI port:

```
#define PORT_CFG_HBR    { 4, 0 }
#define PORT_NV_HBR     2
#define PORT_GFC_HBR    NULL
#define PORT_SFC_HBR    &hal_spi_port_transfer_command
```

Poslední skupinou, kterou lze pojmout jako port, jsou kanály AD převodníku. Makro pro položku *config* zde zastupuje pole, jehož první položka je ukazatel na bázovou adresu registrů ADC modulu, druhá položka je číslo měřené skupiny. Další makro určuje počet měřených kanálů a ukazatel na get funkci směřuje na funkci low-level driveru AD převodníku. Set funkce zde nemá smysl a proto je NULL.

```
#define PORT_CFG_ADC    { (uint32_t) adcREG1, adcGROUP1 }
#define PORT_NV_ADC     12
#define PORT_GFC_ADC    &adc_get_port_val
#define PORT_SFC_ADC    NULL
```

Další struktura *port_def_st* deklarovaná v tomto hlavičkovém souboru slouží pro propojení jména portu s jeho deskriptorem. Obsahuje opět dvě položky:

name ukazatel na řetězec se jménem portu, podle kterého se později vyhledává,

desc ukazatel na deskriptor portu.

V souboru *hal_port_def.c* jsou deklarované a naplněné struktury deskriptorů pro všechny porty a také pole struktur *port_def_st*, které mapuje deskriptory ke jménům portu, podobně jako v případě mapování pinů. I zde je velmi důležitá funkce *hal_port_get_dsc* pro vyhledání deskriptoru portu podle zadaného jména, pracující podobně jako v případě vyhledávání pinů.

7.3 Dekódování odpovědi z SPI

Všechny důležité struktury pro dekodér jsou v souboru *spi_resp_transl.h*. Hierarchie map navržená v sekci 6.8, je tvořena třemi strukturami. První z nich, *spitr_name_map_st*, mapuje název portu na mapu příkazů a odpovídajících odpovědí. Obsahuje tři položky:

spi_name ukazatel na řetězec s názvem portu,

cmd_map ukazatel na mapu příkazů a odpovědí,

num_cmd počet položek v mapě příkazů a odpovědí.

Mapa příkazů a odpovědí je tvořena druhou strukturou *spitr_cmd_map_st*:

cmd_msk maska pro platné bity příkazu, které rozlišují druhy příkazů mezi sebou,

command příkaz pro který se vztahuje odkazovaná odpověď,

field_desc ukazatel na odpověď, respektive mapu popisků a k nim přiřazených bitů odpovědi,

num_fields počet polí v odpovědi.

Třetí struktura *spitr_field_desc_st* je určena pro sestavení mapy názvů polí v SPI odpovědi a k nim přiřazených bitů. Struktura obsahuje jen dvě položky:

field_name ukazatel na řetězec popisující dané pole v odpovědi,

mask maska přiřazující biti odpovědi k řetězci.

Další dvě struktury v hlavičkovém souboru jsou určeny pro sestavení dekódovaných dat do tabulky, která umožňuje položky indexovat a je vhodnější k prohledávání. Tabulka je dána polem struktur *spitr_reg_translate_table_row_st*, definujících její řádky. Řádek obsahuje vždy jméno pole a jeho hodnotu.

Pro dekódování jsou připravené funkce. Nejprve je potřeba nalézt podle názvu portu správnou mapu příkazů a odpovědí. K tomu slouží funkce

```
const spitr_cmd_map_t* get_spi_cmd_map(const char* spi_port_name, int len,
    uint32_t* num_cmdDesc);
```

Funkce očekává jméno portu *spi_port_name* a vrací mapu příkazů a jejich počet *num_cmdDesc*.

Poté je potřeba najít mapu popisů polí a k nim přiřazených bitů, k čemuž lze použít funkci

```
const spitr_field_desc_t* get_spi_field_desc(const spitr_cmd_map_t* cmd_map
    , uint32_t num_cmd, uint32_t cmd, uint32_t* num_fdDesc);
```

která očekává mapu příkazů *cmd_map*, příkaz *cmd* ke kterému se hledá popis odpovědi a počet příkazů v mapě *num_cmd*. Návratovými hodnotami je mapa popisů polí a jejich bitů a počet polí *num_fdDesc*.

Nyní je již možné zavolat funkci pro samotné dekódování

```
int spitr_fill_tr_table(const spitr_field_desc_t* fd, uint32_t num_fields,
    uint32_t value, spitr_reg_translate_table_t* table);
```

Funkce očekává mapu názvů polí a jejich bitů *fd*, počet polí *num_fields*, hodnotu odpovědi k překladu *value* a vrací vyplněnou tabulku *table* a počet řádků.

Hodnoty z odpovědi jsou přímo čteny pomocí masky a makra, který obsahuje trik logické aritmetiky:

```
#define __mfld2val(mask,val) (((val)&(mask))/((mask)&~((mask)<<1)))
```

7.4 FlexRay

Soubor *drv_fray.c* obsahuje skupinu funkcí pro poskytnutí rozhraní k základní konfiguraci a provozu jednotky jako FlexRay uzlu. Jsou zde definované konfigurační, kontrolní a pomocné funkce.

7.4.1 Konfigurační funkce

fray_init Slouží k nastavení globálních parametrů sítě. Z ukazatele na strukturu nesoucí nastavení je překopíruje do konfiguračních registrů FlexRay. Provede také reset příkazu POC na výchozí hodnotu.

fray_config_buffer Slouží k nastavení RX nebo TX bufferu. Očekává následující argumenty.

- **buf_num** Určuje číslo právě konfigurovaného bufferu.
- **mode** Určuje vlastnosti bufferu pomocí bitových příznaků, které jsou definované v hlavičkovém souboru s prefixem FRAY_BUF_.
- **cyc_filter** Udává nastavení pro filtr rámců podle čísla cyklu.
- **frame_id** Přiřazuje bufferu číslo rámce, ve kterém budou přenášená data.
- **payload** Udává maximální velikost dat v rámci v jednotkách 16 bitů.
- **data_pointer** Adresuje první slovo bufferu v message RAM.

Funkce naplní podle zadaných parametrů datové struktury, vypočítá CRC hlavičky, nastaví input buffer a přenese nastavení do hlavičkové části message RAM.

fray_buffer_set_data Slouží k nastavení dat k odeslání v příštím komunikačním cyklu. Jako argumenty očekává číslo konfigurovaného bufferu, ukazatel na začátek dat a velikost dat ve slovech. Provede se nastavení vstupního bufferu, nakopírování dat do datové sekce vstupního bufferu a odeslání do datové sekce message RAM.

7.4.2 Kontrolní funkce

fray_clear_msg_ram Provede vynulování celé message RAM, což je vhodné provést vždy před počátkem konfigurace bufferů.

fray_startup_procedure Spustí proces integrace do sítě nebo inicializace nové sítě. Argument určuje, kterou cestou se bude uzel vydávat. V případě, že se jedná o *coldstart* uzel, bude se pokoušet založit novou síť spolu s dalšími coldstart uzly. V případě neúspěchu a vyčerpání startovacích pokusů přejde do integračního režimu.

fray_wait_for_new_cycle Pokud není povoleno přerušení, je možné tuto funkci použít pro detekci začátku nového komunikačního cyklu.

fray_message_received Slouží pro ověření, zda byla přijata nová zpráva do RX bufferu, jehož číslo je předané jako parametr funkci.

fray_buffer_get_data Slouží pro načtení přijatých dat z RX bufferu. Parametry funkce udávají číslo bufferu, ukazatel na místo v paměti, kam budou data zkopírována, a velikost dat ve slovech. Funkce nastaví výstupní buffer pro přenos dat, přenese data z RX bufferu do datové části výstupního bufferu a poté je zkopíruje na zadané místo v paměti.

fray_halt Funkce pro zastavení datových přenosů na konci testování.

7.4.3 Pomocné funkce

Další skupina obsahuje tzv. pomocné funkce, sloužící pro výpočet CRC, přechod mezi jednotlivými stavy POC a přenosy dat mezi message RAM, vstupním a výstupním bufferem. Tyto funkce jsou používané interně v předcházejících konfiguračních a kontrolních skupinách a kvůli k jejich počtu zde nejsou rozepsané.

7.5 Příklad implementace high-level driveru

Pro příklad implementace high-level driveru byl zvolen ovladač pro H-můstek. Obsahuje funkce pro ovládání PWM, směru otáčení motoru, zapínání a vypínání motoru a v neposlední řadě pro resetování watchdogu. Tyto metody tvoří ucelené rozhraní a následující text obsahuje popis každé z nich. Funkce pro ovládání PWM využívají low-level driver pro NHET,

funkce pro vypínání a nastavení směru využívají ovládání pinů pomocí HAL a pro reset watchdogu se používá SPI low-level driver.

