Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Programovací interface pro systém REMCS

Plzeň 2014 Petr Dallinger

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 8. května 2014

Petr Dallinger

Abstract

The subject of this bachelor's thesis is to design and implement a programming interface for industrial control system REMCS. This work gives its readers a basic outline of commonly used communication a diagnostic protocols. It also shows basic principles of storing data in flash memory cells.

The thesis defines a custom application layer protocol transmitted over the CAN bus.

Primary object of the work is the creation of application software which implements the developed protocol on PC. The aim was to create a easily applicable user interface, which allows programming of the whole control system at the same time.

Abstrakt

Tato bakalářská práce si dává za cíl navrhnout a implementovat programovací rozhraní pro průmyslový řídící systém REMCS. Práce poskytuje čtenáři základní nástin problematiky zápisu do pamětí typu flash, dále popisuje běžně používané komunikační a diagnostické protokoly.

Práce definuje vlastní protokol aplikační vrstvy postavený nad sběrnicí CAN.

Hlavní částí práce je výroba aplikačního software, který implementuje navržený protokol pro PC. Cílem je vytvořit program s jednoduše použitelným uživatelským rozhraním, pomocí kterého je možné programovat celý řídící systém.

Obsah

1 Úvod

Většina společností, které se v současné době orientují na vývoj a výrobu elektronických řídících systémů, bojuje s problémem konkurence. Potřebují svůj systém uvést ve velice krátké době na trh. V případě složitých řídicích systémů to pro ně může představovat překážku, kterou nejsou schopné překonat.

Modulární řídicí systém REMCS je určený pro řízení real-time aplikací v oblasti energetiky a dopravní techniky se zvýšenými požadavky na bezpečnost. Cílovým zákazníkem centra RICE, které tento systém vyvíjí, není konkrétní koncový uživatel. Klade si za cíl dát především menším a středním firmám přístup k vlastní platformě high-tech řídicího systému, na jehož vývoj by jinak neměly kapacity, know-how, nebo pokud potřebují vlastní systém nasadit v krátkém časovém měřítku.

Od RICE tak zákazník dostává HW platformu a SW knihovny pro její ovládání. Výsledný produkt je pak v jeho režii. Je zodpovědná za vývoj výrobku, jeho uvedení do provozu i následnou údržbu celého systému a jeho podporu u koncového zákazníka.

To s sebou přináší požadavek na univerzální SW nástroj, který by umožnil nahrát firmware do všech součástí systému. Požadavek na nástroj, který by byl dostatečně robustní a jednoduchý, aby byl použitelný během celého životního cyklu systému – jak ve fázi vývoje u výrobce, tak třeba servisním technikem koncového zákazníka.

1.1 Účel práce

Cílem této práce je navrhnout a implementovat SW řešení pro flashování firmware do modulů systému REMCS, a to s následujícími vlastnostmi:

- jeden univerzální nástroj pro flashování všech čipů v systému
- distribuce SW prostřednictvím balíčků
- snadná obsluha

Návrhu bude předcházet analýza stávajících průmyslových protokolů pro komunikaci po sběrnici CAN. Komunikace mezi implementovaným SW nástrojem a systémem REMCS by měla být postavena buď jako rozšíření standardního protokolu, nebo na vlastním navrženém protokolu. Požadavkem je plánovaná podpora sběrnic CAN i Ethernet.

2 Rozhraní CAN (Controller Area Network)

Sběrnice CAN vznikla v druhé polovině osmdesátých let dvacátého století. Za jejím zrodem stála společnost Bosch. Komunikační protokol byl vytvořen s ohledem na požadavky automobilového průmyslu – vysokou rychlostí přenosu dat a vysokou mírou zabezpečení komunikace proti chybám.

Od začátku byla sběrnice CAN vyvíjena jako síť decentralizovaná. Vzhledem k neexistenci jednoho nadřazeného uzlu se sběrnice vyznačuje vysokou spolehlivostí. V případě výpadku jednoho uzlu na sběrnici mohou ostatní uzly dále komunikovat.

Maximální přenosová rychlost je 1MBit/s. Rychlosti se v různých systémech mohou lišit, na jedné sběrnici ale musí být vždy shodná.

Celá sběrnice pracuje v režimu multi-master. To znamená, že všechna zařízení na sběrnici mohou pracovat v režimu master a řídit tak komunikaci. Zařízení se stává masterem v okamžiku, kdy začne vysílat. Ostatní uzly musí v tom případě vyčkat na uvolnění sběrnice, mohou pouze poslouchat.

Protokol definuje dominantní a recesivní úroveň. Recesivní úroveň je na sběrnici trvale přítomna v okamžiku, kdy žádný uzel nevysílá. Dominantní úroveň je na sběrnici přítomna v okamžiku, kdy kterýkoli z uzlů vysílá dominantní bit (logická hodnota „0“).

2.1 Řešení kolizí na sběrnici

Protože neexistuje žádný arbitr sběrnice, musí být na úrovni CAN protokolu řešena situace konfliktu na sběrnici. V případě, že jeden z uzlů začne vysílat dříve, sběrnice již není volná a druhý uzel musí počkat. Problém nastane pouze v případě, kdy oba uzly zahájí vysílání v jednom okamžiku. Vysílající stanice poslouchá na sběrnici v okamžiku vysílání. Pokud je právě vysílán recesivní bit a na sběrnici je zjištěna dominantní úroveň, znamená to, že souběžně vysílá i jiný vysílač. Proto zařízení přestane okamžitě vysílat.

2.2 Priority zpráv

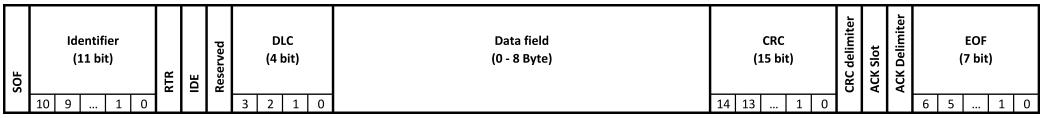
Vzhledem k možnému k použití CAN sběrnice v časově kritických systémech protokol musí řešit i prioritizaci zpráv která zabezpečí včasné doručení důležitých zpráv.

Zařízení na sběrnici CAN nemají žádnou adresu. Stejně tak zprávy vysílané po sběrnici protokolem CAN neobsahují žádnou informaci o cílovém uzlu, kterému jsou určeny, a mohou být přijímány všemi ostatními uzly připojenými ke sběrnici.

Filtrování zpráv je možné na linkové úrovni. Je možné pouze na základě identifikátoru, který je součástí přijímané zprávy. Identifikátoru zprávy je využito i pro nastavení priorit jednotlivých zpráv. Zprávy s nižším ID mají vyšší prioritu – čím vyšší je ID zprávy, tím dříve je vysílán recesivní bit.

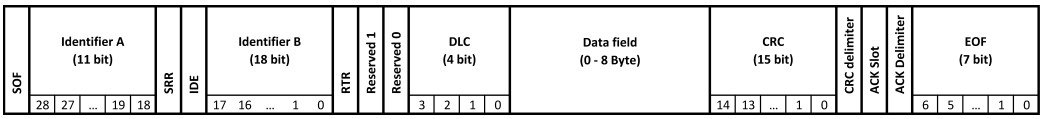
2.3 Struktura zpráv

CAN revize 2.0B definuje 2 formáty zpráv, základní a rozšířený. Základní formát zprávy používá 11Bit identifikátor zprávy. Struktura datového rámce je na obrázku 2.1.



Obrázek 2.1: Formát základní CAN zprávy

Na obrázku 2.2 je struktura rozšířeného datového rámce. Hlavním rozdílem je délka identifikátoru zprávy. Ta je při použití rozšířeného rámce 29 bit.



Obrázek 2.2: Formát rozšířené CAN zprávy

Význam jednotlivých částí datové zprávy podle specifikace CAN 2.0B je následující:

- SOF (Start of Frame) — začátek zprávy, 1 bit, vždy dominant.
- Identifier — Identifikátor zprávy, 11 bit / 29 bit, vysíláno od MSB. V případě rozšířeného rámce rozdělený na dvě části. Na pozici nejvyšších 7 bitů se musí vyskytovat alespoň jeden dominantní bit.
- RTR (Remote Request) — 1 bit, příznak, zda se jedná o datovou zprávu nebo o žádost o vyslání dat. Pro datovou zprávu musí být bit „dominant“, v žádosti o data „recessive“.

- SRR (Subtitute Remote Request) — 1 bit, pouze pro rozšířený formát, musí být „recessive“.
- IDE (Identifier Extension) — identifikace formátu zprávy; pro základní formát zprávy musí být „dominant", pro rozšířený musí být „recessive“.
- Reserved — rezerva, musí být vždy „dominant“.
- DLC (Data Length Code) — 4 bity, počet přenášených dat v bajtech ve zprávě. Dovolené jsou pouze hodnoty 0 až 8.
- Datová oblast (Data Field) — Data zprávy, délka je v DLC, vysíláno od MSB.
- CRC — 15 bitů, kontrolní součet je počítán od SOF do posledního byte z sekce Data field
- CRC delimiter — 1 bit, vždy „dominant“
- ACK Slot — 1 bit, odesilatel vždy vysílá bit „recessive“, uzel, který přijme korektní CRC, vyšle bit „dominant“
- ACK delimiter — 1 bit, vždy „recessive“.
- EOF (End of Frame) — konec zprávy, 7 bit, všechny musí být „recessive“.

3 Rozhraní Ethernet

Rozhraní Ethernet bylo vytvořeno pro použití v lokálních počítačových sítích. V současnosti se jedná o dominantní technologii pro drátové sítě. Jako komunikační médium se používá kroucená dvojlinka. Rozhraní je definováno rodinou normem IEEE 802.3. Ethernet realizuje fyzickou a linkovou vrstvu síťového modelu ISO/OSI.

S nárůstem výkonu počítačů a požadavku na přenosy stále většího objemu dat se rozhraní postupně vyvíjelo. Původně byla maximální přenosová rychlost 10MBit/s. Postupně se staly standardem sítě o přenosových rychlostech 100MBit/s a 1GBit/s. Na optických kabelech mohou být používány i rychlosti 10GBit/s a vyšší.

S rozvojem „chytrých“ zařízení se rozhraní Ethernet stále častěji objevuje i v komerční elektronice, např. v televizorech. Současné mikrokontroléry tak běžně obsahují obvody linkové vrstvy a v některých případech i fyzické vrstvy tohoto síťového rozhraní.

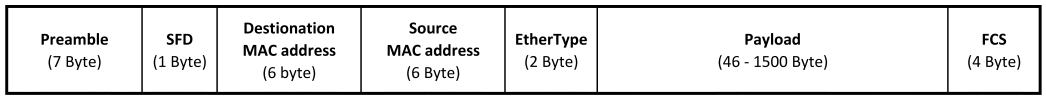
Na obrázku 3.1 je struktura datového rámce Ethernet. Smysl jednotlivých částí rámce je následující:

- Preamble a SFD (Start of Frame Delimiter) – slouží k tomu, aby se zařízení na sběrnici mohla vzájemně synchronizovat. Preamble obsahuje opakující sekvenci bitů „10“. SFD pak v sekvenci pokračuje s výjimkou poslední dvojice. Zde je vždy „11“.

- Source / Destination MAC Address – fyzická adresa odesílatele a příjemce. Za přidělování fyzických adres je zodpovědný výrobce síťového zařízení, adresa by měla být jedinečná.

- Payload – oblast vyhrazená pro data z vyšší síťové vrstvy, např. IP rámec. Podle použitého protokolu se může lišit velikost.

- FCS (Frame Check Sequence) – CRC pro ověření přenášených dat.



