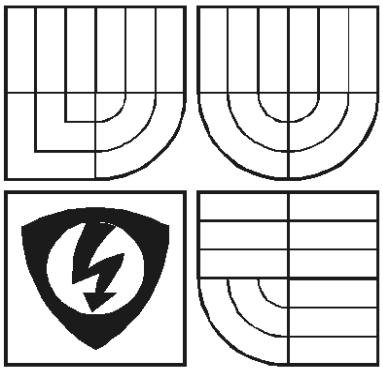
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav radioelektroniky



DÁLKOVÉ MONITOROVÁNÍ OPTICKÉHO BEZKABELOVÉHO SPOJE

bakalářská práce

Studijní obor: Elektronika a sdělovací technika

Jméno studenta: Jaroslav Martínek

Vedoucí bakalářské práce: Doc. Dr. Ing. Zdeněk Kolka

BRNO UNIVERSITY OF TECHNOLOGY

Faculty of Electrical Engineering and Communication

Institute of Radio Electronics

REMOTE MONITORING OF FREE-SPACE OPTICAL LINK

Bachelor Thesis

Study Specialization: Electronics and Communication
Author: Jaroslav Martínek
Supervisor: Dr. Zdeněk Kolka

ABSTRACT

Content of the Bachelor Thesis covers basic concept of test system, as well as the posibility of realisation of the testing laser head and the tester itself. As a communication protocol between the laser header and controling PC was choosen an UDP protocol for its modest requirements. As a controlling microcomputer in the laser header is used Rabbit RMC2200 for which a utility software was writen. It can read measured data and send it to a client computer over Ethernet network via mentioned UDP protocol. Also a web server software was programmed into RCM2200 for easy configuration of test systém over www interface. Because RCM2200 cannot generate accurate signal from timers, which makes it unaplicable for direct measuring, an FPGA circuir Xilinx Spartan 3 was used for it. At a client computer side was programmed monitoring software for operating systems Windows and Unix (Linux), which can receive data from the laser header and log it into file named by actual date. Windows version besides allows live monitoring of received data in form of text and graph.

Prohlášení

Prohlašuji, že svou bakalářskou práci na téma „Dálkové monitorování optického bezkabelového spoje“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny uvedeny v seznamu literatury na konci práce.

V Brně dne 3.6.2005 .....

Poděkování

Děkuji vedoucímu bakalářské práce Doc. Dr. Ing. Zdeňku Kolkovi za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé bakalářské práce.

V Brně dne ....

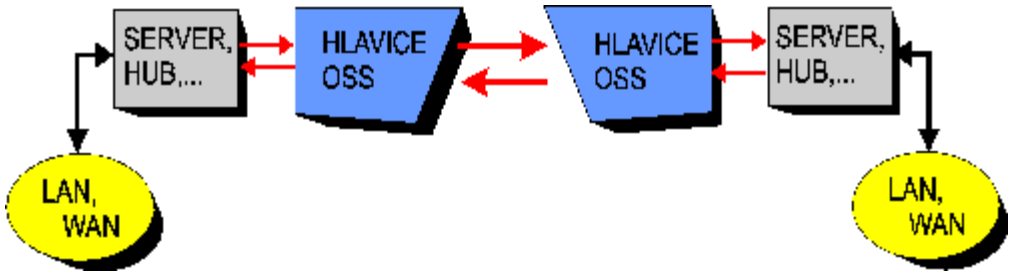
Obsah:

1. Úvod

V posledních letech došlo v souvislosti s rychlým rozvojem výpočetní techniky také k prudkému rozvoji datových komunikací. Ty je možno provozovat různými cestami, ať už klasicky kabelem, optickou linkou, či bezdrátově. Vždy je dobré znát a sledovat parametry těchto datových spojů. Velmi důležitým parametrem takového spoje je kvalita přenosu, reprezentovaná údajem chybovosti, který nám dává jasnou představu o funkci spoje a je možno jej využít pro statistické, dohledové i vývojové účely. Samozřejmě je také výhodné k nashromážděným datům přistupovat dálkově. Tím se zabývá tato bakalářská práce.
Celé zařízení bude složeno z tzv. Embedded-systému, ke kterému bude připojen A/D převodník a tester s obvodem FPGA. Také zde poběží serverová část (konfigurovatelná přes www rozhraní), která bude komunikovat s klientským PC některým z níže popsaných protokolů. Toto klientské PC (OS Windows nebo Unix) zpracuje přijatá data do souboru a případně do grafů tak, aby je bylo možno dále srozumitelně interpretovat.

1.1 Optické spoje

Jak bylo naznačeno výše, tyto spoje jsou velice perspektivní a to hlavně kvůli vysoké přenosové rychlosti a relativně malému rušení. Optický spoj je vždy tvořen dvojicí (případně čtveřicí, pokud je přijímací a vysílací zvlášť) identických hlavic, které realizují spoj typu bod-bod (peer-to-peer), přičemž mezi těmito hlavicemi musí být zajištěna přímá viditelnost. Jelikož se v podstatě jedná pouze o přenosový kanál, je přenos dat protokolově nezávislý a proto je možné takový spoj použít u široké škály datových sítí. V souladu s požadavky těchto sítí byly vyvinuty spoje pro přenosové rychlosti 10 Mbit/s, 16 Mbit/s, 155 Mbit/s a 622 Mbit/s, dosah je (podle typu) od stovek metrů do jednotek kilometrů [2].



Obr. 1.1 – umístění optického spoje v systému

V porovnání s rádiovými spoji mají optické spoje tyto výhody:

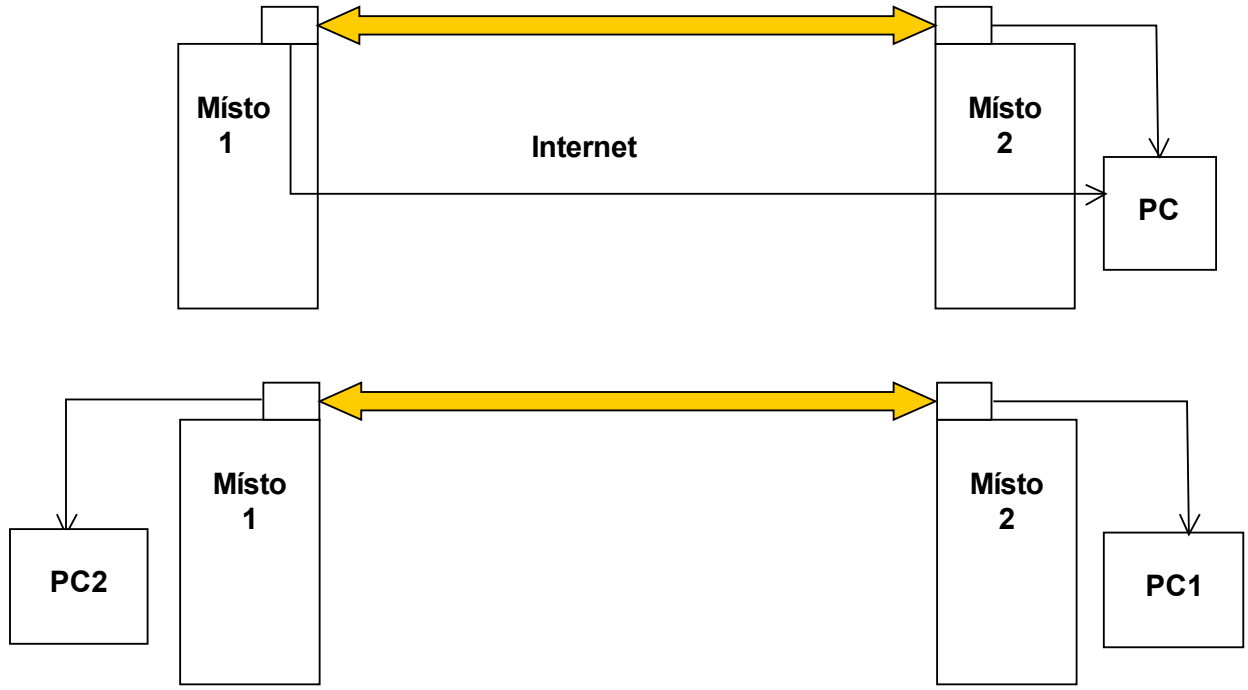
- Vysoké přenosové rychlosti
- Odolnost vůči rušení
- Legislativní volnost světelného spektra (pro optický spoj není třeba licence)
- Nepůsobí elektromagnetický smog
- Nelze je snadno odposlouchávat

Optické spoje mají ovšem i své nevýhody:

- Možné výpadky spoje při nepřízni počasí (hustá mlha)
- Je třeba je velmi přesně zaměřit
- Musí být pevně uchyceny
- Pouze peer-to-peer technologie
- Omezený dosah (exponenciální útlum atmosféry)

1.2 koncepce systému

Koncepce měřícího systému je znázorněna na obrázku 1.2. Na místo budoucího optického spoje (typicky vyvýšená místa, např. výškové budovy) se umístí dvě měřící hlavice, které budou průběžně monitorovat přijímaný výkon, proud diodou a chybovost spoje.