Všechny ostatní high-level drivery dodržují podobnou strukturu v závislosti na funkcích, které jejich periferie poskytuje.

hbr_pwm_set_signal Slouží pro nastavení vlastností PWM. Parametry funkce nastavují periodu v µs a délku aktivní části signálu v procentech periody. Pokud je aktivní generování PWM signálu, změna se projeví ihned.

hbr_pwm_start Slouží k zapnutí generování PWM signálu na pin HBR_PWM můstku. Tato funkce skončí úspěšně jen tehdy, když byla před jejím volání alespoň jednou provedena konfigurace signálu pomocí hbr_pwm_set_signal.

hbr_pwm_stop Slouží k zastavení generování PWM signálu.

hbr_pwm_get_duty Funkce vracející délku aktivní části signálu v procentech periody.

hbr_pwm_get_period Funkce vracející délku periody v µs.

hbr_set_dir Funkce pro nastavení směru otáčení po nebo proti směru hodinových ručiček. Změna se projeví ihned.

hbr_get_dir Funkce vracející nastavený směr otáčení.

hbr_set_en Funkce pro zapnutí a nebo vypnutí motoru.

hbr_get_en Funkce vracející stav motoru.

hbr_spi_wdg_transfer Odstartuje automatické posílání příkazu pro reset watchdogu s periodou 10ms. Při prvním volání je vytvořena nová úloha, která posílá příkaz a po odeslání je vždy volána funkce *hbr_spi_wdg_callback* pro vykonávání jakéhokoliv doplňujícího kódu.

8 Testování

Cílem celé práce bylo provést otestování všech periferií, u kterých to bylo možné. Testování má za cíl odhalit nedostatky hardwarového návrhu a případné výrobní vady dodaných kusů. Práce si neklade za cíl provést důkladné otestování za všech krajních podmínek, ale ověřit základní funkčnost jednotky. Obrázek 8.1 ilustruje propojení dvou jednotek do FlexRay sítě a prostor s vybavením pro testování.

8.1 Použité vybavení

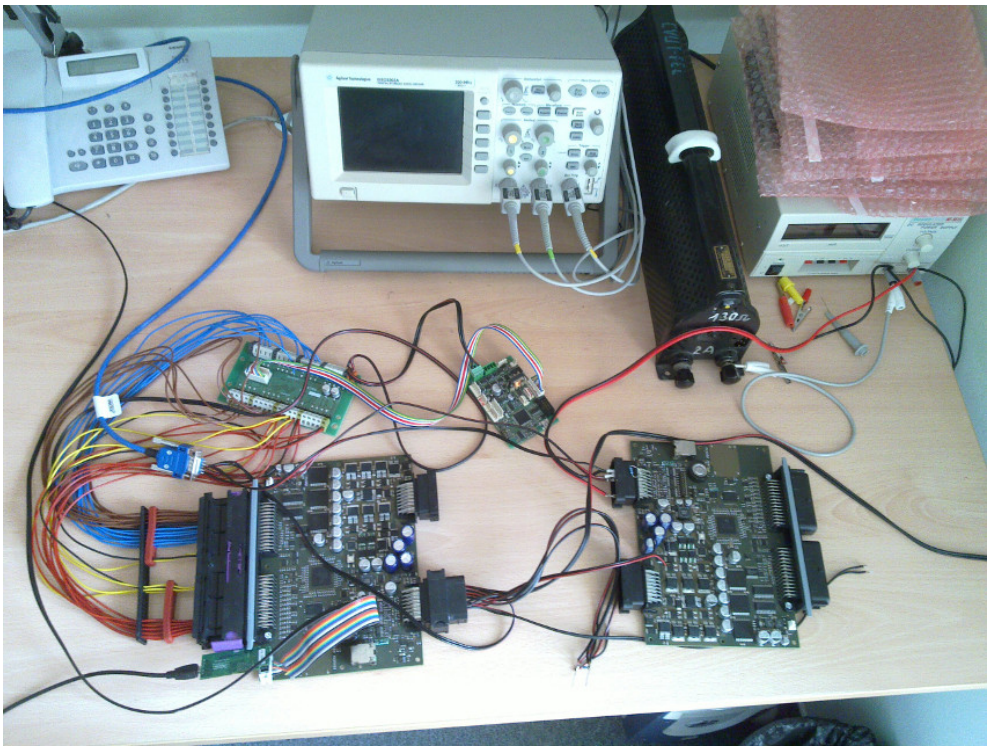
- Multimetr Metex M-3800
- Osciloskop Agilent Technologies DSO3202A
- FlexRay analyzátor Vector VN3600
- Rezistor 130 Ω 2 A
- Motor 12VDC
- IO Expander řízený po SPI
- Kontrolér LMC_CB1 pro převod příkazů z UART na SPI pro IO expander
- Zdroj napětí Manson NP-9615s

8.2 Testování a výsledky

8.2.1 Externí paměť SDRAM

Pro testování externí paměti je připraven příkaz *testsdram* (C.37), který provede zápis dat přes celý adresový prostor pro externí paměť a následně při čtení kontroluje jeho správnost.

Testem bylo ověřeno, že jednotky jsou osazené 64 MB paměťovými moduly, které vykazovaly bezchybnou funkci.



Obrázek 8.1: Testovací plocha a dvě jednotky

8.2.2 6x výstup do 10 A s možností PWM (HOUT)

Výstupy HOUT je potřeba testovat s dostatečnou zátěží, nejlépe indukčního charakteru. Jako dostatečný spotřebič pro zamezení přepnutí do chybového stavu se ukázal školní laboratorní rezistor 130 Ω 2 A. Piny byly testovány jeden po druhém voláním následujících příkazů a měření výstupního napětí multimetrem:

1. *pinvalHOUT1:1* (C.34) – Nastaví výstupní napětí HOUT1 na 12 V.
2. *houtfail 1* (C.25) – Otestuje, zda se HOUT1 nachází ve funkčním nebo chybovém stavu.
3. *pinvalHOUT1:0* (C.34) – Vypne HOUT1.
4. *houtpwm1:(1000,25)* (C.27) – nastaví PWM HOUT1 na 25 %.
5. *houtpwmstart1* (C.28) – spustí generování PWM signálu na HOUT1, na výstupu by mělo být napětí přibližně 3 V.
6. *houtpwmstop1* (C.29) – Vypne PWM na HOUT1

Nyní dojde k odpojení zátěže a otestování detekce selhání pomocí příkazů:

1. *pinvalHOUT1:1* (C.34) – Způsobí přepnutí budiče do chybového stavu z důvodu nepřipojení zátěže.
2. *houtfail 1* (C.25) – Musí detekovat selhání.

Pro otestování čtení hodnoty protékajícího proudu pomocí AD převodníku je možné použít příkaz *portvalHOUTIFBK?* (C.26).

Popsaným postupem testování prošly všechny HOUT piny a nebyl zjištěn žádný hardwarový ani softwarový problém.

8.2.3 6x výstupy do 2 A (MOUT)

Testování MOUT pinů probíhalo připojením každého z nich na multimetr pro měření napětí a aktivováním výstupů zadáním příkazů pro ovládání pinů (C.34). Například pro pin MOUT1 se zadávají příkazy *pinvalMOUT1EN:1* a *pinvalMOUT1IN:1*, kterými dojde k aktivaci napětí 12 V na tomto výstupu. Poté je možné výstup opět deaktivovat příkazem *pinvalMOUT1EN:0*, přepojit multimetr na další pin a stejným postupem ho otestovat.

Posupnou aktivací, proměřením a deaktivací byly otestovány piny MOUT1–MOUT6. U všech bylo změřeno správné napětí 12 V v aktivním stavu a 0 V v neaktivním stavu.

8.2.4 8x digitální výstupy do 100 mA (LOUT)

Digitální výstupy se testovaly připojené na IO expander postupným rozsvěcením a zhasínáním jednotlivých LED pomocí příkazu loutvalX:1 a loutvalX:0 (C.31), kde X nabývalo postupně hodnot 1 až 8. Stav budiče je možné zjistit příkazem loutstat (C.32), který vytiskne tabulku hodnot s popisy.

Postupným zapnutím a vypnutím každého výstupu byla ověřena bezchybná funkce portu.

8.2.5 16x digitální vstup (DIN)

Testování digitálních vstupů nejprve probíhalo ručním přivedením napětí na vstupy a čtením hodnot pomocí příkazů *pinvalDINX?* (C.34), kde X nabývalo hodnot 8–15 a čtením hodnot z SPI pomocí příkazu *dinportval?* (C.13). Měření bylo potřeba opakovat pro nastavení první skupiny pinů DIN1–DIN7 na detekci sepnutí vůči zemi a vůči 12 V. Později se testování zjednodušilo připojením vstupů na IO expander, jeho propojení s deskou LMC_CB1 a nastavováním vstupů posíláním příkazů z počítače po UART.

Výsledky testů ověřily správnou funkčnost hardware i software. Všechny piny detekují správně sepnutí při napětí 6 V.

8.2.6 2x LIN

Pro otestování jedné linky LIN byl navržen příkaz *testlin* (C.33), který vyšle zprávu po interní smyčce a ověří, zda byla přijata. Pomocí osciloskopu bylo ověřeno, že fyzická vrstva je aktivní – byly zachyceny signály.

Druhou linku LIN nebylo možné otestovat, protože sdílí jeden pin se sériovou linkou používanou ke komunikaci s počítačem.