Obrázek 3.1: Struktura ethernetového rámce

3.1 TCP/IP

Protokol TCP/IP je dnes nejpoužívanější technologií v počítačových sítích. Vznikl s požadavkem nezávislosti na linkové úrovni. TCP/IP model vymezuje čtyři vrstvy.

1. Vrstva síťového rozhraní
2. Síťová vrstva
3. Transportní vrstva
4. Aplikační vrstva

Vrstva síťového rozhraní poskytuje služby potřebné pro vlastní ovládání sítě, vysílání a příjem dat. Vzhledem k nezávislosti na přenosové technologii, není v systému TCP/IP blíže specifikována. Dominantní technologií v oblasti lokálních sítí je v současnosti Ethernet.

Síťová vrstva slouží k tomu, aby zajišťovala adresaci, směrování a předávání datagramů mezi odesílatelem a příjemcem. Služby v sítích TCP/IP mají nespojovaný charakter, jde pouze o vysílání jednoduchých nespolehlivých datagramů. Architektura vychází z předpokladu, že za zajištění spolehlivosti přenosu by měly být zodpovědné koncové uzly. Díky tomu nemusí být komunikační síť zbytečně vytěžována. Má pouze vyvinout maximální úsilí přenášené pakety doručit. Nejčastěji používaným protokolem síťové vrstvy je protokol IP.

Transportní vrstva je zodpovědná za zajištění přenosu dat mezi dvěma koncovými účastníky. V případě TCP/IP jde přímo o aplikační programy (pracující na aplikační vrstvě). Podle jejich nároků a požadavků může transportní vrstva regulovat tok dat oběma směry, zajišťovat spolehlivost přenosu, nebo třeba také měnit nespojovaný charakter přenosu (v síťové vrstvě) na spojovaný. Příkladem protokolu transportní vrstvy je TCP pro spolehlivé nebo UDP pro nespolehlivé služby.

Aplikační vrstva je nejvyšší vrstvou protokolu TCP/IP. Jejími entitami jsou samotné aplikační programy. Typickými příklady protokolů aplikační vrstvy mohou být třeba HTTP, FTP nebo SSH.

4 Permanentní paměti s možností zápisu

Flash ROM jsou elektricky mazatelné a programovatelné nevolatilní (tj. neztrácejí data po přerušení napájení) paměti. Tento typ pamětí je požíván nejčastěji v embedded řídících systémech pro uchování programu. Flash paměť je složena z pole buněk organizovaných do řádek a sloupců. Každá paměťová buňka je tvořena jedním unipolárním tranzistorem. Jedná se o modifikovaný MOSFET se dvěma elektrodami (source a drain) a dvěma hradly — řídícím (control gate) a plovoucím (floating gate). Plovoucí hradlo je od okolí izolováno vrstvou SiO_2. Řídící hradlo je použito k výběru příslušné paměťové buňky. Plovoucí hradlo ovlivňuje stav paměťové buňky.

4.1 Čtení flash paměti

V případě, že na plovoucím hradle není náboj, tranzistor reaguje jen na řídící hradlo a otevře se. Paměťová buňka pak nese hodnotu „1“. V případě, že na plovoucím hradle náboj je, po výběru buňky tento náboj působí jako bariéra a tranzistor se neotevře. Buňka pak nabývá hodnoty „0“.

4.2 Zápis flash paměti

Flash paměti umožňují zápis dat po jednotlivých buňkách. Výchozí stav paměťové buňky je „0“. Samotná operace zápisu tak představuje nabití plovoucího hradla nábojem. Toho se docílí tzv. tunelováním. Na řídící hradlo je přivedeno napětí vyšší než běžné provozní. Zvýšené napětí se přivede i na drain. Pokud je proud tekoucí mezi drain a source dostatečně velký, elektrony prorazí izolační bariéru plovoucího hradla. Přitom dojde k jeho nabití.

4.3 Mazání flash pamětí

Proces mazání je největší odlišností od ostatních elektricky programovatelných pamětí. Flash paměti umožňují jen mazání paměti po blocích. Pro smazání paměti je mezi řídící hradlo a source přivedené vyšší napětí opačné polarity. Náboj je tunelováním odsát z plovoucího hradla.

5 Běžně používané diagnostické protokoly

S nárůstem počtu elektronických systémů v automobilech vyvstal požadavek na diagnostické funkce a také funkce pro aktualizaci firmware v zařízeních. Zpočátku byla diagnostické funkce omezené a každý z výrobců používal vlastní metody komunikace. To zvyšovalo náklady na vývoj i údržbu systémů.

S rozvojem digitálních technologií a nástupem standardních komunikačních sběrnic jako například K-Line nebo CAN v automobilech vznikla potřeba standardizace komunikačních protokolů.

Zpočátku neexistovala norma, která by předepisovala automobilkám použití univerzálních rozhraní. Od konce devadesátých let 20. století se začal prosazovat standard OBD II (On Board Diagnostic). Ten předepisuje požadované diagnostické funkce.

Pro definici komunikace po sběrnici K-Line byl vytvořen standard ISO 14230 a vznikl tak protokol KWP2000. S nástupem diagnostiky přes CAN-BUS byl jako nástupce KWP2000 a jeho rozšíření navržen protokol UDS (Unified Diagnostic Services) standardizován byl jako ISO14229. Vzhledem k tomu, že se v současnosti používá spíše UDS, bude dále popisován právě tento protokol.

5.1 UDS

UDS protokol je postaven na aplikační vrstvě. Je tak nezávislý na použitém přenosovém médiu. Existují tak implementace protokolu na rozhraní CAN, Ethernet, ale i na jiných sběrnicích používaných v automobilovém průmyslu.

Samotná komunikace probíhá v módu klient–server. V případě UDS bývá klient nejčastěji nazýván tester. Tester vysílá požadavek, server pak tento požadavek zpracovává.

Požadavky na služby jsou potvrzované. Součástí požadavku i odpovědi jsou adresy testera a serveru. Adresace služeb může být fyzická, kdy je adresován vždy jeden konkrétní sever, nebo funkcionální. Funkcionální adresy tester obvykle používá v případě, že buď nezná fyzickou adresu serveru, nebo požaduje odpověď od všech serverů na sběrnici.

Požadavek na službu má následující strukturu:

Byte	Obsah	Popis
1	SA	Adresa odesílatele zprávy
2	TA	Adresa příjemce
3	TA_type	Fyzická/funkcionální adresa
4	SrvReqID	Identifikátor služby
	SubReqID	Identifikátor podslužby (pokud je definována)
	Parameter 1	Volitelné parametry služeb
	⋮	
N	Parameter m	

Protokol UDS definuje následující skupiny služeb:

- Služby pro přístup k paměti
- Služby pro práci s uloženými daty
- Služby pro upload a download
- Služby pro vstup a výstup
- Služby vzdálených volání

Ne všechny služby musí být vždy dostupné. V následující části budou vysvětleny služby, které jsou obvykle použity v případě flashování. Kompletní popis definovaných služeb rozebírá výše uvedené norma UDS.

DiagnosticSessionControl

Pro zvýšení bezpečnosti zavádí protokol UDS různé diagnostické módy serveru, tzv. session. V různých session jsou dostupné různé sady diagnostických služeb. DiagnosticSesstionControl umožňuje přechod mezi jednotlivými session. Na serveru musí být vždy aktivní právě jedna session.

Pro použití všech služeb musí být v systému definovány minimálně 2 diagnostické session. Nejčastěji bývají označovány default a extended. Ve většině implementací pak bývá použita i tzv. programming, popřípadě security session.

TesterPresent

Bývá zvykem, že z důvodu zvýšení bezpečnosti v případě, kdy tester po definovanou dobu nevysílá požadavky, přejde server zpět do default session. Aby k tomu nedošlo, vysílá tester periodicky zprávu TesterPresent a informuje tím server, že je stále připojen a server musí zůstat v právě aktivní session.

SecurityAccess

Služba slouží pro získání bezpečnostního ověření. Je platný pro aktuální session do okamžiku její změny. Je používáno mechanismu asymetrického klíče. Definovány jsou podslužby requestSeed a sendKey. Tester požádá server o zaslání seed (Sub-ID služby). Server mu v odpovědi seed zašle. Tester poté pošle key. Server provede ověření a v případě úspěchu povolí zabezpečený přístup.

WriteDataByIdentifier

Zápis dat určitého typu na základě identifikátoru. Identifikátory jsou specifické pro daný typ serveru. V některých případech bývá požadován zabezpečený přístup, není povoleno volání v default session.

ReadDataByIdentifier

Čtení dat určitého typu podle zadaného identifikátoru ze serveru, identifikátory jsou specifické pro daný typ serveru. Volání ReadDataByIdentifier není povoleno v default session.

RoutineControl

Služba slouží pro práci s voláním vzdálených akcí. UDS protokol definuje dvě možné metody chování testeru a serveru v případě volání vzdálených rutin.

Tester vyšle požadavek na start vzdálené rutiny serveru. Server začne vykonávat požadovanou akci. V případě první metody přístupu je klient zodpovědný za zastavení akce na straně serveru. Po nějaké době tester musí poslat požadavek na zastavení vzdálené rutiny a následně vyšle serveru požadavek na výsledek vzdálené operace.

V případě použití druhé metody přístupu je zodpovědnost za ukončení na straně serveru. To znamená, že tester neposílá požadavek na ukončení vzdálené rutiny, posílá pouze požadavek na výsledek. V případě že klient požaduje data dříve, než jsou na straně serveru dostupná, server zašle negativní odpověď. V praxi bývá druhá metoda použita častěji.

Standard UDS požaduje službu RoutineControl s dvěma parametry:

- Požadovaná akce vzdálené rutiny (start / stop / získání výsledku); i v případě implementace druhé metody musí server na požadavek zastavení rutiny odpovědět.
- Identifikátor vzdálené rutiny

Samotné vzdálené rutiny serveru standard UDS nedefinuje a jsou vždy závislé na daném systému. Volání RoutineControl není povoleno v default session.

RequestDownload

Touto službou dává tester serveru požadavek na start přenosu bloku dat ve směru od klienta k serveru. Po přijetí požadavku se server musí připravit na přenos a teprve po dokončení příprav odešle pozitivní odpověď. Parametry funkce jsou:

- Identifikátor délky adresy a délky přenášených dat – 4 bity každý, první udává délku adresy v bytech, tedy až 128-bitová adresa a druhý určuje počet bytů parametru určujícího délku přenášených dat. Velikost přenášeného bloku tedy může být až 2^{128} bytů.
- Adresa paměti na serveru, od které se mají začít zapisovat data.
- Délka dat v bytech.

Volání služby není povoleno v default session.

RequestUpload

Služba se chová stejně jako RequestDownload, jediným rozdílem je směr přenášených dat. V tomto případě požaduje klient, aby mu server poslal blok dat, která má na zadané adrese.

TransferData

Služba slouží k samotnému přenosu dat. Směr přenosu a počet dat je určen předchozím voláním RequestUpload nebo RequestDownload.

Součástí datové zprávy je čítač pořadí dat. Při prvním zavolání služby TransferData následujícím po RequestUpload nebo RequestDownload musí být nastaven čítač na hodnotu 1 a v každé další zprávě je tento čítač inkrementován. Volání služby není povoleno v default session.

RequestTransferExit

Služba slouží k informování serveru, že další data již nebudou přenášena, jedná se o ukončení operace Download nebo Upload.

