VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV AUTOMATIZACE A INFORMATIKY

FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

POLOHOVÁ REGULACE MOTORU POMOCÍ 8BIT UP

POSITIONAL MOTION CONTROL BY MEANS OF 8BIT UP

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. PETR JANDA
AUTHOR

VEDOUCÍ PRÁCE Ing. RADOMIL MATOUŠEK, Ph.D.
SUPERVISOR

BRNO 2010

ABSTRAKT

Práce se zabývá polohovým řízením DC motoru Icla D065 firmy SIG Positec Automation GmbH pomocí sběrnice CAN a protokolu CANopen. Jednotlivé části popisují obecné principy a standardy sběrnice CAN a mezinárodně normovaného vysokoúrovňového protokolu pro ovládání systémů CANopen. Pro řízení výše uvedeného motoru byl použit mikrořadič firmy Microchip PIC18F4685 a vývojové prostředí MPLAB® IDE. S využitím programovacího jazyka C byl vytvořen program pro MCU řídící komunikaci s motorem CANopen a ovládání jeho pohybu. Výsledky této diplomové práce budou dále využity při řešení VZ Inteligentní systémy v automatizaci.

ABSTRACT

This thesis deals with positional control of DC motor Icla D065 of company SIG Positec Automation GmbH with aid of CAN-Bus and CANopen protocol. Individual parts describe general principle and standards of CAN-Bus and international normalized higher-level protocol CANopen for systems control. For control this motor was used microcontroller PIC18F4685 from company Microchip and developmental environment MPLAB® IDE. With used C language was made program for MCU control communication with a motor by support of CANopen protocol and its motion control. The result of this thesis will be used in solution of research project Intelligent Systems in Automation.

KLÍČOVÁ SLOVA

Sběrnice CAN, protokol CANopen, PIC18F4685, motor Icla D065, řízení pohonu.

KEYWORDS

CAN-Bus, CANopen protocol, PIC18F4685, motor Icla D065, motor control.

PODĚKOVÁNÍ

Děkuji především vedoucímu diplomové práce Ing. Radomilu Matouškovi, Ph.D. a panu Petru Matějkovi za poskytnutí informací, cenných rad, připomínek a pomoci při realizaci diplomového úkolu. Dále děkuji také Ing. Vilmě Jandové za jazykovou korekturu práce.

OBSAH:

ZADÁNÍ ZÁVĚREČNÉ PRÁCE - 3 -
ABSTRAKT - 5 -
KLÍČOVÁ SLOVA - 5 -
PODĚKOVÁNÍ - 7 -
SEZNAM POUŽITÝCH SYMBOLŮ - 11 -
1. ÚVOD - 13 -
2. SBĚRNICE CAN - 15 -
2.1. Základní vlastnosti - 15 -
2.2. Fyzická vrstva - 15 -
2.2.1. Přenosová média - 15 -
2.2.2. Topologie sběrnice - 16 -
2.2.3. Přístup na sběrnici - 16 -
2.3. CAN protokol - 17 -
2.3.1. Principy datového přenosu - 17 -
2.3.2. Přenos dat v reálném čase - 17 -
2.3.3. Formáty rámce zprávy - 18 -
2.3.4. Formát základního rámce dle specifikace CAN 2.0 A - 18 -
2.3.5. Formát rozšířeného rámce dle specifikace CAN 2.0 B - 19 -
2.3.6. Detekce a oznamování chyb - 19 -
3. PROTOKOL CANopen - 21 -
3.1. Aplikační vrstva CAL - 21 -
3.2. Objektová knihovna - 22 -
3.3. Komunikace - 23 -
3.4. Předdefinovaná sada spojení - 26 -
3.5. Přidělení identifikátoru - 27 -
3.6. Proces startování - 28 -
3.7. Detaily skladby zprávy - 29 -
3.7.1. Řízení NMT modulu - 29 -
3.7.2. NMT startování uzlu - 30 -
3.8. Shrnutí - 30 -
4. UŽITÁ PLATFORMA MIKROPROCESORU - 31 -
4.1. Popis vstupně výstupních pinů - 35 -
4.2. Nastavení oscilátorů - 37 -
4.3. Vnitřní struktura paměti - 38 -

4.4. Organizace programové paměti - 38 -
4.5. Organizace datové paměti (RAM) - 39 -
4.5.1. Organizace datové paměti EEPROM - 41 -
4.6. Přerušení - 41 -
4.6.1. Přerušení na pinech INTx - 42 -
4.6.2. Přerušení portu B při změně stavu - 42 -
4.7. Výstupní porty - 43 -
4.8. Technologie ECAN - 43 -
4.8.1. Operační módy CAN modulu - 45 -
4.8.2. Funkční módy CAN modulu - 47 -
4.8.3. Nastavení přenosové rychlosti - 48 -
4.8.3.1. Popis časového kvanta - 49 -
4.8.3.2. Synchronizační segment - 49 -
4.8.3.3. Propagační segment - 49 -
4.8.3.4. Fázový segment - 49 -
4.8.3.5. Vzorkovací bod - 50 -
5. POPIS OVLÁDANÉHO POHONU IclA D065 - 51 -
5.1. Části pohonu - 51 -
5.2. Operační módy - 53 -
5.3. Vlastnosti okolního prostředí - 53 -
5.4. Rozvržení signálního převodníku - 53 -
5.5. Připojení CAN sběrnice - 55 -
6. CAN OVLÁDÁNÍ POHONU VERZE 1.0 - 57 -
6.1. Hlavní menu ovladače - 59 -
6.1.1. Nastavení motoru - 59 -
6.1.2. Nastavení trasy - 60 -
6.1.3. Manuální řízení - 61 -
6.2. Demonstrační úloha - 61 -
7. ELEKTRONICKÉ ZAPOJENÍ „CAN ovládání pohonu verze 1.0“ - 63 -
8. ZDROJOVÝ KÓD - 67 -
9. ZÁVĚR - 101 -
SEZNAM POUŽITÉ LITERATURY - 103 -

SEZNAM POUŽITÝCH SYMBOLŮ

ACK - ACKnowledge field (pole potvrzení příjmu)
BRP - Baud Rate Prescaler (násobitel přenosové rychlosti)
CAL - CAN Application Layer (aplikační vrstva vyvinuta firmou Philips)
CAN - Controller Area Network (síť řadičů)
CCP1 - Capture / Compare / PWM modul (modul používaný jako: vzorkovač, komparátor, nebo zdroj pulzně šířkové modulace)
CiA - CAN in Automation (mezinárodní sdružení výrobců a uživatelů zabývající se vývojem a podporou vysokoúrovňových CAN protokolů)
CMS - CAN Based Message Specification (specifikace zpráv CAN sběrnice)
COB-ID - Comunication OBject IDetificator (identifikátor komunikačního objektu)
CPU - Central Processing Unit (centrální výpočetní jednotka - procesor)
CRC - Cyclic Redundant Check (součet cyklické redundantní kontroly)
CS - Command Specifier (specifikátor příkazu)
DBT - DistriBuTor (stanice zajišťující dynamické přidělování CAN identifikátorů)
DC - Direct Curent (stejnosměrné napětí)
DIP - Dual In-line Package (provedení součástek s dvěma řadami pinů)
DLC - Data Length Code (kód počtu bytů přenášených dat)
DS - Draft standard (norma)
DSP - Draft standard proposal (norma požadavků na návrh zařízení)
ECAN - Enhanced CAN (rozšířený CAN modul)
EDS - Electronnic DataSheet (elektronická databáze knihovny uzlu)
EEPROM - Electrically Erasable Programmable Read-Only Memory (programovatelná, elektricky vymazatelná paměť určená pouze pro čtení)
EOF - End of Frame (konec zprávy)
FIFO - First In First Out (zásobník typu „první dovnitř, první ven“, kde hodnota do něj uložená jako první, je z něj jako první také vyzvednuta)
GPR - General Purpose Registers (registry pro obecné použití)
ICD - In-Circuit Debugging (rozhraní používané pro ladění programového kódu za chodu, přímo v mikroprocesoru)
ICSP - In-Circuit Serial Programming (sériové komunikační rozhraní používané pro programování mikroprocesorů osazených na desce plošného spoje)
IDE - IDentifier Extension (bit identifikátoru rozšířeného formátu rámce)
IFS - Intermission Frame Space (prostor mezi přenášenými rámci)
ISO - International Organization for Standardization (Mezinárodní organizace pro normalizaci)
LMT - Layer ManagemenT (služby změny parametrů jako jsou přenosová rychlost, adresa uzlu atd.)
MSSP - Master Synchronous Serial Port (synchronně řízený sériový port)
NMT - Network ManagemenT (služby řízení sběrnice)
NRZ - Non Return to Zero (bez návratu k nule, je typ kódování, kde logická 1 je reprezentována např. kladným napětím a logická 0 napětím záporným. 0 V zde Nepředstavuje žádnou logickou úroveň)

OSI	-	Open Systems Interconnection (propojení otevřených systémů)
PDIP	-	Plastic DIP (plastové provedení součástek s dvěma řadami pinů)
PDO	-	Process Data Object (objekt procesních datových zpráv)
PLC	-	Programmable Logic Controller (programovatelný logický automat)
PLL	-	Phase Lock Loop (frekvenční násobitel uzavřené fázové smyčky)
QFN	-	Quad Flat No leads (ploché čtvercové provedení mikroprocesorů bez nožiček)
RAM	-	Random Access Memory (paměť s libovolným přístupem)
RC	-	zpětnovazební oscilátor tvořený rezistory (R), kondenzátory (C) a operačním zesilovačem
RPDO	-	Receive Process Data Objects (objekty určené pro příjem procesních dat)
RTR	-	Remote Transmission Request (vzdálený rámec)
SAE	-	Society of Automotive Engineers (profesní sdružení odborníků z oblasti leteckého, automobilového a dopravního průmyslu)
SDO	-	Service Data Object (objekt servisních datových zpráv)
SFR	-	Special Function Registers (registry speciálních funkcí jsou používány CPU a periferiemi pro ovládání prováděných operací)
SOF	-	Start of Frame (uvozující bit rámce)
SYNC	-	SYNChronization (synchronizační zpráva)
TPDO	-	Transmit Process Data Objects (objekty určené pro vysílání procesních dat)
TQFP	-	Thin Quad Flat Pack (tenké ploché čtvercové provedení mikroprocesorů)
USART	-	Universal Synchronous Asynchronous Receiver Transmitter (modul zastřešující různá sériová komunikační rozhraní, jako např. RS-232)
WDT	-	WatchDog Timer (časovač hlídání chodu programového kódu)

1. ÚVOD

Obsahem diplomové práce je vyvinout komunikační rozhraní pro polohové řízení inteligentního pohonu IclA D065 firmy SIG positec pomocí sběrnice CAN a komunikačního protokolu CANopen. Jádrem komunikačního rozhraní je zvolen 8 bitový mikroprocesor z produktové řady PIC firmy Microchip.

Teoretická část práce je zaměřena na popis automatizační sběrnice CAN, která byla vyvinuta pro použití v automobilovém průmyslu, ale s postupem času se její použití rozšířilo do mnoha automatizačních odvětví. Jedním z nich je i řízení polohovacích zařízení. V rámci práce jsou popsány obecné standardy řídícího protokolu CANopen, který je jedním z vysokoúrovňových protokolů CAN sběrnice používaných pro nastavení parametrů sběrnice a parametrů jednotlivých stanic připojených ke sběrnici. Definovány jsou jednotlivé typy přenášených zpráv a jejich priorit, stejně jako standardní objekty v objektových knihovnách CAN zařízení. Je specifikována vybraná platforma použitého 8 bitového mikroprocesoru PIC18F4685, přičemž popis je orientován na části mikroprocesoru použité pro řešení dané úlohy, kterými jsou oscilátor, vnitřní organizace pamětí, vstupně výstupní porty, přerušení a ECAN modul. Dále specifikuje řízený inteligentní pohon IclA D065, obecný konstrukční popis, popis portů a běžného zapojení. Praktická část diplomové práce pak řeší ovládání vytvořeného rozhraní „CAN ovládaní pohonu verze 1.0“, popis funkčních částí a demonstrační úlohou. Její součástí je také elektronická dokumentace výše zmíněného komunikačního rozhraní.

Obrázky použité v této práci jsou převzaty ze zahraniční literatury a zahraničních serverů. Obsahují anglické názvy, které jsou v oblasti automatizační techniky běžně užívané, proto nejsou překládány. Vyžaduje-li to další výklad, jsou výrazy přeloženy a vysvětleny v rámci textu diplomové práce.

Výsledky této diplomové práce budou dále využity při řešení VZ Inteligentní systémy v automatizaci.

2. SBĚRNICE CAN

2.1. Základní vlastnosti

Sběrnice CAN (Controller Area Network), dále jen sběrnice, je systém sériové sběrnice, který byl původně vyvinut pro aplikaci v automobilovém průmyslu na začátku 80. let dvacátého století. CAN protokol je mezinárodně standardizovaný v ISO 11898-1 a obsahuje popis spojové vrstvy podle sedmi úrovňového ISO/OSI referenčního modelu. CAN, který je v dnešní době podporován více než padesáti výrobci polovodičových součástek, poskytuje dvě komunikační služby: odesílání zpráv (vysílání datových rámců) a požadování zpráv (vzdálený požadavek přenosu). Všechny ostatní poskytované služby jako oznamování chyb, automatické přenesení chybných rámců atd. jsou prováděny automaticky mikrořadičem [1].

Spojová vrstva zajišťuje tyto služby:

- ✓ podpora více řídících CAN uzlů (dále jen uzel) – jakýkoli uzel může poslat zprávu, je-li sběrnice v nečinnosti,
- ✓ podpora komunikačního vysílání – všechny přenášené zprávy jsou přijímány všemi uzly. Všechny přijímající uzly rozhodují o akceptaci těchto zpráv, což zaručuje jak datovou konzistenci tak, že všechny uzly v síti budou používat stejné informace.;
- ✓ sofistikovaný detekční mechanismus a opětovné přenesení chybných zpráv – zaručuje datovou konzistenci v celé síti;
- ✓ nedestruktivní arbitráž sběrnice – jestliže dva nebo více uzlů požaduje současně přenos zprávy, protokol zaručuje, že zpráva s vyšší prioritou získá bezprostřední přístup na sběrnici.

2.2. Fyzická vrstva

CAN protokol definuje spojovou vrstvu a část fyzické vrstvy podle ISO/OSI sedmivrstvého referenčního modelu. Mezinárodní standardizační organizace definuje standard, který obsahuje CAN specifikace a specifikaci části fyzické vrstvy: fyzické signály, které zahrnují bitové kódování a dekódování stejně jako bitové časování a synchronizaci [1].

2.2.1. Přenosová média

Základem pro přenos CAN zpráv, dále jen zpráv, a protikladem pro přístup na sběrnici je schopnost reprezentovat hodnotu dominantního a recesivního bitu. Což je dosud možné pro elektrická a optická média a drátový i bezdrátový přenos. Pro elektrická média jsou definována diferenční výstupní napětí v normách ISO 11898-2, ISO 11898-3, SAE J2411 a ISO 11992.

U optických médií je recesivní úroveň definována „tmou“ a dominantní úroveň „světlem“. Nejběžněji používaným fyzickým médiem pro implementaci sběrnice je diferenčně řízená dvojlinka. Pro základní automobilovou elektroniku se používá jednodrátová sběrnice.

Parametry elektrických médií nás zajímají pouze v případě, že chceme zvětšit délku sběrnice. Šíření signálu, odpor vedení a průřez vedení jsou faktory, které nás

nejvíce zajímají při dimenzování sítě a jsou vyžadovány k dosažení největšího možného datového přenosu při dané délce vedení. Obecně platí pravidlo, že čím delší je sběrnice tím je větší úbytek napětí na jejím konci. Průřez vodičů je vypočítán pomocí dovoleného poklesu napětí signálové úrovně mezi dvěma nejvzdálenějšími uzly v systému a celkovým vstupním odporem všech připojených přijímačů. Dovolený pokles napětí musí být nejvýše takový, aby úroveň signálu mohla být spolehlivě interpretována v jakémkoli přijímacím uzlu.

2.2.2. Topologie sběrnice

Elektrické signály na sběrnici se odráží od konců vodičů. K docílení správného čtení úrovně signálu uzlem ze sběrnice, je tedy důležité vyrušení odráženého signálu. Což je zabezpečeno ukončením sběrnice pomocí ukončovacích odporů na obou koncích a vyhnutím se nevyužitých odbočení sběrnice. Nejlepší poměr přenosu vůči délce sběrnice je docílen jak uzavřením jedno linkové struktury, tak zakončením obou konců linky. Specifická doporučení jsou uvedena v normativních standardech ISO 11898-2 a ISO 11898-3. Sběrnici je možné rozšířit pomocí opakovačů, mostů a bran.

Opakovač přenáší elektrický signál z jednoho segmentu sběrnice do druhého. Je pasivním prvkem sběrnice, který pouze obnovuje úroveň přenášeného signálu. Rozděluje sběrnici na dva fyzicky nezávislé segmenty, což způsobuje navýšení času potřebného pro rozšíření signálu.

Most slouží k propojení dvou logicky oddělených segmentů sběrnice na spojové úrovni (OSI úroveň 2). To znamená, že CAN identifikátory musí být jedinečné v každém z těchto dvou segmentů. Mosty implementují zásobník a mohou předávat zprávy nebo jejich části v časově nezávislých přenosech. Odlišují se od opakovačů, protože mohou předávat zprávy, které nejsou lokální. Opakovače přeposílají pouze elektrické signály včetně CAN identifikátoru.

Brány poskytují propojení sběrnic s různými vysokoúrovňovými protokoly, což znamená, že provádí překládání dat protokolu mezi dva komunikační systémy. Překlad se provádí v aplikační vrstvě (OSI úroveň 7).

2.2.3. Přístup na sběrnici

Propojení mezi sběrnicí a čipem CAN řadiče je realizováno pomocí transceiveru. Transceivery jsou vyráběny pro různá přenosová média dle daných standardů a v podstatě jsou tvořeny vysílačem a přijímačem.

Vysílač poskytuje dostatečnou schopnost řízení výstupu, chrání vestavěný CAN řadič proti přetížení a redukuje elektromagnetické záření. Kromě úpravy signálu mezi čipem a médiem sběrnice musí vysílač obsahovat také řadu přídavných funkcí.

Přijímač zajišťuje definovanou úroveň recesivního signálu a ochraňuje vstupní komparátor řadiče před přepětím ze sběrnice. Rozšiřuje běžný rozsah vstupního komparátoru CAN řadiče a zajišťuje dostatečnou vstupní citlivost. Kromě toho detekuje chyby sběrnice, jako jsou např. přerušení obvodu, zkrat, proražení na kostru…atd.

Funkcí transceiveru může být také galvanické oddělení elektrických obvodů uzlu od obvodu sběrnice.

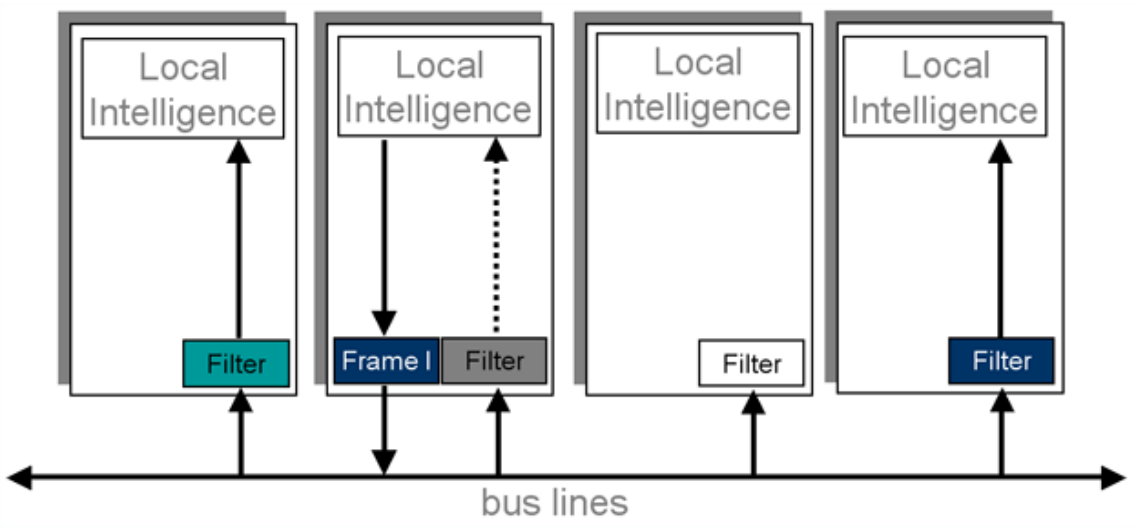
2.3. CAN protokol

CAN protokol je definován mezinárodním standardem ISO 11898. Vedle samotného protokolu je také standardem ISO 16845 definována zkouška přizpůsobivosti CAN čipů, která zaručuje jejich zaměnitelnost [1].

2.3.1. Principy datového přenosu

CAN je založen na „mechanismu komunikačního vysílání“, jehož základem je protokol vysílání pomocí zpráv, který definuje obsah zprávy. Každá zpráva má identifikátor zprávy, který je jedinečný v celé sběrnici, čímž je definován obsah a priorita zprávy. Při vysílání většího počtu uzlů na sběrnici je dle identifikátoru zprávy řízen výběr právě toho uzlu, který bude v danou chvíli vysílat na sběrnici (arbitráž sběrnice).

Výsledkem obsahově orientovaného adresního schématu je dosažení vysokého stupně systému a konfigurační flexibility. Připojení uzlu k existující sběrnici je možné bez nutnosti zásahu do hardwaru nebo softwaru již připojených uzlů, pokud jsou připojované uzly přijímači. Připojení umožňuje modulární koncept sběrnice, který také zajišťuje přijímání mnohonásobných dat a synchronizaci distribuovaných procesů. Datový přenos, viz Obr. 1, není založen na dostupnosti specifických typů stanic, které poskytují základní obsluhu a aktualizaci sběrnice.

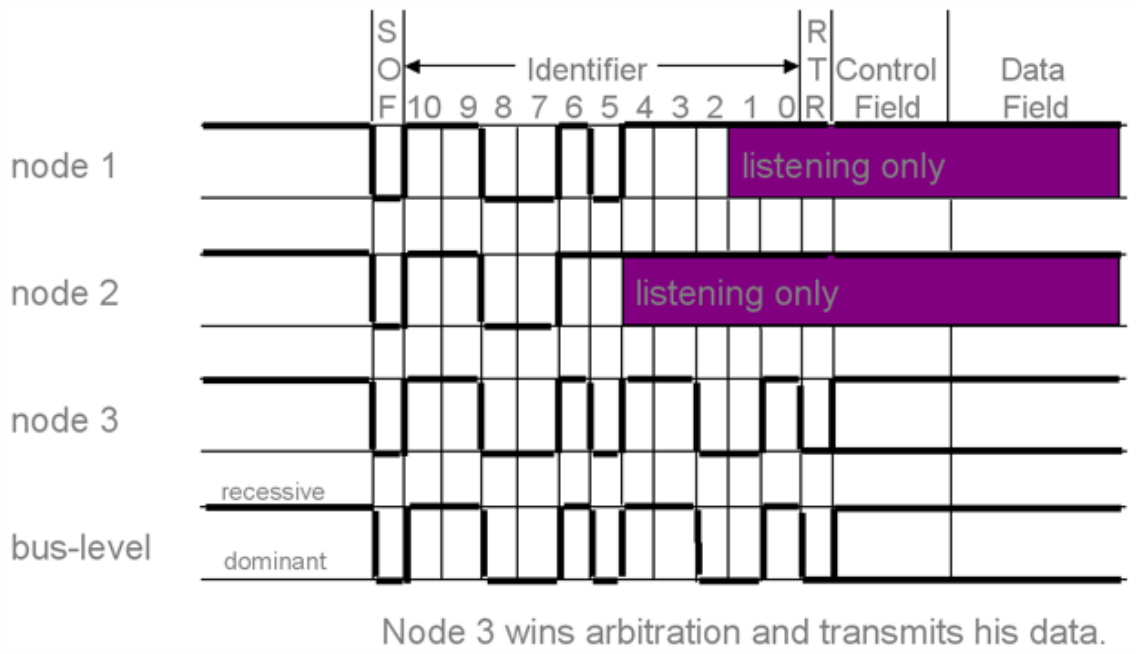


Obr. 1 Princip datového přenosu [2]

2.3.2. Přenos dat v reálném čase

V reálném čase dochází ke značným rozdílům v naléhavosti posílání zpráv po sběrnici. Rychlé výměna parametru např. zátěže motoru, bývá vysílána častěji a tedy s menšími prodlevami než jiné parametry jako např. teplota motoru. Priorita, se kterou je zpráva vysílána vzhledem k jiné méně naléhavé zprávě, je určena identifikátorem zprávy. Priority jsou stanoveny v průběhu návrhu systému formou odpovídajících binárních hodnot, které nemohou být dynamicky měněny. Identifikátor s menší binární hodnotou má větší prioritu.

Konflikty v přístupu na sběrnici jsou řešeny tzv. „bitovou arbitráží“ identifikátoru viz Obr. 2. Zúčastněná stanice sleduje úroveň sběrnice bit po bitu. V případě, že dva uzly vysílají zprávu současně, dochází ke konfliktu tehdy, když každý uzel vyšle rozdílnou úroveň signálu. V této chvíli dominantní stav přepíše recesivní stav na sběrnici a uzel vysílající recesivní stav ukončí vysílání zprávy a pouze naslouchá. Všechny uzly, které ztratily právo vysílat, se automaticky stávají přijímacími uzly zpráv s vyšší prioritou a nemohou opakovat vysílání, dokud není dokončeno odesílání předchozí zprávy.



Obr. 2 Řešení konfliktu více uzlů v přístupu na sběrnici [2]

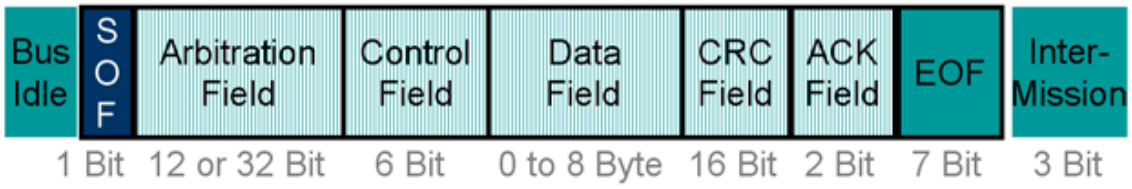
2.3.3. Formáty rámce zprávy

CAN protokol podporuje dva formáty rámce zprávy, které se liší jediným parametrem a to délkou identifikátoru. Základní rámec stanovuje 11 bitovou délku identifikátoru, zatímco rozšířený rámec stanovuje délku identifikátoru na 29 bitů.

2.3.4. Formát základního rámce dle specifikace CAN 2.0 A

Zpráva se základním rámcem začíná start bitem zvaným „Start of Frame (SOF)“, který je následován „arbitračním polem“ skládajícím se z identifikátoru a bitu požadavku vzdáleného přenosu „Remote Transmission Request (RTR)“ užívaného k rozlišení mezi datovým rámcem a rámcem datového požadavku zvaného vzdálený rámec. Dále pak následuje řídící pole, které je složeno z bitu identifikátoru rozšířeného formátu rámce „IDentifier Extension (IDE)“ s rozlišením mezi základním a rozšířeným rámcem následovaný kódem délky dat „Data Length Code (DLC)“ užívaného k určení počtu přenášených bytů v datovém poli „Data field“. Je-li zpráva využívána jako vzdálený rámec, DLC obsahuje počet požadovaných datových bytů. Datové pole, které následuje, může obsahovat až 8 bytů dat. Neporušenost rámce je zaručena následujícím

součtem cyklické redundantní kontroly „Cyclic Redundant Check (CRC)“. Následuje pole potvrzení příjmu „ACKnowledge field (ACK)“, které obsahuje ACK slot a ACK oddělovací znak. Bit v ACK slotu je odeslán jako recesivní bit a je přepsán každým příjemcem, který tuto zprávu obdrží bez chybně, na dominantní. Bezchybné zprávy jsou potvrzeny příjemcem bez ohledu na výsledek akceptačního testu. Konec zprávy je označen „End of Frame (EOF)“ viz Obr. 3. Prostor mezi přenosem rámců „Intermission Frame Space (IFS)“ je minimální počet bitů oddělující jednotlivé zprávy. Dokud nedojde k vysílání některého z uzlů, sběrnice zůstává nečinná.



Obr. 3 Zobrazení formátu datového rámce [2]

2.3.5. Formát rozšířeného rámce dle specifikace CAN 2.0 B

Rozdíl mezi formátem rozšířeného rámce zprávy a formátem základního rámce zprávy je v délce použitého identifikátoru. 29 bitový identifikátor je vytvořen 11 bitovým identifikátorem a 18 bitovým rozšířením. Rozlišení mezi rámci je tvořeno použitím IDE bitu. Nastavení tohoto bitu na úroveň dominantní znamená použití 11 bitového rámce, úroveň recesivní znamená 29 bitový rámec. Jelikož tyto dva formáty mohou být na sběrnici používány, platí, že zpráva s 11 bitovým identifikátorem má vždy vyšší prioritu při kolizním přístupu na sběrnici. Rozšířený formát obsahuje také nevýhody, kterými jsou: delší čekací doba pro přístup na sběrnici, zpráva vyžaduje širší přenosové pásmo a méně kvalitní detekce chyb.

CAN mikrořadiče, které podporují formát rozšířeného rámce zpráv, umožňují také posílat a přijímat zprávy ve formátu základního rámce. Dále existují CAN řadiče, které podporují pouze formát základního rámce. Formát rozšířeného rámce umí rozeznat, ale ignorují ho.

2.3.6. Detekce a oznamování chyb

Na rozdíl od jiných detekčních systémů, CAN protokol neumí používat potvrzování zpráv, ale umí signalizovat chyby bezprostředně po jejich výskytu. Pro detekci chyb jsou implementovány tři mechanismy na úrovni zprávy (spojová vrstva OSI úroveň 2): cyklická redundantní kontrola (CRC), ověření rámce, ACK chyby.

CAN protokol také implementuje dva mechanismy pro detekci chyb na bitové úrovni (fyzická vrstva OSI úroveň 1). Jsou jimi: monitorování a doplňkový bit.