8.2.7 4x digitálně-analogový převodník

Převodník byl zpočátku testován přímým měřením napětí na výstupech multimetrem, později byly výstupy propojeny se vstupy AD převodníku a porovnávány naměřené hodnoty s napětími nastavenými pomocí příkazů *dacpinvalX:Y* (C.6), kde X udává číslo pinu v rozmezí 1–4 a Y udává požadované výstupní napětí v mV.

Testování probíhalo pro napětí 0 V, 500 mV, 5 V a 12 V. Naměřené hodnoty na všech výstupech odpovídaly požadovaným napětím.

8.2.8 12x analogově-digitální převodník

Kanály AD převodníku byly testovány nejprve přivedením napětí 12 V ze zdroje přímo na vstupy ADL a ADH jednotlivých kanálů a přečtením hodnot z portu. V pozdějších fázích se provádělo testování propojením vstupů AD převodníku s výstupy DA převodníku, nastavením výstupního napětí a přečtením hodnot na ADC pomocí příkazu *readadc* (C.3).

Testem byla ověřena bezchybná funkce všech připojených kanálů pro napětí 0 V, 500 mV, 5 V i 12 V.

8.2.9 3x CAN High speed

Testování CAN linek probíhalo jejich propojením v jednu síť, přičemž jeden vodič linky byl záměrně odpojitelný pro vyvolání chyby. Pro odeslání a přijmutí testovacích zpráv slouží příkaz *testcan X Y* (C.4), kde X je číslo zdrojové linky a Y je číslo cílové linky.

Testování probíhalo nejprve posláním zpráv mezi všemi kombinacemi připojených linek. Poté se jeden drát linky 3 odpojil od sítě a vyzkoušela se reakce při posílání zpráv na tuto linku a z této linky.

Při prvním testování se vyskytly potíže s budiči fyzické vrstvy, které nepropouštěly žádný signál na sběrnici. Od výstupů MCU až ke vstupům budičů se signály šířily, ale i přes správné nastavení kontrolních signálů a korektní průchod stavovým automatem obvodů se nedařilo nic odvysílat.

Pomocí osciloskopu bylo zjištěno, že budiče se již během resetu MCU, kdy nejsou definovány hodnoty na výstupech, probudí z úsporného režimu a po uplynutí timeoutu na příjem nějakého signálu se opět uspí. Podle specifikace lze budič z tohoto stavu vyvést pouze tím, že detekuje příchozí komunikaci na sběrnici nebo signálem WAKE s 12V úrovněmi. Původní domněnka, že lze obvod aktivovat i signály pro STB a INH určenými pro řízení ze strany MCU, byla chybná. Proto bylo potřeba problém vyřešit drobnou úpravou, která během resetu podrží budič CAN v neaktivním stavu a ten bude moci být korektně probuzen teprve až po inicializaci MCU a spuštění testů.

Po vyřešení hardwarových problémů bylo možné zprávy odesílat i přijímat bez chyb. Software CAN správně detekuje selhání sítě a vypisuje příslušná hlášení.

8.2.10 Ethernet

Otestování Ethernetu nebylo součástí diplomové práce, ale pro úplnost je zde přesto uvedeno. Proběhlo otestování fyzické vrstvy odesláním testova-

cích paketů příkazem ethernet (C.15) a jejich zachycením na počítači.

Byla odhalena chyba v návrhu hardware, kdy ve schématu chyběla část obvodu. Nedostatek bylo možné opravit přidáním propojek a SMD součástek na patřičná místa a rozhraní Ethernet tím zprovoznit.

8.2.11 1x H-můstek

Pro testování můstku byl použitý malý stejnosměrný motor připojený přímo k výstupům. Sekvence pro spuštění motoru se skládá z těchto příkazů:

1. *hbrwdg* (C.19) – pro zapnutí periodického resetování watchdogu;
2. *hbrdir:1* (C.24) – pro volbu prvního směru;
3. *hbrpwm:(1000,100)* (C.20) – pro nastavení PWM na 100 %;
4. *hbrpwmstart* (C.21) – pro zapnutí generování signálu PWM;
5. *hbren:1* (C.23) – pro zapnutí motoru.

Nyní by se motor měl otáčet plnou rychlostí v jednom směru (záleží na připojení motoru k jednotlivým pólům výstupu). Pro změnu směru a rychlosti otáčení je možné zadat příkazy:

1. *hbrdir:0* (C.24) – změní směr otáčení;
2. *hbrpwm:(1000,50)* (C.20) – nastaví rychlost otáčení na 50 %.

Motor se vypne zadáním příkazu *hbren:0* (C.23). Testování prokázalo bezchybnou funkčnost celého podsystému.

8.2.12 1x FlexRay pro dva kanály

Pro otestování sběrnice FlexRay je potřeba propojit dvě JRVA jednotky do jedné sítě a k dokonalému přehledu dějů na síti byl během testování připojen také FlexRay analyzátor Vector VN3600. Na krátkou vzdálenost postačí i obyčejné propojovací kabely bez zakončovacích odporů, ale na delší vzdálenosti je nezbytné použít kroucenou dvojlinku zakončenou rezistorem s charakteristickou impedancí vedení.

Po zapojení všech uzlů je možné spustit testování tím, že na první jednotce se spustí příkaz *testfrayA* (C.17), povolí se případně záznam analyzátoru a na druhé jednotce se spustí příkaz *testfrayB* (C.18). Jednotky nastartují novou síť a vymění si 100 zpráv. Zaznamenává se počet chyb vzniklých během přenosu a po ukončení přenosu jsou oba uzly zastaveny.

Podle záznamu analyzátoru i výpisů obou testovacích příkazů bylo ověřeno, že rozhraní FlexRay je funkční.

8.2.13 Ovládání napájecích okruhů

Během prvních experimentů se pro zapínání napájecích okruhů používaly příkazy pro ovládání pinů *pinvalVBAT1EN:1*, *pinvalVBAT2EN:1*, *pinvalVBAT3EN:1* a *pinvalVBATEN:1* (C.34). Bohužel spínání napájecích okruhů se ukázalo jako nespolehlivé. Obvody AUIR33401S jsou určeny pro spínání induktivní zátěže a v tomto případě je jimi spínaná zátěž kapacitní. Na začátku, kdy jsou kapacitory tvořící filtr zcela vybité, protéká obvodem krátkou dobu relativně velký proud, který ochrana integrovaného obvodu interpretuje jako zkrat a přepne se do chybového stavu, ze kterého je možné se dostat jedině resetem systému. Dosud nebyl nalezen žádný hardwarový způsob řešení tohoto problému a proto byl částečně vyřešen softwarově.

Pro zapnutí všech napájecích okruhů slouží nyní příkaz *poweron* (C.40), který začne spínat hlavní VBATEN pin pomocí signálu PWM s proměnnou šířkou pulzů. Z počátku jsou pulzy velmi krátké a vzdálené od sebe, postupně se prodlužují a prodleva mezi nimi se zkracuje. Tímto způsobem se postupně nabije kapacitor na takovou úroveň, kdy už při trvalém sepnutí protéká obvodem bezpečný proud a napájecí okruh je v provozu.

Toto řešení však pomohlo jen částečně, protože na některých jednotkách je nutné parametry pulzů upravit tak, aby odpovídaly mírně odlišným vlastnostem kapacitorů.

Výsledkem testu tedy je odhalení nedostatku v návrhu hardware, který je potřeba ještě vyřešit, protože softwarové spínání napájecích okruhů není vždy spolehlivé.

8.2.14 Přechod do úsporného režimu

Tímto testem se ověří korektnost chování budičů fyzických vrstev CAN a LIN, které nesmějí po přechodu do úsporného režimu jednotku samovolně probudit. Systém se uspí příkazem *sleep* (C.2). Během testování se projevil nedostatek při návrhu obvodu u sběrnice LIN, která jednotku vždy okamžitě probudila.

Po provedení opravy již příkaz *sleep* desku přepne do režimu s nízkým odběrem.

8.2.15 Probuzení z úsporného režimu

Jednotka se má probudit při detekci vnějších signálů na sběrnici LIN a CAN. Po přepnutí do úsporného režimu příkazem *sleep* bylo probouzení otestováno přivedením vodiče LIN RX na zem a připojením jedné z CAN linky

ke druhé jednotce, na které byl spuštěný příkaz *testcan*. V obou případech došlo k probuzení jednotky do operačního režimu.

8.3 Automatizace testů

8.3.1 Testovací skript

Z výroby bylo doručeno několik osazených jednotek určených k otestování. Další jednotky budou následovat po osazení konektorů. Pro vyloučení pochybení při ověřování funkčnosti, například opomenutím nějakého testu, byl napsán skript v jazyce Python, který provádí testera krok po kroku testovacím procesem podle plánu, napsaném v konfiguračním souboru. Skript vypisuje pokyny pro testera, odesílá příkazy jednotce a vypisuje očekávaný a skutečný výstup příkazu.

