ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta elektrotechnická

Diplomová práce

2010 Václav Vrtěl

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta elektrotechnická

Síťový server pro CNC stroje

Vedoucí práce
Doc. Jan Bílek

Autor
Václav Vrtěl

Praha 2010

Originální zadání

Prohlášení

Prohlašuji, že jsem svou magisterskou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu § 60 Zákona č.121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne ……. …….

podpis

Poděkování

Chtěl bych poděkovat především své rodině, díky které jsem mohl nerušeně pracovat na projektu. Moje poděkování patří školiteli Tomu Mortensenovi, který mě vedl při psaní implementace protokolů. Chtěl bych také poděkovat Peteru Buble Johnsonovi, který mi umožnil pracovat na tomto projektu v Kodani celých pět týdnů.

Anotace

Cílem této diplomové práce je navrhnout a realizovat zařízení, které by umožňovalo komunikaci prostřednictvím počítačové sítě Ethernet mezi osobním počítačem a CNC strojem vybaveným sériovým portem a rovněž by umožňovalo přenášet stavové signály z CNC stroje do osobního počítače.

Annotation

Purpose of this master thesis is to design and create a system for allowing a communication over Ethernet between a personal computer and a CNC machine equipped with a serial port RS232. The device should allow transmitting state signals from a CNC machine to a personal computer.

Obsah

1 Úvod....1
1.1 Členění práce....2
2 Systém, pro který je zařízení navrhované....3
2.1 Systém NDC-Max....3
2.2 Systém MDC-Max....4
2.3 NC-Base....4
3 Požadavky na zařízení....6
3.1 HW požadavky....6
3.2 SW požadavky....7
3.3 Prostorové požadavky (požadavky na šasi)....8
3.4 Vyrobitelnost zařízení....8
3.5 Ekonomické požadavky....8
4 Výběr procesoru a vývojových nástrojů....10
4.1 Správná volba procesoru....10
4.2 Vybraní kandidáti....11
4.2.1 ATMEL....11
4.2.2 ARM procesory....12
4.2.3 MICROCHIP....15
4.3 Závěr rešerše o výběru vhodného procesoru....16
5 Použité vývojové nástroje....18
5.1 MPLAB IDE....18
5.2 Starter kit....18
5.3 Aplikační knihovny Microchipu....19
6 Software....20
6.1 Verzování softwaru....20
6.2 Microchip TCP/IP Stack....20
6.3 Naprogramované obslužné rutiny....21
6.4 TCP server obsluhující sériový port....22
6.4.1 Implementace serveru....23
6.4.2 Popis zprávy....25
6.4.3 Obsluha příchozích zpráv....26
6.4.3.1 Přijímání TCP paketů....28
6.4.4 Nastavení sériového portu....29
6.4.5 RX, TX FIFO zásobníky....32
6.4.6 Příjem datových zpráv....33
6.4.6.1 Heartbeat....34
6.4.7 Odesílání datových zpráv....34

6.4.8 Obsluha přerušení UART portu....35
6.4.8.1 Informování o stavu sériové linky....36
6.4.9 Výpočet parity a délky slova....36
6.5 Paměť pro uložení nastavení....37
6.5.1 I2C protokol....38
6.5.2 Sériová paměť EEPROM 24CXX....38
6.5.3 PIC32 a rozhraní I2C....39
6.5.4 Aplikační rozhraní pro zápis a čtení....40
6.6 Webové rozhraní....42
6.6.1 Návrh designu webového rozhraní....43
6.6.1.1 Členění webové stránky....43
6.6.1.2 Obsah webového rozhraní....44
6.6.2 Program pro vývoj webových stránek....45
6.6.3 Kaskádové styly....45
6.6.4 Implementace webového rozhraní....46
6.6.4.1 Souborový systém....46
6.6.4.2 Aplikační rozhraní pro prezentaci webových stránek....47
6.6.4.3 Čtení dat od uživatele....51
6.7 UDP server....53
7 Hardware....57
7.1 Klíčové komponenty....57
7.1.1 PIC32MX795F512L....58
7.1.2 DP83848C....58
7.1.3 MAX3243....58
7.2 Výběr součástek....59
7.2.1 Dodavatelé....59
7.3 Elektrické schéma....59
7.3.1 Procesorová deska....60
7.3.1.1 Zapojení procesoru....60
7.3.1.2 Napájení procesorové desky....61
7.3.1.3 Zapojení ethernetového obvodu DP83848C....61
7.3.1.4 Zapojení obvodu MAX3243....62
7.3.1.5 Přídavné konektory....63
7.4 Snímání signálů ze stroje....63
7.4.1 Reléové oddělení signálů....63
7.4.2 Optické oddělení signálů....64
7.4.3 Svorkovnice vstupní části....65
7.5 Plošný spoj....65
7.6 Pájení součástek prototypu....67
7.7 Navržené prototypy....67
7.7.1 Programování prototypu....68
7.8 Box pro zastavění plošného spoje....68
8 Rozpočet....70
9 Závěr....72
Literatura....74

Seznam tabulek

1 Stručné shrnutí požadavků na projekt....7
2 Přehledová kalkulace platformy AVR32....10
3 Přehledová kalkulace platformy ARM....11
4 Přehledová kalkulace platformy PIC32....13
5 Investice do vývoje na platformě PIC32....15
6 Funkce pro řízení rozhraní I2C....34
7 Parametry funkce MPFS.EXE....40
8 Finanční zatížení projektu....63

Seznam obrázků

1 Použité vývojové prostředky....15
2 Znázornění běhu Microchip TCP/IP stacku....17
3 Smyčka obslužných rutin....18
4 Vývojový diagram implementace protokolu....19
5 Formát zprávy....20
6 Princip převodu bitů do pole....26
7 Struktura vkládané zprávy....30
8 Adresa EEPROM podle propojení adresních vývodů....32
9 Výsledný design webového rozhraní....36
10 Vložení proměnné do webové stránky....40
11 Obsluha UDP serveru....46
12 Okno programu Quatechtool pro nastavení síťových parametrů....48
13 Základní doporučené zapojení procesoru PIC32....52
14 Napájecí část procesorové desky....53
15 Připojení oddělovacího transformátoru....54
16 Zapojení obvodu MAX3243....54
17 Reléová část....56
18 Zapojení optoisolátorů....58
19 Svorkovnice pro připojení signálů ze stroje....58
20 Procesorová deska a deska s galvanickým oddělením....59
21 Box pro vestavbu plošného spoje....61

Kapitola 1

Úvod

Zadavateli tohoto projektu jsou firmy CIMCO Integration v Kodani a její distributor pro Českou republiku firma Globus Praha s.r.o. Jedná se o spřátelené firmy zabývající se softwarem pro administraci CNC strojů a CAD/CAM systémy. Po dohodě s těmito firmami vznikl projekt, který se rozhodly finančně zaštítit. Cílem tohoto projektu, který byl pojat jako diplomová práce, je vytvořit elektronické zařízení umožňující přenos dat mezi osobním počítačem a CNC strojem využívající sériovou linku RS232. Zařízení představuje konvertor mezi počítačovou ethernet sítí a sériovým portem RS232. Dalším úkolem zařízení je snímat dvoustavové signály ze strojů a posílat je klientské aplikaci v osobním počítači.

Součástí projektu je výběr vhodného procesoru, na kterém se bude provozovat síťový server, a nalezení vhodných ladících a vývojových prostředků. Vybrat vhodné elektronické součástky a distributora, který by tyto součástky byl schopný zajistit. Vytvořit seznam součástek pro prototyp včetně objednávacích čísel, aby bylo možné je kdykoliv později nakoupit. Navrhnout elektronické schéma a na jeho základě vytvořit desku plošných spojů. Vybrat vhodného výrobce desek plošných spojů a po dohodě s ním nechat vyrobit prototypovou desku.

1.1 Členění práce

V druhé kapitole je stručně popsán systém, který má vyvíjené zařízení používat ke své činnosti. V kapitole číslo tři jsou zmíněné základní požadavky, které jsou kladené na zařízení a které má splňovat. Čtvrtá kapitola je rešerší o výběru vhodné vývojové platformy, na které bude později celý projekt založen. Následující pátá kapitola je věnována použitým vývojovým nástrojům. Šestá kapitola se zabývá softwarem, který byl pro zařízení vyvinut a na kterém stojí celý projekt. Sedmá

kapitola je věnována hardwaru, bez kterého software nemůže existovat a tvoří s ním nedílnou součást. Osmá kapitola je vyhrazena pro informace o finančním zatížení projektu. V poslední deváté kapitole je shrnutí a vyhodnocení celého projektu.

Kapitola 2

Systém, pro který je zařízení navrhované

Jedním ze zadavatelů tohoto projektu je dánská společnost CIMCO Integration se sídlem v Kodani zabývající se vývojem softwaru pro správu CNC strojů CAM systémy. Mezi programy pro správu CNC strojů patří produkty DNC-Max, MDC-Max a NC-Base. Vyjmenované programy využívají pro komunikaci s CNC strojem ethernet nebo sériovou linku, jak je v průmyslu zcela běžné. S rozvojem ethernetových sítí ve výrobních halách závodů vznikl požadavek na vzdálený přenos sériové linky a dalších signálů ze stroje a do stroje přes zabudovanou firemní ethernetovou síť. Tento projekt má být uspokojivým řešením zmíněného požadavku.

2.1 Systém NDC-Max

Prvním z produktů firmy CIMCO Integration je software NDC-Max určený ke komunikaci mezi osobním počítačem a CNC strojem. Tento program zajišťuje přenos programů mezi CNC strojem a počítačem po sériové lince RS232, ethernetu nebo přes síťový server se vzdálenými sériovými linkami. Veškeré interakce mezi CNC strojem a obsluhou jsou průběžně zaznamenávány. Pro zajištění všestrannosti tento program spolupracuje s jakýmikoliv běžně se vyskytujícími CNC stroji, a tak je možné pomocí jednoho programu obsluhovat několik různých CNC strojů. K serveru NDC-Max je možné se připojit pomocí klientského programu přes internet odkudkoliv.

U starších strojů, ale i mnoha nových, se ke komunikaci využívá standardní sériová linka RS232 a jelikož ne vždy je počítač pro obsluhu umístěn v blízkosti stroje, je nutné buď sériovou linku prodloužit od počítače až ke stroji, což není vždy optimální či vůbec možné, nebo využít zavedenou firemní počítačovou ethernet síť, ať již drátovou verzi či bezdrátovou, která je v moderních výrobních podnicích zpravidla velmi dobře dostupná po celé výrobní hale.

Pro případ, že se pro komunikaci používá síť ethernet, je nutné použít zařízení, které by umožnilo převádět data z ethernet sítě na sériovou linku. V podstatě se jedná o mikropočítač, na kterém běží TCP, popřípadě UDP server. K tomuto serveru se připojí klient, který chce využívat jeho služeb. Komunikace mezi serverem a klientem probíhá podle komunikačního protokolu určeného právě pro tento účel. Firma Cimco Integration se rozhodla pro projekt vlastního zařízení, které by tuto konverzi dat zajišťovalo a které by usnadnilo nasazení jejich produktů především v kombinaci s dalším jejich produktem MDC-Max.

2.2 Systém MDC-Max

MDC-Max je program, který vyhodnocuje přijaté signály ze strojů a umožňuje jejich databázové a statistické zpracování. Tento program je určen především pro sledování efektivity výroby a vytíženosti jednotlivých strojů. Je možné tak zjistit, v jakých situacích má stroj největší prostoje (například při výměně nástroje či při čekání na materiál apod.) nebo naopak kdy dosahuje největší produktivity. Rovněž je možné vysledovat, jak dlouho trvá daná úloha na konkrétním stroji a porovnat ji v grafu s jinými stroji.

Program také nabízí živé sledování strojů na obrazovce, např. v kanceláři dílny-vedoucího či třeba ředitele podniku. Program není omezen jen na lokální síť, ale je možné se pomocí klientského programu k němu připojit odkudkoliv přes internet. Rovněž může být zaznamenávána obsluha stroje a tím se může vysledovat např. kdo jaký obrobek vyprodukoval a sledovat tak kvalitu a hospodárnost výroby, o kterou zákazníci usilují.

MDC-Max může sledovat jakékoliv stavové signály, např. vřeteno se točí, probíhá výměna nástroje, vyndávání obrobku nebo sledování kabinových dveří apod., vše záleží na zákazníkovi. Aby toto bylo dost dobře možné, je třeba zařízení, které by snímalo dané signály a pomocí komunikačního protokolu je předávalo přes síť ethernet počítači, na kterém běží MDC-Max server.

2.3 NC-Base

Pro úplnost bude představen poslední produkt, který je určen pro správu CNC strojů. Jedná se o program NC-Base, který je databází programů pro CNC stroje. Umožňuje jejich verzování, přidávání příloh, rozděluje pravomoc podle nastavených uživatelských kont apod. Všechny tři programy jsou vzájemně propojeny a do sebe začleněny.

Kapitola 3

Požadavky na zařízení

Na projekt jsou kladené nároky, které by měly být v projektu dodrženy. Požadavky byly vytvořeny na základě konzultace se zadavateli projektu. Tyto požadavky na výsledné zařízení a i celý projekt lze rozdělit do několika základních skupin a to hardwarové požadavky, softwarové požadavky, ekonomické požadavky a prostorové požadavky, nebo-li požadavky na šasi zařízení a konečná vyrobitelnost celého zařízení v dostupných firmách.

3.1 HW požadavky

Hardware může být libovolné platformy založený na libovolném procesoru a systému. Výsledné zařízení musí být dostatečně rychlé pro obsluhu dvou sériových linek RS232 a 8 digitálních vstupů, jejichž stav má sledovat a o změně informovat připojeného klienta. Výběr platformy byl ponechán na volbě a rozhodnutí autora.

Komunikace se zařízením musí být vedena po drátovém ethernetu standardu 100Base-TX (Fast Ethernet). Tedy má být možné ho připojit do běžné počítačové sítě dnešní doby (rok 2010). Do budoucna se uvažuje o rozšíření komunikace o bezdrátový standard IEEE 802.11b.

Obě sériové linky musí obsahovat všechny modemové signály (RI, RTS, CTS, TX, RX, DSR, DTR, CD). Musí být zajištěno řízení toku jak hardwarové tak i softwarové. Nutnou podmínkou je možnost 7-bitové komunikace s paritou. Vyvedení sériové linky má být pomocí konektoru CANON 9M, standardní 9-kolíkový konektor známý z osobních počítačů. Sériová linka musí být schopna plynule komunikovat rychlostí nejméně 9600 baud/s.

Stavové signály snímané na stroji musí být galvanicky oddělené od tohoto zařízení. Zařízení

musí akceptovat 0 V na daném vstupu jako bit s hodnotou 0 a 12 až 24 V jako bit s hodnotu 1. Je požadováno nejméně 8 takovýchto vstupů. Tyto vstupy musí být zpřístupněny na vnější svorkovnici. Tyto signálové vstupy mohou být řešené jako volitelné rozšíření výsledného produktu. Mohou existovat tedy dvě verze produktu, jedna verze obsahující tyto vstupy a druhá verze bez těchto vstupů.

Zařízení má být složeno z dostupných moderních součástek tak, aby mohly být splněny veškeré požadavky kladené na konečné zařízení. Zařízení musí být, co se týče výběru součástek, navrženo tak, aby byla zajištěna jeho pozdější vyrobitelnost v dostupných montážních firmách.

Zařízení musí obsahovat resetovací tlačítko, které uvede v zařízení všechny hodnoty do předem známého výchozího stavu.

3.2 SW požadavky

Komunikace mezi zařízením a klientským počítačem má probíhat na úrovni TCP/IP protokolu. Samotné dorozumívání TCP serveru běžícím na navrhovaném zařízení a klientem běžícím na osobním počítači se děje pomocí pravidel, která jsou shodná s protokolem Intelisocket [11] určeným právě pro výměnu informací mezi serverem obsluhující sériový port a klientem, který využívá jeho služeb. K serveru se může připojit právě jeden klient. Každý sériový port na zařízení bude obsluhován jedním TCP serverem na daném TCP portu, číslo TCP portu začíná na hodnotě 5000 a každému dalšímu portu je přiřazeno číslo o jednu větší.

Na zařízení má být v provozu i UDP server, který bude naslouchat na UDP portu 49344. Pomocí tohoto UDP serveru má být možné nastavovat parametry nutné pro síťovou komunikaci jako je IP adresa a další. UDP server komunikuje pomocí UDP Quatech protokolu [10].

Na zařízení má rovněž běžet webový server, který je dostupný přes běžný webový prohlížeč a pomocí kterého je možné rovněž nastavovat parametry nutné pro síťovou komunikaci.

Přenos signálů ze stroje má probíhat podle protokolu, který v tomto okamžiku ještě není znám (má být dodán firmou Cimco Integration).

3.3 Prostorové požadavky (požadavky na šasi)

Výsledné zařízení by mělo být zastavěné do vhodného šasi z plastu nebo kovu. Buď navrhnout vlastní šasi, které by se dalo nechat vyrobit nebo využit šasi, která jsou již hotová a dostupná na trhu a popřípadě navrhnout nutné změny, které by se na nich musely udělat, jako vyvrtání otvorů apod.

Na šasi nejsou kladené žádné zvláštní požadavky.

3.4 Vyrobitelnost zařízení

Veškeré navržené díly a součásti musí být na trhu běžně dostupné. Navržené postupy musí být snadno výrobně proveditelné běžnými technologiemi pro výrobu elektronických zařízení. Má být vybrán dodavatel součástí a zvolené díly, které bude nutné objednat, musí být uvedeny včetně objednávacích čísel a kontaktu na dodavatele, u kterého se budou součástky nakupovat.

3.5 Ekonomické požadavky

Zařízení musí využívat takové součástky, které jsou pokud možno nejekonomičtější a s parametry zajišťující splnění ostatních požadavků (dostupnost, funkce apod.). Použité vývojové prostředí včetně knihoven funkcí má být vybráno tak, aby mohlo být splněno zadání a cena byla na přijatelné úrovni pro zadavatele. S vývojovým prostředím je spojena i volba procesoru a platformy, jejichž cena, výkon a využitelnost má být brána v potaz. Konečnou volbu poté schvaluje zadavatel.

č.	Požadavek
1	TCP server komunikující protokolem INTELISOCK
2	UDP server komunikující UDP Quatech protokolem
3	Webové rozhraní pro nastavování síťových parametrů
4	Mikropočítač libovolné platformy
5	Snadno dostupné součástky
6	8 vstupů ze stroje galvanicky oddělené od navrhovaného zařízení
7	2 sériové porty RS232 se všemi modemovými signály
8	Podpora 7-bitového přenosu s paritou
9	Vhodné šasi z libovolného materiálu, vhodného tvaru a velikosti
10	Ekonomicky výhodné řešení

Tabulka 1: Stručné shrnutí požadavků na projekt

Kapitola 4

Výběr procesoru a vývojových nástrojů

Důležitým krokem v tomto projektu byla volba procesoru a s ním spojené vývojové a ladící nástroje. Na trhu existuje celá řada produktů a zkoumat všechny nabízené procesory, či jen jejich platformy, by zabralo mnoho času a mapování trhu s procesory není náplní tohoto projektu. Výběr procesorů vycházel ze známých platforem a výrobců, kteří jsou už léty osvědčeni a mají širokou podporu v komunitě vývojářů, ať již profesionálních či amatérských. Díky tomu existuje mnoho zdrojů informací, například na internetu nebo v různých odborných publikacích.

4.1 Správná volba procesoru

Vhodný procesor nelze volit jenom podle ceny, popřípadě podle vybavenosti nebo jen podle jeho výkonu. S procesorem jsou spojeny nástroje pro jeho programování, ladění a v neposlední řadě i překladač kódu. Mnozí výrobci nabízí své vlastní řešení, jiní se spoléhají na řešení třetích firem, na které se odkazují.

Mezi výrobce, kteří nabízí své vlastní procesory a svá vlastní řešení ladících nástrojů, vývojové prostředí a nespočet příkladů a programových knihoven, patří beze sporu ATMEL a MICROCHIP. Oba výrobci vychází svojí cenovou politikou zákazníkovi vstříc a jsou díky tomu značně oblíbeni. Oba výrobci vyvíjí procesory vlastní koncepce, kromě toho ATMEL nabízí i procesory založené na známé architektuře ARM.

Procesory ARM patří v dnešní době mezi jedny z nejvýkonnějších procesorů a dočkaly se širokého rozšíření do spotřební elektroniky a lze je nalézt prakticky všude. Výrobců těchto procesorů je celá řada, ve svém programu je má mnoho výrobců polovodičových součástek. Avšak ne každý výrobce nabízí podporu pro tyto procesory. Procesory Arm jsou zpravidla podporovány

třetí stranou, což je také umožněno jeho jednotnou platformou. Produkty třetích firem jsou profesionální nástroje nabízející špičkovou kvalitu a dobrou technickou podporu, zároveň jsou však velmi drahé, ceny často překračují statisícové částky. ARM procesory si našly také velkou oblibu mezi amatérskými vývojáři a tak je možné najít ladící nástroje a překladače, které jsou zadarmo. Někteří výrobci ARM procesorů nabízí programové knihovny za 0,0 € nebo za velmi nízký poplatek, které lze komerčně používat.

Rozhodujícími faktory při výběru procesoru byla cena programátoru a ladícího nástroje (zpravidla se jedná o jedno zařízení), cena vývojového prostředí včetně překladače, cena procesoru, cena a vhodnost vývojového kitu jakožto jeho existence vůbec a v poslední řadě dostupnost knihovny pro komunikaci TCP/IP. V úvahu bylo vzato i nabízené pohodlí při vývoji, ladění a nahrávání programů.

Důležitost správné volby procesoru by neměla být podceňována, jelikož může ovlivnit případný vývoj dalších projektů u zadavatelských firem.

4.2 Vybraní kandidáti

Mezi kandidáty byly zařazeny procesory firmy ATMEL, MICROCHIP a procesory s jádrem ARM 7. Důvodem tohoto prvotního výběru byly zkušenosti s procesory ATMEL, povědomí o procesorech ARM a dobrá pověst procesorů MICROCHIP.

4.2.1 ATMEL

Firma Atmel [6] má ve svém výrobním programu jak 8-bitové procesory, tak 32-bitové procesory a vyrábí i procesory založené na jádře ARM. Jelikož ARM procesory jsou jednotnou platformou víceméně nezávislé na výrobci procesoru, nebude se tato kapitola ARM procesory zabývat.