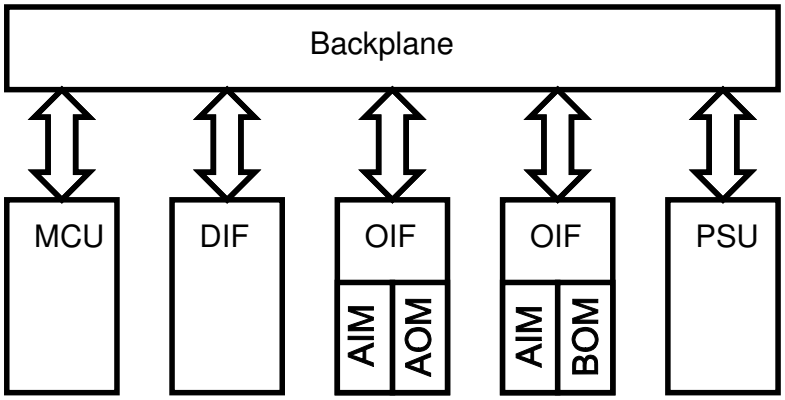
6 Návrh realizace

6.1 Popis systému REMCS

Systém REMCS (RICE Embedded Modular Control System) je komplexní modulární řídicí systém určený pro řízení real-time aplikací v oblasti energetiky a dopravní techniky se zvýšenými požadavky na bezpečnost.

Systém je vyvíjen v rámci Regionálního inovačního centra elektrotechniky (RICE) Západočeské univerzity v Plzni. Je složen ze zásuvných karet a propojovací části, tzv. backplane. Případně je také možné karty používat samostatně.

Příklad konfigurace systému je popsán na obrázku 6.1.



Obrázek 6.1: Příklad konfigurace systému REMCS

Systém může obsahovat karty následujících typů:

- MCU – Mikroprocesorová karta, hlavní řídící jednotka.
- DIF – Direct Interface, karta pro řízení měniče.
- OIF – Open Interface, karta otevřené platformy. Karta může být osazena až dvěma rozšiřujícími moduly.
- PSU – Napájecí zdroj.

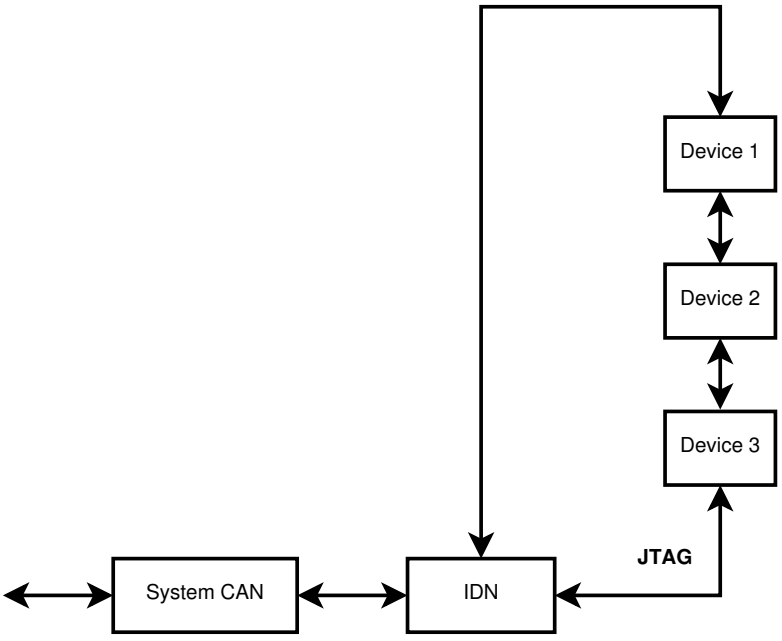
OIF karta v současnosti podporuje následující typy rozšiřujících modulů

- AIM – Analog Input Module, modul analogových vstupů.
- AOM – Analog Output module, modul analogových výstupů.

- BIM – Binary Input Module, modul digitálních vstupů.
- BOM – Binary Output Module, modul digitálních výstupů.
- SII – Sensor Input Interface, modul čidla otáček.

Každá karta obsahuje dva mikrokontroléry. Hlavním procesorem (MMU) je dvoujádrový ARM Cortex R4F z rodiny Hercules. Úkolem tohoto mikrokontoréru je řízení funkcionality celé karty. Ve většině případů nemá přímý kontakt s diagnostickým rozhraním. Druhým mikrokontolérem je pak ARM Cortex M4, který provádí diagnostiku karty a řídí mimo jiné i flashování SW. Tento mikrokontrolér je označován IDN. Je připojen přes rozhraní JTAG k hlavnímu mikrokontroléru a k periferiím (viz obrázek 6.2).

REMCS je koncipován jako master-slave systém. Jedna z karet funguje jako master a řídí přístup ostatních karet na sběrnici.



Obrázek 6.2: Diagnostický subsystém

6.2 Návrh protokolu

Při výběru protokolu přicházela v úvahu dvě řešení. Prvním z nich bylo použití jednoho z existujících diagnostických protokolů, druhým byl návrh protokolu vlastního.

V prvním případě by byl využit protokol UDS, protože nabízí dostatečnou úroveň zabezpečení a je standardizován. Jeho hlavní výhodou je to, že v současné době je používán většinou společností zabývajících se řídicími systémy v oblasti dopravní techniky a je široce podporován.

Pro realizaci ale byla zvolena možnost druhá – návrh protokolu vlastního. Protokol UDS totiž není pro použití v systému REMCS příliš vhodný pro jeho vysoký overhead (pokud by byl použit pouze pro flashování). Komunikace s REMCS probíhá výhradně po sběrnici CAN. Objem dat přenesených jednou CAN zprávou může být maximálně 8 bytů. Při použití protokolu UDS by byla minimálně polovina z celkového objemu komunikace použita na režii samotného protokolu.

6.2.1 CAN zprávy v systému REMCS

Systém REMCS využívá pro komunikaci rozšířené CAN zprávy podle protokolu CAN ve verzi 2.0B. Identifikátor zprávy tak má délku 29 bitů. Bylo rozhodnuto, že identifikátor bude obsahovat zdrojovou a cílovou adresou komunikujícího IDN. Struktura identifikátoru je na obrázku 6.3.

Význam	Message class (5 bit)					Message number (8 bit)								IDN receiver address (8 bit)								IDN sender address (8 bit)							
Bitová pozice	28	27	26	25	24	24	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Obrázek 6.3: Formát CAN Id v systému REMCS

Message class určuje třídu, do které zpráva podle svého účelu patří. Pro zprávy určené pro flashování byla vyhrazena *message class* $0x0C$.

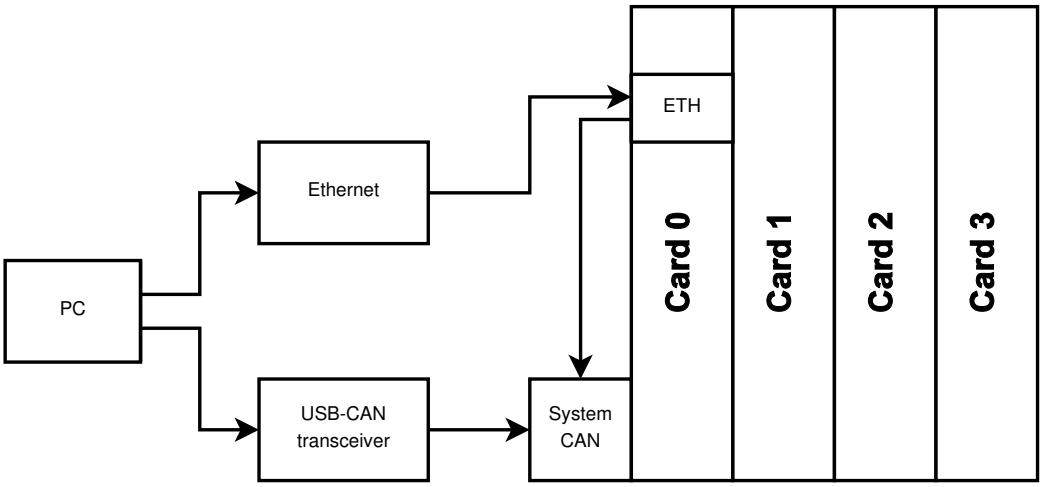
Message number určuje konkrétní typ zprávy uvnitř třídy.

IDN receiver obsahuje adresu příjemce zprávy. Adresa příjemce $0xFF$ značí, že zpráva je určená všem IDN na sběrnici (broadcast zpráva).

IDN sender je adresa odesílajícího IDN. Adresa odesílatele $0xEE$ je rezervována pro externí uzel, tedy například rozhraní pro připojení PC při flashování.

6.2.2 Koncept programovacího interface

PC je připojeno přes zvolené rozhraní k systémové sběrnici CAN jako externí uzel (adresa 0xEE). Komunikuje s IDN jednotlivých karet. Data jsou přenášena v binární podobě po blocích pevně zvolené délky. Program běžící v mikrokontroléru IDN ovládá diagnostickou sběrnici JTAG a zapisuje data do



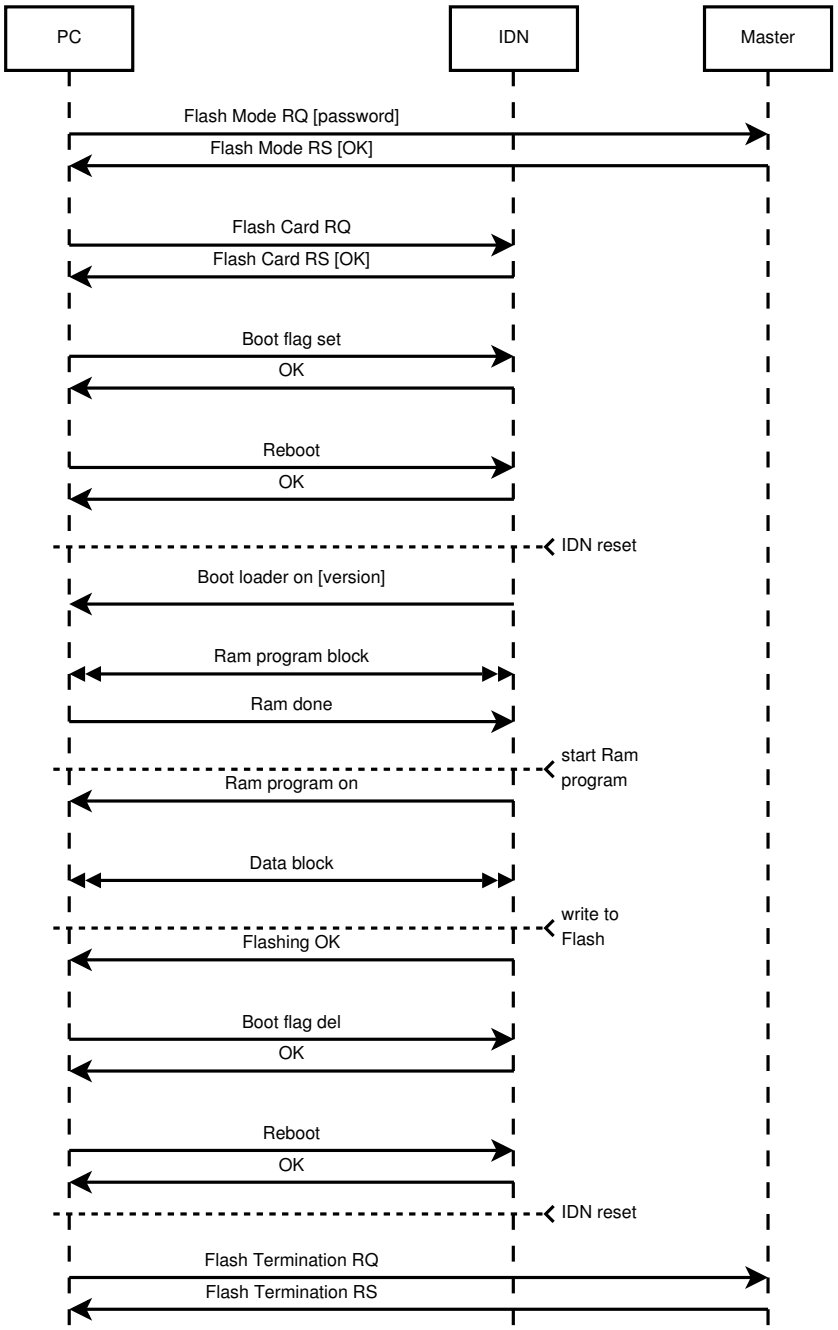
Obrázek 6.4: Komunikace po sběrnici CAN a po Ethernetu

zařízení k ní připojeným. V případě komunikace přes Ethernet je ethernetové rozhraní jedné z karet použito jako ETH-CAN převodník (viz obrázek 6.4).

