VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

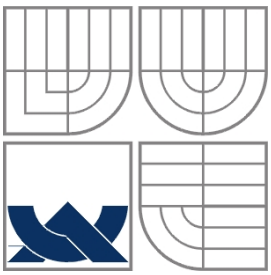
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

UDP A TCP KOMUNIKACE NA FITKITU

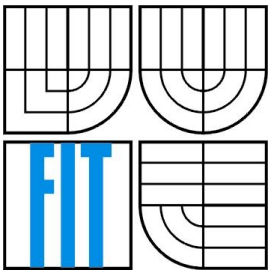
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE MARTIN MUSIL
AUTHOR

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

UDP A TCP KOMUNIKACE NA FITKITU

UDP AND TCP COMMUNICATION FOR FITKIT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MARTIN MUSIL
AUTHOR

VEDOUCÍ PRÁCE — ING. ZDENĚK VAŠÍČEK
SUPERVISOR

BRNO 2010

Abstrakt

Cílem bakalářské práce bylo rozšířit výukovou platformu FITkit o možnost komunikace prostřednictvím protokolů TCP a UDP. Komunikaci zajišťuje Ethernetový modul ENC28J60 od firmy Microchip. Komunikační protokoly zpřístupňuje knihovna enc28j60 implementovaná v jazyce C. V teoretické části práce se zabýváme rozborem síťových protokolů, které budou implementovány. Práce pokračuje popisem vlastností a parametrů připojovaného modulu. Následuje návrh knihovny, při kterém je kladen důraz zejména na omezené výpočetní zdroje platformy. V implementační části práce je popsáno aplikační rozhraní a základní principy fungování knihovny.

Abstract

The aim of this thesis was to extend educational platform FITkit for the possibility of communication via TCP and UDP protocol. Communication is provided by Ethernet module ENC28J60 from Microchip. Communication protocols are provided by library enc28j60, which is implemented in C language. The theoretical part deals with analysis of network protocols to be implemented. Work continues describing the characteristics and parameters of the connected module. Followed by a design library in which is particular emphasis on limited computing resource of the platform. In the implementation of the work is to describe the application interface and the basic principles of the library.

Klíčová slova

Fitkit, ENC28J60, TCP/IP , mikrokontrolér, MSP430, komunikace, Internet, protokol, TCP, UDP

Keywords

Fitkit, ENC28J60, TCP/IP , microcontroller, MSP430, communication, Internet, protocol, TCP, UDP

Citace

Martin Musil: UDP a TCP komunikace na FITkitu, bakalářská práce, Brno, FIT VUT v Brně, 2010

UDP a TCP komunikace na FITkitu

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Zdeňka Vašíčka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Martin Musil
10.5.2010

Poděkování

Tímto bych chtěl poděkovat svému vedoucímu Ing. Zdeňku Vašíčkovi za odborné vedení, konzultace a připomínky, které mi pomohly při řešení bakalářské práce.

Obsah

1. Úvod

Vestavěné systémy se za poslední desetiletí staly nedílnou součástí každodenního života. Již dnes se s nimi můžeme setkat prakticky na každém kroku a to jejich význam a oblasti nasazení ještě významně porostou. Vestavěné systémy tak nalezneme v mobilních telefonech, televizorech, kamerových a zabezpečovacích systémech i automobilovém průmyslu.

S obrovským rozvojem Internetu se pro vestavěná zařízení otevřela nová oblast uplatnění. V dnešní době je kladen stále větší důraz na připojení některých vestavěných systémů do sítě Internet. Může se jednat například o zabezpečovací zařízení, kamery, meteorologické senzory nebo i domácí spotřebiče a termostaty. Výhoda připojení těchto zařízení do sítě Internet je jak hromadná možnost sběru dat a regulace z jednoho místa, tak i vzdálený přístup, třeba i z druhého konce světa.

Tématem práce je navrhnout a implementovat sadu knihovních funkcí, která by umožňovala připojit modul ENC28J60 k výukové platformě FITkit, a tak rozšířit jeho možnosti o připojení k síti Internet. Knihovna musí dále umožňovat komunikaci prostřednictvím protokolů TCP a UDP.

Práce je členěna na několik kapitol. Obsahem druhé kapitoly je rozbor síťového modelu TCP/IP, jeho členění na vrstvy a specifikace operací probíhajících na každé z nich. Třetí kapitola rozebírá jednotlivé síťové protokoly nutné pro implementaci knihovny, jejich specifika, principy činnosti. Ve čtvrté kapitole je popsán modul ENC28J60, je zde prezentováno blokové schéma modulu, popis jednotlivých částí a komunikačního rozhraní. Pátá kapitola se zabývá návrhem komunikační knihovny. Jsou zde rozebrány otázky návrhu, mezi které patří komunikace s Ethernetovým modulem, správa vnitřní paměti modulu a celková struktura komunikační knihovny. V šesté kapitole se nachází celkový popis a implementace knihovny. Jsou zde prezentována řešení jednotlivých dílčích knihoven, jako je knihovna pro práci s pamětí a knihovny komunikačních protokolů, ale i popis aplikačního rozhraní knihovny a jeho základních funkcí. Sedmá kapitola obsahuje dosažené výsledky z měření přenosových rychlostí komunikace.

2. Architektura TCP/IP

Otázka síťové komunikace je věc natolik obsáhlá, že ji nejsme schopni efektivně řešit jako celek. Vzhledem k charakteru počítačových sítí a služeb, které musíme zajistit, se jako nejlepší řešení ukázalo rozdělení základního programového vybavení sítě na hierarchicky uspořádané vrstvy. Každá z nich má na starosti přesně vymezený okruh úkolů, jenž smí provádět. Mechanismy, pomocí kterých tyto úkoly zajišťuje, pak nabízí k využití vyšší vrstvě. Zároveň ale využívá služeb vrstev nižších. Tomuto rozdělení se říká vrstvový model (anglicky *layered model*). Mezi nejznámější modely patří ISO/OSI, který byl standardizován mezinárodní organizací ISO a TCP/IP, který je dnes dominantním standardem pro komunikaci v síti Internet. Informace v této kapitole jsou volně převzaty z [1] a [2].

Zatímco referenční model ISO/OSI definuje sedm vrstev síťového programového vybavení, model TCP/IP je rozčleněn pouze do čtyř. Každý model má odlišnou definici jednotlivých vrstev a protokolů pracujících na ní, proto se jejich vrstvy obecně nedají srovnávat. V praxi je však třeba využívat komunikační zařízení vyhovující ISO OSI pro přenos IP-paketů nebo naopak realizovat služby podle ISO OSI na modelu TCP/IP.

Hlavní odlišnosti mezi těmito dvěma modely vyplývají z rozdílných postojů jejich tvůrců. Model ISO/OSI se snaží vměstnat velkou spoustu funkcí, včetně zajištění spolehlivého přenosu, přímo do komunikační podsítě, která bude v důsledku toho poměrně dosti složitá. Naopak na hostitelské počítače zbude pouze jednoduchá úloha.

Model TCP/IP je zato pravým opakem. O zajištění spolehlivého přenosu se musí postarat koncové stanice. Komunikační podsíť má jednoduchý charakter, nemusí plýtvat prostředky na přeposílání paketů a může lépe využít přenosovou kapacitu. Z toho ovšem plyne, že může docházet ke ztrátě paketů bez upozornění nebo snahy o nápravu. Komunikační síť by ale neměla zahazovat pakety bezdůvodně, měla by naopak vyvíjet maximální snahu pakety doručit a zahazovat je až v nezbytných případech, například při poškození dat během přenosu nebo při výpadku spojení.

Vrstva síťového rozhraní

Vrstva síťového rozhraní zajišťuje přenos jednotlivých bitů po fyzickém médiu, kterým může být například metalický nebo optický kabel, ale také se může jednat o radiové spojení. Úkolem fyzické vrstvy je přizpůsobit se konkrétním přenosovým prostředkům a vytvořit jednotné rozhraní pro přenos dat.

Této vrstvy se týkají standardy, které definují například elektrické a mechanické vlastnosti rozhraní. V standardech je například uvedeno, jaké hodnoty napětí budou reprezentovat logickou 0 a 1, zda se jedná o synchronní nebo asynchronní přenos, na jakém kmitočtu bude přenos dat probíhat, tvar konektoru a počet pinů, jaké signály budou na kterém kabelu přenášeny, jejich význam a časový průběh.

Mezi nejznámější síťové technologie patří Ethernet, Wi-fi, FDDI, PPP a Token ring. V sítích LAN v současné době převládá technologie Ethernet.

Síťová vrstva

Síťová vrstva, někdy také označovaná jako IP vrstva (anglicky také *Internet layer*), již není závislá na přenosové technologii. IP vrstva zajišťuje, aby se datagramy dostaly od odesílatele až k příjemci, a to nejen v rámci lokální sítě, ale i mezi různými podsítěmi mezi kterými mohou ležet i další podsítě.

Tato vrstva je navržena pro maximální přenosovou rychlost na úkor spolehlivosti. Spolehlivost přenosu řeší až vyšší vrstvy, síťová vrstva se nestará o nápravu když dojde k poškození nebo ztrátě dat.

Základní přenosovou jednotkou je IP datagram. Každý z nich nese úplné směrovací informace (zejména IP adresu odesílatele a příjemce), takže síť může přenášet datagramy samostatně. V internetu existuje více možných cest mezi odesílatelem a příjemcem a tak je možné, že každý datagram půjde jinou cestou a v důsledku toho mohou dorazit v jiném pořadí než byly odeslány.

Transportní vrstva

Transportní vrstva zajišťuje přenos mezi dvěma koncovými účastníky, kterými jsou v případě modelu TCP/IP přímo aplikační programy, dále umožňuje regulaci toku dat oběma směry a je schopna zajistit spolehlivý přenos dat.

Vrstva zajišťuje jak spojovaný tak nespojovaný přenos dat a to podle požadavků konkrétní aplikace. Některé aplikace se bez spolehlivého přenosu dat neobejdou, jde o větší část služeb na internetu jako jsou webové aplikace, informační systémy, vzdálené sezení apod., kde si nemůžeme dovolit jakoukoli ztrátu dat. Naopak pro videohovory a podobné služby, jež nevyžadují stoprocentní správnost dat, je výhodnější použít nespojovanou službu, kde se absence zajištění spolehlivosti kladně projeví na rychlosti a odezvě přenosu. V tomto konkrétním případě nám nevadí, když se některý z paketů ztratí či poškodí. Ztráta znamená pouze nepatrnou chybu v zobrazení, která nemusí být ani okem postřehnutelná.

Aplikační vrstva

Na aplikační vrstvě už nepracují protokoly jako na předchozích vrstvách nýbrž samotné síťové aplikace. Na rozdíl od modelu ISO/OSI komunikují přímo s transportní vrstvou. Případné prezentační a relační služby, které existují v modelu OSI jako samostatné vrstvy, si již aplikace musí realizovat sama.

3. Síťové protokoly

Protokol je soubor pravidel podle kterého probíhá komunikace mezi dvěma koncovými body. V nejjednodušší podobě protokol definuje pravidla řídící syntaxi, sémantiku a synchronizaci vzájemné komunikace. Protokoly mohou být realizovány jak hardwarově tak softwarově nebo kombinací obojího. Informace v této kapitole jsou volně převzaty z [1] a [2].

3.1 Ethernet

Ethernet je technologie používaná pro budování lokálních sítí LAN. V referenčním modelu ISO/OSI realizuje fyzickou a linkovou vrstvu, v modelu TCP/IP zase vrstvu síťového rozhraní. Byl vyvinut firmou Xerox původně za účelem sdílet na tehdejší dobu velice drahé tiskárny. Ethernet se však pro svou jednoduchost a snadnou implementaci začal ve stále větší míře používat na vytváření lokálních počítačových sítí LAN. Dnes je jedním z nejrozšířenějších linkových protokolů. Ethernet byl normalizován institutem IEEE jako norma IEEE 802.3. Později byla norma převzata organizací ISO jako ISO 8802-3.

Klasický Ethernet používal sběrnicovou topologii a sběrnice byla tvořena koaxiálním kabelem. V dnešní době se koaxiální kabeláž již nepoužívá, nahradila ji kroucená dvojlinka. Také topologie se změnila ze sběrnicové na hvězdicovou, v jejímž středu je rozbočovač (anglicky *hub*). Rozbočovač ale sběrnici napodobuje, kopíruje signál přicházející z jednoho rozhraní na ostatní. Rozbočovače jsou již většinou nahrazeny přepínači (anglicky *switch*), které jsou na rozdíl od rozbočovačů inteligentní. Switch má v paměti uloženou tabulku se záznamy, které počítače má připojeny na svých rozhraních. Příchozí datagram pak pošle jen na rozhraní, kde je připojen adresovaný počítač nikoli na všechna.