Jestliže nejméně jeden uzel odhalí jednu nebo více chyb použitím mechanismů uvedených výše je aktuální přenos zrušen odesláním chybového rámce. Tím je zabráněno ostatním stanicím v přijetí zprávy, čímž je zajištěna bezchybnost přenosu dat po sběrnici. Po ukončení vysílání chybné zprávy, která byla zrušena, odesílatel automaticky znovu odešle původní zprávu. Uzly mohou znovu soupeřit o přístup na sběrnici.

Jakkoli mohou být popsané metody výkonné a efektivní, v případě poruchy uzlu může nastat případ, že všechny zprávy budou zrušeny. Jestliže uzel nebude zajišťovat vlastní autodetekci, systém sběrnice může být takovou poruchou blokován. CAN protokol z tohoto důvodu obsahuje mechanismus rozlišující ojedinělé chyby od trvalých chyb a místních selhání uzlů. Toto je zajištěno pomocí statického odhadu chybných stavů uzlů se snahou rozpoznat jejich vlastní poruchy a možností převést poškozený uzel do takového operačního módu, aby nebyl negativně ovlivněn celý zbytek sběrnice.

3. PROTOKOL CANopen

CANopen je vysokoúrovňový protokol pro CAN sběrnici. Byl vyvinut jako standard pro sítě s vysokou flexibilitou nastavení. Původně byl navržen především pro sítě ovládající pohybově orientovaná zařízení, jako jsou manipulační systémy. V dnešní době je však využíván v různých aplikačních oblastech, kterými jsou např. medicínská zařízení, terénní vozy, námořní elektronika a vlaková nebo stavební automatizace. Poskytuje standardizované komunikační objekty pro přenos dat v reálném čase, konfigurační data a data síťového managementu. Standardizovaná CANopen zařízení a aplikační profily zjednodušují úlohy začlenění CANopen systému. V dnešní době je již k dispozici mnoho nástrojů a zařízení, která podporují CANopen protokol, a jsou dostupná za rozumnou cenu.

Soubor CANopen specifikací obsahuje profily zařízení a aplikační profily. Obecné i specifické profily zařízení upřesňují jejich logická rozhraní. Aplikační profily popisují sadu virtuálních rozhraní. CANopen zařízení mohou implementovat jedno nebo více virtuálních zařízení.

Motor využitý pro potřeby řešení této diplomové práce se řídí komunikačním profilem DS301 a profilem zařízení DSP 402 vydaným asociací CAN in Automation (CiA). Profil zařízení pro řízení a ovládání pohonu definuje funkční chování řadiče pro servopohony, frekvenční měniče a krokové motory. Také specifikuje operační módy a odpovídající konfigurační parametry. Specifikace obsahuje konečné množství automatů, které upřesňují vnitřní a vnější chování zařízení pro každý stav. Stavy jsou měněny pomocí řídícího slova (control word) obdrženého od řídícího členu a mohou být měněny vnitřními událostmi. Aktuální stav je indikován pomocí stavového slova (status word). Řídící slovo a různé příkazy (např. rychlost) jsou mapovány do přednastaveného RPDO (Receive Process Data Objects). Stavové slovo a různé další aktuální hodnoty (např. pozice) jsou mapovány do TPDO (Transmit Process Data Objects). Specifikace poskytuje sadu běžných přednastavených PDO dostupných pro všechny pohony a také sadu specifických přednastavených PDO použitelných pouze pro specifické třídy pohonu jako jsou servopohony, frekvenční měniče a krokové motory.

3.1. Aplikační vrstva CAL

Aplikační vrstva CAL (CAN Application Layer) vyvinuta firmou Philips Medical Systems je jedním z vysokoúrovňových protokolů pro CAN sběrnice poskytující sadu služeb a protokolů užitečných pro každé zařízení na sběrnici. Protokol byl převzat nezávislou skupinou uživatelů a výrobců CiA, která jej neustále zdokonaluje a publikuje v řadě standardů.

CAL stanovuje čtyři základní služby aplikační vrstvy, kterými jsou:

- ✓ CMS (CAN Based Message Specification) – poskytuje objekty typu proměnná, událost a doména k navrhování a upřesnění přístupu k funkcím uzlu přes CAN rozhraní;
- ✓ NMT (Network ManagemenT) – poskytuje služby pro podporu řízení sběrnice jako je inicializace, zpuštění a zastavení uzlů a detekce poruchy uzlu; řídí se podle konceptu řídící a podřízené stanice, přičemž v daném okamžiku je na sběrnici pouze jedna NMT řídící stanice;

- ✓ DBT (DistriBuTor) – zajišťuje dynamické přidělování CAN identifikátorů (oficiálně zvaných COB-ID, identifikátory komunikačních objektů) uzlům na sběrnici, služba se řídí podle konceptu řídící a podřízené stanice; přičemž v daném okamžiku je na sběrnici pouze jedna DBT řídící stanice;
- ✓ LMT (Layer ManagemenT) – umožňuje změnu parametrů vrstvy, jako jsou např. NMT adresa uzlu nebo změna přenosové rychlosti CAN rozhraní.

CMS definuje 8 úrovní důležitosti zpráv. Každá úroveň obsahuje 220 identifikátorů komunikačních objektů (COB-ID), což znamená, že celkem zabírají 1760 identifikátorů z celkového množství 2032. Zbývající identifikátory jsou rezervovány pro MNT, DBT, LMT služby. Znázornění viz Tab. 1. Zprávy s nižší hodnotou COB-ID mají vyšší prioritu.

Tab. 1 Mapování služeb a objektů CAL pomocí 11 bitového COB-ID

CAN aplikační vrstva	
COB-ID	**Využití**
0	NMT služby start/stop
1 - 220	CMS objekty s prioritou 0
221 - 440	CMS objekty s prioritou 1
441 - 660	CMS objekty s prioritou 2
661 - 880	CMS objekty s prioritou 3
881 - 1100	CMS objekty s prioritou 4
1101 - 1320	CMS objekty s prioritou 5
1321 - 1540	CMS objekty s prioritou 6
1541 - 1760	CMS objekty s prioritou 7
1761 - 2015	NMT služby pro monitorování uzlu
2016 - 2031	NMT, LMT, DBT služby

Standard CAL přebírá 11 bitový identifikátor ze standardu CAN 2.0A jehož oblast hodnot je 0 – 2047, ale z historických důvodů je jeho oblast hodnot omezena na oblast 0 – 2031. Při použití 29 bitového identifikátoru podle CAN 2.0B se počet řádků v Tab. 1 nezmění, pouze dojde k mnohonásobnému zvětšení rozsahu hodnot COB-ID [3].

3.2. Objektová knihovna

Jedná se o uspořádanou skupinu objektů, přičemž každý objekt je určený 16 bitovým indexem. Tyto objekty mohou být strukturami, jejichž prvky jsou popsány pomocí 8 bitových podindexů. Obecný popis rozvržení objektové knihovny viz Tab. 2.

Tab. 2 Obecná struktura CANopen objektové knihovny

Objektová knihovna CANopen	
Index	**Objekt**
0000	nevyužito
0001 – 001F	Statické datové typy (standardní datové typy jako Integer16, Boolean atd.)
0020 – 003F	Komplexní datové typy (předdefinované struktury složené ze standardních datových typů)
0040 – 005F	Komplexní datové typy definované výrobcem
0060 – 007F	Statické datové typy definované v profilu zařízení
0080 – 009F	Komplexní datové typy definované v profilu zařízení
00A0 – 0FFF	vyhrazeno
1000 – 1FFF	Oblast komunikačního profilu (např. typ zařízení, chybový registr, počet podporovaných PDO zpráv)
2000 – 5FFF	Oblast nastavení specifikovaných výrobcem
6000 – 9FFF	Oblast profilu standardizovaných zařízení (např. DSP 402)
A000 – FFFF	vyhrazeno

Pro všechny uzly na sběrnici existuje objektová knihovna. Obsahuje parametry popisující zařízení a jeho chování na sběrnici. Knihovna uzlu existuje ve formě databáze popsané v elektronickém souboru EDS nebo v papírové podobě. Proto není nutné se přes sběrnici dotazovat uzlu na všechny jeho parametry uvedené v objektové knihovně. Uzel pouze potřebuje být schopen prezentovat alespoň všechny povinné položky objektové knihovny a případně některé jiné položky, které tvoří součást konfigurovatelné funkčnosti uzlu.

Profil popisuje pro každý objekt knihovny jeho funkci, jméno, index i podindex, datový typ, zda je objekt povinný či nepovinný a jestli je objekt určen pro čtení, zápis nebo čtení i zápis. Profily definují, které objekty jsou povinné a které volitelné, množství povinných je omezeno na minimum. Volitelné funkce lze přidávat k rozšíření funkčnosti zařízení CANopen. Pokud zařízení vyžaduje více funkcí, než je uvedeno v profilu můžou být tyto funkce do zařízení přidány výrobcem.

Část knihovny popisující komunikační parametry je stejná pro všechny CANopen zařízení tzn., že umístění objektu v objektové knihovně zůstává stejné, ale může mít různé hodnoty [3].

3.3. Komunikace

Zabývá se zprávami posílanými po sběrnici, jejich obsahem a funkčností. Jinými slovy lze hovořit o komunikačním modelu. CAN komunikační model definuje čtyři typy zpráv (komunikačních objektů):

- ✓ administrativní zprávy – služby a protokoly jsou v souladu s prvky služeb CAL, které jsou založeny na koncepci řídící a podřízené stanice: na sběrnici je pouze jedna LMT-, NMT- nebo DBT- řídící stanice a jedna nebo více podřízených stanic;

- ✓ servisní datový objekt (SDO) – poskytuje klientský přístup do položek (objektů) objektové knihovny zařízení (zařízení je serverem) pomocí indexu a podindexu objektu, který je obsažen v několika prvních bytech CAN zprávy. SDO umožňuje přenos dat o libovolné délce, pokud jsou data delší, než 4 bity jsou rozdělena do více zpráv. SDO je typem potvrzované zprávy, odpověď je generována pro každou zprávu. Komunikace prostřednictvím SDO vyžaduje značnou režii;
- ✓ objekt procesních dat (PDO) – využívá se k přenosu dat v reálném čase, data jsou přenášena od pouze jednoho producenta k jednomu nebo více spotřebitelům. Přenos dat je limitován přenášením min. 1 a max. 8 bytů dat, např. jeden objekt procesních dat může přenést 64 vstupních a výstupních (I/O) hodnot nebo čtyři 16 bitové analogové vstupy. Datový obsah PDO je definován pouze přes jeho CAN identifikátor a je znám odesílateli stejně dobře jako příjemci PDO. Každé PDO je popsáno dvěma objekty z komunikační knihovny:
    - PDO komunikační parametr – obsahuje, které COB-ID je použito pro PDO, přenosový typ, časovou periodu apod.;
    - PDO parametr přiřazení – obsahuje seznam objektů z objektové knihovny, které jsou přiřazeny do PDO, včetně jejich velikosti v bitech;

  Obsah PDO zprávy je předdefinován nebo se nastavuje při startu sběrnice. Přiřazení aplikačních objektů do PDO je popsáno v objektové knihovně zařízení pomocí PDO parametru přiřazení a je nastavitelný pomocí SDO zpráv, jestliže proměnná PDO přiřazení je podporována zařízením. PDO může mít několik přenosových režimů:
    - synchronní – synchronizace pomocí obdržení objektu SYNC:
        - acyklický – synchronizovaný s ohledem na synchronizační zprávu, ale ne periodický;
        - cyklický – přenos je spouštěn periodicky, po každé 1,2 … 240 synchronizační zprávě;
    - asynchronní:
        - přenos je spouštěn vzdáleným požadavkem přenosu (pomocí CAN vzdáleného rámce) z jiného zařízení;
        - přenos je spouštěn v důsledku výskytu specifické události zařízení popsané v profilu zařízení (např.: změna vstupní hodnoty, časová událost);

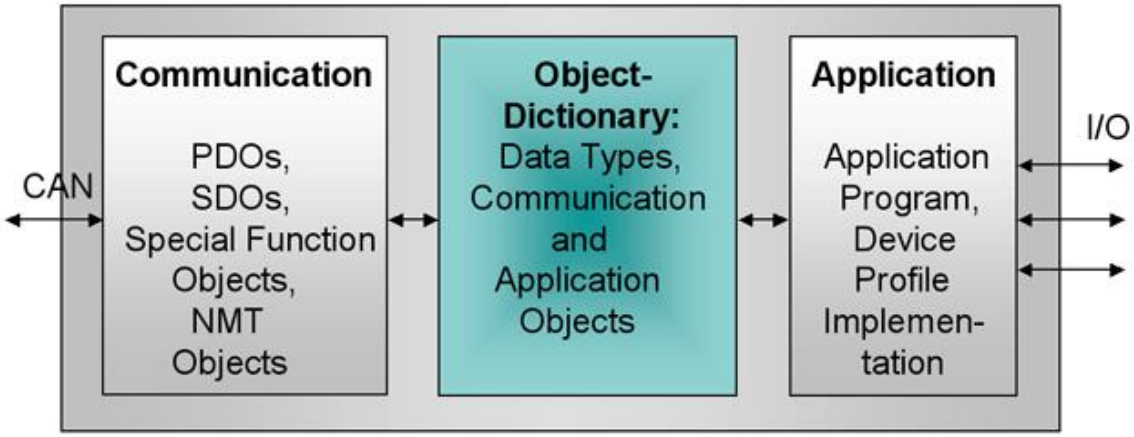
  PDO může být přiřazena časová konstanta nečinnosti, určující minimální čas mezi dvěma po sobě jdoucími PDO přenosy, která zabraňuje tzv. „vyhladovění“ sítě. Může také obsahovat událost spouštěcí periody, pomocí které je periodicky spouštěn PDO přenos, po uplynutí daného času. PDO se provádí jako CMS objekt typu „uložitelná událost“, podle CAL, což znamená, že data jsou přenášena bez režijních nákladů protokolu, tzn. zpráva není potvrzována. Může přenášet max. 8 bytu dat.
- ✓ objekty předdefinovaných zpráv a speciálních funkcí:
    - synchronizační objekt (SYNC):
        - používá se pro synchronizaci úloh sítě, zvláště pak je důležitý pro řídící úlohy. Skutečné vstupní hodnoty jsou více méně současně uloženy v celé síti a následně předány (pokud je to vyžadováno),

výstupní hodnoty jsou aktualizovány podle zprávy přijaté po předchozím SYNC objektu;
    - využívá konceptu řídící a podřízené stanice – řídící stanice vyšle periodický SYNC objekt a podřízené stanice po přijetí SYNC objektu provedou úlohy současně;
    - přenos synchronizačních PDO uvnitř daného časového okna je prováděno s ohledem na přenos objektů SYNC;
    - je realizován jako CMS objekt typu „základní proměnná“;
    - obsahuje COB-ID s nejvyšší prioritou k zajištění pravidelné synchronizace; aby byla zpráva co možná nejkratší, nepřenáší žádné datové byty;
- objekt časového označení:
    - poskytuje aplikačním zařízením odkaz na společný časový rámec;
    - je realizován jako CMS objekt typu „uložitelná událost“;
- nouzový objekt:
    - je vyvolán při výskytu vnitřní poruchy zařízení;
    - je realizován jako CMS objekt typu „uložitelná událost“;
- objekt hlídání uzlu a chodu:
    - je založen na konceptu řídící a podřízené stanice;
    - NMT- řídící uzel monitoruje stavy uzlů – což se nazývá objekt „hlídání uzlu“;
    - uzly volitelně monitorují stav MNT-řídícího uzlu – což se nazývá objekt „hlídání chodu“,
    - detekuje chyby síťových rozhraní zařízení, ale ne selhání zařízení jako takových, tak jsou hlášena prostřednictvím nouzového objektu;
- objekt Boot-up („nabíhání“):
    - je založen na konceptu řídící a podřízené stanice;
    - odesláním tohoto objektu NMT-podřízený uzel oznamuje NMT-řídícímu uzlu, že přešel ze stavu inicializace do předoperačního stavu.

Dva z výše uvedených typů komunikačních objektů jsou užívány pro přenos dat. Implementují dva rozdílné mechanismy přenosu dat:

- ✓ SDO s nízkou prioritou je užíván pro objemný datový přenos mezi zařízeními, typickým využitím je konfigurace zařízení na síti;
- ✓ PDO je užíván pro rychlý přenos dat obsahujících až 8 bytů bez zbytečných nákladů na jejich přenos.

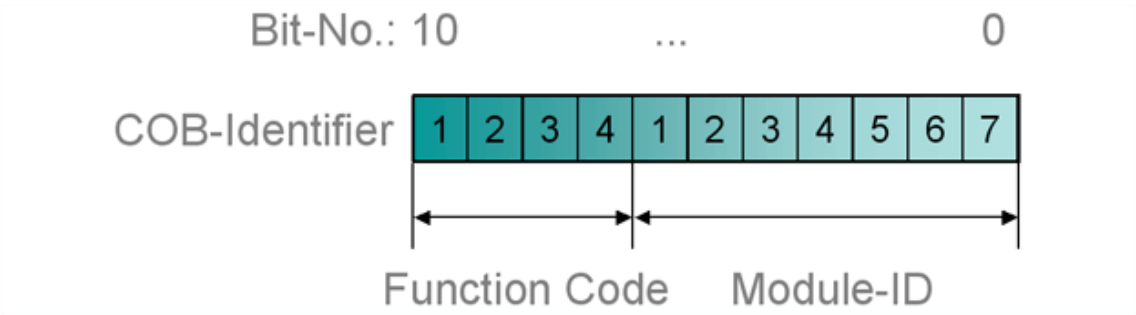
CANopen zařízení podporují řadu služeb pro řízení sítě (administrativní zprávy) a vyžadují minimálně jedno SDO. Každé CANopen zařízení, které vysílá nebo přijímá procesní data, by mělo mít nejméně jedno PDO. Všechny ostatní komunikační objekty jsou nepovinné. Vztahy mezi CAN komunikací, objektovou knihovnou a aplikačním softwarem v CANopen zařízení jsou schematicky znázorněny na Obr. 4.



Obr. 4 Schematické znázornění vztahů mezi komunikaci, objektovou knihovnou a aplikačním softwarem v CANopen zařízení [1]

3.4. Předdefinovaná sada spojení

V zájmu zjednodušení namáhavého nastavování jednoduchých sítí je definováno povinné výchozí schéma přidělování CAN identifikátoru. Identifikátory jsou k dispozici v předoperačním stavu ihned po inicializaci a mohou být změněny pomocí dynamického přidělování. Zařízení poskytuje odpovídající identifikátory pouze pro podporované komunikační objekty. Schéma přiřazení je založeno na rozdělení 11 bitového identifikátoru na 4 bitový funkční kód a 7 bitový identifikátor uzlu (Node-ID). Viz Obr. 5.



Obr. 5 Struktura 11 bitového CAN identifikátoru [1]

Identifikátor uzlu je definován systémovým integrátorem, např. nastavením DIP přepínače na zařízení. Musí ležet uvnitř oblasti od 1 do 127 (nesmí však mít hodnotu 0). Předdefinovaná sada spojení definuje čtyři přijímací PDO objekty, čtyři vysílací PDO objekty, jeden SDO objekt (zabírající 2 CAN identifikátory), jeden nouzový objekt a jeden identifikátor chyby uzlu. Také podporuje vysílání nepotvrzovaných NMT řídících zpráv, objektů časového označení a synchronizačních objektů. Schéma výsledného přiřazení identifikátoru je zobrazeno v Tab. 3 [3].

Tab. 3 Schéma výsledného přiřazení identifikátoru („PDO/SDO vysílání/příjem“ je z pohledu podřízeného uzlu) [3]

Broadcast objects of the CANopen Predefined Master/Slave Connection Set			
Object	**Function code (ID-bits 10-7)**	**COB-ID**	**Communication parameters at OD index**
NMT Module Control	0000	000h	–
SYNC	0001	080h	1005h, 1006h, 1007h
TIME STAMP	0010	100h	1012h, 1013h

Peer-to-Peer objects of the CANopen Predefined Master/Slave Connection Set			
Object	**Function code (ID-bits 10-7)**	**COB-ID** *	**Communication parameters at OD index**
EMERGENCY	0001	081h - 0FFh	1024h, 1015h
PDO 1 (transmit)	0011	181h - 1FFh	1800h
PDO 1 (receive)	0100	201h - 27Fh	1400h
PDO 2 (transmit)	0101	281h - 2FFh	1801h
PDO 2 (receive)	0110	301h - 37Fh	1401h
PDO 3 (transmit)	0111	381h - 3FFh	1802h
PDO 3 (receive)	1000	401h - 47Fh	1402h
PDO 4 (transmit)	1001	481h - 4FFh	1803h
PDO 4 (receive)	1010	501h - 57Fh	1403h
SDO (transmit/server)	1011	581h - 5FFh	1200h
SDO (receive/client)	1100	601h - 67Fh	1200h
NMT Error Control	1110	701h - 77Fh	1016h, 1017h

Přidělování identifikátoru odpovídá sadě připojení dle konceptu řídící a podřízené stanice. Jelikož se identifikátory uzlů navzájem liší, ve skutečnosti s nimi může přímo komunikovat pouze řídící přístroj, který zná všechny identifikátory připojených uzlů (až 127 uzlů). Dva propojené podřízené uzly spolu nemohou komunikovat, protože navzájem neznají své identifikátory.

Z porovnání identifikátorů uvedených v Tab. 1 a Tab. 3 vyplývá, jak jsou objekty CANopen protokolů mapovány do obecných objektů CAL [3].

3.5. Přidělení identifikátoru

Přidělení CAN identifikátoru (COB-ID) v CANopen protokolu může být provedeno třemi způsoby:

- ✓ použitím předdefinované sady spojení řídící a podřízené stanice – přidělení identifikátoru je výchozí, takže není potřebná jeho konfigurace; podporuje-li uzel změnu dat PDO je možné provést jejich konfiguraci tzv. mapováním;
- ✓ úpravou PDO identifikátoru po spuštění (pokud je uzel v předoperačním stavu), s pomocí SDO pro zápis nové hodnoty do příslušného místa v objektové knihovně uzlu;

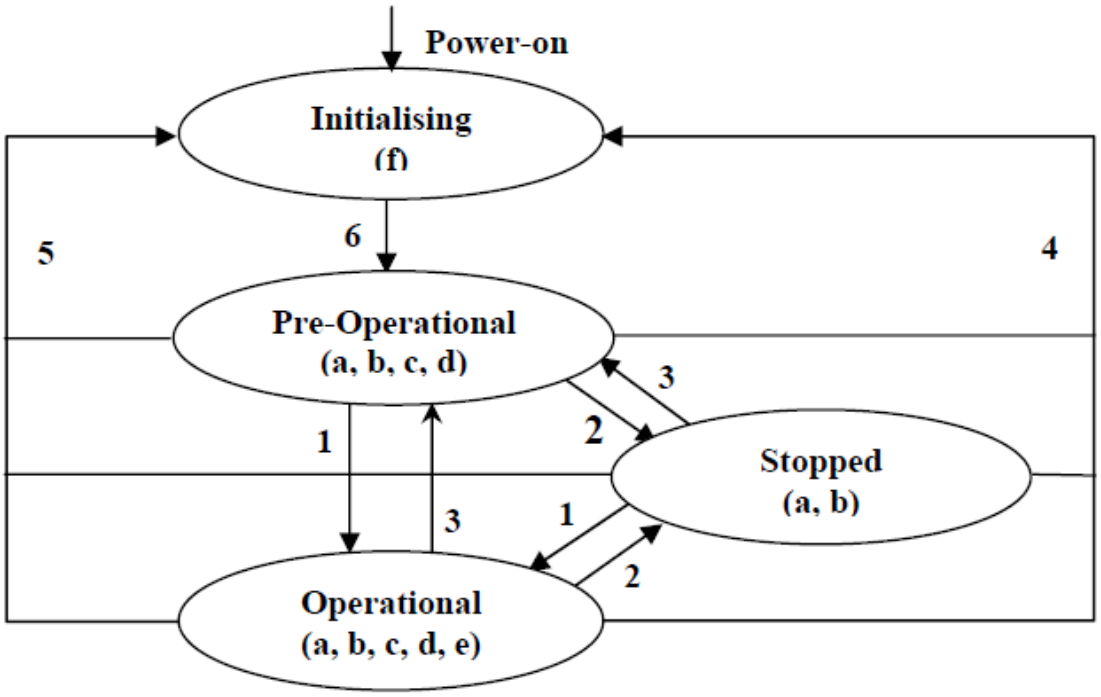
- ✓ použitím CAL DBT služby – uzly připojené do CANopen sítě jsou nejprve identifikovány pomocí jejich přednastavených identifikátorů uzlu; identifikátor uzlu může být konfigurován nastavením DIP přepínače na zařízení nebo použitím služeb CAL LMT. Při inicializaci a startu sítě, síťový řídící uzel nejprve naváže spojení s každým připojeným podřízeným uzlem pomocí zprávy „připojit vzdálený uzel“ (služba CAL NMT). Po navázání spojení jsou uzlům přiděleny CAN identifikátory pro posílání PDO a SDO objektů pomocí distribučních služeb CAL [3].

3.6. Proces startování

Inicializační proces v CANopen síti nabízí tzv. rozšířené startování stejně tak jako tzv. omezené startování. Rozšířené startování je volitelné a omezené startování by mělo být podporováno všemi CANopen zařízeními a uzly. Oba uvedené typy mohou existovat v síti současně.

Uzel podporuje proces rozšířeného startování, jestliže je distribuce identifikátoru prováděna pomocí CAL DBT služeb.

Oba inicializační procesy mohou být vyjádřeny diagramem přechodu stavů uzlu pro omezený startovací proces viz Obr. 6. Diagram stavu rozšířeného startování obsahuje více stavů mezi předoperačním a operačním stavem.



Obr. 6 Diagram přechodových stavů pro omezený start uzlu.
Písmena v závorkách znázorňují, které typy komunikačních objektů mohou být použity při daných stavech uzlů: a) NMT objekt; b) objekt hlídání uzlu; c) objekt SDO; d) nouzový objekt; e) objekt PDO; f) startování [3]

Přechody stavů (1–5 spouštěné službami NMT) a NMT specifikátor příkazu (v závorkách):

1: start vzdáleného uzlu (0x01)
2: ukončení vzdáleného uzlu (0x02)
3: vstup do předoperačního stavu (0x80)
4: obnovení uzlu (0x81)
5: obnovení komunikace (0x82)
6:ukončení inicializace zařízení, automatický přechod do předoperačního stavu, odešle startovací zprávu.

NMT služby jsou používány pro uvedení všech nebo jen vybraných uzlů do provozních stavů. CAN zpráva přenášející tuto NMT službu se stává z CAN hlavičky (COB-ID = 0) a dvou datových bytů. První byte obsahuje požadovanou službu (NMT specifikátor příkazu) a druhý byte obsahuje identifikátor uzlu nebo nulu pro adresaci všech uzlů současně.

V CAN síti existuje jeden uzel, který se chová jako NMT řídící uzel. Vydává NMT zprávy a kontroluje proces inicializace zprávy uzlu. CANopen zařízení podporující pouze omezené startování jsou také nazývána „zařízení s omezenou schopností“. Tato zařízení vstupují do předoperačního stavu automaticky po ukončení inicializace zařízení. V tomto stavu je možné zařízení parametrizovat a přiřadit COB-ID pomocí SDO objektu. Je-li zařízení přepnuto do pohotovostního stavu, je nuceno ukončit veškerou komunikaci, očekávat NMT služby a hlídání uzlu, jestliže jsou tyto služby aktivovány [3].

3.7. Detaily skladby zprávy

Následující podkapitoly se zabývají popisem použití COB-ID v sadě předdefinovaných spojení [3].

3.7.1. Řízení NMT modulu

Pouze NMT řídící stanice vystavuje zprávy řízení NMT modulu. Všechny podřízené stanice musí podporovat služby řízení NMT modulu. Na tyto zprávy se neodpovídá. NMT zpráva má následující formát, který je znázorněn v Tab. 4.

Tab. 4 Formát NMT zprávy a hodnoty specifikátoru příkazů

Vysílání nadřízené stanice

COB-ID	Byte 0	Byte 1
0x000	CS	Adresa stanice (Node-ID)

Specifikátor uzlu s nulovou hodnotou znamená adresaci všech podřízených uzlů. Specifikátor příkazu CS (Command Specifier) může nabývat následujících hodnot:

Specifikátor příkazu (CS)	NMT služba
1	Spustit vzdálenou stanici
2	Zastavit vzdálenou stanici
128	Přepnout do předoperačního stavu
129	Resetovat stanici
130	Resetovat komunikaci

3.7.2. NMT startování uzlu

NMT – podřízený uzel vyšle startovací zprávu pro oznámení NMT řídícímu uzlu o absolvování procesu inicializace a přepnutí do předoperačního stavu, viz Tab. 5.

Tab. 5 Ukázka startovací zprávy

Vysílání podřízené stanice

COB-ID	Byte 0
0x700 + adresa stanice	0

3.8. Shrnutí

Vyrábí – li dva nezávislí výrobci svá zařízení podle směrnic protokolu CANopen a příslušných profilů CANopen zařízení, mohou tato standardizovaná zařízení spolu bezchybně komunikovat v jedné síti po stejné sběrnici. Pro zvýšení úrovně kompatibility by měly být dodrženy následující podmínky:

- ✓ přizpůsobení: zařízení může být připojeno do CANopen sítě bez narušení komunikace ostatních zařízení – kompatibilita na aplikační úrovni,
- ✓ spolupráce: zařízení může vyměňovat data s ostatními uzly v síti – kompatibilita komunikačních profilů,
- ✓ zaměnitelnost: zařízení lze nahradit jiným zařízením – kompatibilita profilu zařízení.