Z hlediska dostatečné výkonové rezervy a orientace na trend a moderní technologie, byla jako kandidát zvolena platforma AVR32, což jsou 32-bitové procesory založené na vlastní architektuře firmy ATMEL. Jedná se o moderní výkonné plně 32-bitové procesory s podporou DMA kanálů a vysokou integrací periférií v čipu procesoru a to vše doplněno jeho nízkou spotřebou. Některé modely jsou vybaveny rozhraním ethernet MAC, které ve spolupráci s obvodem zajišťujícím přístup k fyzické vrstvě za pomocí rozhraní MII (Media Independent Interface) nebo RMII (Reduced Media Independent Interface) vytváří plnohodnotný a velmi výkonný přístup do ethernet sítě. Procesory standardně disponují i několika UART moduly, které lze použít pro

komunikace po sériové lince RS232. Moduly jako I2C či ISP jsou pro tyto procesory naprostou samozřejmostí. K některým modelům je možné připojit i externí SDRAM modul pro zvětšení paměti RAM. Díky tomu je možné na těchto procesorech provozovat i operační systém LINUX.

Mezi výhody procesorů s platformou AVR32 patří velmi kvalitní podpora výrobce, který k vývoji aplikací poskytuje programátorům vývojové prostředí zcela zdarma. Rovněž poskytuje zdarma celou řadu knihoven a funkcí včetně podpory TCP/IP komunikace [5]. Výrobce nabízí i několik vývojových kitů, na kterých je možné vyvíjené aplikace hned zkoušet a provozovat. Takovýto starter kit může být chápán jako nultý prototyp s výhodou okamžitého nasazení a odlaďování dané aplikace.

Mezi velké výhody platformy avr32 patří nízké náklady nutné k zahájení vývoje aplikací na něm založených a hlavně jeho bezproblémové a rychlé připojení k osobnímu počítači, na kterém běží avr-vývojové prostředí.

Popis	Podrobnosti	Cena
Vývojové prostředí	AVR Studio	0,00 Kč
Překladač	AVR-GCC	0,00 Kč
Ladící nástroj	AVR-GDB	0,00 Kč
Vývojový kit	ATEVK1100	3 130,00 Kč
Softwarové knihovny pro vývoj aplikací včetně TCP/IP	UC3	0,00 Kč
Programátor a debugger procesorů AVR32	ATJATGICE2	7 792,00 Kč

Tabulka 2: Přehledová kalkulace platformy AVR32

4.2.2 ARM procesory

Výrobců procesorů s jádrem ARM je na světě celá řada. Výběr konkrétního výrobce není jednoduchým úkonem. ARM procesory jsou obecně podporovány až třetími stranami, samotní výrobci jim prostoru mnoho nevěnují (podobně jako ATMEL nebo MICROCHIP svým platformám), i když i zde by se našly výjimky. ARM procesor je velmi výkonný a některé typy předčí procesor AVR32 od ATMELu či procesor PIC32 od MICROCHIPu. Rovněž jejich proudový odběr může být ve srovnání s jinými platformami v některých situacích nižší. ARM procesor se jeví jako nejlepší kandidát pro nasazení snad v jakékoliv sofistikovanější aplikaci. Procesory Arm jsou bohatě vybavovány různými periferiemi jako je I2C, ISP, UART, ETHERNET MAC, rozhraním pro

připojení externí RAM paměti atd.. Na ARM procesorech lze velmi úspěšně provozovat operační systém LINUX. Pro vývoj programů pro procesory ARM existuje celá řada prostředí, ať již komerčních či bezplatných. Ale ve srovnáni s ATMELem nebo MICROCHIPem působí bezplatná prostředí méně pohodlně a nevzbuzují potřebnou jistotu naprosté bezproblémovosti, čemuž byla přisuzována nejvyšší priorita. Ne všechny ARM procesory jsou v bezplatných vývojových prostředích dobře podporovány a komerční prostředí jsou značně drahá, přestože o jejich profesionalitě nelze pochybovat. Existují i volně dostupné TCP/IP knihovny, které však nedosahují takových kvalit jako nabízí MICROCHIP či ATMEL. Knihovnu TCP/IP nabízí NXP ke svým ARM procesorům, přesto však ve srovnání s konkurenčními platformami není tak bohatě propracovaná, přestože nabízí vše základní.

Existuje celá řada vývojových kitů v celé škále cenového rozpětí. K programování a odladění je ale potřeba programátor. Kvalitní programátor často přesahuje hranici 1000 dolarů, např.: "ULINK*pro* Debug and Trace Unit $1395,0" od firmy Keil. Existují i mnohem levnější řešení založená na OpenOCD pro ladění, překladači GCC, to vše integrované do vývojového prostředí Eclipse. Toto řešení nemusí být 100% spolehlivé a bezproblémové a má i řadu omezení (například na konkrétní typy procesorů).

V tabulce 3 je uvedena stručná kalkulace. Byly použity co možná nejlevnější vývojové nástroje, při volbě jakéhokoliv profesionálního nástroje se ceny velmi dramaticky zvedaly.

Popis	Podrobnosti	Cena
Vývojové prostředí	Eclipse, WinARM	0,00 Kč
Překladač	arm-gcc	0,00 Kč
Ladící nástroj	OpenOCD	0,00 Kč
Vývojový kit	Low cost Olimex LPC2378 prototyping board LPC-P2378, [4]	1 677,60 Kč
Softwarové knihovny pro vývoj aplikací včetně TCP/IP	NXP TCP/IP stack	0,00 Kč
Programátor a debugger procesorů ARM	Keil ULINK pro, [20]	24 412,50 Kč

Tabulka 3: Přehledová kalkulace platformy ARM

4.2.3 MICROCHIP

Posledním kandidátem zařazeným do výběru byl výrobce procesorů MICROCHIP [8] , který je znám především svými procesory „PIC“ a mnoha dalšími polovodičovými součástkami.

Firma MICROCHIP vyrábí jak 8-bitové procesory, tak i procesory 16-bitové. Na základě svých dlouholetých zkušeností a velkého úspěchu se svými procesory se rozhodla vyvinout nový 32-bitový procesor, který představila v roce 2007.

Jelikož procesorová řada PIC32 patří k nejvýkonnější a nejpokrokovější, co MICROCHIP nabízí, byla pozornost zaměřena právě na tento typ procesoru. Procesory PIC32 jsou vybaveny celou řadou periférií, mezi které patří i ethernet MAC umožňující jejich bezproblémové nasazení v ethernet sítích. Další užitečné periférie jsou I2C, SPI, CAN a pro projekt důležitá periférie UART, kterých některé modely PIC32 integrují až 6.

Microchip nabízí celou řadu vývojových kitů, které jsou koncipovány stavebnicově a rozšiřující moduly jsou kompatibilní se všemi procesorovými kity jak hardwarově, tak i v samotném softwaru, který Microchip dodává volně ke svým procesorům. Dodávané softwarové knihovny jsou robustně koncipovány a velmi propracované s bohatou nápovědou a různými tutoriály, které ulehčují začátečníkům start s jejich procesory. Microchip nabízí celou řadu kvalitně propracovaných příkladů použití. K dispozici programátorům je i fórum, které je přímo na webových stránkách Microchipu a do kterého přispívají i samotní vývojáři Microchipu. Podpora ze strany Microchipu je veliká. Microchip dává volně k dispozici i knihovnu pro práci s TCP/IP sítí. TCP/IP knihovna je velmi detailně propracovaná, připravená k okamžitému použití a velmi lehce konfigurovatelná přes jednoduchého průvodce. Podporuje celou řadu ethernet obvodů, přes ethernet (MAC) + ethernet (FYZ) až k mnoha ISP ethernet obvodům. Mezi velké přednosti této knihovny patří i podpora WiFi modulu, kterého je Microchip rovněž výrobcem.

Microchip dává k dispozici zdarma i vývojové prostředí včetně překladače. Od placené verze se může lišit výslednou velikostí zkompilovaného kódu, který umí placená verze zmenšit až o 50% v závislosti na licenci. Neplacená verze neumožňuje využít optimalizace překladače. Microchip nabízí i propracované programátory a debuggery za příznivou cenu. Rovněž vývojové starter kity jsou cenově příznivé. Mezi další výhody patří i dostupnost veškerých produktů Microchipu přes jeho vlastní internetový obchod *microchipdirect [23]*.

Popis	Podrobnosti	Cena
Vývojové prostředí	MPALB microchip	0,00 Kč
Překladač	microchip-gcc	0,00 Kč
Ladící nástroj	MPLAB microchip	0,00 Kč
Vývojový kit	Ethernet starter kit	1 225,00 Kč
Softwarové knihovny pro vývoj aplikací včetně TCP/IP	Microchip application library	0,00 Kč
Programátor a debugger procesorů PIC32	MPLAB ICD 3	3 500,00 Kč

Tabulka 4: Přehledová kalkulace platformy PIC32

4.3 Závěr rešerše o výběru vhodného procesoru

Na základě této krátké rešerše o procesorech a jejich vývojových platformách lze usoudit, že procesory ARM nemohou být v tomto projektu použity, jelikož kvalitní vývojová prostředí překračují akceptovatelnou cenu a vývojové nástroje, které by byly finančně přijatelné, nenabízí takovou profesionální úroveň a jistotu 100% funkčnosti na první zapojení jako konkurenční platformy od Microchipu či Atmelu. Přes zjevně lepší vlastnosti ARM procesorů oproti konkurenci není možné postavit projekt na platformě ARM procesoru.

Dalšími kandidáty se stávají 32-bitové procesory firem Atmel a Microchip. Oba procesory nabízejí dostatečný výkon s velkou rezervou. Hardwarová výbava obou procesorů je na stejné úrovni a podstatně se neliší. Oba výrobci nabízejí zdarma profesionální vývojové prostředí s kompilátorem a propracovaným odlaďováním programů. Rovněž nabízí kvalitní profesionální vývojové starter kity ulehčující začátek vývoje zákaznických aplikací. Oba výrobci ke svým platformám nabízí zdarma různé aplikační softwarové knihovny včetně knihovny pro komunikaci TCP/IP.

Microchip ve srovnání s Atmelem nabízí větší podporu programátorům, například přes diskuzní fórum, do kterého přispívají samotní vývojáři Microchipu. Rovněž Microchip nabízí softwarovou knihovnu pro práci s WiFi sítí a WiFi obvodem, který sám vyrábí. Samotné procesory PIC32 jsou ve srovnání s procesory AVR32 levnější. Tyto skutečnosti jsou rozhodující.

Na základě všech uvedených argumentů byla pro tento projekt vybrána platforma PIC32 od Microchipu [8], která splňuje veškeré požadavky.

Kapitola 5
Použité vývojové nástroje

Pro vývoj na platformě Microchipu PIC32 je zapotřebí vývojového kitu, vývojového prostředí, aplikačních knihoven a studijního materiálu pro jejich použití.

5.1 MPLAB IDE

Vývojové prostředí MPLAB je k dispozici na webových stránkách společnosti Microchip [8]. V současnosti (listopad 2010), je k dispozici ve verzi 8.60. Instalační program nainstaluje všechny potřebné komponenty nutné pro vývoj včetně překladače. MPLAB IDE obsahuje i simulátor procesorů PIC a je tedy možné některé věci testovat bez přítomnosti fyzického PIC procesoru.

5.2 Starter kit

Jelikož se jedná o aplikaci využívající ethernet, byl jako vývojový kit zvolen *Ethernet starter kit*, který obsahuje procesor *PIC32MX795F512L*, 3 tlačítka a 3 LED-diody pro libovolné použití, ethernet konektor, USB-host konektor a USB konektor určený k odlaďování nahrávání programů. Celý starter kit je napájen z hostitelského počítače a nevyžaduje tedy žádný externí zdroj.

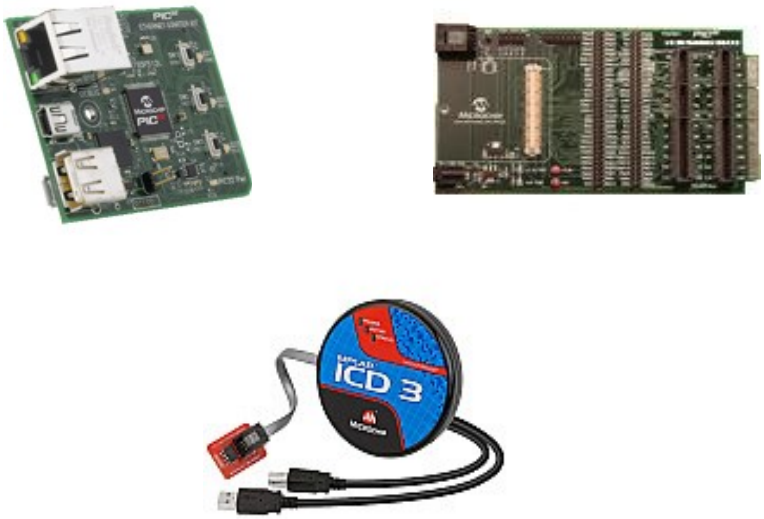
Pro přístup ke všem signálům je potřeba tento vývojový kit připojit k rozšiřující desce *„expansion board“*.

5.3 Aplikační knihovny Microchipu

Na webových stránkách Microchipu [8], je možné stáhnout celou řadu aplikačních příkladů a knihoven a to i TCP/IP stack [9], který zajišťuje komunikaci po drátovém ethernetu, ale i bezdrátovým způsobem pomocí WiFi.

Položka	Obchod	Objednávací číslo	Cena
PIC32 I/O Expansion Board	www.microchipdirect.com [21]	DM320002	53.73 €
PIC32 Ethernet Starter Kit	www.microchipdirect.com [21]	DM320004	53.73 €
MPLAB ICD 3 In-Circuit Debugger	www.microchipdirect.com [21]	DV164035	141.78 €

Tabulka 5: Investice do vývoje na platformě PIC32



Obrázek 1: Použité vývojové prostředky (převzato z [21])

Kapitola 6

Software

Klíčovou záležitostí tohoto projektu je programové vybavení pro vyvíjené zařízení, které má zabezpečit požadované funkce. Pro vývoj softwau bylo využito programu MPLAB IDE od firmy Microchip [8], včetně integrovaného překladače a debuggeru, pro platformu Microsoft Windows.

6.1 Verzování softwaru

Jelikož se jedná o softwarový projekt většího rozsahu, je nutné použít systém pro verzování jednotlivých vývojových kroků. Pomocí verzování je možné při případné chybě se vrátit k funkční verzi a nalézt příčinu nynější nefunkčnosti či opustit nesprávnou větev vývoje, vrátit se k předchozí a začít lepší myšlenkou. Tuto skutečnost si autor během práce na tomto projektu sám ověřil. Verzování může ušetřit mnoho času marným hledáním chyby, která je neustále přehlížena. Verzovací systémy umožňují porovnávání různých verzí mezi sebou se zvýrazněním rozdílných částí kódu.

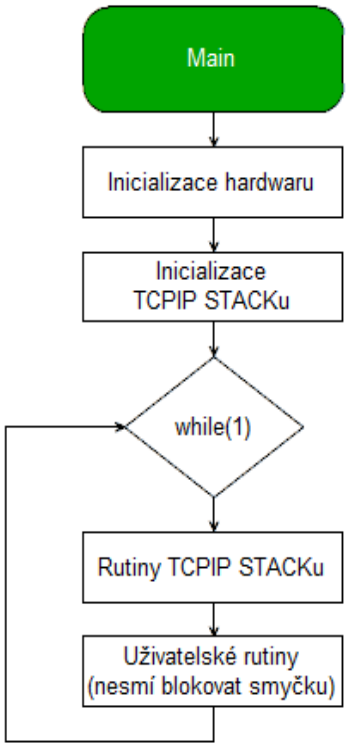
V tomto projektu byl použit verzovací systém *TortoiseSVN verze 1.6.10*.

6.2 Microchip TCP/IP Stack

Programová výbava je založená na knihovně TCP/IP Stack [9] od firmy Microchip. Knihovna je nabízená zcela volně pod licencí "Royality license". Díky této licenci lze knihovnu použít i v komerčních aplikacích, kód lze libovolně měnit a používat podle potřeby. Knihovnu lze používat doživotně.

Knihovna je založená na kooperativním multitaskingu. Je tedy nutné, aby se kód nikde nezastavil, například při čekání na splnění nějaké podmínky a podobně, a aby žádná funkce v

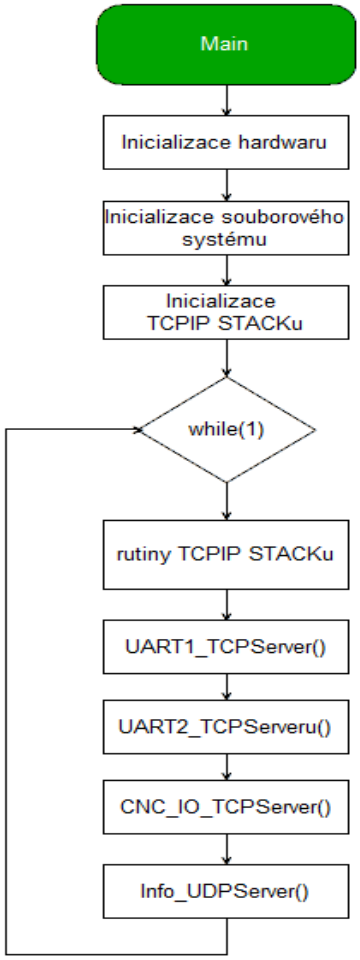
programu netrvala příliš dlouho, jelikož by to případně i znemožnilo správnou funkci síťové komunikace. V programové smyčce TCP/IP stacku je i obsluha síťové komunikace a všech paketů a síťových požadavků. Princip TCP/IP stacku je znázorněn na obrázku 2.



Obrázek 2: Znázornění běhu Microchip TCP/IP stacku

6.3 Naprogramované obslužné rutiny

Pro požadovanou funkčnost vyvíjeného komunikačního zařízení jsou zapotřebí celkem čtyři obslužné rutiny zabezpečující požadované komunikační schopnosti zařízení. Obsluha jednotlivých sériových portů, v tomto případě dvou, vyžaduje dva TCP servery naslouchajících na různých portech. TCP server obsluhující události (signály) získané z CNC stroje. Dále UDP server naslouchající na daném portu, který umožňuje zařízení v síti identifikovat a změnu jeho parametrů jako IP adresy apod. Na obrázku 3 je znázorněn graf obsluhy rutin. A v poslední řadě funkce pro čtení a zápis konfiguračních parametrů TCP/IP Stacku do paměti připojené na rozhraní I2C. TCP/IP Stack nabízí uložení parametrů pouze do pamětí komunikující přes rozhraní ISP.



Obrázek 3: Smyčka obslužných rutin

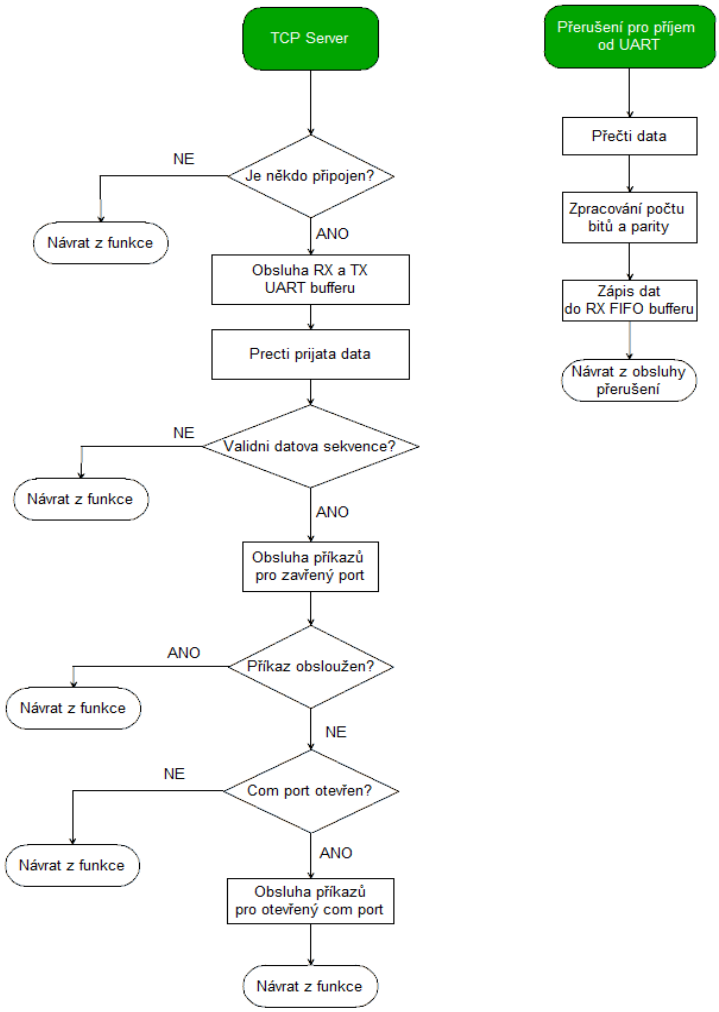
6.4 TCP server obsluhující sériový port

Pro běh komunikace mezi počítačem a zařízením je použit TCP server, který implementuje komunikační protokol Intelisock [11]. Jedná se o protokol firmy Quatech, která jej nabízí volně ke svým produktům pro umožnění komunikace mezi uživatelem a produktem firmy Quatech. DNC server firmy Cimco Integration implementuje mezi jinými i tento protokol. Aby možné ověřit správnou funkci tohoto navrhovaného zařízení aniž by museli vývojáři hned implementovat nový protokol, bylo pro vývoj použito protokolu firmy Quatech. Toto je jedna z podmínek zadavatele. Pro komerční použití se musí implementovat vlastní protokol.

6.4.1 Implementace serveru

Celý protokol je vepsán do funkce `GenericTCPServer()` a `GenereicTCPServer1()`. Obě funkce jsou volány z hlavní programové smyčky. Jsou obsluhovány dva sériové porty, každý sériový port má vlastní TCP server, který naslouchá na daném portu. Číslování TCP portů pro jednotlivé porty RS232 začíná od čísla 5000. Tedy port RS232 číslo 1 bude k dispozici na TCP portu 5000, port RS232 číslo 2 bude k dispozici na TCP portu číslo 5001 atd. Ke každému TCP portu se může připojit nejvýše jeden klient.

Komunikační protokol je navržený jako rozhodovací strom, podle dané události se rozhodne o dalším následném úkonu. Pro lepší představu je přehledový vývojový diagram rozhodování podle událostí zobrazen na obrázku 4.



Obrázek 4: Vývojový diagram implementace protokolu

Quatech protokol je založený na výměně jednoduchých zpráv přesně daného formátu. Všechny zprávy si obě strany vzájemně potvrzují podle daných pravidel.