Prvním krokem ve flashovacím procesu je žádost o povolení flashování od master karty. Aby nemohlo dojít k neoprávněné změně firmware, je vyžadována autorizace heslem. Master karta povolí přístup k systému. Následně flashovací rozhraní musí informovat příslušné karty, že bude probíhat flashování firmware. Třetím krokem je přechod karty do režimu boot loaderu — nastavením příznaku *boot_flag* a restartem karty. Nyní je karta připravena pro příjem dat. Následuje přenos Ram programu. Ten je přenášen jako jednolitý blok dat. Po úspěšném přenosu je Ram program spuštěn. Dále jsou postupně přenášeny jednotlivé bloky firmware a zapisovány do Flash paměti zařízení. Po dokončení přenosu všech bloků je karta nastavena do módu aplikace. Po naflashování všech karet je master informován o dokončení a flash mód je ukončen.

Sekvenční diagram celé komunikace je popsán na obrázku 6.5.

V následující části budou detailně popsány jednotlivé kroky procesu flashování systému REMCS. Struktura jednotlivých zpráv je podrobně popsána později.



Obrázek 6.5: Sekvenční diagram komunikace

Žádost o svolení k flashování

Flashující rozhraní musí nejprve získat svolení pro přístup k systému od master karty. Vzhledem k tomu, že rozhraní nezná adresu masteru, pošle všesměrovou žádost *Flash mode RQ*. Tento typ zprávy je přijímán pouze kartou nastavenou jako master. Obsahem zprávy je heslo pro přístup k systému. Karta odpoví zprávou *Flash mode RS*, která informuje klienta, zda byl přístup povolen.

Detekce přítomných karet

Karty v systému si mezi sebou vyměňují zprávy typu *IDN Card information*. Tyto zprávy slouží k informování ostatních karet (hlavně master karty která systém ovládá) o tom, že je karta přítomná v systému a co dělá. Zprávy obsahují informace o typu karty, o zařízeních k ní připojených a o verzi firmware nahraného na kartě. Klient může poslat zprávu *Card detail RQ*, na kterou IDN odpoví touto informační zprávou. Odesláním zprávy *Card detail RQ* na všesměrovou adresu je možné detekovat všechny karty v systému.

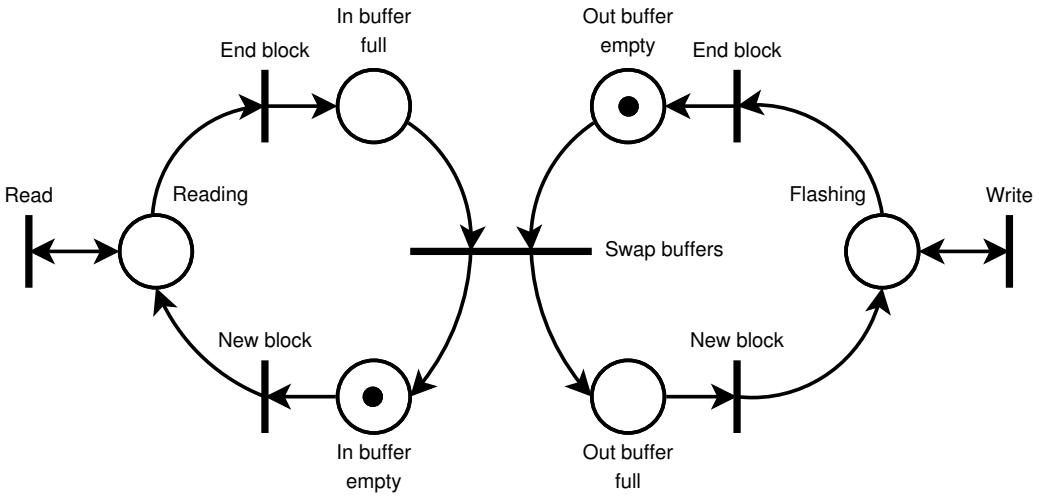
Boot loader, boot_flag

Po restartu karty se v IDN automaticky spouští boot loader. Jde o krátký program na začátku adresního prostoru. Boot loader slouží k inicializaci HW komponent zařízení a k provedení autodiagnostiky karty. Po dokončení práce boot loader předá činnost programu karty.

V případě, že je nastaven speciální příznak — *boot_flag* — boot loader nespustí program, ale přejde do módu terminálu a čeká na příchozí komunikaci. Tento příznak je nastaven zprávou *Bootloader flag set* a zrušen zprávou *Bootloader flag delete*.

Ram program

Prvním přeneseným blokem dat je Ram program (dále *RAMPRG*). *RAMPRG* obsahuje kód pro vlastní zápis dat do perzistentní paměti přes rozhraní *JTAG*. Důvodem pro oddělený *RAMPRG* je bezpečnost. V běžném provozu nesmí být v systému žádný kód, který by umožňoval zápis do paměti programu. Ani v případě chybné funkce tedy nemůže dojít k poškození dat v permanentní paměti. Po restartu se karta vždy musí dostat do nějakého konzistentního stavu.



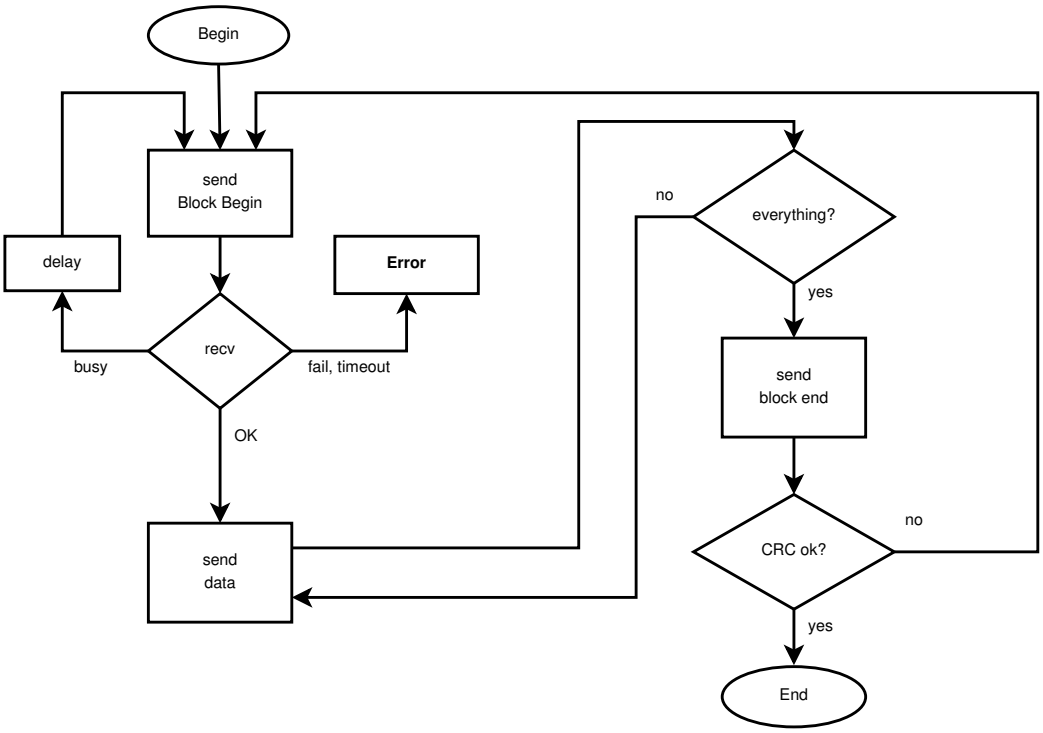
Obrázek 6.6: Činnost *RAMPRG*

Po svém spuštění *RAMPRG* čeká na příjem bloku dat. Po úspěšném přijetí bloku *RAMPRG* začne flashovat zařízení. Z důvodu efektivity *RAMPRG* používá metodu double-buffering (viz obrázek 6.6). *RAMPRG* obsahuje dva buffery. Vstupní buffer pro příjem dat přes CAN a výstupní buffer pro flashování bloku do permanentní paměti. V případě, že je vstupní buffer plný a výstupní buffer prázdný, dojde k výměně významů bufferů – ze vstupního se stane výstupní a naopak.

Blokový přenos dat

Data jsou přenášena po blocích pevné délky. Na začátku přenosu bloku je odeslána zpráva *Ram block start*. Tato zpráva informuje IDN o tom, že bude přenášet blok dat, pro jaké zařízení je blok určen a o počáteční adrese, od které má být blok zapsán. Možné odpovědí IDN na tuto zprávu jsou *IDN Flash OK* (přenos povolen), *IDN Flash Fail* (nastala nějaká chyba) a *IDN Flash Busy* (vstupní buffer IDN je plný).

Následuje přenos samotných dat. K tomu slouží zprávy *Ram block data*. Tyto zprávy kvůli efektivitě spojení (plných 8 bytů v každé zprávě) obsahují pouze data. IDN data zapíše do vstupního bufferu, žádnou odpověď negeneruje. Kontrola, že zpráva byla doručená úspěšně, je prováděna pouze podle ACK zprávy. PC tedy může tyto zprávy vysílat jednu za druhou.



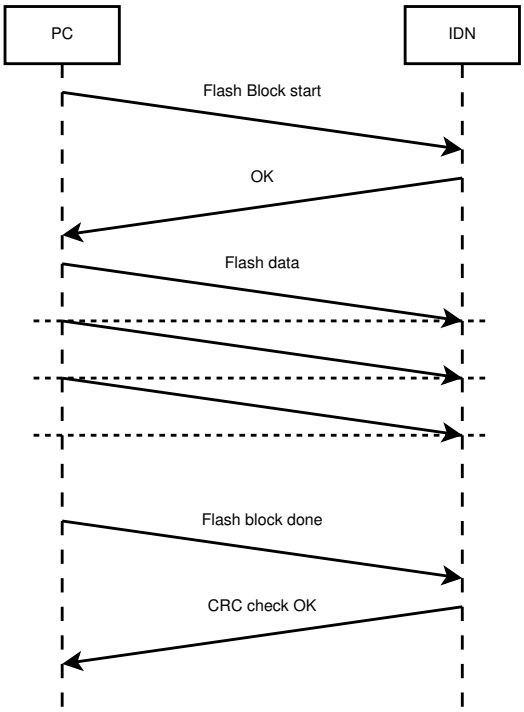
Obrázek 6.7: Blokový přenos dat

Na konci bloku je odeslána zpráva *Ram block stop*. Ta obsahuje kontrolní součet celého bloku dat. IDN zkontroluje CRC a pokud je platné, pošle potvrzení *IDN Flash OK*. V případě vadného bloku IDN pošle *IDN Flash Fail*. PC se pokusí celý blok poslat znovu.

Vývojový diagram přenosu bloku je vidět na obrázku 6.7 a sekvenční diagram na obrázku 6.8.

Ukončení flashování

Po dokončení flashování všech karet musí klient upozornit master kartu o tom, že má zablokovat přístup k systému. K tomu slouží zpráva *Flash mode termination RQ*. Protože klient nezná adresu master karty, je tato zpráva vysílána jako broadcast. Master odpoví zprávou *Flash mode termination RS*.



