VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV RADIOELEKTRONIKY

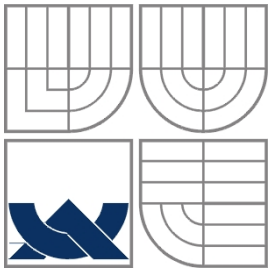
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF RADIO ELECTRONICS

ŘÍDÍCÍ A MONITOROVACÍ JEDNOTKA PRO HLAVICI OPTICKÉHO SPOJE

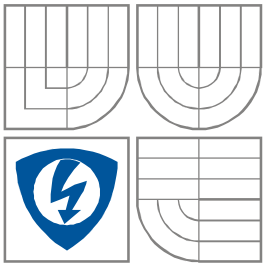
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. MARTIN OVČÁČEK
AUTHOR

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV RADIOELEKTRONIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF RADIO ELECTRONICS

ŘÍDÍCÍ A MONITOROVACÍ JEDNOTKA PRO HLAVICI OPTICKÉHO SPOJE

CONTROL AND MONITORING UNIT FOR OPTICAL LINK STATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. Martin Ovčáček
AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. Aleš Prokeš, Ph.D.
SUPERVISOR

BRNO, 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav radioelektroniky

Diplomová práce

magisterský navazující studijní obor

Elektronika a sdělovací technika

Student: Ovčáček Martin, Bc. ***ID:*** 89513

Ročník: 2 ***Akademický rok:*** 2007/2008

NÁZEV TÉMATU:

Řídící a monitorovací jednotka pro hlavici optického spoje

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s rozhraním Ethernet a prostudujte používané protokoly pro přenos dat. Navrhněte mikroprocesorový systém pro monitorování a nastavování analogových veličin (teplota, úroveň přijímaného signálu, výkon vysílače, apod.) umožňující komunikaci po internetu. Při návrhu preferujte mikroprocesory rady 8051 (Silicon Laboratories, Atmel, apod.). Vyberte vhodný typ s ohledem na dostupná vývojová prostředí. Pro návrh řídící jednotky je možné využít i některé dostupné mikroprocesorové a komunikační moduly.

Vytvořte softwarové moduly pro obsluhu Ethernetového řadiče, monitorování napětí na analogových vstupech mikroprocesoru, komunikaci po sběrnicích I2C a SPI a obsluhu periferií (displej, klávesnice). Vytvořte komunikační software pro vzdálené řízení modulu a pro přenos dat prostřednictvím internetu a ověřte jeho správnou činnost.

DOPORUČENÁ LITERATURA:

[1] KÁLLAY, F., PENNIAK, P. Počítačové sítě a jejich aplikace. Praha: GRADA, 2003.

[2] BLUMM, R. C# Network Programming. Chichester: John Willey & Sons, 2002.

[3] SKALICKÝ, P. Mikroprocesory řady 8051. Praha: BEN - technická literatura, 1998.

Termín zadání: 5.10.2007 ***Termín odevzdání:*** 30.5.2008

Vedoucí práce: doc. Ing. Aleš Prokeš, Ph.D.

prof. Dr. Ing. Zbyněk Raida
předseda oborové rady

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami:

1. Pan/paní

Jméno a příjmení:	Martin Ovčáček
Bytem:	Vrchlického 1124, Hranice, 753 01
Narozen/a (datum a místo):	18. března 1984 v Hranicích

(dále jen „autor“)

a

2. Vysoké učení technické v Brně

Fakulta elektrotechniky a komunikačních technologií
se sídlem Údolní 53, Brno, 602 00
jejímž jménem jedná na základě písemného pověření děkanem fakulty:
prof. Dr. Ing. Zbyněk Raida, předseda rady oboru Elektronika a sdělovací technika
(dále jen „nabyvatel“)

Čl. 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):

   ☐ disertační práce
   ☒ diplomová práce
   ☐ bakalářská práce
   ☐ jiná práce, jejíž druh je specifikován jako .....
   (dále jen VŠKP nebo dílo)

Název VŠKP:	Řídící a monitorovací jednotka pro hlavici optického spoje
Vedoucí/ školitel VŠKP:	doc. Ing. Aleš Prokeš, Ph.D.
Ústav:	Ústav radioelektroniky
Datum obhajoby VŠKP:	_____

   VŠKP odevzdal autor nabyvateli*:

   ☒ v tištěné formě – počet exemplářů: 2
   ☒ v elektronické formě – počet exemplářů: 2

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.

3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.

4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

* hodící se zaškrtněte

Článek 2

Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizovaní výpisů, opisů a rozmnoženin.

2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.

3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti

   - ☒ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)

4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3

Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.

2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.

3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.

4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: 30. května 2008

…….. ……

Nabyvatel Autor

Abstrakt

Náplní Diplomové práce „Řídící a monitorovací jednotka pro hlavici optického spoje“ je návrh a realizace hardwaru a softwaru jednotky, která bude umožňovat vzdálené řízení a přenos dat prostřednictvím sítě internet. Pro potřeby hlavice optického spoje je požadováno, aby jednotka umožňovala monitorování a nastavování analogových veličin pomocí webového rozhraní (webových stránek). Obsahuje popis rozhraní Ethernet a použitých síťových protokolů. Hardware jednotky tvoří mikrokontrolér C8051F120 a ethernetový kontrolér CP2200 firmy Silicon Laboratories.

Klíčová slova

Ethernet, 10BASE-T, CP2200, linkový rámec, datagram, segment, ARP, IP, ICMP, TCP, UDP, HTTP

Abstract

The aim of Master's thesis "Control and monitoring unit for optical link station" is design of hardware and software of unit, which provides remote control and data transfer through Internet. It is required monitoring and setting analog parameters of optical link station by web service. It includes description of Ethernet and net protocols. The core of unit are microcontroller C8051F120 and ethernet controller CP2200 made by Silicon Laboratories.

Key words

Ethernet, 10BASE-T, CP2200, link frame, datagram, segment, ARP, IP, ICMP, TCP, UDP, HTTP

Bibliografická citace

OVČÁČEK, M. *Řídící a monitorovací jednotka pro hlavici optického spoje.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2008. 80 s. Vedoucí diplomové práce doc. Ing. Aleš Prokeš, Ph.D.

Prohlášení

Prohlašuji, že svou diplomovou práci na téma Řídící a monitorovací jednotka pro hlavici optického spoje jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne 30. května 2008 .....
podpis autora

Poděkování

Děkuji vedoucímu diplomové práce doc. Ing. Aleši Prokešovi, Ph.D. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce.

V Brně dne 30. května 2008 .....
podpis autora

Obsah

Seznam obrázků:

Seznam tabulek:

1 Úvod

Náplní Diplomové práce „Řídící a monitorovací jednotka pro hlavici optického spoje“ je návrh hardwaru a softwaru jednotky, která bude umožňovat vzdálené řízení a přenos dat prostřednictvím sítě internet. Pro potřeby hlavice optického spoje je požadováno, aby jednotka umožňovala monitorování a nastavování analogových veličin (teplota, úroveň přijímaného signálu, výkon vysílače, apod.) přes internet, resp. pomocí webového rozhraní.

Diplomová práce bude rozdělena na kapitoly, kde každá kapitola bude řešit určitou část problematiky. Nejprve bude stručně popsáno rozhraní Ethernet, na kterém je síťová komunikace založena, dále budou popsány síťové protokoly umožňující přenos dat, dále bude navržen hardware jednotky podle požadavků zadání a nakonec budou popsány softwarové moduly a vytvořený funkční software jednotky. Hlavice optického spoje není náplní této práce, a proto nebude nijak popisován. Jednotka sice bude určena pro hlavici optického spoje, nicméně tato skutečnost nemá na funkce jednotky ani na principy komunikace žádný vliv.

První část práce bude věnována stručnému popisu rozhraní Ethernet. Budou popsány zejména verze Ethernetu a struktura linkového rámce. Detailnější popis nebude pro potřeby jednotky nutný.

V další části práce budou popsány protokoly pro přenos dat prostřednictvím sítě internet. Protokoly budou rozděleny do příslušných vrstev síťového modelu TCP/IP. Budou popsány pouze protokoly, které jednotka bude používat.

Poté již bude řešen hardware jednotky. Požadováno je použití mikrokontroléru C8051F120 a ethernetového kontroléru CP2200, oba firmy Silicon Laboratories. Bude k dispozici vývojový modul s uvedeným mikrokontrolérem. Bude navržen modul s řadičem CP2200 pro připojení k dodanému modulu.

Poslední část práce bude věnována softwaru pro navržený hardware. Software bude rozdělen do SW modulů, které budou použity pro konečný software jednotky. Softwarové moduly budou popsány, popis se bude týkat parametrů funkcí a jejich použití. Konečný software bude rovněž popsán, popis se však bude týkat spíše algoritmů a hlavních myšlenek programu.

Nejdůležitějším výstupem práce budou zdrojové texty softwarových modulů a konečného funkčního softwaru. Tyto zdrojové texty budou k dispozici na přiloženém CD.

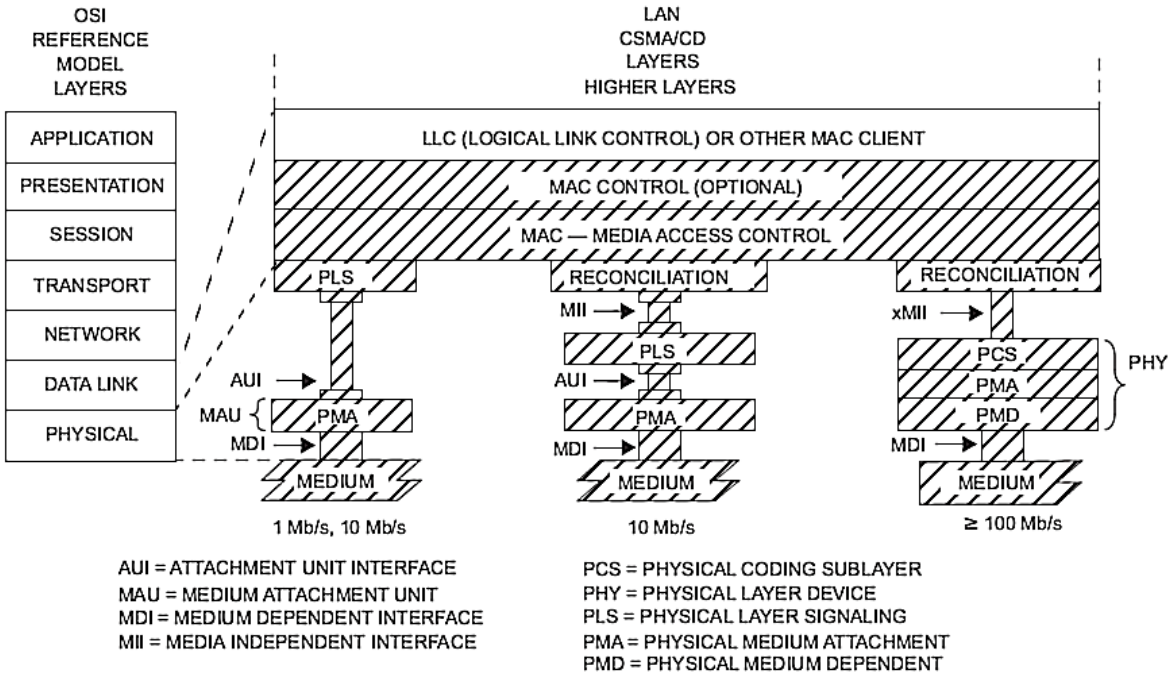
2 Ethernet (IEEE802.3)

Ethernet/IEEE 802.3 technologie jsou v současnosti nejpoužívanější síťové technologie pro LAN. Norma IEEE 802.3 specifikuje fyzickou a linkovou vrstvu síťového modelu ISO OSI (viz kap. 3).

Ve fyzické vrstvě norma definuje vlastnosti přenosového média (impedance, kapacity, útlum, přeslech, apod.), použité úrovně, kódování, přenosové rychlosti, apod. Vlastnosti se liší v závislosti na standardu (10Base-T, 100Base-FX, atd.).

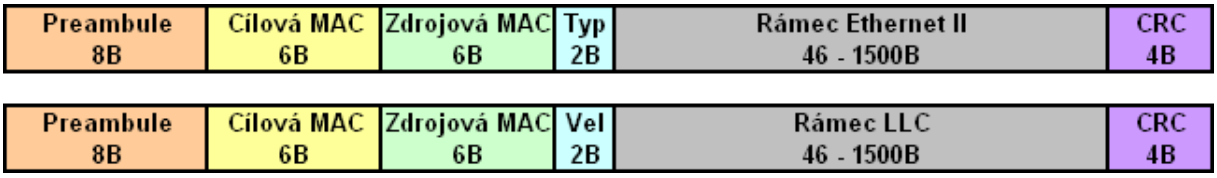
Linková vrstva má pak podle IEEE802.3 definovány přístupové metody k médiu a definuje formát rámců.

Mezi jednotlivými vrstvami norma přesně definuje rozhraní, přes které vrstvy, resp. jejich dílčí součásti kooperují. Architektura IEEE802.3 je na Obr. 1.



Obr. 1 Referenční model IEEE802.3 (převzato z [10]).

Ethernet (mimo 10Gb/s) používá přístupovou techniku CSMA/CD (Carrier-Sense Multiply Access with Collision Detection). CSMA/CD technika je založena na náhodném přístupu uzlů k médiu. Všechny kolize, které mohou nastat (současné vysílání z více uzlů), jsou vyřešeny postupy CSMA/CD. V plně duplexním provozu (kolize nemůže nastat) již není tato metoda potřebná, a proto se nepoužívá. To je případ použití přepínačů (switch) namísto dříve používaných rozbočovačů, opakovačů (hub).



Obr. 2 Struktura rámce Ethernet II a LLC.

Obr. 2 ukazuje strukturu linkového rámce definovaného IEEE802.3. Prvních osm bajtů tvoří tzv. preambule. Ta slouží k synchronizaci a přesné detekci začátku dat. Za preambulí následuje šest bajtů fyzické adresy (MAC adresa) cíle, dále šest bajtů MAC adresy odesilatele. Následuje dvoubajtové pole, které udává typ/velikost rámce. Má-li toto pole hodnotu větší než 1500, jde o typ rámce (IP datagram, ARP, apod.), potom o rámci hovoříme jako o rámci Ethernet II. Pokud je hodnota pole menší nebo rovno 1500, jedná se o velikost rámce, pak hovoříme o rámci jako Ethernet IEEE802.3. Struktura dat by potom měla odpovídat protokolu logické vrstvy LLC IEEE802.3.

Minimální délka rámce bez preambule je 64B, samotná data rámce tedy musejí být minimálně 46B. Maximální velikost dat je 1500B.

Fyzická (MAC) adresa:

MAC adresa je šestibajtová. Nejnižší dva bity prvního bajtu mají zvláštní význam. Bit0 udává, zda se jedná o *Unicast* nebo *Multicast* adresu, Je-li nastaven na 1, je to *Multicast* adresa. Bit1 udává, zda je adresa celosvětově unikátní, nebo je přidělená v rámci jedné společnosti. Je-li tento bit nulový, jedná se o globálně unikátní adresu. Jinak první tři bajty MAC adresy udávají výrobce, další tři pak výrobní číslo daného výrobce.

Např.: **00-0b-3c-02-97-5d** (MAC adresa řadiče CP2200)
00-0b-3c: Cygnal Integrated Products, Inc. (Silicon Laboratories).
02-97-5d: výrobní číslo.

Bit0 prvního bajtu je 0 – unicast, Bit1 je 0 – celosvětově unikátní adresa.

2.1 Ethernet 10Mb/s

2.1.1 10Base-5

Jedná se o původní Ethernet na koaxiálním kabelu. Sběrnici tvoří koaxiální kabel o impedanci 50 Ω, ke které se jednotlivé stanice připojují pomocí speciálních tranceiverů a AUI kabelů.

2.1.2 10Base-2

Ethernet na tenkém koaxiálním. Koaxiální kabel tvoří sběrnici, ke které se připojují jednotlivé stanice přímo. Kabel je impedance 50 Ω (RG-58) nesmí mít žádné odbočky a je na koncích zakončen odpory 50 Ω (tzv. terminátory).

2.1.3 10Base-T

10Base-T je definován normou IEEE802.3i. Jako přenosové médium používá kroucenou dvoulinku. Využívá UTP (Unshielded Twisted Pair) kabely, kde využívá dva páry ze čtyř. Dnes již překonaná síť, která byla ve většině případů nahrazena rychlejší 100 Mbit/s variantou.

10Base-T definuje maximální délku UTP kabelu na 100m (90m pevné a 2x5m pohyblivý kabel pro připojení zařízení). Používá kódování Manchester.

2.1.4 10Base-F

Jako přenosové médium se používá optické vlákno. Používá se pro spojení na větší vzdálenost, nebo spojení mezi objekty, kde nelze použít kroucenou dvoulinku. Tvořila obvykle tzv. pateřní síť, která propojuje jednotlivé menší celky sítě. Dnes je již nahrazována vyššími rychlostmi (Fast Ethernet, Gigabit Ethernet).

2.2 Ethernet 100Mb/s

Dnes velmi rozšířený standard. Označuje se jako Fast Ethernet. Je založen na efektivnějším využití přenosového média (změnou kódování).

2.2.1 100Base-TX

Fast Ethernet, používá dva páry UTP nebo STP kabelu kategorie 5. Pro kódování je použit kód 4B5B doplněný o převod na MLT-3 (Multi-Level Transmit , 3 stavy).

2.2.2 100Base-T4, T2

Standard, který používá UTP kabely kategorie 3, bylo tedy možné jej použít pro již vybudované sítě strukturované kabeláže kat. 3 (staré rozvody v budovách). Bylo tedy možné přejít z 10Mb/s na 100Mb/s bez nutnosti měnit kabeláž na kat. 5.
100Base-T4 používal tři páry k přenosu dat a jeden pár pro detekci kolizí, 100Base-T2 dokonce vystačil již jen se standardními dvěma páry s využitím kódování PAM-5 (5-level Pulse Amplitude Modulation).

2.2.3 100Base-FX

Fast Ethernet používající dvě optická vlákna. Použité kódování je stejně jako u 100Base-TX kódování 4B5B.

2.3 Ethernet 1Gb/s, 10Gb/s

Poslední dobou již i 100Mb/s přestává stačit, a tak byly vytvořeny standardy pro 1Gb/s (gigabitový ethernet) 1000Base-T, 1000Base-CX používající metalická vedení, a pro optické kabely 1000Base-LX a 1000Base-SX, pro sítě 10Gb/s to jsou standardy 10GBase-T, 10GBase-CX4 pro metalická vedení a 10GBase-LX4 pro optická vedení.

Gigabitový ethernet IEE802.3z už mírně mění linkový rámec. Je to z důvodu detekce kolize, linkový rámec je prodloužen, aby doba vysílání rámce byla větší, než doba potřebná k překlenutí maximální vzdálenosti.

Desetigigabitový ethernet IEEE802.3ae již podporuje pouze plně duplexní provoz, a tak přístupová metoda CSMA/CD již není použita.

2.3.1 1000Base-T, CX

Standardy pro sítě s rychlostí 1Gb/s, používající metalická vedení. Standard *–T* využívá kabely UTP/FTP (4 páry) a pro kódování používá kód PAM-5. Standard *–SX* je určen na krátkou vzdálenost, používá kabel TWINAX (kroucený dvoudrát, stíněný), kódování 8B10B.

2.3.2 1000Base-LX, SX

Standardy sítě 1Gb/s využívající optická vlákna. Standard *-SX* používá vícevidové optické vlákno 50/125 µm nebo 62.5/125 µm a vlnová délka je 850 nm, standard *-LX* pak na vlnové délce 1310 nm. Kódování je u obou standardů 8B10B.

Existují ještě nestandardní rozhraní 1000BASE-ZX, LH, LLX., které na vlnové délce 1550 nm umožňují překlenout vzdálenost až 70km.

2.3.3 10GBase-L, CX4, LX4

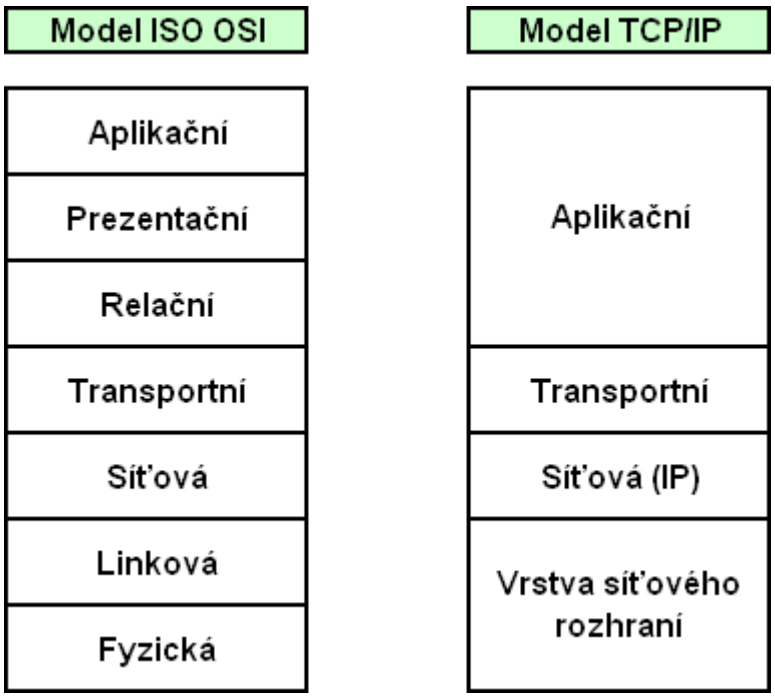
Standardy sítě 10Gb/s. Standard *-L* využívá kabely kategorie 7 pro 100m vzdálenost (kat. 6 pro kratší vzdálenosti, asi 50m) a kód PAM-16, standard *-CX4* využívá čtyři kabely TWINAX do vzdálenosti 15m. Standard *–LX* využívá čtyři optická vlákna na vlnové délce 1310nm.

3 Síťové protokoly

Síťové protokoly se dělí do síťových vrstev. Existují dva základní modely (viz Obr. 3):

- **Model ISO OSI**: Obsahuje sedm vrstev: Aplikační, Prezentační, Relační, Transportní, Síťovou, Linkovou, Fyzickou. Tento model nespecifikuje žádné služby, jedná se pouze o obecnou architekturu, jeho význam v poslední době ustupuje a je používán spíše jako zdroj terminologie.

- **Model TCP/IP**: Jedná se o síťový model, který vznikl jako konkurence modelu ISO OSI. Model TCP/IP je jednodušší a mnohem používanější. Obsahuje pouze čtyři vrstvy: Aplikační, Transportní, Síťovou, Vrstvu síťového rozhraní. Pro popis síťových protokolů bude použit právě model TCP/IP.



Obr. 3 Síťové modely ISO OSI a TCP/IP.

Budou popsány pouze protokoly využité pro Řídící a monitorovací jednotku.

3.1 Vrstva síťového rozhraní

Vrstva síťového rozhraní je nejnižší vrstva modelu TCP/IP. Tato vrstva sdružuje Fyzickou a Linkovou vrstvu modelu ISO OSI. V této vrstvě je řešeno vše co se týká použité přenosové cesty a vysílání a příjmu paketů (rámců). Blíže není tato vrstva specifikována, protože je závislá na použité technologii (Ethernet, PPP, FDDI, Token ring).

3.2 Síťová vrstva

Síťová vrstva již není závislá na přenosové technologii.

Na rozdíl od vrstvy linkové, kdy bylo možné data dopravit jen v rámci lokální sítě (LAN), ve vrstvě síťové lze data dopravovat mezi libovolnými dvěma stanicemi (mezi libovolnými dvěma počítači v Internetu) přes mnohé LAN.

Data jsou od odesilatele k příjemci dopravována přes směrovače (routery), přičemž může existovat i více cest. Není zaručeno, že několik za sebou odeslaných datagramů bude procházet stejnými směrovači.

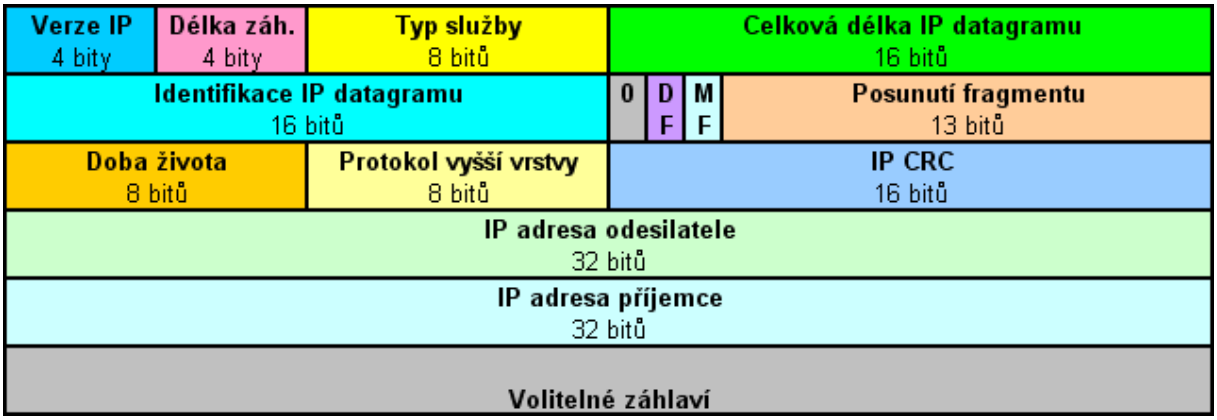
Síťová vrstva poskytuje nespolehlivou datagramovou službu, tzn., že není zaručeno doručení datagramu, není zaručeno ani pořadí datagramů.