Na začátku funkce je kontrolováno, zda-li je připojený k serveru klient. Není-li nikdo připojený, následuje návrat do hlavní programové smyčky. V případě, že UART port byl klientem již otevřen a klient mezi tím ukončil TCP spojení se serverem, je UART port ihned uzavřen a tím pádem je přichystaný k dalšímu připojení. Po připojení klienta systém čeká na příchozí zprávy. Na následující ukázce je zkrácená funkce, která byla teď popisována.

```
void GenericTCPServer(void)
{
          switch(TCPServerState2)
          {
                    case SM_HOME:
                              // Allocate a socket for this server to listen and accept connections on
                              MySocket2 = TCPOpen(0, TCP_OPEN_SERVER, SERVER_PORT2, TCP_PURPOSE_GENERIC_TCP_SERVER);
                              if(MySocket2 == INVALID_SOCKET)
                                        return;
                              TCPServerState2 = SM_LISTENING;
                              StructQueueInit(&fifoRX2, SIZE_OF_FIFO_RX); //initialize fifoUART buffer
                              StructQueueInit(&fifoTX2, SIZE_OF_FIFO_TX); //initialize fifoHOST buffer
                              return;

                    case SM_LISTENING:
                    {         // See if anyone is connected to us
                              if(!TCPIsConnected(MySocket2))
                              {
                                        if(portIsOpened2 == 1) //conection lost, close com port if is opened
                                                  portIsOpened2 = CloseComPort();
                                        return;
                              }
                               ...// do protocol routines
                    }
}
```

6.4.2 Popis zprávy

Každá zpráva má danou strukturu, která je zobrazena na obrázku 5. Pokud není přesně dodržená struktura, je zpráva ignorována.

0x1b	0x1b	0x1b	Reserved	Sequence	Command	Destination port	Length	Data
8 bits	8 bits	8 bits	16 bits	8 bits	8 bits	8 bits	16 bits	Length x 8 bits
Header								

Obrázek 5: Formát zprávy

Header

Každá platná zpráva musí začínat sekvencí 3 bajtů o hodnotě 0x1b šestnáctkové soustavy. Jedná se o klíčovou část zprávy, není-li tato sekvence tří bajtů nalezena na prvních třech pozicích zprávy, je celá zpráva ignorována.

Reserved

Za sekvencí *Header* následuje šestnácti bitové číslo označené jako *Reserved* a v tomto účelu použití

protokolu nemá žádný význam a jeho hodnota může být ignorována.

Sequence

Následujícím osmibitovým číslem je *Sequence*. Číslo *Sequence* je použito pro kontrolu paketů nesoucích data a je v závislosti na pořadí datového paketu inkrementováno. Nulová hodnota je při přenosu dat neplatnou hodnotou.

Command

Další přenášenou hodnotou je číslo *Command* a slouží pro rozlišení druhu zprávy. Toto číslo určuje jaká data jsou přenášena a jak má být s těmito daty nakládáno. Je to jedno z významných čísel posílané zprávy.

Destination port

Mezi další hodnotu, kterou je možné ze zprávy vyčíst je číslo *Destination port*. Pro účel užití tohoto protokolu jeho hodnota nemá platný význam a bude vždy nastavena na nulu.

Length

Posledním řídícím číslem zprávy je šestnáctibitová hodnota *Length* udávající počet bajtů dat, která popřípadě následují za touto dosavadní sekvencí. Tu můžeme označit za datovou hlavičku, ve které jsou všechny potřebné údaje příslušící přenášené zprávě. Hodnota *Length* může být i nulová. Velikost datové přílohy je závislá na druhu posílané zprávy.

Data

Za datovou hlavičkou mohou následovat samotná data. Jejich prezence a případná délka je závislá na druhu konkrétní zprávy. Některé zprávy nenesou žádná data.

6.4.3 Obsluha příchozích zpráv

Obsluha příchozích zpráv je rozdělena na dvě rutiny. Důvodem rozdělení je skutečnost, že se přechází mezi dvěma stavy a to mezi stavem, kdy je otevřen sériový port a má smysl přijímat a zpracovávat data týkající se práce se sériovou linkou a stavem, kdy je sériový port zavřený a má smysl přijímat právě jen zprávy nutné pro jeho nastavení a úspěšné otevření.

Funkce `FindRequest()` kontroluje první tři bajty na výskyt hodnoty 0x1b, jestliže se v datech tato posloupnost na prvních třech pozicích nevyskytuje, funkce vrátí hodnotu -1, v opačném případě vrátí počet datových bajtů v datové příloze. Tato funkce urychluje kód, jelikož při absenci dané sekvence dojde k okamžitému návratu do hlavní programové smyčky a žádná další porovnání

se již neprovádí. Rovněž je vrácená hodnota využita pro kontrolu příjmu všech datových bajtů právě podle počtu udaného v její návratové hodnotě. Může se stát, že data z nějakého důvodu nedorazí všechna a aby nedošlo k nepředvídatelné chybě je takto neúplná zpráva ignorována, což je případně signál pro odesílatele zprávy, že má zprávu opakovat.

Je-li nalezena zpráva, je funkci `alwayes()` předáno číslo zprávy, podle kterého se ve funkci vybere příslušná rutina. Odpovídá-li číslo zprávy rutině obsažené ve funkci `alwayes()`, je po provedení příslušných úkonů vrácena hodnota 1 (TRUE), v případě že zpráva neodpovídá žádné rutině, vrátí funkce hodnotu 0 (FALSE). Tato funkce se provádí vždy, ať je již sériový port otevřený či nikoli. Návratové hodnoty je využito pro urychlení běhu obsluhy portu, jelikož v případě obsloužení dané zprávy dojde k návratu do hlavní programové smyčky a nedochází k dalšímu porovnávání u jednotlivých rutin.

Je-li sériový port otevřený, zpřístupní se i funkce `portOpened()` obsahující obsluhy zpráv určené právě pro otevřený sériový kanál. To však za předpokladu, že nebyla přijatá zpráva již obsloužena funkcí `alwayes()`.

Každá přijatá zpráva, která byla úspěšně provedena, je přeposlána odesílateli zpět i s datovou přílohou, pokud to daný typ zprávy vyžaduje.

Následující výpis je příkladem kódu pro obsluhu příchozích zpráv, v tomto případě se jedná o obsluhu zprávy *setBaudRate*.

```
//6
case 0x42: //B set baudrate + 4 Bytes
{
        baudRate2 = AddBuffer2[0];
        baudRate2 = ((baudRate2 << 8) | AddBuffer2[1]);
        baudRate2 = ((baudRate2 << 8) | AddBuffer2[2]);
        baudRate2 = ((baudRate2 << 8) | AddBuffer2[3]);
        if(portIsOpened2)
                portIsOpened2 = CloseComPort();
        portIsOpened2 = OpenComPort(baudRate2,stopBits2,parity2,wordLength2,fifoTX2,fifoRX2,2);
        BYTE i;
        for(i = 0; i < 10; i++)
                tcpTxData2[i] = HeaderBuffer2[i];
        for(i = 0; i < 4; i++)
                tcpTxData2[i+10] = AddBuffer2[i];
        tcpSend2 = TCPTrySend(tcpTxData2, 14);
        break;
}
```

Zpráva je rozlišená pomocí svého identifikačního čísla, spolu s touto zprávou jsou přenášeny čtyři datové bajty. Tyto čtyři bajty tvoří 32-bitové celé číslo, které uvádí rychlost přenosu sériové linky v baudech. Na začátku rutiny se tedy vytvoří ze čtyř jednotlivých bajtů 32-bitové číslo, které je uložené do globální proměnné, aby bylo dle potřeby kdykoliv přístupné. Zkontroluje se, jestli není sériový port už otevřen. Je-li otevřen, zavře se, aby mohla být nová rychlost nastavena. Po té je UART otevřen na předepsané rychlosti a zpráva je odeslána zpět pro potvrzení úspěšného nastavení nové rychlosti a v datové příloze je odeslána nastavená rychlost.

6.4.3.1 Přijímání TCP paketů

V TCP/IP socketu lze nastavit velikost paketu relativně dle libosti jak pro příjem tak i pro odesílání zvlášť. Jelikož procesor disponuje hardwarovým rozhraním pro práci s ethernetem, jsou přijímané pakety ukládány přímo do operační paměti, a tak je velikost paketu omezena pouze její velikostí. Nesmí být ale opomenuto, že ethernet nepoužívá pouze TCP pakety, ale posílá i jiná data nutná pro režii ethernetového spojení, která rovněž využívají část operační paměti a je nutné mít dostatek volného místa v operační paměti i pro samotné uživatelské aplikace. Proto nesmí být velikost TCP paketu přehnaně veliká, jinak by mohlo dojít k přetečení zásobníku. Výchozí přednastavenou hodnotou od Microchipu pro TCP paket je 20 bajtů. Bez obav ji lze zvětšit alespoň na 60 bajtů. Velikost paketů lze měnit v hlavičkovém souboru *TCPIPConfig.h* pomocí datové struktury `TCPSocketInitializer[]`. Pro snížení datových přenosů v ethernetové síti by samozřejmě byly lepší pakety větší velikosti, jelikož každý TCP paket je potvrzován a čím menší je paket pro daný objem dat, tím více potvrzování se po síti posílá a efektivita přenosu klesá.

Ne vždy je však možné získat z přijímacího zásobníku všech 60 bajtů. Funkce `TCPGetArray()`, která naplní příslušné pole přijatými daty, vrací skutečný počet přijatých dat. Je-li však dat k dispozici méně, je nutné uložit stav, ve kterém se obsluha portu nachází, a vrátit se do hlavní programové smyčky a počkat, než budou v přijímacím zásobníku další data. Není možné čekat například ve `while()` smyčce, až bude k dispozici více dat, došlo by k zablokování veškeré komunikace. Rovněž je žádoucí vyzvednout došlá data, jinak nová data nemusí být v některých případech přijata. (V aplikační poznámce [9] je zmiňováno, že periodické volání funkcí TCP/IP stacku v programové smyčce má za úkol data odesílat z jednotlivých socketů do sítě, o příjmu se již nezmiňuje. Podle zkušeností autora je však nutné i pro příjem dat volat pravidelně funkce TCP/IP stacku v programové smyčce.) Velikost dat v přijímacím bufferu lze zjistit i pomocí funkce `TCPGetIsReady()`. Není tedy nezbytně nutné data z přijímacího bufferu ihned vyjmout, ale podle zkušeností autora je lepší data hned číst. Nová data jsou pak načtena v méně cyklech hlavní

programové smyčky. Pro příjem dat z ethernetu je samozřejmě využíváno přerušení procesoru. Na příkladu následujícího kódu je zvýrazněna funkce pro čtení dat z přijímacího zásobníku. Jedná se o funkci pro vyhledávání platných zpráv `FindRequest()`, která byla zmíněna na začátku.

```
signed int FindRequest(TCP_SOCKET      MySocket, BYTE* AddBuffer, BYTE* HeaderBuffer)
{
          if(TCPGetArray(MySocket, HeaderBuffer, 10) != 10)
                    return -1;
          signed int addLength = TCPGetArray(MySocket, AddBuffer, 1450);
          if(HeaderBuffer[0] == 0x1B && HeaderBuffer[1] == 0x1B && HeaderBuffer[2] == 0x1B) //case 1
                    return addLength;
          else
                    return -1; //request not found
}
```

6.4.4 Nastavení sériového portu

Nastavení parametrů sériové linky jako počet přenášených bitů, počet stop bitů a parita se děje pomocí zpráv *setRegister* nebo *modifyRegister* anebo pomocí zprávy *setWordLengthStopBitsAndParity*.

Zpráva *setWordLengthStopBitsAndParity* obsahuje konkrétní hodnoty pro nastavení délky slova, počet stop bitů a nastavení parity. Zprávy *modifyRegister* a *setRegister* neposílají konkrétní hodnoty pro jednotlivé parametry, ale posílají hodnoty registrů, které slouží pro nastavení parametrů sériové linky. Tyto registry jsou shodné s registry obvodu UART 16550A, které jsou přítomny v osobních počítačích a vytváří asynchronní sériové rozhraní standardu RS232.

Jelikož procesor PIC32 fyzicky nedisponuje takovými registry, jsou registry programově emulovány jako datová proměnná - 8-bitové celé číslo (char). Mezi emulované registry patří MCR (modem control register), LCR (line control register), MSR (modem status register), LSR (line status register). Emulace těchto registrů je vhodná i pro případné budoucí použití externích UART obvodů, které pro svoje nastavování parametrů rovněž používají hodnoty těchto registrů.

```
//UART registers
          BYTE lineStatus2 = 0x00;
          BYTE lineControl2 = 0x00;
          BYTE modemStatus2 = 0xFF;
          BYTE modemControl2 = 0x00;
```

Pro nastavení sériové linky se používají registry LCR a MCR. Registry MSR a LSR reprezentují stav sériové linky. Jelikož se jedná o emulaci registrů, je nutné jejich hodnoty programově, na základě změn signálů, obnovovat a udržovat konzistentní či naopak měnit fyzické nastavení, tzn. stavy signálů právě podle nových hodnot registrů. Na příkladu níže je uvedena funkce, která mění hodnoty registru MCR podle hodnot fyzických signálů. Byl-li registr, změněn vrátí hodnotu 1, jinak vrátí hodnotu 0.

```
//*******************************************************************
//MSR change function
//return 1, if MSR was changed
//return 0, if MSR is without some change
//*******************************************************************
BYTE UpDateMSR()
{
          unsigned char newStatus = !CD2;
          newStatus = (newStatus << 1) | !RI2;
          newStatus = (newStatus << 1) | !DSR2;
          newStatus = (newStatus << 1) | !CTS2;
          newStatus = (newStatus << 4) | 0x00;
          unsigned char newStatusMask = CD2;
          newStatusMask = (newStatusMask << 1) | RI2;
          newStatusMask = (newStatusMask << 1) | DSR2;
          newStatusMask = (newStatusMask << 1) | CTS2;
          newStatusMask = (newStatusMask << 4) | 0x00;

          if(newStatus != modemStatus2)
          {
                    if(hwFlowControl2)
                    {
                              if( (newStatusMask & hwStatusModemMask2) == hwStatusModemMask2)
                                        holdOn2 = 1;
                              else
                                        holdOn2 = 0;
                    }
                    modemStatus2 = newStatus;
                    return 1;
          }
          else
                    return 0;
}
```

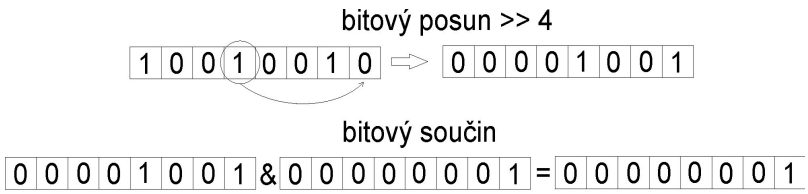
Zpráva *modifyRegister* mění hodnotu registru bit po bitu. Konkrétní bit nastavuje pomocí *SetMask* masky a nuluje ho podle masky *ClearMask*. Příslušně modifikován bude právě ten bit, který je v masce označen jedničkou. Nastavení a mazání toho samého bitu v jednom okamžiku je zakázaná operace. Na následujícím výpisu je funkce měnící zvolený registr bit po bitu.

```
//*******************************************************************
//function for modification of register bit by bit
//*******************************************************************
BYTE ModifyRegisterBits(BYTE clearMask, BYTE setMask, BYTE registerValue)
{
          BYTE bitRegister[8];
          BYTE bitClearMask[8];
          BYTE bitSetMask[8];
          BYTE i;
          for(i = 0; i < 8; i++)
          {
                    bitRegister[7-i] = ((registerValue >> (7-i)) & 0x01);
                    bitClearMask[7-i] = ((clearMask >> (7-i)& 0x01);
                    bitSetMask[7-i] = ((setMask << (7-i) &0x01);
                    if(bitSetMask[7-i])
                              bitRegister[7-i] = 1;
                    if(bitClearMask[7-i])
                              bitRegister[7-i] = 0;
          }
          registerValue = 0;
          for(i = 0; i < 8; i++)
                    registerValue = (registerValue << 1) | bitRegister[7-i];
          return registerValue;
}
```

Jednotlivé bity přijatých modifikačních masek jsou nejdříve nakopírované do pole, kde každá jedna položka pole daného indexu odpovídá hodnotě jednoho bitu na pozici indexu. Převod bitů do pole se děje pomocí bitového posunu, kdy je příslušný bit posunut na první pozici zcela doprava a všechny horní bity jsou vymaskovány pomocí bitového násobení. Princip je ukázán na obrázku 6.

Získají se tak tři nové proměnné typu pole. Původní hodnota registru, která je teď reprezentována polem, může být jednoduše modifikována pomocí porovnávání. Je-li příslušná

hodnota v dané masce jedna, provede se modifikace.



Obrázek 6: Princip převodu bitů do pole

6.4.5 RX, TX FIFO zásobníky

Data přijatá ze sériového portu jsou vkládána do zásobníku, který je implementován jako FIFO fronta. Jedná se tedy o pole bajtů, kdy vkládaná položka se přidává na konec fronty a odebíraná položka se nachází vždy v čele fronty. Pro implementaci FIFO fronty byl použit hlavičkový soubor *struct_queue.h*. Jelikož je implementace FIFO fronty již dávno vyřešená, nemá smysl ji vymýšlet znovu.

Jakmile jsou přijatá data zapsaná do FIFO fronty, následuje menší časové zpoždění, aby se fronta naplnila více daty, a data z fronty začnou být odesílána přes ethernet k připojenému klientovi.

Velikost fronty je nastavena na 1450 bajtů. Aby nebyla data odesílána okamžitě, jakmile jsou ve frontě nějaká přítomna, čeká se, až bude ve frontě alespoň 1000 bajtů anebo po přijetí prvního bajtu uběhla daná doba. Vhodná doba čekání byla experimentálně zvolena a je funkcí přenosové rychlosti sériové linky (1). Kdyby byla data okamžitě odesílána bez jakéhokoliv zpoždění, odesílaly by se velmi malé pakety a rostl by zbytečně provoz v ethernetové síti.

```
TimeDelay = 1000/(baudRate2/11)) * 5 * 40000;        (1)
```

Data, která mají být odeslána přes sériovou linku, jsou rovněž zachytávána do FIFO fronty. V tomto případě už bez žádného zpožďování, jelikož sériový port je mnohem pomalejší než ethernet. Data do fronty jsou přijímána tak dlouho, než je fronta plná. Další data jsou přijata, až se ve frontě vytvoří volné místo. Toto však platí pouze, když je zapnutý FIFO buffer a řízení toku je vypnuté. Jelikož klient vyžaduje potvrzení, že data, která poslal, byla skutečně odeslána přes sériový kanál, je v ostatních případech nutné čekat, až jsou všechna přijatá data odeslána přes

UART. Tento způsob, kdy není využita celá kapacita FIFO fronty, je pomalejší, ale je vyžadován protokolem. Možnost využití celé FIFO kapacity byla implementována a je možné ji programově zapínat. Její efekt se projeví ale pouze jen při neřízeném toku.

Při příjmu zpráv *recieverFlush* a *transmitterFlush* je příslušná FIFO fronta vymazána.

Práce s FIFO frontou není atomická, vyžaduje více operací. Nikdy se nesmí stát, že se budou odebírat data z prázdné fronty, došlo by okamžitě k pádu aplikace. Tato skutečnost musí být při odebírání dat vždy brána v úvahu a než budou data odebírána musí být fronta testována, není-li prázdná, viz následující příklad.

```
if( StructQueueIsNotEmpty(&fifoRX2,SIZE_OF_FIFO_RX) )
{
        p_item = StructQueueRemove(&fifoRX2,SIZE_OF_FIFO_RX);
        dataBuffer[i] = p_item->item;
}
```

Neatomičnost práce s frontou je nutné brát v úvahu, jestliže v kódu z fronty odebíráme data a data do fronty zapisujeme s využitím přerušení. Kdyby došlo k situaci, kdy odebíráme data, snížíme index ve frontě, ale data už nestihneme fyzicky odebrat, a v tom okamžiku se zavolá přerušení od sériového portu a dojde k přidání nové položky, bude FIFO fronta nekonzistentní a dojde ke ztrátě dat. Jelikož je pro příjem dat ze sériové linky využito přerušení UART kanálu, je nutné při práci s frontou brát tuto skutečnost v úvahu. Řešením je před zahájením odebírání dat z fronty vypnout přerušení od UARTu a po dokončení práce s ním přerušení opět zapnout.

6.4.6 Příjem datových zpráv

Datové zprávy se liší od ostatních tím, že jejich datová příloha může mít až 1450 bajtů, tedy je mnohem větší, než u ostatních zpráv. To zpravidla znamená, že není možné přijmout všechna data najednou. Místo toho jsou data posílána po více částech. Pro zabezpečení příjmu datové zprávy je nutné přepínat mezi dvěma režimy příjmu, a to mezi režimem příjmu klasických zpráv a režimem datového přenosu. Je-li přijata hlavička datové zprávy, uloží se hodnota reprezentující počet bajtů datové přílohy. Jsou-li zároveň přijatá nějaká data, počet přijatých bajtů dat se uloží do datové proměnné reprezentující počet došlých dat. Pokud je celkový počet dat větší, uloží se daný stav a při opětovném volání obsluhy TCP serveru nejsou data prohledávána na přítomnost zprávy, ale přišlá data se hned považují za součást předchozí datové přílohy a k datové proměnné reprezentující došlá data se přičte počet právě přijatých bajtů. Tento cyklus se opakuje, až jsou přijaté všechny bajty datové přílohy. Po přijetí všech bajtů se může klientovi teprve odeslat potvrzení, že všechna data byla úspěšně doručena a TCP server se opět přepne do režimu přijímání klasických zpráv.

Mechanismus příjmu více datových bajtů je na následujícím výpisu.

```
if(dataTransfer2)
{
        if(length2 == currentBytes2)
        {
                dataTransfer2 = 0; //no more data transfer
                isAknowledge2 = 1; //all data is in the TX FIFO, host wait for aknowledge
        }
        else
        {
                addLength2 = TCPGetArray(MySocket2, AddBuffer2, 1450);
                WORD i;
                FIFO_ITEM *p_item;
                for(i = 0; i < addLength2; i++)
                {
                        //add one item
                        if(StructQueueIsNotFull(&fifoTX2, SIZE_OF_FIFO_TX))
                        {
                                p_item = StructQueueAdd(&fifoTX2, SIZE_OF_FIFO_TX);
                                p_item->item = AddBuffer2[i];
                        }
                }
                currentBytes2 += addLength2;
                break;
        }
}
```

6.4.6.1 Heartbeat

Klient a server si mezi sebou vyměňují zvláštní druh zprávy, který je označený jako *HeartBeat*. Při inicializaci komunikace pošle server klientovi časovou prodlevu mezi jednotlivými zprávami *Heartbeat*. Tuto prodlevu musí klient dodržovat. Pokud ji nedodrží, server zavře sériový port a ukončí spojení. Na zprávu *Heartbeat* server klientovi obratem odpoví rovněž zprávou *Heartbeat*. Jelikož se však může stát, že inkriminovaná doba, kdy server očekává další zprávu *Heartbeat*, bude zrovna během přenosu datové zprávy, kdy server nepřijímá klasické zprávy, je interval na straně klienta zvětšen 1,5 krát. To je dostatečný čas, aby dorazil *Heartbeat* od klienta a server neukončil komunikaci neoprávněně. Interval pro zprávu *Heartbeat* je uložen v datové proměnné *alive.* Pokud je tato proměnná nastavena na nulu, je systém *Heartbeat* zcela vypnut.

