ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA ELEKTROTECHNICKÁ

Katedra aplikované elektroniky a telekomunikací

DIPLOMOVÁ PRÁCE

MicroCANopen Plus na platformě STM32

Marek Novák **2019**

Originál (kopie) zadání BP/DP

Abstrakt

Předkládaná diplomová práce je zaměřena na implementaci protokolového zásobníku MicroCANopen Plus na platformu STM32. V teoretické části práce je nejprve popsána specifikace standardu CAN. To znamená způsob přístupu na sběrnici, typy rámců, detekce chyb a další. Potom jde o teorie zaměřené na samotný protokol MicroCANopen Plus. Jsou zde popsány základní možnosti tohoto protokolu, jako jsou například procesní a servisní datové objekty nebo network management. V poslední části práce je pak podrobně popsána samotná realizace protokolového zásobníku. Jsou zde vysvětleny jednotlivé funkce a struktura programu.

Klíčová slova

Sběrnice, Protokol stack, MicroCANopen Plus,

Abstract

This master's thesis deals with the implementation of the protocol stack MicroCANopen Plus on the STM32 platform. At first, the theoretical part presents specifications of CAN standard, e.g. access methods, types of frames, error detection etc. A description and basic options of a protocol MicroCANopen Plus are areas of interest as well, e.g. Service and Process Data Objects or network management. The realization of protocol stack itself is discussed in the practical part of the thesis. The program's structure and implemented functions are described in detail.

Key words

Bus, Protocol stack, MicroCANopen Plus,

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně, s použitím odborné literatury a pramenů uvedených v seznamu, který je součástí této diplomové práce.
Dále prohlašuji, že veškerý software, použitý při řešení této diplomové práce, je legální.

.....

podpis

V Plzni dne 27.5.2019 Marek Novák

Poděkování

Tímto bych rád poděkoval vedoucímu diplomové práce Ing. Petru Kristovi, Ph.D., za poskytnutí užitečných rad a cenných zkušeností při realizaci této diplomové práce.

Obsah

Úvod

Cílem diplomové práce je implementování protokolového zásobníku MicroCANopen Plus na platformu STM32. Jako zástupce platformy STM32 se použil vývojový kit Nucleo F429ZI, který obsahuje CAN periférii. K tomuto kitu byl následně připojen modul, který obsahoval CAN budič s konektory a základními perifériemi, jako jsou tlačítka LED diody pro účely testování a vizualizace. Celé testování a kontrola funkčnosti byly umožněny za pomoci převodníku CAN-USB od firmy IMFsoft. Tento převodník umožnil propojení CAN modulu s počítačem přes USB port. V počítači je nainstalován program, který umožňuje kompletní kontrolu komunikace CAN. Lze zde odesílat a přijímat PDO, používat SDO služby, network management a mnoho dalšího.

V práci je provedena základní implementace možností protokolu s důrazem na rozdělení programu na část závislou na hardwaru a na část realizující samotný protokolový zásobník, který by měl být co nejvíce univerzální a nezávislý na cílové platformě. Jsou dvě možnosti, jak implementaci udělat. Jednou je koupení již hotového produktu, ten pro svoje zařízení přizpůsobit a dopsat části, které se týkají přímo platformy. Druhou možností je si vše napsat sám, jako v případě této diplomové práce. Při realizaci byla využita implementace na základě doporučení www.microcanopen.com, kde je možnost nahlédnout do ukázkových zdrojových kódů [4] a stáhnout manuál pro implementaci [5]. V rámci diplomové práce byly implementovány následující služby: servisní a datové služby, network management, adresář deskriptorů a heartbeat. V praxi se lze ve výsledku setkat s mnoha případy, kdy dvě firmy vlastní MicroCANopen Plus nebo jiný vyšší protokol. Po propojení systému ani jedna strana nefunguje správně nebo funguje jen částečně a podobně. Chyba bývá v nedodržení předepsaných pravidel, jako jsou například identifikátory komunikačních objektů COB ID.

V následujících kapitolách se nejprve věnuji popisu specifikace standardu CAN. Řeším například přístup na sběrnice, detekci chyb, komunikační protokol a jiné věci potřebné pro další část diplomové práce. Následuje popis MicroCANopen Plus protokolu. Jedná se o adresář deskriptorů, procesní datové objekty, servisní datové objekty, network management a emergenci. V poslední části práce je věnován prostor realizaci MicroCANopen Plus na platformě STM32. Jsou tu popsány jednotlivé funkce, struktura programu a jeho funkčnost.

Seznam symbolů a zkratek

CAL..... CAN Applications Layer

CAN ..... Controller Area Network

CIA ..... CAN In Automation

COB-ID ..... Komunikační objekt (Communication Object) = CAN-ID

EMCY..... Chybové zprávy a alarmy (Emergency)

LLC ..... Logic Link Control

MAC..... Medium Access Control

NMT ..... Řízení a diagnostika uzlu (Network management)

Node-ID..... Identifikátor uzlu

OD ..... Adresář deskriptorů (Object Dictionary)

PDO ..... Procesní datové objekty (Process Data Objects)

RPDO ..... Přijímací procesní datové objekty (Receive Process Data Objects)

SDO ..... Servisní datové objekty (Service Data Objects)

SYNC ..... Synchronizace zpráv (Synchronisation Object)

TPDO ..... Vysílací procesní datové objekty (Transmit Process Data Objects)

1 CAN bus

Controller Area Network neboli CAN byl vyvinut firmou Bosch v osmdesátých letech. Jedná se o sériovou sběrnici, která měla původně sloužit pro automobilový průmysl. Důvodem nasazení byl rozvoj elektrotechniky a přibývání elektrických komponentů pro komfort, pohodlí a bezpečnost auta i posádky. Mezi tyto prvky řadíme například ABS, airbagy, centrální zamykání, brzdový asistent či klimatizaci. Jejich počet s dobou a každým dalším modelem auta stoupá. Pokud by se spojení realizovalo pomocí vlastní přenosové linky pro každý systém zvlášť, množství dat by způsobovalo přeslechy a jiné chyby s tímto zapojením spojené. Dalším problémem je váha svazků, která jde proti trendu snažit se auto odlehčovat pro snížení emisí a také ušetřit náklady na výrobu. V neposlední řadě by nastal problém se v kabelech vyznat a rozumět každému protokolu, který by se v autě nacházel. Z těchto důvodů byla vyvinuta sériová sběrnice CAN.

Postupem času se CAN začal ve velkém nasazovat nejen v autech, ale i v zabezpečovací technice, komunikaci v průmyslových oblastech a dalších odvětvích. Důvodem je jednoduchost, spolehlivost, vysoká přenosová rychlost a snadná rozšiřitelnost. Díky svým vlastnostem a podpoře ze strany výrobců hardwaru se tento systém rozšířil natolik, že se stal v roce 1993 mezinárodní normou ISO 11898. Později došlo ke standardizaci vyšších vrstev protokolu, jako jsou CANopen, CAN Kingdom ad.

V následující části se věnuji principu činnosti sběrnice CAN.

1.1 ISO/OSI model

Sběrnice CAN z ISO/OSI referenčního modelu využívá pouze tři vrstvy. A to konkrétně vrstvu první – fyzickou, dále druhou – linkovou a poslední, sedmou – vrstvu aplikační, která je použita pouze, pokud se jedná o CAN s vyšší protokolovou vrstvou. Příkladem může být později v této práci řešený MicroCANopen Plus. Na *Obrázku 1* je ISO/OSI referenční model.

Obrázek 1. ISO/OSI referenční model

Linková vrstva je rozdělená na LLC (Logic Link Control) a MAC (Medium Access Control). LLC se stará o řízení datových spojů. To znamená, že podává hlášení o přetížení ve formě Overload Notifications. Mimo to se stará o filtraci přijatých zpráv pomocí takzvaných akceptačních filtrů. LLC je také zodpovědná za přenos dat ze zdroje do cíle. MAC se stará o kódování dat, řízení přístupu na sběrnici, detekci chyb, vkládání doplňkových bitů do zpráv (bit stuffing) a potvrzování o správném přijetí dat.

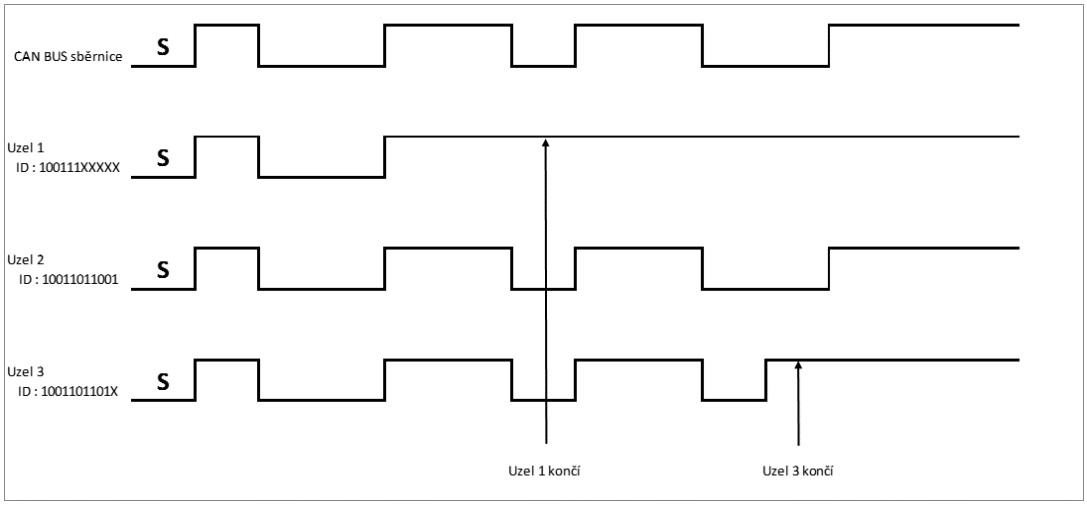
Fyzická vrstva má za úkol bitové kódovaní a dekódovaní, časování a synchronizaci. CAN nemá specifikované úrovně signálů. [2] Pro realizaci fyzického média se používá více norem. Nejčastěji se jedná o ISO 11898, která pojednává o diferenciální sběrnici. Dále se může použít norma ISO 11898-2 pro high speed, ISO 11898-3 pro low speed nebo třeba SAE J2411 pro single wire. Konektor také není pevně definován. Ale časem se některé konektory staly častěji používané. Jde například o konektor D-sub. Hlavním požadavkem

fyzické vrstvy je, aby médium bylo schopno přenášet dominantní nuly a recesivní jedničky [1].

Aplikační vrstva není samotným CAN standardem definována. Používá se až pro CAN s vyšší protokolovou vrstvou. Ta bude probrána v kapitole 2.

1.2 Přístup na sběrnici a řešení kolizí

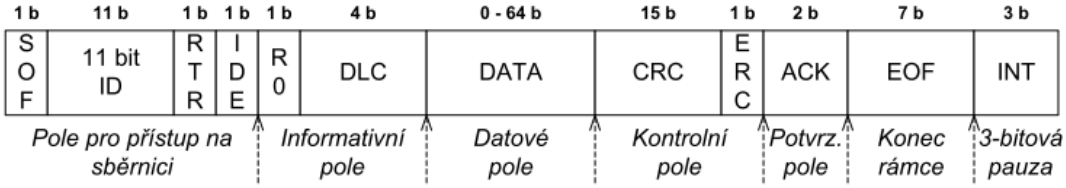
Jednotlivé uzly jsou na sběrnici připojeny principem otevřeného kolektoru. Pokud na sběrnici není připojen žádný uzel, je sběrnice v klidovém stavu. To znamená, že je na sběrnici recesivní stav (logická 1). V takovém případě se může libovolný uzel na sběrnici připojit a začít vysílat data. Může ale nastat situace, kdy by se na sběrnici chtěly připojit dva a více uzlů zároveň. V takovém případě za pomoci bitové arbitráže zůstane vysílat jen jeden uzel. Bitová arbitráž funguje pomocí identifikátorů, kdy každý uzel má svůj jedinečný identifikátor. Jakmile ho na sběrnici začne vysílat, zároveň se na ní monitoruje stav. Pokud by konkrétní uzel poslal recesivní bit, ale na sběrnici by byl bit dominantní, tak se odpojí. Vždy dostane přednost uzel s nejnižší hodnotou identifikátoru. Tento způsob je velmi podobný CSMA/CD (Carrier Sense Multiple Access with Collision Detection). Rozdíl je v okamžiku, kdy se uzel může znovu pokusit o přístup na sběrnici. V klasickém CSMA/CD, který je použit například u Ethernetu, by nastala Jamming sekvence a uzel by se mohl pokusit připojit znovu v náhodném čase. U CAN tomu tak není. Ten po neúspěchu monitoruje stav na sběrnici. V momentě, kdy je sběrnice volná, se znovu pokusí o vyslání dat. Pokud by došlo k vícenásobnému přístupu, tak se provede arbitráž. Z tohoto pohledu se dá říci, že u CAN nenastává kolize. Průběh arbitráže je na *Obrázku 2* [1].



Obrázek 2. Arbitráž CAN

1.3 Komunikační protokol

Komunikační protokol CAN je dvojího typu. Jedna možnost je standardní formát CAN 2.0A o délce 11 bitů. Druhá možnost je CAN 2.0B o délce identifikátoru 29 bitů. Standardní formát CAN 2.0A je na *Obrázku 3.*



Obrázek 3. Standardní formát rámce CAN 2.0A

SOF (Start Of Frame) je takzvaný start bit.

