Fakulta elektrotechnická
Katedra aplikované elektroniky a telekomunikací

Diplomová práce

Převodník protokolu Contact ID z telefonní linky na protokol TCP

Autor práce: Bc. Jindřich Reiser
Vedoucí práce: Ing. Jiří Stifter, Ph.D.

Plzeň 2013

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta elektrotechnická
Akademický rok: **2012/2013**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Jindřich REISER**
Osobní číslo:	**E11N0076P**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Telekomunikační a multimediální systémy**
Název tématu:	**Převodník protokolu Contact ID z telefonní linky na protokol TCP**
Zadávající katedra:	**Katedra aplikované elektroniky a telekomunikací**

Zásady pro vypracování:

1. Seznamte se s protokolem Contact ID a popište princip komunikace.
2. Navrhněte protokol pro přenos zprávy Contact ID po TCP.
3. Navrhněte a zrealizujte převodník protokolu Contact ID z telefonní linky na protokol TCP.
4. Vytvořte aplikaci pro příjem zpráv z převodníku na PC.
5. Otestujte funkčnost celého systému.

Rozsah grafických prací: **podle doporučení vedoucího**

Rozsah pracovní zprávy: **30 - 40 stran**

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

Seznam odborné literatury:

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

Vedoucí diplomové práce: **Ing. Jiří Stifter, Ph.D.**
Katedra aplikované elektroniky a telekomunikací

Konzultant diplomové práce: **Ing. Jiří Stifter, Ph.D.**
Katedra aplikované elektroniky a telekomunikací

Datum zadání diplomové práce: **15. října 2012**

Termín odevzdání diplomové práce: **9. května 2013**

L.S.

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

Doc. Dr. Ing. Vjačeslav Georgiev
vedoucí katedry

V Plzni dne 15. října 2012

Abstrakt

Práce se zabývá návrhem a realizací převodníku protokolu Contact ID, který umožňuje přenášet zprávy ze zařízení elektronického zabezpečovacího systému z telefonní linky na pult centrální ochrany prostřednictvím protokolu TCP. Práce obsahuje popis přenosových protokolů používaných v popisovaném převodníku, návrh a realizaci prototypu, funkci programového vybavení mikrokontroléru, tvorbu aplikace pro příjem zprávy z převodníku a reálného zapojení.

Klíčová slova

Poplachové systémy, Převodník, Contact ID, IP CID

Abstract

Reiser, Jindřich. *Converter of Contact ID protocol on telephone line to TCP protocol* [*Převodník protokolu Contact ID z telefonní linky na protokol TCP*]. Pilsen, 2013. Master thesis (in Czech). University of West Bohemia. Faculty of Electrical Engineering. Department of Applied Electronics and Telecommunications. Supervisor: Jiří Stifter

---

This thesis describes the design and implementation of a protocol converter Contact ID of electronic security system equipment from the telephone line to the alarm response center via TCP. This thesis describes the transmission protocols used in the described converter, design and implementation of a prototype, the microcontroller firmware, application to receive messages from the converter and the real application.

Keywords

Alarm system, Converter, Contact ID, IP CID

Prohlášení

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

Prohlašuji, že jsem svou závěrečnou práci vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 270 trestního zákona č. 40/2009 Sb.

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

V Plzni dne 30. dubna 2013

Bc. Jindřich Reiser

.....

Podpis

Obsah

Seznam obrázků

Seznam tabulek

Seznam symbolů a zkratek

ARP	Address Resolution Protocol. Protokol pro získání adres.
ASCII	American Standard Code for Information Interchange. Americké normované kódy pro výměnu informací.
CRC	Cyclic redundancy check. Cyklická redundantní kontrola.
DHCP	Dynamic Host Configuration Protocol. Protokol pro dynamické získávání konfigurace.
DPS	Deska plošných spojů.
DTMF	Dual-tone multi-frequency. Dvojtónová multi-kmitočtová volba.
EPS	Elektronický požární systém.
EZS	Elektronický zabezpečovací systém.
ICMP	Internet Control Message Protocol. Internetový protokol pro kontrolní zprávy.
IGMP	Internet Group Management Protocol. Internetový protokol pro skupinové řízení.
IO	Integrovaný obvod.
IP	Internet Protocol. Internetový protokol.
ISP	In-system programming. Programování v systému.
MCU	Microcontroler unit - Mikrokontrolér.
OZ	Operační zesilovač.
PCO	Pult centrální ochrany.
PoE	Power over Ethernet. Napájení po Ethernetu.
PSN	Packet-Switched Network. Paketově přepínaná síť.
QoS	Quality of Service. Kvalita služeb.
RTC	Real Time Clock. Hodiny reálného času.
SIP	Session Initiation Protocol. Protokol pro inicializaci relací.
SMTP	Simple Mail Transfer Protocol. Protokol pro přenos jednoduchých pošt.
TCP	Transmission Control Protocol. Protokol pro řízení přenosu.
TELNET	Telecommunication Network. Telekomunikační síť.
UDP	User Datagram Protocol. Protokol pro uživatelské datagramy.
VB.NET	Programovací jazyk Visual Basic s podporou .NET Framework.
VoIP	Voice over Internet Protocol. Protokol pro volání přes internet.

1

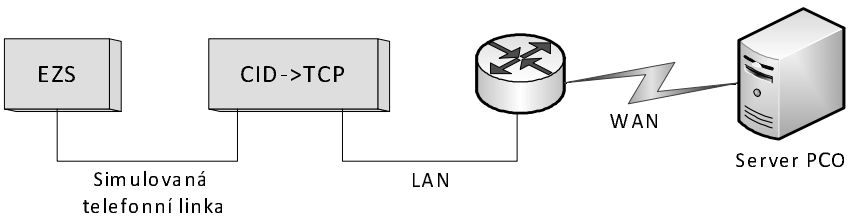
Úvod

V dnešní době již existuje velká řada zabezpečovacích ústředen, které umožňují odesílat zprávy na pult centrální ochrany prostřednictvím sítě Internet. Bohužel téměř všechny používají vlastní protokol, který není normalizován a proto ho podporuje jen velmi malý počet PCO (většinou jen od stejného výrobce, jako je zařízení EZS).

V současnosti je standardizován přenos zpráv na pult centrální ochrany prostřednictvím telefonní linky, kde se využívá protokol Contact ID vyvinutý společností Ademco. Ovšem velkou technickou překážkou je, že pevné telefonní linky se stále méně používají a proto je tento způsob z hlediska budoucnosti neperspektivní.

Tuto situaci se snaží vyřešit několik výrobců PCO, kteří využívají k přenosu zpráv ze zařízení EZS protokol SIP, který se běžně využívá ve VoIP telefonii. Ovšem VoIP technologie nejsou běžně navrhované pro přenos zpráv pro zabezpečovací technologii, kde jsou časté používané délky DTMF tónu 50ms. Takto krátké DTMF tóny přenáší kodeky využívané ve VoIP telefonii zkresleně a proto se nedetekují správně. Další velkou nevýhodou řešení přenášení zpráv prostřednictvím SIP protokolu je využití SIP serveru v jedné lokalitě všemi zařízeními.

Z důvodu, že předchozí způsob přenosu na PCO je velmi nespolehlivý, byl navržen převodník, který je popisován v této práci. Převodník, který by přijímal zprávy z každého zařízení EZS vybavený telefonním komunikátorem a odesílal je přímo na PCO prostřednictvím protokolu TCP, na trhu schází. Převodník nejprve zprávu přijme, zpracuje a poté odešle na pult centrální ochrany v jediném paketu. Další výhodou tohoto řešení je využití šifrování proti nežádoucímu odposlechu. Umístění převodníku v zabezpečovacím řetězci je znázorněno na obr. 1.1



Obr. 1.1: Umístění převodníku v zabezpečovacím řetězci

Tato práce je rozdělena do pěti kapitol. První kapitola se zabývá popisem protokolů, které jsou u převodníku využity. Druhá kapitola se zabývá návrhem a realizaci převodníku. Třetí kapitola popisuje programové vybavení mikrokontroléru. Čtvrtá kapitola je zaměřena na aplikaci přijímající zprávy z převodníku a pátá kapitola popisuje reálné zapojení, užití převodníku a postup jeho testování.

2

Protokoly pro přenos dat ze zabezpečovacích zařízení

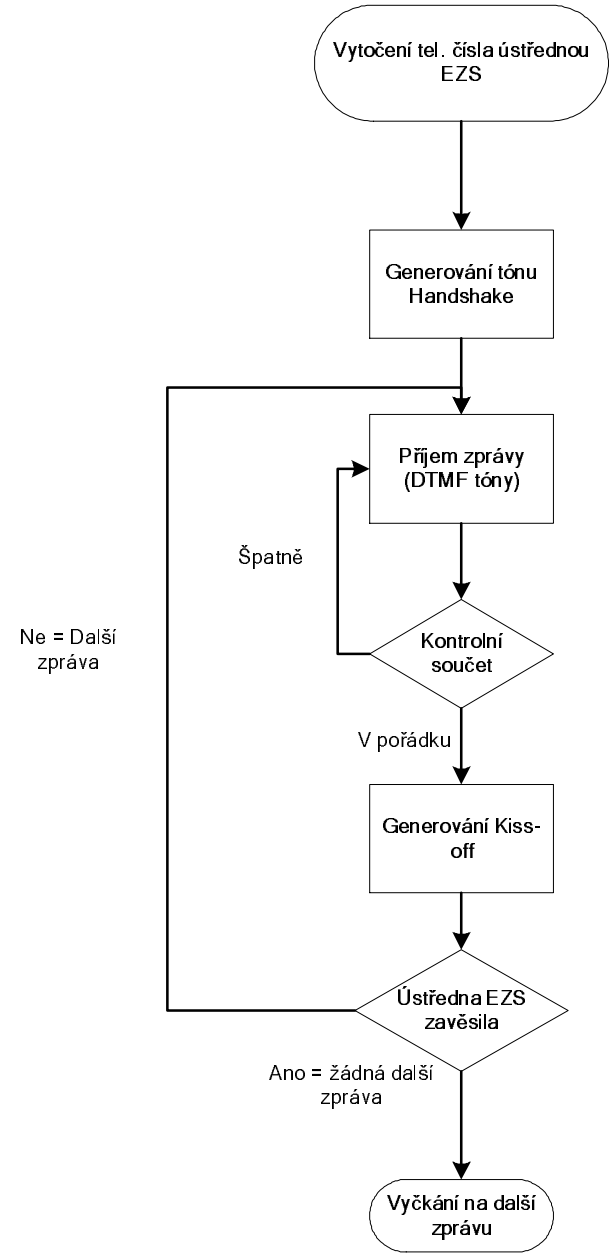
2.1 Protokol Ademco Contact ID

Dnes jedním z nejrozšířenějším a nejpoužívanějším protokolem pro přenos dat z EZS nebo EPS po telefonní lince je Ademco Contact ID, který je standardizován normou americké společnosti SIA DC-05 [1] z roku 1999. Tímto protokolem komunikují prakticky všechna zařízení EZS, EPS a PCO.

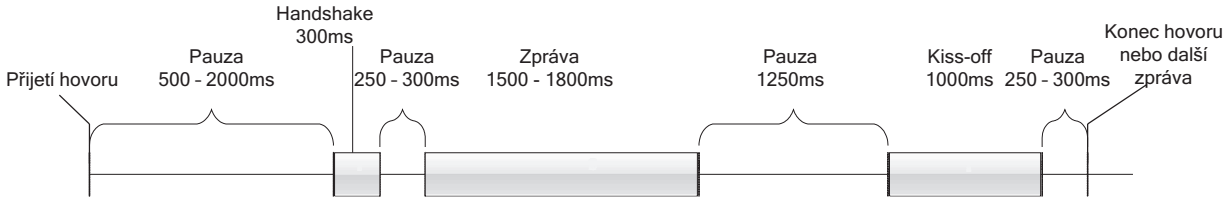
Komunikace protokolu Ademco Contact ID je znázorněna na obrázku 2.1 a časový průběh protokolu je na obrázku 2.2. Přenos dat tohoto protokolu lze rozdělit do tří základních částí. První část je tzv. handshake tón, který vysílá přijímací zařízení, tj. PCO nebo v případě této práce převodník Contact ID na protokol TCP. Tento tón oznamuje, že komunikační kanál je připraven pro přijetí zprávy. Handshake tón se skládá ze dvou harmonických tónů o kmitočtu 1400Hz a 2300Hz s jednotlivými délkami 100ms a tyto tóny jsou oddělené mezerou o délce 100ms. Zpoždění mezi vyzvednutí telefonní linky pultem centrální ochrany a vysláním handshake tóne vysílacím zařízením je typicky 500ms a nebývá delší než 2000ms.

Po handshake tónu se přenáší samotná zpráva ve formě DTMF tónů. Jedná se o sekvenci 16 znaků, které jsou vysílané z EZS nebo EPS na PCO nebo do převodníku popisovaný v této práci. Detailnější popis této části je v kapitole 2.1.1.

Po úspěšném přijetí zprávy je se zpožděním 1250ms po posledním znaku vyslán tzv. Kiss-off tón, který slouží jako potvrzení ze strany přijímacího zařízení (PCO, ...), že přenos proběhl úspěšně a bez chyb. Kiss-off tón je složen z harmonického tónu o kmitočtu 1400Hz, který je 750ms až 1000ms dlouhý. Vysílací zařízení (EZS, EPS, ...) musí zachytit minimálně 400ms tohoto Kiss-off tónu. Po přijetí Kiss-off tónu může vysílací zařízení zavěsit (ukončit přenos) nebo po uplynutí dalších 250ms až 300ms vyslat další zprávu. Pokud vysílací zařízení Kiss-off tón nezachytí nebo zachytí v kratším intervalu než je 400ms, tak je zpráva považovaná za nepřenesenou a přenos se opakuje. Vždy jsou provedeny 4 pokusy



Obr. 2.1: Komunikace protokolu Ademco Contact ID



Obr. 2.2: Časový průběh protokolu Ademco Contact ID

o přenos a poté vysílací zařízení zavěsí a po uplynutí určitého intervalu (zpravidla 1min) se celý přenos opakuje, dokud není zpráva úspěšně doručena.

2.1.1 Zpráva protokolu Ademco Contact ID

Jak již bylo napsáno v kapitole 2.1, tak přenos zprávy je proveden pomocí 16-ti DTMF tónů zpravidla o délce 50ms až 60ms. Mezery mezi jednotlivými znaky jsou dlouhé opět 50ms až 60ms. S takto krátkými intervaly mají problém některé telefonní kodeky používané např. ve VoIP. Hodnoty jednotlivých DTMF znaků jsou zobrazeny v tabulce 2.1.

DTMF znak	Nižší kmitočet [Hz]	Vyšší kmitočet [Hz]	Hodnota
0	941	1336	10
1	697	1209	1
2	697	1336	2
3	697	1477	3
4	770	1209	4
5	770	1336	5
6	770	1477	6
7	852	1209	7
8	852	1336	8
9	852	1477	9
B(*)	941	1209	11
C(#)	941	1477	12
D	697	1633	13
E	770	1633	14
F	852	1633	15

Tab. 2.1: Hodnoty jednotlivých DTMF tónů v protokolu Contact ID dané normou SIA DC-05 [1]

Přenášená zpráva je ve formátu **ACCT MT Q XYZ GG CCC S**:

ACCT – Čtyřmístné číslo objektu, které slouží k identifikaci objektu na zařízení PCO (každý střežený objekt má vzhledem k PCO jedinečné číslo). Hodnoty znaků mohou být 0 až 9 a B až F.

MT – Tyto znaky označují typ protokolu zprávy. U Contact ID se přenáší hodnota 18 nebo novější 98. Některé starší PCO ještě neumí hodnotu 98, proto je vhodnější přenášet pouze hodnotu 18.

Q – Tento znak označuje typ zprávy. Pokud je hodnota znaku rovna **1**, tak se jedná o novou událost nebo příchod do objektu. Hodnota znaku **3** udává ukončení události

(např. ukončení poplachu nebo servisního režimu) nebo odchod z objektu. Má-li znak hodnotu **6**, tak se jedná o stavovou zprávu, že předchozí událost stále trvá.

XYZ – Tyto znaky přenáší kód události. Kódy událostí jsou dány v příloze normy SIA DC-05 Ademco Contact ID [1]. Např. kód události 110 znamená požární poplach, kód 120 je tísňový poplach, kód 131 je narušení perimetru objektu, kód 137 je sabotáž zařízení EZS nebo kód 400 znamená příchod či odchod z objektu.

GG – Dvoumístné číslo skupiny, kde se přenášená událost stala. Skupina se využívá k rozdělení objektu na několik částí, kde můžeme mít označené přízemí budovy např. skupinou 01, patro skupinou 02 apod. Hodnoty znaků můžou být 0 až 9 a B až F. Hodnota 00 udává, že událost se nevztahuje k určité skupině, ale k celkovému objektu.

CCC – Trojmístné číslo zóny, kde se přenášená událost stala. Také se může jednat o identifikační číslo osoby pro události příchod, odchod apod. Hodnoty znaků mohou být 0 až 9 a B až F. Hodnota 000 udává, že se událost nevztahuje k určité zóně nebo osobě, ale k celkovému objektu.

S – Jedná se o kontrolní součet, který je dán vztahem:

$$(\sum_{i=1}^{15} x_i \ + \ S) \ MOD \ 15 \ = \ 0 \tag{2.1}$$

V rovnici 2.1 proměnná x reprezentuje hodnotu jednotlivých znaků zprávy. Hodnoty znaků jsou dány tabulkou 2.1.

Výsledná zpráva pak může mít tvar **1234 18 1 131 01 015 8**. Z této zprávy lze zjistit, že zprávu poslal objekt 1234 a zpráva je odeslána v protokolu Ademco Contact ID. Jedná se v tomto případě o nově vyvolaný poplach v perimetru ve skupině 01 v zóně 015. Poslední znak o hodnotě 8 je kontrolní součet, který je dán vztahem:

$$(1+2+3+4+1+8+1+1+3+1+10+1+10+1+5+8) \ MOD \ 15 \ = \ 0 \tag{2.2}$$

Zpráva o zrušení poplachu z předchozího odstavce bude mít tvar **1234 18 3 131 01 015 6**.

2.2 Protokol Ademco Contact ID prostřednictvím TCP

Velké rozšíření protokolu Ademco Contact ID (popsaného v kapitole 2.1) vedlo k jeho IP modifikaci. Tento protokol je dán normou DC-09 americké společnosti SIA [3] z roku 2012. Pro přenos zprávy se užívá jak protokol TCP, tak i protokol UDP.

2.2.1 Požadavky na přenosovou trasu

Jako přenosové médium se používají klasické média pro přenos IP paketů (tj. Ethernet, 802.11x, GPRS, CDMA). Požadavky na přenosovou síť v ČR udává norma ČSN EN 50136-1-5 [5]. Dle této normy nemusí být síťová zařízení sekundárně napájena pro případ výpadku napájecí sítě, protože PSN umožňuje velký datový tok a tím i kontrolní dohledové zprávy ve velmi krátkých intervalech (od 10 vteřin). Pokud zařízení EZS neodesílá kontrolní zprávy, tak se na PCO zobrazí poplašná zpráva, která indikuje ztrátu spojení.

Norma ČSN EN 50136-1-5 [5] ještě udává požadavky na výkonnost PSN. Pokud je PSN sdílená i s jinými zařízeními, tak musí být zajištěno, aby v žádném případě nedošlo ke ztrátě paketu vlivem plného využití kapacity PSN. V praxi se jedná o správné nastavení priority paketů pomocí služby QoS tak, že pakety z poplachového zařízení budou mít nejvyšší prioritu.

PCO (popř. jiné přijímací zařízení) musí mít statickou veřejnou IP adresu, na kterou se bude vysílací zařízení (např. EZS) připojovat. Vysílací zařízení nemusí mít statickou ani veřejnou adresu. Spojení je zde typu klient-server.

2.2.2 Přenos zprávy události

Při přenosu zprávy protokolem TCP se nejdříve vytvoří spojení s přijímacím zařízením (PCO) a hned poté se odešle samotná zpráva. Po přijetí a zpracování zprávy odešle přijímací zařízení zpět potvrzovací zprávu. Pokud se jedná o kladné potvrzení (vše proběhlo v pořádku), tak vysílací zařízení spojení ukončí. V případě negativního potvrzení se přenos zprávy opakuje.

Přenášená zpráva je ve formátu:
<LF><crc><0LLL><"id"><seq><Rrcvr><Lpref>< #acct> [<pad> |...data...] [x...data...] <timestamp><CR>

LF – ASCII znak "linefeed" označující počátek zprávy. V ASCII tabulce má hodnotu 0A v hexadecimálním tvaru.

crc – Cyklický redundantní kód. Slouží jako základní zabezpečení proti chybám při přenosu zprávy. Výpočet se provádí od <"id"> před začátek <CR>. Výpočet je dán normou SIA DC-07 [2] a výsledek se přenáší jako 4 hexadecimální znaky.

0LLL – Tato část začíná vždy znakem *0* a přenáší informace o délce zprávy. Výpočet se provádí stejně jako crc od <"id"> před začátek <CR> a přenáší se jako 3 hexadecimální znaky.

"id" – Typ přenášeného protokolu. V případě nezašifrované zprávy Ademco Contact ID se zde bude přenášet "ADM-CID", v případě šifrované zprávy se zde bude přenášet "*ADM-CID".

seq – V této části se přenáší pořadové 4-místné číslo sekvence. Počáteční sekvence má pořadové číslo `0001`. Po sekvenci `9999` se přenáší opět sekvence `0001`. Při opakování stejné zprávy (např. z důvodu chyby) se pořadí neinkrementuje.

Rrcvr – Tato část přenáší číslo přijímacího zařízení pro případ, že na stejné adrese je více přijímacích zařízení. Tento prvek je nepovinný, začíná vždy znakem `R` a má délku 1 až 6 hexadecimálních znaků.

Lpref – Zde se přenáší prefix čísla objektu. Ten slouží jako rozšířené informace o objektu. Tato část je povinná a začíná znakem `L`. Prefix může být dlouhý opět 1 až 6 hexadecimálních znaků a v případě, že ho nechceme použít, tak přeneseme pouze `L0`.

acct – Číslo objektu, které je shodné s číslem objektu popisovaném v kapitole 2.1.1.

pad – V případě nešifrované zprávy se zde přenáší opět číslo objektu včetně `#`. V případě šifrované zprávy se zde přenáší pseudonáhodná data, kde jednotlivé byte budou mít hodnotu 0 až 255 s výjimkou hodnot, které v ASCII tabulce reprezentují znaky | (124, x7C), [(91, x5B)] a (93, x5). Výsledný počet pad bytů musí být takový, aby mezi počátečním znakem `[` a posledním znakem před <CR> byl násobek 16-ti. V případě, že je již násobkem 16-ti před přidáním pad bytů, tak se musí přidat přesně dalších 16 pad bytů.

data – Data protokolu Ademco Contact ID (popsaného v kapitole 2.1.1) ve tvaru `QXYZ GG CCC`.

x.data – Pomocné dodatkové informace z vysílacího zařízení. Formát dat je popsán v normě SIA DC-09 [3]. Součástí těchto pomocných dat může být informace o názvu objektu, nadmořské výšce, názvu místnosti nebo MAC adresa. Tato část zprávy je nepovinná.

timestamp – Čas a datum přidání události do fronty. U nešifrovaných zpráv je tato část nepovinná, v případě šifrovaných zpráv je tato část povinná. V případě, že se čas přijímacího zařízení a této části výrazně liší (+20/-40 vteřin), tak na vysílací zařízení je odesláno negativní potvrzení. Toto slouží jako ochrana proti opakujícím se stejným zprávám. Tato část má tvar `_HH:MM:SS,MM-DD-YYYY`.

CR – ASCII znak "carriage return" označující konec zprávy. V ASCII tabulce má hodnotu 0D v hexadecimálním tvaru.

Příklad výsledné zprávy může vypadat takto:
<LF>87CD0035<"ADM-CID">9876R579L789ABC#1234 [#1234|1110 00 129] <CR>
Z této zprávy lze zjistit, že se jedná o zprávu protokolu Ademco Contact ID s pořadovým číslem 9876. Číslo přijímacího zařízení je 579 a vysílací zařízení má prefix 789ABC. Číslo objektu je 1234 a byl zde vyhlášen požární poplach v zóně 129.

2.2.3 Dohledové zprávy

Volitelně mohou mít přijímací a vysílací zařízení nastaveno periodické hlídání spojení. Při těchto dohledových zprávách vysílá vysílací zařízení (např. EZS) v nastaveném intervalu tzv. "Null Message". Pokud přijímací zařízení (např. PCO) nepřijme po nastavenou dobu žádnou dohledovou zprávu, tak na monitoru PCO se zobrazí poplašná zpráva, indikující ztrátu spojení.

Interval odesílání dohledových zpráv z vysílacího zařízení může být odlišný od nastaveného intervalu na přijímací straně, ale nikdy nesmí být na přijímacím zařízení nastaven kratší interval než na vysílacím zařízení. Této nesymetrie se využívá např. pokud chceme ignorovat jeden výpadek spojení. Interval může být nastaven (na obou zařízení) v rozsahu 10 až 3600 vteřin nebo 1 až 1080 hodin. V praxi se doporučují kratší intervaly, pokud to technické možnosti PSN dovolují.

V případě nešifrovaného přenosu bude mít zpráva tvar:
<LF><crc><0LLL><"NULL">0000<Rrcvr><Lpref>< #acct> [] <timestamp><CR>
U dohledových zpráv se u **id** odesílá "NULL" jako indikace, že se jedná o prázdnou zprávu. Číslo sekvence je vždy 0000 a neprovádí se inkrementace. Zbytek zprávy je stejný jako u přenosu popisovaného v kapitole 2.2.2.

V případě šifrovaného přenosu má zpráva tvar:
<LF><crc><0LLL><"*NULL">0000<Rrcvr><Lpref>< #acct> [<pad>]
<timestamp><CR>
Zde oproti nešifrované zprávě je rozdíl v id, kde je hvězdička, a také je zde povinná část <pad>, která má stejné vlastnosti jako v kapitole 2.2.2. Část <timestamp> je v šifrované zprávě povinná.

2.2.4 Potvrzovací zprávy

Po každé přijaté zprávě, která přenáší událost, musí přijímací zařízení odeslat zpět na vysílací zařízení potvrzovací zprávu. Existují celkem 4 typy potvrzovacích zpráv. První typ je tzv. pozitivní potvrzení. Tato zpráva se odesílá pouze v případě, že je vše v naprostém pořádku, tj. souhlasí CRC, délka zprávy, formát zprávy a časový údaj na konci zprávy. Pozitivní potvrzení se může odesílat v nešifrované, ale i v šifrované podobě. Nešifrovaná zpráva má tvar:
<LF ><crc><0LLL><"ACK"><seq><Rrcvr><Lpref>< #acct> [] <CR>
Šifrovaná zpráva je ve tvaru:
<LF><crc><0LLL><"*ACK"><seq><Rrcvr><Lpref>< #acct> [<pad>]
<timestamp><CR>.

Další je tzv. negativní potvrzení. Tato zpráva se přenáší, pokud např. nesouhlasí crc, číslo sekvence nebo je rozdílný <timestamp>. Negativní potvrzení se posílá pouze v nešifrované podobě a má tvar:
<LF><crc><0LLL><"NAK">0000R0L0A0[] <timestamp><CR>

Část <timestamp> se přenáší pokaždé, pro případné srovnání času přijímacího a vysílacího zařízení.

Třetí typ je potvrzení, že zpráva byla přijatá správně, ale nelze ji aktuálně zpracovat. Zpráva bude mít tvar:
<LF><crc><0LLL><"DUH"><seq><Rrcvr><Lpref>< #acct> [] <CR>
Toto potvrzení se nikdy neposílá v šifrované podobě.

Poslední typ není plně potvrzovací zpráva, ale přenos dat z přijímacího zařízení na vysílací zařízení. V normě SIA DC-09 [3] je zatím definováno pro budoucí použití a má tvar:
<LF><crc><0LLL><"RSP"><seq><Rrcvr><Lpref>< #acct> [...data...] <CR>

2.2.5 Šifrování přenosu

Zprávy v protokolu Ademco Contact ID mohou být zašifrovány pomocí šifrování AES s využitím metody CBC (Cipher Block Chaining), která je popsána v sekci 6.2 publikace NIST Special Publication 800-38A (vydání z roku 2001) [4]. Velikost šifrovacího klíče je 128b, 192b nebo 256b.

Na obrázku 2.3 je vidět princip metody Cipher Block Chaining. Na začátku celé operace se provede exkluzivní součet vstupního nešifrovaného textu a předchozího šifrovaného textu (v případě prvního přenosu se užívá tzv. inicializační vektor). Následuje Cipher Block, který provede zašifrování pomocí šifrovacího klíče.

Dešifrování se provádí přesně opačnou metodou. Nejdříve se provede dešifrování v invertovaném Cipher Blocku pomocí šifrovacího klíče a poté následuje exkluzivní součet předcházejícího zašifrovaného textu a s výsledkem dešifrovaného textu z invertovaného Cipher Blocku.

Inicializační vektor nemusí být tajný, ale nesmí být snadno predikovatelný. Tento vektor musí být znám na vysílací i přijímací straně stejně jako šifrovací klíč.

U zprávy Ademco Contact ID se nešifruje celá zpráva, ale pouze část od prvního znaku [do posledního znaku před $< CR >$. Nezašifrovaná zpráva, která má tvar:
<LF>B3680040 <"ADM-CID">0001L000000#1234 [#1234|1140 00 007]
_22:49:34,01-22-2012<CR>
bude v šifrované podobě vypadat:
<LF>4B89007B <"*ADM-CID">0001L000000#1234 [371baac130fe81508f556e6fd2ccfd
8826e9ba186f0fb674bb87c079484e546dff35532aa285936a00c27b6feb053f68<CR>

2.2.6 Chyby v přenosu

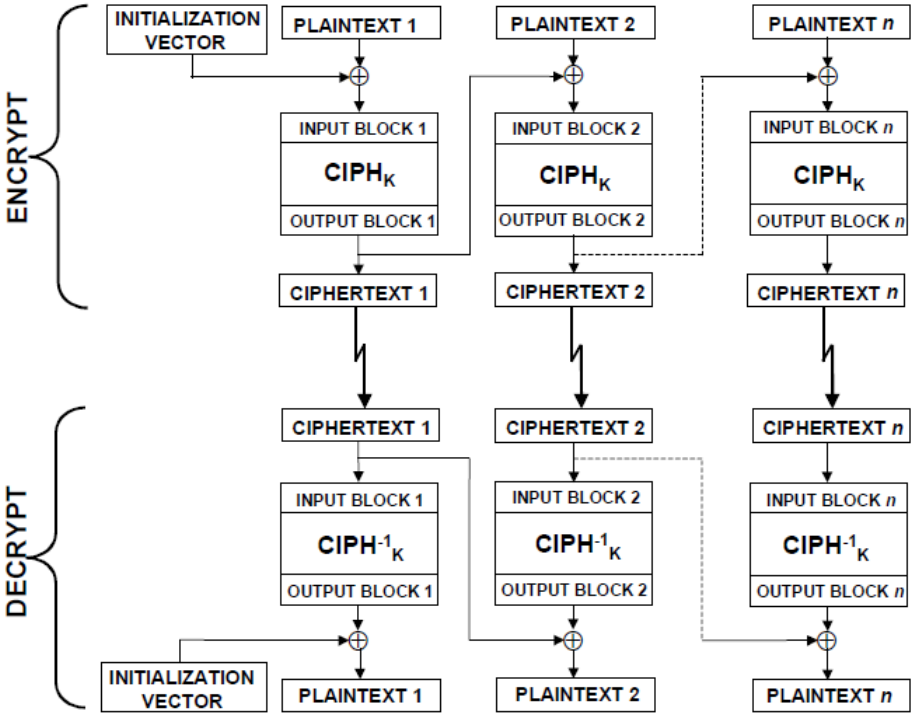
Po přenosu zprávy na přijímací zařízení se může stát, že přijímací zařízení neodpoví v časovém limitu. V tomto případě se po uplynutí nastavitelného intervalu odešle zpráva znovu. Tento interval bývá 5 až 60 vteřin. Počet pokusů o přenos zprávy může být nastavitelný

CBC Encryption:

$$C_1 = CIPH_K(P_1 \oplus IV);$$
$$C_j = CIPH_K(P_j \oplus C_{j-1}) \quad \text{for } j = 2 \ldots n.$$

CBC Decryption:

$$P_1 = CIPH^{-1}_K(C_1) \oplus IV;$$
$$P_j = CIPH^{-1}_K(C_j) \oplus C_{j-1} \quad \text{for } j = 2 \ldots n.$$



Obr. 2.3: Popis metody Cipher Block Chaining (převzato z [4])

a doporučený počet jsou 3 pokusy. V případě, že se přenos nepovede, tak vysílací zařízení indikuje chybu.

V případě, že přijímací zařízení odešle negativní potvrzení, tak vysílací zařízení nejprve aktualizuje svůj čas ze zprávy o potvrzení a poté se pokusí odeslat opětovně zprávu. V tomto případě není mezi zprávami žádný interval. V případě, že se přenos nepovede, tak vysílací zařízení indikuje chybu.

Další možnou odpovědí od přijímacího zařízení je, že zpráva byla správně přijata, ale momentálně nemůže být zpracována. V tomto případě se vysílací zařízení pokusí okamžitě odeslat stejnou zprávu. V případě, že se přenos nepovede, tak vysílací zařízení indikuje chybu.

2.3 Alternativní přenos protokolu Contact ID prostřednictvím TCP

Kromě standardizovaného přenosu protokolu Ademco Contact ID po TCP (popsaného v kapitole 2.2) existuje ještě alternativní přenos, který je užíván u některých zařízeních PCO. Princip je takový, že z vysílacího zařízení na přijímací zařízení se odesílá nešifrovaný text s původní zprávou Ademco Contact ID z telefonní linky dle normy SIA DC-05 [1] (popsaném v kapitole 2.1.1).