Síťová vrstva je často označována jako vrstva IP. To proto, že je založená prakticky na jednom protokolu – protokolu IP. Ve skutečnosti do této vrstvy patří ještě další protokoly. Jsou to:

- protokol ARP, RARP,
- protokol ICMP,
- protokol IGMP.

3.2.1 Protokol IP

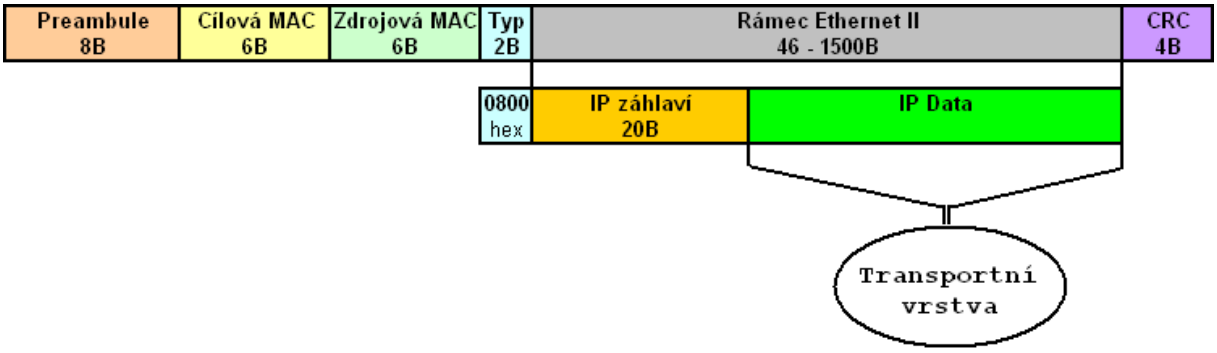
Pokud hovoříme o přenášení dat protokolem IP, pak hovoříme o IP datgramu. IP datagram se skládá z hlavičky IP a přenášených dat. Hlavička má obvykle 20B, avšak není to podmínkou. Může být i větší, vždy však v násobcích 4 bajtů (maximálně 60B viz *Délka záhlaví*). Struktura IP hlavičky je znázorněna na Obr. 4. Stručný popis je uveden v Tab. 1. Protokol IP je popsán normou *RCF 791*.



Obr. 4 Hlavička IP datagramu.

Na Obr. 5 je znázorněno začlenění IP datagramu do linkového rámce. Pole *Typ* v linkovém rámci je nastaveno na hodnotu *0x0800*, což je typ odpovídající IP datagramu. Data rámce Ethernet II tedy tvoří IP datagram. Síťová vrstva prohledá hlavičku a podle protokolu vyšší vrstvy rozhodne, kam má data z datagramu předat, v případě fragmentů

nejprve složí celý datagram. V případě protokolu vyšší vrstvy ICMP jsou data určena přímo vrstvě síťové (např. odpověď příkazu *ping*, nebo zpráva od směrovače, apod.). Pokud je protokol vyšší vrstvy TCP nebo UDP, jsou data z datagramu předána vyšší vrstvě – vrstvě transportní.



Obr. 5 Začlenění IP datagramu do linkového rámce.

Hlavička IP detailně:

Verze IP: Pole *verze IP* je čtyřbitové, udává přímo číslo verze. V dnešní době jsou dvě verze. Verze 4 – velmi rozšířená verze, většina směrovačů je vyrobena pro verzi 4, proto přejít na jinou by bylo velmi zdlouhavé a finančně náročné. Nicméně 4 bajty IP adresy, jak tomu ve verzi 4 je, již dávno přestaly stačit, a tak se vyvinula další verze IP – verze 6. Ta sice začíná pronikat do internetu, některé LAN již mohou na verzi 6 fungovat, nicméně jak bylo uvedeno výše, přechod na verzi 6 je velmi zdlouhavá záležitost. Verze 6 má IP adresu 16-tibajtovou a hlavička má jinou strukturu než verze 4. Pole *Typ* v linkovém rámci je pro IPv6 nastaveno na *0x86DD*. Další popis se bude týkat IPv4.
Délka záhlaví: Toto pole je rovněž čtyřbitové a udává velikost IP hlavičky v násobcích 4 bajtů. Standardní (a také minimální) velikost hlavičky je 20B, pole bude mít tedy hodnotu 5. Pokud by hlavička IP byla rozšířená o volitelné záhlaví, může být hodnota pole vyšší, maximálně však 15, tedy celá IP hlavička nemůže být větší jak 60B.
Typ služby: Toto pole (1B) mělo být využito pro identifikaci služby, pro kterou je datagram určen. Podle tohoto pole by pak směrovače mohly některé služby přeposlat prioritně. Normy *RFC791* a *RFC1349* využití specifikovaly, nicméně se to v praxi nezavedlo – z obavy, že by bylo toto pole zneužito a vyplněno pevně na takovou hodnotu, která by zaručovala nejvyšší prioritu, pak by toto pole ztrácelo svůj význam. Hodnota tohoto pole bývá *0x00*.
Délka datagramu: Dvoubajtové pole udávající celkovou délku IP datagramu, tedy velikost hlavičky plus velikost dat. Dva bajty předurčují, že IP datagram může mít velikost maximálně 65535B.
Identifikace datagramu: Dvoubajtová položka udávající jakési pořadové číslo datagramu. Je využívána při fragmentaci (společně s následujícími položkami *MF* a *Posun fragmentu*). Pokud je datagram rozdělen na fragmenty, mají tyto stejné identifikační číslo.
Příznaky: Tři bity: první je vždy 0, druhý *DF* – Don't Fragment – fragmentace zakázána, třetí bit *MF* – More Fragments – další fragment. Je-li *DF* nastaven na 1, pak je fragmentace zakázána, pokud by takový datagram dorazil na směrovač, který by chtěl defragmentovat, pak by takový datagram zahodil a protokolem ICMP by tuto skutečnost oznámil. Pokud je bit *MF* nastaven na 1, udává, že se nejedná o poslední fragment, tzn., že ještě minimálně jeden fragment dorazí. Dále to znamená, že pole posunutí fragmentu je platné.

Posunutí fragmentu: 13-tibitové pole, které udává ofset v datagramu, kam aktuální fragment patří.
Doba života: Time To Live (TTL), jednobajtové pole, které udává, kolika směrovači ještě datagram může projít. Každý směrovač hodnotu pole zmenšuje (o 1), až je toto pole 0 směrovač datagram zahodí (protokolem ICMP o tom může informovat). TTL bývá nastavována u odesilatele na hodnotu 128 (OS Windows) nebo 64.
Protokol vyšší vrstvy: Jednobajtové pole, které udává protokol, který je obsažen v datech datagramu. Nejzákladnější jsou TCP (0x06), UDP (0x17) a ICMP (0x01).
Kontrolní součet: Toto dvoubajtové pole obsahuje kontrolní součet počítaný z hlavičky IP datagramu. Algoritmus výpočtu je dán normami RFC1071 a RCF1141.
IP odesilatele: Čtyřbajtová síťová (IP) adresa odesilatele (Ipv4).
IP příjemce: Čtyřbajtová síťová (IP) adresa příjemce (Ipv4).
Volitelné záhlaví: Toto pole musí být v násobcích 4 bajtů. V praxi se nepoužívá, protože směrovače datagramy s volitelným záhlavím zahazují.

Stručné shrnutí významu jednotlivých polí hlavičky IP datagramu společně s typickými hodnotami je uvedeno v Tab. 1.

Tab. 1 Stručný popis hlavičky IP datagramu.

Název pole	typ. hodnota	Popis
Verze IP	**4**	Verze protokolu IP
Délka záhl.	**5**	Délka záhlaví v násobcích 4 (stejně jako u TCP)
Typ služby	**0**	nepoužívá se. (RFC791, RFC1349)
Celk. délka		celková délka (hlavička+data)
Identifikace		identifikace IP datagramu, mj. se vzužívá mechanismem fragmentace
Příznak 0	**0**	Tento bit je vždy nastaven na 0
Příznak DF	**1**	DF..Don'tFragment(fragmentace zakázána)
Příznak MF	**0**	MF..MoreFragments(další fragmenty)
Posun frag.	**0**	ofset fragmentu
Doba života	**128 nebo 64**	Doba života IP datagramu (TTL)
Protokol	**0x01**	ICMP
	0x06	TCP
	0x17	UDP
IP CRC		kontrolní součet (RFC1071, RFC1141)
IP odesilatel		IP adresa odesilatele
IP příjemce		IP adresa příjemce
Volitelné		nepoužívá se. (směrovače tyto datagramy zahazují)

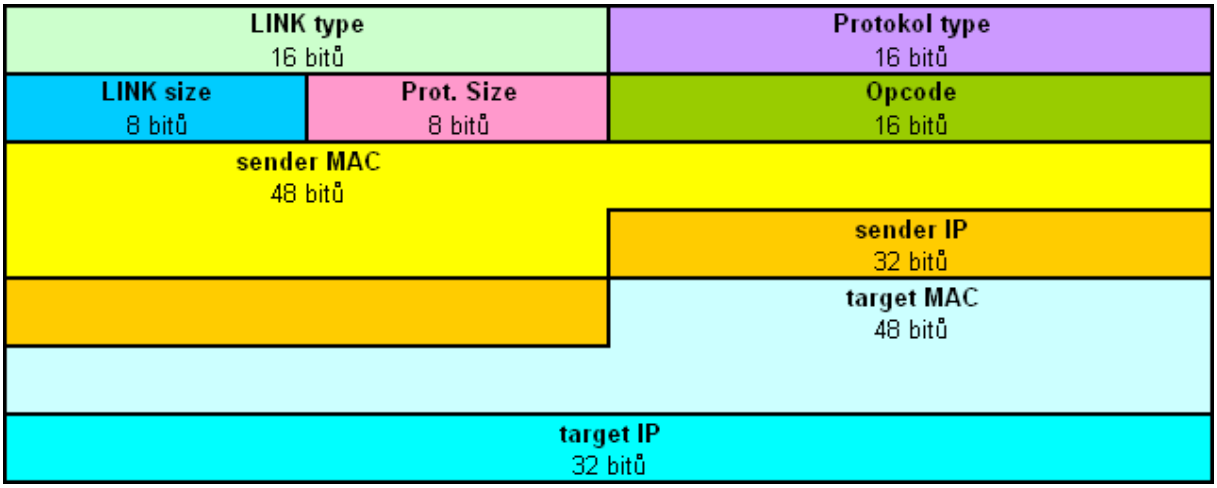
3.2.2 Protokol ARP

Pro komunikaci na LAN nestačí znát IP adresu cíle, ale je potřeba znát i MAC adresu cíle, aby bylo možné odeslat linkový rámec na dané místo. Protokol ARP je definován normou *RCF 826*.

Protokol ARP slouží ke zjištění fyzické (MAC) adresy cíle z jeho síťové (IP) adresy. Struktura protokolu ARP je na Obr. 6.

Linkový rámec nesoucí protokol ARP je na Obr. 7. Pole *Typ* v linkovém rámci je pro protokol ARP vyplněno hodnotou 0x0806. Protože pro linkový rámec Ethernet II a IPv4 je

velikost ARP protokolu 28 bajtů, je do linkového rámce doplněn trailer velikosti 18 bajtů obsahující 0, tak má linkový rámec velikost 64B, tedy minimální možnou. Pozn.: Je možné odesílat i linkové rámce menší než 64B (tzv. *Runt Packet*) není však zaručeno, že je budou všechny stanice podporovat.



Obr. 6 Struktura protokolu ARP.

Pro zjištění MAC adresy z IP se rámec pošle jako *Broadcast* (MAC cíle je FF:FF:FF:FF:FF:FF). Tímto je zaručeno, že rámec přijmou všechny stanice na LAN. Pole *Typ* je vyplněno hodnotou 0x0806, jedná se tedy o protokol ARP. Příjemce ze struktury ARP vyčte údaje a testuje, zda je požadavek na jeho IP. Pokud ne, rámec ignoruje, pokud ano, vytvoří nový rámec s odpovědí a odešle jej na MAC adresu tazatele.

Obr. 7 Linkový rámec nesoucí protokol ARP.

Opcode pro dotaz na MAC adresu z IP adresy má hodnotu 1, odpověď má hodnotu 2. S protokolem ARP úzce souvisí také protokol RARP. Jedná se o zpětný překlad adres, jde tedy o získání IP adresy z adresy MAC.

Struktura protokolu RARP je stejná jako ARP. Liší se pouze operačním kódem (*Opcode*). Pro dotaz RARP je to hodnota 3, pro odpověď potom hodnota 4. Protokol RARP se používá jen výjimečně, je to případ bezdiskových stanic, která po zapnutí zná pouze svou MAC adresu a je nucena protokolem RARP se dotázat, jakou má adresu IP. Na příslušné LAN potom musí existovat RARP server, který odpoví.

Význam jednotlivých polí protokolu ARP včetně typických hodnot je uveden v Tab. 2.

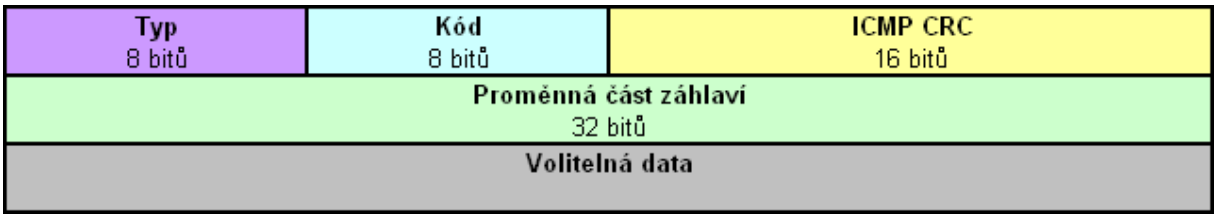
Tab. 2 Stručný popis protokolu ARP.

Název pole	typ. hodnota	Popis
LINK type	**0x0001**	Typ linkového protokolu (**Ethernet II**) (http://www.iana.org)
Prot. type	**0x0800**	Typ síťového protokolu (**IP**)
LINK size	**0x06**	velikost adresy linkové vrstvy
Prot. size	**0x04**	velikost adresy síťové vrstvy
Opcode	**0x01**	požadavek
	0x02	odpověď
Sender MAC		Linková adresa odesilatele
Sender IP		Síťová adresa odesilatele
Target MAC		Linková adresa cíle
Target IP		Síťová adresa cíle

3.2.3 Protokol ICMP

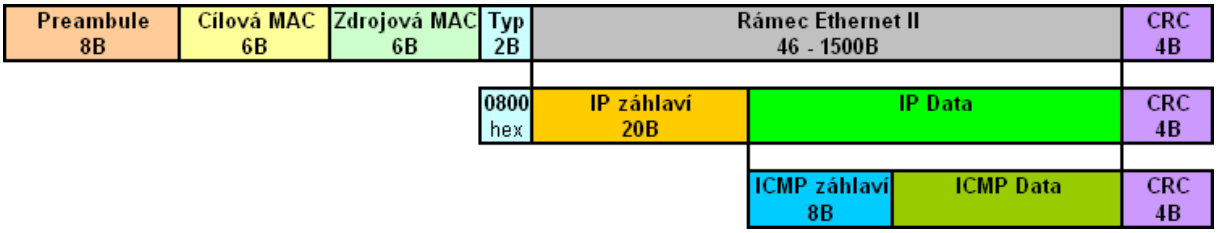
Protokol ICMP (Internet Control Message Protocol) je signalizační protokol, který signalizuje mimořádné události v sítích založených na protokolu IP. Protokol ICMP je definován normou *RCF 792*.

Struktura protokolu ICMP je na Obr. 8. Pakety ICMP jsou baleny do IP datagramu. Začlenění protokolu ICMP do linkového rámce je znázorněno na Obr. 9.



Obr. 8 Protokol ICMP.

První bajt ICMP záhlaví je *Typ služby* ICMP, druhý bajt je *Kód příkazu*. Další dva bajty tvoří *Kontrolní součet*. Další čtyři bajty tvoří *Proměnná část záhlaví*, tato část se svým významem liší podle typu služby. Za záhlavím ICMP následuje datová část ICMP, která se také typ od typu liší. Stručný popis ICMP protokolu včetně typických hodnot pro službu „ECHO“ je uveden v Tab. 3.



Obr. 9 Začlenění protokolu ICMP do IP datagramu a do linkového rámce.

Protokol ICMP může mj. poskytovat následují služby [6]:

- *Echo* – nástroj, pro test dosažitelnosti uzlů v internetu a doby jejich odezvy (*ping*).
- *Nedoručitelný IP datagram* – tuto signalizaci posílá směrovač, když není možné datagram dále směrovat (datagram zahodí a pošle zprávu prostřednictvím ICMP), *Kód* potom určuje přesný důvod (více než 10 případů).
- *Sniž rychlost odesílání* – Pokud je na cestě mezi odesilatelem a příjemcem síť někde přetížena, pak směrovač, který není schopný dále odesílat datagramy signalizuje odesilateli *Sniž rychlost odesílání* (použitelné jen u TCP, UDP tyto zprávy neakceptuje).
- *Změň směrování* – tímto ICMP paketem lze dynamicky měnit směrovací tabulky ve směrovačích. Kód určuje jeden ze čtyř případů.
- *Čas vypršel* – Zahrnuje dva případy: První nastává, když směrovač sníží pole TTL na 0, druhý případ je, že příjemce není schopen z fragmentů složit celý IP datagram.
- *Chybný parametr* – Opět dvě možnosti: První je, když je detekována chyba v hlavičce IP, druhý, když chybí některá požadovaná volitelná položka (nepoužívá se).
- *Žádost o masku subsítě* – Tento paket slouží ke zjištění masky podsítě, kam je tazatel připojen (případ bezdiskových stanic).

Tab. 3 Stručný popis protokolu ICMP (echo).

Název pole	typ. hodnota	Popis
Typ	0x00	Echo - odpověď
	0x08	Echo - požadavek
Kód	0x00	Kód příkazu
ICMP CRC		kontrolní součet
Proměnná		2 Bajty identifikátor, 2 bajty pořadové číslo
Volit. data		max 1472B, aby nedošlo k fragmentaci

3.3 Transportní vrstva

Zatímco vrstva síťová má na starost dopravit data z jedné stanice na druhou (třeba mezi dvěma nebo i více LAN), transportní vrstva řeší předání dat mezi koncovými procesy (aplikacemi). Protože síťová vrstva nezaručuje, že IP datagram opravdu dorazí k cíli, řeší spolehlivost služby protokol vrstvy transportní (TCP).

V transportní vrstvě jsou definovány dva protokoly:

- **UDP** - (User Datagram Protocol) – poskytuje **nespojovanou nespolehlivou službu.**
- **TCP** - (Transmision Control Protocol) – poskytuje **spojovanou spolehlivou službu.**

3.3.1 UDP protokol

Protokol UDP je jednoduchý protokol transportní vrstvy, který poskytuje nespojovanou nespolehlivou službu. UDP je zdokumentován v normě *RFC 768*.

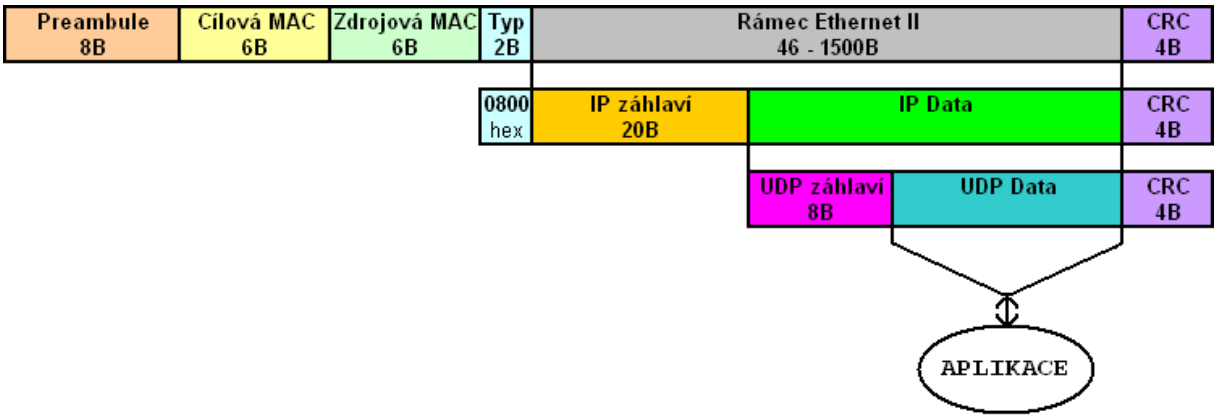
Na Obr. 10 je struktura hlavičky UDP datagramu.

Zdrojový port 16 bitů	Cílový port 16 bitů
Délka datagramu 16 bitů	UDP CRC 16 bitů

Obr. 10 Hlavička UDP datagramu.

Hlavička je velmi jednoduchá, obsahuje pouze zdrojový a cílový port (více o portech viz kap. 3.3.2), velikost UDP datagramu a kontrolní součet, který může být nastaven na *0x0000* (nemusí se povinně počítat, jinak se počítá stejně jako kontrolní součet u TCP, viz kap. 3.3.2). Stručný popis hlavičky UDP je uveden v Tab. 4.

UDP datagramy se balí do IP datagramu, jak je znázorněno na Obr. 11. UDP datagram nese data, která jsou předána přímo konkrétní aplikaci. Pokud by byla požadována spolehlivost služby, musela by ji zajišťovat až aplikace.



Obr. 11 Začlenění UDP datagramu do IP datagramu a do linkového rámce.

Hlavní výhodou protokolu UDP je, že je možné posílat datagramy nejen na konkrétní IP adresu, ale lze posílat také tzv. *oběžníky*.*Oběžníky* mohu být buď všeobecné (*broadcast*) nebo adresní, určeny skupině adres (*multicast*). *Multicast* se používá u distribuce audia a videa, kde nevadí, že je UDP nespolehlivý (ztráta datagramu způsobí jen malou chybu v obraze).

Tab. 4 Stručný popis hlavičky protokolu UDP.

Název pole	typ. hodnota	Popis
Zdrojový port		Zdrojový port
Cílový port		Cílový port
Délka datag.		velikost dat+ velikost hlavičky
UDP CRC	0x0000	Kontrolní součet (není nutný)

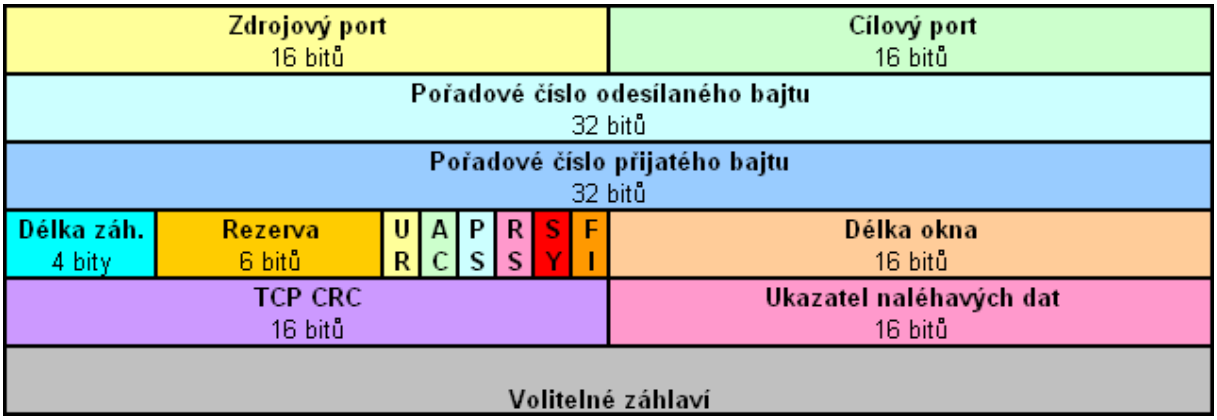
3.3.2 TCP protokol

Na rozdíl od UDP, protokol TCP zaručuje službu spojovanou spolehlivou, proto je také mnohem složitější. Protokol TCP je zdokumentován v normě *RFC 793*.

TCP vyžaduje definované navázaní spojení, přenášená data protistrana potvrzuje a nakonec se spojení definovaným způsobem ukončí. Tento mechanismus je schopen odhalit případnou ztrátu segmentu a definovaným způsoben si vyžádat opětovné odeslání ztraceného TCP segmentu (chybové řízení), také je schopen řídit rychlost odesílání dat (řízení toku). Možnosti protokolu TCP jsou tak rozsáhlé, že by jeho popis převyšoval náplň této práce, budou zde zmíněny jen základní principy využité pro *Řídící a monitorovací jednotku*. Velmi podrobný popis protokolu TCP je v [6], [11].

