ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA ELEKTROTECHNICKÁ

KATEDRA APLIKOVANÉ ELEKTRONIKY A TELEKOMUNIKACÍ

DIPLOMOVÁ PRÁCE

Systém pro měření pomocí protokolu XCP

Bc. Martin Hás **2018**

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta elektrotechnická
Akademický rok: **2017/2018**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Martin HÁS**
Osobní číslo:	**E15N0071P**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Elektronika a aplikovaná informatika**
Název tématu:	**Návrh a aplikace "systému pro měření protokolem CCP/XCP" pro mikroprocesory**
Zadávající katedra:	**Katedra aplikované elektroniky a telekomunikací**

Zásady pro vypracování:

1. Podrobně prostudujte (z hlediska mikroprocesorových systémů) problematiky:
   a)protokol CCP/XCP
   b)testování softwaru
2. Navrhněte prototyp mikroprocesorového systému umožňující měření fyzikálních veličin (napětí, teplota,...) a vyčítání kalibračními protokoly.
3. Realizujte navržený systém a začleňte jej do stávajícího testovacího prostředí.

Rozsah grafických prací: **podle doporučení vedoucího**

Rozsah kvalifikační práce: **40 - 60 stran**

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

Student si vhodnou literaturu vyhledá v dostupných pramenech podle doporučení vedoucího práce.

Vedoucí diplomové práce: **Ing. Kamil Kosturik, Ph.D.**
Katedra aplikované elektroniky a telekomunikací

Datum zadání diplomové práce: **10. října 2017**

Termín odevzdání diplomové práce: **24. května 2018**

Doc. Ing. Jiří Hammerbauer, Ph.D.
děkan

L.S.

Doc. Dr. Ing. Vjačeslav Georgiev
vedoucí katedry

V Plzni dne 10. října 2017

Anotace

Tato diplomová práce se zabývá návrhem a realizací systému s mikrokontrolérem, který komunikuje s nadřazeným systémem pomocí měřícího a kalibračního protokolu XCP, realizovaného po sběrnici CAN. Protokol XCP slouží především k získávání dat a jejich zápisu do systému s mikrokontrolérem. Teoretická část práce se zabývá testováním softwaru a stručně shrnuje základy protokolu XCP. V další části práce jsou uvedeny požadavky zadavatele na realizovaný systém. V praktické části je nejdříve popsán hardware systému, tj. výběr součástek a výsledná deska plošných spojů. Dále je řešeno programové vybavení systému. U toho jsou představeny pomocné vývojové nástroje, dále je uveden vývojový diagram a detailní popis aplikace. Poslední část kapitoly tvoří soupis implementovaných příkazů XCP a objektů na pevně daných adresách v paměti mikrokontroléru.

Klíčová slova

XCP, CCP, kalibrační protokol, měřící systém, STM8

Abstract

This diploma thesis deals with the design and the realization of a system with a microcontroller, which communicates with its master system via the measurement and calibration protocol XCP realized over a CAN bus. XCP protocol serves primarily for the purpose of the acquisition of the data and their writing to the microcontroller system. The theoretical part of this thesis deals with the software testing and briefly summarizes the basics of XCP. In the next part there are stated the requirements for the system as requested by its contractor. In the practical part of the thesis the hardware of the system is described first, that is the choice of the components and the resulting printed circuit board. The software of the system is discussed next. Auxiliary development tools are described and then the application flowchart and the detailed application's description are presented. The last part of the chapter consists of the list of all the implemented XCP commands and objects located on the fixed addresses in the memory of the microcontroller.

Key words

XCP, CCP, calibration protocol, measurement system, STM8

Prohlášení

Předkládám tímto k posouzení a obhajobě diplomovou práci, zpracovanou na závěr studia na Fakultě elektrotechnické Západočeské univerzity v Plzni.

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně, s použitím odborné literatury a pramenů uvedených v seznamu, který je součástí této diplomové práce.

Dále prohlašuji, že veškerý software, použitý při řešení této diplomové práce, je legální.

V Plzni dne 24. 5. 2018 Martin Hás

……..

Poděkování

Tímto bych rád poděkoval společnosti ZF jako zadavateli této práce a především pak Ing. Danielu Fritzovi za vedení při tvorbě této práce.

Další poděkování patří všem lidem z Fakulty elektrotechnické Západočeské univerzity v Plzni, kteří mě provázeli během studia na této škole.

Obsah

Úvod

Zadavatelem tématu této práce i požadavků na řešený systém, stejně tak jako jeho budoucím uživatelem, je testovací oddělení plzeňské pobočky společnosti ZF. Německá skupina ZF je významným celosvětovým dodavatelem komponent v oblasti automobilového průmyslu, který se zaměřuje především na vývoj a výrobu automobilových převodovek. Vzhledem k tomu, že důkladné testování je (obzvláště v automobilovém průmyslu) v dnešní době nedílnou součástí vývoje kterékoliv součásti obsahující nejenom elektroniku a případně i její obslužný software, jakákoliv možnost automatizace testovacího procesu tak může významně zkrátit dobu vývoje takové součásti.

Tato práce se zabývá právě návrhem specifického systému s mikrokontrolérem, který by umožnil automatizované měření na ventilech v tzv. „loadboxu“, jak se v ZF označuje zařízení, které se nachází v rozměrné skříni testovacího simulátoru na principu techniky „hardware-in-the-loop“ a které během testování simuluje např. ovládání automatické převodovky či uzávěrek diferenciálů v automobilu. Ventily jsou konfigurovány pomocí spínání reléových kontaktů. Požadované zařízení proto musí umožnit tuto konfiguraci ventilů spínáním a rozepínáním reléových kontaktů. Vzhledem k tomu, že ventily se při provozu zahřívají, musí systém taktéž podporovat souběžné měření teploty na těchto ventilech. Kromě kombinace digitálních výstupů a analogových vstupů, které budou sloužit primárně k výše uvedenému, by měl systém navíc ještě podporovat standardní komunikační rozhraní, jakými jsou I^2C, UART a SPI. Tato rozhraní mohou být použita k dalšímu rozšíření systému, jehož mikrokontrolér je v základu omezen počtem pinů pouzdra. Umožní se tak například připojení dodatečných vstupně-výstupních expandérů, digitálně-analogových převodníků či teplotních senzorů. Pro umožnění co nejsnazší konektivity by měly být všechny vstupy, výstupy a rozhraní vyvedeny na přístupné konektory. Jako určitý přídavek navíc k návrhu tohoto systému je možné provést volbu nejvhodnějšího typu teplotního senzoru pro měření teploty na ventilech.

Nejdůležitější částí této práce je ovšem implementace podpory protokolu XCP do programového vybavení mikrokontroléru v systému. XCP je v diagnostice a testování v automobilovém sektoru hojně využívaný protokol, neboť umožňuje univerzálním a předepsaným způsobem realizovat komunikaci nadřízeného ovládacího systému s měřícím a/nebo kalibračním systémem. Na straně ovládacího systému pak lze automatizovaným způsobem získávat či zapisovat data, což v našem případě znamená konfigurovat ventily

v loadboxu a získávat údaje o jejich teplotě. Vzhledem k bohaté škálovatelnosti implementace protokolu XCP je samozřejmostí, že jeho implementace v systému by měla umožňovat minimálně veškerou funkcionalitu potřebnou ke konfiguraci a měření teploty na těchto ventilech. Systém s jeho implementací protokolu XCP bude jako přenosové prostředí využívat (původně) automobilovou sběrnici CAN. Vzniklá implementace protokolu XCP bude muset být bezpodmínečně kompatibilní s ovládacím systémem užívaným v ZF.

Nakonec je třeba ještě zmínit, proč jsem si pro svou diplomovou práci zvolil mezi jinými právě toto téma. Diplomovou práci obecně vnímám jako záležitost, kde by člověk měl především co nejvíce uplatnit či prohloubit znalosti a zkušenosti, získané během studia vysoké školy. A z mého pohledu právě práce s tímto tématem vše zmíněné zahrnuje v sobě. To znamená věci, se kterými jsem se často setkal či se je dokonce naučil používat, možná však ne vždy úplně autonomně. V této práci jsou v první řadě záležitosti čistě elektrotechnické – návrh systému, výběr a soupis součástek, deska plošných spojů. Další část tvoří samotné mikrokontroléry a programování – výběr a užití mikrokontroléru, se kterým jsem do té doby neměl zkušenost a jeho programování za neustálého studia manuálu. A do třetice je to práce podle přesně dané specifikace, kterou je samotná implementace protokolu XCP. Konečné zhodnocení toho, jak se mi užití zmíněných věcí v této práci podařilo, už ovšem nechávám na Vás.

Seznam obrázků

Seznam tabulek

Seznam symbolů a zkratek

A/D	Analogově-digitální
ASAM	Association for Standardisation of Automation and Measuring Systems (sdružení výrobců v automobilovém průmyslu)
CAN	Controller Area Network (typ původně automobilové komunikační sběrnice)
CCP	Can Calibration Protocol (předchůdce protokolu XCP)
CS	Chip Select (výběrový signál, např. u SPI)
CTO	Command Transfer Object (objekt pro přenos příkazů)
DAQ	Data Acquisition (akvizice dat)
DPS	Deska plošných spojů
DTO	Data Transfer Object (objekt pro přenos dat)
ECU	Electronic Control Unit (elektronická řídící jednotka)
EEPROM	Electrically Erasable Programmable Read-Only Memory (typ nevolatilní paměti)
EMC	Electromagnetic compatibility (elektromagnetická kompatibilita)
HIL	Hardware-in-the-loop (typ simulátoru v automobilovém průmyslu)
I^2C	Inter-Integrated Circuit (typ sériové sběrnice)
ID	Identifier (identifikátor)
JTAG	Joint Test Action Group (standardizované diagnostické rozhraní)
MTA	Memory Transfer Address (adresa pro paměťový přenos)
ODT	Object Description Table (popisná tabulka objektu)
PC	Personal Computer (osobní počítač)
PID	Packet Identifier (identifikátor paketu)
RAM	Random-access Memory (typ paměti)
SPI	Serial Peripheral Interface (typ sériového rozhraní)
STIM	Stimulation (stimulace)
SWD	Serial Wire Debugging (typ programovacího rozhraní)
SWIM	Single Wire Interface Module (typ programovacího rozhraní)
THT	Through-hole Technology (technologie součástek s vývody skrz desku)
UART	Universal Asynchronous Receiver-Transmitter (typ sériového rozhraní)
USB	Universal Serial Bus (typ sériového rozhraní)

1 Teoretická část

1.1 Testování softwaru

1.1.1 Obecně o softwarovém testování

Testování softwaru je v moderní době jeden z neopomenutelných kroků při vývoji softwaru pro spotřebitelské aplikace. Testování je způsob, jak odhalit chyby v softwaru a ujistit se, že je software vhodný pro praktické použití. Obecně lze definovat, že testování softwaru je způsob zjištění a potvrzení toho, že vytvořený program či aplikace [1]:

- splňuje technické a obchodní požadavky, které byly stanoveny před jeho vytvořením,
- je reálně použitelný,
- může být spolehlivě implementován na stanoveném zařízení,
- reaguje korektně na všechny vstupní stimuly,
- vykonává svou funkci v předepsaném čase.

Softwarové testování není ničím z následujících bodů, či neslouží k žádnému z následujících účelů [2]:

- Softwarové testování není laděním (debuggingem) softwaru. Smyslem ladění je odstraňovat chyby. Existence chyb a jejich přibližné umístění je při ladění známé. Ladění nemusí být dokumentováno. Neexistuje k němu žádná přesná specifikace ani jeho zápis. Ladění je sice výsledkem testování, ale nikdy není jeho náhradou.
- Softwarové testování nikdy nedokáže odhalit naprosto všechny chyby v softwaru. Různé druhy testování mohou odhalit různé druhy chyb. Při testování vždy zůstane určité množství chyb skryto.
- Softwarové testování má za úkol odhalit chyby v softwaru a nikoliv jejich příčiny. Součástí testování tak není žádné napravování chyb či úprava funkcionality. Výsledkem testování softwaru je zápis o testování (test report). Tester v žádném případě nesmí upravovat kód, který je testován. To je práce softwarového vývojáře, který tak činí na základě zápisu, který obdrží od testera softwaru.

Charakter softwarového testování lze naopak upřesnit v následujících bodech [2]:

- Softwarové testování je formální aktivitou. Řídí se strategií a systematickým přístupem. Různé fáze testování se vzájemně doplňují. Testování je vždy přesně specifikováno a zaznamenáváno.
- Softwarové testování je plánovaná aktivita. Způsob práce i předpokládané výsledky

jsou známé předem. Doba trvání jednotlivých fází testování může být spolehlivě odhadnuta. Bod v čase, kdy bude testování softwaru ukončeno, je definovaný.

- Výsledky softwarového testování jsou oficiálními důkazy o kvalitě testovaného softwaru.

1.1.2 Druhy testovacích metod

Metody softwarového testování mohou být obecně rozděleny do následujících skupin [2]:

- statické testy,
- dynamické testy,
- strukturální testy,
- funkční testy.

Během statického testování není testovaný software opravdu spouštěn, ale místo toho je analyzován v tzv. „offline“ režimu. Ve statickém testování lze ověřit i kód, který ještě nemusel nevzniknout – například lze upravit či upřesnit dokumentaci či zpřesnit výsledky a odhady. Samotný kód může být podroben inspekci – může být kontrolována jeho struktura, syntaxe, tok dat, popř. může být ověřena činnost kompilátoru tohoto kódu. Statické testování softwaru v sobě zahrnuje tzv. verifikaci. [1] [2]

Dynamické testy se na rozdíl od statických testů provádějí při běhu testovaného programu. Takto lze testovat i program, který nemusí být kompletně implementovaný – mohou být využity pouze jeho funkční celky (diskrétní funkce nebo moduly). Dynamické testování může probíhat na cílovém zařízení, na emulátoru či na simulátoru. Dynamické testování kromě verifikace zahrnuje i tzv. validaci. [1] [2]

Strukturální testy mohou být také nazývány testy tzv. „bílých skříněk“ (popř. „skleněných skříněk“). Jsou prováděny se znalostí zdrojového kódu testovaného softwaru. Během strukturálního testování softwaru je testovaný systém stimulován určitým způsobem tak, aby se program „vydal“ specifickými cestami, přes požadované větve kódu. Systém je na vstupech zatížen kritickými hraničními hodnotami či dokonce neplatnými vstupními hodnotami. Chování takového systému je pak zaznamenáno a porovnáno s chováním předpokládaným. [2]

Funkční testování může být také nazýváno testováním tzv. „černých skříněk“. V takovém případě je k systému přistupováno tak, jako by nebyla známa jeho vnitřní struktura. Na vstupy

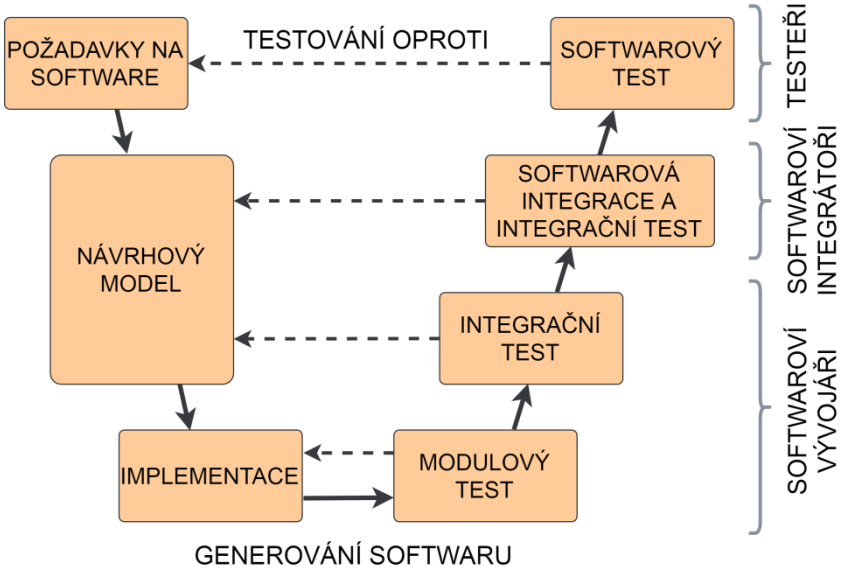
systému jsou přivedeny určité stimuly a výstup systému je zaznamenáván a porovnáván s předpokládanými výsledky. [2]

1.1.3 Fáze testování

Různé druhy testování jsou schopny odhalit různé druhy chyb. Proto nelze spoléhat pouze na jeden typ testování na úkor testů ostatních. Provádět softwarové testování je také nutné během celého procesu vývoje softwaru. Obecně platí pravidlo, že čím později je určitá chyba odhalena, tím složitější a dražší je její odstranění.

Testování softwaru jako celku lze rozdělit do tří různých fází průběhu testování. V různé fázi návrhu a vývoje softwaru jsou prováděny různé fáze softwarového testování. Těmito postupnými fázemi testování jsou:

- modulové testy,
- integrační testy,
- softwarové (systémové) testy.



Obr. 1: *Příklad konkrétního postupu při vývoji a testování softwaru. Vlastní tvorba.*

Modul je nejmenší kompilovatelnou jednotkou kódu. Často nemůže být dostatečně velký na to, aby mohl být otestován funkčním testem. Naopak je ovšem ideálním kandidátem na to, aby byl otestován testem strukturálním. Modul bude obvykle testován nejdříve statickými testy, a až později testy dynamickými. [2]

Jak jsou do sebe během vývoje programu integrovány různé moduly, je nutné postupně testovat jejich vzájemnou interakci. Příkladem integrování zde může být třeba spuštění operačního systému jako základního prvku aplikace a následné přidávání jednotlivých

modulů. Postupným testováním takového systému jako černé skříňky lze odhalit, v kterém bodě integrace systém začne vykazovat chyby. [2]

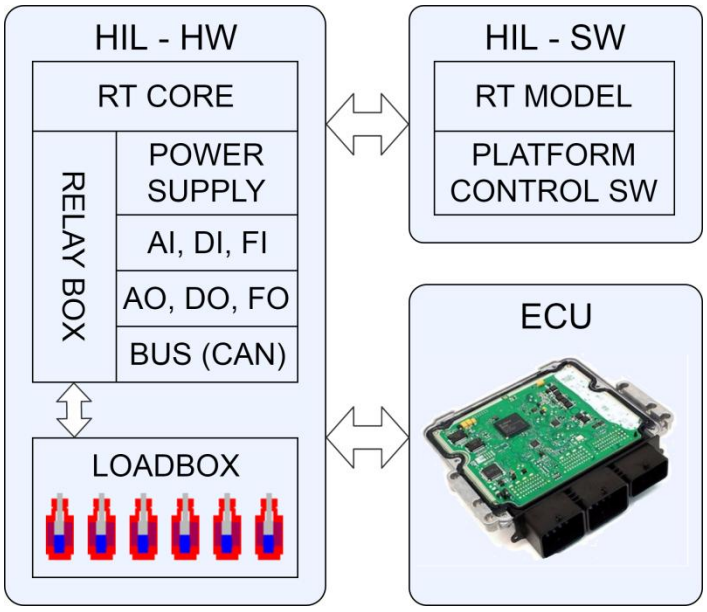
Modulové testy a integrační testy jsou prováděny během postupné tvorby testovaného softwaru softwarovými vývojáři. Během integrování softwaru jsou integrační testy prováděny za pomoci rozšířených testovacích prostředí. Softwaroví vývojáři, softwaroví integrátoři a softwaroví testeři jsou v tomto případě obvykle všichni zapojeni do testování.

Softwarové testy jsou již prováděny na kompletním softwarovém systému v požadovaném provozním prostředí. Softwarové testy jsou prováděny buďto funkčními vývojáři, či samostatnými testery.

1.1.4 Popis simulátoru HIL

Pro vývoj a testování komplexních vestavěných systémů, které pracují v reálném čase, mohou být využity rozsáhlé simulační platformy, anglicky zvané „hardware-in-the-loop“ (HIL). Vzhledem k tomu, že k testování softwaru v oboru automobilového průmyslu nemusí být vždy možné či vhodné testovat aplikaci během jejího vývoje přímo v automobilu za provozu, neboť to většinou znamená velkou časovou i finanční náročnost a obtížnou reprodukovatelnost testování, je obvykle vhodnější takovou aplikaci při vývoji testovat v laboratorním prostředí. [3]

Simulátory HIL umožňují k tomuto účelu procesy v jednotlivých subsystémech skutečného vozu napodobit pomocí složitých matematických modelů tak, aby se jejich výsledek co nejvíce blížil podmínkám v reálném automobilu. Software simulátoru HIL spočívá v tzv. modelu v reálném čase, který běží v časové smyčce (s typickou periodou 1 ms). S každým cyklem je poté přečten stav všech vstupních signálů, za pomoci matematického modelu vypočten stav nový a ten je nastaven na výstupní signály. Obsluha simulátoru může do parametrů matematického modelu během testování vstupovat a ovlivňovat tak běh modelu (například ovládat signály odpovídající rychlosti vozu či natočení volantu nebo vkládat různé chyby).



Obr. 2: *Blokové schéma simulátoru HIL. Vlastní tvorba.*

Během testování je poté řídící jednotka s testovaným softwarem připojena k simulátoru HIL. Řídící jednotka získává ze simulátoru veškerá stimulační data (simulace reálných čidel). Také její výstupy jsou připojeny na reálné či simulované zátěže nebo akční členy. Díky možnosti ovlivňovat vstupní signály řídící jednotky lze pak jednoduše testovat reakce softwaru na vzniklé chyby (např. rozsahy signálů mimo stanovené meze, odpojení signálů či jejich zkraty apod.).

1.2 Protokol XCP

1.2.1 Obecně o XCP

„Univerzální měřící a kalibrační protokol“ XCP vychází z protokolu CCP (CAN Calibration Protocol), který je jako kalibrační protokol určen výhradně pro realizaci na sběrnici CAN. Protokol XCP je jakýmsi jeho vylepšením a „zobecněním“ a k podpoře sběrnice CAN přidává i možnost jeho realizace na mnoha dalších, často užívaných přenosových médiích („X“ v názvu symbolizuje právě variabilní fyzickou vrstvu). Protokoly CCP a XCP jsou standardizovány německou organizací ASAM, sdružující mezinárodní výrobce, dodavatele a vývojáře v oblasti automobilového průmyslu. Klíčovým přispěvatelem protokolu XCP je německá společnost Vector Informatik GmbH. Ta k protokolu XCP poskytuje informační podporu a mnoho vývojových nástrojů.

Obr. 3: *Logo protokolu XCP. Převzato z [4].*

Prvotní verze protokolu CCP byla standardizována roku 1995 a poslední, stále aktuální verze pochází z roku 1999. Kromě rozdílných specifik přenosových vrstev standardizace CCP, na rozdíl od pozdější standardizace XCP, postrádá i některé významné části. Například neobsahuje časové značky doprovázející získávaná data či nemá konzistentní podporu možnosti programování paměti typu flash. Navíc počátkem nového tisíciletí vznikla potřeba podpory modernějších a značně rychlejších přenosových médií, než je sběrnice CAN (např. Ethernet). Proto byla v roce 2003 standardizována první verze protokolu XCP. Od té doby navíc přibyly ještě tři aktualizace. Jednotlivé verze protokolu XCP se vyznačují zpětnou kompatibilitou s verzemi předchozími.

Verze protokolu XCP jsou následující [4]:

- XCP 1.0 (2003) – Specifikace základních prvků pro měření a kalibraci, synchronní měření a stimulace, přepínání stránek a programování nevolatilní paměti. Podpora transportních vrstev CAN, Ethernet (UDP a TCP/IP), SPI a USB.
- XCP 1.1 (2008) – Přidání podpory pro transportní vrstvu FlexRay.
- XCP 1.2 (2013) – Přidání podpory deskriptoru A2L-IF_DATA pro výpočet předpokládaného využití prostředků v řídící jednotce.

- XCP 1.3 (2015) – Přidání podpory základních prvků pro různé stavy řídící jednotky, obcházení chybové obsluhy a časové korelace.