Ethernetový kabel je tvořen čtyřmi páry kroucené dvojlinky. V dřívějších dobách pro propojení počítačů rychlostí 10Mb/s stačily páry dva, jeden se používal pro vysílání a druhý pro příjem dat. Dva páry stačily ještě i na rychlost 100Mb/s, musel být však použit kvalitnější kabel. Pro dosažení vyšších rychlostí byly využity i zbývající dva páry. Je tak možné docílit rychlosti až 1Gb/s. Místo kroucené dvojlinky se již také využívá optických vláken, s nimiž jsme schopni realizovat rychlosti vyšší jak 10Gb/s.

Aby bylo možné počítače v lokální síti jednoznačně identifikovat, každé síťové kartě je hned při výrobě přiřazena celosvětově jedinečná MAC adresa. Její délka je 48 bitů a zapisuje se nejčastěji jako šestice dvojciferných hexadecimálních čísel oddělených dvojtečkami, například „01:23:45:67:89:AB“.

V moderních síťových zařízeních je možné MAC adresu síťové karty změnit, takže není zaručena jednoznačná identifikace zařízení v lokální síti LAN.

Verze Ethernetu

Ethernet se postupně vyvíjel podle zvyšujících se požadavků na přenosovou rychlost. Existuje tak několik různých verzí.

- **Ethernet** – Původní varianta s přenosovou rychlostí 10Mb/s. Definována pro koaxiální kabel, kroucenou dvojlinku.
- **Fast Ethernet** – Rychlejší verze s přenosovou rychlostí 100Mb/s definovaná standardem IEEE 802.3u. V současnosti se dá se považovat za základní verzi Ethernetu, je k dispozici pro kroucenou dvojlinku a optická vlákna.

- **Gigabitový Ethernet** – Zvýšil přenosovou rychlost až na 1Gb/s. Je definován standardem IEEE 802.3z (pro optická vlákna) a IEEE 802.3ab pro kroucenou dvojlinku.
- **Deseti-gigabitový Ethernet** – Zatím poslední standardizovaná verze (IEEE 802.3ae). Jako médium slouží hlavně optická vlákna.

Formát rámce

Základní komunikační jednotka je rámec, skládá se z šesti polí.

Preambule	MAC příjemce	MAC odesílatele	Typ	Data	CRC
8B	6B	6B	2B	46 – 1500B	4B

Obrázek 3.1. Ethernetový rámec

Rámec začíná preambulí. Ta je součástí fyzické vrstvy a slouží k synchronizaci hodin s příjemcem. Následují MAC adresy cílového a zdrojového rozhraní. Položka typ slouží k rozpoznání protokolu vyšší vrstvy. Minimání délka datové části je 46B. Pokud tato podmínka není splněna, rámec musí být doplněn nulami. Rámec je ukončen kontrolním součtem, který je vypočítán ze všech polí s vyjímkou preambule.

3.2 Protokol ARP

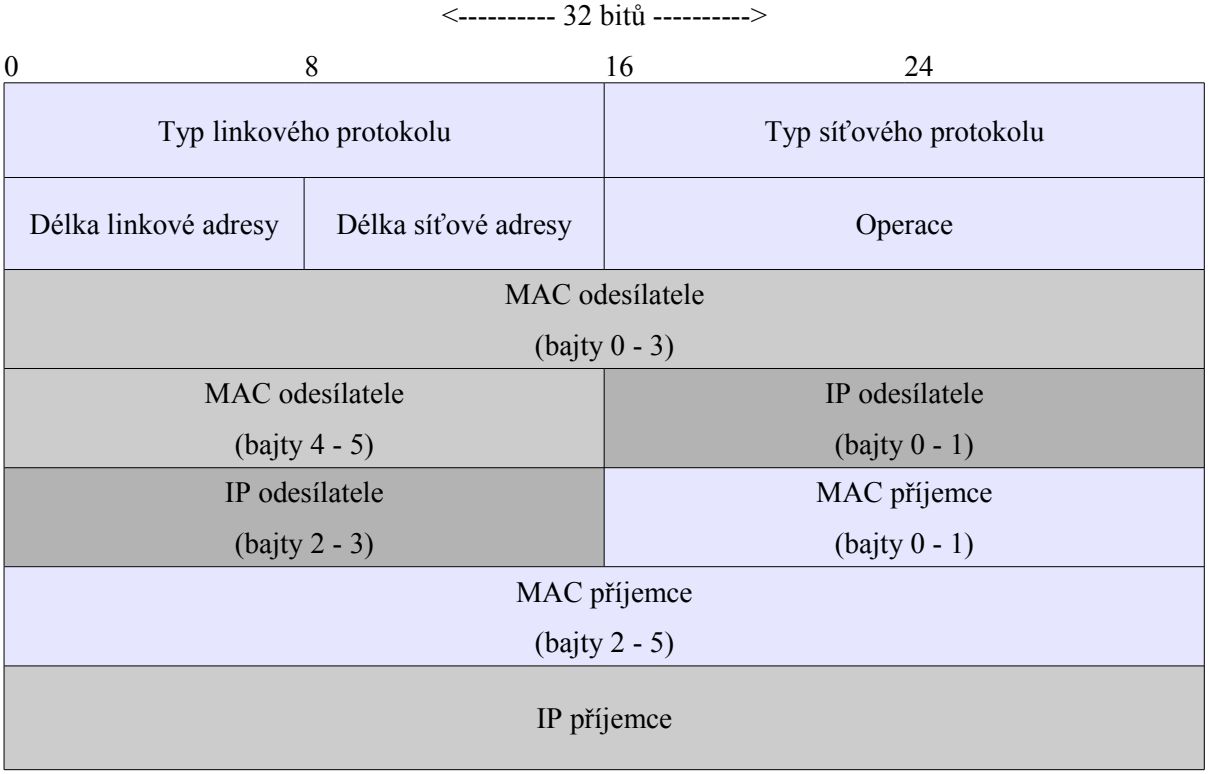
Protokol ARP (z anglického *Address Resolution Protocol*) je protokol využívaný protokolem IPv4 k namapování IP adres na hardwarové MAC adresy používané Ethernetem. Protokol ARP pracuje pod síťovou vrstvou jako součást rozhraní mezi síťovou a linkovou vrstvou ISO/OSI.

Protokol ARP se používá v situaci, kdy je potřeba odeslat IP datagram počítači, který leží ve stejné podsíti jako odesílatel. Odesílatel datagramu sice zná svou IP adresu a IP příjemce, ale aby mohl sestavit Ethernetový rámec a odeslat ho po síti, musí znát také MAC adresu sebe a příjemce. Svou MAC adresu samozřejmě zná, ale problémem je zjistit MAC adresu příjemce. Právě tento problém řeší protokol ARP. Pomocí něho odesílatel odešle ARP dotaz (anglicky *ARP request*) obsahující hledanou IP adresu a údaje o sobě, jako je jeho IP a MAC adresa. Tento dotaz odešle tzv. linkovým broadcastem, takže dotaz obdrží všechny počítače v síti. Vlastník hledané IP adresy pak odpoví přímo odesílateli a sdělí mu svou MAC adresu. Poté už nic nebrání odesílateli sestavit rámec a odeslat ho příjemci.

Informace o MAC adresách odpovídajících jednotlivým IP adresám se ukládají do ARP tabulky. Není tak nutné pro každý odesílaný datagram zjišťovat MAC adresu znovu. Data v ARP tabulce se ale neuchovávají trvale, mají nastavenou dobu platnosti, po kterou mohou být považovány za aktuální. Po vypršení této doby se záznamy z tabulky mažou.

ARP hlavička

ARP žádost se skládá pouze z hlavičky, neobsahuje datovou část. Hlavičku protokolu a její položky znázorňuje obrázek 3.2.



Obrázek 3.2. Hlavička ARP protokolu

První pole, typ linkového protokolu, identifikuje protokol linkové vrstvy. Typ síťového protokolu udává číselný kód protokolu síťové vrstvy, pro který byl ARP dotaz určen. Dále jsou uvedeny údaje o délkách linkové a síťové adresy, Ethernetová MAC adresa má standardní délku 6B, IPv4 adresa zase 4B. Položka operace pak rozlišuje, zda se jedná o ARP dotaz nebo odpověď. Dále následují MAC adresa odesílatele, IP odesílatele, MAC adresa příjemce (při dotazu bývá vyplněna nulami) a IP adresa příjemce.

ARP hlavička bývá vložena do Ethernetového rámce, kde MAC odesílatele je známa a jako MAC příjemce se použije broadcastová MAC adresa (samé jedničky, tzn. FF:FF:FF:FF:FF:FF). Odpověď se již nemusí posílat přes broadcast, ale může být odeslána přímo žadateli.

3.3 Protokol IP

IP protokol je základní protokol celého internetu. Používá datagramy pro komunikaci po síti s přepojováním paketů (anglicky *packet switching network*). Pracuje na síťové vrstvě a je součástí rodiny protokolů TCP/IP.

IP protokol poskytuje služby síťové vrstvy pro propojování počítačů do počítačové sítě. Každý počítač je v ní jednoznačně identifikován jednou nebo více unikátními IP adresami. Základní datovou jednotkou na síťové vrstvě je paket, někdy také nazývaný jako datagram. Každý paket nese

informaci o IP adrese odesílatele a samozřejmě také IP adrese příjemce. Dále obsahuje sadu řídících informací.

Síťový protokol IP přenáší datagramy mezi dvěma uzly pomocí směrovačů (routerů). Nejsložitějším úkolem IP směrovače je výběr optimální linky, na kterou se má datagram přeposlat, aby dosáhl svého cíle v síti. Tento proces se nazývá směrování (anglicky *routing*). Při něm si směrovač vytvoří směrovací tabulku obsahující informace podle kterých se směrovač rozhodne, na jakou linku přepošle příchozí paket. Tento proces se neopakuje pro každý datagram, je totiž časově a výpočetně náročný, proto se směrovací tabulka obnovuje v pravidelných intervalech.

IP sítě běžně používají dynamické směrovací protokoly k nalezení alternativní trasy, pokud se linka stane dočasně nedostupnou. To poskytuje značnou odolnost proti výpadkům v síti, ale stále nezaručuje spolehlivé doručení paketu a tak se obvykle pro komunikaci používají protokoly na vyšších vrstvách. Aplikace, které jsou vůči výpadku imunní používají jednoduchý transportní protokol UDP (anglická zkratka pro *User Data Protocol*), který nezaručuje doručení paketu. Většina aplikací ale potřebuje hlavně spolehlivý přenos. Ten je poskytován transportním protokolem TCP (anglická zkratka pro *Transmission Control Protocol*).

IP adresa

IP adresa je 32 bitová hodnota, která jednoznačně identifikuje zařízení v rámci sítě Internet. Některá zařízení mohou mít více IP adres, ale jednu IP už nemůže mít více zařízení. Protože je IP adresa tvořena 32 bity, je teoreticky možné mít v internetu přes 4,2 miliardy koncových stanic. Ačkoli se toto číslo zdá obrovské, vyčerpání adresního prostoru je s obrovským rozvojem dostupnosti internetu nevyhnutelné, a tak se pomalu začínají zavádět IP adresy s délkou 128 bitů.

Každá adresa se skládá ze dvou částí, adresy sítě,ke které je počítač připojen, a adresy počítače, která identifikuje počítač v lokální síti. Správce konkrétní IP sítě je oprávněn rozdělit si hostitelské adresy podle svých potřeb, bez nutnosti spolupracovat se správci jiných sítí.

IP adresa se obvykle zapisuje v desítkové notaci. To znamená, že každý ze čtyř bajtů adresy je reprezentován jako číslo od 0 do 255. Tyto čísla bývají oddělena tečkami. Jako příklad poslouží adresa 147.229.15.157.

IP adresy dělíme na *unicastové, multicastové a broadcastové*. Unicastová adresa patří vždy jednomu konkrétnímu počítači v internetu. Data se posílají vždy z jednoho počítače druhému. Jedná se o nejběžnější způsob zasílání dat v internetu. Multicastová adresa adresuje určitou skupinu počítačů, využívá se pro posílání dat z jednoho počítače na určitou skupinu dalších počítačů. Tento způsob se využívá například pro streaming videa. Poslední typ, broadcastová adresa, adresuje všechny počítače v dané síti LAN.