4. UŽITÁ PLATFORMA MIKROPROCESORU

Jedním z cílů této diplomové práce bylo vybrat 8 bitový mikroprocesor pro ovládání pohonu Icla D065, o němž bude pojednávat následující kapitola. Na výběr byly mikroprocesory od firem Atmel nebo Microchip. U Atmelu bylo možné vybírat z mikroprocesorů uvedených v Tab. 6a a 6b. [4]

Tab. 6a a 6b Výběr procesorů od firmy Atmel obsahující CAN moduly

Automotive AVR

Devices	Flash (Kbytes)	EEPROM (Bytes)	SRAM (Bytes)	Max I/O	F.max (MHz)	Vcc (V)	10-bit A/D Channels	Analog Comparator	16-bit Timers	8-bit Timer	Brown Out Detector	Ext Interrupts	Hardware Multiplier	Interrupts
AT90CAN128 Automotive	128	4096	4096	53	16	2,7-5,5	8	Yes	2	2	Yes	8	Yes	34
AT90CAN32 Automotive	32	1024	2048	53	16	2,7-5,5	8	Yes	2	2	Yes	8	Yes	34
AT90CAN64 Automotive	64	2048	4096	53	16	2,7-5,5	8	Yes	2	2	Yes	8	Yes	34
ATmega164P Automotive	16	512	1024	32	16	2,7-5,5	8	Yes	1	2	Yes	32	Yes	31
ATmega168 Automotive	16	512	1024	23	16	2,7-5,5	8	Yes	1	2	Yes	26	Yes	26
ATmega169P Automotive	16	512	1024	54	16	2,7-5,5	8	Yes	1	2	Yes	17	Yes	23
ATmega324P Automotive	32	1024	2048	32	16	2,7-5,5	8	Yes	1	2	Yes	32	Yes	31
ATmega32C1 Automotive	32	1024	2048	27	16	2,7-5,5	11	4	Yes	Yes	Yes	4	Yes	31
ATmega32M1 Automotive	32	1024	2048	27	16	2,7-5,5	11	4	Yes	Yes	Yes	4	Yes	31
ATmega48 Automotive	4	256	512	23	16	2,7-5,5	8	Yes	1	2	Yes	26	Yes	26
ATmega644P Automotive	64	2048	4096	32	16	2,7-5,5	8	Yes	1	2	Yes	32	Yes	32
ATmega64C1 Automotive	64	2048	4096	27	16	2,7-5,5	11	4	Yes	Yes	Yes	4	Yes	31
ATmega64M1 Automotive	64	2048	4096	27	16	2,7-5,5	11	4	Yes	Yes	Yes	4	Yes	31
ATmega88 Automotive	8	512	1024	23	16	2,7-5,5	8	Yes	1	2	Yes	26	Yes	26
ATtiny167 Automotive	16	512	512	16	16	2,7-5,5	11	Yes	1	1	Yes	16	--	20
ATtiny24 Automotive	2	128	128	12	16	2,7-5,5	8	Yes	1	1	Yes	12	--	17
ATtiny25 Automotive	2	128	128	6	16	2,7-5,5	4	Yes	--	2	Yes	7	--	15
ATtiny261 Automotive	2	128	128	16	16	2,7-5,5	11	Yes	1	2	Yes	2	--	19
ATtiny44 Automotive	4	256	256	12	16	2,7-5,5	8	Yes	1	8	Yes	12	--	17
ATtiny45 Automotive	4	256	256	6	16	2,7-5,5	4	Yes	--	2	Yes	7	--	15
ATtiny461 Automotive	4	256	256	16	16	2,7-5,5	11	Yes	1	2	Yes	2	--	19
ATtiny84 Automotive	8	512	512	12	16	2,7-5,5	8	Yes	1	1	Yes	12	--	17
ATtiny85 Automotive	8	512	512	6	16	2,7-5,5	4	Yes	--	2	Yes	7	--	15
ATtiny861 Automotive	8	512	512	16	16	2,7-5,5	11	Yes	1	2	Yes	2	--	19
ATtiny87 Automotive	8	512	512	16	16	2,7-5,5	11	Yes	1	1	Yes	16	--	20

Devices	On Chip Oscillator	PWM Channels	RTC	Self Program Memory	SPI	TWI	UART	Watchdog
AT90CAN128 Automotive	Yes	8	Yes	Yes	Yes	--	2	Yes
AT90CAN32 Automotive	Yes	8	Yes	Yes	Yes	--	2	Yes
AT90CAN64 Automotive	Yes	8	Yes	Yes	Yes	--	2	Yes
ATmega164P Automotive	Yes	6	Yes	Yes	1+USART	Yes	2	Yes
ATmega168 Automotive	Yes	6	Yes	Yes	1+USART	Yes	1	Yes
ATmega169P Automotive	Yes	4	Yes	Yes	Yes	USI	1	Yes
ATmega324P Automotive	Yes	6	Yes	Yes	1+USART	Yes	2	Yes
ATmega32C1 Automotive	Yes	4	--	Yes	Yes	Yes	1	Yes
ATmega32M1 Automotive	Yes	10	--	Yes	Yes	Yes	1	Yes
ATmega48 Automotive	Yes	6	Yes	Yes	1+USART	Yes	1	Yes
ATmega644P Automotive	Yes	6	Yes	Yes	1+USART	Yes	2	Yes
ATmega64C1 Automotive	Yes	10	--	Yes	Yes	Yes	1	Yes
ATmega64M1 Automotive	Yes	4	--	Yes	Yes	Yes	1	Yes
ATmega88 Automotive	Yes	6	Yes	Yes	1+USART	Yes	1	Yes
ATtiny167 Automotive	Yes	4	Yes	Yes	1+USI	No	Yes	Yes
ATtiny24 Automotive	Yes	4	--	Yes	USI	USI	--	Yes
ATtiny25 Automotive	Yes	4	--	Yes	USI	USI	--	Yes
ATtiny261 Automotive	Yes	2	--	Yes	Yes	USI	--	Yes
ATtiny44 Automotive	Yes	4	--	Yes	USI	Yes	--	Yes
ATtiny45 Automotive	Yes	4	--	Yes	USI	USI	--	Yes
ATtiny461 Automotive	Yes	2	--	Yes	Yes	USI	--	Yes
ATtiny84 Automotive	Yes	4	--	Yes	USI	USI	--	Yes
ATtiny85 Automotive	Yes	4	--	Yes	USI	USI	--	Yes
ATtiny861 Automotive	Yes	2	--	Yes	Yes	USI	--	Yes
ATtiny87 Automotive	Yes	4	Yes	Yes	1+USI	No	Yes	Yes

Mikroprocesory od firmy Microchip jsou vypsány v Tab. 7 [5].

Tab. 7 Výběr procesorů obsahujících CAN modul firmy Microchip

PIC18Fxx8x												
Devices	Flash (Kbytes)	EEPROM (Bytes)	RAM (Bytes)	Max I/O Pins	F.max (MHz)	Vcc (V)	Digital Communications	Type of CAN module	Capture/Compare/ PWM Peripherals	16-bit Timers	8-bit Timer	Self Program Memory
PIC18F2480	16	256	768	25	40	2,0-5,5	1 -A/E/USART 1 - MSSP(SPI/I2C)	1 x ECAN	1 -CCP 10-bit PWM resolutions	3	1	Yes
PIC18F2580	32	256	1536	25	40	2,0-5,5	2 -A/E/USART 1 - MSSP(SPI/I2C)	1 x ECAN	1 -CCP 10-bit PWM resolutions	3	1	Yes
PIC18F2585	48	1024	3328	25	40	2,0-5,5	3 -A/E/USART 1 - MSSP(SPI/I2C)	1 x ECAN	1 -CCP 10-bit PWM resolutions	3	1	Yes
PIC18F2680	64	1024	3328	25	40	2,0-5,5	4 -A/E/USART 1 - MSSP(SPI/I2C)	1 x ECAN	1 -CCP 10-bit PWM resolutions	3	1	Yes
PIC18F2682	80	1024	3328	25	40	2,0-5,5	5 -A/E/USART 1 - MSSP(SPI/I2C)	1 x ECAN	1 -CCP 10-bit PWM resolutions	3	1	Yes
PIC18F2685	96	1024	3328	25	40	2,0-5,5	6 -A/E/USART 1 - MSSP(SPI/I2C)	1 x ECAN	1 -CCP 10-bit PWM resolutions	3	1	Yes
PIC18F4480	16	256	768	36	40	2,0-5,5	7 -A/E/USART 1 - MSSP(SPI/I2C)	1 x ECAN	1 -CCP 1 -ECCP 10-bit PWM resolutions	3	1	Yes
PIC18F4580	32	256	1536	36	40	2,0-5,5	8 -A/E/USART 1 - MSSP(SPI/I2C)	1 x ECAN	1 -CCP 1 -ECCP 10-bit PWM resolutions	3	1	Yes
PIC18F4585	48	1024	3328	36	40	2,0-5,5	9 -A/E/USART 1 - MSSP(SPI/I2C)	1 x ECAN	1 -CCP 1 -ECCP 10-bit PWM resolutions	3	1	Yes
PIC18F4680	64	1024	3328	36	40	2,0-5,5	10 -A/E/USART 1 - MSSP(SPI/I2C)	1 x ECAN	1 -CCP 1 -ECCP 10-bit PWM resolutions	3	1	Yes
PIC18F4682	80	1024	3328	36	40	2,0-5,5	11 -A/E/USART 1 - MSSP(SPI/I2C)	1 x ECAN	1 -CCP 1 -ECCP 10-bit PWM resolutions	3	1	Yes
PIC18F4685	96	1024	3328	36	40	2,0-5,5	12 -A/E/USART 1 - MSSP(SPI/I2C)	1 x ECAN	1 -CCP 1 -ECCP 10-bit PWM resolutions	3	1	Yes

Základní výběrová kritéria byla:

- ✓ integrovaný CAN modul,
- ✓ dostatek vnitřní programové a datové paměti,
- ✓ min. 6 kanálů pro zdroje vnějšího přerušení,
- ✓ min. 20 I/O pinů.

Vzhledem ke špatné dostupnosti mikroprocesorů firmy Atmel, byl zvolen mikroprocesor PIC18F4685 firmy Microchip. Tento mikroprocesor dodávaný ve 40 pinovém PDIP viz Obr. 7, 44 pinovém TQFP nebo 44 pinovém QFN balení je vybaven 96 kB vnitřní programové flash paměti (s adresovatelným prostorem až 2 MB), 1024 B EEPROM paměti, 3328 B RAM paměti [6]. Dalšími vlastnostmi mikroprocesoru jsou:

- ✓ režimy řízení spotřeby:
  - chod procesoru – CPU zapnuto, periferie zapnuty,
  - nečinný režim – CPU vypnuto, periferie zapnuty (spotřeba proudu běžně klesá na 5,8 µA),
  - režim spánku – CPU vypnuto, periferie vypnuty (spotřeba proudu běžně klesá na 0,1 µA),
- ✓ flexibilní struktura oscilátoru:
  - čtyři krystalové režimy s frekvencí až 40 MHz,
  - PLL – uzavřená fázová smyčka – dostupné pro krystalové a interní oscilátory,

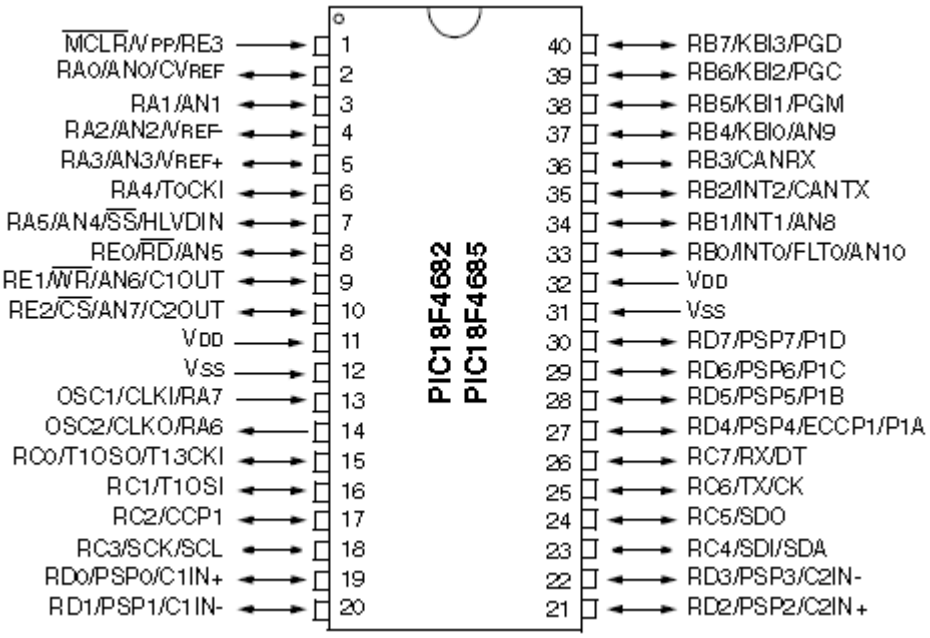
- dva režimy externího RC oscilátoru až 4 MHz,
- dva režimy externího zdroje času až 40 MHz,
- interní oscilátorový blok (8 uživatelsky volitelných frekvencí od 31 kHz do 8 MHz poskytuje kompletní oblast kmitočtů od 31 kHz do 32 MHz při použití PLL),
- sekundární oscilátor pomocí časovače 1 s frekvencí 32 kHz,
- hlídač poruch časového signálu,

✓ speciální rysy mikroprocesoru:
- architektura optimalizovaná pro použití kompilátoru programovacího jazyka C s volitelnou rozšířenou instrukční sadou,
- programová flash paměť se sto tisíci cykly zápisu nebo mazání s udržitelností dat větší než 40 let,
- datová EEPROM paměť s milionem cyklů zápisu nebo mazání s udržitelností dat větší než 40 let,
- softwarově samoprogramovatelný,
- prioritní úrovně přerušení,
- jednocyklový hardwarový násobitel 8 x 8,
- rozšířený watchdog časovač (WDT) – programovatelná perioda od 41 ms do 131 s,
- dvou pinové pěti voltové sériové programování v zapojeném obvodu (ICSP),
- dvou pinové ladění v zapojeném obvodu (ICD),
- pásmo pracovního napětí 2 – 5,5 V,

✓ důležité vlastnosti periferií:
- vysokoproudový zdroj/spotřebič, 25 mA,
- tři externí zdroje přerušení,
- modul použitelný jako zdroj pulzně šířkové modulace, komparátor nebo vzorkovač (CCP1),
- rozšířený CCP1 modul (1, 2 nebo 4 PVM výstupy, volitelná polarita, programovatelná doba snímání, automatické vypnutí a restartování),
- synchronně řízený sériový port (MSSP) obsahující třívodičové SPI (4 módy) a I^2C s módy vysílače i přijímače,
- modul rozšířeného adresovatelného USART – podporuje RS-485, RS-232 (používající vnitřní oscilátorový blok bez potřeby externího krystalu), LIN 1.3, automatické probuzení při start bitu a automatickou detekci přenosové rychlosti,
- deseti bitový ADC převodník s možností využití až 11 kanálů s rychlostí vzorkování až 100 000 vzorků/s,
- duální analogové komparátory s přepínaným vstupem,

✓ vlastnosti rozšířeného CAN modulu (ECAN):
- přenosová rychlost zpráv až 1 $Mb.s^{-1}$,
- vyhovuje specifikaci CAN 2.0 B aktivní,
- plně zpětně kompatibilní s vestavěnými CAN moduly PIC 18xxx8,
- tři operační režimy,
- tři vyhrazené vysílací zásobníky s nastavením priorit,
- dva vyhrazené zásobníky pro příjem,
- šest programovatelných zásobníků pro příjem nebo vysílání,

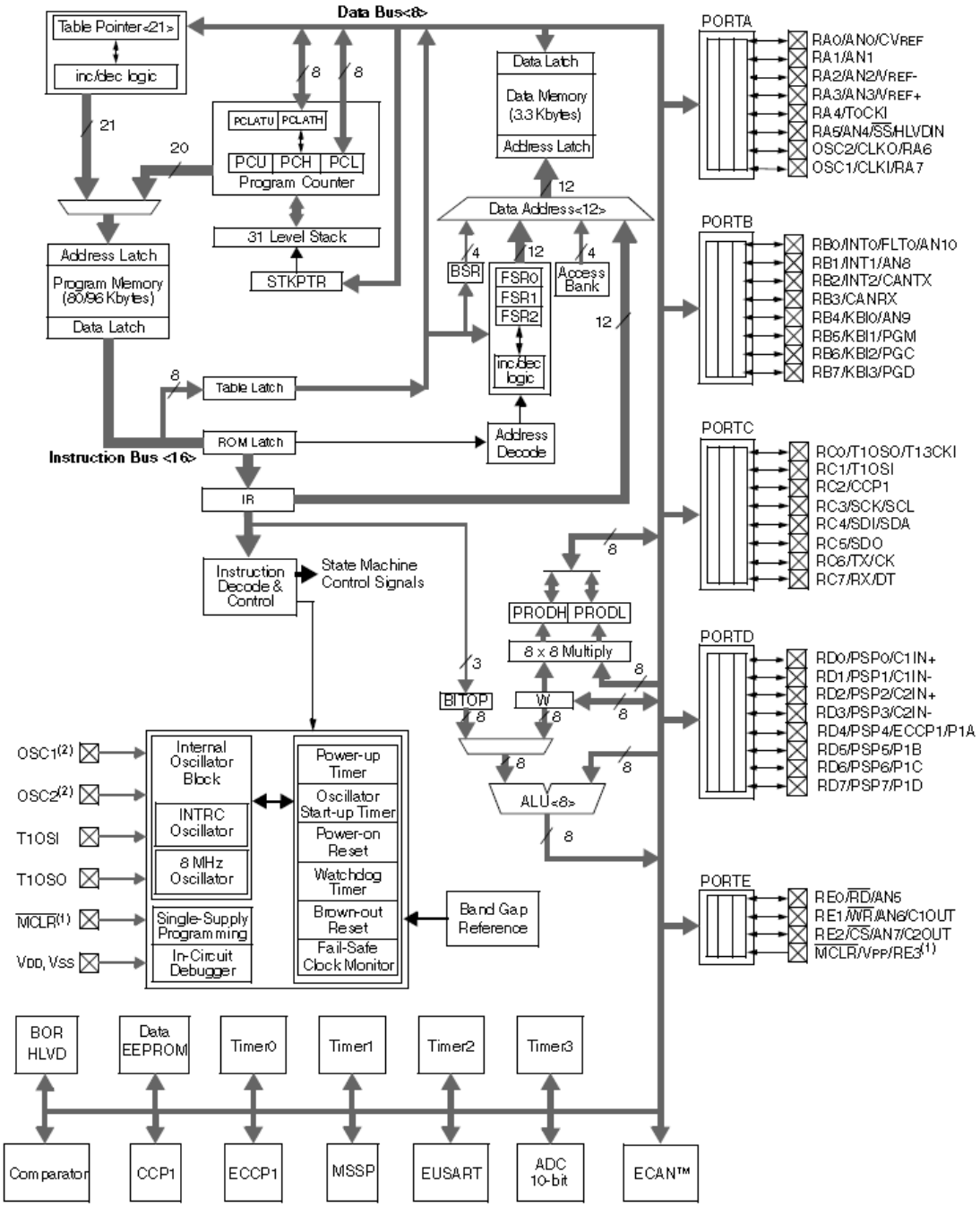
- tři plné 29 bitové přijímací masky,
- šestnáct plných 29 bitových přijímacích filtrů s dynamickým přiřazováním,
- automatické odpovídání na zprávy,
- rozšířené řízení chyb.



Obr. 7 Čtyřiceti pinový PDIP [6]

4.1. Popis vstupně výstupních pinů

Strukturální vazby vybraného mikroprocesoru schematicky vyjadřuje Obr. 8



Note 1: RE3 is multiplexed with $\overline{MCLR}$ and is only available when the $\overline{MCLR}$ Resets are disabled.

2: OSC1/CLKI and OSC2/CLKO are only available in select oscillator modes and when these pins are not being used as digital I/O. Refer to **Section 2.0 "Oscillator Configurations"** for additional information.

Obr. 8 Blokové schéma interních vazeb mikroprocesoru [6]

Detailní popis všech pinů mikroprocesoru PIC 18F4685 je dostupný v manuálu volně přístupném na webových stránkách výrobce [6]. V rámci této diplomové práce budou popsány pouze piny, které byly funkčně využity pro její řešení. Tyto piny jsou sestaveny do Tab. 8:

Tab. 8 Popis pinů použitých v diplomové práci

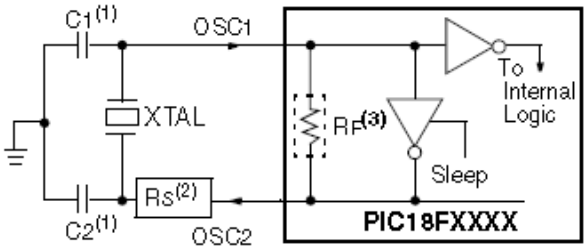
Označení pinu	Číslo pinu	Typ pinu	Popis pinu
RA6/OSC2	14	výstupní	Výstup krystalového oscilátoru připojený ke zdroji hodinového času (krystalu).
RA7/OSC1	13	vstupní	Vstup krystalového oscilátoru připojený ke zdroji hodinového času (krystalu).
RB0/INT0	33	vstupní	Digitální vstup připojený k ovládacímu tlačítku „stop“, generující přerušení 0.
RB1/INT1	34	vstupní	Digitální vstup připojený k ovládacímu tlačítku „enter“, generující přerušení 1.
RB2/CANTX	35	výstupní	Digitální výstup pro vysílání CAN zpráv připojený k řadiči CAN sběrnice.
RB3/CANRX	36	vstupní	Digitální vstup pro přijímání CAN zpráv připojený k řadiči CAN sběrnice.
RB4/KBI0	37	vstupní	Digitální vstup připojený k ovládacímu tlačítku „vlevo“, generující přerušení.
RB5/KBI1	38	vstupní	Digitální vstup připojený k ovládacímu tlačítku „vpravo“, generující přerušení.
RB6/KBI2	39	vstupní	Digitální vstup připojený k ovládacímu tlačítku „nahoru“, generující přerušení.
RB7/KBI3	40	vstupní	Digitální vstup připojený k ovládacímu tlačítku „dolů“, generující přerušení.
RC0	15	výstupní	Digitální výstup určený k řízení LCD displeje připojený na pin RS.
RC1	16	výstupní	Digitální výstup určený k řízení LCD displeje připojený na pin E.
RD0	19	výstupní	Digitální výstupní piny datové sběrnice pro LCD displej.
RD1	20		
RD2	21		
RD3	22		
RD4	27		
RD5	28		
RD6	29		
RD7	30		
V_{SS}	12, 31	napájení	Referenční zemnicí pin pro logické a vstupně-výstupní piny.
V_{DD}	11, 32	napájení	+ 5 V

4.2. Nastavení oscilátorů

Mikroprocesor PIC 18F4685 může pracovat v deseti rozdílných režimech nastavení oscilátoru. Uživatel si může vybrat jeden z následujících režimů:

1. LP krystal s nízkou energetickou náročností
2. XT krystal nebo rezonátor
3. HS vysokofrekvenční krystal nebo rezonátor
4. HSPLL vysokofrekvenční krystal nebo rezonátor s povolenou PLL funkcí
5. RC externí RC člen se čtvrtinovou frekvencí oscilátoru na výstupním pinu RA6
6. RCIO externí RC člen s volným pinem RA 6
7. INTIO1 vnitřní oscilátor se čtvrtinovou frekvencí na výstupním pinu RA6 a volným pinem RA7
8. INTIO2 vnitřní oscilátor s volnými piny RA6 a RA7
9. EC externí zdroj hodinového času se čtvrtinovou výstupní frekvencí
10. ECIO externí zdroj hodinového času s volným pinem RA6

V HS režimu je krystal nebo keramický rezonátor připojen na piny OSC1 a OSC2. Připojení pinů je znázorněné na Obr. 9 a výběr vhodného kondenzátoru pro dané zapojení viz Tab. 9.



Note 1: See Table 2-1 and Table 2-2 for initial values of C1 and C2.
2: A series resistor (RS) may be required for AT strip cut crystals.
3: RF varies with the oscillator mode chosen.

Obr. 9 Připojení krystalu nebo keramického rezonátoru [6]

Tab. 9 Výběr kondenzátoru pro krystalový oscilátor [6]

Osc Type	Crystal Freq	Typical Capacitor Values Tested:	
		C1	C2
LP	32 kHz	33 pF	33 pF
	200 kHz	15 pF	15 pF
XT	1 MHz	33 pF	33 pF
	4 MHz	27 pF	27 pF
HS	4 MHz	27 pF	27 pF
	8 MHz	22 pF	22 pF
	20 MHz	15 pF	15 pF

Capacitor values are for design guidance only.

These capacitors were tested with the crystals listed below for basic start-up and operation. **These values are not optimized.**

Different capacitor values may be required to produce acceptable oscillator operation. The user should test the performance of the oscillator over the expected VDD and temperature range for the application.

See the notes following this table for additional information.

Crystals Used:	
32 kHz	4 MHz
200 kHz	8 MHz
1 MHz	20 MHz

4.3. Vnitřní struktura paměti

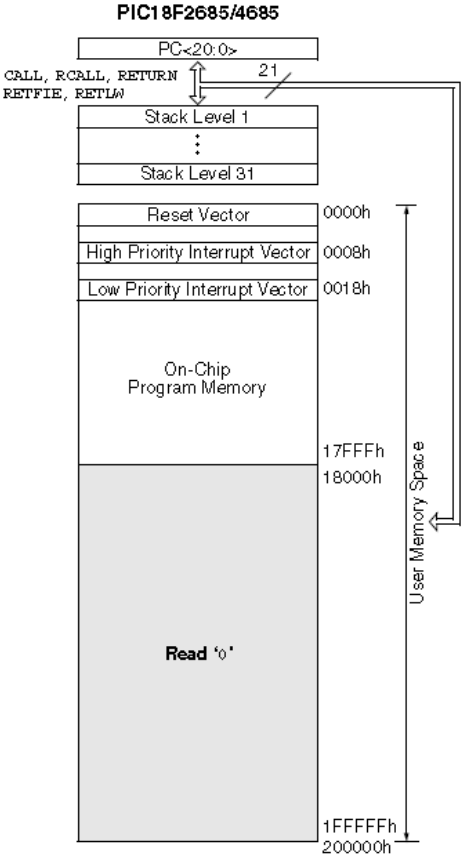
Mikroprocesor obsahuje tři různé druhy paměti:

1. programová paměť (flash),
2. datová paměť (RAM),
3. datová paměť (EEPROM).

Procesor používá harvardskou architekturu, tzn., že programové a datové paměti mají oddělené sběrnice, což umožňuje současný přístup do dvou paměťových prostorů. Z praktického hlediska můžeme datovou EEPROM považovat za periferní zařízení, protože pro přístup a adresaci využívá samostatnou sadu ovládacích registrů [6].

4.4. Organizace programové paměti

Pro přístup do programové paměti je implementován 21 bitový programový čítač, který je schopen adresovat až dva MB paměťového prostoru. Přístup do prostoru mezi horní hranici fyzicky implementované paměti a 2 MB adresy vrátí vždy nulu. Mikroprocesor obsahuje 96 kB flash paměti a může ukládat až 49152 šestnácti bitových instrukcí. Zařízení ze skupiny PIC18 mají dva vektory přerušení na adresách 0008h a 0018h a reset vektor na adrese 0000h. Mapa programové paměti pro PIC18F4685 je zobrazená na Obr. 10.



*Obr. 10 Mapa programové paměti a zásobníku pro PIC*18F468 [6]

Programová paměť je adresovatelná po jednotlivých bytech, ale instrukce v ní uložené jsou dvou nebo čtyř bytové. Nejméně významný byt instrukčního slova je vždy uložen v oblasti programové paměti se sudou adresou. Aby bylo možné udržet zarovnání s instrukční hranicí, inkrementuje programový čítač svůj krok o hodnotu 2. Ukládání instrukcí v programové paměti znázorňuje Obr. 11.

			LSB = 1	LSB = 0	Word Address ↓
	Program Memory Byte Locations →				000000h
					000002h
					000004h
					000006h
Instruction 1:	MOVLW	055h	0Fh	55h	000008h
Instruction 2:	GOTO	0006h	EFh	03h	00000Ah
			F0h	00h	00000Ch
Instruction 3:	MOVFF	123h, 456h	C1h	23h	00000Eh
			F4h	56h	000010h
					000012h
					000014h

Obr. 11 Uložení instrukcí v programové paměti [6]

Flash programovou paměť lze číst, zapisovat a mazat s použitím standardní operace v celém rozsahu vstupního napětí. Čtení z paměti je prováděno po 1 bytu, zatímco zápis a mazání do paměti jsou prováděny po blocích o velikosti 64 bytů. Zápis nebo mazání programové paměti zastaví provádění instrukcí, dokud není operace ukončena. V tomto čase není paměť přístupná, proto nelze provádět programový kód. Zápis a mazání ukončuje vnitřní časovač.

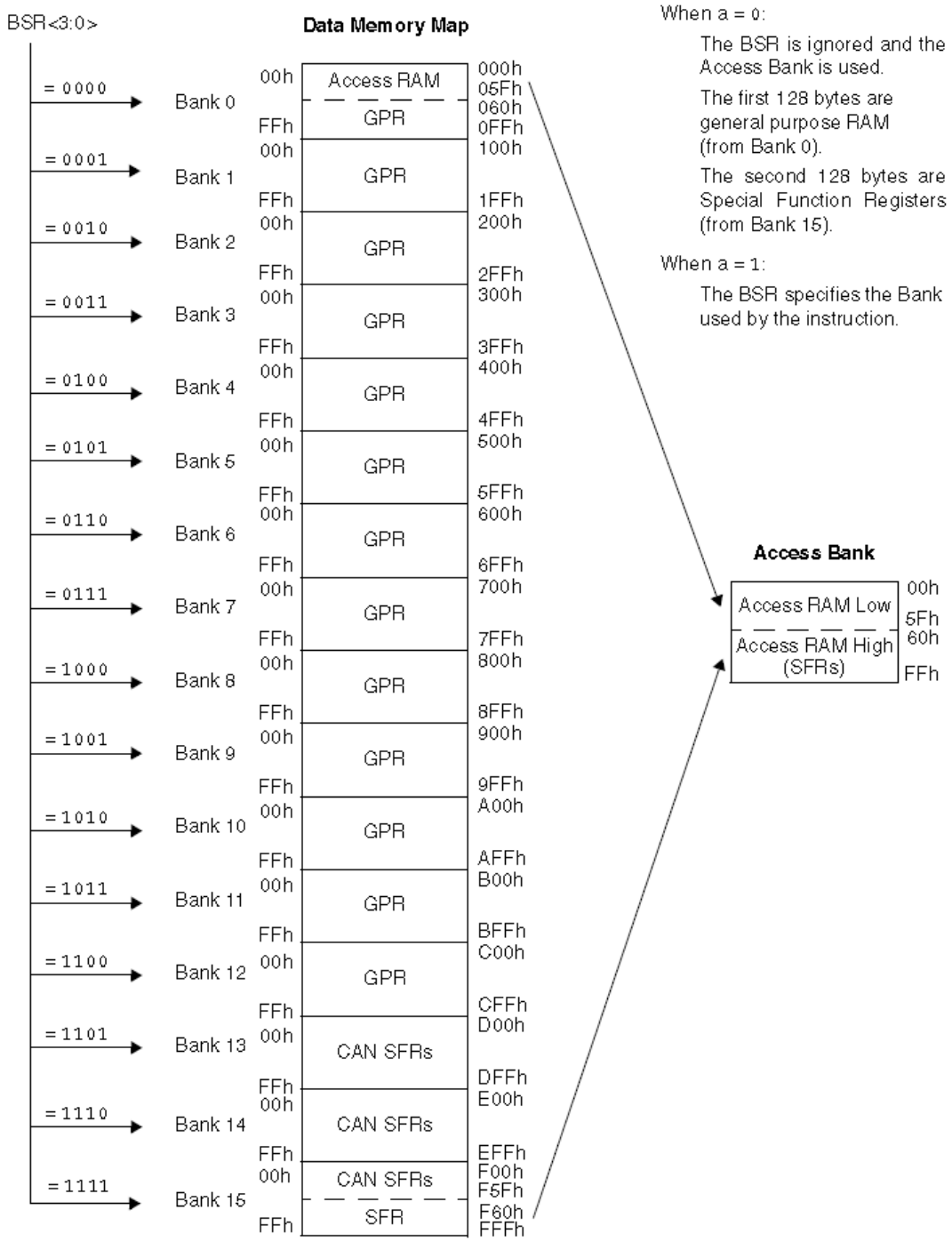
Zapisovaná hodnota do paměti nemusí mít smysl instrukce. V případě, že prováděná instrukce programového kódu nemá správný tvar, je provedena instrukce NOP. Pro práci s programovou pamětí jsou určeny registry EECON1, EECON2, TABLAT a TBLPTR. Vzhledem k uvedeným skutečnostem, lze čtení z paměti provádět přímo, zatím co zápis a mazání dat vyžaduje přídavný uživatelský kód.