ID (Identifier): Identifikátor zprávy o délce 11 bitů.

RTR (Remote Transmission Request): Jeden bit, který oznamuje, jestli zpráva obsahuje data, nebo naopak o data žádá.

IDE (IDentifier Extension): Jeden bit, který určuje, jestli se jedná o standardní, nebo rozšířený rámec.

R0 (Reserve): Jeden bit rezervy pro budoucí využití.

DLC (Date Lenght): Čtyři bity nesoucí informaci o velkosti počtu bajtů dat. Může být přenášeno 0–8 bajtů.

Data Field: Datové pole o možné délce až 8 bajtů.

CRC (Cyclick Redundance Code): 15 bitů cyklického redundantního kódu pro kontrolu správnosti přenosu zprávy.

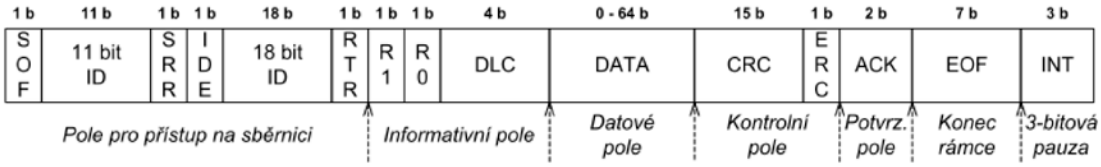
ERC (End cRC): Jeden bit oznamující konec CRC.

ACK (ACKnowlege field): Jedná se o dva bity určené pro přijímače. V tomto časovém úseku přijímače potvrzují správnost přijatých dat tím, že změní recesivní stav sběrnice na dominantní. Druhý bit odděluje potvrzovací bit a mění sběrnici zpět do recesivního stavu.

EOF (End Of Frame): Pole, ve kterém se posílají pouze recesivní bity. Oznamuje se tím ukončení rámce a přijímače dostávají možnost nahlásit chyby.

INT (INTermission): Nutná minimální prodleva před odesláním nových dat. Přijímače mají čas na zpracování přijatých dat.

Komunikační protokol CAN 2.0B se od CAN 2.0A liší jen trochu. Na *Obrázku 4* je rámec zobrazen. V tomto rámci je přidáno druhé pole **ID** o velikosti 18 bitů. Pole **SRR** je v recesivním stavu, protože standardní formát CAN 2.0A má vyšší prioritu než rámec CAN 2.0B. Posledním dalším polem je **R1**. Jedná se opět o jednobitovou rezervu [2].



Obrázek 4. formát rámce CAN 2.0B

1.4 Zabezpečení dat a detekce chyb

Sběrnice CAN disponuje sadou bezpečnostních mechanismů. Jedná se o monitoring, CRC kód, vkládání bitu, kontrolu zpráv a potvrzování přijímaných zpráv.

Monitoring byl probrán v kapitole 1.3. Jedná se o problematiku vícenásobného přístupu na sběrnici.

CRC kód (Cyclic Redundancy Check) je pole ve vysílací zprávě dlouhé 15 bitů. Viz např. *Obrázek 3*. Generujícím polynomem je polynom $x^{15} + x^{10} + x^8 + x^7 + x^4 + x^3 + x + 1$. Libovolný uzel může chybu kontrolního součtu CRC nahlásit.

Doplnění bitu neboli bit stuffing nastane, pokud by byla data po dobu pěti bitů stejné úrovně. V takovém případě se musí vložit bit opačné úrovně, jinak by nebyla dodržena synchronizace jednotlivých uzlů. Při dekódování zprávy se po pěti bitech stejné úrovně naopak následující bit vyjme. Příklad bit stuffingu je na *obrázku Obrázku 5*.

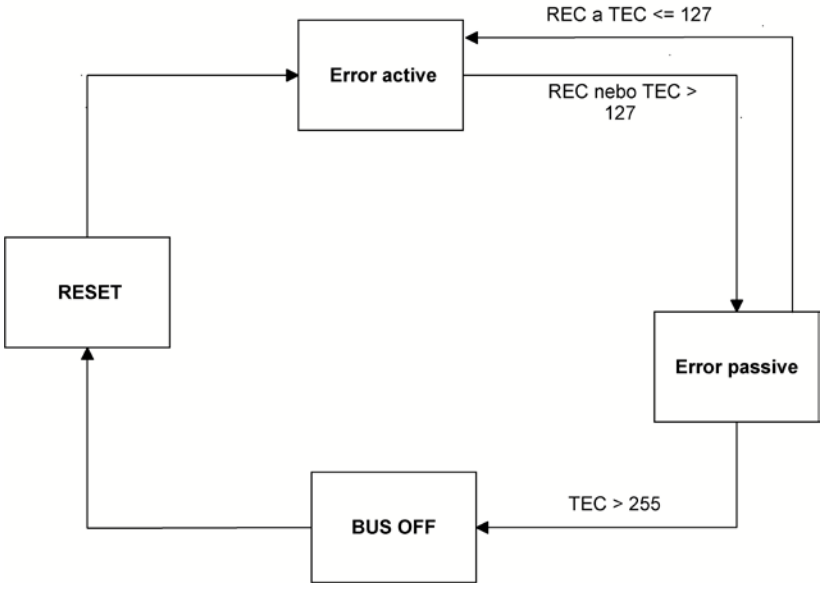
Vysílaná data	10100000 11011111
Vysílaná data + bit stuffing	10100000 **1** 11011111 **0**

Obrázek 5. Bit stuffing

Kontrola zprávy (message frame check) funguje na základě kontroly formátu zprávy, který je daný konkrétní specifikací. Pokud by se přišlo v rámci zprávy na chybu, je záhy vygenerována chyba rámce.

Potvrzení přijetí zprávy (acknowledge) je pole ve vysílané zprávě. Pokud uzel přijme zprávu, musí ji potvrdit v poli ACK tím, že hodnotu změní do dominantního stavu. Tuto změnu provedou i uzly, které zprávu nepřijímají na základě filtrů [2].

Každý uzel má také dva čítače chyb: čítač chyb při vysílání (Transmit error count) a čítač chyb při přijmu (Receive error count). Při každé chybě se do čítače přičte 8 a při správném příjmu nebo odvysílání zprávy odečte 1. Na základě chyb potom uzel přechází mezi třemi stavy: aktivní (Error Active), pasivní (Error Passive) a odpojené (Bus-off). Aktivní sběrnice se podílejí na komunikaci, a pokud detekují jakoukoli chybu, začnou na sběrnici vysílat Active Error Flag. Ten je tvořen šesti po sobě jdoucími dominantními bity, takže naruší bit stuffing. V aktivním stavu zůstává, dokud TEC nebo REC počítadlo nepřesáhne hodnotu 127 chyb. Pasivní stanice se také podílí na komunikaci, ale z pohledu hlášení chyb posílá na sběrnici pouze passive Error Flag. Ten se skládá z šesti po sobě jdoucích recesivních bitů, takže nedojde k rozhození komunikace. Do pasivního stavu se dostane, pokud je na počítadle TEC nebo REC více než 127 chyb. Odpojená sběrnice má odpojený budič, a tím pádem se nepodílí na žádné komunikaci po sběrnici. Do tohoto stavu přechází, pokud je na počítadle TEC více než 255 chyb. Z tohoto stavu se oproti stavu pasivnímu uzel sám nedostane. Musí nastat reset stanice. Přechod mezi stavy je vidět na *Obrázku 6* [1] [2].



Obrázek 6. Přechod mezi stavy uzlu

1.5 Typy rámců

Sběrnice CAN může použít čtyři druhy rámců. Jsou to rámce datové (Data frame), žádost o zprávu (Remote frame), chybový rámec (Error frame) a rámec o přeplnění (Overload frame).

Datový rámec je vysvětlen v samostatné kapitole 1.3. Jedná se o rámec, který posílá data.

Rámec s žádostí o data slouží k vyžádání si dat od jiné stanice. Má skoro stejnou strukturu jako rámec s daty. Ten je vidět na *Obrázku 3*. Pole RTR je v tomto případě v recesivním stavu oproti datové zprávě, která má tento bit v dominantním stavu. Datové pole zde neexistuje. V poli DLC je nula. V poli identifikátoru je pak identifikátor uzlu, od kterého jsou data žádána. Pokud by nastala situace, kdy by bylo žádáno o data a uzel by pod stejným identifikátorem zároveň data posílal, tak by v arbitráži došlo až na RTR. Zde by datová zpráva měla dominantní stav, a tím pádem by zpráva se žádostí ze sběrnice odstoupila.

Chybový rámec slouží k signalizaci chyb. Jakmile nějaký uzel zjistí chybu například v CRC, v porušení bit stuffingu nebo třeba chybu v rámci, odešle chybovou zprávu. Chybový rámec se skládá ze dvou částí. První z nich je ERROR FLAG a druhá část se nazývá ERROR DELIMITER. Error flag slouží k ohlášení chyb a error delimiter jako oddělovač chybového rámce. Rozlišujeme dva druhy chybových zpráv podle toho, jestli jsou poslány z aktivního, nebo pasivního uzlu (Error Active, Error Passive). Pokud je zpráva z aktivního uzlu, stanice posílá error active flag. Ten je složen z šesti po sobě jdoucích dominantních bitů. Tím se poruší bit stuffing a ostatní stanice také začnou vysílat své chybové rámce. Tím na CAN vznikne superpozicí sekvence dominantních úrovní o možné délce šest až dvanáct bitů. Následně stanice pošle osm recesivních bitů, takzvaných error delimiter, a monitoruje sběrnici, dokud recesivní bit nezachytí. Tím končí chybová zpráva. Pokud by nedošlo k zachycení recesivní úrovně, protože vlivem superpozice se tam nemusí projevit, tak se poslání error delimiter opakuje. Odesílatel zprávy s daty na zprávu o chybě reaguje znovu odesláním dat. Při novém odeslání dochází znovu k arbitráži a může se stát, že opravná zpráva bude poslána až později. Pokud by chybovou zprávu poslala pasivní stanice, vyšle passive error flag. Ten je složen z šesti po sobě jdoucích recesivních bitů. Tím opět dojde k porušení bit stuffingu, ostatní stanice se dozví, že nastala chyba, a pošlou active error flag. Passive error flag je ukončen, pokud zachytí šest po sobě jdoucích

bitů shodné úrovně. Ukončení rámce pak nastává stejně jako u aktivní stanice vysláním osmi po sobě jdoucích bitů recesivní úrovně.

Posledním typem rámce je rámec o přeplnění. Ten se skládá z pole OVERLOAD FLAG a OVERLOAD DELIMITER. Pro vyvolání overload flag vedou dvě podmínky. Může se jednat o vnitřní podmínku, která vyžaduje zpoždění další datové zprávy, případně remote zprávy. Druhým důvodem je detekce dominantních bitů v prvním a druhém bitu pole INT (intermission) nebo v osmém bitu error delimiter nebo overload delimiter. Overload flag se stejně jako active error flag skládá z šesti dominantních bitů. Zasláním overload flag dojde ke zničení struktury INT pole a ostatní stanice začnou také posílat overload flag. Pokud by ale došlo k detekci dominantního bitu až na třetí pozici INT pole, bylo by to chápáno jako start of frame a k ničemu by nedošlo. Po odeslání overload flag stanice vyšle overload delimiter. Ten je složen z osmi recesivních bitů. Jakmile ostatní stanice naslouchání zjistí přechod z dominantního do recesivního stavu, přestanou posílat overload flag a vyšlou dalších sedm recesivních bitů [2].

1.6 Synchronizace

Synchronizace je na sběrnici CAN odvozována od změny úrovně signálu, proto se musí do zpráv po pěti bitech stejné úrovně přidávat bit stuffing. Délka jednoho bitu se dělí na čtyři části. Synchronizační segment (SYN_SEG), segment zpoždění průchodu (PROP_SEG) a dvou fázových segmentů (PHASE_SEG1, PHASE_SEG2). Každý tento segment se dále dělí na časová kvanta t_q. Časové kvantum je odvozeno z kmitočtu oscilátoru. **SYN_SEG** se rovná jednomu časovému kvantu. V této oblasti se vždy očekává hrana signálu.

PROP_SEG je velký 1–8 časových kvant a slouží ke kompenzaci signálu na sběrnici vzhledem k rychlosti šíření signálu po sběrnici, zpoždění vysílacích a přijímacích obvodů a dalších zpoždění. **PHASE_SEG** jsou dlouhé 1–8 časových kvant a určují vzorkovací bod. Oba segmenty se mění podle toho, jestli hrana bitu přišla dříve, než ji přijímač očekával, nebo později. V prvním případě se segment 2 musí zkrátit. V případě druhém se segment 1 musí prodloužit. Na *Obrázku 7* je vidět rozdělení jednoho bitu [2].