Z předchozího je patrné, že většina základní funkcionality nutné pro plnohodnotné fungování protokolu XCP byla obsažena již v původní standardizaci XCP 1.0 z roku 2003.

Při návrhu protokolu XCP byl dle jeho tvůrců kladen důraz především na následující základní teze [4]:

- minimální obsazení dostupných prostředků v řídící jednotce a snaha o jejich co nejefektivnější využití,
- efektivní komunikace,
- škálovatelnost,
- přenositelnost,
- jednoduchá implementace v podřízených jednotkách,
- plug-and-play konfigurace jen s několika málo parametry,
- synchronní stimulace dat,
- přesná akvizice naměřených dat díky snímání odběru časových značek.

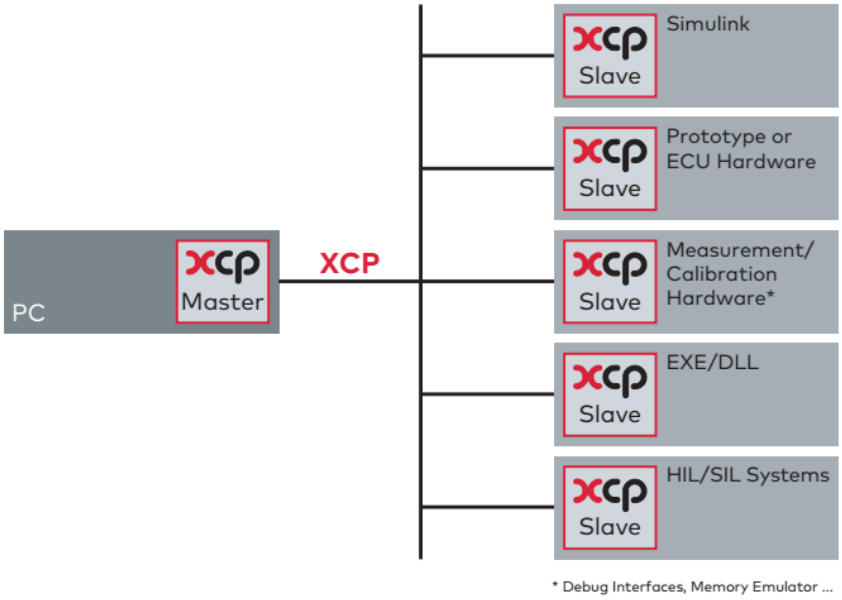
Další z předností protokolu XCP je jeho nezávislost na fyzické a transportní vrstvě a z toho plynoucí možnost využití na mnoha různých přenosových vrstvách. Jak už vyplynulo z předchozího textu, protokol XCP tak lze provozovat v následujících variantách:

- XCP na CAN a CAN FD,
- XCP na SxI (SPI, SCI),
- XCP na Ethernetu (TCP/IP a UDP/IP),
- XCP na USB,
- XCP na FlexRay.

Topologie XCP je založena na principu „master/slave“. To obecně znamená, že komunikaci iniciuje řídící prvek („XCP master“). Jednotkou master bude v nejčastějším případě osobní počítač, na kterém běží specifický software (klasicky např. software CANape od firmy Vector). Toto PC pak bude komunikovat buď přímo, či pomocí nějakého externího zařízení s jednou či více podřízenými jednotkami, označovanými jako „XCP slave“. Jednotek slave může být na společné komunikační sběrnici více a master musí zvolit tu, s níž se pokouší komunikovat.

Podřízenou jednotku slave si lze v nejjednodušším případě představit jako elektronické zařízení typu mikrokontroléru, hradlového pole, programovatelného logického

automatu apod. V automobilové elektronice lze takový prvek obecně označit jako tzv. elektronickou řídící jednotku (electronic control unit, ECU), což je označení pro komplexnější elektronické zařízení, které má na starosti určitý subsystém v automobilu. Ovšem kromě ECU může XCP slave být také například simulačním systémem HIL či diagnostickým prostředím, jak ilustruje následující obrázek.



Obr. 4: *Znázornění struktury protokolu XCP a variability zařízení slave. Převzato z [4].*

Klíčová a zároveň nejdůležitější funkcionalita protokolu XCP je ta, že umožňuje čtení z paměti a zápis do paměti jednotek slave. Elektronické řídící jednotky jsou systémy s diskrétním chováním v čase. Jejich parametry se mění v určitých časových intervalech – tedy pouze, když procesor změří či vypočítá určitou hodnotu a uloží ji do paměti RAM. XCP kromě standardního čtení z paměti podporuje také funkci čtení paměti v přesně stanovených časových intervalech. Tyto intervaly mohou být synchronní vůči událostem, probíhajícím uvnitř ECU a data získaná tímto synchronním měřením tedy přesně korelují s těmito událostmi. Možnost zápisu do paměti jednotky slave umožňuje přesnou optimalizaci parametrů algoritmů probíhajících uvnitř. Protokol XCP i v případě zápisu opět poskytuje podporu pro funkci synchronního přístupu do paměti. Kombinace čtení z paměti a zápisu do ní a možnost jejich vzájemné synchronizace u protokolu XCP tedy vedou k efektivní využitelnosti v ladění systémových procesů uvnitř ECU.

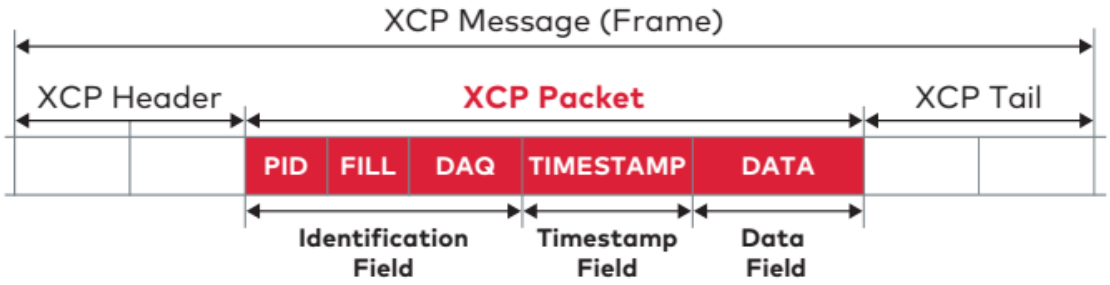
Přístup do paměti se v případě čtení i zápisu provádí se znalostí adres požadovaných parametrů v paměti zařízení slave. Master k tomuto účelu pracuje s tzv. „datovým popisným souborem ASAP2“, zkráceně označovaného jako soubor A2L. Uživatel pak pracuje pouze

se symbolickými názvy objektů uvnitř ECU a tyto názvy jsou pomocí souboru A2L překládány na skutečné adresy, které master využívá při komunikaci s jednotkou slave. Kromě samotné adresy tedy musí soubor A2L znát i datový typ objektu, způsob jeho uložení v paměti či pravidla pro jeho převod na skutečnou fyzikální jednotku. Soubor A2L dále specifikuje přenosovou cestu a její vlastnosti. [5]

1.2.2 Úvod do protokolové vrstvy XCP

Komunikace mezi jednotkami master a slave v prostředí protokolu XCP probíhá výměnou zpráv principem „dotaz – odpověď“. Pro upřesnění je XCP z protokolového hlediska soustavou „soft master/slave“. Ačkoliv master vždy úkoluje jednotku slave, tak té je dovoleno posílat jednotce master některé druhy zpráv plně autonomně. Fyzická přenosová vrstva, na které je protokol XCP realizován, musí vždy zajistit to, aby každá jednotka slave obdržela pouze zprávu, která je jasně určena pouze jí. Takže třeba v případě protokolu XCP na sběrnici CAN je zpráva adresována pomocí CAN ID, u XCP na SPI je adresace řešena selekcí jednotek slave signálem CS apod. Tímto způsobem může být ve stejné síti aktivních několik souběžných komunikačních kanálů typu „single-master/single-slave“, které se ovšem nesmí vzájemně ovlivňovat. [6]

Celý rámec zprávy protokolu XCP je zapouzdřen do rámce zprávy transportní vrstvy (např. zpráva CAN, paket USB). Rámec zprávy XCP je obecně složen ze tří částí: hlavičky XCP, paketu XCP a patičky XCP (viz Obr. 5). Hlavička a patička XCP jsou závislé na protokolu transportní vrstvy (například v případě XCP na sběrnici CAN hlavička úplně chybí a nepovinná patička slouží pouze k zarovnání paketu), zatímco paket XCP je ve zprávě přítomen vždy. Právě paket XCP tvoří jádro zprávy, protože obsahuje datový objekt protokolu XCP.



Obr. 5: *Obecný rámec zprávy protokolu XCP s červeně vyznačenou částí, která označuje paket XCP. Převzato z [4].*

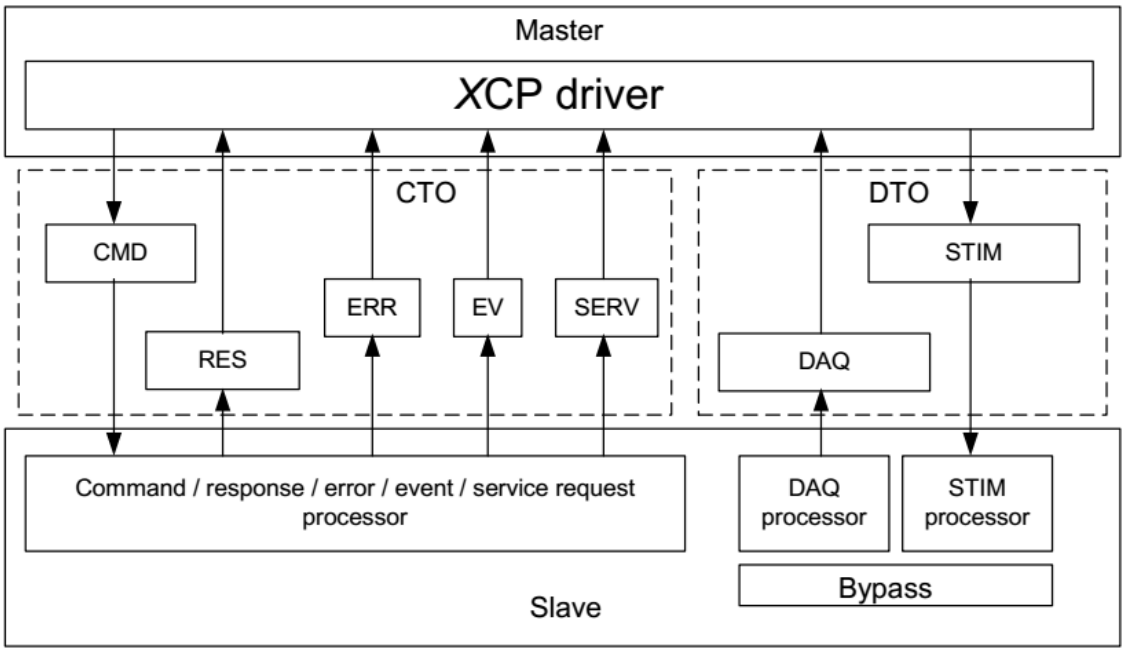
Pakety protokolu XCP mohou být tvořeny dvěma druhy objektů – příkazovými objekty (command transfer objects, CTO) a datovými objekty (data transfer objects, DTO). Objekty CTO jsou používány k přenosu následujících typů zpráv [6]:

- kontrolní příkazy (CMD),
- odezvy na kontrolní příkazy (RES),
- chybová hlášení (ERR),
- hlášení událostí (EV),
- servisní žádosti (SERV).

Pomocí CTO jsou tedy předávány v první řadě příkazy. Slave musí na CMD vždy odpovědět RES nebo ERR. Zbývající typy – EV a SERV – posílá slave sám a asynchronně.

Objekty DTO naproti tomu nepředávají žádné příkazy. DTO slouží výhradně k předávání synchronních dat. Dle směru paketů je lze potom rozlišit na následující typy:

- akvizice dat (DAQ),
- datová stimulace (STIM).



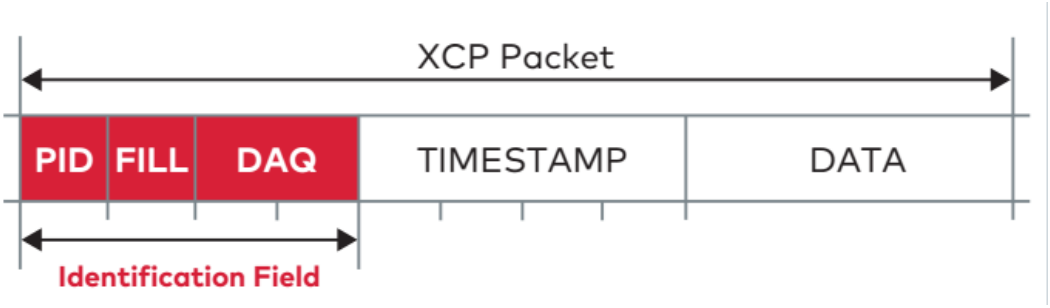
Obr. 6: *Typy příkazových a datových objektů XCP a směry jejich přenosu. Převzato z [6].*

V protokolu XCP se definují dva ukazatele, které určují limit dat v jednom paketu XCP na zvolené přenosové vrstvě. Jsou to:

- MAX_CTO – maximální délka paketu CTO v bytech,
- MAX_DTO – maximální délka paketu DTO v bytech.

Pro XCP, provozované na sběrnici CAN, činí hodnoty MAX_CTO i MAX_DTO 8 bytů (což logicky vyplývá ze shodného maximálního počtu datových bytů jedné zprávy na CANu). V případě zbylých podporovaných transportních protokolů tyto hodnoty činí 8–255 bytů pro MAX_CTO a 8–65535 pro MAX_DTO. [6] Tyto hodnoty jsou pak reálně omezeny především fyzickými limity zařízení slave.

1.2.3 Paket XCP



Obr. 7: *Složení paketu protokolu XCP s vyznačeným identifikačním polem. Převzato z [4].*

Při předávání zpráv musí vždy obě strany být schopny jasně rozlišit povahu přijaté zprávy. K tomu v paketu XCP slouží sekce identifikačního pole. Jejím základem je vždy jednoznačný identifikátor paketu (packet identifier, PID). Velikost PID je 1 byte, může tedy nabývat rozsahu hexadecimálních hodnot 0x00 až 0xFF. Pomocí PID lze ihned rozlišit, zda se jedná o paket CTO či DTO a co tento paket v sobě nese (viz následující tabulka).

Tab. 1: *Rozsahy PID a jim odpovídající typů paketů. Vlastní zpracování dle [4].*

<table>
<tr><th colspan="2">Master → Slave</th><th colspan="2">Slave → Master</th></tr>
<tr><td>0xFF</td><td rowspan="4">CMD</td><td>0xFF</td><td>RES</td></tr>
<tr><td rowspan="2">⋮</td><td>0xFE</td><td>ERR</td></tr>
<tr><td>0xFD</td><td>EV</td></tr>
<tr><td>0xC0</td><td>0xFC</td><td>SERV</td></tr>
<tr><td>0xBF</td><td rowspan="3">STIM</td><td>0xFB</td><td rowspan="3">DAQ</td></tr>
<tr><td>⋮</td><td>⋮</td></tr>
<tr><td>0x00</td><td>0x00</td></tr>
</table>

V případě CTO identifikační pole obsahuje vždy pouze samotné PID. V případě DTO může být PID v identifikačním poli ještě doplněno o položky FILL a DAQ, které slouží k celkovému rozšíření počtu dostupných a stále jednoznačně určených paketů

se synchronními daty. Ve speciálním případě, kdy je přesně známa chronologie odesílaných datových paketů, lze posílat paket DTO i s absencí celého identifikačního pole.

Další částí paketu XCP je nepovinné pole s časovou značkou (timestamp). Ta je dostupná pouze pro pakety DTO. Přítomnost časové značky v paketu DTO, odesílaného jednotkou slave, pomáhá jednotce master přesně určit čas, kdy byla data obsažená ve zprávě získána. Tím se navíc eliminuje nutnost znát dobu, kterou datům trvala cesta přenosovou cestou, neboť časová značka byla odebrána přímo v době jejich vzniku. [5]

Poslední částí paketu XCP je datové pole. V případě paketu CTO datové pole obsahuje data s rozličnými parametry, která doprovázejí identifikátor PID. V paketu DTO pak obsah datového pole tvoří samotná synchronní data.

1.2.4 Pakety CTO

Pakety CTO s kontrolními příkazy (CMD) vždy posílá pouze master jednotce slave. Každému příkazu je jednoznačně přiřazen specifický identifikátor PID v rozsahu hexadecimálních čísel 0xC0 až 0xFF (tj. celkem 63 možných PID pro CMD). Slave je povinen na každý paket s CMD vždy zareagovat buď pozitivní odezvou (RES), anebo zápornou odezvou čili chybovou hláškou (ERR). Paket RES je odeslán, byl-li příkaz úspěšně vykonán. V opačném případě, kdy nebylo možné příkaz vykonat, je použit paket ERR. Paket ERR obsahuje navíc specifický chybový kód, který určuje povahu chybového hlášení (mezi takové patří například: neznámý příkaz, špatná syntaxe příkazu, odepření přístupu do paměti, zaneprázdněnost systému apod.).

Tab. 2: *Struktura paketů CMD, RES a ERR. Vlastní zpracování dle [7].*

Paket	Pozice	Datový typ	Popis
CMD	0	Byte	PID pro CMD (0xC0 až 0xFF)
	1…MAX_CTO−1	Byte	Specifické parametry CMD
RES	0	Byte	PID pro RES (0xFF)
	1…MAX_CTO−1	Byte	Specifické parametry RES
ERR	0	Byte	PID pro ERR (0xFE)
	1	Byte	Chybový kód
	2…MAX_CTO−1	Byte	Specifické parametry ERR

XCP dovoluje pro výměnu paketů s příkazy a jejich odezvami užití celkem tří komunikačních režimů. Základ tvoří standardní komunikační režim, ve kterém vždy jeden

příkaz je následován jednou odezvou (princip dotaz – odpověď). Žádná jiná sekvence v tomto režimu není připuštěna. Standardní komunikační režim je výchozím režimem. Pro urychlení příkazů pro čtení a zápisu do paměti a programování nevolatilní paměti lze použít buď blokový, anebo prokládaný komunikační režim. Blokový režim umožňuje použít blok paketů s příkazy (v případě komunikace od jednotky master), či blok paketů s odezvami (komunikace od jednotky slave). To ovšem znamená, že slave musí být schopen blok příkazů dostatečně rychle zpracovávat. To lze zajistit nastavením specifických parametrů blokového přenosu (minimální časový rozestup v bloku a maximální velikost bloku). V obou předchozích režimech vždy master musí čekat na odezvu od jednotky slave předtím, než může poslat další požadavek. Tuto komplikaci řeší prokládaný komunikační režim, kde master může vyslat více požadavků najednou, které slave postupně zpracovává. Tento režim opět musí být pro korektní fungování omezen specifickým parametrem (maximální velikost fronty příkazů). [7]

Druhou kategorii CTO tvoří pakety s hlášením událostí (EV) a servisními žádostmi (SERV). Pakety EV jsou užívány k tomu, aby slave mohl hlásit specifické asynchronní události. Tyto pakety nejsou potvrzovány, a proto není zajištěno jejich doručení. Implementace paketů EV není povinná. Paket EV vždy obsahuje kód události pro její jednoznačné určení. Mezi takto hlášené události mohou patřit například různé druhy typů dokončení ukládání do nevolatilní paměti či nemožnost stíhat odesílání všech požadovaných synchronních dat. Pakety SERV slouží k požadavku na jednotku master, aby provedla určitou obsluhu zařízení slave. Opět platí, že pakety nejsou potvrzovány a implementace je nepovinná. Paket SERV taktéž obsahuje kód pro jednoznačné určení typu požadavku. Předdefinovaným požadavkem je pouze žádost o resetování jednotky slave, zbytek je uživatelsky definovatelný.

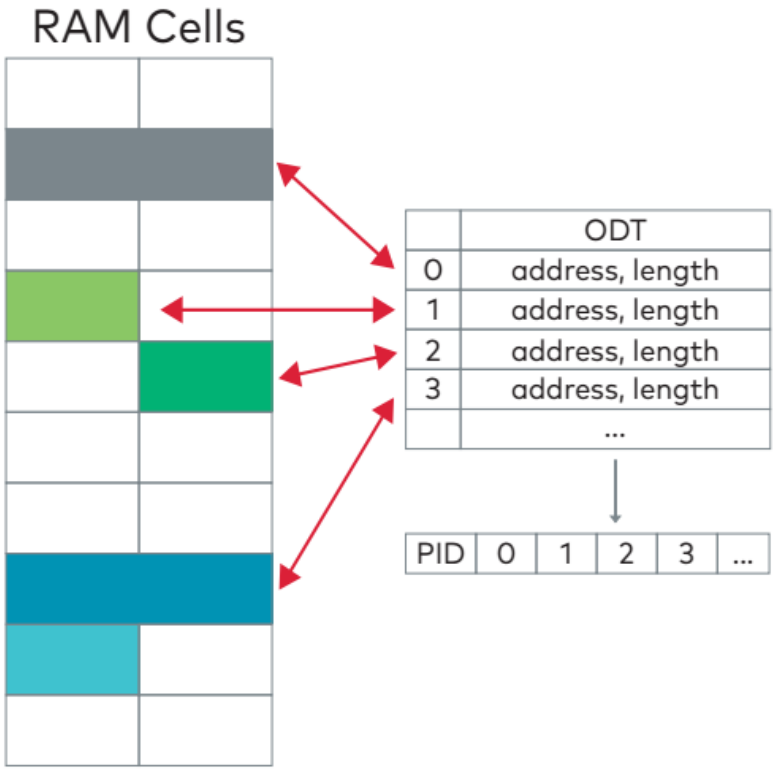
Tab. 3: *Struktura paketů EV a SERV. Vlastní zpracování dle [7].*

Paket	Pozice	Datový typ	Popis
EV	0	Byte	PID pro EV (0xFD)
	1	Byte	Kód události
	2…MAX_CTO−1	Byte	Specifické parametry EV
SERV	0	Byte	PID pro SERV (0xFC)
	1	Byte	Kód požadavku
	2…MAX_CTO−1	Byte	Specifické parametry SERV

1.2.5 Pakety DTO

Pakety DTO slouží pro posílání zpráv, které obsahují synchronní data, získaná pomocí měření či určená pro kalibraci. Rozeznávají se tak dva druhy synchronních datových událostí – datová akvizice (data acquisition, DAQ) a datová stimulace (data stimulation, STIM). Pojmem DAQ se rozumí to, že slave posílá synchronní data jednotce master (dochází k akvizici, tedy získání dat). STIM znamená, že master posílá synchronní data jednotce slave (tedy slave je daty „stimulován“).

DAQ i STIM obecně probíhají podobným způsobem. Na počátku vždy následuje inicializační fáze. V té master během komunikace se zařízením slave nastaví, jaká data a při jaké události se mají posílat a požádá o spuštění tohoto přenosu. V případě, že se jedná o DAQ, vždy, pokud nastane vyžádaná událost, slave zahájí měření a odesílá jednotce master vyžádaná data. V případě kalibrace STIM master posílá data jednotce slave, která si je ukládá stranou. Když nastane požadovaná událost, slave tyto data zapíše na své místo. V obou případech se tak děje, dokud si master nevyžádá zastavení těchto činností. V porovnání se způsobem, kdy by musela probíhat vzájemná výměna příkazů a odezev, tak použití DAQ a STIM umožňuje výrazně efektivnější komunikaci a data jsou při něm čtena či zapisována v přesně stanovených časových okamžicích.



Obr. 8: *Skládání datových elementů do DTO pomocí tabulky ODT. Převzato z [4].*

Vztahy mezi odesílanými datovými objekty a datovými elementy v paměti jednotky slave definují tzv. popisné tabulky objektu (object description tables, ODT). Ty definují, jakým způsobem jsou jednotlivé elementy skládány do paketu DTO. Každá ODT se skládá z určitého počtu položek. Každá taková položka definuje jeden datový element jeho adresou a velikostí v bytech. Jedna ODT potom odpovídá jednomu paketu DTO se specifickým identifikátorem PID. Na jednotce master je, aby zajistila, že celkový počet využitých bytů v každém ODT nepřesáhne svůj limit (stanovený hodnotou MAX_DTO).