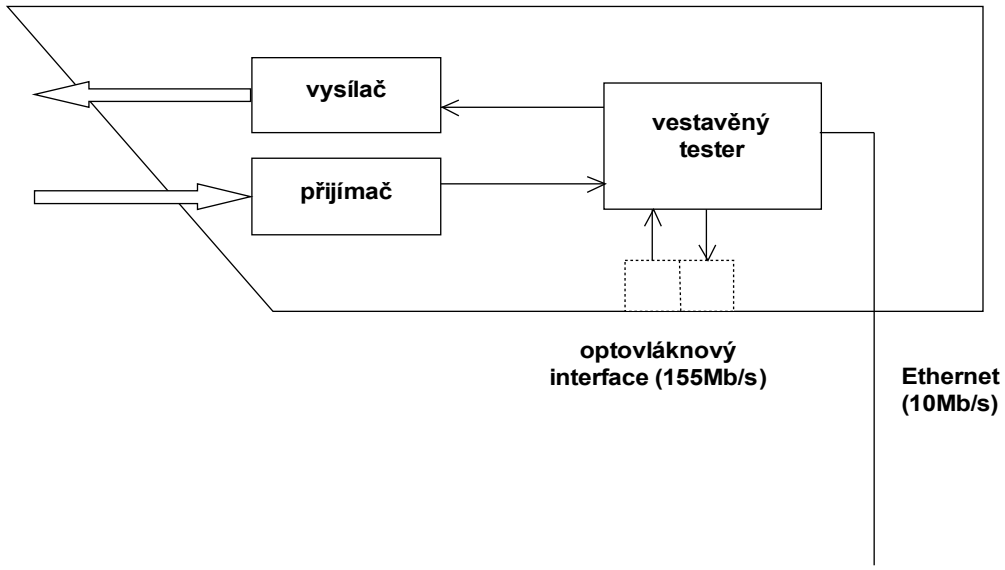


Obr. 1.2 – koncepce systému

Naměřená data jsou následně periodicky odesílána do řídícího počítače (PC), kde jsou ukládána pro další použití, případně vhodně vyhodnocována (tabulka, graf,...). Pro spojení měřících hlavic s PC přicházejí v úvahu dvě možnosti – buď bude každá hlavice připojena ke svému zvláštnímu PC anebo se budo data přenášet sítí Internet do společného PC.

1.3 Měřící hlavice

Měřící hlavice je nejdůležitější a nejkomplexnější součástí systému pro měření chybovosti. Její blokové schéma je znázorněno na obrázku 1.3.

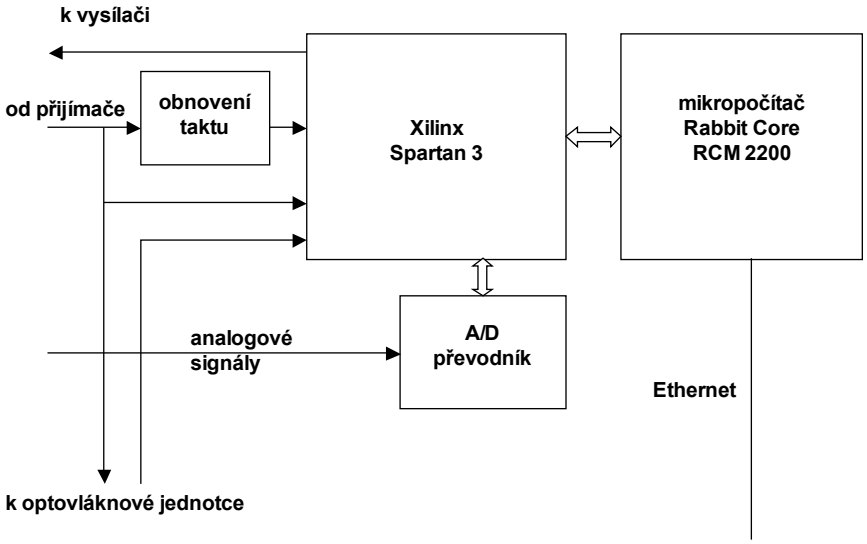


Obr. 1.3 – blokové schéma měřící hlavice

Optický vysílač slouží k převodu a úpravě digitálního signálu na modulaci vhodnou pro přenos světelným spektrem a k vysílání tohoto signálu přes vhodnou optiku vzduchem k protistraně, k čemuž se používá většinou laserových diod, nicméně v levnějších variantách je možno použít také „obyčejné“ LED anebo infračervené diody. **Přijímač** obstarává příjem světelného signálu a jeho zpětnou demodulaci na digitální signál. **Vestavěný tester** generuje měřená data pro vysílač, vyhodnocuje přijatá data z přijímače a převodníků a posílá je po 10Mbit Ethernetu do řídícího PC. Obsahuje také webserver pro pohodlnou konfiguraci hlavice. Hlavice může být také vybavena 155Mbit optovláknovým interfacem, pak může sloužit jako běžný optický spoj pro přenos reálných dat.

1.4 Tester

Vestavěný tester zajišťuje vlastní testování chybovosti datového spoje. Jak je vidět z blokového schématu na obrázku 1.4, funkci testování a sběru dat zajišťuje FPGA obvod Spartan 3 firmy Xilinx. Generuje potřebná data pro vysílač, čte digitální data z přijímače a z A/D převodníku, který slouží pro digitalizaci analogových veličin, jako je přijímaná úroveň signálu nebo proud vysílací diodou.



Obr. 1.4 – blokové schéma testeru

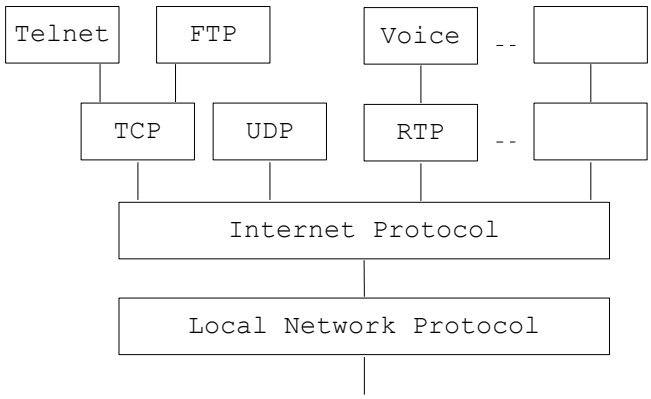
Naměřená data jsou dále z FPGA obvodu vyčítána mikropočítačem Rabbit Core RCM2200, který tato data posílá pomocí Ethernetového rozhraní do řídícího PC, řídí celý tester a zároveň zajišťuje jeho konfiguraci pomocí webového rozhraní.

Úkolem této práce je tedy:

- Vyřešit vhodně způsob odesílání naměřených dat do řídícího PC
- Umožnit konfiguraci hlavice přes www rozhraní
- Zajistit příjem naměřených dat na klientském PC a jejich vyhodnocení pro operační systémy Windows a Linux (Unix)

2. Popis komunikačních protokolů

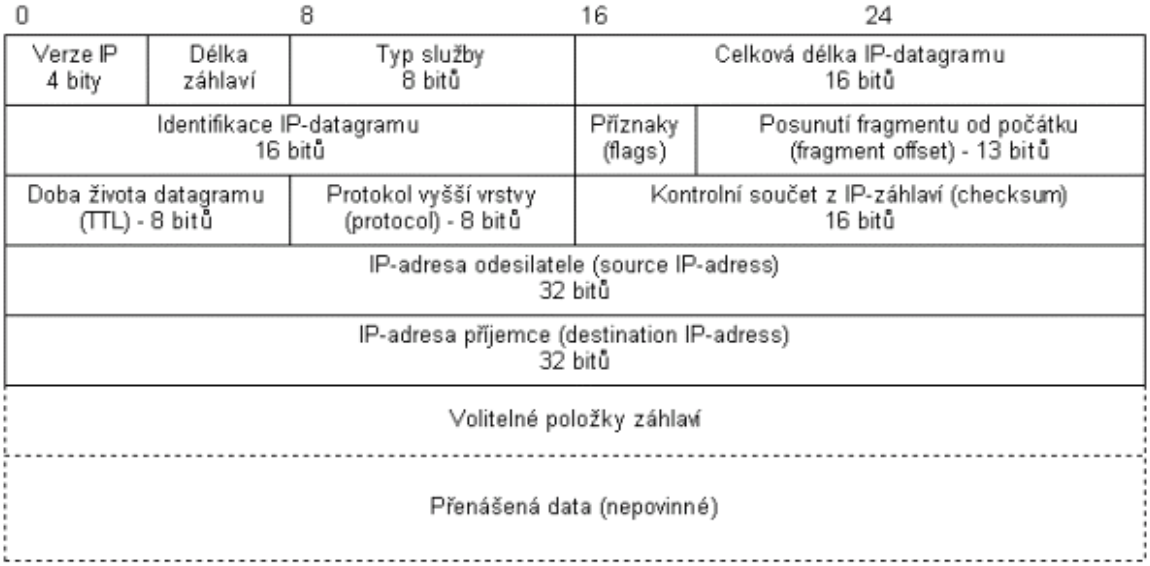
Aby bylo možno navázat komunikaci mezi počítači, je potřeba, aby spolu komunikovaly jednotnou "řečí". Proto byly vytvořeny komunikační protokoly, což jsou v podstatě přesně stanovené normy, dle kterých má komunikace probíhat. Tyto normy schvaluje Mezinárodní normalizační úřad (ISO), organizace ITU nebo IEEE. Na internetu jsou známé normy pod názvem Request For Comment (RFC). Dnes se používá velké množství protokolů, z nichž každý má své místo v hierarchii [3]:



Obr. 2.1 – Hierarchické umístění protokolů

2.1 Protokol IP

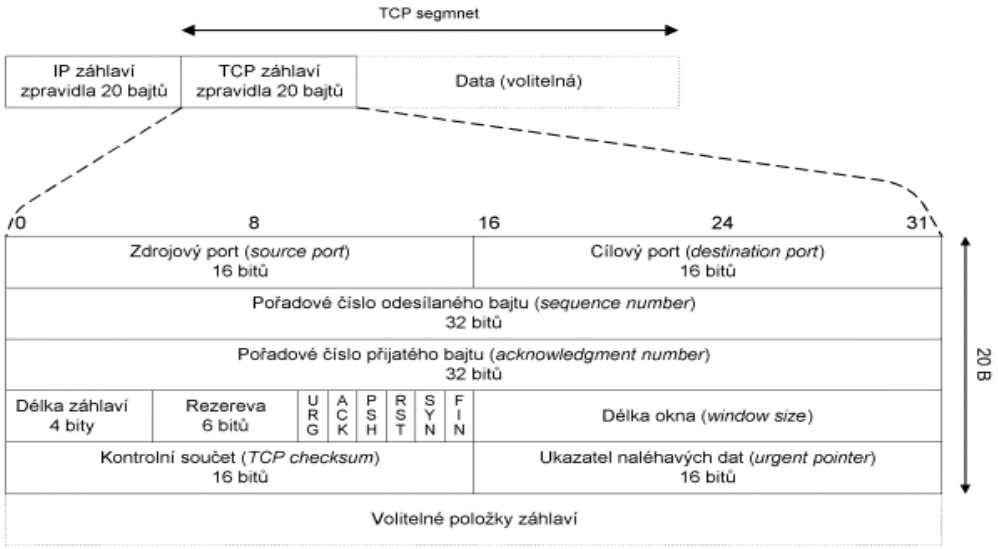
IP (Internet Protocol) je komunikační protokol, na kterém je dnes postaven Internet. IP protokol zajišťuje komunikaci dvou libovolných počítačů v internetu, tedy i přes různé LAN. Komunikace probíhá předáváním tak zvaných IP paketů, IP protokol definuje tento paket jako IP datagram, definuje také jeho přesný vnitřní formát. Každý IP datagram nese ve svém záhlaví úplnou adresu příjemce, díky čemuž je možno přenášet každý datagram samostatně a tyto tak mohou dorazit k adresátovi v jiném pořadí, než byly odeslány. Touto adresou příjemce se rozumí IP adresa, která musí být pro každý počítač jedinečná (ovšem jeden počítač může mít přiděleno více IP adres). IP adresa ve verzi protokolu 4 má 4 byte (ve verzi 6 má 16 byte). Na internetu se ovšem běžně jako adresa nepoužívá toto 4 byte-ové číslo, ale nějaký textový (=lépe zapamatovatelý) řetězec, který je nutné přeložit na IP adresu pomocí DNS serverů [1], [3].



Obr. 2.2 – Struktura IP datagramu

2.2 Protokol TCP

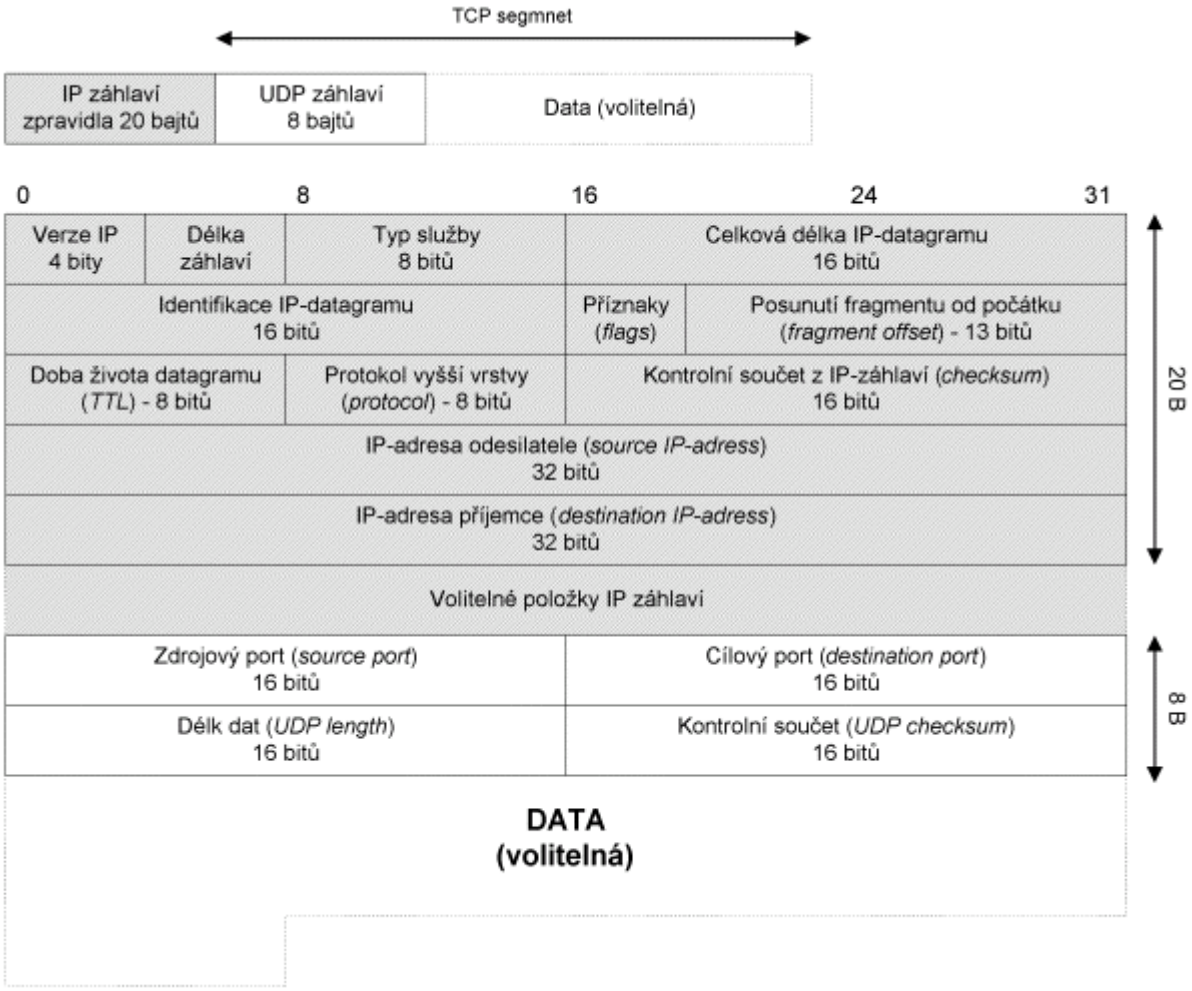
Protokol TCP (Transmision Control Protocol) je dnes asi nejpoužívanější. Jedná se o tak zvanou spojovou službu, což znamená, že před samotnou komunikací se naváže spojení, přičemž příjemce potvrzuje všechna přijímaná data. V případě ztráty dat sí příjemce vyžádá jejich opětovné zaslání. Na konec je nutné spojení ukončit (uzavřít). TCP segment obsahuje svou hlavičku a samotná data, která přenáší. TCP segment bude vložen do IP datagramu (jako data IP datagramu) a odeslán. Pokud je tento segment příliš velký, musí protokol IP provést fragmentaci. Adresou je v tomto případě tzv. port, jedná se o 2 byte-ové číslo a je součástí TCP hlavičky. Každá aplikace, která komunikuje pomocí TCP má přidělen svůj v rámci počítače jednoznačný port. Zjednodušeně lze říci, že zatímco IP protokol zajišťuje komunikaci dvou počítačů, tak TCP protokol zajišťuje komunikaci dvou aplikací na těchto počítačích. TCP port lze tedy považovat za jednoznačnou "adresu" aplikace na počítači. V případě navazování spojení pomocí protokolu TCP je třeba vždy uvést IP adresu a TCP port, čímž se určí s jakým počítačem a s jakou aplikací na něm se bude komunikovat [1], [4].



Obr. 2.3 Struktura TCP segmentu

2.3 Protokol UDP

Protokol UDP (User Datagram Protocol) je v podstatě alternativou k TCP. Jedná se o tak zvanou nespojovanou službu, což znamená, že nedochází k navázání spojení. Odesilatel pouze odešle data na stanovenou IP adresu a daný UDP port a dále se nezajímá o to, zdali data dorazila a zda se nepoškodila. Žádné potvrzení ani nic podobného nepřijde. UDP protokol je vhodný zejména v situacích, kdy by spojení pomocí TCP bylo velkou zátěží pro síť a kdy není třeba 100% zajistit korektnost přijímaných dat. Úspěšně se také využívá vlastnosti, že adresátem UDP datagramu nemusí být pouze jednoznačná IP-adresa, tj. síťové rozhraní konkrétního počítače. Adresátem může být skupina stanic (Broadcast), uplatnění tedy najde u různých streamovaných aplikacích (audio/video stream, např. aplikace RealMedia) atd. Stejně jako u TCP je součástí UDP hlavičky také port, jehož význam je stejný jako v případě protokolu UDP, ovšem jeho číslování je nezávislé na číslování TCP portů. Například aplikace může mít přiřazen TCP port 5000, ale UDP port 5000 může mít přiřazena zcela jiná aplikace [1], [5].