6.4.7 Odesílání datových zpráv

Při odesílání datových zpráv směrem ke klientovi se nejdříve kontroluje pomocí volání funkce *TCPIsPutReady()* počet bajtů, které je možné v dané chvíli odeslat. Pokud je možné odeslat více jak 10 bajtů, to odpovídá 10 bajtům datové hlavičky a alespoň jednoho bajtu datové přílohy, jsou data odeslána. Není-li možné poslat tedy alespoň 11 bajtů, není přenos uskutečněn a další pokus o přenos dat bude až při opětovném průchodu programové smyčky. Při odesílání zpráv je délka datové přílohy rovna počtu bajtů, které je možné v jednom okamžiku odeslat, mínus 10 bajtů hlavičky zprávy. Při tomto procesu se odebírají data z FIFO fronty a z důvodu neatomičnosti, které byly uvedeny v předchozí kapitole, je nutné vypínat přerušení UARTu, jak je patrné na následujícím výpisu. Při každém odeslání datové zprávy je inkrementováno číslo zprávy *sequence number.* Toto číslo je pak využíváno při potvrzení (*ACK*) doručení zprávy klientovi. Kdyby data nebyla

potvrzena, byl by pokus o jejich odeslání opakován. Protokol umožňuje posílaní více datových zpráv ve sledu za sebou. Každá datová zpráva má sekvenční číslo o jedničku větší než ta předchozí. V tom případě se potvrzuje jenom ta poslední zpráva. Tato možnost však není v zařízení využita.

```
//*******************************************************************
//send recieved data through ethernet to host
//*******************************************************************
void SendToHost()
{
          unsigned short wMaxPut = TCPIsPutReady(MySocket2);
          DisableIntU2RX; //disable ISR
          if(wMaxPut > 10 && StructQueueIsNotEmpty(&fifoRX2,SIZE_OF_FIFO_RX)/* && sequenceRecieve < hostPacketCount*/)  //if we can
not send anything, jump into main loop
          {
                    if(StructQueueCount(&fifoRX2,SIZE_OF_FIFO_RX) < (wMaxPut - 10) )
                              lengthRecieve2 = StructQueueCount(&fifoRX2,SIZE_OF_FIFO_RX);
                    else
                              lengthRecieve2 = (wMaxPut - 10); // do not forget bytes for head
                    unsigned short i;
                    FIFO_ITEM *p_item;
                    BYTE *dataBuffer = (BYTE *) malloc(lengthRecieve2 + 10);
                    for(i = 10; i < lengthRecieve2 + 10; i++)
                              if( StructQueueIsNotEmpty(&fifoRX2,SIZE_OF_FIFO_RX) )
                              {
                                        p_item = StructQueueRemove(&fifoRX2,SIZE_OF_FIFO_RX);
                                        dataBuffer[i] = p_item->item;
                              }
                              else
                                        break;
                    EnableIntU2RX; //enable ISR

                    sequenceRecieve2++;
                    BYTE Head[10] = {0x1B,0x1B,0x1B,0x00,0x00,0x00,0x44,0x00,0x00,0x00};
                    Head[4] = (sequenceRecieve2 >> 8);
                    Head[5] = ((sequenceRecieve2 << 8) >> 8);//put the sequence number of the packet
                    Head[8] = (lengthRecieve2 >> 8);
                    Head[9] = ((lengthRecieve2 << 8) >> 8);//put the length number of the current packet
                    for(i = 0; i < 10; i++)
                              dataBuffer[i] = Head[i];
                    TCPPutArray(MySocket2, dataBuffer, lengthRecieve2 + 10);
                    TCPFlush(MySocket2);
                    free(dataBuffer);
          }
          else
          {
                    EnableIntU2RX; //enable ISR
                    return;
          }
}
```

6.4.8 Obsluha přerušení UART portu

Příchozí data ze sériového portu jsou obsluhována pomocí přerušení UARTu. Přerušení lze nastavit na několik úrovní naplnění vstupního zásobníku obvodu UART. Možnosti jsou následující: neprázdný zásobník, zásobník naplněný z ½, zásobník naplněný z ¾ a zásobník plný. Pro potřeby tohoto zařízení je přerušení nastavené na volbu neprázdný zásobník, to znamená, jakmile je v zásobníku alespoň jeden přijatý bajt, je ihned ze zásobníku vyzvednut. V obslužné rutině volané při přerušení se provádí i operace spojené s řízením toku. Je-li nastavené řízení toku, ať již softwarové či hardwarové, kontroluje se zaplnění FIFO fronty. V případě, že ve frontě již není místo, je poslán signál, ať již HW nebo SW, pro pozastavení komunikace. Pro případ, že by nedošlo k okamžitému zastavení komunikace, jsou signály pro pozastavení nastaveny dříve než je fronta zcela plná, aby se zabezpečilo, že další data do pozastavení komunikace budou spolehlivě přijata.

Procesor PIC32 umožňuje multivektorové přerušení, které je v tomto případě zapnuté. Obsluha přerušení je volána nejen v případě příjmu nového bajtu, ale i při chybě parity, přetečení

zásobníku či při chybě rámce. Vektor konkrétního přerušení se testuje až ve funkci obsluhující přerušení UARTu. Všechny chyby jsou zpracovávány a odesílány klientovi jako změna registrů sériové linky. Díky tomu má klient úplný přehled o stavu sériové linky.

6.4.8.1 Informování o stavu sériové linky

Změna status-registrů sériové linky je poskytována klientovi pomocí vkládání hodnot příslušných registrů do přenášeného proudu dat. Toto vkládání zapíná i vypíná klient pomocí zprávy pro to určené. Struktura vkládané zprávy s novou hodnotou registru je zobrazena na obrázku 8. Klient musí sledovat datový tok a při nalezení této zprávy ji musí z proudu dat vyjmout. Aby nedošlo k záměně běžných dat za status registrů, musí být data přicházející ze sériového portu prohledávána, neobsahují-li poznávací sekvenci status zprávy. Je-li v toku dat detekována poznávací sekvence, dva za sebou následující bajty 0x1b, musí být za tuto sekvenci vložen bajt s hodnotou 0xFF, který znamená, že se jedná o běžná data a ne o status zprávu. Prohledávání příchozích dat ze sériového portu je prováděno v obsluze přerušení UARTu. Mezi registry nesoucí informaci o stavu sériové linky patří MSR (modem status register) a LSR (line status register).

LSR obsahuje informace o chybě parity, chybě rámce a přetečení zásobníku.

MSR obsahuje informace o signálech RI (ring indicator), DSR (data set ready), CTS (clear to send)a CD (carrier detect).

0x1b	0x1b	Typ registru	Hodnota registru

Obrázek 7: Struktura vkládané zprávy

6.4.9 Výpočet parity a délky slova

Hardwarový UART přítomný na čipu procesoru PIC32 disponuje pouze 8-bitovou a 9-bitovou délkou slova a umožňuje nastavit lichou paritu, sudou paritu a žádnou paritu. Při použití délky slova 9 bitů je možné nastavit paritu na *mark* (paritní bit je vždy jedna) a *spacer* (paritní bit je vždy nula). V požadavcích na zařízení je délka slova 7 bitů s paritou. To je možné docílit tak, že se bude používat 8-bitová délka slova, přičemž 8. bit bude nastavován podle zvolené parity. Paritní bit při nastavené liché paritě je nastavován na jedna právě tehdy, když je počet jedniček včetně paritního

bitu lichý. Při nastavené sudé paritě je paritní bit jednička právě tehdy, když je počet jedniček včetně paritního bitu sudý. Výpočet parity probíhá podle následujícího kódu.

```
//*********************************************************************
//count parity if odd or even
//*********************************************************************
BYTE CountParity(BYTE data)
{
          BYTE i;
          BYTE mask = 0x01;
          BYTE info = 0x00;
          for(i=0; i < 8; i++)
          {
                    if(data & mask)
                              info++;
                    mask = (mask << 1);
          }
          info++;
          return (info & 0x01); //return 0 if odd, 1 if even parity
}
```

Ve for-cyklu se počítá počet jedniček, v každém cyklu jsou všechny horní bity kromě prvního bitu maskovány a před dalším cyklem jsou posunuty doprava. Tímto jednoduchým cyklováním a maskováním je možné spočítat počet jedniček v datech. Funkce vrátí nulu, když je počet jedniček sudý a jedničku, když je počet jedniček lichý. Samotná kontrola dat na správnou paritu probíhá způsobem ukázaným v následujícím výpisu. Nejdříve je zjištěn počet jedniček v příchozích datech, tím se určí parita, a pak je porovnán s hodnotou paritního bitu. Neshodují-li se tyto skutečnosti, je oznámena chyba parity a je poslána klientovi.

```
switch(parity)
{
          case 1: //control even parity
                    if( !(CountParity(data & 0x7F) == ((data & 0x80) == 0x80)) )
                              lineStatus1 |= 0x04; //found parity error
                    break;
          case 2: //control odd parity
                    if( !(CountParity(data & 0x7F) == ((data & 0x80) == 0x80)) )
                              lineStatus1 |= 0x04;
                    break;
          case 3: //control mark parity
                    if(!(data & 0x80))
                              lineStatus1 |= 0x04;
                    break;
          case 4: //control space parity
                    if((data & 0x80))
                              lineStatus1 |= 0x04;
                    break;
          default:
                    break;
}
```

6.5 Paměť pro uložení nastavení

Pro uložení parametrů síťového nastavení a dalších údajů, jako hesla apod., slouží externí paměť. TCP/IP stack podporuje přímo paměti jako EEPROM nebo FLASH připojené k rozhraní ISP. V tomto projektu se používá mezi procesorem a pamětí komunikační rozhraní I2C. Důvodem použití právě rozhraní I2C byl nedostatečný počet volných kanálů ISP, jelikož jsou sdílené s rozhraním UART a s rozhraním RMII, přes které je připojené ethernetové rozhraní. Aplikační rozhraní pro komunikaci s pamětí připojené přes rozhraní I2C muselo být dodatečně implementováno.

6.5.1 I2C protokol

I2C je rozhraní založené na jednoduchém protokolu master-slave. Umožňuje, aby byl kanál sdílen mezi více periferiemi, pomocí adresování. Periférie rozpoznávají 7-bitovou adresu a 8.bit v adrese značí požadovanou operaci, kterou je čtení nebo zápis. Master, který zahajuje operaci, vyšle *start* podmínku, která značí začátek komunikace, po té následuje bajt se 7-bitovou adresou a bitem označující danou akci. Každý vyslaný bajt potvrzuje slave jedním bitem označovaným jako *ack*, který značí úspěšné přijetí předchozího bajtu. Konec komunikace značí master podmínkou *stop*. Obšírnějším popisem linky I2C se zabývá například literatura [22].

6.5.2 Sériová paměť EEPROM 24CXX

Pro uchování dat byla použita sériová paměť typu EEPROM 24C64. Jedná se o paměť, která je schopná uchovat zapsaná data i bez přítomnosti napájecího napětí po dobu větší než 200 let, což je pro tuto aplikaci uspokojivá hodnota, výrobce Atmel [6] garantuje až 10 000 000 cyklů čtení a zápisů, i tato hodnota je dostačující. Paměť je organizována po 8-bajtových stránkách, což umožňuje v jednom cyklu zapsat až 8 bajtů rychle za sebou. Adresa paměti se nastavuje podle připojení vývodů A0, A1, A2 buď ke kladnému napětí anebo k zápornému napětí (k zemi GND). Pokud je adresový vývod připojen k zápornému napětí (k zemi GND), je daný bit nastavený na 0, pokud ke kladnému napětí, je příslušný bit nastaven na logickou jedničku. Adresa paměti vypadá potom následovně, obrázek 8.

1	0	1	0	A2	A1	A0	$R/\overline{W}$

Obrázek 8: Adresa EEPROM podle propojení adresních vývodů

Po aktivaci paměti pomocí správné adresy jsou odeslány dva bajty nesoucí 12-bitovou adresu, na kterou se mají daná data zapsat nebo číst, v případě čtení. Pokud se jedná o zápis, následuje hned za datovou adresou bajt, který má být zapsán, po tomto bajtu následuje podmínka *stop*. Každý odeslaný bajt, ať je to datová adresa, adresa cílového zařízení nebo zapisovaná či čtená data, je potvrzen jedním *ack* bitem. Pokud je *ack* bit nastaven na logickou 1, k potvrzení nedošlo a operace se musí provést znovu. Je-li bit *ack* nastaven na logickou 0, je přijatý bajt potvrzen a může se pokračovat v komunikaci.

Při čtení je nutné po odeslání adresy, od které se má číst, komunikaci restartovat a adresovat paměť znovu, tentokrát však v režimu čtení. Teprve po té jsou na datovém vodiči k dispozici čtená data.

Paměti 24C umožňují sekvenční čtení a zápis jedné paměťové stránky v jednom cyklu.

Sekvenční čtení znamená, že po prvním přečteném bajtu není vyslána podmínka *stop*, ale dále se pokračuje ve čtení. V paměti je po každém přečteném bajtu zvětšena datová adresa o jedničku a při dalším čtení je na datový vodič vystaven další bajt, který je v paměti organizován hned za předchozím přečteném bajtu. Tento typ čtení celý proces urychluje, jelikož není třeba adresovat zařízení po každém přečteném jednom bajtu.

Je možné zapsat i více jak jeden bajt v jednom cyklu, potom se jedná o stránkový zápis, kdy je možné zapsat najednou až 8 bajtů, které jsou v paměti nejdříve nahrané do zásobníku a průběžně pak ukládané. Je možné zapisovat až na 8 stránek hned za sebou, aniž by se muselo čekat na dokončení operace zápisu. Dohromady je tedy možné rychle za sebou zapsat až 64 bajtů, 8 stránek po 8 bajtech.

6.5.3 PIC32 a rozhraní I2C

Rozhraní I2C je řešeno jako samostatná periférie na čipu procesoru PIC32 a jako taková je řízena parametry příslušného registru jako všechny ostatní periférie v procesoru. PIC32 komunikuje po lince I2C frekvencí až 1MHz, standardní zařízení připojená k rozhraní I2C komunikují většinou rychlostí 100KHz nebo 400KHz. Rychlost komunikace se volí pomocí hodnoty v příslušném registru rozhraní I2C. Důležitou hodnotou rozhraní I2C je tedy jeho přenosová rychlost. Ta se počítá podle následujícího vzorce (2).

$$BRGVAL = \frac{PBCLK}{(2 \cdot FSCK)} - 2 \qquad (2)$$

PBCLK je hodnota frekvence periferní sběrnice procesoru. V tomto případě platí, že je polovinou systémové frekvence procesoru. To je vždy závislé podle nastavení děliček frekvence v procesoru, dle požadavků programátora. FSCK představuje výslednou frekvenci hodinového kmitočtu v Hz linky I2C.

6.5.4 Aplikační rozhraní pro zápis a čtení

Veškeré funkce, které jsou nutné k ovládání rozhraní I2C, jsou k dispozici přes hlavičkový soubor

plib.h. Funkce potřebné pro řízení I2C komunikace jsou v tabulce 6.

`OpenI2C2(I2C_EN,BRG_VAL)`	Inicializace I2C kanálu s požadovanou rychlostí
`StartI2C2()`	Pošle START podmínku
`MasterWriteI2C2(Byte)`	Zapíše 1 bajt
`MasterReadI2C2()`	Přečte 1 bajt
`RestartI2C2()`	Pošle STOP a poté hned START podmínku
`I2C2STATbits.ACKSTAT`	*ack* bit, potvrzení odeslaného bajtu
`IdleI2C2()`	Čekání na *ack* bit
`StopI2C2()`	Pošle STOP podmínku

Tabulka 6: Funkce pro řízení rozhraní I2C

Celý princip komunikace lze vysledovat na následující ukázce kódu. Kód je snadno čitelný a posloupnost příkazů, jak bylo popsáno v předcházejících oddílech, lze dobře sledovat. Tímto jednoduchým způsobem lze komunikovat s jakoukoliv periférií připojenou na kanál I2C. Při větších objemech přenášených dat by bylo možné použít i kanál DMA pro menší zatížení procesoru.

Při sekvenčním čtení není poslána hned za prvním přijatým bajtem *stop* podmínka, nýbrž se pokračuje dále ve čtení až po splnění počtu požadovaných bajtů. Pokud se čtení přeruší tím, že paměť poslední přečtený bajt nepotvrdí, cyklus se ukončí a funkce vrátí počet úspěšně přečtených bajtů.

```
UINT32 ReadArrayI2C(UINT16 address, BYTE* data, UINT32 length)
{
                  ...
       for(index=0; (index < length); index++)
       {
         *data = MasterReadI2C2(); //read data
          data++;
          IdleI2C2();//Wait to complete
          if( I2C2STATbits.ACKSTAT )// 0 when acknowledged
              break;
      }
                ...
      return index; //return the count of successful read data
}
```

Funkce pro zápis i čtení polí mají jako návratovou hodnotu počet úspěšně přečtených nebo zapsaných dat. Pokud nastane jakákoliv chyba a nepodaří se zpracovat ani jeden bajt, funkce vrátí nulu. Funkce pro čtení jednoho bajtu se liší, její návratovou hodnotou je přečtený bajt a v případě neúspěchu -1, funkce vrací znaménkový integer.

```
INT32 ReadByteI2C(UINT16 address)
{
	StartI2C2();//Send the Start Bit
	IdleI2C2();//Wait to complete
	//send the address to read from the serial eeprom
	MasterWriteI2C2( ((SlaveAddress << 1) | 0) ); //write-command
	IdleI2C2();//Wait to complete
	//ACKSTAT is 0 when slave acknowledge. if 1 then slave has not acknowledge the
data.
	if( I2C2STATbits.ACKSTAT )
		return -1; //return -1 if fail
	MasterWriteI2C2(address >> 8); //high  address byte
	IdleI2C2();//Wait to complete
	//ACKSTAT is 0 when slave acknowledge. if 1 then slave has not acknowledge the
data.
	if( I2C2STATbits.ACKSTAT )
		return -1; //return -1 if fail
	MasterWriteI2C2((BYTE) address); //low address byte
	IdleI2C2();//Wait to complete
	//ACKSTAT is 0 when slave acknowledge. if 1 then slave has not acknowledge the
data.
	if( I2C2STATbits.ACKSTAT )
		return -1; //return -1 if fail

	//now send a start sequence again
	RestartI2C2();//Send the Stop condition
	IdleI2C2();//Wait to complete
	MasterWriteI2C2( (SlaveAddress << 1) | 1 ); //transmit read command
	IdleI2C2();//Wait to complete

	BYTE data = MasterReadI2C2(); //read data
	StopI2C2();//Send the Stop condition
	IdleI2C2();//Wait to complete
	return data;//return the read data
}
```

6.6 Webové rozhraní

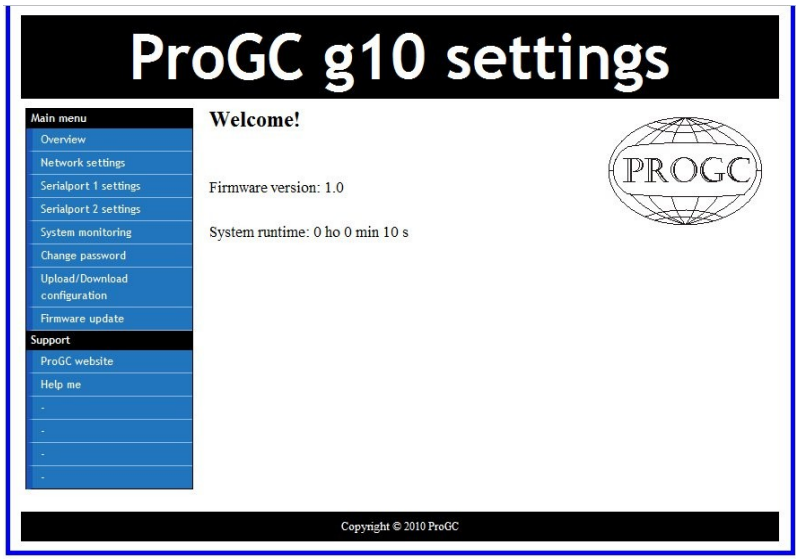
Pro nastavení síťových parametrů, například změny hesla, nahrání nového firmwaru či nové konfigurace, slouží webové rozhraní, které je přístupné přes běžný internetový prohlížeč (například Firefox, Internet Explorer, Opera, Safari a další). Do prohlížeče je nutné zadat IP adresu zařízení, přičemž musí být zajištěno, aby se osobní počítač i zařízení nacházely ve stejné síti. Po zadání IP adresy se zobrazí uvítací stránka s odkazy pro konkrétní nastavení.

Tvorbu webového rozhraní lze rozdělit do více kroků, mezi které patří návrh designu, jeho kódová realizace pomocí html, kaskádových stylů a popřípadě užití i javascriptu a v neposlední řadě implementace webového rozhraní v kódu procesoru PIC32.

6.6.1 Návrh designu webového rozhraní

Webové rozhraní je výkladní skříní produktu, musí být jednoduché, intuitivní, přehledné. Mezi atribut, který nelze opomenout, jistě patří i „líbivé“. Četnost používání webového rozhraní je

závislá na druhu aplikace, ve které je zařízení nasazené. Webové rozhraní bude hrát velkou roli především v budoucnu, kdy bude zařízení implementovat další pokročilé funkce, kterou budou webové rozhraní pro svůj výstup přímo vyžadovat a obsluha bude s nimi v kontaktu neustále. Rovněž tento návrh může být jednotnou šablonou pro návrh webového rozhraní v jiných projektech, které by vyžadovaly tímto rozhraním disponovat. Výsledný design je zobrazen na obrázku 9.