Ucelený soubor tabulek ODT tvoří jeden list DAQ. Každý list DAQ je přidělen k určité události uvnitř jednotky slave. Jednotlivé listy DAQ také mohou být selektivně spouštěny či zastavovány. List DAQ může být také spouštěn s různými možnostmi nastavení (jsou-li takové možnosti podporovány). Takto lze například listu DAQ nastavit režim přidávání časové značky. Časová značka se přidává pouze k paketu s první ODT v listu DAQ. Časová značka je získávána z volně běžícího čítače. Pokročilou funkcí listu DAQ je možnost přepínání jeho směru (tj. směr DAQ či směr STIM).

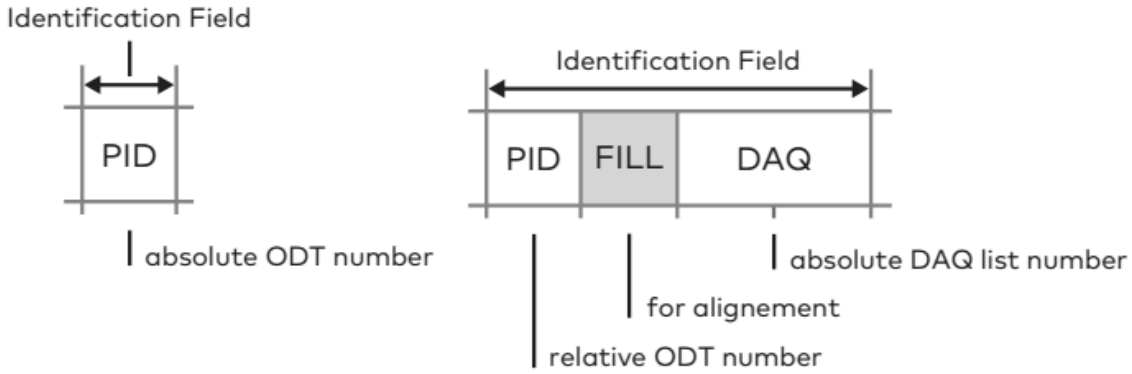
Listy DAQ jsou děleny do následujících tří kategorií:

- předdefinované listy DAQ,
- statické listy DAQ,
- dynamické listy DAQ.

Ve statickém listu DAQ je předem stanovený počet ODT a celkový počet položek pro každou ODT. Elementy v jednotlivých položkách ODT jsou variabilní. Předdefinovaný list DAQ má také předem daný počet ODT a jejich položek, ale jednotlivé položky ODT jsou pevně definované a jejich obsah není proměnný. Největší míru variability umožňují dynamické listy DAQ. U nich má master (namísto pevně daného počtu listů DAQ, ODT a jejich položek) vyhrazenou část paměti zařízení slave o stanovené délce. V ní mohou být dle aktuální potřeby za běhu různě definovány listy DAQ s variabilními počty ODT a jejich položek. Všechny tyto tři typy listů DAQ mohou být užívány souběžně v jedné jednotce slave.

Přidělování identifikátorů paketům DTO může probíhat podle jednoho ze dvou možných způsobů. Identifikační pole pak obsahuje:

- absolutní číslo ODT nebo
- relativní číslo ODT a absolutní číslo listu DAQ.



Obr. 9: *Dva způsoby identifikace paketů DTO. Převzato z [4].*

Při použití absolutních čísel ODT k přidělování PID tvoří identifikační pole pouze samotné PID. Každému listu DAQ je pak přidělen parametr FIRST_PID, což je hodnota identifikátoru PID, kterou nese první ODT z tohoto listu. PID dalších ODT z tohoto listu jsou FIRST_PID+1, FIRST_PID+2 atd. Zařízení slave musí samozřejmě zaručit, aby se PID jednotlivých listů DAQ nepřekrývaly.

Pokud je použit způsob relativního čísla ODT a absolutního čísla listu DAQ, nachází se v identifikačním poli údaj o čísle listu DAQ. Ten může být dlouhý 1 či 2 byty. Identifikátor PID pak obsahuje údaj o pozici ODT uvnitř tohoto listu. Počet možných paketů DTO je tak značně rozšířen. Identifikační pole může obsahovat ještě jeden byte výplně pro zarovnání počtu bytů identifikačního pole na sudý počet.

Protokol XCP umožňuje zařízení slave spustit v tzv. pokračovacím režimu (resume mode). Po resetu se slave podívá, má-li v nevolatilní paměti uložené nějaké listy DAQ z předešlých relací. Pokud tomu tak je, zařízení vstupuje do pokračovacího režimu, kde autonomně začíná odesílat a přijímat data z těchto listů. Z tohoto stavu potom může být jednotka slave přepnuta do stavu CONNECTED (viz sekce Standardní příkazy).

1.3 Příkazy protokolu XCP

Jak již bylo uvedeno, příkazové pakety CMD master používá k zadávání požadavků jednotce slave. Celkový počet příkazů užívaných v protokolu XCP je 57 (s rezervou specifických PID pro dalších 7). Vzhledem k tomu, že XCP je škálovatelné, není nezbytně nutné z tohoto počtu v jednotce slave implementovat každý jednotlivý příkaz. Dostupné příkazy by měl master znát ze souboru A2L. Pokud by se přesto pokusil požádat o provedení příkazu, který v jednotce slave nebyl implementován, měl by obdržet zápornou odezvu s chybovým kódem, který odpovídá chybě neznámého příkazu.

Příkazy protokolu XCP se dělí do pěti základních funkčních skupin. Jsou to:

- standardní příkazy,
- kalibrační příkazy,
- příkazy pro přepínání stránek,
- příkazy pro akvizici a stimulaci dat,
- příkazy pro programování nevolatilní paměti.

Pokud nejsou funkce skupiny příkazů v jednotce slave zapotřebí, není nutné tuto skupinu implementovat (s výjimkou standardních příkazů). Příkazy ve skupinách se dělí na nevolitelné (mandatorní) a volitelné. Je-li potřeba některou ze skupin implementovat, z té musí být implementovány minimálně veškeré mandatorní příkazy. Ze zbývajících volitelných příkazů mohou být implementovány pouze ty, které aplikace vyžaduje – zbytek příkazů může být vynechán.

1.3.1 Standardní příkazy

Tab. 4: *Standardní příkazy protokolu XCP. Vlastní zpracování dle [7].*

Příkaz	PID	Volitelný	Funkce
CONNECT	0xFF	Ne	Naváže logické spojení se slave.
DISCONNECT	0xFE	Ne	Ukončí logické spojení se slave.
GET_STATUS	0xFD	Ne	Vrátí informace o stavu aktuální relace.
SYNCH	0xFC	Ne	Pro synchronizaci komunikace.
GET_COMM_MODE_INFO	0xFB	Ano	Vrátí informace o podpoře kom. režimů.
GET_ID	0xFA	Ano	Poskytne informace o ID zařízení.
SET_REQUEST	0xF9	Ano	Požadavek na uložení do nevolat. paměti.
GET_SEED	0xF8	Ano	Vrátí seed pro výpočet klíče.
UNLOCK	0xF7	Ano	Odemkne zařízení vypočteným klíčem.

SET_MTA	0xF6	Ano	Nastaví paměťový ukazatel MTA.
UPLOAD	0xF5	Ano	Vyčte paměť dané délky z lokace MTA.
SHORT_UPLOAD	0xF4	Ano	Vyčte paměť o dané délce z dané adresy.
BUILD_CHECKSUM	0xF3	Ano	Vypočítá kontrolní součet bloku paměti.
TRANSPORT_LAYER_CMD	0xF2	Ano	Pro příkazy závislé na transportní vrstvě.
USER_CMD	0xF1	Ano	Pro uživatelské příkazy.

Po resetu zařízení slave je pro zahájení komunikace vždy potřeba použít příkaz CONNECT, který zařízení přepne do stavu CONNECTED. Bez této procedury slave nereaguje na další příkazy od jednotky master. Po skončení nutné komunikace lze zařízení slave přepnout zpět do stavu DISCONNECTED příkazem DISCONNECT.

V případě mnoha příkazů pro operace s pamětí protokol XCP vyžívá tzv. adresu pro paměťový přenos (memory transfer address, MTA). Ta se obecně skládá z jedné 32bitové základní adresy a 8bitového adresového doplňku (ten slouží k adresaci v případě možnosti přepínání více stránek paměti). U většiny příkazů, které pro paměťové operace využívají právě MTA, se MTA po úspěšném provedení požadovaného příkazu inkrementuje. To usnadňuje komunikaci v případě nutnosti práce s blokem paměti větším, než je dovoleno pro jeden samostatný příkaz. Tak například stačí příkazem SET_MTA nastavit MTA na požadovanou hodnotu a poté pomocí sekvence příkazů UPLOAD postupně vyčíst požadovaný blok paměti. Odpadá tak nutnost opakovaně zadávat požadovanou zdrojovou či cílovou adresu.

1.3.2 Kalibrační příkazy

Tab. 5: *Kalibrační příkazy protokolu XCP. Vlastní zpracování dle [7].*

Příkaz	PID	Volitelný	Funkce
DOWNLOAD	0xF0	Ne	Zapíše data dané délky do lokace MTA
DOWNLOAD_NEXT	0xEF	Ano	Zapíše data v blokovém komunik. režimu.
DOWNLOAD_MAX	0xEE	Ano	Zapíše data o maximální povolené délce.
SHORT_DOWNLOAD	0xED	Ano	Zapíše data dané délky na danou adresu.
MODIFY_BITS	0xEC	Ano	Modifikuje bity maskami AND a XOR.

Kalibrační příkazy slouží k zápisu do paměti a modifikaci dat v paměti uložených. Jejich základem je příkaz DOWNLOAD. Tento příkaz jako parametry obsahuje číslo s počtem bytů dat k zápisu do paměti a tato data. Příkaz MODIFY_BITS slouží k bitovému maskování sekce délky 16 bitů. Maskuje se podle předpisu maskami AND a XOR. Každý jednotlivý bit

lze kombinací těchto masek nastavit na 1, vyresetovat na 0 či invertovat. Další kalibrační příkazy jsou pouze variacemi na příkaz DOWNLOAD. S výjimkou příkazu SHORT_DOWNLOAD všechny kalibrační příkazy využívají ukazatel MTA.

1.3.3 Příkazy pro přepínání stránek

Tab. 6: *Příkazy protokolu XCP pro přepínání stránek. Vlastní zpracování dle [7].*

Příkaz	PID	Volit.	Funkce
SET_CAL_PAGE	0xEB	Ne	Nastaví přístupový režim stránky.
GET_CAL_PAGE	0xEA	Ne	Vrátí č. aktivní strany s požad. přístupem.
GET_PAG_PROCESSOR_INFO	0xE9	Ano	Vrátí obecné informace o stránkování.
GET_SEGMENT_INFO	0xE8	Ano	Vrátí informace o specifickém segmentu.
GET_PAGE_INFO	0xE7	Ano	Vrátí informace o specifické stránce.
SET_SEGMENT_MODE	0xE6	Ano	Nastaví režim daného segmentu.
GET_SEGMENT_MODE	0xE5	Ano	Vrátí režim daného segmentu.
COPY_CAL_PAGE	0xE4	Ano	Kopíruje stránku do jiné stránky.

Protokol XCP umožňuje rozdělení paměti na sektory, segmenty a stránky. Sektor je fyzický blok paměti. Naproti tomu segment je logickým blokem paměti. Segmenty lze dále logicky rozdělit na jednotlivé stránky. Různé stránky jednoho segmentu popisují data na shodných adresách, ale s rozdílnými parametry, například s odlišnými hodnotami či rozdílným přístupem do jednotlivých stránek. Každý segment musí mít alespoň jednu stránku. Přístupový režim ke stránce lze nastavit tak, že stránku využívá buď ECU, protokol XCP, případně oba.

1.3.4 Příkazy pro akvizici a stimulaci dat

Tab. 7: *Příkazy protokolu XCP pro základní akvizici a stimulaci dat. Vlastní zpracování dle [7].*

Příkaz	PID	Vol.	Funkce
CLEAR_DAQ_LIST	0xE3	Ne	Vynuluje všechny položky ODT listu DAQ.
SET_DAQ_PTR	0xE2	Ne	Nastaví ukazatel na položku ODT.
WRITE_DAQ	0xE1	Ne	Zapíše data do položky ODT.
SET_DAQ_LIST_MODE	0xE0	Ne	Nastaví režim listu DAQ.
GET_DAQ_LIST_MODE	0xDF	Ne	Vrátí nastavený režim listu DAQ.
START_STOP_DAQ_LIST	0xDE	Ne	Zastaví, spustí či vybere list DAQ.
START_STOP_SYNCH	0xDD	Ne	Spustí či zastaví více listů najednou.
WRITE_DAQ_MULTIPLE	0xC7	Ano	Zapíše data do více položek ODT.

READ_DAQ	0xDB	Ano	Přečte data z položky ODT.
GET_DAQ_CLOCK	0xDC	Ano	Vrátí hodnotu čítače pro časové značky.
GET_DAQ_PROCESSOR_INFO	0xDA	Ano	Vrátí obecné informace o listech DAQ.
GET_DAQ_RESOLUTION_INFO	0xD9	Ano	Vrátí informace o rozlišení listů DAQ.
GET_DAQ_LIST_INFO	0xD8	Ano	Vrátí informace o specifickém listu DAQ.
GET_DAQ_EVENT_INFO	0xD7	Ano	Vrátí informace o událostním kanálu.

Tato sekce příkazů slouží k inicializaci a spouštění listů DAQ a také k získávání informací o nich. Podobně jako některé předchozí příkazy využívaly ukazatel MTA, tak příkazy pro čtení a zápis položek ODT pracují s ukazatelem na položku ODT, nastavovanou příkazem SET_DAQ_PTR. Tento ukazatel se taktéž po příslušné operaci s položkou ODT inkrementuje.

Tab. 8: *Příkazy protokolu XCP pro dynamickou alokaci listů DAQ. Vlastní zpracování dle [7].*

Příkaz	PID	Vol.	Funkce
FREE_DAQ	0xD6	Ano	Smaže všechny dynamické listy DAQ.
ALLOC_DAQ	0xD5	Ano	Alokuje listy DAQ.
ALLOC_ODT	0xD4	Ano	Alokuje tabulky ODT listu DAQ.
ALLOC_ODT_ENTRY	0xD3	Ano	Alokuje položky tabulce ODT.

Na počátku inicializační sekvence dynamických listů DAQ musí master vždy použít příkaz FREE_DAQ. Poté následuje příkaz ALLOC_DAQ pro alokaci daného počtu listů DAQ. Další v pořadí je příkaz či série příkazů ALLOC_ODT, kterým se daným listům DAQ přiřazují tabulky ODT. Nakonec následuje příkaz či série příkazů ALLOC_ODT_ENTRY pro alokaci položek k tabulkám ODT. Jiný postup než zmíněný není povolen a slave při chybné sekvenci příkazů odesílá chybovou zprávu.

1.3.5 Příkazy pro programování nevolatilní paměti

Tab. 9: *Příkazy protokolu XCP pro programování nevolatilní paměti. Vlastní zpracování dle [7].*

Příkaz	PID	Vol.	Funkce
PROGRAM_START	0xD2	Ne	Indikuje začátek sekvence programování.
PROGRAM_CLEAR	0xD1	Ne	Vymaže část nevolatilní paměti.
PROGRAM	0xD0	Ne	Zahájí programování segmentu paměti.
PROGRAM_RESET	0xCF	Ne	Indikuje konec sekvence programování.
GET_PGM_PROCESSOR_INFO	0xCE	Ano	Vrátí obecné informace o programování.
GET_SECTOR_INFO	0xCD	Ano	Vrátí informace o specifickém sektoru.
PROGRAM_PREPARE	0xCC	Ano	Příprava k programování nevolatilní paměti.

PROGRAM_FORMAT	0xCB	Ano	Nastaví formát dat před programováním.
PROGRAM_NEXT	0xCA	Ano	Pošle data k programování v blok. režimu.
PROGRAM_MAX	0xC9	Ano	Pošle data maximální délky k programování.
PROGRAM_VERIFY	0xC8	Ano	Zkontroluje obsah naprogramované paměti.

Pojmem „příkazy pro programování nevolatilní paměti“ jsou myšleny takové příkazy, s jejichž pomocí se dá programovat programová (flash) paměť. Programování paměti flash se obecně skládá ze tří částí: příprava dat, samotné programování a konečné ověření. Samotná implementace procesu programování flash paměti je silně závislá na konkrétním typu zařízení.

2 Zadání systému

V této kapitole budou uvedeny požadavky na systém s mikrokontrolérem, jehož realizace je páteří této práce. Některé z těchto požadavků byly v průběhu tvorby práce konzultovány se zadavatelem práce a upraveny či doplněny. V takovém případě je slovní doplnění uvedeno dále v textu.

2.1 Požadavky na hardware

Požadavky na mikrokontrolér samotný jsou následující:

- interface sběrnice CAN s možností přenosové rychlosti až 1 MBaud/s (tzv. high speed CAN),
- standardizované periferní sběrnice jako SPI, I^2C, UART,
- minimálně 24 digitálních výstupů, případně použití expandéru,
- minimálně 8 analogových vstupů,
- interní EEPROM, případně použití externí EEPROM,
- dlouhodobá podpora mikrokontroléru jeho výrobcem.

Požadavky na zbytek hardwaru jsou:

- napájecí napětí v rozsahu 6 až 24 voltů stejnosměrného napětí,
- provozní teplota v rozsahu −40 až +140 °C,
- galvanické oddělení digitálních výstupů – možnost spínání relé,
- příprava pro připojení teplotních senzorů.

Co se týče výběru vhodného mikrokontroléru, bylo navíc doporučeno, aby byl použitý mikrokontrolér ideálně z produkce společností STMicroelectronics nebo Freescale Semiconductors (resp. dnes již koncern NXP).

Horní hranici provozní teploty +140 °C je doporučeno držet, ale není to absolutně nutné. Zařízení musí být minimálně schopno během provozu vydržet maximální teplotu při testování v klimatické komoře +120 °C. Cokoliv nad tuto teplotu je užitečná rezerva.

Napájecí napětí by se ve skutečnosti mělo horní hranicí blížit až k 40 voltům stejnosměrného napětí, aby byla v napájecím napětí zajištěna bezpečnější rezerva.

Po konzultaci bylo také upřesněno, že galvanické oddělení výstupů pro spínání relé není vyžadováno, neboť by to mimo jiné neúměrně zvýšilo cenu systému a vyžadovalo by to i komplikované galvanické oddělení napájecí části. Pro tuto aplikaci postačí ke spínání

relé elementární spínací prvky a jednoduché ochranné prvky pro zabezpečení mikrokontroléru před možnými napěťovými špičkami, způsobenými spínáním induktivní zátěže.

Spínací prvky budou použity ke spínání relé proudem zhruba 20 mA. Spínací prvky by měly být bez problémů schopny sepnout proud o hodnotě do 100 mA.

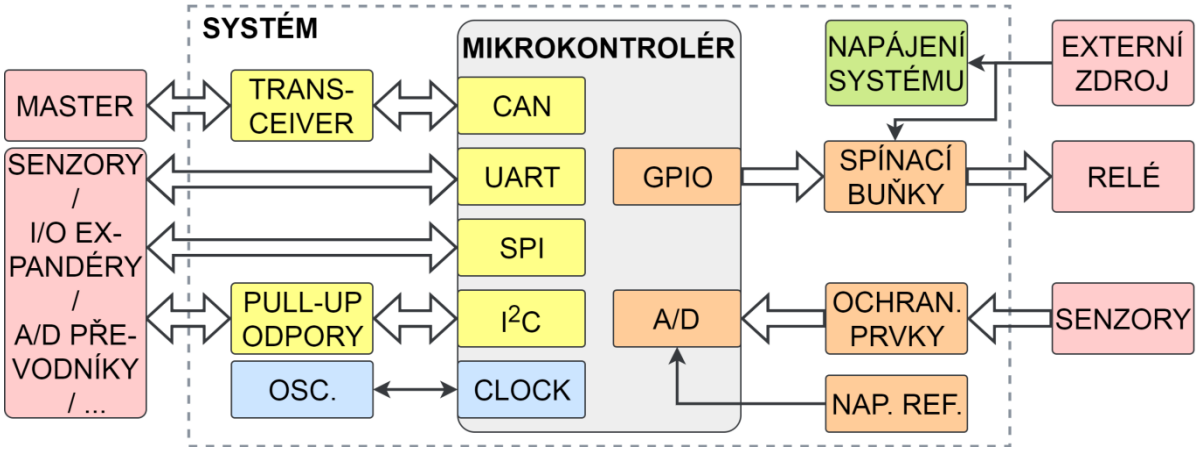
2.2 Požadavky na software

Požadavky na softwarovou aplikaci, která bude naprogramována pro zvolený mikrokontrolér, jsou následující:

- implementace zařízení slave protokolu XCP,
- přítomnost struktury, konfigurovatelné pomocí protokolu XCP a umístěné v paměti EEPROM, která bude obsahovat:
    - variabilní přenosovou rychlost sběrnice CAN,
    - informace o zařízeních, připojených k mikrokontroléru,
    - ID zařízení,
- možnost nastavování a čtení digitálních výstupů a dat z připojených senzorů.

Zařízení také musí mít protokol XCP implementovaný takovým způsobem, aby byl během komunikace jednoznačně kompatibilní se zařízením master, využívaným zadavatelem práce (software CANape).

3 Hardwarové řešení



Obr. 10: *Předpokládané blokové schéma navrhovaného systému. Vlastní tvorba.*

3.1 Prototypová deska plošných spojů

Prototypová deska plošných spojů (DPS) je v případě našeho systému určena především k následujícím účelům:

- zvolení součástek a případné ověření jejich vhodnosti,
- vytvoření základu, ze kterého po opravě možných chyb a vyladění nedostatků vznikne finální návrh,
- umožnění toho, začít co nejdříve programovat samotnou softwarovou aplikaci.

3.1.1 Mikrokontrolér

Volba vhodného mikrokontroléru byla klíčovou otázkou aplikace. Prvotní předpoklad počítal s využitím 32bitového mikrokontroléru řady STM32 od společnosti STMicroelectronics. Tato platforma mikrokontrolérů je v dnešní době velmi oblíbená, často nasazovaná v praxi, univerzální a aplikaci by poskytla prakticky neomezený dostatek prostředků. Kvůli požadavku na značný teplotní rozsah muselo být využití řady STM32 ovšem zamítnuto, neboť tyto mikrokontroléry končí s horní provozní teplotou na hodnotě 125 °C. Navíc u řady STM32 nelze najít konkrétní mikrokontrolér, který by na čipu měl integrovaný kontrolér sběrnice CAN a zároveň paměť EEPROM.

Pozornost se proto zaměřila na ekonomičtější platformu STM8 téhož výrobce. Tato řada 8bitových mikrokontrolérů se dokonce jeví jako příhodnější pro naši konkrétní aplikaci, která neklade na výpočetní výkon mikrokontroléru nijak velké nároky. Produktová řada mikrokontrolérů STM8AF, určených pro automotivní aplikace, u některých čipů zaručuje

spolehlivou funkčnost až do teploty 150 °C. Verze STM8AF52 obsahuje kontrolér sběrnice CAN i integrovanou paměť EEPROM. Po nalezení čipu v pouzdře s nejvhodnějším počtem vývodů byl pro naši aplikaci zvolen mikrokontrolér STM8AF52A9 ve variantě TDY [8].