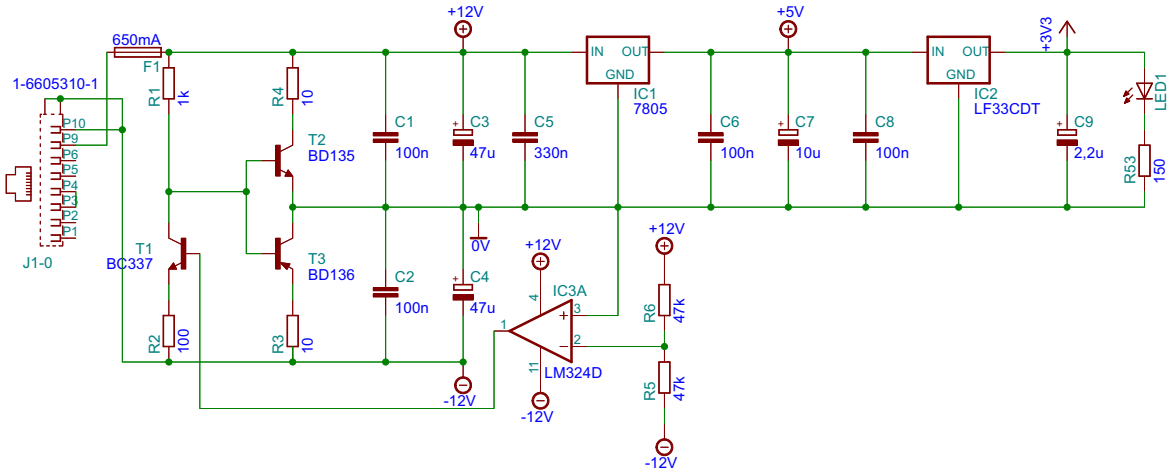
Jako potvrzení, že přenos proběhl v pořádku, přijímací zařízení odešle zpět na vysílací zařízení stejnou zprávu.

3

Popis bloků převodníku

3.1 Blok napájení

Napájení převodníku je zajištěno prostřednictvím PoE s napětím 24V. Celkový odběr převodníku by za standardních podmínek neměl přesáhnout 500mA, ale pro zajištění případně větších proudových odběrů byl vybrán napájecí zdroj PoE 24V/1A. Způsob napájení pomocí PoE byl vybrán proto, aby se snížil počet přívodních kabelů do převodníků a zjednodušila se montáž zařízení. Schéma napájecí části je na obrázku 3.1.



Obr. 3.1: Napájecí blok převodníku

Za vstupním konektorem RJ45 se nachází pojistka F1 o velikost 650mA. Tato pojistka slouží jako mezní proudová ochrana, která chrání hlavně přívodní Ethernetový kabel a napájecí zdroj. Za touto pojistkou je obvod pro převod nesymetrického napětí 24V na symetrické napětí 12V. Tento obvod je tvořen tranzistory T1 až T3 a rezistory R1 až R4. Řízení zajišťuje operační zesilovač LM324D, který má na svém invertujícím vstupu odporový dělič hlídající poloviční napětí. Toto napětí je porovnáváno s vytvořenou virtuální nulou na neinvertujícím vstupu. Výstupní napětí OZ řídí tranzistor T1, který reguluje bá-

zové napětí tranzistorů T2 a T3. Tranzistory T2 a T3 jsou otevřené stejně tak, aby tvořili dvě stejná napětí proti virtuální nule. Maximální proud z výstupu operačního zesilovače přes tranzistor T1 je omezen rezistorem R2.

Ztrátový výkon tranzistoru T1 při maximálním odběru převodníku je do 610mW, tranzistor T2 má ztrátový výkon pod 1mW, tj. není třeba ani chladič. Tranzistor T3 je již třeba chladit, protože jeho ztrátový výkon dosahuje až 3,3W. Chladič byl vypočítáván pro vnější teplotu 45°C a maximální teplotu přechodu tranzistoru 150°C. Při těchto hodnotách bylo zjištěno, že je nutné mít chladič s maximální tepelnou rezistivitou 28,8K/W. Při tvorbě převodníku byla však přidána rezerva a chladič použitý na převodníku má tepelnou rezistivitu 21K/W. Rezistory R1 a R4 mají maximální ztrátový výkon do 0,3W, rezistory R2 a R3 mají ztrátový výkon až 1,5W. Výpočet maximální tepelné rezistivity chladiče je vidět v rovnici 3.1, kde T_j je maximální teplota tranzistoru, K_1 je tepelná rezistivita pouzdra tranzistoru, P je maximální ztrátový výkon a ν_a je teplota okolí.

$$K_2 = \frac{(T_j - K_1 P) - \nu_a}{P} \tag{3.1}$$

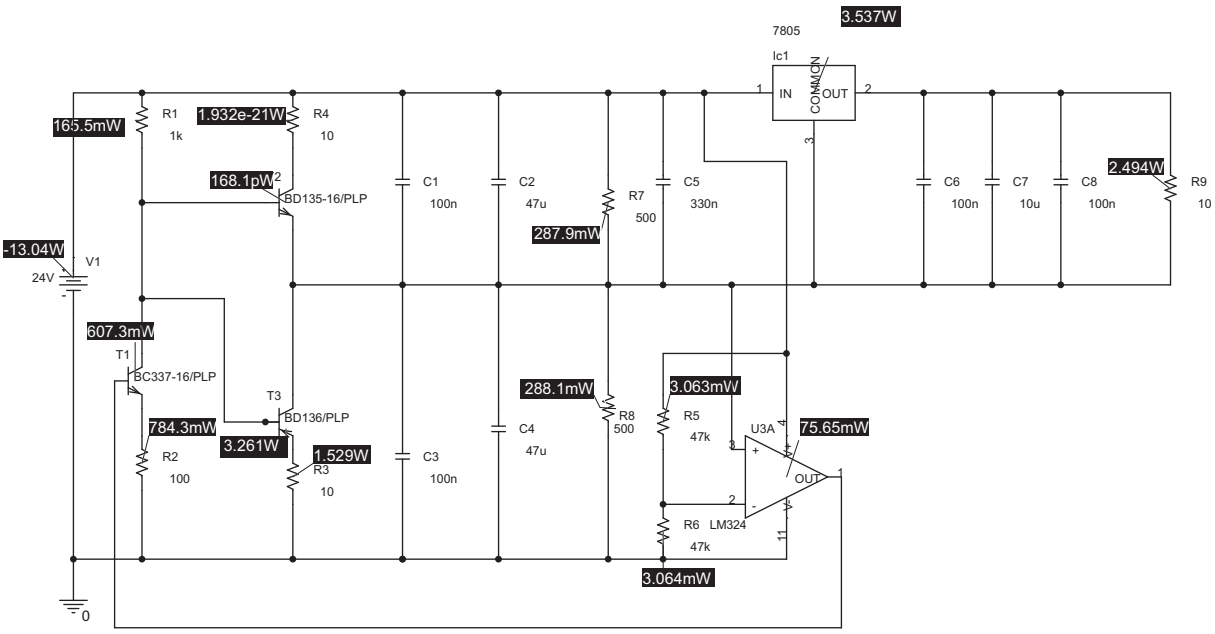
Z napětí 12V je dále tvořeno napětí 5V, které je použito pro napájení IO MT8870DS. Napětí 5V je tvořeno pomocí lineárního stabilizátoru 7805, který má maximální proudové zatížení 1A. V této aplikaci bude však zatížen maximálně proudovým odběrem 400mA a v průměru jen 300mA. I přes to, ale ztrátový výkon je až 3,6W (v průměru 2,5W) a IO musí být chlazen. Pro výpočet byly použity opět hodnoty 45°C vnějšího prostředí a maximální teplota IO 150°C a hodnota tepelné rezistivity chladiče vychází 44K/W. I v tomto případě byla vytvořena rezerva a byl použit chladič s tepelnou rezistivitou 29K/W.

Na závěr je vytvořeno napětí 3,3V pro napájení digitální části převodníku. Toto napětí je získáno pomocí IO LF33CDT, který je napájen napětím 5V. Maximální proudová zátěž napěťové hladiny 3,3V je 300mA a v průměru 250mA. IO LF33CDT má ztrátový výkon 0,5W a není ho třeba chladit pomocí chladiče.

Na obrázku 3.2 jsou odsimulované ztrátové výkony v napájecí části z programu PSpice. V tomto obrázku rezistory R7 a R8 nahrazují ztrátový výkon na telefonní lince a operačního zesilovače. Rezistor R9 slouží jako náhrada za ztrátový výkon všech součástek v napěťové hladině 5V a 3,3V. Ztrátový výkon integrovaného obvodu LF33CDT nebyl odsimulován kvůli chybějící knihovně s touto součástkou.

3.2 Analogová část

První část analogového bloku zajišťuje napájení telefonní linky a její impedanční přizpůsobení. Další část zajišťuje oddělení příchozího a odchozího signálu pomocí aktivní telefonní vidlice. Tato část také zajišťuje zesílení signálu. Poslední část analogové části slouží k přijmu DTMF s využitím integrovaného obvodu MT8870DS.



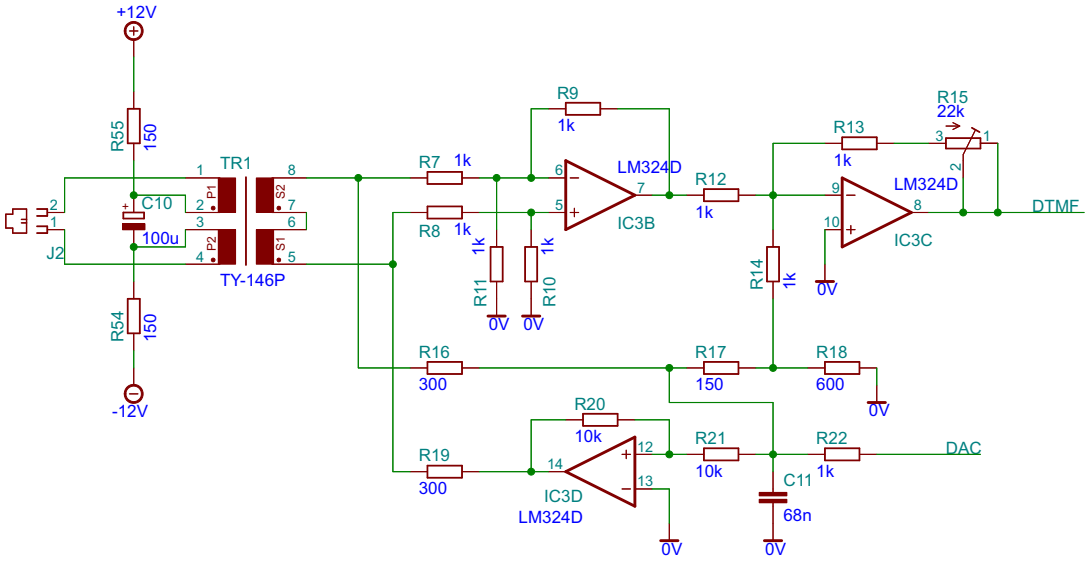
Obr. 3.2: Simulace ztrátových výkonů v napájecí části

Schéma první a druhé analogové části je na obrázku 3.3. Konektor J2 je výstupní konektor typu RJ11 pro telefonní linku směřující k ústředně EZS. Telefonní linka je napájena pomocí fantomového zapojení. C10 je připojen na symetrické napětí 12V přes rezistory R54 a R55. Tyto rezistory zajišťují proudovou ochranu v případě zkratu na telefonní lince. Maximální zkratový proud telefonní linky je 80mA. Přepěťová ochrana na telefonní lince není řešena z důvodu, že zařízení musí být ve vnitřních prostorech s velmi krátkým propojem s ústřednou EZS (v řádech metrů).

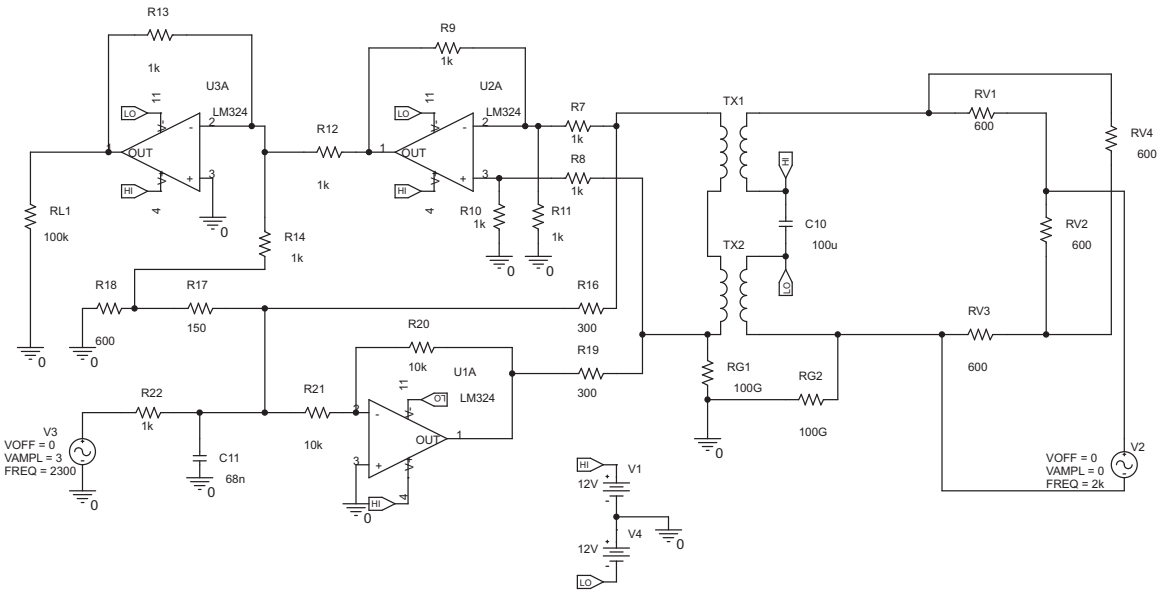
Oddělení příchozího a odchozího směru signálu je provedeno aktivní vidlicí s operačním zesilovačem LM324D. Rezistor R18 zde slouží jako vyvažovací impedance pro telefonní linku, rezistory R12, R13 a R15 určují zesílení příchozího směru signálu. Výpočet zesílení se počítá stejně jako u neinvertujícího zesilovače. Stejným způsobem se nastavuje zesílení na odchozím směru pomocí rezistorů R20 a R21.

Rezistor R22 a kondenzátor C11 slouží jako dolní propust za D/A převodníkem, který má zlomový kmitočet 2340Hz a má strmost 20dB/dek. V oblasti 33,2kHz (nastavený vzorkovací kmitočet D/A převodníku) je útlum dolní propusti 22,5dB. Při nejvyšším kmitočtu 2300Hz a vzorkovací frekvenci 33,2kHz vzniká další produkt vzorkování na frekvenci 30,9kHz. Tato harmonická frekvence původního signálu má na tomto signálu útlum 23dB. Ve spojení s dolní propustí je útlum této harmonické téměř 45,5dB. D/A převodník mikrokontroléru LPC2368 má nejnižší výstupní zatěžovací impedanci 1kΩ, proto rezistor R22 má právě tuto hodnotu.

První a druhá část analogového bloku obvodu byla simulována v aplikaci PSpice, kde bylo použito schéma z obrázku 3.4. Cílem této simulace bylo zjištění útlumu z vysílacího směru do přijímacího směru obvodu. Výstupní napětí bylo generováno v rozsahu 300 až 3400Hz s amplitudou 3V. Vstupní napětí bylo měřeno na rezistoru RL, který simuluje

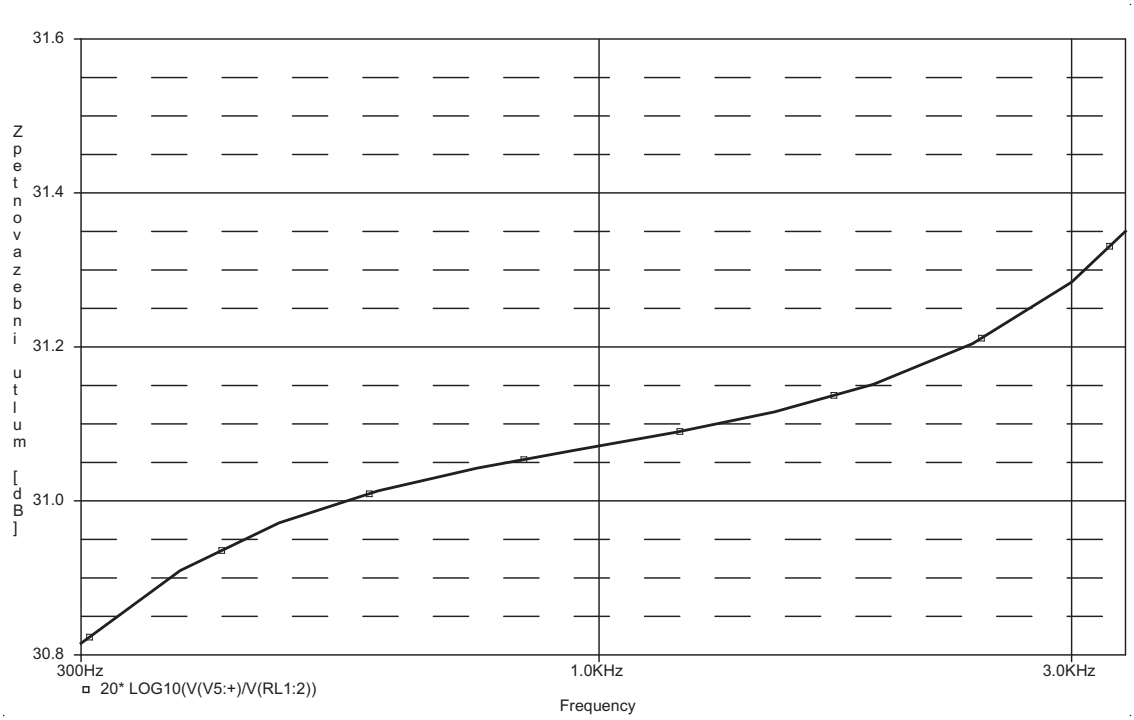


Obr. 3.3: Analogový blok převodníku



Obr. 3.4: Obvod pro simulaci útlumu zpětnovazební smyčky aktivní telefonní vidlice

vstupní impedanci DTMF přijímače. Výsledný graf příčného útlumu je na obrázku 3.5.



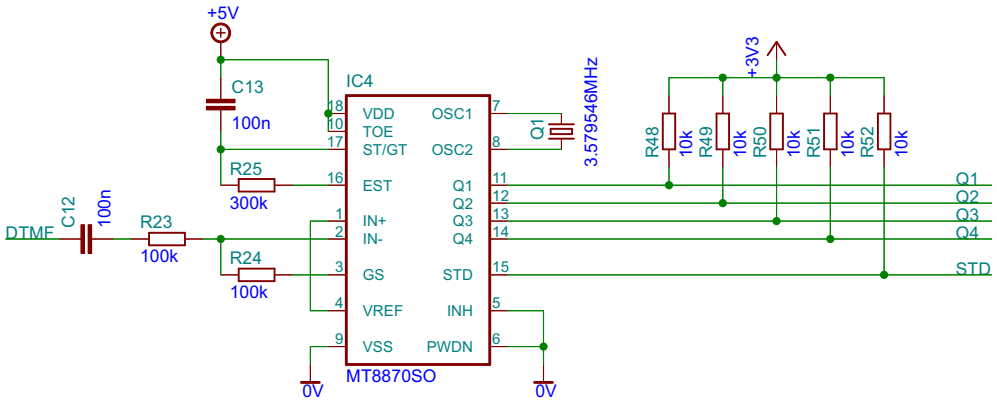
Obr. 3.5: Frekvenční závislost útlumu mezi vysílacím a přijímacím směrem aktivní vidlice

Z grafu lze vyčíst, že v telefonním pásmu je příčný útlum aktivní vidlice 31dB. V praktickém zapojení bylo změřeno, že tato hodnota je nižší o 4dB z důvodu tolerancí součástek.

Schéma bloku detekce DTMF je na obrázku 3.6. Detekce DTMF je provedena pomocí integrovaného obvodu MT8870DS, který analogový signál přijímá na invertujícím vstupu přes vazební kondenzátor C12. Rezistory R23 a R24 určují zesílení signálu. Rezistor R25 a kondenzátor C13 určují dekódovací parametry IO. Hodnota R25 a C13 je převzata z doporučení výrobce. Vstup TOE slouží k aktivaci výstupu, vstup INH určuje, zda se mají dekódovat i DTMF znaky A, B, C, D (aktivace je při nízké úrovni). Vstup PWDN slouží k přepnutí do úsporného režimu, při kterém přestane obvod pracovat. Takt vnitřních logických obvodů je dán krystalem Q1, který má rezonanční kmitočet 3,579546MHz. Výstupy z tohoto dekodéru jsou na pinech Q1 až Q4, které jsou vedeny do mikrokontroléru na vstupy P2.0 až P2.3. Informaci o tom, že IO MT8870DS úspěšně dekódoval DTMF je na pinu STD, tento výstup je veden na pin P2.11 (EINT1) mikrokontroléru.

3.3 Digitální část

Schéma digitální části je na obrázku 3.7. Celý systém je řízen mikrokontrolérem LPC2368 od firmy NXP (dříve Philips). Tento mikrokontrolér je 32bitový s jádrem ARM7. Velikost flash paměti je 512kB, velikost RAM paměti 32kB s vyhrazeným 16kB prostorem



Obr. 3.6: Blok detekce DTMF

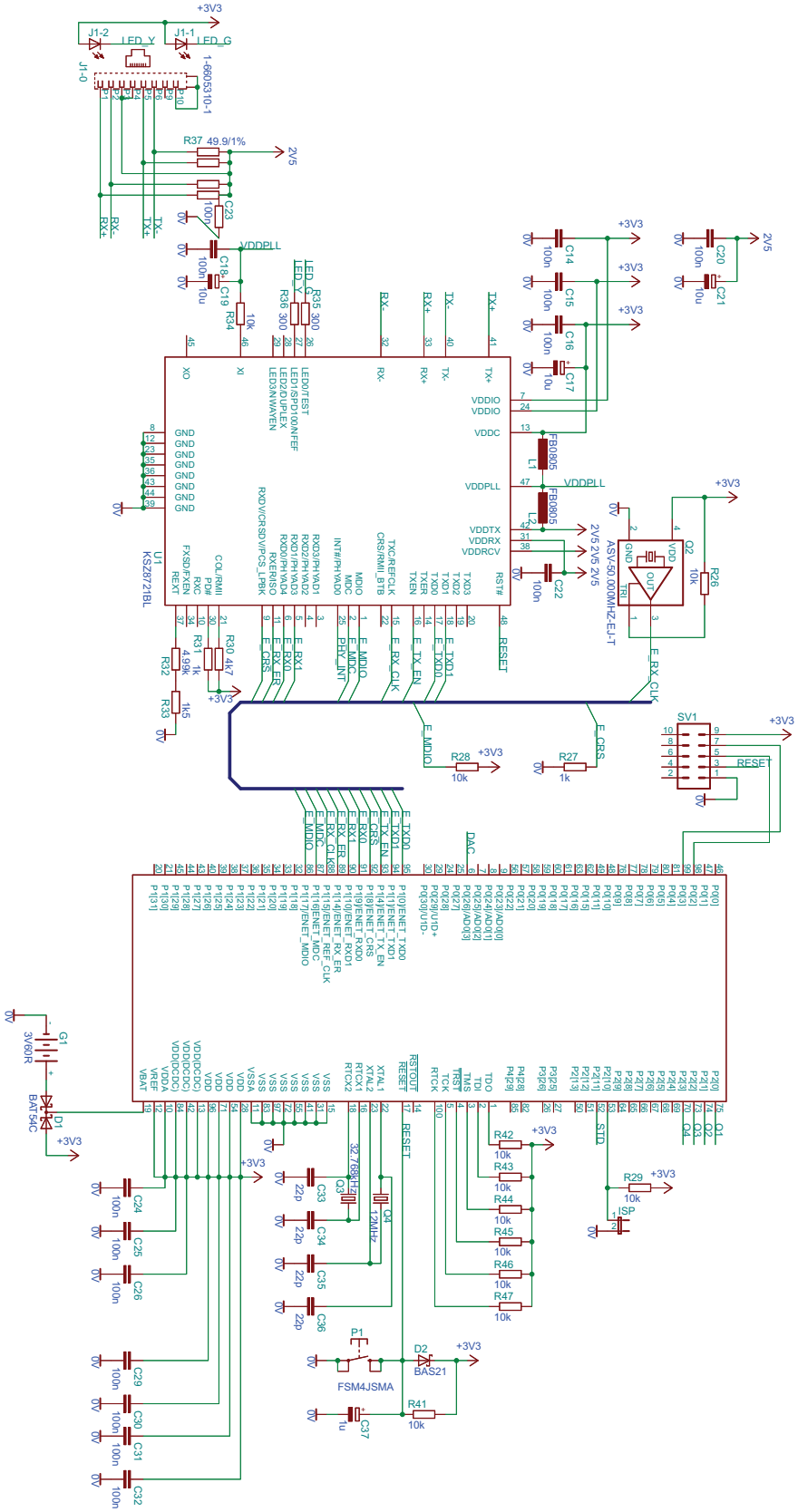
pro Ethernet. MCU má 4 časovače, 6 A/D převodníků, D/A převodník, sběrnice CAN, UART, I2C, SPI a USB. LPC2368 má podporu Ethernetu na vrstvě MAC s podporou spojení RMII i MII s fyzickou vrstvou. Maximální taktovací kmitočet je 72MHz.

Napájení MCU je napětím 3,3V, které je přivedeno na piny VDD a VDD(DCDC). U těchto pinů jsou blokovací kondenzátory (C24 až C32) o velikosti 100nF. MCU je uzemněn přes piny VSS. Odběr MCU je při taktovacím kmitočtu 12MHz maximálně 30mA. Napájení RTC v době, kdy je převodník bez napájecího napětí, je zajištěno prostřednictvím NiMH baterie o napětí 3,6V. Aby baterie nebyla vybíjena v době, kdy je převodník napájen, tak je vstup VBAT přes dvojitou diodu D1 (BAT54C) napájen z napěťové hladiny 3,3V. Odběr z baterie je maximálně 80μA.

Taktovací kmitočet MCU v této aplikaci je 12MHz, který postačuje pro správnou funkčnost převodníku. Takt je tvořen pomocí krystalového rezonátoru Q4, který má na výstupech kondenzátory C35 a C36. Hodnota 22pF je doporučená hodnota z User Manualu [6] v tabulce 33 v sekci 4.2.2. Krystalový rezonátor Q3 má rezonanční kmitočet 32,768kHz, který slouží pro RTC.

Reset mikrokontroléru a fyzické vrstvy Ethernetu je provedeno zapojením daným z User Manualu mikrokontroléru [6]. Reset je aktivní při logické "0" na pinu RESET na mikrokontroléru i na fyzické vrstvě Ethernetu, tj. při napětí 0V, a impuls musí být delší než 3ms. Uzemnění pinu je provedeno tlačítkem P1 a při neaktivním stavu je napětí 3,3V zajištěno rezistorem R41. Reset lze provést i z programátoru přes konektor SV1.

Vstupní a výstupní piny rozhraní JTAG jsou přes rezistory R42 až R47 s hodnotou 10kΩ připojeny na napětí 3,3V, čímž se toto rozhraní deaktivuje. Pro programování se v této práci používá rozhraní UART0 a případný debug se provádí pomocí ostatních periférií MCU (GPIO, text přes UART či Ethernet, apod.). Programátor, který bude popsán v kapitole 3.5, se připojuje k převodníku přes konektor SV1, který zajišťuje pro programá-



Obr. 3.7: Schéma digitální části

tor napájecí napětí 3,3V, odchozí a příchozí směr sběrnice UART0 a možnost externího softwarového resetu. MCU se v této aplikaci programuje v režimu In-System Programming (ISP), který se aktivuje pomocí pinu P2.10. Pro aktivaci ISP režimu musí být na tento pin přivedena logická "0". V případě přivedení logické "1" na tento pin při startu či resetu se spustí program nahraný ve flash paměti.

Fyzickou vrstvu Ethernetu zajišťuje integrovaný obvod KSZ8721BL od společnosti Micrel. Tato fyzická vrstva podporuje 100BASE-TX i 10BASE-T s half duplexem či full duplexem. Spojení s mikrokontrolérem je umožněno přes RMII i přes MII. Integrovaný obvod KSZ8721BL má vnitřní zdroj napětí 2,5V, které se využívá na Ethernetu, a není tedy potřeba externího zdroje tohoto napětí. Tento IO je napájen napětím 3,3V s maximálním odběrem 150mA.

Propojení MCU s fyzickou vrstvou Ethernetu je zde řešeno přes RMII rozhraní. Piny TXD0 a TXD1 slouží pro odchozí směr přenosu dat, pomocí pinu TX˙EN se povoluje odchozí směr přenosu dat, piny RXD0 a RXD1 jsou pro příchozí směr přenosu dat, pin RX˙ER je pro hlášení z fyzické vrstvy do MCU o chybách při příjmu dat. Pin CRS slouží jako kontrola, zda příchozí data jsou v pořádku. Piny CRS jsou uzemněné přes 1kΩ rezistor, při startu se zde nastavuje hodnota pro Loopback Mode. V případě, že při startu nebo resetu je na tomto pinu logická "0", tak se aktivuje normální přenos. V případě logické "1" je přenos ve smyčce, která se používá pro případné testování. Taktovací kmitočet příchozího toku dat je zajištěn krystalovým oscilátorem Q2 s nominální hodnotou frekvence 50MHz a tento takt je přiveden na piny RX˙CLK. Piny MDIO a MDC slouží pro řízení přenosu dat mezi MCU a fyzickou vrstvou Ethernetu.

Na fyzické vrstvě se dále nastavuje pomocí pinu COL/RMII, zda je MCU připojeno rozhraním RMII nebo MII. Pinem PD# se fyzická vrstva aktivuje nebo deaktivuje pro případnou úsporu elektrické energie. Piny TX+, TX-, RX+ a RX- slouží pro datové spojení s konektorem RJ45. Všechny piny jsou přes rezistory R37 až R40 s hodnotou 49,9Ω připojeny na napětí 2,5V. Kontrolní LED diody na konektoru RJ45 se připojují na piny LED0 a LED1. Na tyto piny se připojují katody těchto diod, aby se docílilo toho, že LED budou trvale svítit, pokud bude přenos dat neaktivní, a blikat při přenosu dat.

3.4 Návrh a výroba desky plošného spoje

Návrh DPS byl proveden v aplikaci Eagle verze 6.1 od společnosti CadSoft. Vzhled k použití MCU s pouzdrem LQFP100 bylo nutné použít nejmenší rozměr vodivých cest 0,254mm se stejnou izolační mezerou mezi vodiči. Největší rozměry vodivých cest jsou v napájecí části, kde dosahují až 0,8mm. Rozměr 0,8mm postačuje proudovému odběru 500mA s tím, že se teplota spoje zvýší o méně než 10°C. Izolační mezera zemní plochy je 0,8mm. Při návrhu bylo nutné přistoupit také k použití prokovených děr, kterých je celkem 64. Výsledný návrh DPS je v příloze A.1.

DPS je oboustranná, vyrobena z materiálu FR4 s tloušťkou materiálu 1,5mm a s

CU fólií o tloušťce 18μm. Celkový rozměr DPS je 114mm x 71mm s nepájivou zelenou maskou s jednostranným potiskem a povrchovou úpravou HASL. Výrobu prototypové desky zajistila společnost Pragoboard za 550Kč vč. DPH. Při výrobě 100 kusů DPS je cena již 100,1Kč za kus vč. DPH.

Zapájení součástek bylo u prototypu provedeno ručně na katedře aplikované elektroniky a telekomunikací, proto cena pájení byla v tomto případě nulová. V případě strojního pájení součástek na DP je cena od 300Kč za kus.

V převodníku je celkem 37 kondenzátorů, převážně v SMD provedení (jedinou výjimkou je kondenzátor C10 pro napájení telefonní linky) s pouzdry 0805 a 1206. Celková cena kondenzátorů je 72Kč při výrobě prototypu. Při sériové výrobě 100ks převodníku je cena 51Kč na jednu desku. Dále se na DPS nachází celkem 53 rezistorů za celkovou částku 141Kč (pro sérii 100ks je cena 81Kč na 1 desku). Rezistory jsou zde opět převážně SMD s pouzdry 0805 a 1206, ale nachází se zde i THT rezistory a to s pouzdry 0207 pro nevýkonné uhlíkové rezistory do 1W. Rezistory s výkonem nad 1W mají pouzdro 0411. Rezistor R32 je oproti ostatním rezistorům výrazně dražší vzhledem k jeho atypické hodnotě a velmi malé toleranci. Tento rezistor slouží k impedančnímu vyvážení fyzické vrstvy Ethernetu.

V obvodu se také nachází několik diskrétních polovodičů v celkové ceně 26Kč při výrobě prototypu a 19Kč při výrobě 100ks. Nejdražším integrovaným obvodem je mikrokontrolér, který při nákupu 1ks stojí 400Kč. Při nákupu 100ks je již cena znatelně nižší a to 286Kč za 1ks. Celkově jsou v převodníku integrované obvody za 674Kč při kusové výrobě. Při sériové výrobě 100ks je cena 440Kč. Ostatní součástky v obvodu stojí dohromady 541Kč (při sérii 100ks je to 381Kč). Z těchto součástek je nejdražší konektor RJ45. Celkový seznam součástek s jejich pořizovací cenou za 1ks je v příloze A.2.