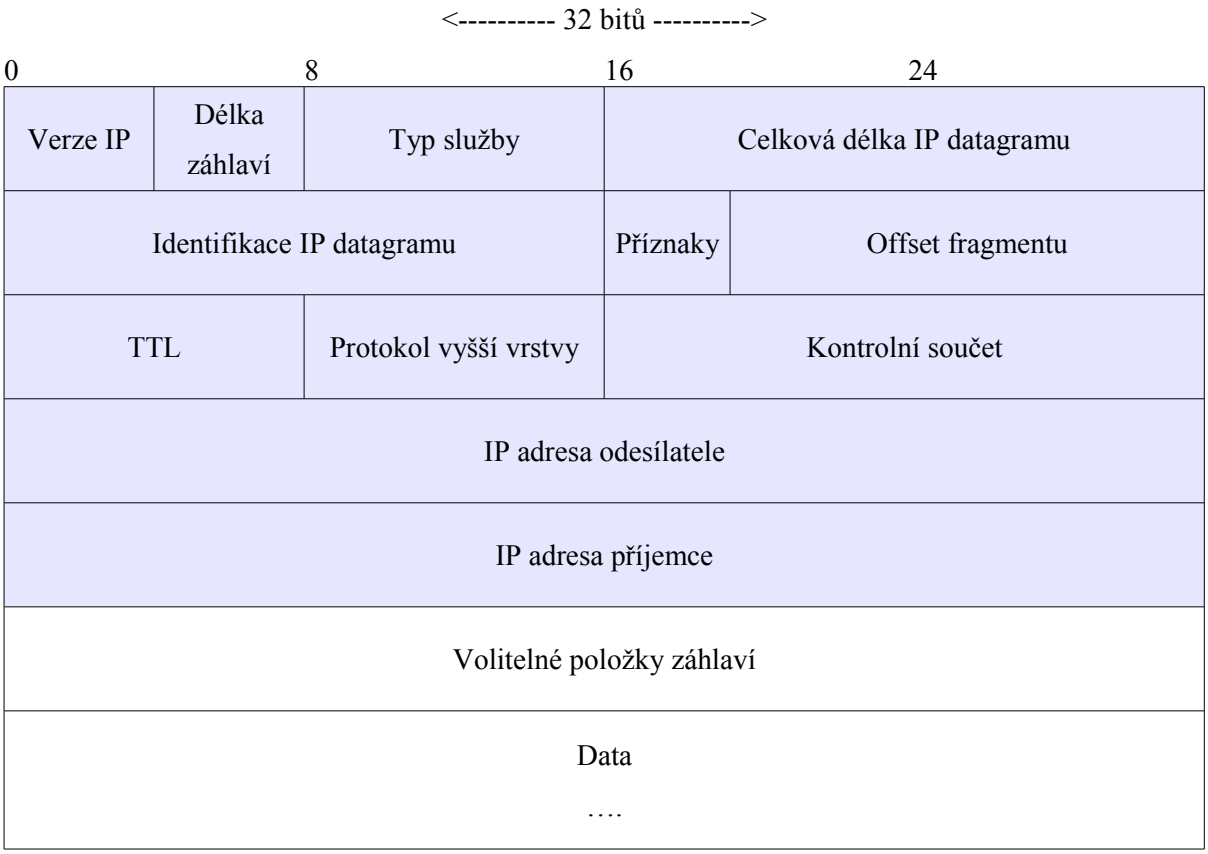
IP adresa sítě je dána logickým součinem IP adresy a další 32-bitové hodnoty, síťové masky, někdy také nazývané jako „maska podsítě“. Maska tak určuje hranici mezi adresou počítače a sítě. Masky mohou být zapsány ve stejné notaci, jako IP adresy, například 255.255.255.0, ale také se můžeme setkat se zkráceným zápisem, kde se za IP adresu počítače zapisuje za lomítko počet logických „1“ v masce, například 193.71.2.15/24 pro masku 255.255.255.0.

Jako příklad pro zachycení vztahu IP adresy, IP adresy sítě a masky podsítě poslouží adresa počítače 193.71.2.15. Pokud je maska podsítě 255.255.255.0, tak adresu sítě dostaneme logickým součinem masky a IP počítače v síti, tzn. IP sítě bude 193.71.2.0. 24 bitová adresa sítě zanechává posledních osm bitů k adresaci stanic uvnitř sítě. To nám zanechává 256 použitelných kombinací, avle 0 je rezervována pro adresu sítě a 255 pro broadcastovou adresu. Zbývá nám adresový prostor pro 254 počítačů v rozsahu 193.71.2.1 – 193.71.2.254.

Síťové masky byly dříve k adresám přiřazovány podle třídy IP adresy. Od roku 1993 se začal používat systém CIDR (anglicky *Classless Inter-Domain Routing*, česky *beztřídní mezidoménové směrování*), ve kterém je možno nastavit libovolnou masku sítě, a tak efektivněji využít adresní prostor.

IP Hlavička

Každý IP datagram se skládá ze dvou částí: hlavičky a datové čísti. Obsah datové části je zřejmý, jsou jím data odesílaná vyšší transportní vrstvou. Proto se zaměříme především na hlavičku datagramu. Ta má minimální velikost 20 bajtů a obsahuje všechny nezbytné informace o datagramu.



Obrázek 3.3. Hlavička IP datagramu

Na prvních čtyřech bitech hlavičky je uložena verze IP protokolu. Kromě klasického IP verze 4 se kvůli nedostatku volných adres stále častěji setkáváme s IP verze 6. Postupně na tuto verzi přejde celý internet. Další položka určuje velikost hlavičky. Na čtyřech bitech lze reprezentovat pouze 16 hodnot, proto je údaj uložen po vydělení čtyřmi. Maximálně tedy může mít hlavička 60 bajtů. Povinné položky zabírají 20 bajtů, zbylých 40 bajtů může posloužit na volitelné položky hlavičky. Typ služby je v praxi nepoužívaná položka, dříve navrhovaná pro zajištění přednostního doručování paketů.

Délka IP datagramu obsahuje údaj o délce dat včetně hlavičky. V následující položce je 16-ti bitové identifikační číslo paketu, které slouží hlavně pro správné sestavení fragmentovaných datagramů. Právě pro fragmentaci jsou vyhrazeny dvě následující položky, v první jsou příznaky, které oznamují, zda je datagram fragmentovaný a druhá obsahuje případný offset vůči začátku datagramu.

Doba života datagramu (TTL) slouží k zamezení nekonečného cestování datagramu po síti. Každý router, přes který datagram projde, sníží hodnotu TTL o jedničku. Pokud se hodnota TTL dostane až na nulu, router datagram zahodí a informuje odesílatele prostřednictvím protokolu ICMP. Počáteční hodnotu položky TTL většinou nastavuje jádro operačního systému, ale dá se nastavit i explicitně.

Položka s protokolem vyšší vrstvy obsahuje číselnou identifikaci protokolu, který využívá IP datagram k přenosu po síti. Samotný IP protokol se nepoužívá ke komunikaci, je vždy využíván protokolem vyšší vrstvy, například TCP či UDP, či služebními protokoly ICMP a IGMG. Protokoly ICMP a IGMG jsou sice formálně součástí protokolu IP, ale chovají se jako protokoly vyšších vrstev, tzn. že v datové části IP datagramu mají svou vlastní hlavičku.

Hlavička IP datagramu obsahuje kontrolní součet dat, ale pouze z hlavičky datagramu. Kontrolní součet se tak musí přepočítat při každé změně v IP hlavičce. Přepočítávání probíhá na každém směrovači, protože je směrovač povinen u každého datagramu snižovat hodnotu TTL.

3.4 Protokol ICMP

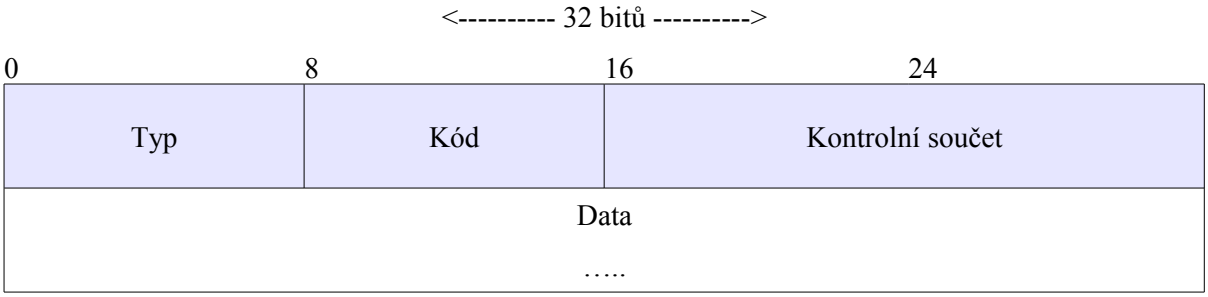
Protokol ICMP (z anglického *Internet Control Message Protocol*) je jedním z nejdůležitějších protokolů z rodiny protokolů TCP/IP. Je definován v dokumentu RFC 792. ICMP je součástí IP protokolu a využívá IP pro přenos svých paketů. Chová se tak částečně jako protokol vyšší vrstvy.

Protože IP protokol není navržen jako spolehlivý, je účelem kontrolních zpráv protokolu ICMP poskytovat zpětnou vazbu o problémech v komunikačním prostředí, nikoli dosáhnout spolehlivosti IP. Protokol ICMP nám může poskytovat informace například o nedostupnosti cílového počítače/routeru, přetížení uzlu, chybě v hlavičce IP protokolu, vypršení TTL, atd. Protokol se dá dále využít ke kontrole správného směrování paketů mezi routery.

Stejně jako není zaručeno doručení IP datagramu, není také zaručeno doručení kontrolních zpráv. Nelze se tedy spolehnout, že nám při ztrátě datagramu bude doručena ICMP zpráva. To v důsledku znamená, že pro zajištění spolehlivého přenosu si musí protokoly vyšších vrstev vytvořit své vlastní kontrolní mechanismy.

ICMP hlavička

Hlavička protokolu z obrázku 3.4 obsahuje pouze tři povinné položky, typ a kód zprávy a kontrolní součet z hlavičky a dat. Za hlavičkou následují data, která závisí na konkrétním kódu a typu zprávy.



Obrázek 3.4. Hlavička ICMP zprávy

3.5 Protokol UDP

UDP protokol (z anglického *User Datagram Service*) je protokol transportní vrstvy definovaný pro použití s protokolem IP. Je definován v dokumentu RFC 768.

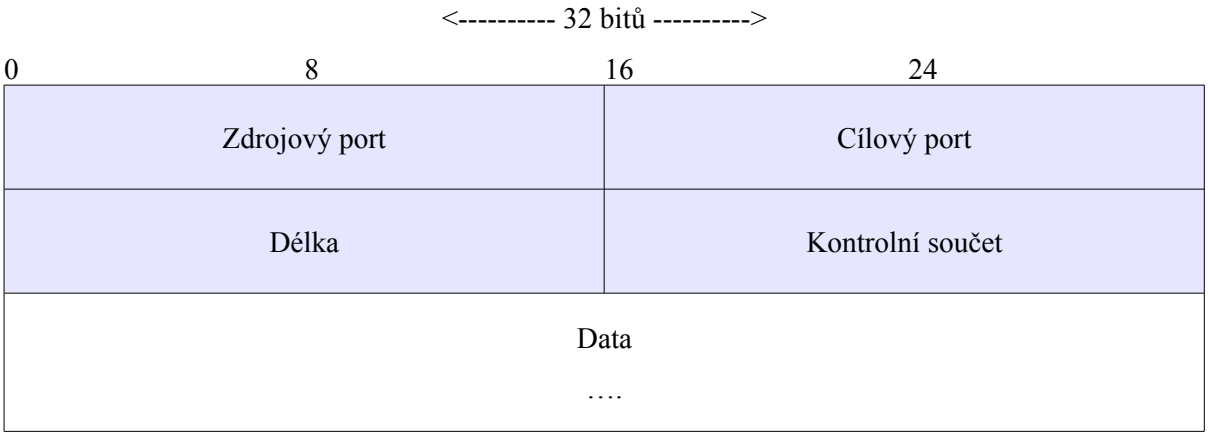
Služby protokolu UDP jsou nespolehlivé a neposkytují žádné záruky doručení či zamezení duplikace datagramů. Není zaručeno ani dodržení pořadí doručení datagramů. Protokol je tím pádem

jednodušší a snižuje režii pro přenos dat. Tato vlastnost je u některých aplikací vítaná, zejména při přenosu hlasu a videa, kde když dojde ke ztrátě jednoho z datagramů, na výsledném vjemu to na kvalitě příliš neubere. Navíc než by byl datagram znovu odeslán, data by již byla neaktuální.

UDP poskytuje minimální, nespolehlivý přenos, ale zato prokazuje nejvyšší možnou snahu pro doručení datagramu (anglicky *best effort service*). V porovnání s ostatními transportními protokoly, zejména TCP, je UDP nespojovaný protokol, to znamená, že nevytváří spojení mezi koncovými stanicemi před odesláním samotných dat. To opět snižuje režii přenosu a snižuje dobu odezvy (anglicky *latency*). Díky těmto vlastnostem poskytuje UDP velmi efektivní a rychlou komunikaci, což může vyhovovat některým aplikacím. Na druhou stranu neobsahuje řízení toku dat, a tak je možné nesprávně navrženou aplikací zahltit přenosovou linku.

UDP hlavička

Formát UDP hlavičky je popsán obrázkem 3.5.



Obrázek 3.5. Hlavička UDP datagramu

První položkou je 16-ti bitové číslo zdrojového portu, které slouží při komunikaci pomocí protokolů TCP a UDP k rozlišení aplikace v rámci počítače. Zdrojový port tak identifikuje aplikaci, která datagram odesílá, cílový port zase aplikaci, které je datagram určen. Délka a kontrolní součet se počítá jak z hlavičky, tak dat.

