ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA ELEKTROTECHNICKÁ

KATEDRA APLIKOVANÉ
ELEKTRONIKY A TELEKOMUNIKACÍ

DIPLOMOVÁ PRÁCE

Ethernetové rozhraní embedded systémů

Vedoucí práce **Ing. Jiří Basl, Ph. D.**
Autor práce **Bc. Pavel Fikar**

2012

ABSTRACT

Content of diploma thesis contains characteristics of an ethernet interface of embedded devices. The thesis describes the reference model ISO/OSI, it defines the differences between the classical ethernet and the implementation of ethernet in embedded devices. One part of the thesis consists of sample application, which has been built on the use of evaluation kit Stellaris of Texas Instruments company. The heart of this evaluation kit is formed by microcontroller with Cortex™ M3 core. In the sample applicatin, LwIP TCP/IP stack was implemented. One chapter is dedicated to this stack as well. The application is supposed to function as Ethernet-RS232 converter and embedded webserver.

KEYWORDS

Embedded Ehternet, Texas Instruments, Stellaris, LwIP TCP/IP stack

ANOTACE

Obsahem diplomové práce je charakteristika ethernetového rozhraní pro embedded systémy. Práce obsahuje popis referenčního modelu ISO/OSI, definuje rozdíly mezi klasickým ethernetem a implementacích ethernetu v embedded zařízeních. Součástí práce je ukázková aplikace postavená na využití vývojového kitu Stellaris od společnosti Texas Instruments, jehož srdce tvoří mikrokontrolér s jádrem Cortex™ M3. V ukázkové aplikaci byl implementován LwIP TCP/IP stack, kterému je věnována v této práci samostatná kapitola. Aplikace plní funkci Ethernet-RS232 převodníku a embedded webserveru.

KLÍČOVÁ SLOVA

Embedded Ethernet, Texas Instruments, Stellaris, LwIP TCP/IP stack

PROHLÁŠENÍ

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

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně, s použitím odborné literatury a pramenů uvedených v seznamu, který je součástí této diplomové práce.

Dále prohlašuji, že veškerý software, použitý při řešení této bakalářské práce, je legální.

V Plzni dne 2. května 2012 Pavel Fikar

PODĚKOVÁNÍ

Tímto bych rád poděkoval vedoucímu diplomové práce Ing. Jiřímu Baslovi, *Ph. D.* za cenné profesionální rady, připomínky a metodické vedení práce.

Seznam ilustrací

Seznam použitých pojmů, symbolů a zkratek

Ethernet - širokopásmový komunikační systém určený k přenosu digitálních datových rámců přes lokální síť

Datagram – datový rámec přenášený síťovou či transportní vrstvou, jehož přenos není opatřen bezpečnostními mechanismy definovanými v případě paketových přenosů [10]

Paket – datový rámec přenášený transportní vrstvou, jehož přenos je opatřen bezpečnostními mechanismy, které jsou schopné detekovat chybný přenos a v tomto případě přenos opakovat

Protokol – definovaná sada pravidel

Protocol Stack – sada protokolů

Soket – dvojice tvořená IP adresou a číslem portu, někdy také označovaná jako koncový bod nebo endpoint

NVIC – Nested Vectored Interrupt Controller, vestavěný vektorový řadič přerušení [16]

LwIP – LightWeight Internet Protocol stack, konkrétní implementace TCP/IP stacku

OSI – Open System Interconnection, propojení systémů nezávislé na jakémkolik firemním řešení

IP – Internet Protocol, protokol síťové vrstvy

ICMP – Internet Control Message Protocol, protokol síťové vrstvy

TCP – Transmission Control Protocol, protokol transportní vrstvy

UDP – User Datagram Protocol, protokol transportní vrstvy

DHCP – Dynam Host Configuration Protocol, protokol aplikační vrstvy

HTTP – HyperText Transfer Protocol, protokol aplikační vrstvy

Pbuf – vniřní reprezentace datagramu či paketu uvnitř LwIP stacku

CCS – Code Composer Studio, vývojové prostředí

Css – Cascade Style Sheet, soubor obsahující pravidla definující vzhled html dokumentu

SSI – Server Side Include, schopnost webserveru generovat dynamická data

CGI – Common Gateway Interface, schopnost webserveru zpracovat data z formulářů

Obsah

1 Úvod

V posledních letech zájem o propojení počítačů a inteligentních elektronických zařízení značně vzrůstá. Ceny těchto technologií neustále klesají díky stále pokročilejším výrobním technologiím a tak se ethernetové komunikační technologie postupně prolínají do všech možných oblastí ať už se jde o drátovou či bezdrátovou technologii. Ethernet se tak v obou těchto podobách postupně dostal do oblastí jako je zdravotní péče, řídící technika, sledovací zařízení, komunikační technika, automobilový průmysl, využití v senzorových sítích etc.

Přestože internetové technologie se v časové linii neustále vyvíjí, prokázal ethernet svojí flexibilitu v neustále se měnícím a vyvíjejícím světě sítí po posledních několik desetiletí. Jednoduše řečeno, co se týče spolehlivosti a rychlosti přenosu dat, ethernet stále drží krok s dobou a náhrada této technologie za jinou technologii je v příštích několika letech nepředstavitelná.

Cílem diplomové práce je prozkoumat možnosti využití ethernetu v embedded systémech, prozkoumat jeho slabé a silné stránky, uvést možnosti implementace ehternetu v embedded systémech, zdůraznit rozdíly implementace ethernetu v embedded zařízeních od implementací v osobních počítačích a v neposlední řadě vytvořit ukázkovou aplikaci, která by názorně možnosti ethernetu v embedded systémech nastínila pro snadnější představu.

2 Ethernet

Ethernet je širokopásmový komunikační systém určený k přenosu digitálních datových rámců přes lokální síť. Éterem (ether) je v tomto případě sdílený komunikační prostředek, tedy pasivní přenosové médium nepodléhající žádnému centrálnímu řízení. Přístup k éteru je rozložen do sítě (net) příslušných stanic a je řízen konkrétním modelem arbitráže. Už tedy víme, odkud se název ethernetu vynořil a zjistili jsme, že jeho vysvětlení je kupodivu velmi jednoduché. Ve chvíli, kdy se začneme zabývat samotnou funkcí ethernetu a mechanismy jimiž disponuje, složitost značně narůstá. Z tohoto důvodu, mluvíme-li o ethernetu, rozdělujeme ho dle referenčních modelů na jednotlivé vrstvy. Nejpoužívanějším referenčním modelem v dnešní době je pak referenční model ISO/OSI popsaný v následující kapitole.

2.1 Model ISO/OSI

Základem chápání ethernetu je jeho rozdělení do jednotlivých vrstev. K tomu slouží referenční model ISO/OSI. Jedná se o referenční model otevřené komunikace, odtud potom zkratka OSI znamenající Open Systems Interconnection. Popis referenčního modelu tvoří nepatrnou část diplomové práce. Je však důležité si uvědomit, že pro embedded ethernet platí stejná pravidla komunikace jako pro ethernet implementovaný v osobních počítačích. Jelikož samotný referenční model je velmi rozsáhlý a vydal by za sepsání několika samostatných svazků, uvádím v této práci pouze nezbytný přehled a detailněji se zaměřuji pouze na protokoly implementované v ukázkové aplikaci.

2.1.1 Vrstvy referenčního modelu ISO/OSI

1. Fyzická vrstva
2. Spojová vrstva
3. Síťová vrstva
4. Transportní vrstva
5. Relační vrstva
6. Prezentační vrstva
7. Aplikační vrstva

Díky rozdělení ethernetu do jednotlivých vrstev je možné stanovit podmínky, za kterých je možné spolehlivě komunikovat mezi sebou po sériové sběrnici. Model je tvořen sedmi vrstvami, z nichž každá plní svoji předem definovanou funkci a službu. Nazváním referenčního modelu otevřeným se rozumí, že tento model není závislý na jakémkoli firemním řešení [3].

Referenčním modelem je vysvětleno jak se odesílaná zpráva postupně po vrstvách zpracovává u odesílatele a jak se následně přijímaná zpráva zpracovává u příjemce. K samotnému přenosu informace dochází přes fyzický spoj mezi každýma dvěma účastníky přenosu. Účastníci komunikace, kteří spolu komunikují například na aplikační vrstvě pak nemají žádné informace o funkci nižších vrstev.

Každá z vrstev má definována pravidla, která řídí komunikaci mezi účastníky. Definuje jak je možné zahájit, provést a ukončit přenos. Dosah každé vrstvy je vždy omezen

na jednu nižší vrstvu a jednu vyšší vrstvu (pokud existují). Každá vrstva poskytuje své funkce nejbližší vyšší vrstvě přes softwarové rozhraní realizované jako komunikační protokol. Ten lze chápat jako soubor pravidel pro komunikaci. Pokud tedy účastník komunikace chce odeslat data v podobě textu, obrázku či jiné informace, je nutné tato data upravit, přidat nějaký druh zabezpečení, opatřit adresami, popřípadě dalšími informacemi, data zakódovat a modulovat. Vedle fyzické vrstvy se na tom podílejí právě protokoly vyšších vrstev, například IP a TCP (Internet Protocol, Transmission Control Protocol). Obecné označení konkrétní softwarové implementace těchto protokolů se pak označuje často používaným termínem „protocol stack."

2.1.2 Fyzická vrstva

Fyzická vrstva je základní vrstvou referenčního modelu ISO/OSI. Fyzická vrstva je tvořena logickou sběrnicí, po které jsou datové pakety přenášeny směrem ke všem účastníkům komunikace. Datové pakety jsou však určeny pouze těm, jejichž adresa je uvedena v adresovém poli přenášeného rámce.

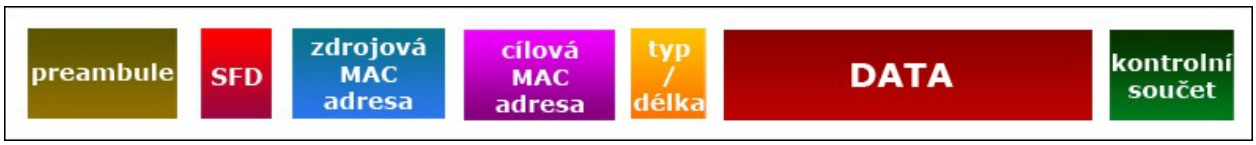
Fyzická vrstva definuje rozložení pinů, použité konektory, napěťové úrovně, vlastnosti a specifikace kabelů, elektrické vlastnosti přenosového média i jeho mechanické vlastnosti [3].

2.1.3 Spojová vrstva

Jelikož síť je obecně využívána mnoha zařízeními od různých výrobců, je každé ethernetové rozhraní těchto zařízení označeno unikátní adresou nazývanou MAC, často označovanou také jako fyzická adresa. Spojová vrstva zajišťuje přenos dat v rámci jedné lokální sítě právě pomocí fyzických adres zařízení.

Jednotlivé bity přenášeného rámce se přenášejí po bitech po fyzickém médiu, samotnému přenosu informačních bitů pak předchází startovací posloupnost. Startovací posloupnost, označovaná také jako preambule, slouží k synchronizaci vysílací stanice a všech přijímacích stanic. Datový rámec obsahuje adresu příjemce, odesílatele, typ zprávy, samotná data a kontrolní součet. Adresy odesílatele a příjemce jsou definovány podvrstvou MAC (Medium Access Control) protokolu spojové vrstvy. Typ zprávy je určen k rozeznání protokolu vyšších vrstev. Kontrolní součet slouží ke kontrole správnosti příjemcem přijatých dat. V případě detekce chyby v přijímaném rámci je tento rámec zahozen bez ohlášení vysílací stanici.

Struktura etherenetového rámce je naznačena v následujícím obrázku.



Ilustrace 1: Struktura ethernetového rámce

- **Preambule** – sekvence střídajících se jedniček a nul potřebná k nastavení bitové synchronizace.
- **SFD (Start Frame Delimiter)** – byte 10101011 určený k rozpoznání začátku rámce.
- **Zdrojová MAC adresa** – 48 bitová fyzická adresa ethernetového rozhraní vysílací stanice.

- **Cílová MAC adresa** – 48 bitová fyzická adresa ethernetového rozhraní přijímací stanice.
- **Typ/Délka** - Typ rámce přenášených dat (specifikace protokolu vyšší vrstvy) nebo počet bytů dat
- **DATA** – 0 až 1500 bytů dat.
- **Padding** – minimální délka rámce je 64 bytů, pokud je rámec kratší je v poli padding doplněn potřebným počtem bitů.
- **Kontrolní součet** – 32 bitové CRC.

Spojová vrstva dále definuje přístupovou metodu ke sdílenému přenosovému médiu. Jelikož je přenosové médium sdíleno několika stanicemi, které mohou ve stejnou chvíli začít vysílat, je třeba definovat pravidla přístupu k tomuto sdílenému médiu. Nejznámější přístupovou metodou je CSMA/CD (Carrier Sense Multiple Access/Collision Detection). Každý z účastníků komunikace má v tomto případě stejné právo využít sdílené přenosové médium v jakémkoli okamžiku, kdy je médium nevyužito. Pokud se však dvě zařízení rozhodnou odeslat data ve stejný okamžik dojde ke kolizi. Pokud stanice detekují kolizi, vyšle signál JAM, kterým ohlásí i ostatním stanicím, že došlo ke kolizi a po náhodném čase vysílání opakuje, pokud je sdílené médium volné k použití.

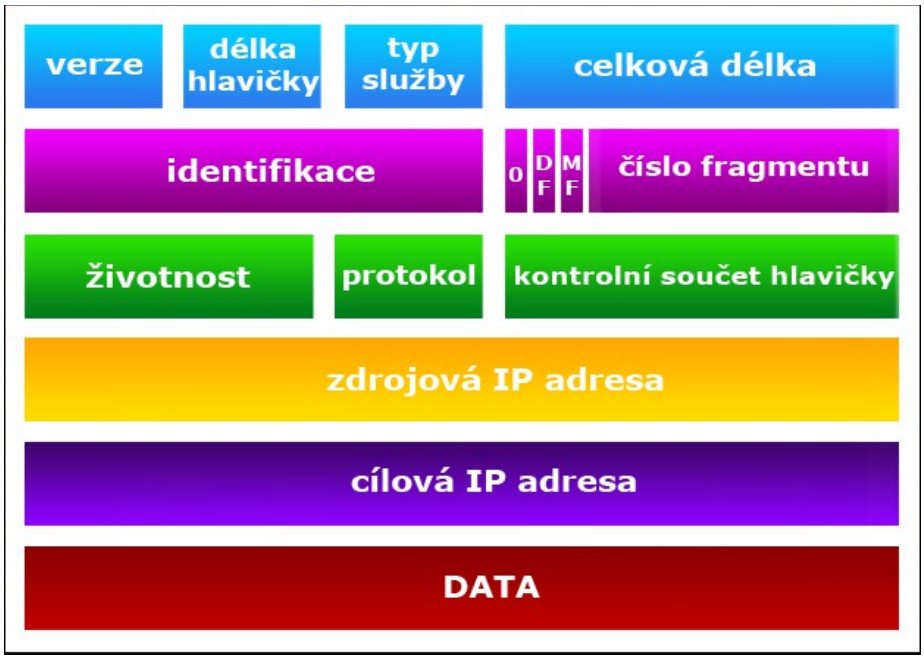
Modernější varianty ethernetu však od sdíleného média a tedy od přístupové metody CSMA/CD ustupují a využívají přepínače s plně duplexním režimem provozu [3].

2.1.4 Síťová vrstva

Úkolem síťové vrstvy je zajistit především síťovou adresaci, směrování a předávání datagramů. Síťová vrstva je pak schopna přenášet data v jedné síti nebo mezi více sítěmi. Jednotlivé sítě jsou rozděleny do několika podsítí a spojené s jinými sítěmi za účelem širokého pokrytí. Předávání datagramů mezi jednotlivými sítěmi je obstaráno systémem bran a směrovačů. Na síťové vrstvě pracuje množství protokolů. Nejdůležitější z nich jsou popsány níže.