Celková pořizovací cena součástek pro prototyp je 1 452Kč, v případě sériové výroby 100ks je cena součástek 971Kč. Kompletní cena prototypu (výroba DPS, pájení, součástky) je 2 302Kč. Při sérii 100ks je cena 1 371Kč za 1 převodník.

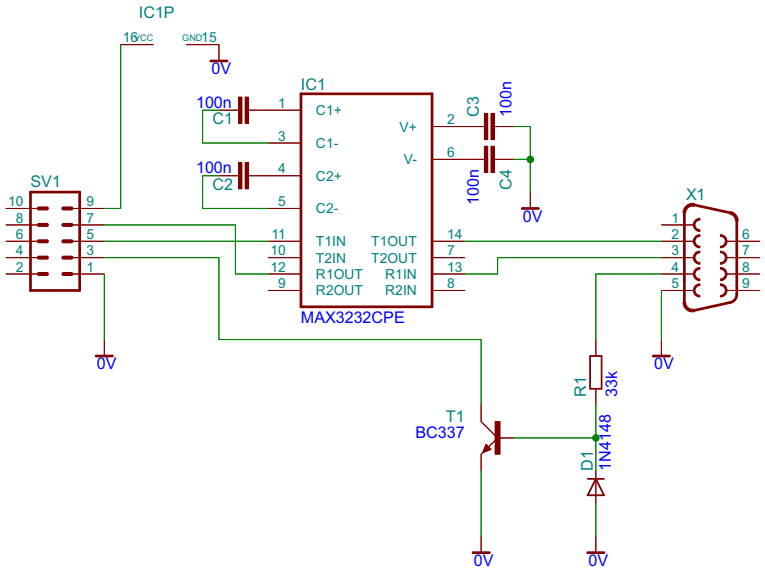
Položka	Cena ks - výroba 1ks	Cena ks - výroba 100ks
Výroba DPS	550,0 Kč	100,1 Kč
Součástky	1 452,0 Kč	971,0 Kč
Osazení DPS	300,0 Kč	300,0 Kč
Celková cena	2 302,0 Kč	1 371,1 Kč

Tab. 3.1: Souhrn ceny výroby 1ks a 100ks převodníku

3.5 Návrh a výroba progrmátoru mikrokontroléru

Pro programování mikrokontroléru byl navržen jednoduchý obvod, který je na obrázku 3.8. V tomto obvodu konvertuje integrovaný obvod MAX3232CPE data z RS232 na UART0

a opačně. Kondenzátory C1 až C4 slouží jako nábojová pumpa pro dosažení požadovaného napětí RS232, které je 12V. Tranzistor T1, který je ovládán pinem 4 konektoru X1, provádí RESET převodníku. Spojení s DPS převodníku je provedeno páskovým 10-ti žilovým vodičem, připojeným na konektor SV1 programátoru a SV1 převodníku. V reálném zapojení je každý vodič párován sousedním vodičem.



Obr. 3.8: Schéma programátoru

4

Programové vybavení mikrokontroléru

Firmware převodníku byl programován ve vývojovém prostředí aplikace Keil μVision verze 4 s podporou TCP stacku RL-TCPnet. Samotné programování bylo psáno v jazyce C a ke kompilování byl použit výchozí kompilátor Keilu. K nahrání hotového programu do FLASH paměti mikrokontroléru byl využit program Flash Magic, který je pro nekomerční účely zdarma.

Firmware v převodníku zpracovává příchozí data (již digitální) z DTMF přijímače MT8870DS a pomocí D/A převodníku odesílá odpovídající odpověď zpět na telefonní linku. Přijatá data jsou dále zkontrolována proti přenosovým chybám (CRC, korektnost zprávy k normě SIA DC-05 - popsán v kapitole 2.1). Zkontrolovaná data jsou zašifrována pomocí šifrování AES s metodou CBC (popsán v kapitole 2.2.5) a jsou odeslána na PCO přes TCP přenos, který je popsán v kapitole 2.2. Kromě těchto základních funkcí převodník umí odesílání kontrolních dohledových zpráv, odesílání e-mailových zpráv a možnost nastavení pomocí TELNETu.

4.1 Inicializace převodníku po startu

Po startu mikrokontroléru se spustí funkce main ze souboru main.c (v příloze B.1.1), kde se příkazem `init_setup()` (v příloze B.1.3) nejprve inicializuje FLASH paměť pro ukládání a načítání dat, které je potřeba uchovat i po restartu zařízení. Dále se provede inicializace RTC hodin a jejich případné nastavení při prvním startu. To se realizuje příkazem `init_RTC()` (v příloze B.1.3).

Jako další krok se provede načtení nastavení z FLASH paměti. V případě, že se jedná o start po resetu nastavení (zjištěno pomocí funkce `get_RESET_setup()`), tak se provede přepis hodnot do výchozího nastavení podle parametrů načtených ze souboru setup.h (v příloze B.1.6).

Po načtení nastavení z FLASH paměti se inicializuje RL-TCPnet stack. Nejprve se

nastaví parametry časovače `TIMER 0` (vyvolané příkazem `init()`) a poté se provede samotný start stacku příkazem `init_TCPnet()`. Tento příkaz je vnitřní funkce knihovny TCP_ARM_L.lib vytvořené společností Keil. Po spuštění RL-TCPnet se načte IP nastavení z FLASH paměti (příkaz `get_ip_setup()`).

Dále se provede inicializace D/A převodníku a časovačů potřebných pro generování tónů příkazem `init_tone()`. Poté příkaz `init_DTMF()` provede nastavení vstupů mikrokontroléru od DTMF přijímače MT8870DS, kde se nastaví i externí přerušení `EINT1`, a nastaví se parametry časovače pro správný chod přijmu dat z MT8870DS. Na závěr se provede inicializace odesílaní a přijímaní dat z TCP (příkaz `init_IPC()`).

Po startu všech periférií a načtení všech nastavení, se program dostane do smyčky, která neustále kontroluje stav časovače `TIMER 0` a provadí se dohled nad Ethernetem. V případě, že se přijímá zpráva z telefonní linky, tak se tato kontrola dočasně deaktivuje. Deaktivace je důležitá z důvodu využití časovačů mikrokontroléru, protože RL-TCPnet potřebuje pro svůj chod všechny dostupné časovače mikrokontroléru a příjem DTMF požaduje dva časovače pro svoji správnou činnost.

4.2 Nastavení a start MAC a PHY vrstvy Ethernetu

Při zadání příkazu `init_TCPnet()` se provede nastavení a start fyzické vrstvy Ethernetu v KSZ8721BL a MAC vrstvy v mikrokontroléru. RL-TCPnet spustí funkci `init_ethernet()` (soubor LPC23xx_EMAC.x - příloha B.1.2), která nejprve inicializuje vnitřní periférie mikrokontroléru. Po startu Ethernetové periférie se ověří typ procesoru, protože mikrokontrolér LPC2368 byl dodáván ve více revizích. Dle použité revize MCU se aktivují a nastaví vstupy a výstupy mikrokontroléru, které slouží pro přenos dat mezi fyzickou vrstvou Ethernetu (KSZ8721BL) a mikrokontrolérem.

Poté se provede reset a vymazání nastavení MAC vrstvy. Po krátké prodlevě, vytvořené `for` cyklem, se provede základní nastavení MAC vrstvy a vypne se dočasně příjem a odesílání dat, aby bylo zajištěno správně nastavení celého Ethernetu.

Po tomto základním nastavení MAC vrstvy se provede inicilaizace a nastavení fyzické vrstvy Ethernetu, prostřednictvím funkce `init_PHY()`. Tato funkce na začátku provede nastavení rozhraní RMII u MAC vrstvy (u fyzické vrstvy je to řešeno hardwarově, viz. kapitola 3.3). Po resetu MAC vrstvy se vyhledá na vstupech mikrokontroléru fyzická adresa PHY KSZ8721Bl a provede se její restart.

Po restartu se realizuje nastavení registrů PHY KSZ8721BL, kde se zapne režim auto-negotiation, který vybere nejlepší přenosové rychlosti a režimu přenosu. Po úspěšné aktivaci auto-negotiation režimu se provede načtení aktuálního nastavení PHY pro nastavení stejné přenosové rychlosti a přenosového režimu i u MAC vrstvy.

Po nastavení a startu fyzické vrstvy se uloží MAC adresa do registrů mikrokontroléru. Dále se provede uložení nastavení MAC vrstvy o rychlosti a přenosovém režimu. Jakmile jsou všechny základní registry MAC vrstvy nastavené, tak se nastaví DMA mikrokon-

troléru pro odesílání a příjem dat. Následuje nastavení filtrů paketů, aktivace přijmu a odeslání dat. Na závěr se nastaví a aktivuje interní přerušení MCU.

Přerušení je aktivováno vždy po příjmu nebo odeslání dat po Ethernetu. V případě příjmu dat se provede nejprve kontrola, zda je přijatý paket validní, tj. zda souhlasí délka paketu, CRC a hlavička. Jestliže jsou všechny tyto věci v paketu v pořádku, tak se data předají do RL-TCPnet stacku. Tento stack v převodníku přijímá, zpracovává a odesílá ARP, ICMP, IGMP. Dále UDP pakety pro DHCP a TCP pro Ademco Contact ID zprávy, TELNET a SMTP.

4.3 Příjem zprávy protokolu Contact ID z telefonní linky

Po startu převodníku se ve funkci main (kapitola 4.1) spustí příkaz `init_DTMF()`, který mimo jiné také aktivuje externí přerušení `EINT1`, na který je přiveden výstup STD z DTMF přijímače MT8870DS. Tento výstup se aktivuje vždy, když DTMF přijímač zaznamená validní DTMF tón.

Po aktivaci externího přerušení `EINT1` se načte symbol ze vstupů P2.0 až P2.3 mikrokontroléru a provede se jeho úprava podle převodní tabulky z datasheetu [9]. Po načtení a úpravě symbolu se podle aktuálního režimu, reprezentovaného proměnnou `DTMF_phase`, budou provádět další kroky popsané níže.

V případě, že proměnná `DTMF_phase` je rovna 0, tak se jedná o začátek vytáčení telefonního čísla převodníku. Je-li jako první symbol zaznamenán #, tak se jedná o ruční reset nastavení nebo restart zařízení a `DTMF_phase` se nastaví do režimu 9. Při jiné volbě se zvolí režim 1 a volba se zaznamená. Po uložení volby se aktivuje časovač, který hlídá rozestupy mezi DTMF symboly. Pokud časovač provede přerušení dříve, než se zaznamená další symbol, tak se proměnná `DTMF_phase` nastaví zpět do hodnoty 0 a uložená volba se vymaže.

Při zaznamenání symbolu v režimu 1 se provede uložení přijatého symbolu a kontrola, zda se již přijatá volba neshoduje s nastaveným telefonním číslem převodníku. Pokud tomu tak je, tak se změní proměnná `DTMF_phase` na číslo 2 a aktivuje se časovač, který vytvoří časovou prodlevu mezi dokončením volby telefonního čísla a handshake tónem. Jakmile tato časová prodleva vyprší, tak se změní proměnná `DTMF_phase` na hodnotu 3 a příkazem `tone_hand()` se vyvolá funkce pro přehrání handshake tónu.

Handshake tón je vytvořen v souboru tone.c, který je vypsán v příloze B.1.4. V tomto souboru jsou definovány proměnné `sin1400` a `sin2300`, který v poli obsahují číselně vyjádřené body na sin křivce 1400Hz a 2300Hz, s amplitudou 10000, při vzorkovacím kmitočtu 32200Hz. Vzorkovací kmitočet je zde tvořen časovačem *Timer 2*. Při každém přerušení se vypočte z proměnných `sin1400` a `sin2300` aktuální hodnota pro D/A převodník. Hodnota pro D/A převodník obsahuje číselný offset 512 (polovina 10-bitového rozsahu D/A převodníku), aby se dodrželo symetrické napětí. Velikost amplitudy určuje proměnná `TONE_level`

v rozsahu 0 až 512.

Po vyslání handshake tónu se změní režim příjmu DTMF na hodntou 4 a spustí se časovač hlídající maximální dobu očekávání příjmu zprávy protokolu Contact ID. Pokud čas časovače vyprší dříve než přijde první symbol zprávy, tak se znovu odešle handshake tón. Handshake tón se opakuje maximálně pětkrát, poté se změní režim zpět na hodnotu 0 a odešle se e-mailová zpráva o chybě příjmu zprávy Contact ID ze zařízení EZS.

Po příjmu prvního symbolu zprávy protokolu Contact ID se změní režim příjmu dat na hodnotu 5 a očekává se přijetí celkem 15-ti DTMF symbolů. Jakmile je celá zpráva přijata, tak se provede kontrola přijatého kontrolního součtu na konci zprávy a vyhodnocuje se, zda se jedná o Contact ID zprávu, tj. má-li na 5. a 6. symbolu hodnotu 18 nebo 98. Pokud je zpráva validní, tak se změní režim příjmu dat na hodnotu 6 a spustí se časovač vytvářející rozestup mezi přijetím zprávy a odesláním kiss-off tónu. Zároveň se nastaví proměnná `IPC_phase` na hodnotu 6, která dává informaci o tom, že po skončení příjmu dat z telefonní linky se má odeslat zpráva protokolu Contact ID prostřednictvím TCP protokolu.

Po kiss-off tónu se změní režim příjmu dat z telefonní linky na hodnotu 8 a očekává se případná další zpráva. V případě, že se přijme další symbol,tak se změní režim příjmu dat na hodnotu 5 a pokračuje se stejně jako u první zprávy. Převodník může přijmout maximálně 3 zprávy před odesláním zpráv na PCO. Po odeslání všech zpráv může převodník přijmout opět další 3 zprávy. Tento počet byl zvolen tak, aby při normálním provozu nezpomaloval příjem dat z EZS a nehrozilo riziko, že při ztrátě napájení se ztratí velký počet zpráv. Pokud je tento počet zpráv docílen, tak převodník další zprávy nepřijímá. Nepřijaté zprávy jsou uloženy zpravidla v zařízení EZS, které se je pokusí po krátkém intervalu znova odeslat.

Pokud již žádná nová zpráva po kiss-off tónu není přijatá, tak se proměnná `DTMF_phase` vrátí na hodnotu 0 a příjem dat ze zařízení EZS je ukončen.

Při režimu příjmu dat 9 se očekává volba, zda-li uživatel chce zařízení resetovat do výchozího nastavení nebo zda jen restartovat. Reset nastavení je proveden volbou "1" a restart zařízení je po volbě "2".

4.4 Odesílání zpráv protokolu Contact ID prostřednictvím TCP

Podobně jako u přenosu zpráv po telefonní lince je zde přenos rozdělen do fází, které jsou určeny proměnnou `IPC_phase`. Pokud se aktuálně nepřenáší žádná zpráva, tak má tato proměnná hodnotu 0. V této fázi se nevykonává žádná činnost z pohledu přenosu dat prostřednictvím TCP a čeká se na příjem dat ze zařízení EZS nebo na přerušení časovačem pro dohledovou zprávu NULL.

Jakmile dojde k přerušení časovače, tak se vytvoří zpráva NULL. Tyto zprávy jsou dvojího typu, jak je popsáno v kapitole 2.2.3, a to šifrovaná verze a nešifrovaná verze

dohledové zprávy NULL.

U nešifrované zprávy se nejprve zkontroluje, zda uživatel požaduje odesílání časového údaje, který je u nešifrované zprávy nepovinný. Pokud si uživatel tento údaj přeje odesílat, tak se zjistí aktuální datum a čas z RTC hodin prostřednictvím funkce `RTCGetTime` a tento údaj se přidá na konec zprávy ve formátu popsaném v kapitole 2.2.2. Před tento časový údaj se přidá ještě prefix čísla objektu a číslo objektu. Poté následuje ověření, zda se přenáší prefix čísla přijímacího zařízení. Poté je tento údaj dán na konec zprávy, před který se umístí číslo sekvence (u NULL je to 0000) a typ zprávy. Na závěr je spočítáno CRC a délka zprávy. Výsledky výpočtu CRC a délky zprávy jsou přidány na začátek celé zprávy.

U šifrované verze je tvorba zprávy velmi podobná s tím rozdílem, že časový údaj je zde povinný a před tento časový údaj se dává pseudonáhodná sekvence pad, která je popsaná v kapitole 2.2.5. Sekvence pad a časový údaj je zašifrován pomocí AES s metodou CBC prostřednictvím funkcí ze souborů aes.c a aes.h, které jsou převzaty z [10]. Zbytek tvorby zprávy je stejný jako u nešifrované zprávy s jedninou výjimkou, že u typu zprávy je *, která slouží pro přijímací zařízení jako indikace, že se jedná o šifrovanou zprávu.

Jakmile dojde k vytvoření zprávy, tak se změní fáze přenosu na hodnotu 1 a naváže se spojení se serverem pultu centrální ochrany příkazem `tcp_connect`. Pokud se spojení nepodaří úspěšně navázat, tak se inkrementuje proměnná zaznamenávající počet chyb při přenosu dat prostřednictvím TCP. Pokud je počet těchto chyb stejný jako uživatelem nastavená mez, tak se odešle varovná zpráva prostřednictvím e-mailu.

Úspěšné navázaní spojení je potvrzeno tak, že RL-TCPnet vyvolá funkci `tcp_callback` s parametrem `TCP_EVT_CONNECT`. Poté je změněna fáze přenosu na hodnotu 2 a vytvořená zpráva se odešle po Ethernetu. Třetí fáze nastane, jakmile RL-TCPnet potvrdí přijetí zprávy přijímacím zařízením vyvoláním funkce `tcp_callback` s parametrem `TCP_EVT_ACK`. Potom se očekává přijetí potvrzovací zprávy z pultu centrální ochrany. Pokud do nastaveného času tato zpráva nepřijde, tak se zpráva na PCO odesílá znovu, takto se to provede celkem třikrát. Pokud se zpráva neodešle úspěšně (PCO neodpoví), tak se inkrementuje proměnná počítající chyby při přenosu prostřednictvím TCP. Navíc je okamžitě odeslána zpráva prostřednictvím e-mailu o neúspěšném odeslání zprávy na PCO.

Příchozí potvrzovací zpráva je přijata a zpracovaná funkcí `receive_mess`, která nejprve změní fázi přenosu dat po TCP na hodnotu 4. Poté je u zprávy vypočtena délka zprávy a CRC a pokud hodnoty souhlasí s údajem na začátku přijaté zprávy, tak se vyhodnocuje typ příchozí zprávy. Pokud je zpráva ACK, tak se vynulují chybové proměnné a zkontroluje se, zda nejsou další zprávy k odeslání. Pokud ano, tak se změní fáze přenosu dat na hodnotu 6. Pokud již nejsou další zprávy k dispozici, tak se TCP spojení ukončí a změní se proměnná `IPC_phase` zpět na hodnotu 0.

V případě, že přijatá zpráva je typu NAK, tak se nejprve synchronizuje příchozí časový údaj RTC hodin s údajem ze zprávy. Poté se změní fáze přenosu dat na hodnotu 5, která zajišťuje přeposlání zprávy na pult centrální ochrany. Další možnou zprávou je DUH,

která oznamuje, že PCO momentálně nebylo schopné zprávu zpracovat. V tomto případě se opět změní fáze přenosu dat na hodnotu 5.

Přenosová fáze 6 znamená, že v paměti je neodeslaná zpráva o události. Tato zpráva má opět šifrovanou a nešifrovanou verzi. Tvorba zprávy je stejná jako u dohledových zpráv NULL s výjimkou, že se před časový údaj přidává informace o události ve formátu popsané v kapitole 2.2.2.

Zpracování informací o události pro přenos prostřednictvím TCP protokolu tvoří funkce `IPC_convert`, která z proměnné `DTMF_symbol` převede data přijatá z telefonní linky na tvar požadovaný při přenosu události prostřednictvím TCP. Vytvoření spojení, přenos dat a potvrzení je stejné jako u dohledové zprávy. Při pozitivním potvrzení ACK je zde navíc odeslána e-mailová zpráva uživateli s informací o události.

4.5 Odesílání e-mailových zpráv

Aby byl uživatel co nejrychleji informován o aktivitě v hlídaném objektu, tak byla vytvořena funkce pro odesílání jednoduchých zpráv. Uživatel si může nastavit, které zprávy mu budou posílány, z kterého e-mailového účtu a na kterou adresu mu budou zprávy odesílány. Samozřejmě musí nastavit parametry SMTP serveru a to IP adresu SMTP serveru, port pro odchozí poštu a autorizační parametry.

Odesílání zprávy v převodníku probíhá tak, že funkce, která požaduje odeslání zprávy, nejprve naplní řetězec `MAIL_sub`, pro předmět zprávy, a řetězec `MAIL_body`, který obsahuje samotnou zprávu. Po naplnění řetězců vyvolá funkci `send_mail`, který vytvoří spojení se SMTP serverem.

Funkce `send_mail` využívá funkcí RL-TCPnet, která hned na začátku vyvolá funkci `smtp_accept_auth`. Tato funkce vrací booleovskou hodnotu o tom, zda SMTP server vyžaduje autorizaci. Po připojení na SMTP server je postupně vyvolávána funkce `smtp_cbfunc`, která má v parametru informaci o hodnotě, kterou aktuálně SMTP server žádá. Tato funkce tak postupně odesílá uživatelské jméno, heslo, odchozí adresu, adresu příjemce, předmět zprávy a na závěr obsah zprávy.

O stavu odeslané zprávy a připojení na SMTP server dává informaci funkce `smtp_cback`, která je opět vyvolaná RL-TCPnet stackem.

4.6 Nastavení převodníku přes TELNET

Nastavení převodníku je možné provést přes rozhraní TELNET. Podporu připojení a zpracování poskytuje RL-TCPnet, který si pro tento účel vyhrazuje použití TCP portu 22. RL-TCPnet podporuje připojení TELNETu s i bez požadavku na autorizaci, v této aplikaci byla použita verze s autentizací přihlašovacím jménem, které je vždy admin a měnitelné heslo, které je ve výchozím nastavení též admin.

Veškeré ovládání a zpracování dat z TELNETu je v souboru telnet.c. Při připojení klienta na místní TCP port 23, je spuštěn prostřednictvím RL-TCPnet funkce `tnet_cbfunc`. Tato funkce vrací požadované hodnoty, které jsou specifikované v parametru funkce.

Jakmile je klient úspěšně připojen a přihlášen, tak již může zadávat jednotlivé příkazy. Příjem příkazu zpracovává funkce `tnet_process_cmd`, který v parametru v proměnné `cmd` má celý řetězec, který uživatel zadal. Pomocí funkce `tnet_ccmp` lze pak následně zjistit, zda uživatel tento příkaz zadal. Parametr se poté načítá funkcí `sscanf` a případný text, který se má uživateli na obrazovce objevit se příkazem `sprintf` uloží do proměnné `buf`, jejíž pointer je deklarován v parametru funkce `tnet_process_cmd`. Jako výsledek funkce se vrací hodnota `len`, která je vrácena z funkce `sprintf`. Výsledný kód pro jeden příkaz pak vypadá takto:

```
if (tnet_ccmp (cmd, "IP") == __TRUE) {
 if (len > 4) {
  sscanf((const char *)(cmd+3),"%d.%d.%d.%d",&ipa[0],&ipa[1],&ipa[2],
  &ipa[3]);
  for (i=0;i<4;i++)
  {
    if (ipa[i] > 255)
    {
      len = sprintf((char *)buf,"\r\n IP => Neplatna adresa");
      return (len);
    }
  }
  ip = 1;
  len = sprintf((char *)buf,"\r\n IP => Nova adresa bude %d.%d.%d.%d
  Hodnotu ulozite prikazem SAVE_IP",ipa[0],ipa[1],ipa[2],ipa[3]);
  return (len);
 }
}
```

Podobným způsobem jsou zpracované téměř všechny příkazy prostřednictvím rozhraní TELNET. Výjimkou je příkaz `HELP`, který nedokáže najednou zobrazit celou nápovědu z důvodu velkého rozsahu, proto se nejdříve vypíše první část nápovědy a k proměnné `len` se přičte hodnota 0x4000, která zajistí, že hned po ukončení funkce `tnet_process_cmd` se tato funkce spustí hned znova, kdy se vypíše další část nápovědy, až se nakonec vypíše celá.

Dalším příkazem, který využívá přičítání určité hodnoty k proměnné `len` je příkaz `QUIT`, který TCP spojení s klientem ukončí. Hodnota, která se k proměnné `len` přičítá je 0x8000.

Všechny nastavovací příkazy s výjimkou příkazy pro IP nastavení převodníku ukládají hned po přijetí data do FLASH paměti převodníku pomocí funkce `NVOL_SetVariable`

nebo funkcemi ze souboru set.c, který je popsán v příloze B.1.3, a nastavení se ihned uvede do praxe. IP nastavení se nejprve uloží do pomocných proměnných, odkud se poté uloží do FLASH paměti vyvoláním příkazu `SAVE_IP`. Po dokončení uložení IP nastavení se zařízení restartuje a IP nastavení se aplikuje. Důvodem pro tento postup je, že se nejprve mohou dokončit všechny nastavení před tím, než se aplikují. Pokud by došlo k postupné aplikaci jako u ostatních nastavení, tak by hrozilo, že při změně IP adresy převodníku nebo IP adresy výchozí brány, že by zařízení nemuselo být již v síti dostupné.

4.7 Ukládání a načítání dat z FLASH paměti

Pro ukládání dat, které musí být dostupné i po restartu zařízení, je vyhrazen ve FLASH paměti prostor v sektorech 12 a 13. Každý z těchto sektorů má velikost 32kB, velikost proměnné, které se do FLASH paměti ukládají, je 48 bytů. Z těchto 48 bytů jsou první čtyři byty vyhrazeny pro data funkcí ze souboru flash_nvol.c, takže pro data je k dispozici 44 bytů. Nastavený maximální počet proměnných je 50, které stačí pro všechna uložená nastavení. První sektor slouží jako hlavní, kam se ukládají všechny data. V případě, že první sektor nebude správně pracovat, tak se data začnou ukládat do druhého sektoru.

Veškeré zdrojové kódy pro ukládání dat do FLASH paměti jsou převzaty z [7]. Jedinou nutnou úpravou je nastavení parametrů FLASH paměti mikrokontroléru, které v tomto převodníku jsou takovéto:

```
#define SECTOR1_STARTADDR 0x00028000
#define SECTOR1_NUM       12
#define SECTOR2_STARTADDR 0x00030000
#define SECTOR2_NUM       13
#define SECTOR_SIZE 0x8000
#define CPU_CLK 48000

#define IAP_LOCATION 0x7FFFFFF1

#define MAX_VARIABLES 50
#define MAX_VARIABLE_SIZE 44
#define INVALID_VAR_OFFSET 0
```

Před samotným používáním dat z FLASH paměti je potřeba inicializace, která se provede příkazem `NVOL_init`, která se spouští hned při startu převodníku. Pro zápis dat se využije funkce `NVOL_SetVariable`, kde první parametr určuje pozici ve FLASH paměti. V tomto firmware jsou pozice definované v souboru setup.h, který je v plném rozsahu vypsán v příloze B.1.6. Druhým parametrem je pointer samotné proměnné, která se má ukládat, a posledním parametrem je velikost ukládané proměnné. Pro načtení dat z FLASH paměti slouží funkce `NVOL_GetVariable`, kde první parametr je pozice proměnné, která

je shodná s pozicí při ukládání. Druhý parametr je opět pointer proměnné, kam se budou data ukládat, a třetí parametr je velikost ukládaných dat.

5

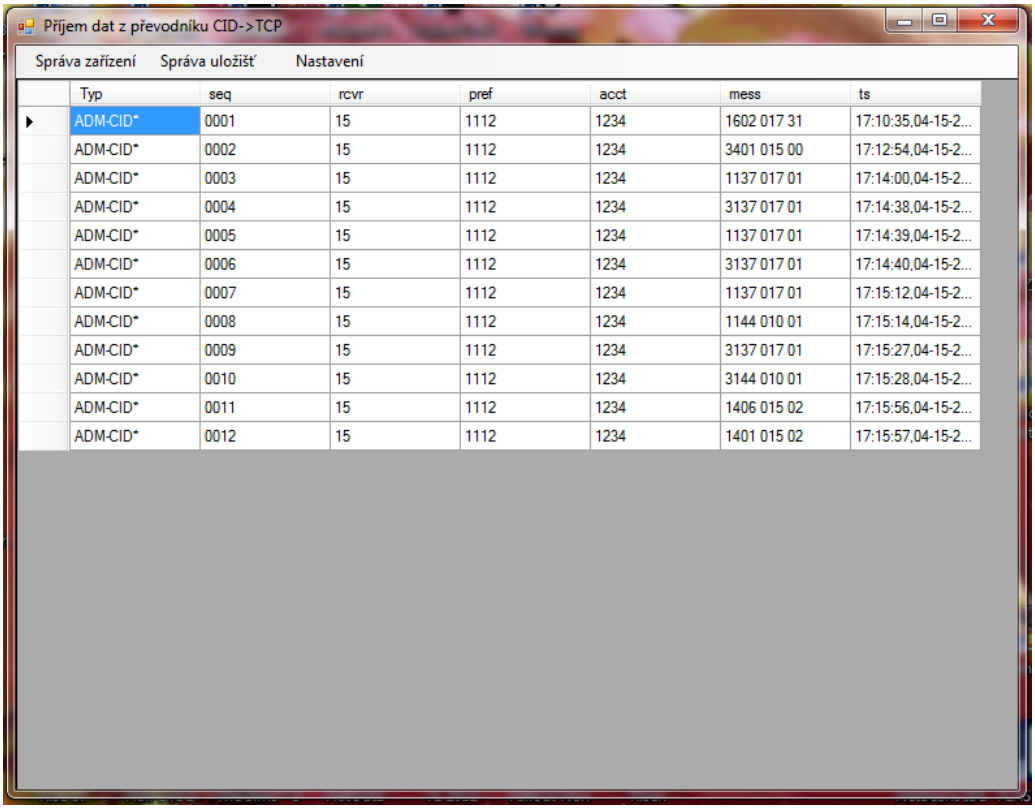
Aplikace pro příjem zpráv z převodníku

Pro pulty centrální ochrany, které nepodporují příjem zpráv protokolu Ademco Contact ID prostřednictvím TCP (podle normy SIA DC-09 [3]), byla vytvořena aplikace, která tyto zprávy přijímá, zpracovává a ukládá do databází Microsoft SQL, MySQL a do textového souboru (např. s příponou .csv). Tato aplikace je vytvořena tak, aby byla spuštěna na pozadí operačního systému a nerušila ostatní aplikace. Informace z aplikace jsou uživateli předávány prostřednictvím systémových "bublinových" zpráv. Zdrojový kód aplikace byl vytvořen v Microsoft Visual Studiu 2008 v jazyku VB.NET. Software je plně 32-bitový s podporou i 64-bitového systému.

Po startu aplikace se objeví hlavní okno aplikace (form1.vb), kde lze přehledně vidět v tabulce (*DataGridView1*) příchozí zprávy ze zařízení, které byly přijaty v průběhu běhu aplikace. V horní části okna je nabídka s možností zobrazit okna pro správu zařízení (kapitola 5.1), správu úložišť (kapitola 5.3) a nastavení (kapitola 5.2). Náhled okna je vidět na obrázku 5.1.

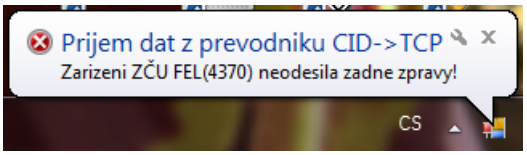
Při načítání aplikace po startu se nejprve upraví velikost tabulky (*DataGridView1*) podle velikosti hlavního okna aplikace. Tento krok se provede vždy, když se změní velikost tohoto okna. Následně se provede načtení informací z databáze zařízení do struktury `dev_info.chint`. Po načtení informací se aktivuje časovač *Timer1*, který hlídá maximální intervaly od poslední zprávy z jednotlivých zařízení. Následuje aktivace vlákna `readThread` pro čtení dat z TCP spojení. Na závěr se aktivuje časovač *Timer2*, který pravidelně aktualizuje hlavní okno aplikace a provede se nastavení systémové ikony a zpráv.

Časovač *Timer1* má nastavený interval přerušení 1 vteřinu. Po každém přerušení se provede nejprve u každého zařízení přičtení 1 vteřiny k době, kdy naposled odeslali platnou zprávu (tj. bylo odesláno pozitivní potvrzení). Tento časový údaj se následně porovná s uživatelem nastaveným časovým intervalem maximální nečinnosti zařízení a pokud je shodný, tak se změní proměnná `flag` ve struktuře `dev_info.chint` na hodnotu 1, které znamená, že zařízení neodpovídá a uživateli se zobrazí systémová zpráva, která je vidět



Obr. 5.1: Hlavní okno aplikace

na obrázku 5.2.



Obr. 5.2: Systémová zpráva

Časovač *Timer2* má nastavený interval přerušení také 1 vteřinu. Po přerušení časovač zkontroluje, zda nepřibyla nová zpráva, která se má zobrazit v tabulce v hlavním okně, v tomto případě se spustí funkce `add_list`, která zajistí potřebnou aktualizaci dat. Dále časovač provádí kontrolu, zda některá z jiných části této aplikace nepotřebuje zobrazit systémovou zprávu. Přímé zobrazení systémové zprávy umožňuje pouze hlavní okno (*form1*), jiné okna nebo třídy musí využít tohoto časovače.