3.6 Protokol TCP

Protokol TCP je jedním z protokolů transportní vrstvy. Na rozdíl od protokolu UDP zajišťuje spolehlivý přenos, doručení datagramů v pořadí odeslání a řízení toku dat. Úplný popis protokolu je uveden v dokumentu RFC 793.

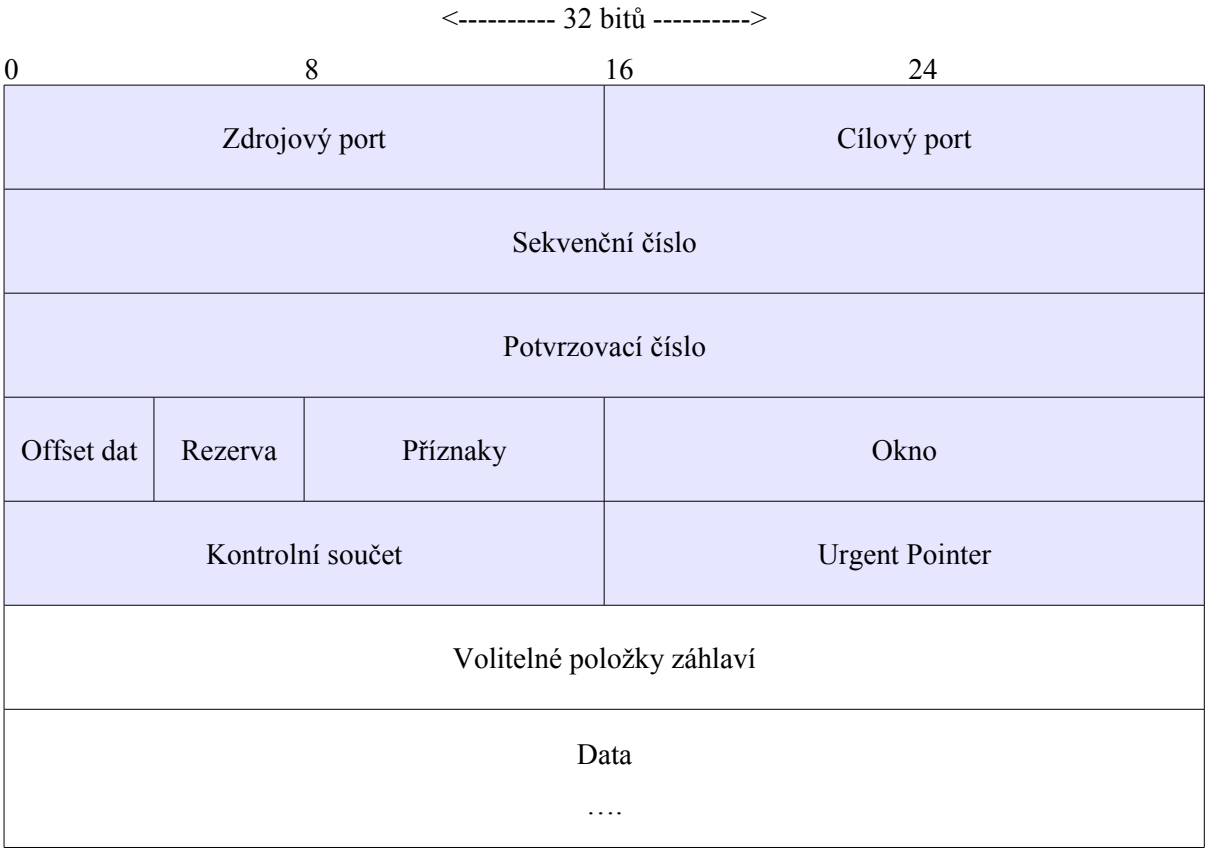
TCP je spojovaným protokolem. To znamená, že před samotným přenosem dat musí mezi koncovými body proběhnout navázání spojení a po skončení se musí spojení zase korektně ukončit. Prioritou TCP je korektní a spolehlivé doručení dat, nikoli nejkratší doba odezvy nebo maximální rychlost spojení. Spolehlivost spočívá především v systému číslování paketů a potvrzování jejich přijetí druhou stranou spojení, v případě nepotvrzení je datagram odeslán znovu. Protokol také přijatá data správně sestaví, pokud byly datagramy doručeny v nesprávném pořadí. K zajištění korektnosti přijatých dat slouží kontrolní součet, který je počítán z tvz. pseudohlavičky a dat. Pseudohlavička se

skládá pouze z některých položek normální TCP hlavičky, a to z IP adres odesílatele a příjemce, délky datové části a kódu protokolu.

Mezi důležité vlastnosti TCP protokolu patří kontrola toku dat. Ta je těsně spjata s poměrem odeslaných/potvrzených datagramů. Protokol pro zvýšení propustnosit spojení neposílá datagramy po jednom, ale po skupinách, tzv. oknech. Pokud jsou všechny pakety druhou stranou přijaty a potvrzovány, odesílací okno se zvětšuje a dovoluje odeslat více datagramů zaráz. Jakmile se ale pakety začnou ztrácet, začne protokol odesílací okno zmenšovat, až nebude docházet ke ztrátám. Díky tomuto mechanismu dokáže protokol optimálně vytížit přenosovou linku a nedochází tak k zahlcení.

TCP hlavička

TCP datagram se skládá z hlavičky protokolu a aplikačních dat. Hlavička má minimální velikost 20B a její formát je popsán obrázkem 3.6.



Obrázek 3.6. Hlavička TCP datagramu

První položkou je 16-ti bitové číslo zdrojového portu, které slouží při komunikaci pomocí protokolů TCP a UDP k rozlišení aplikace v rámci počítače. Zdrojový port tak identifikuje aplikaci, která datagram odesílá, cílový port zase aplikaci, které je datagram určen. Sekvenční číslo udává pořadové číslo prvního bajtu dat. Potvrzovací číslo se uplatňuje pouze s příznakem ACK a je v něm uloženo pořadové číslo posledního přijatého bajtu. Položka offset udává velikost hlavičky.

Příznaky TCP paketu jsou následující:

- **URG** – urgent pointer
- **ACK** – potvrzení
- **PSH** – aplikační data
- **RST** – reset spojení
- **SYN** – synchronizace sekvenčních čísel
- **FIN** – oznámení, že odesílatel odeslal všechna data, chce uzavřít spojení

Velikost okna udává, kolik bajtů je odesílatel schopen v dané chvíli přijmout. Souvisí s velikostí a obsazením vstupního bufferu. Kontrolní součet je počítán z hlavičky protokolu a odesílaných dat.

Průběh komunikace

Pro navázání spojení používá TCP tzv. trojcestný handshake. Při něm pošle klient na server datagram s nastaveným příznakem SYN a náhodně zvoleným sekvenčním číslem. Server může buďto spojení odmítnout příznakem RST nebo spojení akceptuje a klientovi pošle svoje náhodně vygenerované sekvenční číslo spolu s příznaky SYN a ACK. Přiznakem ACK tak potvrdí příchod klientova datagramu. Klient následně potvrdí příchod datagramu ze serveru odesláním příznaku ACK.

V tuto chvíli je spojení ustanoveno a obě strany si mohou začít posílat data. Při odesílání datagramu obsahujícího aplikační data se zpravidla nastavuje příznak PSH, ale není to pravidlem. Každý odeslaný datagram nese svoje sekvenční číslo, které je pořadovým číslem prvního odesílaného bajtu. Protější strana pak potvrzuje přijetí příznakem ACK a potvrzovacím číslem, které udává pořadové číslo posledního přijatého bajtu. Příznaky ACK a PSH se mohou kombinovat, je tak možné data odeslat a zároveň potvrdit přijatá data.

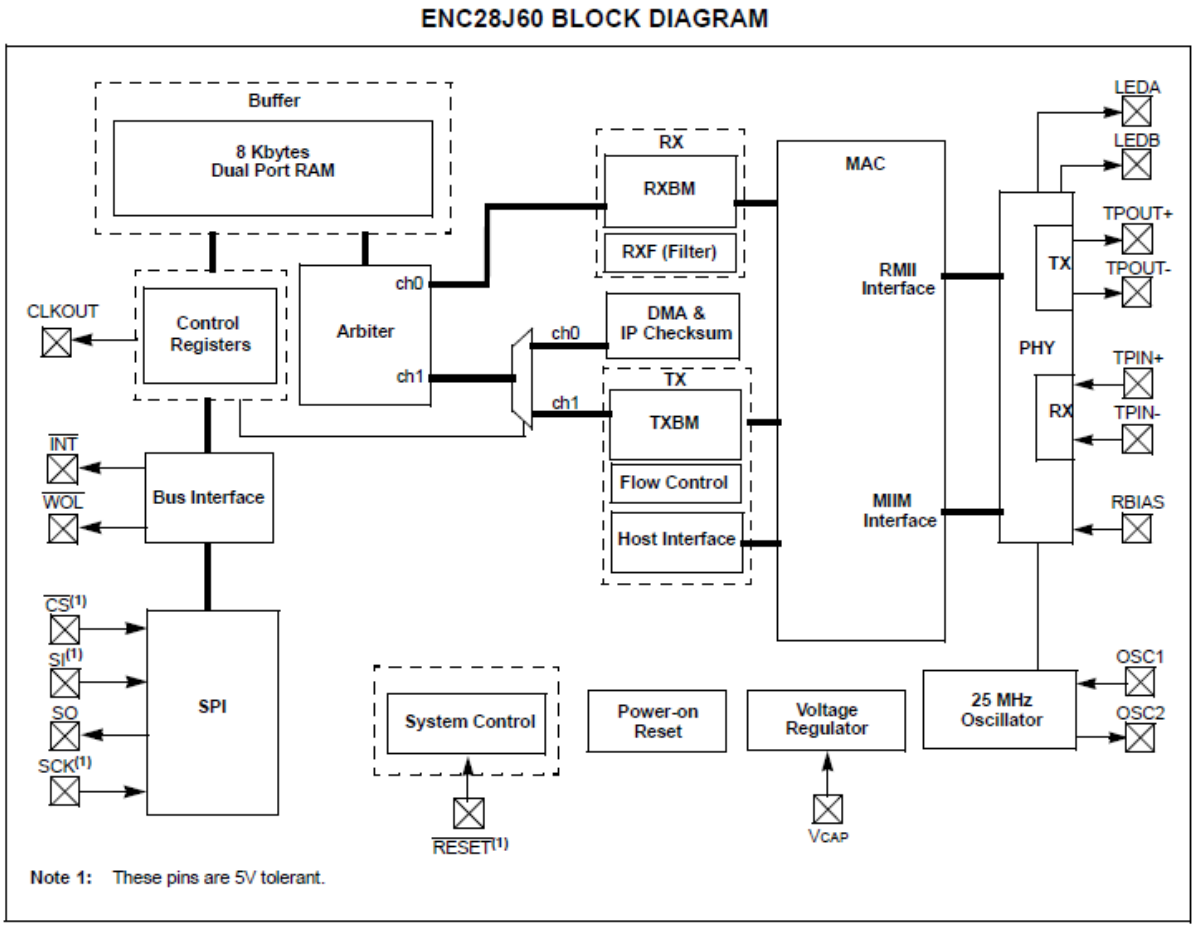
Pokud jedna ze stran již nemá žádná data k odeslání, oznámí to protější straně příznakem FIN a pokusí se spojení uzavřít. Druhá strana má na výběr ze dvou možností. Buď odešle příznak FIN a ACK a spojení uzavře také, nebo může přijetí příznaku FIN pouze potvrdit a pokračovat v odesílání, dokud neodešle všechna svá data. Strana uzavírající spojení je musí akceptovat a potvrzovat s příznaky ACK. Až se rozhodne ukončit spojení i druhá strana, odešle příznaky FIN a ACK. Strana, která uzavírala spojení jako první pak musí příznak FIN ACK potvrdit zasláním datagramu s příznakem ACK. Spojení ale může úplně ukončit teprve až po normou stanoveném timeoutu, protože není jisté, zda poslední potvrzení dorazilo a zda není nutné ho přeposlat. Naopak druhá strana po obdržení posledního potvrzení může spojení uzavřít okamžitě.

4. Ethernetový modul ENC28J60

ENC28J60 je samostatný ethernetový řadič se sériovým SPI rozhraním (zkratka anglického *Serial Peripheral Interface*). Je navržen tak, aby zprostředkoval Ethernetové rozhraní pro mikrokontroléry s rozhraním SPI. Informace uvedené v této kapitole byly čerpány z [5].

Modul splňuje standardy IEEE 802.3 pro komunikaci v síti internet. Dále zajišťuje filtrování, aby omezil počet příchozích paketů a tím odlehčil řídicímu mikrokontroléru. Obsahuje DMA modul pro zvýšení datové propustnosti a hadrware pro výpočet CRC paketů. Komunikační rozhraní s mikrokontrolérem tvoří dva piny určené pro přerušení a SPI s přenosovou rychlostí až 10Mb/s. Další dva piny jsou vyhrazeny pro LED a indikaci síťové aktivity.