2.1.4.1 IP protokol

Síťová vrstva je nejčastěji reprezentována protokolem IP. IP protokol pracuje s IP adresami, které se skládají jak z adresy sítě tak poté z konkrétní adresy stanice v síti. IP adresa pak musí být v síti jedinečná. IP protokol se dnes objevuje ve verzi 4 a 6, označované IPv4 a IPv6. Struktura datagramu IPv4 je naznačena v následujícím obrázku.

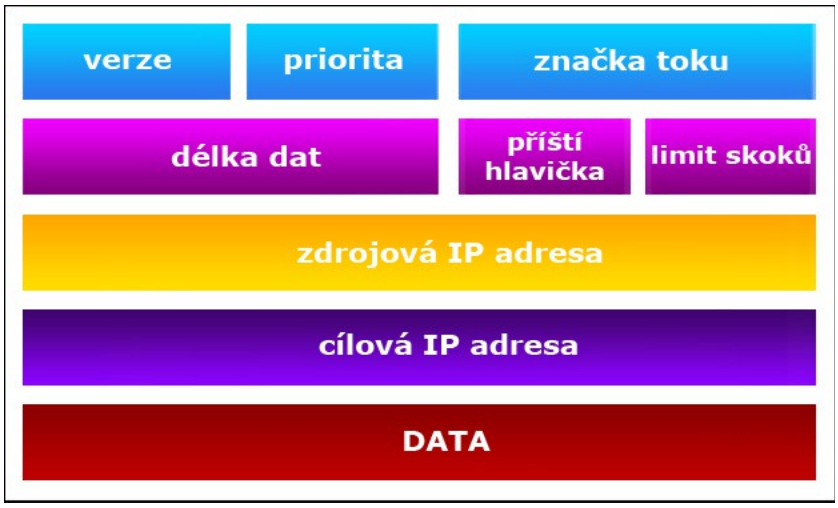


Ilustrace 2: Struktura datagramu protokolu IPv4

- **Verze** – 4 bity označující formát hlavičky.
- **Délka hlavičky** – 4 bity udávající délku hlavičky.
- **Typ služby** – 1 byte – umožňuje definovat prioritu.
- **Celková délka** – 2 byty – celková délka datagramu.
- **Identifikace** – 2 byty – identifikátor zadán odesílatelem k umožnění opětovnému složení fragmentované zprávy.
- **0** – 1 bit rezervován – musí být nula.
- **DF (Don't fragment)** – 1 bit, je-li nastaven na 1 pak není zpráva fragmentována.
- **MF (More fragments)** – 1 bit, je-li nastaven na 0 pak se jedná o poslední fragment.
- **Číslo fragmentu** – 13 bitů udávajících pozici fragmentu ve zprávě (udáváno v oktetech[1]).
- **Životnost** – 1 byte, jeho hodnota v tomto poli je při každém zpracování datagramu snížena o 1, po dosažení nuly je datagram zahozen a není již dále směrován.
- **Protokol** – 8 bitů označující protokol vyšší vrstvy.
- **Kontrolní součet hlavičky** – 16 bitový kontrolní součet hlavičky.
- **Zdrojová IP adresa** – 32 bitová IP adresa odesílatele.
- **Cílová IP adresa** – 32 bitová IP adresa příjemce.
- **DATA**

1 jeden oktet je 64 bitů

IPv6 pak nabízí v porovnání s IPv4 nesrovnatelně větší počet IP adres. Adresovací systém je v něm rozšířen z 32 bitů na 128 bitů. Protokol IPv6 však zdokonaluje také další funkce a urychluje výsledný přenos informace například tím, že při adresování není třeba analyzovat celou hlavičku IP datagramu. IPv6 také přináší zabezpečení, podobně jako rozšířený IPv4, v podobě zašifrovaného obsahu datagramu zabraňujícímu neoprávněnému čtení. Struktura datagramu protokolu IPv6 je zobrazena v následujícím obrázku.



Ilustrace 3: Struktura datagramu protokolu IPv6

- **Verze** – 4 bity označující verzi hlavičky.
- **Priorita** – 1 bit označující prioritu.
- **Značka toku** – 20 bitové pole určené pro zvýšení kvality real-time přenosů, kterým dokáže řídit datový tok. Unikátní označení toku dat umožní směrovačům mezi vysílací a přijímací stanicí směrovat datagramy stejnou cestou, aby zajistili jednotnost pořadí, v jakém mají být datagramy doručeny.
- **Délka dat** – 2 byty označující délku dat.
- **Příští hlavička** – 1 byte, informace o protokolu vyšší vrstvy nebo o přítomnosti rozšířené hlavičky.
- **Limit skoků** – 1 byte, jeho hodnota je při každém zpracování datagramu snížena o 1, po dosažení nuly je datagram zahozen a není již dále směrován.
- **Zdrojová IP adresa** – 128 bitová IP adresa vysílací stanice.
- **Cílová IP adresa** – 128 bitová IP adresa přijímací stanice.
- **DATA**

2.1.4.2 ICMP – Internet Control Message Protocol

ICMP zprávy jsou generovány následkem chyb v IP datagramech nebo za účelem diagnostiky sítě či směrování datagramů. Názorným příkladem případu vygenerování chybové ICMP zprávy je pokles čísla v poli TTL (Time To Live) na 0. V takovém případě je vygenerována chybová zpráva, která je následně odeslána původnímu odesilateli, který je tak informován o nedosažitelnosti cílové stanice. Hlavička ICMP protokolu je umístěna za hlavičkou IP

protokolu, v datové části datagramu. Struktura ICMP zprávy je znázorněna v následujícím obrázku.

Ilustrace 4: Struktura ICMP zprávy

- **Typ** – slouží k identifikaci typu zprávy.
- **Kód** – blíže specifikuje význam zprávy příslušného typu. Pro každý typ zprávy jsou definovány různé kódy.
- **Kontrolní součet** – 16 bitový kontrolní součet
- **Volitelné parametry** – až 128 bitové pole. Toto pole může obsahovat například identifikační číslo, číslo sekvence nebo masku podsítě, IP adresu brány a další informacemi. Informace obsažená v tomto poli je závislá na typu zprávy.

Příklad typů a kódů používaných v ICMP zprávách

- **Typ 0 – *Echo reply* (echo)**

  Odpověď na žádost *Echo request*. Pole parametrů obsahuje identifikátor a číslo sekvence.

- **Typ 8 – *Echo request* (žádost o echo)**

  Žádost odesílaná nástrojem *PING (Packet Internet Groper)*. Účelem této žádosti je prozkoumat konektivitu sítě. Pole parametrů obsahuje identifikátor a číslo sekvence.

- **Typ 3 – *Destination unreachable* (cíl nedosažitelný)**
  - Kód 0 – cílová síť nedosažitelná
  - Kód 1 – cílový hostitel nedosažitelný
  - a další

Příkaz *Echo request* lze vyvolat v příkazovém řádku operačního systému Windows nebo v terminálu operačních systémů Linux příkazem *ping* s parametrem tvořící IP adresu hostitele jehož dosažitelnost chceme tímto způsobem diagnostikovat. Například: *ping 192.168.0.1*.

2.1.4.3 ARP – Address Resolution Protocol

Dalším protokolem pracujícím na síťové vrstvě je ARP protokol. Účelem tohoto protokolu je získat fyzickou adresu zařízení s příslušnou IP adresou. Pokud se příjemce nachází ve stejné podsíti jako odesilatel, je třeba odeslat data přímo příjemci a k sestavení příslušného ethernetového rámce je nezbytná znalost jeho fyzické adresy. V takovém případě odešle nejprve odesilatel žádost *ARP request* linkovým broadcastem. Tomu odpovídá fyzická adresa *FF:FF:FF:FF:FF:FF*. Linkový broadcast obdrží všichni účastníci lokální sítě, její hranice však nepřekročí. Účastník sítě s příslušnou IP adresou pak odešle zpět odpověď *ARP reply*, kde uvádí vlastní IP adresu i fyzickou adresu.

Informace o fyzických adresách, které odpovídají jednotlivým IP adresám účastníků sítě si uživatelé zapisují do *ARP cache*. Odesilatel se tak nemusí dotazovat na příslušnou

fyzickou adresu pokaždé, když potřebuje odeslat nějaká data. Záznamy uvedené v *ARP cache* je pak nutno obnovit vždy po vypršení jejich platnosti. V příkazovém řádku operačního systému Windows je možné nechat zobrazit obsah *ARP cache* příkazem *arp -a*. V operačních systémech s Linuxovým jádrem pak stačí za tímto účelem příkaz *arp* bez parametru.

2.1.5 Transportní vrstva

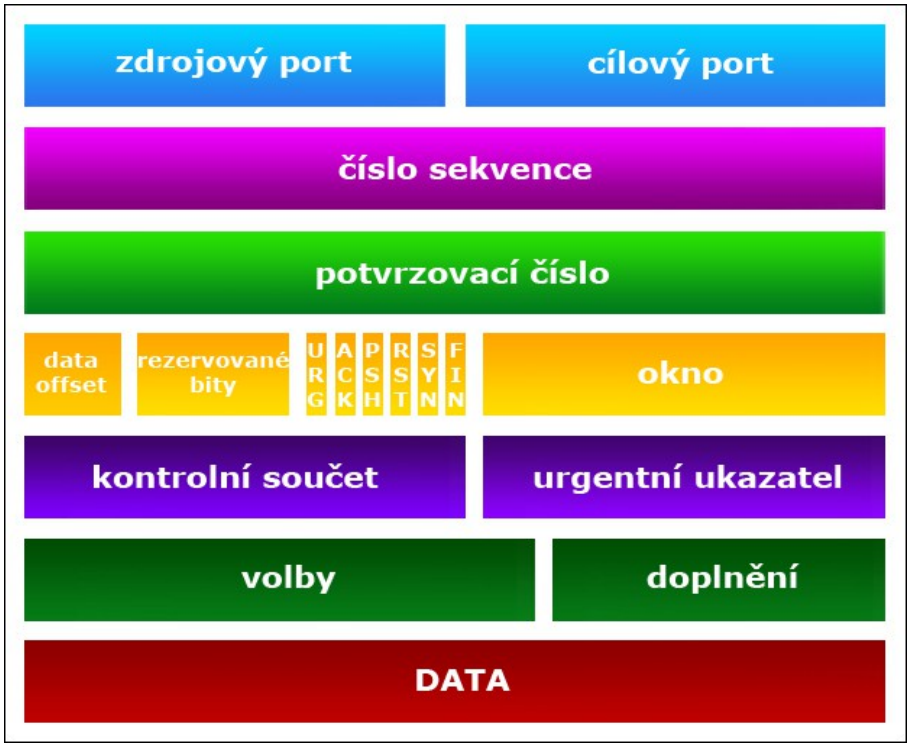
Účelem transportní vrstvy je realizace spojení pro uživatelské počítačové programy. Mezi dvěma stanicemi může být v jeden okamžik navázáno hned několik těchto spojení. Je tedy možný běh několika programů běžících v jeden okamžik a využívající každý své vlastní nezávislé spojení.

2.1.5.1 TCP (Transmission Control Protocol)

Nejznámějším a také nejvyužívanějším protokolem v transportní vrstvě je TCP (Transmission Control Protocol). Tento protokol je navržen tak, aby umožňoval bezchybný přenos dat. Zabezpečení bezchybného přijetí dat přijímací stanicí je dosaženo potvrzovacím cyklem. Pokud přijímací stanice nepotvrdí bezchybné přijetí dat, opakuje vysílací stanice vysílání nepotvrzeného paketu. Tento algoritmus je schopen detekovat i napravit chybné přijetí paketu. V praxi je označován jako Nagelův algoritmus [8].

Nespornou výhodou TCP protokolu je jeho schopnost vytvořit plně duplexní přenosový kanál mezi vysílací a přijímací stanicí. Server s veřejnou IP adresou může kdykoli přijmout zprávu od jakékoli stanice na určitém portu, na kterém naslouchá. Vysílací stanice vytvoří plně duplexní spojení a i když by byla za normálních podmínek nedosažitelná přes svoji IP adresu, protože leží v jiné síti, může dostat odpověď od serveru. Tohoto mechanizmu využívá celá řada aplikací či aplikačních protokolů pracujících nad transportní vrstvou. Například HTTP, email či SSH. Typickým příkladem je právě HTTP. Webový server naslouchá na portu 80. Klientská stanice zašle požadavek GET s žádostí o určitá data a následně data, o která zažádal obdrží v rámci stejného spojení, které klient vytvořil. Po úspěšném přijetí dat od serveru je spojení uzavřeno.

Struktura TCP paketu je znázorněna v následujícím obrázku.



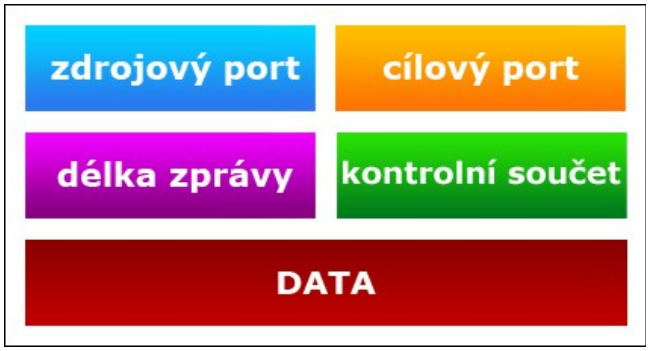
Ilustrace 5: Struktura TCP paketu

- **Zdrojový port** – 16 bitové číslo označující číslo zdrojového portu.
- **Cílový port** – 16 bitové číslo označující číslo cílového portu.
- **Číslo sekvence** – 4 byty, udává pozici prvního datového bytu v celkové přenášené zprávě. Je-li nastaven SYN bit jedná se o počáteční číslo sekvence.
- **Potvrzovací číslo** – 4 byty, je-li nastaven ACK bit jedná se o číslo sekvence, které očekává přijímač v příštím kroku.
- **Data offset** – 4 bity
- **Rezervované bity** – 6 rezervovaných bitů
- **URG** – 1 bit, nastavení priority
- **ACK** – 1 bit, označuje platnost čísla sekvence z pole udávajícího potvrzovací číslo.
- **PSH** – 1 bit, Push bit slouží k okamžitému natlačení dat do aplikace běžící na aplikační vrstvě.
- **RST** – 1 bit, slouží k znovusestavení spojení.
- **SYN** – 1 bit, je nastaven jedná-li se o počáteční číslo sekvence.
- **FIN** – 1 bit, označení posledních vysílaných dat.
- **Velikost okna** – 16 bitů udávající maximální počet oktetů, které je odesílatel schopen přijmout.
- **Kontrolní součet** – 16 bitový kontrolní součet hlavičky i dat zároveň.

- **Urgentní ukazatel** – 16 bitový ukazatel na první urgentní datový byte ve zprávě je-li nastaven bit URG.
- **Volby** – 0 až 44 bytů – specifikace dalších možností TCP paketu.
- **Doplnění** – slouží k doplnění (hlavička sestává z 32 bitových částí).
- **DATA**

2.1.5.2 UDP (User Datagram Protocol)

Dalším možným protokolem transportní vrstvy je UDP (User Datagram Protocol). Na rozdíl od TCP protokolu nevyžaduje UDP protokol potvrzení přijetí zprávy či sestavení speciálního přenosové kanálu, zaručujícího příjem dat v určitém pořadí a spolehlivé přijetí dat. Datagramy tak mohou dorazit v pozměněném pořadí, některé mohou dorazit vícekrát a některé se mohou bez náhrady ztratit. Díky těmto vlastnostem může být využití UDP protokolu vhodnou možností především u časově kritických aplikací, kde přijetí dat v daný okamžik je mnohem důležitější než ztráta času s jeho potvrzování a jeho opětovným odesíláním a čekáním na opožděná data, která už pak ve chvíli jejich přijetí přijímací stanicí mohou postrádat význam. Struktura UDP datagramu je velmi jednoduchá a je znázorněna na následujícím obrázku.