SYNC_SEG	PROP_SEG	PHASE_SEG1	PHASE_SEG2

Obrázek 7. Rozdělení jednoho bitu

2 Protokol MicroCANopen Plus

Jedná se o standardizovanou sedmou vrstvu ISO/OSI vrstvy CAN, u které v základu stála společnost Philips Medical Systems. Protokol je otevřený a zdarma. Hlavním důvodem pro vznik vyšších vrstev CAN (7. vrstva ISO/OSI) byla vzájemná kompatibilita, zaměnitelnost a snadná integrace do stávajícího zařízení. Jako příklad vyšších vrstev lze uvést CANopen, MicroCANopen, MicroCANopen Plus, SAE J1939 a mnoho dalších. Základní specifikace aplikační vrstvy CAN je dána sdružením CiA (CAN in Automation) pod označením CAL (CAN Applications Layer) dokumenty CiA 201–207. CAL specifikuje pouze obecné komunikační procedury, ale už ne datové obsahy a specifické komunikační objekty. Definuje aplikačně nezávislé objektově orientované prostředí pro možnou implementaci distribuovaných systémů fungujících na sběrnici CAN. Obsahuje čtyři základní služby aplikační vrstvy, kterými jsou:

CMS (**CAN Based Message Specification**) – objekty a služby pro komunikaci,

DBT (Identifier Distributor) – přidělování CAN identifikátorů (COB-ID),

NMT (Network Management) – řízení sběrnice (inicializace, spuštění a zastavení uzlů atd., detekce poruch uzlu),

LMT (Layer Management) – změny parametrů vrstvy jako například NMT adresa uzlu, přenosová rychlost uzlu.

MicroCANopen Plus je zjednodušená verze plného protokolu CANopen podle CiA 301 a CiA 302. Protokol implementuje pouze základní vlastnosti, služby a funkce. Ale i přes svoji zjednodušenost musí být plně kompatibilní s CANopen, tím pádem i s možností rozšíření na plnohodnotný CANopen, kdy se na již používanou strukturu jen přidají další funkcionality. V následující *Tabulce 1* je vidět porovnání mezi CANopen, MicroCANopen a MicroCANopen Plus. V následujících kapitolách budou pak probrány některé služby MicroCANopen Plus. Například adresář deskriptorů, procesní datové objekty, servisní datové objekty, network management a emergency. V těchto kapitolách popisujících teoretické základy MicroCANopen bylo čerpáno ze zdrojů [1] a [3].

Tabulka 1. Porovnání CANopen, MicroCANopen a MicroCANopen Plus

	CANopen	**MicroCANopen**	**MicroCANopen Plus**
Přenosová rychlost [kb/s]	10, 20, 50, 125, 250, 500, 800, 1000	10, 20, 50, 125, 250, 500, 800, 1000	10, 20, 50, 125, 250, 500, 800, 1000
Max. počet uzlů v segmentu	127	127	127
Network management	NMT nebo autostart	NMT nebo autostart	NMT nebo autostart
Heartbeat	Ano	Ano	Ano
Node guarding	Ano	Ne	Ano
EMCY	Ano	Ne	Ano
Konfigurace uzlů	Ano	Ne, dáno předem	Ano
Položky v OD	Volitelná velikost a typ	Datové typy max. do 32 bit	Datové typy max. do 32 bit
Počet PDO	1024	8	1024
Přístup k PDO přes OD	Ano	Ne	Ano
Konfigurace a mapování PDO v OD	Ano	Ne	Ano

2.1 Adresář deskriptorů (OD)

Adresář deskriptorů, nebo také slovník objektů, je vyhledávací tabulka adresující veškeré proměnné uzlu, které jsou přístupné ze sítě. Jádrem každého CANopen uzlu je vždy vlastní slovník objektů. Do slovníku může být zapisováno či z něho čteno pomocí SDO služeb, které budou probrány později v kapitole 2.3. Slovník obsahuje popis konfigurace konkrétního CANopen uzlu. To znamená, že nese informace o aplikačních, stavových, komunikačních a konfiguračních parametrech. Pro přístup se používá SDO služby pomocí takzvaného multiplexoru. Každý objekt má svůj jedinečný index, který je reprezentován jako 16bitové číslo. To znamená, že maximální možný počet záznamů ve slovníku je 65 535. Každý objekt obsahuje ještě alespoň jeden takzvaný podzáznam adresovaný pomocí subindexů o maximální velikosti osmi bitů. Dohromady má tedy celý

multiplexor délku 24 bitů (16 + 8). V SDO zprávách je multiplexor obsažen v poli m. Položky ve slovnících jsou rozděleny do osmi částí podle *Tabulky 2*.

Tabulka 2. Rozdělení slovníku poruch

Index	**Popis**
0x0000	Rezerva
0x0001–0x025F	Definice datových typů
0x0260–0x0FFF	Rezerva
0x1000–0x1FFF	Komunikační profil
0x2000–0x5FFF	Specifikace výrobce
0x6000–0x9FFF	Standardizovaný profil zařízení
0xA000–0xBFFF	Standardizovaný profil rozhraní
0xC000–0xFFFF	Rezerva

V rozsahu adres 0x0001–0x025F se nacházejí definice datových typů. Jedná se o statické datové typy, komplexní datové typy, výrobcem definované datové typy, profilem zařízení definované statické typy a profilem zařízení definované dynamické typy.

Komunikační profil na adrese v rozsahu 0x1000–0x1FFF je určen především k nastavení komunikačních parametrů, jako je například PDO. Dále obsahují povinné údaje zařízení, které nelze vynechat. Jednotlivé povinné údaje s příslušným indexem a subindexem jsou uvedeny v *Tabulce 3*.

Tabulka 3. Povinné údaje komunikačního profilu

Index	**Subindex**	**Popis**	**Význam**
0x1000	0x00	Device type	Typ zařízení
0x1001	0x00	Error Registr	Chybový registr
0x1017	0x00	HeartBeat time	Hodnota HB periody
0x1018	0x00	Number of Subindex	Počet subindexů
0x1018	0x01	Vendor ID	Kód výrobce
0x1018	0x02	Product ID	Číslo produktu
0x1018	0x03	Revision Number	Číslo verze
0x1018	0x04	Serial Number	Sériové číslo

Objekty na adrese 0x2000–0x5FFF jsou otevřené pro specifikaci výrobcem. Mohou obsahovat data a konfigurace aplikace, které nejsou obsažené CANopen standardem.

Na adrese 0x6000–0x9FFF se nachází standardizovaný profil zařízení, který obsahuje proměnné profilu zařízení, například podle profilu DS-401, který specifikuje CANopen interface pro moduly s analogovým a digitálním vstupem a výstupem.

2.2 Procesní datové objekty (PDO)

Procesní datové objekty jsou dvojího druhu. Jeden pro odesílání, takzvaný TPDO, druhý pro příjem, takzvaný RPDO. Přestože SDO služby jsou komfortní prostředek pro přenos dat a monitorování uzlu, mají jednu nevýhodu. Tou nevýhodou je vysoká časová náročnost. Ta je neslučitelná s možností dělat aplikace s přenosem v reálném čase. To je důvod, proč se využívají procesní datové objekty. Časově kritické aplikační proměnné jsou totiž přímo mapovány do těchto objektů.

Maximální možná délka odesílaných dat může být až 8 bajtů. Zpráva obsahuje COB-ID, které je předdefinované podle *Tabulky 4*. COB-ID je složeno z Code + Node-ID. *Obrázek 8* pak zobrazuje strukturu TPDO zprávy. Přijímací objekty jsou stejné jako objekty vysílací. Rozdíl je jen v COB-ID, které je možno vidět v *Tabulce 5*. Čísla uzlů Node-ID se mohou pohybovat v rozsahu 1–127 pro TPDO i RPDO. Takže například vysílací objekt TPDO2 bude mít COB-ID 0x282. Naopak přijímací objekt RPDO2 bude mít COB-ID 0x302.

Tabulka 4. COB-ID TPDO

COB-ID	**TPDOx**
0x180+ Node-ID	TPDO1
0x280+ Node-ID	TPDO2
0x380+ Node-ID	TPDO3
0x480+ Node-ID	TPDO4

COB ID	**Počet bajtů**	**Data**
Viz tab.4	Max. 8	0-8 bajtů

Obrázek 8. TPDO zpráva

Tabulka 5. COB-ID RPDO

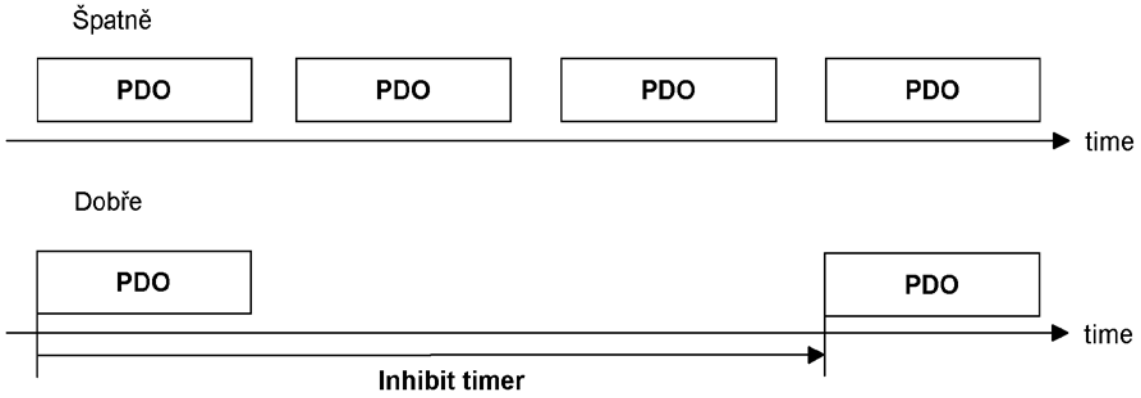
COB-ID	**RPDOx**
0x200+ Node-ID	RPDO1
0x300+ Node-ID	RPDO2
0x400+ Node-ID	RPDO3
0x500+ Node-ID	RPDO4

TPDO zprávy lze posílat pomocí čtyř hlavních podporovaných spouštěcích metod:

- Event driven (COS – Change of state),
- Time driving,
- Individual polling,
- Synchronized, group polling.

První metoda, Event driven, je založená na změnách v datech (změnou je událost – event). To znamená, že pokud nastane jakákoli změna v datech, případně konkrétní změna, dosažení určitého limitu nebo třeba minimálního rozdílu, dojde k přednosu dat pomocí TPDO zprávy. Pokud například sledujeme sadu digitálních vstupů a událost je nastavená na jakoukoli změnu, k události nedojde, dokud se na vstupech nic nezmění. Tato metoda má ale jeden problém. Pokud by se data měnila neustále, byla by celá dostupná šířka pásma vysílání zabrána touto událostí a nestíhalo by se dělat nic jiného. Proto je pro každý TPDO objekt definován takzvaný inhibit timer. V tomto čase je zakázáno spouštění dalšího TPDO objektu a je znovu možné až po uplynutí této doby. Konkrétní doba je závislá na povaze aplikace. Tímto opatřením však může docházet ke ztrátě dat, které budou chodit v čase inhibit timer. Na *Obrázku 9* je ukázka popsané problematiky. Dalším problémem může být analogový výstup nebo vstup. Pokud například snímáme teplotu, nepotřebujeme získávat každou změnu. Stačí nám třeba jen výrazné změny. Tento problém je řešen specifikací CiADS401, která podporuje konfigurovatelnou delta detekci. Pak systém rozpozná změnu pouze v případě, kdy dojde ke změně o hodnotu delta definovanou uživatelem, případně na minimum nebo maximum. Protože ale TPDO může obsahovat mnoho proměnných procesů, a dokonce i směs analogových a digitálních hodnot, je detekce změn velmi složitá, a to hlavně v případě, kdy jeden TPDO obsahuje více

takových odlišných vstupních hodnot. Proto se doporučuje nemíchat analogové a digitální hodnoty pouze v TPDO, případně upustit od detekčního mechanismu CiADS401.



Obrázek 9. Inhibit timer

Další metodou je Time driven. Tato metoda má pevně nastaven čas (event time), po jehož uplynutí se spustí odeslání TPDO. Pokud je například zadán čas 10 ms, dojde každých 10 ms k odeslání TPDO. Čas se zadává s rozlišením milisekund a může trvat různě dlouho. Konkrétní doba záleží na konkrétní aplikaci. Každý uzel má svůj event time a ve výchozím stavu nejsou nijak synchronizovány. Tato metoda je oproti Event driven jednoduchá, ale bohužel posílá data, i když nedojde k jejich změně.

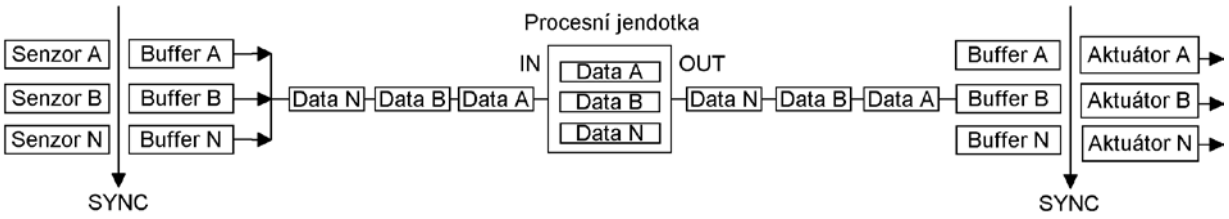
Třetí metoda, Individual polling, není doporučována. Individual polling je implementována pomocí funkce Remote Request a zde dochází k problému. Všichni výrobci nedodržují jednotný standard a může docházet k nekompatibilitě. Další nevýhodou jsou velké režijní náklady způsobené vzdáleným dotazem na TPDO. Proto se doporučuje využít poslední metody.