Obrázek 9: Výsledný design webového rozhraní

6.6.1.1 Členění webové stránky

Stránka je rozdělena na tři sloupce. V prvním sloupci zleva je umístěna nabídka s jednotlivými odkazy. Prostřední sloupec je zvolen jako pracovní. V tomto sloupci se zobrazují formuláře pro vstupní data, stejně jako informace pro obsluhu. Třetí sloupec zleva je zvolen jako dekorativní a je v něm umístěn obrázek loga projektu. Logo je vlastním dílem autora. ProGC je zkratkou „PROject Globus Cimco“. V záhlaví stránky se nachází nadpis, který je zobrazen na všech stránkách, stejně jako zápatí stránky, které obsahuje „copyright“ a rok vzniku stránky.

6.6.1.2 Obsah webového rozhraní

Webová stránka obsahuje odkazy pro nastavování parametrů či sledování stavu zařízení. Prvním odkazem „Overview“ je samotná úvodní stránka, na které se zobrazuje aktuální verze firmwaru a celková doba činnosti zařízení od jeho zapnutí.

Dalším odkazem „Network settings“ se zobrazí nejdříve výzva pro zadání hesla administrátora a poté se zobrazí stránka s formulářem pro zadání síťových parametrů (viz. Příloha C). Po zadání všech parametrů dojde k restartu zařízení na požadovaných parametrech.

Třetím odkazem je stránka s formulářem pro nastavení parametrů sérové linky. Momentálně tato možnost není využita. Stránka najde své uplatnění v budoucnu, kdy bude možné propojovat dvě tato zařízení přes ethernet, která budou pracovat jako virtuální sériový kabel přes síť ethernet (viz. Příloha C).

Odkaz „System monitoring“ nemá vytvořenou žádnou svoji stránku. Jeho užití autor zamýšlí v budoucnu (viz. Příloha C)..

Dalším odkazem je „Change password“. Na této stránce je možné změnit heslo administrátora. Do formuláře se zadává předchozí heslo a heslo nové, které je třeba zadat dvakrát stejně pro kontrolu překlepů (viz. Příloha C)..

Na stránce „Upload/Download configuration“ bude možné nahrát soubor s úplnou konfigurací zařízení, či stáhnout aktuální konfiguraci zařízení do osobního počítače. Tato možnost není zatím implmentována, jedná se o budoucí záměr autora (viz. Příloha C).

Posledním odkazem v rámci webového zařízení je stránka „Firmware update“ pro nahrávání nového firmwaru do zařízení. Pro svoji složitost není zatím tato funkce aktivní a v době psaní této práce na ní autor stále pracuje, stejně jak na ostatních neimplementovaných funkcích, které jsou netriviálními úkoly vyžadující značnou časovou dotaci (viz. Příloha C).

Poslední odkazy slouží pro přímý přechod na stránky podpory výrobku a na stránky samotného výrobce. Tyto odkazy jsou zatím neplatné, správnými údaji budou doplněny v momentě, kdy tyto stránky budou existovat.

Celkem je použito 8 webových stránek. Každý odkaz má svoji webovou stránku, stránka *config.html* používá po nastavení parametrů síťové komunikace ještě jednu webovou stránku *reboot.html*, která informuje o probíhajícím restartu.

6.6.2 Program pro vývoj webových stránek

Pro psaní HTML kódu webových stránek byl využit nástroj PSPad. Jedná se o textový editor, který je možné používat bezplatně. Editor umí zvýraznit HTML syntaxe a rovněž má zabudovaný i webový prohlížeč, pomocí kterého je možné ihned zhlédnout aktuální design psané stránky.

6.6.3 Kaskádové styly

Kaskádové styly dnes patří mezi moderní způsoby formátování obsahu webové stránky. Formátování je v omezené míře možné i pomocí HTML tagů, ale některé prvky formátování se již bez kaskádového stylu neobejdou. Kaskádové styly umožňují mimo jiné relativně jednoduchým způsobem měnit design na všech dotčených stránkách stejně a najednou. Kaskádové styly lze na sebe vrstvit a vždy platí ta poslední. Je to příjemná vlastnost, které lze dobře využít. Další informace o možnostech kaskádových stylů je možné najít v literatuře [3] nebo [7].

V projektu je využito externího souboru, který obsahuje formátovací hodnoty. Kaskádové styly používají následující syntaxi [3]:

```
selektor

{

      vlastnost1: hodnota;

      vlastnost2: hodnota

}
```

V HTML kódu se pak dodržuje tento zápis:

```
<span id="selektor"> ovlivněný text</p>
```

Detailnímu popisu kaskádových stylů se autor nebude věnovat, jelikož to není náplní tohoto projektu. Kaskádové styly byly pouze jen prostředkem pro dosažení daného úkolu.

6.6.4 Implementace webového rozhraní

Dalším krokem nutným po dokončení designu webového rozhraní je jeho implementace v kódu procesoru PIC32. Tato implementace podléhá pravidlům, která je nutné například pomocí prezentací od Microchipu [12], [13] a [14] nastudovat a některé detaily se naučit z ukázkových příkladů. Tento vývoj implementace je zdlouhavá záležitost, jelikož každá změna vyžaduje překompilování zdrojových kódů, jejich naprogramování do procesoru a poté vyčkání než, dojde k ethernetovému spojení mezi zařízením a osobním počítačem, aby mohla být funkčnost webového rozhraní ověřena přes webový prohlížeč.

6.6.4.1 Souborový systém

Jelikož webové stránky jsou ve své podstatě obyčejné textové soubory a odkazy v nabídce odkazují opět na soubory, musí být tyto soubory v paměti organizovaně uspořádány. Nejlepší formou je implementovat souborový systém. Takový souborový systém umožňuje minimálně výběr konkrétního souboru a následně jeho čtení. Mezi pokročilejší funkce souborového systému patří kopírování, mazání a zápis nových souborů apod. Nejedná se zde o souborový systém, jaký se používá na osobních počítačích, tento souborový systém má sloužit výhradně jen pro realizaci webového serveru. Samotný souborový systém se zpravidla vytváří při kompilaci, kdy je reprezentován polem bajtů, s jehož strukturou dokáže pracovat implementované aplikační rozhraní souborového systému.

Microchip pro usnadnění práce programátorům nabízí možnost využít svého souborového systému *„Microchip File systém (MPFS)“*. Jedná se o jednoduchý souborový systém, který umí data ze zkompilovaného souborového systému pouze číst. Je navržený právě pro aplikaci webového serveru, popřípadě FTP serveru. Tento souborový systém nabízí názvům souborů osm znaků plus tři znaky jako přípona. Nerozlišuje mezi malými a velkými písmeny. Jeho začlenění do systému je snadné a nevyžaduje znalost způsobu ukládání dat, vše se děje vůči programátorovi transparentně.

Před výslednou kompilací musí být složka se soubory konvertována buď do „C“ souboru nebo do binárního tvaru „bin“. To je závislé na místě, kde bude souborový systém uložen. Microchip umožňuje umístit tento souborový systém buď do programové paměti procesoru nebo do externí paměti. Jako externí paměť může být použita flash paměť nebo paměť typu EEPROM. Aktuální verze TCP/IP stacku podporuje externí paměti pouze na rozhraní ISP. V aplikačních poznámkách Microchipu lze dohledat i podporu pro paměti připojené pomocí rozhraní I2C. Toto rozhraní se však pro ukládání souborového systému nehodí, jelikož rychlost rozhraní I2C je pro tento druh použití nedostatečná. Při použití externí paměti je možné na začátku souborového systému vyhradit určitý počet bajtů pro uložení konfiguračních dat aplikace. Pro konvertování souborového systému do použitelné podoby slouží program *MPFS.EXE*. Jeho parametry jsou uvedeny v tabulce 7.

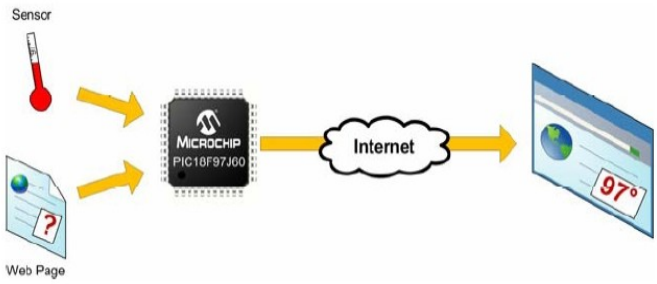
```
mpfs.exe [/?] [/c] [/b] [/r<Block>] <InputDir> <OutputFile>

Vysvětlivky:
/? : zobrazí nápovědu
/c : vygeneruje souborový systém jako „C“ soubor (uložení v programové paměti procesoru)
/b : vygeneruje binární soubor (uložení v externí paměti – flash, EEPROM)
/r : rezervuje určitý počet bajtů na začátku souborového systému v paměti
    (pouze v /b módu  výchozí hodnota = 32)
<InputDir>: adresář, který obsahuje vstupní soubory
<OutputFile>: jméno výstupního souboru
```

Tabulka 7: Parametry funkce MPFS.EXE (převzato z [24])

Pro snazší použití tohoto programu se využívá dávkového souboru „mpfs.bat“, který spouští konvertování se všemi potřebnými parametry.

6.6.4.2 Aplikační rozhraní pro prezentaci webových stránek

Od webových stránek jsou očekávány tři základní akce. První akcí je, že webová stránka zobrazí aktuální údaje, které se budou obnovovat při každém jejím novém načtení, tedy pokaždé, když si je prohlížeč opětovně vyžádá. Druhou akcí je, že daná hodnota bude aktualizována pravidelně nebo podle nějakého řádu, k tomuto účelu slouží Javascript. Server posílá webovému prohlížeči na základě Javascriptu XML soubor, který obsahuje aktualizované hodnoty a po vyjmutí jednotlivých hodnot z XML souboru jsou dosazeny do příslušného místa na stránce. Třetí akcí je načítání údajů od uživatele přes webové rozhraní.



Obrázek 10: Vložení proměnné do webové stránky (převzato z [12])

Datová proměnná je v HTML kódu indikována pomocí dvou znaků ~~, mezi které je název proměnné vepsán. Takto napsaná proměnná je potom voláním *callback* funkce nahrazena patřičnou hodnotou.

Příklad může být ilustrován na proměnné firmware. Na úvodní straně webového rozhraní se zobrazuje vedle nápisu *Firmware* krátký textový řetězec nesoucí informaci o aktuálním firmwaru. V HTML kódu je tato proměnná zapsána mezi znaky ~~ následujícím způsobem:

```
<div id="variable">Firmware version: <span id="firmware">~firmware~</span>
<br/><br />
                      System runtime: <span id="runTime">?</span> s<br /><br />
 </div>
```

V programovém kódu je nyní nutné implementovat novou funkci *HTTPPrint_firmware()*. Každá funkce, která bude v HTML nahrazovat danou proměnou, musí začínat následovně *HTTPPrint_* a za podtržítkem se připíše název nahrazované proměnné. Tato funkce se nachází v souboru *CustomHTTPApp.c*.

Data v HTML kódu jsou nahrazena textovým řetězcem. Je tedy nutné zavolat funkci, která náhradu provede. Voláním funkce TCPPutString() vložíme na místo proměnné v HTML kódu ~firmaware~ řetězec obsahující číslo aktuální verze firmwaru. Úplná implementace je ukázána na následujícím výpisu kódu.

```
void HTTPPrint_firmware(void)
{
      TCPPutROMString(sktHTTP, (ROM void*)FIRMWARE);
}
```

Je-li vkládán delší řetězec, je postup obdobný s pár úpravami. Aby se předešlo případu, kdy není možné odeslat celý řetězec, kontroluje se velikost výstupního zásobníku pomocí funkce TCPIsPutReady(). Funkce vrátí počet bajtů, které je možné v aktuálním stavu odeslat. Není-li dostatek místa v zásobníku, nastaví se stavová proměnná *curHTTP.callbackPos* na hodnotu jedna a funkce se opustí. V dalším průchodu programové smyčky je funkce volána znovu a pokud je už splněna podmínka dostatečného volného místa v zásobníku, vloží se požadovaný řetězec do výstupního zásobníku a proměnná *curHTTP.callbackPos* se nastaví zpět na hodnotu nula.

```
void HTTPPrint_longString(void)
{
    if(strlen((char*)lonfString) > TCPIsPutReady(sktHTTP))
    {// Not enough space
        curHTTP.callbackPos = 0x01;
        return;
    }
    TCPPutString(sktHTTP, longString;
    curHTTP.callbackPos = 0x00;
}
```

Proměnná dosazovaná do stránky popsaným způsobem je obnovována při každém novém vyžádání stránky. Pokud je nutné měnit hodnotu na stránce nezávisle, může být k tomuto účelu využit Javascript. Příkladem může být celkový čas běhu zařízení umístěný na uvítací stránce, který se neustále aktualizuje.

Nejdříve se na HTML stránce zvolí místo, kde se aktualizovaná hodnota bude vyskytovat. Tento kód může vypadat například takto:

```
<div id="variable">Firmware version:
<spanid="firmware"><td>~firmware~</td></span>
<br /><br />
System runtime: <span id="runTime">?</span> s<br /><br />
</div>
```

Proměnná se bude aktualizovat na místě elementu jménem „runTime“. Pokud hodnota nebude k dispozici, bude na jejím místě otazník, stejně jak je to patrné z předchozího kódu.

Javascript pro aktualizaci bude vypadat následovně.

```
<script type="text/javascript">
<!--
// Parses the xmlResponse from status.xml and updates the value
function updateStatus(xmlData) {

  // Update the runTime value
      document.getElementById('runTime').innerHTML = getXMLValue(xmlData,
'runTime');


}
setTimeout("newAJAXCommand('status.xml', updateStatus, true)",500);
//-->
</script>
```

Nyní je nutné vytvořit textový XML soubor obsahující element „runTime“. Tento krátký soubor bude posílán zařízením na základě žádosti z Javascriptu. Javascript poté najde příslušný element a dosadí správou hodnotu na HTML stránku. I nyní je nutné do XML souboru napsat

proměnnou, která se má aktualizovat, mezi znaky „~~”. Místo tohoto řetězce bude dosazena požadovaná zpráva.

```
<response>
<runTime>~runTime~</runTime>
</response>
```

Na straně procesoru bude implementována funkce callback, která se bude jmenovat *HTTPPrint_runTime()*. Opět za podtržítkem následuje jméno aktualizované proměnné.

```
void HTTPPrint_runTime(void)
{
      BYTE AN1String[8];
      BYTE AN2String[8];
      uitoa(runTime/3600, (BYTE*)AN0String);//houres
      uitoa((runTime - ((runTime/3600)*3600))/60 , (BYTE*)AN1String); //minutes
      uitoa(runTime - ((runTime - (runTime/3600)*3600)/60)*60,
(BYTE*)AN2String); //seconds
      TCPPutString(sktHTTP, AN0String);
      TCPPutString(sktHTTP, " ho ");
      TCPPutString(sktHTTP, AN1String);
      TCPPutString(sktHTTP, " min ");
      TCPPutString(sktHTTP, AN2String);
}
```

Ve funkci *HTTPPrint_runTime()* se vypočítávají z celkového počtu sekund hodiny a minuty, aby se hodnota zobrazovala uživatelsky přívětivě.

6.6.4.3 Čtení dat od uživatele

Pro čtení dat od uživatele jsou dvě možnosti. První je metoda GET. Je to jednodušší způsob, na Microchip platformě je ale omezen velikostí zásobníku na 100 bajtů. Jelikož je velikost dat omezená, nebyl tento způsob získávaní dat od uživatele v projektu použit.

Druhou možností je metoda POST. Metoda Post není již limitována velikostí dat. Její zpracování je obtížnější než zpracování metody GET.

Na HTML stránce musí být umístěn element *form*, který bude charakterizován metodou POST:

```
<form method="post" ...>
```

Jako odpověď budou odeslána všechna výstupní dat z formuláře. Výhodou této metody sice

je neomezená velikost posílaných dat a také plná kontrola nad přijímacím zásobníkem, jak bude ukázáno později, nevýhodou je však složitější zpracování. Data mohou dorazit v různém pořadí, není zajištěná žádná předem daná posloupnost.

Kód implementovaný na HTML stránce pro změnu hesla je na následující ukázce. Na stránce jsou umístěny tři vstupní textová pole, omezená na 30 znaků. Vkládá se nejdříve původní heslo a poté dvakrát nové heslo.

```
<form method="post" action="password.html" name="password">
  <fieldset>
        <div><label>Old password:</label>
        <td>
       <input type="text" name="oldPassword0" maxlength="30" /></td></div>
        <div><label>New password:</label>
        <td>
        <input type="text" name="newPassword1" maxlegth="30" /></td></div>
        <div><label>New password again:</label>
        <td>
       <input type="text" name="newPassword2" maxlength="30" /></td></div>
        <div><label><span id="mesage">~mesage~</span></label></div>
       <div> </div>
     <div><input type="submit" class="sm" value="Change password" /></div>
  </fieldset>
</form>
```

V kódu procesoru se do funkce *HTTP_IO_RESULT HTTPExecutePost()* připíše nová funkce, která bude obsluhovat metodu POST pro změnu hesla. Identifikace POST-metody spočívá nejdříve v kontrole názvu souboru HTML stránky, ze které POST-metoda pochází. Kontrola se děje právě ve funkci *HTTPExecutePost()*. V případě změny hesla pochází POST-metoda ze souboru *password.htm*. Aktuální jméno souboru, které je součástí žádosti o obsluhu, je porovnáváno s názvem *password.htm.* Souhlasí-li název, zavolá se obsluha příslušné POST-metody.

```
if(!strcmppgm2ram((char*)filename, "password.htm"))
            return HTTPPostPassword();
```

Funkce pro obsluhu změny hesla se jmenuje *HTTPPostPassword()* a má tři návratové hodnoty. Nejsou-li načtená dostatečná data, aby mohla být použita, funkce vrací hodnotu HTTP_IO_NEED_DATA. Dokud nejsou načtena všechna data, funkce vrací hodnotu HTTP_IO_WAITING a je volána opakovaně dokud není vrácena hodnota HTTP_IO_DONE.

Dostatečný počet přijatých dat lze kontrolovat funkcí *TCPIsGetReady(sktHTTP)*, která vrací počet bajtů v přijímacím zásobníku zvoleného TCP portu. Příjem dalších dat se provádí pomocí smyčky *while()*, kdy se porovnává hodnota *curHTTP.byteCount*, která je při každém novém čtení dat snižována až na hodnotu jedna, což signalizuje konec příjmu dat. Uvnitř této smyčky dochází k

porovnání jména elementu. Pokud jméno elementu souhlasí s požadovaným jménem, je hodnota elementu načtená do proměnné. Nesmí být opomenuto, že data mohou být posílána v různém pořadí. To znamená, že původní heslo nemusí být přijato jako první, ale třeba jako poslední. Na následujícím výpisu je ukázána funkce pro změnu hesla. Funkce z důvodu svého většího rozsahu je ve výpisu zkrácena, neobsahuje samotnou záměnu hesla.

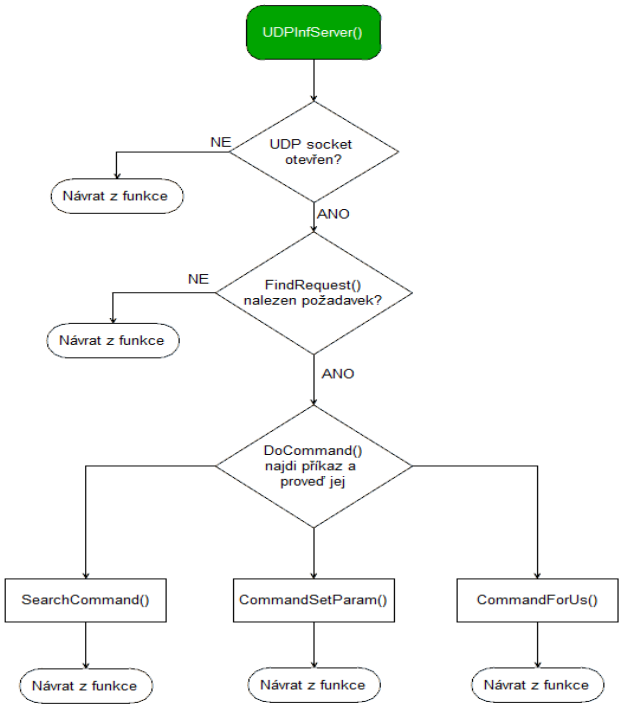
```
static HTTP_IO_RESULT HTTPPostPassword(void)
{
      if(TCPIsGetReady(sktHTTP) < curHTTP.byteCount)
            return HTTP_IO_NEED_DATA;

      while(curHTTP.byteCount)
      {
            // Read a form field name
            if(HTTPReadPostName(curHTTP.data, 14) != HTTP_READ_OK)
                  return HTTP_IO_WAITING;

            // Read a form field value
            if(HTTPReadPostValue(curHTTP.data + 14, sizeof(curHTTP.data)-14-
2) != HTTP_READ_OK)
                  return HTTP_IO_WAITING;

            // Parse the value that was read
            if(!strcmppgm2ram((char*)curHTTP.data, (ROM char*)"oldPassword0"))
            {// Read new static IP Address
                  memcpy((void*)oldPassword, (void*)curHTTP.data+14, 30);
                        return HTTP_IO_WAITING;
            }
            else if(!strcmppgm2ram((char*)curHTTP.data, (ROM
char*)"newPassword1"))
            {// Read new gateway address
                  memcpy((void*)newPassword1, (void*)curHTTP.data+14, 30);
                        return HTTP_IO_WAITING;
            }else if(!strcmppgm2ram((char*)curHTTP.data, (ROM
char*)"newPassword2"))
            {// Read new gateway address
                  memcpy((void*)newPassword2, (void*)curHTTP.data+14, 30);
                        return HTTP_IO_WAITING;
            }
      }
      ...
      return HTTP_IO_DONE;
}
```

6.7 UDP server

Pokud není známa IP adresa zařízení, tak se k webovému rozhraní nedostaneme. Je sice možné resetovacím tlačítkem uvést zařízení do „továrního“ nastavení, ale tím budou veškerá nastavení nenávratně ztracena. Lepším řešením je využít služeb UDP serveru, který není vázán na konkrétní IP adresu. UDP server přijímá zprávy na portu 49344 od všech IP adres dané sítě a také všem odpovídá. To znamená, že na daném segmentu sítě si UDP zprávu mohou přečíst všichni, tedy

pokud UDP zpráva je vysílána všem. UDP zpráva může být samozřejmě směřována i na konkrétní IP adresu. UDP server je kompatibilní s protokolem Quatech UDP [10], který firma poskytuje volně pro podporu svých zařízení. Pro komerční použití bude muset být implementován protokol vlastní. Přes rozhraní UDP je možné nastavovat síťové parametry jako jsou IP adresa, maska sítě a podobně. UDP server je programován jako rozhodovací strom. Jeho struktura je patrná z obrázku 11. Prvním voláním funkce *UDPInfServer()* je otevřen UDP socket a pokud je jeho otevření úspěšné, přepne se server do stavu naslouchání.