Modul ENC28J60 je tvořen sedmi základními bloky (viz obrázek 4.1). Obsahuje SPI rozhraní, které slouží pro komunikaci s mikrokontrolérem. Další blok tvoří registry pro řízení a monitorování stavu modulu. Pro ukládání příchozích a odchozích paketů obsahuje modul buffer tvořený 8kB dual-port RAM paměti. Další součástí je arbitr, který řídí přístup do paměti při DMA přenosu. Bus Interface (rozhraní sběrnice) interpretuje příkazy přijaté přes SPI. MAC (Medium Access Control) implementuje MAC logiku podle standardu IEEE 802.3. Poslední částí je modul fyzické vrstvy, který kóduje a dekóduje analogová data pro přenos po kroucené dvojlince.



Obrázek 4.1. Blokový diagram modulu ENC28J60 [5]

Rozhraní SPI

SPI je sériové synchronní rozhraní určené primárně pro připojení periferií k mikrokontroleru. Dovoluje jak spojení dvou zařízení, tak i propojení více zařízení za podmínky, že komunikovat mohou současně vždy jen dvě zařízení. Rozhraní pracuje ve *full-duplex* režimu, to znamená, že v jednom okamžiku se mohou přenášet data oběma směry. Komunikace pomocí SPI vyžaduje dva datové vodiče (MISO, MOSI), jeden vodič pro hodinový signál (CLK) a alespoň jeden vodič pro výběr zařízení (CS), se kterým proběhne komunikace.

Každé připojené zařízení může být v jednom ze dvou režimů, buď *master* nebo *slave*. *Master* zařízení může být pouze jediné, ostatní jsou typu *slave*. *Master* generuje hodinový signál na sběrnici a signálem na vodič CS vybírá periferii v režimu *slave*, se kterou chce komunikovat.

Řídící registry

Řídící registry poskytují hlavní komunikační rozhraní mezi mikrokontrolerem (dále uC) a řídící logikou ethernetového modulu. Zápis dat do těchto registrů dovoluje uC řídit ethernetovou komunikaci, naopak čtení z některých registrů slouží k monitorování průběhu operací a stavů řadiče. Registry jsou rozděleny do čtyř bank po 32 registrech, adresují se 5-ti bitovou adresou. Posledních 5 adres každé banky je společných pro všechny banky. Tyto registry slouží k řízení celého modulu a tímto způsobem je vyřešen efektivní přístup bez nutnosti přepínat banku.

Registry jsou rozděleny do tří základních skupin. Skupina ETH jsou registry nastavující chování modulu jako celku, MAC jsou registry řídící MAC vrstvu (*Media Access Control*) a MII (*Media Independent Interface*) registry řídící rozhraní fyzické vrstvy.

Buffer

Buffer slouží k uložení příchozích a odchozích paketů. Má kapacitu 8KB a dá se nastavením registrů rozdělit na dvě libovolně velké části, kam se budou ukládat příchozí pakety a kam budeme ukládat pakety pro odeslání. Část vyhrazená pro příjem paketů se chová stejně jako kruhová fronta FIFO a mechanismy čtení jsou uspůsobené tak, aby se dal paket přečíst jednoduše jako stream. Je zde implementováno i několik automatických kontrolních mechanismů, které brání přepsat ještě nepřečtené části fronty nově příchozími pakety. Část pro odesílání paketů se naopak chová jako klasický paměťový blok a je zcela v režii mikrokontroléru.

Instrukční sada rozhraní SPI

Provoz ENC28J60 zcela závisí na příkazech od hostitelského mikrokontroleru přes rozhraní SPI. Tyto příkazy sestávají z instrukcí o jednom nebo více bajtech, které zpřístupňují kontrolní registry a vnitřní buffer (viz obrázek 4.2). Každá instrukce sestává ze tří bitů určujících operaci a 5-ti bitového argumentu. V závislosti na typu operace následují data.

SPI™ INSTRUCTION SET FOR THE ENC28J60

Instruction Name and Mnemonic	Byte 0								Byte 1 and Following							
	Opcode			Argument					Data							
Read Control Register (RCR)	0	0	0	a	a	a	a	a	N/A							
Read Buffer Memory (RBM)	0	0	1	1	1	0	1	0	N/A							
Write Control Register (WCR)	0	1	0	a	a	a	a	a	d	d	d	d	d	d	d	d
Write Buffer Memory (WBM)	0	1	1	1	1	0	1	0	d	d	d	d	d	d	d	d
Bit Field Set (BFS)	1	0	0	a	a	a	a	a	d	d	d	d	d	d	d	d
Bit Field Clear (BFC)	1	0	1	a	a	a	a	a	d	d	d	d	d	d	d	d
System Command (Soft Reset) (SC)	1	1	1	1	1	1	1	1	N/A							

Legend: a = control register address, d = data payload.

Obrázek 4.2. Instrukční sada SPI modulu ENC28J60 [5]

Komunikace a řízení probíhá pomocí sedmi instrukcí. Instrukce RCR přečte obsah kontrolního registru na adrese uložené v argumentu. Pokud se jedná od registr ze skupiny ETH, odpověď je odeslána bezprostředně po dokončení přenosu instrukce. Přístup k ostatním registrům je časově náročnější a tak se před samotnou odpovědí odešle tzv. *dummy-byte*. Nejde o nic jiného než o bezvýznamný bajt, který vyplní čekací dobu. Instrukcí RBM lze přečíst jeden bajt z paměti příchozích paketů z adresy určené hodnotou registru ERDPT. Přečtený bajt odešle zpět přes rozhraní SPI a adresa v registru ERDPT se inkrementuje. Aby nebylo nutné pro každý bajt tuto proceduru opakovat, stačí, aby mikrokontroler nechal pin CS na SPI rozhraní v log. 0 a modul bude odesílat sekvenčně následující bajty v paměti tak dlouho, dokud pin CS nenastavíme zpět do log. 1.

K zápisu do registru slouží instrukce WCR. Ta zapíše na adresu registru v argumentu hodnotu bajtu poslaného bezprostředně po instrukci. Instrukce WBM zapíše bajt zaslaný za instrukcí na adresu z registru EWRPT. Hodnota registru se po zápisu automaticky inkrementuje a tak můžeme stejnou instrukcí zapsat bajt na následující adresu. Můžeme také využít obdobného přístupu jako u instrukce RMB, kdy po zaslání instrukce můžeme sekvenčně zapisovat data do paměti dokud nenastavíme bit CS do log. 1.

Instrukcemi BFS a BFC je možné přímo nastavovat a mazat jednotlivé bity v registrech podle masky, kterou předáme argumentem instrukce. Pro restartování modulu a nastavení registrů na defaultní hodnoty slouží instrukce SC.

5. Návrh komunikační knihovny

Návrh knihovny pro modul ENC28J60 sestává z několika nezávisle pracujících částí, kterými se budeme v této kapitole postupně zabývat. První z nich je programové řešení komunikace mikrokontroléru rodiny MSP430 se samotným modulem. Tato část zahrnuje funkce realizující jednotlivé instrukce jimiž je možné modul obsluhovat. Druhá část bude pojednávat o správě paměťových prostředků a optimalizaci paměťové náročnosti. Mezi nejpalčivější problémy při návrhu celé knihovny patří právě nedostatek paměti RAM jak na straně mikrokontroléru, tak modulu. V poslední části se budeme zabývat návrhem modelu TCP/IP, jeho rozdělením na vrstvy realizující jednotlivé síťové protokoly a volbou optimálního rozhraní mezi těmito vrstvami.

5.1 Komunikace s modulem

Jak již bylo zmíněno v předchozí kapitole, komunikace s modulem probíhá přes rozhraní SPI. Modul je tak připojen přes samostatnou linku k MCU přes obvod FPGA, který vykonává pouze funkci propojky. Dva samostatné SPI kanály se nacházejí pouze na Fitkitech od verze 2.0, proto není možné Ethernetový modul připojit na verzích starších.

Návrh komunikačního rozhraní

Komunikace s modulem probíhá přes rozhraní SPI, na kterém je implementován jednoduchý komunikační protokol obsahující sedm instrukcí (viz Kapitola 5). Každá instrukce se skládá ze tří bitů identifikujících instrukci a pěti adresních bitů. Z počtu bitů adresy nám vyplývá, že maximální počet registrů je 2^5, což je 32. Registrů je ale podstatně více, proto jsou rozděleny do čtyř bank podle významu. V bance 0 se nachází registry pro práci s vnitřní pamětí, banka 1 obsahuje nastavení vstupních filtrů, banka 2 nastavení parametrů fyzické vrsvy modulu a banka 3 registry pro nastavení MAC adresy a diagnostiku modulu. Posledních pět registrů je společných pro každou banku, obsahují základní ovládání modulu a také slouží k výběru banky.

Mezi základní instrukce patří čtení a zápis do vnitřních registrů modulu (instrukce RCR a WCR). Zápis do registru se realizuje odesláním dvou bajtů bezprostředně za sebou přes rozhraní SPI. První bajt obsahuje kód instrukce a adresu registru, druhý pak nový obsah registru, jak lze zjistit z tabulky v kapitole 5. Čtení registrů probíhá obdobně, MCU odešle instrukci s adresou a okamžitě po odeslání začne modul odesílat obsah naadresovaného registru. Při čtení ale přichází na řadu problém s přístupovou dobou registrů. Pouze registry, jejichž jméno začíná na „E“, jsou přístupné okamžitě, u ostatních je před samotnými daty odeslán tzv. *Dummy byte*, vyplňovací bajt, který nenese žádné informace, pouze vyplní dobu čekání na obsah registru. Na základě adresy registru však nelze odlišit tyto dvě skupiny registrů, proto je volba režimu čtení výhradně na programátorovi.

Všechny registry jsou pouze osmibitové, avšak pro adresaci 8kB paměti je zapotřebí 16 bitů. To je řešeno spojením dvou registrů uvnitř modulu, z nichž ten s nižší adresou obsahuje nižší bajt a druhý vyšší bajt slova. Při zápisu do takového registru musí být dodržena jedna zásada, nižší bajt musí být zapsán první. Je tomu tak z hlediska synchronizace, nižší bajt se nepřepíše okamžitě, ale čeká se až na zápis ho vyššího bajtu, aby byla hodnota rozšířeného registru přepsána v jednom okamžiku. Z tohoto důvodu vznikly v knihovně pomocné funkce realizující snadné čtení a zápis 16-ti bitových registrů.

Další dvě instrukce (RBM a WBM) slouží pro práci s vnitřním bufferem modulu. Dovolují tak číst a zapisovat z adres paměti nastavených v kontrolních registrech. Pro usnadnění operací s pamětí existuje vnitřní mechanismus, který automaticky zvyšuje hodnoty adres v kontrolních registrech, je tak možné sekvenčně číst z paměti bez nutnosti měnit adresy před každým čtením. Tento mechanismus je v knihovně trvale zapnutý, výrazně snižuje režii přenosu a dále v kombinaci s možností posílat ne pouze jeden, ale i více bajtů paměti s jednou instrukcí dále výrazně snižuje režii a tak zefektivňuje přenos bloků dat mezi modulem a MCU. Využití blokových přenosů má své opodstatnění, SPI přenos je totiž úzkým hrdlem celého systému, a tak je každé snížení režie přenosu přínosem pro rychlejší práci s modulem.

Instrukce BFS (*Bit Field Set*) slouží k nastavení pouze určitých bitů v registru podle masky. Jedná se o operaci OR registru s maskou. Druhá instrukce BFC je opakem BFS, realizuje operaci NAND registru s maskou. Tyto instrukce jsou v knihovně také implementovány.

Poslední instrukcí je reset modulu (RST). Po zavolání fukce realizující reset jsou nastavení modulu resetována a pro správnou funkci je ho zapotřebí znovu inicializovat.

Inicializace modulu

Před použitím pro příjem a odesílání paketů musí být modul řádně inicializován. Jedná se o posloupnost příkazů, které musí být provedeny po každém restartu modulu. Inicializace se skládá ze dvou částí - určení uspořádání vnitřní paměti a nastavení vlastností síťové vrstvy.

Paměť má velikost 8kB a je možné ji libovolně rozdělit na část pro příjem a část pro odesílání rámců (volba velikosti obou částí bude rozebrána v kapitole 6.2). Zatímco organizace a způsob využití místa v odesílací části odpovídá potřebám uživatele, v přijímací části platí striktní formát ukládání rámců. Každý rámec začíná ukazatelem do vnitřní paměti, kde bude následovat další přijatý rámec, poté jsou uloženy čtyři bajty informací o vlastnostech rámce a nakonec následuje samotný rámec. Jednotlivé rámce se ukládají za sebe. Část pro příjem funguje jako kruhový buffer, tzn. když se rámec již nevejde na konec bufferu, začne se ukládat od začátku.