8.3.2 Konfigurace skriptu

Syntaxe konfiguračního souboru je jednoduchá a obsahuje jen několik příkazů a možnost řádkových komentářů ve stylu jazyka C, tedy pomocí dvojice znaků `//`. Následující příkazy slouží k definování testovacího procesu.

showmsg Zobrazí testerovi zprávu uvedenou na dalších řádcích. Blok zpráv je ukončen příkazem *end*, po kterém skript čeká na stisk klávesy Enter.

cmd Slouží pro vyznačení bloku příkazů. Textové řetězce uvnitř bloku jsou po řádcích odeslány na sériovou linku jako příkazy a je přečten jejich výstup. Po každém příkazu musí následovat blok *output*. Skript pokaždé před odesláním nového příkazu čeká na potvrzení klávesou Enter. Blok příkazů je ukončen příkazem *end*.

output Předpokládaný výstup předchozího příkazu je definován tímto blokem, ukončeným příkazem *end*.

end Slouží k ukončení bloku.

8.3.3 Spuštění skriptu a ovládání

Pro spuštění skriptu je potřeba mít nainstalovaný interpretr jazyka Python ve verzi nejlépe 2.6 s knihovnou PySerial .

Spuštění pod operačním systémem Linux je možné pomocí příkazu

```
python testscript.py /dev/ttyXXX path_to_config
```

V prostředí příkazové řádky operačního systému Windows je možné použít podobný příkaz, pouze cesta k sériovému portu je nahrazena aliasem COM s příslušným číslem. *Path_to_config* udává cestu ke konfiguračnímu souboru, podle kterého probíhá testování.

Ovládání skriptu je přímočaré. Po každé zprávě pro testera se před dalším krokem čeká na stisk klávesy Enter a stejně tak před odesláním každého příkazu je očekáváno potvrzení klávesou Enter.

8.4 Shrnutí výsledků testů

Periferie	Provedené testy	Výsledek testu
MOUT	Aktivace výstupu	OK
Externí paměť SDRAM	Zápis a čtení vzoru přes celý adresní prostor	OK
HOUT	Aktivace výstupu, detekce chyby, PWM	OK
DIN	Čtení hodnot na vstupu, konfigurace programovatelných pinů	OK
LIN1	Poslání a přijmutí vzorové zprávy pomocí interní smyčky, změření signálů na fyzickém konektoru	OK
DAC	Nastavení požadovaného napětí a výstupu, aktivace výstupu	OK
ADC	Přečtení hodnot na všech kanálech převodníku	OK
CAN	Odeslání a přijmutí vzorové zprávy, detekce a zotavení z chyb na síti	Zjištěna a opravena hardwarová závada, nyní OK
Ethernet	Odeslání testovacího paketu a přijmutí na PC.	Zjištěna a opravena hardwarová závada, nyní OK
HBR	Resetování watchdogu, volba směru, aktivace výstupu, PWM	OK

Periferie	Provedené testy	Výsledek testu
MOUT	Aktivace výstupu	OK
FlexRay	Iniciace sítě, odeslání a přijmutí zprávy mezi dvěma propojenými jednotkami	OK
VBAT	Aktivace napájecího okruhu	Zjištěna závada, oprava pomocí software, nutno ladit parametry pro každou jednotku zvlášť.
Přechod do úsporného režimu		OK
Probuzení z úsporného režimu	Přivedením signálu na LIN nebo CAN	OK

9 Závěr

Diplomová práce měla za cíl vytvořit software pro obsluhu periferií a komunikačního rozhraní FlexRay automobilové řídící jednotky navržené katedrou řídící techniky. Hlavní účel software byl otestovat dodané jednotky na výrobní chyby a nedostatky v návrhu, sekundární účel byl vytvořit jednotné rozhraní pro ovládání vstupně výstupních portů využitelné v navazujících projektech v prostředí Matlab/Simulink.

Na počátku vývoje bylo nejprve zprovozněno vývojové prostředí Code Composer Studio od Texas Instruments a založen git repositář pro zdrojové kódy vyvíjeného software. V průběhu vývoje byla již existující wiki projektu doplněna o nové sekce obsahující seznam nízkoúrovňových SPI příkazů jednotlivých periferií, návod pro rozšíření sady příkazů pro interpretr CMDPROC a dokumentaci hotových příkazů. Byla provedena kontrola schématu jednotky a sjednocení názvosloví mezi projektovou wiki a popisy ve schématu.

Během příprav a vývoje před dodáním prototypů z výroby byla aktivována komunikace přes sériovou linku a došlo k integraci interpretru příkazů CMDPROC na procesor TMS570LS3137 a operační systém FreeRTOS. Vznikly také první příkazy pro ovládání rozhraní CAN a analogově-digitálního převodníku. Vše bylo testováno na vývojovém kitu se stejným mikrokontrolérem.

Po dokončení výroby prototypů a dodání prvních kusů jednotek pokračoval vývoj již na cílovém hardware. Byla navržena a vytvořena abstrakční vrstva pro jednotné ovládání pinů a jejich slučování do portů nezávisle na tom, ke které periferii MCU (NHET, GPIO, DMM) nebo expandéru SPI jsou připojeny. Nad touto vrstvou byly implementovány vysokoúrovňové ovladače, které nabízejí sadu funkcí pro pohodlné ovládání periferií bez nutnosti znát detaily hardwarového návrhu. Sada příkazů pro interpretr byla rozšířena o příkazy ovládající vstupy, výstupy, převodníky a sběrnice CAN a LIN tak, aby bylo možné ověřit jejich správnou funkci.

Proběhla analýza kódu pro rozhraní FlexRay poskytnutého katedrou měření. Část kódu byla upravena pro potřeby projektu a začleněna do ovladače FlexRay. Vznikla sada příkazů k testování komunikace mezi dvěma propojenými jednotkami.

Všechny dodané jednotky s osazenými konektory byly podrobeny poloautomatickému testování pomocí vytvořeného skriptu. Během testování byly odhaleny nedostatky v návrhu hardware pro modul CAN, LIN a Ethernet. Opravy byly zaneseny do schémat a jednotky odeslány na opravu.

Jeden nedostatek dosud opravený nebyl. Jedná se o spínání napájecích okruhů, u kterého byl objevený problém se spínáním kapacitní zátěže (filtračních kapacitorů) pomocí spínačů určených pro indukční zátěž, které však jako jediné splňovaly bezpečnostní normy. Bylo navržené softwarové řešení, které kapacitory nabíjelo pomocí PWM s proměnnou délkou periody. Tento způsob problém řeší jen částečně, protože je potřeba pro každou jednotku nalézt správné parametry PWM, aby spínání bylo spolehlivé. Do budoucna je nutné provést hardwarovou úpravu návrhu s využitím jiného obvodu.

Nakonec byla vytvořena dokumentace k vytvořeným příkazům a k interpretru příkazů na projektové wiki. Čtenář ji může nalézt v dodatcích diplomové práce.

Navazující projekty využívající tuto řídící jednotku budou moci využívat vysokoúrovňové ovladače a abstrakční vrstvu, vytvořené v této práci, a je tak možné soustředit se rovnou na vyšší funkce software a jejich implementaci například v prostřední Matlab/Simulink. Další práce na nižších úrovních může spočívat v implementaci rozhraní pro konfiguraci a ovládání FlexRay, CAN a LIN. Je možné také navázat portováním TCP/IP zásobníku pro TMS570LS3137 a FreeRTOS a nahradit tak komunikaci po sériové lince spojením pomocí Ethernetu.

Literatura

[1] 100 greatest innovations. *Channel4* [online]. 2009 [cit. 2013-04-09]. Dostupné z: http://www.channel4.com/4car/100-greatest/innovations/enginetech.html

[2] BARRY, Richard. *USING THE FREERTOS REAL TIME KERNEL: a Practical Guide* [online]. 1.0.5. 2009 [cit. 2013-04-11].

[3] BOSCH. *CAN Specification* [online]. 2.0. 1991 [cit. 2013-04-15]. Dostupné z: http://www.bosch-semiconductors.de/media/pdf_1/canliteratur/can2spec.pdf

[4] *CAN bus*. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001-2013 [cit. 2013-04-15]. Dostupné z: http://en.wikipedia.org/wiki/CAN_bus

[5] En Cars Globe. *Volkswagen 411* [online]. 2012 [cit. 2013-04-15]. Dostupné z: http://encarsglobe.com/data_images/models/volkswagen-411/volkswagen-411-13.jpg

[6] *Engine control unit*. In: Wikipedia: the free encyclopedia [online]. 24.9.2005. San Francisco (CA): Wikimedia Foundation, 2001-2013, 2.4.2013 [cit. 2013-04-09]. Dostupné z: http://en.wikipedia.org/wiki/Engine_control_unit

[7] FLEXRAY CONSORCIUM. FlexRay *Communications System Protocol Specification: Version 2.1* [online]. Revision A. 2005 [cit. 2013-04-15]. 6.5

[8] *FlexRay*. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001-2013 [cit. 2013-04-15]. Dostupné z: http://en.wikipedia.org/wiki/FlexRay

[9] FREESCALE. *33972/A/T: MULTIPLE SWITCH DETECTION INTERFACE* [online]. Rev. 19.0. 2012 [cit. 2013-04-15]. Dostupné z: http://cache.freescale.com/files/analog/doc/data_sheet/MC33972.pdf?pspll=1 5.8, D.2

[10] INTERNATIONAL RECTIFIER. *AUIR33401S* [online]. 2007 [cit. 2013-04-15]. Dostupné z: http://www.irf.com/product-info/datasheets/data/auir33401s.pdf 5.11, D.4