Poslední metoda, Synchronized or group polling, je vhodná, pokud chceme synchronizovat aplikace orientované na pohyb nebo aplikace, kde je potřeba mít zároveň v jednom okamžiku více informací, aby nedocházelo k jitteru. Synchronizovaná komunikace je v CANopen implementována pomocí signálu SYNC. Zpráva SYNC je na *Obrázku 10*. Tento signál je specifická zpráva bez dat a slouží pouze pro synchronizaci. Princip synchronizace dat je vidět na *Obrázku 11*. Senzory neustále čtou vstupní data a ukládají je do vyrovnávací paměti, takže je uložena vždy kopie nejnovější informace. Po příchodu signálu SYNC přestanou senzory aktualizovat vyrovnávací paměť a data se začnou sériově vysílat. Tak se do procesní jednotky dostanou data z jednoho okamžiku ze všech senzorů. Z procesní jednotky jsou pak data přenesena do vstupních bufferů akčních

členů, ale ne dál na výstup. Až když přijde druhý signál SYNC, data se paralelně uplatní na výstupu.

COB-ID	**Počet bajtů**	**Data**
0x80	0	–

Obrázek 10. Zpráva SYNC



Obrázek 11. Princip SYNC

2.3 Servisní datové objekty (SDO)

Servisní datové objekty SDO umožňují přímý komunikační kanál mezi master stanicí a proměnnými, které jsou mapované v adresáři deskriptorů OD konkrétního uzlu. Komunikace probíhá systémem klient – server v režimu point-to-point. Vlastník slovníku je server a žadatel o data ze slovníku klient. SDO služba může být trojího druhu. SDO download tehdy, jestliže klient nahrává data serveru; SDO upload, pokud klient žádá data ze serveru; a Abort SDO Transfer pro možnost přerušení přenosu. Podle typu použitého protokolu dále rozlišujeme komunikace SDO upload a SDO download služeb na jednorázové (expedited), segmentové (segmented) a blokové (block).

2.3.1 Jednorázový protokol (expedited)

Jednorázový přenos je pro MicroCANopen Plus volen jako základní. Při použití tohoto protokolu nesmí žádná proměnná v OD překročit velikost čtyři bajty. Pokud je potřeba přenést více než 4 bajty dat, je potřeba použít segmentový protokol. Protokol je postaven na jednoduchém principu, který se skládá pouze z jedné žádosti SDO a jedné odpovědi SDO jak pro SDO download, tak pro SDO upload. Princip je popsán u segmentového protokolu v další kapitole.

2.3.2 Segmentový protokol (segmented)

Druhý protokol umožňuje posílání celých segmentů o maximální možné velikosti dat až sedm bajtů. V tomto protokolu už nestačí pouze jedna žádost a odpověď SDO. Celý proces zahrnuje minimálně čtyři části pro SDO download i SDO upload.

SDO Download

- Initiate SDO Download – Request (proběhne 1x)
- Initiate SDO Download – Response (proběhne 1x)
- Download SDO Segment – Request (proběhne, kolikrát je potřeba)
- Download SDO Segment – Response (proběhne, kolikrát je potřeba)

SDO Upload

- Initiate SDO Upload – Request (proběhne 1x)
- Initiate SDO Upload – Response (proběhne 1x)
- Upload SDO Segment – Request (proběhne, kolikrát je potřeba)
- Upload SDO Segment – Response (proběhne, kolikrát je potřeba)

SDO downloads

Klient, obvykle uzel, který se pokouší konfigurovat slave uzel, odešle nejprve žádost Initiate SDO download – Request pomocí identifikátoru COB-ID 0x600 + Node-ID uzlu, kterému je adresována. Tím je požádáno o zápis do jeho OD. Server poté zprávou Initiate SDO download – Response klientovi odpoví. Tím oznamuje, že žádost byla úspěšně zpracována. COB-ID zprávy je 0x580 + Node-ID klienta. Zpráva Request obsahuje pole **ccs** (Client Command Specifier), které udává typ služby. Pro Request je to jednička. Dále je tu **x**, které slouží jako rezerva. Bit **n** nese informaci o počtu bitů, které neobsahují data, pokud bit e a s je roven jedné. Pole **e** se rovná jedné, pokud jde o protokol rychlého přenosu. Je-li udána velikost dat, je bit **s** roven jedné. Pole **m** obsahuje informace o multiplexeru, indexu a subindexu z OD. Poslední pole o velikosti čtyř bajtů je potom určeno jako datové pole, pokud se jedná o jednorázový protokol podle kapitoly 2.3.1. V opačném případě zůstává jen jako rezerva. Na *Obrázku 12* je vidět struktura popsané zprávy.

0								1	2	3	4	5	6	7
7	6	5	4	3	2	1	0	m			Data rychlého přenosu nebo rezerva			
ccs = 1			x	n		e	s							

Obrázek 12. Initiate SDO Download – Request

Zpráva Response obsahuje pole **scs** (Server Command Specifier), které je obdobou pole ccs. Pro Response je vyplněno trojkou. Dále jsou tu pole **x** a **m**, které mají stejný význam jako ve zprávě Request. Poslední pole je pak jen rezerva. Na *Obrázku 13* je zpráva Initiate SDO Download – Response.

0								1	2	3	4	5	6	7
7	6	5	4	3	2	1	0	m			Rezerva			
scs = 3			x											

Obrázek 13. Initiate SDO Download – Response

Pokud v Initiate SDO Download bylo pole e rovno 0, mohou se přenášet data až o velikosti 7 bitů pomocí segmentového přenosu dat tak dlouho, jak bude potřeba pro přenos všech dat. Nejprve klient odešle zprávu Download SDO Segment – Request, která obsahuje pole **ccs** rovné nule. Bit **t**, který je s prvním poslaným segmentem roven nule, se s každým dalším poslaným segmentem neguje. Pole **n** značí počet bajtů, které neobsahují žádná data. Bit **c** je nastaven na jedničku, pokud se jedná o poslední segment dat. V opačném případě je nulový. Poslední pole o velikosti 7 bajtů obsahuje data, která putují od klienta k serveru. Na *Obrázku 14* je zpráva vidět.

0								1	2	3	4	5	6	7
7	6	5	4	3	2	1	0	Data						
ccs = 0			t	n			c							

Obrázek 14. Download SDO Segment – Request

Po příjmu dat v serveru odešle server odpověď, která klientovi oznamuje úspěšné zpracování přijatých dat. Zpráva obsahuje pole scs rovné 1, bit **t** v první odpovědi

nastavený na nulu a s každou další odpovědí negovaný, dále pole **x** jako rezervu a poslední pole o velikosti jednoho bajtu také jako rezervu. Zpráva je na *Obrázku 15.*

0								1	2	3	4	5	6	7
7	6	5	4	3	2	1	0	Rezerva						
scs =1			t	x										

Obrázek 15. Download SDO Segment – Response

SDO Upload

SDO upload funguje prakticky stejným způsobem jako SDO download. Oproti SDO download ale klient žádá o data ze serveru. Komunikace začíná zprávou klienta Initiate SDO Upload – Request s žádostí o data. COB-ID zprávy je 0x600 + Node-ID serveru, z něhož je o data žádáno. Zpráva obsahuje pole ccs = 2. Zbytek zprávy není třeba komentovat. Na *Obrázku 16* je zpráva znázorněna.

0								1	2	3	4	5	6	7
7	6	5	4	3	2	1	0	m			rezerva			
ccs = 2			x											

Obrázek 16. Initiate SDO Upload – Request

Server klientovi oznamuje zprávou Initiate SDO Upload – Response přijatý požadavek na čtení ze svého OD. COB-ID zprávy je 0x580 + Node-ID. Zpráva podle *Obrázku 17* obsahuje pole scs = 2 a další již dříve vysvětlená pole. Pokud by se jednalo o protokol jednorázového přenosu, končí komunikace zde a v bajtech 4 až 7 by byla poslána data klientovi.

0								1	2	3	4	5	6	7
7	6	5	4	3	2	1	0	m			Data rychlého protokolu nebo rezerva			
scs = 2			x	n		e	s							

Obrázek 17. Initiate SDO Upload – Response

V další části klient posílá zprávu Download SDO Upload – Request s ccs = 3 podle *Obrázku 18*. Obratem dostane odpověď ze serveru ve zprávě Download SDO Upload s daty, která žádal. Zpráva je vidět na *Obrázku 19*.

0								1	2	3	4	5	6	7
7	6	5	4	3	2	1	0	Rezerva						
ccs = 3			t	x										

Obrázek 18. Upload SDO Segment – Request

0								1	2	3	4	5	6	7
7	6	5	4	3	2	1	0	Data						
scs = 0			t	n			c							

Obrázek 19. Upload SDO Segment – Response

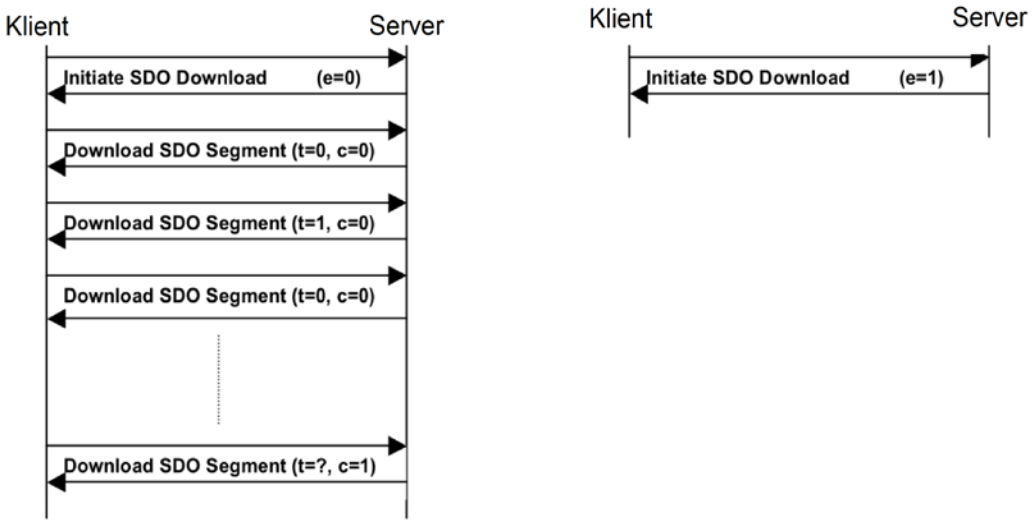
Abort SDO Transfer

Poslední službou je Abort SDO transfer. Ta serveru nebo klientovi umožňuje kdykoli z jakéhokoli důvodu ukončit přenos. Týká se jak segmentového, tak jednorázového protokolu. Důvodem přerušení přenosu je velmi často nenalezení žádaného záznamu v OD nebo nesprávná délka dat – například je zapisována čtyřbajtová hodnota do dvoubajtové položky v OD. Zpráva pro oba směry přenosu mezi klientem a serverem je stejná. Pole **cs** má hodnotu 4. Pole **x** opět značí rezervu a poslední čtyři bajty obsahují kód konkrétní chyby, která vyvolala přerušení přenosu. Seznam error kódu je v [1] na straně 501 v tabulce H. 1 SDO Abort Codes. Na *Obrázku 20* je Abort SDO Transfer zpráva vidět.

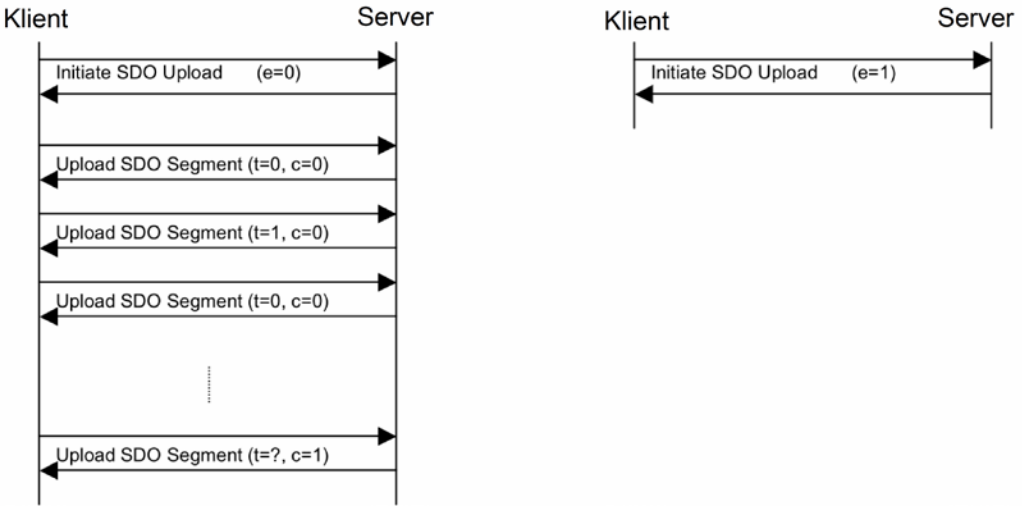
0								1	2	3	4	5	6	7
7	6	5	4	3	2	1	0	m			Error kód			
cs = 4			x											

Obrázek 20. Abort SDO Transfer

Celkový pohled na průběh komunikace SDO download je znázorněn na *Obrázku 21*, kde levý obrázek ukazuje segmentový protokol a obrázek vpravo průběh pro jednorázový přenos. Na *Obrázku 22* je stejným způsobem znázorněn SDO upload [3].