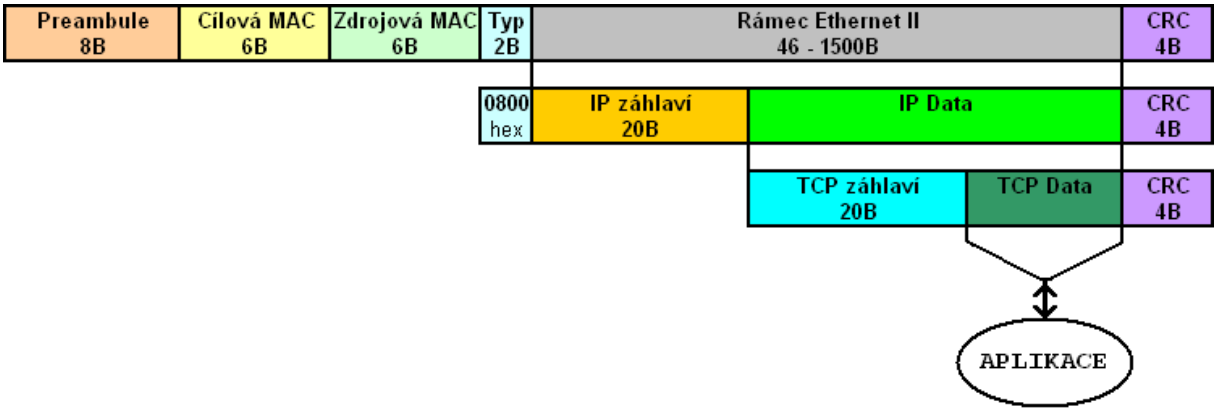
Protokol TCP tedy spolehlivě přenáší data od jedné konkrétní aplikace na straně odesilatele ke konkrétní aplikaci na straně příjemce. Aplikace je dána číslem portu. Číslo portu je dvoubajtové, může být tedy v rozsahu 0 až 65535. Tento rozsah se dělí na tři dílčí rozsahy. Čísla 0 až 1023 jsou pevně svázaná s konkrétními aplikačními protokoly (21–FTP, 80-HTTP, apod.) Rozsah čísel 1024-49151 je určen pro běžné uživatelské aplikace a procesy. Tyto čísla portů jsou registrována u *Internet Assigned Numbers Authority* (IANA). Čísla portů v rozsahu 49152-65535 nejsou registrována a lze je použít pro dynamické či soukromé účely.
Pozn.: Protokol UDP má čísla portů definována stejně. Lze komunikovat na stejném portu protokolem TCP a UDP, aplikace jsou potom odděleny právě protokolem transportním.

Struktura hlavičky TCP segmentu je uvedena na Obr. 12. Již na první pohled je zřejmé, že se jedná o složitější protokol než je UDP. Stručný popis významu jednotlivých polí v hlavičce je uveden v Tab. 5.



Obr. 12 Hlavička TCP segmentu.

TCP segment je balen (stejně jako UDP) do IP datagramu. Začlenění TCP segmentu do IP datagramu a poté do linkového rámce Ethernet II je uvedeno na Obr. 13.



Obr. 13 Začlenění TCP segmentu do IP datagramu a do linkového rámce.

TCP hlavička podrobně:

- **Zdrojový port** – (*source port*) - číslo portu aplikace na straně odesilatele, na tento port se bude posílat potvrzení, případně data.
- **Cílový port** – (*destination port*) - číslo portu aplikace na straně příjemce.
- **Pořadové číslo odesílaného bajtu** – (*sequence number*) – sekvenční číslo odesílaného bajtu. Při navazování spojení (nastaven bit *SYN*) je generováno náhodné číslo *ISN (Initial Sequence Number)*, při komunikaci je potom bráno relativní číslo oproti *ISN*.
- **Pořadové číslo přijatého bajtu** – (*acknowledgment number*) – platné při nastaveném bitu *ACK*, potvrzuje přijatá data do tohoto čísla, resp. další data očekává s tímto sekvenčním číslem.
- **Délka záhlaví** – (*data offset*) – vyjadřuje velikost TCP hlavičky v násobcích 4 bajtů, velikost hlavičky prakticky udává ofset dat v TCP segmentu..
- **Rezerva** – (*reserved*) – šest bitů nastavených na 0. Pozn.: Norma *RCF3168* definuje nejnižší dva rezervované bity jako *CWR* (*Congestion Window Reduced) flag* a *ECE (ECN-Echo).*
- **Příznakové bity** – (*flags*) – šestice bitů, kde každý má svůj specifický význam.
  URG – (urg*ent*) – příznak, že TCP segment nese naléhavá data, potvrzuje platnost pole *Ukazatel naléhavých dat*, které ukazuje na naléhavá data (ofset v datech TCP segmentu).
  ACK – (ack*nowledgment*) – potvrzuje platnost pole *Pořadové číslo přijatého bajtu.*
  PSH – (p*ush*) – příznak aplikačních dat, data mají být předána aplikaci (*push* funkce).
  RST – (reset) – reset spojení, tímto příznakem bude ukončeno spojení (bez nutnosti standardního ukončení), také se používá k odmítnutí spojení.
  SYN – (*synchronize*) – při navazování spojení, synchronizace sekvenčního čísla – sekvenční číslo je *ISN*.
  FIN – (*finish*) – při ukončování spojení, obě strany musejí odeslat segment s nastaveným *FIN* a protistrana jej musí potvrdit.
- **Délka okna** – (*windows size*) – vyjadřuje přírůstek k *Pořadové číslo přijatého bajtu*, který je příjemce schopen přijmout.
- **Kontrolní součet** – (*check sum*) – kontrolní součet se počítá podle normy *RFC1071*, na rozdíl od UDP je v TCP povinný. Počítá se z tzv. *Pseudozáhlaví*, hlavičky TCP a TCP dat. *Pseudozáhlaví* je část hlavičky IP, je to IP odesilatele, IP příjemce, protokol vyšší vrstvy (bráno dvoubajtově, vyšší bajt je 0) a celkové délky IP datagramu.

- **Ukazatel naléhavých dat** – (*urgent pointer*) – platné při nastaveném *URG*, přičtením ukazatele k pořadovému číslu odesílaného bajtu, dostaneme konec úseku naléhavých dat. Je využíván např. programem *TELNET*.
- **Volitelné záhlaví** – (*options*) – musí mít velikost v násobcích 4 bajtů, používá se při navazování spojení, kdy je zde informace o velikosti *MSS* (maximum segment size), popř. další parametry, apod.

Stručné shrnutí popisu hlavičky TCP, včetně typických hodnot je uveden v Tab. 5.

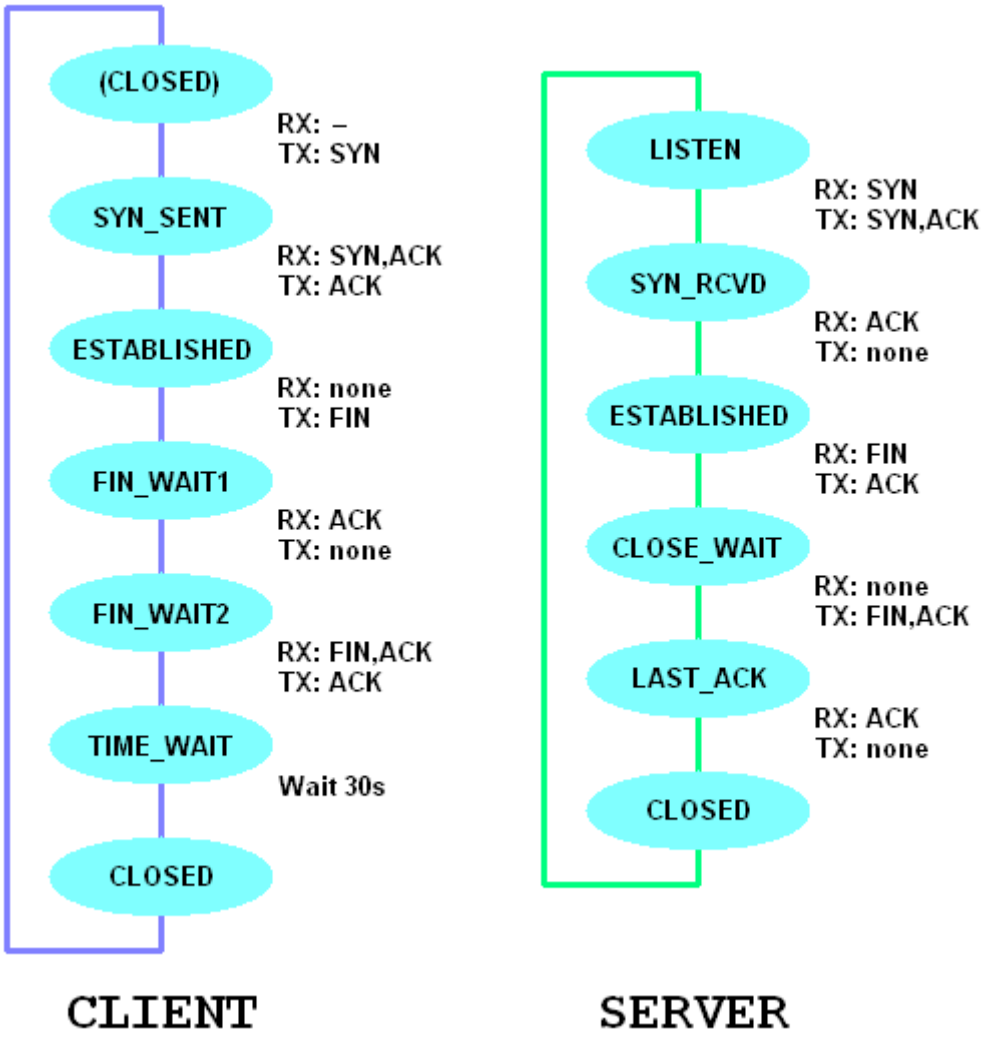
Jak již bylo zmíněno, protokol TCP umožňuje chybové řízení a řízení toku dat. Využívá k tomu tzv. metodu okna. Každý TCP segment nese v hlavičce informaci o velikosti okna – velikost volné paměti přijímacího bufferu. Pokud je velikost okna na 0, nebudou se posílat další data, dokud příjemce neodešle segment s nenulovou velikostí okna. Tohoto řízení toku se využívá, když aplikace není schopna rychle odebírat data z přijímacího bufferu. Přijatá data se musejí potvrzovat, potvrzování může probíhat kumulativně, tzn., že není potvrzován každý segment, ale určitý blok. Existuje několik scénářů vzniku chyby – ztráta datového segmentu, ztráta potvrzovacího segmentu, a podle toho potom několik možností řešení ztráty – opakované odeslání datového segmentu, potvrzení následujícím potvrzovacím paketem (potvrzuje další data, čímž potvrdí i data předchozí. Detailní popis je v [6], [11].

Tab. 5 Stručný popis hlavičky protokolu TCP.

Název pole	typ. hodnota	Popis
Zdrojový port		Zdrojový port
Cílový port		Cílový port
Pořad. odesl.		*Sequence Number* (Sekvenční číslo odesílaného bajtu, ISN (Initial Sequence Number) při nastaveném SYN)
Pořad. přijat.		*Acknowledgment Number* (Pořadové číslo přijatého bajtu)
Délka záhlaví	**5**	Délka záhlaví v násobcích 4 (stejně jako u IP)
Rezerva	**0**	Rezerva, musí být 0
Flags	**URG**	Příznak naléhavých dat, která se mají zpracovat přednostně (pole *Ukazatel naléhavých dat* je platné)
	ACK	Příznak platnosti pole *Pořadové číslo přijatého bajtu*. Nastaven vždy kromě segmentu s nastaveným SYN.
	PSH	Příznak, že segment nese aplikační data. Není ustáleno!
	RST	Reset - odmítnutí, ukončení spojení.
	SYN	Navázání spojení - reset pořadových čísel dat.
	FIN	Odesilatel odeslal všechna data a uončuje spojení, protistrana může ještě vysílat, nebo také ukončit.
Délka okna	**1460**	Velikost dat, které je stanice schopna přijmout (velikost volného bufferu).
TCP CRC		Kontrolní součet (RFC1071)
Ukaz. naléh	**0**	Ukozatel naléhavých dat (platné při nastaveném URG)
Volitelné		(Velikost musí být v násobcích 4 bajtů) Využito při nastaveném SYN - navazování spojení, nese informaci o MSS

Protokol TCP je stavový protokol. Vyplývá to z vlastnosti, že je to protokol spojovaný. Při navazování spojení a při uzavírání spojení protokol mění svůj stav, aby bylo možné určit, v jaké fázi se přesně nachází. Stavy jsou rozdílné pro stanici, která spojení navazuje a stanicí, na kterou se připojuje. Podle toho lze o stanicích hovořit jako o *Clientovi* a *Serveru*. Možné stavy Clienta a Serveru jsou znázorněny na Obr.14.

Client – stanice, která spojení navazuje – odešle TCP segment (dále jen paket) s nastaveným bitem SYN, pole *Pořadové číslo odesílaného bajtu* bude naplněno náhodnou hodnotou (*ISN*) a ve volitelné části hlavičky bude informace o velikosti *MSS*. Tento paket se odešle na požadovaný port serveru (např.: 80). Odesláním paketu je clientův stav ***SYN_SENT***. Server musí být na daném portu ve stavu ***LISTEN***, aby bylo možné spojení navázat. Příjmem paketu přejde do stavu ***SYN_RCVD*** a zároveň odešle paket a nastaveným SYN, ACK, pole *Pořadové číslo odesílaného bajtu* bude naplněno náhodnou hodnotou (*ISN*) a ve volitelné části hlavičky bude informace o velikosti *MSS*. Client tento paket přijme, změní svůj stav na ***ESTABLISHED*** a odešle paket s nastaveným ACK. Server tento paket příjme a změní stav na ***ESTABLISHED***. Nyní je spojení navázáno a může probíhat datová komunikace mezi clientem a serverem (plně duplexní)). Spojení může uzavřít libovolná strana. V tomto případě uzavírá spojení client. Client tedy odešle paket s nastaveným FIN,ACK a změní stav na ***FIN_WAIT1*** (aktivní uzavření). Server obdrží paket, změní stav na ***CLOSE_WAIT*** a odešle paket s nastaveným ACK. Client paket příjme a změní stav na ***FIN_WAIT2*** (neposílá nic). Potom server pasivně uzavírá odesláním paketu s nastaveným FIN,ACK a změní stav na ***LAST_ACK***. Client tento paket příjme změní stav na ***TIME_WAIT*** a odešle paket s nastaveným ACK. Server přijme paket a změní stav na ***CLOSED***, čímž je spojení uzavřeno. Client ještě vyčká 30 sekund a poté změní stav na ***CLOSED***. Podrobnější popis je v [6].



Obr. 14 Stavy protokolu TCP pro spojení CLIENT-SERVER.

3.4 Aplikační vrstva

Vazbu mezi transportní a aplikační vrstvou tvoří tzv. *sockety*. *Socket* je definován číslem portu (zdrojového i cílového) a transportním protokolem. Každý proces (aplikace) využívá *socket* (nebo i více) pro předávání aplikačních dat mezi aplikační a transportní vrstvou (nemůže nastat situace, že by aplikaci byla předána data, která jí nepatří). Formát aplikačních dat, popř. i jejich význam či časování definují aplikační protokoly. Existuje celá řada aplikačních protokolů, většinou jsou spjaty s nějakým číslem portu. Např. je to pro TCP na portu 80 *HyperText Transfer Protocol* (HTTP), *File Transfer Protocol* (FTP) na portu 21, *Simple Mail Transfer Protocol* (SMTP) na portu 25, nebo pro UDP *Domain Name System* (DNS) na portu 53, atd. Tyto protokoly jsou určeny pro komunikaci Klient-Server, číslo portu odpovídá cílovému portu, na který se klienti připojují. Existují také aplikační protokoly pro komunikaci Bod-Bod, případně hybridní (uživatelé pro navázání spojení potřebují služby serveru, poté komunikují bod-bod).

3.4.1 HTTP protokol

Protokol HTTP (HyperText Transfer Protokol) používají webové prohlížeče a webové servery. Jedná se o aplikační protokol pro komunikaci Klient-Server. Klienti se připojují na server na port TCP/80, posílají na server požadavky a server jim posílá odpovědi. Jedná se o protokol bezstavový. Existují tři verze: HTTP/0.9, HTTP/1.0, HTTP/1.1. V dnešní době je nejrozšířenější verze HTTP/1.1, verze 0.9 se nepoužívá, verze 1.0 se používá zřídka (používá se pro jednoduchou správu přes webové rozhraní např. u tiskáren, měřících stanic, apod. Bude použit i pro *Řídící a monitorovací jednotku*). Protokol HTTP/1.0 je popsán normou *RCF 1945*.

Komunikace protokolem HTTP může být *perzistentní* a *neperzistentní*. Při neperzistentním spojení se pro každý požadavek (každý objekt webové stránky) otevírá nové spojení. Perzistentní spojení otevře jedno spojení a v tom posílá všechny požadavky. Pokud se pošle více požadavků za sebou bez čekání na odpověď, pak hovoříme o tzv. *pipeningu*.

HTTP/1.1 má implicitní nastavení na perzistentní komunikaci, HTTP/1.0 umožňuje pouze neperzistentní komunikaci.

Tab. 6 Obecný formát dotazu HTTP.

metoda	SP	URL	SP	verze	CR	LF
1. řádek hlavičky					CR	LF
2. řádek hlavičky					CR	LF
.						
.						
n. řádek hlavičky					CR	LF
CR	LF					
data (nepovinné)						

Obecný formát HTTP dotazu je uveden v Tab. 6. První pole požadavku tvoří *metoda*.

Metoda může být:

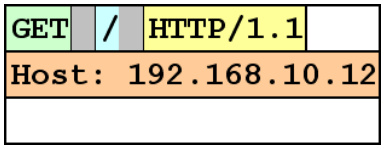
- *GET* – pro stažení objektu,
- *POST* – pro odeslání parametrů (údaje z html formulářů) na server,
- *HEAD* – požadavek na hlavičku odpovědí (samotná data se neposílají).

Pro verzi HTTP/1.1 jsou to dále:

- *PUT* – upload souboru na server,
- *DELETE* – mazání souboru ze serveru,
- *OPTIONS* – dotaz na vlastnosti serveru,
- *TRACE* – dotaz pro určení, kolik *proxy* je na cestě na server.

Za polem *metoda* následuje pole obsahující znak mezery, déle je pole s adresou URL, následuje opět pole s mezerou, poté je pole s verzí protokolu a řádek je ukončen standardními znaky *CR* a *LF*. Další řádky jsou hlavičkové, povinná je jen hlavička „Host:“. Hlavičkové řádky jsou také standardně ukončeny CR, LF. Za hlavičkovými řády následuje prázdný řádek (pouze znaky CR, LF) a po prázdném řádku již následují případná data.

Konkrétní příklad http požadavku je na Obr. 15.



Obr. 15 Příklad HTTP požadavku.

Obecný formát HTTP odpovědi udává Tab. 7. První pole je verze protokolu, následuje mezera, dále je kód výsledku, následuje opět mezera a poté následuje textový popis výsledku. Další formát je shodný s HTTP požadavkem.

Tab. 7 Obecný formát odpovědi HTTP.

verze	SP	kód	SP	popis	CR	LF
1. řádek hlavičky					CR	LF
2. řádek hlavičky					CR	LF
.						
.						
n. řádek hlavičky					CR	LF
CR	LF					
data						

Kód výsledku je tříciferný, kde první cifra určuje druh odpovědi, zbylé dvě jsou upřesňující.

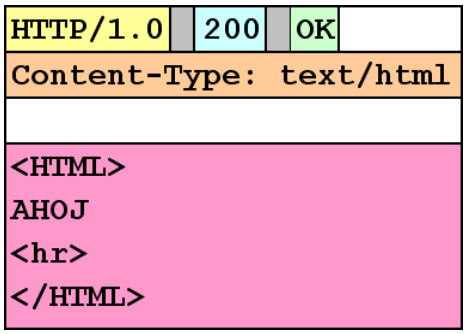
Možné druhy odpovědi:

1xx – informativní odpověď.
2xx – kladná odpověď, úspěch operace.
3xx – přesměrování.
4xx – chyba klienta.
5xx – chyba serveru.

Nejčastější kódy (včetně textového popisu):

100 Continue - pokračování.
200 OK - úspěch.
403 Forbidden - neoprávněný přístup.
404 Not Found - objekt nenalezen.

Konkrétní příklad HTTP odpovědi je znázorněn na Obr. 16.



Obr. 16 Příklad HTTP odpovědi.

4 Hardware

Vzhledem k požadavku řízení a monitorování přes internet, musí být hardware navržený tak, aby jej bylo možné připojit k síti ethernet. Existují dvě základní varianty připojení k síti ethernet. První je použití mikrokontroléru s implementovanou periferií Ethernet již na čipu, nebo druhá, použitím ethernetového kontroléru, který se dodatečně připojí k libovolnému mikrokontroléru.

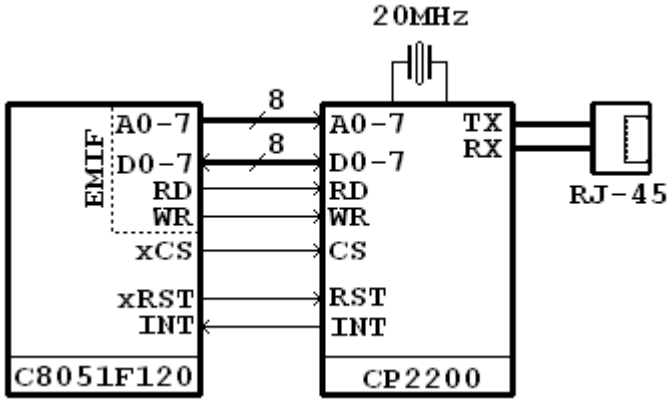
První varianta se zdá být výhodnější, protože je vše v jednou pouzdře a není potřeba vyrábět složitou desku s plošnými spoji, nevýhodou však je, že takový mikrokontrolér má své jádro, svou instrukční sadu a jeho programování by mohlo činit potíže. Také jsme limitováni paměťovým prostorem, který je pevně dán typem mikrokontroléru.

Druhá varianta má nevýhodu v nutnosti výroby desky plošného spoje pro propojení ethernetového kontroléru a použitého mikrokontroléru. Nicméně lze použít jakýkoliv mikrokontrolér, třeba takový, jehož architekturu dobře známe. Navíc pokud se obě části – ethernetový kontrolér a mikrokontrolér – rozdělí na samostatné HW moduly, je možné podle požadavků modul mikrokontroléru vyměnit třeba za modul s jiným (výkonnějším, levnějším, apod.) mikrokontrolérem.

Pro Řídící a monitorovací jednotku je zvolena druhá varianta – mikrokontrolér a externí ethernetový kontrolér. Tato varianta byla zvolena s ohledem na zadání a hlavně z důvodu, že modul s mikrokontrolérem byl včetně ladících nástrojů dán k dispozici pro vývoj a tvorbu této diplomové práce.

Hardware Řídící a monitorovací jednotky je založen na ethernetovém kontroléru (řadiči) CP2200 [4] a mikrokontroléru C8051F120 [3]. Oba obvody vyrábí firma Silicon Laboratories.

Blokové schéma připojení kontroléru CP2200 k mikrokontroléru C8051F120 je na Obr. 17. Připojení je realizováno jako připojení externí paměti, tzn. pomocí osmibitové datové sběrnice, osmibitové adresní sběrnice, řídících signálů pro čtení *RD*, zápis *WR*, a signálu pro výběr obvodu *CS*. Navíc je zde signál *RST*, pro HW reset ethernetového kontroléru a signál *INT*, který indikuje požadavek na přerušení od ethernetového kontroléru.



Obr. 17 Blokové schéma připojení kontroléru CP2200 k mikrokontroléru C8051F120.

Hardware jednotky je tvořen dvěma moduly:

- **Modul C8051F120** – osazen mikrokontrolérem C8051F120.
- **Modul EthernetCP2200** – osazen řadičem CP2200.

4.1 Modul C8051F120

Rozmístění prvků modulu C8051F120 je na Obr. 18. Modul je osazen těmito základními prvky:

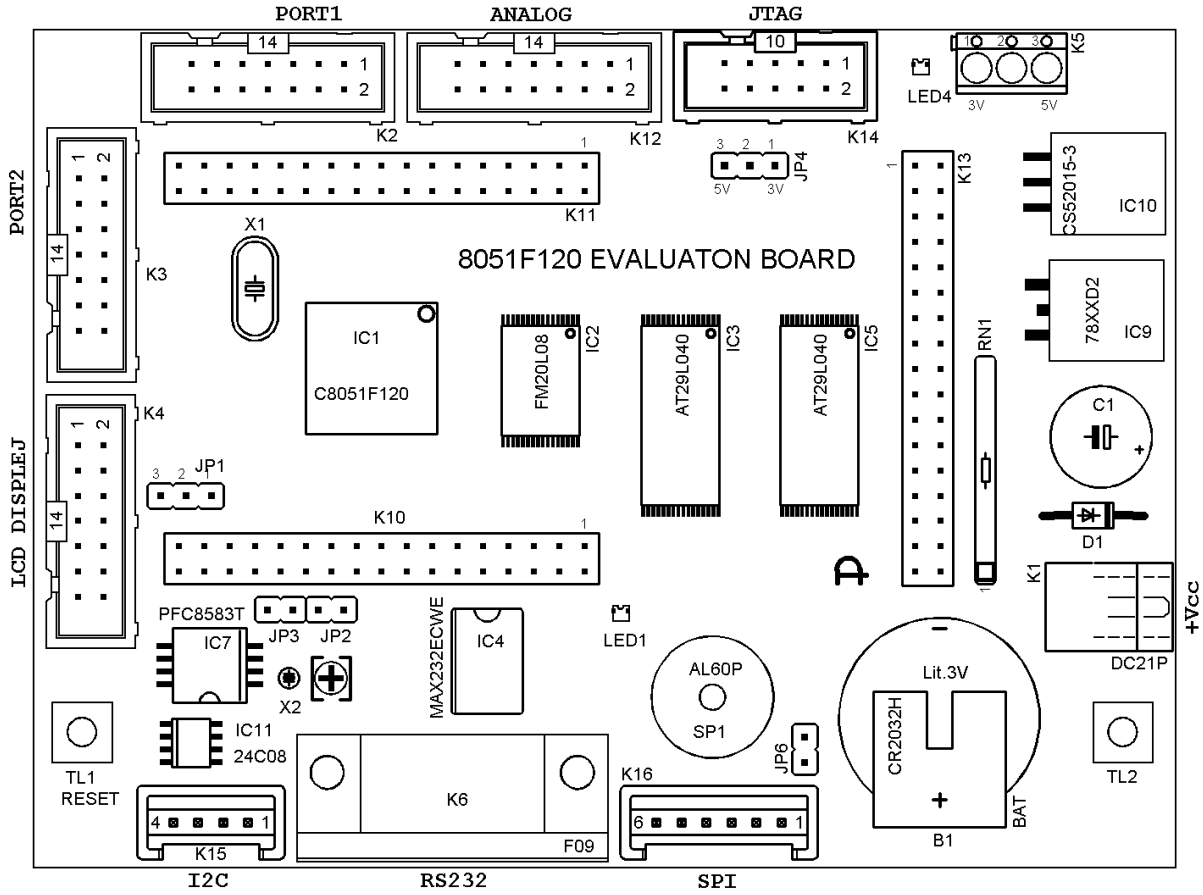
- mikrokontrolérem C8051F120 (IC1),
- pamětí FRAM FM20L08 (IC2),
- pamětí FLASH 2xAT29L040 (IC3, IC4),
- obvodem reálného času PCF8583 (IC7),
- pamětí EEPROM 24C08 (IC11),
- převodníkem úrovní MAX232 (IC4),
- stabilizátory napětí 3V a 5V (IC10, IC9),
- konektory pro připojení dalšího modulu (K10, K11, K13),
- konektory s vyvedenými porty P1 a P2 (K2, K3),
- konektorem pro LCD (K4),
- konektorem pro připojení analogových signálů (K12),
- konektorem pro JTAG (K14),
- konektorem pro sběrnici I^2C (K15),
- konektorem pro sběrnici ISP (K16),
- konektorem pro připojení RS232 (K6),
- napájecím konektorem,
- atd.