Obrázek 6.8: Přenos bloku dat

6.2.3 Použité zprávy

Následuje popis struktury jednotlivých zpráv a jejich užití.

Reboot

Jméno	Reboot	DLC	0
Message class	$0x0C$	Message number	$0x01$

Zpráva slouží k restartu karty. Může být odeslána pouze kartou master nebo externím uzlem. Zpráva neobsahuje žádná data.

Boot loader flag set

Jméno	Boot loader flag set	DLC	0
Message class	$0x0C$	Message number	$0x02$

Nastaví příznak *boot_flag*. Při příštím restartu se IDN zastaví v režimu boot loaderu.

Boot loader flag delete

Jméno	Boot loader flag delete	DLC	0
Message class	$0x0C$	Message number	$0x03$

Odstraní příznak *boot_flag*. Při příštím restartu IDN po provedení boot loaderu standardně přejde do režimu aplikace.

Flash mode request

Jméno	Flash mode RQ	DLC	8
Message class	$0x0C$	Message number	$0x04$

Data	7	6	5	4	3	2	1	0
0	Password [0]							
1	Password [1]							
2	Password [2]							
3	Password [3]							
4	Password [4]							
5	Password [5]							
6	Password [6]							
7	Password [7]							

Žádost o povolení k flashování celého systému. Tuto zprávu odesílá externí uzel jako broadcast (cílová adresa 0xFF). Zprávu přijímá pouze karta pracující jako master. Data zprávy obsahují heslo pro přístup k systému.

Flash mode response

Jméno	Flash mode RS	DLC	1
Message class	$0x0C$	Message number	$0xFB$

Data	7	6	5	4	3	2	1	0
0	Status							

Odpověď na zprávu *Flash mode RQ*. Podle toho jestli karta master povolila nebo zakázala flashování vrací následující hodnoty:

Status	Význam
0x01	Flashování bylo povoleno
0xFF	Flashování nebylo povoleno
0x55	Chybné heslo

Flash mode termination request

Jméno	Flash mode termination RQ	DLC	0
Message class	$0x0C$	Message number	$0x10$

Zpráva, kterou je karta master upozorněna na to, že flashování systému bylo dokončeno. Zpráva je vysílána od externího uzlu na broadcast. Na zprávu reaguje pouze karta v režimu master.

Flash mode termination response

Jméno	Flash mode termination RS	DLC	0
Message class	$0x0C$	Message number	$0xEF$

Odpověď na zprávu *Flash card RQ*. Karta je připravená k flashování dat.

Flash card request

Jméno	Flash card RQ	DLC	0
Message class	$0x0C$	Message number	$0x06$

Podobně jako *Flash mode RQ* slouží k požádání systému o povolení k flashování. V tomhle případě o flashování jedné karty.

Flash card response

Jméno	Flash card RS	DLC	0
Message class	$0x0C$	Message number	$0xF9$

Odpověď na zprávu *Flash card RQ*. Karta je připravená k flashování dat.

Ram block start

Jméno	Ram block start	DLC	5
Message class	$0x0C$	Message number	$0x08$

Data	7	6	5	4	3	2	1	0
0	Device identification							
1	Target address [0]							
2	Target address [1]							
3	Target address [2]							
4	Target address [3]							

Začátek bloku dat pro přenos. První byte dat slouží k identifikaci zařízení na kartě. Zbývající 4 byty určují počáteční adresu, od které se má blok dat

zapisovat. Adresa je přenášená od nejvýznamnějšího bytu po nejméně významný (big endian). V závislosti na stavu karty IDN odpoví zprávou *IDN Flash OK*, *IDN Flash Fail* nebo *IDN Flash Busy*.

Ram block stop

Jméno	Ram block stop	DLC	4
Message class	$0x0C$	Message number	$0x09$

Data	7	6	5	4	3	2	1	0
0	CRC32 [0]							
1	CRC32 [1]							
2	CRC32 [2]							
3	CRC32 [3]							

Konec bloku dat pro přenos. Data obsahují kontrolní součet dat bloku (CRC-32). Je použit stejný kontrolní polynom, který používá Ethernet, protože mikrokontrolér IDN má pro tento výpočet implementovanou HW jednotku.

Ram block data

Jméno	Ram block data	DLC	8
Message class	$0x0C$	Message number	$0x0A$

Data	7	6	5	4	3	2	1	0
0	Data [0]							
1	Data [1]							
2	Data [2]							
3	Data [3]							
4	Data [4]							
5	Data [5]							
6	Data [6]							
7	Data [7]							

Surová data. Data jsou zapisována do vstupního bufferu karty na narůstající adresy.

Ram done

Jméno	Ram done	DLC	4
Message class	$0x0C$	Message number	$0x0B$

Data	7	6	5	4	3	2	1	0
0	Address [0]							
1	Address [1]							
2	Address [2]							
3	Address [3]							

V případě, že právě přeneseným blokem dat byl strojový kód *RAMPRG*, PC užije tuto zprávu pro jeho spuštění. Data obsahují adresu na první instrukci *RAMPRG,* na kterou IDN provede nepodmíněný skok. Tato zpráva smí být použita pouze tehdy, když IDN pracuje v režimu boot loaderu.

IDN Flash OK

Jméno	IDN Flash OK	DLC	0
Message class	$0x10$	Message number	$0x10$

Potvrzení, že předchozí operace proběhla úspěšně.

IDN Flash Fail

Jméno	IDN Flash Fail	DLC	0
Message class	$0x10$	Message number	$0x11$

Předchozí operace byla neúspěšná. Program zkusí poslat příkaz znovu.

IDN Flash Busy

Jméno	IDN Flash Busy	DLC	0
Message class	$0x10$	Message number	$0x12$

Předchozí operace nebyla provedena, protože karta je zaneprázdněná. Program musí chvíli počkat a zkusit příkaz poslat znovu.

Card detail request

Jméno	Card detail RQ	DLC	0
Message class	$0x0C$	Message number	$0x05$

Program si vyžádá informace o kartě. Informace jsou vráceny ve zprávě *IDN Card information*. PC může vysláním tohoto požadavku jako broadcastu detekovat najednou všechny karty připojené do systému.

IDN Card information

Jméno	IDN Card information	DLC	8
Message class	$0x10$	Message number	$0x04$

Data	7	6	5	4	3	2	1	0
0	Card type							
1	Card variant							
2	Device list							
3	Firmware revision MSB							
4	Firmware revision LSB							
5	Reserved							
6	Reserved							
7	Reserved							

Položka Card type je výčtový typ různých druhů karet systému REMCS. V současnosti jsou definované tyto hodnoty:

Card type	Význam
$0x01$	BPL (backplane)
$0x02$	DIF
$0x03$	MCU
$0x04$	OIF

Pole Card variant informuje o tom, do které vývojové úrovně daná karta patří.

Card variant	Význam
$0x01$	Alfa
$0x02$	Beta
$0x03$	Gamma
$0x04$	RC (Release Candidate)
$0x05$	Release

Device list je bitové pole, které určuje, která zařízení jsou implementovaná na dané desce.

Device list	Význam
$0x01$	MMP (Master Microprocessor — Hercules)
$0x02$	FPGA
$0x04$	USB port
$0x08$	Ethernet port

Pole Firmware revision je 16 bitů dlouhé číslo, které odpovídá SVN revizi firmwaru nahranému na dané kartě.

Tuto zprávu odesílají IDN, aby byly ostatní karty informovány o jejich přítomnosti. Zpráva je automaticky vyslána při každém resetu karty. Flasho-

vací program si může o tyto zprávy explicitně zažádat zprávou *Card detail RQ*.

6.3 Vstupní soubory

Pro provedení práce byl požadavek na podporu vstupních souborů obsahujících program ve formátech Motorola S-Record a bin. Pro konfiguraci byl navržen soubor ve formátu XML. Tento formát byl zvolen z důvodu snadného čtení i generování programem.

6.3.1 Motorola S-Record

Také někdy nazývaný SREC nebo S19. Formát souboru, kterým je možné popsat binární data souborem v ASCII kódování. Soubor je tedy možné zobrazit jakýmkoliv textovým editorem.

Soubor se dělí na jednotlivé záznamy. Každý záznam je v souboru na samostatném řádku. Data jsou popsána v hexadecimální podobě. Každý byte dat je zapsán jako dvojice ASCII znaků. Znaky mohou nabývat hodnot '0'–'9' a 'A'–'F'. První znak v páru popisuje hodnotu horní 4 bity a druhý znak popisuje spodní 4 bity zapisovaného bytu.

Každý řádek začíná značkou, podle které se rozlišuje typ záznamu — S0–S9. Následující dva znaky obsahují délku zbytku záznamu v bytech. Další je adresa – podle typu záznamu má délku 4, 6 nebo 8 bytů a je zapsaná ve formátu big endian. Následují samotná data. Na konci záznamu je ještě 2 byty dlouhý kontrolní součet — binární součet jednotlivých bytů délky, adresy a dat omezený na dvě hexadecimální číslice.

Typy záznamů jsou následující:

- S0 — Informace o datech - jméno souboru, verze, ...
- S1, S2, S3 — Datová sekvence s adresou dlouhou 16, 24, resp. 32 bitů.
- S5, S6 — Informují o počtu datových záznamů, které se doposud v souboru vyskytly. Počet je umístěn v adresním poli dlouhém 16, resp. 24 bitů.
- S7, S8, S9 — Počáteční adresa programu. Adresa je dlouhá 32, 24, resp. 16 bitů.

6.3.2 Bin

Soubor typu *bin* obsahuje binární data. Vzhledem k tomu, že *bin* obsahuje jen čistá data, je požadována ještě adresa, od které mají být data zapsána. V případě, že se jedná o strojový kód, který má být spuštěn (RAM program), musí být ještě určena adresa první instrukce programu.

6.3.3 Načítání souborů

V programu jsou všechny typy souborů firmware převáděny do binární podoby rozdělené do stránek zvolené velikosti. K tomu je určená třída *Reader*. Funkce pro čtení jednotlivých formátů vstupních dat přijímají zvolený *Reader* jako výstupní argument a zapisují do něj načtená data. Jeden *Reader* může postupně přijmout data z několika různých souborů. Klient tak může například flashovat datové a programové segmenty zvlášť. Pokud více souborů zapisuje do stejného úseku paměti, budou zapsána data z posledního načteného souboru. Jeden *Reader* udržuje data, která budou přenesena do jednoho konkrétního zařízení na jedné kartě.