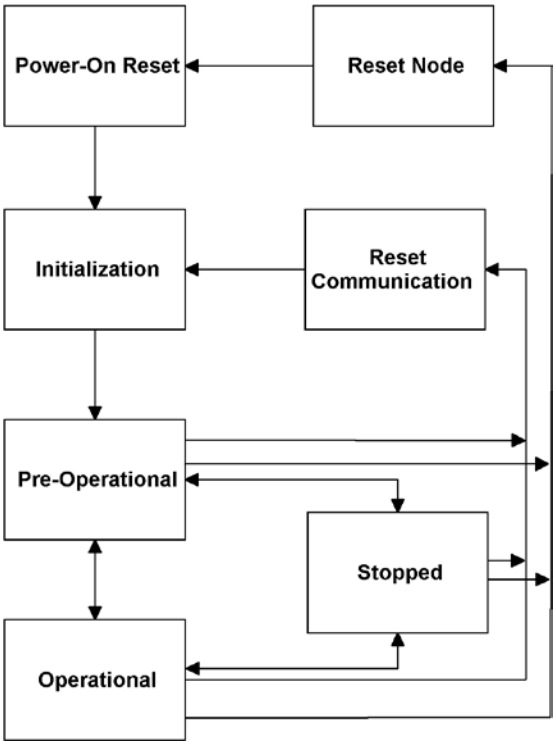
Obrázek 21. Celkový pohled na komunikaci SDO download



Obrázek 22. Celkový pohled na komunikaci SDO upload

2.4 Network management (NMT)

Každý slave uzel zapojený do komunikace musí mít implementovaný network management stavový automat, který slouží k řízení přechodů mezi jednotlivými stavy. Diagram možných přechodů mezi stavy je na *Obrázku 23.*



Obrázek 23. Network management

Po zapnutí napájení se uzel nejprve dostane do stavu Power-On Reset a automaticky přechází do stavu Initialization. V tomto stavu dochází ke kompletní inicializaci rozhraní a komunikace. Při dokončení inicializace se uzel pokusí poslat takzvanou bootup zprávu. Pokud dojde k úspěšnému přenosu této zprávy, uzel se dostává do stavu Pre-operational. V tomto stavu začíná posílat pravidelnou zprávu heartbeat, pomocí které signalizuje svoji bezchybnou činnost uzlu. Pokud network management nezaznamená v zadaném čase zprávu heartbeat, může se domnívat, že byl uzel ztracen. V takovém případě může řídicí stanice podniknout příslušné opatření. K možným opatřeným řadí re-inicializace uzlu, případně restartování celého uzlu. Zpráva je posílána podle aplikace s přesností jedné milisekundy. Vedle heartbeat je ještě možnost posílání zprávy node guarding. Ta ale není doporučována, protože zabere dvakrát tak velké pásmo přenosu. To je dáno tím, že funguje na principu dotazu a odpovědi. Master stanice se zeptá příslušného uzlu a ten v maximální akceptovatelné době musí odpovědět. V Pre-operational módu nemůže uzel posílat ani přijímat žádné procesní objekty TPDO a RPDO, ale může zpracovávat servisní datové služby SDO a posílat chybové zprávy EMCY a již zmíněný heartbeat. Chybové zprávy EMCY budou podrobněji probrány v kapitole 2.5. Pro přechod do Operational módu je potřeba obdržet od master stanice zprávu Start Node. Jakmile se uzel dostane do stavu Operational, už plně funguje. To znamená, že již zpracovává jak služby SDO a EMCY, tak

i TPDO a RPDO zprávy. Kromě toho dále generuje heartbeat. Uzel může do stavu Operational přejít i alternativním způsobem. A to tak, že funguje v takzvaném autostart režimu. V tomto režimu může příslušný uzel ze stavu Pre-operational do stavu Operational přejít plně automaticky ihned po své inicializaci. Takováto stanice se obejde bez master stanice s tou podmínkou, že zná svoji konfiguraci a mapování jednotlivých procesních proměnných TPDO a RPDO. Přehled toho, co může uzel v jakém stavu dělat, je popsán v *Tabulce 6*.

Tabulka 6. Možnosti ve stavech uzlu CAN

	Initializing	**Pre-operational**	**Operational**	**Stopped**
Bootup	ANO			
SDO		ANO	ANO	
Emergency		ANO	ANO	
SYNC/TIME		ANO	ANO	
Heartbeat / node guarding		ANO	ANO	ANO
PDO			ANO	

Network management řídí uzel mezi třemi stavy – Operational, Pre-operational a Stopped. Pro řízení jsou použity takzvané NMT zprávy. COB-ID takové zprávy je 0x000. Datové pole je velké dva bajty. V prvním bajtu je příkaz, který informuje uzel, do kterého stavu má přejít. V druhém bajtu je informace o Node-ID příjemce, aby bylo možné určit, kterému uzlu je zpráva určena. Pokud je Node-ID nulové, je zpráva určena všem uzlům. *Tabulka 7* obsahuje seznam kódů network managementu, které jsou v bajtu 0, a na *Obrázku 24* je znázorněna NMT zpráva.

COB-ID	**Počet bajtů**	**Data**	
0x000	2	0	1
		Kód zprávy	Node-ID příjemce

Obrázek 24. NMT zpráva

Tabulka 7. Kódy NMT

Kód	**Funkce**
1	Přechod do Operational
2	Přechod do Stopped
128	Přechod do Pre-operational
129	Reset uzlu
130	Reset komunikace

V módech Stopped, Operational a Pre-operational slave stanice vysílají informaci heartbeat také ve zprávě dané struktury. Ta má COB-ID složené z kódu 0x700 + Node-ID uzlu, který zprávu posílá. Například pro uzel jedna bude COB-ID 0x701. Samotná zpráva je veliká jeden bajt a obsahuje 7bitovou zprávu o svém aktuálním stavu. Poslední bit je rezerva. Hodnoty zpráv pro jednotlivé stavy jsou vidět v *Tabulce 8*. *Obrázek 25* pak zobrazuje NMT heartbeat zprávu. Kapitola je napsána pomocí zdrojů [1] a [3].

COB-ID	**Počet bajtů**	**Data**							
0x700+ Node-ID	1	7	6	5	4	3	2	1	0
		r	Zpráva o stavu						

Obrázek 25. NMT heartbeat zpráva

Tabulka 8. Heartbeat kódy zpráv

Kód	**Funkce**
0	Bootup
4	Stopped
5	Operational
127	Pre-operational

2.5 Emergency (EMCY)

Každému uzlu slave je vždy přiřazena jedna emergency zpráva (EMCY). Jako identifikátor COB-ID se používá 0x80 + Node-ID uzlu. Takže například identifikátor pro uzel 5 bude 0x85. Zpráva se pak skládá z osmi datových bajtů, kde první dva bajty jsou použity pro error kód. Třetí bajt je použit pro kopii error registru. Posledních pět bajtů je pak využito pro specifické chyby dané přímo konkrétním výrobcem. Struktura zprávy je vidět na *Obrázku 26* [1].

<table>
<tr><th>COB-ID</th><th>Počet bajtů</th><th colspan="8">Data</th></tr>
<tr><td rowspan="2">0x80 + Node-ID</td><td rowspan="2">8</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td></tr>
<tr><td colspan="2">Error kód</td><td>Error registr</td><td colspan="5">Specifický kód výrobce</td></tr>
</table>

Obrázek 26. Struktura zprávy EMCY

Obvykle jsou mimořádné události hlášeny jen jednou. To znamená, že jednou nahlášená zpráva se považuje za stále aktuální. Pokud by došlo ke změně a chyba byla odstraněna, uzel odešle emergency zprávu s kódem chyby 0x0000 [1]. Seznam vybraných error kódů je v *Tabulce 9*. Nejsou tu však ani zdaleka uvedeny všechny. Kompletní seznam lze dohledat v [1].

Tabulka 9. Error kódy EMCY

Error kódy	Popis
0000h–00FFh	Žádná chyba, Error reset
2000h–20FFh	Proud
3000h–30FFh	Napětí
4000h–40FFh	Teplota
5000h–50FFh	Hardware zařízení
6000h–60FFh	Software zařízení
FF00h–FFFFh	Specifikace zařízení

3 Realizace protokolu MicroCANopen Plus

V této kapitole se věnuji samotné realizaci protokolového zásobníku MicroCANopen Plus. Napsaný program je určen pro slave uzel a má implementované čtyři vysílací objekty TPDO, označené TPDO1 až TPDO4. Dále obsahuje stejný počet přijímacích objektů, označených RPDO1 až RPDO4. Další částí jsou SDO služby, které umožňují zapsání hodnoty do adresáře deskriptorů, případně z něho vyčíst žádanou hodnotu. Program obsahuje i heartbeat pro hlášení stavu uzlu a související službu network management označovanou jako NMT. Samozřejmostí je implementace adresáře deskriptorů s povinnými položkami a dalšími položkami pro PDO mapping, nastavení parametrů PDO služeb, heartbeat ad. Všechny principy výše popsaných možností jsou vysvětleny podrobně v kapitole 2.

Služba TPDO slouží k posílání zpráv. V tomto programu je zvolena možnost odesílání na základě time driving. Čas, v němž se konkrétní zpráva odesílá, je stejně jako kupříkladu délka zprávy a Node-ID nastavitelný. Délku zprávy a periodu vysílání je možné nastavit nejen v include souboru, ale i pomocí SDO služeb. Nastavení těchto parametrů lze vykonat také dynamicky, protože se jedná o MicroCANopen Plus (v Object Dictionary 0x14xx a 0x18xx), a je zde tudíž implementovaná i možnost PDO mapping (v Object Dictionary 0x16xx a 0x1Axx). V rámci této diplomové práce je použita u vysílacího objektu TPDO1.

Služby RPDO slouží k příjmu zpráv, a to opět s možností nastavení délky přijaté zprávy jak pomocí include souboru, tak pomocí SDO služby.

SDO služby spolupracují se slovníkem deskriptorů, který lze ručně libovolně doplňovat o další položky. Pomocí SDO služeb lze v této práci do slovníku zapisovat a také z něho žádané hodnoty vyčíst a poslat stanici, která si o data žádá.

Heartbeat je ve výchozím stavu nastaven na 500 ms, přičemž je možné ho pomocí SDO služeb změnit. NMT protokol potom obsahuje všechny stavy od bootup až po Operational mód. Pomocí include souboru se nastavuje, jestli bude uzel fungovat ve standardním režimu přes Pre-operational mód, nebo pomocí takzvaného autostartu.

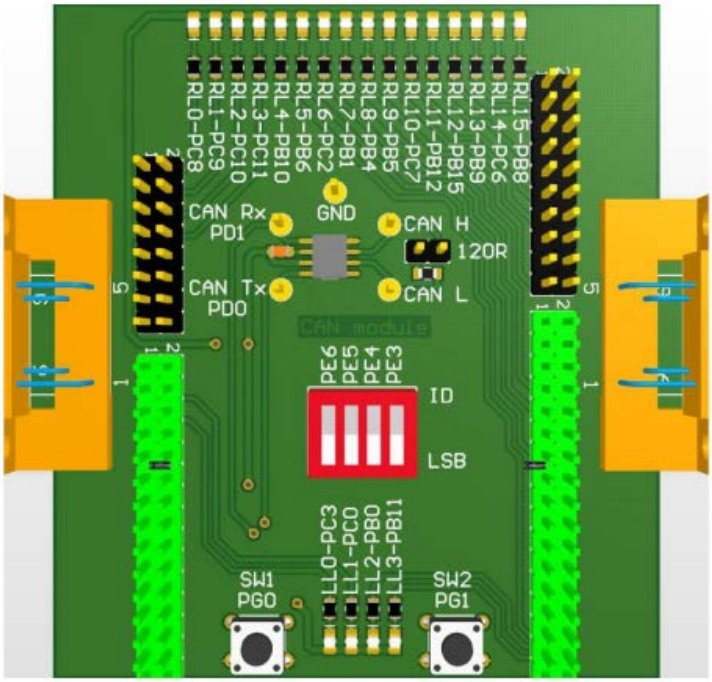
V následujících podkapitolách nejprve uvádím hardware, na kterém je diplomová práce realizována. Následně věnuji pozornost samotným zdrojovým souborům a funkcím v nich obsaženým. Jednotlivé zdrojové soubory a hlavičkové soubory lze dohledat na CD v příloze A podle následující Tabulky 10.

Tabulka 10. Přehled zdrojových a hlavičkových souborů ve složce

Zdrojový soubor, hlavičkový soubor	**složka**
include.h	MCO_plus\src\include.h
mcohw.c	MCO_plus\src\mcohw.c
MCO.c	MCO_plus\src\MCO.c
inicializace.c	MCO_plus\src\inicializace.c
main.c	MCO_plus\src\main.c

3.1 Použitý hardware

Jak už bylo zmíněno v úvodu, coby zástupce platformy STM32 byl použit vývojový kit Nucleo F429ZI, který obsahuje CAN periférii. Při programování bylo využito referenčního manuálu [6] a datového listu [7]. K tomuto kitu byl ještě připojen modul obsahující CAN budič s konektory a základní periférie, jako jsou tlačítka a LED diody pro účely testování a vizualizace. Kit byl i s dokumentací poskytnut a vytvořen Katedrou aplikované elektrotechniky. *Obrázek 27* ukazuje nakreslený modul s popisem tlačítek a dalších periférií. V příloze B na CD je schéma zapojení modulu. Jádrem modulu je budič SN65HVD230. Celé testování a kontrola funkčnosti bylo provedeno pomocí katedrou zapůjčeného převodníku CAN-USB od firmy IMFsoft [8]. Tento převodník umožnil propojení CAN modulu s počítačem přes USB port. Program USB-CAN adapter V4.15 nainstalovaný v počítači pak posloužil ke kompletní správě komunikace CAN. Lze tu odesílat a přijímat PDO, používat SDO služby, network management a mnoho dalšího.