Ilustrace 6: Struktura UDP datagramu

- **Zdrojový port** – 16 bitové číslo označující číslo zdrojového portu.
- **Cílový port** – 16 bitové číslo označující číslo cílového portu.
- **Délka zprávy** – 16 bitové číslo udávající délku zprávy
- **Kontrolní součet** – volitelně může UDP datagram obsahovat 16 bitový kontrolní součet, pokud není kontrolní součet implementován je pole doplněno nulami. U IPv6 je kontrolní součet povinný.
- **DATA**

V některých aplikacích může být nevýhodou UPD protokolu to, že nevytváří plně duplexní spojení mezi komunikujícími stanicemi. Vlastně nevytváří žádné spojení. Aplikace jednoduše naslouchá na určeném portu, popřípadě odešle z jiného portu data na konkrétní vzdálený soket[2] příjemce. Problém může nastat v případě, že příjemce nelze identifikovat pomocí IP adresy. Takovýto případ nastává pokud se cílová stanice nenachází ve stejné síti a ani nevlastní veřejnou IP adresu. Pomocí IP adresy pak jednoduše není dosažitelná. S touto

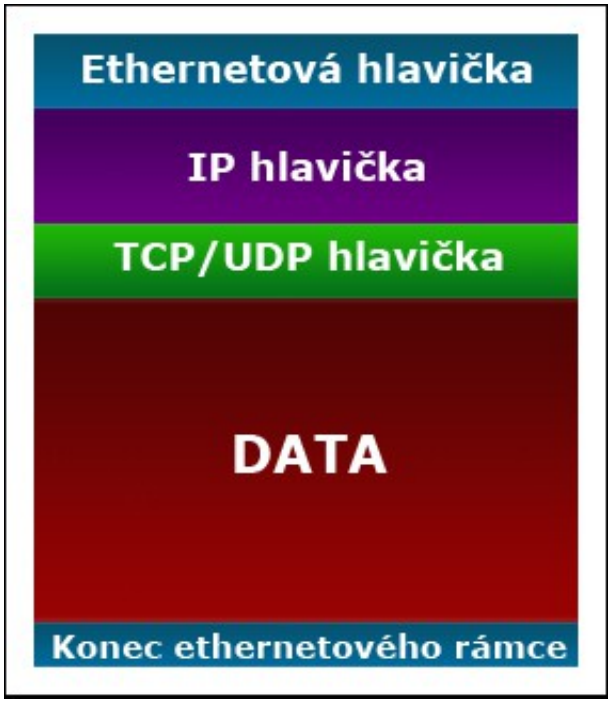
2 soket – dvojice tvořená IP adresou a číslem portu, někdy také označovaná jako endpoint

vlastností je třeba počítat při návrhu konkrétní aplikace. Pokud tedy stanice s veřejnou IP adresou přijme data přes UDP protokol od stanice, která neleží ve stejné síti, musí vysílací stanice počítat s tím, že nikdy nedostane od přijímací stanice s veřejnou IP adresou odpověď. Pokud by si aplikace tento druh komunikace vyžadovala bylo by nutné použít TCP protokol, který vytvoří potřebné plně duplexní spojení.

Výhodou UDP protokolu naopak může být jeho nenáročnost, na kterou je v embedded systémech kladen vysoký důraz. Ale zároveň i přenos dat v časově kritických real-time aplikacích, kde více než na spolehlivost je kladen důraz na to, aby byla data doručena včas. Právě proto je UDP využíván i v aplikacích jako je VoIP[3] či IPTV[4] [17].

2.1.6 Výsledný tvar ethernetového rámce

Pokud tedy využijeme znalostí referenčního modelu ISO/OSI můžeme si představit následující sestavení dat přenášených sítí.



Ilustrace 7: Struktura přenášených dat

2.1.7 Prezentační vrstva

Nižší vrstvy referenčního modelu ISO/OSI dělají vše proto, aby data k příjemci dorazila ve stejném tvaru v jakém byla odeslána stanicí vysílací. I když však data dorazí k příjemci zcela nezměněna, nezaručuje to, že budou na přijímací stanci stejně zobrazena. Přijatá data mohou být například odeslána v jiném kódování než v jakém se je bude přijímací stanice snažit zobrazit. Jiný problém by mohl způsobit způsob zobrazování čísel s plovoucí desetinnou čárkou či samotné rozsahy číselných proměnných a podobně.

Úkolem prezentační vrstvy je tedy vyřešit problémy se správnou prezentací přenesených dat. Samotný význam dat již však předmětem prezentační vrstvy není a tedy se jím ani nezabývá.

3 VoIP – Voice over Internet Protocol – hlasové komunikace a média přenášená přes IP protokol
4 IPTV – Internet Protocol TeleVision – televizní služby poskytované přes IP protokol

2.1.8 Aplikační vrstva

Na rozdíl od prezentační vrstvy, která se nezabývá vlastním významem přenášených dat, jsou právě tyto informace aplikační vrstvou rozeznávány. Jednotlivé aplikace běžící na aplikační vrstvě tak rozeznávají například význam jednotlivých čísel, zda se jedná o počet přihlášení do systémové databáze, počet zaměstnanců, aktuální teplotu a podobně. Stejně jako na nižších vrstvách i na aplikační vrstvě se můžeme setkat s různými protokoly. Tyto protokoly mohou být vázány ke konkrétní aplikaci, ale existují i aplikační protokoly, které jsou v některých případech nezbytné pro správný chod sítě a s aplikací běžící na aplikační vrstvě nemají nic společného. Případem takového protokolu je například DHCP protokol.

2.1.8.1 DHCP protokol (Dynamic Host Configuration Protocol)

DHCP protokol poskytuje nástroje pro přenos síťových konfiguračních informací přes ethernetové rozhraní. Jeho hlavní funkcí je schopnost přiřazení znovupoužitelných síťových adres jednotlivým účastníkům sítě.

DHCP protokol je postaven na principu komunikace klient-server. Existují tři různé druhy přiřazení síťových adres jednotlivým klientům:

- Automatické přiřazení – přiřazuje klientovi permanentní síťovou adresu.
- Dynamické přiřazení – přiřazuje klientovi síťovou adresu na určitou dobu.
- Manuální přiřazení – síťová adresa je přiřazena klientovi na základě pokynu administrátora nebo pravidel jím definovaných.

Z těchto tří metod však pouze dynamické přiřazení síťových adres umožňuje jejich znovupoužití. V okamžiku kdy klientovi vyprší platnost síťové adresy a nepožádá o její prodloužení (například proto, že už není v síti přítomen), může být tato síťová adresa přiřazena dalšímu klientovi.

Ve spojení s DHCP protokolem v síti můžeme najít tři různé typy stanic:

- DHCP server – účastník sítě, který přiřazuje konfigurační parametry jednotlivým účastníkům sítě, tj. DHCP klientům.
- DHCP klient – účastník sítě, který obdržuje konfigurační parametry od DHCP serveru. Konfiguračními informacemi je myšlena například síťová adresa.
- BOOTP přenosový agent – účastník sítě, který předává DHCP zprávy mezi DHCP klientem a DHCP serverem. Umožňuje konfigurace několika sítí najednou za použití pouze jednoho DHCP serveru [6].

2.1.8.2 HTTP protokol (HyperText Transfer Protocol)

HTTP je protokol na aplikační vrstvě. Původní verze 0.9 protokolu byla jednoduchým textovým protokolem určeným k přenosu informace přes internet. Verze 1.0 umožnila zprávě být ve formátu MIME zprávy. MIME zpráva se skládá z typu a jeho volitelných podtypů, tzv. meta informací. Například typ řetězec tak může mít definován podtyp uvádějící jemu příslušící kódování. Nyní se v drtivé většině případů používá verze 1.1, která navíc bere v úvahu hierarchii proxy serverů, potřebu stabilního spojení, virtuální stanice atd [7].

Protokol je navržen pro komunikaci typu žádost/odpověď. Nesporně nejpoužívanější žádostí protokolu HTTP je žádost typu GET. Pokud chce klient zažádat o dokument uložený na serveru vyšle žádost v následujícím formátu:

GET http://www.server.cz/zadany_soubor.html HTTP/1.1

Server na žádost může odpovědět třemi různými způsoby:

- vrátí požadovaný dokument a kód 200
- odpoví, že požadovaný dokument se nachází jinde a příslušný kód 3xx
- oznámí chybu indikující jiný problém a příslušný kód začínající 4xx nebo 5xx

Pokud server vrací požadovaný dokument, odpovídá v následujícím formátu:

HTTP/1.1 200 OK
hlavička 1
hlavička 2 (popřípadě další hlavičky níže, pokud je třeba)
prázdný řádek
samotný dokument

Odpověď tedy může vypadat následovně:

HTTP/1.1 200 OK
Content-type: text/html
Date: Mon, 1 June 2012 08:00:00 GMT

<html>
<head>
<title>Nazev stranky</title>
</head>
<body>
...
</body>
</html>

3 Embedded Ethernet

Embedded ethernet je definován jako single-chip implementace ethernetového rozhraní. Jinými slovy řečeno, o embedded ethernetu hovoříme pokud embedded zařízení disponuje možnostmi komunikovat s okolním světem pomocí ethernetového rozhraní [20].

Důvod proč embedded ethernet specifikujeme a oddělujeme ho od klasického internetu je, že požadavky na embedded zařízení jsou především jejich jednoduchost za účelem minimalizace rozměrů a jejich váhy. Díky tomu embedded zařízení nedisponují zdaleka tak velkým výpočetním výkonem jako výkonné počítače a také nároky na využití paměti musí být co nejvíce minimalizovány.

3.1 lwIP stack

LwIP (LightWeight IP) stack je implementace TCP/IP stacku určená pro embedded systémy. Zaměřuje se na snížení nároků na využití paměti a na snížení velikosti kódu. LwIP stack využívá zjednodušené API, které nevyžaduje nadbytečné kopírování dat, což má příznivý dopad na výpočetní a paměťové nároky.

LwIP je open source TCP/IP stack sepsaný v jazyce C. Je schopen pracovat s, i bez operačního systému. Velikost kódu se typicky pohybuje v rozmezí od 25 do 40 kB zatímco nároky na využití operační paměti se pohybují od 15 kB až k pár desítkám kB [12].

3.1.1 Charakteristické vlastnosti

- IP protokol umožňující přeposílání datagramů přes více síťových rozhraní
- ICMP protokol pro údržbu a diagnostiku sítě
- UDP protokol
- TCP protokol
- DHCP protokol pro automatické přiřazení IP adresy klientským stanicím
- ARP protokol
- Podpora více síťových rozhraní
- a další

3.1.2 Implementace

LwIP stack sestává především z definice TCP/IP protokolů, nicméně obsahuje mnoho dalších funkcí pro správu paměti, funkcí k výpočtu kontrolních součtů a dalších funkcí. Protokoly nižších vrstev jsou obhospodařovány hardwarově. Naopak protokoly vyšších vrstev již musí být součástí konkrétní aplikace.

Soubor TPC/IP protokolů je dle referenčního modelu rozdělen do vrstev. Každý z implementovaných protokolů má svůj vlastní vstupní bod. Některé TCP/IP implementace rozdělují jednotlivé protokoly do samostatných procesů. Tento přístup odděluje velmi pečlivě jednotlivé vrstvy. Na rozdíl od nich LwIP implementace používá model, ve kterém jsou všechny protokoly zařazeny v jednom hlavním procesu odděleném od jádra operačního systému. Tento přístup umožňuje v některých mezních případech spolupráci mezi vrstvami.

Jako příklad bych uvedl výpočet kontrolního součtu příchozího TCP segmentu. Za tímto účelem je třeba znát IP adresu cílové stanice. V případě LwIP stacku si transportní vrstva může IP adresu vyčíst sama z IP hlavičky, jelikož její struktura je mu známa a obejde se tak předávání IP adresy jako parametru čímž se sníží nároky na využití paměti. Umístění LwIP do vlastního procesu umožňuje také jak použití s operačním systémem tak i bez něj. Aby byla zachována univerzálnost této implementace, není z programu přistupováno nikdy přímo k samotným vnitřním datovým strukturám, nýbrž vždy přes volání funkcí a procedur s využitím časovačů a mechanizmu předávání zpráv. Správa paměti potom musí být schopna zacházet s buffery proměnné velikosti.

Správa paměti LwIP využívá dynamické datagramové buffery zvané *pbuf*. Jejich struktura umožňuje jejich alokování v dynamické paměti stejně jako jejich ukládání do paměti statické. Správa paměti obstarává mechanismy alokace a dealokace blízké paměti. Zároveň správa paměti hlídá využití paměti tak, aby data TCP/IP stacku nezabrala celou dostupnou paměť.

LwIP stack uchovává datové struktury jednotlivých síťových rozhraní uspořádané ve formě propojených seznamů. Každá datová struktura uvádí ukazatel na následující datovou strukturu, jméno síťového rozhraní a jeho příslušnou IP adresu. Dále obsahuje dva funkční ukazatele. První ukazuje na funkce, která zpracovává příchozí data. Druhý potom ukazuje na ovladač zařízení, který je využíván k odeslání dat po fyzické vrstvě.

3.1.3 LwIP API

API (Application Program Interface) označuje rozhraní, které umožňuje programátorovi vytvořit aplikaci využívající soubor funkcí a procedur či tříd nějaké knihovny nebo i operačního systému. V případě LwIP stacku jde tedy o funkce a procedury, které je možné volat z programu za účelem využití možností, které LwIP stack nabízí.

LwIP API je sestaveno tak, že nevyžaduje žádné kopírování dat mezi aplikací a stackem. LwIP stack dokáže vnitřně manipulovat s daty samostatně.

3.1.4 Správa paměti a systém *pbuf*

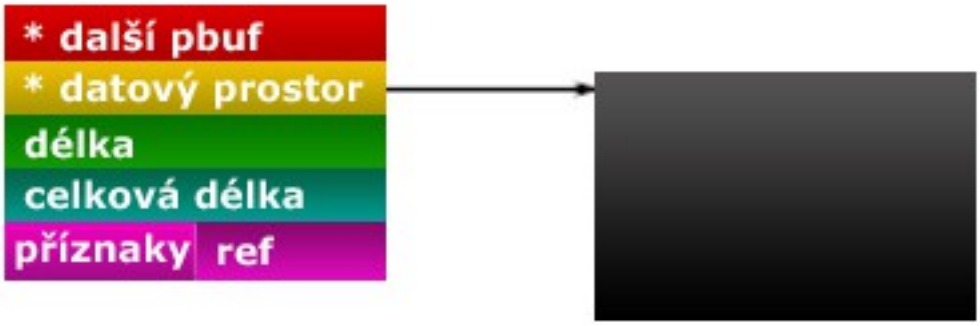
Správa paměti v komunikačních procesech musí být schopna akumulovat příchozí zprávy o proměnné velikosti. Tato velikost se pohybuje od krátkých odpovědí ICMP protokolu o velikosti pouhých několik bytů až po plně využitý TCP datagram čítající několik set až tisíc bytů cenných dat.