Nastavení síťové vrstvy se skládá z těchto kroků. Prvním krokem je nastavení MAC adresy modulu. Ta se zapisuje do vyhrazených registrů a za běhu modulu se nesmí měnit. Dalším důležitým nastavením je režim přenosu, který by měl být v knihovně implicitně nastaven na full duplex. Mezi další vlastnosti patří výpočet a kontrola CRC, *padding*, maximální délka a filtr příchozích rámců. Při volbě automatického výpočtu CRC modul za každý odeslaný rámec připojí vypočtené CRC. Tato volba by měla být zapnuta, protože vlastní výpočet CRC by byl zbytečný. Stejně jako *padding* (česky *výplň*), tedy doplnění rámce nulami na minimální délku 60 bajtů. Nastavení maximální délky příchozího rámce by se mělo shodovat s maximální délkou ethernetového rámce, aby zbytečně nedocházelo k zahazování velkých rámců.

5.2 Správa paměti

Mezi největší problémy při implementaci modelu TCP/IP patří vedle slabého výpočetního výkonu MCU zejména nedostatek operační paměti. Této skutečnosti bude při návrhu přihlíženo doslova na každém kroku. Také velká většina všech omezení bude právě na vrub nedostatku paměti RAM.

Buffery pro práci s pakety

Pro příjem paketů a následné zpracování dat je zapotřebí paket uložit do nějakého bufferu, kde s ním budeme moci pracovat. Logicky se nabízí možnost stáhnout celý příchozí paket z modulu do bufferu v RAM paměti MCU. Zde pak může dojít ke zpracování, přičemž hlavní výhodou by byl náhodný

přístup do dat v paketu a také nízká režie přenosu přes rozhraní SPI. Hlavním záporem této varianty je ale paměťová náročnost. Maximální velikost paketu v síti Ethernet je 1518B, což je necelá pětina z 8kB operační paměti MCU. Z toho důvodu při implementaci knihovny přistoupíme ke kompromisu. Paket je po přijetí uložen v paměti modulu, proto ho nemusíme zbytečně kopírovat do MCU a přístup k datům se bude odehrávat přímo přes SPI rozhraní. Data se tak dají číst obdobně jako z BSD socketů a aplikace si vytvoří čtecí buffer pouze o velikosti, která bude postačovat pro zpracování. Tímto krokem dojde k velké úspoře RAM za cenu o něco větší režie přenosu přes SPI rozhraní, protože místo souvislého čtení celého paketu budeme číst po blocích.

Obdobný postup uplatníme i s pakety pro odesílání. Data budou zapisována přímo do modulu, kde pro ně bude vyhrazen prostor v odesílací části paměti.

Volba velikosti a počtu odesílacích bufferů

Počet odesílacích bufferů bude přímo závislý na vlastnostech a parametrech transportních a nižších protokolů. Podle zadání musíme umožnit odesílání protokolů TCP, UDP, ICMP a ARP. Pro protokol UDP vyhradíme v paměti Ehternetového modulu odesílací buffer o velikosti 1514B, aby bylo možné odesílat i datagramy největší možné velikosti. Pro protokol TCP je nutné vytvořit samostatný buffer pro účely přeposílání nedoručených datagramů. V tomto bufferu pak datagram zůstane tak dlouho, dokud jeho přijetí nebude potvrzeno. S tímto přístupem přichází hlavně omezení na odesílání datagramů. Je tak možné odeslat pouze jeden datagram a další teprve až po doručení předchozího. Bylo by sice možné zajistit více bufferů a tím zvětšit velikost odesílacího okna, ale tento požadavek by zabral daleko více paměti, které máme již tak nedostatek. Proto se toto řešení jeví jako optimální.

Další dva protokoly, ARP a ICMP, budou potřebovat každý svůj vlastní buffer. ARP zejména proto, že slouží vyšším protokolům ke zjišťování MAC adresy. Pokud totiž není v ARP tabulce překlad IP na MAC adresu, musí se probíhající odesílání paketu pozastavit a odeslat ARP žádost. Teprve po přijetí ARP odpovědi může posílání pokračovat. Velikost ARP bufferu je pouze 42 bajtů, stejně jako maximální velikost ARP paketu. Buffer pro odchozí ICMP zprávy má velikost dostačující pro odpověď na zprávu ECHO odeslanou programem Ping, 114 bajtů.

5.3 Struktura navržené knihovny

Celou knihovnu je vhodné hned z počátku rozdělit na několik nezávislých bloků a navrhnout správné rozhraní mezi nimi. Nejvýhodnější rozdělení bude bezpochyby podle podle vrstev TCP/IP modelu, kdy každý soubor knihovny bude obsahovat jednu vrstvu modelu. Každá z vrstev pak bude obsahovat velice podobné rozhraní jak směrem nahoru v hierarchii (příjem) tak směrem dolů (odesílání).

Příjem paketů

Existují dvě možnosti, jak zjistit nově příchozí paket: buď pomocí přerušení z modulu nebo pomocí cyklického testování vnitřního registru. Obsluha přerušením má však jednu zásadní nevýhodu, zpracování celého paketu bude jistě časově náročné a to by v kombinaci s častým příchodem paketů mohlo vyústit v situaci, kdy se MCU bude starat pouze o příchozí pakety a nebude vykonávat další činnost. Tuto situaci by bylo dále nutné implementačně ošetřit, a proto bude jednodušší naimplementovat cyklickou kontrolu vnitřního registru. Získáme tím i výhodu, že budeme moci obsluhu příchozího paketu volat dle požadavku dané aplikace.

Úkolem obslužné funkce bude zjistit přítomnost nového paketu ve vstupním bufferu a zareagovat vhodnou obsluhou. Tím je myšleno volání funkce linkové vrstvy, která paket začne

zpracovávat a nakonec podle atributů hlavičky zavolá příslušnou obslužnou funkci vrstvy vyšší. Například při příchodu rámce obsahujícího IP paket zavolá vyšší síťovou vrstvu a při příchodu ARP dotazu zavolá obslužnou funkci protokolu ARP. Podobně bude pracovat i vrstva síťová a transportní pouze s tím rozdílem, že transportní vrstva je poslední a zde bude docházet k předávání dat aplikaci.

Na každé vrstvě bude docházet ke kontrole hlaviček protokolů. Pakety protokolů, které nebudou implementovány budou zahazovány, stejně jako pakety fragmentované. U příchozích paketů nebude prováděna ani kontrola kontrolních součtů, které se vyskytují v každém z protokolů, pouze bude samotným modulem prováděna automatické kontrola CRC příchozích rámců.

Odesílání paketů

Při odesílání paketů budeme postupovat přesně opačným směrem. Aplikace transportní vrstvě předá data s IP adresou, na kterou je chce odeslat, ta připojí hlavičku a opět zavolá obslužnou funkci vrstvy nižší. Až poslední z vrstev, linková, pouze připíše hlavičku a nastaví modul, aby odeslal data po síti.

Při odesílání paketů musíme do hlaviček protokolů uvádět i vlastnosti celého rámce, jako je délka nebo kontrolní součet. Za tímto účelem budou v knihovně implementovány příslušné funkce. Kontrolní součet bude vypočítáván automaticky jak u TCP, tak i u UDP datagramů. Náročnost na výpočetní prostředky bude minimální při využití DMA řadiče v modulu, který je schopen spočítat kontrolní součet za zlomek času, který by zabralo přenést zpět datagram přes rozhraní SPI a spočítat ho na straně mikrokontroléru.

Práce s buffery

Při zpracování příchozích paketů musí mít všechny vrstvy k paketu přístup, aby mohly zpracovávat hlavičky svých protokolů. Proto vytvoříme jednotné rozhraní, které bude provádět přístup k paketu uloženému ve vnitřní paměti modulu. Každá vrstva si na začátku zpracování z tohoto rozhraní načte pouze svoji hlavičku, zpracuje z ní údaje a zavolá vyšší vrstvu. Po přečtení hlaviček transportní vrstvou už v bufferu zbude pouze datová část datagramu, kterou bude moci aplikace zpracovat.

Při odesílání paketů použijeme obdobný přístup. Zde se ale situace trochu komplikuje, protože buffer již není jeden, ale čtyři. Proto naimplementujeme rozhraní, knihovnu, která bude mít za úkol správu těchto čtyř bufferů a bude implementovat jednotný přístup při práci s kterýmkoli z nich. Sama aplikace rozhraní inicializuje, zvolí, který z bufferů se použije, zapíše přes něj data, která chce poslat a předá buffer transportní vrstvě. Ta pak, stejně jako všechny nižší vrstvy, připíše svoji hlavičku a zavolá nižší vrstvu. Až poslední, linková vrstva provede odeslání paketu.

6. Způsob implementace knihovny

6.1 Modul pro obsluhu ENC28J60

Funkce zajišťující komunikaci s ethernetovým modulem jsou uloženy v souboru *enc28j60_spi*. Pro komunikaci skrze SPI bylo použito knihovny *fspi* z repozitáře Fitkitu, ve které jsou definovány funkce a makra pro obsluhu druhého kanálu SPI. V modulu knihovny jsou implementovány funkce jak pro čtení, zápis a bitové operace s registry, tak pro čtení a zápis znaků i bloků paměti do vnitřního bufferu. Navíc jsou implementovány funkce pro čtení i zápis do tzv. rozšířených 16b registrů. Dále se pro usnadnění práce s textovými protokoly v modulu nachází upravená funkce pro čtení řádků z bufferu (znakem konce řádku se rozumí znak LF).

Funkce pro přenos bloků neobsahují žádné kontrolní mechanismy, jako je například kontrola hranic přijatého rámce. Všechny kontroly rozsahů apod. musí být prováděny na vyšší vrstvě.

Inicializace ENC28J60

Pro inicializaci modulu ENC28J60 slouží funkce ENC28J60_init(). Funkce zapouzdřuje inicializaci SPI rozhraní, restart modulu, čekání na stabilizaci oscilátoru a nastavení řídících registrů.

Rozhraní SPI není vedeno přímo z MCU na piny Fitkitu, ale je vedeno přes obvod FPGA. Je tedy zapotřebí vytvořit a inicializovat propojení v jazyce VHDL. Po inicializaci SPI se provede softwarový restart modulu pomocí k tomu určné instrukce. Následuje rozdělení bufferu na části podle rozboru v kapitole 6.2. Ačkoli je velká část 8kB bufferu přidělena pro odesílací část a na příjem paketů zbylo necelých 5kB, praktická implementace a zkoušky v provozu nakonec ukázaly, že toto rozdělení nezpůsobuje zahazování příchozích paketů z důvodu nedostatku paměti.

Při inicializaci potřebujeme také nastavit MAC adresu modulu. Jako defaultní byla zvolena adresa 00:23:8B:6B:38:64. V případě, kdy potřebujeme adresu změnit je možné před samotnou inicializací provést změnu pomocí funkce *set_mac()*. Po inicializaci modulu se již MAC adresa měnit nedá.

Mezi další inicializační nastavení patří zejména zarovnávání rámců na 60 bajtů délky, automatický výpočet CRC pro každý rámec a vstupní filtr, který je nastaven na příjem broadcast a unicast rámců o délce maximálně 1518B. Modul je dále nastaven pro práci ve full-duplex módu.

Po nastavení všech důležitých registrů je povolen příjem paketů a modul může být používán.

6.2 Modul pro přístup k paměti

Buffer pro příjem paketů

Funkce pro práci se vstupním bufferem jsou obsaženy v souboru *enc28j60_spi_rx*. Patří mezi ně funkce pro čtení rámců, zjišťování vlastností rámce (délka, počet ještě nepřečtených bajtů z rámce atd.) a správu paměti (uvolňování místa). Názvy funkcí patřících ke knihovně mají prefix „rx“.

Při příchodu rámce paketu je vstupní buffer inicializován voláním funkce *rx_init()*. K volání dochází uvnitř modulu knihovny, proto ho není nutné inicializovat v aplikaci. Pokud se jedná o první inicializaci bufferu, tak se čtecí ukazatel nastaví na začátek paměti vyhrazení pro příjem paketů. Když ne, použije se ukazatel na další rámec získaný při předchozí inicializaci bufferu. Tento ukazatel se nachází vždy na začátku každého uloženého rámce a je nutné ho ukládat, abychom neztratili kontext

mezi přijatými rámci. Délka přijatého rámce bývá vždy uložena ve stavovém vektoru, který se nachází v bufferu za ukazatelem a má délku 4 bajty.

Funkce *rx_getc()*, *rx_read()* a *rx_readline()* implementované v modulu dovolují číst buffer po jednom bajtu, blocích i řádcích (pokud se jedná o text). Jsou navíc navrženy tak, že si samy hlídají meze rámce a nedovolí přečíst více bajtů, než kolik zbývá do konce rámce. Všechny tyto funkce mají jako návratovou hodnotu počet skutečně přečtených bajtů. Data se mohou přečíst, podobně jako u *BSD socketů*, pouze jednou. Náhodný přístup do bufferu není podporován. Pro čtení více než jednoho bajtu je doporučeno využívat funkce blokového přenosu dat, zejména kvůli výrazně nižší režii a tím pádem nižší době přenosu přes rozhraní SPI, která potažmo vede i k nižší odezvě aplikace při komunikaci po síti.