Ve stavu naslouchání je volána funkce *FindRequest()*. Funkce nejdříve zkontroluje, jestli je v přijímacím zásobníku počet bajtů odpovídající délce dat požadované protokolem. Funkce v dalším kroku porovná číslo požadavku ve zprávě s čísly požadavků, které server umí obsloužit. Pokud je nalezen platný požadavek, zavolá se funkce *DoCommand()*, která podle čísla požadavku zavolá obslužnou funkci. Po provedení požadovaného příkazu následuje návrat do programové smyčky.



Obrázek 11: Obsluha UDP serveru

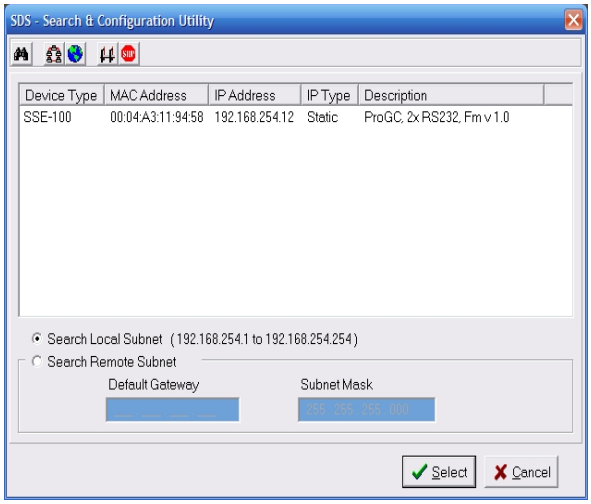
Funkce *CommandForUs()* a *CommandSetParam()* obsluhují jenom ty zprávy, které jsou posílány výhradně na IP adresu zařízení, která byla v předchozím příkazu odeslána do počítače. Funkce *CommandForUs()* navíc porovnává MAC adresu zaslanou ve zprávě se svojí MAC adresou. Je to nutné zabezpečení, jelikož tento požadavek slouží ke změně hesla.

Aby mohla být jednotlivá data jednoduše přečtena z příchozího řetězce, je s výhodou využito ukazatelů. Příchozí data se přetypují na strukturu *UdpPacket*. Struktura *UdpPacket* se skládá ze struktur *UDP_INF_HEADER* a ze struktury *COMMAND_x* odpovídající příslušnému příkazu. Přístup k jednotlivým datům v příchozí zprávě se tím usnadní a zpřehlední.

Jednotlivé struktury vypadají následovně:

```
typedef struct _COMMAND_0
{
   char     Name[10];
}COMMAND_0;

typedef struct _COMMAND_1
{
   unsigned char     LocalIpAddress[LEN_IP_ADDR]; //
   unsigned char     GatewayAddress[LEN_IP_ADDR]; //
   unsigned char     IpAddressMask[LEN_IP_ADDR];  //
   unsigned char     StaticOrDhcp;                // static = 0, dhcp = 1
   unsigned short    ProductIdWord;               // orginal product identifier
read from H/W
   char              FwRevNumber[LEN_REV];        // FW revision - ASCII
   char              FriendlyName[LEN_FRIENDLY];  //
}COMMAND_1;

typedef struct _COMMAND_4
{
   unsigned short    Status;
}COMMAND_4;

typedef struct _UDP_INF_HEADER
{
   WORD   Length;                     // length of header sizeof(UDP_HEADER)
   unsigned char    Command;
   unsigned char    MacAddress[LEN_MAC_ADDR];   // six byte ethernet MAC
address
   unsigned char    Broadcast;                  // set to indicate if response
from ESD should be
                                                // broadcast back or sent
directly to sender IP address
   char             UserName[LEN_USERNAME];     //
   char             PassWord[LEN_PASSWORD];     //
}UDP_INF_HEADER;

typedef struct _UDP_PACKET
{
   UDP_INF_HEADER       Header;        // header

   union
   {
      COMMAND_0    CmdData0;     // initial search command - sds responds with
command 1
      COMMAND_1    CmdData1;     // ipCfg data reponse sent for command 1
      COMMAND_1    CmdData2;     // ipCfg data sent with command 2..same as
COMMAND_1
      COMMAND_4    CmdData4;     // command status reponse
   } Data;
}UdpPacket, *UdpPacket_P;
```

Na obrázku 12 je zobrazeno okno programu Quatechtool firmy Quatech [23] s přihlášeným zařízením ProGC. Přes tento program lze zařízení nastavovat a resetovat.



Obrázek 12: Okno programu Quatechtool pro nastavení síťových parametrů

Kapitola 7

Hardware

Software popsaný v předchozí kapitole byl testován na vývojovém kitu. Vývojový kit lze chápat jako nultý prototyp, obsahuje všechny klíčové komponenty, které bude obsahovat i výsledné zařízení a všechny funkce softwaru i samotná jeho funkčnost lze na něm spolehlivě ověřit. Přesto výsledné zařízení bude jiné. Bude obsahovat dostupné součástky a ty se mohou lišit od vývojového kitu. I samotný návrh, především plošného spoje, se nemusí vyhnout všem chybám, přestože autor dělá maximum, aby jim předešel. Chyby ve fázi vývoje hardwaru mají zcela jiný charakter než chyby v softwaru. Chyba v softwaru se odstraní přepsáním firmwaru procesoru a její opravu lze hned otestovat. Tuto možnost vývoj hardwaru neumožňuje. Hardware se skládá nejen ze součástek správných parametrů a vlastností, ale také z plošného spoje. Chyba typu špatné hodnoty součástky se dá vyřešit její jednorázovou výměnou a opětovným otestováním. Chyba v návrhu plošného spoje může ale způsobit nemalou komplikaci. Elektrické schéma může být zcela v pořádku, přesto však plošný spoj může být špatně navržený. Chyba v návrhu plošného spoje často znamená i časové zdržení, jelikož výroba desky se musí zadávat externí firmě a s tím je spojená i administrativa. Rovněž samotná výroba plošného spoje trvá několik dní. Je nutné dodržet celou řadu doporučení, jak by měl plošný spoj vypadat [15], aby byla zajištěná řádná činnost výsledného elektrického zařízení. Know-how jak navrhovat plošné spoje se učí spíše praxí, jelikož teoretická doporučení pro tvorbu kvalitního spoje nelze vždy dodržet, ať už jsou těmi důvody rozměry součástek, rozměry samotného plošného spoje nebo dostupná technologie výroby či návrhu.

7.1 Klíčové komponenty

Mezi klíčové komponenty patří integrované obvody PIC32MX795F512L, obvod DP83848C a obvod MAX3243. Tyto tři součástky jsou základem zařízení. Jejich absence by celý projekt značně

zkomplikovala.

7.1.1 PIC32MX795F512L

PIC32 je procesor amerického výrobce Microchip [8]. Procesory řady PIC32 jsou plně 32-bitové, založené na jádře RISC. Procesor pracuje na frekvenci až 80 MHz. Model PIC32MX795F512L [19] disponuje operační pamětí o velikosti 128 KB a programovou flash pamětí o velikosti 512KB. Obsahuje vnitřní oscilátory 8 MHz a 32 KHz, které jsou nekalibrované a pro stabilnější taktovací frekvenci se připojuje externí krystal. Procesor je napájen napětím 2,5 V až 3,6 V. Vybrané vývody procesoru jsou 5 V tolerantní. Procesor disponuje 5 rozhraními I2C, 4 ISP , 5 kanály UART a rozhraním ethernet MAC 10/100 Mbit /s podporující standard RMII a MII. Mezi klíčové vlastnosti pro projekt patří zmíněné rozhraní ethernet MAC a dostatečný počet UART rozhraní. Procesor je možné programovat přímo v uživatelské aplikaci za pomocí třívodičového rozhraní ICSP (Microchip standard). Procesor má 100 pinů, z nichž 85 lze použít jako vstupy a výstupy k obecnému použití.

7.1.2 DP83848C

Obvod DP83848C [17] je výrobkem firmy National Semiconductors. Obvod tvoří fyzickou vrstvu ISO/OSI modelu počítačové sítě. S procesorem může komunikovat pomocí rozhraní RMII, MII nebo SNI. Obvod automaticky detekuje kolize, komunikuje rychlostí 10 i 100 Mbit/s, rozpozná přímé i překřížené kabely a přizpůsobí se. Tři připojené LED diody lze naprogramovat na požadovanou funkci: detekce kolize, aktuální mód (10 nebo 100Mbit), aktivita linky.

7.1.3 MAX3243

Mezi výrobce tohoto polovodičového čipu patří firmy MAXIM a Texas Instruments. Obvod zajišťuje přechod mezi úrovněmi linky RS232 a CMOS úrovní, navíc je vybaven ochranou proti elektrostatické elektřině ± 15 kV, podle modelu HMB (Human-Body model). Disponuje 5 vstupy úrovně RS232, následně 5 výstupy úrovně CMOS a 3 vstupy úrovně CMOS s následně 3 výstupy úrovně RS232. Obvod lze napájet napětím 3V až 5,5 V, viz [18].

7.2 Výběr součástek

Kromě klíčových komponent obsahuje zařízení velkou spoustu pasivních součástek jako kondenzátory, odpory, konektory apod. Mezi klíčové vlastnosti těchto komponent patří především

typ pouzdra a v neposlední řadě i jejich elektrické vlastnosti. Aby byla velikost výsledného plošného spoje co možná nejmenší, byl vybrán pro odpory a kondenzátory typ pouzdra 0805. Jedná se o běžnou velikost součástek v provedení SMD, což jsou součástky určené pro povrchovou montáž. Velikost 0805 byla zvolena i pro svoji dobrou dostupnost u dodavatelů elektronických součástek a samotná velikost pouzder je ještě dostatečně veliká, aby je bylo možné osazovat ručně při výrobě prototypu. Výjimku tvořily tantalové kondenzátory, které v této velikosti nejsou dobře dostupné a tak byly použity v pouzdru typu B. Konektory jsou standardizované a příliš modifikací nenabízí, návrh se přizpůsobil dostupným konektorům.

Nejdůležitější bylo použít takové součástky, které jsou dodavatelé schopni v krátkém čase poskytnout.

7.2.1 Dodavatelé

Prvním zvoleným dodavatelem byla firma GME [16], se kterou se autor seznámil v dřívější době. Tento dodavatel nebyl schopný poskytnout všechny potřebné komponenty. Dalším dodavatelem byla firma Farnell [2], která se ukázala být relativně dobrou volbou. Procesory a vývojové nástroje byly zakoupeny přímo u výrobce přes jeho internetový obchod Microchip-direct. Použité součástky jsou přímo spojené s dodavatelem, nelze vybrat součástku, kterou není možné koupit. Aby bylo možné součástky objednat, bylo nutné strávit dlouhé hodiny vyhledáváním objednávacího čísla vhodné součástky v nepřeberném množství nabízených produktů různých výrobců, parametrů a dostupností.

7.3 Elektrické schéma

Elektrická schémata byla vytvořena v programu Eagle ve verzi „free“. Volná verze limituje velikost plošného spoje na velikost 100 x 100 mm a na použití nejvýše dvou vrstev. Úplné elektrické schéma je k nahlédnutí v Příloze B.

7.3.1 Procesorová deska

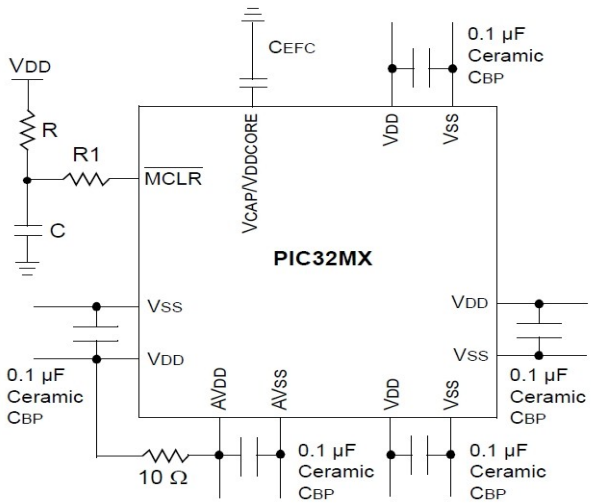
Procesorová deska je hlavní díl obsahující samotný procesor, ethernetový obvod , oscilátory, krystaly, převodníky úrovně signálů, napájecí část a konektory. Elektrické zapojení procesorové desky bylo vytvořeno na základě doporučeného zapojení výrobců použitých polovodičových obvodů.

7.3.1.1 Zapojení procesoru

Největší pozornost bylo třeba věnovat zapojení procesoru PIC32MX795F512L a ethernetovému obvodu DP83848C.

Klíčovou záležitostí je zajistit dobré napájení. Procesor obsahuje celkem 14 vývodů určených pro jeho napájení, včetně napájení analogové části. Podle doporučení výrobce mají být blízko k vývodům připojené blokovací kondenzátory o kapacitě 100nF. Důležité pro činnost procesoru je zapojení napájecího vývodu (pin 85). Tímto vývodem je napájen vnitřní stabilizátor pro napájení samotného jádra procesoru. Vývod je nutné blokovat dvěma kapacitami C_{EFC} 0,1 nF a 10µF. Kondenzátory mohou být tantalové nebo keramické, musí být nízko impedanční (1 ohm). Při nedržení parametrů procesor nefunguje správně. Tuto skutečnost si autor nedobrovolně ověřil.

Procesor PIC32 ze své podstaty nepotřebuje resetovací obvod, přestože je možné jej připojit. Hodnoty resetovacího obvodu výrobce stanovuje pouze jen jako orientační a uvádí, že je nutné správnou hodnotu určit až na konkrétním zapojení. Důvodem, proč procesor nevyžaduje resetovací obvod, je přítomnost vnitřního regulátoru napětí pro jádro procesoru. Reset procesoru vyvolá náběžná hrana napájení. Na obrázku 13 je uvedeno základní doporučené zapojení procesoru PIC32.



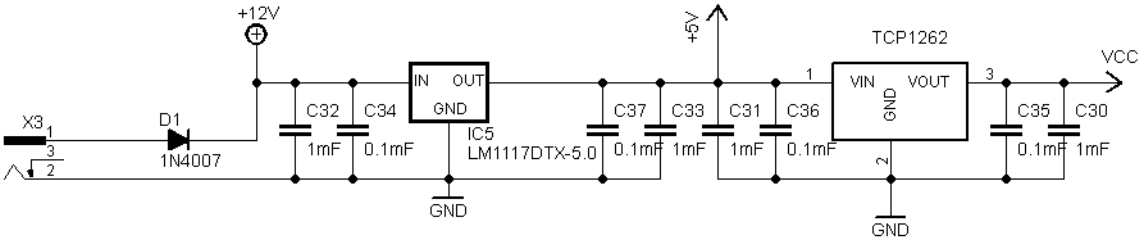
Obrázek 13: Základní doporučené zapojení procesoru PIC32 (převzato z [19])

Krystalový obvod má být pomocí blokovacích kondenzátorů připojen k zemi. Hodnota blokovacích kondenzátorů může být v rozsahu 10 pF až 33 pF. Větší hodnota se nedoporučuje, poněvadž by mohla zcela zabránit rozkmitání oscilátoru. Čím je kapacita větší, tím déle trvá

oscilačnímu obvodu se rozkmitat na požadované frekvenci. Nevhodná hodnota může mít za následek nesprávný takt oscilačního obvodu. Procesor se programuje pomocí třívodičového zapojení ICSP. Programátor se připojuje k vývodům označeným PGC, PGD a MCLR. Vývod MCLR je součástí resetovacího obvodu. Je-li resetovací obvod připojen, je nutné zabezpečit odpojení resetovacího obvodu při programování, podle doporučení výrobce třeba zkratovací propojkou („jumper“). Rezonanční krystaly musí být umístěné co nejblíže k vývodům procesoru, ke kterým jsou připojené.

7.3.1.2 Napájení procesorové desky

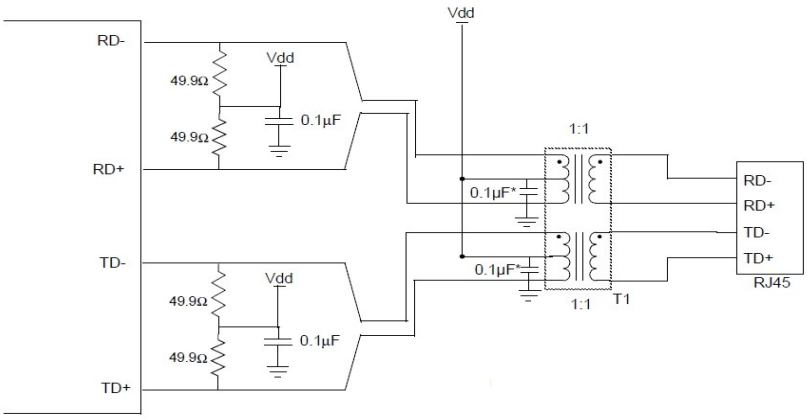
Samotný procesor je napájen napětím 3,3 V, které zajišťuje lineární stabilizátor TCP1262-3.3V. Tímto napětím jsou napájeny všechny komponenty. Stabilizátoru TCP1262 předchází stabilizátor LM1117, jehož stabilizované napětí je 5V. Napětím 5V je napájená přídavná deska zabezpečující galvanické oddělení pro signály ze stroje. Na vstupní napájecí části je zapojená dioda v propustném směru. Zabraňuje náhodnému přepólování, Je na ní trvalá ztráta napětí kolem 0,7 V. Ke stabilizátoru napětí LM1117 je možné připojit napětí 5 V až 15 V. Na obrázku 14 je zobrazená napájecí část.



Obrázek 14: Napájecí část procesorové desky

7.3.1.3 Zapojení ethernetového obvodu DP83848C

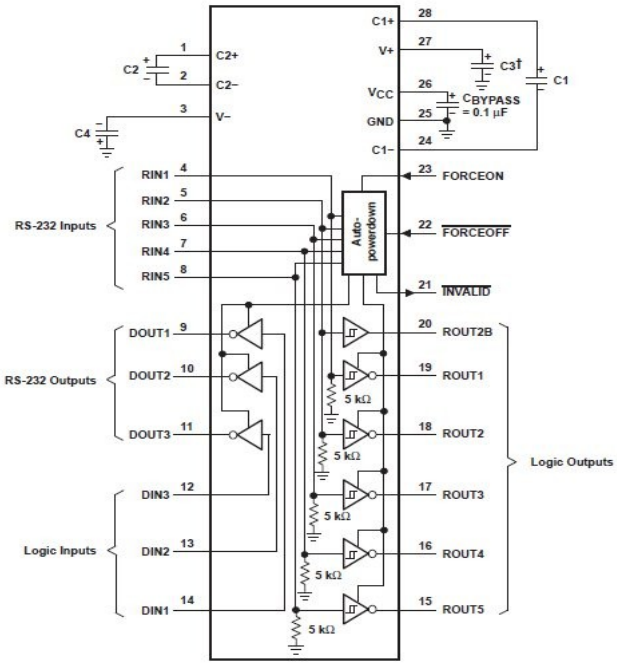
Obvod DP83848C je připojen k procesoru v režimu RMII a je propojen s procesorem podle zapojení, které Microchip použil na vývojovém kitu. V zapojení ethernetového nejsou žádné záludnosti, pouze blokovací kondenzátory by měly být umístěny co možná nejblíže k obvodu. Obvod obsahuje také vnitřní stabilizátor pro napájení jádra obvodu. Napájení vnitřního stabilizátoru by mělo být blokováno tantalovými kondenzátory. Připojení oddělovacího transformátoru bylo provedeno podle doporučení výrobce, viz obrázek 15.



Obrázek 15: Připojení oddělovacího transformátoru (převzato z [17])

7.3.1.4 Zapojení obvodu MAX3243

Obvod MAX3243 [18] je nábojovou pumpou, která převádí úrovně signálu mezi RS232 a CMOS. Obvod je zapojen podle doporučení výrobce obrázek 16.



Obrázek 16: Zapojení obvodu MAX3243 (převzato z [18])

7.3.1.5 Přídavné konektory

Do schematu bylo zapojeno několik přídavných konektorů. Konektor pro připojení k programátoru, konektor pro připojení druhé desky obsahující galvanické oddělení signálů ze stroje a konektor pro připojení desky obsahující elektroniku pro bezdrátový ethernet.

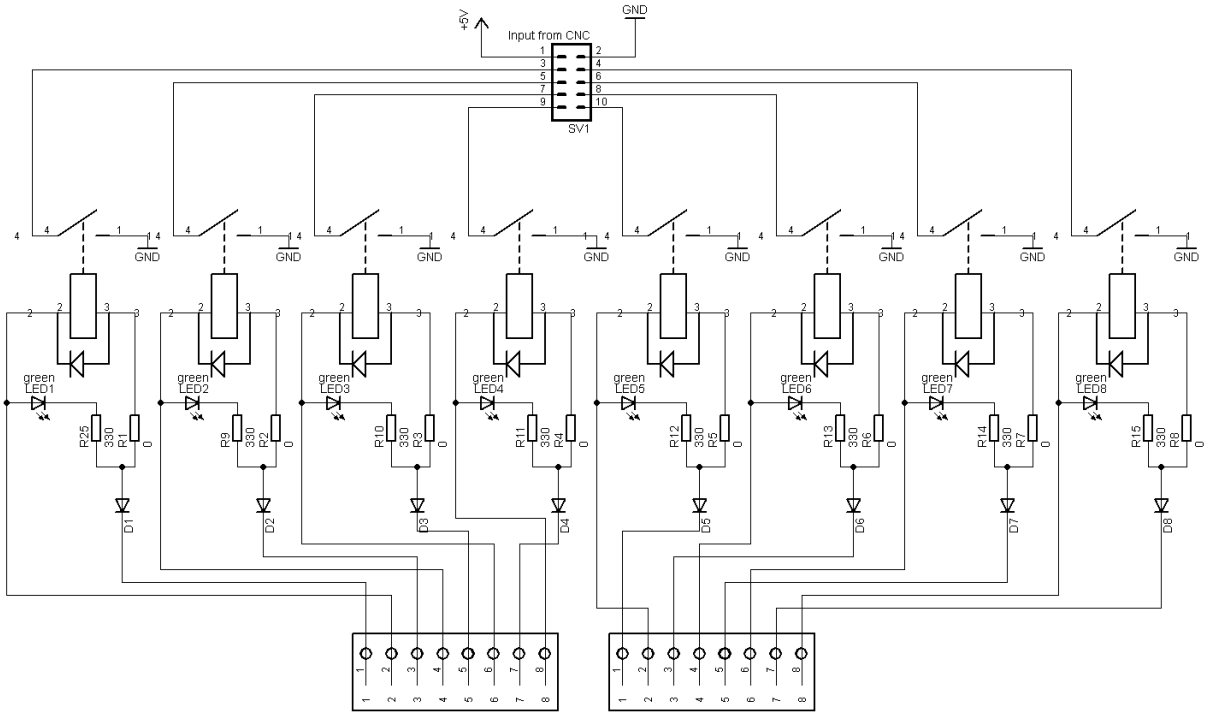
(WiFi část nebyla implementována, dodavatel [8] oznámil dobu dodání na druhou polovinu února 2011.)