Tab. 10: *Vybrané vlastnosti mikrokontroléru STM8AF52A9TDY. Vlastní zpracování dle [8].*

STM8AF52A9TDY	
Počet vývodů pouzdra	64
Maximální frekvence jádra	24 MHz
Paměť FLASH	128 Kbyte
Paměť RAM	6 Kbyte
Paměť EEPROM	2 Kbyte
Interface sběrnice CAN	High-speed 1 MBit/s CAN 2.0B
Další komunikační rozhraní	USART, LIN, SPI, I^2C
Čítače	3x16bitový, 1x8bitový
A/D převodník	10bitový, 16 kanálů
Napájecí napětí	3 až 5,5 V
Provozní teplotní rozsah	−40 až +150 °C

3.1.2 Napájecí obvod

Vzhledem k rozsahu možného napájecího napětí zvoleného mikrokontroléru přicházely v úvahu napájecí soustavy 3,3 V či 5 V. Zvoleno bylo napájecí napětí 5 V z důvodu větší robustnosti výsledného systému. Konverzi napětí ze vstupního rozsahu 6 až 24 V zajišťuje měnič typu step-down ve formě integrovaného obvodu A6985F výrobce STMicroelectronics. Ten je určen primárně pro napájení elektronických automotivních systémů z napájecích soustav s napětím 12 a 24 voltů. Konkrétně byla použita varianta tohoto obvodu s označením A6985F5V, disponující fixním výstupním napětím 5 V, která ušetří nutnost přiřadit na výstup navíc zpětnovazební napěťový dělič a pravděpodobně nepatrně zlepší i přesnost výstupního napětí. Tento integrovaný obvod je ze svého výstupu schopen dodávat až 500 mA proudu, což se pro naši aplikaci jeví jako více než postačující. Obvod A6985F opět dokáže pracovat v širokém rozmezí teplot od −65 do +150 °C. [9]

Dále tento obvod poskytuje možnost nastavení spínací frekvence od 250 kHz do 2 MHz, dále výstup pro resetování připojeného zařízení při náběhu napájení a nastavitelnou prodlevu tohoto výstupu. A6985F umožňuje funkci ve dvou různých režimech. Režim nízkého šumu slouží pro zachování co nejkvalitnější regulace v celém rozsahu odebíraného proudu, zatímco

režim nízkého odběru zajišťuje vyšší účinnost měniče při nižším odběru proudu, ale zhoršuje kvalitu regulace. [9]

Napájecí obvod mimo součástek k těmto účelům obsahuje pro funkci měniče nutnou vstupní kapacitu (10 µF a 1 µF), výstupní kapacitu (10 µF) a výstupní induktor k vyhlazování proudu (22 µH). Návrh celku byl výrazně ulehčen použitím internetového simulačního nástroje eDesignSuite, poskytovaného výrobcem měniče. Součástí napájení je i vstupní EMC filtr, který by měl sloužit k utlumení možného nežádoucího rušení generovaného zpět do napájecí sítě spínáním měniče.

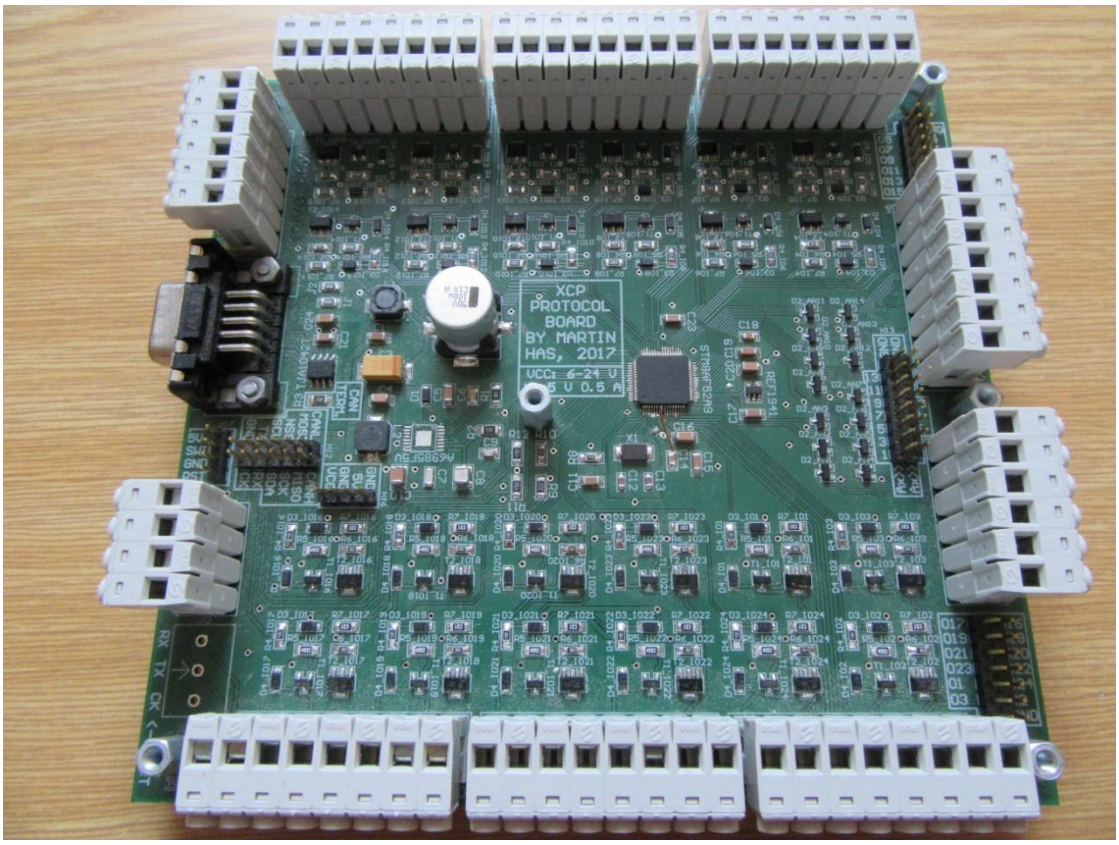
3.1.3 Ostatní obvody

Kromě mikrokontroléru samotného a napájecího obvodu tvoří zbytek systému:

- Krystalový oscilátor pro přesné časování mikrokontroléru s rezonanční frekvencí 16 MHz a širokým provozním teplotním rozsahem. K němu je připojena dvojice zatěžovacích kondenzátorů o kapacitě 12 pF.
- Integrovaný obvod REF1941 jako zdroj referenčního napětí 4,096 V pro integrovaný A/D převodník. Obvod zároveň společně slouží i jako napájení této analogové části mikrokontroléru (referenční napětí REF1941 lze zatížit odebíraným proudem až do hodnoty 20 mA). Maximální provozní teplota tohoto obvodu činí 150 °C, stabilita jeho referenčního napětí je zaručena do teploty 125 °C.
- Transceiver pro sběrnici CAN typu TJA1042. Ten slouží jako mezivrstva, která propojuje ovladač sběrnice CAN v mikrokontroléru (vytváří digitální signály RX a TX) a samotnou fyzickou vrstvou sběrnice CAN, tvořenou diferenčními signály CANL a CANH. Sběrnici CAN lze na straně desky zakončit terminátorem.
- Dvacet čtyři spínačů, z nichž každý je tvořen dvojící bipolárních tranzistorů. Tyto spínače jsou určeny pro spínání připojených reléových kontaktů. Tranzistory jsou zapojeny jako tzv. „high side“ spínače. Tedy připojený reléový kontakt je jednou stranou spojen s nulovým potenciálem a druhou stranou je k němu připínáno a odepínáno napájecí napětí. Samozřejmostí jsou ochranné diody pro vybíjení energie z připojených indukčností. Pro případ proražení spínacích tranzistorů jsou výstupy mikrokontroléru dodatečně chráněny diodami.
- Čtrnáct analogových vstupů do mikrokontroléru je dodatečně chráněno diodovými napěťovými omezovači.
- Všechny integrované obvody obsahují dostatečný počet blokovacích kondenzátorů.

3.1.4 Deska plošných spojů

Deska plošných spojů byla navržena v návrhovém systému Altium Designer. Deska plošných spojů má velikost zhruba 140 x 150 mm, na svou relativně jednoduchou funkci je tedy značně velká. Její velkou část zabírají buňky s tranzistorovými spínači. Po okolí desky to jsou pak poměrně robustní svorkovnice, které zajišťují, že veškeré vývody z mikrokontroléru mohou být fixně spojeny se zbytkem světa. Tyto konektory jsou ještě doplněny klasickými „pinheadery“, které umožňují realizovat „měkčí“ spojení se světem a ulehčují diagnostiku signálů. Diferenční signály CANL a CANH sběrnice CAN jsou vyvedeny na tzv. „D-Sub“ konektor v rozmístění doporučeném právě pro sběrnici CAN.



Obr. 11: *Fotografie osazené prototypové DPS. Vlastní tvorba.*

3.1.5 Zhodnocení prototypové DPS

Na prototypové desce bohužel byly udělány dvě chyby. První chybou bylo prohození napájecích kontaktů u mikrokontroléru (signály značené VDD a VSS). Tato chyba byla relativně jednoduše opravitelná fyzickou změnou cest na desce. Druhou a značnější chybou byl špatně udělaný „footprint“ čipu napájení (v dokumentaci je tento čip znázorněn ze spodu, zatímco jeho footprint byl dělán seshora). Díky tomu bylo napájení nefunkční a během programování musela být deska napájena přímo 5 volty na mikrokontrolér.

Naproti těmto dvěma nepříjemnostem byl ovšem zbytek desky funkční a umožňoval bezproblémové programování samotné aplikace. I přesto zbýval prostor pro vylepšení některých věcí, která byla promítnuta do pozdější finální desky plošných spojů.

3.2 Finální deska plošných spojů

Finální deska plošných spojů vychází z návrhu prototypové desky. Obecné rozložení funkčních celků na desce zůstává nezměněné. Samozřejmě jsou opraveny dvě chyby přítomné na prototypové desce.

Největší změnou prošel obvod napájení. Rozsah napájecího napětí byl rozšířen na 5–38 V (38 V je horní hranicí napětí, které napájecí čip dokáže zpracovat) a tomu musel být celý napájecí obvod předimenzován. Dále byl čip A6985F5V nahrazen obecnějším a lépe dostupnějším A6985F bez fixního výstupního napětí. Před EMC filtr byl předřazen varistor k omezení možných přepěťových událostí v napájecím napětí.

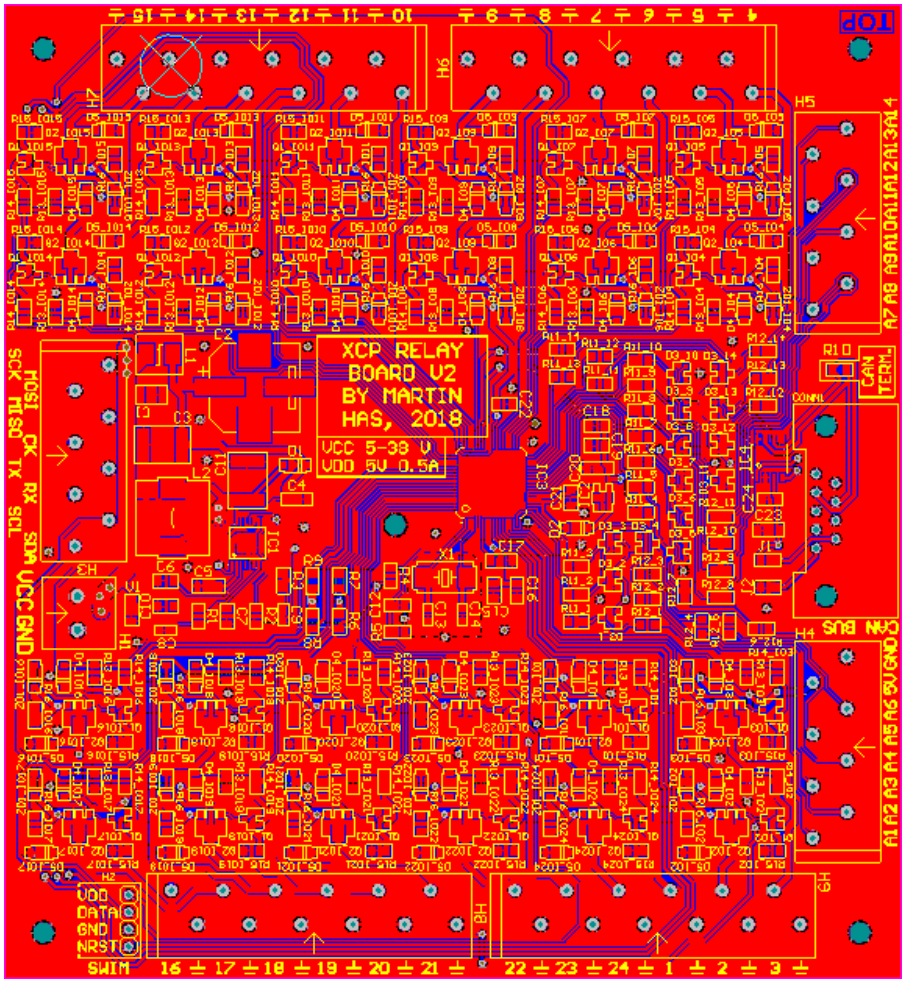
Velkou změnou prošly buňky pro spínání reléových kontaktů. Zapojení s dvojicí bipolárních tranzistorů nahradilo zapojení s dvojicí tranzistorů MOSFET (resp. jedním tranzistorem typu NMOS a jedním typu PMOS). Vzhledem k tomu, že hradlo tranzistoru PMOS je spínáno oproti napájecímu napětí, které může dosahovat až hodnoty 38 V, bylo nutné přidat Zenerovu diodu, která rozdíl napětí mezi hradlem tranzistoru a elektrodou source s přivedeným napájecím napětím omezí na hodnotu maximálně 10 V.

Použitý typ tranzistoru PMOS je teoreticky schopen sepnout proud až do hodnoty 3 A. Při udávaném odporu kanálu v sepnutém stavu, který činí 200 mΩ, by při takovém zatížení ovšem vykazoval ztrátový výkon 1,8 W, což je nad jeho tepelné možnosti. Vzhledem k tomu, že se na desce nachází 24 těchto tranzistorových buněk a přívodní napájecí konektor by měl být zatížitelný proudem do 16 A (přívodní cesty na DPS k těmto buňkám by měly být schopny vést více proudu než samotný konektor), lze po odečtení max. 0,7 A na napájení obvodů a připojených senzorů získat hodnotu zhruba 0,64 A sepnutelného proudu při použití každé buňky. Tato hodnota je tak o mnohonásobně vyšší, než požadovaných 100 mA a dává dostatečnou rezervu pro širší možnosti využití.

Obvody ochrany analogových vstupů mikrokontroléru prošly pouze mírnou úpravou, kde byla k napěťovým omezovačům se dvojicí Schottkyho diod přidána dvojice rezistorů. Jeden z nich omezuje průtok proudu (při přepětí či záporném napětí na analogovém vstupu) přidaným napěťovým omezovačem a druhý průtok proudu vnitřními omezovacími diodami

v mikrokontroléru. Tak by měly být analogové vstupy jednoduše a dostatečně chráněny v případě, že je na ně přivedeno napětí, které je mimo stanovené meze.

Rozměr finální desky bylo možné mírně zmenšit na hodnotu zhruba 132 x 143 mm. Toho bylo dosaženo zmenšením pouzder mnoha diskrétních součástek, jejich větším nahuštěním k sobě a užitím odlišných konektorů s menší roztečí vývodů. Nadbytečné pinheadery byly vypuštěny, zbyl pouze jeden k programování mikrokontroléru přes protokol SWIM. Celkem deska obsahuje 315 jednotlivých součástek.



Obr. 12: *Finální DPS, viděná v návrhovém systému Altium Designer. Vlastní tvorba.*

Nakonec je třeba zmínit, že se jako příhodný teplotní senzor pro naši aplikaci jeví senzor LMT87. Tento analogový senzor je vyráběn společností Texas Instruments a dokáže měřit v rozmezí teplot −50 až +150 °C s přesností ±0,4 °C. Jeho výstupní napětí se pohybuje od 3277 mV (−50 °C) do 538 mV (+150°C), je tedy ideální pro naše referenční napětí 4,096 V. Jeho převodní charakteristika vykazuje mírnou nelinearitu, jejíž odchylku lze snadno minimalizovat linearizací této charakteristiky po částech. LMT87 se navíc vyrábí v příhodném THT pouzdře TO-92.

4 Softwarové řešení

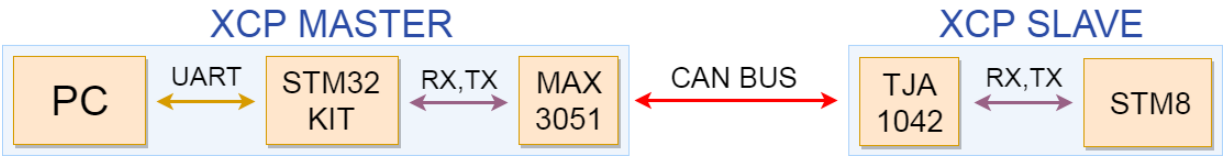
4.1 Vývojové nástroje

4.1.1 Pomocné nástroje

Před programováním samotné aplikace zařízení slave protokolu XCP bylo nutné vyřešit způsob, jak tuto aplikaci spojit s počítačem, který bude simulovat jednotku master. Zadavatelem práce bylo sice umožněno vypůjčit si kombinovaný analyzátor FlexRay/CAN Vector VN7600, toto zařízení však bylo ve firmě dosti vytíženo a i vzhledem k jeho vysoké pořizovací ceně bylo vhodné najít lepší způsob propojení.

Jednou ze zvažovaných možností byla emulace protokolu XCP na sběrnici CAN za pomoci sériového přenosu přes rozhraní UART rovnou do počítače. To by znamenalo nutnost v mikrokontroléru průběžně bufferovat příchozí data až do obdržení celé zprávy. Tento způsob by sice byl relativně jednoduchý k naprogramování, ale stejně bychom se nevyhnuli nutnosti dříve či později muset podporu sběrnice CAN implementovat.

Proto byla nakonec zvolena metoda komunikace přes sběrnici CAN mezi naším zařízením slave a vývojovým kitem NUCLEO–F446RE s mikrokontrolérem STM32F446RE. Tento kit je následně ovládán pomocí sériového přenosu z počítače (resp. emulací sériového přenosu pomocí USB). Kit tedy slouží jako jakýsi interface mezi sběrnicí CAN a sériovým přenosem přes rozhraní UART čili jako primitivní „analyzátor“ sběrnice CAN. Ke kitu NUCLEO–F446RE bylo pouze nutné připojit transceiver sběrnice CAN typu MAX3051 a zakončovací terminátor sběrnice.



Obr. 13: *Blokové schéma vývojového řetězce. Vlastní tvorba.*

Toto řešení sice umožňuje komunikaci mezi PC jako jednotkou master a mikrokontrolérem STM8 jako zařízením slave, ale přesto s sebou nese určitá úskalí při časování mezi rozhraním UART a sběrnicí CAN. Kit NUCLEO–F446RE „tiskne“ přijaté i odeslané zprávy sběrnice CAN do počítače sériovým přenosem ve formátu, který je patrný z následující sekvence zpráv:

```
1236,1,7F0,5,000000CE,003FA000,48506\n

1237,1,7F0,3,00000000,00820801,751\n
```

```
1238,1,7F0,7,004B004B,1F00A702,1114\n
```

Čárky ve zprávě tvoří oddělovače mezi jednotlivými elementy zprávy. Prvek „\n“ je znak začátku nové řádky, který značí ukončení stávající zprávy a počátek zprávy nové. Jednotlivé elementy zprávy, řazeny postupně tak, jak jdou ve zprávě za sebou, jsou následující:

- číslo zprávy,
- číslice značící, zda byla zpráva přijata (1), nebo odeslána (0),
- standardní identifikátor zprávy (hexadecimální číslo),
- počet bytů zprávy,
- spodní čtveřice bytů zprávy (hexadecimální číslo),
- horní čtveřice bytů zprávy (hexadecimální číslo),
- čas mezi jednotlivými zprávami v milisekundách.

Tab. 11: *Orientační srovnání doby přenosu zprávy mezi CAN a UART. Vlastní zpracování.*

		BEST CASE			WORST CASE		
	Baudrate [bit/s]	Bytů dat	Bitů zprávy	Čas [ms]	Bytů dat	Bitů zprávy	Čas [ms]
UART	115200	30	300	2,6	38	380	3,3
CAN	20000	8	111	5,55	2	63	3,15

Předchozí tabulka orientačně znázorňuje, jak dlouho může zprávě trvat přenos po sběrnici CAN nejnižší možnou podporovanou přenosovou rychlostí našeho systému a jak dlouho bude trvat její následný přenos rozhraním UART do počítače nejvyšší standardní přenosovou rychlostí. Z údajů vyplývá, že například při nepřetržitém příjmu zpráv není zaručeno, že budou všechny mít dostatek času k „vytištění“ do počítače – záleží na počtu bytů jejich dat.

Tento problém je naštěstí irelevantní během klasické výměny příkazů a odezev v protokolu XCP. Ta ve standardním přenosovém režimu XCP probíhá principem dotaz – odpověď. Zařízení master tedy vždy čeká na odezvu jednotky slave před odesláním požadavku nového.

Bohužel však tento problém nemusí být úplně zanedbatelný při posílání zpráv se synchronními daty. Tato data se obvykle posílají najednou „v dávce“ několika zpráv. Před implementací bufferu příchozích zpráv v kitu opravdu docházelo k tomu, že několik zpráv v dávce nemuselo být správně zobrazeno. Právě to vedlo k nutnosti začít příchozí zprávy ve vývojovém kitu bufferovat. Tím se odbourala nutnost užívat pro přenos po sběrnici

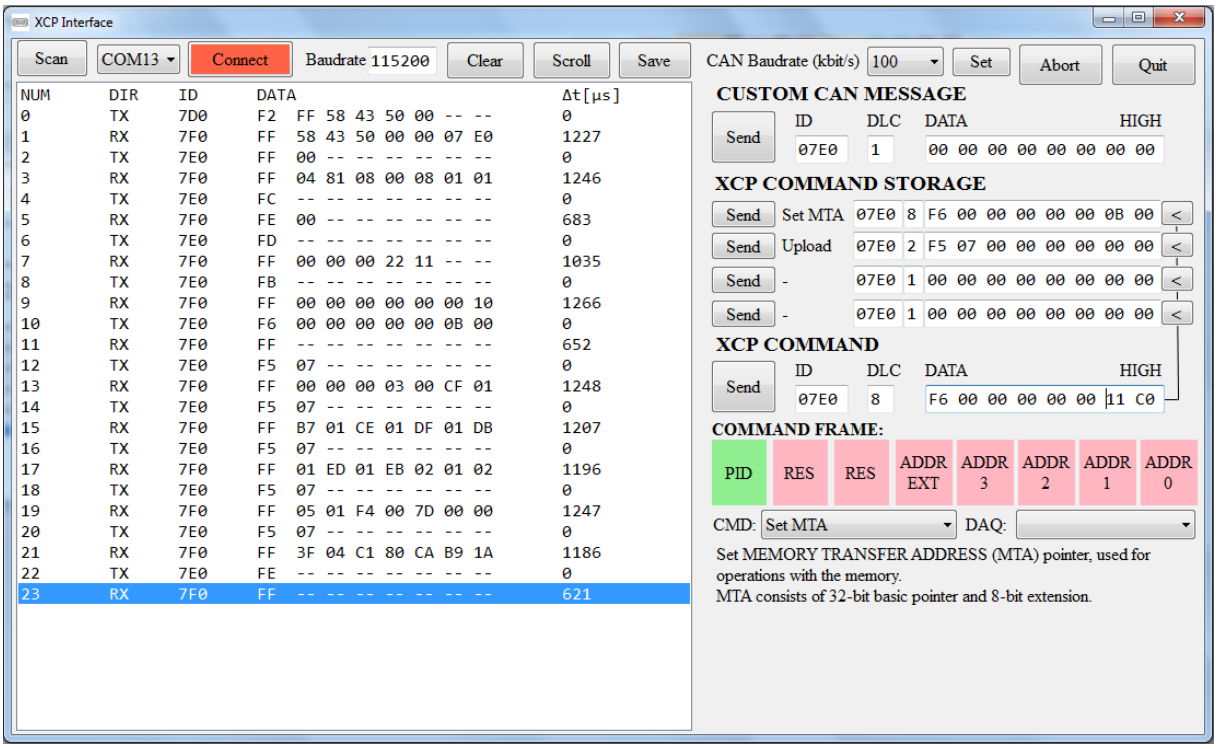
CAN nejnižší dovolenou přenosovou rychlost. Při odesílání synchronních dat je i tak ovšem nutné mít dostatečně velkou klidovou rezervu do příští spouštěcí události, aby všechny nahromaděné zprávy ve vývojovém kitu měly dostatek času k postupnému odeslání rozhraním UART. O tomto způsobu komunikace mezi sběrnicí CAN a rozhraním UART lze tedy obecně říci, že dokáže velmi efektivně fungovat, uživatel je ovšem nucen dodržovat jisté zásady.