Napájení modulu se přivádí na konektor K1. Rozsah napájecího napětí je asi +6V až +16V (omezeno maximálním napětím vstupního filtračního kondenzátoru). Modul je chráněn proti přepólování sériovou diodou. Napájecí napětí je stabilizováno na +5V a pak na +3,3V. Obě tyto stabilizovaná napětí jsou rozvedena po celém modulu, k příslušným obvodům a na konektory.

Tl1 na modulu slouží jako reset C8051F120, signál tohoto tlačítka je rovněž vyveden na konektor *K10* a lze jej tedy rovněž použít pro reset dalších obvodů.

Na modulu jsou prvky k volnému použití. Všechny tyto prvky jsou vyvedeny na konektor *K10*. Z konektoru pak mohou být napojeny podle potřeby na zvolený pin mikrokontroléru. Jsou to:

- tlačítko Tl2,
- indikační LED1,
- siréna SP1.



Obr. 18 Rozmístění prvků modulu C8051F120.

4.1.1 Popis konektorů pro připojení periferií k modulu C8051F120

Zde budou popsány využité konektory pro připojení periferií k modulu C8051F120.

Konektor K14:

Konektor *K14* slouží k připojení JTAG debug adaptéru, pomocí kterého je možné procesor programovat a krokovat a ladit program.

Konektor K12:

Konektor *K12* slouží k připojení analogových signálů. Obsahuje osm analogových vstupů, dva analogové výstupy, vstup a výstup referenčního napětí, je zde vyvedena analogová zem a napětí +3,3V nebo +5V, volitelné jumperem na *JP4*. Zapojení konektoru K12 je v Tab. 8.

Konektor K4:

Konektor *K4* slouží k připojení LCD displeje. Tento konektor zabírá port P3 mikrokontroléru. Pro osmibitovou komunikaci jsou zde definovány signály *K47,48,49,410*, které jsou vyvedeny na konektor *K10*. Odtud se mohou podle potřeby napojit na definované piny mikrokontroléru. Pro kontrast displeje může být použita linka P3.0, nebo napětí z odporového trimru *R8*. Toto se volí jumperem na *JP1*. Zapojení konektoru *K4* je uvedeno v Tab. 8.

Tab. 8 Zapojení konektoru K12 a K4.

Význam pinů konektoru K12		
Pin	**Funkce**	**Popis**
1	AIN0	vstup ADC0
2	AIN1	vstup ADC1
3	AIN2	vstup ADC2
4	AIN3	vstup ADC3
5	AIN4	vstup ADC4
6	AIN5	vstup ADC5
7	AIN6	vstup ADC6
8	AIN7	vstup ADC7
9	VCC3/5	napájení +3,3V nebo +5V
10	AGND	analogová zem
11	DAC0	výstup DAC0
12	DAC1	výstup DAC1
13	VREF01	VREF0, VREF2 (spojeno)
14	VREF	VREFD, VREF (spojeno)

Význam pinů konektoru K4		
Pin	**Funkce**	**Popis**
1	DGND	vstup ADC0
2	VCC5	vstup ADC1
3	CONT	pin P3.0 nebo trimrR8
4	P3.1	pin P3.1 (RS_LCD)
5	P3.2	pin P3.2 (RW_LCD)
6	P3.3	pin P3.3 (E_LCD)
7	K47	--- (viz text)
8	K48	--- (viz text)
9	K49	--- (viz text)
10	K410	--- (viz text)
11	P3.4	pin P3.4 (D4_LCD)
12	P3.5	pin P3.5 (D5_LCD)
13	P3.6	pin P3.6 (D6_LCD)
14	P3.7	pin P3.7 (D7_LCD)

Konektor K6:

Konektor *K6* slouží k připojení k rozhraní RS232. Jedná se o klasický 9-tipinový DSUB konektor, zásuvku. Ze signálů RS232 se využívá pouze *TxD, RxD, RTS, CTS*. Signály *TxD* a *RxD* jsou napojeny přímo na mikrokontrolér (piny P0.0 a P0.1), signály *RTS* a *CTS* jsou vyvedeny na konektor *K10*.

4.1.2 Popis konektorů K13, K11 a K10 modulu C8051F120

Konektor K13:

Konektor *K13* je určen pro připojení externí paměti k mikrokontroléru. Má vyvedenou paměťovou sběrnici, datovou sběrnici, řídící signály *RD* a *WR*, signál pro výběr obvodu *CEEXT*, a napájecí napětí +3,3V a +5V. Na tento konektor je tedy možné připojit řadič CP2200. Zapojení konektoru je uvedeno v Tab. 9.

Konektor K11:

Konektor *K11* obsahuje osm analogových vstupů, dva analogové výstupy, vstupy referenčních napětí, vstupy komparátorů, je zde vyvedeno analogové napětí +3,3V a digitální napájení +3,3V a je zde vyveden port P1. Signály *A, B, C, D* jsou napojeny na konektor *K2*. Zapojení konektoru *K11* je v Tab. 10.

Konektor K10:

Na konektoru *K10* jsou vyvedeny porty P0 a P2, dále indikační LED, siréna, tlačítko, signály RTS a CTS rozhraní RS232, sběrnice I2C z konektoru *K15* (připojeno k RTC a EEPROM), reset C8051F120, požadavek na přerušení od obvodu RTC. Signály *K47,48,49,410* jsou napojeny na konektor *K4* a představují čtyři datové bity displeje LCD. Zapojení konektoru *K10* je v Tab. 11.

Tab. 9 Zapojení konektoru K13.

Význam pinů konektoru K13					
Popis	**Funkce**	**Pin**	**Pin**	**Funkce**	**Popis**
digitální zem	DGND	**1**	**2**	VCC3	napájení +3,3V
adresa 4	A4	**3**	**4**	A5	adresa 5
adresa 6	A6	**5**	**6**	A7	adresa 7
adresa 12	A12	**7**	**8**	A15	adresa 15
adresa 16	A16	**9**	**10**	A18	adresa 18
signál pro zápis	WR	**11**	**12**	A17	adresa 17
adresa 14	A14	**13**	**14**	A13	adresa 13
adresa 8	A8	**15**	**16**	A9	adresa 9
adresa 11	A11	**17**	**18**	A3	adresa 3
adresa 2	A2	**19**	**20**	A1	adresa 1
adresa 0	A0	**21**	**22**	D0	data 0
data 1	D1	**23**	**24**	D2	data 2
data 3	D3	**25**	**26**	D4	data 4
data 5	D5	**27**	**28**	D6	data 6
data 7	D7	**29**	**30**	A10	adresa 10
signál pro čtení	RD	**31**	**32**	CEEXT	extrerní chip enable
digitální zem	DGND	**33**	**34**	VCC5	napájení +5V

Tab. 10 Zapojení konektoru K11.

Význam pinů konektoru K11					
Popis	**Funkce**	**Pin**	**Pin**	**Funkce**	**Popis**
vstup - komparátoru 1	CP1-	**1**	**2**	CP1+	vstup + komparátoru 1
vstup - komparátoru 0	CP0-	**3**	**4**	CP0+	vstup + komparátoru 0
napájení +3,3V	VDD3A	**5**	**6**	AGND	analogová zem
vstup ADC0	AIN0	**7**	**8**	AIN1	vstup ADC1
vstup ADC2	AIN2	**9**	**10**	AIN3	vstup ADC3
vstup ADC4	AIN4	**11**	**12**	AIN5	vstup ADC5
vstup ADC6	AIN6	**13**	**14**	AIN7	vstup ADC7
napájení +3,3V	VCC3	**15**	**16**	DGND	digitální zem
výstup DAC0	DAC0	**17**	**18**	DAC1	výstup DAC1
VREF0, VREF2 (spojeno)	VREF01	**19**	**20**	VREF	VREFD, VREF (spojeno)
pin P1.0 (INT0)	P1.0	**21**	**22**	P1.1	pin P1.1 (INT1)
pin P1.2	P1.2	**23**	**24**	P1.3	pin P1.3
pin P1.4	P1.4	**25**	**26**	P1.5	pin P1.5
pin P1.6	P1.6	**27**	**28**	P1.7	pin P1.7
---	A	**29**	**30**	B	---
---	C	**31**	**32**	D	---
napájení +3,3V	VCC3	**33**	**34**	DGND	digitální zem

Tab. 11 Zapojení konektoru K10.

Význam pinů konektoru K10					
Popis	**Funkce**	**Pin**	**Pin**	**Funkce**	**Popis**
indikační LED (LED1)	LED1	**1**	**2**	SP	siréna
tlačítko (TI2)	TLEXT	**3**	**4**	VDD3B	+3V záloha BAT
CTS (RS232)	R1OUT	**5**	**6**	T1IN	RTS (RS232)
pin P0.0 (TX0)	P0.0	**7**	**8**	P0.1	pin P0.1 (RX0)
pin P0.2 (SCK)	P0.2	**9**	**10**	P0.3	pin P0.3 (MISO)
pin P0.4 (MOSI)	P0.4	**11**	**12**	P0.5	pin P0.5 (NSS)
pin P0.6 (I2C_SDA)	P0.6	**13**	**14**	P0.7	pin P0.7 (SCL)
napájení +5V	VCC5	**15**	**16**	DGND	digitální zem
pin P2.0	P2.0	**17**	**18**	P2.1	pin P2.1
pin P2.2	P2.2	**19**	**20**	P2.3	pin P2.3
pin P2.4	P2.4	**21**	**22**	P2.5	pin P2.5
pin P2.6	P2.6	**23**	**24**	P2.7	pin P2.7
data I2C	SDA	**25**	**26**	SCL	clock I2C
reset C8051F120	RST	**27**	**28**	INT8583	přerušení od RTC
---	K48	**29**	**30**	K410	---
---	K47	**31**	**32**	K49	---
napájení +5V	VCC5	**33**	**34**	DGND	digitální zem

4.2 Modul EthernetCP2200

Schéma zapojení modulu EthernetCP2200 je na Obr. 19. Fotografie modulu bez konektoru pro ethernet je na Obr. 24. **Modul kromě ethernetového řadiče obsahuje ještě řadič USB (schéma i část desky plošného spoje byla dodána), tento však není náplní této práce a dále nebude zmiňován**.

4.2.1 Popis modulu EthernetCP2200

Jak již bylo uvedeno, jádro modulu tvoří ethernetový řadič CP2200 firmy Silicon Laboratories. Kromě datasheetu [4] firma dodává také příklady kódů pro obsluhu řadiče viz [1], [2]. Tento obvod zahrnuje integrovaný IEEE 802.3 Ethernet MAC - Media Access Controller (kontrolér pro přístup k médiu), fyzickou vrstvu 10BASE-T a 8kB FLASH.

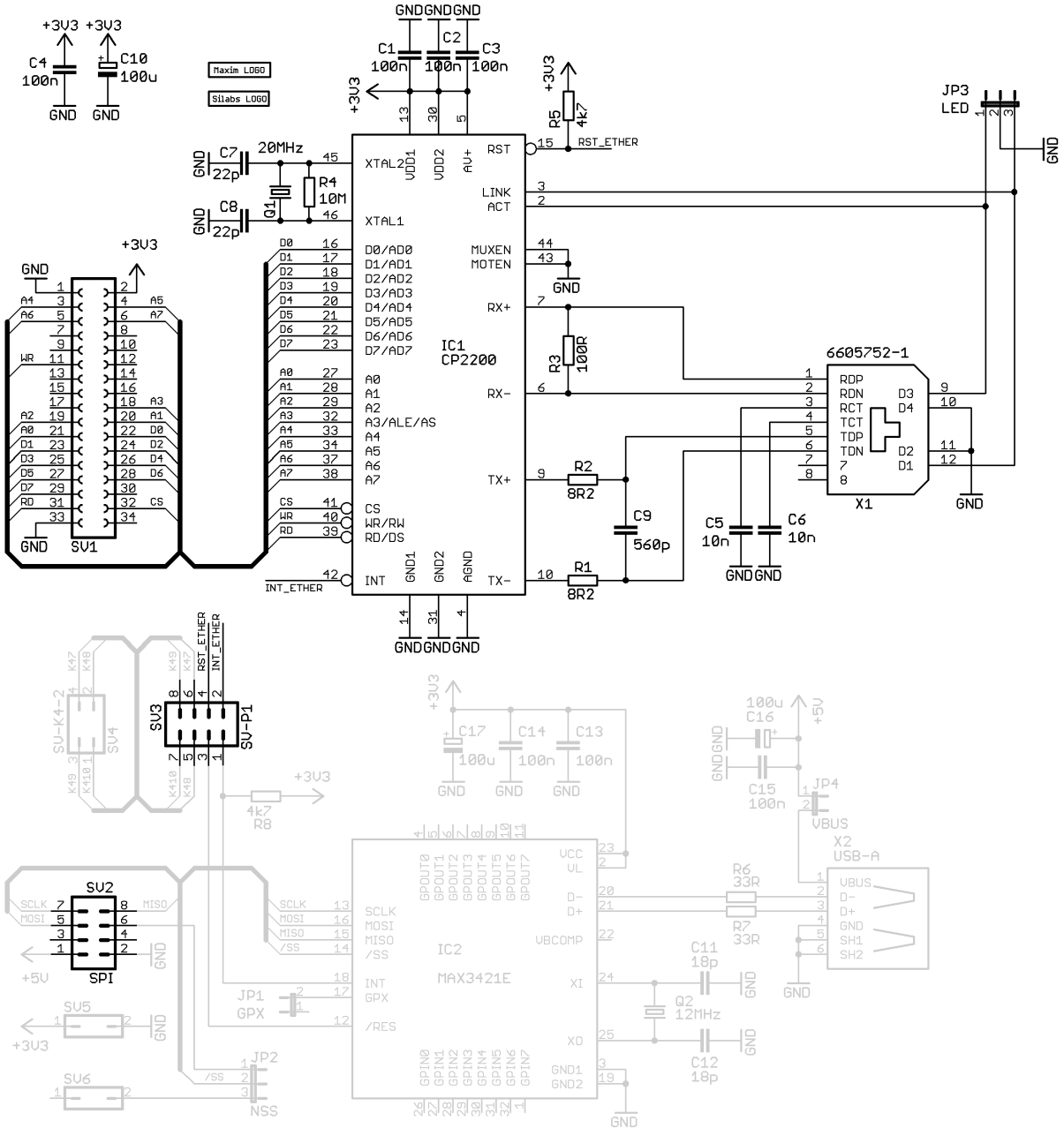
Hodinový kmitočet obvodu CP2200 je tvořen krystalem Q1 (musí být 20MHz ± 50 ppm viz [4]), kondenzátory C7 a C8 a rezistorem R4. Piny *MUXEN* a *MOTEN* jsou uzemněny, tím se definuje formát sběrnice. Resetovací pin je přes rezistor R5 držen na vysoké úrovni. Napájení analogové a digitální části CP2200 je společné a není nijak odděleno.

Modul se připojuje k modulu C8051F120 přes konektory *SV1* a *SV-P1* (konektor *SP1* je osazen pouze z důvodů mechanické pevnosti). Konektor *SV1* tvoří sběrnici pro připojení externí paměti, tedy přímo k připojení CP2200. Na konektoru *SV-P1* jsou využity pouze dva piny, a to *RST_ETHERNET* a *INT_ETHERNET*. 4 piny konektoru vedoucí na konektor *SV4* (není osazen) jsou určeny pro osmibitové připojení displeje (viz modul C8051F120), tyto nejsou pro tento účel použity (připojení displeje je čtyřbitové) a mohou tudíž být použity např. pro indikaci stavů, např pomocí přídavných LED.

Modul je napájen napětím +3,3V, přivedeným přes konektor SV1 z modulu C8051F120. Napájecí napětí je blokováno kondenzátory C1,2,3,4 a C10.

Modul je osazen ethernetovým konektorem 6605762-1 (*X1)*, který má již integrovaný magnetický obvod a dvě indikační LED (link, active) společně s rezistory. Magnetický obvod má vysílací vinutí v poměru 1:2,5, aby byla zachována impedance 100Ω. Při použití jiného převodového poměru není zaručena správná funkce a není garantována maximální vzdálenost 100m k aktivnímu prvku.

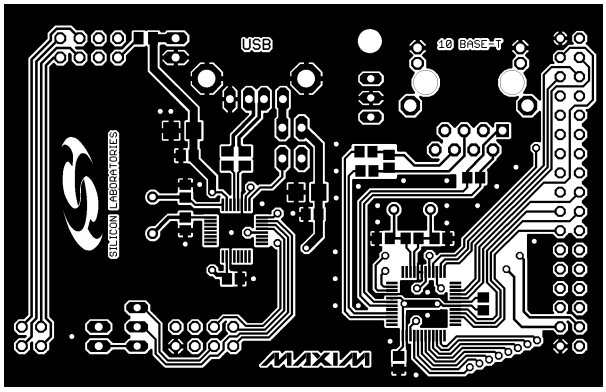
Pro případ použití jiného typu ethernetového konektoru je modul doplněn konektorem *JP3*. Nejedná se o konfigurační konektor, nýbrž o konektor pro připojení externích indikačních LED (link, active). **Připojené externí LED musejí mít v sérii rezistor, aby proud nepřesáhl 10mA (alespoň 100Ω)!**



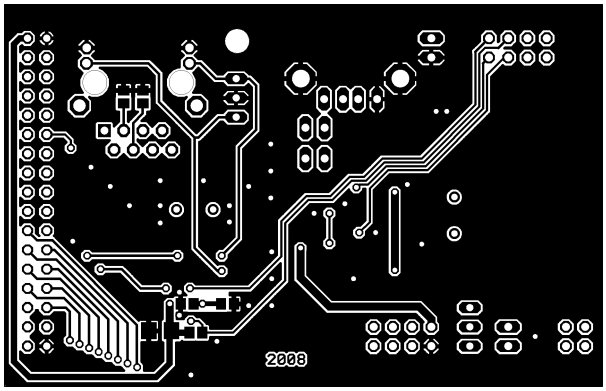
Obr. 19 Schéma zapojení modulu EthernetCP2200.

4.2.2 Deska s plošnými spoji modulu EthernetCP2200

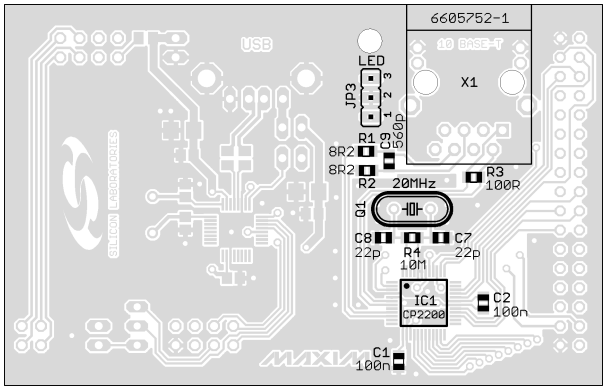
Deska s plošnými spoji modulu EthernetCP2200 je na Obr. 20 a Obr. 21. Rozměry desky jsou 79x50,4mm. Na Obr.22 a Obr. 23 je pak uvedeno osazení desky. Deska je koncipována tak, aby ji bylo možné přímo zasunout do modulu C8051F120 bez nutnosti použití propojovacích kabelů.



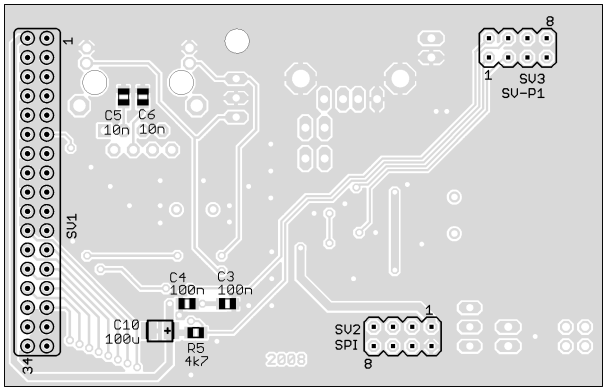
Obr. 20 Horní strana desky EthernetCP2200 (1:1).



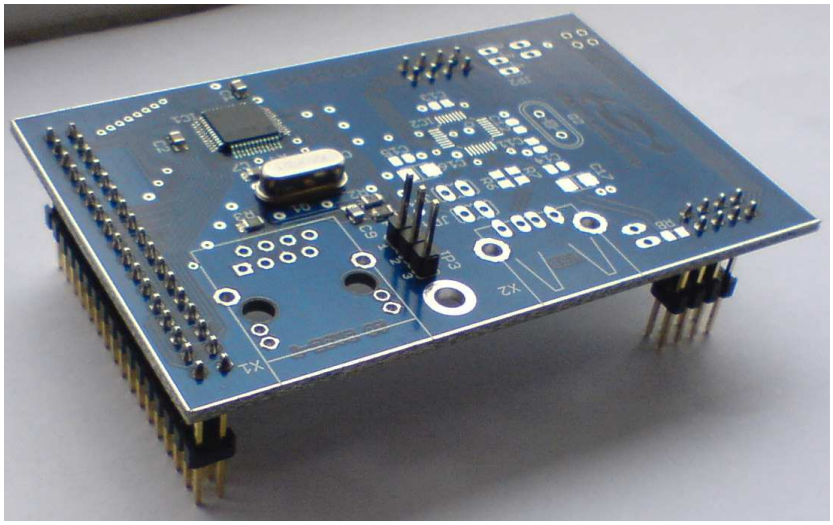
Obr. 21 Spodní strana desky EthernetCP2200 (1:1).



Obr. 22 Osazení horní strany desky EthernetCP2200.



Obr. 23 Osazení spodní strany desky EthernetCP2200.



Obr. 24 Fotografie modulu EthernetCP2200.

5 SW moduly

Všechny softwarové moduly jsou psány v prostředí *Keil uVision3* v jazyce *C*. Zdrojový kód každého modulu je uložen do souboru ****.c***. Každý modul má definovány prototypy funkcí, případně nějaká makra, tyto jsou pak uloženy v hlavičkovém souboru ****.h***. Všechny soubory budou k dispozici na přiloženém CD.

Seznam vytvořených modulů:

- Modul ADC - obsluha A/D převodníku.
- Modul I2C - obsluha sběrnice I2C.
- Modul SPI - obsluha sběrnice SPI.
- Modul LCD - obsluha LCD displeje.
- Modul UART - obsluha sériové linky.

- Modul PHY_LINK - obsluha eth. řadiče CP2200 (fyzická a linková vrstva).
- Modul NETWORK - modul síťové vrstvy.
- Modul TRANSPORT - modul transportní vrstvy.
- Modul APPLICATION - modul aplikační vrstvy.

- Modul TIMER - obsluha časovače.
- Modul BOARD - modul desky (konfigurace HW).
- Modul MAIN - hlavní modul (obsahuje funkci `main()`).

5.1 typedefs.h

Protože jsou v softwarových modulech některé datové typy přetypovány nebo jsou použity datové struktury či uniony, bude zde uveden výpis souboru, ve kterém je vše definováno.