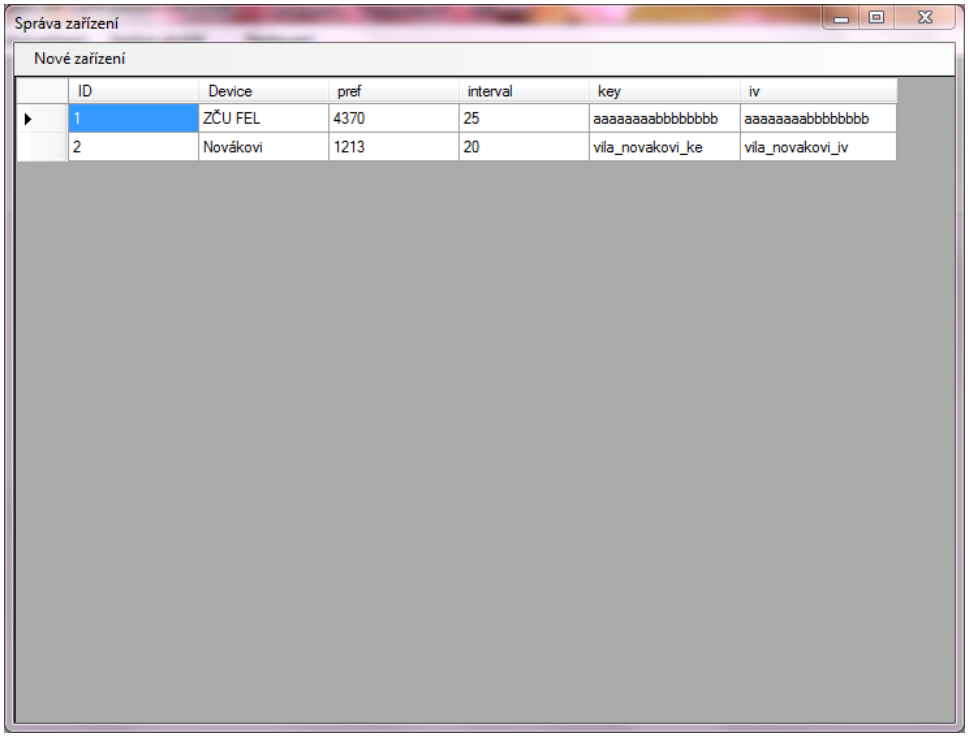
5.1 Správa zařízení

Tato aplikace byla vytvořena tak, aby mohla přijímat zprávy i z více zařízení. Proto bylo zapotřebí vytvořit okno pro správu zařízení, které je propojeno s databází, obsahující informaci o těchto zařízeních.

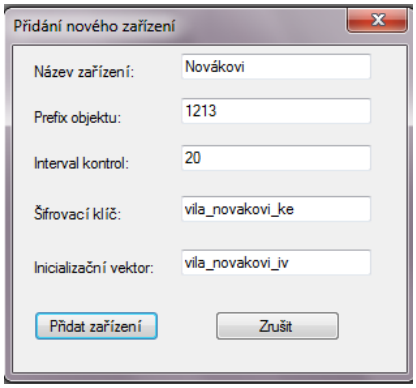
Databáze je vytvořena v aplikaci Microsoft Access 2010, v které je tabulka nazvaná

`devices` a obsahuje 6 sloupců. První je sloupec id, který se při přidání nového záznamu automaticky inkrementuje. Další sloupce jsou Device (název zařízení), pref (prefix objektu), int (maximální interval nečinnosti zařízení), key (šifrovací klíč AES CBC) a iv (inicializační vektor AES CBC). Databáze je zaheslovaná.

Okno pro správu zařízení je vidět na obrázku 5.3. Tabulka (*DataGridView1*) zobrazuje obsah databáze popisované v předchozím odstavci. Tato tabulka umožňuje zároveň úpravu jednotlivých buněk a i vymazání celé řádky. Všechny tyto úpravy se projeví v databázi. V horní části okna se opět nachází nabídka s možností otevření nového okna pro přidání nového zařízení. Okno pro přidání nového zařízení je vidět na obrázku 5.4.



Obr. 5.3: Okno pro správu zařízení



Obr. 5.4: Okno pro přidání nového zařízení

Při přidání nového zařízení se nejprve ověří, zda všechny zadané hodnoty jsou validní, tj. zda je prefix objektu jedinečný (kontroluje funkce `kontrola_prefix`), zda je interval

v rozsahu 10 až 3600 vteřin nebo zda šifrovací klíč a inicializační vektor mají správnou délku. Po zkontrolování všech těchto hodnot se provede uložení do databáze prostřednictvím funkce `Novy_zaznam`, která provede i přidání zařízení do struktury `dev_info.chint`, aby zařízení mohlo být používáno okamžitě.

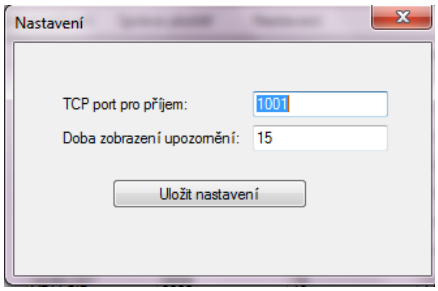
Při editaci buňky v tabulce je postup velmi podobný. Nejprve se zkontroluje zda nová hodnota v buňce je validní a poté dojde k přepisu hodnoty v databázi. Po uložení databáze se tabulka aktualizuje a aktualizace se provede i ve struktuře `dev_info.chint`. V případě vymazání se u struktury `dev_info.chint` změní proměnná `flag` na hodnotu 2, která znamená, že zařízení bude dále ignorováno. Po restartu aplikace se toto zařízení již do struktury nenačte.

5.2 Nastavení základních parametrů aplikace

Aplikace pro příjem zpráv z převodníku umožňuje i nastavení lokálního TCP portu pro příjem zpráv z převodníku. Dalším možným nastavením je délka zobrazení systémové zprávy. Nastavený čas je ve vteřinách. Obě položky se ukládají do registru operačního systému Windows do adresy:

`HKEY_CURRENT_USER\Software\VB and VBA Program Settings\CID->TCP`

Náhled okna pro nastavení aplikace je na obrázku 5.5.

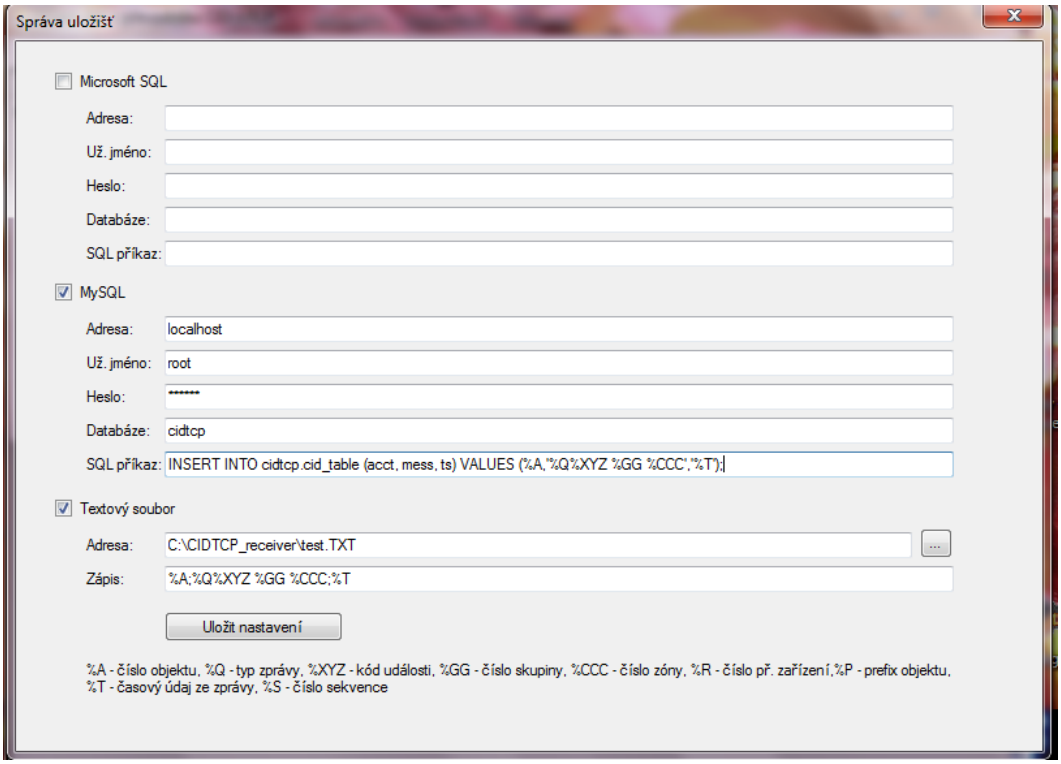


Obr. 5.5: Okno pro nastavení parametrů aplikace

5.3 Ukládání zpráv do databáze SQL a do textového souboru

Přijaté zprávy z převodníku prostřednictvím protokolu TCP mohou být uloženy do databází Microsoft SQL a MySQL nebo do textového souboru. Výběr úložišť a jejich nastavení se provádí v okně Správa úložišť (*form4*). Toto okno je na obrázku 5.6.

Pro ukládání dat v MSSQL se zvolí adresa SQL serveru a jeho přihlašovací údaje. Následuje název databáze a prostor pro zadání SQL příkazu. V zadaným SQL příkazu jsou nahrazeny hodnoty, které chce uživatel do databáze zapsat zástupnými řetězci, které začínají %. Tyto řetězce jsou poté ve zdrojovém kódu nahrazeny hodnotou přijaté ze zprávy.



Obr. 5.6: Okno pro správu uložišť

Pro proces uložení zprávy do databáze MSSQL byla vytvořena funkce `ToMSSQL`, která se nachází v class5.vb. Tato funkce nejprve rozdělí zprávu na jednotlivé části podle normy SIA DC-05 popsané v kapitole 2.1.1. Dalším krokem je vytvoření připojovacího řetězce a nahrazení zástupných znaků v SQL příkazu hodnotami ze zprávy. Následně dojde k připojení k SQL serveru, vykonání příkazu a k odpojení. V případě chyby se zobrazí uživateli systémová zpráva.

Parametry pro nastavení databáze MySQL jsou stejné jako u MSSQL, i průběh procesu je velmi podobný. Rozdílem oproti ukládání dat do MSSQL je, že u MySQL se používá jiná knihovna než v případě MSSQL. Tato knihovna má i trochu odlišné postupy připojení a vytvoření příkazu.

Pro ukládání do textového souboru je zapotřebí nastavit cestu k textovému souboru, kterou lze vyplnit stisknutím tlačítka vedle TextBoxu a obsah zapsané řádky. Obsah je opět nahrazen zástupnými znaky jako v případě SQL databází. Při procesu ukládání dat se textový soubor otevře a data se zapíší na novou řádku. Poté je soubor uzavřen. V případě chyby se opět uživateli objeví systémová zpráva.

5.4 Příjem zpráv z převodníku prostřednictvím protokolu TCP

Kontinuální příjem zpráv z převodníku prostřednictvím TCP protokolu je zajištěn vláknem `readThread`, které je spouštěno hned při startu aplikace. Toto vlákno neustále kon-

troluje stav připojení zařízení a přijímá od nich data. Po přijetí jsou data zpracována funkcí `check_message`.

Funkce `check_message` nejprve zkontroluje délku zprávy a CRC s údajem v hlavičce zprávy. Pokud tyto údaje souhlasí, tak se provede rozšifrování zprávy a načtení dat ze zprávy do proměnných, které se ve funkci dále zpracovávají.

Po načtení dat do proměnných se zkontroluje typ zprávy. Pokud se jedná o nešifrovanou kontrolní zprávu NULL, která neobsahuje časový údaj, tak se okamžitě odesílá pozitivní potvrzení prostřednictvím funkce `create_ACK`. Při pozitivním potvrzení se vynuluje proměnná `resp` ve struktuře `dev_info.chint`, počítající dobu od poslední úspěšné komunikace zařízení. V případě, že zpráva obsahuje časový údaj, tak se tento údaj nejprve zkontroluje. V případě, že je v limitu dané normou SIA DC-09 [3], tak se odešle stejné pozitivní potvrzení jako v případě zprávy bez časového údaje. Pokud časový údaj není v normou daném limitu, tak se odešle negativní potvrzení pomocí funkce `create_NAK`.

U šifrované kontrolní zprávy je časový údaj povinný, proto se zde kontrola shody času provádí vždy. V případě, že časový údaj je v povoleném rozsahu, tak se odešle zašifrované pozitivní potvrzení prostřednictvím funkce `create_ACKen`. V opačném případě se odesílá negativní potvrzení pomocí funkce `create_NAK`.

U nešifrované zprávy ADM-CID se také zkontroluje případný časový údaj. Pokud je tento časový údaj v normou daném rozsahu, nebo tento časový údaj není, tak se odešle pozitivní potvrzení pomocí funkce `create_ACK`. Stejně jako u kontrolní zprávy dojde k vynulování proměnné `resp` ve struktuře `dev_info.chint`, zpráva se uloží do listu pro tabulku v hlavním okně a provede se zápis do databáze SQL popsané v kapitole 5.3.

U šifrované zprávy je časový údaj povinný, takže kontrola časového údaje se provádí vždy. V případě pozitivního potvrzení se odešle zašifrované pozitivní potvrzení prostřednictvím funkce `create_ACKen`. V opačném případě se odesílá negativní potvrzení pomocí funkce `create_NAK`.

Pokud zpráva je správně dlouhá a má správné CRC a přesto typ zprávy není touto aplikací dekódován, tak se odešle zpráva typu DUH, která znamená, že aplikace není schopna tuto zprávu zpracovat.

V případě špatného CRC nebo špatné délky zprávy se neodesílá žádná zpráva a je vyčkáváno na opravenou zprávu.

5.5 Šifrování a dešifrování zpráv

Pokud je přijata zašifrovaná zpráva, tak se nejprve provede nastavení šifrovacího klíče a inicializačního vektoru do proměnných `AES_key` a `AES_iv`, podle prefixu objektu zařízení. Toto nastavení je provedeno funkcí `AESset`. Zbytek dešifrování provede funkce `AESdec`. Zašifrovaná část je přenášena v hexadecimálním tvaru, proto se provede na začátku převod z hexadecimálního tvaru na znaky ASCII. Po převodu se provede dešifrování pomocí knihoven .NET Framework, která jsou ve VB.NET k dispozici.

V případě, že se dešifrování nepodaří (např. z důvodu špatného klíče nebo inicializačního vektoru), tak funkce vrátí prázdný řetězec.

U šifrování zprávy je postup přesně opačný. Nejprve se zpráva zašifruje pomocí knihoven .NET Framework. Zašifrovaná zpráva se převede z ASCII znaků do hexadecimálního tvaru. Tato zpráva se poté odešle. Šifrování provádí funkce `AESenc`.

5.6 Technické parametry aplikace

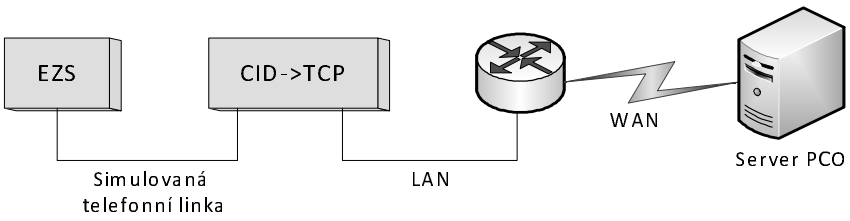
- **Operační systém:** MS Windows XP, Vista, 7, 8, Server 2003, Server 2008
- **Podpora systémů:** 32-bitové i 64-bitové systémy
- **verze .NET Framework:** Verze 3.5 a vyšší
- **Minimální aplikační vybavení OS:** MS Access Database Engine 2010 a vyšší, MySQL Connectors
- **Verze MySQL serveru:** Verze MySQL 5 a vyšší
- **Verze MSSQL serveru:** Verze MSSQL 2005 a vyšší
- **Síťové požadavky:** Ethernetová síť s podporou TCP/IP s veřejnou IP adresou

6

Použití v praxi a testování reálného zapojení

6.1 Připojení k zařízení EZS a PCO

Převodník popisovaný v této práci přijímá zprávy protokolu Contact ID ze simulované telefonní linky, na kterou je připojeno zařízení EZS. Tyto zprávy jsou převodníkem zpracovány a odeslány na PCO prostřednictvím protokolu TCP. Pakety protokolu TCP jsou přenášeny skrze lokální ethernetovou síť, která musí mít možnost se spojit se sítí, ke které je připojen server PCO. Základní zapojení převodníku v praxi je vidět na obrázku 6.1.



Obr. 6.1: Základní zapojení převodníku v praxi

K připojení převodníku k zabezpečovací ústředně EZS se využívá standardního telefonního dvouvodičového kabelu s konektorem RJ11. Telefonní linka je napájena napětím 24V. Zkratový proud telefonní linky je omezen na 80mA. Při zkratu není poškozeno, ani přetíženo žádné interní zapojení převodníku. Zakončovací impedance telefonní linky je 600Ω.

Napájení převodníku je řešeno prostřednictvím PoE 24V (doporučeno použití zdroje s proudovou zátěží 1A). Napájecí zdroj je doporučeno připojit co nejblíže k převodníku vzhledem ke ztrátám na Ethernetovém vedení. Maximální proud po Ethernetovém vedení je omezen vstupní pojistkou 650mA. Vzhledem k použití napájení prostřednictvím PoE je nutné, aby byl přívodní Ethernetový kabel, mezi převodníkem a napájecím zdrojem, 8-vodičový, jinak může hrozit, že převodník nebude správně fungovat.

Připojení k síti WAN je nutné provést tak, že pakety odesílané převodníkem budou

mít nejvyšší prioritu. Toto lze dosáhnout použitím QoS, který dnes umí prakticky každý router či modem. Při instalaci převodníku je nutné se přesvědčit, že připojení k PCO povoluje přenos dat na protokolu, který převodník používá (v základním nastavení je to TCP port 1001).

Převodník nesmí být umístěn v prostorech, který neumožňují snadný odvod tepla. V opačném případě hrozí přehřátí a porucha přístroje.

6.2 Nastavení převodníku

Nastavení převodníku se provádí prostřednictvím připojením TELNETu, který je na převodníku přijímán na TCP portu 23. Po připojení je požadováno přihlášení. Uživatelské jméno je vždy admin. Heslo je měnitelné, ale v základním nastavení je také admin.

Po přihlášení je uživatel vyzván již k zadání příkazu. Pro vypsání všech dostupných příkazu pro nastavení převodníku je příkaz `HELP`. Veškeré příkazy jsou rozdělené do 6 skupin. První skupina je pro IP nastavení převodníku, druhá se zabývá nastavením přijmu zpráv ze zařízení EZS po telefonní lince. Třetí skupina se zabývá nastavením pro odesílání zpráv na PCO. Čtvrtá skupina je pro nastavení e-mailových zpráv. V páté skupině se nastavuje pouze výstupní úroveň signálu po telefonní lince. Poslední skupina je pro ovládání připojení prostřednictvím TELNETu.

Ve skupině pro IP nastavení převodníku se nastavuje IP adresa převodníku (příkaz `IP`), maska sítě (příkaz `SM`) a IP adresa výchozí brány (příkaz `GW`). Dalším možným nastavením je IP adresa primární a sekundární DNS brány (příkaz `DNS1` a příkaz `DNS2`). Také je možné změnit MAC adresu převodníku a to lze udělat příkazem `MAC`. Pro aktivaci nebo deaktivaci získávání IP nastavení ze serveru DHCP je použit příkaz `DHCP`. Název zařízení, kterým se bude převodník prezentovat v ethernetové síti a který bude používat v e-mailových zprávách, lze měnit příkazem `DEV`. Pro uložení a aplikaci všech nastavení se zadává příkaz `SAVE_IP`. Po zadání tohoto příkazu se převodník restartuje.

Druhá skupina nastavuje parametry přijmu zpráv ze zařízení EZS. Maximální interval mezi DTMF symboly při odesílání zprávy nebo telefonního čísla ze zařízení EZS se nastavuje příkazem `EZS_ME`. Pro ruční volbu (reset nastavení nebo restart zařízení pomocí připojeného telefonu) se nastavuje tento interval pomocí příkazu `EZS_MM`. Tyto intervaly lze zvolit v intervalu 100 až 2500ms. Dalším parametrem je interval mezi koncem volby telefonního čísla převodníku zařízením EZS a vysláním Handshake tónu. Tento parametr se nastavuje příkazem `EZS_BH`. Interval se volí v rozsahu 500 až 2000ms. Příkazem `EZS_BM` se volí interval vyčkávání na zprávu po vyslání handshake tónu. Parametr se volí v rozsahu 500 až 4000ms. Interval odeslání kissoff tónu po úspěšném přijetí zprávy z telefonní linky se nastavuje příkazem `EZS_BK`, kde parametr se zadává v rozsahu 1000 až 1500ms. Posledním příkazem v této skupině je telefonní číslo převodníku (příkaz `EZS_TN`, který musí mít délku v rozsahu 3 až 9 číslic.

Skupina pro nastavení odesílání zpráv na PCO obsahuje příkazy pro nastavení IP

adresy serveru PCO (příkaz `PCO_IP`) a vzdálený TCP port pro odesílání zpráv (příkaz `PCO_PORT`). Dále se zde volí, zda má být přenos šifrován (příkaz `PCO_EN`), jaké je číslo přijímacího zařízení (příkaz `PCO_RCVR`), prefix objektu (příkaz `PCO_PREF`) a číslo objektu (příkaz `PCO_ACCT`). Číslo objektu lze překládat ze zpráv přijaté za zařízení EZS nebo lze pro zprávy na PCO použít zadané číslo objektu. Tato funkce se ovládá příkazem `PCO_ANT`. U nešifrovaných zpráv není povinné přenášet časový údaj, možnost odesílání se volí příkazem `PCO_TS`. Interval odesílání kontrolních zpráv se nastavuje příkazem `PCO_IN`, tento interval musí být v rozsahu 10 až 3600 vteřin. Na závěr jsou zde příkaz pro nastavení šifrovacího klíče (příkaz `PCO_KEY`) a inicializačního vektoru (příkaz `PCO_IV`).

Další skupina se zabývá nastavením e-mailových zpráv. Prvním parametry jsou e-mailová adresa odesilatele zprávy (příkaz `MAIL_FROM`), příjemce zprávy (příkaz `MAIL_TO`) a IP adresa SMTP serveru (příkaz `MAIL_IP`). Dále se nastavují autorizační parametry příkazy `MAIL_AUTH`, `MAIL_USER` a `MAIL_PASS`. Další příkazy slouží pro nastavení, které zprávy si uživatel přeje přijímat.

Nastavení úrovně výstupního signálu telefonní linky se volí příkazem `TONE_LEVEL`. Parametr se volí v rozsahu 0 až 512, kdy hodnota 450 odpovídá výstupní úrovni 0dBm na konektoru převodníku. Hodnota 0 znamená, že převodník nebude generovat výstupní signál. Závislost zadané hodnoty a výstupní úrovně je lineární.

Do poslední skupiny patří příkaz `HELP` a příkaz `CH_PSWD`, který slouží pro změnu přihlašovacího heslo pro nastavení převodníku. Posledním příkaze v této skupině je `QUIT`, který ukončí relaci.

6.3 Nastavení aplikace pro příjem zpráv z převodníku

Pro správnou funkci aplikace pro příjem zpráv z převodníku je zapotřebí, aby na serveru byl nainstalován .NET Framework verze 3.5 a vyšší. Aplikace také potřebuje engine pro přístup do databáze Access. Pokud se aplikace užívá pro ukládání dat do MySQL databáze je zapotřebí nainstalovat MySQL Connectors for .NET. Všechny potřebné engine, knihovny a aplikace jsou volně k dispozici na stránkách společnosti Microsoft a MySQL.

Aby mohla aplikace přijímat data z převodníků, tak je nutné nastavit výjimku v použitém firewallu. Ve výchozím nastavení aplikace přijímá zprávy na TCP portu 1001. Tento port je také nutné mít povolený v routeru a v hardwarovém firewallu. Server PCO musí mít veřejnou IP adresu, na kterou se budou převodníky připojovat.

Aplikace umožňuje přijímat zprávy z více převodníků. Pro přidání nového zařízení je zapotřebí zadat název zařízení, které je dále použito pouze pro účely této aplikace. Dalším parametrem je prefix objektu, který musí být v databázi zařízení jedinečný, protože podle tohoto parametru se převodníky od sebe identifikují. Zadaný prefix je číselný v desítkové soustavě. Přenášený prefix je v hexadecimálním tvaru. Dalším nastavitelným parametrem je maximální interval, kdy aplikace nepřijme z převodníku žádná validní data. Tento

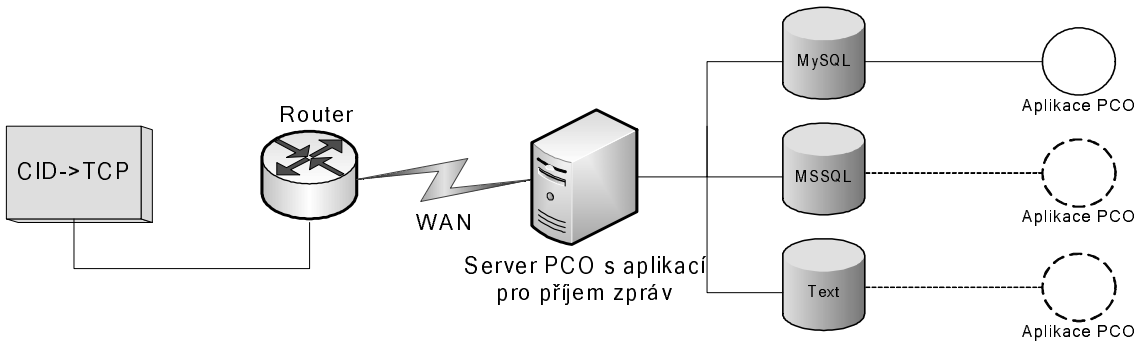
parametr by se měl nastavovat delší než interval mezi kontrolními zprávami u převodníku. Posledními parametry jsou šifrovací klíč a inicializační vektor, který může být pro každý převodník jiný.

Přijatá data se ukládají do SQL databází nebo do textového souboru. U SQL databází je zapotřebí nastavit adresu SQL serveru, přihlašovací údaje a název databáze, kam se mají data ukládat. Dalším parametrem je SQL příkaz, kterým se budou data do databáze ukládat. Příklad jak může tento SQL příkaz vypadat je zde:

```
INSERT INTO cidtcp.cid_table (acct, mess, ts) VALUES
(%A,'%Q%XYZ %GG %CCC','%T');
```

Pro ukládání dat do textového souboru, se nejprve zvolí cesta k souboru. Poté se zvolí data, která mají být uložena. V tomto případě se použijí stejné zástupné řetězce jako u SQL databáze.

Funkce aplikace pro příjem zpráv z převodníku je znázorněna v blokovém schématu na obr. 6.2. Přijaté zprávy na serveru jsou prostřednictvím aplikace pro příjem zpráv uloženy do databáze, odkud si tyto data převezme aplikace PCO.



Obr. 6.2: Blokové schéma funkce aplikace pro příjem zpráv

6.4 Popis činnosti v reálném provozu

V době, kdy zařízení EZS nevysílá žádné zprávy, tak převodník v pravidelných intervalech odesílá kontrolní dohledové zprávy. Tyto kontrolní dohledové zprávy zaznamená aplikace pro příjem zpráv, ale neukládá je do žádné databáze. Pouze vynuluje proměnou zaznamenávající interval od poslední validní zprávy. V případě, že se po nastavenou dobu nepodaří odeslat žádnou validní zprávu, tak se na PC PCO objeví systémová zpráva o nečinnost zařízení.

Jakmile zařízení EZS zaznamená událost, kterou chce odeslat na PCO, tak komunikátor EZS provede volbu telefonního čísla převodníku. Po úspěšné volbě telefonního čísla se přenese zpráva protokolu Contact ID, který je popsán v kapitole 2.1. Okamžitě po přijetí zprávy naváže převodník TCP spojení s pultem centrální ochrany a odešle zpracovanou zprávu, která je popsána v kapitole 2.2. Pokud je přenesená zpráva v pořádku, tak PCO odešle pozitivní potvrzení a TCP spojení se ukončí. Pokud se jedná o zprávu, kterou si

uživatel přeje odeslat prostřednictvím e-mailu, tak se po ukončení spojení s PCO naváže spojení se SMTP serverem a odešle se e-mailová zpráva na zvolenou adresu.

Pokud je aktivován překlad čísla objektu, tak přijaté číslo objektu ze zprávy protokolu Contact ID ze zařízení objektu bude ignorováno a pro odeslání na PCO bude použito nastavené číslo objektu. Pokud překlad čísla objektu není aktivován, tak příchozí číslo objektu ze zprávy ze zařízení EZS je použito i pro odeslání na PCO.

6.5 Testování v reálném provozu

Při testování převodníku v reálném zapojení byla použita zabezpečovací ústředna JA-82K s komunikátorem JA-80V od společnosti Jablotron. Připojení k převodníku bylo provedeno dvouvodičovým telefonním kabelem o délce 10m. Délka kabelu mezi PoE zdrojem a převodníkem byla 3m a od PoE zdroje k modemu byla délka kabelu 10m.

Připojení k internetu bylo realizováno prostřednictvím ADSL připojením s maximálním rychlosti stahování dat 3,5Mb/s a s rychlostí odesílání dat 256kb/s. Na modemu nebyl nastaven QoS a v průběhu testu bylo připojení paralelně používáno ke stahování dat o průměrné rychlosti 3,2Mb/s jako simulace zátěže internetového připojení. Internetové připojení mělo neveřejnou IP adresu. Převodník získal IP nastavení z DHCP serveru.

Aplikace pro příjem zpráv z převodníku byla nainstalována na serveru s veřejnou IP adresou. Server je připojen k poskytovateli prostřednictvím WiFi sítě pracující v kmitočtovém pásmu 5GHz na vzdálenost 3,5km. Rychlost internetového připojení serveru je symetricky 6Mb/s s vytížením v průběhu testu na 40 až 60%. Ukládání dat bylo provedeno do MySQL databáze nacházející se na jiném serveru ve stejné lokální sítí.

Zasílání kontrolních zpráv z převodníku bylo nastavené v intervalu 10 vteřin. V aplikaci byla nastavena doba hlídání nečinnosti zařízení 15 vteřin.

Prvních 20 minut testu zařízení EZS neodeslalo žádnou zprávu. Přenos v této době nevykázal žádné chyby. Poté byl simulován odchod z objektu. Zpráva se přenesla do aplikace bez žádného problému. Doba mezi zakódováním objektu a přijetím zprávy do aplikace byla 8 vteřin. E-mailová zpráva byla doručena také v pořádku.

Po dalších 10 minutách byl vyhlášen několikanásobný poplach (poplach ve smyčce 1 a 2, následovaný sabotáží ústředny EZS). Všechny tři zprávy byly odeslány v pořádku na server v celkovém čase 16 vteřin od prvního poplachu. E-mailové zprávy byly opět doručeny bez problému. V době, kdy převodník přijímal zprávy z ústředny EZS, se na serveru objevila systémová zpráva, která oznamovala, že zařízení neodesílá žádné zprávy, ovšem 4 vteřiny poté se opět objevila zpráva, že již zprávy odesílá a přijaté zprávy se objevily v hlavním okně aplikace.

Po zrušení poplachu a příchodu oprávněné osoby následoval interval 20 minut, kdy opět zařízení EZS neodeslalo žádnou zprávu. Po těchto 20 minutách byl simulován výpadek spojení a zároveň bylo provedeno zakódování objektu. Během 7 vteřin od přerušení spojení se na serveru objevila systémová zpráva, že zařízení neodesílá žádné zprávy. Výpadek

byl simulován na dobu 1 minuty. Okamžitě po konci výpadku byla zpráva o zakódování objektu odeslána na server. E-mailová zpráva přišla během několika vteřin po přijetí zprávy do aplikace na serveru.

Dalším simulovaným výpadkem byl výpadek na straně serveru. Zde se pouze aplikace pro příjem zpráv ukončila a bylo provedeno odkódování objektu. 20 vteřin po odkódování objektu byla přijata e-mailová zpráva o tom, že převodník nemůže odeslat zprávu na pult centrální ochrany.

Veškeré přijaté zprávy na serveru byly také v pořádku zaznamenány v databázi MySQL.

Testování ukázalo, že převodník i aplikace pro příjem zpráv z převodníku pracují správně a přesně jak bylo navrženo. Veškeré zprávy byly doručeny v pořádku a bez zpoždění s výjimkou, kdy byl testován výpadek připojení k internetu. Během testování se objevila jedna chyba a to, že se na krátkou chvíli objevila systémová zpráva o nečinnosti převodníku v době, kdy převodník přijímal velké množství zpráv ze zařízení EZS po telefonní lince.

6.6 Technické parametry převodníku

- **Napájecí napětí:** 24V DC prostřednictvím PoE
- **Maximální příkon:** 15W
- **Průměrný příkon:** 8,5W
- **Napětí telefonní linky:** 24V
- **Maximální proud telefonní linky:** 80mA
- **Zakončovací impedance telefonní linky:** 600Ω
- **Konektor telefonní linky:** RJ-11
- **Protokol pro příjem zpráv z EZS:** Ademco Contact ID DC-05
- **Konektor Ethernetu:** RJ-45
- **Přenosová rychlost Ethernetu:** 10/100Mbit/s
- **Podpora protokolů po Ethernetu:** DHCP, SMTP, ICMP, IGMP, ARP, TCP
- **Protokol pro odesílání zpráv na PCO:** Contact ID DC-09
- **Metoda šifrování zpráv:** AES CBC 128b, 192b nebo 256b dlouhý klíč + 128b dlouhý inicializační vektor

7

Závěr

V rámci této diplomové práce byl navržen a vytvořen funkční prototyp převodníku pro příjem zpráv protokolu Contact ID z telefonní linky na protokol TCP. Převodník při příjmu splňuje normu SIA DC-05 [1], která stanovuje parametry a formát přenosu dat při přenosu zprávy protokolu Contact ID po telefonní lince.

Převodník přijaté zprávy převede do formátu, které splňují normy SIA DC-07 [2] a SIA DC-09 [3]. Tyto zprávy jsou následně odeslány na pult centrální ochrany, které tyto normy podporují, nebo do aplikace pro příjem zpráv z převodníku, která tyto zprávy uloží do MSSQL nebo MySQL databáze, popř. do textového souboru.