Dále je k dispozici pár praktických funkcí, jako je například test prázdnosti bufferu (*rx_isempty()*) nebo funkce vracející počet ještě nepřečtených bajtů (*rx_left()*).

Buffer pro odesílání paketů

Funkce pro práci s odesílacím bufferem jsou obsaženy v souboru *enc28j60_spi_tx*. Pro odesílání paketů jsou implementovány celkem čtyři různě velké buffery (tato volba je zdůvodněna v kapitole 6.2). Modul knihovny vytváří rozhraní, které síťovou aplikaci oddělí od skutečnosti, že pracuje se čtyřmi různými buffery. V modulu je proto implementována sada univerzálních funkcí, která s buffery pracuje jednotným způsobem. Názvy těchto funkcí mají prefix „tx“.

Inicializace bufferu je v případě odesílání datagramů na straně aplikace. Do parametru inicializační funkce *tx_init()* je třeba uvést, jakým protokolem chceme data odesílat. Funkce pak na základě uvedeného protokolu vybere a nainicializuje příslušný buffer. Vnitřní proměnné a všechny výpočty (volné místo, velikost bufferu atd.) ve funkcích jsou navrhnuty tak, aby počítaly s konstantami. Ty je pak možné za běhu měnit a tak jednoduše přepínat mezi buffery, aniž by došlo ke ztrátě dat nebo jiným vedlejším efektům. Proto je také možné mít pro zápis do čtyř i velikostně různých bufferů pouze jednu obecnou funkci.

Zápis do bufferů může probíhat dvěma způsoby, buď po jednom bajtu nebo blokově. Funkce *tx_putc()* pro zápis po bajtech a tx_write() po blocích jsou implementovány tak, že si samy kontrolují, aby nedošlo k přetečení bufferu. K dispozici je ještě makro *tx_write_str()*, které sice nakonec volá funkci *tx_write()*, ale předtím provede výpočet délky řetězce předaného parametrem.

Výstupní buffer nepodporuje náhodný přístup, pouze je možné vrátit funkcí *tx_rewind()* zapisovací ukazatel zpět na začátek a přepisovat buffer. To se v kombinaci s funkcí *tx_skip()* realizující přeskočení části bufferu hodí zejména pro dodatečné úpravy datagramu před odesíláním.

Před odesláním datagramu je nutné buffer řádně uzavřít a připravit k odeslání, k tomu účelu slouží funkce *tx_close()*. Ta označí aktuální pozici zapisovacího ukazatele jako konec bufferu. Data zapsaná později budou sice zapsána do bufferu, ale již nebudou odeslána.

6.3 Moduly komunikačních protokolů

Aplikační rozhraní

Systém obsluhy příchozích paketů je založen na tzv. *callback handlerech*. Ke zjednodušení definice a práce s handlery byla vytvořena makra *TCP_APP_HANDLER()*, *UDP_APP_HANDLER()* a *ICMP_APP_HANDLER()*. V nich je pak možné pomocí funkcí modulu *enc28j60_spi_rx* příchozí paket zpracovat. Handlery je nezbytné v programu zaregistrovat k obsluze příchozích paketů. Při volání jim jsou navíc v parametrech předány ukazatele na hlavičky protokolů. Všechny mají přístup k IP hlavičce, dále vždy k hlavičce svého protokolu. Tím je možné zjistit například IP adresu

odesílatele paketu a další užitečné informace. V těle handlerů je také možné pomocí modulu *enc28j60_spi_tx* vytvořit a zaslat odpověď na příchozí paket, ale není to podmínkou.

Handler protokolu TCP může být zaregistrován vždy jen jeden, a to z toho důvodu, že knihovna nepodporuje více paralelních TCP spojení. Tento handler je volán jen v případě příchodu paketu, který obsahuje aplikační data.

UDP handlerů může být v jeden čas zaregistrováno více. Knihovna umožňuje poslouchat zaráz až na šesti portech. Handler se pak váže k číslu portu a při příchodu paketu na tento port je zavolána obslužná funkce.

ICMP handler smí být zaregistrován pouze jeden. Zde již není implementováno žádné třídění příchozích zpráv. Zaregistrováním handleru jsou všechny automatické ICMP odpovědi zablokovány a veškeré ICMP zprávy jsou předávány aplikaci k obsluze.

Obsluha knihovny

Obslužná funkce *ENC28J60_idle()* zajišťující zpracování příchozích paketů se nachází v souboru *enc28j60_eth*. Funkce se dotazuje modulu, zda nebyl doručen nový paket a v případě že ano, vyvolá kroky k jeho zpracování a předání dat aplikaci. Voláním této funkce se zpracuje nejvýše jeden příchozí rámec. Je doporučeno ji volat v hlavní smyčce programu.

Protokol UDP

Funkce pro komunikaci pomocí protokolu UDP jsou obsaženy v souboru *enc28j60_udp*. Obslužné funkce dovolují odesílat UDP datagramy přímo z libovolného lokálního portu na libovolnou IP adresu a port. Naopak příjem datagramů může probíhat pouze na šesti zaregistrovaných portech, v případě využívání DHCP klienta knihovny pouze na pěti.

Pro odeslání UDP datagramu tak stačí pouze inicializovat tx-buffer, zapsat do něj data a uzavřít. Následným voláním funkce *udp_send()* se datagram odešle. Při odesílání datagramu se vždy vypočítává kontrolní součet datagramu, který je následně dopsán do hlavičky.

Registrace callback handleru k lokálnímu portu probíhá zavoláním funkce *udp_bind()*. Handler musí být definován makrem *UDP_APP_HANDLER()*. Ten bude poté volán s příchodem každého paketu na zaregistrovaný port. Registrace portu se dá také zrušit, a to prostřednictvím funkce *udp_unbind()*.

Protokol ICMP

Protokol ICMP není určen pro komunikaci, ale pouze k oznamování chybových stavů jako je například nedostupnost služby, nedosažitelnost počítače/směrovače. S tím je počítáno při implementaci aplikačního rozhraní. Funkce pro komunikaci pomocí protokolu ICMP jsou obsaženy v souboru *enc28j60_icmp*. Modul knihovny pro jednoduchost automaticky odpovídá pouze na zprávy typu „*ECHO*“. Všechny ostatní zprávy jsou ignorovány. Pokud ale aplikace z nějakého důvodu potřebuje odposlouchávat ICMP zprávy, může zaregistrovat handler funkcí *icmp_bind()*. Od okamžiku registrace budou všechny ICMP zprávy předávány k obsluze aplikaci a automatické odpovědi nebudou prováděny. ICMP modul neprovádí třídění zpráv podle typu a proto je možné registrovat pouze jeden handler, na který budou přeposílány zprávy všech typů. Funkcí *icmp_unbind()* se zruší registrace handleru a aktivuje se automatický režim odpovědi.

ICMP zprávy je možné také odesílat, stačí inicializovat, naplnit a uzavřít tx-buffer s daty, která chceme připojit k ICMP zprávě a zavolat funkci *icmp_send()*, která zprávu odešle.

Protokol TCP

Komunikace prostřednictvím protokolu TCP je na rozdíl od UDP stavová a tak potřebuje protokol k provozu pomocné mechanismy. Jedním z nich je socket. V tomto modulu se konkrétně jedná o strukturu, která reprezentuje TCP spojení a uchovává stavové informace. Základ obsluhy TCP tvoří stavový automat zajišťující přechod mezi jednotlivými fázemi spojení. Ten je doslova jádrem protokolu, protože zajišťuje agendu spojenou s ustavením a ukončením spojení, s odesíláním datagramů, jejich případným přeposíláním a také potvrzováním příchozích paketů. Ke správné funkci TCP spojení je zapotřebí cyklického volání časovače *tcp_timer()*, kterým se řídí veškeré timeouty a přeposílací mechanismy stavového automatu. Doba mezi dvěmi volání časovače určují dobu, po které se nepotvrzený paket znovu odešle, doporučeno je rozmezí 250 – 500ms.

Nové TCP spojení je třeba inicializovat, vytvořit socket. K tomu slouží funkce *create_tcp_socket()*, která zaregistruje handler TCP_APP_HANDLER k libovolnému portu. Modul protokolu TCP pro nedostatek vnitřní paměti Ethernetového modulu dovoluje pouze jedno TCP spojení, proto nelze vytvoři více socketů. V dalším kroku je možné se připojit voláním *tcp_connect()* ke vzdálenému serveru nebo funkcí *tcp_bind()* vytvořit TCP server.

Úspěšné navázání spojení se serverem nebo naopak připojení připojení klienta k místnímu serveru je možné testovat funkcí *tcp_connected()*. Jakmile tato funkce vrátí kladnou hodnotu, spojení je ustanoveno a je možné začít odesílat data. Nedílnou součástí odesílání je korektní práce s tx-bufferem. Paket je pak možné odeslat voláním funkce *tcp_send()*. Součástí odesílání je i výpočet kontrolního součtu z hlavičky a dat datagramu.

Po ustanovení spojení je možné spojení kdykoli aktivně ukončit voláním *tcp_disconnect()*. Po ukončení jednoho spojení je třeba čekat určitou dobu, než můžeme spojení pokládat za uzavřené. Znovupoužitelnost TCP socketu prověří volání *tcp_is_idle()*.

Socket je možné odstranit funkcí *delete_tcp_socket()* ovšem za předpokladu, že spojení je korektně ukončeno (*tcp_is_idle()*). Poté je možné vytvořit jiný socket, například na jiném portu.

Výše zmíněné funkce a mechanismy zajišťující chod TCP protokolu jsou obsaženy v souboru *enc28j60_tcp*.

Protokol DHCP

Protokol DHCP běží nad protokolem UDP a je využíván pro automatickou konfiguraci síťových nastavení. DHCP modul implementuje pouze základní funkcionalitu DHCP protokolu, a to získání síťové adresy, masky, adresy routeru a serveru DNS.

DHCP klient je inicializován voláním funkce *dhcp_init()*. Ta provede zaregistrování vlastního UDP handleru definovaného v knihovně, v němž je implementován stavový automat. Pro správnou funkci automatu je zapotřebí volat funkci *dhcp_timer()* v intervalu přibližně 500 – 1000ms. Tato doba bude i intervalem znovuposílání DHCP datagramů.

Pro kontrolu úspěšného získání DHCP nastavení slouží funkce *dhcp_ready()*. Jakmile vrátí funkce kladnou hodnotu, síťová adresa byla úspěšně nastavena a je možné začít využívat knihovnu pro komunikaci v síti Internet. Návratovou hodnotu je ale přesto nutné dále cyklicky testovat, protože například při odpojení Ethernetového kabelu nebo resetování routeru se provede automatický restart DHCP klienta a je nutné počkat na znovunačtení síťových nastavení.

Výše zmíněné funkce a mechanismy zajišťující chod DHCP protokolu jsou obsaženy v souboru *enc28j60_dhcp*.

6.4 Obsluha základních protokolů

Protokol ARP

Pro odeslání paketu je nezbytné kromě IP adresy cíle znát i MAC adresu jeho síťové karty. Právě překlad IP adres na MAC adresy zajišťuje ARP protokol. Ten spravuje také tabulku s mapováním. Tabulka má kapacitu šest položek. K vypršení platnosti záznamů nedochází automaticky, ale pokud je to třeba, je možné vyčistit ARP tabulku voláním *arp_table_clear()*. Pokud je tabulka plná, začnou se přepisovat nejstarší záznamy.

K odesílání ARP dotazů dochází automaticky bez zásahu aplikace a to tehdy, kdy neexistuje příslušný mapovací záznam v tabulce. O odesílání dotazů a správu tabulky se stará IP protokol, který služby ARP využívá.

Protokol IP

Při implementaci IP protokolu muselo dojít kvůli nedostatku paměťových prostředků k jednomu kompromisu: protokol nepodporuje fragmentaci paketů. Příchozí fragmentované pakety jsou tak automaticky zahazovány. Dále se zahazují i veškeré volitelné položky záhlaví.

IP protokol při přijetí nového paketu nejprve zkontroluje, zda je fragmentován a pokračuje odstraněním volitelných položek hlavičky. Zde také dochází k odstranění výplně (*paddingu*), kterým se doplňují příliš krátké rámce. Podle položky „protokol“ z hlavičky je následně zavolána obslužná funkce jednoho z vyšších protokolů.