Obr. 2.4 – Struktura UDP datagramu

2.4 Protokol SNMP

Protokol SNMP (Simple Network Management Protocol) patří do rodiny tzv. Služebních protokolů. Slouží k výměně informací, které se člení na jednotlivé proměnné a jež jsou uspořádány do hierarchického stromu, tzv. MIB (Management Information Base), podobně jako např. adresářová struktura souborů na disku. Slouží ke komunikaci mezi managery a agenty na řízených uzlech sítě. Je to asynchronní, transakčně orientovaný protokol založený na modelu klient/server. Strana, která posílá požadavky (snmp klient), může být např. jednoduchý snmp browser či složitý NMS (Network Management Systém), na straně zařízení je snmp agent (snmp server), který na požadavky odpovídá. Výjimku tvoří tzv. trapy, které agenti vysílají asynchronně při výskytu jednotlivých událostí (výpadek proudu, větráku, překročení mezních údajů, objevení nového zařízení). Pro přenos dat se používá protokol UDP, přičemž je definováno přesně místo, kam se mohou připojovat uživatelské aplikace jednotlivých firem, které spravuje organizace IANA (Internet Assigned Numbers Authority). Pro komunikaci se využívá port 161, trap je odesílán na port 162.

SNMP ve verzích 1 a 2 používá následující příkazy:

- *get-request* -získání informace z MIB.
- *get-next-request* - umožňuje managerovi získat informace o objektech v MIB bez znalosti jejich přesných jmen, umožňuje postupné procházení celým hierarchickým stromem.
- *set-request* - změna hodnoty proměnné agenta.
- *trap* - vysílaný bez předchozího vyžádání, agent jej zasílá managerovi jako reakci na danou událost, zpráva zůstává nepotvrzená, proto agent nemá jistotu, zda byla doručena.
- *get-response* - agent vykoná tuto operaci jako reakci na předchozí příkazy - je to vlastně odpověď agenta managerovi. Odpověď obsahuje i dotaz, protože protokol nezajišťuje souvislost mezi dotazem a odpovědí.
- *get-bulk* - operace, která je součástí SNMP v.2. Umožňuje vyžádat si k přečtení celou skupinu informací z MIB, čímž se mnohdy urychluje komunikace.
- *inform* - umožňuje komunikaci dvou managerů mezi sebou.

MIB (Management Information Base) je databáze, která dovoluje jednoznačně identifikovat informace využívané systémem správy. Aby mohl SNMP manager i agent tyto informace získat a předávat, tak je nutná znalost struktury MIB [1], [6].

2.5 Protokol HTTP

HTTP (Hyper Text Transfer Protocol) je jednoduchý aplikační protokol vystavený nad protokolem TCP. Funguje na principu klient - server) dotaz – odpověď, přičemž jakákoliv aktivita musí být vyvolána klientem. Komunikace se serverem probíhá přes TCP (server většinou používá port 80, ale není to podmínkou). Úplný dotaz/odpověď musí mít specifikovánu metodu, URI (absolutní nebo relativní cesta k souboru nebo úplné URL dokumentu), verzi a hlavičky. Data jsou vždy od hlavičky odděleny prázdným řádkem, v případě dotazu se prázdným řádkem serveru oznamuje konec dotazu. V některých případech následuje po hlavičkách i tělo dotazu, opět oddělené jedním prázdným řádkem [1], [7], [10].

Příklad jednoduchého HTTP 1.1 dotazu:

```
GET / HTTP/1.1
Host: localhost
```

GET je metoda dotazu, / značí kořenový adresář serveru, HTTP 1.1 verzi protokolu. U této verze je povinné uvést hlavičku "*Host: server*". Dotaz je možno doplnit podmínkami, z nichž nejdůležitější jsou *If-Modified-Since*, *If-Unmodified-Since*, *If-Match*, *If-None-Match* a *If-Range*.
Odpověď serveru:

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Sat, 08 May 2004 13:39:47 GMT
Content-Type: text/html
Accept-Ranges: bytes
Last-Modified: Mon, 12 Apr 2004 20:30:02 GMT
ETag: "0180eecc20c41:7fc"
Content-Length: 107

<html>
...
</html>
```

Zde je vidět, že tělo odpovědi (v tomto případě HTML stránka) následuje také po hlavičce a po prázdném řádku.

Další používané metody dotazu jsou:

- *POST* - Pomocí této metody se dají v těle dopravit na server informace od uživatele (velmi často se POST používá pro odeslání rozsáhlejších dat z webových formulářů, pro upload souboru a podobně).
- *HEAD* - chová se stejně jako GET, ale v odpovědi se nepřenáší tělo, proto se hodí například ke zjištění, zda objekt existuje (při kontrole odkazů na stránce).
- *PUT/DELETE/MOVE/COPY* - jsou určeny k manipulaci se soubory. Aby bylo možné tyto metody použít, musí být povolen zápis do vybraného adresáře. V praxi se ovšem příliš nepoužívají.
- *OPTIONS* - slouží ke zjištění informací o daném kontextu (nebo "*" pro celý server). Klient může zjistit, které dotazy může na daný kontext zaslat.
- *TRACE* - se používá ke sledování cesty celého dotazu. V těle odpovědi jsou potom všechny dotazy jednotlivých systémů, kterými požadavek procházel. Tato metoda je používaná administrátory a webovými programátory, kteří chtějí zjistit, proč jim server vrací například prošlý (expirovaný) dokument apod.
- *CONNECT* - slouží k tunelování HTTP protokolu (například SSL).

Protokol HTTP ve verzi 1.1 dále definuje velké množství hlaviček pro dotazy i odpovědi:

Hlavičky v dotazech:

- *Accept* - Hlavičky tohoto typu indikují, co všechno je schopen klient zpracovat. Server pak vybere nejvhodnější alternativu. Patří sem hlavičky *Accept* (MIME typy dokumentů), *Accept-Charset* (znaková sada, v českém prostředí velmi důležité), *Accept-Encoding* (kódování přenášených dat, většinou slouží k výběru komprese) a *Accept-Language* (jazyk dokumentu).
- *Connection* - V protokolu HTTP 1.1 je definován parametr *close*, který požaduje okamžité uzavření spojení po přenosu prvního vyžádaného dokumentu.
- *Referer* - V této hlavičce je přenášeno URI stránky, ze které byl odkaz vygenerován.

- *Host* - Jelikož je u HTTP 1.1 možno provozovat více virtuálních serverů na jediné IP adrese, musí klient v této hlavičce specifikovat jméno serveru, s nímž chce komunikovat.
- *User-Agent* - Pomocí této hlavičky dojde k identifikaci klienta (prohlížeče), ať už ke statistickým účelům nebo pro přizpůsobení obsahu.

Hlavičky odpovědi:

- *Content* - Popisují tělo odpovědi. Mohou obsahovat například délku obsahu (*Content-Length*), jazyk (*Content-Language*), typ dokumentu (*Content-Type*) a další atributy.
- *Server* - Tato hlavička slouží serveru k vlastní identifikaci (obvykle zde najdeme jeho jméno, verzi a někdy i další informace).
- *Expires* - Server může prostřednictvím tohoto údaje sdělit, kdy vyprší platnost dokumentu. Po uplynutí této doby by si měl klient stáhnout novou verzi.

3. Programové řešení komunikace pomocí UDP

3.1 Řešení v systémech Windows

Pokud chceme v programu pro MS Windows používat sokety, je nejprve třeba inicializovat knihovnu soketů (winsock.dll) pomocí funkce `WSAStartup`. Tato využívá strukturu `WSADATA`, jež vypadá následovně:

- `WORD wVersion` - číslo verze WinSock. Ve vyšším bytě je umístěno vedlejší číslo verze (číslo podverze). V nižším bytě je umístěno hlavním čísle verze. Zde je vhodné si pomoci makry HIBYTE a LOBYTE.
- `WORD wHighVersion` - číslo maximální verze WinSock. Číslo verze udává maximální možnou verzi WinSock, která je použitelná pro dll soubor na lokálním počítači. Hlavní číslo verze a číslo podverze jsou umístěny stejně jako u předchozího atributu.
- `char szDescription[WSADESCRIPTION_LEN + 1]` - textový popis knihovny soketů. Jedná se o klasický C řetězec zakončený znakem s ASCII 0. Identifikátor `WSADESCRIPTION_LEN` je makro. Udává maximální počet znaků, který může mít popis.
- `char szSystemStatus[WSASYS_STATUS_LEN + 1];` - textový popis stavu Soketů. Opět se jedná o klasický C řetězec. Identifikátor `WSASYS_STATUS_LEN` je makro udávající maximální počet znaků řetězce.
- `unsigned short iMaxSockets;` - maximální počet soketů, které může aplikace (teoreticky) použít.
- `unsigned short iMaxUdpDg;` - maximální velikost jednoho UDP datagramu. Údaj je v bytech. Je-li 0, potom není velikost datagramu implementací soketů nějak omezená. O UDP datagramech si povíme něco v dalších dílech.
- `char FAR *lpVendorInfo;` - ukazatel na další informace o konfiguraci počítače.

Funkci `WSAStartup` předáme ukazatel na instanci této struktury, tato nám poté instanci naplní smysluplnými údaji. Hlavička funkce vypadá takto:

```
int WSAStartup(WORD wVersionRequested, LPWSADATA lpWSAData);
```

Prvním parametrem je číslo požadované verze. Dolní byte je hlavní číslo verze, horní byte je číslo podverze. Druhým parametrem je ukazatel na `WSADATA`. Zkratka LP znamená Pointer. Ukazatel ukazuje na alokovanou ale neinicializovanou (má náhodné hodnoty) instanci struktury `WSADATA`, tato bude po zavolání funkce naplněná. Funkce vrací v případě úspěšné inicializace číslo 0, poté jsme připravení pracovat se sokety.
Pro ukončení práce se sokety se (typicky na konci programu) zavolá funkce `WSACleanup()`.

Pro odeslání UDP paketu poslouží funkce sendto, která má následující zápis:

```
int sendto(SOCKET s, const char *buf, int len, int flags, const
struct sockaddr *to, int tolen)
```

Tato funkce odešle data daným soketem na danou adresu a port. Ukazatel `to` ukazuje na strukturu, kterou zaplníme IP adresou cílového počítače a číslem UDP portu aplikace na cílovém počítači. Při každém volání sendto můžeme jedním soketem posílat data na libovolné místo. Posledním parametrem je délka struktury, na kterou se odkazuje parametr `to`. Funkce vrací počet skutečně odeslaných bytů nebo hodnotu makra `SOCKET_ERROR` v případě chyby.

Pro příjem paketu slouží funkce recvfrom:

```
int recvfrom(SOCKET s, char *buf, int len, int flags, struct
sockaddr *from, int *fromlen)
```

Tato funkce přijme data daným soketem.

Vlastní realizace byla provedena ve vývojovém prostředí Borland C++ Builder 6.0 jako GUI aplikace (typické pro toto prostředí). Pro lepší používání bylo také použito překladu doménových jmen na IP adresy a naopak pomocí funkcí `gethostbyname` a `gethostbyaddr`.

3.2 Řešení v systémech Unix

Princip řešení v systémech Unix je ve své podstatě totožný, práce se sockety se pro naše účely liší pouze v několika drobnostech.

Je třeba se postarat o deklarace socketových funkcí. Není třeba sockety inicializovat ani s nimi ukončovat práci, tímto odpadá použití funkcí `WSAStartup` a `WSACleanup`. Struktura `hostent` je stejná jako ve Windows, liší se pouze typem atributů `h_addr_type` a `h_length`, které nejsou typu `short int` ale `int`.

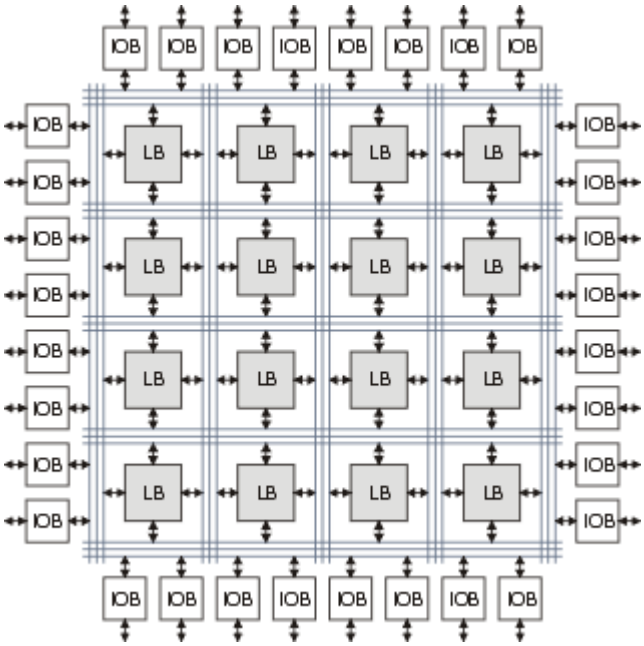
Program byl napsán jako konzolová aplikace, zkompilován kompilátorem g++. Program očekává jako parametr adresu a port cílového PC. Na něj odešle UDP paket čeká na příjem, takto se děje v cyklu 4x. Opět je pro zjednodušení zadávání použito překladu z doménových jmen na IP adresy pomocí funkce `gethostbyname` [11].

4. Xilinx Spartan

4.1 Úvod

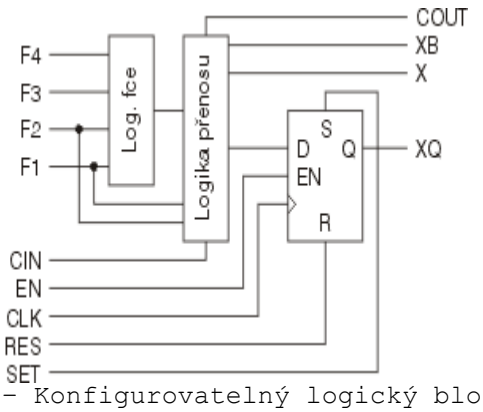
Všechny programovatelné součástky se souhrnně označují PLD, což znamená Programmable Logic Device. Číslicové programovatelné součástky je možné podle vnitřní struktury rozdělit do tří skupin. První skupinou jsou klasické PLD, druhou komplexní PLD a do třetí skupiny patří obvody typu FPGA.

Obvody typu FPGA (Field Programmable Gate Array) mají z programovatelných obvodů nejobecnější strukturu a obsahují nejvíce logiky. Současné největší obvody FPGA obsahují až 6 milionů ekvivalentních hradel (typické dvouvstupové hradlo NAND). Typickou strukturu obvodu FPGA znázorňuje následující obrázek.



Obr. 4.1 – Struktura FPGA obvodu

Bloky označené IOB (Input/Output Block) představují vstupně-výstupní obvody pro každý v-v pin FPGA. Tyto bloky obvykle obsahují registr, budič, multiplexer a ochranné obvody. Bloky LB (Logic Block) představují vlastní programovatelné logické bloky. Všechny bloky mohou být různě propojeny globální propojovací maticí. Nejpoužívanější struktura konfigurovatelného logického bloku je znázorněna na obrázku 4.2.



Obr. 4.2 – Konfigurovatelný logický blok

FPGA obvykle umožňují propojit některé signály logických bloků přímo se sousedním bez nutnosti využívat globální propojovací matici. Takovéto spoje mají mnohem menší zpoždění a umožňují tak realizovat například rychlé obvody šíření přenosu, což je nezbytné pro sčítačky nebo násobičky.
Kromě bloků znázorněných na předchozích obrázcích integrují výrobci do FPGA další prvky. Většina moderních FPGA obsahuje několik bloků rychlé synchronní statické paměti RAM. Velmi často obvody FPGA obsahují PLL (Phase Locked Loop) nebo DLL (Delay Locked Loop) pro obnovení charakteristik hodinového signálu, případně pro násobení nebo dělení jeho frekvence [12].

4.2 Xilinx Spartan 3

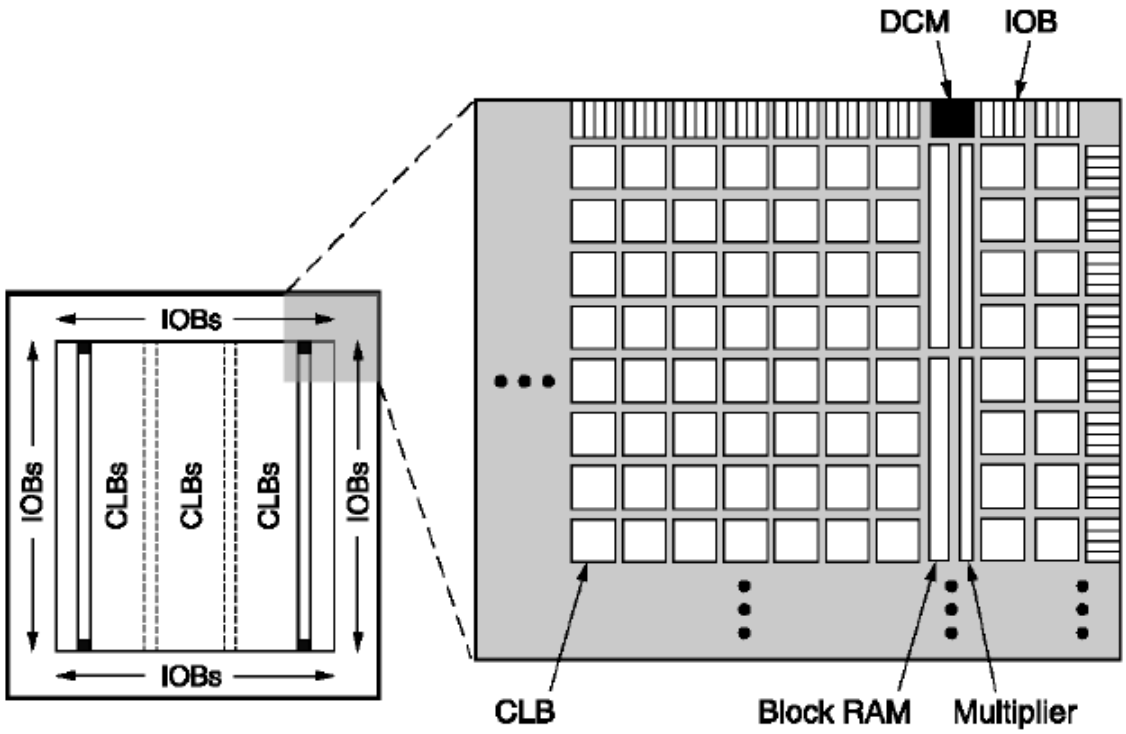
FPGA obvod Xilinx Spartan 3 je posledním z vývojové řady obvodů Spartan. Oproti předchozím řadám se liší v počtu systémových hradel a logických buněk, velikosti RAM, počtu I/O a implementací některých nových bloků do architektury jako jsou DCMs nebo násobičky. Počet systémových hradel se pohybuje od 50000 do 5000000.

Hlavní rysy architektury SPARTAN-3:

- nová 90nm technologie výroby
- až 5 miliónů systémových hradel nebo 74880 logických buněk
- hodinový signál až 326MHz
- 3 separátní napájecí zdroje: 1.2V , 2.5V , 3.3V
- až 784 I/O pinů, 622Mb/s přenosová rychlost pinu
- podpora až 17 úrovňových standardů
- Double Data Rate (DDR) podpora
- integrovaná struktura pro konstrukci rychlých sčítaček
- 4 až 104 integrovaných násobiček 18 x 18
- JTAG port pro testování
- až 1872Kb blokové RAM
- až 520Kb distribuované RAM
- až 4 Digital Clock Manager (DCM) umožňující frekvenční syntézu
- 8 globalních hodinových linek
- plná podpora vývojového prostředí Xilinx ISE

Architektura je složena z 5 základních funkčních/logických bloků:

- **IOBs** (Input/Output Blocks) řídí tok dat mezi I/O pinem a vnitřní logikou
- **CLBs** (Configurable Logic Blocks) obsahující LUTs (Look-Up Tables) na principu paměti RAM
- **Block RAM** umožňující ukládaní dat ve formátu 18Kb dual-port blocks.
- **násobičky** umožňující vynásobení dvou 18-bitových čísel
- **DCM** (Digital Clock Manager) poskytuje autokalibraci, plně digitální řešení distribuce zpoždění, násobení, dělení a fázový posun hodinového signálu



Obr. 4.2 – Uspořádání základních bloků v architektuře