4.5. Organizace datové paměti (RAM)

Datová paměť v zařízeních PIC18 je implementována jako statická RAM. Každý registr v datové paměti má 12 bitový registr umožňující adresovat až 4096 bytů datové paměti. Paměťový prostor je rozdělen do 16 bank, z nichž každá obsahuje 256 bytů. Organizace datové paměti je vyobrazena na Obr. 12.

Datová paměť dále obsahuje registry speciálních funkcí (SFR) a registry pro běžné použití (GPR). Registry SFR jsou používány pro ovládání a kontrolu stavu mikroprocesoru a periferních funkcí, zatímco GPR jsou užívány pro ukládání dat a pracovních dat jednotlivých operací v uživatelských aplikacích. Jakékoliv čtení neimplementované paměti je čteno jako 0.

Instrukční sada a architektura umožňuje práci se všemi bankami. Přístup k celé datové paměti je umožněn ve třech režimech: přímého, nepřímého a indexovaného adresování. Zařízení PIC18 implementují přístupovou banku z důvodu zajištění přístupu k běžně užívaným registrům během jednoho cyklu. Tato banka je 256 bytů široká a poskytuje rychlý přístup do registrů speciálních funkcí a do spodní poloviny GPR banky 0 bez použití BSR (registr pro výběr banky) [6].



Obr. 12 Mapa datové paměti [6]

4.5.1. Organizace datové paměti EEPROM

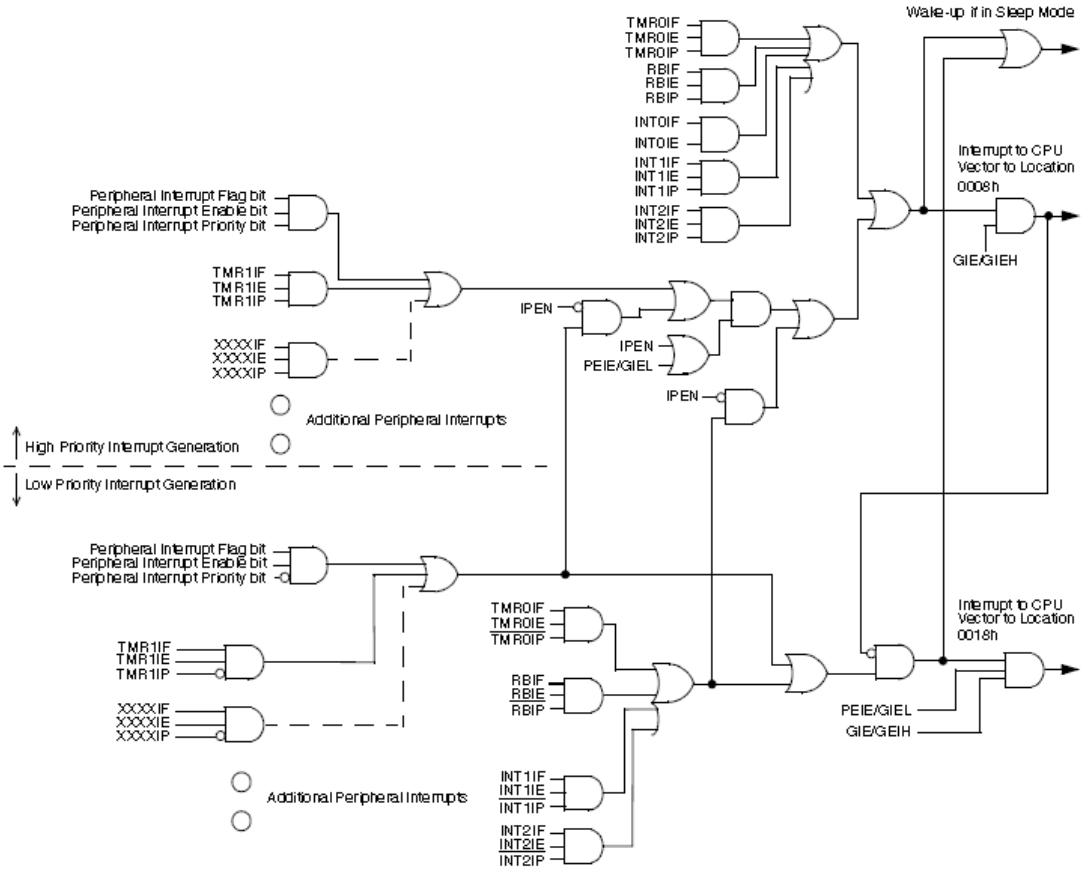
Datová EEPROM je oblast paměti udržující data i při výpadku napájení, oddělená od datové RAM a programové flash paměti. Používá se pro dlouhodobé skladování programových dat a není mapovaná v souborovém registru ani v programové paměti, protože je nepřímo adresovatelná pomocí registru speciálních funkcí. Z paměti lze číst nebo do ní zapisovat pomocí standardních operací v celém rozsahu vstupního napětí. Pro práci s EEPROM se využívá pět registrů: EECON1, EECON2, EEDATA, EEADR a EEADRH.

4.6. Přerušení

PIC18F4685 obsahuje několik zdrojů přerušení s rozdělením do dvou prioritních úrovní (vysoká a nízká). Vektor vysoké úrovně priority je umístěn na adrese 000008h a nízké priority na adrese 000018h. Přerušení s vyšší prioritou má za následek přerušení provádění programového kódu s nižší prioritou. Pro ovládání operací přerušení je vyhrazeno 13 registrů: RCON, INTCON, INTCON2, INTCON3, PIR1, PIR2, PIR3, PIE1, PIE2, PIE3, IPR1, IPR2 a IPR3. Je doporučováno použití hlavičkových souborů firmy Microchip dodávané s MPLAB® IDE pro symbolické pojmenování bitů těchto registrů což umožňuje kompilátoru starat se o automatické umístění bitů uvnitř specifikovaného registru. Každý zdroj přerušení má tři bity pro ovládání jeho chování. Funkce těchto bitů jsou:

- ✓ příznak – indikuje skutečnost vzniku přerušení,
- ✓ povolení – umožňuje provedení odskoku na adresu vektoru přerušení, jestliže je příznak nastaven,
- ✓ priorita – slouží pro výběr mezi vysokou a nízkou úrovní.

Používání priorit je povoleno nastavením IPEN bitu v RCON. Jestliže jsou povoleny prioritní úrovně, pak se používají dva bity pro globální ovládání těchto přerušení. Nastavení GIEH bitu v INTCON registru povolí všechna přerušení s vysokou prioritou a nastavení GIEL bitu v INTCON registru povolí všechna přerušení s nízkou prioritou. Jestliže jsou nastaveny bity příznaku přerušení, povolení a příslušného povolení globálního přerušení pak přerušení přejde bezprostředně na adresu 000008h nebo 000018h závisející na nastavení bitu priority. Jednotlivá přerušení mohou být zakázána pomocí jejich odpovídajících bitů povolení. Pokud je IPEN bit nastaven na nulu (defaultní nastavení) jsou prioritní úrovně zakázány a přerušení je kompatibilní s PIC zařízeními střední úrovně. V kompatibilním módu nemají jednotlivé prioritní bity žádný efekt. V tomto případě se z GIEL bitu stává PEIE bit, který v INTCOM registru povoluje/zakazuje všechny periferní zdroje přerušení a z GIEH bitu se stává GIE bit, který povoluje/zakazuje všechny zdroje přerušení [6]. Všechna přerušení pak odskakují na adresu 000008h. Schéma přerušení viz Obr. 13.



Obr. 13 Logické schéma přerušení [6]

4.6.1. Přerušení na pinech INTx

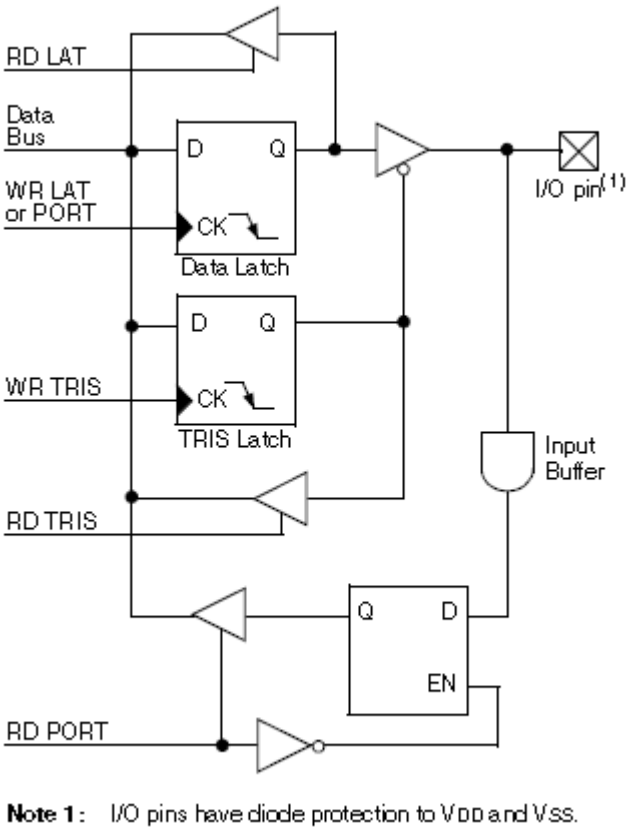
Externí přerušení na pinech RB0/INT0, RB1/INT1 a RB2/INT2 jsou přerušení s přepínanou hranou vzniku. V případě, že odpovídající INTEDGx bit v INTCON2 registru je nastaven na hodnotu 1 je přerušení přepnuto náběžnou hranou signálu, jestliže je příslušný bit nastaven na hodnotu 0 je přerušení přepnuto na spádovou hranu signálu. Vyskytne-li se odpovídající hrana signálu na pinu RBx/INTx je příslušný bit příznaku INTxIF nastaven na hodnotu 1. Toto přerušení může být zakázáno nastavením odpovídajícího bitu povolení INTxIE na hodnotu 0. Bit příznaku INTxIF musí být nastaven na hodnotu 0 v obslužné rutině přerušení před znovu povolením přerušení. Prioritní úroveň přerušení pro INT1 a INT2 je určena hodnotou obsaženou v prioritním bitu INT1IP a INT2IP v INTCON3 registru. Pro INT0 neexistuje prioritní bit. Toto přerušení je vždy přerušením s vysokou úrovní.

4.6.2. Přerušení portu B při změně stavu

Nastane-li změna vstupního stavu na portu B je nastaven bit příznaku RBIF v registru INTCON na hodnotu 1. Přerušení může být povoleno/zakázáno nastavením nebo vymazáním bitu povolení RBIE v registru INTCON. Úroveň priority přerušení pro port B je určena hodnotou obsaženou v prioritním bitu RBIP v registru INCON2.

4.7. Výstupní porty

PIC18F4685 obsahuje čtyři 8 pinové porty A, B, C a D a jeden čtyř pinový port E. Vlastnosti každého portu jsou řízeny třemi registry: TRIS (přepínání vstup/výstup), PORT (čtení hodnoty portu) a LAT (uzamčení výstupní hodnoty portu). Zjednodušený model obecného vstupně/výstupního portu bez převodníku na další periferie viz Obr. 14 [6].



Obr. 14 Operace obecného vstupně/výstupního portu [6]

4.8. Technologie ECAN

Vybraný mikroprocesor pro účely řešení této práce obsahuje sofistikovanější řadič CAN sběrnice. Tento ECAN modul je plně zpětně kompatibilní s CAN moduly dostupnými v PIC18CXX8 a PIC18FXX8 sériích. CAN modul je sériový převodník, který je využitelný pro komunikaci s ostatními mikrořadiči. Převodník a protokol byly navrženy pro komunikaci v prostředí s rušivými vlivy.

ECAN modul je komunikační řadič implementující protokol CAN 2.0A a B tak, jak je definováno ve specifikaci firmy BOSCH. Modul podporuje CAN 1.2, CAN 2.0A, CAN 2.0B pasivní a CAN 2.0B aktivní verzi tohoto protokolu. Modul má následující vlastnosti:

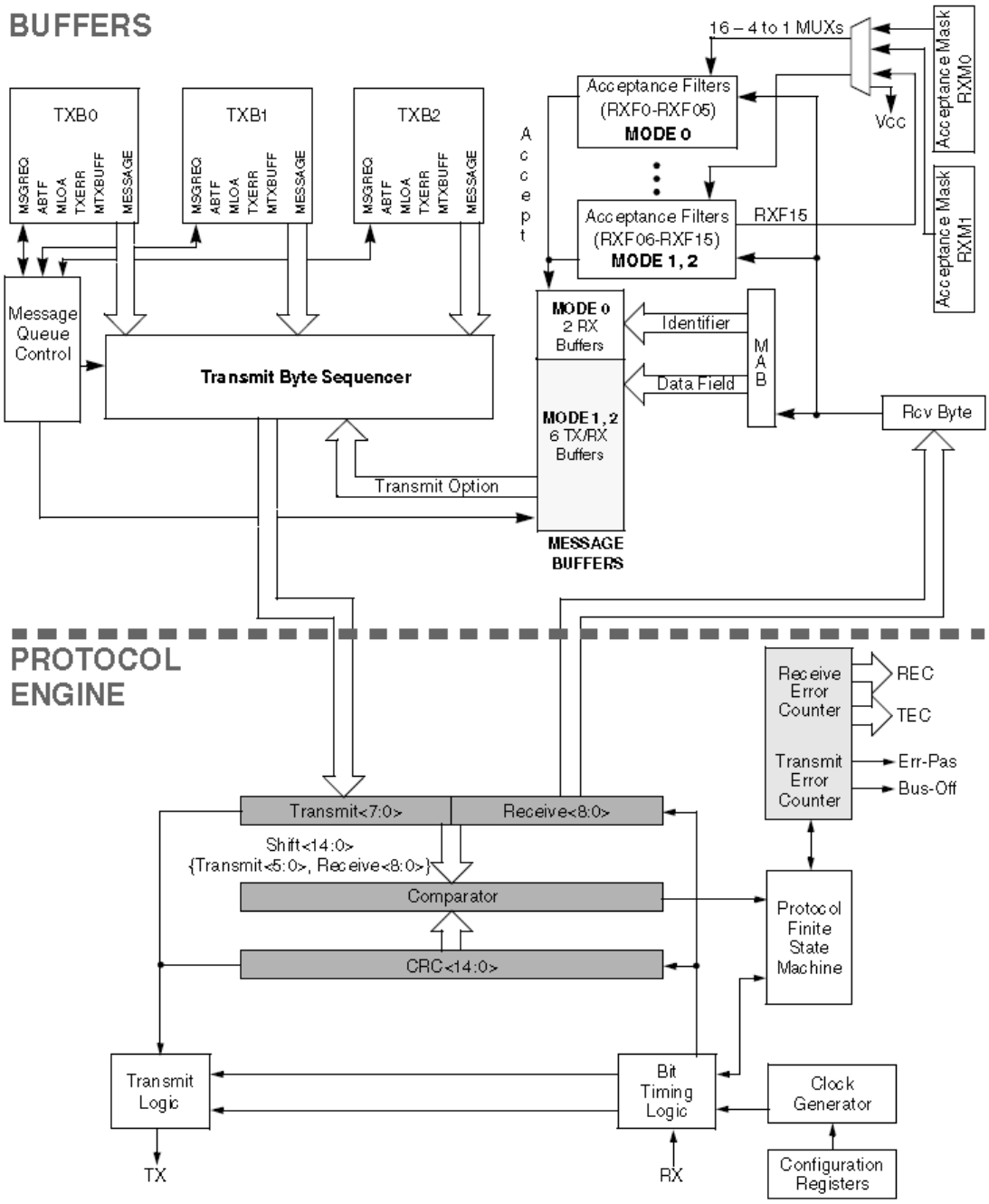
- ✓ podpora datového filtru Device Net™,
- ✓ standardní a rozšířené datové rámce,
- ✓ datová délka 0 – 8 bytů,

- ✓ programovatelná přenosová rychlost až 1 $Mb.s^{-1}$,
- ✓ tři funkční módy,
- ✓ podpora vzdálených rámců s automatickým držením,
- ✓ přijímač s dvojitým zásobníkem s priorizací přijímaných zpráv,
- ✓ šest programovatelných zásobníků pro příjem a vysílání zpráv,
- ✓ šestnáct plně akceptačních filtrů, které mohou být přiřazeny jedné ze čtyř masek,
- ✓ dvě masky akceptačních filtrů, které mohou být přiřazeny ke kterémukoli filtru,
- ✓ jeden akceptační filtr, který umožňuje použití jako masky akceptačního filtru nebo akceptačního filtru,
- ✓ tři zásobníky vyhrazené pro vysílání,
- ✓ programovatelný zdroj hodinového taktu.

Modul CAN sběrnice se stává z jádra protokolu, zásobníku zpráv a řízení viz Obr. 15. Jádro CAN protokolu automaticky drží všechny funkce pro přijímání a vysílání zpráv po CAN sběrnici. Zprávy jsou přenášeny po uložení dat do příslušných registrů. Stav a chyby mohou být ověřeny přečtením příslušných registrů. Každá zpráva detekovaná na sběrnici je ověřena neabsolvuje-li chyby a po té porovnána s filtry z důvodu rozhodnutí o přijetí a uložení zprávy v jednom ze dvou přijímacích registrů. CAN modul podporuje následující typy rámců:

- ✓ standardní datový rámec,
- ✓ rozšířený datový rámec,
- ✓ vzdálený rámec,
- ✓ chybový rámec,
- ✓ generování a detekování mezirámcového prostoru.

CAN modul používá piny RB2/CANTX a RB3/CANRX pro připojení na sběrnici. V normálním módu automaticky změní stav pinu TRISB2. Uživatel se musí přesvědčit, že pin TRISB3 je nastaven správně [6].



Obr. 15 Blokové schéma jádra protokolu a CAN zásobníků [6]

4.8.1. Operační módy CAN modulu

Mikroprocesory této řady mají vestavěno šest základních operačních módů:

1. konfigurační,
2. vypnuto,
3. operační,
4. naslouchání,
5. vratný,
6. rozpoznávání chyb.

Zadáme-li požadavek na změnu módu, procesor vyřídí změnu teprve po odeslání všech čekajících zpráv. Z tohoto důvodu musí uživatel ověřit, jestli se zařízení nastavilo do požadovaného módu dříve, než bude provádět další operace.

Ad 1) Konfigurační mód - CAN modul by měl být před prvním použitím inicializován, což je možné pouze v případě, že je modul v konfiguračním módu. V tomto módu lze nastavit konfigurační registry, registry akceptačních masek a registry akceptačních filtrů. Registry používané ke konfiguraci modulu nemůžou být měněny za chodu, čímž chrání modul před náhodnými chybami uživatele. CAN modul nemůže být přepnut do konfiguračního módu v případě, že probíhá přenos zpráv. V konfiguračním módu nemůže modul vysílat ani přijímat zprávy, čítač chyb je smazán a příznak chyb zůstává nezměněn. Programátor získává přístup do konfiguračních registrů, které mají v ostatních módech omezený přístup.

Ad 2) Mód vypnuto – v tomto módu CAN modul nevysílá ani nepřijímá zprávy. Funguje podobně jako vypínač jakékoliv jiné periferie, tzn., že interní hodiny modulu jsou zastaveny, dokud není modul aktivován. V případě, že je modul aktivní čeká na 11 recesivních bitů na CAN sběrnici, čímž detekuje, je-li sběrnice v nečinnosti a teprve po té se přepne do módu vypnuto. Mód umožňuje aktivovat funkci probuzení. Při této funkci se modul po aktivaci ze sběrnice automaticky přepne do operačního módu. Např. jestliže byl modul přepnut z operačního módu do módu vypnuto, při detekci aktivity na sběrnici je modul probuzen a automaticky přepnut do operačního módu. První zpráva, která probuzení způsobila je označena jako neplatná a je odstraněna. Operační modul negeneruje žádný chybový rámec. V módu vypnuto se vstupní a výstupní piny používané modulem stanou běžnými vstupně výstupními piny.

Ad 3) Operační mód – Je standardním módem CAN modulu. Zařízení v tomto módu aktivně monitorují všechny zprávy posílané po sběrnici a generují bity správného přijetí zpráv, chybové rámce atd. Jde o jediný mód, v kterém PIC18F4685 posílá zprávy po sběrnici.

Ad 4) Naslouchací mód – mód umožňuje přijímání všech zpráv včetně chybných. Používá se pro aplikace monitorující sběrnici nebo pro detekování přenosové rychlosti. Pro automatickou detekci přenosové rychlosti je nezbytné, aby na sběrnici byly připojeny dva uzly na víc, které mezi sebou navzájem komunikují. Přenosová rychlost může být určena empiricky testováním různých hodnot, dokud není přijata korektní zpráva. Jde o tzv. tichý mód, při kterém nejsou vysílány žádné zprávy včetně chybových rámců nebo bitů potvrzení správného přijetí. Filtry a masky mohou být použity k přijmutí specifických zpráv nebo mohou být nastaveny na hodnotu 0 pro příjem všech zpráv. Chybový čítač je v tomto stavu módu smazán a deaktivován.

Ad 5) Vratný mód – umožňuje vnitřní vysílání zpráv z vysílacích zásobníků do zásobníků přijímacích bez aktuálního vysílání zpráv na sběrnici. Bývá používán pro vývoj systémů a jejich testování. V tomto módu je ignorován bit potvrzení správného přijetí zprávy a modul umožňuje přijímání zpráv, které sám vysílá jako by šlo o zprávy vyslané jiným uzlem. Jde opět o tzv. tichý mód, při kterém nejsou vysílány žádné zprávy včetně chybových rámců nebo bitů potvrzení správného přijetí. V tomto módu se z CANTX pinu stává běžný vstupně výstupní pin. Filtry a masky mohou být použity k přijmutí specifických zpráv nebo mohou být nastaveny na hodnotu 0 pro příjem všech zpráv.

Ad6) Mód rozpoznávání chyb – přijímá všechny zprávy kromě zpráv chybných.

4.8.2. Funkční módy CAN modulu

Kromě šesti operačních módů popsaných v předchozí kapitole disponuje ECAN modul také třemi funkčními módy nazvanými mód 0, mód 1 a mód 2.

Mód 0 – je navržen tak, aby byl plně kompatibilní s CAN moduly používanými v PIC18CXX8 a PIC18FXX8. Jde o defaultní funkční mód. Všechny kódy napsané pro uvedené typy mikroprocesorů lze použít i pro tento ECAN modul bez nutnosti provedení úprav. V tomto módu jsou dostupné následující zdroje:

- ✓ tři vysílací zásobníky – TXB0, TXB1 a TXB2,
- ✓ dva přijímací zásobníky – RXB0 a RXB1,
- ✓ dvě akceptační masky – pro každý přijímací zásobník jedna – RXM0 a RXM1,
- ✓ šest akceptačních filtrů – dva pro RXB0 a čtyři pro RXB1 – RXF0 až RXF5.

Mód 1 – je podobný módu 0 s rozdílem většího počtu přístupných zdrojů. Obsahuje šestnáct akceptačních filtrů a dvě akceptační masky. Akceptační filtr 15 může být použit jako akceptační filtr nebo akceptační maska. Obsahuje navíc tři vysílací a dva přijímací zásobníky a dalších šest zásobníků zpráv. Každý z těchto šesti zásobníků může být naprogramován jako přijímací, vysílací nebo pro automatické odpovědi zpráv. Čtrnáct ze šestnácti akceptačních filtrů může být dynamicky přiřazeno ke kterémukoliv přijímacímu zásobníku a akceptační masce. Jestliže je přijímací zásobník naprogramován na používání standardních identifikátorů zpráv (CAN 2.0A) část registru akceptačního filtru může být použita jako datový filtr. Délka filtru je programovatelná od 0 do 18 bitů. Tato funkčnost zjednodušuje implementaci vysokoúrovňových protokolů jako je DeviceNet™ protokol. V tomto módu jsou dostupné následující zdroje:

- ✓ tři vysílací zásobníky – TXB0, TXB1 a TXB2,
- ✓ dva přijímací zásobníky – RXB0 a RXB1,
- ✓ šest programovatelných zásobníků s možností automatických odpovědí – B 0-5,
- ✓ šestnáct dynamicky přiřaditelných akceptačních filtrů – RXF0 až RXF15,
- ✓ dvě akceptační masky – RXM0, RXM1 a RXF15 s možností naprogramování jako třetí akceptační maska.

Mód 2 – umožňuje vytvoření přijímacího FIFO zásobníku ze dvou nebo více přijímacích zásobníků. Žádný z akceptačních filtrů není speciálně přiřazen některému z přijímacích zásobníků, proto každý filtr, který je povolen a přiřazen k FIFO přijímacímu zásobníku, může generovat přijetí, čímž dojde k aktualizaci FIFO zásobníku. Délka FIFO zásobníku je programovatelná s rozsahem od 2 do 8 přijímacích zásobníků a je určena prvním programovatelným zásobníkem, který je nakonfigurován jako vysílací. Např. Je-li zásobník B2 naprogramován jako vysílací, potom je FIFO zásobník sestaven z RXB0, RXB1, B0 a B1. Jeho délka tedy činí čtyři zásobníky. V tomto módu jsou dostupné následující zdroje:

- ✓ tři vysílací zásobníky – TXB0, TXB1 a TXB2,
- ✓ dva přijímací zásobníky – RXB0 a RXB1,
- ✓ šest programovatelných zásobníků s možností automatických odpovědí – B 0-5,
- ✓ šestnáct akceptačních filtrů – RXF0 až RXF15,
- ✓ dvě akceptační masky – RXM0, RXM1 a RXF15 s možností naprogramování jako třetí akceptační maska.

4.8.3. Nastavení přenosové rychlosti

Všechny uzly na CAN sběrnici musí používat stejnou nominální přenosovou rychlost. Specifikace této sběrnice používá kódování bez návratu do nulového stavu (NRZ), které neumí zakódovat hodinový takt do datového proudu. Přijatý hodinový takt musí být proto vrácen přijímacími uzly a synchronizován s hodinovým taktem vysílače. Vzhledem k rozlišnosti oscilátorů a přenosových časů jednotlivých uzlů musí přijímač používat nějaký typ PLL synchronizovaný s hranou datového přenosu pro získání synchronizace hodinového taktu. Vzhledem k tomu, že data jsou kódována NRZ, je nezbytné vložit bitovou vycpávku k zajištění výskytu hrany nejméně jedenkrát za šest bitů.

PIC18F4685 má implementován DPLL pro časování bitu, který je konfigurován pro synchronizaci příchozích dat a poskytuje nominální časování vysílaných dat. DPLL dělí každý bit do několika časových segmentů skládajících se z minimálních časových period zvaných časová kvanta (T_Q). Funkce časování sběrnice prováděné během bitového rámce, jako jsou synchronizace na lokální oscilátor, kompenzace prodlevy síťového přenosu a pozice vzorkovacího bodu, jsou definovány programovatelnou časovou logikou bitu DPLL.

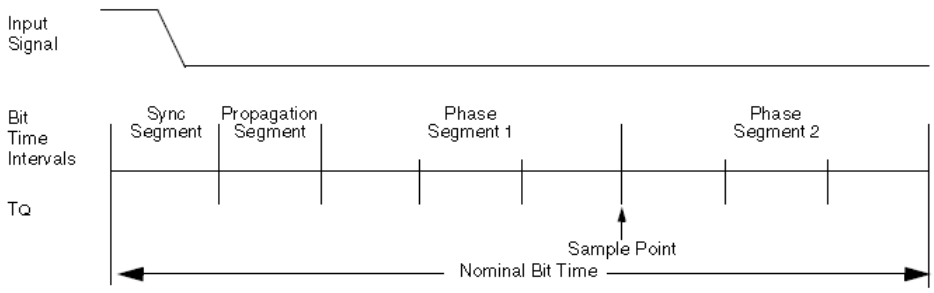
Zařízení na sběrnici musí používat stejnou přenosovou rychlost, ale nevyžadují stejnou časovou frekvenci hlavního oscilátoru. Pro různé hodinové frekvence jednotlivých zařízení by přenosová rychlost měla být zvyšována vhodným nastavením násobitele přenosové rychlosti (BRP) a počtu časových kvant v každém segmentu. Nominální přenosová rychlost je hodnota přenesených bitů za sekundu v případě ideálního vysílače s ideálním oscilátorem bez resynchronizace a může dosahovat hodnoty maximálně 1 $Mb.s^{-1}$. Nominální bitový čas je definován jako:

$$T_{BIT} = \frac{1}{Nominální\ přenosová\ rychlost}.$$

Uvažujeme-li, že nominální čas bitu je rozdělen do samostatných nepřekrývajících se časových segmentů. Jsou jimi:

- ✓ synchronizační segment (Sync_seg),
- ✓ propagační časový segment (Prop_seg),
- ✓ fázový segment 1 (Phase_seg1),
- ✓ fázový segment 2 (Phase_seg2).