6.3.4 Konfigurační soubor flashování

Proces flashování je řízen konfiguračním souborem ve formátu XML. Soubor obsahuje informace o kartách připojených do systému, o zařízeních použitých na kartách. Dále obsahuje cesty k souborům, které mají být do zařízení zapsané.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <flash-tool>
3    <ram-program jmp-address="0xAB001F">
4      <file src="ramprg.bin" type="bin" address="0xAB001F"/>
5    </ram-program>
6    <card id="0x0" name="MCU">
7      <device id="0x0" name="MMU">
8        <file src="mmu.s19" type="s19"/>
9      </device>
10     <device id="0x1" name="FPGA">
11       <file src="mmu.bin" type="bin" address="0xFA0000"/>
12     </device>
13     <device id="0x4" name="ETH"/>
14   </card>
15 </flash-tool>
```

Kořenový element souboru je nazvaný *flash-tool*. Potomky uzlu *flash-tool* musí být právě jeden element *ram-program* a libovolný počet elementů *card*.

Tag *ram-program* popisuje *RAMPRG*, který má být při flashování použit. Má jeden volitelný argument *jmp-address*, který obsahuje adresu první instrukce *RAMPRG*. Tento argument je potřeba, pokud je *RAMPRG* dodáván jako soubor typu *bin*. Soubor ve formátu *s19* má tuto adresu uloženou v sobě. Tag *ram-program* musí obsahovat jediný soubor.

Element *card* má povinné argumenty *id* a *name*. Argument *id* obsahuje unikátní adresu karty. Je to číslo délky 1 byte zapsané v hexadecimální soustavě. Argument *name* souží pro její krátký popis, podle kterého je možné karty rozlišovat. Potomky elementu card jsou žádný nebo více elementů *device*.

Obdobně jako *card* i uzel *device* požaduje argumenty *id* a *name*. Jejich formát a význam je shodný. Tag *device* obsahuje seznam souborů (tagů *file*), které mají být na dané zařízení nahrané.

Uzel *file* požaduje argument *src*, který udává absolutní nebo relativní cestu k souboru firmware. Druhým povinným argumentem je *type* — formát v jakém je soubor uložen. V současnosti jsou podporovány hodnoty „bin“ pro binární soubory a „s19“ pro soubory ve formátu Motorola S-Record. Pro soubory typu bin je navíc potřeba parametr *address*, ve kterém je uložena adresa, od které se mají data do paměti zapisovat.

6.4 Komunikace přes Ethernet

Pro návrh řešení komunikace pro rozhraní Ethernet bylo potřeba vybrat síťové protokoly. Na síťové vrstvě byla situace jasná, byl zvolen protokol IP. Na transportní úrovni přicházely v úvahu protokoly dva — UDP a TCP.

UDP nabízí nespojované služby bez zajištění spolehlivosti. Vzhledem k tomu, že neřeší spolehlivost, má menší režii a je rychlejší. Naproti tomu TCP nabízí zabezpečení spolehlivosti přenosu za cenu jeho nižší rychlosti přenosu.

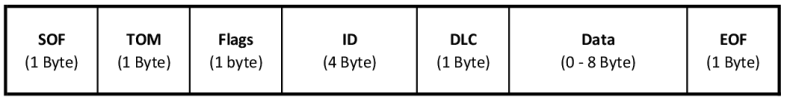
Pro návrh byl zvolen protokol TCP. Vzhledem k velikosti dat přenášených mezi PC s kartami v systému REMCS není faktor rychlosti problémem, samotný přenos po CAN sběrnici je řádově pomalejší než Ethernet. Stejně tak čas potřebný pro operaci zápisu do flash paměti je podstatně vyšší.

Na aplikační úrovni bylo rozhodnuto, že bude vytvořen vlastní protokol. Jednotlivé CAN zprávy budou odesílány v samostatných TCP zprávách. Posíláním krátkých zpráv sice roste režie protokolu, ale vzhledem k rychlosti Ethernetu je zatížení LAN nevýznamné. Přináší to s sebou menší HW nároky na adaptér ETH-CAN. Složitost SW adaptéru bude také nižší.

Přenášena musí být ID, DLC a datový obsah CAN zpráv. Vše ostatní může být vytvořeno na straně převodníku při vytváření CAN zprávy.

Výsledná realizace dosud nebyla provedena, proto nebylo možné navržené řešení vyzkoušet. Předpokládá se však, že navrhované řešení by mělo být funkční a zcela splňovat požadavky systému na robustnost a rychlost.

Navržené zprávy mají formát popsaný na obrázku 6.9.



Obrázek 6.9: Struktura CAN zprávy přenášené přes TCP/IP

- SOF (Start of Frame) — položka identifikuje použitý protokol.
- TOM (Type of Message) — význam této zprávy.
- Flags — řídící příznaky.
- ID — identifikátor CAN zprávy.
- DLC — délka dat v bytech.
- Data — data přenášené zprávy.
- EOF (End of Frame) — značka konce přenášené zprávy.

Pro pole SOF bude nastavena hodnota $0xEC$, hodnotou pole EOF bude $0xCE$.

Položka TOM může nabývat následujících hodnot:

TOM	Význam	Směr
$0xAC$	Zpráva z PC na sběrnici CAN	PC → CAN
$0xCA$	Zpráva ze sběrnice CAN do PC	CAN → PC
$0xCF$	Buffer je plný	CAN → PC
$0xC0$	Zpráva přijata do bufferu	CAN → PC
$0x05$	ACK přijat	CAN → PC

Flags je bitové pole, které nastavuje příznaky posílané zprávy.

Bit	7	6	5	4	3	2	1	0
Význam	nepoužité						RTR	IDE

Pole ID obsahuje identifikátor CAN zprávy, ke které se zpráva vztahuje. Jedná se o 32bitové číslo v síťovém formátu, tj. big endian.

Význam a formát položek DLC, IDE, RTR a Data je stejný jako v případě CAN zprávy.

7 Popis implementace

V této části jsou popsány významné funkce a datové struktury, které byly vytvořeny při implementaci programu. Kompletní programová dokumentace byla vytvořena nástrojem `doxygen` a je dostupná v elektronické podobě na přiloženém médiu.

Aplikace je napsaná v programovacím jazyce C++. Program využívá vlastnosti objektově orientovaného programování. Byl primárně vyvíjen pro operační systém MS Windows, ale nebyly použity žádné metody, které by omezovaly přenositelnost na jiné operační systémy. Pro tvorbu GUI byl zvolen nástroj `wxFormBuilder`.

Při vývoji byl použit překladač gcc ve verzi 4.7.2 distribuovaný v balíku mingw. Aplikace využívá knihovnu libXML2 (použitá verze je 2.7.8 — aktuální verze v době psaní této práce) pro přístup ke XML zdrojům. Ta dále závisí na knihovnách iconv a zlib. Pro tvorbu GUI používá aplikace knihovnu wxWidgets. Použitá verze wxWidgets je 2.8.12, knihovna je staticky linkovaná.

7.1 Rozhraní pro přístup ke sběrnici CAN

Z důvodu modularity byly ovladače k zařízením k přístupu na CAN sběrnici opatřeny společným rozhraním. Jednotlivé ovladače jsou přeloženy samostatně jako dynamické knihovny, soubory .dll v OS Windows a soubory .so v UNIXových operačních systémech (v dalším textu jsou oba typy knihoven nazývány souhrnně DLL). Knihovny DLL se k programu připojují za běhu jako zásuvné moduly.

Společné komunikační rozhraní je vytvořené jako třída, od které jsou jednotlivé implementace odvozeny použitím dědičnosti. Vzhledem k tomu, aby byly vytvořené DLL přenositelné mezi různými překladači C++, je na společné rozhraní kladeno velké množství požadavků. Samotná třída rozhraní je totiž překládaná dvakrát, jednou při překladu připojované DLL a jednou při překladu aplikace, ke kterému je knihovna připojována. Je nutné, aby se při překladu zachovalo stejné ABI (Application Binary Interface). Problémy s kompatibilitou ABI způsobuje nespecifický předpis, jak má být organizován přeložený kód. Typickými problémy jsou různé dekorace názvů funkcí pro linker (name mangling), různá organizace datových položek v paměti (aligning) nebo odlišná struktura tabulky virtuálních metod (vtable).

Name mangling je vyřešen tak, že jediná funkce, která je exportovaná z knihovny — create_interface — má nastavené dekorování jazyka C (tj.

žádné). Veškerá další funkcionalita je zajištěna virtuálními metodami. Bez ohledu na to, jaký používá překladač name mangling, budou ve výsledném kódu metody volány pouze jako položky v tabulce virtuálních metod. Třída CANInterface je ryze abstraktní, všechny její metody jsou ryze virtuální. Jediný kód, který třída rozhraní obsahuje, jsou překladačem automaticky vytvořené konstruktory a destruktor, které v našem případě *nesmí* vykonávat žádnou aktivitu.

Problém různého zarovnávání datových položek je vyřešen tak, že třída CANInterface prostě žádné datové položky nemá.

Pokud by dva různé překladače C++ generovaly objekty s nekompatibilní tabulkou virtuálních metod, nebylo by možné, aby spolu takové třídy komunikovaly. Na platformě Windows je kompatibilita zajištěna standardy technologie COM (Component Object Model), která by jinak nemohla fungovat. Univerzální zajištění kompatibility však neexistuje.

Provádění funkce v DLL nesmí za žádných okolností skončit výjimkou. Vzhledem ke sdílené povaze knihovny nesmí být paměť alokovaná knihovnou (použitím *malloc* nebo *new*) uvolněna mimo knihovnu.

Interface poskytuje následující globální funkci:

```
extern "C" __declspec(dllexport) CANInterface* __cdecl
    create_interface();
```

Za poněkud komplikovanou deklarací se skrývá funkce, která vytvoří instanci třídy implementující rozhraní CANInterface a vrátí ukazatel na tuto instanci. Funkce je deklarovaná extern „C“, aby nedošlo k deformaci názvu funkce (name mangling). Funkce je exportována z DLL knihovny.

Třída CANInterface poskytuje následující metody:

```
virtual connection_state status();
```

Metoda vrátí stav připojení ke sběrnici CAN. V případě odpojitelných rozhraní (USB-CAN transceiver) také zjistí, zda je potřebný hardware připojen k PC.

```
virtual status_type recv_blocking(const Filter *f,
    CANMessage &message, int timeout);
```

Blokující příjem zpráv. Prvním parametrem je filtr, který nastavuje, který identifikátor nebo skupinu identifikátorů CAN zpráv má aplikace přijmout. Druhý parametr je výstupní a předává přijaté zprávy aplikaci. Poslední parametr určuje maximální dobu čekání.

```
virtual status_type send_blocking(const CANMessage&
    message, int timeout);
```

Blokující odesílání zpráv. Metoda odešle zprávu na CAN sběrnici a počká, dokud nepřijme potvrzení o úspěšném doručení.

```
virtual status_type set_param(param_meaning meaning,
    param_type type, size_t length, const void *value);
```

Nastavení parametrů spojení. Které parametry rozhraní nabízí, záleží na konkrétní implementaci. Výčtovým typem v prvním argumentu funkce je vybráno k jakému účelu nastavený parametr slouží. Zbývající argumenty popisují samotný parametr ve formátu TLV (tj. Type-Length-Value). Implementace rozhraní přijímá pouze parametry jí určené a ignoruje ostatní. Připojení je možné pouze pokud jsou nastavené všechny potřebné parametry.

```
virtual status_type connect();
```

Metoda se pokusí připojit k rozhraní. Pokud nejsou nastavené potřebné parametry, metoda skončí neúspěšně.