SPARTAN-3 využívá pro konfiguraci paměťových buněk RAM a tedy při vypnutí napájení je daná konfigurace ztracena. Proto je potřeba zajistit nahrání konfigurace při každém zapnutí FPGA. Konfigurační data jsou automaticky čtena z externího zdroje dat (PROM, JTAG, FPGA) a to buď sériově nebo paralelně. Dále také umožňuje vybrat jeden ze 17 možných úrovňových standardů pro jednotlivé piny (single-ended) a jeden ze 6 pro rozdílový výstup (differential).

IOBs (Vstupně výstupní bloky)
IOB poskytuje programovatelné obousměrné rozhranní mezi I/O pinem a interní logikou FPGA. Struktura IOBs obsahuje 3 základní signálové cesty: vstupní, výstupní a 3-stavová.

- Vstupní cesta přenáší data z pinu do vnitřní logiky přes volitelné programovatelné zpoždění na linku I. Další možné výstupy jsou IQ1 a IQ2 přes klopné obvody.
- Výstupní cesta z vnitřní logiky vede přes linky O1 a O2 na výstupní pin.
- 3-stavový výstup je řízen linkami T1 a T2.

CLBs (Configurable Logic Blocks)
CLB je tvořena 4 LC (Logic Cell), které jsou uspořádány do dvojic (řezy) tak, že daná dvojce má společný carry přenos. Oba řezy obsahují dva logické funkční generátory, dva ukládací elementy, multiplexer, carry logiku. Těmito bloky jsou tvořeny logické, aritmetické a paměťové funkce. Levý pár má navíc další dvě funkce: ukládaní dat pomocí distribuované RAM a posun dat s 16-bitovým registrem.

Bloková RAM
Všechny obvody rodiny SPARTAN-3 obsahují blokovou RAM uspořádanou po blocích velikosti 18Kbit. Pro uložení velkého množství dat je efektivnější použít blokovou RAM než distribuovanou RAM. Kapacita paměti je 18432bitů bez parity nebo 16384bitů s paritou. Paměť také umožňuje dvouportový přístup – může být konfigurována jako Single-Port nebo Dual-Port RAM. Paměť je na čipu rozmístěna po blocích tvořící sloupce.
Paměť lze konfigurovat do různých „rozměrů“. Dle šířky datového slova odpovídá šířka adresové sběrnice a také podle typu paměti (Dual-Port nebo Single-Port).

Násobičky
Obvody SPARTAN-3 obsahují násobičky 18x18 bitů s 36bitovým výstupem. Mohou pracovat jako asynchronní i jako synchronní. Vstupní datové slovo může být reprezentováno dvojkovým doplňkem (buď 18bitové znaménkové nebo 17bitové neznaménkové). Kaskádním řazením lze vytvořit násobičky pracující s širším datovým slovem než jen 18bitů.

DCM (Digital Clock Manager)
DCM řídí, upravuje a distribuuje hodinový signál po celém čipu. Plní 3 základní funkce: eliminace časového zpoždění, frekvenční syntéza, fázový posun signálu. Všechny obvody obsahují 4 DCM (kromě XC3S50, ten má pouze 2 DCM).

Globální síť hodinového signálu
FPGA SPARTAN-3 obsahují 7 globálních hodinových signálů GCLK0-GCLK7. Ty vstupují do pole multiplexerůkde jsou přepnuty a pokračují do DCM nebo do globální sítě hodinového signálu. V DCM jsou upraveny a vedou zpět do pole multiplexerů a pokračují do globální sítě hodinového signálu.

Propojovací síť

- Long lines – vhodné pro distribuci globálního signálu s malým zpožděním, propojuje každou šestou CLB
- Hex lines – také vhodné pro vedení rychlých signálů s malým zpožděním, navíc umožňuje efektivnější propojení, protože jsou spojeny s každou třetí CLB
- Double lines – spojují každou druhou CLB s vysokou flexibilitou propojení
- Direct lines – propojuje nejbližší sousední CLB

Registry

Obvody Spartan 3 obsahují množství interních registrů, které kontrolují konfiguraci a vyčítání z obvodu. Jejich výčet je uveden v následující tabulce.

Tabulka 4.1 – Registry obvodu Spartan 3

Jméno	Označení	Read/Write	Binární adresa
Cyclic Redundancy Check	CRC	R/W	00000
Frame Address Register	FAR	R/W	00001
Frame Data Input Register	FDRI	W	00010
Frame Data Output Register	FDRO	R	00011
Command Register	CMD	R/W	00100
Control Register	CTL	R/W	00101
Mask Register	MASK	R/W	00110
Status Register	STAT	R	00111
Legacy Output Register	LOUT	W	01000
Configuration Options Register	COR	R/W	01001
Multiple Frame Write Register	MFWR	W	01010
Frame Length Register	FLR	R/W	01011
(Rezervováno)	–	–	01100
(Rezervováno)	–	–	01101
Product IDCODE Register	IDCODE	R/W	01110
Partial Reconfiguration Register	SNOWPLOW	W	01111

5. Modul Rabbit RCM2200

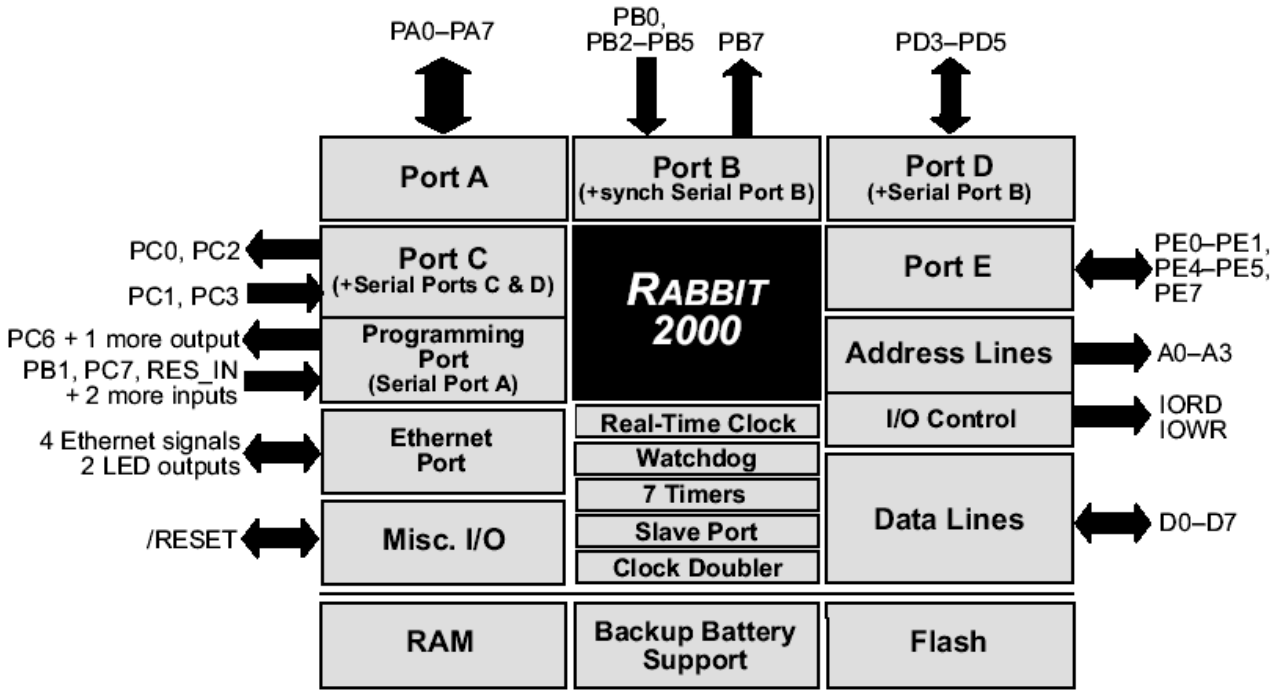
Obr. 5.1 – Vzhled modulu

Je to procesorový modul s mikroprocesorem Rabbit 2000 pracujícím na frekvenci 22,1MHz, 128kB statické RAM a 256kB Flash. Dále obsahuje dvoje hodiny, obvody pro reset a kontrolu zálohování hodin reálného času, I/O sběrnici, paralelní a seriové porty a ethernetový port, který činí tento modul vhodným pro internetové aplikace.

5.1. Vlastnosti modulu RCM2200

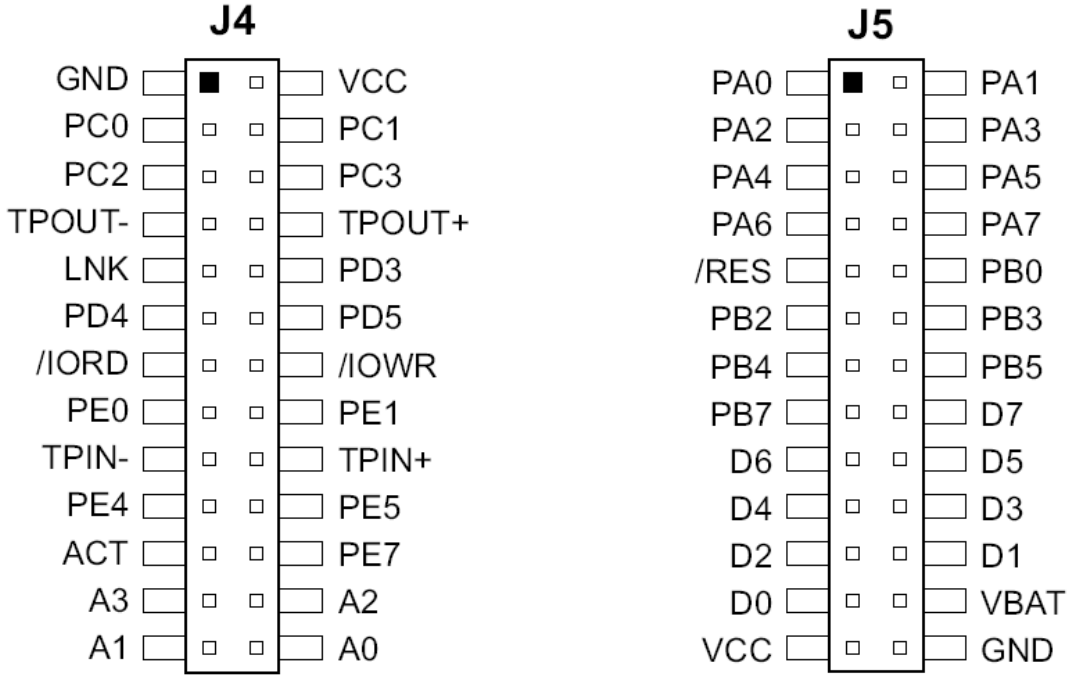
- Malá velikost: 41mm x 58mm x 22mm
- Mikroprocesor: Rabbit 2000 na 22,1MHz
- 26 paralelních
- 8 datových
- 4 adresní
- Paměťové vstupy/výstupy
- Vnější resetovací vstup
- Pět 8-bitových časovačů a dva 10-bitové časovače
- 256K-512K flash paměti, 128K-512K paměti SRAM
- Hodiny reálného času
- Watchdog
- Možnost uživatelské zálohovací baterie skrze připojení na konektoru J5
- 10Base-T RJ-45 Ethernetový port
- Tři CMOS kompatibilní seriové porty s maximální asynchronní přenosovou rychlosti 691200 bps a maximální synchronní přenosovou rychlostí 5529600 bps.

5.2. Hardware



Obr. 5.2 – Podsystémy modulu

Modul má 26 paralelních I/O vodičů seskupených v pěti 8-bitových portech na konektoru J4 a J5. 16 obousměrných I/O vodičů je umístěno na pinech PA0–PA7, PD3–PD5, PE0–PE1, PE4, PE5 a PE7. Více napoví následující obrázek.