Časové segmenty (a také nominální čas bitu) jsou vytvořeny z násobku časových kvant, viz Obr. 16.



Obr. 16 Rozdělení času bitu [6]

Podle definice je nominální čas bitu programovatelný minimálně od 8 do maximálně 25 T_Q a minimální nominální čas bitu je 1µs což odpovídá maximální přenosové rychlosti 1 Mb.s^{-1}. Aktuální nominální délka času bitu je dána následujícím vztahem:

$$Nominální\ čas\ bitu = T_Q \cdot (Sync_seg + Prop_seg + Phase_seg1 + Phase_seg2).$$

Časové kvantum je pevná jednotka odvozená z periody oscilátoru. Je také definována pomocí programového násobitele přenosové rychlosti, jehož číselná hodnota se pohybuje v rozmezí od 1 do 64. Matematicky je časové kvantum vyjádřeno rovnicí:

$$T_Q(\mu s) = \frac{2\cdot(BRP+1)}{F_{OSC}\ (MHz)} \quad nebo \quad T_Q(\mu s) = 2 \cdot (BRP + 1) \cdot T_{OSC}(\mu s),$$

kde:

F_{OSC} je frekvence hodinového taktu,
T_{OSC} je odpovídající perioda oscilátoru,
BRP nabývá hodnot od 0 do 63.

Výše zmíněná rovnice vyjadřuje efektivní frekvenci oscilátoru používanou mikroprocesorem. Např. Je-li použit 10 MHz krystal v HS módu, pak F_{OSC} je také rovna 10 MHz a T_{OSC} je rovno 100 ns. Jestliže tento 10 MHz krystal použijeme s HSPLL módem, pak efektivní frekvence F_{OSC} je rovna 40 MHz a T_{OSC} je rovno 25 ns [6].

4.8.3.1. Popis časového kvanta

Časová kvanta jsou fixní jednotky odvozené od periody oscilátoru a násobitele přenosové rychlosti. Frekvence oscilátoru v různých uzlech musí být koordinovány, aby byly schopny poskytovat systému správný nominální čas bitu. Nyní je nutné se zmínit, že počet časových kvant v rámci jednoho časového bitu je programovatelné od 4 do 25 a použitelné minimum je 8 T_Q. Není zaručeno, že bit s délkou menší než 8 T_Q bude pracovat správně.

4.8.3.2. Synchronizační segment

Jedná se o část bitu používaná k synchronizaci různých CAN uzlů na sběrnici. V průběhu doby synchronizačního segmentu je očekávána vstupní hrana signálu. Délka synchronizačního segmentu má pevnou délku 1 T_Q.

4.8.3.3. Propagační segment

Tato část bitu je určena ke kompenzaci fyzické časové prodlevy na sběrnici. Prodleva je složena ze signálního času linky sběrnice a vnitřní prodlevy uzlu. Délka tohoto segmentu může být programována od 1 do 8 T_Q.

4.8.3.4. Fázový segment

Používá se k optimálnímu určení vzorkovacího bodu přijímaného bitu v rámci nominálního časového bitu. Vzorkovací bod se nastavuje mezi fázový segment 1 a 2. Tyto segmenty mohou být prodlužovány nebo zkracovány při resynchronizačním procesu. Konec fázového segmentu 1 určuje místo vzorkování v rámci časového bitu. Fázový segment 1 je programovatelný v rozmezí od 1 do 8 T_Q. Fázový segment 2

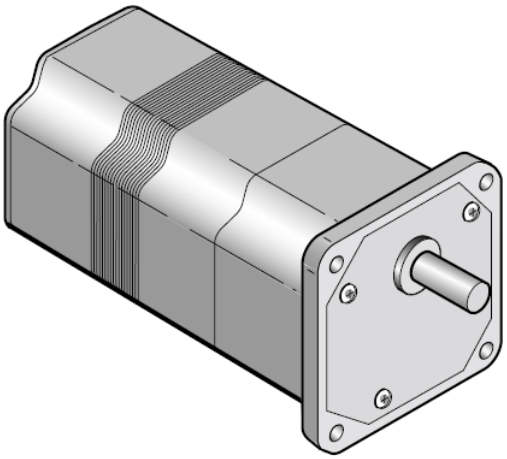
poskytuje prodlevu před dalším vysíláním dat a je taktéž programovatelný v rozmezí od 1 do 8 T_Q. Vzorkovací bod by neměl být nastaven na pozdější dobu než je přibližně 80 % bitového času.

4.8.3.5. Vzorkovací bod

Jde o časový bod, ve kterém je čtena hodnota sběrnice. Tímto je získána hodnota přijímaného bitu. Vzorkovací bod je nastaven na konci fázového segmentu 1. V případě, že je časování nastaveno příliš pomalu a obsahuje příliš mnoho časových kvant, je možné specifikovat více vzorkovacích bodů. Hodnotu pak určuje největší hodnota vzorku.

5. POPIS OVLÁDANÉHO POHONU IclA D065

Cílem diplomové práce je ovládání servomotoru IclA D065 (Obr. 17) pomocí sběrnice CAN. V této kapitole bude popsán výše zmíněný pohon, který patří do skupiny servomotorů firmy SIG positec, vyznačující se proměnnou rychlostí pohonu s permanentním buzením a synchronním (DC) servomotorem. Má vestavěnou napájecí a ovládací elektroniku, síťové připojení, motor a převodovku.



Obr. 17 Integrovaný pohon IclA D065 [7]

Název IclA je zkráceným tvarem pohonu s integrovanou uzavřenou smyčkou (Integrated Closed Loop Actuator). Současná a požadovaná hodnota pozice je udržována v uzavřené kontrolní smyčce. Mezi charakteristické vlastnosti pohonu patří:

- ✓ kompaktní konstrukce,
- ✓ integrované poziční funkce,
- ✓ komunikační převodník pro provádění operací na sběrnici,
- ✓ provoz se změnou směru otáček za provozu,
- ✓ vysoký brzdící a točivý moment.

Pohon je vyráběn se čtyřmi druhy převodovek, rozměry a elektrické připojení jsou pro všechny čtyři varianty stejné. Označení pohonu je IclA D065 DC024 SXXX, kde XXX vyjadřuje typ převodovky.

5.1. Části pohonu

Pohon obsahuje následující části [7], viz Obr. 18:

Převodovka: pohon je vybaven tří nebo čtyř rychlostní převodovkou. Kovové převody jsou uloženy v samomazných bronzových pouzdrech. Třírychlostní převodovky dosahují účinnosti 77,8 %, čtyřrychlostní pak 71,6 %.

Motor: bezuhlíkový, permanentně buzený DC motor pracuje s vnitřním rozlišením 12 dílků za jednu otáčku. Odstupňování převodů je v rozmezí od 213 do 1378 dílků na jednu otáčku hnací hřídele.

Poziční senzor: s řídící elektronikou tvoří čidlo kvaziabsolutní hodnoty. Tři Hallovy senzory v pozičním zařízení snímají aktuální pozici rotoru v rozlišení 12 dílků na jednu otáčku a monitorují stav změny motoru. Poziční čítač po té převede aktuální pozici na 32 bitovou absolutní hodnotu. Po ukončení pozičního procesu, ale před vypnutím pohonu jsou do interní datové paměti uloženy absolutní hodnota pozice a stav změny motoru. Při opětovném zapnutí motoru čidlo kvaziabsolutní hodnoty určí, zda došlo k mechanickému pohybu hřídele v rozmezí 2, 4, 8 nebo 10 dílků za otáčku.

Napájecí elektronika: provádí poziční instrukce řídící elektroniky pro řízení motoru. Ochranné funkce nepřetržitě monitorují zařízení proti podpětí, přepětí a ochraňují zařízení před přetížením. Je-li motor v klidu, řídící systém vypne napěťový zesilovač, aby nedošlo k jeho přehřátí.

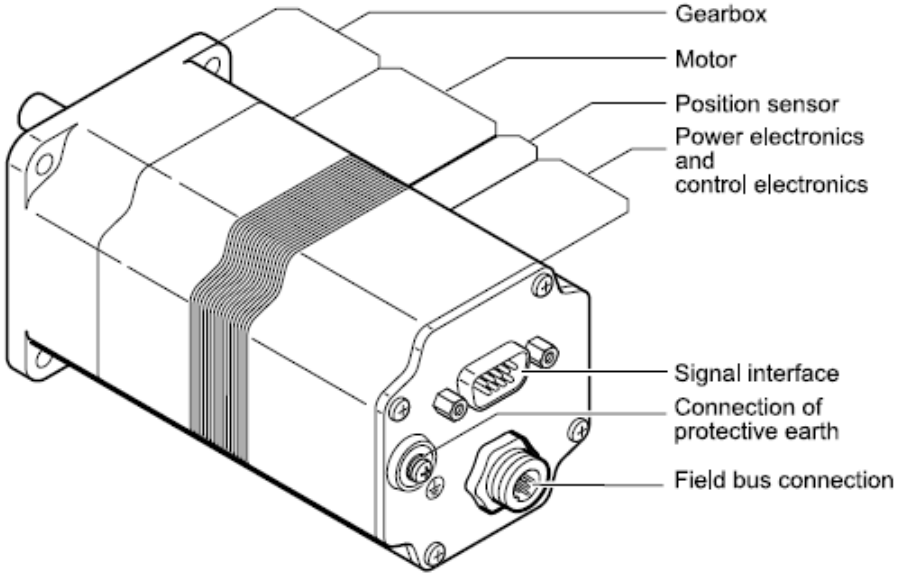
Signální převodník: poziční zařízení je napájeno přes signální převodník a zároveň nabízí připojení nouzového vypínacího signálu a ovládacích signálů pro manuální režim.

Napájení: napájení zesilovače a řídící elektroniky je 24 V ss. Může být připojeno přímo bez nutnosti dalších úprav týkajících se potlačení rušení.

Připojení pro nouzové vypnutí: reprezentuje v signálním převodníku jeden ze dvou pohotovostních kanálů, který slouží pro aktivaci vestavěné ochranné funkce pro nouzové vypnutí.

Připojení pro manuální řízení: pohon může být řízen manuálně pomocí dvou kontaktů. Signály jsou připojeny přes signální převodník.

Připojení sběrnice: reprezentuje komunikační převodník pohonu. Připojení na sběrnici umožňuje pohonu jeho integraci do sítě zařízení, které jsou ovládány např. PLC.



Obr. 18 Popis jednotlivých částí motoru [7]

5.2. Operační módy

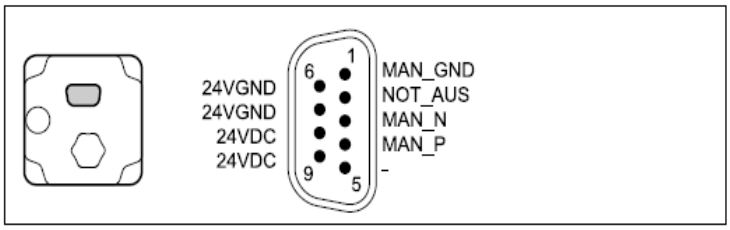
Pohon pracuje ve třech operačních módech: manuálním, pozičním a referenčním. Po připojení na napájení se pohon automaticky nastaví do manuálního módu. V manuálním módu se poziční zařízení pohybuje v dané pracovní oblasti proměnnou rychlostí. Řízení pohonu v manuálním módu může být prováděno buď připojením externích signálů, nebo po sběrnici. V pozičním módu můžeme řídit posun z bodu A do bodu B. Řízení pohybu je definováno lichoběžníkovou charakteristikou, která je dána nominální hodnotou rychlosti a náběžných hran zrychlení a zpomalení [7].

5.3. Vlastnosti okolního prostředí

Zařízení by mělo být provozováno za daných podmínek. Teplota okolního prostředí by se měla pohybovat v rozmezí 0 – 60 °C. Transportní a skladovací teplota je od – 40 do + 70 °C. Relativní vlhkost by se měla pohybovat v rozmezí 15 – 85 % a nesmí nastat kondenzace. V případě instalace zařízení v rámci nadmořské výšky do 1000 m nad střední hladinou moře nedochází k ovlivnění funkčnosti motoru. Stupeň elektrotechnické ochrany bez těsnění hřídele je IP54, s těsněním hřídele pak IP65 [7].

5.4. Rozvržení signálního převodníku

Signální převodník je opatřen devíti pinovým D Sub konektorem, který je znázorněn na Obr. 19 včetně popisu jednotlivých pinů.

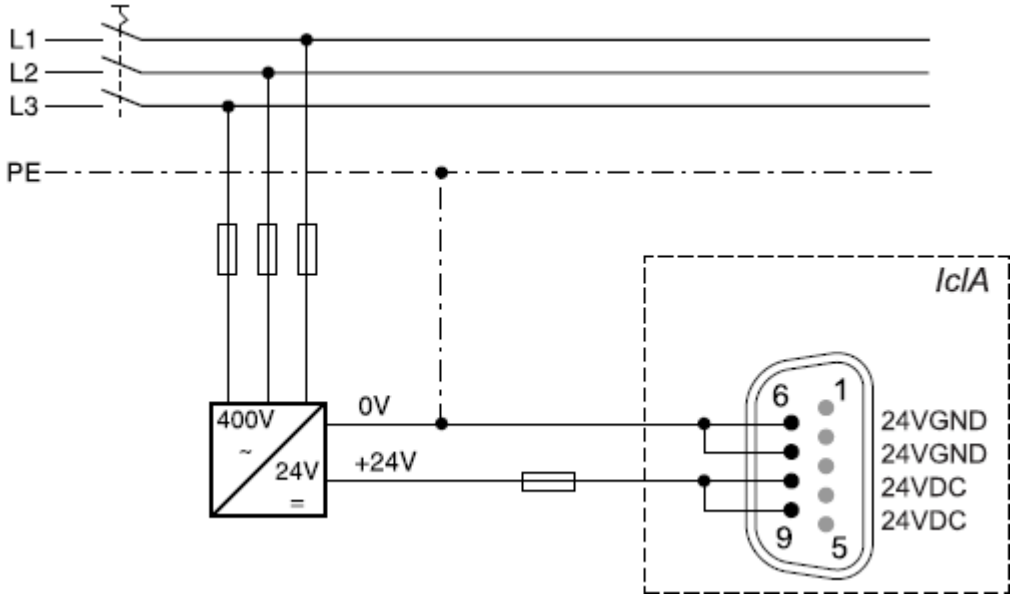


Pin	Signal	active	Meaning	I/O
1	MAN_GND	–	Earth, reference potential for control signals MAN_P and MAN_N	–
2	NOT_AUS	low	Control signal for emergency shutdown via safety switch	I
3	MAN_N	low	Control signal for negative manual movement	I
4	MAN_P	low	Control signal for positive manual movement	I
5	–	–	not assigned	–
6	24VGND	–	GND for 24 V_{DC} power supply	–
7	24VGND	–	GND for 24 V_{DC} power supply	–
8	24VDC	–	24 V_{DC} power supply	I
9	24VDC	–	24 V_{DC} power supply	I

Obr. 19 Zapojení D Sub konektoru [7]

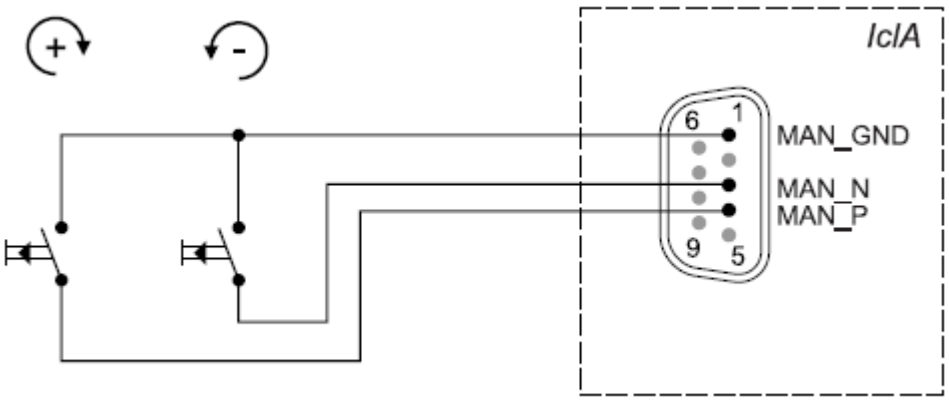
Zdrojem napětí může být transformátor s třífázovým usměrňovačem, ale není nutné použít vyhlazovací kondenzátor. Ukázka připojení napájení viz Obr. 20. Zvlnění usměrněného napětí však musí být udržováno v rozmezí 2 V, aby byl zaručen správný chod motoru. +24 V pro motor je zapojeno přes piny 8 a 9 D Sub konektoru a přes piny 6 a 7 je řešeno zapojení zemění. Minimální průřez přívodního kabelu je 1,5 mm^2.

Ochrana elektrického obvodu závisí na velikosti průřezu přívodního kabelu, ale nejmenší přípustná hodnota jištění je 6 A. Jestliže délka přívodního kabelu přesahuje 2 m, musí být přívodní vodiče spirálovitě vinuty.



Obr. 20 Schéma připojení napájení [7]

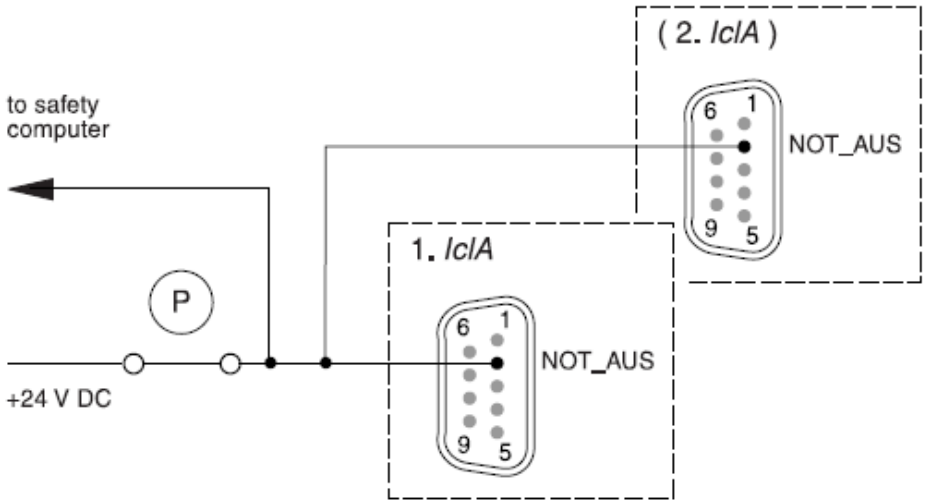
Ovládací signály manuálního módu se připojují přes piny 1, 3 a 4 D Sub konektoru, viz Obr. 21.



Obr. 21 Ovládací piny manuálního řízení [7]

Na pin 1 se připojuje zem, na pin 3 je připojen signál řízení motoru v opačném směru chodu a na pin 4 je připojen ovládací signál pro řízení chodu motoru v běžném směru. Průřez zapojených vodičů by měl být v rozmezí 0,25 – 1,25 mm^2. Maximální délka vodičů při nejmenším průřezu je 15 m.

Funkce nouzového vypnutí je aktivována přes pin nouzového vypnutí (2). V případě, že dojde k přerušení napájení pinu je pohon okamžitě zastaven. Zapojení pinu nouzového napětí viz Obr. 22.

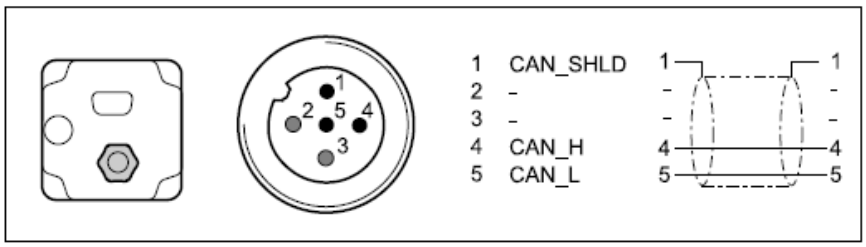


Obr. 22 Zapojení pinu nouzového napětí [7]

Specifikace přívodních vodičů je stejná jako u manuálního řízení. K vyřazení funkce nouzového vypnutí je třeba provést propojení pinů 2, 8 a 9 [7].

5.5. Připojení CAN sběrnice

Pohon je připojen na CAN sběrnici přes pěti pinový zásuvný konektor, který je znázorněn na Obr. 23 včetně popisu zapojení pinů.



Pin	Signal	Meaning	I/O
1	CAN_SHLD	Screen, PE connection	–
2	–	not assigned	–
3	–	not assigned	–
4	CAN_H	Data wire, dominant high	I/O
5	CAN_L	Data wire, dominant low	I/O

Obr. 23 Konektor pro připojení pohonu na CAN sběrnici [7]

Pohon se připojuje ke sběrnici pomocí T - rozbočovače, do kterého je připojen přívodní a výstupní kabel. Je-li pohon umístěn jako poslední v řadě, pak je nutné místo výstupního kabelu připojit zakončující 120 Ω odpor. Tímto způsobem je třeba zakončit oba konce sítě. Maximální délka kabelu v síti závisí na přenosové rychlosti použité v síti. Obecně platí, že čím je vyšší rychlost, tím je kratší délka sběrnice [7]. Závislost délky kabelu na přenosové rychlosti ukazuje Tab. 10.

Tab. 10 Závislost délky kabelu na přenosové rychlosti

přenosová rychlost [kBd]	délka kabelu [m]	přenosová rychlost [kBd]	délka kabelu [m]
1000	25	100	600
800	50	50	1000
500	100	20	2500
250	250	10	5000

6. CAN OVLÁDÁNÍ POHONU VERZE 1.0

V rámci diplomové práce byl sestrojen ovládací prvek „CAN ovládaní pohonu verze 1.0“ (dále jen ovladač) viz Obr. 24. Zařízení slouží k ovládání pohonu po CAN sběrnici.

Obr. 24 Vytvořený ovladač CAN ovládání pohonu.

Ovladač je tvořen obalem, který ve svém jádru ukrývá vybraný naprogramovaný mikroprocesor PIC18F4685 a další elektrosoučástky. Dále je ovladač vybaven displejem jako součástí komunikačního rozhraní s uživatelem, konektorem vstupního napětí (6 Vss), hlavním vypínačem, konektorem pro připojení na CAN sběrnici opatřeným 120 Ω zakončovacím terminátorem a šesti ovládacími tlačítky viz Obr. 25.

Tlačítka pro pohyb v menu DOPRAVA a DOLEVA,

Tlačítka ZVÝŠIT a SNÍŽIT pro nastavování číselných hodnot,

OK Tlačítko ENTER pro potvrzení volby výběru a zadání číselné hodnoty

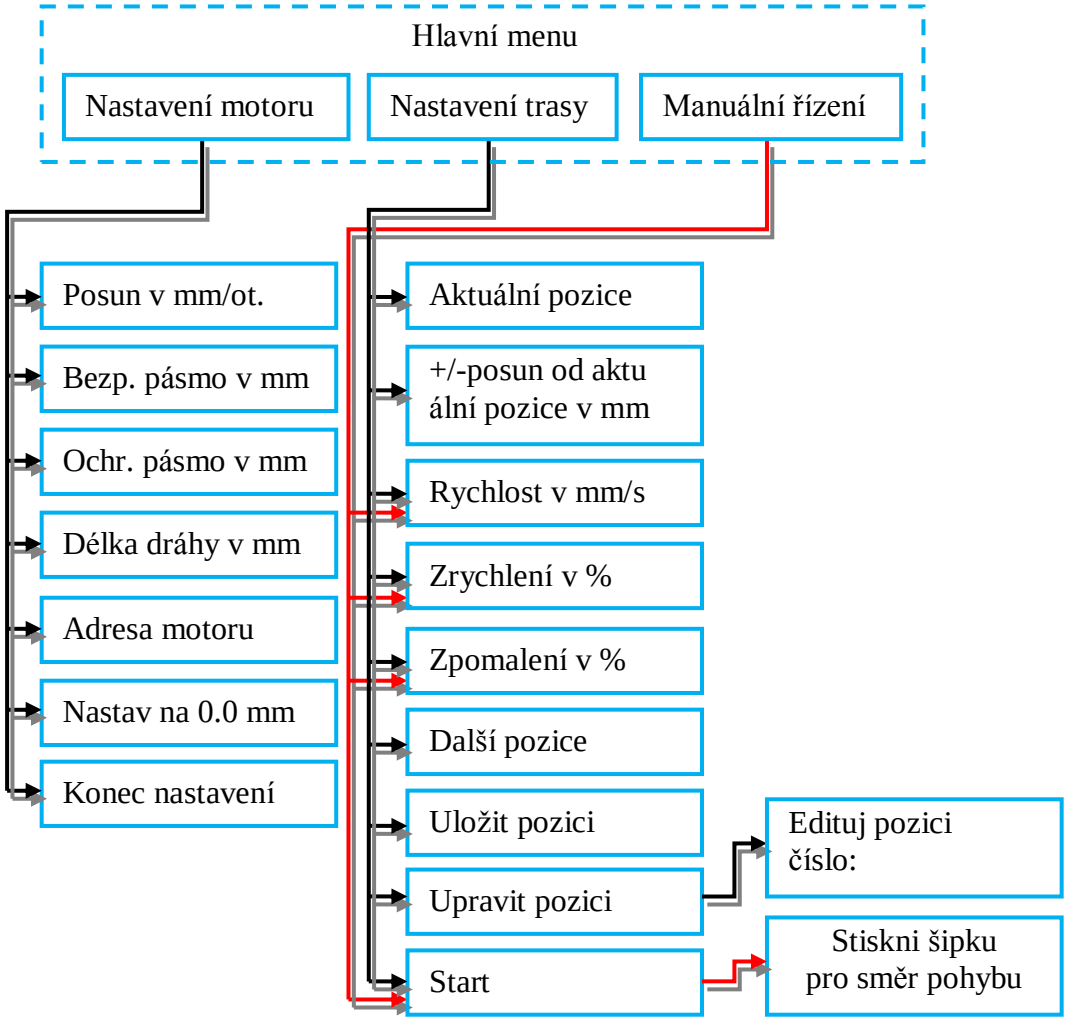
Tlačítko nouzového přerušení pohybu

Obr. 25 Ovládací tlačítka ovladače.

Po připojení do sítě a po inicializaci začne ovladač komunikovat s pohonem, aby zjistil, zda již byly nastaveny parametry právě připojeného pohonu. V případě, že se jedná o první spojení s pohonem, je uživatel vyzván k zadání adresy pohonu a poté k nastavení jeho parametrů, kterými jsou:

- ✓ posun [mm za otáčku],
- ✓ bezpečnostní pásmo [mm],
- ✓ ochranné pásmo [mm],
- ✓ délka dráhy [mm].

Po zadání požadovaných údajů, jsou odeslány pohonu a uloženy ve vnitřní flash paměti. Program přejde na úroveň hlavního menu, aby uživatel mohl začít ovládat pohon. Struktura ovládacího menu je vyobrazena v Obr. 26.



Obr. 26 Blokové schéma rozvržení ovládacího menu.

6.1. Hlavní menu ovladače

Hlavní menu obsahuje tři položky, které jsou popsány v následujících kapitolách

6.1.1. Nastavení motoru

Výběrem položky „nastavení motoru“ z hlavního menu se uživateli zpřístupní následující podnabídky.

Posun v mm/ot. – pro nastavení poměru posunu pojezdu [mm] k otáčkám výstupní hřídele motoru. Po volbě této položky se na displeji zobrazí následující text:

```
Posun:
                00.00
```

podtržená číslice označuje editační pozici. Pro změnu takto označené číslice se používají tlačítka „ZVÝŠIT“ a „SNÍŽIT“ (Obr. 25) a pro změnu pozice tlačítka „VPRAVO“ a „VLEVO“ (Obr. 25). Po zadání číselné hodnoty položky se stlačením tlačítka „ENTER“ (Obr. 25) vrací běh programu zpět do podmenu „Nastavení motoru“.

Bezp. pásmo v mm, Ochr. pásmo v mm – použitý pohon IclA D065 umožňuje nastavit bezpečnostní a ochranné pásmo, které chrání motor před fyzickým poškozením. Tato pásma se rovnají hodnotám rozdílu vzdáleností S0, D0 nebo S1, D1 a D0,W0 nebo D1, W1 jak je patrné z obrázku Obr. 27.

Obr. 27 Zobrazení délky dráhy, bezpečnostního a ochranného pásma v oblasti pohybu pojezdu

Stejně jako u nastavení posunu i zde se na displeji zobrazí zadání číselné hodnoty a ovládací tlačítka (Obr. 25) mají stejný význam.

Délka dráhy v mm – slouží k nastavení vzdálenosti bodů W0 a W1 (pracovní oblast pohonu a pojezdu). Po zvolení se na displeji zobrazí:

```
Draha:
            00000.00
```

a ovládací tlačítka (Obr. 25) mají stejný význam jako u nastavení posunu.

Adresa motoru - v případě záměny pohonu za jiný s odlišnou adresou, lze v této položce upravit adresu pohonu pro vnitřní potřeby ovladače. Displej zobrazí:

```
Adresa motoru:
                000
```

pro ovládání se používají stejná tlačítka (Obr. 25), jako u nastavení posunu se stejným významem.

Nastav na 0.0 mm – je pouze servisní funkcí, kdy se po stlačení tlačítka „ENTER“ (Obr. 25) pozice motoru vrátí do počátečního stavu (W0 = 0 mm). Na displej se vypíše:

FRCIME na 0	a poté	Aktualni pozice: +00032.27

.

Konec nastavení – je poslední položkou v této nabídce a slouží pro odeslání nastavených hodnot pohonu a k návratu do nadřízené nabídky „Hlavní menu“. Pro aktivaci slouží tlačítko „ENTER“ (Obr. 25) a po jeho použití se na displeji zobrazí:

ODESILANI DAT PRO MOTOR	a poté	Nastaveni trasy .