Zprávy se mohou přenášet nezašifrované nebo zašifrované. Šifrovaní zprávy je provedeno pomocí AES s metodou CBC s využitím 128b, 192b nebo 256b dlouhým šifrovacím klíčem a 128b dlouhým inicializačním vektorem. Tato šifrovací metoda není v dnešní době ještě překonaná.

Jako doplňková služba byla implementována do převodníku služba SMTP klienta, která uživateli převodníku odešle základní e-mailové zprávy o událostech, které se staly ve střeženém objektu.

Nastavení převodníku je prováděno prostřednictvím připojení TELNET, kde lze nastavit velký počet parametrů pro konfiguraci IP, nastavení časových intervalů na telefonní lince, nastavení odesílání zpráv na PCO, nastavení e-mailových zpráv, výstupní úrovně signálu na telefonní lince apod.

Výroba prototypu stála 2 302Kč. Při sériové výrobě 100ks se tato cena sníží na 1 371Kč. Tato cena lze také snížit některými úpravami zapojení, jako výběr jiného mikrokontroléru, jiného konektoru RJ45 apod.

Budoucí verze tohoto převodníku může být vylepšena pro uživatele příjemnějšímu nastavování, např. prostřednictvím webových stránek. Dále je možné rozšířit počet přijímaných protokolů na telefonní lince a přidat nové protokoly pro odesílání zpráv na PCO.

Převodník byl testován v reálném provozu. Test byl prováděn v různých situacích (normální přenos zpráv, velký počet zpráv apod.) a převodník vždy přenesl zprávu na PCO kromě doby, kdy byl simulován výpadek internetového spojení. Okamžitě po opětovném navázání internetového spojení byla zpráva bez prodlení odeslána. Aplikace pro příjem

zpráv z převodníku úspěšně všechny zprávy přijala a uložila do požadovaných databází.

Literatura

[1] SIA. *SIA DC-05-1999.09: Digital Communication Standard - Ademco Contact ID Protocol - for Alarm System Communications.* SIA, 1999.

[2] SIA. *SIA DC-07-2001.12: Digital Communications Standard Receiver-to-Computer Interface Protocol for Central Station Equipment Communications* SIA, 2001.

[3] SIA. *SIA DC-09-2012A: SIA Digital Communication Standard – Internet Protocol Event Reporting.* SIA, 2012.

[4] NIST. *NIST 800-38A: Recommendation for Block Cipher Modes of Operation.* NIST, 2001.

[5] ÚNMZ. *ČSN EN 50136-1-5: Poplachové systémy - Poplachové přenosové systémy a zařízení - Požadavky na paketově přepínanou síť PSN.* ÚNMZ, 2009.

[6] NXP. *UM10211 LPC23xx User Manual.* NXP, 2012.

[7] NXP. *AN11008 Flash based non-volatile storage.* NXP, 2011.

[8] Micrel. *KSZ8721BL/SL Datasheet rev.1.3.* Micrel, 2009.

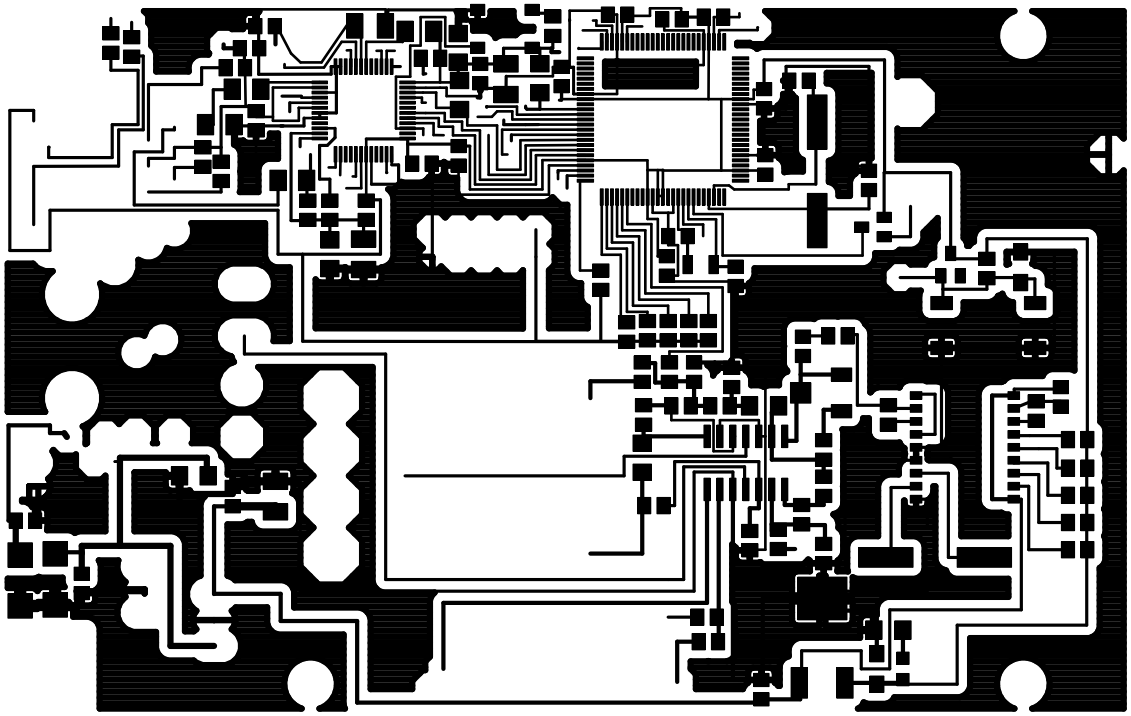
[9] Mitel. *MT8870D/MT8870D-1 Datasheet.* Mitel, 2005.

[10] Tanaka, Inaka. *Optimized AES source code for embedded systems*[online] 2012 [Cit. 25.3.2013]. Dostupné z: http://embeddedknowledge.blogspot.cz/2012/03/optimized-aes-source-code-for-embedded.html

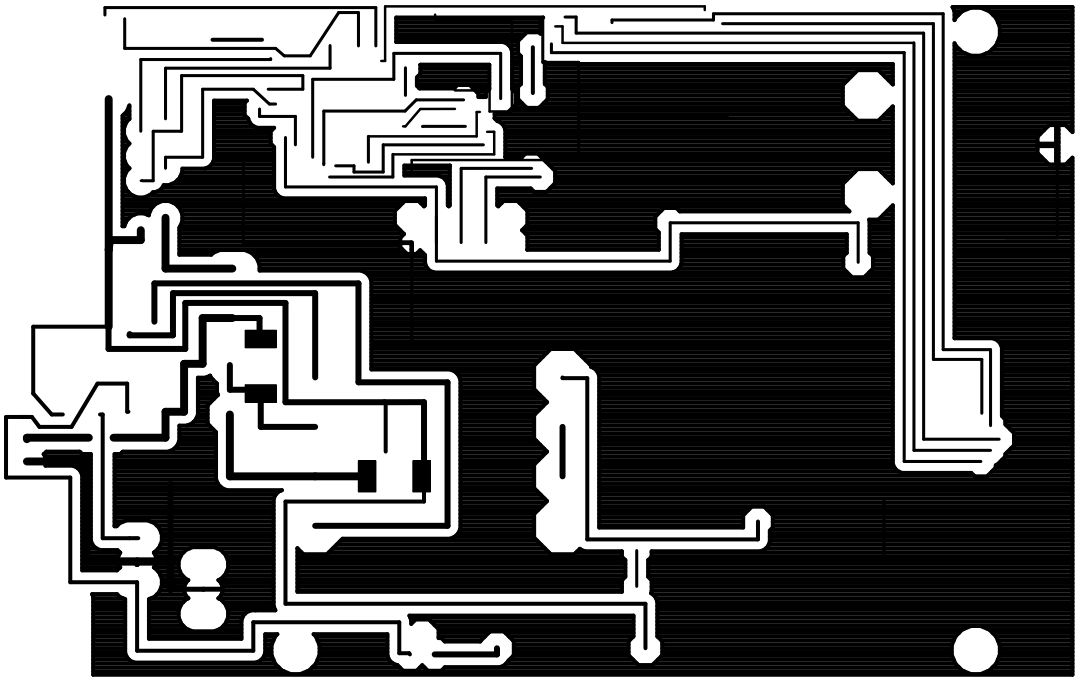
Příloha A

Návrh DPS

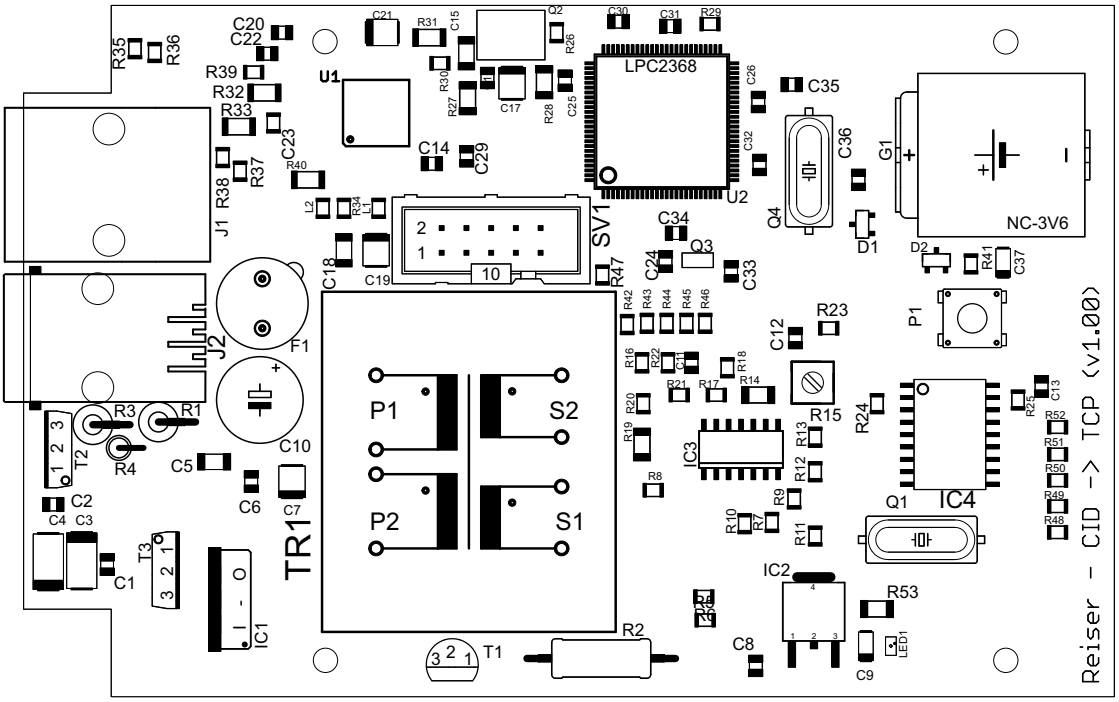
A.1 Výsledné návrhy DPS



Obr. A.1: TOP vrstva návrhu DPS převodníku



Obr. A.2: BOT vrstva návrhu DPS převodníku



Obr. A.3: Vrstva nepájivé masky návrhu DPS převodníku

A.2 Seznam součástek a jejich ceny

Označení	Hodnota	Pouzdro	Cena 1ks	Cena 100ks	ks
R17	150	0805	0,50 Kč	0,36 Kč	1
R53	150	1206	0,50 Kč	0,24 Kč	1
R16, R35, R36	300	0805	0,50 Kč	0,36 Kč	3
R19	300	1206	0,50 Kč	0,24 Kč	1
R18	600	0805	2,08 Kč	1,50 Kč	1
R22	680	0805	0,50 Kč	0,24 Kč	1
R4	10/0,5W	0207	1,95 Kč	0,25 Kč	1
R3	10/1W	0411	1,97 Kč	0,35 Kč	1
R2	100/1W	0411	1,97 Kč	0,35 Kč	1
R23, R24	100k	0805	0,50 Kč	0,36 Kč	2
R20, R21, R26, R27, R29, R34, R41, R42, R43, R44, R45, R46, R47, R48, R49, R50, R51, R52	10k	0805	0,50 Kč	0,36 Kč	18
R28	10k	1206	0,50 Kč	0,24 Kč	1
R7, R8, R9, R10, R11,	1k	0805	0,50 Kč	0,24 Kč	7
R12, R13 R14, R31	1k	1206	0,50 Kč	0,24 Kč	2
R1	1k/1W	0411	1,97 Kč	0,35 Kč	1
R33	1k5/1%	1206	1,00 Kč	0,85 Kč	1
R25	300k	0805	0,50 Kč	0,24 Kč	1
R32	4.99k/1%	1206	67,75 Kč	30,00 Kč	1
R5, R6	47k	0805	0,50 Kč	0,24 Kč	2
R37, R38, R39	49.9/1%	0805	1,00 Kč	0,70 Kč	3
R40	49.9/1%	1206	1,75 Kč	1,25 Kč	1
R30	4k7	0805	0,50 Kč	0,24 Kč	1
R15	22k	3314	36,25 Kč	31,00 Kč	1

Tab. A.1: Seznam rezistorů v převodníku vč. cen za 1ks při nákupu 1ks a 100ks

Označení	Hodnota	Pouzdro	Cena 1ks	Cena 100ks	ks
C1, C2, C6, C8, C11, C12, C13, C14, C16, C18, C18, C20, C22, C23, C24, C25, C26, C27, C28, C29, C30, C31, C32	100n	0805	1,25 Kč	0,77 Kč	22
C15	100n	1206	1,25 Kč	0,65 Kč	1
C10	100u / 35V	Axiální 8,5mm	4,94 Kč	3,27 Kč	1
C7, C17, C19, C21	10u	SMD B	2,90 Kč	2,40 Kč	4
C37	1u	SMD A	2,10 Kč	1,53 Kč	1
C9	2,2u	SMD A	1,90 Kč	1,57 Kč	1
C33, C34, C35, C36	22p	0805	0,91 Kč	0,50 Kč	4
C5	330n	1206	1,65 Kč	1,25 Kč	1
C3, C4	47u	SMD C	8,50 Kč	6,80 Kč	2

Tab. A.2: Seznam kondenzátoru v převodníku vč. cen za 1ks při nákupu 1ks a 100ks

Označení	Součástka	Hodnota	Pouzdro	Cena 1ks	Cena 100ks	ks
D1	Dioda	BAT54C	SOT23	4,23 Kč	2,53 Kč	1
D2	Dioda	BAS21	SOT23	3,80 Kč	2,50 Kč	1
LED1	LED Dioda	Zelená (Uf=2,2V)	0805	2,75 Kč	2,38 Kč	1
T1	Tranzistor NPN	BC337	TO92	1,60 Kč	1,28 Kč	1
T2	Tranzistor NPN	BD135	TO126	6,35 Kč	5,08 Kč	1
T3	Tranzistor PNP	BD136	TO126	6,75 Kč	5,30 Kč	1

Tab. A.3: Seznam diskrétních polovodičů v převodníku vč. cen za 1ks při nákupu 1ks a 100ks

Označení	Součástka	Typ	Pouzdro	Cena 1ks	Cena 100ks	ks
IC1	U regulátor	7805	TO220	15,34 Kč	6,72 Kč	1
IC2	U regulátor	LF33CDT	DPAK	36,93 Kč	19,64 Kč	1
IC3	Operační zes.	LM324D	SO14	7,56 Kč	5,45 Kč	1
IC4	DTMF receiver	MT8870SO	SO-18W	33,20 Kč	33,20 Kč	1
U1	Ethernet PHY	KZS8721BL	TQFP48	179,54 Kč	88,79 Kč	1
U2	uKontrolér	LPC2368	TQFP100	401,13 Kč	286,01 Kč	1

Tab. A.4: Seznam integrovaných obvodů v převodníku vč. cen za 1ks při nákupu 1ks a 100ks

Označení	Součástka	Hodnota/Typ	Pouzdro	Cena 1ks	Cena 100ks	ks
F1	Pojistka	650mA	5mm	14,52 Kč	9,44 Kč	1
G1	Baterie	3,6V	Varta 3V60R	49,00 Kč	31,85 Kč	1
J1	Konektor	RJ45	1-6605310-1	240,79 Kč	156,51 Kč	1
J2	Konektor	RJ11	520250-1	31,46 Kč	20,45 Kč	1
L1, L2	Tlumivka	FB0805	0805	9,98 Kč	6,49 Kč	2
P1	Tlačítko	FSM4JSMA	FSM4JSMA	6,35 Kč	4,13 Kč	1
Q1	Krystal	3.579546MHz	HC49S	11,50 Kč	7,47 Kč	1
Q2	Kryst. rezon.	ASV-50MHZ-EJ-T	7050	48,10 Kč	31,26 Kč	1
Q3	Krystal	32.768kHz	3215	15,73 Kč	10,23 Kč	1
Q4	Krystal	12MHz	HC49S	13,31 Kč	8,65 Kč	1
SV1	Konektor	MLW10G	ML10	5,50 Kč	3,58 Kč	1
TR1	Trafo	TY-146P		84,13 Kč	84,13 Kč	1

Tab. A.5: Seznam ostatních součástek v převodníku vč. cen za 1ks při nákupu 1ks a 100ks

Příloha B

Zdrojové kódy

B.1 Zdrojové kódy pro programové vybavení mikrokontroléru

B.1.1 main.c

```
1
2  /*******************************************************
3  Diplomová práce - Bc. Jindrich Reiser
4  ------------------------------------
5
6  Prevodnik protokolu Contact ID z telefonni linky na protokol TCP
7  *******************************************************/
8
9  #include <stdio.h>
10 #include <LPC23xx.H>            // LPC23xx definice
11 #include "set.h"                // Definice pro nastaveni prevodniku
12 #include "IPCID.h"              // Definice IP Contact ID
13 #include "DTMF.h"               // Definice prijmu DMTF
14 #include "tone.h"               // Definice vysilani tonu
15 #include "type.h"               // Definice typu promennych
16 #include "irq.h"                // Definice preruseni
17 #include "rtc.h"                // Definice Real-time Clock
18 #include "var.h"                // Definice promenych
19
20
21 int tim_0 = 0;                 //Promenna pro casovac
22
23 #define MCLK 48000000             /* Hlavni takt procesoru 48MHz (12*4MHz)         */
24 #define TCLK       5              /* Casovac bude mit 5 taktu za sekundu (200ms)       */
25 #define TCNT (MCLK/TCLK/4)        /* Doba casovace              */
26
27
28 BOOL tick;                      //Promenna pro casovac
29
30
31
32 /*******************************************************
33 init - inicializace casovace pro Ethernet
34 *******************************************************/
35
36 static void init () {
37    T1TCR = 1;
38    T1MCR = 3;
39    T1MR0 = TCNT - 1;
40 }
41
42
43
```

```
44
45 /*----------------------------------------------------
46 timer_poll - kontrola casovace pro Ethernet
47 ----------------------------------------------------*/
48
49 static void timer_poll () {
50    if (T1IR & 1)
51                {
52       T1IR = 1;
53       timer_tick ();
54       tick = __TRUE;
55                        tim_0++;
56
57                 if (tim_0 == (5 * IPC_interval))        //Spousteni dohledovych zprav NULL
58                 {
59                         Timer0Handler();
60                         tim_0 = 0;
61                 }
62
63                 if (IPC_phase != 0)
64                 //Spousteni TCP cinnosti dle jednotlivych fazi (IPC_phase = 0 - stand by)
65                 {
66                        IP_check();
67                 }
68
69                 if (SET_reset == 1) set_RESET_setup();        //Spousteni resetu nastaveni
70
71    }
72 }
73
74
75
76 /*--------------------------- main ------------------------------------------*/
77
78 int main (void) {
79         int active_reset = 0;
80
81         init_setup();                           //Inicializace systemoveho nastaveni
82         init_RTC();                             //Spusteni a nastaveni RTC hodin
83         active_reset = get_RESET_setup();
84         //Zjisteni, zda byl pred restartem spusten reset nastaveni
85         if (active_reset == 0)
86         {                                       //Pokud ne, nastavi se promenne z FLASH pameti
87                 get_mac_setup();
88                 get_DTMF_setup ( );
89                 get_PCO_setup ( );
90                 get_MAIL_setup ( );
91                 get_TONE_setup ( );
92         }
93         else
94         {
95         //Pokud ano, provede se nahrani vychoziho nastaveni do FLASH
96                 set_DTMF_setup ( );
97                 set_PCO_setup ( );
98                 set_MAIL_setup ( );
99                 set_TONE_setup ( );
100                set_DHCP_setup();
101        }
102
103        init ();                                //Start casovace
104  init_TcpNet ();                           //Inicializace Ethernetu a funkci TCP stacku
105        if (active_reset == 0) get_ip_setup();
106        //Nacteni nastaveni IP - pouze v pripade, ze nebyl spusten reset nastaveni
107        tone_init();                            //Inicializace D/A prvodniku
108        DTMF_init();                            //Inicializace prijmu dat z MT8870DS (DTMF receiver)
109        init_IPC();                             //Inicializace odesilani dat na PCO
110
111        while (1)
112        {
113                if (DTMF_phase == 0)
114                //V pripade pouzivani tlf. linky, docasne nepouzivat Ethernet
115                {
116                timer_poll ();
```

```
117     main_TcpNet ();
118                 }
119   }
120 }
121
```

B.1.2 Změněná část LPC23_EMAC.c

```
 1
 2 void PHYInit( void )
 3 {
 4   int i, regValue;
 5   int timeout;
 6   //Nastaveni RMII rozhranni, deleni hodin 20-ti, nevuzivani preambule a neskenovani inkrementace
 7   MAC_MCFG = 0x801F;
 8   for ( i = 0; i < 0x40; i++ );         //Kratka pauza pred restartem
 9   MAC_MCFG = 0x001F;                    //Restart
10   MAC_MCMD = 0;
11
12  //Nastaveni, ze se jedna o RMII
13   MAC_COMMAND |= 0x0200;
14   MAC_SUPP = 0x0100;
15
16         //Vyhledani fyzicke adresy fyzicke vrstvy Ethernetu
17         for(PHY_ADDR=0;PHY_ADDR<0x1F;PHY_ADDR++)
18         {
19                 regValue = read_PHY (0x00);
20                 if (regValue != 0xFFFF) break;
21         }
22
23
24   for ( i = 0; i < 0x100; i++ );    //Kratka pauza
25   write_PHY( 0x0000, 0x8000 );      //Restart PHY
26   for ( i = 0; i < 0x20; i++ );     //Kratka pauza
27
28         //Vyckani na dokonceni restartu PHY
29   timeout = 0x1000000 * 4;
30   while ( timeout != 0 )
31   {
32                 regValue = read_PHY( 0x0000 );
33                 if ( (regValue & 0x8000) == 0x0000 )        break;
34                 timeout--;
35   }
36
37   write_PHY( 0x0000, 0x1000 | 0x0200 );        //Nastaveni PHY
38          //Vyckani na dokonceni nastaveni PHY
39         timeout = 0x1000000;
40         while ( timeout != 0 )
41         {
42           regValue = read_PHY( 0x0001 );
43                 if ( (regValue & (0x0020|0x0004))        == (0x0020|0x0004) )        break;
44           timeout--;
45         }
46
47
48         //Nacteni informaci o rychlosti a typu spojeni
49         regValue = read_PHY( 0x001F );
50         regValue &= 0x001C;
51
52         switch ( regValue )
53         {
54           case 0x0004:                      //10MBit - Half Duplex
55                         Speed = SPEED_10;
56                         Duplex = HALF_DUPLEX;
57                         break;
58           case 0x0008:
59                         Speed = SPEED_100;          //100MBit - Half Duplex
60                         Duplex = HALF_DUPLEX;
61                         break;
62           case 0x0014:
63                         Speed = SPEED_10;           //10MBit - Full Duplex
```

```
64                          Duplex = FULL_DUPLEX;
65                          break;
66            case 0x0018:
67                          Speed = SPEED_100;          //100MBit - Full Duplex
68                          Duplex = FULL_DUPLEX;
69                          break;
70            default:
71            //100MBit - Full Duplex - nepodarilo se korektni nacteni, bude pouzita vychozi hodnota
72                          Speed = SPEED_100;
73                          Duplex = FULL_DUPLEX;
74                          break;
75          }
76
77    return;
78          }
79
80  /*--------------------------- init_ethernet ---------------------------------*/
81
82  //Tato funkce je oproti originalu upravena
83  void init_ethernet (void) {
84    int regVal;
85    int i;
86
87    //Aktivace periferie
88    regVal = PCONP;
89    regVal |= 0x40000000;
90    PCONP = regVal;
91
92    for ( i = 0; i < 0x100; i++ );             //Kratka pauza
93          regVal = MAC_MODULEID;                   //Zjisteni typu procesoru
94          //Nastaveni vstupu a vystup
95    if ( regVal == ((0x3902 << 16) | 0x2000) )
96    {
97                  PINSEL2 = 0x50151105;               //Starsi typ (jeste Philips)
98    }
99    else
100   {
101                 PINSEL2 = 0x50150105;               //Novejsi typ (jiz NXP)
102   }
103   PINSEL3 = 0x00000005;
104
105
106
107         MAC_MAC1 = 0xCF00;                      //Reset MAC vrstvy
108   MAC_COMMAND = 0x0038;                       //Reset nastaveni
109
110         //Kratka pauza a reset
111   for ( i = 0; i < 0x04; i++ );
112   MAC_MAC1 = 0x0;
113
114         //Vypnuti prijmu a odesilani dat a nastaveni MAC do zakladniho nastaveni
115   EMAC_TxDisable();
116   EMAC_RxDisable();
117         MAC_MAC2 = 0x00;
118
119   //Nastaveni MAC vrstvy - doporucen nastaveni z User Manualu
120   MAC_IPGR = 0x0012;
121   MAC_CLRT = 0x370F;
122   MAC_MAXF = 0x0600;
123
124         PHYInit();        //Spusteni inicializace PHY
125
126
127
128
129   /* Nastaveni MAC adresy */
130   MAC_SA0 = ((U32)own_hw_adr[5] << 8) | (U32)own_hw_adr[4];
131   MAC_SA1 = ((U32)own_hw_adr[3] << 8) | (U32)own_hw_adr[2];
132   MAC_SA2 = ((U32)own_hw_adr[1] << 8) | (U32)own_hw_adr[0];
133
134         //Nastaveni rychlosti a parametru linky
135         if ( (Speed == SPEED_10) && (Duplex == HALF_DUPLEX) )
136   {
```

```
137                 MAC_MAC2 = 0x30;
138                 MAC_SUPP &= ~0x0100;
139                 MAC_COMMAND |= 0x0240;
140                 MAC_IPGT = 0x0012;
141   }
142   else if ( (Speed == SPEED_100) && (Duplex == HALF_DUPLEX) )
143   {
144                 MAC_MAC2 = 0x30;
145                 MAC_SUPP |= 0x0100;
146                 MAC_COMMAND |= 0x0240;
147                 MAC_IPGT = 0x0012;
148   }
149   else if ( (Speed == SPEED_10) && (Duplex == FULL_DUPLEX) )
150   {
151                 MAC_MAC2 = 0x31;
152                 MAC_SUPP &= ~0x0100;
153                 MAC_COMMAND |= 0x0640;
154                 MAC_IPGT = 0x0015;
155   }
156   else if ( (Speed == SPEED_100) && (Duplex == FULL_DUPLEX) )
157   {
158                 MAC_MAC2 = 0x31;
159                 MAC_SUPP |= 0x0100;
160                 MAC_COMMAND |= 0x0640;
161                 MAC_IPGT = 0x0015;
162   }
163   /* Inicializace DMA pro prijem a odesilani */
164   rx_descr_init ();
165   tx_descr_init ();
166
167   MAC_MAC1 = 0x0002;             //Prijem vsech ramcu
168
169   /* Nastaveni RX filtru */
170         MAC_RXFILTERCTRL = RFC_MCAST_EN | RFC_BCAST_EN | RFC_PERFECT_EN;
171
172   MAC_INTCLEAR = 0xFFFF;        /* Vycisteni presruseni */
173   /* Nastaveni preruseni */
174         MAC_INTENABLE = INT_RX_DONE | INT_TX_DONE;
175         //Zapnuti prijmu a odesilani dat
176         MAC_COMMAND |= 0x01;
177   MAC_MAC1 |= 0x01;
178         MAC_COMMAND |= 0x02;
179
180            /* Konfigurace preruseni */
181   VICVectAddr21 = (U32)interrupt_ethernet;
182         return;
183 }
184
```