Obr. 5.3 – Pinout konektorů J4 a J5

Tabulka 5.1 – Popisy a využití vývodů konektoru J4

Konektor J4				
Pin	**Jméno**	**Hlavní použití**	**Jiné použití**	**Poznámka**
1	GND			
2	VCC			
3	PC0	Výstup	TXD	
4	PC1	Vstup	RXD	
5	PC2	Výstup	TXC	
6	PC3	Vstup	RXC	
7	TPOUT-			Vysílací port sítě
8	TPOUT+			
9	LNK			Indikace připojení sítě
10	PD3	Paralelní programovatelné vstupy/výstupy		
11	PD4		ATXB výstup	
12	PD5		ARXB vstup	
13	/IORD	Vstup		
14	/IOWR	Výstup		
15	PE0	Paralelní programovatelné I/O	Ovládání I0 nebo vstup INT0A	
16	PE1		Ovládání I1 nebo vstup INT1A	
17	TPIN-			Přijímací port sítě
18	TPIN+			
19	PE4	Paralelní programovatelné I/O	Ovládání I4 nebo vstup INT0B	
20	PE5		Ovládání I5 nebo vstup INT1B	
21	ACT			Idikace aktivity sítě
22	PE7	Paralelní programovatelný vstup/výstup	Ovládání I7 nebo chip select slave portu /SCS	
23-26	A[3:0]			Adresní sběrnice procesoru

Tabulka 5.2 – Popisy a využití vývodů konektoru J5

Konektor J5				
Pin	**Jméno**	**Hlavní použití**	**Jiné použití**	**Poznámka**
1-8	PA[0:7]	Paralelní programovatelné vstupy/výstupy	SD0-SD7	
9	/RESET	Resetovací výstup	Resetovací vstup	
10	PB0	Vstup	Hodiny sériového portu CLKB vstup n. výstup	
11	PB2	Vstup	/SWR	
12	PB3	Vstup	/SRD	
13	PB4	Vstup	SA0	Adresní vodiče sekundárního portu
14	PB5	Vstup	SA1	
15	PB7	Výstup	/SLAVEATTN	
16-23	D[7:0]	Vstupy/výstupy		
24	VBAT	Vstup 3V baterie		
25	VCC	Napájení +5V		
26	GND	Zem		

Porty procesoru Rabbit 2000 jsou nastavitelné, takže výchozí hodnoty mohou být programově změněny.

Piny PA0-PA7 mohou být použity pro připojení Rabbit 2000 jako sekudnárního k jinému procesoru. Sekundární port taktéž používá piny PB2-PB5, PB7 a PE7.

Piny PE0, PE1, PE4 a PE5 mohou být použity pro aplikaci až dvou externích přerušení. PB0 je možno využít pro přístup k hodinám sériového portu B procesoru. PD4 se může naprogramovat jako sériový výstup pro port B, PD5 může být sériovým vstupem portu B.

PC4, PC5, PD0, PD1, PE2, PE3 a PE6 jsou využívány pro komunikaci s řídícím čipem ethernetového rozhraní.

5.3 Sériová komunikace

Modul RCM2200 nemá přímo integrovaný převaděč RS-232 nebo RS-485, který proto může být umístěn na vnější desce.

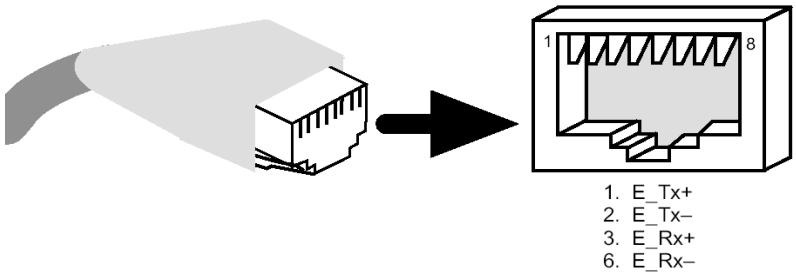
– Sériové porty

Modul má čtyři sériové porty označované jako port A, B, C a D. Všechny tyto porty mohou pracovat v asynchronním módu do rychlosti systémových hodin dělené 64. Asynchronní přenos může obsahovat 7 nebo 8 datových bitů, přičemž je podporován i devátý bit, používaný k označení prvního byte datového slova. Porty A a B mohou také pracovat v časovaném módu, kdy časovací výstup postupně časuje vstupní a výstupní data. Pokud je česování prováděno procesorem Rabbit 2000, přenosová rychlost může být do 80% kmitočtu systémových hodin děleného 128 (nebo 138000 bps pro časování 22,1MHz).

Port A je dostupný pouze na programovacím portu, takže jeho využití je poněkud nepraktické.

– Ethernetový port

Rozložení vývodů ethernetového konektoru RJ-45 je na následujícím obrázku.



Obr. 5.4 – Rozložení vývodů konektoru RJ-45

Vedle tohoto konektoru jsou umístěny dvě indikační LED diody, jedna informuje o připojení kabelu a druhá o přenosu dat. Signály k těmto LED jsou také vyvedeny na koektoru J4 a mohou tak být použity pro indikaci na externí desce.

– Programovací port

Jako programovací port je využit sériový port A, který je umístěn na 10ti pinovém konektoru J1. Používá se pro nastartování RCM2200 do módu, kdy je možné přenést do modulu program a spustit jej. Taktéž se přes něj přenáší data při krokování/diagnostikování programu přímo v modulu. Přes programovací port je možno modul také zresetovat.

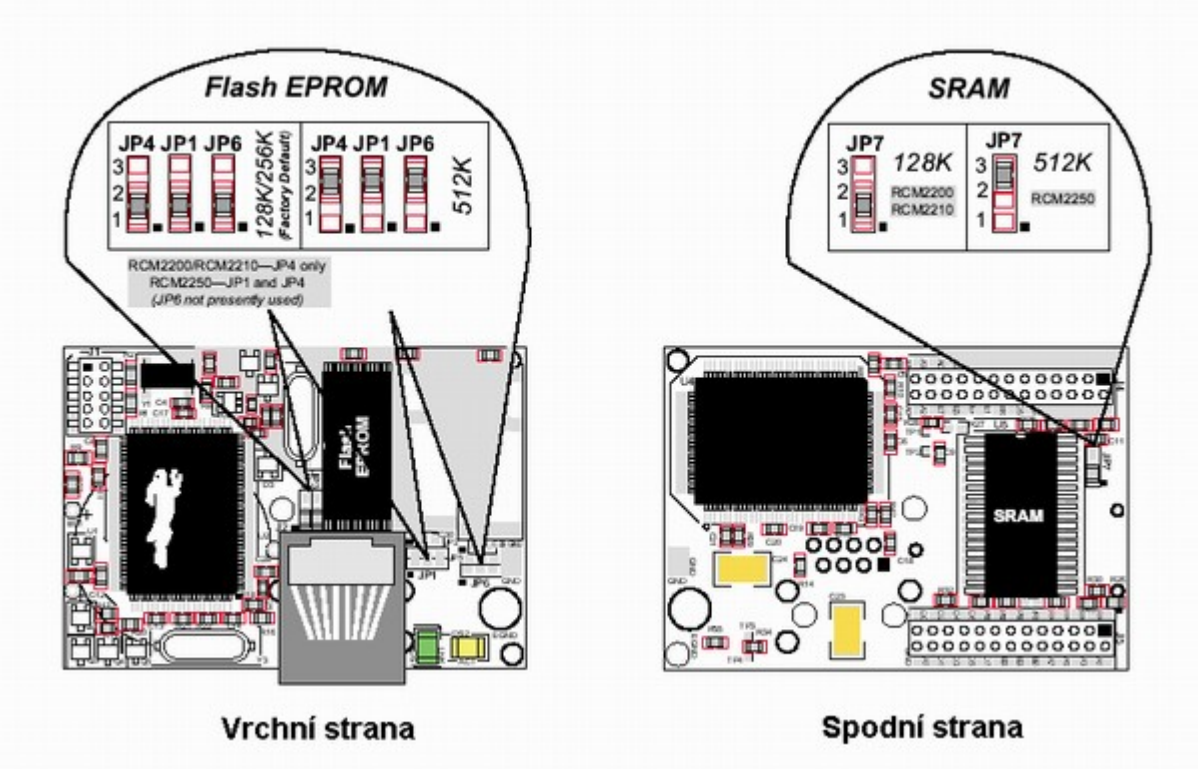
Obsahuje dva stavové piny, které určují startovací mód. Jejich logická hodota je velmi důležitá pro chování modulu po resetu.

Také obsahuje diagnostický pin, kde je moždo posílat digitální data informující o stavu modulu.

5.4 Paměť

- SRAM

Modul RCM2200 je možno rozšířit o další paměť a to od 32K do 512K v SOIC pouzdře. V základní výbavě se moduly dodávají se 128K nebo 512K SRAM, velikost paměti je nastavena jumpery v poli J7 (jumpery jsou v tomto případě nahrazeny 0Ω SMD rezistory).



Obr. 5.5 – Umístění jumperů na desce modulu

- Flash

Stejně jako SRAM i Flash paměť je možno rozšířit a to o 128K až 512K v pouzdře TSOP. Ve standardní verzi jsou na modulu jedna nebo dvě 256K Flash paměti. Nastavení velikosti paměti se provádí podobně jako u SRAM.

Komunikace mezi mikropočítačem RMC2200 a obvodem Spartan 3 bude probíhat po sběrnici pomocí příznaků /IOWR a /IORD. Obvod Spartan vyvolá každou vteřinu přerušení a mikropočítač přečte potřebná data – počet chyb za minulou sekundu a počet milisekund defektního signálu. Dále přečte z A/D převodníku údaje o přijímaném signálu a proudu diodou.

6. Programování modulu RCM2200

6.1 Vývojové prostředí Dynamic C

Prostředí Dynamic C bylo vyvinuto speciálně pro vývoj software mikroprocesorových systémů, umožňuje rychlou kompilaci a interaktivní krokování v reálném prostředí.

Při vývoji software je možno zvolit, zdali se má pro kompilaci a spouštění programu používat paměť Flash nebo statická RAM. Výhoda práce v RAM je hlavně v šetření životnosti Flash, jejíž počet přepisů je omezen na zhruba 100 000.

6.2 Odesílání UDP

Odesílání UDP paketu je realizováno pomocí funkce `udp_send()` z knihovny `UDP.LIB`