6.1.2. Nastavení trasy

Aktuální pozice – slouží pro zobrazení pozice v rámci nastavování trasy. Před zadáním prvního uzlu trasy je zobrazena aktuální poloha pohonu, po zadání alespoň jednoho uzlu trasy jsou k aktuální pozici pohonu připočteny všechny vzdálenosti nastavených uzlů dráhy a poté zobrazeny.

+/- posun od aktuální pozice v mm – je prvním ze čtyř nastavení jednotlivých uzlů trasy pojezdu. Hodnota této položky je vztažena k předchozí položce „Aktuální pozice“ a udává délku posunu v kladném (ve směru hodinových ručiček) nebo záporném (proti směru hodinových ručiček) směru. Kladná hodnota značí kladný směr posunu. Ovládání se provádí tlačítky (Obr. 25), jak bylo popsáno na začátku kapitoly. Pro návrat do předchozího menu slouží tlačítko „ENTER“ (Obr. 25).

Rychlost v mm/s – hodnota udává rychlost posunu [$mm \cdot s^{-1}$].

Zrychlení v %, Zpomalení v % - hodnota se nastavuje v procentech. Rozsah hodnot je 0 – 99, pro určení procent se k nastavené hodnotě musí připočíst hodnota 1.

Další pozice – stlačením tlačítka „ENTER“ (Obr. 25) se nastavené hodnoty uloží do vnitřní RAM paměti a uvolní předchozí položky pro nastavení nového uzlu trasy. Trasu je možno plánovat pomocí maximálně 20 uzlů.

Uložit pozici – tato položka způsobí pouze uložení hodnot parametrů uzlu, ale hodnoty jsou i nadále přístupné editaci.

Upravit pozici – po volbě této položky se na displeji zobrazí nabídka pro volbu pozice uzlu a po potvrzení volby tlačítkem „ENTER“ (Obr. 25) se hodnoty daného uzlu nahrají do jednotlivých položek, kde je uživatel může editovat.

Start – nabídku „Nastavení trasy“ uzavírá položka „Start“, která odešle jednotlivé uzly trasy do pohonu. Zatím co je prováděna požadovaná akce, je na displeji zobrazena aktuální poloha.

6.1.3. Manuální řízení

Zvolení této nabídky zpřístupní volby nastavení profilu pohybu manuálního řízení:

- ✓ Rychlost v mm/s,
- ✓ Zrychlení v %,
- ✓ Zpomalení v %,

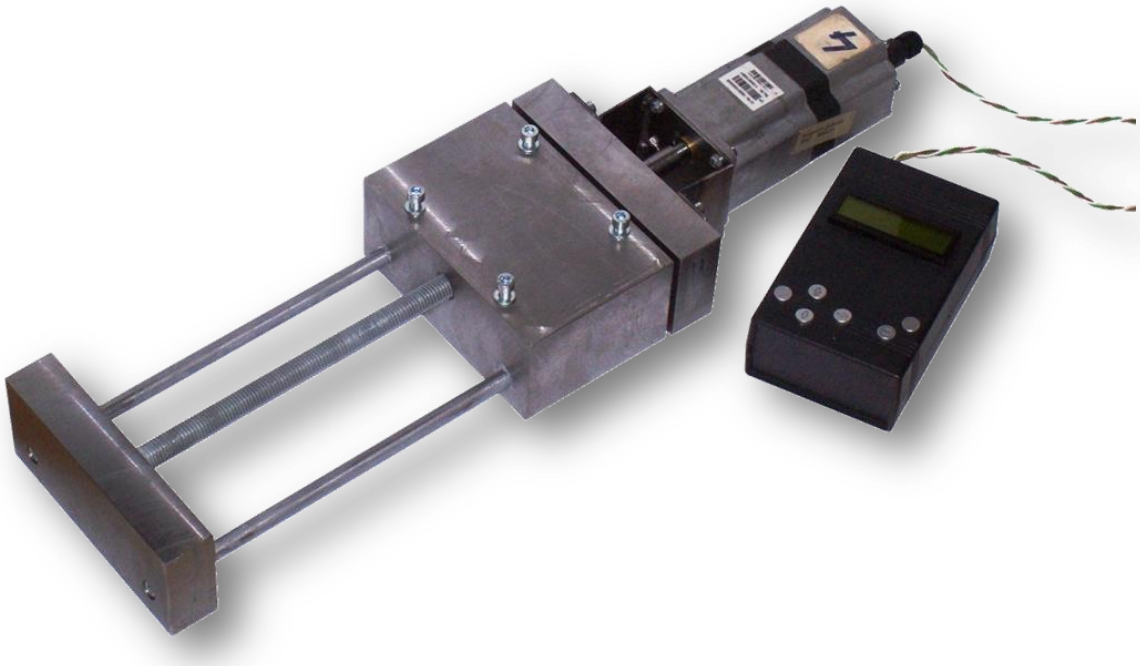
které byly popsány v předchozí kapitole.

Volbou poslední položky „Start“ jak je vyobrazeno v Obr. 26 se zpřístupní přímé ovládání chodu pohonu v kladném i záporném směru. Jednotlivé směry jsou ovládány tlačítky „DOPRAVA“ a „DOLEVA“ (Obr. 25). Pohon se otáčí daným směrem tak dlouho, dokud je tlačítko směru aktivní, nebo dokud pohon nedojede na konec dráhy, body W0 a W1 viz Obr. 27.

Pro ukončení manuálního řízení a návrat do hlavní nabídky na položku „Manuální řízení“ se používá tlačítko „ENTER“ (Obr. 25).

6.2. Demonstrační úloha

Demonstrační úloha ovládání inteligentního pohonu byla provedena na řízení pojezdu jednoosého suportu viz Obr. 28.



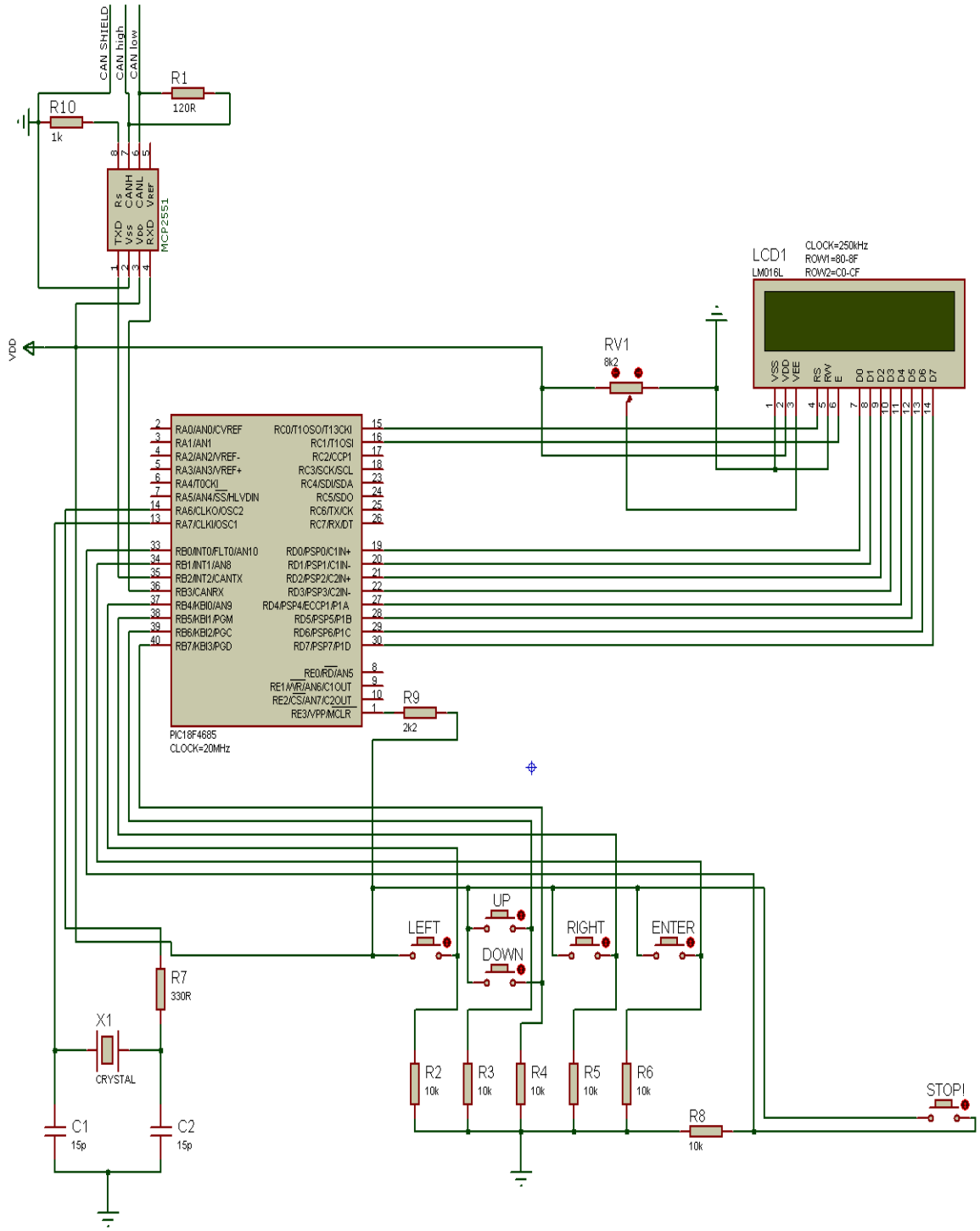
Obr. 28 Provedení demonstrační úlohy – jednoosý suport

7. ELEKTRONICKÉ ZAPOJENÍ „CAN ovládání pohonu verze 1.0“

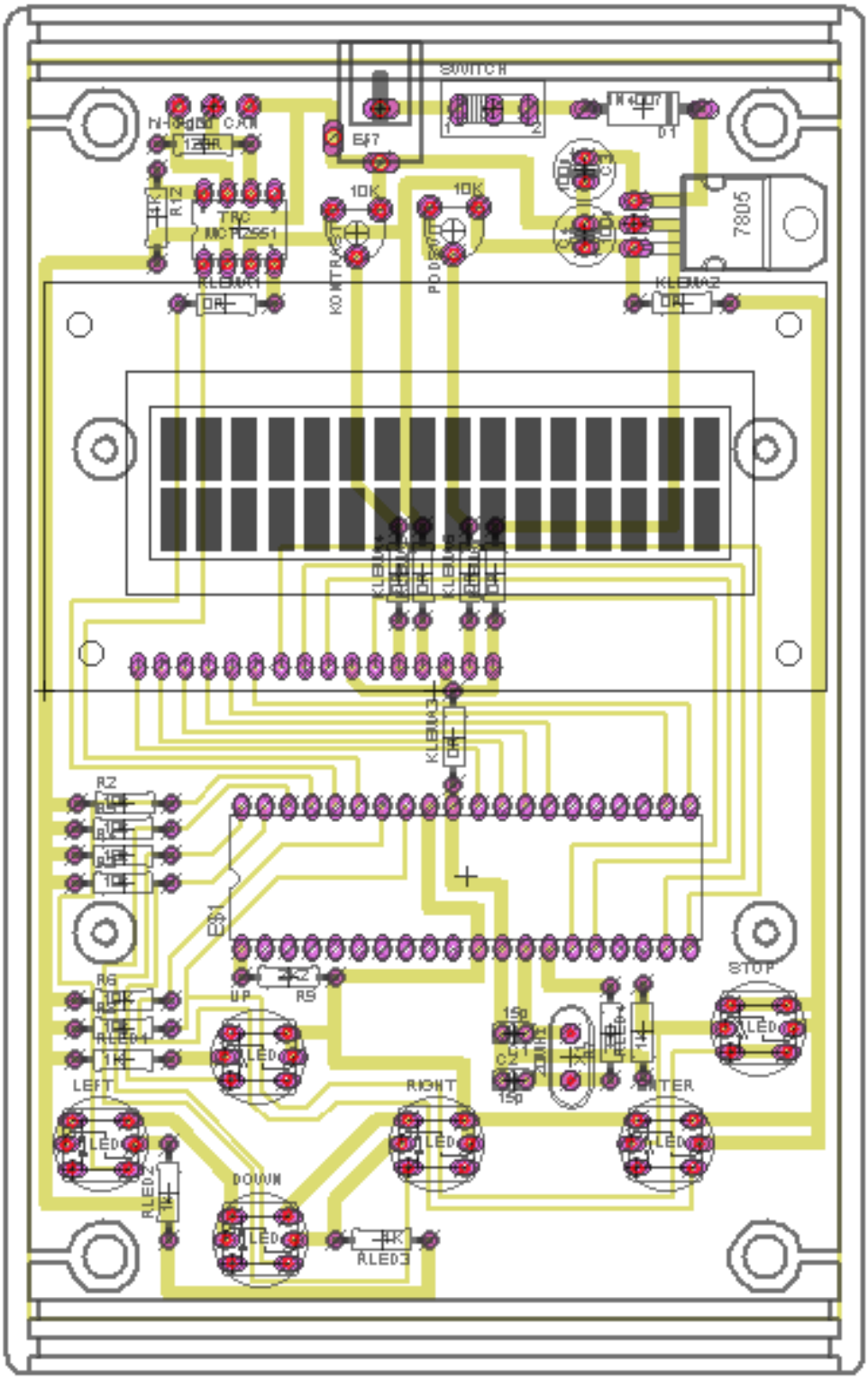
V rámci sestavení výše popsaného ovladače bylo prvotně teoreticky navrženo schéma elektronického zapojení „CAN ovládaní pohonu verze 1.0“, které je uvedeno na Obr. 29. Zapojení obsahuje zvolený mikroprocesor, který spolu s LCD panelem tvoří jádro celé sestavy. Mikroprocesor PIC18F4685 má za úkol řídit komunikaci mezi veškerými periferiemi. Další částí zapojení je již zmíněný LCD panel, který je ovládán porty C a D mikroprocesoru a spolu s šesti ovládacími tlačítky (port B) tvoří komunikační rozhraní mezi ovladačem a uživatelem. Takt procesoru udává 20 MHz krystal připojený na piny 6 a 7 portu A. Poslední zmíněnou elektrosoučástkou je transceiver MCP2551 obsluhující příjem a vysílání zpráv po CAN sběrnici. Zvládá přenos rychlostí až 1 $Mb \cdot s^{-1}$, je vhodný pro použití s 12V nebo 24V sběrnicí. Následuje podrobný výpis součástek:

- rezistor R1 – 120 Ω, 1 ks, cca 5,- Kč,
- rezistory R2 ÷ R6 a R8 – 10 kΩ, 6ks, cca 33,- Kč,
- rezistor R7 – 330 Ω, 1 ks, cca 5,- Kč,
- rezistor R9 – 2k2, 1 ks, cca 5,- Kč,
- rezistor R10 – 1 kΩ, 1 ks, cca 5,- Kč,
- trimer RV1 – 8k2, 1 ks, cca 5,50 Kč,
- LCD panel LCD1- 16 znaků, 2 řádky, 1 ks, cca 125,- Kč,
- ovládací tlačítka UP, DOWN, LEFT, RIGHT, ENTER, STOP! – 6 ks, cca 210,- Kč,
- krystal X1- 20 MHz, 1 ks, cca 10,- Kč
- kondenzátory C1, C2 – 15pF, 2 ks cca 16,- Kč,
- řídící mikroprocesor PIC18F4685firmy Microchip, 1 ks, cca 250,- Kč,
- CAN transceiver MCP2551 firmy Microchip, 1 ks, cca 40,- Kč.

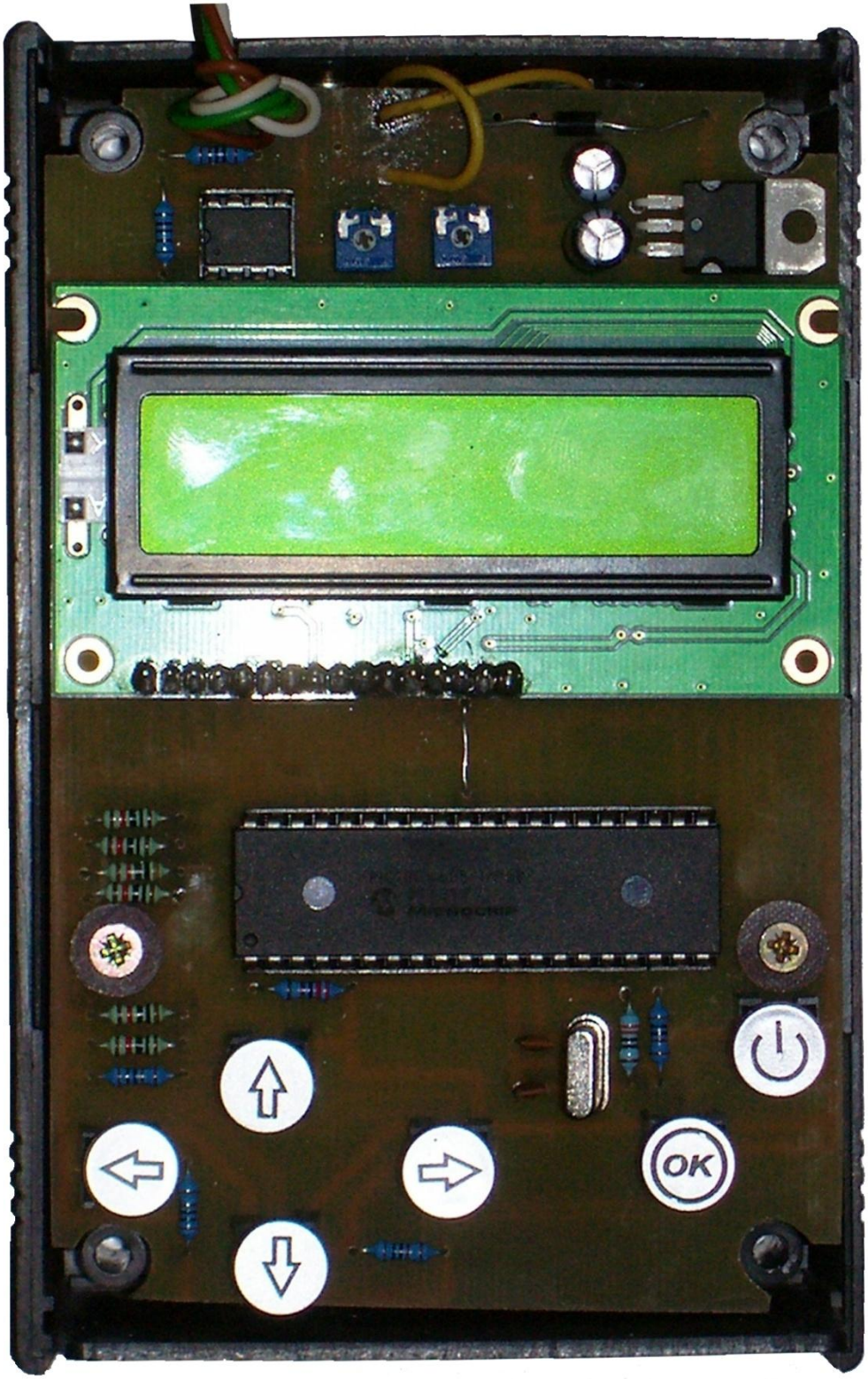
S využitím schématu elektronického zapojení byl následně připraven návrh PCB desky plošného spoje (cca 200,- Kč) pro osazení elektronických součástek, který je zobrazen na Obr. 30. Podle daného návrhu pak byl vyroben reálný fungující plošný spoj využitý k výrobě ovladače. Viz. Obr 31. Cena celého ovladače při maloobchodních cenách včetně DPH je přibližně 1100,- Kč.



Obr. 29 Schéma elektronického zapojení „CAN ovládání pohonu verze 1.0“



Obr. 30 Návrh desky plošného spoje



Obr. 31 Vnitřní konstrukce ovladače

8. ZDROJOVÝ KÓD

Souběžně se sestavováním schématu samotného HW zapojení řídícího obvodu, jak bylo ukázáno v předchozí kapitole, byl vyvíjen řídící algoritmus pro zvolený mikroprocesor PIC18F4685. Tento řídící algoritmus zajišťuje chod vnitřní struktury a periférií mikroprocesoru, interakci mezi uživatelem a řízeným pohonem.

Pro sestavení zdrojového kódu a jeho následnou kompilaci, bylo použito vývojové prostředí MPLAB® IDE verze 8.40.00.00 ve studentské verzi, které poskytuje jak komfortní prostředí pro vývoj dané aplikace včetně ladících nástrojů, tak standardní prostředky programovacího jazyka C.

Ukázka zdrojového kódu souboru Buttons.c diplomové práce na téma: „Polohová regulace motoru pomocí 8bit uP“.

/* Vložení hlavičkových souborů standardních knihoven a knihoven vytvořených v rámci DP. */

```
#include <stdlib.h>
#include <string.h>      Hlavičkové soubory standardních
#include <math.h>        knihovních funkcí.
#include <delays.h>
#include "ECAN.h"        Hlavičkový soubor knihovny pro
                         obsluhu funkcí ECAN modulu.
#include "Def's.h"       Hlavičkový soubor s obecnými
                         definicemi používaných ve většině
                         zdrojových souborů.
#include "LCD.h"         Hlavičkový soubor knihovny funkcí
                         určených pro komunikaci a ovládání
                         LCD panelu.
#include "Buttons.h"     Hlavičkový soubor knihovny funkcí
                         ovládání tlačítek, obsahuje definice
                         jmenných konstant.
#include "DataStorage.h" Hlavičkový soubor knihovny funkcí
                         obsahuje definici datové struktury a
                         deklarace funkcí pro komunikaci
                         s touto datovou strukturou.
#include "My_CAN.h"      Hlavičkový soubor knihovny funkcí
                         používaných pro ovládání a komunikaci
                         s řízeným pohonem.
#include "Strings.h"     Hlavičkový soubor obsahující definice
                         znakových řetězců uložených
                         v programové flash paměti, sloužících
                         ke komunikaci s uživatelem.
#include "NumOps.h"      Hlavičkový soubor knihovny funkcí pro
                         zaokrouhlování proměnných a převody
                         proměnných na řetězce ASCII znaků.
#include "FlashOps.h"    Hlavičkový soubor deklarující funkci
                         pro zápis hodnot do programové flash
                         paměti.
```

/* Definice jmenných konstant pro převod a zaokrouhlení čísel typu DOUBLE. */

```
#define _DenaryPlaces      5
#define _ShPRDenaryPlaces  2
#define _MADenaryPlaces    3
```

// Texty uložené v programové Flash paměti.

```
extern const near rom char *pPrintedMenu1[];
extern const near rom char *pPrintedMenu2[];
extern const near rom char *pPrintedMenu3[];
extern const near rom char *pPrintedMenu4[];
extern const near rom char *pPrintedMenu5[];
extern const near rom char *pPrintedMenu6[];
extern const near rom char *pPrintedMenu7[];
extern const near rom char *pPrintedMenu8[];
extern const near rom char PrintedMenu31b[];
extern const near rom char PrintCAN1[];
extern const near rom char PrintCAN2[];
extern const near rom char PrintClrFault1[];
extern const near rom char PrintClrFault2[];
extern const near rom char PrintNodeFault[];
extern const near rom char PrintValueFault[];
extern const near rom char PrintMotion[];
extern const near rom char PrintManMove_OrientationA[];
extern const near rom char PrintManMove_OrientationB[];
```

// Deklarace proměnných užívaných ke komunikaci s pohonem IclA.

```
extern unsigned long Prof_velocity;        // Index:6081h, Sub-
index:00h, max 900 [Inc/s]
extern unsigned long Prof_accel;                // Index:6083h,
Sub-index:00h, max 5000 [Inc/s^2]
extern unsigned long Prof_decel;                // Index:6084h,
Sub-index:00h, max 2000 [Inc/s^2]
extern double   Target_pos;                // Index:607Ah, Sub-
index:00h, max 2147483646 [Inc]
```

/*--- P A R A M E T E R Y C A N Z P R Á V Y ----------*/

```
extern unsigned long   COB_ID;
extern unsigned char   DLen;
extern ECAN_RX_MSG_FLAGS rxFlags;
```

/*--- P A R A M E T E R Y C A N Z P R Á V Y ----------*/

// Definice polí pro formátovaný textový výstup na LCD.

```
char caNum[]    ="-00000.00",
      caShift[] ="00.00",
      caPct[]   ="00",
      caMotAddr[]="000",
      NodeNum   = 0;
const char _SPACE[]   ="  ";
```

```
volatile char   cPosition,
                Num_i      = 0,
                cLevel     = _LCDMainMenu,
                cPrevLevel = 0,
                cMenu1     = 0,
                cMenu2     = 0,
                cMenu3     = 0,
                cMenu8     = 0;
extern eResult  eManual, eStart;
```

// Datová struktura nastavení jednotlivých uzlů trasy.

```
extern const stTrcSet* Data_ptr;
```

/* Datová struktura uložená ve Flash paměti, obsahuje provozní nastavení pohonu a sériové číslo pohonu. */

```
extern rom DriveData ROMDDS;
```

/* Pracovní kopie předchozí struktury uložená v RAM a ukazatele na obě struktury. */

```
extern DriveData DDS;
extern DriveData* RamDDS_ptr;
extern const rom DriveData* rom RomDDS_ptr;
```

// Proměná indikující ukončení cyklu v main. c řádek 109.

```
extern eResult _CONTINUE;
```

/* Deklarace interních funkcí:
Fce VarClr přebírá ukazatel na pole znaků a v tomto poli nastavuje jednotlivé znaky na hodnotu "0" ASCII. Fce ChkVal přebírá ukazatel na pole znaků a porovnává uživatelem nastavenou hodnotu v tomto poli obsaženou s maximální nebo minimální přípustnou hodnotou pro daný parametr. Je-li hodnota větší než maximum nebo menší než minimum, je do argumentu zapsána hodnota maxima nebo minima a funkce vrací hodnotu 1 jako příznak změny. Je-li hodnota argumentu ponechána beze změny, vrací funkce hodnotu 0. */

```
void VarClr(char* _Var);
char ChkVal(char* pArg);
```

/*--- D E F I N I C E F U N K C Í ----------*/

// Definice fce použité při inicializaci portu tlačítek.

```
void Btn_reg_set(void)
{
// Inicializace registru pro kontrolu resetovacích funkcí.
RCON = 0x00;
// Inicializace registru kontroly přerušení.
INTCON = 0x00;
// Inicializace registru kontroly přerušení.
INTCON2 = 0x00;
// Inicializace registru kontroly přerušení.
INTCON3 = 0x00;
// Zablokování využívání priorit přerušení.
RCONbits.IPEN = 0;
// Celkové vypnutí všech přerušení.
INTCONbits.GIE = 0;
```

```
// Povolení přerušení na pinu RB0.
INTCONbits.INT0IE = 1;
// Povolení přerušení na portu B, piny <7:4>.
INTCONbits.RBIE = 1;
// Přerušení na pinu RB0 je vyvoláno náběžnou hranou signálu.
INTCON2bits.INTEDG0 = 1;
// Přerušení na pinu RB1 je vyvoláno náběžnou hranou signálu.
INTCON2bits.INTEDG1 = 1;
// Vypnutí pull-up rezistorů.
INTCON2bits.RBPU = 1;
// Povolení přerušení na pinu RB1.
INTCON3bits.INT1IE = 1;
// Nastavení pinů portu B jako vstup, nastavení se netýká pinů RB2 a RB3.
_TRISBTNPORT = _TRISBTNPORT | 0xF3;}

/* Fce Button Fnc přebírá aktuální a předcházející hodnotu tlačítkového portu. Je to funkce,
která provádí jednotlivé akce přiřazené ovládacím tlačítkům v závislosti na fázi programu. */
void ButtonFnc(unsigned char ActiveBtn, unsigned char PassedBtn)
{

	// Deklarace a definice lokálních proměnných.
	unsigned char c1, c2;
	char i = 0, j = 0, DP = 0, _DPlc = 0, _POS = 0, _Num[10];
	double _dNum;
	long SPL = 0, SPDL = 0, SPSL = 0;
	char* pArr;
	unsigned double _Rate;
	eSignum _Sign = _UNSIGNED;
	static char DataNodeLevel = 0;
	eResult _END = FALSE;

	// První úroveň rozřazení podle aktivního tlačítka.
	// Aktivní tlačítko je "šipka dolů".
	if	((ActiveBtn == KEY_DOWN) && (ActiveBtn ^ PassedBtn))
	{

		// Program se nachází ve fázi nastavení číselné hodnoty.
		if (cLevel == _LCDNumber)
		{
			/* Nastavujeme-li položky "Ochranné pásmo", "Bezpečnostní pásmo",
			"Délka dráhy" z menu "Nastavení motoru", nebo položku "Posun od
			aktuální pozice" z menu "Nastavení trasy" */
			if (((cPrevLevel == _LCDMotorSetUp) &&
			((cMenu2 == _PM2_SAFETY_LIM) ||
			(cMenu2 == PM2_DRIVE_LIM) ||
			(cMenu2 == _PM2_POSITION_LIM))) ||
			((cPrevLevel == _LCDMovementSetUp) &&
			(cMenu3 == _PM3_SHIFT_POS)))

				// Je použito pole caNum.
				pArr = caNum;
```

/* Nastavujeme-li položku "Posun v mm/ot." z menu "Nastavení motoru", nebo nastavujeme položku "Rychlost" buď z menu "Nastavení trasy", nebo pro manuální řízení */

```
else if(((cPrevLevel == _LCDMotorSetUp) &&
(cMenu2 == _PM2_SHIFT_PER_REV)) ||
(((cPrevLevel == _LCDMovementSetUp) ||
(cPrevLevel == _ManMove)) &&
(cMenu3 == _PM3_VELOCITY)))

      // Je použito pole caShift.
      pArr = caShift;
```

/* Nastavujeme-li položky "Zrychlení" a "Zpomalení" z menu "Nastavení trasy", nebo pro manuální řízení */

```
else if(((cPrevLevel == _LCDMovementSetUp) ||
(cPrevLevel == _ManMove)) &&
((cMenu3 == _PM3_ACCELERATION) ||
(cMenu3 == _PM3_DECELERATION)))

      // Je použito pole caPct.
      pArr = caPct;
```

/* Nastavujeme-li položku "Adresa motoru" z menu "Nastavení motoru" */

```
else if((cPrevLevel == _LCDMotorSetUp) &&
(cMenu2 == _PM2_MOTOR_ADR))
      // Je použito pole caMotAddr.
      pArr = caMotAddr;
```

/* Hodnota aktuálně měněné číslice je uložena do pomocné proměnné. */

```
c1 = pArr[Num_i];
```