O zobrazení zpráv v počítači a jejich odesílání se stará aplikace, vytvořená v prostředí Windows Forms v programu Microsoft Visual Studio. Nástroje prostředí Windows Forms umožňují jednoduchý přístup k virtuálnímu sériovému portu. Základem aplikace je zpracování dat, přijímaných po sériovém portu. Pokud aplikace obdrží znak nového řádku, zpracuje dosavadní obdržený řádek tak, že provede jeho rozdělení na jednotlivá pole podle čárek, které slouží jako oddělovače. Posléze tato jednotlivá pole interpretuje a vytiskne do prvku typu ListBox, použitého k zobrazení dat na obrazovku.

Druhá klíčová funkcionalita této aplikace je možnost zadat vývojovému kitu s STM32 požadavek na odeslání zprávy po sběrnici CAN. Uživatel si může samozřejmě u požadované zprávy nastavit ID, počet datových bytů a požadovaná data. Po kliknutí na tlačítko Send se zpráva po sériovém portu odešle do kitu. Nejdříve se posílá specifický uvozovací znak, po jehož přijetí mikrokontrolér na kitu očekává přesný počet zbývajících znaků zprávy. Po přijetí posledního z nich tuto zprávu mikrokontrolér nechává odeslat po sběrnici CAN.

Další funkcionalitou aplikace, která navíc výrazně zjednodušuje práci s protokolem XCP, je předpřipravený seznam příkazů, které byly implementovány v jednotce slave. Uživateli je kromě textového popisu zvoleného příkazu zobrazen i rámec příkazu s popisem jednotlivých bytů. Dále jsou automaticky předvyplněny kolonky pro ID, počet datových bytů, byte s identifikátorem PID a eventuálně zbytek parametrů příkazu na výchozí hodnotu. Uživatel má také možnost si svůj modifikovaný příkaz uložit do „skladiště“ na jednu ze čtyř položek pro rychlé použití.

Z okna aplikace je také možné měnit přenosovou rychlost, kterou komunikuje vývojový kit po sběrnici CAN. Opět se tak děje odesláním specifického znaku, po kterém následují data s novou přenosovou rychlostí. Lze také užít tlačítka Abort, které opět vysílá specifický znak, po jehož přijetí kit zruší odesílání zprávy po sběrnici CAN. Aplikace také umožňuje uložení výpisu zpráv z prvku ListBox do textového souboru.

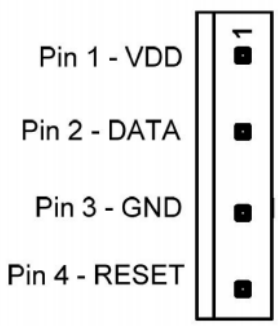


Obr. 14: *Ukázka aplikace pro komunikaci se zařízením XCP slave. Vlastní tvorba.*

4.1.2 Vývojové nástroje mikrokontroléru STM8

Vzhledem k tomu, že jako mikrokontrolér pro implementaci zařízení slave protokolu XCP byl zvolen čip platformy STM8, bylo nutné vybrat k tomuto mikrokontroléru správné vývojové nástroje.

Jedním z praktických prvků, které odlišují mikrokontroléry řady STM32 a STM8, je rozdílný interface pro jejich programování. Mikrokontroléry STM32 se obvykle programují pomocí tzv. rozhraní „JTAG/serial wire debugging“, zkráceně SWD, které se připojuje klasickým plochým kabelem pro JTAG s 20 piny. Naproti tomu mikrokontroléry řady STM8 pro programování využívají tzv. „single wire interface module“, zkráceně SWIM. Každý čip této řady má na křemíku zabudovaný modul SWIM pro komunikaci pomocí protokolu SWIM a samostatný modul pro debuggování. [10] Programování a debuggování mikrokontroléru řady STM8 pomocí SWIM využívá 4 pinový konektor. Pin 1 je určený k měření napájecího napětí, pin 2 je datový vodič SWIM, pin 3 zemní vodič a pin 4 slouží k resetování zařízení.

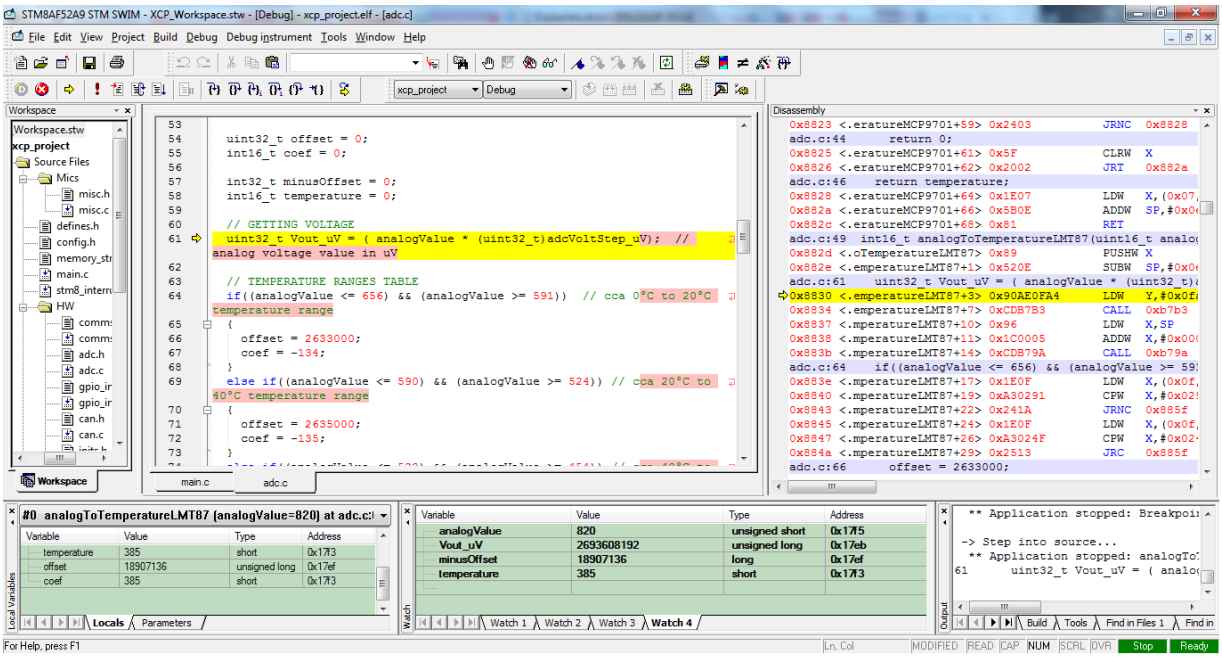


Obr. 15: *Konektor protokolu SWIM. Převzato z [11].*

Původní předpoklad byl, že jako programátor mikrokontroléru bude použit vestavěný programátor ST-LINK/V2-1, užívaný u populárních vývojových kitů Discovery a Nucleo (minimálně u těch, které obsahují čipy STM32). Bohužel se ukázalo, že zrovna tato verze programátoru podporou protokolu SWIM nedisponuje. Bylo tak nutné zvolit „stand-alone" programátor ST-LINK, který podporuje zároveň platformy STM8 i STM32.

Z dostupných vývojových prostředí se jako efektivní varianta jevilo použití prostředí od STMicroelectronics jménem ST Visual Develop. ST Visual Develop je stáhnutelné a plně využívatelné bez jakýchkoliv poplatků. Nevýhodou tohoto prostředí je jeho značná archaičnost. K editaci samotného programu je proto lepší využívat vhodnější software. Výhodou pro začátečníka je jeho relativní jednoduchost a přehlednost. Největší síla ST Visual Develop tkví ve velice dobrém debuggeru. Ten i na dnešní poměry obsahuje všechno důležité: breakpointy, krokování programu, okna nahlížení na proměnné, rozpis příkazů jazyka C do instrukcí v assembleru, prohlížení paměti a hardwarových registrů atd.

Co naopak prostředí ST Visual Develop neobsahuje v základu v sobě, je kompilátor jazyka C či assembleru do strojového kódu programovaného zařízení. Kompilátor pro STM8 lze vybírat minimálně ze čtyř možností. Z tohoto počtu jsou dva kompilátory dostupné zdarma. Jako nejlepší řešení se jevil kompilátor CXSTM8 firmy Cosmic, neboť je právě prostředím ST Visual Develop bezproblémově podporován a doporučen k použití v kombinaci s tímto prostředím je i společností STMicroelectronics. Integrace CXSTM8 do ST Visual Develop je díky tomu velice jednoduchá. Jedinou podmínkou bezplatného užívání kompilátoru CXSTM8 je nutnost jeho registrace.



Obr. 16: *Debuggování programu v prostředí ST Visual Develop. Vlastní tvorba.*

Užitečným nástrojem, obsaženým v balíku spolu s prostředím ST Visual Develop, je program ST Visual Programmer. Ten slouží k samostatnému programování a prohlížení nevolatilních pamětí cílového zařízení. Ačkoliv o nahrávání vytvořeného softwaru do programové paměti typu flash v mikrokontroléru se samozřejmě stará samo vývojové prostředí ST Visual Develop, nástroj ST Visual Programmer se přesto hodí k prohlížení a úpravu dat v paměti EEPROM, popř. k modifikaci tzv. „option bytů“ (těmi lze upravovat pokročilá nastavení mikrokontroléru).

4.2 Struktura aplikace

Samotná aplikace s implementací zařízení slave protokolu XCP pro mikrokontrolér STM8AF52A9 je napsána v programovacím jazyce C takovým způsobem, aby byla přeložitelná použitým kompilátorem CXSTM8. Aplikace je rozdělena do jednotlivých modulů, které sestávají ze zdrojových a/nebo hlavičkových souborů. Moduly lze rozdělit do několika logických funkčních celků. Tyto funkční celky jsou následující:

- mandatorní části aplikace,
- části aplikace pro práci s hardwarem mikrokontroléru,
- části aplikace s implementací protokolu XCP,
- ostatní části aplikace.

Mandatorní části aplikace obsahují tyto moduly:

- main.c – Zdrojový soubor, který obsahuje hlavní funkci main. Ta je v programu spouštěna jako první. Funkce main obsahuje volání inicializačních funkcí a nekonečný cyklus s hlavním rozhodovacím blokem.
- config.h – Hlavičkový soubor, který obsahuje definice různých parametrů pro konfiguraci zařízení. Jsou v něm také definovány struktury na pevně daných místech v paměti zařízení.
- defines.h – Tento soubor obsahuje především definice bitů některých proměnných.
- memory_structures.h – Obsahuje většinu struktur, užívaných v programu, které jsou definované jako vlastní datové typy.
- stm8_interrupt_vectors.c – V tomto souboru se nachází tabulka vektorů přerušení mikrokontroléru. Samotné funkce s obsluhami těchto přerušení se pak vždy nacházejí ve zdrojových souborech, odpovídajících dané funkci tohoto přerušení.

Další částí aplikace jsou zdrojové a hlavičkové soubory, které slouží především k přímé práci s hardwarem mikrokontroléru. Mezi ně patří následující moduly:

- adc.c, adc.h – Funkce pro práci s analogově-digitálním převodníkem mikrokotroléru. Obsažena je navíc tabulka jednotlivých kanálů multiplexoru převodníku, řazených dle chronologického pořadí převodu.
- can.c, can.h – Soubory s funkcemi, určenými výhradně pro obsluhu kontroléru sběrnice CAN, tj. inicializace kontroléru CAN a obsluha příjmu a odesílání zpráv.
- GPIO_init.c, GPIO_init.h – Tyto soubory obsahují obecné funkce pro inicializaci stavu vstupně-výstupních buněk digitálních portů a tabulky s pozicemi digitálních

portů pro různé funkční bloky. Dále také obsahují specifické funkce pro a import a export stavů portů pro reléové výstupy z a do registrů v paměti.

- inits.c, inits.h – Funkce pro nastavení a zabezpečení hodinového systému, nastavení různých priorit vektorů přerušení a počáteční inicializaci digitálních a analogových vstupů a výstupů.
- timers.c, timers.h – Obsahuje funkce pro inicializaci a obsluhu přerušení čítačů.
- memory.c, memory.h – Rozličné funkce pro zápis do různých typů paměti mikrokontroléru a ostatní funkce, určené pro práci s pamětí.
- uart.c, uart.h – Tyto funkce slouží pro inicializaci a práci s komunikačním rozhraním UART.
- spi.c, spi.h – Funkce, pro inicializace a práci s komunikačním rozhraním SPI.
- i2c.c, i2c.h – Inicializace a práce s komunikačním rozhraním I^2C.
- comms.c, comms.h – Tyto soubory zastřešují práci s předchozími třemi typy komunikačních rozhraní. Obsaženy v nich jsou navíc funkce pro volání jednotlivých inicializací komunikačních rozhraní.

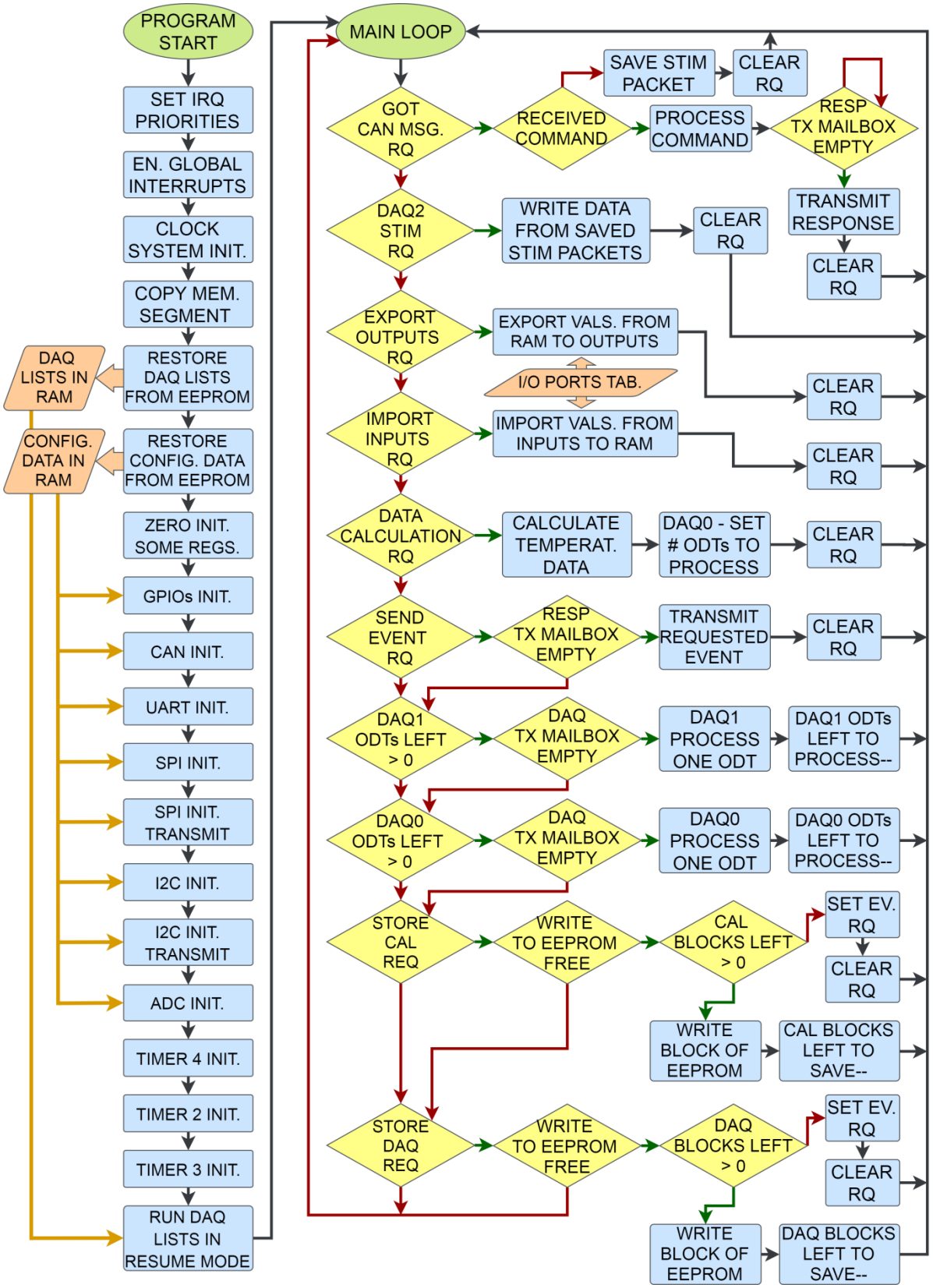
Následující soubory slouží k samotné práci s protokolem XCP:

- XCP_config.h – Hlavičkový soubor s definicemi pro konfiguraci protokolu XCP.
- XCP.c, XCP.h – Tyto soubory obsahují hlavní funkci protokolu XCP, a to sice funkci s identifikací přijatého paketu XCP a voláním odpovídajícího příkazu či uložením paketu pro stimulaci.
- XCP_cmds.c, XCP_cmds.h – V těchto souborech jsou obsaženy funkce se všemi implementovanými příkazy ze skupin standardních a kalibračních příkazů.
- XCP_DAQ.c, XCP_DAQ.h – Tyto soubory obsahují funkce s implementovanými příkazy pro akvizici dat a funkce s různými obsluhami pro akvizici dat.
- XCP_responses.c, XCP_responses.h – V těchto souborech lze nalézt funkce pro odesílání odezev protokolu XCP. Navíc jsou zde obsaženy funkce, které váží protokol XCP na přenosové prostředí (v našem případě sběrnici CAN).

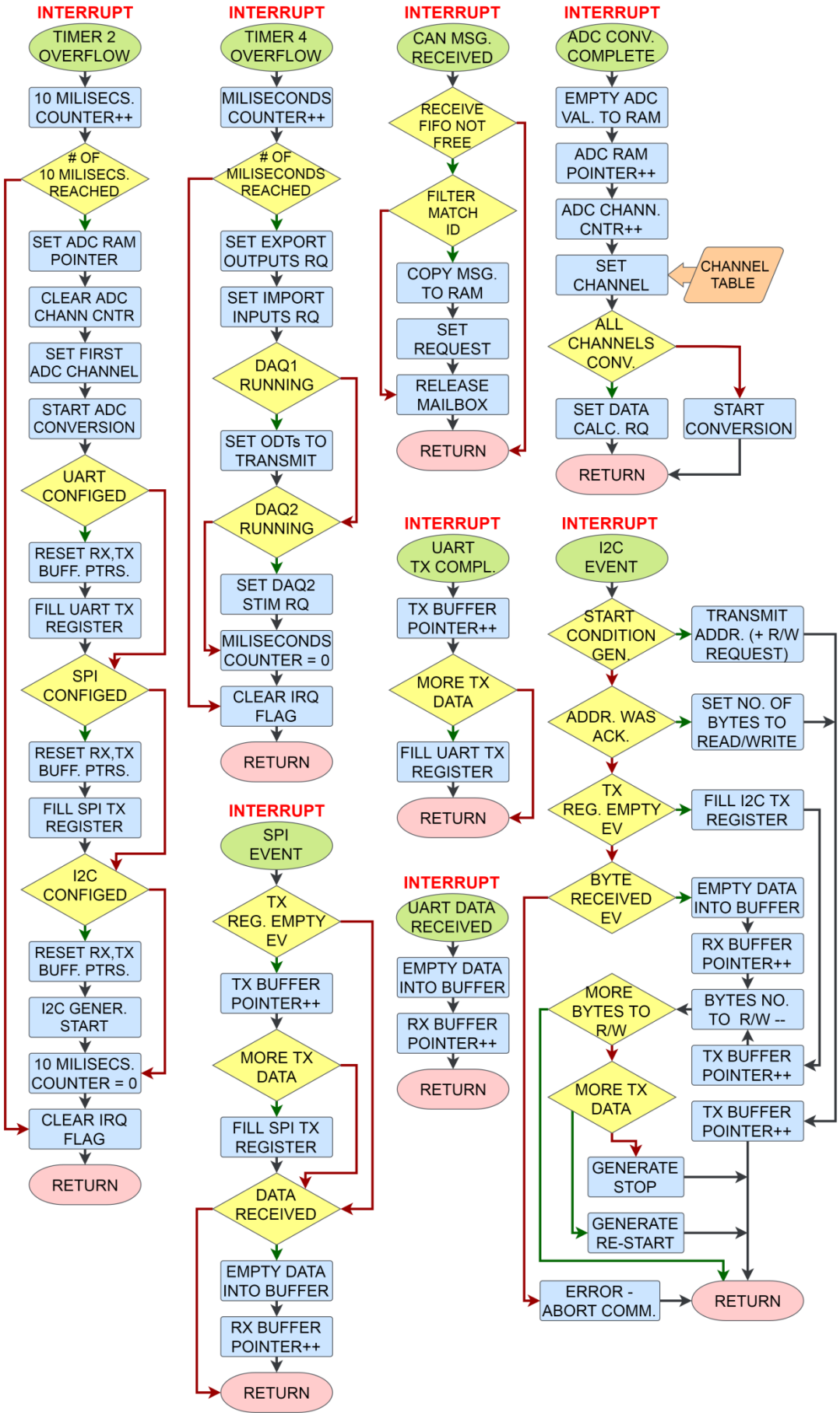
Poslední část tvoří moduly, které nelze jednoznačně zařadit mezi předchozí skupiny:

- sensors.c, sensors.h – Tyto soubory obsahují funkce, které slouží k přepočtu číselných hodnot z teplotních senzorů na analogových vstupech na skutečnou teplotu.
- misc.c, misc.h – Zde jsou obsaženy ostatní pomocné funkce, které nebylo možné logicky přiřadit do ostatních souborů.

4.3 Vývojové diagramy aplikace



Obr. 17: *Vývojový diagram synchronní části aplikace. Vlastní tvorba.*



Obr. 18: *Vývojový diagram asynchronní části aplikace. Vlastní tvorba.*

4.4 Popis programu

4.4.1 Inicializační část aplikace

Po spuštění programu jako první samozřejmě probíhá řada inicializačních procedur. Jako první se spouští procedura, která upravuje rozdílné stupně priorit vektorů přerušení. Následně je v řadiči přerušení globálně povoleno spouštění obsluh přerušení.

První nutnou inicializací je inicializace hodinového systému mikrokontroléru. Hodiny mikrokontroléru jsou po náběhu napájení generovány z vnitřního RC oscilátoru s rezonanční frekvencí 16 MHz s předděličkou frekvence nastavenou tak, aby byl generován pulz o frekvenci 2 MHz. Zdroj hodin je přepnut z vnitřního RC oscilátoru na externí krystalový oscilátor, který dokáže zajistit mnohem větší stabilitu frekvence generovaného signálu. Rezonanční frekvence tohoto oscilátoru je 16 MHz a po nastavení dělícího poměru předděličky rovného 1 je tato frekvence i frekvencí hodin mikrokontroléru. Dále je povolena obsluha přerušení tzv. bezpečnostního systému hodin. Ta je volána v případě detekce selhání externího oscilátoru. V takovém případě se automaticky přepíná na interní RC oscilátor jako zdroj hodin s dělícím poměrem rovným 8 a obsluha přerušení tento dělící poměr pouze nastaví roven 1 (tj. vrací zpět frekvenci 16 MHz).