Výpis souboru *typedefs.h*:

```
typedef unsigned char byte;
typedef unsigned int  word;
//------------------------------------------------------------------------
typedef union UINT
{
  unsigned int  Int;
  unsigned char Char[2];
} UINT;
//------------------------------------------------------------------------
typedef union ULONG
{
  unsigned long Long;
  unsigned int  Int[2];
  unsigned char Char[4];
} ULONG;
//------------------------------------------------------------------------
```

```
typedef union MACADDRESS
{
  unsigned int  Int[3];
  unsigned char Char[6];
} MACADDRESS;
//---------------------------------------------------------------------------
typedef union IPADDRESS
{
  unsigned long Long;
  unsigned int  Int[2];
  unsigned char Char[4];
} IPADDRESS;
//---------------------------------------------------------------------------
struct t_client
{
      byte              state;      //aktualni stav spojeni
      word              time;       //timeout,  dekrementuje  se  po  urcitem
intervalu, az je 0 nastal timeout
      MACADDRESS        mac;        //MAC adresa stanice, ktera se pripojila
      IPADDRESS         ip;         //IP adresa stanice, ktera se pripojila
      UINT              port;       //port  stanice,  ktera  se  pripojila,  na
tento se bude posilat (port clienta)
      UINT              xport;      //port, na ktery se stanice pripojila
      ULONG             xseq;       //posledni prijate seq cislo
      ULONG             seq;        //seq cislo, ktere se bude posilat
      byte              page;       //stranka odesilanych dat
      word              addr;       //adresa odesilanych dat
      byte              flag;       //odeslany priznakovy Bajt
      UINT              len;        //delka odeslanych dat
      UINT              xlen;       //delka prijatych dat
      byte              xflag;      //prijaty priznakovy Bajt
      unsigned long     size;       //celkova velikost odesilanych dat
      byte              err;        //pocitani chyb
      byte              stat;       //(cislo souboru, který se bude posilat)
};
//---------------------------------------------------------------------------
struct t_files                      //            (32B)
{
      byte              name[24];   //jmeno souboru   (24B)
      byte              scode;      //          (1B)
      byte              page;       //stranka  (1B)
      word              addr;       //adresa   (2B)
      unsigned long     size;       //velikost (4B)
};
//---------------------------------------------------------------------------
struct t_stat_files
{
      word              addr;       //adresa
      word              size;       //velikost
};
```

5.2 Modul ADC

Funkce modulu ADC jsou určeny pro obsluhu A/D převodníku mikrokontroléru C8051F120.

Soubory modulu ADC:

Zdrojové:	**adc.c**
Hlavičkové:	**adc.h**

Proměnné definované v modulu ADC:

žádné.

Modul zahrnuje následující funkce:

5.2.1 ADC_INIT

Popis: Funkce slouží k inicializaci A/D převodníku.

Prototyp: `ADC_Init(void);`

Příklad volání: `ADC_Init();`

Parametry: **žádné.**

Vrácená hodnota: **žádná.**

5.2.2 get_ADC

Popis: Funkce vrací průměr 256 změřených hodnot daného kanálu převodníku se zvolenými paramerty.

Prototyp: `word get_ADC(byte channel, byte gain);`

Příklad volání: `Ain0 = get_ADC(0x00, ADC_gain1);`

Parametry: **1.** *channel* – typ *unsigned char*, kanál převodníku, který se bude měřit. Možné hodnoty:
ADC_chan_temp, 0 až 7, ostatní viz [3].
2. *gain* - typ *unsigned char*, zisk předzesilovače převodníku.
Možné hodnoty:
ADC_gain_temp, ADC_gain_1, ADC_gain_2, ADC_gain_4, ADC_gain_8, ADC_gain_16, ADC_gain_05
– zesílení: temp(2);1; 2; 4; 8; 16; 0,5

Vrácená hodnota: **ADC_val** – typ *unsigned int*, průměr 256 naměřených hodnot.

5.3 Modul I2C

Funkce modulu I2C jsou určeny pro obsluhu SMBus periferie mikrokontroléru C8051F120. SMBus je plně kompatibilní se sběrnicí I^2C.

Soubory modulu I2C:
Zdrojové: **i2c.c**
Hlavičkové: **i2c.h**

Proměnné definované v modulu I2C:

žádné.

Modul zahrnuje následující funkce:

5.3.1 I2C_INIT

Popis: Funkce slouží k inicializaci SMBus (I^2C) a nastavení rychlosti komunikace.

Prototyp:
```
void I2C_Init(byte clock_rate);
```

Příklad volání:
```
I2C_Init(c_i2cCR50);
```

Parametry: **1.** *clock_rate* – typ *unsigned char*, udává rychlost sběrnice
Možné hodnoty:
(v závislosti na taktu 8051 viz [3]):
c_i2cCR50 – cca 50kHz, při taktu 33,3MHz.

Vrácená hodnota: **žádná.**

5.3.2 I2C_wrAddrs

Popis: Funkce zapíše do zařízení na sběrnici I^2C (dané adresou zařízení) adresu paměťové oblasti, ze které se bude číst, nebo na kterou se bude zapisovat. Funkce je pouze pomocná, je volána jinými funkcemi a samostatně ji volat nelze. Funkce vrací kód výsledku operace (úspěch nebo chybový kód).

Prototyp: `byte I2C_wrAddrs(byte dev_addr, word addr);`

Příklad volání: `status = I2CwrAddrs(dev_RTC, RTC_ctrl);`

Parametry:
1. *dev_addr* – typ *unsigned char*, adresa zařízení.
Možné hodnoty:
dev_EEP – adresa zařízení 24C08 (sériová eeprom).
dev_RTC – adresa zařízení RCF8583 (obvod reálného času).
2. *addr* – typ *unsigned int*, adresa v paměťovém prostoru.
Možné hodnoty:
Libovolné (Podle zařízení).

Vrácená hodnota: **status** – typ *unsigned char*.
NO_ERROR – operace proběhla úspěšně.
Chybové kódy:
eI2C_FF – chyba start bitu.
eI2C_F0 – chyba device (nebyl ACK).
eI2C_E1 – chyba při zápisu vyššího bajtu adresy.
eI2C_E0 – chyba při zápisu nižšího bajtu adresy.

Pozn.:
dev_addr **-** Nejnižší bit udává, bude-li ***addr*** (adresa paměť. prostoru) zapisována do zařízení jako jedno nebo dvoubajtová. Pro dvoubajtovou adresu je nejnižší bit nastaven (např. sériové eeprom 24C64). Pro jednobajtovou adresu je nejnižší bit vynulován (např. pro RCF8583, nebo i 24C08* - zde se ovšem z dev_addr berou nějaké bity, které tvoří vyšší bity adresy *addr* viz [16]).

5.3.3 I2C_trans

Popis: Funkce zapíše do zařízení dané *dev_addr* na adresu danou *addr* data *I2C_data*. Funkce vrací kód výsledku operace (úspěch nebo chybový kód).

Prototyp:
```
byte I2C_trans(byte dev_addr,word addr,
               byte I2C_data);
```

Příklad volání: `status = I2C_trans(dev_RTC,RTC_ctrl,0x04);`

Parametry:
1. *dev_addr* – typ *unsigned char*, adresa zařízení.
Možné hodnoty:
dev_EEP – adresa zařízení 24C08 (sériová eeprom).
dev_RTC – adresa zařízení RCF8583 (obvod reálného času).
2. *addr* – typ *unsigned int*, adresa v paměťovém prostoru.
Možné hodnoty:
Libovolné (Podle zařízení).
3. *data* – typ *unsigned char*, data zapisovaná na danou adresu.

Vrácená hodnota: **status** – typ *unsigned char*.
NO_ERROR – operace proběhla úspěšně.
Chybové kódy:
eI2C_D0 - chyba při zápisu datového bajtu.
chybové kódy funkce I2C_wrAddrs.

5.3.4 I2C_transbuff

Popis: Funkce zapíše do zařízení dané *dev_addr* data od adresy dané *addr* z bufferu *buff*. Funkce vrací kód výsledku operace (úspěch nebo chybový kód).

Prototyp:
```
byte I2C_transbuff(byte dev_addr,
                   byte *buff, word addr,
                   byte count);
```

Příklad volání:
```
status = I2C_transbuff(dev_RTC,
                       datetime,0x00,
                       sizeof(datetime));
```

Parametry:
1. *dev_addr* – typ *unsigned char*, adresa zařízení.
Možné hodnoty:
dev_EEP – adresa zařízení 24C08 (sériová eeprom).
dev_RTC – adresa zařízení RCF8583 (obvod reálného času).
2. ***buff* – adresa bufferu, ze kterého se budou brát data.
3. *addr* – typ *unsigned int*, adresa v paměťovém prostoru.
Možné hodnoty:
Libovolné (Podle zařízení).
4. *count* – typ *unsigned char*, velikost zapisovaných dat.
(hodnota 0 odpovídá 256B).

Vrácená hodnota: **status** – typ *unsigned char*.
NO_ERROR – operace proběhla úspěšně.
Chybové kódy:
eI2C_D0 - chyba při zápisu některého datového bajtu.
chybové kódy funkce I2C_wrAddrs.

5.3.5 I2C_receive

Popis: Funkce přečte ze zařízení dané *dev_addr* z adresy *addr* jeden bajt dat. Funkce vrací hodnotu dvoubajtovou, data jsou umístěny v horním bajtu a to jen v případě nulového nižšího bajtu (nižší bajt je kód výsledku operace).

Prototyp: `int I2C_receive(byte dev_addr,word addr);`

Příklad volání: `stat16 = I2C_trans(dev_RTC,RTC_ctrl);`

Parametry:
1. *dev_addr* – typ *unsigned char*, adresa zařízení.
Možné hodnoty:
dev_EEP – adresa zařízení 24C08 (sériová eeprom).
dev_RTC – adresa zařízení RCF8583 (obvod reálného času).
2. *addr* – typ *unsigned int*, adresa v paměťovém prostoru.
Možné hodnoty:
Libovolné (Podle zařízení).

Vrácená hodnota: **stat16** – typ *int*.
Nižší bajt:
NO_ERROR – operace proběhla úspěšně. Vyšší bajt potom nese přečtená data.
Chybové kódy:
eI2C_8F - chyba opakovaného start bitu.
eI2C_80 - chyba dev_addr+R.
eI2C_82 - chyba NACK.
chybové kódy funkce I2C_wrAddrs.

Vyšší bajt:
Přečtená data – pouze je-li nižší bajt 0, jinak neplatný.

5.3.6 I2C_recbuff

Popis: Funkce přečte ze zařízení dané *dev_addr* od adresy *addr* blok dat daný velikostí *count* a přečtená data uloží do *buff*. Funkce vrací kód výsledku operace (úspěch nebo chybový kód).

Prototyp:
```
byte I2C_recbuff(byte dev_addr,
                 byte *buff, word addr,
                 byte count);
```

Příklad volání:
```
status = I2C_transbuff(dev_RTC,
                       datetime,0x00,
                       sizeof(datetime));
```

Parametry:
1. *dev_addr* – typ *unsigned char*, adresa zařízení.
Možné hodnoty:
dev_EEP, *dev_RTC*, *apod.*
2. ***buff* –** adresa bufferu, do kterého se budou ukládat data.
3. *addr* – typ *unsigned int*, adresa v paměťovém prostoru.
Možné hodnoty:
Libovolné (Podle zařízení).
4. *count* – typ *unsigned char*, velikost zapisovaných dat.
(hodnota 0 odpovídá 256B).

Vrácená hodnota: **status** – typ *unsigned char*.
NO_ERROR – operace proběhla úspěšně.
Chybové kódy:
eI2C_8F - chyba opakovaného start bitu.
eI2C_80 - chyba dev_addr+R.
eI2C_81 – chyba ACK.
eI2C_82 - chyba NACK.
chybové kódy funkce I2C_wrAddrs.

5.4 Modul SPI

Funkce modulu SPI jsou určeny pro obsluhu SPI periferie mikrokontroléru C8051F120. SPI je konfigurováno jako *Master* s čtyřvodičovým připojením.

Soubory modulu SPI:
Zdrojové: **spi.c**
Hlavičkové: **spi.h**

Proměnné definované v modulu SPI:

žádné.

Modul zahrnuje následující funkce:

SPI_INIT
SPI_exchange
SPI_SS0
SPI_SS1

5.4.1 SPI_INIT

Popis: Funkce slouží k inicializaci SPI, nastavení rychlosti komunikace a konfiguraci režimu přenosu dat.

Prototyp:
```
void SPI_Init (byte cfg, byte clock_rate);
```

Příklad volání:
```
SPI_Init(c_spiPHA1+c_spiPOL1,c_spiCR100);
```

Parametry: **1.** *cfg* – typ *unsigned char*, udává konfiguraci přenosu dat.
Možné hodnoty:
c_spiPHA1 – fáze (druhá hrana SCK).
c_spiPOL1 – polarita (SCK je log. 1 v *iddle* stavu).
(kombinace).

2. *clock_rate* – typ *unsigned char*, udává rychlost sběrnice.
Možné hodnoty:
(v závislosti na taktu 8051 viz [3]):
c_spiCR100 – cca 100kHz, při taktu 33,3MHz.

Vrácená hodnota: **žádná.**

5.4.2 SPI_exchange

Popis: Funkce slouží k výměně dat mezi master/slave. Master (8051) odešle na sběrnici bajt *SPI_data* a zároveň přijímá bajt od Slave. Zařízení, se kterým se má komunikovat, musí mít předem nastaven signál NSS na log. 0 a tato úroveň musí být udržována po celou dobu přenosu bajtu.

Prototyp:
```
byte SPI_exchange(byte SPI_data);
```

Příklad volání:
```
rxdata = SPI_exchange(0x55);
```

Parametry: **1.** *data* – typ *unsigned char*, data, která se odešlou po sběrnici.

Vrácená hodnota: **rxdata** – typ *unsigned char***,** přijatá data ze sběrnice.

5.4.3 SPI_SS0

Popis: Nejedná se o funkci, ale pouze makro (skok a návrat z podprogramu by byl zbytečný). Nastaví signál *Slave Select* (NSS) na log. 0, čímž zařízení připojené na signál NSS aktivuje, tzn., že se zařízením bude možné potom komunikovat. Pro více zařízení na sběrnici SPI je nutné, aby každé mělo svůj signál NSS, tedy by bylo nutné definovat výstupní linku se signálem NSS pro každé další zařízení.

Prototyp:
```
(#define SPI_SS0() NSSMD0 = 0)
```

Příklad volání:
```
SPI_SS0();
```

Parametry: **žádné.**

Vrácená hodnota: **žádná.**

5.4.4 SPI_SS1

Popis: Nejedná se o funkci, ale pouze makro (skok a návrat z podprogramu by byl zbytečný). Nastaví signál *Slave Select* na log. 1, čímž připojené zařízení deaktivuje (další posílání dat po sběrnici bude daným zařízením ignorováno).

Prototyp: `(#define SPI_SS1() NSSMD0 = 1)`

Příklad volání: `SPI_SS1();`

Parametry: **žádné.**

Vrácená hodnota: **žádná.**

5.5 Modul LCD

Funkce modulu LCD jsou určeny pro obsluhu LCD displeje. Typ použitého displeje je BTHQ22005VSS. Jedná se dvouřádkový displej s 20-ti znaky na řádek. Komunikace s displejem je čtyřbitová. Komunikace je zjednodušená tak, že se čeká na vykonání instrukce displeje pevnou dobu (netestuje se příznak zaneprázdnění).

Soubory modulu LCD:

Zdrojové:	**lcd.c**
Hlavičkové:	**lcd.h**

Proměnné definované v modulu LCD:

žádné.

Modul zahrnuje následující funkce:

LCD_INIT
LCD_putins
LCD_put
LCD_string
LCD_text
LCD_hex
LCD_setpos

5.5.1 LCD_INIT

Popis: Funkce slouží k inicializaci LCD displeje. Nastaví čtyřbitovou komunikaci, 2 řádky, vypne kurzor, vymaže displej, zapne posuv kurzoru, vypne posuv displeje a nastaví kurzor na první řádek, první znak. (Případně definuje znaky s ordinální hodnotou 0 až 7).

Prototyp: `LCD_Init(void);`

Příklad volání: `LCD_Init();`

Parametry: **žádné.**

Vrácená hodnota: **žádná.**

5.5.2 LCD_putins

Popis: Funkce zapíše instrukci do displeje.

Prototyp: `void LCD_putins(byte lcd_data);`

Příklad volání: `LCD_putins(0x80);`

Parametry: **1.** *data* – typ *unsigned char*, instrukce zapisovaná do displeje.
Možné hodnoty: viz [8], [13].

Vrácená hodnota: **žádná.**

5.5.3 LCD_put

Popis: Funkce zapíše znak do displeje na aktuální pozici kurzoru.

Prototyp: `void LCD_put(byte lcd_data);`

Příklad volání: `LCD_putins('A');`

Parametry: **1.** *data* – typ *unsigned char*, instrukce zapisovaná do displeje.
Možné hodnoty: viz [8], [13].
0x00 – 0x07; 0x20 – 0x7F; (0xA0 - 0xFF)

Vrácená hodnota: **žádná.**

5.5.4 LCD_string

Popis: Funkce vypíše na LCD displeji řetězec od aktuální pozice kurzoru.

Prototyp: `void LCD_string(char code *string);`

Příklad volání: `LCD_string(”Ahoj”);`

Parametry: **1. ***_string_ – (ukazatel na) nulou zakončený řetězec v datové oblasti *code*.

Vrácená hodnota: **žádná.**

5.5.5 LCD_text

Popis: Funkce vypíše na LCD displeji řetězec na zadaný řádek.

Prototyp: `void LCD_text(byte *src,byte line);`

Příklad volání: `LCD_text(&text_01,line1);`

Parametry: **1. ***_src_ – adresa pole 20-ti znaků.
2. *line* – typ *unsigned char*, číslo řádku, na který se znaky vypíší.
Možné hodnoty:
line1 – první řádek.
line2 – druhý řádek.

Vrácená hodnota: **žádná.**

5.5.6 LCD_hex

Popis: Funkce vypíše na aktuální pozici kurzoru dva znaky vyjadřující hexadecimální hodnotu vstupního bajtu.

Prototyp: `void LCD_hex(byte hex_data);`

Příklad volání: `LCD_hex(0xAA);`

Parametry: **1.** *data* – typ *unsigned char*, data, která se mají v hexa formátu zobrazit na displeji..

Vrácená hodnota: **žádná.**

5.5.7 LCD_setpos

Popis: Funkce nastaví kurzor na danou pozici.

Prototyp: `void LCD_setpos(byte line, byte col);`

Příklad volání: `LCD_setpos(line1,0);`

Parametry: **1**. *line* – typ *unsigned char*, číslo řádku (číslováno od 0), na který se přesune kurzor.
Možné hodnoty:
line1 – první řádek.
line2 – druhý řádek.
2. *col* – typ unsigned char, číslo sloupce (číslováno od 0), na který se přesune kurzor.
Možné hodnoty:
0x00 –0x13

Vrácená hodnota: **žádná.**

5.6 Modul UART

Funkce modulu UART jsou určeny pro obsluhu sériové sběrnice mikrokontroléru C8051F120. Využívá se přerušení od přijatého a odeslaného bajtu. Jsou zavedeny kruhové buffery pro vysílání a příjem. Data na odeslání se zapisují do *tbuf* a o odesílání se stará přerušovací systém. Stejně tak přijatá data se ukládají do *rbuf* a software voláním příslušné funkce případná data vyčte.

Parametry linky: start bit, 8 datových bitů, žádný paritní bit, jeden stop bit.

Soubory modulu UART:

Zdrojové:	**uart.c**
Hlavičkové:	**uart.h**

Proměnné definované v modulu UART:

`static byte xdata tbuf[TBUF_SIZE];`
kruhový vysílací buffer sériové linky.
Typ: *unsigned char[]*, Umístění: *xdata (on chip)*, Velikost *TBUF_SIZE*.

`static byte xdata rbuf[RBUF_SIZE];`
kruhový přijímací buffer sériové linky.
Typ: *unsigned char[]*, Umístění: *xdata (on chip)*, Velikost *RBUF_SIZE*.

`static byte data t_in, t_out, r_in, r_out;`
ukazatele začátku, resp. konce dat ve vysílacím, resp. v přijímacím bufferu.
Typ: *unsigned char*, Umístění: *data*, Velikost *1B (každá)*.

```
static bit ti_restart;
```
Nastavuje se při vyprázdnění *tbuf*, značí, že se přerušení musí generovat softwarově (aby se nastartovalo posílání dat).
Typ: *bit*, (Umístění: *data*), Velikost: 1bit.

Modul zahrnuje následující funkce:

5.6.1 UART0_INIT

Popis: Funkce slouží k inicializaci UART a nastavení rychlosti komunikace (baudrate).

Prototyp: `void UART0_Init(byte baudrate);`

Příklad volání: `UART0_Init(br_115200);`

Parametry: **1.** *baudrate* – typ *unsigned char*, udává přenosovou rychlost.
Možné hodnoty:
(v závislosti na taktu 8051 viz [3]):
br_115200 – přenosová rychlost 115200Bd, při taktu 33,3MHz.

Vrácená hodnota: **žádná.**

5.6.2 UART0_putchar

Popis: Funkce uloží vstupní bajt do vysílací fronty – do vysílacího bufferu. Pokud je vysílací buffer plný, funkce vrací hodnotu *0xFF (-1)*, jinak vrací hodnotu *0*.

Prototyp: `byte UART0_putchar(byte c);`

Příklad volání: `status = UART0_putchar('A');`

Parametry: **1.** *char* – typ *unsigned char*, znak, který se má odeslat (bude uložen do *tbuf*).

Vrácená hodnota: **status –** typ *unsigned char*.
False (NO_ERROR) – operace proběhla úspěšně.
True – chyba, vysílací buffer je plný.

5.6.3 UART0_getchar

Popis: Funkce přečte přijatý bajt z přijímacího bufferu. V případě, že je přijímací buffer prázdný, funkce vrátí hodnotu *0xFFFF (–1)*, jinak vrací přijatý bajt (ve spodním bajtu vrácené dvoubajtové hodnoty, vyšší bajt bude nulový).

Prototyp: `int UART0_getchar(void);`

Příklad volání: `stat16 = UART0_getchar();`

Parametry: **žádné.**

Vrácená hodnota: **stat16 –** typ *int*.
Nižší bajt:
Platná data – pouze v případě, je-li vyšší bajt 0.

Vyšší bajt:
False (NO_ERROR) – nižší bajt obsahuje platná data.
True – přijímací buffer prázdný, žádná platná data.

5.6.4 UART0_putstring

Popis: Funkce pošle po sériové lince daný řetězec, navíc pošle znaky konce řádku *CR* a *LF* (*0x0D* a *0x0A*).

Prototyp: `void UART0_putstring(char code *string);`

Příklad volání: `UART0_putstring(”Ahoj”);`

Parametry: **1. ****string* – (ukazatel na) nulou zakončený řetězec v datové oblasti *code*.

Vrácená hodnota: **žádná.**

5.6.5 UART0_ISR

Popis: Obsluha přerušení od sériové linky. V případě, že přerušení bylo vyvoláno odesláním znaku, obsluha odešle další bajt z vysílacího bufferu. V případě přerušení od přijatého znaku, se přijatá data uloží do přijímacího bufferu.

Prototyp: `interrupt 4`

Příklad volání: `(interrupt)`

Parametry: **žádné (interrupt).**

Vrácená hodnota: **žádná (interrupt).**

5.7 Modul PHY_LINK

Funkce modulu PHY_LINK jsou určeny pro obsluhu ethernetového řadiče CP2200. Modul odpovídá vrstvě síťového rozhraní podle modelu TCP/IP, nebo podle ISO OSI vrstvě fyzické a linkové.

Soubory modulu CP2200:
Zdrojové: **phy_link.c, cp2200_reg.c**
Hlavičkové: **phy_link.h, cp2200_reg.h**

Proměnné definované v modulu PHY_LINK:

```
extern MACADDRESS data MYMAC;
```
MAC adresa obvodu CP2200.
Typ: *MACADDRESS* viz kap. 5.1, Umístění: *data* (*může být *code*), Velikost *6B.*

```
extern MACADDRESS data DESTMAC;
```
MAC adresa cíle, používá se při posílání paketu.
Typ: *MACADDRESS* viz kap. 5.1, Umístění: *data*, Velikost *6B.*

```
extern byte xdata RX_BUFF[1518];
```
přijímací buffer, přijatý paket se umístí do tohoto bufferu.
Typ: *unsigned char[]*, Umístění: *xdata(on chip),* Velikost *1518B (6+6+2+1500+4).*

```
extern byte xdata TX_BUFF[1500];
```
vysílací buffer, obsahuje uživatelská data linkového rámce, která se budou posílat.
Typ: *unsigned char[]*, Umístění: *xdata(on chip),* Velikost *1500B.*

```
extern byte data PacketInfo;
```
odpovídá registru CPINFOH právě načteného paketu.
Typ: *unsigned char*, Umístění: *data*, Velikost *1B.*

Proměnné typu *bit*, (slouží k diagnostice):
```
extern bit LinkStatusChange;
extern bit LinkStatus;
extern bit EthernetInit;
extern bit inhibit;
extern bit nmp;
extern bit RxFull;
```

Modul zahrnuje následující funkce:

5.7.1 CP_INIT

Popis: Funkce slouží k inicializaci kontroléru CP2200, kontrolér se resetuje a nastaví do požadovaného módu. Navíc se načte unikátní MAC adresa do proměnné *MYMAC* z paměti *FLASH* řadiče CP2200.

Prototyp:
```
byte CP_Init(byte eth_mode);
```

Příklad volání:
```
status = CP_Init(MODE_Autonegotiation);
```

Parametry:
1. *eth_mode* – typ *unsigned char*, udává mód, ve kterém se CP2200 pokusí připojit do sítě.
Možné hodnoty:
MODE_Autonegotiation – připojení funkcí Autoneg. viz [4]
MODE_FullDuplex – plně duplexní připojení
MODE_HalfDuplex – poloduplexní připojení

Vrácená hodnota:
status – typ *unsigned char*.
NO_ERROR – pro úspěšnou inicializaci.
Chybové kódy:
RSTHI_ERROR – chyba uvolnění RST pinu.
OSC_ERROR – chyba oscilátoru.
SELFINIT_ERROR – chyba vlastní inicializace.
EMIF_ERROR – chyba rozhraní.
MEMAA_ERROR, *MEM55_ERROR* – chyby zápisu do bufferu.
LASTRST_ERROR – poslední reset nebyl způsoben pinem RST.
AUTONEG_ERROR – chyba funkce Autonegotiation.
AUTONEGTIMEOUT_ERROR – timeout Autonegotiation.