[11] Local Interconnect Network. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001-2013 [cit. 2013-04-15]. Dostupné z: http://en.wikipedia.org/wiki/Local_Interconnect_Network

[12] MICROCHIP. *MCP4902/4912/4922: Datasheet* [online]. 2010 [cit. 2013-04-15]. Dostupné z: http://ww1.microchip.com/downloads/en/DeviceDoc/22250A.pdf 5.7, D.3

[13] NXP. *TJA1021 LIN 2.1/SAE J2602 transceiver: Product data sheet* [online]. Rev. 7. 2011 [cit. 2013-04-15]. Dostupné z: http://www.nxp.com/documents/data_sheet/TJA1021.pdf 5.14, D.10

[14] NXP. *TJA1041A High-speed CAN transceiver: Product data sheet* [online]. Rev. 04. 2008 [cit. 2013-04-15]. 5.5, D.8

[15] NXP. *TJA1082 FlexRay node transceiver: Product data sheet* [online]. Rev. 6. 2012 [cit. 2013-04-15]. Dostupné z: http://www.nxp.com/documents/data_sheet/TJA1082.pdf 5.9, D.9

[16] ON SEMICONDUCTOR. *NCV7608: Octal Configurable Low/High Side Driver* [online]. 2010 [cit. 2013-04-25]. Rev. 2. Dostupné z: http://www.onsemi.com/pub_link/Collateral/NCV7608-D.PDF 5.13, D.6

[17] STM. *L99H01 Motor bridge driver for automotive applications: Datasheet* [online]. Rev. 4. 2012 [cit. 2013-04-15]. Dostupné z: http://www.st.com/st-web-ui/static/active/en/resource/technical/document/datasheet/CD00230289.pdf 5.10

[18] STM. *VNH5019A-E: Automotive fully integrated H-bridge motor driver* [online]. Rev 8. 2011 [cit. 2013-04-15]. Dostupné z: http://www.st.com/st-web-ui/static/active/en/resource/technical/document/datasheet/CD00234623.pdf 5.12, D.5, D.7

[19] TEXAS INSTRUMENTS. *Code Composer Studio* [online]. 1995, 2013 [cit. 2013-04-11]. Dostupné z: http://www.ti.com/tool/ccstudio#descriptionArea 5.1

[20] TEXAS INSTRUMENTS. *HAL Code Generator tool* [online]. 1995, 2013 [cit. 2013-04-11]. Dostupné z: http://www.ti.com/tool/halcogen 5.1

[21] TEXAS INSTRUMENTS. *TMS570LS31x/21x 16/32-Bit RISC Flash Microcontroller: Technical Reference Manual* [online]. 2012 [cit. 2013-04-11]. Dostupné z: http://www.ti.com/litv/pdf/spnu499a

[22] TEXAS INSTRUMENTS. *TMS570LS31x Hercules Development Kit* [online]. 2013 [cit. 2013-04-22]. Dostupné z: http://www.ti.com/tool/tmdx570ls31hdk 3.2

[23] ZEMAN, *Martin. Programové vybavení převodníku Ethernet/-FlexRay* [online]. ČVUT FEL katedra měření, 2012. Bakalářská práce. ČVUT. Vedoucí práce Jan Novák. Dostupné z archivu závěrečných prací katedry měření. 1.1.1, 5, 6.5

A Tvorba příkazu pro CMDPROC

Kořenový seznam příkazů je definován v souboru *cmd.c*, který obsahuje také některé specifické příkazy, které neodpovídají žádné z dalších kategorií. Příkazy s podobným zaměřením by však měly být sloučeny do podseznamů a odděleny do jiných zdrojových souborů tak, jak je naznačeno v příkladu níže.

A.1 Definice příkazu

Každý příkaz se skládá ze tří povinných částí. Funkce s rutinou vykonávanou příkazem, deskriptoru a reference na deskriptor v seznamu příkazů. Rutina příkazu musí být definována ve funkci s následující hlavičkou:

```
int cmd_do_something(cmd_io_t *cmd_io, const struct cmd_des *des,
    char *param[])
```

cmd_io Ukazatel na zásobník vstupně výstupních funkcí pro sériový port;

des ukazatel na deskriptor příkazu, jemuž patří tato funkce;

param ukazatel na pole parametrů předaných při volání příkazu.

Deskriptor příkazu je definován strukturou *cmd_des_t*, obsahující šest položek:

code udává zda příkaz používá operační znaky : a ?;

mode specifikuje mód příkazu – pro čtení, pro zápis, pro čtení i zápis;

name ukazatel na řetězec se jménem příkazu;

help ukazatel na řetězec s nápovědou;

fnc ukazatel na funkci s rutinou příkazu;

info ukazatel na proměnnou, do které může příkaz zapisovat nebo číst z ní hodnoty (volitelné).

Po vytvoření funkce příkazu, definici a naplnění deskriptoru je potřeba přidat referenci na deskriptor do příslušného seznamu nebo podseznamu příkazů. Seznam příkazů musí být vždy zakončen prázdnou položkou (NULL).

A.2 Příklad přidání nového příkazu

Předpokládejme, že je potřeba přidat sadu příkazů pro ovládání imaginární LED, pro kterou již je definován driver v souboru *drv_led.c*.

Vytvoříme nový zdrojový soubor *cmd_led.c* a hlavičkový soubor *cmd_led.h*. Hlavičkový soubor obsahuje pouze externí deklaraci seznamu pro novou sadu příkazů:

```
#ifndef CMD_LED_H_
#define CMD_LED_H_

#include "cmd_proc"  // Include command processor header file
extern cmd_des_t const *cmd_list_led[];   // External definition of list,
    to be visible in root file commands.c

#endif /* CMD_LED_H_ */
```

Zdrojový soubor obsahuje definici funkce pro zapnutí a vypnutí LED, dva deskriptory příkazů ukazující na tyto funkce a nový seznam příkazů dle deklarace v hlavičkovém souboru:

```
#include "cmd_led.h"
#include "cmdio_tisci.h"     // IO stack for RS-232 output transfer
#include "drv_led.h"     // Our LED driver

// Turn LED on command routine
int cmd_do_led_on(cmd_io_t *cmd_io, const struct cmd_des *des, char *param[]) {
    led_setbit(1);
    print("LED␣is␣on/r/n");
    return 0;
}

// Turn LED off command routine
int cmd_do_led_off(cmd_io_t *cmd_io, const struct cmd_des *des, char *param[]) {
    led_setbit(0);
    print("LED␣is␣off/r/n");
    return 0;
}

// Command descriptor for LED on command
cmd_des_t const cmd_des_led_on = {
    0, 0,
    "ledon", "Turns␣the␣LED␣on",
    cmd_do_led_on, (void *)&cmd_list_led
};

// Command descriptor for LED off command
cmd_des_t const cmd_des_led_off = {
    0, 0,
    "ledoff","Turns␣the␣LED␣off",
    cmd_do_led_off, (void *)&cmd_list_led
};

// Sublist of command descriptors for LED control
// List can include another sublists and must be terminated by NULL
```

```
cmd_des_t const *cmd_list_led[] = {
    &cmd_des_led_on,
    &cmd_des_led_off,
    NULL
};
```

Tím je hotová definice nového příkazu a nyní je potřeba modifikovat soubor *cmd.c* s kořenovým seznam příkazů. Modifikace spočívá ve vložení nového hlavičkového souboru na začátek kódu a vložení podseznamu pomocí direktivy CMD_DES_INCLUDE_SUBLIST na požadované místo:

```
...
// Include our header file
#include "cmd_led.h"
...
// Include our sublist into the root list
/** @brief Main list of commands */
cmd_des_t const *cmd_list_main[] = {
    &cmd_des_help,
    &cmd_des_sleep,
    CMD_DES_INCLUDE_SUBLIST(cmd_list_adc),
    ...
    CMD_DES_INCLUDE_SUBLIST(cmd_list_led),
    ...
    NULL
};
```

Po přeložení a nahrání programu do jednotky bude výpis příkazu *help* (C.1) obsahovat nově vložené položky.

B Manuál pro navázání komunikace jednotky s počítačem

Pro fyzické propojení jednotky a počítače je nutné použít převodník napěťových úrovní RS-232 a TTL.

Je možné použít libovolný emulátor terminálu, který je schopný přistupovat k sériovému portu. V diplomové práci byl použit GTKTerm, , který vyžaduje nastavení následujících hodnot parametrů:

- baud rate: 9600;
- parity: None;
- bits: 8;
- stopbits: 1;
- flow control: None.

Pokud je terminál správně nastavený a jednotka připojena ke zdroji napětí, zobrazí se uvítací text a výzva. Od té chvíle je možné zadávat příkazy ukončené znakem End Of Line.

C Dokumentace k příkazům

C.1 Help

Popis

Tento příkaz bez parametrů vypíše seznam všech dostupných příkazů s krátkou nápovědou pro každý z nich. Pokud je předán název příkazu jako parametr, vypíše nápovědu jen pro něj.

Syntaxe

```
--> help [command]
```

Příklad

```
--> help pinval
```

Vypíše nápovědu pro příkaz *pinval*.