Následuje kopírování dat z nevolatilní paměti do paměti RAM. Z programové paměti je kopírován segment paměti, který obsahuje funkci, jež při běhu musí být vykonávána z paměti RAM (zápis bloku paměti EEPROM). Dále je obsah paměti EEPROM kopírován do paměti RAM. Tato data sestávají z konfiguračních dat zařízení o velikosti 1024 bytů a prostoru uložených listů DAQ o velikosti 1024 bytů. Z důvodu rychlejšího přístupu k těmto datům a především možnosti jejich okamžité změny je k těmto datům přistupováno právě v paměti RAM. Zpět do nevolatilní paměti EEPROM jsou tato data v aplikaci ukládána až při takovém požadavku od uživatele.

Dalším krokem jsou inicializace vstupních a výstupních portů mikrokontroléru. „Reléové výstupy“ mikrokontroléru jsou inicializovány buď jako vstupy, či výstupy. Digitální buňky na analogových vstupech jsou inicializovány jako vstupy. Stejně jako u následujících konfigurací se tak děje na základě konfiguračních dat, uložených do paměti EEPROM z předchozích relací.

Kontrolér sběrnice CAN je inicializován na požadovanou přenosovou rychlost. Filtry sběrnice CAN jsou nastaveny tak, aby propustily pouze zprávy s identifikátorem ve standardním formátu (11 bitů), které souhlasí s identifikátorem zařízení master

či identifikátorem pro broadcast – tyto identifikátory jsou opět nakonfigurovány z předchozích relací. Je povolena obsluha přerušení při příjmu zprávy po sběrnici CAN.

Následují konfigurace standardních komunikačních rozhraní. Je-li jejich funkcionalita požadována, inicializují se ovladače komunikace pro rozhraní UART, SPI a I^2C do zvolených režimů s požadovanými přenosovými rychlostmi. Nachází-li se nějaká data v inicializačních vysílacích bufferech rozhraní I^2C a SPI, je zahájeno jednorázové vysílání těchto dat.

Jako další prvek je inicializován analogově-digitální převodník. Tomu je nastaven vstupní hodinový signál o frekvenci 2 MHz. Jedna konverze v takovém případě trvá zhruba 7 µs. Vzhledem k tomu, že převodník v tomto typu mikrokontroléru nepodporuje tvz. „scan mode“, tedy převod sekvence požadovaných kanálů „v jedné dávce“, musí být postupný převod všech požadovaných kanálů řízen „ručně“. V programu je nadefinována tabulka kanálů převodníku tak, jak uživatel chce, aby byly postupně převáděny. Jedna proměnná je určena k čítání kanálů v tabulce a jedna k adresaci bufferu pro uložení konverzí. Zarovnání výstupních dat je nastavené tak, aby bylo umožněno jejich jednoduché vyčtení jako datový typ uint16. Je provedena inicializační konverze libovolného kanálu, která by dle dokumentace měla dopadnout chybně. Po jejím dokončení je povolena obsluha přerušení při dokončení konverze.

Další částí úvodních inicializací je nastavení čítačů v mikrokontroléru. Ze čtyř dostupných čítačů aplikace využívá jeden 8bitový a dva 16bitové čítače. Pro odměřování kratších časových úseků je používán 8bitový čítač TIM4. Tomu je nastavena perioda čítání 1 ms. K odměřování delších časových úseků aplikaci slouží čítač TIM2. Ten má základní časovou periodu 10 ms. Oba tyto čítače mají povolené obsluhy přerušení při přetečení a mají specifické proměnné, které budou počítat jejich přetečení. Posledním využitým čítačem v aplikaci je čítač TIM3. Ten se od zbylých dvou čítačů liší tím, že je volnoběžný. Nemá tedy obsluhu přerušení, do jeho chodu není nijak zasahováno a při přetečení sám počítá od začátku. Perioda čítání tohoto čítače je rovna zhruba 524,28 ms. Čítač TIM3 slouží aplikaci pro odběr časových známek pro pakety DTO.

Nakonec je proveden proces zjišťování, zdali je možné aplikaci spustit v pokračovacím režimu. Program prohledá všechny listy DAQ, které byly obnoveny z paměti EEPROM, a pokud je u některého z listů nastaven požadavek na spuštění v pokračovacím režimu, pak je tento list spuštěn. Je-li takto spuštěn alespoň jeden list, aplikace vstupuje do pokračovacího režimu a tato událost je indikována odesláním odpovídajícího paketu EV.

4.4.2 Asynchronní část aplikace

Asynchronní část aplikace je ta část, jejímž úkolem je okamžitě reagovat na vzniklé události. Tato část aplikace využívá obsluh vzniklých přerušení.

Z hlediska protokolu XCP je „nejdůležitější“ obsluha přerušení ta obsluha, která je volána v případě přijetí zprávy po sběrnici CAN. Vzhledem k nastavení filtrů pro příjem zpráv hardware mikrokontroléru vyvolává přerušení pouze v případě, že byla přijata zpráva od zařízení master či zpráva s identifikátorem pro broadcast. Ostatní nežádoucí zprávy jsou vyfiltrovány. Odpovídá-li pak číslo filtru pro příjem zpráv číslu požadovanému, je zpráva překopírována do paměti RAM. Následně je nastaven požadavek na její rozklíčování v synchronní části aplikace. Paměť s přijatou zprávou v hardwaru pro sběrnici CAN je potom uvolněna.

Pokud dojde k přetečení čítače TIM4 či TIM2, v odpovídající obsluze přerušení je inkrementována ta proměnná, která čítá počet načítaných přetečení tohoto čítače. Když dosáhne jedna z těchto proměnných požadované hodnoty, je tato proměnná vynulována a jsou provedeny specifické akce, spjaté s touto proměnnou. Tyto hodnoty čítání, které jsou tedy ekvivalentní časovým intervalům, jsou uživatelsky upravitelné a lze je uložit do paměti EEPROM. Tak je zaručena uživatelská variabilita dvojice procesů uvnitř mikrokontroléru.

Hodnotu čítání u TIM4 lze označit jako interval obnovy výstupů a jejich nastavování či vyčítání uživatelem pomocí DAQ. Prováděné akce v tomto případě sestávají z:

- nastavení požadavku na import a export stavů „reléových výstupů“,
- nastavení počtu ODT listu DAQ1 k odeslání (je-li tento list DAQ spuštěn),
- nastavení požadavku na provedení stimulace dat listu DAQ2.

V případě TIM2 slouží tento interval obecně k zahájení měření a komunikace. V takovém případě jsou prováděny tyto akce:

- nastavení prvního kanálu k převodu u analogově-digitálního převodníku, inicializace ukazatele bufferu pro výsledky převodu a spuštění převodu,
- požadavek na zahájení komunikace přes standardní komunikační rozhraní, tj. inicializace ukazatelů vysílacích a přijímacích bufferů a naplnění vysílacích hardwarových registrů daty (v případě SPI a UART) či vygenerování podmínky START u I^2C (ovšem jen v případě, že jsou řadiče rozhraní konfigurovány a jsou dostupná nějaká data k vysílání).

Pokud analogově-digitální převodník dokončí jeden převod, je vygenerováno přerušení, jehož obsluha výsledek převodu překopíruje z hardwarového registru převodníku do bufferu pro výsledky převodu a inkrementuje ukazatel na tento buffer. Pokud ještě nebyly převedeny všechny kanály z tabulky kanálů k převodu, následuje nastavení dalšího kanálu a spuštění převodu. V opačném případě, kdy už je převod všech požadovaných kanálů dokončen, je pouze nastaven požadavek na výpočet dat z výsledků převodů.

Poslední součástí asynchronní části aplikace jsou přerušení od standardních komunikačních rozhraní. V případě rozhraní UART dostupné vzniklé události sestávají ze dvou přerušení. První z nich je přerušení v případě příchozích dat, zapsaných do přijímacího hardwarového registru. V takovém případě je v obsluze přerušení obsah tohoto registru vyčten do přijímacího bufferu v paměti RAM. Posléze je ukazatel na tento buffer inkrementován. Pokud se stane, že je dat na příjmu příliš mnoho a ukazatel vyjede mimo oblast bufferu, je opět nastaven na začátek. Druhým typem přerušení u rozhraní UART je přerušení při vyprázdnění vysílacího hardwarového registru. V takovém případě je inkrementován ukazatel na vysílací buffer. Dále se kontroluje, byla-li již data z vysílacího bufferu odeslána všechna. Pokud tomu tak není, je vysílací registr naplněn dalšími daty.

V případě přerušení pro rozhraní SPI existuje pouze jeden druh přerušení a tím je obecná událost rozhraní SPI. V její obsluze se posléze musí otestovat, zda nastala událost příjmu dat či událost prázdného vysílacího registru. Při následném řešení těchto událostí se postupuje analogicky jako při řešení dvou rozdílných obsluh přerušení u rozhraní UART.

Vzhledem ke komplexnosti sběrnice I^2C, která je vyšší, než komplexnost rozhraní SPI a UART, je pro rozhraní I^2C dostupné sice jenom jedno obecné přerušení, to ovšem může obsahovat následující události a jejich obsluhu:

- Vygenerování podmínky START – V tomto případě se z vysílacího bufferu odesílá adresa cílového zařízení a požadavek na čtení či zápis. Ukazatel na vysílací buffer je inkrementován.
- Potvrzení (ACK) odeslané adresy – Nastavuje se počet bytů, které budou vyslány či přijaty (jejich počet je uložen jako hodnota za adresou ve vysílacím bufferu). Bude-li přijat pouze jeden byte dat, ta nebudou potvrzena a je generována podmínka START či STOP. Ukazatel na vysílací buffer je opět inkrementován. Podle směru toku dat může nastat pouze jeden z následujících dvou typů událostí.
- Data byla přijata a uložena v registru – Data jsou vyčtena do přijímacího bufferu, jeho ukazatel je inkrementován a zkontroluje se jeho pozice. Počet zbývajících dat k přijetí

je dekrementován. Budou-li následující data posledními, nebudou potvrzena a je generována podmínka START (v případě dalších dat k vysílání) či STOP.

- Vysílací registr je prázdný – Zbývají-li další data k odeslání, je naplněn vysílací registr (byty dat k odeslání se nacházejí ve vysílacím bufferu za hodnotou s počtem bytů k odeslání). Ukazatel na vysílací buffer se inkrementuje a počet zbývajících dat se dekrementuje. Byla-li již odeslána všechna data, generuje se START či STOP.

4.4.3 Synchronní část aplikace

Synchronní část aplikace sestává z nekonečného cyklu, ve kterém jsou testovány podmínky, které mohou být nastavovány kdekoliv v aplikaci. Podmínky jsou testovány postupně takovým způsobem, že pokud byla nějaká podmínka splněna a odpovídající akce byla vykonána, vrací se celý cyklus na začátek – lze tedy prohlásit, že podmínky jsou řazeny sestupně dle jejich priorit. Po vykonání akce je obvykle proměnná, která rozhoduje v podmínce, vyresetována či dekrementována.

První a z hlediska aplikace „nejvýznamnější“ podmínkou je požadavek na rozklíčování zprávy ze sběrnice CAN, která byla uložena v paměti RAM. Nejdříve je dle PID paketu rozhodnuto, jedná-li se o příkazový paket či paket s daty pro stimulaci. Obsahuje-li paket příkaz, je dle svého PID identifikován v tabulce příkazů a odtud je volána odpovídající funkce. V případě, že požadovaný příkaz není implementován, odesílá se chybová hláška ERR_CMD_UNKNOWN. Každá funkce s jednotlivým příkazem má obvykle takovou strukturu, že nejprve probíhá kontrola toho, zda se nacházíme ve stavu CONNECTED. Následuje kontrola počtu datových bytů paketu a zjištění, zdali se parametry příkazu nacházejí v požadovaných mezích. Potom je proveden příkaz samotný a vygenerována odpověď. Na pakety typu RES, EV a SERV je aplikací využívána jedna odesílací schránka kontroléru sběrnice CAN z celkových tří. Není-li tato schránka volná, pokud chce aplikace odeslat odpověď na příkaz, vyčká se na její uvolnění.

Byl-li po sběrnici CAN přijat paket s daty pro stimulaci, aplikace podle čísla PID (využívá se identifikace paketů dle absolutního čísla ODT) projde dostupné listy DAQ, u každého zjistí, zda je list ve směru stimulace a zda je aktivní. V případě, že jsou předchozí podmínky splněny, aplikace dále spočítá rozsah jeho PID a podle něho rozhodne, zda přijatý paket patří tomuto listu DAQ. Pokud ano, tak nakonec zjistí adresu úložiště v paměti pro pakety tohoto listu a posléze paket uloží na odpovídající pozici.

Dalším prvkem rozhodovacího cyklu je testování, zdali byla splněna podmínka pro požadavek na událost stimulace listu DAQ2. V případě, že je tato podmínka splněna, aplikace postupně bere z úložiště paketů tohoto listu DAQ paket za paketem a data z nich zapisuje na požadovaná místa v paměti. Pokud aplikace narazí na prázdnou položku ODT, přeskočí zbytek celé tabulky ODT. Pokud aplikace narazí na celou prázdnou tabulku ODT, přeskočí celý zbytek listu DAQ. Může tak být významně ušetřen čas v případě ne zcela naplněného listu DAQ, neboť u protokolu XCP se obecně předpokládá, že zbytek položek je prázdný.

Následující dvě podmínky jsou sice testovány zvlášť, ale principiálně jsou shodné. Jsou jimi požadavky na export výstupních a import vstupních hodnot z „reléových výstupů“. V paměti RAM je definována sekvence vstupních a výstupních „stínových“ registrů, ze kterých jsou data přenášena na výstupy a opačně. Uživatel tak nemusí znát přesnou pozici každé buňky, ale pro přístup k ní může využít sadu těchto registrů. Vzhledem k tomu, že data jsou přenášena na všechny výstupní buňky a importována ze všech vstupních buněk zároveň, musí mít ovšem uživatel přehled o tom, zda byla odpovídající buňka inicializována jako vstupní či jako výstupní, aby mohl rozhodnout, zda má smysl zapisovat do výstupního či číst data ze vstupního stínového registru.

Další testovanou podmínkou je požadavek na výpočet dat z hodnot získaných digitalizací z analogových vstupů. V případě splnění této podmínky se každá hodnota z bufferu s digitalizovanými hodnotami přepočítá přes požadovanou funkci a uloží do bufferu s přepočítanými hodnotami. Požadovanou přepočtovou funkci pro každý analogový vstup udává odpovídající číslo v konfiguračních datech. U analogových vstupů se jako výchozí zařízení předpokládá použití senzorů pro měření teploty. Naše přepočtová funkce je obecně definována jako funkce, která jako vstupní parametr přijímá proměnnou typu uint16 a výstupním parametrem je proměnná typu int16. Pokud uživatel nadefinuje novou přepočtovou funkci, musí její konfigurační číslo nadefinovat do tabulky ve funkci pro volání odpovídající přepočtové funkce. Po přepočtení hodnot je nastaven celkový počet ODT v listu DAQ0, které zbývá odeslat (tj. spouštěcí událost listu DAQ0) za předpokladu, že je tento list aktivní.

Podmínka pro žádost o odeslání paketu s hlášením události je testována spolu s ověřením, zda je požadovaná vysílací schránka kontroléru sběrnice CAN prázdná. To je z důvodu, aby aplikace nemusela čekat na uvolnění schránky (na rozdíl třeba od odpovědi po vykonání

příkazu, kde aplikace může vyčkat, neboť příkaz má v aplikaci maximální prioritu). V případě volné schránky je odpovídající typ ohlášení události připraven k odeslání.

Následujícími částmi v cyklu je obsluha odesílání datových paketů listů DAQ1 a DAQ0. Místo toho, aby bylo odeslání paketů jednoho listu řešeno najednou, ve spouštěcí události odpovídajícího listu je vždy pouze nastaven celkový počet paketů, které zbývají k odeslání. V podmínce je poté testováno, zda je toto číslo větší než nula. Poté je paket požadovaného čísla naplněn daty podle konkrétní tabulky ODT. I zde platí dříve zmíněné zjednodušení – pokud je detekována prázdná položka ODT, je přeskočen zbytek jedné ODT a pokud je detekována celá prázdná tabulka ODT, je přeskočen zbytek listu DAQ (tj. počet zbývajících paketů k odeslání je vynulován). Po připravení odeslání jednoho paketu s daty je počet zbývajících paketů k odeslání dekrementován.

Část aplikace pro odesílání datových paketů užívá dvě zbývající odesílací schránky kontroléru sběrnice CAN. Spolu s testováním kladného zbývajícího počtu paketů se ověřuje, zdali je alespoň jedna z těchto dvou odesílacích schránek prázdná. Bez této podmínky by byla aplikace nucena čekat při pokusu odeslat datový paket. V kombinaci s testováním počtu zbývajících paketů je tak zaručeno, že i během obecného procesu odesílání datových paketů může být tato činnost kdykoliv přerušena například v případě příjmu příkazu od zařízení master, a po vykonání tohoto příkazu může původní činnost opět pokračovat.

Posledními z testovaných podmínek je testování na požadavek na uložení do paměti typu EEPROM. Požadované uložení může být dvojího typu – požadavek na uložení kalibračních dat (to jsou v případě naší aplikace veškerá konfigurační data) a požadavek na uložení listů DAQ. Požadavky se zadávají pomocí příkazu XCP s názvem SET_REQUEST, resp. jeho parametrů STORE_CAL_REQ a STORE_DAQ_REQ. V případě zadání požadavku na uložení listů DAQ je pouze u těch listů, jež byly předtím „zvoleny“ pomocí příkazu START_STOP_DAQ_LIST, nastaven parametr pro spuštění v pokračovacím režimu. Kromě samotného požadavku na provedení akce se u obou typů požadavků nastaví počet bloků EEPROM, které zbývají k uložení. V samotném cyklu se pak kromě samotného požadavku na uložení do paměti EEPROM testuje, zda je zápis do paměti EEPROM volný a zda je počet zbývajících bloků k uložení kladný. Až poté je započat zápis jednoho bloku paměti a dekrementován zbývající počet bloků k zápisu. Pokud je při testování podmínek zjištěn nastavený požadavek na uložení, volný přístup do paměti EEPROM, ale nulový zbývající počet bloků, značí to, že proces ukládání byl dokončen. V takovém případě je nastaven požadavek na vygenerování odpovídajícího hlášení události EV.

4.5 Seznam implementovaných příkazů XCP

V aplikaci byly z pěti celkových skupin příkazů XCP implementovány celkem tři skupiny, a to skupiny standardních příkazů, kalibračních příkazů a příkazů pro akvizici dat. Žádný z příkazů pro přepínání stránek ani z příkazů pro programování zařízení nebyl v naší aplikaci zapotřebí, proto byly tyto dvě skupiny úplně vynechány.

Ze sekce standardních příkazů byly vynechány pouze dva příkazy, související s odemykáním zabezpečeného zařízení a dále příkaz pro výpočet kontrolního součtu sekce paměti. Výpis implementovaných standardních příkazů pak vypadá takto:

- CONNECT
- DISCONNECT
- GET_STATUS
- SYNCH
- GET_COMM_MODE_INFO
- GET_ID
- SET_REQUEST
- SET_MTA
- UPLOAD
- SHORT_UPLOAD
- TRANSPORT_LAYER_CMD
- USER_CMD

Z kalibračních příkazů byly vynechány příkazy, které jsou pouze variací na základní příkaz DOWNLOAD a pro naši aplikaci by neměly prakticky žádný přínos navíc. Implementovány tak byly pouze následující dva příkazy:

- DOWNLOAD
- MODIFY_BITS

V aplikaci byly implementovány prakticky všechny příkazy, které jsou využívány při práci se statickými listy DAQ. Vynechán byl pouze jeden příkaz pro získání detailních informací o událostech uvnitř jednotky slave. Zbytek příkazů, které v aplikaci implementovány jsou, je následující:

- CLEAR_DAQ_LIST
- SET_DAQ_PTR
- WRITE_DAQ

- SET_DAQ_LIST_MODE
- GET_DAQ_LIST_MODE
- START_STOP_DAQ_LIST
- START_STOP_SYNCH
- READ_DAQ
- GET_DAQ_CLOCK
- GET_DAQ_PROCESSOR_INFO
- GET_DAQ_RESOLUTION_INFO
- GET_DAQ_LIST_INFO

4.6 Pevné paměťové struktury

Vzhledem k tomu, že protokol XCP pracuje s paměťovými objekty na známých adresách, jsou v aplikaci důležitá data umístěná na pevně daných adresách v paměti. Definici jejich struktur lze nalézt v souboru memory_structures.h a jejich pevné umístění v souboru config.h. Zde budou uvedeny z důvodu vyšší přehlednosti a jednoznačnosti. Nutné je také zmínit, že použitý mikrokontrolér pracuje s vícebytovými datovými typy (např. datové typy uint16 či uint32) ve formátu společnosti Motorola (tzv. „big-endian“).

4.6.1 Konfigurační data

Konfigurační data slouží k úvodním inicializacím či k získávání informací o zařízení, ale také jsou používána při běhu programu. Všechna konfigurační data jsou po startu zařízení obnovována z nevolatilní paměti EEPROM a uživatelským požadavkem tam mohou být zpět uložena. Některá konfigurační data se při jejich změně mohou projevit ihned. K provedení nových počátečních inicializací je ovšem po uložení dat do nevolatilní paměti vyžadován restart zařízení. Konfigurační data zde budou rozdělena do tří oddělených funkčních bloků.

Tab. 12: *Konfigurační data zařízení a sběrnice CAN. Vlastní zpracování.*

Funkce struktury	Adresa (hex.)	Datový typ	Počet	Popis parametru
Konfigurace zařízení	C00	uint8	1	Perioda obnovy výstupů v milisekundách.
	C01	uint8	1	Perioda měření a vysílání v desítkách milisek.
	C06	char	32	ID zařízení.
Konfigurace sběrnice CAN	C26	uint32	1	Přenosová rychlost sběrnice CAN.
	C2A	uint16	1	Standardní CAN ID zařízení master (11 bitů).
	C2C	uint16	1	Standardní CAN ID zařízení slave (11 bitů).
	C2E	uint16	1	Standardní CAN ID pro broadcast (11 bitů).

Tab. 13: *Podporované hodnoty přenosových rychlostí sběrnice CAN. Vlastní zpracování.*

Podporované hodnoty parametru „Přenosová rychlost sběrnice CAN“ [Baud/s]							
20000	50000	83333	100000	125000	250000	500000	1000000

Vzhledem k tomu, že aplikace z důvodu vyšší možnosti flexibility dovoluje upravovat dva časové parametry (perioda obnovy výstupů a perioda měření a vysílání), musí uživatel volit tyto parametry vhodně takovým způsobem, aby aplikace mohla ve stanovených časových intervalech provést všechny úkony, spjaté s těmito časovými intervaly.