Obrázek 27. Modul s budičem CAN

3.2 Popis programu

Celý program se skládá ze tří hlavních zdrojových souborů (.c), které obsahují jednotlivé funkce. Všechny funkce jsou popsány v dalších kapitolách podle toho, v jakém zdrojovém souboru se nacházejí. Výjimku tvoří pouze funkce *MCO_ProcessStack*, jíž je jako jádru protokolového zásobníku věnována samostatná kapitola. Vedle těchto hlavních zdrojových souborů se nacházejí i další, které slouží k nastavení jednotlivých periférií mikrokontroléru. Součástí programu je také hlavičkový soubor *include.h*, důležitý pro počáteční nastavení výchozích hodnot PDO objektů, heartbeat periody, node_ID, rychlosti komunikace a hodnot do přijímacích filtrů. To je zařízeno pomocí příslušných struktur. Jsou tu připojeny všechny ostatní hlavičkové soubory programu, takže v něm stačí do každého zdrojového souboru připojit jen tento hlavní hlavičkový soubor *include.h.* V hlavičkovém souboru jsou všechny struktury a výchozí hodnoty, které lze měnit, přehledně okomentovány a není potřeba provádět další popis. Další hlavičkové soubory obsažené v programu náleží vždy konkrétnímu zdrojovému souboru. Celý program je realizován v programovacím jazyce C a je kompletně na CD v příloze A. Na následujících řádkách je vidět struktura programu.

- main
  - void SysTick_Handler(void)
  - void CAN1_TX_IRQHandler(void)
  - void CAN1_RX0_IRQHandler(void)
  - void CAN1_RX1_IRQHandler(void)
  - void CAN1_SCE_IRQHandler(void)
  - int main (void)

- MCO.c
  - void MCO_Init (unsigned int Baudrate,unsigned char Node_ID,unsigned int Heartbeat)
  - void MCO_InitRPDO(unsigned char PDO_NR,unsigned int CAN_ID, unsigned char len)
  - void MCO_InitTPDO(unsigned char PDO_NR, unsigned int CAN_ID, unsigned int event_time, unsigned int inhibit_time,unsigned char len)
  - void MCO_ProcessStack(void)
  - void MCOUSER_ResetCommunication(void)
  - void SDO_Search(uint16_t indexx, uint8_t subindex)
  - void SDO_Download(uint16_t indexx, uint8_t subindex, uint32_t SDO_Download_data)
  - void MCOUSER_ResetApplication(void)

- mcohw.c.
  - unsigned char MCOHW_Init(unsigned int BaudRate)
  - unsigned char MCOHW_SetCANFilter(unsigned int CANID)
  - void MCOHW_PushMessage(int *p_TransmitBuf)
  - void MCOHW_PullMessage(int *p_ReceiveBuf)
  - void MCOHW_Response_SDO_Message(int *p_SDO_TransmitBuf_ini, int *p_SDO_TransmitBuf_Data)

3.2.1 Konfigurační, stavové a datové rozhraní

Tato kapitola se věnuje rozhraní, které se stará o data, konfiguraci a předávání informací o stavu protokolového zásobníku. Jedná se o sadu proměnných a struktur.

Konfigurační rozhraní

Jedná se o tři struktury: MCO_NIT, RPDO_Init a TPDO_INIT. Jednotlivé struktury jsou popsány tabulkami 11–13. Tabulka obsahuje název struktury, jména položek ve struktuře, popis položek, název proměnné s výchozím nastavením položky a výchozí hodnotu. Dále jsou tu dva #define soubory pro nastavení akceptačních filtrů. Jedná se o #define dCANID 0x180 a #define mask 0x780.

Tabulka 11. Struktura MCO_INIT

MCO_INIT			
Jméno položky	**Popis položky**	**Define proměnná**	**hodnota**
BaudRate	Nastavení baud rate uzlu	dBaudRate	125
Node_ID	Node-ID uzlu	dNode_ID	0x1
Heartbeat	Perioda heartbeat uzlu	dHeartbeat	500

Tabulka 12. Struktura RPDO_Init

RPDO_Init			
Jméno položky	**Popis položky**	**Define proměnná**	**hodnota**
PDO_NR	Číslo RPDO		
CAN_ID	Číslo CAN_ID		
len	Délka zprávy	dlen_RDPOx	8

x u dlen_RPDOx má hodnotu 1–4 podle čísla RPDO

Tabulka 13. Struktura TPDO_INIT

RPDO_Init			
Jméno položky	**Popis položky**	**Define proměnná**	**hodnota**
PDO_NR	Číslo RPDO		
CAN_ID	Číslo CAN_ID		
event_time	Perioda vysíláno TPDO	devent_time_TPDOx	211, 251, 311, 353
inhibit_time	Inhibit_time	dinhibit_time_TPDOx	0
len	Délka zprávy	dlen_TPDOx	4
last_time	Čas posledního odeslání	dlast_time	0

x opět značí konkrétní TPDO objekt

Stavové rozhraní

Zde jsou proměnné, které souží k předávání stavu různých částí programu. Jedná se o stavy schránek PDO objektů, network management a další. V *Tabulce 14* je seznam těchto proměnných se stručným popisem funkce. Podrobnější popis je v dalších kapitolách, věnujících se detailnímu popisu programu.

Tabulka 14. Stavové rozhraní

Název	Popis
signal_HB	Čas pro odeslání heartbeat
signal_TPDO1	Čas pro odeslání TPDO1
signal_TPDO2	Čas pro odeslání TPDO2
signal_TPDO3	Čas pro odeslání TPDO3
signal_TPDO4	Čas pro odeslání TPDO4
signal_prijem	Informace o příjmu rámce
signal_prijem_SDO	Informace o příjmu rámce SDO
signal_Error	Informace o chybovém stavu
start	Zapnutí systému
inicializace	Systém je v inicializačním stavu
NMT	Network management – stav uzlu

Datové rozhraní

Toto rozhraní slouží k posílání rámců. V následující tabulce je opět přehled jednotlivých proměnných, přes které se přenos realizuje.

Tabulka 15. Datové rozhraní

Název	Popis
ReceiveBuf	Buffer pro příjem
TransmitBuf	Buffer pro odesílání
SDO_TransmitBuf_Data	Buffer pro odesílání datové části SDO
SDO_TransmitBuf_ini	Buffer pro odesílání inicializační části SDO
SDO_ReceiveBuf_data	Buffer pro příjem datové části SDO
SDO_ReceiveBuf_ini	Buffer pro příjem inicializační části SDO

3.2.2 Zdrojový soubor main.c

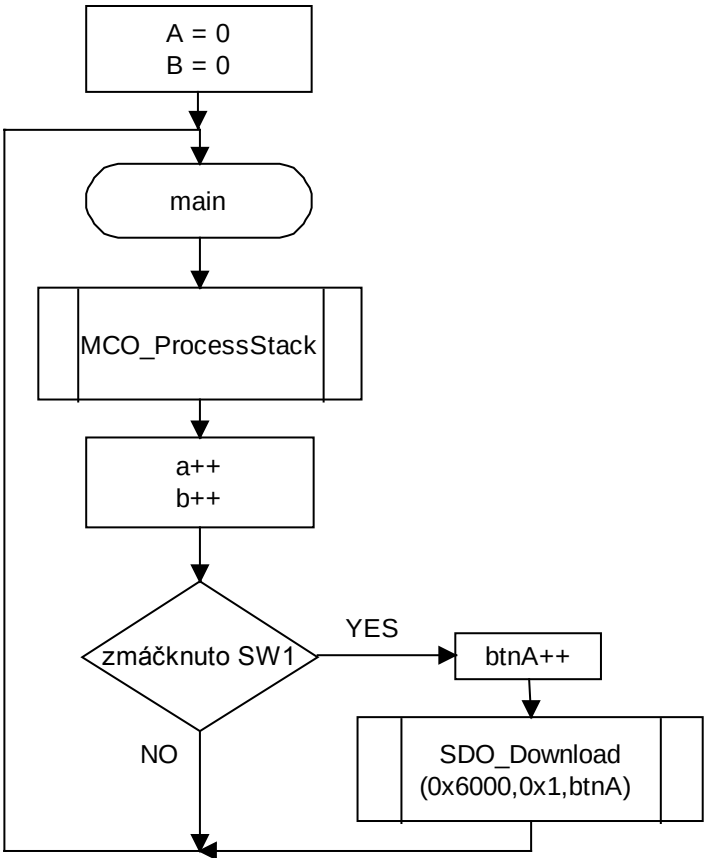
Tento zdrojový soubor obsahuje obsluhu všech přerušení. Konkrétně se jedná o *SysTick Handler*, který slouží k počítání času. U *CAN1_TX_IRQHandler* nastává přerušení

po odeslání rámce. U *CAN1_RX0_IRQHandler* a *CAN1_RX1_IRQHandler* dochází k přerušení při příjmu zprávy. Zdrojový soubor pak ještě obsahuje funkci main.

int main (void)

Funkce obsahuje nekonečnou smyčku while, ve které běží celý program. Na Obrázku 28 je vývojový diagram funkce main.

Jako první se volá funkce *MCO_ProcessStack*. Následně se inkrementují proměnné *a, b*, které v programu simulují data, jež jsou pak poslána po sběrnici. Jako poslední probíhá testování stavu tlačítka *SW1*. Pokud dojde k jeho zmáčknutí, inkrementuje se proměnná *BtnA* a pomocí funkce *SDO_Download* se do adresáře deskriptorů na pozici analogového vstupu zapíše aktuální hodnota inkrementované proměnné. Testování tlačítka slouží k simulaci využití PDO mappingu, který je realizován pomocí vysílacího objektu TPDO1. Vysílací objekt TPDO1 má data na vyslání namapované na adresu, kde jsou uložena analogová data. PDO mapping je možný právě u MicroCANopen Plus.



Obrázek 28. Vývojový diagram funkce main

void SysTick Handler(void)

Jedná se o přerušení od periférie *SysTick* (časovač jádra). Na začátku obsluhy přerušení dochází k inkrementaci proměnné *tikani*. Tímto způsobem je získáván počet milisekund od počátku programu. Časový údaj 1 ms je volitelný. Stačí změnit nastavení registrů v obslužném zdrojovém souboru *inicializace.c*. Takto zvolený čas je ale standardní dle doporučení. Dále v obsluze dochází k rozvětvení programu podle toho, která událost má podle aktuálního času nastat.

První událost je platná, pokud je vyhodnoceno, že nastal čas periody heartbeat a zároveň je NMT ve stavu Operational, Pre-operational nebo Stopped. Vyhodnocení času události se provede rozdílem aktuálního času a času, kdy nastala poslední splněná podmínka (*tikani – last_HB*). Rozdíl musí být větší nebo roven nastavené hodnotě periody HB v *include.h*. Takto postavená podmínka je důležitá pro vyhnutí se práci s modulo, které nemusí vždy fungovat, například pokud je program zaneprázdněn jinou činností. Systém pomocí rozdílu funguje i při obsluze přerušení v jiném čase než přesně modulem definovaném. Pokud je podmínka splněna, proměnná *signal_HB* se nastaví do true, což slouží jako informace pro samotný protokolový zásobník, kde dochází ke shození příznaku. Aktuální čas se uloží do proměnné *last_HB*, aby mohl systém vyhodnocení rozdílu dvou časů dál fungovat.

Další čtyři události se starají o obsluhu TPDO objektů 1 až 4. Podmínky jsou konstruovány všechny obdobně jako v předchozím případě, pomocí rozdílů časů. Tentokrát se jedná o rozdíl (*tikani – set_TPDO[x].last_time*). Menšitel je hodnota posledního času události ze struktury konkrétního TPDO, kde se za x dosazují čísla 0–3 podle objektu TPDO. Vedle správného rozdílu musí být ještě NMT ve stavu Operational. Pokud jsou obě podmínky platné, nastaví se *signal_TPDOx* (x nabývá hodnoty 1–4 podle konkrétního TPDO) do stavu true a do položky *last_time* ve struktuře TPDO se uloží aktuální čas. Proměnné *signal_TPDOx* jsou v tomto případě také za spolupráce s funkcí *ProcessStack*, která obsluhuje celý protokolový zásobník, shozeny.

void CAN1_TX_IRQHandler (void)

Tato funkce obsluhuje přerušení, které je vyvoláno po odeslání zprávy. Funkce se rozděluje na čtyři větve. Jednotlivé větve se rozlišují na základě proměnné *number_T*. Tato proměnná se rovná nule, pokud jde o odeslání TPDO1, jedničce, pokud jde o TPDO2, dvojce při TPDO3, trojce při TPDO4, šestce pro heartbeat a nakonec sedmičce u služby

SDO. U jednotlivých TPDO nejprve dojde ke zrušení příznaku interrupt v příslušném registru a následně se testuje, jestli nečeká nějaká zpráva na odeslání (*signal_TPDOx* = true). Pokud by se tak stalo, proběhne nastavení proměnné *number_TPDOx* = 0, 1, 2 nebo 3 podle toho, jakou TPDO zprávu bude později funkce *MCO_initTPDO* nastavovat. Dále se nastaví proměnná *number_T* = 0, 1, 2 nebo 3 opět podle toho, o jakou TPDO zprávu se jedná. Proměnná se později použije ve funkci *MCOHW_PushMessage*, aby funkce věděla, pro jaký TPDO má nastavit registry a poslat data. Následně se do bufferu přepíšou data k poslání, zavolá funkce *MCO_initTPDO* pro nastavení TPDO a nakonec se zavolá funkce *MCOHW_PushMessage* pro samotné odeslání dat.