7.4 Snímání signálů ze stroje

Snímané signály ze stroje jsou dvoustavové. Ze stroje se snímají signály, které ovládají spínací relé, kontrolní světla nebo signály, které mají přímou souvislost se sledovanou činností CNC stroje. Napětí těchto signálů má ve většině případů hodnotu 0 V nebo 24V. U některých strojů se můžeme setkat i s napětím 12 V. Signály nelze přímo propojit se zařízením. CNC stroje jsou velmi drahá zařízení a proto je na místě maximální opatrnost při jakémkoliv zásahu do stroje, i kdyby byla sebe banálnější. Proto je nutné signály galvanicky oddělit. K tomuto účelu je možné použít například relé nebo optické galvanické oddělení.

7.4.1 Reléové oddělení signálů

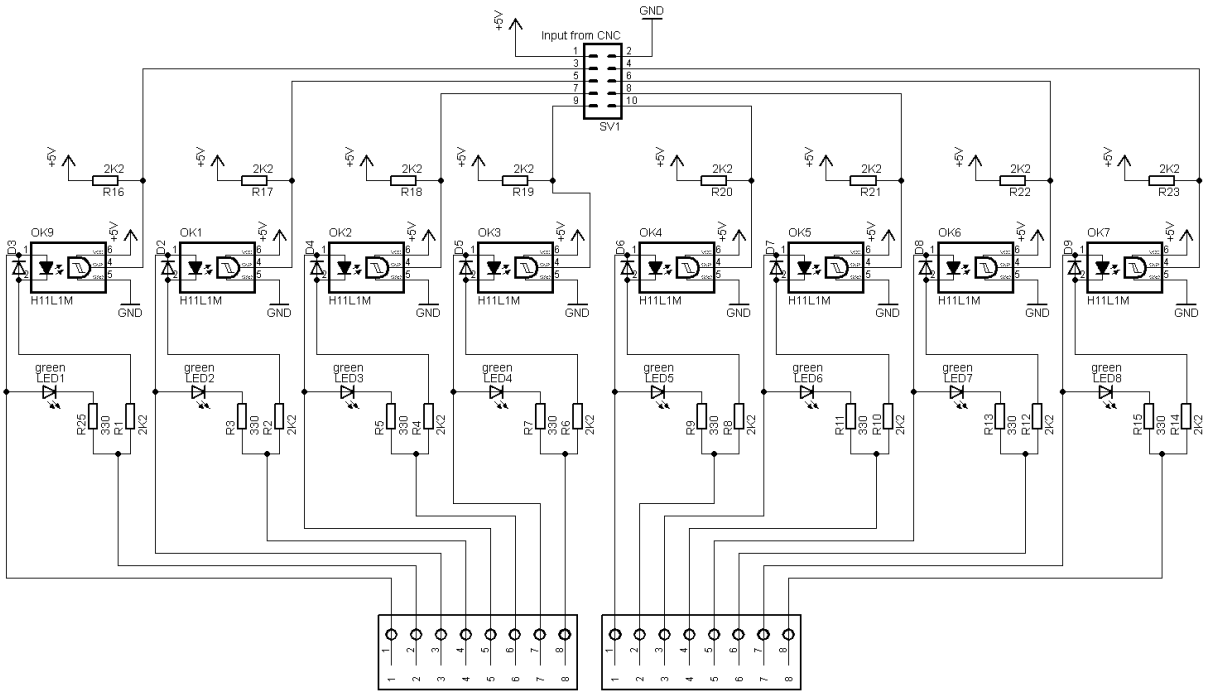
Reléové oddělení je již praxí odzkoušeným způsobem, jak galvanicky oddělit signály snímané z CNC stroje. V praxi se používají relé připojená na DIN lištu, signál ze stroje relé sepne a I/O modul pošle přes ethernet zprávu o změně stavu. Ve vyvíjeném zařízení je to obdobné s tím rozdílem, že relé jsou již obsažená přímo v zařízení a není tedy žádných externích relé zapotřebí. Spínací kontakty relé jsou připojeny k portu procesoru a v sepnutém stavu jsou spínané k zemi. Relé jsou opatřeny diodou, při odpojení se v diodě maří energie vznikající v cívce relé při odpojení proudu. Dioda chrání spínací tranzistor před nedovoleným proudem opačné polarity, který by jím jinak protékal. Autor měl možnost vidět používání relé v praxi. Používaná relé nebyla opatřená diodou. Pro 100% jistotu, že se elektronice stroje nic nestane, je moudřejší diodu použít. Aby nedošlo ke zkratu při opačném připojení relé, kdy by dioda byla připojena do spínacího okruhu v propustném směru, byla do série zařazena ještě jedna bezpečnostní dioda zabraňující případnému zkratu. Sepnuté relé může být zároveň signalizováno rozsvícením LED diody. Schéma reléové části je na obrázku 17.



Obrázek 17: Reléová část

7.4.2 Optické oddělení signálů

Dalším způsobem galvanického oddělení je použití optoisolátorů. I tento způsob se v praxi již používá a patří mezi častější varianty snímání signálů. Zařízení bylo navrženo tak, aby se mohl díl se vstupní částí podle potřeby vyměnit buď za optoisolátor nebo za relé. Jako oddělovač je použit optoisolátor se Schmittovým klopným obvodem. Tento optoisolátor vyžaduje napájení 5 V a výrobce zaručuje elektrickou pevnost až do napětí 5 kV. Výstupem optoisolátoru je definovaná TTL úroveň. Na výstupu je tranzistor s otevřeným kolektorem, je nutné připojit zdvihací odpor pro definování vysoké úrovně. Paralelně s integrovanou diodou byla zapojena druhá dioda, aby se při přepólování zabránilo jejímu zničení. Pomocí předřadného odporu je definován proud protékající diodou na hodnotu 40 mA. Na obrázku 18 je schéma zapojení optoisolátorů.



Obrázek 18: Zapojení optoisolátorů

7.4.3 Svorkovnice vstupní části

Na plošném spoji jak reléové části, tak i optoisolátorové části, jsou umístěny dva konektory, do kterých je možné zasunout svorkovnici se šrouby, do niž se připevní přívodní kabely se signálem, obrázek 19.

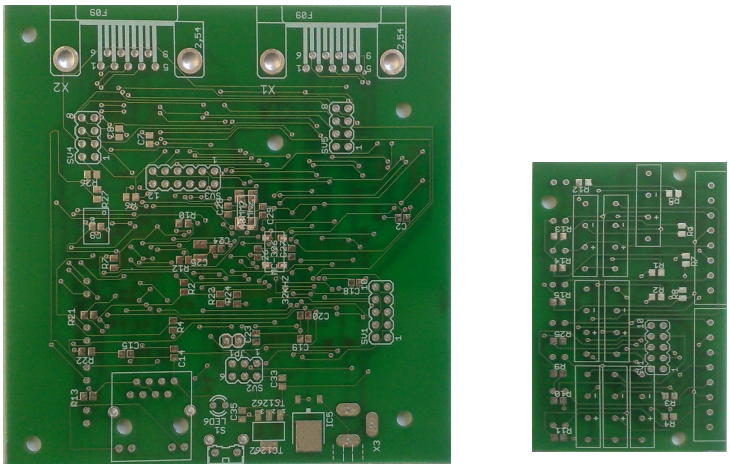
Obrázek 19: Svorkovnice pro připojení signálů ze stroje (převzato z [2])

7.5 Plošný spoj

Při návrhu bylo třeba brát na zřetel rozmístění součástek, aby odpovídalo pokud možno doporučením výrobců jednotlivých komponent. Zařízení se skládá celkem ze dvou plošných spojů. Při návrhu se musely zajistit rozměry desek tak, aby se správně překrývaly a propojovací konektory byly správně umístěny pod sebou. Nesměly být opomenuty ani montážní otvory pro sešroubování desek k sobě. Dalším kritériem byl samotný rozměr plošného spoje. Plošný spoj bude umístěn v boxu, rozměry desky tomu musí odpovídat.

Po správném nastavení rozměrů plošného spoje a rozmístění součástek následovalo vytvoření vodivých cest. K tomuto účelu byl použit automatický router vestavěný v programu Eagle. Aby byla automatická tvorba cest úspěšná, bylo zapotřebí nastavit všechny parametry jako šířka vodivé linky, velikost propojovacího otvoru, odstupy od součástek apod. Tyto parametry bylo nutné ladit, aby byl „auto-router“ úspěšný. Ruční návrh vodivých cest by vzhledem ke složitosti motivu byl velmi časově náročný. Hotový hlavní díl plošného spoje a deska s galvanickým oddělením jsou na obrázku 20.

Výroba plošného spoje vyžadovala vyšší třídu přesnosti a ne každý výrobce plošných spojů umí tuto třídu přesnosti dodržet. S vyšší třídou přesnosti roste i cena plošného spoje. První prototyp byl vyroben firmou PCB Benešov. Cenová nabídka na vyhotovení druhého prototypu firmou PCB Benešov překračovala 15000 Kč. To byla neakceptovatelná částka. Druhý prototyp vyrobila firma Petr Huráb Havířov za několikanásobně nižší cenu. Deska ale musela být předělána na rozměry, které tato firma uměla vyrobit, tzn. síla linky 0,2 mm a propojky 0,5 mm. Firma PCB Benešov žádná omezení nemá a je schopná vyrobit prakticky jakýkoliv plošný spoj.



Obrázek 20: Procesorová deska a deska s galvanickým oddělením

7.6 Pájení součástek prototypu

Pájení SMD součástek se liší od pájení běžných součástek a ruční pájení je obecně hodně pracné a v některých případech až nemožné. SMD součástky jsou určené pro strojovou výrobu plošného spoje.

Pro vytvoření vodivého spojení se nanáší malé množství cínu. Odměřit tak malé množství cínu z trubičkové pájky je takřka nemožné. Proto se pro pájení SMD součástek používá pájecí pasta, která je směsí cínu a dalších látek. Pasta se prodává v tubě podobné injekční stříkačce pro snazší dávkování. Autor ji doplnil injekční jehlou, aby bylo možné pastu přesně dopravit na požadované místo. Pasivní součástky bylo možné připájet pomocí hrotové pájky. Procesor a ethernetový obvod vyžadoval zcela jiný postup.

Výstupní piny procesoru jsou od sebe vzdálené 0,3 mm a procesor jich má celkem 100. Sebemenší pohyb vyvolal okamžité posunutí procesoru. Jako první byl připájen jeden pin procesoru a poté na protější straně druhý pin. Tím byl procesor zabezpečen proti posunutí. Pájecí pasta se nanesla v malém množství kolem všech pinů procesoru a pomocí horkého vzduchu byla pasta s cínem roztavena a tím došlo k připájení. Poté byla kolem pinů procesoru nanesena tekutá kalafuna a hrotovou pájkou prohřáty všechny piny. Tento postup zabezpečil 100% připájení procesoru k plošnému spoji.

Stejný postup byl dodržen i pro ethernetový obvod.

7.7 Navržené prototypy

Celkem byly navrženy 3 prototypy.

První zkušební prototyp ověřoval zapojení součástek a měl být umístěn v boxu, který dodavatel přes veškerá ujišťování před objednáním nedodal. I některé další součástky potkal stejný osud a nebyly dodány. Přesto se na tomto prototypu mohlo ověřit samotné zapojení zařízení. Zprovoznění nebylo jednoduché. Po prvním připájení procesoru sice vše nasvědčovalo dobrému kontaktu, pravda byla ale jinde. Po dlouhých oživovacích pokusech s přesvědčením, že procesor je dobře připájený a překontrolování všeho, byl osazen druhý plošný spoj, který již fungoval. Bylo zjištěno, že problémem byl přece jen špatně připájený procesor.

Druhý prototyp, který se od třetího prototypu liší pouze tvarem plošného spoje, byl navržen pro součástky jiného spolehlivějšího dodavatele. V době psaní tohoto textu byly plošné spoje dodány a k osazení a k odzkoušení ještě nedošlo.

Třetí prototyp byl stejný jako druhý, lišil se ve velikosti a tvaru plošného spoje, byl určen do

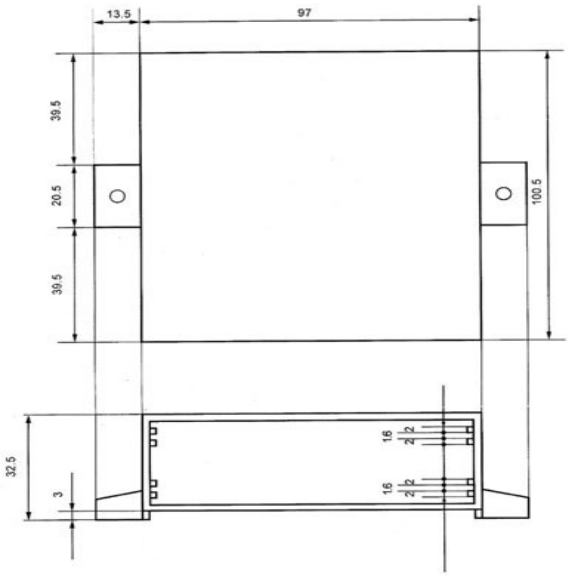
jiného boxu.

7.7.1 Programování prototypu

Naprogramování procesoru zabudovaného na plošném spoji bylo prováděno pomocí ladícího nástroje MPLAB ICD 3. Jedná se o produkt firmy Microchip. Programátor používá k nahrávání programu tři vodiče DATA, CLOCK, MCLR. Je nutné připojit i dva signály napájení GND, Vcc. Programátor se připojuje k počítači přes USB port. Programované zařízení může být napájeno i přes samotný programátor.

7.8 Box pro zastavění plošného spoje

Původně vybraný box nebyl dodavatelem dodán. Byl vybrán alternativní box. Jedná se o plastový box černé barvy a o vnitřních rozměrech 91 x 92 mm (měřeno posuvným měřítkem). Box má po stranách úchyty k přišroubování. Plošný spoj se vkládá do drážek, viz obrázek 21.



Obrázek 21: Box pro vestavbu plošného spoje (převzato z [16])

Tento box slouží pouze pro prototyp. Do boxu mají být vyvrtány otvory pro sériové porty a kontrolní LED diody.

V budoucnu by mělo být zařízení vestavěné do boxu vlastní konstrukce. Materiálem má být hliníkový plech, který by byl ve větším množství strojně nařezán na požadovaný tvar a naohýbán. Vnější strana boxu by byla povrchově upravena. Tato konstrukce boxu vyžaduje více časové dotace a zůstává ve fázi návrhu.

Kapitola 8

Rozpočet

Zajímavou částí projektu je jeho cena. Od začátku se autor snažil používat cenově výhodné komponenty, vývojové prostředky a výrobce. V tabulce 8 je stručné shrnutí cen. V konečné částce jsou započítané položky nejen za elektronické komponenty, ale i za vývojové nástroje, výrobu plošných spojů a zařízení nutné pro sestavení prototypu.

Počet	Popis	Hodnota	Cena
1	Programming 32-bit microcontrollers in C	kniha	755 Kč
4	PCB Benešov, výroba DPS	1. prototyp	4 157 Kč
12	PETR HURAB Havířov, výroba DPS	2. a 3. prototyp	3 881 Kč
1	WIFI – vývojový kit	WIFI	1 200 Kč
1	Programátor procesoru MPLAB ICD 3	programátor	3 800 Kč
1	PIC32 STARTER KIT	vývojová deska	1 756 Kč
1	Rozšiřovací deska pro PIC32 STARTER KIT	vývojová deska	1 756 Kč
1	Pájecí multifunkční stanice	páj. Stanice	9 870 Kč
1	Farnell objednávka 1 elektrické komponenty	Farnell 1	2 814 Kč
1	Farnell objednávka 2 elektrické komponenty	Farnell 2	272 Kč
1	Farnell objednávka 3 elektrické komponenty	Farnell 3	269 Kč
1	Farnell objednávka 4 elektrické komponenty	Farnell 4	35 Kč
1	Farnell objednávka 5 elektrické komponenty	Farnell 5	930 Kč
1	Farnell objednávka 6 elektrické komponenty	Farnell 6	138 Kč
1	Farnell objednávka 7 elektrické komponenty	Farnell 7	54 Kč
1	Farnell objednávka 8 elektrické komponenty	Farnell 8	1 000 Kč
1	Elektrické komponenty a ostatní komponenty	ostatní	3 737 Kč
		Celkem	36 424 Kč

Tabulka 8: Finanční zatížení projektu

Veškeré komponenty zakoupené pro projekt jsou podložené daňovým dokladem.

Kapitola 9

Závěr

Jedná se o velký projekt skládající se z více částí. Každá část by mohla být zpracovávaná jako samostatná práce. Obstarávat všechny části projektu bylo velmi vyčerpávající a stresující. Projekt se skládá z částí jako tvorba webového rozhraní, implementace TCP protokolu, UDP protokolu, I2C komunikace, implementace webového rozhraní do kódu procesoru, konstrukce hardwaru a box pro zastavění výsledného plošného spoje. Programování procesoru PIC32 bylo díky podpoře Microchipu hodně usnadněno, přesto bylo nutné se naučit zacházet s programovými knihovnami, což zabralo podstatnou část vývojového času a autor se ji učil v podstatě neustále v průběhu celého projektu.

Autorovi se podařilo splnit všechny části projektu. Pouze nebyl implementován protokol pro přenos stavových signálů ze stroje. Důvodem bylo nedodání přenosového protokolu školitelem, který pro svou velkou časovou vytíženost jej nestihl poskytnout.

Program v procesoru je implementován na principu kooperativního víceúlohového systému. Bylo by zajímavé program implementovat pomocí RTOS systému, což je zjednodušený operační systém s preemptivním plánováním. Při kooperativním víceúlohovém systému je nutné, aby se kód nikde nezastavil v čekací rutině, ale aby pokračoval bez zastavení dál. To je náročné především na ukládání stavu, ve kterém se systém v jednotlivých obsluhách právě nachází. Samotný vývoj softwaru probíhal na vývojovém kitu a mohl tak být hned přezkoušen.

Vývoj hardwaru byl mnohem složitější a vyžádal si hodně času. Do této doby měl autor zkušenosti s procesory s jádrem ATMEGA či 8051. Procesor PIC32 je mnohem sofistikovanější komponenta. Elektrické schéma bylo vytvořeno podle doporučení výrobců jednotlivých komponent a dodržet všechna doporučení bylo mnohdy velmi náročné, především co se týče rozmístění

pasivních součástek, aby nepřekročily doporučenou vzdálenost od dané komponenty. Byla použitá technologie povrchové montáže SMT. Odpory a kondenzátory mají velikost pouzder 0805. Byly použité ne zcela běžné komponenty, jako je samotný procesor PIC32MX795F512L nebo ethernetový čip DP83848C či MAX3243. Největším úskalím vývoje projektu byly enormně dlouhé objednávací doby některých komponent, což značně celý projekt zdržovalo. Další nepříjemností ze strany dodavatele byla například neshoda technického popisu objednávané součástky s došlou součástkou. Výroba plošného spoje patřila také mezi složitější úkony. Jelikož se jedná o SMT technologie, bylo nutné vzhledem k pouzdrům procesoru a ethernetového čipu volit vyšší třídu přesnosti, což výrobu prodražovalo a proto byl hledán výrobce s výhodnou cenou. To vyžadovalo patřičnou administrativu. K prodlevám v projektu přispěly i přepravní služby, které měly za úkol dopravit vyrobené plošné spoje. Přestože byla zásilka poslána jako expresní a jednalo se o vnitrostátní přepravu, byla dodána až po celých 8 dnech od data odeslání. V době psaní této práce nejsou ještě k dispozici všechny komponenty pro sestavení konečného zařízení. První prototyp byl už sestaven a odzkoušen. Byl však navržen pro komponenty, které dodavatel nedodal. Mezi nedodané komponenty patří konektor RJ45 s transformátorem, oscilační obvod, taktovací krystal a box pro zastavění plošného spoje. Po této zkušenosti byly součástky objednávány u jiného dodavatele. Ani výběr jednotlivých součástek nebyl jednoduchý a vyžádal si dlouhé hodiny vybírání z nepřeberného množství výrobců, cen a dostupností.

Původně byla zamýšlena i podpora pro bezdrátový Ethernet, ale Microchip (výrobce bezdrátového adaptéru) oznámil dobu dodání na druhou polovinu února 2011.

V praxi jsou takováto zařízení vyvíjená týmem inženýrů a využívá se firemního „know-how“ a řady již v rámci firmy hotových modulů, které se pak jenom spojují a popřípadě vylepšují. K dovedení projektu k naprosté dokonalosti by bylo třeba více času.

Literatura

[1] DI JASIO, Lucio. *Programming 32-bit Microcontrollers in C : Exploring the PIC32*. Oxford : Newnes, 2008. 527 s. ISBN 9780750687096.

[2] *Farnell : přední světový distributor elektronických součástek a produktů pro oblast údržby, oprav a renovací* [online]. 2010 [cit. 2010-12-19]. Dostupné z WWW: <http://cz.farnell.com/>.

[3] J*ak psát web, návod na html stránky : o tvorbě internetových stránek* [online]. 2010, 1 2.12.2010 [cit. 2010-12-19]. Dostupné z WWW: <http://www.jakpsatweb.cz/>. ISSN 1801-0458.

[4] *A FREE real time operating system (RTOS) for small embedded systems : The FreeRTOS Project* [online]. 2010 [cit. 2010-12-19]. Dostupné z WWW: <http://www.freertos.org/>.

[5] *Application note: AVR32817 : Getting Started with the 32-bit AVR UC3 Software Framework lwIP TCP/IP Stack* [online]. San Jose, USA : Atmel Corporation, 2010 [cit. 2010-12-19]. Dostupné z WWW: <http://www.atmel.com/dyn/resources/prod_documents/doc32147.pdf>.