Tab. 14: *Konfigurační data „reléových výstupů“ a analogových vstupů. Vlastní zpracování.*

Název struktury	Adresa (hex.)	Datový typ	Počet	Popis parametru
Konfigurace „reléových výstupů“	C34	uint8	1	Inicializační stav „reléového výstupu“ 1.
	C35	char	16	Informace o připojeném zařízení.
	C45	uint8	1	Inicializační stav „reléového výstupu“ 2.
	C46	char	16	Informace o připojeném zařízení.
	C56	uint8	1	Inicializační stav „reléového výstupu“ 3.
	C57	char	16	Informace o připojeném zařízení.
	C67	uint8	1	Inicializační stav „reléového výstupu“ 4.
	C68	char	16	Informace o připojeném zařízení.
	C78	uint8	1	Inicializační stav „reléového výstupu“ 5.
	C79	char	16	Informace o připojeném zařízení.
	C89	uint8	1	Inicializační stav „reléového výstupu“ 6.
	C8A	char	16	Informace o připojeném zařízení.
	C9A	uint8	1	Inicializační stav „reléového výstupu“ 7.
	C9B	char	16	Informace o připojeném zařízení.
	CAB	uint8	1	Inicializační stav „reléového výstupu“ 8.
	CAC	char	16	Informace o připojeném zařízení.
	CBC	uint8	1	Inicializační stav „reléového výstupu“ 9.
	CBD	char	16	Informace o připojeném zařízení.
	CCD	uint8	1	Inicializační stav „reléového výstupu“ 10.
	CCE	char	16	Informace o připojeném zařízení.
	CDE	uint8	1	Inicializační stav „reléového výstupu“ 11.
	CDF	char	16	Informace o připojeném zařízení.
	CEF	uint8	1	Inicializační stav „reléového výstupu“ 12.
	CF0	char	16	Informace o připojeném zařízení.
	D00	uint8	1	Inicializační stav „reléového výstupu“ 13.
	D01	char	16	Informace o připojeném zařízení.
	D11	uint8	1	Inicializační stav „reléového výstupu“ 14.
	D12	char	16	Informace o připojeném zařízení.
	D22	uint8	1	Inicializační stav „reléového výstupu“ 15.
	D23	char	16	Informace o připojeném zařízení.

	D33	uint8	1	Inicializační stav „reléového výstupu“ 16.
	D34	char	16	Informace o připojeném zařízení.
	D44	uint8	1	Inicializační stav „reléového výstupu“ 17.
	D45	char	16	Informace o připojeném zařízení.
	D55	uint8	1	Inicializační stav „reléového výstupu“ 18.
	D56	char	16	Informace o připojeném zařízení.
	D66	uint8	1	Inicializační stav „reléového výstupu“ 19.
	D67	char	16	Informace o připojeném zařízení.
	D77	uint8	1	Inicializační stav „reléového výstupu“ 20.
	D78	char	16	Informace o připojeném zařízení.
	D88	uint8	1	Inicializační stav „reléového výstupu“ 21.
	D89	char	16	Informace o připojeném zařízení.
	D99	uint8	1	Inicializační stav „reléového výstupu“ 22.
	D9A	char	16	Informace o připojeném zařízení.
	DAA	uint8	1	Inicializační stav „reléového výstupu“ 23.
	DAB	char	16	Informace o připojeném zařízení.
	DBB	uint8	1	Inicializační stav „reléového výstupu“ 24.
	DBC	char	16	Informace o připojeném zařízení.
Konfigurace analogových vstupů	DCC	uint8	1	Inicializační stav analogového vstupu 1.
	DCD	uint8	1	Přepočtová funkce analogového vstupu 1.
	DCE	char	16	Informace o připojeném zařízení.
	DDE	uint8	1	Inicializační stav analogového vstupu 2.
	DDF	uint8	1	Přepočtová funkce analogového vstupu 2.
	DE0	char	16	Informace o připojeném zařízení.
	DF0	uint8	1	Inicializační stav analogového vstupu 3.
	DF1	uint8	1	Přepočtová funkce analogového vstupu 3.
	DF2	char	16	Informace o připojeném zařízení.
	E02	uint8	1	Inicializační stav analogového vstupu 4.
	E03	uint8	1	Přepočtová funkce analogového vstupu 4.
	E04	char	16	Informace o připojeném zařízení.
	E14	uint8	1	Inicializační stav analogového vstupu 5.
	E15	uint8	1	Přepočtová funkce analogového vstupu 5.
	E16	char	16	Informace o připojeném zařízení.

	E26	uint8	1	Inicializační stav analogového vstupu 6.
	E27	uint8	1	Přepočtová funkce analogového vstupu 6.
	E28	char	16	Informace o připojeném zařízení.
	E38	uint8	1	Inicializační stav analogového vstupu 7.
	E39	uint8	1	Přepočtová funkce analogového vstupu 7.
	E3A	char	16	Informace o připojeném zařízení.
	E4A	uint8	1	Inicializační stav analogového vstupu 8.
	E4B	uint8	1	Přepočtová funkce analogového vstupu 8.
	E4C	char	16	Informace o připojeném zařízení.
	E5C	uint8	1	Inicializační stav analogového vstupu 9.
	E5D	uint8	1	Přepočtová funkce analogového vstupu 9.
	E5E	char	16	Informace o připojeném zařízení.
	E6E	uint8	1	Inicializační stav analogového vstupu 10.
	E6F	uint8	1	Přepočtová funkce analogového vstupu 10.
	E70	char	16	Informace o připojeném zařízení.
	E80	uint8	1	Inicializační stav analogového vstupu 11.
	E81	uint8	1	Přepočtová funkce analogového vstupu 11.
	E82	char	16	Informace o připojeném zařízení.
	E92	uint8	1	Inicializační stav analogového vstupu 12.
	E93	uint8	1	Přepočtová funkce analogového vstupu 12.
	E94	char	16	Informace o připojeném zařízení.
	EA4	uint8	1	Inicializační stav analogového vstupu 13.
	EA5	uint8	1	Přepočtová funkce analogového vstupu 13.
	EA6	char	16	Informace o připojeném zařízení.
	EB6	uint8	1	Inicializační stav analogového vstupu 14.
	EB7	uint8	1	Přepočtová funkce analogového vstupu 14.
	EB8	char	16	Informace o připojeném zařízení.

Tab. 15: *Podporované hodnoty konfig. dat „reléových výstupů“ a analog. vstupů. Vlastní zpracování.*

Stav „reléového výstupu“		Stav analogového vstupu		Přepočtová funkce	
0	Odpojené zařízení	0	Odpojené zařízení	0	Nepřepočítávat
1	Vstupní inicializace	1	Analogový vstup	1–255	Číslo funkce
2	Výstupní inicializace	2–255	Neznámý stav		
3–255	Neznámý stav				

Tab. 16: *Konfigurační data standardních komunikačních rozhraní. Vlastní zpracování.*

Název struktury	Adresa (hex.)	Datový typ	Počet	Popis parametru
Konfigurace I^2C	EC8	uint8	1	Inicializační stav kontroléru I^2C.
	EC9	uint32	1	Přenosová rychlost I^2C.
	ECD	char	16	Informace o připojeném zařízení.
Vysílací buffer I^2C	EDD	uint8	1	Počet platných bytů cyklického bufferu I^2C.
	EDE	uint8	80	Cyklický vysílací buffer I^2C.
Inicializační buffer I^2C	F2E	uint8	1	Počet platných bytů inicializač. bufferu I^2C.
	F2F	uint8	80	Inicializační vysílací buffer I^2C.
Konfigurace UART	F7F	uint8	1	Inicializační stav kontroléru UART.
	F80	uint32	1	Přenosová rychlost UART.
	F84	char	16	Informace o připojeném zařízení.
Vysílací buffer UART	F94	uint8	1	Počet platných bytů vysílacího bufferu UART.
	F95	uint8	20	Cyklický vysílací buffer UART.
Konfigurace SPI	FA9	uint8	1	Inicializační stav kontroléru SPI.
	FAA	uint32	1	Přenosová rychlost SPI.
	FAE	char	16	Informace o připojeném zařízení.
Vysílací buffer SPI	FBE	uint8	1	Počet platných bytů cyklického bufferu SPI.
	FBF	uint8	30	Cyklický vysílací buffer SPI.
Inicializační buffer SPI	FDD	uint8	1	Počet platných bytů inicializač. bufferu SPI.
	FDE	uint8	30	Inicializační vysílací buffer SPI.

Tab. 17: *Podporované přenosové rychlosti komunikačních rozhraní. Vlastní zpracování.*

Rozhraní	Podporované přenosové rychlosti [Baud/s]				
I^2C	400000	370000	350000	320000	300000
	270000	250000	220000	200000	170000
	150000	100000	50000	30000	20000
SPI	8000000	4000000	2000000	1000000	
	500000	250000	125000	62500	
UART	244–1000000				

Vysílací buffery pro rozhraní SPI a UART pracují na jednoduchém principu – všechna data, nacházející se v bufferu, jsou postupně vysílána počínaje událostí začátku měření

a vysílání. U sběrnice I^2C je situace složitější – následující tabulka ilustruje způsob, jakým je třeba zapsat komunikační data do vysílacího bufferu.

Tab. 18: *Ukázka sekvence ve vysílacím bufferu rozhraní I^2C . Vlastní zpracování.*

Data	Popis
`0x90`	Adresa příjemce + požadavek na zápis dat
`0x03`	Počet bytů k zapsání
`0xAC`	Data k zápisu 1
`0x1B`	Data k zápisu 2
`0x80`	Data k zápisu 3
`0x91`	Adresa příjemce + požadavek na čtení dat
`0x04`	Počet bytů ke čtení

Rozhraní SPI a I^2C navíc obsahují inicializační buffery, jejichž data jsou vysílána pouze po startu programu. Každý vysílací buffer obsahuje příslušné číslo s počtem platných bytů bufferu, které musí souhlasit s počtem datových bytů požadovaných uživatelem k vysílání.

4.6.2 Získávaná a zapisovaná data

Tab. 19: *Získávaná a zapisovaná data. Vlastní zpracování.*

Název struktury	Adresa (hex.)	Datový typ	Počet	Popis parametru
Data z A/D převodníku	`B00`	uint16	14	„Syrové“ hodnoty z kanálů A/D převodníku.
	`B24`	int16	14	Přepočtené hodnoty z kanálů A/D převodníku.
Stínové registry	`B48`	uint8	3	Stínový registr vstupních hodnot.
	`B4B`	uint8	3	Stínový registr výstupních hodnot.
Přijímací buffer UART	`B4E`	uint8	1	Počítadlo přijatých hodnot v bufferu UART.
	`B4F`	uint8	40	Přijímací buffer rozhraní UART.
Přijímací buffer SPI	`B77`	uint8	1	Počítadlo přijatých hodnot v bufferu SPI.
	`B78`	uint8	40	Přijímací buffer rozhraní SPI.
Přijímací buffer I^2C	`BA0`	uint8	1	Počítadlo přijatých hodnot v bufferu I^2C.
	`BA1`	uint8	80	Přijímací buffer rozhraní I^2C.

Data z A/D převodníku odpovídají vstupům A1–A14. Obdobným způsobem odpovídají 3 byty stínových registrů vstupních i výstupních hodnot „reléovým výstupům“ O1–O24.

Počítadla přijímacích bufferů komunikačních rozhraní jsou s každým komunikačním cyklem nulována. Platnost dat bufferů je zaručena pouze do dalšího komunikačního cyklu.

Závěr

Zadavatelem této práce byl požadován systém, který po stránce hardwaru splní zadané požadavky a umožní implementaci protokolu XCP. K realizaci hardwarové části tohoto systému byly navrženy celkem dvě desky plošných spojů. První z těchto desek obsahovala některé více či méně závažné chyby. Druhá a konečná deska plošných spojů měla za úkol tyto nedostatky napravit a zlepšit některé části návrhu. Tato deska plošných spojů obsahuje celkem zhruba 315 jednotlivých komponent a její rozměry činí přibližně 132 x 143 mm.

Systém uživateli poskytuje 24 tranzistorových buněk na digitálních výstupech mikrokontroléru, které jsou určeny pro spínání relé, případně jakékoliv jiné zátěže. Připojená zátěž je spojena s nulovým potenciálem a spínána oproti napájecímu napětí systému, které se pohybuje v rozmezí 5–38 V. Všechny buňky najednou by teoreticky měly být schopny sepnout proudovou zátěž až do hodnoty 0,64 A. Systém dále disponuje 14 analogovými vstupy do 10bitového analogově-digitálního převodníku v mikrokontroléru. Tyto vstupy jsou dodatečně chráněny proti přepětí. Jejich předpokládaným účelem je připojení analogových teplotních senzorů (jako nejvhodnější senzor pro tento účel byl zvolen senzor LMT87). Referenční napětí převodníku je 4,096 V a napětí na analogových vstupech by samozřejmě tuto hodnotu nemělo běžně překračovat. V systému jsou navíc dostupně vyvedeny komunikační sběrnice UART, SPI a I^2C.

V mikrokontroléru typu STM8AF52A9 byla implementována aplikace se zařízením slave pro protokol XCP na komunikační sběrnici CAN. Implementována byla značná část příkazů pro mandatorní práci s protokolem, kalibraci a synchronní získávání dat. Celkem bylo implementováno 26 příkazů protokolu XCP.

Systém obsahuje dva parametry, díky kterým lze upravit dvojici časových intervalů procesů v systému. Jeden z těchto intervalů je určen pro digitální výstupy a druhý pro měření. Uživatel musí zaručit takové nastavení těchto parametrů, které umožní systému dostatečný čas pro provedení všech spřažených akcí. Kromě identifikátoru zařízení systém také obsahuje měnitelné parametry se třemi druhy identifikátorů a s přenosovou rychlostí pro sběrnici CAN. Dále jsou v paměti přítomny inicializační stavy dostupných vstupů a výstupů a informace o zařízeních, která k nim byla připojena. To samé platí i pro trojici komunikačních rozhraní – u nich jsou navíc dostupné automatizované vysílací buffery. Všechna tato konfigurační data lze nechat uložit do nevolatilní paměti EEPROM pomocí příkazu XCP s názvem

SET_REQUEST. Na začátku každé relace jsou tato data z paměti EEPROM obnovena zpět do paměti RAM.

Implementace protokolu XCP umožňuje uživateli různé způsoby ovládání digitálních výstupů pro připojení relé. Pokud je využíván princip „polling“, tedy opakované „manuální“ zadávání dat, mohou být k tomuto účelu využívány kalibrační příkazy DOWNLOAD (pro změnu jednoho či více bytů najednou) či MODIFY_BITS (pro přístup k jednotlivým bitům). Uživatel může k výstupům přistupovat pomocí stínových registrů, ze kterých jsou v nastavitelném časovém intervalu tyto hodnoty přenášeny na hardwarové výstupy. Kromě principu polling lze využít také druhého způsobu, který spočívá v nakonfigurování listu DAQ2, jež umožňuje synchronní stimulaci výstupů na relé. Dostupný je také list DAQ1, který může být použit ke zpětnému synchronnímu čtení těchto hodnot.

Analogové vstupy jsou s danou periodou čteny do bufferu a případně matematicky přepočítávány přes požadované funkce do dalšího bufferu. K získávání těchto dat lze opět užívat oba výše zmíněné způsoby. Princip polling zde spočívá v použití příkazu UPLOAD (popř. variace SHORT_UPLOAD) pro čtení těchto hodnot. Sofistikovanější způsob pro synchronní získávání těchto dat spočívá v konfiguraci listu DAQ0. Ten je poté periodicky spouštěn po dokončení přepočtu těchto dat přes požadované funkce.

Každý ze tří nakonfigurovaných listů DAQ lze pomocí příkazů START_STOP_DAQ a SET_REQUEST uložit do paměti EEPROM. Takový list je poté na začátku příští relace obnoven do paměti RAM a automaticky spuštěn v pokračovacím režimu.

Trojice dostupných komunikačních rozhraní UART, SPI a I^2C umožňuje rozšíření celého systému o další zařízení. Každé toto rozhraní může doplnit systém o jeden či více analogově-digitálních převodníků, vstupně-výstupních expandérů, teplotních senzorů či komunikaci s dalšími zařízeními s mikrokontroléry. Největší potenciál v tomto ohledu nabízí rozhraní I^2C, a to z toho důvodu, že se jedná o sběrnici, u které je využívána adresace dostupných zařízení. Přes I^2C lze tak připojit kupříkladu množství dodatečných teplotních senzorů (např. senzory typu TC74 či DS1631+). V případě rozhraní SPI a UART je selekce připojených zařízení problematická a u našeho systému jsou proto tato rozhraní vhodná především k připojení pouze jednoho zařízení. Se zařízeními na trojici rozhraní je zahajována komunikace z vysílacích bufferů v periodách, stanovených časovým intervalem pro měření. U rozhraní I^2C a SPI je navíc dostupný buffer, určený pro počáteční inicializaci zařízení. S událostmi získávání dat z těchto rozhraní není ovšem jednoznačně synchronizován žádný list DAQ. K tomuto účelu musí být proto buď užívána metoda polling, anebo využit některý

z ostatních dostupných listů DAQ, v kterémžto případě by ovšem nemusela být zaručena jednoznačná konzistentnost dat.

Výsledná aplikace tedy podporuje veškerou funkcionalitu, která byla od tohoto zařízení požadována – nastavování výstupů, čtení senzorů, obsluha komunikačních rozhraní, určitá míra přizpůsobení zařízení a poskytování informací o něm a připojených zařízení.

Zařízení jsme v pobočce firmy ZF nejprve zkoušeli na jeho kompatibilitu s obslužným softwarem CANape. Dále bylo testováno čtení senzorů metodou polling i pomocí konfigurace listu DAQ. Nakonec bylo ozkoušeno nastavování výstupu metodou polling. Výsledek těchto zkoušek byl velmi uspokojivý.

Já osobně mohu dodat už jenom to, že doufám, že toto zařízení bude dobře a spolehlivě sloužit svému účelu.

Použitá literatura

[1] Wikipedia contributors. *Software testing* [online]. San Francisco: Wikipedia, The Free Encyclopedia, 2018 [cit. 18. 5. 2017]. Dostupné z: *https://en.wikipedia.org/w/index.php?title=Software_testing&oldid=835845707*

[2] De Wille, Eberhard. *Software Testing* [online]. Schmidmühlen: the-software-experts, 2018 [cit. 18. 5. 2017]. Dostupné z: *http://www.the-software-experts.com/e_dta-sw-test-principles.php*

[3] Wikipedia contributors. *Hardware-in-the-loop simulation* [online]. San Francisco: Wikipedia, The Free Encyclopedia, 2018 [cit. 18. 5. 2017]. Dostupné z: *https://en.wikipedia.org/w/index.php?title=Hardware-in-the-loop_simulation&oldid=824434752*

[4] Patzer, Andreas a Zaiser, Rainer. *XCP – The Standard Protocol for ECU Development*. Stuttgart: Vector Informatik GmbH, 2014 [cit. 18. 5. 2017].

[5] Vector Informatik. *Description Files for Internal ECU Parameters* [online]. Stuttgart: Vector Informatik GmbH, 2014 [cit. 5. 2. 2018]. Dostupné z: *https://vector.com/vi_datadescription_ecu1_en.html*

[6] Association for Standardization of Automation and Measuring Systems. *XCP Version 1.0 – Part 1 – Overview*. Höhenkirchen: ASAM e.V., 2003 [cit. 5. 2. 2018].

[7] Association for Standardization of Automation and Measuring Systems. *XCP Version 1.0 – Part 2 – Protocol Layer Specification*. Höhenkirchen: ASAM e.V., 2003 [cit. 5. 2. 2018].

[8] STMicroelectronics. *Automotive 8-bit MCU, with up to 128 Kbyte Flash, data EEPROM, 10-bit ADC, timers, LIN, CAN, USART, SPI, I2C, 3 to 5.5 V* [online]. Ženeva: STMicroelectronics N.V., 2016 [cit. 18. 5. 2017]. Dostupné z: *http://www.st.com/content/ccc/resource/technical/document/datasheet/e0/31/79/8c/82/d7/40/21/CD00184072.pdf/files/CD00184072.pdf/jcr:content/translations/en.CD00184072.pdf*

[9] STMicroelectronics. *Automotive 38 V, 500 mA synchronous step-down switching regulator with 30 µA quiescent current* [online]. Ženeva: STMicroelectronics N.V., 2016 [cit. 18. 5. 2017]. Dostupné z: *http://www.st.com/content/ccc/resource/technical/document/datasheet/55/83/d4/c6/6e/6c/4f/ab/DM00176243.pdf/files/DM00176243.pdf/jcr:content/translations/en.DM00176243.pdf*

[10] STMicroelectronics. *STM8 SWIM communication protocol and debug module* [online]. Ženeva: STMicroelectronics N.V., 2016 [cit. 5. 2. 2018]. Dostupné z: *http://www.st.com/content/ccc/resource/technical/document/user_manual/ca/89/41/4e/72/31/49/f4/CD00173911.pdf/files/CD00173911.pdf/jcr:content/translations/en.CD00173911.pdf*

[11] STMicroelectronics. *ST-LINK/V2 in-circuit debugger/programmer for STM8 and STM32* [online]. Ženeva: STMicroelectronics N.V., 2016 [cit. 5. 2. 2018]. Dostupné z: *http://www.st.com/content/ccc/resource/technical/document/user_manual/65/e0/44/72/9e/34/41/8d/DM00026748.pdf/files/DM00026748.pdf/jcr:content/translations/en.DM00026748.pdf*

Přílohy

Ukázka komunikace XCP výsledného systému

CAN ID: MASTER=7E0, SLAVE=7F0, BROADCAST=7D0

ÚVODNÍ SEKVENCE

ID	*PID*	*PARAMETRY*	*INFORMACE*
7D0	F2	FF 58 43 50 00 -- --	GET_SLAVE_ID, ‚IDENTIFY BY ECHO‘ MODE
7F0	FF	58 43 50 00 00 07 E0	OK, ASCII ‚XCP‘, MASTER’S_ID=0x000007E0
7E0	F6	00 00 00 00 00 11 C0	CMD, NO RESPONSE, NOT CONNECTED
7E0	FC	-- -- -- -- -- -- --	CMD, NO RESPONSE, NOT CONNECTED
7E0	FF	00 -- -- -- -- -- --	CONNECT, MODE=NORMAL
7F0	FF	04 81 08 00 08 01 01	OK, MAX_CTO=8, MAX_DTO=8, MODES & VERS…
7E0	FC	-- -- -- -- -- -- --	SYNCH (FOR SYNCHRONIZATION PURPOSES)
7F0	FE	00 -- -- -- -- -- --	NO, ERROR_CODE=SYNCH
7E0	FB	-- -- -- -- -- -- --	GET_COMM_MODE_INFO
7F0	FF	00 00 00 00 00 00 10	OK, ONLY STD. COMM. MODE + ADDIT. INFO
7E0	FD	-- -- -- -- -- -- --	GET_STATUS
7F0	FF	00 00 00 12 13 -- --	OK, STATUS INFO, SESSIONCONFIGID=0x1213

ČTENÍ ANALOGOVÝCH TEPLOTNÍCH SENZORŮ LMT87 METODOU POLLING

7E0	F6	00 00 00 00 00 0B 04	SET_MTA, ADDR=0x0B04
7F0	FF	-- -- -- -- -- -- --	OK
7E0	F5	02 -- -- -- -- -- --	UPLOAD, BYTES=2
7F0	FF	02 33 -- -- -- -- --	OK, AN3_RAW=0x0233=563~2.252V
7E0	F6	00 00 00 00 00 0B 28	SET_MTA, ADDR=0x0B28
7F0	FF	-- -- -- -- -- -- --	OK
7E0	F5	02 -- -- -- -- -- --	UPLOAD, BYTES=2
7F0	FF	0B 22 -- -- -- -- --	OK, SENSOR03_TRUE=0x0B22=2850~28.5°C
7E0	F4	02 00 00 00 00 0B 06	SHORT_UPLOAD, BYTES=2, ADDR=0x0B06

```
7F0    FF  02 44 -- -- -- -- --    OK, AN4_RAW=0x0244=580~2.32V

7E0    F4  02 00 00 00 00 0B 2A    SHORT_UPLOAD, BYTES=2, ADDR=0x0B2A
7F0    FF  09 0C -- -- -- -- --    OK, SENSOR04_TRUE=0x090C=2310~23.1°C
```