5.7.2 CP_LinkStatus

Popis: Funkce vrací hodnotu registru *PHYCN* kontroléru CP2200. Bit 0 odpovídá stavu připojení (*true* – připojeno, *false* - odpojeno).

Prototyp: `byte CP_LinkStatus(void);`

Příklad volání: `status = CP_LinkStatus();`

Parametry: **žádné.**

Vrácená hodnota: **status** – typ *unsigned char*.
Vrací hodnotu registru **PHYCN** viz [4].

5.7.3 CP_ISR

Popis: Obsluha přerušení od kontroléru CP2200. Nastavuje diagnostické bity, při příjmu paketu kontroluje zaplnění bufferu CP2200 a případně zakazuje příjem dalších paketů.

Prototyp: `interrupt 0`

Příklad volání: `(interrupt)`

Parametry: **žádné (interrupt).**

Vrácená hodnota: **žádná (interrupt).**

5.7.4 CP_RxValid

Popis: Funkce testuje platnost přijatého paketu, je-li paket platný, musí se načíst funkcí CP_Receive.

Prototyp: `byte CP_RxValid(void);`

Příklad volání: `status = CP_RxValid();`

Parametry: **žádné**

Vrácená hodnota: **status** – typ *unsigned char*.
False – Paket nebyl přijat.
True (RXVALID) – Paket byl přijat.

5.7.5 CP_Receive

Popis: Funkce načte přijatý paket z kontroléru CP2200 do přijímacího bufferu *RX_BUFF*. V případě vyprázdnění bufferu CP2200 povolí příjem dalších paketů (mohl být totiž zakázaný).

Prototyp: `word CP_Receive(void);`

Příklad volání: `packet_len = CP_Receive();`

Parametry: **žádné.**

Vrácená hodnota: **length** – typ *unsigned int.*
Velikost přijatého paketu. V případě hodnoty 0 nebyl paket přijat zcela korektně.

5.7.6 CP_send

Popis: Funkce odešle paket na zadanou MAC adresu cíle, paket bude mít zadanou délku a zadaný typ. Data paketu se budou brát ze zadané adresy.

Prototyp:
```
void CP_send(MACADDRESS *pDESTMAC,
             byte *buffer,
             word b_length,
             word packet_type);
```

Příklad volání: CP_send(&DESTMAC,TX_BUFF,0x20,type_IP);

Parametry:
1. *dest_mac* – adresa proměnné typu *MACADDRESS*, proměnná je MAC adresa cíle.
2. ***buffer** – adresa bufferu, odkud se budou brát data.
3. *length* – typ *unsigned int*, velikost uživatelských dat linkového rámce.
4. *packet_type* – typ *unsigned int*, udává typ linkového rámce.
Možné hodnoty:
TYPE_IP – typ ethernetového rámce bude nastaven na typ *IP*.
TYPE_ARP – typ ethernetového rámce bude nastaven typ *ARP*.

Vrácená hodnota: **žádná**.

5.7.7 CP_send_pause

Popis: Funkce odešle řídící MAC rámec *PAUSE*. Tento rámec informuje protizařízení na linkové vrstvě, že má na daný čas pozastavit vysílání (např. případ zahlcení přijímacího bufferu). Čas je daný parametrem, který je násobkem 51,2us. Tento paket je možné posílat pouze ve *Full Duplex* módu a protistrana jej musí podporovat (nastavený příslušný bit v *Autonegotiation Page*). Odeslání rámce s parametrem 0 okamžitě povolí protistraně vysílat, i když ještě nevypršel čas určený předchozím *pause* rámcem.

Prototyp: `void CP_send_pause (word pause);`

Příklad volání: CP_send_pause(1000);

Parametry: **1.** *pause* – typ *unsigned int*, udává násobek 51,2us, po který protistrana nebude posílat pakety.

Vrácená hodnota: **žádná**.

5.7.8 CP_flash_rdByte

Popis: Funkce přečte z paměti flash kontroléru CP2200 z dané adresy jeden bajt.

Prototyp: `byte CP_flash_rdByte(word Addr);`

Příklad volání: `data = CP_flash_rdByte(word Addr);`

Parametry: **1.** *Addr* - typ *unsigned int*, adresa paměťové pozice, ze které se bude číst.
Možné hodnoty:
0x0000 – 0x1FFF

Vrácená hodnota: **data –** typ *unsigned char*. Vrací přečtená data.

5.7.9 CP_flash_rdBuff

Popis: Funkce přečte z paměti *FLASH* kontroléru CP2200 z dané adresy zadaný počet bajtů do bufferu dané adresy.

Prototyp:
```
void CP_flash_rdBuff(word Addr,
                     byte *buffer,
                     word count);
```

Příklad volání: `CP_flash_rdBuff(0x0000,&buff,8);`

Parametry: **1.** *Addr* – typ *unsigned int*, adresa paměťové pozice, ze které se bude číst.
Možné hodnoty:
0x0000 – 0x1FFF
2. **buffer* – adresa bufferu, kam se uloží přečtená data.
3. *count* – velikost čtených dat.

Vrácená hodnota: **žádná**.

5.7.10 CP_flash_pgErase

Popis: Funkce smaže jednu stránku paměti FLASH kontroléru CP2200. Stránky jsou 512B. Číslo stránky je dáno adresou (Addr>>9). Poslední stránka (od Addr=0x1E00) se nemaže, protože obsahuje unikátní MAC danou výrobcem.

Prototyp:
```
byte CP_flash_pgErase(word Addr);
```

Příklad volání:
```
status = CP_flash_pgErase(word Addr);
```

Parametry: **1.** Addr: typ *unsigned int*, adresa stránky paměti FLASH, která se bude mazat.
Možné hodnoty:
0x0000 – 0x1DFF

Vrácená hodnota: **status –** typ *unsigned char*,
NO_ERROR – operace proběhla úspěšně.
Chybové kódy:
eCP_flashEF – neplatná adresa.
eCP_flashE7 – samovolný reset (problém s napájením).
eCP_flashE0 – timeout při mazání.

5.7.11 CP_flash_wrByte

Popis: Funkce zapíše jeden bajt na danou adresu paměti FLASH kontroléru CP2200. Na posledních 6-ti bajtech je uložena unikátní MAC daná výrobcem, proto se na tyto adresy zapisovat nebude.

Prototyp:
```
byte CP_flash_wrByte(word Addr,
                     byte wrdata);
```

Příklad volání:
```
status = CP_flash_wrByte(0x0000,'A');
```

Parametry: **1.** *Addr* - typ *unsigned int*, adresa místa, na které se bude zapisovat.
Možné hodnoty:
0x0000 – 0x1FF9
2. *wrdata* – typ *unsigned char*, zapisovaná data.

Vrácená hodnota: **status –** typ *unsigned char*,
NO_ERROR – operace proběhla úspěšně.
Chybové kódy:
eCP_flashEF – neplatná adresa.
eCP_flashE7 – samovolný reset (problém s napájením).
eCP_flashE0 – timeout při zápisu.

5.8 Modul NETWORK

Funkce modulu NETWORK spadají do síťové vrstvy a mají za úkol řešit síťové protokoly IP, ARP a ICMP.

Soubory modulu NETWORK:
Zdrojové: **network.c**
Hlavičkové: **network.h**

Proměnné definované v modulu NETWORK:

```
extern IPADDRESS data MYIP;
```
IP adresa jednotky.
Typ: *IPADDRESS* viz kap. 5.1, Umístění: *data*, Velikost *4B*.

```
extern IPADDRESS data DESTIP;
```
IP adresa cíle, používá se při posílání paketu.
Typ: *IPADDRESS* viz kap. 5.1, Umístění: *data*, Velikost *4B*.

Modul zahrnuje následující funkce:

protokol_ARP
protokol_IP
ICMP

5.8.1 protokol_ARP

Popis: Funkce se volá při příjmu linkového rámce s hodnotou *type=0x0806* (tím je dáno, že paket nese protokol ARP). Funkce zkontroluje , zda je dotaz na překlad IP adresy na MAC určen této jednotce (dotazovaná IP odpovídá hodnotě *MYIP*). V tom případě se odešle linkový rámec s odpovědí ARP s vyplněnou MAC adresou na hodnotu *MYMAC*. Funkce netestuje, zda se jedná o protokol ARP nebo RARP, testování IP je dostačující.

Prototyp: `void protokol_ARP(void);`

Příklad volání: `protokol_ARP();`

Parametry: **žádné.**

Vrácená hodnota: **žádná.**

5.8.2 protokol_IP

Popis: Funkce se volá při příjmu linkového rámce s hodnotou *type=0x0800* (tím je dáno, že paket nese protokol IP). Funkce kontroluje, zda-li je IP datagram fragmentován, pokud ano, takový datagram je ignorován. Jestliže fragmentován není, testuje se IP adresa cíle, ta musí odpovídat *MYIP*. Pokud ne, datagram se ignoruje, pokud IP adresa odpovídá, pak podle pole *protokol vyšší vrstvy* je volána příslušná funkce vyšší vrstvy, případně funkce ICMP. Pozn.: Kontrolní součet IP není kontrolován.

Prototyp: `void protokol_IP(void);`

Příklad volání: `protokol_IP();`

Parametry: **žádné.**

Vrácená hodnota: **žádná.**

5.8.3 ICMP

Popis: Funkce je volána z funkce *protokol_IP*, pokud IP datagram nese protokol ICMP. Tato funkce testuje, zda jde o požadavek *ECHO* (příkaz ping), pokud ano, funkce odešle odpověď na tento požadavek. Využívá se přijatého paketu – pouze se změní potřebné hodnoty a odešle se jako odpověď. Pokud nešlo o požadavek *ECHO*, je paket ignorován.

Prototyp: `void ICMP(void);`

Příklad volání: `ICMP();`

Parametry: **žádné.**

Vrácená hodnota: **žádná.**

5.9 Modul TRANSPORT

Funkce modulu TRANSPORT spadají do transportní vrstvy a mají za úkol řešit transportní protokoly TCP a UDP.

Soubory modulu TRANSPORT:

Zdrojové:	**transport.c**
Hlavičkové:	**transport.h**

Proměnné definované v modulu TRANSPORT:

`extern struct t_client xdata` **`clients[max_conn];`**
Datová struktura sdružující informace o socketu (klientovi), obsahuje informace jako je IP adresa, MAC adresa, ukazatel odesílaných dat, stav TCP, apod.
Typ: *t_client[]* viz kap. 5.1, Umístění: *data*, Velikost: *max_conn*40B*.

`byte data` **`UDP_lock;`**
Používá se k uzamčení komunikace protokolem UDP (v případě chyby, nebo úmyslu).
Typ: *unsigned char*, Umístění: *data*, Velikost: *1B*.

`extern word` **`IP_identify;`**
Identifikace IP datagramu, hodnota se odesílá v hlavičce IP datagramu, s každým odeslaným IP datagramem se hodnota proměnné inkrementuje.
Typ: *unsigned int*, Umístění: *data*, Velikost: 2*B*.

`extern bit` **`ARQ;`**
Tento bit povoluje metodu ARQ (Automatic Repeat reQuest) u protokolu TCP – při ztrátě paketu se odešle poslední paket znovu.
Typ: *bit*, (Umístění: *data*), Velikost: *1bit*.

Modul zahrnuje následující funkce:

UDPsend
TCPsend
IP_UDP
IP_TCP
TCPtimer

5.9.1 UDPsend

Popis: Funkce slouží k odeslání UDP datagramu. Cíl je dán MAC adresou, IP adresou a čísly portů. Není to tedy funkce striktně transportní vrstvy, funkce postupně přechází až k nejnižší vrstvě. Data UDP datagramu jsou brána z *UDPbuff* a jejich velikost je dána *udp_length*. Pokud je k *udp_length* přičtena konstanta 0x8000, potom data UDP datagramu jsou tvořena přímo daty *TX_BUFF* od ofsetu *0x1C*. Velikost je dána (*udp_length & 0x7FFF)*, adresa *UDPbuff* je potom irelevantní. Toho se využívá k tomu, aby nedocházelo ke zbytečnému přesunu dat mezi *UDPbuff* a *TX_BUFF*.

Prototyp:

```
void UDPsend(word srcPort,
             word dstPort,
             MACADDRESS *pDESTMAC,
             IPADDRESS *pDESTIP,
             byte *UDPbuff,
             word udp_length);
```

Příklad volání:
```
UDPsend(55550,
        55550,
        &DESTMAC,
        &DESTIP,
        strUNLOCK,
        sizeof(strUNLOCK)-1);
```

Parametry:
1. *srcPort* – typ *unsigned int*, číslo zdrojového port.
2. *dstPort* – typ *unsigned int*, číslo cílového port.
3. **pDESTMAC* – adresa proměnné typu *MACADRESS*, proměnná udává MAC adresu cíle.
4. **pDESTIP* – adresa proměnné typu *IPADDRESS*, proměnná udává IP adresu cíle.
5. **UDPbuff* – adresa bufferu s odesílanými daty.
6. *udp_length* – typ *unsigned int*, velikost odesílaných dat.
Možné hodnoty:
0x0000 až 0x05C0 – tyto hodnoty určují velikost dat v UDPbuff.
0x8000 až 0x85C0 – velikost dat je hodnota (*udp_length & 0x7FFF)*, data nejsou brána z *UDPbuff* (adresa může být libovolná), odesílají se data přímo z *TX_BUFF*. Data v *TX_BUFF* začínají na ofsetu 0x1C. Od tohoto ofsetu se plní *TX_BUFF* daty, které se mají odeslat.

Vrácená hodnota: **žádná.**

5.9.2 TCPsend

Popis: Funkce odešle TCP segment z daného *socketu*. Číslo *socketu* je parametrem funkce. Opět funkce neodpovídá pouze transportní vrstvě, ale prochází postupně až do vrstvy síťového rozhraní. Před voláním funkce musí být nastaveny všechny potřebné parametry *socketu* (MAC, IP, čísla portů, odesílané příznakové bity, adresa a velikost dat, atd.). Struktura *socketu* je definována v souboru *typedefs.h* (viz kap. 5.1).

Prototyp:
```
void TCPsend(byte client_nr);
```

Příklad volání:
```
TCPsend(4);
```

Parametry:
1. *client_nr* – typ *unsigned char* – číslo klienta, kterému se bude TCP segment posílat.
Možné hodnoty:
0 až (max_conn-1)

Vrácená hodnota: **žádná.**

5.9.3 IP_UDP

Popis: Funkce je volána z funkce *protokol_IP*, to nastává v případě příjmu UDP datagramu. Funkce plní rovněž službu aplikační vrstvy. Pokud dorazí UDP datagram na port *55550*, pak se testuje, o jaký příkaz jde a podle toho se vykoná příslušná akce (zápis bloku dat do paměti, čtení z pamětí, apod.). Vždy se posílá UDP datagram zpět (buď potvrzovací paket, nebo přímo vyžádaná data). To umožňuje kontrolu, zda UDP datagram dorazil (UDP poskytuje nespolehlivou službu) a tím do jisté míry zajistit spolehlivost služby, ovšem až v aplikační vrstvě.

Prototyp: `void IP_UDP(void);`

Příklad volání: `IP_UDP();`

Parametry: **žádné.**

Vrácená hodnota: **žádná.**

5.9.4 IP_TCP

Popis: Funkce je volána z funkce *protokol_IP*, to nastává v případě příjmu TCP segmentu. Jedná se o zásadní funkci protokolu TCP, z přijatého TCP segmentu podle hlavičkových informací rozeznává číslo *socketu*, ke kterému segment patří, popř. přiděluje číslo *socketu* při navazování spojení. Dále kontroluje sekvenční a potvrzovací čísla TCP a podle přijatých příznakových bitů nastavuje stav TCP. Tato funkce neřeší služby vrstvy aplikační, proto je nutné po této funkci volat funkci aplikační vrstvy (funkce `app()`). Pozn.: Kontrolní součet TCP není kontrolován.

Prototyp: `void IP_TCP(void);`

Příklad volání: `IP_TCP();`

Parametry: **žádné.**

Vrácená hodnota: **žádná.**

5.9.5 TCPtimer

Popis: Funkce se volá z obsluhy přerušení časovače Timer3. Od tohoto časového intervalu jsou odvozeny veškeré *timeouty* protokolu TCP. Každým voláním funkce dojde dekrementaci pole *time* ve struktuře *socketu* (pokud je pole nulové k dekrementaci nedochází – stav, který nemá *timeout*). Pokud dojde dekrementací k vynulování pole, nastal *timeout* a podle aktuálního stavu TCP se vykoná příslušná akce. Zároveň se počítá počet chyb a pokud počet chyb dovrší určitou hranici, *socket* (a tím i dané spojení) se uzavře. To proto, aby nedocházelo k opětovnému posílání donekonečna.

Prototyp: `void TCPtimer(void);`

Příklad volání: `TCPtimer();`

Parametry: **žádné.**

Vrácená hodnota: **žádná.**

5.10 Modul APPLICATION

Funkce modulu APPLICATION spadají do aplikační vrstvy a mají za úkol řešit aplikační protokoly (HTTP).

Soubory modulu APPLICATION:
Zdrojové: **application.c**
Hlavičkové: **application.h**

Proměnné definované v modulu APPLICTION:

`struct t_stat_files code` **`status_code[7];`**
Datová struktura sdružující informace adrese a velikosti status kódů (viz kap 4.4.3).
Typ: *t_stat_files[]* viz kap. 5.1, Umístění: *code*, Velikost: 28*B*.

Modul zahrnuje následující funkce:

app
http

5.10.1 app

Popis: Funkce aplikační vrstvy. Z této funkce jsou volány funkce aplikačních protokolů podle čísla cílového portu (pro port 80 je volána funkce `http()`). Dále funkce podle stavu *socketu* odesílá TCP segmenty s příslušnými příznakovými bity a případně daty (funkce požadované příznaky daného *socketu*, zavolá funkci *TCPsend(socket_nr)* a nastaví nový stav *soketu*). Toto funkce postupně vykonává pro všechny *sockety* (u uzavřených *soketů* pochopitelně funkce neposílá žádné segmenty).

Prototyp: `void app(void);`

Příklad volání: `app();`

Parametry: **žádné.**

Vrácená hodnota: **žádná.**

5.10.2 http

Popis: Funkce protokolu HTTP. Pro požadavky GET vyhledává požadovaný soubor, pokud je daný soubor nalezen, nastaví se ve struktuře *soketu* adresa, kde se soubor nachází, v případě nenalezení souboru se nastaví adresa na soubor s chybovým kódem. Poté se nastav stav pro odeslání dat. Samotné odesílání TCP segmentu se děje ve funkci `app()`.

Prototyp: `void http(byte client_nr);`

Příklad volání: `http(4);`

Parametry: **1.** *client_nr* – typ *unsigned char* – číslo klienta (*socketu*), který komunikuje protokolem HTTP.
Možné hodnoty:
0 až (max_conn-1)

Vrácená hodnota: **žádná.**

5.11 Modul TIMER

Funkce modulu TIMER jsou určeny pro obsluhu časovače Timer3 mikrokontroléru C8051F120. Časovač slouží k časování různých událostí v mikrokontroléru a je použit také k časování protokolu TCP.

Soubory modulu TIMER:

Zdrojové:	**timer.c**
Hlavičkové:	**timer.h**

Proměnné definované v modulu TIMER:

```
volatile word data sys_timer;
```

systémový strojový čas, je inkrementován každé přetečení časovače (v obsluze přerušení).
Typ: *unsigned int*, Umístění: *data*, Velikost: *2B.*

Modul zahrnuje následující funkce:

Timer3_INIT
get_systime
Timer3_ISR

5.11.1 Timer3_INIT

Popis: Funkce inicializuje časovač *Timer3*. Časovač pak periodicky generuje přerušení podle nastaveného intervalu.

Prototyp: `void Timer3_Init(word counts);`

Příklad volání: `Timer3_Init(c_Timer3);`

Parametry: **1.** *counts* – typ unsigned *int*, udává hodnotu, na kterou bude časovač naplněn po přetečení (udává periodu přetečení).
Možné hodnoty:
(v závislosti na taktu 8051 viz [3]):
c_time8ms – přetečení každých 8ms, při taktu 33,3MHz.

Vrácená hodnota: **žádná.**

5.11.2 get_systime

Popis: Funkce vrací hodnotu systémového strojového času. Čas je v jednotkách určených nastavením časovače *Timer3*. Při inicializaci časovače s parametrem *c_time8ms*, bude časová jednotka přibližně 8ms. Přetečení systémového času v tomto případě nastane přibližně každých 8,7 minuty.

Prototyp: `word get_systime(void);`

Příklad volání: `s_time = get_systime();`

Parametry: **žádné.**

Vrácená hodnota: **s_time –** typ unsigned *int*. Vrací systémový strojový čas.

5.11.3 Timer3_ISR

Popis: Obsluha přerušení časovače Timer3. Inkrementuje *sys_time*, dává časový základ pro další události a volá funkci TCP_Timer, která řeší časové události protokolu TCP.

Prototyp: `interrupt 14`

Příklad volání: `(interrupt)`

Parametry: **žádné (interrupt).**

Vrácená hodnota: **žádná (interrupt).**

5.12 Modul BOARD

Funkce modulu BOARD slouží k inicializaci HW modulu C8051F120 a obsahuje funkce pro kopírování bloku dat mezi různými typy pamětí.

Soubory modulu BOARD:

Zdrojové:	**board.c**
Hlavičkové:	**board.h**

Proměnné definované v modulu BOARD:

žádné.

Modul zahrnuje následující funkce:

PORT_INIT
SYSCLK_INIT
EMIF_INIT
mem_wr
mem_rd

Pozn.: Funkce pro inicializaci různých částí mikrokontoléru C8051F120 lze pohodlně vygenerovat pomocí programu *Config2*, který poskytuje firma Silicon Laboratories. Proto není nutné, aby měly funkce vstupní parametry, vše lze snadno nastavit ve zmiňovaném programu.

5.12.1 PORT_INIT

Popis: Funkce inicializuje porty mikroprocesoru, nastaví porty podle potřeby na vstupní/výstupní, nastavuje *crossbar* a povoluje periferie. Funkce se volá po resetu mikrokontroléru.

Prototyp: `void PORT_Init(void);`

Příklad volání: `PORT_Init();`

Parametry: **žádné.**

Vrácená hodnota: **žádná.**

5.12.2 SYSCLK_INIT

Popis: Funkce inicializuje systémový takt mikrokontroléru, zapíná a nastavuje PLL. Kmitočet je odvozen z vnitřního oscilátoru a je nastaven na 33,3MHz. Funkce se volá po resetu mikrokontroléru.

Prototyp: `void SYSCLK_Init(void);`

Příklad volání: `SYSCLK_Init();`

Parametry: **žádné.**

Vrácená hodnota: **žádná.**

5.12.3 EMIF_INIT

Popis: Funkce inicializuje rozhraní pro připojení externí paměti. Nastavuje mód na *Split-mode (banked), non-multiplexed* na portech *P4 - P7*, časování sběrnice nastavuje na hodnotu *c_EMI0TC_CP2200*. *XRAM page* nastavuje na hodnotu *0x20* (paměťový prostor CP2200).

Prototyp:
```
void EMIF_Init (void);
```

Příklad volání:
```
EMIF_Init();
```

Parametry: **žádné.**

Vrácená hodnota: **žádná.**

5.12.4 mem_wr

Popis: Funkce kopíruje data z paměti *xdata* na čipu do externí paměti dané stránkou paměti. Během vykonávání funkce je globálně zakázáno přerušení.
Pozn.: pokud by (*addr* + *size*) > 65536, potom během kopírování zákonitě dojde k přetečení adresy z *0xFFFF* na *0x0000* a tím dojde k inkrementaci čísla stránky. Znamená to, že „přetečená“ data se zapíší do následující stránky (u paměti FLASH to neplatí, mělo by se takovému přetečení zamezit).

Prototyp:
```
void mem_wr(byte *src, byte page,
            word addr, word size);
```

Příklad volání:
```
mem_wr(src_buff, pg_FLASH0_0,0x0000,256);
```

Parametry:
1. **src* – ukazatel na buffer, ze kterého se budou brát data..
2. *page* – typ *unsigned char*, stránka paměti, ze které se bude číst.
Možné hodnoty:
pg_FRAM0_0 až 1 – stránky paměti FRAM.
pg_FLASH0_0 až 7, *pg_FLASH1_0 až 7* – paměť FLASH.
pg_chip_XRAM – paměť *xdata* (*RAM*) na čipu.
pg_chip_DATA – paměť *data* (*RAM*) na čipu.
pg_chip_FLASH – paměť *code* (*FLASH*) na čipu.
3. *addr* – typ *unsigned int*, adresa bloku dat v dané stránce.
4. *size* – typ *unsigned int*, velikost bloku dat.
Možné hodnoty:
*0x0001 až Velikost_Bufferu(na který ukazuje *src)*.

Vrácená hodnota: **žádná.**

5.12.5 mem_rd

Popis: Funkce kopíruje data z paměti dané stránkou paměti do paměti *xdata* na čipu. Během vykonávání funkce je globálně zakázáno přerušení.
Pozn.: pokud by (*addr* + *size*) > 65536, potom během kopírování zákonitě dojde k přetečení adresy z *0xFFFF* na *0x0000* a tím dojde k inkrementaci čísla stránky. Znamená to, že „přetečená“ data se budou číst z následující stránky.