B.1.3 set.c

```
1
2  /*********************************************
3  set.c - funkce pro nastavovani
4
5  *********************************************/
6
7  #include "LPC23xx.h"            /* LPC23xx/24xx definice */
8  #include "type.h"               // Definice typu promennych
9  #include "rtc.h"                // Definice Real-time Clock
10 #include "flash_nvol.h"         // Definice ukladani do Flash pameti
11 #include "setup.h"              // Definice promenych
12 #include <net_config.h>         // Definice nastaveni RL-TCPnet
13 #include <stdio.h>
14 #include <string.h>
15
16
17 extern struct tnet_cfg  tnet_config;        //Nastaveni TELNETu
18 #define tnet_auth_passw tnet_config.Passw   //Pristupove heslo TELNETu
19
20 RTCTime local_time;                         //Promenna pro RTC cas
```

```
21 extern LOCALM localm[];                    //Promenna pro nastaveni IP adres
22 extern U8 own_hw_adr[];                    //Promenna pro nastaveni MAC adresy
23 extern U8 lhost_name[];                    //Promenna pro nastaveni sitoveho nazvu zarizeni
24
25
26 /*****************************************************
27 Funkce: set_DHCP_setup
28 Zapnuti DHCP po resetu nastaveni
29 *****************************************************/
30 void set_DHCP_setup(void)
31 {
32         int dhcp = 1;
33         NVOL_SetVariable(FLASH_IP_DHCP, (UNSIGNED8 *)&dhcp, sizeof(dhcp));
34         return;
35 }
36
37
38 /*****************************************************
39 Funkce: set_TONE_setup
40 Ziskani hodnot pro TONE
41 *****************************************************/
42 void set_TONE_setup(void)
43 {
44         NVOL_GetVariable(FLASH_TONE_LEV, (UNSIGNED8 *)&TONE_level, sizeof(TONE_level));
45         return;
46 }
47
48 /*****************************************************
49 Funkce: get_TONE_setup
50 Ziskani hodnot pro TONE
51 *****************************************************/
52 void get_TONE_setup(void)
53 {
54         NVOL_GetVariable(FLASH_TONE_LEV, (UNSIGNED8 *)&TONE_level, sizeof(TONE_level));
55         return;
56 }
57
58
59 /*****************************************************
60 Funkce: set_MAIL_setup
61 Ziskani hodnot pro MAIL
62 *****************************************************/
63 void set_MAIL_setup (void)
64 {
65         char pom_adr[16];
66         int pom_set;
67         int pom_MAIL_en_dp;
68         int pom_MAIL_en_ipc;
69         int pom_MAIL_en_10;
70         int pom_MAIL_en_11;
71         int pom_MAIL_en_12;
72         int pom_MAIL_en_13;
73         int pom_MAIL_en_14;
74         int pom_MAIL_en_15;
75         int pom_MAIL_en_20;
76         int pom_MAIL_en_3;
77         int pom_MAIL_en_40;
78
79         NVOL_SetVariable(FLASH_IPC_ME, (UNSIGNED8 *)&IPC_maxerr, sizeof(IPC_maxerr));
80         NVOL_SetVariable(FLASH_MAIL_FROM, (UNSIGNED8 *)&MAIL_from, sizeof(MAIL_from));
81         NVOL_SetVariable(FLASH_MAIL_TO, (UNSIGNED8 *)&MAIL_to, sizeof(MAIL_to));
82         NVOL_SetVariable(FLASH_MAIL_AUTH, (UNSIGNED8 *)&MAIL_auth, sizeof(MAIL_auth));
83         NVOL_SetVariable(FLASH_MAIL_USER, (UNSIGNED8 *)&MAIL_user, sizeof(MAIL_user));
84         NVOL_SetVariable(FLASH_MAIL_PASS, (UNSIGNED8 *)&MAIL_pass, sizeof(MAIL_pass));
85         sprintf(pom_adr,"%03d %03d %03d %03d",MAIL_adr[0],MAIL_adr[1],MAIL_adr[2],MAIL_adr[3]);
86         NVOL_SetVariable(FLASH_MAIL_ADR, (UNSIGNED8 *)&pom_adr, sizeof(pom_adr));
87         //Ulozeni parametru pro odesilani mailu
88         pom_MAIL_en_dp = MAIL_en_dp * 0x01;
89         pom_MAIL_en_ipc = MAIL_en_ipc *  0x02;
90         pom_MAIL_en_10 = MAIL_en_10 * 0x04;
91         pom_MAIL_en_11 = MAIL_en_11 * 0x08;
92         pom_MAIL_en_12 = MAIL_en_12 * 0x10;
93         pom_MAIL_en_13 = MAIL_en_13 * 0x20;
```

```
94          pom_MAIL_en_14 = MAIL_en_14 * 0x40;
95          pom_MAIL_en_15 = MAIL_en_15 * 0x80;
96          pom_MAIL_en_20 = MAIL_en_20 * 0x100;
97          pom_MAIL_en_3 = MAIL_en_3 * 0x200;
98          pom_MAIL_en_40 = MAIL_en_40 * 0x400;
99          pom_set = pom_MAIL_en_dp + pom_MAIL_en_ipc + pom_MAIL_en_10 + pom_MAIL_en_11 + pom_MAIL_en_12 +
100         pom_MAIL_en_13 + pom_MAIL_en_14 + pom_MAIL_en_15 + pom_MAIL_en_20 + pom_MAIL_en_3 +
101         pom_MAIL_en_40;
102         NVOL_SetVariable(FLASH_MAIL_SET, (UNSIGNED8 *)&pom_set, sizeof(pom_set));
103         return;
104 }
105
106
107 /*********************************************************
108 Funkce: get_MAIL_setup
109 Ziskani hodnot pro MAIL
110 *********************************************************/
111 void get_MAIL_setup (void)
112 {
113         char pom_from[33];
114         char pom_to[33];
115         char pom_user[33];
116         char pom_pass[33];
117         char pom_adr[16];
118         int pom_set;
119         U8 ip1;
120         U8 ip2;
121         U8 ip3;
122         U8 ip4;
123
124         NVOL_GetVariable(FLASH_IPC_ME, (UNSIGNED8 *)&IPC_maxerr, sizeof(IPC_maxerr));
125         if(NVOL_GetVariable(FLASH_MAIL_FROM, (UNSIGNED8 *)&pom_from, sizeof(pom_from)))
126         {
127                 sscanf(pom_from,"%s",MAIL_from);
128         }
129         if(NVOL_GetVariable(FLASH_MAIL_TO, (UNSIGNED8 *)&pom_to, sizeof(pom_to)))
130         {
131                 sscanf(pom_to,"%s",MAIL_to);
132         }
133         NVOL_GetVariable(FLASH_MAIL_AUTH, (UNSIGNED8 *)&MAIL_auth, sizeof(MAIL_auth));
134         if(NVOL_GetVariable(FLASH_MAIL_USER, (UNSIGNED8 *)&pom_user, sizeof(pom_user)))
135         {
136                 sscanf(pom_user,"%s",MAIL_user);
137         }
138         if(NVOL_GetVariable(FLASH_MAIL_PASS, (UNSIGNED8 *)&pom_pass, sizeof(pom_pass)))
139         {
140                 sscanf(pom_pass,"%s",MAIL_pass);
141         }
142         if(NVOL_GetVariable(FLASH_MAIL_ADR, (UNSIGNED8 *)&pom_adr, sizeof(pom_adr)))
143         {
144                 sscanf(pom_adr,"%d %d %d %d",&ip1,&ip2,&ip3,&ip4);
145                 MAIL_adr[0] = ip1;
146                 MAIL_adr[1] = ip2;
147                 MAIL_adr[2] = ip3;
148                 MAIL_adr[3] = ip4;
149         }
150         //Ziskani parametru pro odesilani mailu
151         if(NVOL_GetVariable(FLASH_MAIL_SET, (UNSIGNED8 *)&pom_set, sizeof(pom_set)))
152         {
153          MAIL_en_dp = pom_set && 0x01;
154          MAIL_en_ipc = pom_set && 0x02;
155          MAIL_en_10 = pom_set && 0x04;
156          MAIL_en_11 = pom_set && 0x08;
157          MAIL_en_12 = pom_set && 0x10;
158          MAIL_en_13 = pom_set && 0x20;
159          MAIL_en_14 = pom_set && 0x40;
160          MAIL_en_15 = pom_set && 0x80;
161          MAIL_en_20 = pom_set && 0x100;
162          MAIL_en_3 = pom_set && 0x200;
163          MAIL_en_40 = pom_set && 0x400;
164         }
165         return;
166 }
```

```
167
168
169 /*******************************************************
170 Funkce: Set_PCO_setup
171 Ziskani hodnot pro PCO
172 *******************************************************/
173 void set_PCO_setup (void)
174 {
175         unsigned char pom_adr[16];
176         char pom_rcvr[7];
177         char pom_pref[7];
178         char pom_acct[5];
179
180         sprintf(pom_adr,"%03d %03d %03d %03d",IPC_ADR[0],IPC_ADR[1],IPC_ADR[2],IPC_ADR[3]);
181         NVOL_SetVariable(FLASH_IPC_ADR, (UNSIGNED8 *)&pom_adr, sizeof(pom_adr));
182         NVOL_SetVariable(FLASH_IPC_PORT, (UNSIGNED8 *)&IPC_PORT, sizeof(IPC_PORT));
183         NVOL_SetVariable(FLASH_IPC_PROT, (UNSIGNED8 *)&IPC_PROTOCOL, sizeof(IPC_PROTOCOL));
184         sprintf(pom_rcvr,"%06x",IPC_rcvr);
185         NVOL_SetVariable(FLASH_IPC_RCVR, (UNSIGNED8 *)&pom_rcvr, sizeof(pom_rcvr));
186         sprintf(pom_pref,"%06x",IPC_pref);
187         NVOL_SetVariable(FLASH_IPC_PREF, (UNSIGNED8 *)&pom_pref, sizeof(pom_pref));
188         sprintf(pom_acct,"%04x",IPC_acct);
189         NVOL_SetVariable(FLASH_IPC_ACCT, (UNSIGNED8 *)&pom_acct, sizeof(pom_acct));
190         NVOL_SetVariable(FLASH_IPC_ANT, (UNSIGNED8 *)&IPC_ANT, sizeof(IPC_ANT));
191         NVOL_SetVariable(FLASH_IPC_TS, (UNSIGNED8 *)&IPC_ts, sizeof(IPC_ts));
192         NVOL_SetVariable(FLASH_IPC_INT, (UNSIGNED8 *)&IPC_interval, sizeof(IPC_interval));
193         NVOL_SetVariable(FLASH_IPC_KEY, (UNSIGNED8 *)&IPC_key, sizeof(IPC_key));
194         NVOL_SetVariable(FLASH_IPC_IV, (UNSIGNED8 *)&IPC_iv, sizeof(IPC_iv));
195         return;
196 }
197
198 /*******************************************************
199 Funkce: get_PCO_setup
200 Ziskani hodnot pro PCO
201 *******************************************************/
202 void get_PCO_setup (void)
203 {
204         unsigned char pom_adr[16];
205         char pom_key[33];
206         char pom_iv[33];
207         char pom_rcvr[7];
208         char pom_pref[7];
209         char pom_acct[5];
210         U8 ip1;
211         U8 ip2;
212         U8 ip3;
213         U8 ip4;
214
215         if(NVOL_GetVariable(FLASH_IPC_ADR, (UNSIGNED8 *)&pom_adr, sizeof(pom_adr)))
216         {
217                 sscanf(pom_adr,"%d %d %d %d",&ip1,&ip2,&ip3,&ip4);
218                 IPC_ADR[0] = ip1;
219                 IPC_ADR[1] = ip2;
220                 IPC_ADR[2] = ip3;
221                 IPC_ADR[3] = ip4;
222         }
223
224         NVOL_GetVariable(FLASH_IPC_PORT, (UNSIGNED8 *)&IPC_PORT, sizeof(IPC_PORT));
225         NVOL_GetVariable(FLASH_IPC_PROT, (UNSIGNED8 *)&IPC_PROTOCOL, sizeof(IPC_PROTOCOL));
226
227         if(NVOL_GetVariable(FLASH_IPC_RCVR, (UNSIGNED8 *)&pom_rcvr, sizeof(pom_rcvr)))
228         {
229                 sscanf(pom_rcvr,"%06x%*s",&IPC_rcvr);
230         }
231
232         if(NVOL_GetVariable(FLASH_IPC_PREF, (UNSIGNED8 *)&pom_pref, sizeof(pom_pref)))
233         {
234                 sscanf(pom_pref,"%06x%*s",&IPC_pref);
235         }
236
237         if(NVOL_GetVariable(FLASH_IPC_ACCT, (UNSIGNED8 *)&pom_acct, sizeof(pom_acct)))
238         {
239                 sscanf(pom_acct,"%04x%*s",&IPC_acct);
```

```
240         }
241
242         NVOL_GetVariable(FLASH_IPC_ANT, (UNSIGNED8 *)&IPC_ANT, sizeof(IPC_ANT));
243         NVOL_GetVariable(FLASH_IPC_TS, (UNSIGNED8 *)&IPC_ts, sizeof(IPC_ts));
244         NVOL_GetVariable(FLASH_IPC_INT, (UNSIGNED8 *)&IPC_interval, sizeof(IPC_interval));
245
246         if(NVOL_GetVariable(FLASH_IPC_KEY, (UNSIGNED8 *)&pom_key, sizeof(pom_key)))
247         {
248                 sscanf(pom_key,"%s%*s",IPC_key);
249         }
250
251         if(NVOL_GetVariable(FLASH_IPC_IV, (UNSIGNED8 *)&pom_iv, sizeof(pom_iv)))
252         {
253                 sscanf(pom_iv,"%s%*s",IPC_iv);
254         }
255
256         return;
257 }
258
259
260 /*******************************************************
261 Funkce: set_DTMF_setup
262 Ulozeni hodnot pro DTMF
263 *******************************************************/
264 void set_DTMF_setup (void)
265 {
266         NVOL_SetVariable(FLASH_DTMF_maxEZS, (UNSIGNED8 *)&DTMF_maxEZS, sizeof(DTMF_maxEZS));
267         NVOL_SetVariable(FLASH_DTMF_maxMan, (UNSIGNED8 *)&DTMF_maxMan, sizeof(DTMF_maxMan));
268         NVOL_SetVariable(FLASH_DTMF_befHand, (UNSIGNED8 *)&DTMF_befHand, sizeof(DTMF_befHand));
269         NVOL_SetVariable(FLASH_DTMF_befMess, (UNSIGNED8 *)&DTMF_befMess, sizeof(DTMF_befMess));
270         NVOL_SetVariable(FLASH_DTMF_befKiss, (UNSIGNED8 *)&DTMF_befKiss, sizeof(DTMF_befKiss));
271         NVOL_SetVariable(FLASH_TLF_number, (UNSIGNED8 *)&TLF_number, sizeof(TLF_number));
272         return;
273 }
274
275 /*******************************************************
276 Funkce: get_DTMF_setup
277 Ziskani hodnot pro DTMF
278 *******************************************************/
279 void get_DTMF_setup (void)
280 {
281         NVOL_GetVariable(FLASH_DTMF_maxEZS, (UNSIGNED8 *)&DTMF_maxEZS, sizeof(DTMF_maxEZS));
282         NVOL_GetVariable(FLASH_DTMF_maxMan, (UNSIGNED8 *)&DTMF_maxMan, sizeof(DTMF_maxMan));
283         NVOL_GetVariable(FLASH_DTMF_befHand, (UNSIGNED8 *)&DTMF_befHand, sizeof(DTMF_befHand));
284         NVOL_GetVariable(FLASH_DTMF_befMess, (UNSIGNED8 *)&DTMF_befMess, sizeof(DTMF_befMess));
285         NVOL_GetVariable(FLASH_DTMF_befKiss, (UNSIGNED8 *)&DTMF_befKiss, sizeof(DTMF_befKiss));
286         NVOL_GetVariable(FLASH_TLF_number, (UNSIGNED8 *)&TLF_number, sizeof(TLF_number));
287         return;
288 }
289
290
291 /*******************************************************
292 Funkce: get_mac_setup
293 Ziskani hodnot pro MAC adresu
294 *******************************************************/
295 void get_mac_setup (void)
296 {
297         unsigned char flash_mac[18];
298         U8 m1;
299         U8 m2;
300         U8 m3;
301         U8 m4;
302         U8 m5;
303         U8 m6;
304         U8 mac_adr[6];
305
306         if (NVOL_GetVariable(FLASH_IP_MAC, (UNSIGNED8 *)&flash_mac, sizeof(flash_mac)))
307         //Zjisteni, zda se jedna o prvni start
308   {
309                 sscanf(flash_mac,"%x %x %x %x %x %x",&m1,&m2,&m3,&m4,&m5,&m6);
310                 mac_adr[0] = m1;
311                 mac_adr[1] = m2;
312                 mac_adr[2] = m3;
```

```
313                 mac_adr[3] = m4;
314                 mac_adr[4] = m5;
315                 mac_adr[5] = m6;
316                 mem_copy (&own_hw_adr, &mac_adr, 6);
317         }
318         return;
319 }
320
321 /*************************************************************
322 Funkce: set_RESET_setup
323 RESET nastaveni prevodniku
324 *************************************************************/
325 int set_RESET_setup (void)
326 {
327         unsigned char pom_reset = 1;
328         unsigned char pom2;
329         NVOL_SetVariable(FLASH_RESET_SETUP, (UNSIGNED8 *)&pom_reset, sizeof(pom_reset));
330         if (NVOL_GetVariable(FLASH_RESET_SETUP, (UNSIGNED8 *)&pom2, sizeof(pom2)))
331          NVOL_SetVariable(FLASH_RESET_SETUP, (UNSIGNED8 *)&pom_reset, sizeof(pom_reset));
332         //Watchdog reset
333         WDTC = 0x00000FFF;      // kratky interval
334         WDMOD = 0x03;           // nastaveni resetu MCU
335         WDFEED = 0xAA;          // zapnuti watchdogu
336         WDFEED = 0x55;
337   return;
338 }
339
340
341 /*************************************************************
342 Funkce: get_RESET_setup
343 RESET nastaveni prevodniku
344 *************************************************************/
345 unsigned char get_RESET_setup (void)
346 {
347         unsigned char pom;
348         unsigned char pom2 = 0;
349         if (NVOL_GetVariable(FLASH_RESET_SETUP, (UNSIGNED8 *)&pom, sizeof(pom)))
350   {
351                 NVOL_SetVariable(FLASH_RESET_SETUP, (UNSIGNED8 *)&pom2, sizeof(pom2));
352                 //Navrat RESETu do vychozi hodnoty vypnuto
353                 return pom;
354         }
355         else
356         {
357                 return 0;
358         }
359 }
360
361 /*************************************************************
362 Funkce: get_ip_setup
363 Ziskani hodnot pro IP
364 *************************************************************/
365 void get_ip_setup (void)
366 {
367         unsigned char pom_adr[16];
368         int flash_dhcp;
369         U8 ip1;
370         U8 ip2;
371         U8 ip3;
372         U8 ip4;
373         unsigned char pom[16];
374          U8 ip_adr[4];
375
376         unsigned char flash_dev[15];
377         LOCALM ip_config;
378
379         if (NVOL_GetVariable(FLASH_IP_DHCP, (UNSIGNED8 *)&flash_dhcp, sizeof(flash_dhcp)))
380         //Zjisteni, zda je zapnute DHCP
381   {
382                 if (flash_dhcp == 0)
383                 //Pokud je DHCP vypnute, tak se nacte IP nastaveni
384                 {
385                         dhcp_disable();
```

```
386                     if(NVOL_GetVariable(FLASH_IP_ADR, (UNSIGNED8 *)&pom_adr, sizeof(pom_adr)))
387                     {
388                             sscanf(pom_adr,"%d %d %d %d",&ip1,&ip2,&ip3,&ip4);
389                             ip_adr[0] = ip1;
390                             ip_adr[1] = ip2;
391                             ip_adr[2] = ip3;
392                             ip_adr[3] = ip4;
393                             memcpy(&ip_config.IpAdr,&ip_adr,sizeof(ip_adr));
394                     }
395                     else
396                     //Pokud neni nalezena IP adresa, tak se pouzije vychozi hodnota 192.168.0.100
397                     {
398                             ip_adr[0] = 192;
399                             ip_adr[1] = 168;
400                             ip_adr[2] = 0;
401                             ip_adr[3] = 100;
402                             memcpy(&ip_config.IpAdr,&ip_adr,sizeof(ip_adr));
403                      }
404
405                     if(NVOL_GetVariable(FLASH_IP_SM, (UNSIGNED8 *)&pom_adr, sizeof(pom_adr)))
406                             {
407                             sscanf(pom_adr,"%d %d %d %d",&ip1,&ip2,&ip3,&ip4);
408                             ip_adr[0] = ip1;
409                             ip_adr[1] = ip2;
410                             ip_adr[2] = ip3;
411                             ip_adr[3] = ip4;
412                             memcpy(&ip_config.NetMask,&ip_adr,sizeof(ip_adr));
413                     }
414                     else
415                     //Pokud neni nalezena maska site, tak se pouzije vychozi hodnota 255.255.255.0
416                     {
417                             ip_adr[0] = 255;
418                             ip_adr[1] = 255;
419                             ip_adr[2] = 255;
420                             ip_adr[3] = 0;
421                             memcpy(&ip_config.NetMask,&ip_adr,sizeof(ip_adr));
422                      }
423
424                     if(NVOL_GetVariable(FLASH_IP_GW, (UNSIGNED8 *)&pom_adr, sizeof(pom_adr)))
425                             {
426                             sscanf(pom_adr,"%d %d %d %d",&ip1,&ip2,&ip3,&ip4);
427                             ip_adr[0] = ip1;
428                             ip_adr[1] = ip2;
429                             ip_adr[2] = ip3;
430                             ip_adr[3] = ip4;
431                             memcpy(&ip_config.DefGW,&ip_adr,sizeof(ip_adr));
432                     }
433                     else
434                     //Pokud neni nalezena vychozi brana, tak se pouzije vychozi hodnota 192.168.0.1
435                     {
436                             ip_adr[0] = 192;
437                             ip_adr[1] = 168;
438                             ip_adr[2] = 0;
439                             ip_adr[3] = 1;
440                             memcpy(&ip_config.DefGW,&ip_adr,sizeof(ip_adr));
441                      }
442
443                     if(NVOL_GetVariable(FLASH_IP_DNS1, (UNSIGNED8 *)&pom_adr, sizeof(pom_adr)))
444                             {
445                             sscanf(pom_adr,"%d %d %d %d",&ip1,&ip2,&ip3,&ip4);
446                                     ip_adr[0] = ip1;
447                             ip_adr[1] = ip2;
448                             ip_adr[2] = ip3;
449                             ip_adr[3] = ip4;
450                             memcpy(&ip_config.PriDNS,&ip_adr,sizeof(ip_adr));
451                     }
452                     else
453                     //Pokud neni nalezen DNS server, tak se pouzije vychozi hodnota 0.0.0.0
454                     {
455                             ip_adr[0] = 0;
456                             ip_adr[1] = 0;
457                             ip_adr[2] = 0;
458                             ip_adr[3] = 0;
```

```
459                                 memcpy(&ip_config.PriDNS,&ip_adr,sizeof(ip_adr));
460                          }
461
462                         if(NVOL_GetVariable(FLASH_IP_DNS2, (UNSIGNED8 *)&pom_adr, sizeof(pom_adr)))
463                                 {
464                                 sscanf(pom_adr,"%d %d %d %d",&ip1,&ip2,&ip3,&ip4);
465                                 ip_adr[0] = ip1;
466                                 ip_adr[1] = ip2;
467                                 ip_adr[2] = ip3;
468                                 ip_adr[3] = ip4;
469                                 memcpy(&ip_config.SecDNS,&ip_adr,sizeof(ip_adr));
470                         }
471                         else
472                         //Pokud neni nalezen DNS server, tak se pouzije vychozi hodnota 0.0.0.0
473                         {
474                                 ip_adr[0] = 0;
475                                 ip_adr[1] = 0;
476                                 ip_adr[2] = 0;
477                                 ip_adr[3] = 0;
478                                 memcpy(&ip_config.SecDNS,&ip_adr,sizeof(ip_adr));
479                          }
480
481                          mem_copy (&localm[NETIF_ETH], &ip_config, sizeof(ip_config));
482                          //Ulozeni IP nastaveni do RL-TCPnet stacku
483                 }
484         }
485         //Nacteni nazvu sitoveho zarizeni a nazvu objektu
486         if(NVOL_GetVariable(FLASH_IP_DEV, (UNSIGNED8 *)&flash_dev, sizeof(flash_dev)))
487         {
488                 memcpy (&IPC_name,&flash_dev,sizeof(flash_dev));
489                 memcpy (&lhost_name, &flash_dev,sizeof(flash_dev));
490         }
491
492         //Nacteni hesla pro TELNET
493         if(NVOL_GetVariable(FLASH_TELNET_PASS, (UNSIGNED8 *)&pom, sizeof(pom)))
494         {
495                 sscanf(pom,"%s",tnet_auth_passw);
496         }
497         return;
498 }
499
500
501 /*********************************************************
502 Funkce: RTCSetTime
503 Nastaveni RTC hodin
504 *********************************************************/
505 void RTCSetTime( RTCTime Time )
506 {
507   RTC_SEC = Time.RTC_Sec;
508   RTC_MIN = Time.RTC_Min;
509   RTC_HOUR = Time.RTC_Hour;
510   RTC_DOM = Time.RTC_Mday;
511   RTC_DOW = Time.RTC_Wday;
512   RTC_DOY = Time.RTC_Yday;
513   RTC_MONTH = Time.RTC_Mon;
514   RTC_YEAR = Time.RTC_Year;
515   return;
516 }
517
518 /*********************************************************
519 Funkce: RTCGetTime
520 Zisk casu z RTC hodin
521 *********************************************************/
522 RTCTime RTCGetTime( void )
523 {
524   RTCTime LocalTime;
525
526   LocalTime.RTC_Sec = RTC_SEC;
527   LocalTime.RTC_Min = RTC_MIN;
528   LocalTime.RTC_Hour = RTC_HOUR;
529   LocalTime.RTC_Mday = RTC_DOM;
530   LocalTime.RTC_Wday = RTC_DOW;
531   LocalTime.RTC_Yday = RTC_DOY;
```

```
532   LocalTime.RTC_Mon = RTC_MONTH;
533   LocalTime.RTC_Year = RTC_YEAR;
534   return ( LocalTime );
535 }
536
537 /*****************************************************
538 Funkce: init_RTC
539 Inicializace RTC hodin
540 *****************************************************/
541
542 void init_RTC( void )
543 {
544         int temp;
545   RTC_AMR = 0;
546   RTC_CIIR = 0;
547   RTC_CCR = 0;
548   RTC_PREINT = PREINT_RTC;
549   RTC_PREFRAC = PREFRAC_RTC;
550
551         if (!NVOL_GetVariable(FLASH_IPC_ADR, (UNSIGNED8 *)&temp, sizeof(temp)))
552         //Zjisteni, zda se jedna o prvni start
553   {
554                 //V pripade, ze se jedna o prvni start, tak se nastavi datum 15.3.2013 a cas 12:00:00
555                 local_time.RTC_Sec = 0;
556                 local_time.RTC_Min = 0;
557                 local_time.RTC_Hour = 12;
558                 local_time.RTC_Mday = 15;
559                 local_time.RTC_Wday = 5;
560                 local_time.RTC_Yday = 15;
561                 local_time.RTC_Mon = 3;
562                 local_time.RTC_Year = 2013;
563                 RTCSetTime( local_time );
564         }
565         RTC_CCR |= CCR_CLKEN;
566   RTC_ILR = ILR_RTCCIF;
567   return;
568 }
569
570 /*****************************************************
571 Funkce: init_setup
572 Inicializace FLASH pameti
573 *****************************************************/
574
575 void init_setup(void)
576 {
577         NVOL_Init();
578 }
579
580
581
```

B.1.4 tone.c

```
1
2  /*********************************************
3  Tone.c - funkce pro vysilani z DAC
4
5  *********************************************/
6
7  #include "LPC23xx.h"            /* LPC23xx/24xx definice */
8  #include "type.h"               // Definice typu promennych
9  #include "irq.h"                // Definice preruseni
10 #include "var.h"                // Definice promenych
11 #include "tone.h"               // Definice vysialni z DAC
12 #include "DTMF.h"               // Definice prijmu DTMF
13
14 float t;
15 int pom;
16 float fs = 32200;               // Vzorkovaci frekvence v Hz
17 //Body sinusove krivky 1400Hz a 2300Hz pri 32200Hz vzorkovacim kmitoctu
18 float sin1400[23] = {0,2698,5196,7308,8879,9791,9977,9423,8170,6311,
```

```
3984,1362,-1362,-3984,-6311,-8170,-9433,-9977,-9791,-8879,-7308,-5196,-2698};
float sin2300[14]        = {0.3,4338,7819,9749,9749,7818,4339,
0,-4339,-7818,-9749,-9749,-7818,-4339};

static void tone_T_stop(void);


/*********************************************************
Funkce: DACsin
Na vystupu DAC tvori sinusovy ton
*********************************************************/
void DACsin (float f)
{
        DWORD i;
        float vyp;
        if (f == 0)
        //Vystupni kmitocet je 0Hz
        {
                vyp = 0;
        }
        else if (f == 1400)
        //Vystupni kmitocet je 1400Hz
        {
                vyp = 512 + TONE_level * (sin1400[pom]/10000);
                pom++;
                if (pom == 23) pom = 0;
        }
        else
        //Vystupni kmitocet je 2300Hz
        {
                vyp = 512 + TONE_level * (sin2300[pom]/10000);
                pom++;
                if (pom == 14) pom = 0;
        }
        //Predani hodnoty do D/A prevodniku
        i = vyp;
        DACR = (i << 6) | DAC_BIAS;
        t++;
}


/*********************************************************
Funkce: Timer2Handler
Preruseni po vyprseni casovace
*********************************************************/
void Timer2Handler (void) __irq
{
        float tmax;
  T2IR = 1;               /* zruseni preruseni */
  IENABLE;

        switch (TONE_phase)
        {
                case 1:
                //Handshake ton 1400Hz~100ms
                {
                        DACsin(1400);
                        tmax = fs / 10;
                        if (t == tmax)
                        {
                                DACsin(0);
                                t = 0;
                                TONE_phase = 2;
                        }
                        break;
                }

                case 2:
                //Handshake ton 0Hz~100ms
                {
                        DACsin(0);
                        tmax = fs / 10;
                        if (t == tmax)
```

```
92                          {
93                                  DACsin(0);
94                                  t = 0;
95                                  TONE_phase = 3;
96
97                          }
98                          break;
99                  }
100
101                 case 3:
102                 //Handshake ton 2300Hz~100ms
103                 {
104                         DACsin(2300);
105                         tmax = fs / 10;
106                         if (t == tmax)
107                         {
108                                 DACsin(0);
109                                 t = 0;
110                                 TONE_phase = 0;
111                                 tone_T_stop();
112                                 DTMF_phase = 4;
113                                 DTMF_T_start();
114                         }
115                         break;
116                 }
117
118                 case 4:
119                 //Kiss-off ton 1400Hz~1s
120                 {
121                         DACsin(1400);
122                         if (t == fs)
123                         {
124                                 DACsin(0);
125                                 t = 0;
126                                 TONE_phase = 0;
127                                 if (DTMF_quant == 3)
128                                 {
129                                         DTMF_phase = 0;
130                                         DTMF_number = 0;
131                                         IPC_phase = 6;
132                                 }
133                                 else
134                                 {
135                                 tone_T_stop();
136                                 DTMF_phase = 8;
137                                 DTMF_T_start();
138                                 }
139                         }
140                         break;
141                 }
142
143         }
144
145   IDISABLE;
146   VICVectAddr = 0;        /* potvrzeni preruseni */
147 }
148
149 /*********************************************************
150 Funkce: tone_T_stop
151 Stop casovace pro DAC
152 *********************************************************/
153
154 void tone_T_stop (void)
155 {
156         T2TCR = 0;
157 }
158
159 /*********************************************************
160 Funkce: tone_T_start
161 Casovac pro DAC
162 *********************************************************/
163
164 void tone_T_start (void)
```

```
{
        int Ts;
        pom = 0;
        T2TCR = 0x02;               /* vynulovani casovace */
        T2PR  = 0;                  /* casovac bez deleni */
        Ts = (12000000/fs)-1;
        T2MR0 = Ts;
        T2IR  = 0xff;               /* vynulovani preruseni */
        T2MCR = 0x03;               /* po preruseni resetovat */
        T2TCR = 0x01;
        install_irq( TIMER2_INT, (void *)Timer2Handler, HIGHEST_PRIORITY );
        //Nastaveni preruseni pri TIMER2 - spusteni funkce Timer3Handler
        return;
}


/*********************************************************
Funkce: tone_hand
Aktivace Handshake tonu
*********************************************************/

void tone_hand(void)
{
        tone_init();
        TONE_phase = 1;
        tone_T_start();
}

/*********************************************************
Funkce: tone_kiss
Aktivace Kiss off tonu
*********************************************************/

void tone_kiss(void)
{
        tone_init();
        TONE_phase = 4;
        tone_T_start();
}


/*********************************************************
Funkce: tone_init
Nastaveni vystupu procesoru
*********************************************************/

void tone_init(void)
{
        PCONP |= 0x400000;            // Povoleni TIMER2
        PINSEL1 |= 0x00200000;        // Nastaveni pinu P0.26 jako vystup D/A prevodniku
        t = 0;
        TONE_phase = 0;
}