[6] *Atmel Corporation : Industry Leader in the Design and Manufacture of Advanced Semiconductors* [online]. 2010 [cit. 2010-12-19]. Dostupné z WWW: <http://www.atmel.com/>.

[7] STANÍČEK, Petr. *CSS kaskádové styly : kompletní průvodce.* Vyd. 1. Praha : Computer Press, 2003. 178 s. ISBN 80-7226-872-4.

[8] *Microchip Technology Inc.* [online]. 2010 [cit. 2010-12-19]. Dostupné z WWW: <http://www.microchip.com/>.

[9] RAJBHARTI, Nilesh. *Application note: AN833 : Microchip TCP/IP Stack Application Note* [online]. U.S.A. : Microchip Technology Inc, 2008 [cit. 2010-12-19]. Dostupné z WWW: <http://ww1.microchip.com/downloads/en/AppNotes/00833c.pdf>.

[10] *SDS-Serial Device Server : UDP-Application Utility Interface* [online]. Hudson, Ohio : Quatech, Inc., 2010 [cit. 2010-12-19]. Dostupné z WWW: <www.quatech.com>.

[11] *SDS-Serial Device Server : Intellisock SDK* [online]. Hudson, Ohio : Quatech, Inc., 2010 [cit. 2010-12-19]. Dostupné z WWW: <www.quatech.com>.

[12] WOOD, Elliott. *TCP/IP Networking Part 1 : Web-Based Status Monitoring* [online]. Chandler, Arizona : Microchip Technology Inc, 2007 [cit. 2010-12-19]. Dostupné z WWW: <http://techtrain.microchip.com/webseminars/>.

[13] WOOD, Elliott. *TCP/IP Networking Part 2 : Web-Based Control* [online]. Chandler, Arizona : Microchip Technology Inc, 2007 [cit. 2010-12-19]. Dostupné z WWW: <http://techtrain.microchip.com/webseminars/>.

[14] WOOD, Elliott. *TCP/IP Networking Part 3 : Advanced Web-Based Control* [online]. Chandler, Arizona : Microchip Technology Inc, 2007 [cit. 2010-12-19]. Dostupné z WWW: <http://techtrain.microchip.com/webseminars/>.

[15] ABEL, Martin . *Plošné spoje se SMD, návrh a konstrukce.* Vyd. 1. Pardubice : Platan, 2000. 218 s. ISBN 80-902733-2-7.

[16] *GM Electronic* [online]. 2010 [cit. 2010-12-19]. Dostupné z WWW: <http://www.gme.cz/>.

[17] *DP83848C PHYTER* [online]. Santa Clara, California : National Semiconductor Corporation, 2008 [cit. 2010-12-19]. Dostupné z WWW: <http://www.national.com/ds/DP/DP83848C.pdf>.

[18] *MAX3243* [online]. Dallas, Texas : Texas Instruments Incorporated, 2004 [cit. 2010-12-19]. Dostupné z WWW: <http://focus.ti.com/lit/ds/symlink/max3243.pdf>.

[19] *PIC32MX5XX/6XX/7XX Family Data Sheet* [online]. U.S.A. : Microchip Technology Inc, 2010 [cit. 2010-12-19]. Dostupné z WWW: <http://ww1.microchip.com/downloads/en/DeviceDoc/61156F.pdf>. ISBN 978-1-60932-717-0.

[20] *Digi-Key* [online]. 2010 [cit. 2010-12-20]. Dostupné z WWW: <http://parts.digikey.com/>.

[21] *Microchip Direct* [online]. 2010 [cit. 2010-12-20]. Dostupné z WWW: <http://www.microchipdirect.com/>

[22] *I2C. In Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 2010 [cit. 2010-12-20]. Dostupné z WWW: <http://en.wikipedia.org/wiki/I%C2%B2C>.

[23] *Quatech* [online]. 2010 [cit. 2010-12-20]. Dostupné z WWW: <www.quatech.com>.

[24] *Microchip TCP/IP Stack* [online]. U.S.A. : Microchip Technology Inc, 2002 [cit. 2010-12-20]. DostupnWWW: <http://www.modtronix.com/products/sbc44ec/tcpipstk.pdf>.

Příloha A

Soupis součástek

Tabulka A.1: Soupis součástek pro jeden prototyp

Dodavatel Farnell [2]

Název	Hodnota	Objednávací číslo	Cena
C (0805)	0.1 microF (50V/10%)	1759143	0,22 Kč
C (0805)	10 microF (10V/10%)	1759421	0,82 Kč
C (0805)	11pF (50V/10%)	-	
C (0805)	20pF (50V/10%)	-	
C (1206)	4.7 microF (16V/10%)	1759427	0,57 Kč
C (0805)	1 microF (16V/10%)	1759479	0,85 Kč
C tantal	6.8 microF (20V/10%)	-	
C (0805)	6.8 microF (10V/10%)	-	
R (0805)	49.9 R	1501925	21,17 Kč
R (0805)	250 R	-	
R (0805)	33 R	9333088	0,73 Kč
R (0805)	2.2 k	9332812	0,73 Kč
R (0805)	510 R	9333312	0,73 Kč
R (0805)	1.5 k	9332600	0,73 Kč
R (0805)	4.87 k	1469925	0,73 Kč
R (0805)	4.7 k	9333266	0,73 Kč
R (0805)	1 k	9332383	0,73 Kč
R (0805)	330 R	9333037	0,73 Kč
R (0805)	100 K	9332405	0,73 Kč
R (0805)	470 R	9333258	0,73 Kč
R (0805)	100 R	9332375	0,73 Kč
R (0805)	22 R	9332847	0,73 Kč
R (0805)	240 R	9332871	0,73 Kč
IO	MAX3243EIDW	1053595	35,39 Kč
J	CONN_MAGJACK_SI-60062-F	1572195	138,38 Kč
IO	TC1262-2.5_SOT223 (3.3V)	1296594	53,71 Kč
LED	LED, 3MM, ULTRA-GRN	1142523	1,86 Kč
CON	TERMINAL BLOCK, 8WAY	1761538	58,78 Kč
CON	HEADER, 3.5MM, HORIZ	1761546	20,99 Kč
IO	LM1117MPX-5.0	1652314	22,19 Kč
IO	PIC32M795F512L	1778491	268,87 Kč
IO	DP83848C	1286817	118,16 Kč
BUTTON	Button	1555987	3,66 Kč
IO	Rezonator 50 Mhz	1640953	87,87 Kč

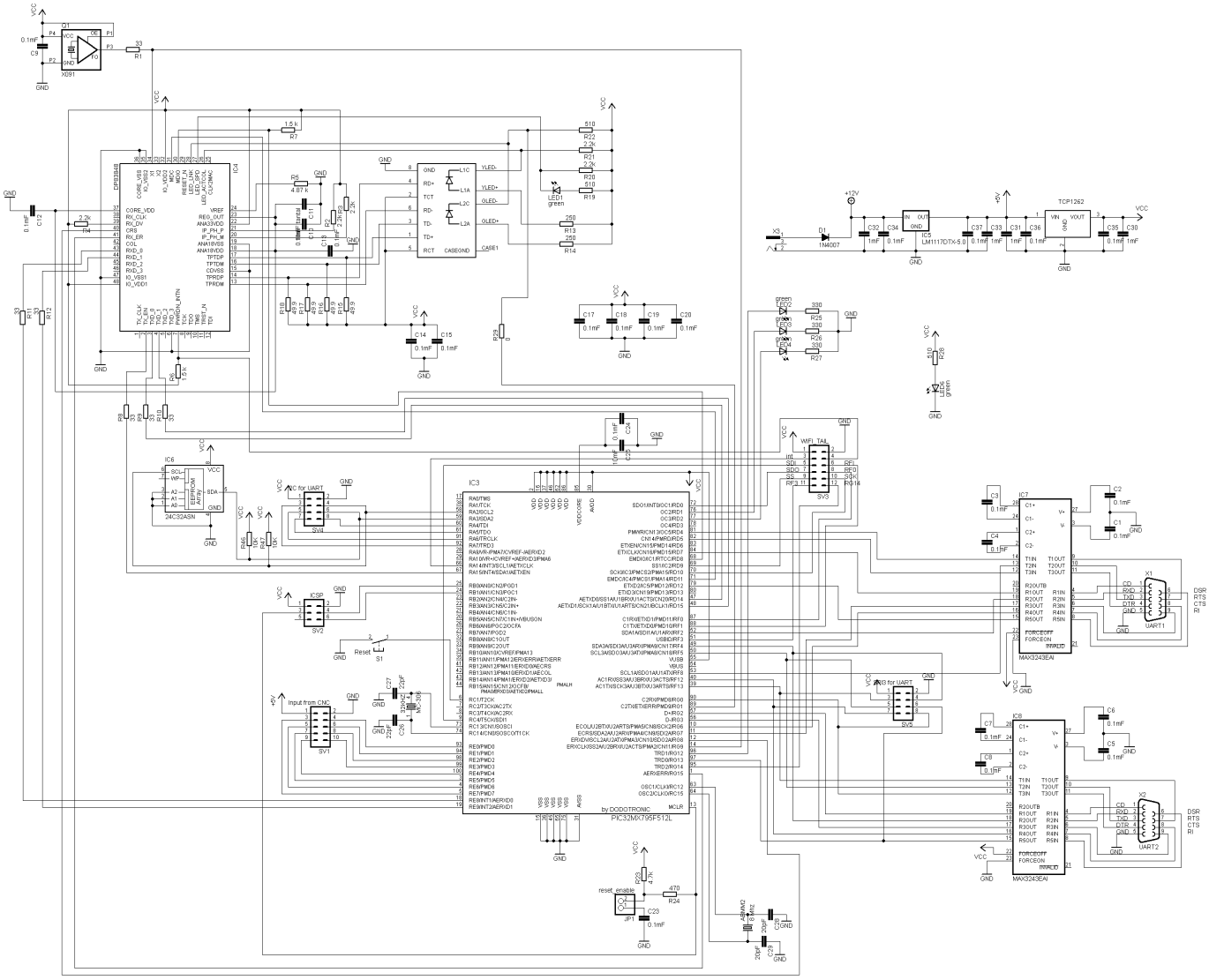
Dodavatel GME [16]

CONN	SUPPLY CONNECTOR (DJ05B)	806-049	5,40 Kč
CONN	CANON 9 PIN	801-037	9,60 Kč
Relle	Bez diody – RR1A24-2K	634-150	26,80 Kč

Příloha B

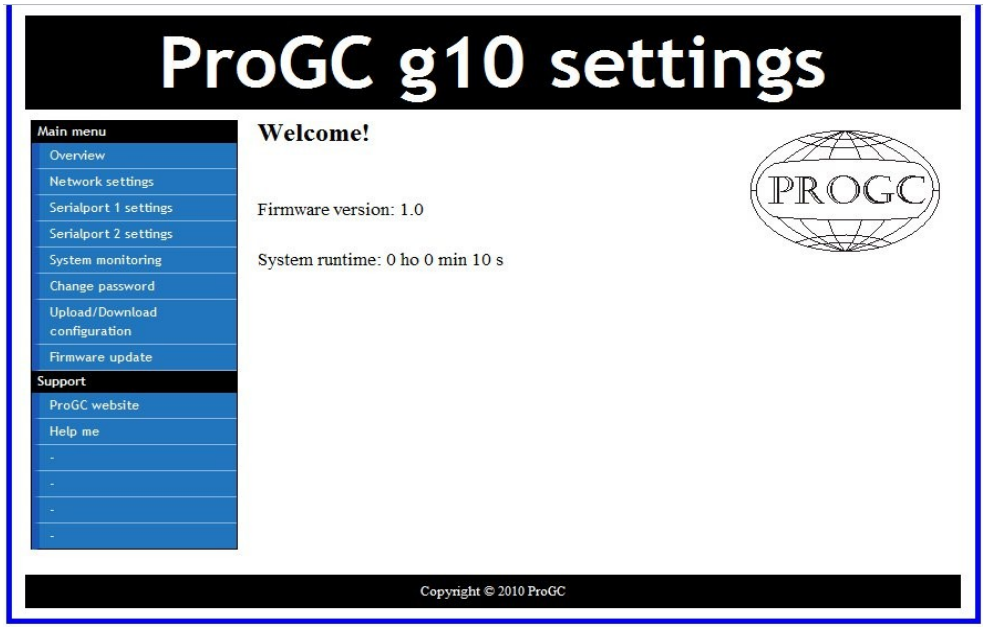
Elektrické schéma

Obrázek B.1: Elektrické schema zapojení

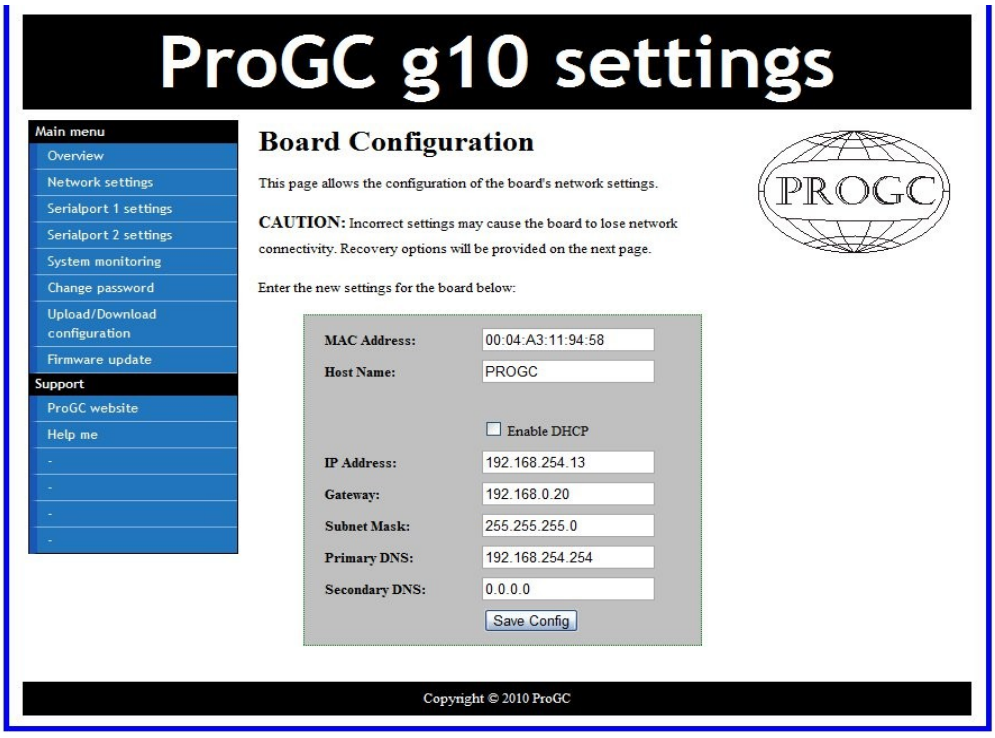


Příloha C

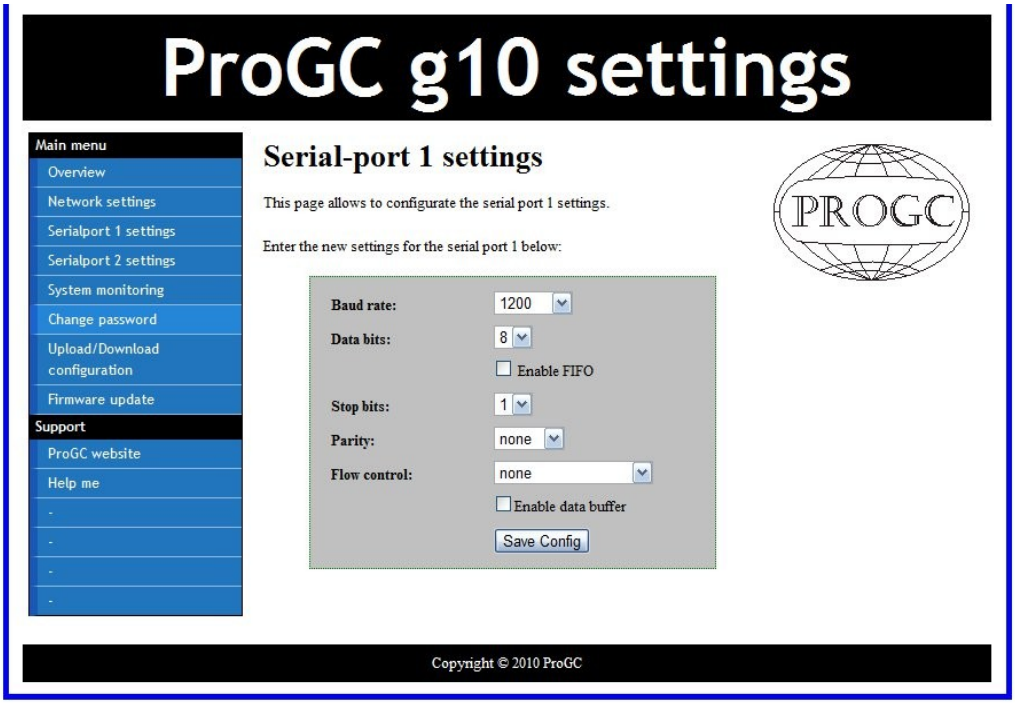
Obrazová příloha



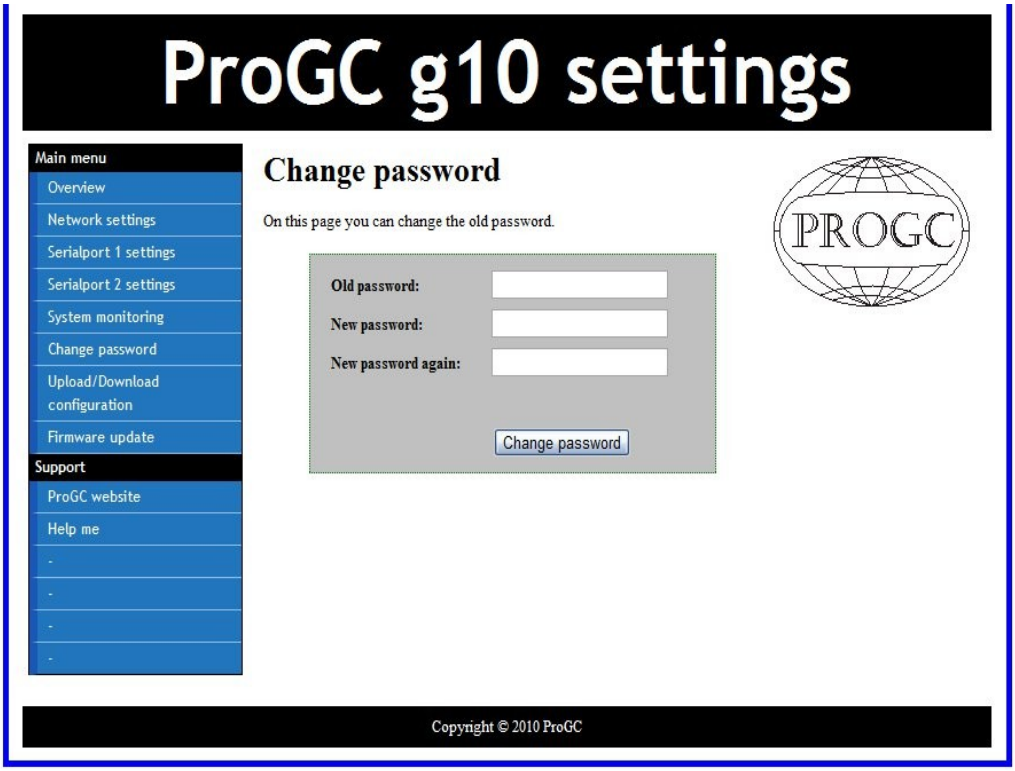
Obrázek C.1: Úvodní stránka webového rozhraní



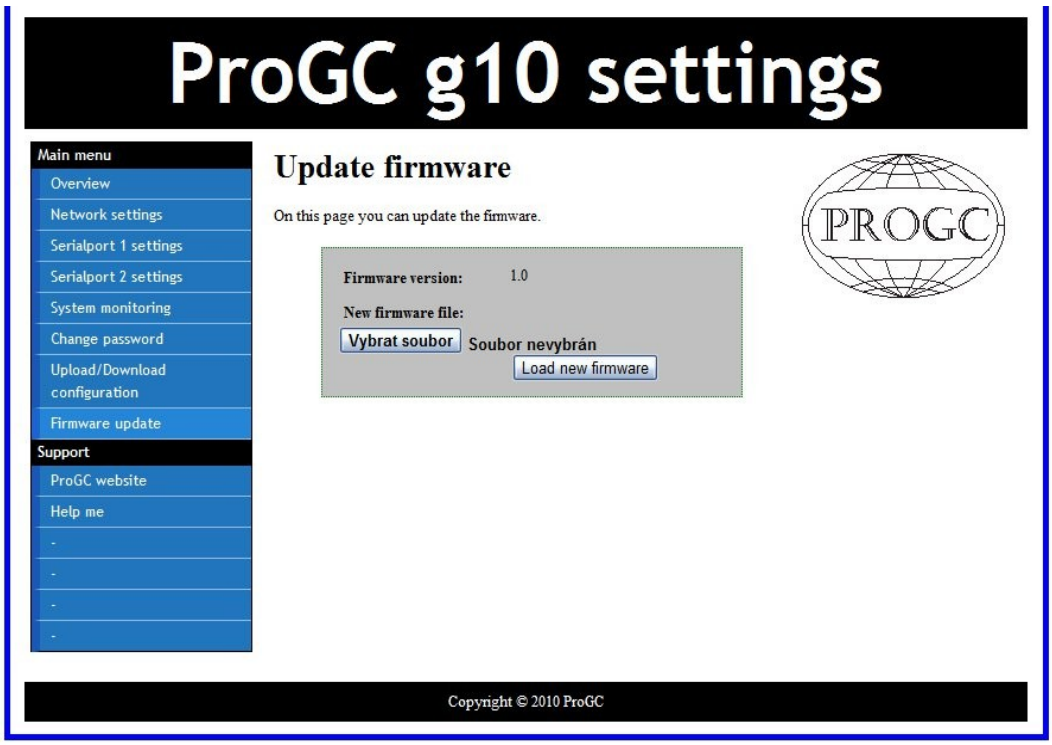
Obrázek C.2: Nastavení síťového rozhraní



Obrázek C.1: Nastavení sériového portu



Obrázek C.1: Změna hesla



Obrázek C.1: Nahrání nového firmwaru

Příloha D

Obsah přiloženého CD

Adresář	Obsah
SW/TCPIP	Software pro PIC32
SW/WebPages	Zdrojové kódy webového rozhraní
HW	Schema zařízení a podklady pro výrobu DPS v Eagle 5.8.0
DOC	Tento dokument ve formátu pdf