3.1.5 Pbuf – paketové/datagramové buffery

Pbuf je vnitřní reprezentací paketu/datagramu uvnitř LwIP stacku. Pro přehlednost budu nadále hovořit o paketech, i když se v případě UDP protokolu může jednat o datagramy. Tyto buffery jsou navrženy za účelem minimalizace paměťových a výkonových nároků stacku. Struktura těchto bufferů umožňuje jak ponechání dat přímo v dynamické paměti tak i jejich uložení do statické paměti. Pbuf může být spojen do seznamu nazývajícího se *pbuf řetězec* (pbuf chain). Obsah jednoho paketu tak může být rozdělen do několika *pbuf*. *Pbuf řetězec* může být složen z různých typů *pbuf*. Každý z uvedených typů *pbuf* je vhodný pro jiné použití. Celkem se můžeme setkat se třemi typy *pbuf*:

- ***PBUF_ROM***

  Datová část paketu je uložena v paměti, která není spravovaná systémem *pbuf*. Může jít například o data uložená v paměti ROM, která je spravovaná samotnou aplikací. *Pbuf_rom* je vhodný ve chvíli, kdy aplikace odesílá data, která jsou alokována v paměti, která je spravována aplikací. Tato data často nejsou dotčena jejich zpracování TCP/IP stackem, často také bývají uloženy v paměti ROM. Struktura *pbuf_rom* je uvedena v následujícím obrázku.



Ilustrace 8: Struktura pbuf_rom

- ***PBUF_RAM***

  Datová část paketu je uložena v dynamické paměti spravované systémem *pbuf*. Hlavičky, které jsou předřazeny datům z *pbuf_rom* jsou uloženy v *pbuf_ram*, které jsou uloženy v *řetězci pbuf* před příslušným *pbuf_rom*.

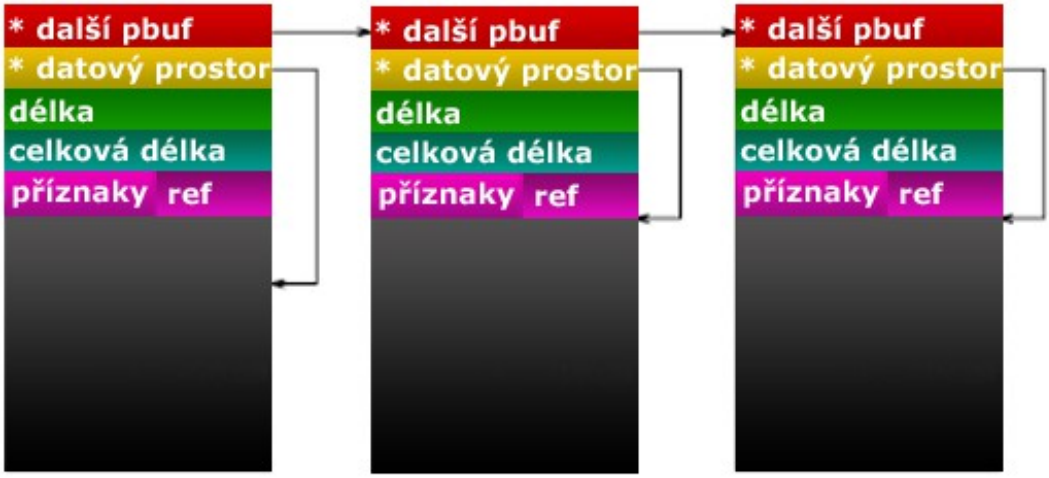
Ilustrace 9: pbuf_ram v řetězci následovaný pbuf_rom

Využití *pbuf_ram* je vhodné i v případě, kdy stanice odesílá dynamicky generovaná data. V takovém případě *systém pbuf* alokuje paměť nejen pro aplikační data, ale i pro hlavičky jednotlivých protokolů, které budou předsazeny samotným datům. Velikost připojovaných hlaviček se nastavuje při kompilaci. Tato situace je znázorněna v následujícím obrázku.

Ilustrace 10: struktura nezřetězeného pbuf_ram

- ***PBUF_POOL***

  Sestává ze seznamu *pbuf* pevné velikosti. *Pbuf_pool* je využíván nejčastěji ovladačem zařízení. Operace alokování paměti pro jednotlivý *pbuf* je rychlá a hodí se k použití v rutině přerušení.



Ilustrace 11: zřetězený pbuf_pool

V podstatě všechny příchozí *pbuf* jsou typu *pbuf_pool* a všechny odchozí jsou buď typu *pbuf_rom* nebo *pbuf_ram*.

Vnitřní struktura *pbuf* se skládá ze dvou ukazatelů, dvou polí označujících délku, příznakových bitů a pole *ref* označující referenční číslo pbuf. Ukazatel *další* ukazuje na další *pbuf*, pokud je součástí *řetězce pbuf*. Ukazatel *datový prostor* ukazuje na první byte dat uložených v *pbuf*. Pole *délka* udává délku dat uložených v obsahu *pbuf*. Pole *celková délka* udává délku dat uložených v obsahu *pbuf*, ke kterému náleží a všech následujících *pbuf*. *Celková délka* je tedy součet pole *délka* a pole *celková délka* v následujícím *pbuf*. Příznakové bity identifikují typ *pbuf* a pole *ref* obsahuje referenční číslo *pbuf*. Pole *délka* a *celková délka* jsou 16 bitové, příznakové *bity* a pole *ref* jsou 4 bitové. Velikost ukazatelů záleží na architektuře použitého procesoru.

K manipulaci s *pbuf* jsou k dispozici následující funkce:

- ***pbuf_alloc*** – slouží k alokaci paměti libovolného ze tří uvedených typů *pbuf.*
- ***pbuf_ref*** – slouží k inkrementaci referenčního čísla.
- ***pbuf_free*** – slouží k dealokaci paměti. Postupně snižuje referenční číslo a dealokuje všechny *pbuf* dokud nedosáhne referenční číslo nuly. Tak dojde ke kompletní dealokaci *pbuf řetězce*.
- ***pbuf_realloc*** – zmenší *pbuf* tak, aby zabíral pouze nezbytnou část paměti potřebnou k uložení obsažených dat.
- ***pbuf_header*** – pozmění ukazatel na datový prostor a pozmění pole *délka* a *celková délka* tak, aby umožnil připojení hlaviček jednotlivých protokolů k datům.
- ***pbuf_chain*** – slouží k zřetězení *pbuf*.
- ***pbuf_dechain*** – slouží k rozdělení *pbuf řetězce.*

3.1.6 Správa paměti

LwIP stack využívající schéma pbuf spravuje blízké paměťové oblasti, ve kterých se stará o alokaci, dealokaci paměti a může zmenšovat velikost alokovaných paměťových bloků dle potřeby. LwIP stack využívá pouze poskytnutou část systémové paměti, aby nedošlo k úplnému využití paměti právě TCP/IP stackem a nezbyla již žádná paměť pro další procesy.

Každá z paměťových oblastí spravovaných LwIP stackem má na svém začátku ukazatel na předchozí paměťový blok, ukazatel na následující paměťový blok a příznakový bit, který udává zda je paměť alokovaná nebo není.

Pokud je třeba alokovat paměť je vyhledávána a alokována první volná dostatečně velká část paměti. Při dealokaci paměti jsou zároveň zkontrolovány sousední bloky a pokud jsou volné jsou spojeny do jednoho většího nevyužitého bloku, aby se zamezilo fragmentaci.

3.1.7 IP zpracování

LwIP stack implementuje pouze základní funkce IP zpracování. Umožňuje odesílat, přijímat a přeposílat pakety, ale ačkoli pro většinu aplikací to nepřináší potíže, nedokáže odesílat či přijímat fragmentované pakety. Pokud je embedded zařízení schované za proxy serverem je žádoucí, aby proxy server všechny příchozí fragmentované IP pakety znovu seskládal. Pokud tak neučiní, jsou fragmentované IP pakety v tichosti zahozeny.

3.1.7.1 Příjímání IP paketů

Zpracování přijatých IP paketů začíná ve chvíli, kdy je ovladačem síťového zařízení zavolána funkce *ip_input()*. V první fázi je rozpoznána verze IP protokolu, vyčtena délka hlavičky a zkontrolován kontrolní součet nad IP hlavičkou. Pokud kontrolní součet odpovídá je vyčtena cílová IP adresa a je překontrolováno, zda odpovídá síťové adrese síťového rozhraní. Pokud adresy odpovídají, je dle obsahu pole obsahujícího identifikátor protokolu vyšší vrstvy rozhodnuto kam je paket předán pro další zpracování.

3.1.7.2 Odesílání IP paketů

Odchozí pakety jsou zpracovávány funkcí *ip_output()*, která využívá funkce *ip_route()*. Funkce *ip_route()* vybere vhodné síťové rozhraní, pokud jich je v systému zapojeno hned několik. Vhodné síťové rozhraní je rozpoznáno z jeho masky podsítě. Pokud síťová adresa rozhraní překrytá maskou podsítě odpovídá cílové adrese překryté maskou podsítě, je síťové rozhraní označeno jako vhodné a přeneseno jako parametr funkci *ip_output_if()*. Tato funkce se postará o vyplnění všech polí hlavičky IP protokolu a vypočítá kontrolní součet IP hlavičky.

3.1.7.3 Přeposílání IP paketů

K přeposílání paketů dochází právě ve chvíli, kdy žádné ze síťových rozhraní embedded zařízení nemá přidělenu síťovou adresu odpovídající cílové síťové adrese uvedené v hlavičce IP protokolu. V takovém případě je paket předán funkci *ip_forward()*. Funkce se postará o snížení hodnoty v poli TTL (Time To Live). Pokud výsledná hodnota v poli TTL dosáhla nuly, je paket zahozen a původnímu odesilateli je odeslána ICMP chybová zpráva. Pokud je pole TTL nenulové je třeba před přeposláním paketu ještě upravit kontrolní součet hlavičky z důvodu změny obsahu pole TTL. Poté může být IP paket přeposlán přes vhodné síťové rozhraní, které je nalezeno stejným principem jako při odesílání IP paketů.

3.1.8 ICMP zpracování

ICMP protokol i jeho zpracování je velice jednoduché. Pokud je při zpracování přijatého IP paketu identifikován protokol ICMP je paket předán ke zpracování funkci *icmp_input()*, která dekóduje hlavičku ICMP protokolu provede vhodnou interakci. V některých případech je paket předán ke zpracování vyšším vrstvám. Například zpráva oznamující nedostupnost cílové stanice (destination unreachable) může být odeslána protokoly transportní vrstvy, především potom UDP protokolem. Za tímto účelem je vyvolána funkce *icmp_dest_unreach()*.

Široce využívané jsou v ICMP protokolu zprávy *Echo Request* a *Echo Reply*. K jejich zpracování dochází ve zmíněné funkci *icmp_input()*, která zajistí přehození cílové a zdrojové síťové adresy a obsah zprávy zamění z *Echo Request* za *Echo Reply*. Funkce se také postará o úpravu kontrolního součtu. Pro samotné odeslání ICMP zprávy je zpráva předána zpět síťové vrstvě, tedy funkci *ip_output()*.

3.1.9 UDP zpracování

Mluvíme-li o UDP zpracování, posouváme se od síťové vrstvy vzhůru na vrstvu transportní. Struktura UDP datagramu je velice jednoduchá a výhody i nevýhody použití UDP protokolu na transportní vrstvě byly shrnuty v minulých kapitolách.

Jelikož na jednom zařízení je možné spravovat více UDP spojení, je stav každého spojení odpovídajícímu jednomu konkrétnímu soketu uložen v proměnné udp_pcb *(UDP Protocol Control Block)*. Udp_pcb je proměnná typu struktura v následujícím tvaru:

```
struct udp_pcb {
   struct udp_pcb *next;
   struct ip_addr local_ip, dest_ip;
   u16_t local_port, dest_port;
   u8_t flags;
   u16_t chksum_len;
   void (* recv)(void *arg, struct udp_pcb *pcb, struct pbuf *p);
   void *recv_arg;
};
```

Jak naznačuje první položka struktury představující *UDP PCB bloky* jsou udržovány v propojeném seznamu. Ve chvíli, kdy dorazí příchozí UDP datagram je vyhledán příslušný *UDP PCB blok* a datagram je pak předán funkci, která se stará o samotné zpracování dat. Ukazatel na tuto funkci je uložen v *PCB bloku* a představuje ho položka ***recv*. Struktura *UDP PCB bloku* obsahuje:

- ***next** - ukazatel na další PCB blok v seznamu UDP PCB bloků
- ***local_ip*** – místní síťová adresa
- ***dest_ip*** – vzdálená síťová adresa
- ***local_port*** – místní číslo portu
- ***dest_port*** – vzdálené číslo portu
- ***flags*** – příznakové bity identifikují způsob generování kontrolního součtu. Generování kontrolního součtu je možné úplně vypnout nebo využít UDP Lite, kde kontrolní součet pokrývá pouze část datagramu.
- ***chksum_len*** – udává, jak velká část datagramu je pokryta kontrolním součtem, pokud je použita metoda UDP Lite generování kontrolního součtu.
- ***recv*** – callback funkce, která je zavolána při přijímání dat pro příslušné UDP spojení
- ***recv_arg*** – volitelný parametr, který je předáván funkci ***recv*** volané při přijímání dat

Odesílání dat pomocí UDP protokolu není o nic složitější než jejich příjem. Z aplikační vrstvy je volána funkce *udp_send()*, která využívá funkci *udp_output()* starající se o vyplnění polí v hlavičce UDP protokolu a výpočet kontrolního součtu, pokud je zapotřebí. Nakonec je datagram předán síťové vrstvě k IP zpracování a následnému odeslání.

3.1.10 TCP zpracování

TCP je protokol pohybující se na transportní vrstvě. Umožňuje spolehlivý přenos streamu dat kanálem vytvořeným mezi 2 účastníky sítě a jeho předání vyšším vrstvám. Kód obstarávající TCP zpracování zabírá zhruba polovinu z celého kódu LwIP stacku, což už samo o sobě naznačuje, že se bude jednat o velmi robustní protokol.

Proces přijímání dat začíná na síťové vrstvě ve funkci *ip_input()*. Pokud po ověření IP hlavičky při procesu přijímání dat zjistí síťová vrstva, že protokolem vyšší vrstvy je TCP protokol, jsou data předána funkci *tcp_input()*, kde je provedena kontrola kontrolního součtu a přečtení pole *Volby* pro bližší specifikaci TCP paketu. Zároveň se rozhoduje dle čísla portu,

ke kterému spojení patřičný TCP paket náleží. Paket je dále zpracován funkcí *tcp_process()*. Tato funkce se chová jako stavový automat. Pokud se stav spojení nachází ve stavu, kdy je možné data ze sítě přijímat tak dále předává paket ke zpracování funkci *tcp_recieve()*, která se následně postará o předání dat aplikační vrstvě.

Pokud chce aplikační vrstva odeslat data pomocí TCP protokolu, je volána funkce *tcp_write()*, která následně předává řízení procesu odesílání dat funkci *tcp_enqueue()*. Funkce *tcp_enqueue()* se postará o rozložení dat na bloky o přenositelné velikosti pokud je to zapotřebí. Dále funkce *tcp_output()* zjistí zda je možné v daný okamžik odeslat data. Pokud je to možné tak jsou data odeslána pomocí funkcí *ip_route()* a *ip_output_if()*.