MODIFIKACE VÝSTUPU O16 PŘES STÍNOVÝ REGISTR (FYZICKY GPIOE BIT3)

```
7E0    F4  01 00 00 00 00 50 14    SHORT_UPLOAD, BYTES=1, ADDR=0x5014
7F0    FF  00 -- -- -- -- -- --    OK, GPIOE_ODR=00

7E0    F4  01 00 00 00 00 0B 4C    SHORT_UPLOAD, BYTES=1, ADDR=0x0B4C
7F0    FF  00 -- -- -- -- -- --    OK, SHADOW_REG_BYTE1=00

7E0    F6  00 00 00 00 00 0B 4C    SET_MTA, ADDR=0x0B4C
7F0    FF  -- -- -- -- -- -- --    OK

7E0    F0  01 80 -- -- -- -- --    DOWNLOAD, BYTES=1, DATA=0x80
7F0    FF  -- -- -- -- -- -- --    OK

7E0    F4  01 00 00 00 00 0B 4C    SHORT_UPLOAD, BYTES=1, ADDR=0x0B4C
7F0    FF  80 -- -- -- -- -- --    OK, SHADOW_REG_BYTE1=80

7E0    F4  01 00 00 00 00 50 14    SHORT_UPLOAD, BYTES=1, ADDR=0x5014
7F0    FF  08 -- -- -- -- -- --    OK, GPIOE_ODR=08

7E0    F6  00 00 00 00 00 0B 4C    SET_MTA, ADDR=0x0B4C
7F0    FF  -- -- -- -- -- -- --    OK

7E0    F0  01 00 -- -- -- -- --    DOWNLOAD, BYTES=1, DATA=0x00
7F0    FF  -- -- -- -- -- -- --    OK
```

MODIFIKACE VÝSTUPŮ POMOCÍ MODIFIKACE BITŮ

```
7E0    F4  01 00 00 00 00 50 0F    SHORT_UPLOAD, BYTES=1, ADDR=0x500F
7F0    FF  00 -- -- -- -- -- --    OK, GPIOD_ODR=00

7E0    F4  01 00 00 00 00 0B 4D    SHORT_UPLOAD, BYTES=1, ADDR=0x0B4D
7F0    FF  00 -- -- -- -- -- --    OK, SHADOW_REG_BYTE2=00

7E0    F6  00 00 00 00 00 0B 4D    SET_MTA, ADDR=0x0B4D
7F0    FF  -- -- -- -- -- -- --    OK

7E0    EC  00 FF FF 00 35 -- --    MODIFY_BITS, &MSK=0xFFFF, XORMSK=0x0035
```

```
7F0    FF  -- -- -- -- -- -- --    OK

7E0    F4  01 00 00 00 00 0B 4D    SHORT_UPLOAD, BYTES=1, ADDR=0x0B4D
7F0    FF  35 -- -- -- -- -- --    OK, SHADOW_REG_BYTE2=35

7E0    F4  01 00 00 00 00 50 0F    SHORT_UPLOAD, BYTES=1, ADDR=0x500F
7F0    FF  34 -- -- -- -- -- --    OK, GPIOD_ODR=34

7E0    F6  00 00 00 00 00 0B 4D    SET_MTA, ADDR=0x0B4D
7F0    FF  -- -- -- -- -- -- --    OK

7E0    EC  00 FF CF 00 04 -- --    MODIFY_BITS, &MSK=0xFFCF, XORMSK=0x0004
7F0    FF  -- -- -- -- -- -- --    OK

7E0    F4  01 00 00 00 00 0B 4D    SHORT_UPLOAD, BYTES=1, ADDR=0x0B4D
7F0    FF  01 -- -- -- -- -- --    OK, SHADOW_REG_BYTE2=01

7E0    F4  01 00 00 00 00 50 0F    SHORT_UPLOAD, BYTES=1, ADDR=0x500F
7F0    FF  00 -- -- -- -- -- --    OK, GPIOD_ODR=00
```

DAQ - KONFIGURAČNÍ SEKVENCE

```
7E0    D8  00 00 00 -- -- -- --    GET_DAQ_LIST_INFO, DAQ=0
7F0    FF  06 0A 07 00 00 -- --    OK, MAX_ODT=10, MAX_ENTRY=7, ...

7E0    D8  00 00 01 -- -- -- --    GET_DAQ_LIST_INFO, DAQ=1
7F0    FF  06 05 07 00 01 -- --    OK, MAX_ODT=5, MAX_ENTRY=7, ...

7E0    DA  -- -- -- -- -- -- --    GET_DAQ_PROCESSOR_INFO
7F0    FF  9C 00 03 00 02 00 00    OK, MAX_DAQ=3, MAX_EVENT_CHANNEL=2, ...

7E0    D9  -- -- -- -- -- -- --    GET_DAQ_RESOLUTION_INFO
7F0    FF  01 04 01 04 42 CC CC    OK, GRANULARITY & TIMESTAMP INFO

7E0    E3  00 00 00 -- -- -- --    CLEAR_DAQ_LIST, DAQ=0
7F0    FF  -- -- -- -- -- -- --    OK

7E0    E2  00 00 00 00 00 -- --    SET_DAQ_PTR, DAQ=0, ODT=0, ENTRY=0
7F0    FF  -- -- -- -- -- -- --    OK

7E0    E1  FF 02 00 00 00 0B 04    WRITE_DAQ, SIZE=2, ADDR=0x0B04
7F0    FF  -- -- -- -- -- -- --    OK
```

```
7E0    E2  00 00 00 01 00 -- --    SET_DAQ_PTR, DAQ=0, ODT=1, ENTRY=0
7F0    FF  -- -- -- -- -- -- --    OK

7E0    E1  FF 02 00 00 00 0B 34    WRITE_DAQ, SIZE=2, ADDR=0x0B34
7F0    FF  -- -- -- -- -- -- --    OK

7E0    E2  00 00 00 02 00 -- --    SET_DAQ_PTR, DAQ=0, ODT=2, ENTRY=0
7F0    FF  -- -- -- -- -- -- --    OK

7E0    E1  FF 04 00 00 00 01 80    WRITE_DAQ, SIZE=4, ADDR=0x0180
7F0    FF  -- -- -- -- -- -- --    OK

7E0    E1  FF 02 00 00 00 01 84    WRITE_DAQ, SIZE=2, ADDR=0x0184
7F0    FF  -- -- -- -- -- -- --    OK

7E0    E0  10 00 00 00 00 01 00    SET_DAQ_LIST_MODE, DAQ=0, MODE=TIMESTP.
7F0    FF  -- -- -- -- -- -- --    OK

7E0    DE  01 00 00 -- -- -- --    START_STOP_DAQ_LIST, START, DAQ=0
7F0    FF  00 -- -- -- -- -- --    OK, FIRST_PID=0
```

DAQ BĚŽÍ (ČTENÍ ANALOGOVÉHO TEPLOTNÍHO SENZORU MCP9701)

```
7F0    00  89 47 00 D3 -- -- --    TIMESTAMP=0x8947, ADC_RAW_VL=0x00D3=211
7F0    01  08 E9 -- -- -- -- --    TEMPERATURE=0x08E9=2281~22.81°C
7F0    02  CC AA 45 89 52 04 --    RAND_MEM_VALS=0xCCAA4589, 0x5204

7F0    00  A1 B1 00 D6 -- -- --    TIMESTAMP=0xA1B1, ADC_RAW_VL=0x00D6=214
7F0    01  09 26 -- -- -- -- --    TEMPERATURE=0x0926=2342~23.42°C
7F0    02  CC AA 45 89 52 04 --    RAND_MEM_VALS=0xCCAA4589, 0x5204

7F0    00  BA 1B 00 D5 -- -- --    TIMESTAMP=0xBA1B, ADC_RAW_VL=0x00D5=213
7F0    01  09 12 -- -- -- -- --    TEMPERATURE=0x0912=2322~23.22°C
7F0    02  CC AA 45 89 52 04 --    RAND_MEM_VALS=0xCCAA4589, 0x5204

7E0    DE  00 00 00 -- -- -- --    START_STOP_DAQ_LIST, STOP, DAQ=0
7F0    FF  00 -- -- -- -- -- --    OK, FIRST_PID=0
```

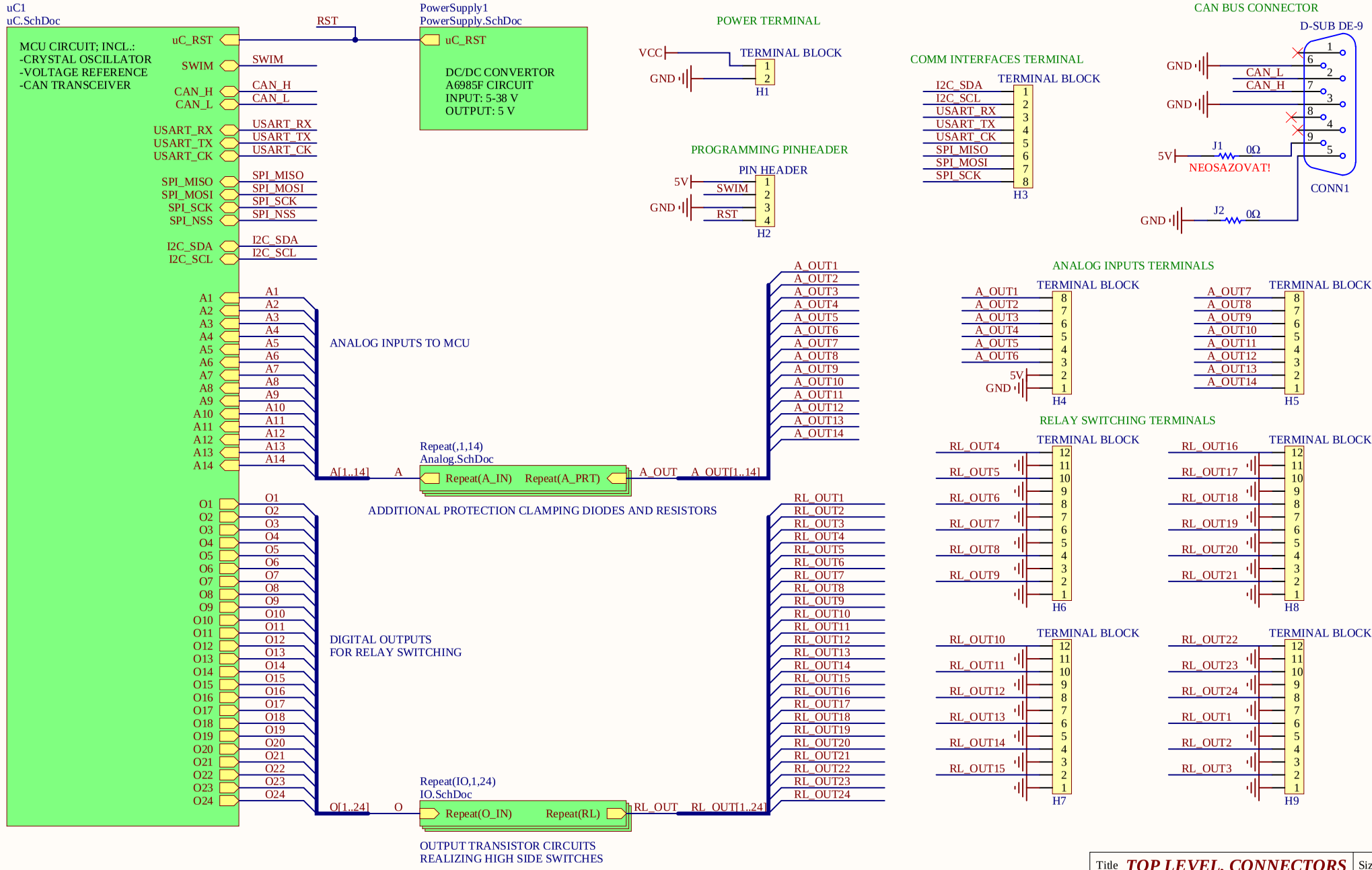
UKLÁDÁNÍ DO EEPROM (EEPROM ADRESA 0x4000–0x4799, OBRAZ V RAM 0xC00–0x1000)

```
7E0    F4  04 00 00 00 00 40 02    SHORT_UPLOAD, BYTES=4, ADDR=0x4002
7F0    FF  FF FF FF FF -- -- --    OK, DATA
```

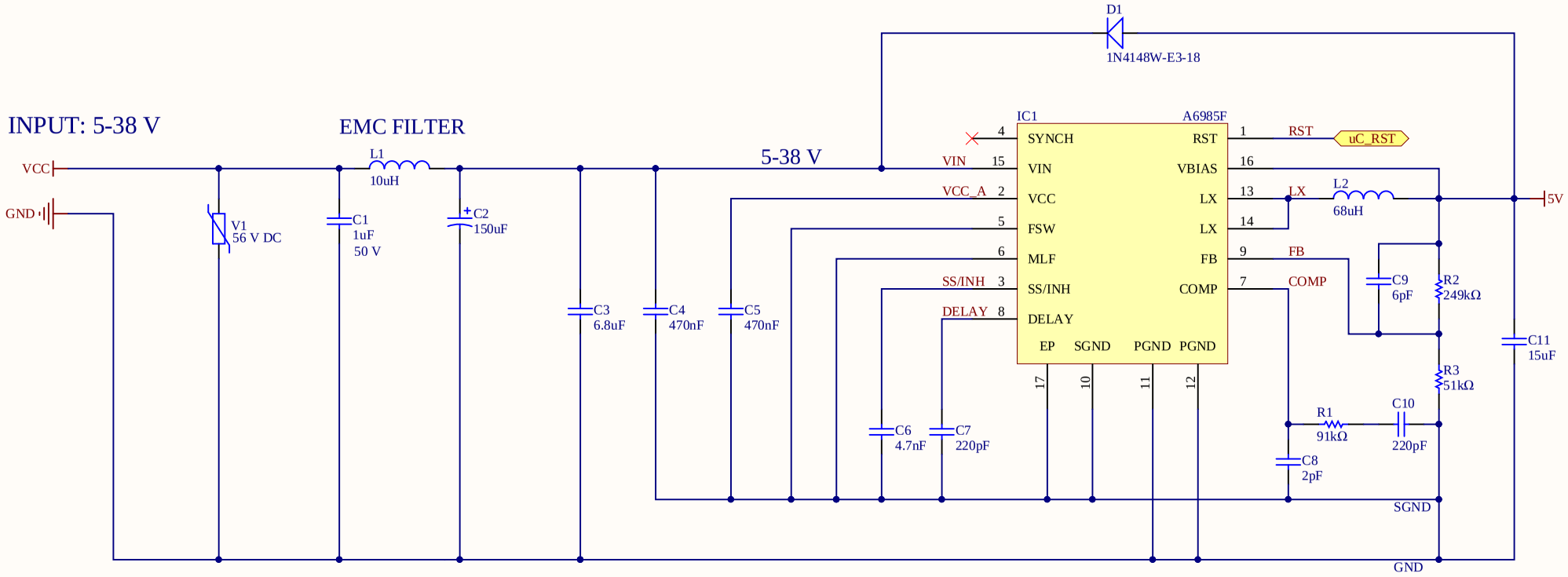
```
7E0    F4  04 00 00 00 00 0C 02    SHORT_UPLOAD, BYTES=4, ADDR=0x0C02
7F0    FF  FF FF FF FF -- -- --    OK, DATA

7E0    F6  00 00 00 00 00 0C 02    SET_MTA, ADDR=0x0C02
7F0    FF  -- -- -- -- -- -- --    OK

7E0    F0  04 AA BB CC DD -- --    DOWNLOAD, BYTES=4, DATA=0xAABBCCDD
7F0    FF  -- -- -- -- -- -- --    OK

7E0    F4  04 00 00 00 00 0C 02    SHORT_UPLOAD, BYTES=4, ADDR=0x0C02
7F0    FF  AA BB CC DD -- -- --    OK, DATA

7E0    F4  04 00 00 00 00 40 02    SHORT_UPLOAD, BYTES=4, ADDR=0x4002
7F0    FF  FF FF FF FF -- -- --    OK, DATA

7E0    F9  01 00 00 -- -- -- --    SET_REQUEST, MODE=STORE_CAL_REQ
7F0    FF  -- -- -- -- -- -- --    OK, SAVING...
7F0    FD  03 -- -- -- -- -- --    EVENT=EV_STORE_CAL

7E0    F4  04 00 00 00 00 40 02    SHORT_UPLOAD, BYTES=4, ADDR=0x4002
7F0    FF  AA BB CC DD -- -- --    OK, DATA

7E0    FE  -- -- -- -- -- -- --    DISCONNECT
7F0    FF  -- -- -- -- -- -- --    OK
```

Schémata finální DPS

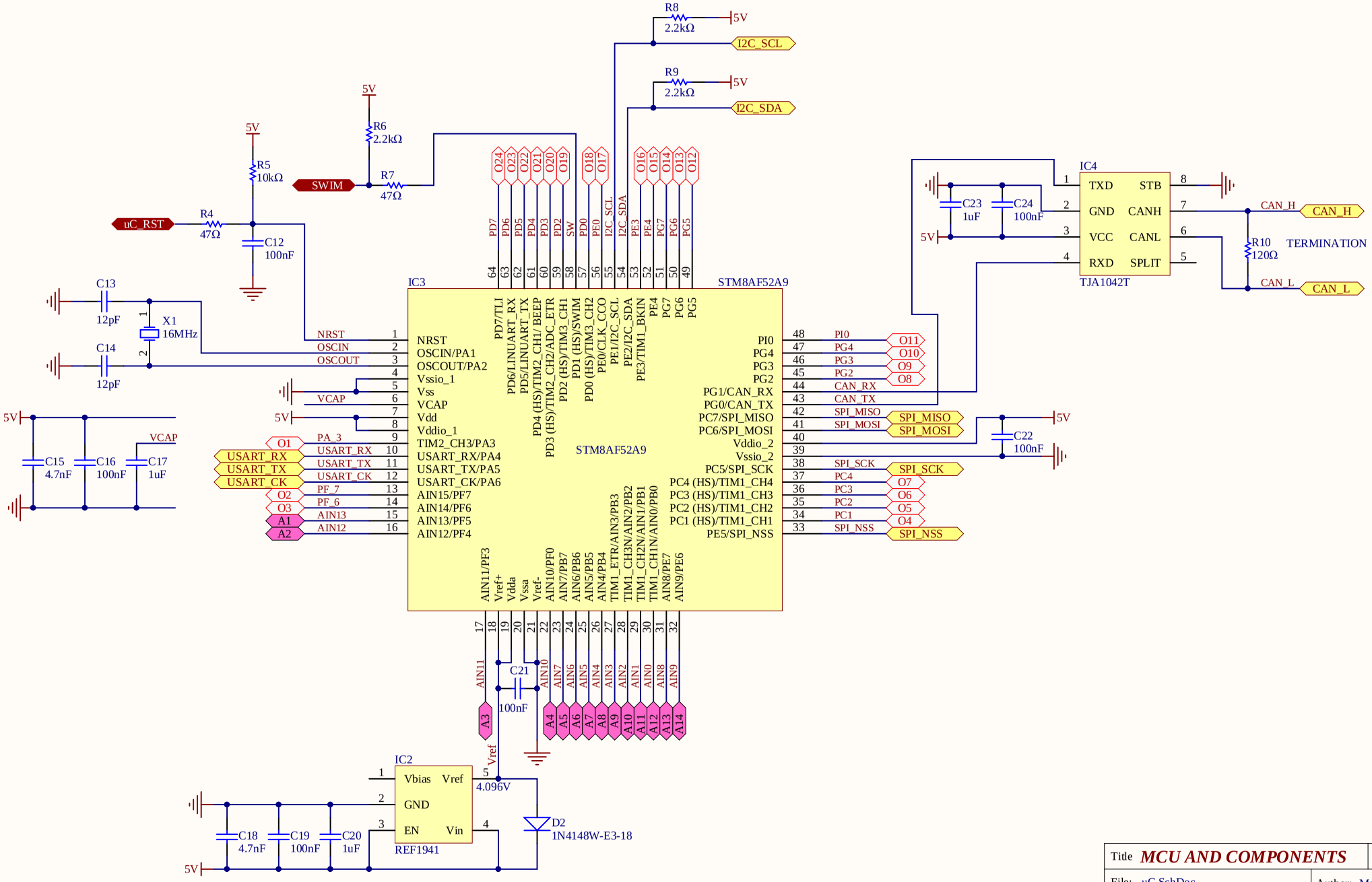
Na následujících čtyřech stranách se nacházejí schémata k finální desce plošných spojů. Schémata mají hlavní zastřešující strukturu Top_Level.SchDoc, ve které jsou instancovány prvky z ostatních schémat. V této struktuře jsou také přítomny připojovací konektory.

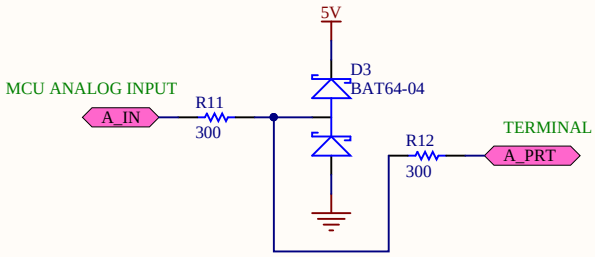


Title	***TOP LEVEL, CONNECTORS***	Size: A4
File: Top_Level.SchDoc		Author: Martin Hás
Date: 11.5.2018		Sheet 1 of 5

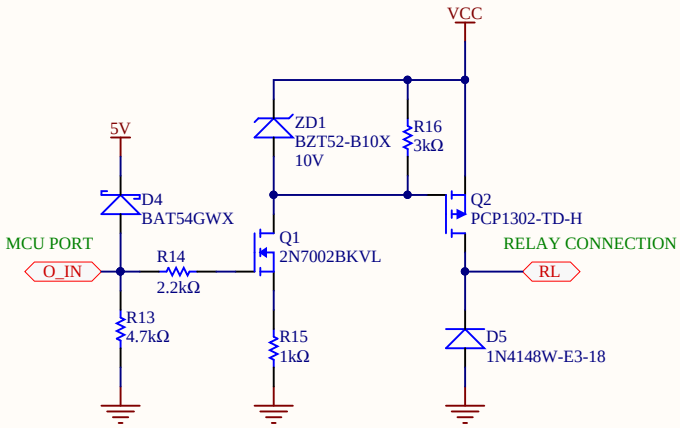


Title **POWER SUPPLY**		Size: A4
File: PowerSupply.SchDoc	Author: Martin Hás	
Date: 11.5.2018	Sheet 2 of 5	


R8
2.2kΩ
5V
I2C_SCL
R9
2.2kΩ
5V
I2C_SDA
5V
R6
2.2kΩ
SWIM
R7
47Ω
5V
R5
10kΩ
uC_RST
R4
47Ω
C12
100nF
C13
12pF
X1
16MHz
C14
12pF
IC3
STM8AF52A9
NRST
OSCIN/PA1
OSCOUT/PA2
Vssio_1
Vss
VCAP
Vdd
Vddio_1
TIM2_CH3/PA3
USART_RX/PA4
USART_TX/PA5
USART_CK/PA6
AIN15/PF7
AIN14/PF6
AIN13/PF5
AIN12/PF4
PI0
PG4
PG3
PG2
PG1/CAN_RX
PG0/CAN_TX
PC7/SPI_MISO
PC6/SPI_MOSI
Vddio_2
Vssio_2
PC5/SPI_SCK
PC4 (HS)/TIM1_CH4
PC3 (HS)/TIM1_CH3
PC2 (HS)/TIM1_CH2
PC1 (HS)/TIM1_CH1
PE5/SPI_NSS
IC4
TJA1042T
TXD
GND
VCC
RXD
STB
CANH
CANL
SPLIT
C23
1uF
C24
100nF
5V
CAN_H
CAN_L
R10
120Ω
TERMINATION
C22
100nF
5V
SPI_MISO
SPI_MOSI
SPI_SCK
SPI_NSS
C15
4.7nF
C16
100nF
C17
1uF
VCAP
C21
100nF
IC2
REF1941
Vbias
GND
EN
Vref
Vin
4.096V
C18
4.7nF
C19
100nF
C20
1uF
D2
1N4148W-E3-18
Title MCU AND COMPONENTS
Size: A4
File: uC.SchDoc
Author: Martin Hás
Date: 11.5.2018
Sheet 3 of 5



Title	***ANALOG PROTECTION***	Size:	A5
File:	Analog.SchDoc	Author:	Martin Hás
Date:	11.5.2018	Sheet	4 of 5



Title	***RELAY OUTPUTS***	Size:	A5
File:	IO.SchDoc	Author:	Martin Hás
Date:	11.5.2018	Sheet	5 of 5