Při odesílání paketu je kromě sestavení platné hlavičky nezbytné zjistit MAC adresu cíle, kterou vyžaduje nižší (linková) vrstva. Nejprve je dobré si uvědomit, že pokud IP adresa cíle neleží ve stejné podsíti, musíme tento rámec odeslat sice se stávající IP adresou, ale s MAC adresou brány. Teprve poté přichází na řadu vyhledávání v ARP tabulce. Pokud se jedná o první paket zaslaný na tuto IP adresu, v tabulce ještě není příslušné mapování IP na MAC, a tak se odesílání stávajícího paketu pozastaví a místo něj se odešle ARP dotaz. Na ARP odpověď se čeká pomocí smyčky s timeoutem, která prování volání obslužné funkce *ENC28J60_idle()*. Tím pádem můžeme přijmout ARP odpověď, získat z ní MAC adresu, a tak pokračovat v odesílání původního paketu. Pokud odpověď nedojde před vypršením timeoutu, paket je zahozen a selhání se oznámí aplikaci prostřednictvím návratové hodnoty.

7. Dosažené rychlosti komunikace

S implementovanou knihovnou bylo provedeno několik testů na rychlost zpracování paketů. Jedná se především o dobu odeslání různě velkých datagramů protokolů TCP a UDP, doba odezvy na ICMP zprávu ECHO a rychlost odpovědi na ARP dotaz. Nakonec proběhl test maximální propustnosti při komunikaci protokoly TCP a UDP. Modul byl připojen přímo k ethernetovému rozhraní počítače, na kterém byly časové údaje odečítány z programu Wireshark.

Nejprve proběhl test odezvy automatických odpovědí. Na ARP dotaz zaslaný na FITkit bylo odpovězeno v průměru za 1,3ms. Program Ping dostal odpovědi v průměru za 2,4ms.

Dále proběhl test odesílání různě velkých datagramů protokoly TCP a UDP. Jedná se jen o dobu odeslání datagramů, je s ní spojena režie inicializace bufferu, přenosu datagramu přes rozhraní SPI, sestavení hlaviček protokolů a odeslání přes Ethernetové rozhraní. Data byla přes rozhraní SPI přenášena po blocích 50B. Průměry naměřených časů jsou uvedeny v tabulce 7.1.

velikost datagramu	prázdný	730B	1460B
TCP	1,1ms	5,1ms	9,3ms
UDP	1,0ms	4,9ms	9,0ms

Tabulka 7.1, Průměrné doby odesílání datagramů

Z doby odeslání prázdného datagramu je patrné, že režie knihovny, která vzniká sestavováním a zápisem hlaviček, se pohybuje okolo 1ms. Tato režie je pro různě velké pakety téměř stejná. Jediným časovým rozdílem mezi zpracováním prázdného a maximálního datagramu je doba výpočtu kontrolního součtu, která se díky využití vnitřních mechanismů modulu pohybuje pod hranicí 0,1ms i pro největší možný datagram. Z porovnání s dalšími sloupci lze tak vyvodit závěr, že největší vliv na dobu odeslání datagramů má vlastní přenos dat přes rozhraní SPI.

Pro test propustnosti síťového rozhraní FITkitu byly vytvořeny speciální testovací aplikace jak pro FITkit, tak pro počítač. Při testu příjmu dat generovala počítačová aplikace data, která v zápětí odesílala na FITkit. Data byla aplikací na FITkitu pro jednoduchost zpracována pouze přečtením z bufferu. Pro test odesílání byl použit obdobný přístup. Aplikace běžící na FITkitu generovala datagramy, které přijímala aplikace na počítači. Čtení i zápis dat do bufferů byl prováděn po blocích 50B a velikost všech datagramů byla 550B. Dosažené rychlosti platí pouze pro jednosměrný přenos dat. Průměry naměřených rychlostí jsou uvedeny v tabulce 7.2.

	rychlost příjmu dat	rychlost odesílání dat
TCP	150kB/s	100kB/s
UDP	165kB/s	155kB/s

Tabulka 7.2, Průměrné rychlosti přenosu dat

Maximální rychlost komunikace pomocí protokolu TCP je znatelně nižší oproti UDP. Je to dáno hlavně tím, že s protokolem TCP je spjata větší režie přenosu v podobě potvrzování přijatých paketů.

8. Závěr

Cílem práce bylo implementovat knihovnu pro modul ENC28J60, která by umožnila FITkit připojit do sítě Internet. Podle zvoleného návrhu byla knihovna úspěšně implementována a testována. Byl tak vytvořen základ pro jednoduchou tvorbu síťových aplikací, které mohou pro komunikaci využívat protokolů TCP a UDP. Dále je možné zpracovávat a odesílat zprávy protokolu ICMP. Součástí práce je i několik vzorových aplikací.

Knihovna byla implementována s ohledem na omezené zdroje, zejména nedostatek paměti RAM. To s sebou přináší několik omezení komunikace. Protokol UDP umožňuje odesílat datagramy o velikosti až 1472B bez omezení, ale příjem je možný pouze na šesti libovolných portech. Protokol TCP pro svou složitost a náročnost dovoluje pouze jedno spojení. Příjem paketů není omezen, ale odesílat lze vždy pouze jeden paket o maximální velikosti 1460B, další je možno odeslat až po doručení předchozího. Zpracování příchozích ICMP zpráv není omezeno, ale protože ICMP není komunikačním protokolem tak odesílat lze pouze krátké zprávy, maximálně 72B.

Knihovnu by v budoucnu bylo možné rozšířit například o podporu protokolu IPv6, na který pomalu přechází celý Internet nebo o implementaci vyšších protokolů, například DNS. Při použití Ethernetového modulu s větším vnitřním bufferem by bylo možné odstranit některé nevýhody současné implementace, například zvýšit počet souběžných TCP spojení.

Literatura

[1] Dostálek, L.: *Velký průvodce protokoly TCP/IP a systémem DNS*. Computer Press, 2002

[2] Fairhurst, G.: Internet Communications Engineering [online]. Dostupný z WWW: <http://www.erg.abdn.ac.uk/users/gorry/eg3567/index.html>

[3] Odom, W.: *Počítačové sítě*. Computer Press , 2005

[4] Orebaugh, A., Ramirez, G., Burke, J.: *Wireshark a Ethereal*. Computer Press, 2008

[5] ENC28J60 datasheet [online]. Dostupný z WWW: <http://ww1.microchip.com/downloads/en/DeviceDoc/39662c.pdf>

[6] Platforma Fitkit – schéma zapojení [online]. Dostupný z WWW: <http://merlin.fit.vutbr.cz/FITkit/download/modul_eth_doc.pdf>

[7] Modul Ethernet – schéma zapojení [online]. Dostupný z WWW: <http://merlin.fit.vutbr.cz/FITkit/download/modul_eth_doc.pdf>

Seznam příloh

Příloha 1

Příklady použití knihovny

Získání adresy z DHCP serveru

Příklad demonstruje využití knihovny ENC28J60 pro získání IP adresy z DHCP serveru.

```
#include <enc28j60/ip_stack.h>

int main(void){

        ENC28J60_init();                        //inicializace modulu ENC28J60
        dhcp_init();                            //inicializace protokolu DHCP

        while (1){
                ENC28J60_idle();                //obslužná funkce knihovny

                if(dhcp_ready()){
                        //.....                 //adresa byla získána z DHCP serveru
                }
        }
}

//...
//přerušení jednou za 500ms
        dhcp_timer();                           //volani dhcp časovače
//...
```

Nastavení statické IP adresy

```
        //...
        set_ip(192,168,1,1);                    //nastavení lokální IP adresy
        set_netmask(255,255,255,0);             //nastavení síťové masky
        set_gateway(192,168,1,254);             //nastavení IP adresy brány
        //...
```

Odesílání a příjem UDP datagramů

Příklad demonstruje využití knihovny ENC28J60 pro komunikaci protokolem UDP.

```
#include <enc28j60/ip_stack.h>

UDP_APP_HANDLER(app_handler){
        //zpracování příchozích datagramů
}

int main(void){
        ENC28J60_init();                        //inicializace modulu ENC28J60
        //nastavení IP adresy
        udp_bind(1234,app_handler);             //příjem UDP datagramů z portu 1234

        while (1){
                ENC28J60_idle();                //obslužná funkce knihovny
        }
}
//odeslání UDP datagramu
if( tx_init(UDP_PROTO) == 0){                   // inicializaci bufferu je nutné testovat
        tx_write_str(„hello world“);
        tx_write(„\r\n“,2);
        tx_close();
        udp_send(cilovy_port, lokalni_port, cilova_ip_adresa);
}
```

Zpracování a odesílání ICMP zpráv

Příklad demonstruje využití knihovny ENC28J60 pro zpracování ICMP zpráv (s DHCP).

```
#include <enc28j60/ip_stack.h>

ICMP_APP_HANDLER(app_handler){
        // zpracování příchozích ICMP zpráv
}

int main(void){
        ENC28J60_init();                        //inicializace modulu ENC28J60
        //nastavení IP adresy
        icmp_bind();                            //manuální příjem ICMP zpráv

        while (1){
                ENC28J60_idle();                //obslužná funkce knihovny
        }
}
//odeslání ICMP datagramu
if( tx_init(ICMP_PROTO) == 0){                  // inicializaci bufferu je nutné testovat
        tx_write(.....);
        tx_close();
        icmp_send(cilova_ip, typ_zpravy, kod_zpravy, ID_datagramu, sekvencni_cislo_datagramu);
}
```

TCP klient

Příklad demonstruje využití knihovny ENC28J60 pro vytvoření TCP klienta (s DHCP nastavením).

```
#include <enc28j60/ip_stack.h>

TCP_APP_HANDLER(app_handler){
        // zpracování příchozích datagramů
}

int main(void){
        //...
        //nastavení časovače na interval 500ms
        ENC28J60_init();                                //inicializace modulu ENC28J60
        dhcp_init();                                    //inicializace protokolu DHCP
        create_tcp_socket(1234,app_handler);            //vytvoření socketu na portu 1234

        while (1){
                ENC28J60_idle();                        //obslužná funkce knihovny

                if(dhcp_ready()){                       //adresa byla získána z DHCP serveru
                        //...
                        tcp_connect(dest_ip,dest_port); //připojení k serveru
                        //...
                }
        }
}

//přerušení jednou za 500ms
//...
        dhcp_timer();           //voláni DHCP časovače
        tcp_timer();            //volání TCP časovače
//...

//odeslání dat TCP protokolem
//...
if( tx_init(TCP_PROTO) == 0){                           // inicializaci bufferu je nutné testovat
        tx_write_str(„hello world“);
        tx_write(„\r\n“,2);
        tx_close();
        tcp_send();
}
//...
```

TCP server

Příklad demonstruje využití knihovny ENC28J60 pro vytvoření TCP serveru (s pevnou IP adresou).

```
#include <enc28j60/ip_stack.h>

TCP_APP_HANDLER(app_handler){
        // zpracování příchozích datagramů
}

int main(void){
        ENC28J60_init();                        //inicializace modulu ENC28J60
        set_ip(192,168,1,1);                    //nastavení IP adres a masky
        set_netmask(255,255,255,0);
        set_gateway(192,168,1,254);
        create_tcp_socket(1234,app_handler);    //vytvoření socketu na portu 1234
        tcp_bind();                             //vytvoří server, poslouchání na portu socketu

        while (1){
                ENC28J60_idle();                //obslužná funkce knihovny
        }
}

//přerušení jednou za 500ms
        tcp_timer();                            //volání TCP časovače

//odeslání dat TCP protokolem
//...
if( tx_init(TCP_PROTO) == 0){                   // inicializaci bufferu je nutné testovat
        tx_write_str(„hello world“);
        tx_write(„\r\n“,2);
        tx_close();
        tcp_send();
}
//...
```

Jednoduchý Telnet klient

Na následujícím příkladě si demonstrujeme použití protokolu TCP pro vytvoření Telnet klienta.

```
#include <enc28j60/ip_stack.h>

TCP_APP_HANDLER(app_handler){
        char buffer[50];
        while(rx_left() > 0)                                    //přečtení celého paketu
                rx_getline(&buffer,50);
                //zpracování příkazu
}

//funkce pro obsluhu terminálu
        //příkaz CONNECT
                create_tcp_socket(53875,app_handler);           //vytvoření socketu
                tcp_connect(dest_ip,dest_port);                 //připojení k serveru

        //příkaz DISCONNECT
                tcp_disconnect();                               //aktivní ukončení spojení

        //odeslání příkazu
                if(tcp_connected()){                            //test, zda je možné odesílat data
                        tx_init(TCP_PROTO);                     //inicializace odesílacího bufferu
                        tx_write_str(prikaz);                   //zápis příkazu z terminálu
                        tx_close();                             //uzavření bufferu
                        tcp_send();                             //odeslání datagramu
                }
//konec obsluhy

int main(void){
        ENC28J60_init();                                        //inicializace ENC28J60
        dhcp_init();                                            //inicializace DHCP

        while (1){
                ENC28J60_idle();                                // obsluha modulu
        }
return 0;
}

//přerušení jednou za 500ms
        tcp_timer();                                            //volání TCP časovače
        dhcp_timer();                                           //volání DHCP časovače
```