Stav každého TCP spojení je uložen v proměnné *tcp_pcb (TCP Protocol Control Block)*. Tcp_pcb je proměnná typu struktura v následujícím tvaru:

```
struct tcp_pcb {
    struct tcp_pcb *next;
    enum tcp_state state;
    void (* accept)(void *arg, struct tcp_pcb *newpcb);
    void *accept_arg;
    struct ip_addr local_ip;
    u16_t local_port;
    struct ip_addr dest_ip;
    u16_t dest_port;
    u32_t rcv_nxt, rcv_wnd;
    u16_t tmr;
    u32_t mss;
    u8_t flags;
    u16_t rttest;
    u32_t rtseq;
    s32_t sa, sv;
    u32_t rto;
    u32_t lastack;
    u8_t dupacks;
    u32_t cwnd, u32_t ssthresh;
    u32_t snd_ack, snd_nxt, snd_wnd, snd_wl1, snd_wl2, snd_lbb;
    void (* recv)(void *arg, struct tcp_pcb *pcb, struct pbuf *p);
    void *recv_arg;
    struct tcp_seg *unsent, *unacked,  *ooseq;
};
```

Tyto *tcp_pcb* proměnné jsou uchovávány v propojeném seznamu. Tento seznam tak obsahuje tcp*_pcb* každého existujícího TCP spojení. Ve chvíli, kdy dorazí příchozí TCP paket je vyhledán příslušný TC*P PCB blok* a datagram je pak předán funkci, která se stará o samotné zpracování dat. Ukazatel na tuto funkci je uložen v *PCB bloku* a představuje ho položka ***recv.* Struktura *UDP PCB bloku* dále obsahuje:

- ***next*** - ukazatel na další PCB blok v seznamu UDP PCB bloků
- ***tcp_state*** – stav TCP spojení
- ***local_ip*** – místní síťová adresa
- ***dest_ip*** – vzdálená síťová adresa
- ***local_port*** – místní číslo portu
- ***dest_port*** – vzdálené číslo portu

- ***rcv_nxt, rcv_wnd*** – proměnné uchovávající informace o příjemci
- ***mss*** – maximální velikost segmentu
- ***rttest*** – odhadovaný čas k odeslání a potvrzení přijetí paketu
- ***rtseq*** – číslo sekvence pro příslušný ***rttest***
- ***sa, sv –*** průměrná hodnota ***rttest*** a její kolísání
- ***rto*** – pokud do uplynutí této doby nepřijde potvrzení o přijetí odeslaného paketu, je paket odeslán znovu. K výpočtu této doby se využívají hodnoty polí ***rttest, rtseq, sa*** a ***sv***
- ***lastack*** – číslo sekvence, které bylo potvrzeno přijetí
- ***dupacks*** – počet potvrzení pro číslo sekvence uložené v proměnné ***lastack***
- ***cwnd, ssthresh*** – dodatečné parametry komunikace
- ***snd_ack, snd_nxt, snd_wnd, snd_wl1, snd_wl2, snd_lbb*** – proměnné obsahující informace používané pří odesílání dat. Obsahují například nejvyšší číslo sekvence potvrzeným přijetím, číslo sekvence očekávané v dalším kroku a další
- ****recv*** – callback funkce, která je zavolána při přijímání dat pro příslušné TCP spojení
- ****recv_arg*** – volitelný parametr, který je předáván funkci ***recv*** volané při přijímání dat
- ****unsent, *unacked, *ooseq*** – jedná se o fronty využívané při procesu odesílání a přijímání dat. ****unsent*** obsahuje data přijatá od aplikace, určená k odeslání, která však ještě odeslána nebyla. ****unacked*** obsahuje data, která již byla odeslána, ale nebylo potvrzeno jejich přijetí. Přijatá data jsou řazena do fronty v proměnné ****ooseq***

Proměnné ****unsent, *unacked, *ooseq***,ve kterých jsou seřazena data jsou datového typu struktura nazvané tcp_seg (TCP segment). Tyto proměnné mají následující strukturu:

```
struct tcp_seg {
  struct tcp_seg *next;
  u16_t len;
  struct pbuf *p;
  struct tcp_hdr *tcphdr;
  void *data;
  u16_t rtime;
}
```

Tyto struktury jsou vnitřní reprezentací TCP segmentu. Uvedená pole mají následující význam:

- ****next*** – ukazatel na další TCP segment v seznamu
- ***len*** – délka TCP segmentu. Pokud se jedná o segment obsahující data, jedná se o délku dat. Jedná-li se o prázdný segment s příznakovými bity SYN nebo FIN je pole ***len*** nastaveno na hodnotu 1.
- ****p*** – ukazatel na *pbuf* obsahující aktuální segment
- ****tcphdr*** – ukazatel na tcp hlavičku obsaženou v segmentu

- ***data** – ukazatel na datovou část obsaženou v segmentu
- ***rtime** – pokud do uplynutí této doby nepřijde potvrzení o přijetí odeslaného paketu, je paket odeslán znovu

Data, která je třeba odeslat jsou ve funkci tcp_*enqueue()* rozdělena na části patřičné velikosti. Tyto části jsou poté zabaleny do příslušného *pbuf* a ten je posléze uložen ve struktuře *tcp_seg*. V příslušném *pbuf* je vytvořena TCP hlavička. Segmenty jsou následně řazeny do fronty k odeslání v seznamu ***unsent*** zmíněném výše. Funkce tcp_enqueue() se snaží vyplnit segmenty maximálním možným množstvím dat. Za tímto účelem může řetězit struktury *pbuf* vkládané do segmentu. Posléze je volána funkce *tcp_output()*, která doplní chybějící části TCP hlavičky a odešle paket využitím funkcí *ip_route()* a *ip_output_if()*. TCP segment je poté přesunut do seznamu ***unacked***, kde sečká do té doby než bude jeho přijetí potvrzeno příjemcem.

3.2 Alternativní protokolové implementace

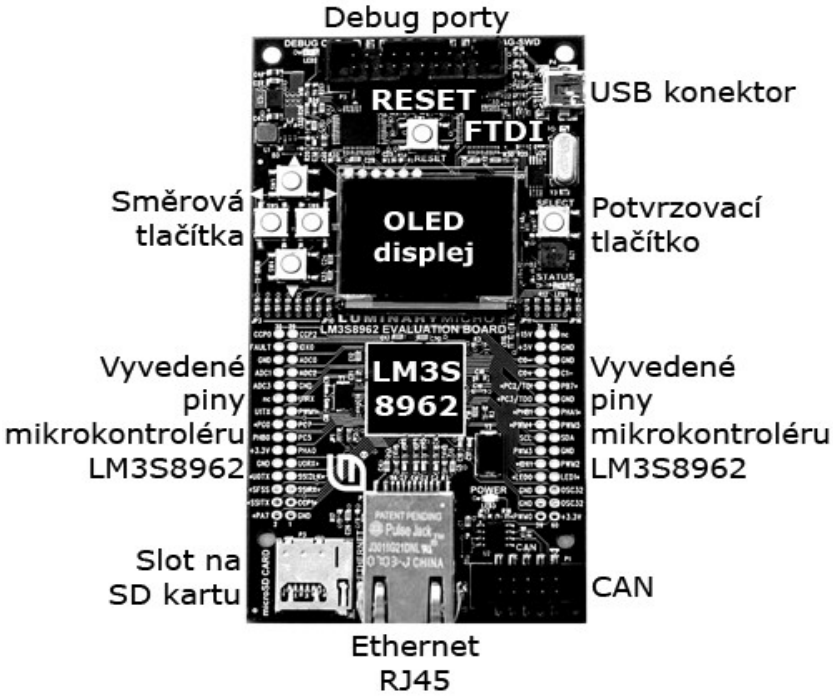
LwIP TCP/IP není zdaleka jedinou možností. Existuje celá řada konkurenčních komerčních i open sourceových TCP/IP stacků, které je možno implementovat. Několik z nich pro představu uvádím v následujícím seznamu:

- ***µIP*** – jednoduchý TCP/IP stack schopný pracovat i na 8bitových mikrokontrolérech. Původně vytvořen Adamem Dunkelsem, nyní vyvíjen širší skupinou vývojářů. Kód zabírá několik kilobajtů, paměťové nároky jsou sníženy na několik stovek bajtů.
- ***Interpeak IPNET*** – komerční plnohodnotný TCP/IP stack
- ***EtherNut*** – opensource hardwarový a softwarový projekt, kterému jde o návrh embedded zařízení s vlastním operačním systémem a vlastním implementovaným TCP/IP stackem.
- ***Web51*** – open source TCP/IP projekt založený na mikrokontrolérech rodiny 8051 [15]
- ***CPC/IP*** – další menší implementace TCP/IP stacku pro 8 bitové mikrokontroléry [14]
- ***GBA TCP/IP/PPP stack*** – TCP/IP stack s podporou PPP
- ***Picnic*** – hardwarový návrh s mikrokontrolérem firmy PIC a ethernetovým čipem obsahujícím software schopný zprovoznit webserver
- ***uC/IP*** – další implementace TCP/IP stacku pro mikrokontroléry [13]
- ***TinyTCP*** – nenáročná implementace IP, TCP a FTP
- ***OpenTCP*** – open source TCP/IP stack určený pro embedded systémy
- ***CMX Micronet*** – komerční TCP/IP stack
- ***LiveDevices*** – komerční TCP/IP stack
- ***NicheStack*** – komerční TCP/IP stack poskytovaný společností InterNiche
- ***Kadak KwikNet*** – komerční TCP/IP stack

4 Stellaris EK-LM3S8962

Při vývoji ukázkové aplikace, která je součástí této diplomové práce byl využit vývojový kit poskytovaný společností Texas Instruments, Stellaris EK-LM3S8962. Jedná se o vývojový kit vhodný k vývoji prototypů embedded zařízení, vhodný pro návrh softwaru i hardwaru konečného zařízení.

Vývojový kit obsahuje ARM® Cortex™ M3 mikrokontrolér spolu s řadou externích periferií. Jmenovitě pak OLED displej, vstupní tlačítka, CAN rozhraní, Ethernetové rozhraní, výstupní diody, malý reproduktor a další. Všechny vstupy i výstupy mikrokontroléru jsou navíc vyvedeny tak, aby bylo možné k nim připojit vnější signály či sondy. Rozložení základních částí vývojového kitu je znázorněno v následující ilustraci.



Ilustrace 12: Rozložení částí vývojového kitu Stellaris LM3S8962

4.1 Code Composer Studio

Při koupi vývojového kitu Stellaris EK-LM3S8962 dostane uživatel k dispozici licenci pro používání vývojového prostředí Code composer studio (dále jen CCS).

4.1.1 O vývojovém prostředí

Jedná se o pokročilé vývojové prostředí s přátelským grafickým rozhraním. Do CCS je možno také importovat projekty z vývojového prostředí Eclipse. CCS je navrženo tak, aby co nejvíce urychlilo proces vývoje softwaru, jeho testování a analyzování chování mikrokontroléru v reálném čase. Výhodou vývojového kitu Stellaris je také fakt, že kromě vývojového

prostředí dostane k dispozici také řadu ukázkových projektů, které názorně ukazují využití jednotlivých periferií mikrokontroléru.

Detailní popis vývojového rozhraní CCS je nad rámec této diplomové práce a pokud má čtenář o tyto informace zájem, je možné je dohledat v dokumentaci samotného CCS či v tutoriálu dodávaném s vývojovým kitem Stellaris, kde je detailně popsáno využití jak ukázkových aplikací tak využití vývojového prostředí za účelem vytvoření vlastního projektu za využití dostupných knihoven potřebných k jednoduchému ovládání periferií mikrokontroléru a rychlému vývoji výsledné aplikace.

4.1.2 Code Composer Studio – často kladené otázky (FAQ)

I když jsou ve zmíněných dokumentech dostupné skoro všechny potřebné informace k seznámení se s vývojovým prostředím, je nasnadě zmínit alespoň několik informací, které by mohli uživatele neseznámeného s vývojem projektů pro vývojový kit Stellaris v začátcích zaskočit. Je zřejmé, že každý uživatel se s počátky vývoje potýká po svém a zaregistruje jiné postřehy a nesnáze. V této kapitole se pokusím tedy podělit o postřehy, které by z mého pohledu mohli být užitečné.

4.1.2.1 Linkování souborů

Pokud jste vytvořili knihovnu v jiném projektu a chcete ji znovu využít v novém projektu nebo jednoduše chcete využít dostupnou knihovnu třetí strany pak není třeba knihovnu kopírovat do adresáře vašeho projektu a popřípadě vytvářet složitou adresářovou strukturu pokud se jedná o více propojených souborů. Jednoduše stačí připojit požadovanou knihovnu kliknutím pravým tlačítkem myši na název projektu, vybrat položku *Link Files to Project* a potřebné soubory tímto způsobem k projektu připojit.

4.1.2.2 Nastavení velikosti zásobníku při zprovozňování webserveru

Snažíte-li se zprovoznit webserver na mikrokontroléru LM3S8962, je možné, že se vaše snažení dospěje do bodu, kdy nebudete moci najít žádnou chybu ve vašem kódu a i přesto se mikrokontrolér po prvním přístupu k webové prezentaci dostane do nekonečné smyčky jedné z chybových rutin a přestane reagovat na jakékoli vnitřní či vnější podněty.

Chcete-li mikrokontrolér donutit, aby se choval jako webserver, je nutné zvětšit velikost zásobníku ve vlastnostech projektu. Klikněte pravým tlačítkem na název projektu a vyberte volbu *Properties*. V levém sloupečku vyberte *Basic Options* u položky *TMS470 Linker*. Hodnotu pole *Set C system stack size(--stack_size, -stack)* nastavte na 1024. Proveďte nový build projektu a vše by mělo být v pořádku.

4.2 ARM® Cortex™ M3

Procesory ARM® Cortex™ řady M jsou navrženy speciálně pro aplikace kde je kladen důraz na nízkou cenu a nízkou energetickou náročnost. Tento procesor je nástupcem velmi úspěšného procesoru ARM7, který byl vyráběn s nákladem vyšším než jedné miliardy ročně a byl využíván širokou paletou světoznámých výrobců v různých aplikacích, od mobilních telefonů až po automobilový průmysl. Jen pro představu, řada mikrokontroléru Cortex™ M3 byla představena společností ARM v roce 2006 [16].

4.2.1 Výhody Cortex™ M3 architektury

Jak již bylo řečeno, nezanedbatelnými výhodami Cortex architektury je vysoký výkon v závěsu s nízkou cenou. Nejsou to však jediné výhody, díky kterým tato architektura vyčnívá. Nespornou výhodou je také právě jednoduchost tvorby aplikací, která bývá jedním z nejdůležitějších kritérií při výběru mikrokontroléru. Instrukční sada *Thumb-2* umožňuje programátorovi velmi jednoduše tvořit efektivní strojový kód bez nutnosti znalosti samotné architektury mikrokontroléru za použití programovacího jazyka C bez použití assembleru a přístupu k systémovým registrům. Usnadnění vývoje programátorům sníží dobu uvedení produktu na trh, sníží náklady na vývoj a samozřejmě také zvýší konkurenceschopnost. Tyto vlastnosti jsou v dnešní době základními stavebními kameny úspěchu.

Cena za mikrokontrolér je dána cenou výrobních technologií a jelikož moderní výrobní technologie v čistém prostoru jsou velmi drahé je zapotřebí k nízké ceně přispět miniaturizací a vysokým nákladem vyráběných zařízení. Cortex™ M3 procesory jsou vyráběny technologií 0,18 μm a je v nich z důvodu úspory místa na křemíku aplikováno jádro čítající pouhých 33 tisíc tranzistorů [21]. Jedná se o nejmenší ARM procesorové jádro, které je v dnešní době v průmyslu vůbec použitelné. Za účelem úspory místa není uzpůsobeno pouze jádro, ale také k němu přidružené periferie.