```
virtual void destroy();
```

Explicitní metoda pro odpojení od interface a dealokaci prostředků. Funguje jako obdoba virtuálního destruktoru.

7.1.1 Struktura Filter

Struktura *Filter* slouží k vyčlenění zpráv, které jsou v daném okamžiku důležité. Jedná se o spojový seznam položek. Každá položka obsahuje ID zprávy a masku. Přijatá CAN zpráva vyhovuje dané položce filtru, pokud se ID zprávy a ID filtru shodují ve všech bitech, ve kterých je v masce bit s hodnotou „1“. Filtr s maskou rovnou 0 přijme jakoukoli zprávu, filtr s maskou se samými jedničkami přijímá pouze zprávy s přesně se shodujícím ID. V případě, že se ve spojovém seznamu vyskytuje více filtrů, budou přijaty zprávy odpovídající kterémukoli z nich.

7.1.2 Struktura CANMessage

Tato struktura slouží k uchování jedné CAN zprávy. Struktura obsahuje id zprávy, délku přenášených dat a data samotná. Dále je uchováván příznak pro rozlišení základního a rozšířeného formátu CAN zprávy a příznak pro rozlišení zda se jedná o datovou zprávu nebo žádost o data (*RTR*).

7.1.3 CANCommon

Třída *CANCommon* definuje metody, které jsou společné pro všechny implementace rozhraní. Tato třída je zodpovědná za správu bufferu příchozích

zpráv a za kontrolu, zda byly odchozí zprávy úspěšně doručeny (kontrola ACK).

7.1.4 USB-CAN rozhraní

Pro testování byl použit HW od firmy Imfsoft (USB-CAN Triple drivers V4.5). Pro komunikaci s PC zařízení obsahuje čip FTDI (FT245RL). To je převodník z USB portu na sériové rozhraní UART. Komunikaci přes směrnici CAN řídí procesor Atmel T89C51CC01 připojený k budičům signálu. Pro bližší informace viz [6].

Tento adaptér byl zvolen pro svou nízkou cenu. Dále z důvodu, že zadavatel již předem několik exemplářů vlastnil. Zařízení je plánováno pro použití jako diagnosticý nástroj. Jako prostředek pro přenos většího objemu dat, kdy je požadováno potvrzování a ověřování komunikace s důrazem na rychlost, se ukázal jako nevhodný. Implicitně převodník posílá data do PC v cyklu 16 ms. Ručním nastavením vlastností převodníku FTDI je možné latenci snížit na nejméně 2 ms. Toto způsobí zvýšení rychlosti komunikace mezi FTDI a počítačem. Firmware v procesoru Atmel s tímto zřejmě nepočítá. Z nezjištěného důvodu převodník stále posílá významná data s periodou 16ms a zbývající kapacitu USB přenosu nevyužívá. Použitím jiného adaptéru a pravděpodobně i úpravou firmware adaptéru současného by se situace zlepšila.

Dalšího urychlení by bylo možné dosáhnout úpravou protokolu komunikace — přidáním sekvenčního čísla do ID datových zpráv. Převodník disponuje s osmi paralelními message centry. Každé z message center je schopné nezávisle odesílat jednu zprávu. Pořadí odesílání zpráv je řízeno prioritou podle protokolu CAN. Je tedy možné předat adaptéru současně až 8 zpráv. Není však zaručeno, že tyto zprávy budou doručeny ve stejném pořadí v jakém byly odeslány.

Vzhledem k přidělenému rozsahu ID by sekvenční číslo mělo délku 4 bity. Pro přenos by byla použila metoda klouzajícího okénka.

Metoda klouzajícího okénka je často využívána v počítačových sítích. Umožňuje posílání několika zpráv najednou bez porušení sekvenčnosti. Každá zpráva obsahuje sekvenční číslo. Je definována velikost okénka — počet zpráv, které mohou být najednou odeslány. Příjemce akceptuje příchozí zprávu pouze pokud její sekvenční číslo spadá mezi hodnoty uvnitř okénka. Metoda klouzajícího okénka je detailně popsána v libovolné literatuře popisující architekturu počítačových sítí, viz např. [9].

Rozhraní je řízeno funkcemi v souboru CANxx.c. Jedná se o modifikovanou verzi ovladače poskytovaného výrobcem HW. Rozhraní ovladače je obaleno funkcemi třídy USB-CAN, aby byla zajištěna kompatibilita s jinými rozhraními.

7.2 Funkce pro flashování karet

Vlastní interface pro programování karet se skládá ze dvou tříd. Třída FlashCard slouží k nahrávání dat do všech zařízení připojených ke konkrétní kartě, tzn. k jednomu IDN. Instance třídy FlashCard přijme metodou *add_pages* stránky určené pro danou kartu. Metodou *apply* je spuštěn vlastní flashovací proces. Třída FlashSystem provede dávkově flashování všech karet v systému.

7.3 Přístup k XML konfiguraci

Pro přístup ke XML datům program používá knihovnu *libXML2*. Knihovna zajišťuje načítání dat z XML souboru (viz sekce 6.3.4) , validaci formátu dat podle XSD stylu i zpětný zápis dat do souboru. Přístup k XML datům je prováděn metodou DOM stromu. Metody pro načítání i zápis XML souboru jsou definované ve třídě XMLConfig.

7.4 GUI

Pro grafické uživatelské rozhraní byla použitá platformně nezávislá knihovna wxWidgets. Hojně byla využitá technologie XRC. Rozložení ovládacích prvků jednotlivých oken aplikace je nastaveno v samostatném XML souboru a za běhu programu je tento soubor načítán. Proto není pro změnu vzhledu GUI potřeba opětná kompilace programu, a dokonce ani přístup ke zdrojovému kódu.

7.5 Překlad projektu

Pro automatický překlad projektu je použit program `make`. V adresáři se zdrojovými kódy jsou soubory *Makefile* a *include.mk*. Soubor *Makefile* řídí samotný překlad, soubor *include.mk* definuje cesty k potřebným knihovnám na lokálním stroji. V případě, že jsou knihovny nainstalované na jiném místě, je potřeba soubor *include.mk* editovat.

Sdílenou knihovnu je možné přeložit samostatně příkazem `make dll`. Plné GUI může být přeloženo příkazem `make flashing-tool`, distribuční GUI příkazem `make flashing-tool-lite`. Spuštěním nástroje `make` bez parametrů se provede překlad obou částí projektu.

7.6 Instalace a spuštění

V adresáři programu jsou potřeba následující soubory:

1. Používané sdílené knihovny — lze je zdarma stáhnout z internetu:
   - libxml2.dll
   - iconv.dll
   - zlib1.dll
   - libgcc_s_dw2-1.dll
   - libstdc++-6.dll
2. Konfigurační soubory GUI:
   - edit_device.xrc
   - FlashTool.xrc
   - FlashToolLite.xrc
   - hex_dialog.xrc
   - menu.xrc
   - usb_can_dialog.xrc
3. Soubory komunikačních knihoven — v současnosti pouze USB_CAN.dll. Pro její funkci je dále potřeba nainstalovat ovladače ze stránek výrobce HW.
4. Spustitelné soubory
   - FlashTool.exe
   - FlashToolLite.exe

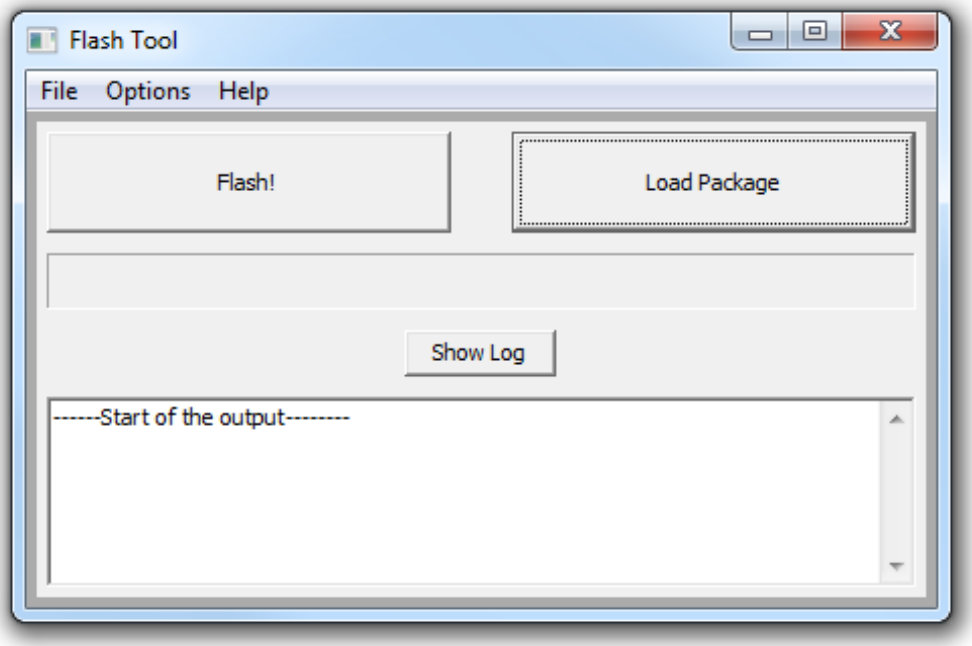
Pro spuštění komplentního GUI souží soubor `FlashTool.exe`. Soubor `FlashToolLite.exe` spouští distribuční GUI.

8 Uživatelský manuál GUI

Požadavkem práce bylo vytvoření dvou různých grafických uživatelských rozhraní.

8.1 Distribuční GUI

Distribuční GUI (viz obrázek 8.1) je určené pro servisní techniky. Jedná se o minimalistické rozhraní, které je shopný ovládat kdokoliv, aniž by měl větší povědomí o řídícím systému. Toto GUI umožňuje pouze naflashovat předem připravený distribuční balíček. Balíček (řídící XML soubor) je možné zvolit stiskem tlačítka *Load Package*. Flashování je zahájeno stiskem tlačítka *Flash!*. Tlačítko Flash! je povolené pouze v případě že byla připojená komunikační knihovna z nabídky File (viz dále).

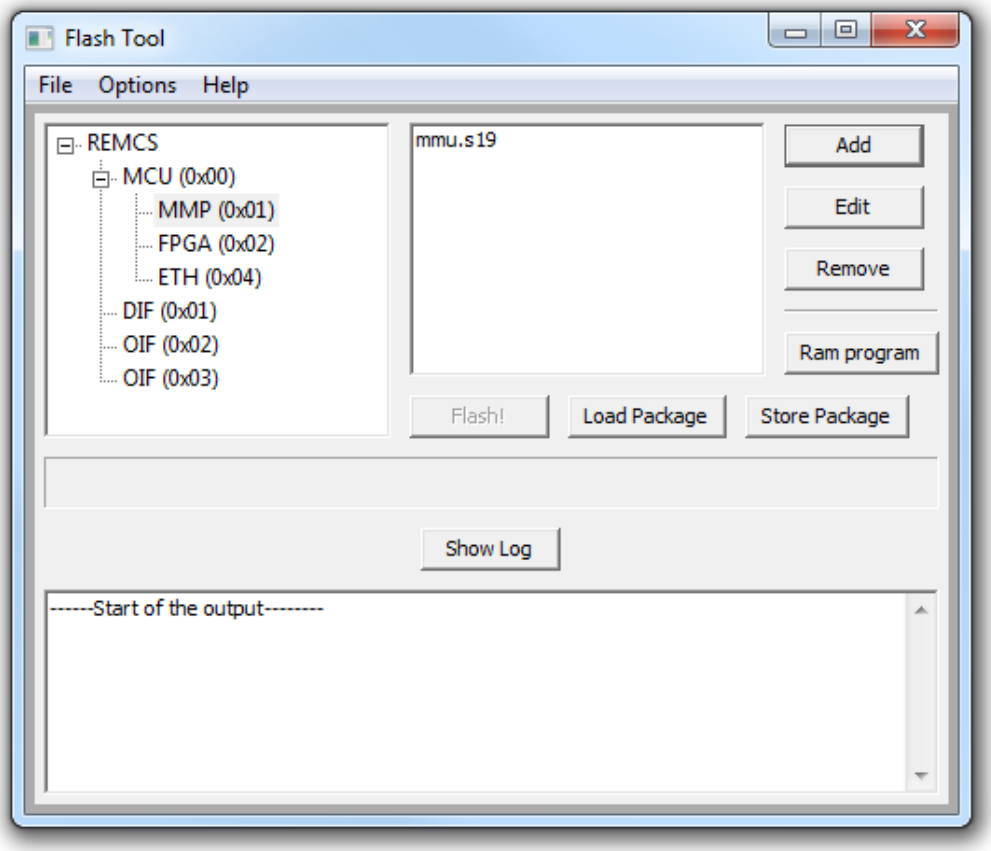


Obrázek 8.1: Distribuční GUI

8.2 Plné GUI

Plné GUI (obrázek 8.2) slouží pro vývojáře systému. Umožňuje graficky zobrazit strukturu připojeného systému. Toto GUI umožňuje vytváření balíčků

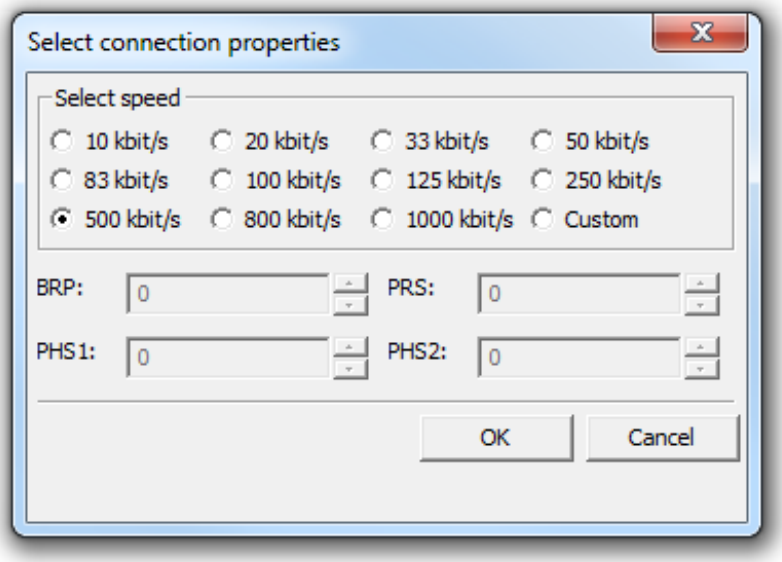
pro distribuci.