```

B.1.5 telnet.c

```

/*********************************************
telnet.c - prijem nastaveni pres TELNET

*********************************************/

#include <Net_Config.h>
#include <string.h>
#include <stdio.h>
#include <LPC23xx.H>             // LPC23xx definice
#include "var.h"                 // Definice promennych
#include "set.h"                 // Definice nastaveni
#include "flash_nvol.h"          // Definice ukladani do Flash pameti
```

```
14
15 /* Net_Config.c */
16 extern struct tcp_cfg   tcp_config;
17 extern struct tnet_cfg  tnet_config;
18 #define tcp_NumSocks    tcp_config.NumSocks
19 #define tcp_socket      tcp_config.Scb
20 #define tnet_EnAuth     tnet_config.EnAuth
21 #define tnet_auth_passw tnet_config.Passw
22
23 /* ANSI znaky */
24 #define CLS     "\033[2J"
25 #define TBLUE   "\033[37;44m"
26 #define TNORM   "\033[0m"
27
28 int help_count = 0;
29 unsigned int ipa[4] = {0,0,0,0};
30 unsigned int sma[4] = {0,0,0,0};
31 unsigned int gwa[4] = {0,0,0,0};
32 unsigned int pdnsa[4] = {0,0,0,0};
33 unsigned int sdnsa[4] = {0,0,0,0};
34 unsigned int maca[6] = {0,0,0,0,0,0};
35 unsigned int dhcp;
36 char dev[33];
37 int ip = 0;
38 int sm = 0;
39 int gw = 0;
40 int pdns = 0;
41 int sdns = 0;
42 int mac = 0;
43 int DH = 0;
44 int de = 0;
45
46 extern void set_MAIL_setup (void);
47
48 typedef struct {
49   U8 id;
50   U8 nmax;
51   U8 idx;
52 } MY_BUF;
53 #define MYBUF(p)        ((MY_BUF *)p)
54
55 static U8 const tnet_header[] = {
56   CLS "\r\n"
57
58   TBLUE "*============================================================*\r\n" TNORM
59   TBLUE "*            Nastaveni prevodniku CID->TCP                   *\r\n" TNORM
60   TBLUE "*============================================================*\r\n" TNORM
61   };
62 static U8 const tnet_help1[] = {
63   "\r\n\r\n"
64   "    Dostupne prikazy:\r\n"
65   "    ----------------------------\r\n"
66   "    IP x.x.x.x       - zmena IP adresy prevodniku\r\n"
67   "    SM x.x.x.x       - zmena masky site\r\n"
68   "    GW x.x.x.x       - zmema vychozi brany\r\n"
69   "    DNS1 x.x.x.x     - zmena DNS serveru 1\r\n"
70         "    DNS2 x.x.x.x     - zmena DNS serveru 2\r\n"
71   "    MAC x-x-x-x-x-x  - zmena MAC adresy prevodniku\r\n"
72   "    DHCP x           - Povoleni DHCP 0-zakazo 1-povoleno\r\n"
73   "    DEV x            - Zmena nazvu objektu a zarizeni\r\n"
74         "    SAVE_IP          - Ulozi nastaveni IP a restartuje prevodnik\r\n"
75         "    ----------------------------\r\n"};
76 static U8 const tnet_help2[] = {
77         "    EZS_ME x         - Max. interval mezi DTMF u volby z EZS (ms)\r\n"
78         "    EZS_MM x         - Max. interval mezi DTMF u man. volby (ms)\r\n"
79         "    EZS_BH x         - Interval pred Handshake tonem (ms)\r\n"
80         "    EZS_BM x         - Max. interval pred prijetim zpravy po HT (ms)\r\n"
81         "    EZS_BK x         - Interval pred Kiss-off tonem (ms)\r\n"
82         "    EZS_TN x         - Telefonni cislo prevodniku\r\n"
83         "    ----------------------------\r\n"};
84 static U8 const tnet_help3[] = {
85         "    PCO_IP x.x.x.x   - IP adresa PCO\r\n"
86         "    PCO_PORT x       - Port PCO\r\n"
```

```
87          "    PCO_EN x         - Nastaveni siforvani 0-bez sifr. 1-sifr.\r\n"
88          "    PCO_RCVR x       - Cislo prijimace PCO 0-zadne cislo\r\n"
89          "    PCO_PREF x       - Prefix cisla objektu pro PCO\r\n"
90          "    PCO_ACCT x       - Cislo objektu\r\n"
91          "    PCO_ANT x        - Preklad cisel objektu 0-bez prekl. 1-prekl.\r\n"
92          "    PCO_TS x         - Odesilat u nesif. zprav datum a cas 0-ne 1-ano\r\n\r\n"
93          "    PCO_IN x         - Interval kontrolnich zprav 10-3600s\r\n"
94          "    PCO_KEY x        - Sifrovaci klic delka 16,24 nebo 32 zn.\r\n"
95          "    PCO_IV x         - Inicializacni vektor delka 16 znaku\r\n"
96          "    ------------------------------\r\n"};
97  static U8 const tnet_help4[] = {
98          "    MAIL_ME x        - Max. pocet IP chyb pred odesl. mailu\r\n"
99          "    MAIL_FROM x      - Odesilatel mailu\r\n"
100         "    MAIL_TO x        - Prijemce mailu\r\n"
101         "    MAIL_AUTH x      - SMTP server pozaduje auth. 0-ne 1-ano\r\n"
102         "    MAIL_USER x      - Uz.jmeno SMTP\r\n"
103         "    MAIL_PASS x      - Heslo SMTP\r\n"
104         "    MAIL_IP x.x.x.x  - IP adresa SMTP\r\n"
105         "    MAIL_PORT x      - Port SMTP\r\n"
106         "    MAIL_EN_DP x     - Mail pri chybe prijmu z EZS 1-ano 0-ne\r\n"
107         "    MAIL_EN_IPC x    - Mail pri chybe odesl. na PCO 1-ano 0-ne\r\n"
108         "    MAIL_EN_10 x     - Mail pri lek. poplachu 1-ano 0-ne\r\n"
109         "    MAIL_EN_11 x     - Mail pri pozarnim poplachu 1-ano 0-ne\r\n"
110         "    MAIL_EN_12 x     - Mail pri tisnovem poplachu 1-ano 0-ne\r\n"
111         "    MAIL_EN_13 x     - Mail pri naruseni objektu 1-ano 0-ne\r\n"
112         "    MAIL_EN_14 x     - Mail pri poplachu/poruse 1-ano 0-ne\r\n"
113         "    MAIL_EN_15 x     - Mail pri poplachu 24hod smycky 1-ano 0-ne\r\n"
114         "    MAIL_EN_20 x     - Mail pri dohledovem poplachu 1-ano 0-ne\r\n"
115         "    MAIL_EN_3 x      - Mail pri technickem poplachu 1-ano 0-ne\r\n"
116         "    MAIL_EN_40 x     - Mail pri prichodu/odchodu 1-ano 0-ne\r\n"
117         "    ------------------------------\r\n"};
118 static U8 const tnet_help5[] = {
119         "    TONE_LEVEL x     - Hlasitost zvuku z prevodniku 0-512\r\n"
120         "    ------------------------------\r\n"};
121 static U8 const tnet_help6[] = {
122         "    CH_PSWD x        - Zmena prihlasovaciho hesla\r\n"
123         "    HELP             - Zobrazi tuto napovedu\r\n"
124         "    QUIT             - Ukonci pripojeni\r\n"};
125
126
127
128 /*********************************************************
129 Funkce: tnet_cbfunc
130 Funkce vyvolana TELNETem
131 *********************************************************/
132
133 U16 tnet_cbfunc (U8 code, U8 *buf, U16 buflen) {
134
135   U16 len = 0;
136         U8 pom[33];
137   buflen = buflen;
138
139
140   switch (code) {
141     case 0:                         //Odeslani hlavicky
142       len = str_copy (buf, (U8 *)&tnet_header);
143       break;
144
145     case 1:                         //Zobrazeni jmena zarizeni
146                         sprintf(pom,"\r\n%s> ",IPC_name);
147       len = str_copy (buf, pom);
148       break;
149
150     case 2:                         //Zobrazeni textu pri prihlaseni
151       len = str_copy (buf, CLS "\r\nNastaveni prevodniku CID->TCP,"
152                                " prosim prihlaste se...\r\n");
153       break;
154
155     case 3:                         //Zobrazeni textu pri pozadavku uz.jmena
156       len = str_copy (buf, "\r\nUz.jmeno: ");
157       break;
158
159     case 4:                         //Zobrazeni textu pri pozadavku hesla
```

```
160       len = str_copy (buf, "\r\nHeslo: ");
161       break;
162
163     case 5:                         //Zobrazeni textu pri selhani prihlaseni
164       len = str_copy (buf, "\r\nPrihlasovani selhalo");
165       break;
166
167     case 6:                         //Zobrazeni textu pri dlouhe necinnosti
168       len = str_copy (buf, "\r\nVyprsel cas prihlaseni.\r\n");
169       break;
170   }
171   return (len);
172 }
173
174
175 /*************************************************************
176 Funkce: tnet_process_cmd
177 Funkce vyvolana pri zadani prikazu do TELNETu
178 *************************************************************/
179
180 U16 tnet_process_cmd (U8 *cmd, U8 *buf, U16 buflen, U32 *pvar) {
181
182   REMOTEM rm;
183   U16 len = 0;
184         int i;
185         int pom;
186         char tmp[33];
187         char pom_adr[18];
188
189         //Zobrazeni napovedy
190         if (help_count > 0)
191                 {
192                         switch (help_count)
193                         {
194                                 case 1:
195                                         len = str_copy (buf,(U8 *)tnet_help2);
196                                         help_count++;
197                                         return (len | 0x4000);
198                                 case 2:
199                                         len = str_copy (buf,(U8 *)tnet_help3);
200                                         help_count++;
201                                         return (len | 0x4000);
202                                 case 3:
203                                         len = str_copy (buf,(U8 *)tnet_help4);
204                                         help_count++;
205                                         return (len | 0x4000);
206                                 case 4:
207                                         len = str_copy (buf,(U8 *)tnet_help5);
208                                         help_count++;
209                                         return (len | 0x4000);
210                                 case 5:
211                                         len = str_copy (buf,(U8 *)tnet_help6);
212                                         help_count = 0;
213                                         return (len);
214                         }
215                 }
216
217
218         len = strlen ((const char *)cmd);
219         //Prikaz IP
220   if (tnet_ccmp (cmd, "IP") == __TRUE) {
221     if (len > 4) {
222       sscanf ((const char *)(cmd+3),"%d.%d.%d.%d",&ipa[0],&ipa[1],&ipa[2],&ipa[3]);
223                         for (i=0;i<4;i++)
224                         {
225                                 if (ipa[i] > 255)
226                                 {
227                                         len = sprintf ((char *)buf,"\r\n IP => Neplatna adresa");
228                                         return (len);
229                                 }
230                         }
231                         ip = 1;
232                         len = sprintf ((char *)buf,"\r\n IP => Nova adresa bude %d.%d.%d.%d
```

```
233                         Hodnotu ulozite prikazem SAVE_IP",ipa[0],ipa[1],ipa[2],ipa[3]);
234                         return (len);
235                         }
236     }
237
238                 //Prikaz SM
239                 if (tnet_ccmp (cmd, "SM") == __TRUE) {
240     if (len > 4) {
241       sscanf ((const char *)(cmd+3),"%d.%d.%d.%d",&sma[0],&sma[1],&sma[2],&sma[3]);
242
243                         for (i=0;i<4;i++)
244                         {
245                                 if (sma[i] > 255)
246                                 {
247                                         len = sprintf ((char *)buf,"\r\n SM => Neplatna adresa");
248                                         return (len);
249                                 }
250                         }
251
252                         sm = 1;
253                         len = sprintf ((char *)buf,"\r\n SM => Nova maska site bude %d.%d.%d.%d
254                         Hodnotu ulozite prikazem SAVE_IP",sma[0],sma[1],sma[2],sma[3]);
255                         return (len);
256                         }
257     }
258
259                 //Prikaz GW
260                 if (tnet_ccmp (cmd, "GW") == __TRUE) {
261     if (len > 4) {
262       sscanf ((const char *)(cmd+3),"%d.%d.%d.%d",&gwa[0],&gwa[1],&gwa[2],&gwa[3]);
263                         for (i=0;i<4;i++)
264                         {
265                                 if (gwa[i] > 255)
266                                 {
267                                         len = sprintf ((char *)buf,"\r\n GW => Neplatna adresa");
268                                         return (len);
269                                 }
270                         }
271
272                         gw = 1;
273                         len = sprintf ((char *)buf,"\r\n GW => Nova vychozi brana bude %d.%d.%d.%d
274                         Hodnotu ulozite prikazem SAVE_IP",gwa[0],gwa[1],gwa[2],gwa[3]);
275                         return (len);
276                         }
277     }
278
279                 //Prikaz DNS1
280                 if (tnet_ccmp (cmd, "DNS1") == __TRUE) {
281     if (len > 6) {
282       sscanf ((const char *)(cmd+5),"%d.%d.%d.%d",&pdnsa[0],&pdnsa[1],&pdnsa[2],&pdnsa[3]);
283
284                         for (i=0;i<4;i++)
285                         {
286                                 if (pdnsa[i] > 255)
287                                 {
288                                         len = sprintf ((char *)buf,"\r\n DNS1 => Neplatna adresa");
289                                         return (len);
290                                 }
291                         }
292
293                         pdns = 1;
294                         len = sprintf ((char *)buf,"\r\n DNS1 => Nova adresa DNS1
295                         bude %d.%d.%d.%d Hodnotu ulozite prikazem SAVE_IP",pdnsa[0],pdnsa[1],
296                         pdnsa[2],pdnsa[3]);
297                         return (len);
298                         }
299     }
300
301                 //Prikaz DNS2
302                 if (tnet_ccmp (cmd, "DNS2") == __TRUE) {
303     if (len > 6) {
304       sscanf ((const char *)(cmd+5),"%d.%d.%d.%d",&sdnsa[0],&sdnsa[1],&sdnsa[2],&sdnsa[3]);
305
```

```
306                         for (i=0;i<4;i++)
307                         {
308                                 if (sdnsa[i] > 255)
309                                 {
310                                         len = sprintf ((char *)buf,"\r\n DNS2 => Neplatna adresa");
311                                         return (len);
312                                 }
313                         }
314
315                         sdns = 1;
316                         len = sprintf ((char *)buf,"\r\n DNS2 => Nova adresa DNS2 bude %d.%d.%d.%d
317                         Hodnotu ulozite prikazem SAVE_IP",sdnsa[0],sdnsa[1],sdnsa[2],sdnsa[3]);
318                         return (len);
319                         }
320     }
321
322                 //Prikaz MAC
323                 if (tnet_ccmp (cmd, "MAC") == __TRUE) {
324     if (len >= 5) {
325       sscanf ((const char *)(cmd+4),"%x-%x-%x-%x-%x-%x",&maca[0],
326       &maca[1],&maca[2],&maca[3],&maca[4],&maca[5]);
327
328                         for (i=0;i<6;i++)
329                         {
330                                 if (maca[i] > 255)
331                                 {
332                                         len = sprintf ((char *)buf,"\r\n MAC => Neplatna adresa");
333                                         return (len);
334                                 }
335                         }
336
337                         mac = 1;
338                         len = sprintf ((char *)buf,"\r\n MAC => Nova MAC adresa bude
339                         %x-%x-%x-%x-%x-%x Hodnotu ulozite prikazem
340                          SAVE_IP",maca[0],maca[1],maca[2],maca[3],maca[4],maca[5]);
341                         return (len);
342                         }
343     }
344
345                 //Prikaz DHCP
346                 if (tnet_ccmp (cmd, "DHCP") == __TRUE) {
347     if (len == 6) {
348       sscanf((const char *)(cmd+5),"%d",&dhcp);
349                         if (dhcp == 1)
350                         {
351                                 len = sprintf ((char *)buf,"\r\n DHCP =>
352                                 DHCP bude nove povoleno. Hodnotu ulozite prikazem SAVE_IP");
353                         }
354                         else
355                         {
356                                 dhcp =0;
357                                 len = sprintf ((char *)buf,"\r\n DHCP =>
358                                 DHCP bude nove zakazano. Hodnotu ulozite prikazem SAVE_IP");
359                         }
360                         DH =1;
361                         return (len);
362                         }
363                         else
364                         {
365                                 len = sprintf ((char *)buf,"\r\n DHCP => Neplatny parametr");
366                                 return (len);
367                         }
368     }
369
370                 //Prikaz DEV
371                 if (tnet_ccmp (cmd, "DEV") == __TRUE) {
372     if ((len > 5) && (len < 19)) {
373       mem_copy (dev, &cmd[4], 32);
374                         de = 1;
375                         len = sprintf ((char *)buf,"\r\n DEV =>
376                         Nove jmeno objektu a zarizeni bude: %s Hodnotu ulozite prikazem SAVE_IP",dev);
377                         return (len);
378                         }
```

```
379                         else
380                         {
381                                 len = sprintf ((char *)buf,"\r\n DEV => Neplatna delka nazvu -
382                                 nazev musi byt delsi nez 1 znak a kratsi nez 16");
383                                 return (len);
384                         }
385     }
386
387                 //Prikaz SAVE_IP - ulozeni promennych pri IP nastaveni
388                 if (tnet_ccmp (cmd, "SAVE_IP") == __TRUE) {
389
390     if (ip == 1)
391                 {
392                         sprintf(pom_adr,"%03d %03d %03d %03d",ipa[0],ipa[1],ipa[2],ipa[3]);
393                         NVOL_SetVariable(FLASH_IP_ADR, (UNSIGNED8 *)&pom_adr, sizeof(pom_adr));
394                         if(!NVOL_GetVariable(FLASH_IP_ADR, (UNSIGNED8 *)&tmp, sizeof(tmp)))
395                          NVOL_SetVariable(FLASH_IP_ADR, (UNSIGNED8 *)&pom_adr, sizeof(pom_adr));
396                 }
397
398                 if (sm == 1)
399                 {
400                         sprintf(pom_adr,"%03d %03d %03d %03d",sma[0],sma[1],sma[2],sma[3]);
401                         NVOL_SetVariable(FLASH_IP_SM, (UNSIGNED8 *)&pom_adr, sizeof(pom_adr));
402                         if(!NVOL_GetVariable(FLASH_IP_SM, (UNSIGNED8 *)&tmp, sizeof(tmp)))
403                          NVOL_SetVariable(FLASH_IP_SM, (UNSIGNED8 *)&pom_adr, sizeof(pom_adr));
404                 }
405
406                 if (gw == 1)
407                 {
408                         sprintf(pom_adr,"%03d %03d %03d %03d",gwa[0],gwa[1],gwa[2],gwa[3]);
409                         NVOL_SetVariable(FLASH_IP_GW, (UNSIGNED8 *)&pom_adr, sizeof(pom_adr));
410                         if(!NVOL_GetVariable(FLASH_IP_GW, (UNSIGNED8 *)&tmp, sizeof(tmp)))
411                          NVOL_SetVariable(FLASH_IP_GW, (UNSIGNED8 *)&pom_adr, sizeof(pom_adr));
412                 }
413
414                 if (pdns == 1)
415                 {
416                         sprintf(pom_adr,"%03d %03d %03d %03d",pdnsa[0],pdnsa[1],pdnsa[2],pdnsa[3]);
417                         NVOL_SetVariable(FLASH_IP_DNS1, (UNSIGNED8 *)&pom_adr, sizeof(pom_adr));
418                         if(!NVOL_GetVariable(FLASH_IP_DNS1, (UNSIGNED8 *)&tmp, sizeof(tmp)))
419                          NVOL_SetVariable(FLASH_IP_DNS1, (UNSIGNED8 *)&pom_adr, sizeof(pom_adr));
420                 }
421
422                 if (sdns == 1)
423                 {
424                         sprintf(pom_adr,"%03d %03d %03d %03d",sdnsa[0],sdnsa[1],sdnsa[2],sdnsa[3]);
425                         NVOL_SetVariable(FLASH_IP_DNS2, (UNSIGNED8 *)&pom_adr, sizeof(pom_adr));
426                         if(!NVOL_GetVariable(FLASH_IP_DNS1, (UNSIGNED8 *)&tmp, sizeof(tmp)))
427                          NVOL_SetVariable(FLASH_IP_DNS1, (UNSIGNED8 *)&pom_adr, sizeof(pom_adr));
428                 }
429                 if (mac == 1)
430                 {
431                         sprintf(pom_adr,"%02x %02x %02x %02x %02x
432                          %02x",maca[0],maca[1],maca[2],maca[3],maca[4],maca[5]);
433                         NVOL_SetVariable(FLASH_IP_MAC, (UNSIGNED8 *)&pom_adr, sizeof(pom_adr));
434                         if(!NVOL_GetVariable(FLASH_IP_DNS1, (UNSIGNED8 *)&tmp, sizeof(tmp)))
435                          NVOL_SetVariable(FLASH_IP_DNS1, (UNSIGNED8 *)&pom_adr, sizeof(pom_adr));
436                 }
437
438                 if (DH == 1)
439                 {
440                         NVOL_SetVariable(FLASH_IP_DHCP, (UNSIGNED8 *)&dhcp, sizeof(dhcp));
441                         if(!NVOL_GetVariable(FLASH_IP_DHCP, (UNSIGNED8 *)&tmp, sizeof(tmp)))
442                          NVOL_SetVariable(FLASH_IP_DHCP, (UNSIGNED8 *)&dhcp, sizeof(dhcp));
443                 }
444
445                 if (de == 1)
446                 {
447                         NVOL_SetVariable(FLASH_IP_DEV, (UNSIGNED8 *)&dev, sizeof(dev));
448                         if(!NVOL_GetVariable(FLASH_IP_DEV, (UNSIGNED8 *)&tmp, sizeof(tmp)))
449                          NVOL_SetVariable(FLASH_IP_DEV, (UNSIGNED8 *)&dev, sizeof(dev));
450                 }
451
```

```
452                 //Restart zarizeni
453     WDTC = 0x00000FFF;
454                 WDMOD = 0x03;
455                 WDFEED = 0xAA;
456                 WDFEED = 0x55;
457
458     return (len | 0x8000);
459   }
460
461         //Prikaz EZS_ME
462         if (tnet_ccmp (cmd, "EZS_ME") == __TRUE) {
463     if (len > 7) {
464       sscanf ((const char *)(cmd+7),"%d",&pom);
465
466                                 if ((pom < 2501) && (pom > 99))
467                                 {
468                                         DTMF_maxEZS = pom;
469                                         NVOL_SetVariable(FLASH_DTMF_maxEZS, (UNSIGNED8 *)&DTMF_maxEZS,
470                                         sizeof(DTMF_maxEZS));
471                                         if (!NVOL_GetVariable(FLASH_DTMF_maxEZS, (UNSIGNED8 *)&tmp, sizeof(tmp)))
472                                          NVOL_SetVariable(FLASH_DTMF_maxMan, (UNSIGNED8 *)&DTMF_maxEZS,
473                                          sizeof(DTMF_maxEZS));
474                                         len = sprintf ((char *)buf,"\r\n EZS_ME => Nova hodnota je %d ms",
475                                         DTMF_maxEZS);
476                                         return (len);
477                                 }
478                                 else
479                                 {
480                                         len = sprintf ((char *)buf,"\r\n EZS_ME =>
481                                         Neplatny interval (Interval musi byt v rozsahu 100 - 2500ms");
482                                         return (len);
483                                 }
484                         }
485     }
486
487                 //Prikaz EZS_MM
488                 if (tnet_ccmp (cmd, "EZS_MM") == __TRUE) {
489     if (len > 7) {
490       sscanf ((const char *)(cmd+7),"%d",&pom);
491
492                                 if ((pom < 2501) && (pom > 99))
493                                 {
494                                         DTMF_maxMan = pom;
495                                         NVOL_SetVariable(FLASH_DTMF_maxMan, (UNSIGNED8 *)&DTMF_maxMan,
496                                         sizeof(DTMF_maxMan));
497                                         if (!NVOL_GetVariable(FLASH_DTMF_maxMan, (UNSIGNED8 *)&tmp, sizeof(tmp)))
498                                          NVOL_SetVariable(FLASH_DTMF_maxMan, (UNSIGNED8 *)&DTMF_maxMan,
499                                          sizeof(DTMF_maxMan));
500                                         len = sprintf ((char *)buf,"\r\n EZS_MM => Nova hodnota je %d ms",
501                                         DTMF_maxMan);
502                                         return (len);
503                                 }
504                                 else
505                                 {
506                                         len = sprintf ((char *)buf,"\r\n EZS_MM =>
507                                         Neplatny interval (Interval musi byt v rozsahu 100 - 2500ms");
508                                         return (len);
509                                 }
510                         }
511     }
512
513                 //Prikaz EZS_BH
514                 if (tnet_ccmp (cmd, "EZS_BH") == __TRUE) {
515     if (len > 7) {
516       sscanf ((const char *)(cmd+7),"%d",&pom);
517
518                                 if ((pom < 2001) && (pom > 499))
519                                 {
520                                         DTMF_befHand = pom;
521                                         NVOL_SetVariable(FLASH_DTMF_befHand, (UNSIGNED8 *)&DTMF_befHand,
522                                          sizeof(DTMF_befHand));
523                                         if (!NVOL_GetVariable(FLASH_DTMF_befHand, (UNSIGNED8 *)&tmp, sizeof(tmp)))
524                                          NVOL_SetVariable(FLASH_DTMF_befHand, (UNSIGNED8 *)&DTMF_befHand,
```

```
525                                       sizeof(DTMF_befHand));
526                                     len = sprintf ((char *)buf,"\r\n EZS_BH => Nova hodnota je %d ms",
527                                     DTMF_befHand);
528                                     return (len);
529                             }
530                             else
531                             {
532                                     len = sprintf ((char *)buf,"\r\n EZS_BH =>
533                                     Neplatny interval (Interval musi byt v rozsahu 500 - 2000ms");
534                                     return (len);
535                             }
536                     }
537     }
538
539                 //Prikaz EZS_BM
540                 if (tnet_ccmp (cmd, "EZS_BM") == __TRUE) {
541
542     if (len > 7) {
543       sscanf ((const char *)(cmd+7),"%d",&pom);
544
545                             if ((pom < 4001) && (pom > 499))
546                             {
547                                     DTMF_befMess = pom;
548                                     NVOL_SetVariable(FLASH_DTMF_befMess, (UNSIGNED8 *)&DTMF_befMess,
549                                      sizeof(DTMF_befMess));
550                                     if (!NVOL_GetVariable(FLASH_DTMF_befMess, (UNSIGNED8 *)&tmp, sizeof(tmp)))
551                                      NVOL_SetVariable(FLASH_DTMF_befMess, (UNSIGNED8 *)&DTMF_befMess,
552                                       sizeof(DTMF_befMess));
553                                     len = sprintf ((char *)buf,"\r\n EZS_BM => Nova hodnota je %d ms",
554                                     DTMF_befMess);
555                                     return (len);
556                             }
557                             else
558                             {
559                                     len = sprintf ((char *)buf,"\r\n EZS_BM =>
560                                     Neplatny interval (Interval musi byt v rozsahu 500 - 4000ms");
561                                     return (len);
562                             }
563                     }
564     }
565
566                 //Prikaz EZS_BK
567                 if (tnet_ccmp (cmd, "EZS_BK") == __TRUE) {
568     if (len > 7) {
569       sscanf ((const char *)(cmd+7),"%d",&pom);
570
571                             if ((pom < 1501) && (pom > 999))
572                             {
573                                     DTMF_befKiss = pom;
574                                     NVOL_SetVariable(FLASH_DTMF_befKiss, (UNSIGNED8 *)&DTMF_befKiss
575                                     , sizeof(DTMF_befKiss));
576                                     if (!NVOL_GetVariable(FLASH_DTMF_befKiss, (UNSIGNED8 *)&tmp, sizeof(tmp)))
577                                      NVOL_SetVariable(FLASH_DTMF_befKiss, (UNSIGNED8 *)&DTMF_befKiss,
578                                       sizeof(DTMF_befKiss));
579                                     len = sprintf ((char *)buf,"\r\n EZS_BK => Nova hodnota je %d ms",
580                                     DTMF_befKiss);
581                                     return (len);
582                             }
583                             else
584                             {
585                                     len = sprintf ((char *)buf,"\r\n EZS_BK =>
586                                     Neplatny interval (Interval musi byt v rozsahu 1000 - 1500ms");
587                                     return (len);
588                             }
589                     }
590     }
591
592                 //Prikaz EZS_TN
593                 if (tnet_ccmp (cmd, "EZS_TN") == __TRUE) {
594     if (len > 7) {
595       sscanf ((const char *)(cmd+7),"%d",&pom);
596
597                             if ((pom < 1000000000) && (pom > 99))
```

```
598                                 {
599                                         TLF_number = pom;
600                                         NVOL_SetVariable(FLASH_TLF_number, (UNSIGNED8 *)&TLF_number,
601                                         sizeof(TLF_number));
602                                         if (!NVOL_GetVariable(FLASH_TLF_number, (UNSIGNED8 *)&tmp, sizeof(tmp)))
603                                          NVOL_SetVariable(FLASH_TLF_number, (UNSIGNED8 *)&TLF_number,
604                                          sizeof(TLF_number));
605                                         len = sprintf ((char *)buf,"\r\n EZS_TN => Nove telefonni
606                                         cislo je %d",TLF_number);
607                                         return (len);
608                                 }
609                                 else
610                                 {
611                                         len = sprintf ((char *)buf,"\r\n EZS_TN => Zvolte
612                                         prosim telefonni cislo o delce 3 az 9 cislic");
613                                         return (len);
614                                 }
615                         }
616     }
617
618                 //Prikaz PCO_IP
619                 if (tnet_ccmp (cmd, "PCO_IP") == __TRUE) {
620     if (len > 4) {
621       sscanf ((const char *)(cmd+7),"%d.%d.%d.%d",&ipa[0],&ipa[1],&ipa[2],&ipa[3]);
622
623                         for (i=0;i<4;i++)
624                         {
625                                 if (ipa[i] > 255)
626                                 {
627                                         len = sprintf ((char *)buf,"\r\n PCO_IP => Neplatna adresa");
628                                         return (len);
629                                 }
630                         }
631
632                         sprintf(pom_adr,"%03d %03d %03d %03d",ipa[0],ipa[1],ipa[2],ipa[3]);
633                         IPC_ADR[0] = ipa[0];
634                         IPC_ADR[1] = ipa[1];
635                         IPC_ADR[2] = ipa[2];
636                         IPC_ADR[3] = ipa[3];
637                         NVOL_SetVariable(FLASH_IPC_ADR, (UNSIGNED8 *)&pom_adr, sizeof(pom_adr));
638                         if(!NVOL_GetVariable(FLASH_IPC_ADR, (UNSIGNED8 *)&tmp, sizeof(tmp)))
639                          NVOL_SetVariable(FLASH_IPC_ADR, (UNSIGNED8 *)&pom_adr, sizeof(pom_adr));
640
641                         len = sprintf ((char *)buf,"\r\n PCO_IP => Nova adresa PCO
642                         je %d.%d.%d.%d",IPC_ADR[0],IPC_ADR[1],IPC_ADR[2],IPC_ADR[3]);
643                         return (len);
644                         }
645     }
646
647                 //Prikaz PCO_PORT
648                 if (tnet_ccmp (cmd, "PCO_PORT") == __TRUE) {
649     if (len > 9) {
650       sscanf ((const char *)(cmd+9),"%d",&pom);
651
652                                 if ((pom < 65536) && (pom > 0))
653                                 {
654                                         IPC_PORT = pom;
655                                         NVOL_SetVariable(FLASH_IPC_PORT, (UNSIGNED8 *)&IPC_PORT, sizeof(IPC_PORT));
656                                         if (!NVOL_GetVariable(FLASH_IPC_PORT, (UNSIGNED8 *)&tmp, sizeof(tmp)))
657                                          NVOL_SetVariable(FLASH_IPC_PORT, (UNSIGNED8 *)&IPC_PORT, sizeof(IPC_PORT));
658                                         len = sprintf ((char *)buf,"\r\n PCO_PORT => Novy vzdaleny port
659                                         je %d",IPC_PORT);
660                                         return (len);
661                                 }
662                                 else
663                                 {
664                                         len = sprintf ((char *)buf,"\r\n PCO_PORT =>
665                                         Neplatna hodnota (Hodnota musi byt v rozsahu 1 - 65535");
666                                         return (len);
667                                 }
668                         }
669     }
670
```

```
671                 //Prikaz PCO_EN
672                 if (tnet_ccmp (cmd, "PCO_EN") == __TRUE) {
673     if (len == 8) {
674       sscanf((const char *)(cmd+7),"%d",&pom);
675                         if (pom == 1)
676                         {
677                                 IPC_PROTOCOL = 1;
678                                 NVOL_SetVariable(FLASH_IPC_PROT, (UNSIGNED8 *)&IPC_PROTOCOL, sizeof(IPC_PROTOCOL));
679                                 if (!NVOL_GetVariable(FLASH_IPC_PROT, (UNSIGNED8 *)&tmp, sizeof(tmp)))
680                                  NVOL_SetVariable(FLASH_IPC_PROT, (UNSIGNED8 *)&IPC_PROTOCOL, sizeof(IPC_PROTOCOL));
681                                 len = sprintf ((char *)buf,"\r\n PCO_EN => Prenos bude sifrovan");
682                         }
683                         else
684                         {
685                                 IPC_PROTOCOL = 0;
686                                 NVOL_SetVariable(FLASH_IPC_PROT, (UNSIGNED8 *)&IPC_PROTOCOL, sizeof(IPC_PROTOCOL));
687                                 if (!NVOL_GetVariable(FLASH_IPC_PROT, (UNSIGNED8 *)&tmp, sizeof(tmp)))
688                                  NVOL_SetVariable(FLASH_IPC_PROT, (UNSIGNED8 *)&IPC_PROTOCOL, sizeof(IPC_PROTOCOL));
689                                 len = sprintf ((char *)buf,"\r\n PCO_EN => Prenos bude nesifrovan");
690                         }
691                         return (len);
692                         }
693                         else
694                         {
695                                 len = sprintf ((char *)buf,"\r\n PCO_EN => Neplatny parametr");
696                                 return (len);
697                         }
698     }
699
700                 //Prikaz PCO_RCVR
701                 if (tnet_ccmp (cmd, "PCO_RCVR") == __TRUE) {
702     if (len > 9) {
703       sscanf ((const char *)(cmd+9),"%x",&pom);
704
705                                 if ((pom <= 0xFFFFFF) && (pom >= 0))
706                                 {
707                                         IPC_rcvr = pom;
708                                         NVOL_SetVariable(FLASH_IPC_RCVR, (UNSIGNED8 *)&IPC_rcvr, sizeof(IPC_rcvr));
709                                         if (!NVOL_GetVariable(FLASH_IPC_RCVR, (UNSIGNED8 *)&tmp, sizeof(tmp))
710                                         ) NVOL_SetVariable(FLASH_IPC_RCVR, (UNSIGNED8 *)&IPC_rcvr, sizeof(IPC_rcvr));
711                                         len = sprintf ((char *)buf,"\r\n PCO_RCVR => Nova hodnota je %x",IPC_rcvr);
712                                         return (len);
713                                 }
714                                 else
715                                 {
716                                         len = sprintf ((char *)buf,"\r\n PCO_RCVR =>
717                                         Neplatny interval (Interval musi byt v rozsahu 0 - FFFFFF");
718                                         return (len);
719                                 }
720                         }
721     }
722
723                 //Prikaz PCO_PREF
724                 if (tnet_ccmp (cmd, "PCO_PREF") == __TRUE) {
725     if (len > 9) {
726       sscanf ((const char *)(cmd+9),"%x",&pom);
727
728                                 if ((pom <= 0xFFFFFF) && (pom >= 0))
729                                 {
730                                         IPC_pref = pom;
731                                         NVOL_SetVariable(FLASH_IPC_PREF, (UNSIGNED8 *)&IPC_pref, sizeof(IPC_pref));
732                                         if (!NVOL_GetVariable(FLASH_IPC_PREF, (UNSIGNED8 *)&tmp, sizeof(tmp)))
733                                          NVOL_SetVariable(FLASH_IPC_PREF, (UNSIGNED8 *)&IPC_pref, sizeof(IPC_pref));
734                                         len = sprintf ((char *)buf,"\r\n PCO_PREF => Nova hodnota je %x",IPC_pref);
735                                         return (len);
736                                 }
737                                 else
738                                 {
739                                         len = sprintf ((char *)buf,"\r\n PCO_PREF =>
740                                         Neplatny interval (Interval musi byt v rozsahu 0 - FFFFFF");
741                                         return (len);
742                                 }
743                         }
```

```
744     }
745
746                 //Prikaz PCO_ACCT
747                 if (tnet_ccmp (cmd, "PCO_ACCT") == __TRUE) {
748     if (len > 9) {
749       sscanf ((const char *)(cmd+9),"%x",&pom);
750
751                                 if ((pom <= 0xFFFF) && (pom >= 0))
752                                 {
753                                         IPC_acct = pom;
754                                         NVOL_SetVariable(FLASH_IPC_ACCT, (UNSIGNED8 *)&IPC_acct, sizeof(IPC_acct));
755                                         if (!NVOL_GetVariable(FLASH_IPC_ACCT, (UNSIGNED8 *)&tmp, sizeof(tmp)))
756                                          NVOL_SetVariable(FLASH_IPC_ACCT, (UNSIGNED8 *)&IPC_acct, sizeof(IPC_acct));
757                                         len = sprintf ((char *)buf,"\r\n PCO_ACCT =>
758                                         Nova hodnota je %x",IPC_acct);
759                                         return (len);
760                                 }
761                                 else
762                                 {
763                                         len = sprintf ((char *)buf,"\r\n PCO_ACCT =>
764                                         Neplatny interval (Interval musi byt v rozsahu 0 - FFFF");
765                                         return (len);
766                                 }
767                         }
768     }
769
770                 //Prikaz PCO_ANT
771                 if (tnet_ccmp (cmd, "PCO_ANT") == __TRUE) {
772     if (len == 9) {
773       sscanf((const char *)(cmd+8),"%d",&pom);
774                         if (pom == 1)
775                         {
776                                 IPC_ANT = 1;
777                                 NVOL_SetVariable(FLASH_IPC_ANT, (UNSIGNED8 *)&IPC_ANT, sizeof(IPC_ANT));
778                                 if (!NVOL_GetVariable(FLASH_IPC_ANT, (UNSIGNED8 *)&tmp, sizeof(tmp)))
779                                  NVOL_SetVariable(FLASH_IPC_ANT, (UNSIGNED8 *)&IPC_ANT, sizeof(IPC_ANT));
780                                 len = sprintf ((char *)buf,"\r\n PCO_ANT =>
781                                 Cislo objektu se bude prekladat");
782                         }
783                         else
784                         {
785                                 IPC_ANT = 0;
786                                 NVOL_SetVariable(FLASH_IPC_ANT, (UNSIGNED8 *)&IPC_ANT, sizeof(IPC_ANT));
787                                 if (!NVOL_GetVariable(FLASH_IPC_ANT, (UNSIGNED8 *)&tmp, sizeof(tmp)))
788                                  NVOL_SetVariable(FLASH_IPC_ANT, (UNSIGNED8 *)&IPC_ANT, sizeof(IPC_ANT));
789                                 len = sprintf ((char *)buf,"\r\n PCO_ANT =>
790                                 Cislo objektu se nebude prekladat");
791                         }
792                         return (len);
793                         }
794                         else
795                         {
796                                 len = sprintf ((char *)buf,"\r\n PCO_ANT =>
797                                 Neplatny parametr");
798                                 return (len);
799                         }
800     }
801
802                 //Prikaz PCO_TS
803                 if (tnet_ccmp (cmd, "PCO_TS") == __TRUE) {
804     if (len == 8) {
805       sscanf((const char *)(cmd+7),"%d",&pom);
806                         if (pom == 1)
807                         {
808                                 IPC_ts = 1;
809                                 NVOL_SetVariable(FLASH_IPC_TS, (UNSIGNED8 *)&IPC_ts, sizeof(IPC_ts));
810                                 if (!NVOL_GetVariable(FLASH_IPC_TS, (UNSIGNED8 *)&tmp, sizeof(tmp)))
811                                  NVOL_SetVariable(FLASH_IPC_TS, (UNSIGNED8 *)&IPC_ts, sizeof(IPC_ts));
812                                 len = sprintf ((char *)buf,"\r\n PCO_TS =>
813                                 Vzdy se bude prenaset datum a cas");
814                         }
815                         else
816                         {
```

```
817                                 IPC_ts = 0;
818                                 NVOL_SetVariable(FLASH_IPC_TS, (UNSIGNED8 *)&IPC_ts, sizeof(IPC_ts));
819                                 if (!NVOL_GetVariable(FLASH_IPC_TS, (UNSIGNED8 *)&tmp, sizeof(tmp)))
820                                  NVOL_SetVariable(FLASH_IPC_TS, (UNSIGNED8 *)&IPC_ts, sizeof(IPC_ts));
821                                 len = sprintf ((char *)buf,"\r\n PCO_TS =>
822                                 Datum a cas se bude prenaset je u sifr. zprav");
823                         }
824                         return (len);
825                         }
826                         else
827                         {
828                                 len = sprintf ((char *)buf,"\r\n PCO_TS =>
829                                 Neplatny parametr");
830                                 return (len);
831                         }
832     }
833
834                 //Prikaz PCO_IN
835                 if (tnet_ccmp (cmd, "PCO_IN") == __TRUE) {
836     if (len > 7) {
837       sscanf ((const char *)(cmd+7),"%d",&pom);
838
839                                 if ((pom < 3601) && (pom > 9))
840                                 {
841                                         IPC_interval = pom;
842                                         NVOL_SetVariable(FLASH_IPC_INT, (UNSIGNED8 *)&IPC_interval,
843                                         sizeof(IPC_interval));
844                                         if (!NVOL_GetVariable(FLASH_IPC_INT, (UNSIGNED8 *)&tmp, sizeof(tmp)))
845                                          NVOL_SetVariable(FLASH_IPC_INT, (UNSIGNED8 *)&IPC_interval,
846                                          sizeof(IPC_interval));
847                                         len = sprintf ((char *)buf,"\r\n PCO_IN =>
848                                         Nova hodnota intervalu je %d s",IPC_interval);
849                                         return (len);
850                                 }
851                                 else
852                                 {
853                                         len = sprintf ((char *)buf,"\r\n PCO_IN =>
854                                         Neplatny interval (Interval musi byt v rozsahu 10 - 3600 s");
855                                         return (len);
856                                 }
857                         }
858     }
859
860                 //Prikaz PCO_KEY
861                 if (tnet_ccmp (cmd, "PCO_KEY") == __TRUE) {
862     if ((len == 24) || (len == 32) || (len == 40)) {
863                                 mem_copy (IPC_key, &cmd[8], 32);
864                                 NVOL_SetVariable(FLASH_IPC_KEY, (UNSIGNED8 *)&cmd[8], 32);
865                                 if (!NVOL_GetVariable(FLASH_IPC_KEY, (UNSIGNED8 *)&tmp, sizeof(tmp)))
866                                  NVOL_SetVariable(FLASH_IPC_KEY, (UNSIGNED8 *)&cmd[8], 32);
867                                 len = sprintf ((char *)buf,"\r\n PCO_KEY =>
868                                 Novy sifrovaci klic je %s",IPC_key);
869                                 return (len);
870                         }
871                         else
872                         {
873                                 len = sprintf ((char *)buf,"\r\n PCO_KEY =>
874                                 Neplatna delka klice - klic musi byt dlouhy 16,24 nebo 32 znaku");
875                                 return (len);
876                         }
877     }
878
879                 //Prikaz PCO_IV
880                 if (tnet_ccmp (cmd, "PCO_IV") == __TRUE) {
881     if (len == 23) {
882                                 mem_copy (IPC_iv, &cmd[7], 17);
883                                 NVOL_SetVariable(FLASH_IPC_IV, (UNSIGNED8 *)&cmd[7], 17);
884                                 if (!NVOL_GetVariable(FLASH_IPC_IV, (UNSIGNED8 *)&tmp, sizeof(tmp)))
885                                  NVOL_SetVariable(FLASH_IPC_IV, (UNSIGNED8 *)&cmd[7], 17);
886                                 len = sprintf ((char *)buf,"\r\n PCO_IV =>
887                                 Novy inicializacni vektor je %s",IPC_iv);
888                                 return (len);
889                         }
```

```
890                         else
891                         {
892                                 len = sprintf ((char *)buf,"\r\n PCO_IV =>
893                                 Neplatna delka inicializacniho vektoru - vektor musi byt dlouhy 16");
894                                 return (len);
895                         }
896     }
897
898                 //Prikaz MAIL_ME
899                 if (tnet_ccmp (cmd, "MAIL_ME") == __TRUE) {
900     if (len > 8) {
901       sscanf ((const char *)(cmd+8),"%d",&pom);
902
903                                 if ((pom < 1001) && (pom > 9))
904                                 {
905                                         IPC_maxerr = pom;
906                                         set_MAIL_setup();
907                                         len = sprintf ((char *)buf,"\r\n MAIL_ME =>
908                                         Nova hodnota je %d",IPC_maxerr);
909                                         return (len);
910                                 }
911                                 else
912                                 {
913                                         len = sprintf ((char *)buf,"\r\n MAIL_ME =>
914                                         Neplatna hodnota (Hodnota musi byt v rozsahu 10 - 1000");
915                                         return (len);
916                                 }
917                         }
918     }
919
920                 //Prikaz MAIL_PORT
921                 if (tnet_ccmp (cmd, "MAIL_PORT") == __TRUE) {
922     if (len > 10) {
923       sscanf ((const char *)(cmd+10),"%d",&pom);
924
925                                 if ((pom < 65536) && (pom > 0))
926                                 {
927                                         MAIL_port = pom;
928                                         set_MAIL_setup();
929                                         len = sprintf ((char *)buf,"\r\n MAIL_PORT =>
930                                         Novy SMTP port je %d",MAIL_port);
931                                         return (len);
932                                 }
933                                 else
934                                 {
935                                         len = sprintf ((char *)buf,"\r\n MAIL_PORT =>
936                                         Neplatna hodnota (Hodnota musi byt v rozsahu 1 - 65535");
937                                         return (len);
938                                 }
939                         }
940     }
941
942                 //Prikaz MAIL_FROM
943                 if (tnet_ccmp (cmd, "MAIL_FROM") == __TRUE) {
944     if (len > 10) {
945                                 mem_copy (MAIL_from, &cmd[10], 32);
946                                 set_MAIL_setup();
947                                 len = sprintf ((char *)buf,"\r\n MAIL_FROM =>
948                                 Nova adresa je %s",MAIL_from);
949                                 return (len);
950                         }
951                         else
952                         {
953                                 len = sprintf ((char *)buf,"\r\n MAIL_FROM =>
954                                 Neplatny parametr");
955                                 return (len);
956                         }
957     }
958
959                 //Prikaz MAIL_TO
960                 if (tnet_ccmp (cmd, "MAIL_TO") == __TRUE) {
961     if (len > 8) {
962                                 mem_copy (MAIL_to, &cmd[8], 32);
```

```
963                                 set_MAIL_setup();
964                                 len = sprintf ((char *)buf,"\r\n MAIL_TO =>
965                                 Nova adresa je %s",MAIL_to);
966                                 return (len);
967                         }
968                         else
969                         {
970                                 len = sprintf ((char *)buf,"\r\n MAIL_TO =>
971                                 Neplatny parametr");
972                                 return (len);
973                         }
974     }
975
976                 //Prikaz MAIL_USER
977                 if (tnet_ccmp (cmd, "MAIL_USER") == __TRUE) {
978     if (len > 10) {
979                                 mem_copy (MAIL_user, &cmd[10], 32);
980                                 set_MAIL_setup();
981                                 len = sprintf ((char *)buf,"\r\n MAIL_USER =>
982                                 Nove uz.jmeno je %s",MAIL_user);
983                                 return (len);
984                         }
985                         else
986                         {
987                                 len = sprintf ((char *)buf,"\r\n MAIL_USER =>
988                                 Neplatny parametr");
989                                 return (len);
990                         }
991     }
992
993                 //Prikaz MAIL_PASS
994                 if (tnet_ccmp (cmd, "MAIL_PASS") == __TRUE) {
995     if (len > 10) {
996                                 mem_copy (MAIL_pass, &cmd[10], 32);
997                                 set_MAIL_setup();
998                                 len = sprintf ((char *)buf,"\r\n MAIL_PASS =>
999                                 Nove heslo je %s",MAIL_pass);
1000                                return (len);
1001                        }
1002                        else
1003                        {
1004                                len = sprintf ((char *)buf,"\r\n MAIL_PASS =>
1005                                Neplatny parametr");
1006                                return (len);
1007                        }
1008    }
1009
1010                //Prikaz MAIL_AUTH
1011                if (tnet_ccmp (cmd, "MAIL_AUTH") == __TRUE) {
1012    if (len == 11) {
1013      sscanf((const char *)(cmd+10),"%d",&pom);
1014                        if (pom == 1)
1015                        {
1016                                MAIL_auth = 1;
1017                                set_MAIL_setup();
1018                                len = sprintf ((char *)buf,"\r\n MAIL_AUTH =>
1019                                SMTP pozaduje authentifikaci");
1020                        }
1021                        else
1022                        {
1023                                MAIL_auth = 0;
1024                                set_MAIL_setup();
1025                                len = sprintf ((char *)buf,"\r\n MAIL_AUTH =>
1026                                SMTP nepozaduje authentifikaci");
1027                        }
1028                        return (len);
1029                        }
1030                        else
1031                        {
1032                                len = sprintf ((char *)buf,"\r\n MAIL_AUTH =>
1033                                Neplatny parametr");
1034                                return (len);
1035                        }
```

```
1036    }
1037
1038                //Prikaz MAIL_IP
1039                if (tnet_ccmp (cmd, "MAIL_IP") == __TRUE) {
1040    if (len > 9) {
1041      sscanf ((const char *)(cmd+8),"%d.%d.%d.%d",&ipa[0],&ipa[1],&ipa[2],&ipa[3]);
1042
1043                        for (i=0;i<4;i++)
1044                        {
1045                                if (ipa[i] > 255)
1046                                {
1047                                        len = sprintf ((char *)buf,"\r\n MAIL_IP =>
1048                                        Neplatna adresa");
1049                                        return (len);
1050                                }
1051                        }
1052
1053                        sprintf(pom_adr,"%03d %03d %03d %03d",ipa[0],ipa[1],ipa[2],ipa[3]);
1054                        MAIL_adr[0] = ipa[0];
1055                        MAIL_adr[1] = ipa[1];
1056                        MAIL_adr[2] = ipa[2];
1057                        MAIL_adr[3] = ipa[3];
1058                        set_MAIL_setup();
1059
1060                        len = sprintf ((char *)buf,"\r\n MAIL_IP =>
1061                        Nova adresa SMTP je %d.%d.%d.%d",MAIL_adr[0],MAIL_adr[1],MAIL_adr[2],MAIL_adr[3]);
1062                        return (len);
1063                        }
1064    }
1065
1066                //Prikaz MAIL_EN_DP
1067                if (tnet_ccmp (cmd, "MAIL_EN_DP") == __TRUE) {
1068    if (len == 12) {
1069      sscanf((const char *)(cmd+11),"%d",&pom);
1070                        if (pom == 1)
1071                        {
1072                                MAIL_en_dp = 1;
1073                                set_MAIL_setup();
1074                                len = sprintf ((char *)buf,"\r\n MAIL_EN_DP =>
1075                                Pri chybe prijmu z EZS se odesle mail");
1076                        }
1077                        else
1078                        {
1079                                MAIL_en_dp = 0;
1080                                set_MAIL_setup();
1081                                len = sprintf ((char *)buf,"\r\n MAIL_EN_DP =>
1082                                Pri chybe prijmu z EZS se neodesle mail");
1083                        }
1084                        return (len);
1085                        }
1086                        else
1087                        {
1088                                len = sprintf ((char *)buf,"\r\n MAIL_EN_DP =>
1089                                Neplatny parametr");
1090                                return (len);
1091                        }
1092    }
1093
1094                //Prikaz MAIL_EN_IPC
1095                if (tnet_ccmp (cmd, "MAIL_EN_IPC") == __TRUE) {
1096    if (len == 13) {
1097      sscanf((const char *)(cmd+12),"%d",&pom);
1098                        if (pom == 1)
1099                        {
1100                                MAIL_en_ipc = 1;
1101                                set_MAIL_setup();
1102                                len = sprintf ((char *)buf,"\r\n MAIL_EN_IPC =>
1103                                Pri chybe odeslani na PCO se odesle mail");
1104                        }
1105                        else
1106                        {
1107                                MAIL_en_ipc = 0;
1108                                set_MAIL_setup();
```

```
1109                                len = sprintf ((char *)buf,"\r\n MAIL_EN_IPC =>
1110                                Pri chybe odeslani na PCO se neodesle mail");
1111                        }
1112                        return (len);
1113                        }
1114                        else
1115                        {
1116                                len = sprintf ((char *)buf,"\r\n MAIL_EN_IPC =>
1117                                Neplatny parametr");
1118                                return (len);
1119                        }
1120      }
1121
1122                //Prikaz MAIL_EN_10
1123                if (tnet_ccmp (cmd, "MAIL_EN_10") == __TRUE) {
1124      if (len == 12) {
1125        sscanf((const char *)(cmd+11),"%d",&pom);
1126                        if (pom == 1)
1127                        {
1128                                MAIL_en_10 = 1;
1129                                set_MAIL_setup();
1130                                len = sprintf ((char *)buf,"\r\n MAIL_EN_10 =>
1131                                Pri lek. poplachu se odesle mail");
1132                        }
1133                        else
1134                        {
1135                                MAIL_en_10 = 0;
1136                                set_MAIL_setup();
1137                                len = sprintf ((char *)buf,"\r\n MAIL_EN_10 =>
1138                                Pri lek. poplachu se neodesle mail");
1139                        }
1140                        return (len);
1141                        }
1142                        else
1143                        {
1144                                len = sprintf ((char *)buf,"\r\n MAIL_EN_10 =>
1145                                Neplatny parametr");
1146                                return (len);
1147                        }
1148      }
1149
1150                //Prikaz MAIL_EN_11
1151                if (tnet_ccmp (cmd, "MAIL_EN_11") == __TRUE) {
1152      if (len == 12) {
1153        sscanf((const char *)(cmd+11),"%d",&pom);
1154                        if (pom == 1)
1155                        {
1156                                MAIL_en_11 = 1;
1157                                set_MAIL_setup();
1158                                len = sprintf ((char *)buf,"\r\n MAIL_EN_11 =>
1159                                Pri pozarnim poplachu se odesle mail");
1160                        }
1161                        else
1162                        {
1163                                MAIL_en_11 = 0;
1164                                set_MAIL_setup();
1165                                len = sprintf ((char *)buf,"\r\n MAIL_EN_11 =>
1166                                Pri pozarnim poplachu se neodesle mail");
1167                        }
1168                        return (len);
1169                        }
1170                        else
1171                        {
1172                                len = sprintf ((char *)buf,"\r\n MAIL_EN_11 =>
1173                                Neplatny parametr");
1174                                return (len);
1175                        }
1176      }
1177
1178                //Prikaz MAIL_EN_12
1179                if (tnet_ccmp (cmd, "MAIL_EN_12") == __TRUE) {
1180      if (len == 12) {
1181        sscanf((const char *)(cmd+11),"%d",&pom);
```

```
1182                        if (pom == 1)
1183                        {
1184                                MAIL_en_12 = 1;
1185                                set_MAIL_setup();
1186                                len = sprintf ((char *)buf,"\r\n MAIL_EN_12 =>
1187                                Pri tisnovem poplachu se odesle mail");
1188                        }
1189                        else
1190                        {
1191                                MAIL_en_12 = 0;
1192                                set_MAIL_setup();
1193                                len = sprintf ((char *)buf,"\r\n MAIL_EN_12 =>
1194                                Pri tisnovem poplachu se neodesle mail");
1195                        }
1196                        return (len);
1197                        }
1198                        else
1199                        {
1200                                len = sprintf ((char *)buf,"\r\n MAIL_EN_12 =>
1201                                Neplatny parametr");
1202                                return (len);
1203                        }
1204     }
1205
1206                 //Prikaz MAIL_EN_13
1207                 if (tnet_ccmp (cmd, "MAIL_EN_13") == __TRUE) {
1208     if (len == 12) {
1209       sscanf((const char *)(cmd+11),"%d",&pom);
1210                        if (pom == 1)
1211                        {
1212                                MAIL_en_13 = 1;
1213                                set_MAIL_setup();
1214                                len = sprintf ((char *)buf,"\r\n MAIL_EN_13 =>
1215                                Pri naruseni objektu se odesle mail");
1216                        }
1217                        else
1218                        {
1219                                MAIL_en_13 = 0;
1220                                set_MAIL_setup();
1221                                len = sprintf ((char *)buf,"\r\n MAIL_EN_13 =>
1222                                Pri naruseni objektu se neodesle mail");
1223                        }
1224                        return (len);
1225                        }
1226                        else
1227                        {
1228                                len = sprintf ((char *)buf,"\r\n MAIL_EN_13 =>
1229                                Neplatny parametr");
1230                                return (len);
1231                        }
1232     }
1233
1234                 //Prikaz MAIL_EN_14
1235                 if (tnet_ccmp (cmd, "MAIL_EN_14") == __TRUE) {
1236     if (len == 12) {
1237       sscanf((const char *)(cmd+11),"%d",&pom);
1238                        if (pom == 1)
1239                        {
1240                                MAIL_en_14 = 1;
1241                                set_MAIL_setup();
1242                                len = sprintf ((char *)buf,"\r\n MAIL_EN_14 =>
1243                                Pri poplachu/poruse se odesle mail");
1244                        }
1245                        else
1246                        {
1247                                MAIL_en_14 = 0;
1248                                set_MAIL_setup();
1249                                len = sprintf ((char *)buf,"\r\n MAIL_EN_14 =>
1250                                Pri poplachu/poruse se neodesle mail");
1251                        }
1252                        return (len);
1253                        }
1254                        else
```

```
1255                        {
1256                                len = sprintf ((char *)buf,"\r\n MAIL_EN_14 =>
1257                                Neplatny parametr");
1258                                return (len);
1259                        }
1260    }
1261
1262                //Prikaz MAIL_EN_15
1263                if (tnet_ccmp (cmd, "MAIL_EN_15") == __TRUE) {
1264    if (len == 12) {
1265      sscanf((const char *)(cmd+11),"%d",&pom);
1266                        if (pom == 1)
1267                        {
1268                                MAIL_en_15 = 1;
1269                                set_MAIL_setup();
1270                                len = sprintf ((char *)buf,"\r\n MAIL_EN_15 =>
1271                                Pri poplachu 24hod smycky se odesle mail");
1272                        }
1273                        else
1274                        {
1275                                MAIL_en_15 = 0;
1276                                set_MAIL_setup();
1277                                len = sprintf ((char *)buf,"\r\n MAIL_EN_15 =>
1278                                Pri poplachu 24hod smycky se neodesle mail");
1279                        }
1280                        return (len);
1281                        }
1282                        else
1283                        {
1284                                len = sprintf ((char *)buf,"\r\n MAIL_EN_15 =>
1285                                Neplatny parametr");
1286                                return (len);
1287                        }
1288    }
1289
1290                //Prikaz MAIL_EN_20
1291                if (tnet_ccmp (cmd, "MAIL_EN_20") == __TRUE) {
1292    if (len == 12) {
1293      sscanf((const char *)(cmd+11),"%d",&pom);
1294                        if (pom == 1)
1295                        {
1296                                MAIL_en_20 = 1;
1297                                set_MAIL_setup();
1298                                len = sprintf ((char *)buf,"\r\n MAIL_EN_20 =>
1299                                Pri dohledovem poplachu se odesle mail");
1300                        }
1301                        else
1302                        {
1303                                MAIL_en_20 = 0;
1304                                set_MAIL_setup();
1305                                len = sprintf ((char *)buf,"\r\n MAIL_EN_20 =>
1306                                Pri dohledovem poplachu se neodesle mail");
1307                        }
1308                        return (len);
1309                        }
1310                        else
1311                        {
1312                                len = sprintf ((char *)buf,"\r\n MAIL_EN_20 =>
1313                                Neplatny parametr");
1314                                return (len);
1315                        }
1316    }
1317
1318                //Prikaz MAIL_EN_3
1319                if (tnet_ccmp (cmd, "MAIL_EN_3") == __TRUE) {
1320    if (len == 11) {
1321      sscanf((const char *)(cmd+10),"%d",&pom);
1322                        if (pom == 1)
1323                        {
1324                                MAIL_en_3 = 1;
1325                                set_MAIL_setup();
1326                                len = sprintf ((char *)buf,"\r\n MAIL_EN_3 =>
1327                                Pri technickem poplachu se odesle mail");
```

```
1328                        }
1329                        else
1330                        {
1331                                MAIL_en_3 = 0;
1332                                set_MAIL_setup();
1333                                len = sprintf ((char *)buf,"\r\n MAIL_EN_3 =>
1334                                Pri technickem poplachu se neodesle mail");
1335                        }
1336                        return (len);
1337                        }
1338                        else
1339                        {
1340                                len = sprintf ((char *)buf,"\r\n MAIL_EN_3 =>
1341                                Neplatny parametr");
1342                                return (len);
1343                        }
1344     }
1345
1346                 //Prikaz MAIL_EN_40
1347                 if (tnet_ccmp (cmd, "MAIL_EN_40") == __TRUE) {
1348     if (len == 12) {
1349       sscanf((const char *)(cmd+11),"%d",&pom);
1350                        if (pom == 1)
1351                        {
1352                                MAIL_en_40 = 1;
1353                                set_MAIL_setup();
1354                                len = sprintf ((char *)buf,"\r\n MAIL_EN_40 =>
1355                                Pri prichodu/odchodu se odesle mail");
1356                        }
1357                        else
1358                        {
1359                                MAIL_en_40 = 0;
1360                                set_MAIL_setup();
1361                                len = sprintf ((char *)buf,"\r\n MAIL_EN_40 =>
1362                                Pri prichodu/odchodu se neodesle mail");
1363                        }
1364                        return (len);
1365                        }
1366                        else
1367                        {
1368                                len = sprintf ((char *)buf,"\r\n MAIL_EN_40 => Nep
1369                                latny parametr");
1370                                return (len);
1371                        }
1372     }
1373
1374                 //Prikaz TONE_LEVEL
1375                 if (tnet_ccmp (cmd, "TONE_LEVEL") == __TRUE) {
1376     if (len > 11) {
1377       sscanf ((char *)(cmd+11),"%d",&pom);
1378
1379                                if ((pom < 513) && (pom >= 0))
1380                                {
1381                                        TONE_level = pom;
1382                                        set_TONE_setup();
1383                                        len = sprintf ((char *)buf,"\r\n TONE_LEVEL =>
1384                                        Nova hodnota je %d",pom);
1385                                        return (len);
1386                                }
1387                                else
1388                                {
1389                                        len = sprintf ((char *)buf,"\r\n TONE_LEVEL =>
1390                                        Neplatna hodnota (Hodnota musi byt v rozsahu 0 - 512");
1391                                        return (len);
1392                                }
1393                        }
1394     }
1395
1396          //Prikaz QUIT - ukonci spojeni
1397   if (tnet_ccmp (cmd, "QUIT") == __TRUE) {
1398     len = str_copy (buf, "\r\nOdpojeni...\r\n");
1399     return (len | 0x8000);
1400   }
```

```
1401
1402          //Prikaz CH_PSWD - zmena hesla
1403    if (tnet_ccmp (cmd, "CH_PSWD") == __TRUE && tnet_EnAuth) {
1404      if (len > 8)
1405                  {
1406        mem_copy (tnet_auth_passw, &cmd[8], 20);
1407                          NVOL_SetVariable(FLASH_TELNET_PASS, &cmd[8], 20);
1408        len = sprintf ((char *)buf, "\r\n CH_PSWD => Nove heslo
1409        je: %s",tnet_auth_passw);
1410                          return (len);
1411                  }
1412    }
1413
1414
1415          //Prikaz HELP
1416    if (tnet_ccmp (cmd, "HELP") == __TRUE || tnet_ccmp (cmd, "?") == __TRUE) {
1417      len = str_copy (buf,(U8 *)tnet_help1);
1418                  help_count = 1;
1419      return (len | 0x4000);
1420    }
1421
1422          //Nenalezen zadany prikaz
1423    len = str_copy  (buf, "\r\n==> Neznamy prikaz: ");
1424    len += str_copy (buf+len, cmd);
1425    return (len);
1426  }
1427
1428
```