4.2.2 Základní rysy

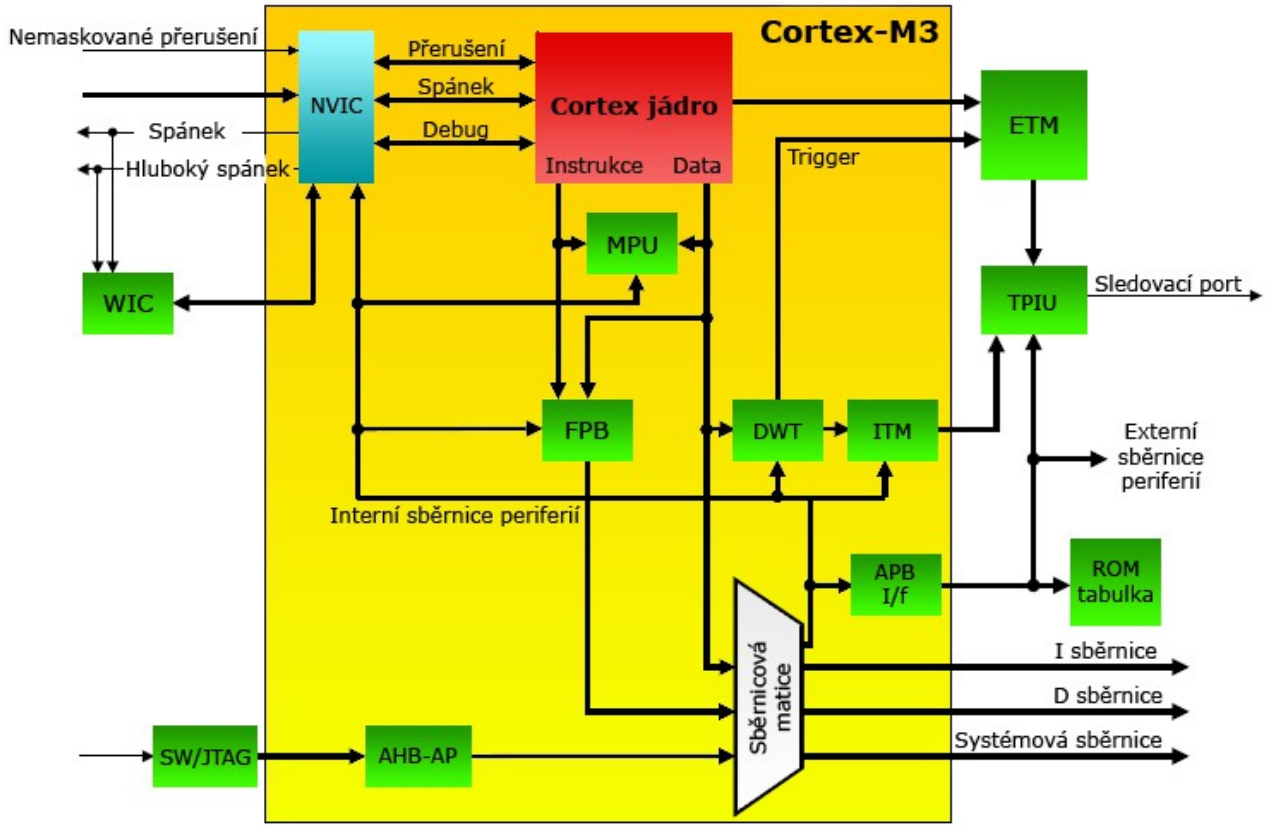
- **RISC procesorové jádro**
  - vysoce výkonná 32-bitová jednotka CPU
  - 3 stupňový pipeline s nízkou latencí
- **Instrukční sada**
  - optimální směs 16-bitových a 32-bitových instrukcí označovaná jako Thumb-2
  - velikost kódu o 30% menší v porovnání s 8-bitovými zařízeními
- **Harvardská architektura**
  - sběrnice oddělené pro data a pro instrukce
- **Jednoduchý přechod z ARM7 mikrokontroléru na Cortex™ M3**
- **Režimy spánku**
  - až 240 přerušení schopných probudit mikrokontrolér z režimu spánku
  - 3 různé režimy spánku
- **Hardwarové dělení a násobení**
  - možnost hardwarového dělení a násobení v jednom taktu
- **Vysoká konektivita**
  - UART / USB
  - Ethernet
  - CAN
- **Správa přerušení – NVIC (Nested Vectored Interrupt Controller)**
  - nízká latence přerušení

- teoreticky až 256 úrovní priorit přerušení
- hardwarově realizovaná přerušení
- možnost nemaskovaných přerušení pro kritické úlohy
- NVIC je účelně integrován s režimy spánku

- **Snadné možnosti ladění**
  - JTAG a Serial-Wire Debug porty
  - Watchpointy
  - Breakpointy
  - Trace point analyzátor k testování stavů vstupů a výstupů mikrokontroléru
- **a mnohé další vlastnosti**

4.2.3 Blokové schéma

Na obrázku uvedeném níže je znázorněna architektura jádra mikrokontroléru založeném na bázi Cortex™ M3. Jsou zde znázorněny jednotlivé komponenty, ze kterých se jádro skládá, jejich hierarchie a implementace samotného procesoru.



Ilustrace 13: Blokové schéma jádra Cortex M3

- **Procesorové jádro** – jádro architektury ARMv7-M
- **NVIC** – konfigurovatelný blok pro správu přerušení, blízce spjatý s procesorovým jádrem.

- **Sběrnicová matice** – spojuje procesor a debugovací rozhraní s externími sběrnicemi uvedenými níže.
- **I sběrnice** – instrukční a vektorová sběrnice pro instrukce a vektory z programového prostoru.
- **D sběrnice** – sběrnice pro data a debugovací přístup k programovému prostoru.
- **Systémová sběrnice** – sběrnice pro instrukce, data a debugovací přístup k systémovému prostoru.
- **Interní sběrnice periferií** – interní sběrnice pro data a debugovací účely.
- **Externí sběrnice periferií** - externí sběrnice pro data a debugovací účely.
- **WIC (Wake-up Interrupt Controller)** – konfigurovatelný řadič přerušení, pokud se systém nachází v některém z režimů spánku.
- **MPU** – jednotka zajišťující správu oprávnění k zápisu do paměťových členů.
- **DWT (Data Watchpoint and Trace)** – možnost nakonfigurovat komparátory, které při shodě s daty spustí signál *trigger*. Slouží pro debugovací účely.
- **ITM** – umožňuje softwarové nebo hardwarové trasování. Výstupem jsou pakety směřované k debugovacímu rozhraní.
- **APB** – rozšiřující sběrnice periferií
- **FPB** – možnost nakonfigurovat hardwarové breakpointy.
- **ETM** – makrobuňka umožňující trasování instrukcí.
- **TPIU** – tato jednotka funguje jako můstek mezi ITM a ETM pokud jsou implementované a sledovacím portem pro debugovací účely.
- **ROM tabulka** – tabulka obsahující adresy jednotlivých paměťových prvků v systému.
- **AHB-AP** - volitelně implementovatelný debugovací port umožňující přístup ke všem paměťovým registrům.
- **SW/JTAG** – konfigurovatelná jednotka zpřístupňující debugovací rozhraní. Má přístup ke všem registrům a paměťovým prvkům v systému včetně procesorových registrů.

4.2.4 Třístupňový pipeline

K dosažení vysokého výkonu za zachování pracovní frekvence z důvodu zachování nízké spotřeby byl v jádrech Cortex™ M3 implementován 3 stupňový pipeline. Jedná se o paralelní zpracování instrukcí znázorněné na následujícím obrázku.

Ilustrace 14: Třístupňový pipeline

Před samotným zpracováním instrukcí je třeba instrukci vyzvednout z paměti, poté dekódovat a následně provést. Z tohoto důvodu byl v jádrech Cortex™ M3 implementován tento zřetězený systém zpracování instrukcí. Tento způsob vede k výraznému zrychlení jejich zpracování.

4.2.5 Správa přerušení – NVIC

Architektura Cortex™ M3 přistupuje jiným způsobem také ke správě přerušení. K této správě je v jeho architektuře vyčleněn konfigurovatelný blok NVIC[5], který je velmi blízce propojen s procesorovým jádrem. Tento blok je počátečně nastaven tak, že generuje nemaskovaná přerušení od 32 fyzických bloků, které je možno rozložit do 8 prioritních úrovní. Tento režim je vhodné použít pro kritická zpracování přerušení v náročných aplikacích.

Blok NVIC dále nabízí možnost konfigurace pro příjem libovolného počtu fyzických přerušení v rozsahu od 1 do 240, které je možno rozložit teoreticky až do 256 prioritních úrovní. NVIC obsahuje tzv. handler tabulky obsahující vektory přerušení, které obsahují odkaz na část kódu, která má být vykonána při daném přerušení.

Jelikož jsou přerušení obsluhována hardwarově, je zaručena menší latence vykonání přerušení a následného návratu k vykonávání předešlých instrukcí. Při volání přerušení je jednoduše adresa handleru přerušení, které bylo vyvoláno, zavedena pomocí instrukční sběrnice a dojde k přesunu v programové paměti. NVIC zároveň obsahuje tabulku obsahující vektory všech probíhajících přerušení. V případě volání více přerušení v jeden okamžik tak není nutné před každým přerušením zálohování dat ve stavových registrech a jejich opětovné obnovení ihned po skončení přerušení, ale k obnovení dat dojde během několika málo cyklů až po skončení posledního přerušení. Tento způsob přerušení je označován jako *dokování přerušení.* Je mnohem efektivnější a je ocenitelný pak především v kritických aplikacích, kde dochází k velkému počtu přerušení. Časová úspora oproti klasickým architekturám, které zpracovávají přerušení softwarově je pak patrná.

5 Ukázková aplikace

5.1 Úvod

Ukázkovou aplikaci, která je součástí této diplomové práce tvoří embedded zařízení (vývojový kit Stellaris), jehož srdcem je procesorem ARM Cortex-M. Jedná se o 32-bitový mikrokontrolér obsahující mimo standardních periferií také implementované ethernetové rozhraní. V mikrokontroléru je implementována fyzická vrstva ethernetu, vyšší vrstvy je pak třeba kontrolovat softwarově. Program je vytvořený v programovacím jazyce C za použití vývojového prostředí Code Composer Studio. Embedded zařízení je vybaveno ethernetovým rozhraním. Za účelem jeho využití byl implementován LwIP stack popsaný v jedné z předchozích kapitol.

5.2 Definované funkce embedded zařízení

V ukázkové aplikaci se podařilo implementovat všechny požadované funkce. Hlavním a také prvotním požadavkem bylo implementovat vhodný TCP/IP stack. V zařízení byl implementován LwIP TCP/IP stack popsaný v jedné z předchozích kapitol. Na tomto základu byli nadále vystavěny následující funkce embedded zařízení.

5 NVIC (Nested Vectored Interrupt Controller) – vložený vektorový řadič přerušení

5.2.1 IP nastavení

Aby bylo zařízení konkurenceschopné a v reálném světě sítí použitelné, bylo nutné umožnit jak manuální tak automatické přiřazení IP adresy zařízení, masky podsítě a brány sítě. Uživatel má možnost ručního nastavení IP adresy zařízení, masky podsítě a brány sítě pomocí tlačítek na embedded zařízení. Zároveň má však také možnost využít automatického přiřazení IP adresy od DHCP serveru. Pokud se zařízení nachází v DHCP režimu a ve stanoveném časovém intervalu IP adresu, masku podsítě a bránu sítě od DHCP serveru neobdrží, nastaví předem definované standardní IP nastavení.

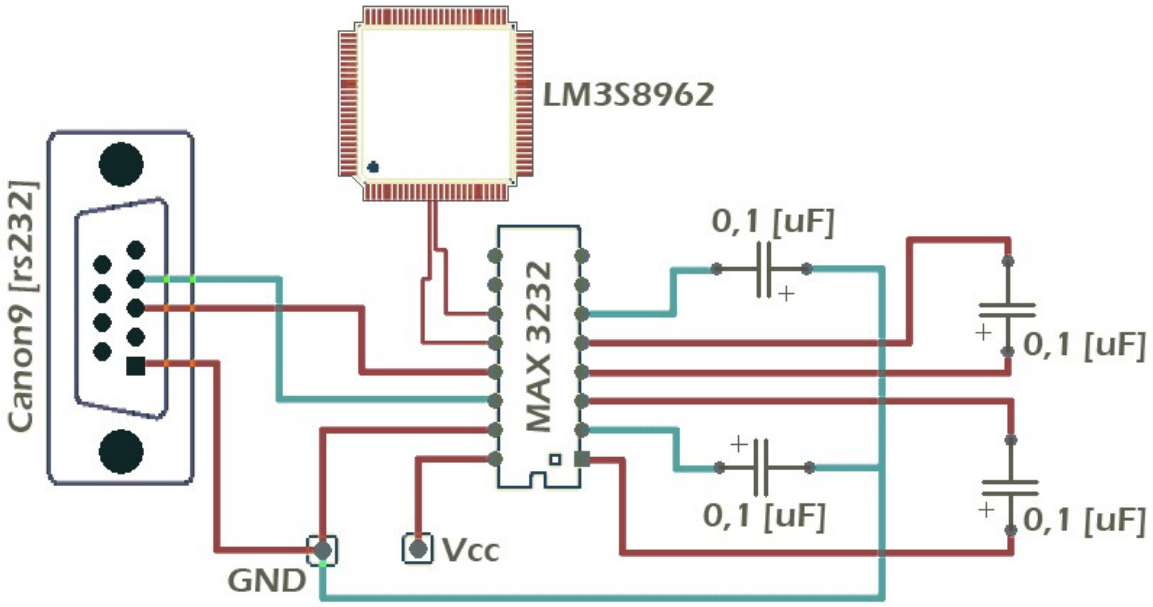
Zvolit režim statického či dynamického přiřazení IP adresy, masky podsítě a brány sítě je možné kromě místního nastavení pomocí tlačítek na embedded zařízení také pomocí webového rozhraní. Pokud je zvolena statická IP adresa zařízení, pak je možné přímo na webovém rozhraní zadat požadovanou IP adresu, masku podsítě i bránu sítě. Obě rozhraní jsou detailněji popsána v jedné následujících kapitol v této diplomové práci.

5.2.2 Převodník Ethernet-RS232

Ethernet-RS232 převodník slouží k převodu informací přenesených přes ethernetové rozhraní na sériové rozhraní a naopak. Funkce převodníku může být velice cenná pro celou řadu uživatelů. V průmyslu i mimo něj stále existuje mnoho zařízení, která komunikují pomocí sériového rozhraní. Na druhou stranu v dnešní době již velká většina počítačů, především přenosných, právě sériové rozhraní RS232 postrádá. Jedinou možností jak tuto absenci obejít je pak použít převodníku z USB rozhraní na RS232 nebo právě převodníku Ethernet-RS232. Nespornou výhodou převodníku Ethernet-RS232 však zůstává fakt, že může být přístupný z kteréhokoli místa, ze kterého je jeho IP adresa dosažitelná. Pokud tedy embedded zařízení bude disponovat veřejnou IP adresou, bude možné jej ovládat z jakéhokoli místa na světě, kde bude mít uživatel přístup k internetu. To se může zdát už jako opravdu ocenitelná změna v porovnání s 10 metrovým sériovým kabelem.

Aplikace umožňuje přenos informací přes ethernetové rozhraní buď pomocí UDP protokolu nebo pomocí TCP protokolu. Výhody a nevýhody jednotlivých protokolů byly již výše shrnuty v kapitole diskutující o referenčním modelu ISO/OSI, konkrétně popisující protokoly transportní vrstvy. Uživatel má tedy možnost volby a záleží pouze na konkrétní aplikaci, který z protokolů se uživatel rozhodne implementovat.

Co se týče sériového rozhraní, je využita periferie UART1. UART1 byl vybrán namísto periferie UART0, jelikož UART0 je využíván vývojovým prostředím zároveň pro debugovací účely. Veškeré sériové periferie mikrokontroléru Stellaris pracují v TTL napěťových úrovních. Pro převod na napěťové úrovně RS232 je tedy třeba použít nějaký druh napěťové translace. V ukázkové aplikaci byla využita napěťová translace pomocí obvodu MAX3232, která provádí translaci z TTL napěťových úrovní na úrovně RS232 a naopak. Zapojení je velice jednoduché, vyžaduje jeden integrovaný obvod a čtyři externí kondenzátory. Integrovaný obvod je napájen napěťovým zdrojem 3,3 V. Zapojení je naznačeno na následujícím obrázku.



Ilustrace 15: Napěťová translace TTL / RS232

S velmi malými modifikacemi tohoto zapojení je možné pak embedded zařízení přestavit z Ethernet-RS232 převodníku na Ethernet-USB převodník a to dokonce bez nutnosti změny programového kódu v mikrokontroléru za použití obvodu FTDI. Popřípadě pak s menšími úpravami hardwaru i softwaru je možné přestavit zařízení na Ethernet-RS485 či Ethernet-CAN převodník.