/* Nachází-li se kurzor na pozici znaménka je provedena jeho změna. */

```
if ((cPrevLevel == _LCDMovementSetUp) &&
(cMenu3 == _PM3_SHIFT_POS) &&
(Num_i == _SignPos))
{
      if((unsigned)pArr[Num_i] == '+')
            pArr[Num_i] = '-';
      else
            pArr[Num_i] = '+';
}
```

/* Je-li nastavována adresa motoru, roluje hodnota v rozmezí 127 až 1. */

```
else if((cPrevLevel == _LCDMotorSetUp) &&
(cMenu2 == _PM2_MOTOR_ADR))
{
      SPL = DDS.NODE_ADDRESS;
      SPL--;
      SPL +=127;
      SPL--;
```

```
        SPL %= 127;
        SPL++;
        DDS.NODE_ADDRESS = SPL;
        VarClr(pArr);
        i = DDS.NODE_ADDRESS;
        j = 0;
        while(i>0)
        {
                i /= 10;
                j++;
        }
        j = 3-j;
        btoa(DDS.NODE_ADDRESS,pArr+j);
}
/* Ve všech ostatních případech je aktuální hodnota číslice
dekrementována o 1. Hodnota číslice je vybírána ze smyčky číslic
0 až 9. */
else
{
        c2 = '0';
        if (c1 == '0')
                c1+=10;
        pArr[Num_i] = ((c1 - c2)-1)+c2;
}
i = cPosition;
// Vypnutí kurzoru na LCD.
SetCursor(OFF);
/* Ověření aktuální hodnoty nastavené na LCD je-li v rozsahu
povolených hodnot. */
if(ChkVal(pArr))
{
        /* V případě změny hodnoty na max. nebo min. hodnotu, je do
        pomocné proměnné uložena počáteční pozice na LCD kde bude
        tato hodnota vypsána. */
        if((cPrevLevel == _LCDMovementSetUp) &&
        (cMenu3 == _PM3_SHIFT_POS))
                j = _LCDStartPosSNum;
        else if((cPrevLevel == _LCDMotorSetUp) &&
        (cMenu2 == _PM2_MOTOR_ADR))
                j = _LCDStartAddrPos;
        else if(((cPrevLevel == _LCDMovementSetUp)
        || (cPrevLevel == _ManMove)) &&
        (cMenu3 == _PM3_VELOCITY))
                j = _LCDStartPosVel;
        else
                j = _LCDStartPosUNum;
        // Vytisknutí hodnoty na LCD.
        PrintTextRAMLCD(pArr, j);
}
else
{
```

```
                if((cPrevLevel == _LCDMotorSetUp) &&
                (cMenu2 == _PM2_MOTOR_ADR))
                        // Tisk hodnoty adresy motoru.
                        PrintTextRAMLCD(pArr,_LCDStartAddrPos
                        );
                else
                        /* Hodnota nebyla změněna, na LCD je přepsána pouze
                        daná číslice. */
                        SetNumLCD(pArr[Num_i], cPosition);
        }

        // Nastavení pozice kurzoru LCD pod změněnou číslici.
        SetCursorPos(i);

        // Zapnutí kurzoru na LCD.
        SetCursor(ON);
}

/* Uživatel se nachází v menu pro editaci uzlů trasy a vybírá číslo uzlu, který
bude editován. */
else if(cLevel == _EditPosNum)
{
        SetCursor(OFF);

        /* Smyčka pro výběr čísla uzlu. Čísla rolují od poslední použité pozice
        po nulu. */
        if(!DataNodeLevel)
                DataNodeLevel =
                GetLastUsedPosition(Data_ptr);
        else
                DataNodeLevel--;

        // Pomocná proměnná i obsahuje zobrazovanou hodnotu.
        i = DataNodeLevel + 1;

        // Převod hodnoty na ASCII znak.
        btoa(i, _Num);

        // Určení počáteční pozice tisku proměnné na LCD.
        _POS = _LCDEndPosition - strlen(_Num);

        // Tisk hodnoty na LCD.
        PrintTextRAMLCD(_SPACE,_LCDStartEditPos);
        PrintTextRAMLCD(_Num,_POS);

        // Nastavení pozice kurzoru a jeho zobrazení na LCD.
        SetCursorPos(_LCDEndPosition - 1);
        SetCursor(ON);
}
}
```

```
// Aktivní tlačítko je "šipka nahoru".
else if    ((ActiveBtn == KEY_UP) &&
(ActiveBtn ^ PassedBtn))
{

        //Program se nachází ve fázi nastavení číselné hodnoty.
        if(cLevel == _LCDNumber)
        {

                /* Nastavujeme-li položky "Ochranné pásmo", "Bezpečnostní pásmo",
                "Délka dráhy" z menu "Nastavení motoru", nebo položku "Posun od
                aktuální pozice" z menu "Nastavení trasy" */
                if (((cPrevLevel == _LCDMotorSetUp) &&
                ((cMenu2 == _PM2_SAFETY_LIM) ||
                (cMenu2 == _PM2_DRIVE_LIM) ||
                (cMenu2 == _PM2_POSITION_LIM))) ||
                ((cPrevLevel == _LCDMovementSetUp) &&
                (cMenu3 == _PM3_SHIFT_POS)))

                        // je použito pole caNum.
                        pArr = caNum;

                /* Nastavujeme-li položku "Posun v mm/ot." z menu "Nastavení
                motoru", nebo nastavujeme položku "Rychlost" buď z menu
                "Nastavení trasy", nebo pro manuální řízení */
                else if(((cPrevLevel == _LCDMotorSetUp) &&
                (cMenu2 == _PM2_SHIFT_PER_REV)) ||
                (((cPrevLevel == _LCDMovementSetUp) ||
                (cPrevLevel == _ManMove)) &&
                (cMenu3 == _PM3_VELOCITY)))

                        // Je použito pole caShift.
                        pArr = caShift;

                /* Nastavujeme-li položky "Zrychlení" a "Zpomalení" z menu
                "Nastavení trasy", nebo pro manuální řízení */
                else if(((cPrevLevel == _LCDMovementSetUp) ||
                (cPrevLevel == _ManMove)) &&
                ((cMenu3 == _PM3_ACCELERATION) ||
                (cMenu3 == _PM3_DECELERATION)))

                        // Je použito pole caPct.
                        pArr = caPct;

                /* Nastavujeme-li položku "Adresa motoru" z menu "Nastavení
                motoru" */
                else if((cPrevLevel == _LCDMotorSetUp) &&
                (cMenu2 == _PM2_MOTOR_ADR))

                        // Je použito pole caMotAddr.
                        pArr = caMotAddr;
```

/* Hodnota aktuálně měněné číslice je uložena do pomocné proměnné. */

```
c1 = pArr[Num_i];
```

/* Nachází-li se kurzor na pozici znaménka je provedena jeho změna mezi hodnotami + a -. */

```
if ((cPrevLevel == _LCDMovementSetUp) &&
(cMenu3 == _PM3_SHIFT_POS) &&
(Num_i == _SignPos))
{
      c2 = '+';
      c1 -= c2;
      c1 += 0x02;
      c1 %= 0x04;
      pArr[Num_i] = c1 + c2;
}
```

/* Je-li nastavována adresa motoru, roluje hodnota v rozmezí 1 až 127 a poté je vytištěna na LCD. */

```
else if((cPrevLevel == _LCDMotorSetUp) &&
(cMenu2 == _PM2_MOTOR_ADR))
{
      DDS.NODE_ADDRESS %= 127;
      DDS.NODE_ADDRESS++;
      VarClr(pArr);
      i = DDS.NODE_ADDRESS;
      j = 0;
      while(i>0)
      {
            i /= 10;
            j++;
      }
      j = 3-j;
      btoa(DDS.NODE_ADDRESS,pArr+j);
}
```

/* Ve všech ostatních případech je aktuální hodnota číslice inkrementována o 1. Hodnota číslice je vybírána ze smyčky číslic 0 až 9. */

```
else
{
      c2 = '0';
      c1 -= c2;
      c1++;
      c1 = c1 % 10;
      pArr[Num_i] = c1 + c2;
}
i = cPosition;
```

// Vypnutí kurzoru na LCD.

```
SetCursor(OFF);
```

```
        /* Ověření aktuální hodnoty nastavené na LCD je-li v rozsahu
        povolených hodnot. */
        if(ChkVal(pArr))
        {

                /* V případě změny hodnoty na max. nebo min. hodnotu, je do
                pomocné proměnné uložena počáteční pozice na LCD kde bude
                tato hodnota vypsána. */
                if((cPrevLevel == _LCDMovementSetUp) &&
                (cMenu3 == _PM3_SHIFT_POS))
                        j = _LCDStartPosSNum;
                else if ((cPrevLevel == _LCDMotorSetUp) &&
                (cMenu2 == _PM2_MOTOR_ADR))
                        j = _LCDStartAddrPos;
                else if(((cPrevLevel == _LCDMovementSetUp)
                || (cPrevLevel == _ManMove)) &&
                (cMenu3 == _PM3_VELOCITY))
                        j = _LCDStartPosVel;
                else
                        j = _LCDStartPosUNum;

                // Vytisknutí hodnoty na LCD.
                PrintTextRAMLCD(pArr, j);
        }
        else
        {
                if((cPrevLevel == _LCDMotorSetUp) &&
                (cMenu2 == _PM2_MOTOR_ADR))
                        // Tisk hodnoty adresy motoru.
                        PrintTextRAMLCD(pArr,
                        _LCDStartAddrPos);
                else
                        /* Hodnota nebyla změněna, na LCD je přepsána pouze
                        daná číslice. */
                        SetNumLCD(pArr[Num_i], cPosition);
        }

        // Nastavení pozice kurzoru LCD pod změněnou číslici.
        SetCursorPos(i);

        // Zapnutí kurzoru na LCD.
        SetCursor(ON);
}
/* Uživatel se nachází v menu pro editaci uzlů trasy a vybírá číslo uzlu, který
bude editován. */
else if(cLevel == _EditPosNum)
{
        // Kurzor LCD vypnut.
        SetCursor(OFF);

        // Číslo uzlu zvětšeno o 1
        DataNodeLevel++;
```

```
            /* Smyčka pro výběr čísla uzlu. Čísla rolují od nuly po poslední
            použitou pozici. */
            if((!(Data_ptr+DataNodeLevel)->Used) ||
            (DataNodeLevel == _LASTNODE+1))
                  DataNodeLevel = 0;

            // Pomocná proměnná i obsahuje zobrazovanou hodnotu.
            i = DataNodeLevel + 1;

            // Převod hodnoty na ASCII znak.
            btoa(i, _Num);

            // Určení počáteční pozice tisku proměnné na LCD.
            _POS = _LCDEndPosition - strlen(_Num);

            // Tisk hodnoty na LCD.
            PrintTextRAMLCD(_SPACE,_LCDStartEditPos);
            PrintTextRAMLCD(_Num,_POS);

            // Nastavení pozice kurzoru a jeho zobrazení na LCD.
            SetCursorPos(_LCDEndPosition - 1);
            SetCursor(ON);
      }
}

// Aktivní tlačítko je "šipka vpravo".
else if ((ActiveBtn == KEY_RIGHT) &&
(ActiveBtn ^ PassedBtn))
{
      /* Je-li zapnuto manuální řízení, je cílový uzel dráhy nastaven na 0,0 mm v
      absolutních souřadnicích (počátek) a tento uzel je předán motoru. Poté je motor
      uveden do chodu a spustí se monitorování držení tlačítka. Pohon jede tak
      dlouho, dokud je tlačítko aktivní, nebo dokud nedosáhne nastaveného cíle trasy.
      V případě deaktivace tlačítka je motor zastaven. */
      if(eManual)
      {
            Data_ptr->Target_Inc = _ORIGIN;
            Data_ptr->Used = 1;
            ActivateMotor(_ABSOLUT);
      }
      // Rolování položek hlavního menu vpravo.
      else if    (cLevel == _LCDMainMenu)
      {
            cMenu1++;
            cMenu1 %= _Menu1Count;
            ClearLCD();
            PrintTextPGMLCD( pPrintedMenu1[cMenu1],
            LCD_ROW_1);
```

```
                /* Získání aktuální pozice pohonu a její zobrazení na LCD. */
                if(cMenu1 == _PM1_ACT_POS)
                {
                        #ifndef _CAN_DISABLED_
                                SPL = ReadActPos();
                                DDS.ACTUAL_POSITION_MM = ((double)SPL
                                / DDS.GEAR_INC_RATIO)*
                                DDS.SHIFT_PER_REVOLUTION;
                        #endif
                        fltoa(_Num, DDS.ACTUAL_POSITION_MM,
                        _DenaryPlaces, _DecimalPlaces, _UNSIGNED);
                        PrintTextRAMLCD(_Num, LCD_ROW_2+ 4);
                }
        }
        // Rolování položek menu "Nastavení motoru" vpravo.
        else if    (cLevel == _LCDMotorSetUp)
        {
                cMenu2++;
                cMenu2 = cMenu2 % _Menu2Count;
                ClearLCD();
                PrintTextPGMLCD(
pPrintedMenu2[cMenu2],LCD_ROW_1);
        }
        // Rolování položek menu "Nastavení trasy" vpravo.
        else if    (cLevel == _LCDMovementSetUp)
        {
                cMenu3++;
                if((NodeNum == _LASTNODE) &&
                (cMenu3 == _PM3_NEXT_POSITION))
                        cMenu3++;
                cMenu3 = cMenu3 % _Menu3Count;
                ClearLCD();
                PrintTextPGMLCD( pPrintedMenu3[cMenu3],
                LCD_ROW_1);
                if (cMenu3 == _PM3_SHIFT_POS)
                        PrintTextPGMLCD(PrintedMenu31b,LCD_ROW_2);
        }
        /* Nastavení editační pozice a pozice kurzoru LCD při nastavování číselných
        hodnot položek. Obsahuje-li číslo desetinou tečku, je její pozice při nastavování
        přeskočena. Kurzor a editační pozice rotuje od prvního číselného znaku nebo
        znaménka po poslední číselný znak. */
        else if (cLevel == _LCDNumber)
        {
                if(cPrevLevel == _LCDMotorSetUp)
                {
                        if(cMenu2 == _PM2_SHIFT_PER_REV)
                        {
                                i = _LCDShRtoLen;
                                j = _LCDStartPosShRto;
                                DP = _DecPointPosShRto;
                        }
```

```
        else if(cMenu2 == _PM2_MOTOR_ADR)
        {
                _END = TRUE;
        }
        else
        {
                i = _LCDUNumLen;
                j = _LCDStartPosUNum;
                DP =_DecimalPointUNum;
        }
}
else if ((cPrevLevel == _LCDMovementSetUp) ||
(cPrevLevel == _ManMove))
{
        if(cMenu3 == _PM3_SHIFT_POS)
        {
                i = _LCDSNumLen;
                j = _LCDStartPosSNum;
                DP =_DecimalPointSNum;
        }
        else if(cMenu3 ==_PM3_VELOCITY)
        {
                i = _LCDShRtoLen;
                j = _LCDStartPosVel;
                DP =_DecPointPosShRto;
        }
        else
        {
                i = _LCDPctLen;
                j = _LCDStartPosPct;
                DP =0x7F;
        }
}
/* Podmínka není vykonána pouze, je-li nastavována hodnota adresy
motoru. */
if (_END == FALSE)
{
        SetCursor(OFF);
        do
        {

                /* Posunutí kurzoru a editační pozice o 1 vpravo. */
                Num_i++;
                Num_i = Num_i % i;
                cPosition = j + Num_i;
                ShiftCursor(RIGHT);
        }
        while (Num_i == DP); // Přeskočení desetinné tečky.
        if(!Num_i)
                SetCursorPos(cPosition);
        SetCursor(ON);
}
```

```
        }
        // Rolování položek menu "Manuální řízení" vpravo.
        else if (cLevel == _ManMove)
        {
              if (cMenu3 == _PM3_START)
                    cMenu3 = _PM3_VELOCITY;
              else if (cMenu3 == _PM3_DECELERATION)
                    cMenu3 = _PM3_START;
              else
                    cMenu3++;
              ClearLCD();
              PrintTextPGMLCD( pPrintedMenu3[cMenu3],
              LCD_ROW_1);
        }

// Vynulování proměnné.
_END = FALSE;
}

// Aktivní tlačítko je "šipka vlevo"
else if    ((ActiveBtn == KEY_LEFT) &&
(ActiveBtn ^ PassedBtn))
{

        /* Je-li zapnuto manuální řízení, je cílový uzel dráhy nastaven na délku dráhy v
        absolutních souřadnicích (konec dráhy) a tento uzel je předán motoru. Poté je
        motor uveden do chodu a spustí se monitorování držení tlačítka. Pohon jede tak
        dlouho, dokud je tlačítko aktivní, nebo dokud nedosáhne nastaveného cíle trasy.
        V případě deaktivace tlačítka, je motor zastaven. */
        if(eManual)
        {

              Data_ptr->Target_Inc = DDS.PATH_LENGTH_MM /
              DDS.SHIFT_PER_REVOLUTION * DDS.GEAR_INC_RATIO;
              Data_ptr->Used = 1;
              ActivateMotor(_ABSOLUT);

        }
        // Rolování položek hlavního menu vlevo.
        else if(cLevel == _LCDMainMenu)
        {
              if (cMenu1 == 0)
                    cMenu1 = _Menu1Count;
              cMenu1--;
              ClearLCD();
              PrintTextPGMLCD( pPrintedMenu1[cMenu1],
              LCD_ROW_1);

              /* Získání aktuální pozice pohonu a její zobrazení na LCD. */
              if(cMenu1 == _PM1_ACT_POS)
              {
                    #ifndef _CAN_DISABLED_
```

```
                SPL = ReadActPos();
                DDS.ACTUAL_POSITION_MM = ((double)SPL
                / DDS.GEAR_INC_RATIO)*
                DDS.SHIFT_PER_REVOLUTION;
            #endif
            fltoa(_Num, DDS.ACTUAL_POSITION_MM,
            _DenaryPlaces, _DecimalPlaces, _UNSIGNED);
            PrintTextRAMLCD(_Num, LCD_ROW_2+ 4);
        }
    }
```

// Rolování položek menu "Nastavení motoru" vlevo.

```
    else if    (cLevel == _LCDMotorSetUp)
    {
        if (cMenu2 == 0)
            cMenu2 = _Menu2Count;
        cMenu2--;
        ClearLCD();
        PrintTextPGMLCD( pPrintedMenu2[cMenu2],
        LCD_ROW_1);
    }
```

// Rolování položek menu "Nastavení trasy" vlevo.

```
    else if    (cLevel == _LCDMovementSetUp)
    {
        if (cMenu3 == 0)
            cMenu3 = _Menu3Count;
        cMenu3--;
        if((NodeNum == _LASTNODE) &&
        (cMenu3 == _PM3_NEXT_POSITION))
            cMenu3--;
        ClearLCD();
        PrintTextPGMLCD( pPrintedMenu3[cMenu3],
        LCD_ROW_1);
        if (cMenu3 == _PM3_SHIFT_POS)
            PrintTextPGMLCD(PrintedMenu31b,
            LCD_ROW_2);
    }
```

/* Nastavení editační pozice a pozice kurzoru LCD při nastavování číselných hodnot položek. Obsahuje-li číslo desetinou tečku, je její pozice při nastavování přeskočena. Kurzor a editační pozice rotuje od posledního po první číselný znak nebo znaménko. */

```
    else if (cLevel == _LCDNumber)
    {
        if(cPrevLevel == _LCDMotorSetUp)
            if(cMenu2 == _PM2_SHIFT_PER_REV)
            {
                i = _LCDShRtoLen;
                j = _LCDStartPosShRto;
                DP = _DecPointPosShRto;
            }
            else if(cMenu2 == _PM2_MOTOR_ADR)
            {
                _END = TRUE;
```

```
            }
            else
            {
                  i = _LCDUNumLen;
                  j = _LCDStartPosUNum;
                  DP =_DecimalPointUNum;
            }
      else if ((cPrevLevel == _LCDMovementSetUp) ||
      (cPrevLevel == _ManMove))
            if(cMenu3 == _PM3_SHIFT_POS)
            {
                  i = _LCDSNumLen;
                  j = _LCDStartPosSNum;
                  DP =_DecimalPointSNum;
            }
            else if(cMenu3 ==_PM3_VELOCITY)
            {
                  i = _LCDShRtoLen;
                  j = _LCDStartPosVel;
                  DP =_DecPointPosShRto;
            }
            else
            {
                  i = _LCDPctLen;
                  j = _LCDStartPosPct;
                  DP =0x7F;
            }
      /* Podmínka není vykonána pouze, je-li nastavována hodnota adresy
      motoru. */
      if (_END == FALSE)
      {
            if (!Num_i)
            {
                  SetCursor(OFF);
                  Num_i = i;
                  SetCursorPos(j+Num_i);
            }
            do
            {
                  /* Posunutí kurzoru a editační pozice o 1 vlevo. */
                  Num_i--;
                  ShiftCursor(LEFT);
            }
            while(Num_i == DP);    // Přeskočení desetinné tečky.
            if(Num_i == (i-1))
                  SetCursor(ON);
            cPosition = j + Num_i;
      }
}
```

```
        // Rolování položek menu "Manuální řízení" vlevo.
        else if (cLevel == _ManMove)
        {
              if (cMenu3 == _PM3_VELOCITY)
                    cMenu3 = _PM3_START;
              else if (cMenu3 == _PM3_START)
                    cMenu3 = _PM3_DECELERATION;
              else
                    cMenu3--;
              ClearLCD();
              PrintTextPGMLCD( pPrintedMenu3[cMenu3],
              LCD_ROW_1);
              if (cMenu3 == _PM3_SHIFT_POS)
                    PrintTextPGMLCD(PrintedMenu31b,
                    LCD_ROW_2);
        }

// Vynulování proměnné.
_END = FALSE;
}

// Aktivní tlačítko je "Enter (OK)".
else if    ((ActiveBtn == KEY_ENTER) &&
(ActiveBtn ^ PassedBtn))
{
        /* Výstup z manuálního řízení a vstup do hlavního menu na položku "Manuální
        řízení", následně je zinicializována datová struktura nastavení uzlů trasy a
        smazáno LCD. */
        if(eManual)
        {
              eManual          = FALSE;
              cLevel           = _LCDMainMenu;
              cPrevLevel       = _LCDMainMenu;
              cMenu1           = _PM1_MANUAL_CONTROL;
              DataInit(Data_ptr);
              ClearLCD();
              PrintTextPGMLCD( pPrintedMenu1[cMenu1],
              LCD_ROW_1);
        }
        // Vstup do jednotlivých položek hlavního menu.
        else if (cLevel == _LCDMainMenu)
        {
              /* Uložení hodnoty stávající úrovně menu do položky předchozí úroveň
              menu. */
              cPrevLevel = _LCDMainMenu;

              // Vstup do nastavení motoru.
              if(cMenu1 == _PM1_MOTOR_SETUP)
              {
                    cLevel = _LCDMotorSetUp;
                    ClearLCD();
                    cMenu2 = 0;
```

```
            PrintTextPGMLCD(pPrintedMenu2[cMenu2],
            LCD_ROW_1);
        }
```

/* Vstup do nastavení trasy. Před vstupem je zjištěno pořadové číslo naposledy použitého uzlu trasy a je uloženo do proměnné NodeNum. Dále je zjištěna aktuální pozice motoru a uložena do datové struktury DDS a položky ACTUAL_POSITION_MM. */

```
        else if(cMenu1 == _PM1_TRACE_SETUP)
        {
            cLevel = _LCDMovementSetUp;
            ClearLCD();
            cMenu3 = _PM3_SHIFT_POS;
            PrintTextPGMLCD(pPrintedMenu3[cMenu3],
            LCD_ROW_1);
            PrintTextPGMLCD(PrintedMenu31b,LCD_ROW_2);
            NodeNum = GetLastUsedPosition(Data_ptr);
            #ifndef _CAN_DISABLED_
                SPL = ReadActPos();
                DDS.ACTUAL_POSITION_MM = ((double)SPL
                / DDS.GEAR_INC_RATIO)*
                DDS.SHIFT_PER_REVOLUTION;
            #endif
        }
```

/* Vstup do menu manuálního řízení. Při vstupu je zinicializována datová struktura nastavení uzlů trasy. Číslo aktuálního uzlu je uloženo do proměnné NodeNum. */

```
        else if(cMenu1 == _PM1_MANUAL_CONTROL)
        {
            cPrevLevel = _LCDMainMenu;
            cLevel = _ManMove;
            cMenu3 = _PM3_VELOCITY;
            ClearLCD();
            PrintTextPGMLCD(pPrintedMenu3[cMenu3],
            LCD_ROW_1);
            DataInit(Data_ptr);
            NodeNum = 0;
        }
    }
```

// Vstup do jednotlivých položek menu pro nastavení motoru.

```
    else if (cLevel == _LCDMotorSetUp)
    {
        if ((cMenu2 == _PM2_SHIFT_PER_REV) ||
        (cMenu2 == _PM2_SAFETY_LIM) ||
        (cMenu2 == _PM2_DRIVE_LIM) ||
        (cMenu2 == _PM2_POSITION_LIM) ||
        (cMenu2 == _PM2_MOTOR_ADR))
        {
```

/* Při pokusu o nastavení jedné z položek "Bezpečnostní pásmo", "Ochranné pásmo", nebo "Délka dráhy" je v případě nenastavené položky "Posun v mm/ot" uživatel vyzván k jejímu nastavení. */

```
if(((cMenu2 == _PM2_SAFETY_LIM) ||
(cMenu2 == _PM2_DRIVE_LIM) ||
(cMenu2 == _PM2_POSITION_LIM)) &&
(!DDS.SHIFT_PER_REVOLUTION))
{
    cMenu2 = _PM2_SHIFT_PER_REV;
    PrintTextPGMLCD(PrintValueFault,
    LCD_ROW_1);
    PrintTextPGMLCD(pPrintedMenu2[cMenu2],
    LCD_ROW_2);
    for (i=0; i<2; i++)     // pauza 1 sec
        Delay10KTCYx(250);
    ClearLCD();
    PrintTextPGMLCD(pPrintedMenu2[cMenu2],
    LCD_ROW_1);
}
else
{
    /* Vstupujeme na úroveň nastavení hodnot jednotlivých
    položek. */
    cPrevLevel = _LCDMotorSetUp;
    cLevel = _LCDNumber;
    ClearLCD();
    PrintTextPGMLCD(pPrintedMenu5[cMenu2],
    LCD_ROW_1);
    /* Nastavení parametrů čísla a jeho zobrazení pro
    jednotlivé položky. */
    if (cMenu2 == _PM2_SHIFT_PER_REV)
    {
          i = _PM7_ShRto;
          _POS = _LCDStartPosShRto;
          _Sign = _UNSIGNED;
          _dNum =
          DDS.SHIFT_PER_REVOLUTION;
          _DPlc = _ShPRDenaryPlaces;
          j = _DecimalPlaces;
          pArr = caShift;
     }
     else if(cMenu2 == _PM2_MOTOR_ADR)
     {
          //i = _PM7_Velct;
          _POS = _LCDStartAddrPos;
          _Sign = _UNSIGNED;
          _dNum = DDS.NODE_ADDRESS;
          _DPlc = _MADenaryPlaces;
          j = _DecimalPlacesZero;
          pArr = caMotAddr;
     }
```

```
                else
                {
                        i = _PM7_Val;
                        _POS = _LCDStartPosUNum;
                        _Sign = _UNSIGNED;
                        if(cMenu2 == _PM2_SAFETY_LIM)
                                _dNum =
                                DDS.SOFT_POS_SAFETY_LIM_MM;
                        else if(cMenu2 ==
                        _PM2_DRIVE_LIM)
                                _dNum =
                                DDS.SOFT_POS_DRIVE_LIM_MM;
                        else if(cMenu2 ==
                        _PM2_POSITION_LIM)
                                _dNum =
                                DDS.PATH_LENGTH_MM;
                        _DPlc = _DenaryPlaces;
                        j = _DecimalPlaces;
                        pArr = caNum;
                }

                /* Převod čísla typu float na řetězec znaků a jeho tisk na
                LCD. Předávané parametry jsou: ukazatel na pole
                znaků, číselná hodnota, počet dekadických míst, počet
                desetinných míst, je-li to číslo se znaménkem. */
                fltoa(pArr, _dNum, _DPlc, j, _Sign);
                PrintTextRAMLCD(pArr, _POS);

                /* Nastavení pozice kurzoru a jeho zobrazení na LCD.
                */
                if(cMenu2 == _PM2_MOTOR_ADR)
                        _POS += 2;
                SetCursorPos(_POS);
                cPosition = _POS;
                SetCursor(ON);
        }
}
/* Uložení nastavených položek nastavení motoru. V případě, že
některá z položek není nastavena, je uživatel vyzván k doplnění této
hodnoty. Je-li vše v pořádku, je na LCD vypsáno " Odesílání dat pro
motor", délkové rozměry převedeny na inkrementy motoru, odeslány
motoru a uloženy do Flash paměti. */
else if (cMenu2 == _PM2_END_SETING)
{
        if(!(DDS.SOFT_POS_SAFETY_LIM_MM &&
        DDS.SOFT_POS_DRIVE_LIM_MM &&
        DDS.PATH_LENGTH_MM))
        {
                if(!DDS.SOFT_POS_SAFETY_LIM_MM)
                        cMenu2 = _PM2_SAFETY_LIM;
                else if(!DDS.SOFT_POS_DRIVE_LIM_MM)
                        cMenu2 = _PM2_DRIVE_LIM;
```

```
                    else if(!DDS.PATH_LENGTH_MM)
                         cMenu2 = _PM2_POSITION_LIM;
                    PrintTextPGMLCD(PrintValueFault,
                    LCD_ROW_1);
                    PrintTextPGMLCD(pPrintedMenu2[cMenu2]
                    , LCD_ROW_2);
                    for (i=0; i<2; i++)   // pauza 1 sec
                         Delay10KTCYx(250);
                    ClearLCD();
                    PrintTextPGMLCD(pPrintedMenu2[cMenu2]
                    , LCD_ROW_1);
               }
               else
               {
                    cLevel = cPrevLevel;
                    PrintTextPGMLCD(
                    PrintCAN1,LCD_ROW_1);
                    PrintTextPGMLCD(
                    PrintCAN2,LCD_ROW_2);
                    _Rate = DDS.GEAR_INC_RATIO /
                    DDS.SHIFT_PER_REVOLUTION;
                    SPL = Roundx((DDS.PATH_LENGTH_MM *
                    _Rate));
                    SPDL =
                    Roundx((DDS.SOFT_POS_DRIVE_LIM_MM *
                    _Rate));
                    SPSL =
                    Roundx((DDS.SOFT_POS_SAFETY_LIM_MM *
                    _Rate));
                    SendMotorSetup (SPL, SPDL, SPSL);
                    FlashTBLWrite(&ROMDDS, &DDS,
                    sizeof(DDS));
                    ClearLCD();
                    cMenu1 = _PM1_TRACE_SETUP;
                    PrintTextPGMLCD(pPrintedMenu1[cMenu1]
                    , LCD_ROW_1);
               }
          }
          // Servisní nastavení motoru na počáteční pozici.
          else if(cMenu2 == _PM2_0_INC)
          {
               PrintTextPGMLCD(PrintMotion,LCD_ROW_1);
               SetStartPos();
               PrintTextPGMLCD(pPrintedMenu2[cMenu2],
               LCD_ROW_1);
          }
     }
     // Vstup do jednotlivých položek menu pro nastavení trasy.
     else if (cLevel == _LCDMovementSetUp)
     {
          cPrevLevel = cLevel;
```