Obrázek 8.2: Plné GUI

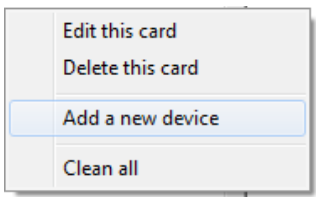
Společnými prvky obou verzí GUI jsou aplikační menu v horní liště okna, tlačítka pro flashování a práci s balíčky, indikátor průběhu flashování a textové pole, do kterého je vypisován záznam prováděných akcí (Log).

Menu File obsahuje položky pro připojení a odpojení knihovny CAN rozhraní. V současné verzi je dostupná pouze knihovna pro USB-CAN transceiver od firmy Imfsoft. Klepnutím na příslušnou položku menu se otevře dialog pro nastavení parametrů rozhraní. Vzhled dialogu je vidět na obrázku 8.3.



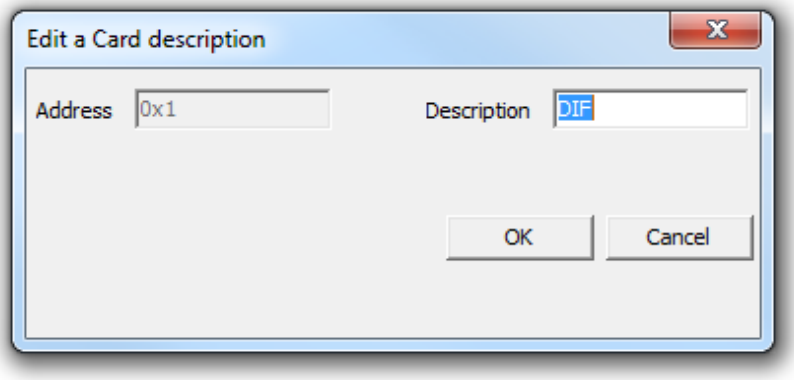
Obrázek 8.3: Nastavení parametrů rozhraní

V levém horním rohu plného GUI je panel, který zobrazuje strukturu řídícího systému. Systém je popsán jako strom. Kořenem stromu je uzel nazvaný REMCS. Jeho poduzly jsou jednotlivé karty. Poduzly karet jsou zařízení přítomná na kartě. Stiskem pravého tlačítka myši na některém z uzlů se otevře kontextové menu, jaké je vidět na obrázku 8.4. Položky kontextového menu jsou závislé na tom, zda byl označen kořen stomu, karta nebo zařízení.



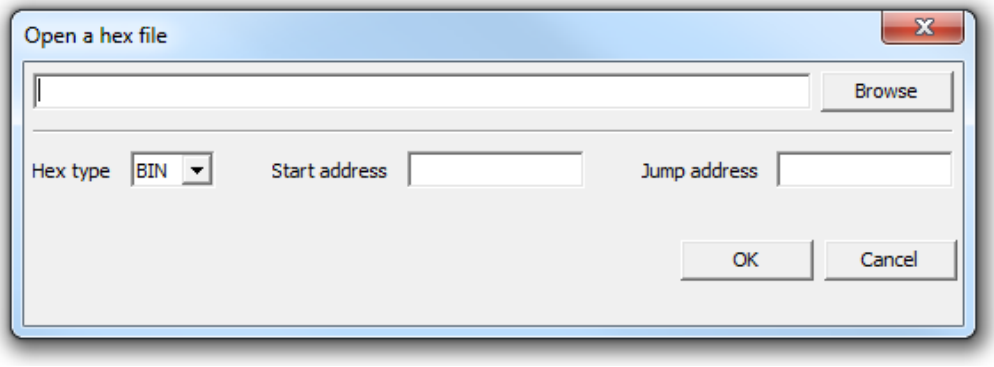
Obrázek 8.4: Kontextové menu karty

Uživatel má možnost přidávat či odebírat karty i zařízení a měnit jim jejich jmenovky. Položka Clean all vyprázdní celý strom. Příklad dialogu pro změnu jmenovky karty je znázorněn na obrázku 8.5. V případě přidávání nové karty, popř. zařízení, je potřeba nastavit unikátní adresu (dvě hexadecimální číslice) a stručnou popisku, podle které může operátor kartu identifikovat. Popisku lze kdykoli editovat, adresa je nastavena pevně.



Obrázek 8.5: Editace jmenovky karty

Napravo od stromu zařízení se nachází seznam souborů, které mají být naflashované na označené zařízení. Vpravo a pod seznamem souborů se nachází panely ovládacích tlačítek. Tlačítka Add, Edit a Remove umožňují přidat, upravit, resp. odebrat soubor, který má být naflashován na označené zařízení. Tlačítko Ram program souží k nastavení souboru s RAMPRG. Soubory HEX jsou nastavované dialogem na obrázku 8.6.



Obrázek 8.6: Dialog pro otevření hex souboru

V horní části dialogového okna je pole pro nastavení souboru. Pod ním jsou pole pro nastavení typu vstupního souboru (zatím pouze bin a s19), počáteční adresu souboru a adresu začátku spustitelného kódu. Adresy jsou zapsány jako hexadecimální čísla délky 32 bitů.

9 Závěr

Tato bakalářská práce se zabývala vytvořením programového vybavení sloužícího k flashování dat do jednotek řídicího systému REMCS.

Nejprve byla provedena analýza existujících průmyslových protokolů pro flashování dat přes CAN a Ethernet. Zvažovalo se užití existujícího protokolu UDS. Jako vhodnější řešení se však díky vysoké režii UDS ukázalo použití vlastního, nově definovaného protokolu.Výsledný protokol byl definován s důrazem na bezpečnost a robustnost vzhledem k plánovanému nasazení.

V další části bakalářské práce byl navržen protokol pro komunikaci přes sběrnici CAN. V následující kapitole byl tento návrh implementován. Realizace byla otestovaná s USB-CAN převodníkem od firmy Imfsoft. Výsledky testování potvrdily, že navržené řešení je funkční. Pro další použití by nicméně bylo z důvodu nižší rychlosti vhodnější použít jiný hardware. Software je na to plně připraven. Vhodným pokračováním této práce by byla implementace knihoven pro jiná USB rozhraní, která by mohla být následně použita v praxi.

Pro komunikaci přes Ethernet bude využit ETH-CAN adaptér. V práci byl vytvořen síťový protokol postavený na aplikační vrstvě TCP/IP popisující komunikaci mezi PC a adaptérem ETH-CAN. Vzhledem ke skutečnosti, že převodník v době odevzdání ještě nebyl zadavatelem práce implementován, ověření této části návrhu nebylo možné. Bude provedeno v dalších krocích dle plánu vývoje systému REMCS.

V druhé části práce byla vyvinuta aplikace pro PC, která implementuje navržený komunikační protokol. Aplikace poskytuje dvě různá grafická uživatelská rozhraní. Plné GUI je určené pro vývojáře systému. Toto GUI umožňuje zobrazit strukturu připojeného systému a vytvářet distribuční balíčky. Distribuční GUI je určené servisním technikům. Toto GUI umožňuje pouze naflashování předem připraveného distribučního balíčku do systému.

Pro snadnou údržbu byl veškerý zdrojový kód komentován tak, aby bylo možné vygenerovat programátorskou dokumentaci pomocí nástroje doxygen.

Posledním bodem zadání této práce bylo vytvoření knihovny pro mikrokontrolér implementující navržený protokol. V rámci vývoje systému REMCS bylo rozhodnuto, že tuto část projektu vzhledem ke komplikovanému řešení flashovacích rutin a vysokým nárokům na provedení výsledného kódu vytvoří vývojový tým systému REMCS.

Seznam použitých zkratek

ABI Application Binary Interface

ACK Acknowledgement — potvrzení přijetí zprávy

BPL Backplane — Spojovací karta systému REMCS

CAN Controller Area Network (viz kapitolu 2)

CRC Cyclic Redundancy Check

DIF Direct Interface — typ karty systému REMCS

DLL Dynamic-link library — dynamicky linkovaná knihovna

ETH Ethernet (viz kapitolu 3)

GUI Graphic User Interface — Grafické uživatelské rozhraní

HW Hardware

IDN Diagnostický mikrokontrolér karet REMCS.

JTAG Join Test Action Group — diagnostická sběrnice

LSB Least Significant Bit — nejméně významný bit.

MCU Microcontroller Unit — typ karty systému REMCS

MOSFET Metal Oxide Semiconductor Field Effect Transistor

MSB Most Significant Bit — nejvýznamější bit.

OIF Open Interface — typ karty systému REMCS

RAMPRG Ram program — program zajišťující zápis dat do Flash paměti.

REMCS RICE Embedded Modualar Control System

RICE Regionální inovační centrum elektrotechniky

SVN Subversion — systém pro správu a verzování zdrojových kódů

SW Software

UDS Unified Diagnostic Services

XRC XML Based Resource System — Součást wxWidgets. Systém, který umožňuje popsat GUI souborem ve formátu XML.

Literatura

[1] *Dokumentace systému Doxygen*, 2014. URL: http://www.stack.nl/~dimitri/doxygen/manual/index.html.

[2] Atmel. *Atmel T89C51CC01 datasheet*, 2008. URL: http://www.atmel.com/Images/doc4129.pdf.

[3] Future Technology Devices International Ltd. (FTDI). *FT245RL Datasheet*, 2011. URL: http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT245R.pdf.

[4] Future Technology Devices International Ltd. (FTDI). *D2XX Programmer's Guide*, 2012. URL: http://www.ftdichip.com/Support/Documents/ProgramGuides/D2XX_Programmer's_Guide(FT_000071).pdf.

[5] Pavel Herout. *Učebnice jazyka C*, 2004.

[6] Imfsoft. *Dokumentace USB-CAN adaptéru Imfsoft Triple drivers*, 2008. URL: http://imfsoft.com/hardware/usb-can-adapter-triple-drivers-v4-2-high-low-one-wire.

[7] ISO. *ISO 14229 — Unified diagnostic services (UDS) — Specification and requirements*, 2006.

[8] Robert Bosch GmBH. *CAN Specification*, 1991. URL: http://www.bosch-semiconductors.de/media/pdf_1/canliteratur/can2spec.pdf.

[9] Andrew S. Tanenbaum. *Computer Networks 4th Edition.*

[10] Pavel Turjanica. *Technická specifikace systému REMCS*, 2014.

[11] Daniel Veillard. *Dokumentace knihovny LibXML2*, 2014. URL: http://xmlsoft.org/html/index.html.

[12] Josef Štengl. *REMCS CAN message list*, 2014.

A Flashování službami protokolu UDS