C.2 Sleep

Popis

Příkaz přepne LIN a CAN budiče do úsporného režimu, což zapříčiní uspání celého systému. Jen externí signály na LIN nebo CAN dokáží jednotku probudit.

Syntaxe

```
--> sleep
```

C.3 Readadc

Popis

Příkaz přečte hodnoty na všech kanálech 1–12 analogově-digitálního převodníku a vypíše hodnoty v hexadecimální formě do tabulky.

Syntaxe

```
--> readadc
```

C.4 Testcan

Popis

Příkaz otestuje sběrnici CAN zasíláním a přijímáním testovacích zpráv pomocí externí smyčky. Při selhání sítě jí umí příkaz resetovat do funkčního stavu.

Jsou očekávány dva parametry. První určuje číslo zdrojové linky, druhý číslo sílové linky.

Příkaz přenáší během testu 100 zpráv, počítá počet TX a RX chyb a případné vypršení časových limitů. Na konci testu je vypsán stav sítě a počet chyb.

Syntaxe

```
--> testcan SOURCE DESTINATION
    kde SOURCE se nerovná DESTINATION
    kde SOURCE a DESTINATION jsou v~rozsahu 1-3
```

Příklad

```
--> testcan 1 2
```

Pošle 100 krát zprávu z CAN1 na CAN2

C.5 Dacpinon

Popis

Příklad umožňuje číst a nastavovat stav pinů digitálně-analogového převodníku (DAC). Pin může být zapnut (ON) nebo vypnut (OFF). Při zapnutí se na výstupu objeví nastavené výstupní napětí.

První parametr udává číslo pinu (1–4), operační znak : pro zápis nebo ? pro čtení stavu. Pokud je použit operační znak :, očekává se druhý parametr, který udává nový stav 1 (ON) nebo 0 (OFF).

Příkaz vždy vypisuje aktuální stav pinu.

Syntaxe

```
--> dacpinonPIN[:|?][STATE]
    kde PIN je v~rozsahu 1-4
    kde : unamená nastav STATE na pin PIN
    kde ? Znamená čti stav pinu PIN
    kde STATE smí být 0 - OFF nebo 1 - ON
```

Příklad

```
--> dacpinon1:1
```

Zapne pin DAC1, vypíše aktuální stav pinu (měl by tedy být zapnutý)

```
--> dacpinon2:0
```

Vypni pin DAC2, vypiš aktuální stav pinu (měl by tedy být vypnutý)

```
--> dacpinon2?
```

Vypiš stav pinu DAC2.

C.6 Dacpinval

Popis

Příkaz poskytuje funkci pro čtení a nastavení napětí na DA pinu.

Očekává číslo pinu (1–4) jako první parametr, operační znak : pro zápis nebo ? pro čtení a eventuálně druhý parametr, určující napětí k nastavení v milivoltech.

Příkaz vždy vytiskne aktuální napětí na pinu.

Syntaxe

```
--> dacpinvalPIN[:|?][VALUE]
    kde PIN je v~rozsahu 1-4
    kde : znamená nastav napětí VALUE na pin PIN
    kde ? Znamená čti napětí z~pinu PIN
```

Příklad

```
--> dacpinval1:8000
```

Nastaví napětí 8 V na pin DAC1, vytiskne aktuální napětí (což by mělo být 8000).

```
--> dacpinval2:500
```

Nastaví napětí 0.5 V na pin DAC2, vytiskne aktuální napětí (což by mělo být 500).

```
--> dacpinval2?
```

Přečte a vytiskne napětí na pinu DAC2.

C.7 Dacstat

Popis

Vytiskne seznam všech DAC pinů, jejich stav (ON/OFF) a napětí v mV.

Syntaxe

```
--> dacstat
```

C.8 Dinportst

Popis

Příkaz slouží k nastavení pinu digitálního vstupu jako aktivního a nebo tří-stavového. Očekává dva parametry. První nastavuje neprogramovatelné piny, druhý programovatelné piny. Každý parametr je 8 bitová hodnota, jejíž nejméně významný bit je přiřazen prvnímu pinu, druhý druhému pinu atd. 1 nastaví pin jako tří-stavový, 0 jako aktivní.

Syntaxe

```
--> dinportst:(SGVAL,SPVAL)
    kde SGVAL je v~rozsahu 00 - FF
    kde SPVAL je v~rozsahu 00 - FF
```

Příklad

```
--> dinpotrst:(00,03)
```

Nastaví piny SP0 a SPI1 jako tří-stavové, piny SP2–SP7 a SG0–SG7 jako aktivní.

SPVAL = 0b00000011

SGVAL = 0b00000000

```
--> dinportst:(AA,FF)
```

Nastaví piny SP0–SP7, SG1, SG3, SG5, SG7 jako tří-stavové, SG0, SG2, SG4 a SG6 jako aktivní.

SPVAL = 0b11111111

SGVAL = 0b10101010

C.9 Dinportpr

Popis

Příkaz slouží k nastavení programovatelných DIN pinů jako spínaných k zemi a nebo ke 12 V. Příkaz očekává jeden 8 bitový argument. První bit zleva (LSB) je přiřazen prvnímu programovatelnému pinu, druhý druhému atd. 1 nastaví pin pro detekci sepnutí vůči 12 V, 0 vůči zemi.

Syntaxe

```
--> dinportpr:(SPVAL)
    kde SPVAL is in range 00 - FF
```

Příklad

```
--> dinpotrpr:(03)
```

Nastavit piny SP0 a SP1 pro detekci sepnutí vůči 12 V, SP2–SP7 vůči zemi.

SPVAL = 0b00000011

C.10 Dinportint

Popis

Příkaz slouží k povolení nebo zakázání generování přerušení a funkce wake-up na pinech DIN. Příkaz očekává dva parametry, každý z nich zadaný 8 bitovou hodnotou. První parametr konfiguruje neprogramovatelné piny, druhy programovatelné. První bit zleva (LSB) je přiřazen prvnímu pinu, druhý druhému atd. 1 povoluje přerušení a wake-up, 0 je deaktivuje.

Syntaxe

```
--> dinportint:(SGVAL,SPVAL)
    kde SGVAL je v~rozsahu 00 - FF
    kde SPVAL je v~rozsahu 00 - FF
```

Příklad

```
--> dinpotrint:(00,03)
```

Povolí přerušení a wake-up funkci na pinech SP0 a SP1, deaktivuje je na pinech SP2–SP7 a SG0–SG7.

SPVAL = 0b00000011

SGVAL = 0b00000000

```
--> dinportint:(AA,FF)
```

Povoluje přerušení a wake-up funkci na pinech SP0–SO7, SG1, SG3, SG5 , SG7 a deaktivuje je na pinech SG0, SG2, SG4 a SG6.

SPVAL = 0b11111111

SGVAL = 0b10101010

C.11 Dinreset

Popis

Příkaz resetuje obvod ovládající DIN do výchozího stavu.

Syntaxe

```
--> dinreset
```

C.12 Dinswst

Popis

Příkaz ruší příznak přerušení a přečte nový stav vstupů.

Syntaxe

```
--> dinswst
```

C.13 Dinportval

Popis

Příkaz přečte hodnoty všech DIN pinů a vytiskne je jako hexadecimální hodnotu. První bit zprava (LSB) udává stav prvního pinu, druhý druhého atd.

Syntaxe

```
--> dinportval?
```

C.14 Dinstat

Popis

Příkaz přečte odpověď ovládajícího obvodu po SPI, dekóduje jí a vytiskne ve formě čitelné tabulky.

Syntaxe

```
--> dinstat
```

C.15 Ethernet

Popis

Příkaz odešle několik pokusných paketů. Žádné platné síťové spojení není realizováno, slouží jen pro otestování fyzické vrstvy.

Syntaxe

```
--> ethernet
```

C.16 Fraystat

Popis

Příkaz přijme odpověď z budiče fyzické vrstvy po SPI, dekóduje jí a vytiskne ve formě tabulky. Očekává příponu specifikující číslo rozhraní FRAY.

Syntaxe

```
--> fraystatPORT
    kde PORT je v~rozsahu 1-2
```

Příklad

```
--> fray1
```

Vypíše status rozhraní FlexRay1

C.17 TestfrayA

Popis

Příkaz vytvoří z jednotky uzel A a čeká na vytvoření uzlu B na jiné jednotce, fyzicky propojené s uzlem A. Potom společně iniciují novou FlexRay síť a vymění si 100 zpráv. Během přijmu se počítají chyby vzniklé při přenosu a po skončení komunikace je uzel zastaven příkazem HALT.

Syntaxe

```
--> testfrayA
```

C.18 TestfrayB

Popis

Příkaz vytvoří z jednotky uzel B a čeká na vytvoření uzlu A na jiné jednotce, fyzicky propojené s uzlem B. Potom společně iniciují novou FlexRay síť a vymění si 100 zpráv. Během přijmu se počítají chyby vzniklé při přenosu a po skončení komunikace je uzel zastaven příkazem HALT.

Syntaxe

```
--> testfrayB
```

C.19 Hbrwdg

Popis

Příkaz nastartuje opakované odesílání příkazu reset watchdog s periodou 10ms. Přenos probíhá na pozadí. Tento příkaz musí být zavolán před nebo okamžitě po povolení H-můstku příkazem *hbren*.