/* Položka aktuální pozice v tomto menu, získá pozici z motoru. V případě, že již je nastaven jeden nebo více uzlů trasy, jsou jejich hodnoty přičteny k aktuální pozici motoru a poté je výsledná hodnota zobrazena na LCD. */

```
if(cMenu3 == _PM3_ACT_POS)
{
      cLevel = _CANActPos;
      _dNum = DDS.ACTUAL_POSITION_MM;
      j = GetLastUsedPosition(Data_ptr);
      for(i=0;i<=j;i++)
            _dNum += ((double)(Data_ptr+i)->
            Target_Inc / DDS.GEAR_INC_RATIO)*
            DDS.SHIFT_PER_REVOLUTION;
      fltoa(_Num, _dNum,5,2,_UNSIGNED);
      PrintTextRAMLCD(_Num,LCD_ROW_2);
}
else if ((cMenu3 == _PM3_SHIFT_POS) ||
(cMenu3 == _PM3_VELOCITY)
cMenu3 == _PM3_ACCELERATION) ||
(cMenu3 == _PM3_DECELERATION))
{
      cLevel = _LCDNumber;
      ClearLCD();
      PrintTextPGMLCD(pPrintedMenu6[cMenu3 -
       _SUBMENU_SHIFT], LCD_ROW_1);
```

/* Nastavení položky "Posun od aktuální pozice". Byla-li již nějaká hodnota nastavena a uživatel se vrací k nastavení této hodnoty, je tato hodnota opět zobrazena. Jinak je zobrazena nulová hodnota. */

```
      if (cMenu3 == _PM3_SHIFT_POS)
      {
            _POS = _LCDStartPosSNum;
            _Sign = _SIGNED;
            _dNum = (((double)(Data_ptr +
            NodeNum)-> Target_Inc *
            DDS.SHIFT_PER_REVOLUTION)/
            DDS.GEAR_INC_RATIO);
            pArr = caNum;
            _DPlc = _DenaryPlaces;
            j = _DecimalPlaces;
      }
```

/* Nastavení položky "Rychlost". Byla-li již nějaká hodnota nastavena a uživatel se vrací k nastavení této hodnoty, je tato hodnota opět zobrazena. Jinak je zobrazena nulová hodnota. */

```
      else if (cMenu3 == _PM3_VELOCITY)
      {
            _POS = _LCDStartPosVel;
            _Sign = _UNSIGNED;
            _dNum = ((Data_ptr+NodeNum)->
            Velocity_Inc / DDS.GEAR_INC_RATIO) *
            DDS.SHIFT_PER_REVOLUTION;
```

```
            pArr = caShift;
            _DPlc = _ShPRDenaryPlaces;
            j = _DecimalPlaces;
      }
      /* Nastavení položky "Zrychlení" a "Zpomalení". Byla-li již
      nějaká hodnota nastavena a uživatel se vrací k nastavení této
      hodnoty, je tato hodnota opět zobrazena. Jinak je zobrazena
      nulová hodnota. */
      else
      {
            _POS = _LCDStartPosPct;
            _Sign = _UNSIGNED;
            if(cMenu3 == _PM3_ACCELERATION)
                  _dNum = ((Data_ptr+NodeNum)->
                  Acceleration_Inc
                  /_Accel_Increment) - 1;
            else if(cMenu3 == _PM3_DECELERATION)
                  _dNum = ((Data_ptr+NodeNum)->
                  Deceleration_Inc
                  /_Decel_Increment) - 1;
            _DPlc = _ShPRDenaryPlaces;
            pArr = caPct;
            j = _DecimalPlacesZero;
            PrintTextPGMLCD(pPrintedMenu7[
            _PM7_Pct], LCD_ROW_2);
      }

      /* Převod čísla typu FLOAT na řetězec znaků a jeho vytisknutí
      na LCD. */
      fltoa(pArr, _dNum, _DPlc, j, _Sign);
      PrintTextRAMLCD(pArr, _POS);
      SetCursorPos(_POS);

      /* Nastavení pozice kurzoru a její zobrazení na LCD. */
      cPosition = _POS;
      SetCursor(ON);
}
/* Volba položky "Další pozice" označí uzel jako použitý, najde nový
volný uzel a zkopíruje do něj hodnoty rychlosti, zrychlení a zpomalení z
předchozího uzlu. Jestliže není již žádný uzel volný, oznámí to
uživateli. */
else if(cMenu3 == _PM3_NEXT_POSITION)
{
      (Data_ptr+NodeNum)->Used = 1;
      i = GetFreePosition(Data_ptr);
      if (i != 0x7f)
      {
            (Data_ptr+i)->Acceleration_Inc =
            (Data_ptr+NodeNum)->Acceleration_Inc;
            (Data_ptr+i)->Deceleration_Inc =
            (Data_ptr+NodeNum)->Deceleration_Inc;
```

```
            (Data_ptr+i)->Velocity_Inc =
            (Data_ptr+NodeNum)->Velocity_Inc;
      }
      NodeNum  = i;
      cPrevLevel = _LCDMainMenu;
      cLevel = _LCDMovementSetUp;
      cMenu3 = _PM3_SHIFT_POS;
      ClearLCD();
      if(NodeNum == 0x7f)
      {
            PrintTextPGMLCD(PrintNodeFault,
            LCD_ROW_1);
            cMenu3 = _PM3_START;
      }
      else
            PrintTextPGMLCD(PrintedMenu31b,
            LCD_ROW_2);
      PrintTextPGMLCD(pPrintedMenu3[cMenu3],
      LCD_ROW_1);
}
```

/* Volba položky "Uložit pozici", stejně jako předchozí položka, označí uzel jako použitý. S tím rozdílem, že nezpůsobí přechod na nový uzel trasy. Na LCD vytiskne "Uloženo". */

```
else if(cMenu3 == _PM3_SAVE_POSITION)
{
      cPrevLevel = _LCDMainMenu;
      (Data_ptr+NodeNum)->Used = 1;
      PrintTextPGMLCD(pPrintedMenu8[_PM8_SAVED],
      LCD_ROW_1);
      for (i=0; i<2; i++)   // pauza 1 sec
            Delay10KTCYx(250);
      PrintTextPGMLCD(pPrintedMenu3[cMenu3],
      LCD_ROW_1);
      for (i=0; i<2; i++)   // pauza 1 sec
            Delay10KTCYx(250);
}
```

/* Položka "Upravit pozici" umožní výběr uzlu trasy, který chce uživatel opravit. Nabídne uživateli všechna čísla uzlů, které již byly uloženy. */

```
else if(cMenu3 == _PM3_EDIT_POSITION)
{
     cLevel = _EditPosNum;
     DataNodeLevel =
     GetLastUsedPosition(Data_ptr);
     i = DataNodeLevel + 1;
     PrintTextPGMLCD(pPrintedMenu4[
     _PM4_SELECT_POS_1row], LCD_ROW_1);
     PrintTextPGMLCD(pPrintedMenu4[
     _PM4_SELECT_POS_2row], LCD_ROW_2);
     btoa(i, _Num);
     _POS = _LCDEndPosition - strlen(_Num);
     PrintTextRAMLCD(_Num, _POS);
```

```
            cPosition = _POS;
            SetCursorPos(_POS);
            SetCursor(ON);
        }
        /* Položka "Start" slouží pro zahájení pohybu motoru po nastavených
        uzlech trasy. Nastaví proměnnou eStart na hodnotu TRUE, která
        způsobí ukončení smyčky provádějící obsluhu ovládacích tlačítek v
        souboru main.c. */
        else if(cMenu3 == _PM3_START)
        {
            eStart = TRUE;
            PrintTextPGMLCD(PrintCAN1 ,LCD_ROW_1);
            PrintTextPGMLCD(PrintCAN2 ,LCD_ROW_2);
        }

    }
    // Úroveň nastavování číselných hodnot.
    else if (cLevel == _LCDNumber)
    {
        // Vypnutí kurzoru na LCD.
        SetCursor(OFF);
        if (cPrevLevel == _LCDMotorSetUp)
        {

            /* Potvrzení volby adresy motoru. Adresa je uložena do Flash
            paměti a následuje vynulování řetězce obsahujícího hodnotu
            adresy. */
            if(cMenu2 == _PM2_MOTOR_ADR)
            {
                i = DDS.NODE_ADDRESS;
                memcpypgm2ram(&DDS,(const rom void*)
                &ROMDDS, sizeof(ROMDDS));
                DDS.NODE_ADDRESS = i;
                FlashTBLWrite(&ROMDDS, &DDS,
                sizeof(DDS));
                VarClr(caMotAddr);
                _CONTINUE = TRUE;
            }
            /* Uložení délky dráhy do pracovní datové struktury DDS a
            následuje vynulování řetězce obsahujícího hodnotu délky dráhy.
            */
            else if(cMenu2 == _PM2_POSITION_LIM)
            {
                DDS.PATH_LENGTH_MM = atof(caNum);
                    // v mm
                VarClr(caNum);
            }
            /* Uložení hodnoty ochranného pásma do pracovní datové
            struktury DDS a následuje vynulování řetězce obsahujícího
            hodnotu ochranného pásma. */
            else if(cMenu2 == _PM2_DRIVE_LIM)
            {
```

```
            // v mm
            DDS.SOFT_POS_DRIVE_LIM_MM =
            atof(caNum);
            VarClr(caNum);
        }
        /* Uložení hodnoty bezpečnostního pásma do pracovní datové
        struktury DDS a následuje vynulování řetězce obsahujícího
        hodnotu bezpečnostního pásma. */
        else if(cMenu2 == _PM2_SAFETY_LIM)
        {
            // v mm
            DDS.SOFT_POS_SAFETY_LIM_MM =
            atof(caNum);
            VarClr(caNum);
        }
        /* Uložení hodnoty posunu v mm/ot. do pracovní datové
        struktury DDS a následuje vynulování řetězce obsahujícího
        hodnotu posunu v mm/ot. */
        else if(cMenu2 == _PM2_SHIFT_PER_REV)
        {
            //[mm za otáčku]
            DDS.SHIFT_PER_REVOLUTION =
            atof(caShift);
            VarClr(caShift);
        }

        /* Vymazání displeje a výpis aktuální položky menu nastavení
        motoru. */
        ClearLCD();
        PrintTextPGMLCD( pPrintedMenu2[cMenu2],
        LCD_ROW_1);
    }
    // Uložení položek nabídek "Nastavení trasy" a "Manuální řízení".
    else if ((cPrevLevel == _LCDMovementSetUp) ||
    (cPrevLevel == _ManMove))
    {
        /* Uloží hodnotu položky "posun od aktuální pozice" do
        aktuálního uzlu v inkrementech a následuje vynulování řetězce
        obsahujícího hodnotu posunu. */
        if(cMenu3 == _PM3_SHIFT_POS)
        {
            Target_pos = atof(caNum);  // v mm
            (Data_ptr+NodeNum)->Target_Inc =
            Roundx(Target_pos /
            DDS.SHIFT_PER_REVOLUTION *
            DDS.GEAR_INC_RATIO);  //v inkrementech
            VarClr(caNum);
        }
        /* Uloží hodnotu položky "Rychlost" do aktuálního uzlu v
        inkrementech za sekundu a následuje vynulování řetězce
        obsahujícího hodnotu rychlosti. */
        else if(cMenu3 == _PM3_VELOCITY)
```

```
                {
                        _dNum = atof(caShift);
                        (Data_ptr+NodeNum)->Velocity_Inc =
                        Roundx((_dNum /
                        DDS.SHIFT_PER_REVOLUTION)*
                        DDS.GEAR_INC_RATIO);  /* v inkrementech za
                        sekundu */
                        VarClr(caNum);
                }
                /* Uloží hodnotu položky "Zrychlení" do aktuálního uzlu v
                inkrementech za sekundu² a následuje vynulování řetězce
                obsahujícího hodnotu zrychlení. */
                else if(cMenu3 == _PM3_ACCELERATION)
                {
                        Prof_accel = atol(caPct);  // v %
                        Prof_accel++;
                        (Data_ptr+NodeNum)->Acceleration_Inc
                        = Prof_accel * _Accel_Increment; /* v
                        inkrementech za sekundu^2 */
                        VarClr(caPct);
                }
                /* Uloží hodnotu položky "Zpomalení" do aktuálního uzlu v
                inkrementech za sekundu² a následuje vynulování řetězce
                obsahujícího hodnotu zpomalení.  */
                else if(cMenu3 == _PM3_DECELERATION)
                {
                        Prof_decel = atol(caPct);  // v %
                        Prof_decel++;
                        (Data_ptr+NodeNum)->Deceleration_Inc
                        = Prof_decel * _Decel_Increment;
                                /* v inkrementech za sekundu^2 */
                        VarClr(caPct);
                }

                /* Vymazání displeje a výpis aktuální položky menu nastavení
                trasy. */
                ClearLCD();
                PrintTextPGMLCD( pPrintedMenu3[cMenu3],
                LCD_ROW_1);
                if (cMenu3 == _PM3_SHIFT_POS)
                        PrintTextPGMLCD(PrintedMenu31b,
                        LCD_ROW_2);
        }
        // Přechod na předchozí úroveň menu.
        cLevel = cPrevLevel;
        cPrevLevel = _LCDMainMenu;
        Num_i = 0;
}
/* Přechod z úrovně zobrazení aktuální pozice do menu nastavení trasy položka
"Aktuální pozice". */
else if(cLevel == _CANActPos)
{
```

```
        cLevel = cPrevLevel;
        cPrevLevel = _LCDMainMenu;
        ClearLCD();
        PrintTextPGMLCD( pPrintedMenu3[cMenu3],
        LCD_ROW_1);
}
/* Výstup z položky "Upravit pozici" způsobí načtení zvoleného uzlu do
jednotlivých položek nastavení trasy a umožní tak jeho editaci. */
else if(cLevel == _EditPosNum)
{
        cLevel = cPrevLevel;
        cPrevLevel = _LCDMainMenu;
        cMenu3 = _PM3_SHIFT_POS;
        NodeNum = DataNodeLevel;
        ClearLCD();
        PrintTextPGMLCD(pPrintedMenu3[cMenu3],
        LCD_ROW_1);
        PrintTextPGMLCD(PrintedMenu31b, LCD_ROW_2);
}
// Úroveň manuálního řízení.
else if(cLevel == _ManMove)
{
        /* Přístup k nastavení položek rychlosti, zrychlení a zpomalení. */
        cPrevLevel = cLevel;
        if ((cMenu3 == _PM3_VELOCITY) ||
        (cMenu3 == _PM3_ACCELERATION) ||
        (cMenu3 == _PM3_DECELERATION))
        {
                cLevel = _LCDNumber;
                ClearLCD();
                PrintTextPGMLCD(pPrintedMenu6[cMenu3 -
                _SUBMENU_SHIFT], LCD_ROW_1);

                // Nastavení formátu čísla.
                if (cMenu3 == _PM3_VELOCITY)
                {
                        _POS  = _LCDStartPosVel;
                        _Sign      = _UNSIGNED;
                        _dNum      = ((Data_ptr+NodeNum)->
                        Velocity_Inc / DDS.GEAR_INC_RATIO)*
                        DDS.SHIFT_PER_REVOLUTION;
                        pArr  = caShift;
                        _DPlc      = _ShPRDenaryPlaces;
                        j          = _DecimalPlaces;
                }
                else
                {
                        _POS  = _LCDStartPosPct;
                        _Sign = _UNSIGNED;
                        if(cMenu3 == _PM3_ACCELERATION)
```

```
                                    _dNum = (Data_ptr->
                                    Acceleration_Inc /
                                    _Accel_Increment) - 1;
                              else if(cMenu3 == _PM3_DECELERATION)
                                    _dNum = (Data_ptr->
                                    Deceleration_Inc /
                                    _Decel_Increment) - 1;
                              _DPlc = _ShPRDenaryPlaces;
                              pArr = caPct;
                              j = _DecimalPlacesZero;
                        }
                        /* Převedení čísla na textový řetězec a jeho tisk na LCD. */
                        fltoa(pArr, _dNum, _DPlc, j, _Sign);
                        PrintTextRAMLCD(pArr, _POS);

                        /* Nastavení pozice kurzoru a jeho zobrazení na LCD. */
                        SetCursorPos(_POS);
                        cPosition = _POS;
                        SetCursor(ON);
                  }
                  /* Spuštění manuálního řízení, na LCD je zobrazeno hlášení o volbě
                  směru posunu */
                  else if(cMenu3 == _PM3_START)
                  {
                        eManual = TRUE;
                        PrintTextPGMLCD(PrintManMove_OrientationA,
                        LCD_ROW_1);
                        PrintTextPGMLCD(PrintManMove_OrientationB,
                        LCD_ROW_2);
                  }
            }
      }
}

/* Funkce VarClr přebírá ukazatel na pole znaků a nastavuje jednotlivé proměnné, které
obsahovaly číslice na hodnotu "0" ASCII. */
void VarClr(char* _Var)
{
      size_t _VarLen;
      int i;

      _VarLen = strlen(_Var);
      for(i=0;(unsigned)i<_VarLen;i++)
            if (((unsigned)*(_Var+i)>='0') &&
            ((unsigned)*(_Var+i)<='9'))
                  *(_Var+i)='0';
            else
                  continue;
}
```

/* Funkce ChkVal přebírá ukazatel na pole znaků a porovnává uživatelem nastavenou hodnotu v tomto poli obsaženou s maximální nebo minimální přípustnou hodnotou pro daný parametr. Je-li hodnota větší než maximum nebo menší než minimum, je do argumentu zapsána hodnota maxima nebo minima a funkce vrací hodnotu 1 jako příznak změny. Je-li hodnota argumentu ponechána beze změny, vrací funkce hodnotu 0. */

```
char ChkVal(char* pArg)
{
#define _Min 0

      double dVal = 0.0, dAbsVal = 0.0, dMax = 0.0, dRes = 0.0;
      long lMax = 0, lMin = 0, lVal = 0;
      char _RES = 0, _DecP, _DenP;
      eSignum _S;
      int i;

      // Převod textového řetězce na číselnou hodnotu typu FLOAT.
      dVal = atof(pArg); // v [mm]

      // Uložení hodnoty do pomocné proměnné.
      dRes = dVal;

      // Získání absolutní hodnoty.
      dAbsVal = fabs(dVal); // Absolutní hodnota v [mm]

      // Získávání maximálních a minimálních přípustných hodnot argumentu.
      if(cPrevLevel == _LCDMotorSetUp)
      {
            if(cMenu2 == _PM2_SAFETY_LIM)
            {
                  dMax = (2*DDS.SOFT_POS_DRIVE_LIM_MM) +
                  DDS.PATH_LENGTH_MM;
                  // Desetinná hodnota [mm].
                  dMax *= (DDS.GEAR_INC_RATIO /
                  DDS.SHIFT_PER_REVOLUTION);
                  // Desetinná hodnota [Inc].
                  lMax = Roundx(dMax); // 32 bitová celá hodnota [Inc]
                  lMax = _SSL_MAX - lMax;
                  // Maximální nastavitelná hodnota [Inc].
                  _DecP      = _DecimalPlaces;
                  _DenP      = _DenaryPlaces;
                  _S         = _UNSIGNED;
            }

            else if(cMenu2 == _PM2_DRIVE_LIM)
            {
                  dMax = (2*DDS.SOFT_POS_SAFETY_LIM_MM) +
                  DDS.PATH_LENGTH_MM; // desetinná hodnota [mm]
                  dMax *= (DDS.GEAR_INC_RATIO /
                  DDS.SHIFT_PER_REVOLUTION);
                  // Desetinná hodnota [Inc].
                  lMax = Roundx(dMax); // 32 bitová celá hodnota [Inc]
                  lMax = _SDL_MAX - lMax;
```

```
                // Maximální nastavitelná hodnota [Inc].
                _DecP      = _DecimalPlaces;
                _DenP      = _DenaryPlaces;
                _S         = _UNSIGNED;
        }
        else if(cMenu2 == _PM2_POSITION_LIM)
        {
                dMax  = 2*(DDS.SOFT_POS_SAFETY_LIM_MM +
                DDS.SOFT_POS_DRIVE_LIM_MM);  // desetiná hodnota [mm]
                dMax  *= (DDS.GEAR_INC_RATIO /
                DDS.SHIFT_PER_REVOLUTION);
                // Desetinná hodnota [Inc].
                lMax  = Roundx(dMax);  // 32 bitová celá hodnota [Inc]
                lMax  = _SPL_MAX - lMax;
                // Maximální nastavitelná hodnota [Inc].
                _DecP      = _DecimalPlaces;
                _DenP      = _DenaryPlaces;
                _S         = _UNSIGNED;
        }
        // Hodnoty ostatních proměnných není potřeba kontrolovat.
        else
                goto ChkValEnd;
}
else if(cPrevLevel == _LCDMovementSetUp)
{
        if(cMenu3 == _PM3_SHIFT_POS)
        {
                dMax  = DDS.PATH_LENGTH_MM;
                // Desetinná hodnota [mm].
                dMax  *= (DDS.GEAR_INC_RATIO /
                DDS.SHIFT_PER_REVOLUTION);
                // Desetinná hodnota [Inc].
                lMax  = Roundx(dMax);  // 32 bitová celá hodnota [Inc].
                lMin  = _Min;     // Minimální nastavitelná hodnota [Inc]
                dMax  = DDS.ACTUAL_POSITION_MM;
                dMax  *= (DDS.GEAR_INC_RATIO /
                DDS.SHIFT_PER_REVOLUTION);
                // Desetinná hodnota [Inc].
                lMax  -= Roundx(dMax);// Long integer value[Inc].
                lMin  += Roundx(dMax);// 32 bitová celá hodnota [Inc].

                /* Získávání maximální a minimální nastavitelné hodnoty parametru z
                hodnot předchozích uložených uzlů. Vezme se hodnota maximálního
                možného posunu daným směrem a odečte se od ní (pohyb vlevo), nebo
                se k ní přičte (pohyb vpravo) hodnota každého uloženého uzlu. */
                i = 0;

                while((Data_ptr+i)->Used)
                {
                        lMax -= (Data_ptr+i)->Target_Inc;
                        lMin += (Data_ptr+i)->Target_Inc;
                        i++;
```

```
            }
            _DecP      = _DecimalPlaces;
            _DenP      = _DenaryPlaces;
            _S         = _SIGNED;
      }
      else if(cMenu3 == _PM3_VELOCITY)
      {
            lMax  = _Max_Velocity;
            _DecP      = _DecimalPlaces;
            _DenP      = _ShPRDenaryPlaces;
            _S         = _UNSIGNED;
      }
      // Hodnoty zrychlení není potřeba kontrolovat.
      else if(cMenu3 == _PM3_ACCELERATION)
            goto ChkValEnd;
      // Hodnoty zpomalení není potřeba kontrolovat.
      else if(cMenu3 == _PM3_DECELERATION)
            goto ChkValEnd;
}
else if (cPrevLevel == _ManMove)
{
      if(cMenu3 == _PM3_VELOCITY)
      {
            lMax  = _Max_Velocity;
            _DecP      = _DecimalPlaces;
            _DenP      = _ShPRDenaryPlaces;
            _S         = _UNSIGNED;
      }
      // Hodnoty zrychlení není potřeba kontrolovat.
      else if(cMenu3 == _PM3_ACCELERATION)
            goto ChkValEnd;
      // Hodnoty zpomalení není potřeba kontrolovat.
      else if(cMenu3 == _PM3_DECELERATION)
            goto ChkValEnd;
}

// Absolutní hodnota argumentu převedená na inkrementy motoru.
dAbsVal *= (DDS.GEAR_INC_RATIO / DDS.SHIFT_PER_REVOLUTION);

// Totéž jako předchozí proměnná dAbsVal, ale celočíselně.
lVal = Roundx(dAbsVal);

/* Porovnávací funkce, porovná hodnotu argumentu proti maximální nebo minimální
možné hodnotě. */
if((cPrevLevel == _LCDMotorSetUp) ||
((cPrevLevel == _ManMove) && (cMenu3 == _PM3_VELOCITY)) ||
((cPrevLevel == _LCDMovementSetUp) &&
(cMenu3 == _PM3_VELOCITY)))
{
      if(lVal > lMax)
      {
            dMax = lMax;
```

```
                dRes = (dMax / DDS.GEAR_INC_RATIO) *
                DDS.SHIFT_PER_REVOLUTION;        // [mm]
                fltoa(pArg, dRes, _DenP, _DecP, _S);
                _RES = 1;
            }
            else
                goto ChkValEnd;
        }
        else if ((cPrevLevel == _LCDMovementSetUp) &&
        (cMenu3 == _PM3_SHIFT_POS))
        {
            if(dVal >= 0)
            {
                if(lVal > lMax)
                {
                    dMax = lMax;
                    dRes = (dMax / DDS.GEAR_INC_RATIO) *
                    DDS.SHIFT_PER_REVOLUTION;        // [mm]
                    fltoa(pArg, dRes, _DenP, _DecP, _S);
                    _RES = 1;
                }
                else
                    goto ChkValEnd;
            }
            else
            {
                if(lVal > lMin)
                {
                    dVal = lMin;
                    dRes = (dVal / DDS.GEAR_INC_RATIO) *
                    DDS.SHIFT_PER_REVOLUTION;        // [mm]
                    dRes *= -1;
                    fltoa(pArg, dRes, _DenP, _DecP, _S);
                    _RES = 1;
                }
                else
                    goto ChkValEnd;
            }
        }
ChkValEnd:
return _RES;
}
/*--- K O N E C  D E F I N I C E  F U N K C Í ---------------------------*/
```

9. ZÁVĚR

Cílem diplomové práce bylo realizovat polohové řízení DC motoru Icla D065 firmy SIG Positec Automation GmbH pomocí sběrnice CAN a protokolu CANopen. Popsat standardy sběrnice CAN a protokolu CANopen, stejně jako platformu použitého 8 bitového mikrořadiče PIC18F4685 firmy Microchip, zkonstruovat komunikační rozhraní a vytvořit jeho elektronickou dokumentaci.

Na základě doporučené literatury a internetových zdrojů byla vypracována teoretická část práce zabývající obecnými principy CAN sběrnice a protokolu CANopen. Dále byl popsán vybraný mikroprocesor PIC18F4685 s ohledem na aplikaci v komunikačním rozhraní, pohon IclA D065 a komunikační rozhraní „CAN ovládaní pohonu verze 1.0“ včetně demonstrační úlohy jednoosého suportu.

Po naprogramování řídícího programu mikroprocesoru a zkompilování výsledné pracovní verze bylo zjištěno, že pro potřeby dané aplikace lze použít i mikroprocesory nižších řad, které obsahují minimálně 48 kB vnitřní programové flash paměti, neboť výsledná velikost programu je 42 kB. Po nahrání programu do komunikačního rozhraní a připojení komunikačního rozhraní k pohonu IclA D065 spolu se suportem, byla ověřena funkčnost rozhraní jak manuálním řízením, tak přednastavením bodů trasy.

Vybraný mikroprocesor PIC18F4685 je vysoce výkonný a bylo by možné s jeho pomocí řídit více pohonů současně např.: pohyb ve dvou nebo třech osách, nebo synchronizovaný pohyb až 126 pohonů. Lze předpokládat, po praktických zkouškách na demonstračním jednoosém suportu, že by jej bylo možné využít i v náročných úlohách jako je např. řízení obráběcích CNC center.

Jako součást elektronické přílohy diplomové práce jsou dodány kompletní zdrojové kódy vytvořené aplikace a návrh desky plošného spoje. Tyto zdroje mohou být dále využity pro vývoj a zdokonalování ovládání pohonu po CAN sběrnici. Dalším vývojovým stupněm by tak mohlo např. být ovládání pohonů ve dvou a více osách po CAN sběrnici.

SEZNAM POUŽITÉ LITERATURY

[1] CAN in Automation. CiA [on line]. 2001, June 2005 [cit. 2009-03-12]. Dostupné z: <www.canopen.org>.

[2] Softing AG. Softing [on line]. 2003, 1st of January [cit. 2009-03-20]. Dostupné z: <http://www.softing.com/home/en/index.php>.

[3] BOTERNBROOD, H. *CANopen, high-level protokol for CAN-bus Version 3.0*. [PDF dokument]. NIKHEF, Amsterdam, March 2000 [cit. 2010-05-20]. Dostupné z: <http://www.nikhef.nl/pub/departments/ct/po/doc/CANopen30.pdf>.

[4] Atmel Corporation [databáze on line]. California, San Jose: Atmel Corporation, 2008. [cit. 2010-05-14]. Dostupné z: <http://www.atmel.com/dyn/products/param_table_v2.asp?family_id=607&OrderBy=part_no&Direction=ASC>.

[5] Microchip technology Inc. [databáze on line]. USA, Arizona, Chandler, 2006. [cit. 2010-05-14]. dostupné z:http://www.microchip.com/ParamChartSearch/chart.aspx?branchID=50&mid=10&lang=en&pageId=7414.května 2010

[6] 8-bit PIC Microcontroller with ECAN Technology [on line]. 2006, January 2007 [cit. 2010-03-05]. Dostupné z: <http://ww1.microchip.com/downloads/en/DeviceDoc/39761c.pdf >

[7] Schneider Electric Motion Deutschland GmbH. Schneider electric [on line]. 2006, April 2008 [cit. 2009-01-10]. Dostupné z: <http://www.schneider-electric-motion.com/ leden 2009>.