Prototyp:
```
void mem_rd(byte *dest, byte page,
            word addr, word size);
```

Příklad volání:
```
mem_rd(dest_buff,pg_FLASH0_0,0x0000,256);
```

Parametry:
1. **dest* – ukazatel na buffer, kam se uloží přečtená data.
2. *page* – typ *unsigned char*, stránka paměti,ze které se bude číst. Možné hodnoty:
pg_FRAM0_0 až 1 – stránky paměti FRAM.
pg_FLASH0_0 až 7, *pg_FLASH1_0 až 7* – paměť FLASH.
pg_chip_XRAM – paměť *xdata* (*RAM*) na čipu.
pg_chip_DATA – paměť *data* (*RAM*) na čipu.
pg_chip_FLASH – paměť *code* (*FLASH*) na čipu.
3. *addr* – typ *unsigned int*, adresa bloku dat v dané stránce.
4. *size* – typ *unsigned int*, velikost bloku dat.
Možné hodnoty:
*0x0001 až Velikost_Bufferu(na který ukazuje *dest)*.

Vrácená hodnota: **žádná.**

5.13 Modul MAIN

Modul MAIN obsahuje hlavní funkci – funkci `main()`. Mimo to obsahuje funkce jednotlivých úloh, které jsou postupně volány v hlavní smyčce funkce `main()`.

Soubory modulu MAIN:
Zdrojové: **main.c**
Hlavičkové: **main.h**

Proměnné definované v modulu MAIN:

žádné (proměnné jsou definovány v modulu *VARIABLES*).

Modul zahrnuje následující funkce:

wait_us
main
task_monitor
task_ethernet
task_uart
task_ad

5.13.1 wait_us

Popis: Funkce slouží ke zpoždění, zpoždění je nastavitelné parametrem. Jedná se o jednoduchou funkci realizovanou zpožďovací smyčkou, proto požadovaný čas zpoždění není přesný, v případě obsluhy přerušení během zpožďovací smyčky se čas může značně prodloužit.

Prototyp: `void wait_us(word us);`

Příklad volání: `wait_us(1000);`

Parametry: **1.** *us* – typ *unsigned int*, udává přibližný čas v mikrosekundách.

Vrácená hodnota: **žádná.**

5.13.2 main

Popis: Hlavní funkce programu. Na začátku jsou volány inicializační funkce z ostatních modulů a jsou nastaveny výchozí hodnoty proměnných. Poté následuje nekonečná smyčka, ve které jsou postupně volány funkce jednotlivých úloh:
task_monitor - úloha pro zobrazování údajů na displeji.
task_ethernet - úloha pro komunikaci prostřednictvím internetu.
task_uart - úloha obsluhy přijatých znaků ze sériové linky.
task_ad - úloha pro měření analogových veličin.

Prototyp: `nemá`

Příklad volání: `(je volána po resetu mikrokontroléru)`

Parametry: **žádné.**

Vrácená hodnota: **žádná.**

6 Software jednotky

Software byl napsán v jazyce *C* v prostředí *Keil uVision3* s využitím SW modulů popsaných v kap. 5. V této kapitole nebudou moduly a jejich funkce znovu popisovány, bude zde popsána spíše hlavní myšlenka programu a použité algoritmy pro řešení dílčích úkolů.

Software má řešit následující požadavky:

- Měření úrovní na analogových vstupech, periodické ukládání naměřených hodnot do souboru.
- Zobrazování údajů na displeji LCD.
- Komunikace prostřednictvím sítě internet - pomocí webového rozhraní (webová stránka zobrazená v prohlížeči) monitorovat analogové vstupy a nastavovat úrovně výstupních linek, nastavovat parametry jednotky, možnost stahování souborů s naměřenými hodnotami.

Software je koncipován pouze jako ukázkový (demo), který nebude prakticky používán pro monitorování a řízení. Slouží pouze jako ukázka možností jednotky. Pro konkrétní použití jednotky bude potřeba software upravit, aby měřené úrovně odpovídaly požadovaným veličinám, aby byly korektně zobrazovány na webových stránkách apod. Konkrétní využití jednotky bude také otázkou hardwarového připojení na konkrétní zařízení (např. hlavici optického spoje).

6.1 Obecný popis

Software je optimalizován pro systémový kmitočet mikrokontroléru 33,3MHz (přesněji 32,6MHz; PLL násobitel 12, dělitel 9, vstupní kmitočet 24,5MHz on chip). Tato frekvence je maximální možná, aby bylo možno zapisovat do externí paměti FRAM pomocí rozhraní EMIF. Při vyšších kmitočtech zápis do FRAM selhával. Může být použit i nižší kmitočet, pro obsluhu rozhraní ethernet je ovšem žádoucí kmitočet co nejvyšší, aby byla jednotka schopna zpracovávat všechny přijaté pakety.

Většina proměnných je definována v paměťovém prostoru *data*, výjimku tvoří datové buffery, které jsou díky své velikosti v oblasti *xdata* na čipu (do oblasti *data* by se nevešly). Tato konfigurace je výhodná zejména z pohledu rychlosti, kdy díky možnosti přímé adresace oblasti *data*, lze kód optimalizovat k vysokému výpočetnímu výkonu. Také použití oblasti *xdata* na čipu je výhodná, protože přístup do této paměti může být rychlejší, než případný přístup do paměti mimo čip, viz kap. 6.1.2.

6.1.1 Rozdělení na úlohy

Hlavní smyčku programu tvoří tři dílčí úlohy: **task_ad** pro měření analogových veličin, **task_monitor** pro zobrazování údajů na displeji a **task_ethernet** pro komunikaci prostřednictvím síťového rozhraní ethernet. Pozn.: pro účely vývoje softwaru byla ještě využita úloha **task_uart** pro komunikaci přes sériovou linku, tato však pro funkci jednotky není nutná.

Zatímco úloha **task_ethernet** se provádí v každém cyklu nekonečné smyčky (síťová komunikace vyžaduje častou co nejrychlejší obsluhu), úlohy **task_ad** a **task_monitor** se provádí jen občas (asi dvakrát za sekundu, častěji obnovovat displej není potřeba, protože by to člověk nebyl schopný vnímat, perioda měření úrovní také dostačuje, nepředpokládají se rychlé změny napětí na vstupech). Pochopitelně toto přidělování prostředků jednotlivým úlohám lze podle potřeby změnit.

6.1.2 Organizace paměti

Mikrokontrolér C8051F120 má k dispozici tři základní paměťové prostory:

- ***data*** – 128 bajtů, přímá adresace, patří sem i oblast *bdata*, která je obsažená v oblasti *data* a umožňuje adresování jednotlivých bitů, dalších 128B je nepřímo adresovatelných označovaných jako *idata*.
- ***xdata*** – externí paměť, možnost adresace 64KB, z toho 8KB je umístěno na čipu, umožňuje několik režimů, kdy je možné paměť na čipu vypnout a umožnit tak celé adresní rozsah paměti mimo čip. Do oblasti *xdata* spadá i oblast 256B prostoru označovaného *pdata*. Oblast *pdata* se využívá pro adresaci v rozsahu 256B na definované stránce. Tato oblast je použita pro přístup k registrům řadiče CP2200.
- ***code*** – paměť programu, 128KB, lze použít pro některé konstanty.

Externí paměť (oblast *xdata*) může být konfigurována jako (viz [3]) :

- Internal XRAM Only
- Split Mode without Bank Select
- Split Mode with Bank Select
- External Only

Jednotka využívá dva z těchto módů, a to *Split Mode with Bank Select* jako výchozí mód a *External Only* při přístupu do pamětí mimo čip.

Vývojová deska C8051F120 má osazeny paměti FRAM (128KB) a FLASH (2x512KB). Protože je možné adresovat pouze 64KB, musí se pro zpřístupnění celých kapacit pamětí použít stránkování. Číslem stránky se pak ovládají vyšší bity adresy a signály CE jednotlivých pamětí. V programu jsou definovány čísla stránek:

- *pg_FRAM0_0* až *pg_FRAM0_1* - pro paměť FRAM.
- *pg_FLASH0_0* až *pg_FLASH0_7* - pro první paměť FLASH.
- *pg_FLASH1_0* až *pg_FLASH1_7* - pro druhou paměť FLASH.
- *pg_chip_FLASH* - paměť FLASH na čipu.
- *pg_chip_XRAM* - pro paměť na čipu, od adresy 0x2000 je v této stránce také 256B paměťového prostoru řadiče CP2200. Tato stránka je výchozí, při přepnutí na jinou stránku se musí globálně zakázat přerušení. Je to z důvodu, že v paměti na čipu jsou definovány buffery *RX_BUFF* a *TX_BUFF*, které využívá řadič CP2200. Buffery jsou úmyslně ve stejné stránce jako řadič CP2200, aby při kopírování dat z řadiče do bufferů a naopak nemuselo docházet k přepínání stránky paměti. Režie přepínání by značně prodloužila dobu kopírování dat.

Pozn.: Kromě uvedených stránek je definována ještě *pg_chip_DATA*, ta ovšem nedefinuje externí paměť, nýbrž paměťový prostor *data*. Je definována pro univerzální použití funkcí *mem_wr* a *mem_rd* modulu *board* viz kap. 5.12.

6.2 TASK_AD

Měření úrovní na analogových vstupech je velmi jednoduchá úloha. Měří se postupně čtyři kanály a do příslušných proměnných se ukládají přímo hodnoty z AD převodníku. Pro praktické využití je to pochopitelně nedostačující, protože taková hodnota je nicneříkající. Každý vstup bude prakticky připojen k nějakému čidlu či převodníku a naměřené hodnoty se přepočítají na příslušné jednotky (např. při měření výkonu to mohou být *mW*, při měření teploty *°C*, apod.).

Kromě úrovní analogových vstupů je v této úloze měřena teplota čipu mikrokontroléru. Využívá se k tomu integrované teplotní čidlo. Naměřená hodnota je na rozdíl od předchozích přepočítána na Celsiovy stupně. Rozsah teploty může být z hlediska formátu proměnné 00.0°C až 99.9°C. Mimo tento rozsah bude teplota udávána chybně.

6.3 TASK_MONITOR

V této úloze se na LCD displeji zobrazují údaje o aktuálním čase a datumu, stav připojení k síti Ethernet a zobrazují se údaje dvou naměřených hodnot úrovní na analogových vstupech.

Informace o datumu a čase je brána z obvodu reálného času (PCF8583) připojeného pomocí sběrnice I^2C. Před každou aktualizací displeje se přečte aktuální datum a čas.

V této úloze dále dochází k ukládání měřených dat do souboru. Podle hodin reálného času a podle nastavené periody ukládání dochází k uložení aktuálního času do hlavičky souboru, dále se do hlavičky ukládá ukazatel dat a příznak přetečení pamětí. Do souboru je potom zaznamenávána teplota čipu a dále hodnoty z prvních dvou analogových vstupů. Formát souboru je uveden v Tab. 12.

Tab. 12 Formát souboru s ukládanými daty.

	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07		
0x00	**Datum a čas posledního zápisu**					**PER**	**Ukazatel**			**hlavička**
0x08	**full**	**Reserved (nuly)**								
0x10	Teplota (např.:"24.5")				Ain0 (hex)		Ain1 (hex)			**data**
0x18	Teplota (např.:"24.5")				Ain0 (hex)		Ain1 (hex)			
0x20	Teplota (např.:"24.5")				Ain0 (hex)		Ain1 (hex)			
0x28	Teplota (např.:"24.5")				Ain0 (hex)		Ain1 (hex)			
0x30	Teplota (např.:"24.5")				Ain0 (hex)		Ain1 (hex)			
0x38	Teplota (např.:"24.5")				Ain0 (hex)		Ain1 (hex)			
0x40	Teplota (např.:"24.5")				Ain0 (hex)		Ain1 (hex)			
.	Teplota (např.:"24.5")				Ain0 (hex)		Ain1 (hex)			
.	Teplota (např.:"24.5")				Ain0 (hex)		Ain1 (hex)			
.	Teplota (např.:"24.5")				Ain0 (hex)		Ain1 (hex)			

Datum a čas posledního zápisu (5B) je brán přímo z obvodu reálného času a formát je tedy shodný s formátem tohoto obvodu viz [12]. Pole *PER* (1B) udává periodu zápisu do souboru, odpovídá desítkám sekund (pro hodnotu 1 tedy bude docházet k ukládání každých 10s). Pole *Ukazatel* (2B) je ofset v souboru, kde jsou poslední zapisovaná data, pokud nedošlo k přetečení souboru, udává toto pole také velikost souboru. Pole *full* (1B) udává, zda-li došlo k přetečení pole a přepisují se nejstarší data novými, nebo nikoliv. Pro hodnotu *0x00* nedošlo k přetečení (velikost souboru udává pole *Ukazatel*), pro hodnotu 0x0*1* k přetečení došlo, velikost souboru je pak 65536B. Následujících 7 bajtů je rezerva.

Data každého zápisu mají 8 bajtů. První čtyři udávají teplotu čipu. Formát teploty je ASCII – desítky, jednotky, desetinná tečka, desetiny stupně. Po teplotě následují dva bajty s naměřenou hodnotou analogového vstupu 0, poté následují dva bajty pro analogový vstup 1. Formát těchto údajů je přímo údaj z AD převodníku, tedy číslo v rozsahu *0x0000* až *0xFFFF*.

6.4 TASK_ETHERNET

Tato úloha obsluhuje rozhraní ethernet a umožňuje vzdálené řízení modulu prostřednictvím internetu a webového rozhraní.

Přijímání paketu si vyžaduje následující strukturu programu:

```
if (CP_RxValid())             // test, jsou-li nejake prijate packety
{
  if (CP_Receive())           // nacteni prijateho packetu do RX_BUFF
  {
    if (((RX_BUFF[ofs_TYPE+0])<<8)+(RX_BUFF[ofs_TYPE+1])==type_I
    {
      protokol_IP();          // zpracovani prijateho IP packetu
    }
    else
    if (((RX_BUFF[ofs_TYPE+0])<<8)+(RX_BUFF[ofs_TYPE+1])==type_ARP)
    {
      protokol_ARP();         // zpracování prijateho ARP packetu
    }
  }
}

app();                        // zpracovani aplikacnich dat, posilani
                              // potvrzovacich paketu TCP, apod.
```

Nejprve se musí testovat, zda-li byl nějaký paket správně přijat (`CP_RxValid()`), pokud ano, je nutné jej co nejrychleji vyčíst z bufferu řadiče CP2200 (aby řadič měl volné místo pro přijímání dalších paketů). Program je koncipován tak, že se paket vždy načte do bufferu *RX_BUFF*. Vyčtení paketu z řadiče do *RX_BUFF* vykonává funkce `CP_Receive()`. Další zpracování paketu se děje z *RX_BUFF*. Pomocí definovaných ofsetů v tomto bufferu je zjištěn typ a podle něj je volána příslušná funkce na zpracování (`protokol_IP()` nebo `protokol_ARP()`). Funkce protokolu IP volá příslušné funkce vyšší vrstvy, kde dochází k dalšímu zpracování přijatého paketu až k aplikačním datům.

Jako poslední funkce, kterou je nutné volat je funkce `app()`. Ta je volána vždy, bez ohledu na to, jestli byl nějaký paket přijat. Funkce vykonává akce podle stavu TCP, resp. stavu jednotlivých *socketů*. Na jeden přijatý paket totiž může být potřeba vykonat více než jedna akce, případně se může stav *socketu* měnit díky *timeoutu* a nikoliv jen příjmem paketu.

6.4.1 Komunikace prostřednictvím TCP

Přijímané TCP segmenty se zpracovávají ve funkci `IP_TCP()`, která je volána z funkce `protokol_IP()` poté, co je zjištěno, že se jedná o TCP segment.

Funkce `IP_TCP()` mění stav socketu podle přijatých příznakových bitů. Stavy socketu jsou oproti standardním stavům TCP (kap. 3.3.2) rozšířeny o další stavy, které přesně definují, jaká operace se má vykonat, případně na jaký segment se čeká. Přehled všech stavů společně s použitými časy pro timeout a stavem následovaným po timeoutu je uveden v Tab. 13.

Tab. 13 Výpis možných stavů soketu, příslušných časů pro T.O. a změna stavu po T.O.

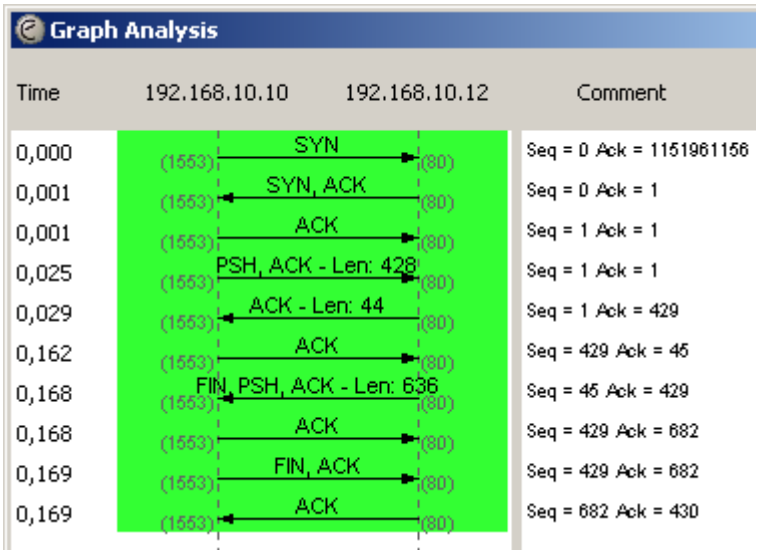
	stav	*čas TO*	*stav po TO*
TCP states	s_LISTEN	t_none	
	s_SYN_RCVD	t_timeout	s_CLOSE
	s_ESTABLISHED	t_a_fin	s_FIN
	s_FIN_WAIT1	t_timeout	s_SEND_AGAIN_F2
	s_FIN_WAIT2	t_fw2	s_RST
	s_TIME_WAIT	t_tw	s_CLOSE
	s_CLOSE_WAIT	t_none	
	s_LAST_ACK	t_timeout	s_SEND_AGAIN_F
	s_CLOSE	t_none	
APP	s_DATA_RCVD	t_none	
	s_DATA_RCVD_FIN	t_none	
	s_STAT_SND	t_none	
WAIT	s_WAIT_ACK	t_timeout	s_SEND_AGAIN
	s_WAIT2_ACK	t_timeout	s_SEND_AGAIN
	s_WAIT3_ACK	t_timeout	s_SEND_AGAIN_S
CMD states	s_STAT_DATA	t_none	
	s_NEXT_DATA	t_none	
	s_FIN	t_none	
	s_RST	t_none	
	s_ACK	t_none	
	s_ACK_FIN	t_none	
	s_ACK_TW	t_none	
	s_SYN	t_none	
ARQ	s_SEND_AGAIN_S	t_none	
	s_SEND_AGAIN	t_none	
	s_SEND_AGAIN_F2	t_none	
	s_SEND_AGAIN_F	t_none	

Definováním nových stavů se obsluha protokolu TCP zjednodušila. Není potřeba v TCP stavu ESTABLISHED sledovat čísla odeslaného bajtu (*sequence number*, dále jen číslo SEQ) a čísla potvrzovací (*acknowledgment number*, dále jen číslo ACK), aby bylo možné kompletovat přijatá a odesílaní data. Definováním speciálních stavů pro každou možnou situaci společně s nastavením velikosti okna na 1460 se docílilo komunikace *Stop and Wait ARQ* viz [11]. Znamená to, že každý odeslaný TCP segment musí být nejprve potvrzen a až poté lze posílat další. Příklad komunikace se znázorněnými stavy po každém TCP segmentu je znázorněna v Tab. 14.

Tab. 14 Příklad TCP komunikace s uvedenými stavy.

	timeout	state	funkce
TCP komunikace s ukoncenim od serveru			
	t_none	s_LISTEN	x
SYN →	x	s_SYN	ip_tcp
SYN,ACK ←	t_timeout	s_SYN_RCVD	app
ACK →	t_a_fin	s_ESTABLISHED	ip_tcp
PSH,ACK len=100 →	t_none	s_DATA_RCVD	ip_tcp
ACK len=44 ←	x	s_STAT_DATA	http (app)
	t_timeout	s_WAIT3_ACK	app
ACK →	t_none	s_STAT_SND	ip_tcp
FIN,PSH,ACK len=400 ←	x	s_NEXT_DATA	http (app)
	t_timeout	s_FIN_WAIT1	app
ACK →	t_fw2	s_FIN_WAIT2	ip_tcp
FIN,ACK →	t_none	s_ACK_TW	ip_tcp
ACK ←	t_tw	s_TIME_WAIT	app
	(t_none)	s_CLOSE	timer3_isr

Příklad reálné komunikace je znázorněn na Obr. 25. Komunikace byla sledována programem Ethereal. Komunikace znázorňuje otevření webové stránky ve webovém prohlížeči. Tato reálná komunikace koresponduje s příkladem v Tab. 14.



Obr. 25 Příklad reálné komunikace (sledováno programem Ethereal).

Funkce `IP_TCP()`nejprve u přijatého TCP segmentu testuje, je-li nastaven příznak *SYN*. Pokud ano, vyhledá se volný *socket* a změní se jeho stav na *s_SYN*. Tím začne navazování spojení.

Pokud není nastaven příznak *SYN*, vyhledá se podle IP adresy a čísla portu *socket*, ke kterému TCP segment patří. Zkontroluje se číslo ACK a podle nastavených příznakových bitů se změní stav *socketu*. Změny stavů jsou znázorněny v Tab. 15.

Číslo ACK se porovnává s číslem, které vznikne součtem odeslaného SEQ čísla a velikosti odeslaných dat (případně se ještě přičte 1 jestliže byl odeslán příznak *FIN* nebo *SYN*). Musí tedy platit:

$$xACK = SEQ + len + 1 \cdot (flag == (f\_SYN \mid f\_FIN)),$$

kde je xACK .. přijaté číslo ACK,
SEQ .. odeslané číslo SEQ,
len .. velikost odeslaných dat,
flag .. odeslaný bajt s příznakovými bity.

V případě souhlasných čísel se pak přijaté číslo ACK uloží do struktury *socketu* jako číslo SEQ, které se bude posílat, vynuluje se velikost odesílaných dat, vynuluje se bajt příznaků, uloží se velikost přijatých dat a přijatý příznakový bajt. Tím je *socket* připraven na další operace.

Poté následuje zmiňovaná změna stavu podle přijatých příznakových bitů.

Tab. 15 Změna stavu socketu v závislosti na přijatých příznakových bitech.

stav	přijaté příznakové bity					
	SYN	*ACK*	*PSH, ACK*	*FIN, PSH, ACK*	*FIN, ACK*	*RST, ACK*
default --->			s_DATA_RCVD	s_RST	s_ACK_FIN	s_CLOSE
s_LISTEN	s_SYN					
s_SYN_RCVD		s_ESTABLISHED				
s_ESTABLISHED		*(s_DATA_RCVD)*		s_DATA_RCVD_FIN		
s_FIN_WAIT1		s_FIN_WAIT2			s_ACK_TW	
s_FIN_WAIT2					s_ACK_TW	
s_TIME_WAIT					s_ACK_TW	
s_LAST_ACK		s_CLOSE				
s_WAIT_ACK		s_FIN				
s_WAIT2_ACK		s_NEXT_DATA				
s_WAIT3_ACK		s_STAT_SND				

Do této chvíle byly řešeny pouze přijímané TCP segmenty. Odesílání TCP segmentů se děje ve funkci `app()`.

Funkce `app()`volá funkce aplikační vrstvy (`http()`) a podle stavu *socketu* nastavuje obsah TCP segmentu, voláním funkce (`TCPsend()`) takový segment odesílá a podle obsahu změní stav *socketu*.

Funkce odesílání TCP segmentu vyplní odesílaný paket (data linkového rámce) hlavičkou IP, hlavičkou TCP a případně odesílanými daty. V IP hlavičce vyplňuje IP adresu cíle z datové struktury *socketu* (viz kap. 5.1), v TCP hlavičce ze struktury vyplňuje cílový a zdrojový port, odesílané SEQ číslo. Odesílané číslo ACK se vypočítá podle vztahu:

$$ACK = xSEQ + xlen + 1 \cdot (xflag == (f\_SYN \mid f\_FIN)),$$

kde je ACK .. odesílané číslo ACK,
xSEQ .. přijaté číslo SEQ,
xlen .. velikost přijatých dat,
xflag .. přijatý bajt s příznakovými bity.

Poslední funkcí, která se zaobírá protokolem TCP je funkce `TCPtimer()`. Toto funkce slouží k ošetření ztráty TCP segmentů. Pokud nedojde do určeného času potvrzovací segment, je vyvolán timeout a nastaví se stav socketu, který takovou situaci ošetří. Většinou se nastaví stav pro opětovné odeslání posledního TCP segmentu. Počet, kolikrát se zopakuje odeslání segmentu je omezen, aby nedocházelo k nekonečnému odesílání jednoho paketu. Mohlo by to nastat, když by se klient nečekaně odpojil ze sítě. Proto je po určeném počtu znovuodeslání segmentu spojení striktně uzavřeno (nastavení stavu socketu na *s_CLOSE*).

Na závěr popisu TCP komunikace je třeba podotknout, že komunikace je velmi zjednodušená a zdaleka nesplňuje TCP komunikaci tak, jak tomu je u složitých systému typu PC.

Jednotka neumožňuje řízení toku dat, jak by měl protokol TCP zajišťovat, není sledována velikost okna klienta, který je připojen. Vzhledem k tomu, že je komunikace řešena jako *Stop and Wait ARQ*, měly by se všechny složitější chybové stavy vyřešit na straně klienta. Díky této myšlence je možné jednotku řešit způsobem, jakým řešena je a neměla by nastat situace, že dojde k selhání přenosu. To lze prakticky jen těžko ověřit, protože simulovat různé chybové stavy je velmi složité a není možné simulovat veškeré možné chybové stavy. Nicméně při softwarovém simulování ztráty paketu (úmyslné vynechání odeslání paketu a úmyslné přeskočení přijatého paketu) k žádnému selhání protokolu *Stop and Wait ARQ* nedošlo (ztracené pakety se poslaly znovu a komunikace pokračovala).