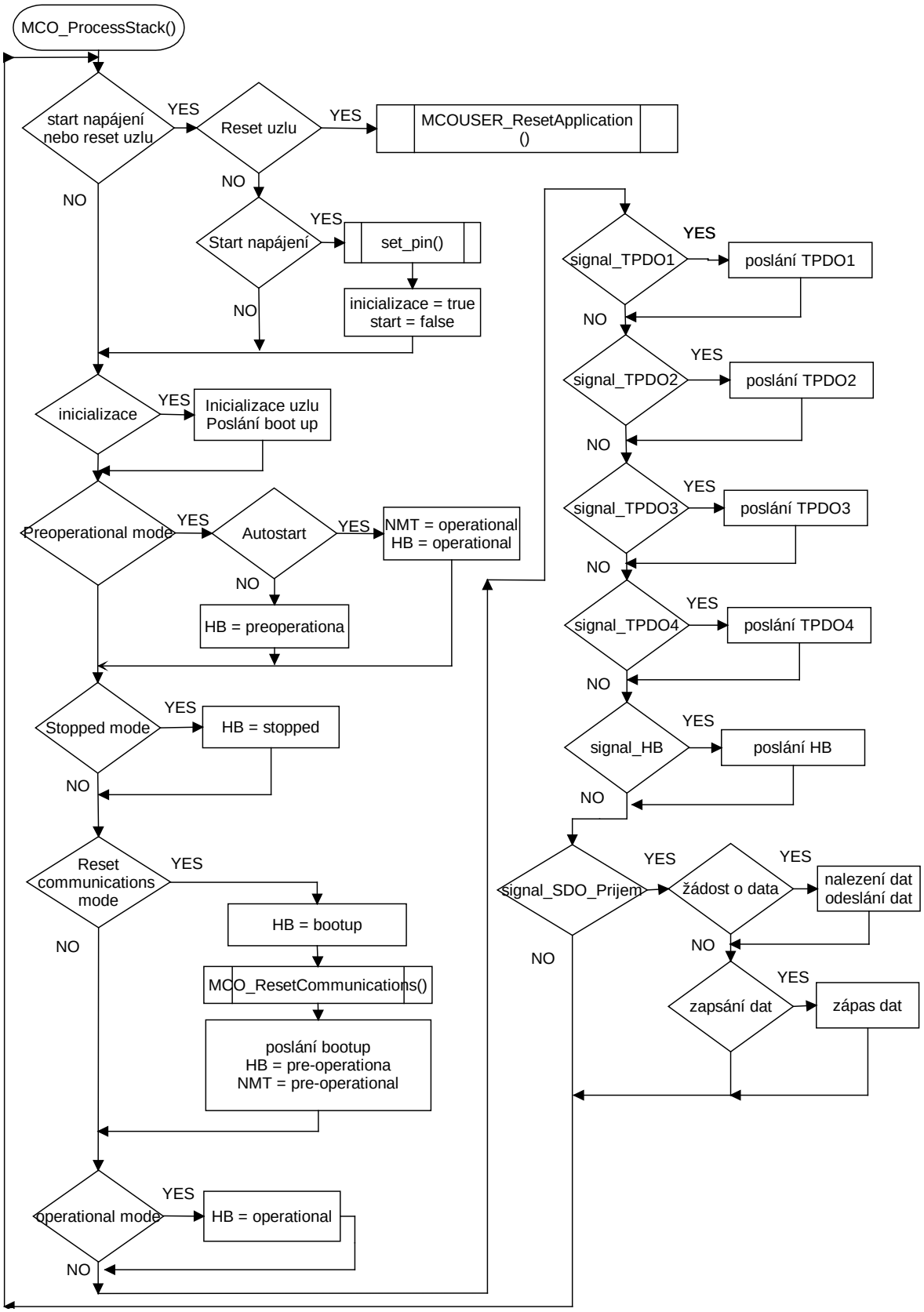
void CAN1_RX0_IRQHandler (void), void CAN1_RX1_IRQHandler (void)

Toto přerušení je vyvoláno, když dojde k příjmu zprávy. Mikrokontrolér má dva vektory přerušení určené pro příjem zprávy. RX0 je určen pro přijímací schránku 0 a RX1 pro přijímací schránku 1. Při každém vyvolání přerušení proběhne nastavení proměnné signal_prijem = 1 nebo 2 podle toho, které přerušení nastalo. Tato informace je důležitá pro funkci *MCOHW_PullMessage*, která přijatou zprávu dále zpracovává. V dalším kroku se funkce *MCOHW_PullMessage* zavolá a předají se jí přijatá data. Tím končí obsluha přerušení.

3.2.3 ProcessStack

Tato funkce obstarává řízení celého protokolového zásobníku. Na *Obrázku 29* je vidět vývojový diagram. Nejprve se testuje, jestli došlo k prvnímu startu systému, nebo k resetu uzlu. Pokud nastala první možnost, to znamená, že proměnná start = true a NMT je ve stavu reset aplikace, zavolá se funkce *set_pin* pro kompletní nastavení všech registrů mikrokontroléru. Poté se proměnná inicializace nastaví na true a proměnná start do false. Pokud nastane druhá možnost, zavolá se funkce *MCOUSER_ResetApplications*, která zařídí další kroky k realizaci resetu uzlu. Dalším krokem je testování proměnné inicializace. Pokud je ve stavu true, proběhne inicializace uzlu, tzn. zavolání funkce *MCO_Init*, která nastaví přenosovou rychlost, HB a Node-ID. Po skončení inicializace se pošle ještě zpráva bootup, proměnné HB a NMT se nastaví do stavu Pre-operational a inicializace do stavu false. Po tomto kroku je testováno, jestli je uzel v Pre-operational módu. Pokud ano, dále se testuje, jestli uzel běží v režimu autostart, nebo ne. Pokud je

povolen autostart režim, uzel přechází automaticky do stavu Operational, to znamená nastavení NMT a HB do stavu Operational. Pokud uzel nemá umožněn autostart, NMT a HB zůstává v režimu Pre-operational. V dalším kroku *ProcessStack* se testuje stav reset communication. Dojde-li k tomuto stavu, musí se nastavit *HB* do stavu bootup, zavolá se funkce *MCO_ResetCommunication* a odešle se bootup zpráva. Nakonec se *HB* a *NMT* nastaví do stavu Pre-operational. V další části je testováno, jestli nepřišel signál pro odeslání TPDO1–TPDO4 objektu, HB, případně k příjmu SDO služby. To se pozná podle proměnných *signal_TPDO1*, *signal_TPDO2*, *signal_TPDO3*, *signal_TPDO4*, *signal_HB* nebo *signal_SDO_prijem* = true. Při *signal_TPDO1* = true se nejprve nastaví proměnné *number_TPDO* a *number_T* = 0. To je důležité pro funkce *MCO_Init* a *MCOHW_PushMessage*. Následně se pomocí funkce *SDO_Search* provede mapping. Tím se získá hodnota pro *TransmitBuf*, která bude odeslána. Poté se už jen zavolá funkce *MCO_InitTPDO* a *MCOHW_PushMessage*. U ostatních signálů – pro TPDO2, TPDO3 nebo TPDO4 – vše probíhá stejně jako u TPDO1, s výjimkou TPDO mapping, místo nějž se do proměnné *TransmitBuf* zapíše proměnná *a*, nebo *b*. Proměnné *number_TPDO* a *number_T* jsou pak nastaveny na číslo 1–3 podle konkrétního TPDO. Pokud se jedná o signál pro HB, nastaví se *number_T* = 6, *TransmitBuf* = HB a opět pomocí funkce *MCOHW_PushMessage* dojde k odeslání. Pokud se jedná o signál SDO služby, nastává složitější situace. Musí se nejprve podle ccs hodnoty rozhodnout, jestli je o data žádáno, nebo jestli budou naopak přijímána. V obou případech se nastaví proměnná *number_T* = 7 a do pomocných proměnných se vyčte index a subindex. Další postup se ale liší. Při příjmu dat se volá funkce *SDO_Download* pro zapsání dat na správnou pozici. Následně se zavolá funkce *MCOHW_Response_SDO_message*, které se předají dva parametry. První parametr obsahuje inicializační část a druhý data a v tomto případě zůstanou data prázdná. Pokud se ale jedná o žádost o data, zavolá se funkce *SDO_Search* pro získání žádaných dat. Poté se pomocí funkce *MCOHW_Response_SDO_message* data odešlou. Druhým parametrem tentokrát není nula, ale žádaná data.



Obrázek 29. Vývojový diagram ProcessStack

3.2.4 Zdrojový soubor MCO.c

Druhým zdrojovým souborem je MCO.c. Soubor obsahuje pole s jednotlivými položkami slovníku deskriptorů. Právě tady lze do adresáře ručně dopisovat, případně upravovat již připravené položky. Dále jsou v MCO zdrojovém souboru samotné funkce, kterým se teď budu podrobněji věnovat.

void MCO_Init (unsigned int Baudrate,unsigned char Node_ID,unsigned int Heartbeat)

Jedná se o funkci, která nevrací žádnou návratovou hodnotu, ale sama očekává tři parametry. Jde o přenosovou rychlost Baudrate, číslo uzlu Node-ID a frekvenci Heartbeat. Funkce, jak už očekávané hodnoty napovídají, slouží k inicializaci nebo re-inicializaci protokolového zásobníku, pokud dojde k resetu komunikace, popř. celého uzlu. Pokud se předávají nulové hodnoty, jako v základním nastavení této diplomové práce, tak se používají předdefinované hodnoty z hlavičkového souboru *include.h*. V opačném případě se použije hodnota, která je funkci přímo předána.

Ve funkci se nejdříve přiřadí předdefinované hodnoty příslušné položce ve struktuře *set_MCO_INIT*. Následně dojde k nastavení přenosové rychlosti pomocí funkce *MCOHW_Init*, která je popsána v další části, věnované zdrojovému souboru mcohw.c. Poté proběhne čtyřikrát while smyčka, pomocí níž se do struktury *set_RPDO[number_RPDO]*, která se stará o nastavení RPDO, nastaví hodnoty z hlavičkového souboru *include.h*. Jedná se o číslo PDO_NR, CAN_ID a len. Poté se ještě zavolá funkce *MCO_InitRPDO*, která se stará o nastavení hodnot ze struktury do registrů samotného mikrokontroléru. Funkce bude popsána později. Po nastavení RPDO se stejným způsobem nastaví i data do struktury TPDO. V tomto případě se jedná o *PDO_NR, CAN_ID, event_time, inhibit_time, len* a *last_time*. Oproti nastavení RPDO se u TPDO nevolá žádná funkce pro nastavení hodnot ze struktur do registrů. To je dáno nedostatkem vysílacích boxů samotného mikroprocesoru, a tím pádem nemožností nastavit vysílací parametry předem do registrů. Nastavení se provádí vždy až těsně před odesláním zprávy. Na samotném konci funkce se pak ještě nastaví příjem NMT, aby mohl být uzel spuštěn.

void MCO_InitRPDO (unsigned char PDO_NR,unsigned int CAN_ID, unsigned char len)

Tato funkce se stará o zapsání hodnot ze struktury RPDO do registrů v mikroprocesoru. Skládá se ze čtyř větví. Podle *PDO_NR*, které bylo RPDO objektu přiřazeno v předchozí funkci, se vybere správná větev a hodnoty z již naplněné struktury

set_RPDO se nastaví do registrů. Na konci každého nastavení se pak ještě nastaví pomocí funkce *MCOHW_SetCANFilter* filtr přijímacího objektu RPDO. Funkce pro nastavení bude popsána později v rámci zdrojového souboru mcohw.c. Na konci funkce pak proběhne aktivace filtru a deaktivace inicializace filtru.