Zvolit protokol transportní vrstvy a IP adresu cílového zařízení, na které jsou data ze sériové linky přenášena může uživatel pomocí místní nabídky za pomocí tlačítek na embedded zařízení nebo přes webové rozhraní.

5.2.3 WebServer

Web server tvoří jednu z nejnáročnějších aplikací běžících na embedded zařízení. Z výše popsaného referenčního modelu ISO/OSI vyplývá, že je pro jeho funkci zapotřebí implementovat všechny vrstvy modelu. Fyzická a spojová vrstva je plnohodnotně implementováni již v mikrokontroléru Stellaris LM3S8962. Softwarově je pak pro funkci webserveru zapotřebí implementovat IP protokol síťové vrstvy, TCP protokol transportní vrstvy. Na transportní vrstvě webserver naslouchá na portu 80 a TCP spojení je zakládáno klienty, kteří se k webserveru připojují. Na aplikační vrstvě je pak implementován protokol HTTP/1.1.

Funkci webserveru v ukázkové aplikaci umožnila implementace knihovny, která je v podstatě součástí LwIP stacku a využívá jeho API. Tato knihovna byla vytvořena původně Adamem Dunkelsem a v dnešní době je i nadále součástí LwIP projektu, který je nyní vyvíjen širokou skupinou vývojářů.

5.2.3.1 Datové úložiště pro data poskytovaná webserverem

Kromě požadavků na TCP/IP processing a implementaci HTTP protokolu je třeba zajistit úložiště pro samotnou webovou prezentaci poskytovanou klientům. LwIP stack

implementovaný v mikrokontroléru Stellaris umožňuje dvě možnosti pro uchování webové prezentace poskytované webserverem:

- Externí paměť - SD karta
- Vnitřní paměť programu

Možnost uchování webové prezentace na externí SD kartě nabízí velké datové úložiště nicméně v přístupu k externí paměti se projeví znatelně větší latencí než při přístupu k vnitřní paměti programu. Naopak výhodou tohoto řešení je snadná možnost aktualizace webové prezentace. Za účelem aktualizace stačí aktualizovat soubory na SD kartě a není třeba přeprogramovat mikrokontrolér jako je tomu v druhém případě.

V ukázkové aplikaci byla aplikována druhá možnost, tedy uchování webové prezentace ve vnitřní paměti programu. Jednotlivé soubory webové prezentace bylo nejprve nutno převést na pole bajtů a tato pole následně uložit v paměti programu jako definované proměnné již při programování mikrokontroléru. Tato pole bajtů jsou pak odesílána namísto samotných souborů uložených v systému souborů na paměťové kartě. Výhodou je rychlejší zpracování vnitřní paměti. Nevýhodou tohoto řešení je omezené datové úložiště v porovnání s SD kartou a složitější způsob aktualizace webové prezentace.

5.2.3.2 Převod webové prezentace na pole bajtů za účelem uchování webové prezentace ve vnitřní paměti programu

K převodu webové prezentace byl využit veřejně dostupný program *makefsdata.* Jednotlivé verze LwIP TCP/IP stacku se od sebe mírně liší a knihovna webserveru taktéž. Nástroj *makefsdata* se ovládá z příkazové řádky. Jako parametr při volání nástroje *makefsdata* se zadává název adresáře, ve kterém se nachází systém souborů, který je zamýšleno převést na pole bajtů do výsledného souboru. Přiložený nástroj *makefsdata* vytváří výstupní soubor *fsdata.c*. Pro použití v současné verzi LwIP stacku implementované v ukázkové aplikaci je třeba soubor přejmenovat na *lmi-fsdata.h.* Tento soubor je třeba připojit k projektu a odstranit jeho první 3 řádky. Na konci souboru se dále nachází seznam několika proměnných typu struktura, ze kterých je zapotřebí odstranit poslední položku.

Nástroj *makefsdata* je možné použít na převod jak textových tak binárních souborů. V ukázkové aplikaci je názorně předvedena možnost zahrnutí obrázku do webové prezentace, *css* souboru obsahujícího kaskádové styly k nastavení vzhledu webových dokumentů a *favicon*.

5.2.3.3 Implementace TCP a HTTP

Webserver očekává příchozí TCP spojení na portu 80, na kterém naslouchá. TCP spojení je založeno klientem, který vysílá HTTP žádost typu GET. TCP zpracování bylo popsáno v jedné z předchozích kapitol zabývajících se implementací TCP protokolu v LwIP TCP/IP stacku. Informace o HTTP protokolu, formování žádostí a odpovědí byli podány v kapitole pojednávající o HTTP protokolu na aplikační vrstvě, která je součástí kapitoly pojednávající o referenčním modelu ISO/OSI.

Webserver přijímá požadavky od všech klientů. Přístup k webové prezentaci má tedy uživatel, který má přístup k IP adrese zařízení ze svého síťového umístění. Bude-li embedded zařízení disponovat veřejnou IP adresou, bude přístupné z kteréhokoli místa na světě, kde má uživatel přístup k internetu. Pro prohlížení webové prezentace může uživatel využít libovolný dostupný internetový prohlížeč. Webová prezentace byla bez potíží otestována v prohlížečích

Microsoft Internet Explorer, Mozilla Firefox, Google Chrome, Safari, Konqueror a Opera, včetně její mobilní verze.

5.2.3.4 SSI - Server Side Include

Server Side Include je rozšířením klasického webserveru, které umožňuje generování dynamického obsahu webových prezentací poskytovaných mikrokontrolérem. Namísto statických webových prezentací se tak dá do webových dokumentů zahrnout například obsah proměnných, paměti, stav embedded zařízení, IP adresa, maska podsítě či adresa brány. Je také možné využít funkcí a procedur implementovaných v mikrokontroléru a výsledky nechat zobrazit ve webové prezentaci.

Ke vkládání dynamických informací do poskytovaných webových dokumentů dochází při jejich odesílání přes TCP protokol. Webové dokumenty jsou při odesílání zároveň čteny a pokud program narazí na definovaný tag uvnitř dokumentu. Dokumenty obsahující SSI tagy musí mít příponu *.shtm, .shtml* nebo *.ssi* a SSI tagy musí být uváděny ve formátu *<!--#jmenoTagu-->*.

Zkušenému uživateli jistě neunikne, že formát odpovídá formátu komentáře v HTML dokumentech. Na tomto místě je vhodné zmínit, že dynamické informace nejsou zaměňovány za tyto tagy, ale jsou vkládány za ně. Jelikož se jedná o formát komentáře tak tagy ve výsledném dokumentu nejsou viditelné, ale jsou viditelné ve zdrojovém kódu.

Zároveň je ovšem nutné dodržet umístění tagů v HTML kódu. Není možné vkládat tagy do vnitřku jiných HTML tagů jinak by došlo k poškození konzistence HTML kódu. Nevýhodou tohoto řešení může být v některých případech například nemožnost vložení dynamické informace do parametru *value* v HTML tagu *input*. Pokud by tedy bylo zapotřebí vložit dynamickou hodnotu do nějakého HTML formulářového pole, bylo by zapotřebí využít sofistikovanějšího způsobu, pravděpodobně za použití javascriptu.

SSI funkci je zapotřebí povolit odkomentováním následujícího řádku v souboru lwipopts.h:

```
#define INCLUDE_HTTPD_SSI
```

Dále je nutné definovat pole SSI tagů v poli **ssi_tags[]* a SSI handler, tedy funkci, která je volána pokud program narazí na uvedený SSI tag. Tato funkce vrací jako výsledek požadovanou dynamickou hodnotu, která je následně vložena do HTML dokumentu a odeslána klientovi. Odkaz na SSI handler je nastaven zavoláním inicializační funkce *http_set_ssi_handler()*. Jak může volání této funkce vypadat je uvedeno v následující ukázce kódu.

```
// definice SSI tagů
static char const *ssi_tags[] = { "Tag1", "Tag2", "Tag3" };

// definice SSI handleru
int ssi_handler(int iIndex, char *pcInsert, int iInsertLen)
{
   /* kód handleru */
}

// volání inicializační funkce
http_set_ssi_handler(&ssi_handler, ssi_tags, 3);
```

5.2.3.5 CGI - Common Gateway Interface

Common Gateway Interface je další rozšiřující funkcí webserveru, která umožňuje zpracovat data z webových formulářů. Spolu s funkcí *Server Side Include* popsanou výše tak dávají dohromady plně výkonný webserver, který je schopen plnohodnotné funkce i na embedded zařízení zcela bez omezení.

CGI funkci je zapotřebí povolit odkomentováním následujícího řádku v souboru lwipopts.h:

```
#define INCLUDE_HTTPD_CGI
```

Pokud v HTML dokumentu definujeme formulář, je třeba definovat jeho parametr *action*, který odkazuje na cílový dokument či funkci, která data z formuláře zpracovává. Cílový dokument či funkce pak obsahuje kód, který se postará o zpracování informací zaslaných metodou GET nebo POST. Metodu předávání dat z formuláře specifikujeme nastavením parametru formuláře *method.* V případě CGI zpracování je třeba parametr *method* nastavit na hodnotu *GET*. Data z formulářů tak budou obsažena v URL a embedded zařízení z něj tato data bude moci extrahovat k dalšímu zpracování.

Funkce, které se starají o zpracování těchto dat nazýváme CGI handlery. Ty je třeba nejprve definovat v proměnné *cgi_handlers[]*. Tato proměnná obsahuje uspořádané dvojice {*cílový_dokument, odkaz_na_handler}*. Cílový dokument je název dokumentu uvedeného v poli *action* v definici HTML formuláře. Tento dokument fyzicky neexistuje, ale jeho název je obsažen v HTTP žádosti i s daty formuláře při jeho potvrzení. Pokud webserver identifikuje název definovaného dokumentu v HTTP žádosti, extrahuje předávaná data z formuláře, zavolá příslušný CGI handler a data mu předá jako parametry. Odkazy na CGI handlery je třeba nastavit zavoláním inicializační funkce *http_set_cgi_handlers()*. Inicializace CGI funkce webserveru znázorňuje následující ukázka kódu.

```
// definice CGI handleru
static tCGI const cgi_handlers[] = { { "dokument.html" , &cgi_handler } };

// volání inicializační funkce
http_set_cgi_handlers(cgi_handlers, 1);
```

5.3 Uživatelské rozhraní

Uživatelské rozhraní dostupné k ovládání embedded zařízení sestává ze dvou hlavních částí:

- místní ovládání pomocí tlačítek na embedded zařízení a nabídky zobrazené na OLED displeji
- ovládání přes webové rozhraní

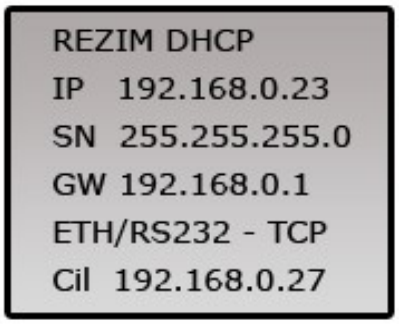
5.3.1 Místní ovládání

Místní ovládání disponuje několika úrovni nabídek umožňující základní nastavení funkce zařízení a zobrazení úvodní obrazovky, která zobrazuje aktuální stav zařízení základní informace o jeho aktuální konfiguraci. K ovládání slouží sada tlačítek. Nabídka a stav zařízení se zobrazuje na integrovaném OLED displeji. Rozložení jednotlivých prvků je patrné z následující ilustrace.

Ilustrace 16: Místní ovládání embedded zařízení

5.3.1.1 Aktuální stav zařízení

Nachází-li se zařízení v pohotovostním režimu a není-li zobrazena nabídka je na OLED displeji zobrazen aktuální stav zařízení se základními informacemi o aktuální konfiguraci. Tato obrazovka vypadá následovně.



Ilustrace 17: Aktuální stav zařízení

Na obrazovce uživatel může vidět režim přiřazování IP adresy, IP adresu zařízení, masku podsítě, IP adresu brány sítě, protokol vybraný pro funkci Ethernet-RS232 převodníku a cílovou IP adresu pro Ethernet-RS232 převodník, na kterou přeposílá data přijatá na sériovém rozhraní.

5.3.1.2 Hlavní nabídka

Do hlavní nabídky v embedded zařízení uživatel vstoupí stisknutím potvrzovacího tlačítka. Nabídka skrývá možnosti IP nastavení, nastavení Ethernet-RS232 převodníku a zobrazení aktuálního stavu zařízení. Hlavní nabídka je znázorněna v následující ilustraci.

Ilustrace 18: Hlavní nabídka

5.3.1.3 Nastavení IP

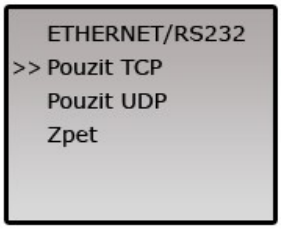
Zvolí-li uživatel v hlavní nabídce volbu *Nastavení IP,* zobrazí se mu možnosti nastavení režimu přiřazování IP adresy, masky podsítě a brány sítě. Uživatel může nastavit vlastní

hodnoty těchto položek a následně změny uložit a aplikovat, popřípadě se vrátit v nabídce zpět.

Ilustrace 19: Nastavení IP voleb

5.3.1.4 Nastavení Ethernet-RS232 převodníku

Pakliže uživatel v hlavní nabídce zvolí volbu *Ethernet/RS232*, dostane možnost vybrat požadovaný protokol pro funkci převodníku. Nabídka vypadá následovně.



Ilustrace 20: Nastavení převodníku

5.3.2 Sledování stavu a konfigurace přes webové rozhraní

Kromě možnosti sledování stavu a konfigurace embedded zařízení pomocí tlačítek a místní nabídky může uživatel využít webového rozhraní. Toto webové rozhraní je dostupné z jakéhokoli webového prohlížeče. K přístupu k webovému rozhraní stačí do řádku adresy zadat URL ve tvaru *http://ip_adresa_zařízení (např.: http://192.168.0.23)*.

Přes webové rozhraní uživatel může zkontrolovat stav zařízení a jeho konfiguraci nebo přímo zařízení konfigurovat. Zobrazení stavu a konfigurace zařízení je umožněno díky implementaci výše zmíněné SSI funkce webserveru. Možnost změnit konfiguraci je možná díky implementaci výše zmíněné CGI funkce webserveru. Následující obrázky znázorňují možnosti webového rozhraní.

Ilustrace 21: Nastavení IP přes webové rozhraní

Ilustrace 22: Nastavení převodníku přes webové rozhraní

5.4 Inicializace Ethernetového řadiče

Za účelem využití ethernetového rozhraní mikrokontroléru LM3S8962 je třeba inicializovat ethernetový řadič. Z registru ethernetového řadiče je vyčtena fyzická adresa ethernetového

řadiče, která je dále předána jako parametr spolu s počáteční IP adresou, maskou podsítě a IP adresou brány sítě funkci *lwIPInit()*.

V souboru *startup_ccs.c* je také zapotřebí specifikovat handler přerušení od ethernetového řadiče na *lwIPEthernetHandler.*

5.5 Implementace IP protokolu

IP protokol síťové vrstvy je zcela implementován jako součást LwIP TCP/IP stacku. Není tedy třeba žádných úprav kódu ani přidávání dodatečných rutin. Inicializací ethernetového řadiče zmíněné v předchozím bodě a zavoláním funkce *lwIPInit()* dojde k nastavení všech potřebných parametrů a handlerů. Ve chvíli kdy přijde IP paket je volána funckce *ip_input()* a další zpracovní IP paketů probíhá dle předpisů popsaných v kapitole pojednávající o LwIP TCP/IP stacku.

5.6 Implementace UDP protokolu

UDP je jednoduchý protokol určený pro přenos textových informací. Jeho implementace v ukázkové aplikaci je po bližším prozkoumání také velice jednoduchá. O inicializaci UDP protokolu se starají volání funkcí *udp_rx_init()* a *udp_tx_init()*. Obě funkce vytvoří a nastaví příslušný *udp_pcb (protocol control block)*.