6.4.2 Komunikace prostřednictvím UDP

Protokol UDP umožňuje na rozdíl od TCP velmi jednoduchou komunikaci. Nejedná se totiž o stavový protokol (nespojovaná služba), a tak není potřeba definovat žádné stavy. Za to ovšem protokol UDP nabízí pouze nespolehlivou službu, takže nevíme, zda-li UDP datagram dorazil k cíli nebo nikoliv. Toto lze však obejít v aplikační vrstvě, kdy se po každém příjmu UDP datagramu odešle UDP datagram zpět (třeba s užitečnými daty, nebo jen s potvrzovacími daty), čímž dojde k potvrzení příjmu. Toho bude v jednotce využíváno.

Komunikace prostřednictvím UDP bude v jednotce využívána pro správu pamětí, popř. pro zjištění stavu TCP komunikace a v neposlední řadě může být také použita pro monitorování naměřených hodnot.

Pro tyto účely byl navržen komunikační protokol aplikační vrstvy. Jedná se o jednoduchou komunikaci Dotaz-Odpověď, kdy odpověď prakticky potvrdí přijetí požadavku a tím vykonání požadované operace. Struktura protokolu je naznačena v Tab. 16. Klient vždy posílá UDP datagram (Dotaz) na port 55550 jednotky s určitým požadavkem. Jednotka odešle nazpět UDP datagram s odpovědí. Port, na který odpověď odesílá se liší podle druhu požadavku. Můžou to být porty 55550, 55551, 55552, 55553, 55555. Hodnoty portů jsou voleny náhodně, ovšem z rozsahu pro soukromé účely (49152-65535 viz *Internet Assigned Numbers Authority*, nebo kap. 3.3.2). Čísla portů lze změnit, musí se ovšem změnit komunikační software klienta (aplikace na PC).

Protokol obsahuje pouze příkazy, které byly nutné pro návrh tohoto ukázkového softwaru. Pro širší použití bude nutné protokol doplnit dalšími příkazy podle potřeby. Je možné protokol doplnit o stejné funkce monitorování a řízení jednotky, jaké jsou požadovány přes webové rozhraní a dát tak uživateli alternativu ovládání jednotky. To by ovšem znamenalo potřebu řídící aplikace na PC.

Příkazy komunikačního protokolu:

DATAr - slouží pro čtení dat z paměti, parametry jsou: stránka paměti, adresa, velikost dat (nesmí být větší jak 1464, aby se data vešly do UDPdatagramu).
- jednotka pošle na port 55552 hlavičku a přečtená data.

DATAw - slouží k zápisu dat do paměti, parametry: stránka paměti, adresa, zapisovaná data. Velikost dat je opět limitována na 1464 (1500(MTU)-20(IP hlavička)-8(UDP hl.)-8(protokol hlavička)), pro paměť FLASH musí být velikost 256B a další zápis se může provádět až za 20ms.
- odpověď na port 55550, potvrzující text.

DATAg - slouží k přečtení naměřených hodnot.
- odpověď na port 55555, hlavička a naměřená data.

ROOTr - slouží k přečtení „root“ sektoru (viz souborový systém kap.6.4.3).
- odpověď na port 55551, přímo 512B požadovaných dat.

ROOTw - slouží k zápisu „root“, parametry: 512B dat.
- odpověď na port 55550, potvrzující text.

ECHO - slouží k ověření funkce komunikace, vrátí zpět „ECHO“.
- odpověď na port 55550, text „ECHO“.

TCP s - slouží ke zjištění stavů socketů.
- odpověď na port 55550, text se stavy.

TCP d - požadavek na zaslání struktury všech socketů.
- odpověď na port 55553, 320B požadovaných dat.

CFG_r - přečte konfiguraci ARQ.
- odpověď na 55550, potvrzovací text a „1“ nebo „0“.

Tab. 16 Komunikační protokol pro správu jednotky (transport. UDP).

	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0A	0x0B	0x0C	0x0D	0x0E	0x0F	0x10	0x11
	"D"	"A"	"T"	"A"	"r"	page	addr		size									
55552:	"D"	"A"	"T"	"A"	"r"	page	addr		data									
	"D"	"A"	"T"	"A"	"w"	page	addr		data									
55550:	"D"	"A"	"T"	"A"	"w"	" "	hex_page		" "	hex_addr				" "	hex_size			
	"D"	"A"	"T"	"A"	"g"													
55555:	"D"	"A"	"T"	"A"	"g"	teplota				Ain0		Ain1		Ain2		Ain3		rezerva
	"R"	"O"	"O"	"T"	"r"													
55551:	root_data_512																	
	"R"	"O"	"O"	"T"	"w"	root_data_512												
55550:	"R"	"O"	"O"	"T"	"w"													
	"E"	"C"	"H"	"O"														
55550:	"E"	"C"	"H"	"O"														
	"T"	"C"	"P"	" "	"s"													
55550:	text																	
	"T"	"C"	"P"	" "	"d"													
55553:	struct_clients (320B)																	
	"C"	"F"	"G"	"_"	"r"													
55550:	"C"	"F"	"G"	"_"	"r"	cfg												
	"C"	"F"	"G"	"_"	"w"	cfg												
55550:	"C"	"F"	"G"	"_"	"w"	cfg												
	"U"	"N"	"L"	"O"	"C"	"K"	kod	/kod										
55550:	"U"	"N"	"L"	"O"	"C"	"K"	" "	"O"	"K"									
	jinak																	
55550:	"errLOCK!_unlock"															hex_kod		

CFG_w - nastaví konfiguraci ARQ, parametr: „1“ nebo „0“.
- odpověď na 55550, potvrzovací text (stejný jako požadavek).

UNLOCK - požadavek na odemknutí (povolení) komunikace, parametry: kód a neg. kód.
- odpověď na port 55550, text „UNLOCK OK“.

Pro neznámý příkaz:
- odpověď na port 55550, text „errLOCK!_unlock“ a hex kód (tento kód se musí použít pro unlock!).

Pro navržený protokol byl pochopitelně vytvořen rovněž software pro PC, aby bylo možné s jednotkou komunikovat. Software byl vytvořen programem *Borland Delphi 7* a bude včetně zdrojových kódů přiložen na CD. Tento software bude ještě popsán v kap. 6.4.4.

6.4.3 Webový server

Jednotka v podstatě představuje jednoduchý webový server. Rozdíl je jen v tom, že jednotka je schopna měřit úrovně napětí, případně nastavovat výstupní linky, apod.

Funkce, která webový server tvoří je `http()`. Funkce je volána z `app()`, pokud je cílový port 80, což odpovídá protokolu HTTP.

Jednotka je postavena na protokolu HTTP/1.0 a podporuje pouze příkaz „GET“. Název souboru může být maximálně 24 znaků. Zpracovává se pouze první řádek požadavku, tzn., že veškeré hlavičkové řádky a případná data jsou ignorovány. Předpokládá se, že celý požadavek bude umístěn pouze v jednom TCP segmentu, resp. v jednom linkovém rámci.

Souborový systém:

Jednotka nemá souborový systém jako takový, má pouze jakousi strukturu pro 16 souborů, která obsahuje jméno (24znaků), scode, stránku paměti, adresu, velikost. Struktura je definovaná v souboru *typedefs.h* viz kap. 5.1. Celá struktura je velká 512B a je umístěna na adrese *ROOT_ADDR*, ve stránce *ROOT_PG*. Z hlediska komunikačního protokolu musí být stránka v paměti FRAM (zapisuje se totiž vždy celých 512B).

Tato struktura, nazvěme ji *ROOT*, tedy obsahuje seznam dostupných souborů. Soubory nesmí být fragmentovány a neměly by přesahovat do jiné stránky paměti (za určitých okolností mohou).

Jednotka nemá definovány žádné funkce pro vytváření souborů, proto je nutné všechny soubory do jednotky ukládat pomocí příkazů komunikačního protokolu popsaného v kap. 6.4.2 (použitím PC softwaru). Tato metoda ukládání souboru není uživatelsky moc přátelská, lepší by bylo vytvořit jiný souborový systém, např. FAT12 nebo FAT16 a vytvořit funkce jednotky pro práci se soubory. To je ovšem vzhledem k tomu, že se soubory nebudou prakticky měnit, zcela zbytečné, navíc by to zpomalilo načítání stránek. Proto tento velmi nedokonalý systém bude dostačující.

Některé soubory jsou povinné a některé musejí být na určené pozici v ROOTu.

Na nulté pozici (číslo souboru 0) musí být soubor *INDEX.HTML*. Vychází to z toho, že pokud v adrese URL není název souboru, jedná se o požadavek na soubor *index.html*. Program je koncipován tak, že pokud není název, bude číslo souboru 0.

První pozice v ROOTu je rezervován pro soubor, který bude modifikován podle naměřených hodnot. Jedná se o soubor, který bude načítán do *html rámce* a bude po určitém intervalu automaticky obnovován. Název souboru není pevný.

Ostatní pozice v ROOTu již nejsou v této verzi programu nijak vymezeny nebo omezeny.

Odesílání objektů:

Po přijetí požadavku „GET“ na konkrétní soubor (objekt) se tento soubor vyhledá v ROOTu. Číslo položky v ROOTu udává číslo souboru. Pro toto číslo souboru se z ROOTu přečte hodnota pole *scode*. Toto pole udává číslo status kódu, který se bude posílat. Status kódem je zde myšleno nejen samotný status (např. *200 OK*), ale zároveň hlavičkový řádek, který informuje o typu souboru (html, jpg, gif, apod.). Číslo status kódu udává položku v poli

status_code (definovaném v modulu *application*) se strukturou *t_stat_files* (viz kap. 5.1). Struktura obsahuje adresu dat a jejich velikost. Tyto údaje se vloží do struktury socketu, číslo souboru se uloží do položky *stat* a odešle se TCP segment. Po potvrzení tohoto segmentu se z položky *stat* vezme číslo souboru a z dané položky ROOTu se vezme adresa a velikost dat samotného souboru a posílají se segmenty, až je celý soubor odeslán. Pokud by číslo status kódu bylo *0xFF*, žádný status kód se neposílá, posílají se přímo data souboru. Toho se může využít u jiných protokolů než HTTP, u kterých se žádné status kódy neposílají.

Tento způsob odesílání souboru (status kód a data zvlášť, třeba z různých pamětí) je zvolený proto, aby nebylo nutné skládat status kód a data za sebe do nějakého speciálního bufferu. Odesílání by se sice zjednodušilo, protože by se odesílal pouze jeden blok dat a použití by bylo univerzálnější, nicméně by to vedlo ke zpomalení komunikace (z důvodů kopírování dat do speciálního odesílacího datového bufferu) a k větší složitosti celého programu.

6.4.4 Software pro PC

Aby bylo možné komunikovat s modulem pomocí komunikačního protokolu popsaného v kap. 6.4.2, bylo nutné vytvořit příslušný software pro PC.

Pro tyto účely byly vytvořeny tři aplikace:

- Program UDP – pro ukládání a čtení dat z pamětí,
- Program TCP_clients – pro zobrazování IP připojených klientů,
- Program Get_data – pro monitorování měřených veličin.

Všechny programy byly vytvořeny pouze pro účely tvorby diplomové práce, nejsou tedy řešeny jako uživatelské aplikace. Ovládání programů není uživatelsky příliš přátelské, nicméně pro účel, pro který byly programy vytvořeny, je to dostačující.

Software byl vytvořen v prostředí *Borland Delphi 7*. Pro komunikaci prostřednictvím UDP byla použita komponenta *IdUDPServer*.

Program UDP

Tento program byl vytvořen za účelem zápisu dat do paměti, resp. ukládání souborů do jednotky. Formulář programu je na Obr. 26.

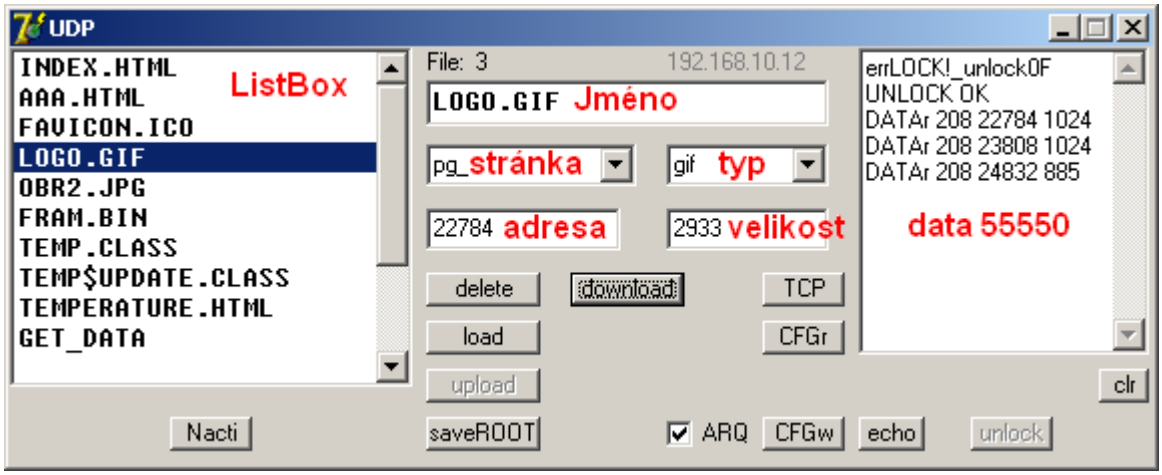
Vlevo je komponenta *ListBox*, ve které se zobrazují názvy souboru z ROOTu. Načtení ROOTu se provede kliknutím na tlačítko „Nacti“.

Označením některého souboru v ListBoxu se zobrazí jméno, typ souboru, stránka paměti, adresa a velikost souboru v prostřední části formuláře. Všechny tyto parametry lze editovat a poté uložit pomocí tlačítka *saveROOT*.

Vymazání souboru se provádí tlačítkem *delete* (dojde k vymazání jména a nastavení výchozích hodnot ostatních polí) a následným stiskem tlačítka *saveROOT*.

Zvolený soubor lze také stáhnout pomocí tlačítka *download*. Po jeho stisku se nabídne SaveDialog, kde se nastaví cesta, kam se má soubor uložit. Jméno souboru bude použito z ROOTu, lze ho však změnit.

Pozn.: Zle stáhnout libovolnou část paměti (nemusí to být soubor). Vyplní se všechna pole a stiskne tlačítko *download*. (nesmí se stisknout tlačítko *saveROOT*! Tím by došlo k vytvoření fiktivního souboru.)



Obr. 26 Program UDP.

Tlačítko *load* slouží k načtení souboru z disku počítače, nabídne se OpenDialog, ve kterém se zvolí požadovaný soubor. Vyplní se pole se jménem a pole velikost souboru. Adresu a stránku paměti musí uživatel zadat ručně. Není nijak kontrolováno, zda-li na zadané adrese už není nějaký soubor uložen, uživatel musí toto kontrolovat sám a adresu se stránkou musí zadávat s velkým rozmyslem, aby nedošlo k přepsání jiného souboru. Načtený soubor se uloží na příslušnou adresu tlačítkem *upload*. Poté musí následovat uložení ROOTu pomocí tlačítka *saveROOT*. Tím je procedura uložení souboru dokončena a od této chvíle je jednotka schopna uložený soubor používat.
Pozn.: Pokud by z nějakého důvodu bylo potřeba pouze naplnit paměť daty a nevytvářet soubor, postup je stejný až na to, že se nakonec neuloží ROOT.

Vpravo na formuláři je umístěno *Memo*, ve kterém jsou zobrazována všechna data přicházející na port 55550. Jedná se o potvrzovací texty příkazů. Uživatel tím dostává potvrzení, že se příkaz vykonal. Kdyby potvrzovací text nepřišel, uživatel musí příkaz zopakovat. Pod komponentou *Memo* je tlačítko *crl*, které slouží k jeho vymazání.

Tlačítko *TCP* slouží ke zjištění aktuálních stavů socketů. Stavy jsou vypsány v komponentě *Memo* v hexadecimálním formátu.

Tlačítko *CFGr* je určeno pro přečtení konfigurace jednotky (konfigurace ARQ). Odpověď je zobrazena v *Memo*.

Tlačítko *CFGw* slouží k nastavení konfigurace ARQ. Nastavuje se na hodnotu True/False podle stavu *CheckBoxu* umístěného vedle tlačítka.

Tlačítko *echo* slouží k ověření komunikace, po jeho stisku se musí v *Memo* vypsat „ECHO“.

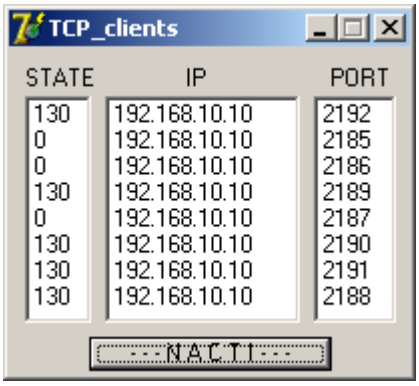
Tlačítko *unlock* slouží k odemknutí komunikace v případě, že byla komunikace uzamčena.

Program TCP_clients

Tento program byl vytvořen pro monitorování a diagnostiku TCP spojení. Formulář programu je na Obr. 27.

Formulář obsahuje tři *ListBoxy*. Levý *ListBox* obsahuje stavy *socketů*, prostřední pak IP adresu a pravý udává číslo portu.

Po stisku tlačítka - - -*N A C T I* - - - se všechny *ListBoxy* aktualizují.



Obr. 27 Program TCP_clients.

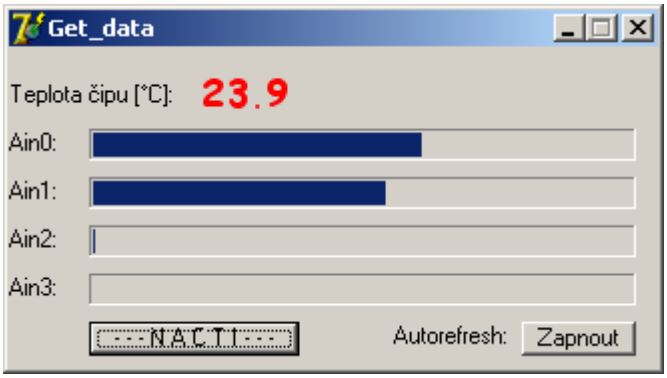
Program Get_data

Tento program byl vytvořen pro monitorování naměřených hodnot. Dále slouží k demonstraci možnosti monitorování pomocí komunikace komunikačním protokolem s využitím UDP, jenž je jakousi alternativou monitorování přes webové rozhraní. Formulář programu je na Obr. 28.

Program zobrazuje teplotu čipu ve stupních Celsia a naměřené úrovně na čtyřech analogových vstupech pomocí komponentů *ProgressBar*.

Stiskem tlačítka - - -*N A C T I* - - - se všechny hodnoty aktualizují.

Druhé tlačítko souží k *Zapnutí* a *Vypnutí* autoobnovování, pokud se autoobnova zapne, jsou měřené hodnoty aktualizovány dvakrát za sekundu.



Obr. 28 Program Get_data.

7 Závěr

V práci bylo stručně popsáno rozhraní Ethernet, jeho verze a struktura linkového rámce. Dále byly popsány použité síťové protokoly, jejich hlavičky a význam v síťovém modelu TCP/IP.

Pro dodaný modul s mikroprocesorem C8051F120 byl navržen a realizován modul EthernetCP2200 s řadičem CP2200 pro připojení jednotky k síti Ethernet. Modul byl podle zadání realizován jako společný modul pro rozhraní Ethernet a USB. USB však nebylo náplní této práce.

Hlavní náplní práce bylo vytvořit software, který bude umožňovat vzdálené řízení jednotky s využitím webového rozhraní. Byly vytvořeny softwarové moduly pro obsluhu ethernetového řadiče, monitorování napětí na analogových vstupech, komunikaci po sběrnicích I2C a SPI a obsluhu displeje LCD. Dále byly vytvořeny SW moduly umožňující komunikaci pomocí síťových protokolů všech vrstev modelu TCP/IP.

Pomocí vytvořených softwarových modulů byl vytvořen software pro mikrokontrolér C8051F120 umožňující monitorování a řízení jednotky přes internet pomocí webového rozhraní. Tento software plní funkci webového serveru.

Společně se softwarem pro mikrokontrolér byl vytvořen software pro PC, který umožňuje správu jednotky – ukládání souborů pro webový server (webové stránky), stahování dat z pamětí jednotky, monitorování měřených hodnot, apod. Tento software byl vytvořen pro potřeby vývoje jednotky a není primárně určen pro uživatelské řízení.

Pro demonstraci funkce jednotky byly napsány webové stránky v kódu HTML. Stránky obsahují aktuální datum a čas na jednotce, aktuální teplotu čipu mikrokontroléru a aktuální naměřené úrovně na analogových vstupech. Další stránka slouží k nastavení data a času a další nastavení jednotky. Tyto stránky jsou pouze jako ukázka možností jednotky, nicméně požadované funkce plní. Pro praktické užívání jednotky s hlavicí optického spoje bude efektivnější využití alternativ k HTML kódu, např. využití JAVA appletů.

Připojení jednotky k hlavici optického spoje nebylo náplní této práce, proto konkrétní funkce byla testována pouze na obecné úrovni. Bylo testováno načítání webových stránek v prohlížečích Mozilla Firefox 1.0, Internet Explorer 6.0 a Opera 5. Byla také simulována ztráta paketů a tím byla testována schopnost implementovaného protokolu TCP opravovat chyby v přenosu. Během testování nenastala žádná vážná situace, kdy by přenos selhal.

8 Literatura

[1] *EMBEDDED ETHERNET DEVELOPMENT KIT USER'S GUIDE*, Silicon Laboratories 2006. 34 stran. Dostupné na WWW: <http://www.silabs.com/public/documents/tpub_doc/evbdsheet/Microcontrollers/en/Ethernet-DK.pdf> [cit. říjen 2006].

[2] *TCP/IP LIBRARY PROGRAMMER'S GUIDE*, Silicon Laboratories 2006. 56 stran. Dostupné na WWW: <http://www.silabs.com/public/documents/tpub_doc/anote/Microcontrollers/Precision_Mixed-Signal/en/an237.pdf> [cit. říjen 2006].

[3] *C8051F120/1/2/3/4/5/6/7, C8051F130/1/2/3*, Silicon Laboratories 2005. Datasheet. 350 stran. Dostupné na WWW: <http://www.silabs.com/public/documents/tpub_doc/dsheet/Microcontrollers/Precision_Mixed-Signal/en/C8051F12x-13x.pdf> [cit. říjen 2006].

[4] *SINGLE-CHIP ETHERNET CONTROLLER*, Silicon Laboratories 2006. Datasheet. 106 stran. Dostupné na WWW: <http://www.silabs.com/public/documents/tpub_doc/dsheet/Microcontrollers/Interface/en/CP2200.pdf> [cit. říjen 2006].

[5] *Wikipedie: Otevřená encyklopedie: Ethernet*, Dostupné na WWW: <http://cs.wikipedia.org/wiki/Ethernet> [cit. listopad 2006].

[6] Libor Dostálek, Alena Kabelová. *Velký průvodce protokoly TCP/IP a systémem DNS*. Computer Press Praha 2000.

[7] Doc. Ing. Ivo Provazník, Ph.D. *Počítače a programování 1*. Skriptum VUT v Brně, 2002.

[8] OUWEHAND, Peter. How to use a HD-44780 based charakter LCD. Dostupné na WWW: <http://ouwehand.net/~peter/lcd/lcd.shtml> [cit. březen 2007].

[9] *THE I2C-BUS SPECIFICATION*, Philips Semiconductors Dostupné na WWW: <http://www.esacademy.com/faq/i2c/I2C_Bus_specification.pdf > [cit. březen 2007].

[10] *CARRIER SENSE MULTIPLE ACCESS WITH COLLISION DETECTION (CSMA/CD) ACCESS METHOD AND PHYSICAL LAYER SPECIFICATIONS*, IEEE Standard for Information technology Dostupné na WWW: <http://standards.ieee.org/getieee802/download/802.3-2005_section1.pdf> [cit. květen 2007].

[11] Doc. Dr. Ing. Zdeněk Kolka, *Počítačové a komunikační sítě*. Skriptum VUT v Brně, 2007.

[12] *PCF8583 CLOCK/CALENDAR WITH 240X8-BIT RAM*, Philips 1997. Datasheet. 29 stran. Dostupné na WWW: <http://www.datasheetcatalog.org/datasheet/philips/PCF8583_5.pdf > [cit. březen 2007].

[13] *SPECIFICATION FOR BTHQ 22005VSS-SMN-LED WHITE*. Batron 2002. 17 stran. Dostupné na WWW: <http://www.farnell.com/datasheets/47103.pdf> [cit. leden 2008].

[14] *FM20L08 1MBIT BYTEWIDE FRAM MEMORY*. Ramtron 2005. Datasheet. 14 stran. Dostupné na WWW: <http://www.ramtron.com/files/datasheets/FM20L08T_r1.8.pdf> [cit. březen 2008].

[15] *AT29LV040A 4-MEGABIT FLASH MEMORY*, Atmel 2005. Datasheet. 16 stran. Dostupné na WWW: <http://www.atmel.com/dyn/resources/prod_documents/doc0334.pdf> [cit. březen 2008].

[16] *TWO-WIRE SERIAL EEPROM 24C08*. Atmel 2007. 27 stran. Dostupné na WWW: <http://www.atmel.com/dyn/resources/prod_documents/doc0180.pdf> [cit. květen 2008].