B.1.6 setup.h

```
1
2  /*********************************************
3  Header setup.h - globalni promenne a definice
4
5  *********************************************/
6
7  #ifndef __SETUP_H
8  #define __SETUP_H
9
10 //Definice pozic pro ukladani ve FLASH pameti
11 #define FLASH_IP_ADR 1
12 #define FLASH_IP_SM 2
13 #define FLASH_IP_GW 3
14 #define FLASH_IP_DNS1 4
15 #define FLASH_IP_DNS2 5
16 #define FLASH_IP_MAC 6
17 #define FLASH_IP_DHCP 7
18 #define FLASH_IP_DEV 8
19 #define FLASH_DTMF_maxEZS 9
20 #define FLASH_DTMF_maxMan 10
21 #define FLASH_DTMF_befHand 11
22 #define FLASH_DTMF_befMess 12
23 #define FLASH_DTMF_befKiss 13
24 #define FLASH_TLF_number 14
25 #define FLASH_IPC_ADR 15
26 #define FLASH_IPC_PORT 16
27 #define FLASH_IPC_PROT 17
28 #define FLASH_IPC_RCVR 18
29 #define FLASH_IPC_PREF 19
30 #define FLASH_IPC_ACCT 20
31 #define FLASH_IPC_ANT 21
32 #define FLASH_IPC_TS 22
33 #define FLASH_IPC_INT 23
34 #define FLASH_IPC_KEY 24
35 #define FLASH_IPC_IV 25
36 #define FLASH_IPC_ME 26
37 #define FLASH_MAIL_FROM 27
38 #define FLASH_MAIL_TO 28
39 #define FLASH_MAIL_AUTH 29
40 #define FLASH_MAIL_USER 30
```

```
41 #define FLASH_MAIL_PASS 31
42 #define FLASH_MAIL_ADR 32
43 #define FLASH_MAIL_PORT 33
44 #define FLASH_MAIL_SET 34
45 #define FLASH_TONE_LEV 35
46 #define FLASH_TELNET_PASS 36
47 #define FLASH_RESET_SETUP 37
48
49 typedef unsigned char   U8;
50 typedef unsigned short  U16;
51
52 //Promenne pro DTMF
53 int DTMF_symbol[3][16] = {{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
54 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}};
55 //Uloziste DTMF symbolu z telefonni linky
56 int DTMF_number = 0;
57 //Pocet prijatych DTMF symbolu
58 int DTMF_phase = 0;
59 //Faze prijmu DTMF
60 int DTMF_maxEZS = 2000;
61 //Maximalni doba vyckavani na dalsi symbol z EZS [ms]
62 int DTMF_maxMan = 2000;
63 //Maximalni doba vyckavani na dalsi symbol pri manualni volbe [ms]
64 int DTMF_befHand = 500;
65 //Odstup handshake tono po poslednim symbolu volby [ms]
66 int DTMF_befMess = 2000;
67 //Maximalni doba vyckavani na CID zpravu po handshake tonu [ms]
68 int DTMF_befKiss = 1250;
69 //Odstup kiss-off tonu po uspednem prijeti zpravy [ms]
70 int TLF_number = 1213;
71 //Telefonni cislo prevodniku
72 int TLF_act = 0;
73 //Aktualne vytocene telefonni cislo
74 int DTMF_errors = 0;
75 //Pocet chyb pri prijmu DTMF
76 int DTMF_quant = 0;
77 //Pocet ulozenych CID zprav v DTMF_symbol
78
79 //Promenne pro Contact ID
80 int CID_MOK = 0;
81 //Je zprava validni? 0 - ne 1 - ano
82 int CID_TCP = 0;
83 //Je zprava odeslana po TCP
84 int CID_ACCT[4] = {0,0,0,0};
85 //Prijate cislo objektu
86 int CID_Q = 0;
87 //Prijaty symbol Q dle normy DC-05
88 int CID_XYZ[3] = {0,0,0};
89 //Prijate symboly XYZ dle normy DC-05
90 int CID_GG[2] = {0,0};
91 //Prijate symboly GG dle normy DC-05
92 int CID_CCC[3] = {0,0,0};
93 //Prijate symboly CCC dle normy DC-05
94
95 //Promenne pro IP CID
96 U8 IPC_ADR[4] = {192,168,0,101};
97 //IP adresa PCO - vychozi hodnota 192.168.0.101
98 U16 IPC_PORT = 1001;
99 //TCP port PCO - vychozi port 1001
100 int IPC_PROTOCOL = 1;
101 //(ne)sifrovany prenos - vychozi hodnota Sifrovany prenos
102 int IPC_seq = 1;
103 //Cislo sekvence
104 int IPC_rcvr = 0;
105 //Cislo prijimaciho zarizeni - vychizi hodnota 0 - nebude se prenaset
106 int IPC_pref = 0x0;
107 //Cislo prefixu objektu - vychozi hodnota 0
108 int IPC_acct = 0x1234;
109 //Cislo objektu - vychozi hodnota 1234
110 int IPC_ANT = 0;
111 //Povolen preklad cisla objektu - vychozi hodnota vypnuto
112 int IPC_ts = 1;
113 //Prenaseni datumu a casu ve zpravach? - vychozi hodnota ano
```

```
114 int IPC_interval = 10;
115 //Interval dohledovych zprav - vychozi hodnota 10vterin
116 int IPC_phase = 0;
117 //Faze odesilani zpravy po TCP
118 int IPC_terr = 0;
119 //Pocet chyb v jednom kroku
120 int IPC_rerr = 0;
121 //Pocet chyb pri jednom odesilani
122 int IPC_errors = 0;
123 //Celkovy pocet chyb odesilani po TCP
124 char IPC_key[33] = "aaaaaaaabbbbbbbb";
125 //Sifrovaci klic AES CBC - vychozi hodnota aaaaaaaabbbbbbbb (128bitovy klic)
126 char IPC_iv[17] = "aaaaaaaabbbbbbbb";
127 //Inicializacni vektor AES CBC - vychozi hodnota aaaaaaaabbbbbbbb
128 unsigned char IPC_name[16] = "CIDTCP";
129 //Nazev objektu - vychozi hodnota CIDTCP
130 int IPC_maxerr = 100;
131 //Maximalni pocet chyb pred odeslanim mailu - vychozi hodnota 100
132
133 //Promenne pro mail a hlaseni po mailu
134 char MAIL_from[33] = "xyz@xyz.com";
135 //Mailova adresa zarizeni - vychozi hodnota xyz@xyz.com
136 char MAIL_to[33] = "zyx@xyz.com";
137 //Mailova adresa prijemce        - vychozi hodnota zyx@xyz.com
138 char MAIL_user[33] = "xyz@xyz.com";
139 //Uzivatelske jmeno SMTP - vychozi hodnota xyz@xyz.com
140 char MAIL_pass[33] = "Password123";
141 //Heslo SMTP - vychozi hodnota Password123
142 char MAIL_sub[50];
143 //Predmet mailove zpravy
144 char MAIL_body[200];
145 //Obsah mailvoe zpravy
146 U8 MAIL_adr[4] = {192,168,0,102};
147 //IP adresa SMTP serveru - vychozi hodnota 192.168.0.102
148 U16 MAIL_port = 25;
149 //SMTP port - vychozi hodnota 25
150 int MAIL_auth = 1;
151 //Potreba SMTP authorizace - vychozi hodnota zapnuto
152 int MAIL_en_dp = 1;
153 //Odeslat mail pri problemu s prijemem z EZS - vychozi ANO
154 int MAIL_en_ipc = 1;
155 //Odeslat mail pri problemu s odeslanim na PCO - vychozi ANO
156 int MAIL_en_10 = 1;
157 //Odeslat mail pri lekarskem poplachu - vychozi ANO
158 int MAIL_en_11 = 1;
159 //Odeslat mail pri pozarnim poplachu - vychozi ANO
160 int MAIL_en_12 = 1;
161 //Odeslat mail pri tisnovem poplachu - vychozi ANO
162 int MAIL_en_13 = 1;
163 //Odeslat mail pri naruseni objektu - vychozi ANO
164 int MAIL_en_14 = 1;
165 //Odeslat mail pri poplachu/poruse - vychozi ANO
166 int MAIL_en_15 = 1;
167 //Odeslat mail pri poplachu 24hod smycky - vychozi ANO
168 int MAIL_en_20 = 1;
169 //Odeslat mail pri dohledovem problemu - vychozi ANO
170 int MAIL_en_3 = 1;
171 //Odeslat mail pri technickem probelmu - vychozi ANO
172 int MAIL_en_40 = 1;
173 //Odeslat mail pri odchodu/prichodu - vychozi ANO
174
175
176 //Promenne pro Tone
177 int TONE_phase = 0;
178 float TONE_level = 400;
179
180 int SET_reset = 0;
181
182 #endif
183
```