Syntaxe

```
--> hbrwdg
```

C.20 Hbrpwm

Popis

Příkaz slouží k nastavení parametrů PWM pro H-můstek. Příkaz očekává operační znak : pro nastavení hodnot nebo ? pro čtení předchozí konfigurace. V případě znaku : následují dva další parametry. První udává délku periody v µs, druhý velikost aktivní části cyklu v procentech periody.

Syntaxe

```
--> hbrpwm[:|?][(PER,DUTY)]
    kde : znamená nastav parametry a argument v~závorce je povinný
    kde ? znemaná čti předchozí nastavení
    kde PER je větší než 0
    kde DUTY je v~rozsahu 0-100
```

Příklad

```
--> hbrpwm:(1000,25)
```

PWM na H-můstku bude mít periodu 1 ms a bude aktivní po 250 µs v každém cyklu.

```
--> hbrpwm?
```

Vytiskne aktuální periodu v µs a délku aktivního cyklu v procentech periody.

C.21 Hbrpwmstart

Popis

Pokud byla PWM pro H-můstek nastavena dříve příkazem *hbrpwm,* spustí tento příkaz generování signálu.

Syntaxe

```
--> hbrpwmstart
```

C.22 Hbrpwmstop

Popis

Příkaz zastaví generování PWM signálu.

Syntaxe

```
--> hbrpwmstop
```

C.23 Hbren

Popis

Příkaz nastavuje nebo čte hodnotu povolovacího pinu H-můstku. Očekává operační znak : pro nastavení nebo ? pro čtení. V případě nastavení je očekáván další parametr udávající hodnotu.

Syntaxe

```
--> hbren[:|?]val
    kde : znamená nastavit
    kde ? znamená číst
    kde val je 0 nebo 1
```

C.24 Hbrdir

Popis

Příkaz slouží k nastavení nebo čtení hodnoty pinu určujícím směr otáčení motoru. Očekává operační znak : pro nastavení nebo ? pro čtení. V případě čtení je očekáván další parametr udávající směr.

Syntaxe

```
--> hbrdir[:|?]val
    kde : znamená nastavit
    kde ? znamená číst
    kde val je 0 nebo 1
```

C.25 Houtfail

Popis

Příkaz testuje stav budiče HOUT, který může být buď v operačním a nebo chybovém stavu.

Syntaxe

```
--> houtfail pin
    kde pin je v~rozsahu 1-6
```

C.26 Readhifbk

Popis

Příkaz čte analogové hodnoty zpětné vazby budiče pinů HOUT.

Syntaxe

```
--> readhifbk
```

C.27 Houtpwm

Popis

Příkaz slouží k nastavení parametrů PWM pro HOUT pin. Příkaz očekává první parametr udávající číslo pinu, operační znak : pro nastavení hodnot nebo ? pro čtení předchozí konfigurace. V případě znaku : následují dva další parametry. První udává délku periody v µs, druhý velikost aktivní části cyklu v procentech periody.

Syntaxe

```
--> houtpwmPIN[:|?][(PER,DUTY)]
    kde PIN je v~rozsahu 1-6
    kde : znamená nastav parametry a argument v~závorce je povinný
    kde ? znamená čti předchozí nastavení
    kde PER je větší než 0
    kde DUTY je v~rozsahu 0-100
```

Příklad

```
--> houtpwm1:(1000,25)
```

HOUT1 PWM bude mít periodu 1 ms a bude aktivní 250 µs.

```
--> houtpwm1?
```

Aktuální perioda PWM pinu HOUT1 v µs a délka aktivní části cyklu v procentech periody.

C.28 Houtpwmstart

Popis

Pokud byly nastaveny parametry PWM pomocí příkazu houtpwm, nastartuje tento příkaz generování PWM signálu na vybraném pinu.

Syntaxe

```
--> houtpwmstartPIN
    kde PIN je v~rozsahu 1-6
```

C.29 Houtpwmstop

Popis

Příkaz zastaví generování PWM signálu na vybraný pin.

Syntaxe

```
--> houtpwmstopPIN
    kde PIN je v~rozsahu 1-6
```

C.30 Loutwr

Popis

Příkaz slouží k nastavení nebo čtení hodnoty všech pinů portu LOUT. Očekává operační znak : pro nastavení nebo ? pro čtení hodnoty a případný další parametr ve formě hexadecimálního čísla udává hodnoty pinů na portu. První bit zprava (LSB) udává hodnotu prvního pinu, druhý bit druhého pinu atd.

Syntaxe

```
--> loutwr[:|?][(VALUE)]
    kde ? znamená čti hodnoty z~portu LOUT
    kde ? znamená čti hodnoty z~portu LOUT
```

Příklad

```
--> loutwr:(FA)
```

LOUT1 = 0
LOUT2 = 1
LOUT3 = 0
LOUT4 = 1
LOUT5–8 = 1

C.31 Loutval

Popis

Příkaz slouží k nastavení nebo čtení hodnoty všech vybraného pinu portu LOUT. Očekává parametr udávající číslo pinu, operační znak : pro nastavení nebo ? pro čtení hodnoty a případný další parametr udávající hodnotu pinu na portu.

Syntaxe

```
--> loutvalPIN[:|?][VAL]
    kde PIN je v~rozsahu 1-8
    kde : znamená nastav hodnotu VAL na PIN
    kde ? znamená čti hodnotu z~pinu PIN
    Kde VAL je 1 nebo 0
```

Příklad

```
--> loutval1:1
```

LOUT1 = 1

C.32 Loutstat

Popis

Příkaz dekóduje odpověď z SPI integrovaného obvodu řídícího LOUT do tabulky čitelné člověku.

Syntaxe

```
--> loutstat
```

C.33 Testlin

Popis

Příkaz slouží k otestování rozhraní LIN1 posíláním testovací zprávy a její přijímání přes interní smyčku, přičemž je možné zachytit fyzické signály na portu pomocí osciloskopu.

Syntaxe

```
--> testlin
```

C.34 Pinval

Popis

Příkaz slouží pro nastavení nebo čtení hodnot z vybraného pinu. Je očekávána přípona s názvem pinu, který má být čten nebo zapisován. Následuje operační znak : pro nastavení nebo ? pro čtení hodnoty. Poslední parametr určuje hodnotu, která má být na pin zapsána.

Seznam pinů je možné obdržet příkazem *pinlist*.

Syntaxe

```
--> pinvalNAME[:|?][VAL]
    kde NAME je řetězec
    kde : znamená přiřad' VAL pinu NAME
    kde ? znamená čti hodnotu z~pinu NAME
    kde VAL je 0 nebo 1
```

Příklad

```
--> pinvalHBREN:1
```

Nastaví pin HBR_EN na 1

```
--> pinvalHBREN?
```

Čte hodnotu z pinu HBR_EN

C.35 Pindir

Popis

Příkaz slouží pro nastavení nebo přečtení směru pinu. Je očekávána přípona se jménem pinu, operační znak : pro nastavení nebo ? pro čtení směru a případně parametr udávající směr, který má být nastaven. 1 znamená výstup, 0 znamená vstup.

Seznam pinů je možné obdržet příkazem *pinlist*.

Syntaxe

```
--> pinvdirNAME[:|?][DIR]
    kde NAME je řetězec
    kde : znamená přiřad' DIR pinu NAME
    kde ? znamená čti směr pinu NAME
    kde DIR je 0 nebo 1
```

Příklad

```
--> pindirHBREN:1
```

Nastaví pin HBR_EN jako výstupní.

```
--> pindirHBREN?
```

Přečte směr pinu HBR_EN.

C.36 Portval

Popis

Příkaz slouží k nastavení nebo čtení hodnoty všem pinům specifikovaného portu. Očekává příponu udávající jméno portu, operační znak : pro nastavení nebo ? pro čtení hodnoty a případný další parametr ve formě hexadecimálního čísla udává hodnoty pinů na portu. První bit zprava (LSB) udává hodnotu prvního pinu, druhý bit druhého pinu atd.

Syntaxe

```
--> portvalNAME:(VAL)
    kde NAME je řetězec
    kde : znamená přiřad' hodnotu VAL portu NAME
    kde ? znamená čti hodnotu portu  NAME
    kde VAL je v~rozsahu 00 - FF

--> portvalLOUT:(FA)
```

LOUT1 = 0
LOUT2 = 1
LOUT3 = 0
LOUT4 = 1
LOUT5–8 = 1

C.37 Testsdram

Popis

Příkaz slouží k otestování připojeného SDRAM modulu přes celý adresní prostor. Detekuje platnou SDRAM kapacitu.

Syntaxe

```
--> testsdram
```

C.38 Spitr

Popis

Příkaz dekóduje odpovědi z SPI periferií do tabulky s popisky čitelné člověkem. Každá SPI odpověď je ve formě bajtů, ve kterých je zakódovaná informace na jednotlivých bitech nebo skupinách bitů. Tento příklad informaci transformuje do člověku čitelné podoby.