Ve funkci *udp_rx_init()* je nastaveno naslouchání na příslušném portu od všech IP adres a s příslušným *udp_pcb* je provázán handler, který je volán pokud je přijat UDP datagram na příslušném portu. V příslušném handleru jsou následně zpracována přijatá data a pokud je UDP protokol využit k funkci Ethernet-RS232 převodníku jsou data odeslána na sériové rozhraní.

Funkce *udp_tx_init()* nastavuje port, ze kterého jsou odesílány UDP datagramy a dále nastavuje cílovou IP adresu, která však může být později změněna v konfiguraci přes webové rozhraní. K samotnému odesílání dat slouží funkce *udp_sendto()*, které se jako parametr předává příslušný *upd_pcb*, nově vytvořený *pbuf,* do jehož datové části jsou vložena data, která je třeba odeslat. Dalším parametrem je cílová adresa a cílový port.

5.7 Implementace TCP protokolu

TCP protokol je v porovnání s UDP protokolem již o poznání složitější. Tato složitost se také projevuje na obsáhlosti kódu, který pokrývá jeho obhospodařování. Kód tvořící implementaci TCP protokolu tvoří přibližně 50% kódu LwIP stacku. Kód v samotné ukázkové aplikaci není však natolik složitý.

Pro inicializaci je potřeba přinutit mikrokontrolér, aby naslouchal na požadovaném portu a přijímal příchozí spojení. K této inicializaci dochází ve funkci *tcp_rx_init(),* která zvolí port, na kterém očekává příchozí spojení, vytvoří příslušný *tcp_pcb* a nastaví požadovaný handler, který je volán pokud dojde k přijmutí TCP paketu na příslušný port. Handler se stará o zpracování přijatých dat, odeslání potvrzení o jejich správném přijetí a stará se také o ukončení spojení po přijetí posledního paketu.

Pokud jde o odesílání dat pomocí TCP protokolu, je inicializace spojení prováděna bezprostředně před odesláním dat a spojení je následně uzavřeno příjemcem po úspěšném přijetí všech dat. O navázání spojení s cílovým koncovým bodem se stará funkce *tcp_tx_init(),* kde se vytvoří požadovaný *tcp_pcb,* naváže se spojení s požadovanou cílovou IP

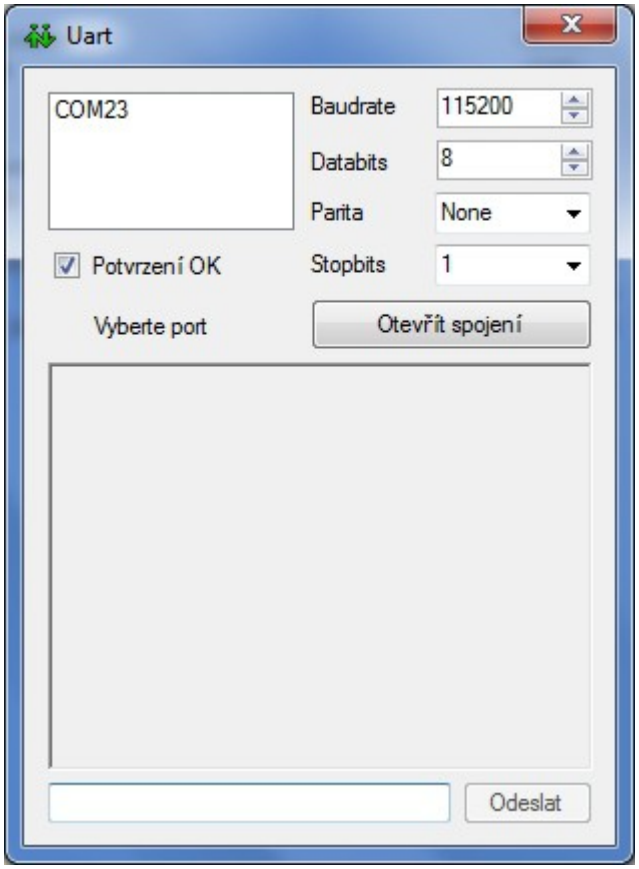
adresou na zadaném portu a nastaví handler *tcp_tx_connected()*, který je zavolán po úspěšném navázání spojení. Pokud dojde k úspěšnému navázání spojení, jsou odeslána požadovaná data uvnitř handleru tcp*_tx_connected()*.

5.8 Sada doplňkových aplikací

Ukázková aplikace je doplněna o sadu doplňkových aplikací vytvořených pro osobní počítač s operačním systémem Windows. Tyto doplňkové aplikace komunikují s embedded zařízením a je možné je využít k otestování komunikace a správnosti funkce zařízení. Aplikace byly vytvořeny ve vývojovém prostředí Microsoft Visual Studio 2008, v programovacím jazyce C# a pro svůj běh potřebují .NET framework, který je součástí většiny v dnešní době dostupných počítačů vybavených operačním systémem Windows.

5.8.1 Aplikace pro komunikaci přes sériové rozhraní

Pro testování Ethernet-RS232 převodníku byla navržena aplikace umožňující obousměrnou komunikaci přes dostupná sériová rozhraní. Aplikace je schopna detekovat dostupné COM porty, uživatel si může vybrat požadovanou konfiguraci komunikace a poté odesílat textové zprávy, popřípadě sledovat data přijatá přes sériové rozhraní. Grafické uživatelské rozhraní je znázorněno v následujícím obrázku.

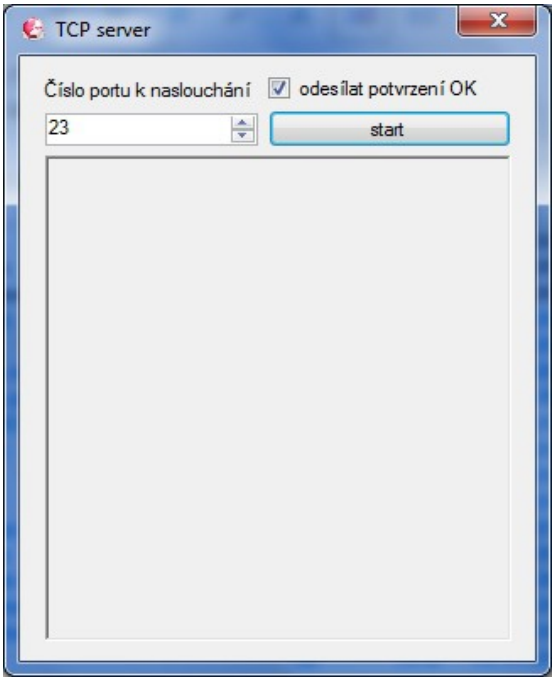


Ilustrace 23: Printscreen aplikace pro komunikaci přes sériové rozhraní

5.8.2 Aplikace pro naslouchání příchozích TCP spojení

Aplikace, která naslouchá na požadovaném portu akceptuje příchozí TCP spojení a vypíše přijatá data. Slouží k otestování správné funkce Ethernet-RS232 převodníku ve chvíli, kdy

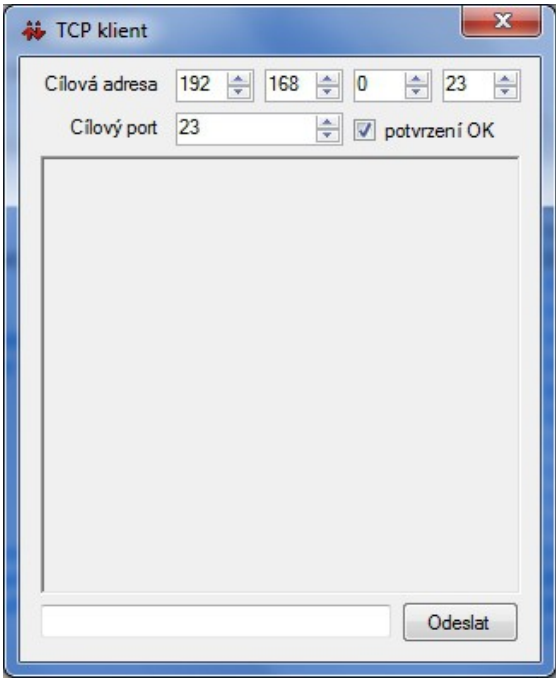
je převodník nakonfigurován pro přenos informací přes TCP protokol na straně Ethernetu. Vzhled grafického uživatelského rozhraní je zobrazen v následujícím obrázku.



Ilustrace 24: Printscreen aplikace akceptující příchozí TCP spojení

5.8.3 Aplikace vytvářející TCP spojení

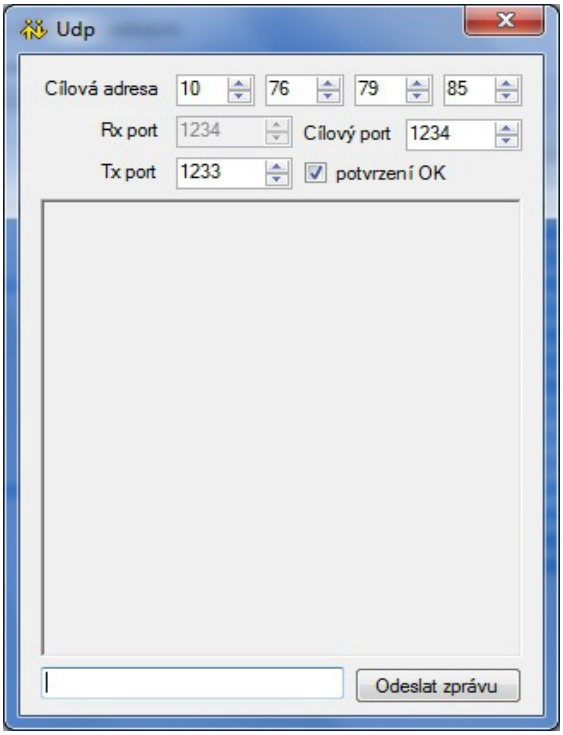
Další potřebnou aplikací k otestování Ethernet-RS232 převodníku je aplikace, která je schopna navázat nové TCP spojení a odeslat příjemci zadaná data. Vzhled grafického uživatelského rozhraní je zobrazen na následujícím obrázku.



Ilustrace 25: Printscreen aplikace navazující nové TCP spojení

5.8.4 Aplikace pro testování komunikace přes UDP protokol

Ethernet-RS232 převodník může kromě TCP protokolu využívat také UDP protokol. Za účelem otestování této komunikace byla postavena aplikace, která dokáže odesílat data na zadanou IP adresu a číslo portu a zároveň dokáže přijímat data a zobrazit je v grafickém rozhraní aplikace. Vzhled aplikace je znázorněn v následujícím obrázku.



Ilustrace 26: Printscreen aplikace pro testování komunikace přes UDP protokol

5.8.5 Internetový prohlížeč

Součástí sady doplňkových aplikací je také jednoduchý internetový prohlížeč využívající vykreslovací jádro prohlížeče Microsoft Internet Explorer. Vzhled uživatelského rozhraní této aplikace je znázorněn v následujícím obrázku.

Ilustrace 27: Printscreen internetového prohlížeče

6 Závěr

Jak ze shrnutých informací vyplývá, ethernet v dnešní době vyniká díky mnoha jeho přednostem. Velmi důležitou vlastností, za kterou je ethernet oceňován velkým množstvím příznivců je jeho všestrannost. Ethernet je možné využívat v kancelářských počítačích i malých embedded zařízeních nedisponujících zdaleka tak velkým výpočetním výkonem a tak může být ethernet implementován do systému řízeného jednoduchým a levným 8-bitovým mikrokontrolérem. Přes ethernetové rozhraní je možné přenášet krátké zprávy o délce několika bajtů až po obrovské soubory o velikosti několika gigabajtů. Jeho nespornou výhodou je existující sada propracovaných protokolů vyšších vrstev zajišťující bezpečný přenos informace a její adresování k požadovanému příjemci, který může sídlit ve stejné místnosti stejně jako na opačné straně zeměkoule. Ethernet implementovaný v embedded zařízení tak otevírá možnosti ať už snadného decentralizovaného řízení nebo vzdáleného sběru dat.

Skutečnost, že ethernet v jakékoli podobě je využíván s dlouholetou tradicí velmi usnadňuje návrh cílového embedded zařízení. Není třeba navrhovat hardwarová rozhraní či vytvářet a definovat pravidla komunikace od základů. Stejně tak jako u ukázkové aplikace postavené na vývojovém kitu Stellaris LM3S8962, která je součástí této práce, je možné využít i u mnoha dalších mikrokontrolérů již implementovanou fyzickou vrstvu referenčního modelu ISO/OSI a přitom zachovat nízkou koncovou cenu zařízení a samozřejmě také znatelně zkrátit vývoj konkrétní aplikace. Vyšší vrstvy je vždy třeba implementovat programově. Nicméně je možné využít volně šiřitelných implementací TCP/IP stacku tak, jak tomu bylo učiněno v ukázkové aplikaci.

7 Seznam příloh

[1] Zdrojové kódy ukázkové aplikace, k dispozici je celý projekt realizovaný ve vývojovém prostředí Code Composer Studio

[2] Zdrojové kódy webové prezentace implementované v projektu ukázkové aplikace

[3] Nástroj pro převod webové prezentace na bajtová pole implementovatelná v embedded webserveru

[4] Zdrojové kódy sady doplňkových aplikací

[5] Spustitelné verze sady doplňkových aplikací

[6] Dokumentace vývojového kitu Stellaris LM3S8962

8 Reference

[1] „Ethernet: Distributed Packet Switching for Local Computer Networks,“,Robert M. Metcalfe and David R. Boggs , Communications of the ACM, 7/1976, Volume 19, Number 7

[2] „Design and Implementation the LwIP TCP/IP Stack,“ Adam Dunkels, Únor 2001

[3] „Průmyslový Ethernet II: Referenční model ISO/OSI ,“ František Zezulka, Ondřej Hynčica, AUTOMA 3/2007, 86-90

[4] „An Ethernet Address Resolution Protocol,“ RFC 826, David C. Plumer, MIT, Listopad 1982

[5] „Internet Control Message Protocol,“ RFC 792, J. Postel, ISI, DARPA Internet Program Protocol Specification, Září 1982

[6] „Dynamic Host Configuration Protocol,“ RFC 2131, R. Droms, Bucknell university, Březen 1997

[7] „Hypertext Transfer Protocol,“ RFC 2616, R. Fielding a další, Compaq/W3C/MIT/Microsoft/Xerox, Červen 1999

[8] „Transmission Control Protocol,“ RFC 793, Information Science Institute, University of Southern California, Září 1981

[9] „User Datagram Protocol,“ RFC 768, J. Postel, Srpen 1980

[10] „Computer Networking: A Top-Down Approach,” Kurose, James F. & Ross, Keith W., 2007

[11] „Using the Stellaris Ethernet Controller with Lightweight IP (lwip),“ Texas Instruments, 2008-2010

[12] „Programming memory-constrained networked embedded systems,“ Adam Dunkels, Únor 2007

[13] „uC/IP stack,“ http://ucip.sourceforge.net/

[14] „CPC/IP stack,“ http://www.cepece.info/cpcip/

[15] „web51 stack,“ http://web51.hw-server.com/

[16] „Cortex-M3 Technical Reference Manual,“ ARM Limited, 2008

[17] „User Datagram Protocol,“ Wikipedia, The Free Encyclopedia, datum poslední midifikace Květen 2012

[18] „EK-LM3S8962 Firmware Development Package,“ Texas Instruments Incorporated, 2007-2010

[19] „Embedded Ethernet and Internet Complete: Designing and Programming Small Devices for Networking,“ Jan Axelson, Duben 2004

[20] „Embedded Ethernet,“ EmbeddedEthernet.com, datum posledního zobrazení Duben 2012

[21] „Úvod do architektury Cortex M3,“ Pandatron.cz, elektrotechnický magazín, Únor 2010