MCO_InitTPDO (unsigned char PDO_NR, unsigned int CAN_ID, unsigned int event_time, unsigned int inhibit_time,unsigned char len)

Tato funkce se stará o nastavení registrů mikroprocesoru pomocí TPDO struktury *set_TPDO* stejným způsobem jako předchozí funkce, s tím rozdílem, že na vyhodnocení, která větev má být použita, se používá pomocná proměnná *number_TPDO*, a ne přímo položka *PDO_NR* jako v předchozí funkci. To je způsobeno tím, že se nastavení registrů provádí voláním funkce z protokolového zásobníku až těsně před odesláním zprávy, a ne z funkce *MCO_Init*. Důvodem je nedostatek vysílacích schránek. Problém byl již popsán výše u popisu funkce *MCO_Init*.

void SDO_Search (uint16_t indexx, uint8_t subindex)

Tato funkce slouží pro službu SDO. Pokud přijde žádost o získání konkrétní hodnoty ze slovníku deskriptorů, tato funkce pomocí předaných parametrů index a subindex prohledá slovník a najde žádaná data. Funkce funguje tak, že ve while smyčce se nejprve projíždí indexy ve slovníku a porovnávají se s požadovaným indexem. Pokud nastane shoda, program se ve slovníku posune o jednu položku a porovná subindex. Pokud i tentokrát nastane shoda, provede se ve slovníku posun o dvě pozice k datům. Získaná data se zapíší se do proměnné *SDO_TrasmitBuf_Data*. Také se posunem zjistí velikost přijatých dat a zapíší se do proměnné *velikost_SDO*. Nakonec se nastaví proměnná *found*, která indikuje nalezení položky, do stavu true. Není-li nalezena shoda v indexu, program se ve slovníku posune o 4 pozice a porovná další index. Pokud neshoda nastane až v subindexu, provede se nejprve posun o jednu pozici zpět, aby mohl být vzápětí opět proveden posun o 4 pozice na další index slovníku. While cyklus běží stále dokola, dokud ve slovníku nenarazí na položku plnou 0xFF hodnot, která značí konec slovníku, nebo pokud proměnná *found* není ve stavu true.

SDO_Download (uint16_t indexx, uint8_t subindex, uint32_t SDO_Download_data)

Funkce slouží opět pro službu SDO, tentokrát však ne při požadavku o data, ale při požadavku o přepsání dat ve slovníku. Funkce funguje stejně jako ta předchozí, ale protože se jedná o zápis do slovníku, data se při nalezení správné pozice pomocí indexu a subindexu nezapisují do proměnné *SDO_TrasmitBuf_Data*. Funkce má jeden parametr navíc, obsahující data, která byla přijata a mají být zapsána do slovníku. Po nalezení pozice se program posune o dvě pozice dopředu do dat a přepíše je. Protože některá přepsaná data se mohou používat i v samotném protokolovém zásobníku, jako například frekvence HB, ještě se pomocí větvení nová hodnota slovníku kopíruje do struktury *set_TPDO, set_MCO_INIT* a tak dále podle toho, kam je potřeba. V rámci této práce je možné takto kopírovat *event_time* TPDO objektů a heartbeat.

void MCOUSER_ResetCommunication (void)

Pokud je potřeba restartovat komunikaci, je volána tato funkce. Ta v sobě obsahuje pouze volání funkce *MCO_Init*, popsané výše. Zavoláním této funkce dojde k re-inicializaci komunikace. Konkrétně se znovu nastaví komunikační rychlost, perioda vysílání HB a číslo uzlu.

void MCOUSER_ResetApplication (void)

Tato funkce je volána, pokud dojde k příkazu na reset aplikace. Tato situace je v reálných podmínkách velmi nebezpečná, protože pokud by se například jednalo o řízení rozjetého motoru a podobně, mohly by nastat velké škody. Proto může tato funkce být velmi složitá a obsahovat mnoho reakcí na různé situace konkrétního procesu. Požadavek na konkrétní reakci musí být aplikaci předán pomocí stavového rozhraní mezi aplikací a komunikačním driverem. Aplikace na základě těchto informací vyhodnotí, co se bude dále vykonávat. V rámci tohoto programu je problém simulován následovně. Funkce nejprve rozsvítí LED LL0–LL3 na vývojovém kitu a inkrementuje proměnou *Reset_Node*, která plní funkci počítadla tohoto stavu. Následně se čeká na zmáčknutí tlačítka SW2, kterým se simuluje zásah obsluhy zařízení. Po zmáčknutí tlačítka se proměnná HB nastaví do stavu bootup. NMT sloužící k obsluze network management se nastaví do resetu aplikace a proměnná start do stavu true. Tím dostává protokolový zásobník informaci, aby provedl kompletní reset uzlu jako po zapnutí napájení.

void MCO_ProcessStack (void)

Tato funkce byla popsána zvlášť v kapitole 3.2.2 *ProcessStack.*

3.2.5 Zdrojový soubor mcohw.c

Třetím zdrojovým souborem je mcohw.c. Obsahuje pět funkcí, které jsou pro každý mikroprocesor jedinečné. Jedná se totiž o funkce přímo závislé na použitém hardwaru.

unsigned char MCOHW_Init (unsigned int BaudRate)

Tato funkce slouží k nastavení přenosové rychlosti. Je volána například při inicializaci funkcí *MCO_Init* nebo při resetu komunikace funkcí *MCOUSER_ResetCommunication*. Ve funkci je switch, který nastaví přenosovou rychlost na základě předaného parametru, obsahujícího žádanou rychlost. Pokud je žádaná přenosová rychlost v seznamu, provede se nastavení registrů a do proměnné set se nastaví hodnota true. Podle této hodnoty se na konci funkce může zvolit návratová hodnota 1, pokud byla nastavena přenosová rychlost nebo 0. Nula se nastaví v případě, že nebyla žádaná přenosová rychlost v seznamu možností nalezena a nastaví se výchozí hodnota 125 kbit/s.

unsigned char MCOHW_SetCANFilter (unsigned int CANID)

Funkce nastavuje filtry příjmu, takzvané akceptační filtry, pro RPDO. Pomocí hlavičkového souboru *include.h* lze předdefinovat hodnotu masky a CANID. Tímto lze dosáhnout určité možnosti modifikace. V rámci této diplomové práce je filtr přednastaven na příjem adres pro RPDO1 v rozsahu 0x180–0x180 + Node-ID (max. 127), RPDO2 pak v rozsahu 0x280–0x280 + Node-ID (max. 127) a obdobně i pro RPDO3 a RPDO4. Tento rozsah byl zvolen pro možnou křížovou komunikaci mezi více uzly. Proměnná *nof* slouží k inkrementaci registru filtru při nastavování jednotlivých filtrů pro konkrétní RPDO.

*void MCO_PushMessege (int *p_TransmitBuf)*

Jedná se o jednu z nejdůležitějších funkcí. Obstarává totiž posílání TPDO objektů a heartbeat. Jejím jediným parametrem je **p_TransmitBuf,* který obsahuje data k odeslání. Ve funkci se pomocí proměnné *number_T* pozná, který TPDO objekt se má připravit. Pro

TPDO1 náleží *number_T* = 1, pro TPDO2 je to *number_T* = 2 a tak dále až po TPDO4, kde je hodnota *number_T* = 3. Heartbeat pak má hodnotu *number_T* = 6. Při skoku do správné větve if se nejprve překopírují data do vysílacího registru a následně se nastaví příslušný příznak odesílání. Protože je v mikrokontroléru nedostatek vysílacích schránek, musí se TPDO1 a TPDO2 dělit o TxMailBox0 a TPDO3 s TPDO4 o TxmailBox1. Heartbeat má vlastní schránku TxmailBox2.

*void MCOHW_pullMessage (int *p_ReceiveBuf)*

Funkce obstarává příjem TPDO, SDO i NMT. Je volána, pokud dojde k přerušení od příjmu zprávy. Nejprve se rozhodne pomocí proměnné *signal_prijem*, jestli se jedná o příjem z přerušení RX0, nebo RX1. Poté se skočí do správné větve if a zjistí se z registru příjmu adresa, ze které byla data vyslána. Pak nastává dělení do tří větví. První větev se pozná podle adresy příjmu 0x00. To informuje o příjmu příkazu z NMT. V takovém případě se do proměnné *NMT* zapíše, do jakého stavu má uzel přejít. Druhá větev se pozná podle toho, že je uzel v operačním módu, adresa je rozdílná od nuly a zároveň není rovna adrese 0x600 + Node-ID. V takovém případě se jedná o příjem RPDO zpráv. Pomocí ukazatele **p_ReceiveBuf* se zapíše přijatá zpráva a následně se schránka uvolní. Třetí větev se pozná podle toho, že NMT stav je v operačním nebo pre-operačním módu a zároveň je adresa rovna 0x600 + Node-ID. V takovém případě se jedná o SDO službu. Proto následuje získání datové a inicializační části a přesun do příslušných proměnných *SDO_ReceiveBuf_ini* a *SDO_ReceiveBuf_data* pro další použití v protokolovém zásobníku. Dále už se jen uvolní schránka příjmu a proměnná *signal_SDO_Prijem* se nastaví do stavu true, aby protokolový zásobník poznal, že musí obstarat obsluhu SDO.

*void MCOHW_Response_SDO_Message(int*p_SDO_TransmitBuf_ini, in*p_SDO_Transit Buf_Data*)

Tato funkce obstarává také odesílání, ale tentokrát odpovědi SDO služby. Funkce je vytvořena pro větší přehlednost. Při odesílání přes funkci *MCOHW_pushMessage* by bylo třeba psát další podmínky a kód by se stal nepřehledným. Ve funkci proběhne jen nastavení vysílací schránky a následné odeslání zprávy. Samotné sestavení zprávy probíhá už ve funkci *ProcessStack* a funkci jsou pak předány už jen dva parametry. Jeden parametr obsahuje inicializační část odpovědi SDO a druhý datovou část zprávy. Pro odeslání se používá TxMailbox2 jako u HB.

Závěr

Cílem diplomové práce bylo implementovat komunikační protokol MicroCANopen Plus na mikrokontrolér řady STM32. Jako zástupce byl zvolen kit Nucleo F429ZI. První kapitola se věnuje všeobecnému popisu specifikace standardu CAN. Druhá kapitola je již zaměřená výhradně na specifikaci komunikačního profilu CANopen DS-301. Jsou zde popsány komunikační a servisní objekty a další náležitosti tohoto profilu.

Třetí kapitola je pak věnována samotné realizaci protokolového zásobníku. Je zde kladen důraz na rozdělení projektu do dvou vrstev. Jedna vrstva je hardwarově závislá na samotném Nucleo kitu a funguje v reálném čase. To znamená, že je využito přerušení, která obsluhují odesílání, příjem zpráv a zajišťují počítání času. Tím je zajištěna co nejmenší prodleva, pokud dojde k události vyžadující reakci v reálném čase. Jedná se typicky o PDO objekty. Druhá vrstva už implementuje MicroCANopen Plus protokolové služby. Jedná se procesní datové objekty, kde je na TPDO1 umožněn i PDO mapping. Celkově je v práci použito čtyř TPDO objektů a čtyř RPDO objektů. Každý z nich je možné modifikovat pomocí *include.h* souboru a částečně také pomocí SDO služeb. Dále jsou tu servisní datové objekty pro přístup do adresáře deskriptorů. Samotný adresář deskriptorů obsahuje specifikací povinné i další vybrané položky. Příkladem mohou být položky procesních datových objektů jako např. event time. Další službou, která je v práci implementována, je network management, který umožňuje řízení uzlu od nadřazené master stanice. S touto službou souvisí i možnost volby, jestli uzel bude pracovat v klasickém, nebo autostart režimu. Druhou službou navázanou na network management je služba heartbeat. Ta v základní konfiguraci diplomové práce běží s periodou 500 ms, lze ji ale modifikovat pomocí SDO služeb nebo před spuštěním uzlu pomocí přepsání *include.h* souboru.

V rámci této kapitoly je věnován prostor detailnímu popsání jednotlivých funkcí programu, které se starají o chod protokolového zásobníku a použitých struktur programu. Celý tento protokolový zásobník je periodicky volán z nekonečné while smyčky samotné main funkce. Tím je zajištěna malá prodleva programu. Protokolový zásobník je realizován v jazyce C, a ne v jeho objektové podobě C++. Důvodem je jeho používání ve škole v rámci pětiletého studia.

Práce může sloužit jako dobrý základ pro další rozšiřování služeb, které MicroCANopen Plus umožňuje.

Seznam literatury a informačních zdrojů

[1] PFEIFFER, Olaf, Andrew AYRE a Christian KEYDEL. *Embedded Networking with CAN and CANopen*. San Clemente, CA: RTC Books, 2013. ISBN 978-0-9765116-2-5.

[2] Robert Bosch GmbH, "CAN Specification 2.0," 1991

[3] CANopen application layer and communication profile", CiA Draft Standard 301, Version 4.02

[4] Embedded system academy,inc. [online].MCOv2.00 [cit. 01. 5. 2019]. Dostupné z: https://www.canopenbook.com/files/MCOv2.00.zip

[5] Embedded system academy,inc. [online].Micro CANopen Plus User Manual [cit. 01.5.2019]. Dostupné z: http://www.esacademy.org/products/getfile.php?filename=MicroCANopen%20Manual.pdf

[6] STMicroelectronic [online katalogový list]. RM0090 Reference manual. ©2019 [cit. 01. 5. 2019]. Dostupné z: https://www.st.com/content/ccc/resource/technical/document/reference_manual/3d/6d/5a/66/b4/99/40/d4/DM00031020.pdf/files/DM00031020.pdf/jcr:content/translations/en.DM00031020.pdf

[7] STMicroelectronic [online katalogový list].STM32F429xx STM32F429xx. ©2018 [cit. 01. 5. 2019]. Dostupné z: **https://www.st.com/resource/en/datasheet/stm32f429zi.pdf**

[8] USB-CAN adapter – TRIPLE drivers V4.2 (high/ low/ one wire). IMFsoft, s.r.o. [online]. © 2006 – 2019 [cit. 01.5.2019]. Dostupné z: http://imfsoft.com/hardware/usb-can-adapter-triple-drivers-v4-2-high-low-one-wire

Přílohy

Přílohy na CD:

- Příloha A – kód programu
- Příloha B – schéma modulu CAN