Příkaz očekává příponu udávající jméno periferie následované operačním znakem ? a dalšími dvěma parametry. První je kód SPI příkazu a druhá je odpověď získaná zasláním předchozího kódu příkazu. Žádný přenos není vykonáván.

Syntaxe

```
--> spitrNAME?(CMD,RESP)
    kde NAME je řetězec
    kde CMD je v~rozsahu 0 - FFFFFFFF
    kde RESP je v~rozsahu 0 - FFFFFFFF
```

Příklad

```
--> spitrFRAY1(FFFF,2B10)
```

Přeloží odpověď 2B10 z budiče FlexRay linky 1 získanou příkazem FFFF odeslaným například pomocí *spimst* do čitelné tabulky.

C.39 Spimst

Popis

Toto je nízkoúrovňový příkaz pro odesílání příkazů na SPI rozhraní. Příkaz očekává jako první parametr číslo SPI rozhraní, operační znak :, adresu zařízení na lince jako druhý parametr a data k odeslání, oddělená po bajtech čárkou, v závorce.

Příkaz odešle data a vypíše získanou odpověď.

Syntaxe

```
--> spimstSPI:ADDR([DATA])
    kde SPI je v~rozsahu 0 - 3
    kde ADDR je v~rozsahu 0 - 3
    kde DATA je ve formátu XX,XX,XX...
    kde XX je v~rozsahu 00 - FF
```

Příklad

```
--> spimst1:0(7F,00,00)
```

Odešle příkaz reset na DIN periferii.

C.40 Poweron

Popis

Příkaz slouží k zapnutí napájecích okruhů na jednotce. Pomalu nabije filtrační kapacitory pomocí PWM s proměnným časováním.

Poweron musí být použit před jakýmkoliv přístupem na periferie závislé na tomto okruhu, jinak jsou zcela nefunkční.

Parametry PWM se mohou lišit pro různé jednotky v závislosti na vlastnostech kapacitorů.

Syntaxe

```
--> poweron
```

C.41 Poweroff

Popis

Příkaz vypne napájecí okruhy na jednotce.

Syntaxe

```
--> poweroff
```

D Tabulka pinů

D.1 Analogově-digitální převodník

Označení vstupu na jednotce	Označení na MCU	Příkazy
ADIN1H	AD1IN0	readadc (C.3)
ADIN1L		
ADIN2H	AD1IN1	
ADIN2L		
ADIN3H	AD1IN2	
ADIN3L		
ADIN4H	AD1IN3	
ADIN4L		
ADIN5H	AD1IN4	
ADIN5L		
ADIN6H	AD1IN5	
ADIN6L		
ADIN7H	AD1IN6	
ADIN7L		
ADIN8H	AD1IN7	
ADIN8L		
ADIN9H	AD1IN8	
ADIN9L		
ADIN10H	AD1IN9	
ADIN10L		
ADIN11H	AD1IN10	
ADIN11L		
ADIN12H	AD1IN11	
ADIN12L		

D.2 Digitální vstupy

Označení vstupu na jednotce	Řídící integrovaný obvod	Označení na MCU	Příkazy
DIN0	MC33972 [9]	SPI1:0	dinportval (C.13), dinstat (C.14), dinswst (C.12), dinreset (C.11), dinportint (C.10), dinportpr (C.9), dinportst (C.8)
DIN1			
DIN2			
DIN3			
DIN4			
DIN5			
DIN6			
DIN7			
DIN8		SPI1:0	
		GIOA 0	
DIN9		SPI1:0	
		GIOA 1	
DIN10		SPI1:0	
		GIOA 2	
DIN11		SPI1:0	
		GIOA 3	
DIN12		SPI1:0	
		GIOA 4	
DIN13		SPI1:0	
		GIOA 5	
DIN14		SPI1:0	
		GIOA 6	
DIN15		SPI1:0	
		GIOA 7	

D.3 Digitálně-analogový výstup

Označení výstupu na jednotce	Řídící integrovaný obvod	Označení na MCU	Příkazy
DAC1	MCP4922 [12]	SPI3:0	dacpinval (C.6), dacstat (C.7), dacpinon (C.5)
DAC2			
DAC3	MCP4922 [12]	SPI3:1	
DAC4			

D.4 Výstup do 10 A (HOUT)

Označení výstupu na jednotce	Řídící integrovaný obvod	Expandované piny	Označení na MCU	Příkazy
HOUT1	AUIR33401 [10]	HOUT1_DIAG	NHET1 17	houtfail (C.25), readhifbk (C.26), houtpwm (C.27), houtpwmstart (C.28), houtpwmstop (C.29)
		HOUT1_IN	NHET1 16	
		HOUT1_IFBK	AD1IN 19	
HOUT2		HOUT2_DIAG	NHET1 19	
		HOUT2_IN	NHET1 18	
		HOUT2_IFBK	AD1IN 20	
HOUT3		HOUT3_DIAG	NHET1 21	
		HOUT3_IN	NHET1 20	
		HOUT3_IFBK	AD1IN 21	
HOUT4		HOUT4_DIAG	NHET1 23	
		HOUT4_IN	NHET1 22	
		HOUT4_IFBK	AD1IN 22	
HOUT5		HOUT5_DIAG	NHET1 27	
		HOUT5_IN	NHET1 25	
		HOUT5_IFBK	AD1IN 23	
HOUT6		HOUT6_DIAG	NHET1 31	
		HOUT6_IN	NHET1 29	
		HOUT6_IFBK	AD1IN 24	

D.5 Výstup do 2 A (MOUT)

Označení výstupu na jednotce	Řídící integrovaný obvod	Expandované piny	Označení na MCU	Příkazy
MOUT1	VNH5019A [18]	MOUT1_IN	NHET1 9	pinval (C.34), portval (C.36)
		MOUT1_EN	DMM_DATA 9	
MOUT2		MOUT2_IN	NHET1 14	
		MOUT2_EN	DMM_DATA 10	
MOUT3		MOUT3_IN	GIOB 7	
		MOUT3_EN	GIOB 5	
MOUT4		MOUT4_IN	GIOB 6	
		MOUT4_EN	GIOB 4	
MOUT5		MOUT5_IN	GIOB 3	
		MOUT5_EN	GIOB 1	
MOUT6		MOUT6_IN	GIOB 2	
		MOUT6_EN	GIOB 1	

D.6 Výstup do 100 mA (LOUT)

Označení výstupu na jednotce	Řídící integrovaný obvod	Označení na MCU	Příkazy
LOUT1	NCV7608 [16]	SPI1:1	loutstat (C.32), loutval (C.31), loutwr (C.30)
LOUT2			
LOUT3			
LOUT4			
LOUT5			
LOUT6			
LOUT7			
LOUT8			

D.7 H-můstek

Označení výstupu na jednotce	Řídící integrovaný obvod	Expandované piny	Označení na MCU	Příkazy
HBR1, HBR2	L99H01 [18]	HBR_EN	NHET1 1	hbrwdg (C.19), hbrpwm (C.20), hbrpwmstart (C.21), hbrpwmstop (C.22), hbren (C.23), hbrdir (C.24)
		HBR_DIR	NHET1 4	
		HBR_PWM	NHET1 7	
		SPI	SPI4:0	

D.8 CAN High speed

Označení výstupu na jednotce	Řídící integrovaný obvod	Expandované piny	Označení na MCU	Příkazy
CANH 1, CANL 1	TJA1041 [14]	TXD	CAN1TX	testcan (C.4)
		RXD	CAN1RX	
		EN	DMM_DATA 13	
		STB	DMM_DATA 11	
CANH 2, CANL 2		TXD	CAN2TX	
		RXD	CAN2RX	
		EN	DMM_DATA 13	
		STB	DMM_DATA 11	
CANH 3, CANL 3		TXD	CAN3TX	
		RXD	CAN3RX	
		EN	DMM_DATA 13	
		STB	DMM_DATA 11	

D.9 FlexRay

Označení výstupu na jednotce	Řídící integrovaný obvod	Expandované piny	Označení na MCU	Příkazy
FRAY1-BP, FRAY1-BM	TJA1082 [15]	TXD	FRAYTX1	testfrayA (C.17), testfrayB (C.18)
		RXD	FRAYRX1	
		TXEN	FRAYTXEN1	
		SPI	SPI4:1	
FRAY2-BP, FRAY2-BM		TXD	FRAYTX2	
		RXD	FRAYRX2	
		TXEN	FRAYTXEN2	
		SPI	SPI4:2	

D.10 LIN

Označení výstupu na jednotce	Řídící integrovaný obvod	Expandované piny	Označení na MCU	Příkazy
LIN1	TJA1021 [13]	TXD	LINTX	testlin (C.33)
		RXD	LINRX	
		SLP	DMM_DATA 15	
LIN2		TXD	SCITX	
		RXD	SCIRX	
		SLP	DMM_DATA 14	

D.11 Napájecí okruhy

Označení výstupu na jednotce	Označení na MCU	Příkazy
VBAT_EN	DMM_DATA 3	poweron (C.40), poweroff (C.41), pinval (C.34)
VBAT1_EN	DMM_DATA 0	pinval (C.34)
VBAT2_EN	DMM_DATA 1	pinval (C.34)
VBAT3_EN	DMM_DATA 2	pinval (C.34)