```
int udp_send( udp_Socket *s, char *buffer, int len );
```

kde:

`s`	je ukazatel na datovou strukturu socketu
`buffer`	je zásobník obsahující UDP datagram
`len`	je délka UDP datagramu

Návratová hodnota této funkce je:

`>0`	počet odeslaných bajtů
`-1`	chyba
`-2`	chyba protože nebyla zjištěna hardwarová adresa

Funkce odesílání UDP je volána v intervalech definovaných proměnnou a běží paralelně s dalšími procesy pomocí funkce `costate`.

6.3 Využití časovače (timeru)

Bylo využito časovače B, což je 10-bitový čítač spouštěný perclk/2 nebo perclk/16, perclk je taktovací kmitočet periferií a bývá obvykle nastaven na frekvenci hlavního oscilátoru CPU.

Pro taktovací kmitočet je tak možno dosáhnout těchto časů/kitočtů:

Tabulka 6.1 – Obnovovací doby / frekvence čítače

Vstup časovače B	Obnovení	Čas	Kmitočet
Perclk/2	0,135µs	138,88µs	7200Hz
Perclk/16	1,085µs	1,11ms	900Hz

Časovač byl realizován těmito funkcemi:

```
WrPortI(SPCR, &SPCRShadow, 0x84);
```
- povolí port A pro výstup

```
SetVectIntern(0x0B, timerb_isr);
```
- nastaví přerušení (co udělat při impulzu z časovače)

```
WrPortI(TBCR, &TBCRShadow, tmrb);
```
- nastaví taktování časovače podle proměnné tmrb (0x01 pro perclk/2)

```
WrPortI(TBL1R, NULL, 0x00);
WrPortI(TBM1R, NULL, 0x00);
```
- nastaví výchozí hodnoty časovače

```
WrPortI(TBCSR, &TBCSRShadow, 0x03);
```
- povolí časovač B a jeho přerušení

```
WrPortI(TBCSR, &TBCSRShadow, 0x00);
```
- zakáže časovač B a jeho přerušení

Funkce volaná při přerušení časovače:

```
#asm
timerb_isr::
     push af             ; uloží registry
     push hl

     ioi  ld a, (TBCSR)  ; nahraje přerušení B1 a B2
                         ; mělo by být voláno co nejdříve

     ld   hl, PADR
     ioi  ld (hl), 0xFF  ; pošle puls na port A
     ioi  ld (hl), 0x00

     ld   a, 00h
     ioi  ld (TBL1R), a  ; nastaví další hodnotu B1 (0000h)
     ioi  ld (TBM1R), a  ; musí se vždy obnovit

done:
     pop  hl             ; obnovení registrů
     pop  af

     ipres               ; obnovení přerušení
     ret                 ; návrat
#endasm
```

Jak je vidět, vždy po dokončení čítání časovače je na port A vyslán impuls, frekvence impulsů je dle předchozího 7200Hz resp. 900Hz. Toho bylo využito pro měření, zdali je možné využít časovač pro vzorkování měřených údajů chybovosti. Bohužel se ale ukázalo, že generování pulsů tímto způsobem je velice nepřesné (odchylka ±10%) a jeho využití pro tento účel je tedy nevhodné.

6.4 HTTP server

HTTP server umožňuje přes síť zobrazovat HTML stránky na klientech (web prohlížečích). V našem případě také poskytuje informace o stavu modulu a umožňuje jeho konfiguraci. Je implementován v knihovně HTTP.LIB a jeho základní programový zápis vypadá takto:

```
#define TCPCONFIG 1

#memmap xmem
#use "dcrtcp.lib"
#use "http.lib"
/* zde je třeba nastavit soubory, proměnné atd. ke kterým má mít
http server přístup */
main(){
     sock_init();
     http_init();
     tcp_reserveport(80);
     while (1) {
          http_handler();
     }
}
```

Prvně je třeba nadefinovat mime-typy souborů:

```
const HttpType http_types[] =
{
   { ".shtml", "text/html", shtml_handler}, // ssi
   { ".html", "text/html", NULL},           // html
   { ".cgi", "", NULL},                     // cgi
   { ".gif", "image/gif", NULL},            // gif
   { ".jpg", "image/jpeg", NULL}            // jpg

};
```

poté musíme přidat soubory, které bude server obsluhovat, na flash.

```
const HttpSpec http_flashspec[] =
{
 {HTTPSPEC_FILE, "/",            index_html, NULL, 0, NULL, NULL},
 {HTTPSPEC_FILE, "/index.shtml", index_html, NULL, 0, NULL, NULL},
 {HTTPSPEC_FILE, "/saved.html",  saved_html, NULL, 0, NULL, NULL},
 {HTTPSPEC_FILE, "/logo.jpg",    logo_jpg,   NULL, 0, NULL, NULL},
 {HTTPSPEC_FILE, "/field.gif",   field_gif,  NULL, 0, NULL, NULL},
 {HTTPSPEC_VARIABLE, "udpsopt",  0, udpsopt, PTR16, "%s", NULL},
 {HTTPSPEC_VARIABLE, "tmrbopt",  0, tmrbopt, PTR16, "%s", NULL},
 {HTTPSPEC_FUNCTION, "/udpcreset.cgi", 0, udpc_reset, 0, NULL,
  NULL},
};
```

Přidány jsou HTML stránky, přípona .shtml značí dynamickou stránku, proměnné a CGI funkce

Jelikož chceme používat formuláře, musíme si je nadefinovat

```
FormVar setform[6];
```

- tato definice nastaví formulář se šesti proměnnými.

```
form = sspec_addform("config.html", setform, 6, SERVER_HTTP);
function = sspec_addfunction("saved", saved, SERVER_HTTP);
sspec_setformepilog(form, function);
sspec_setuser(form, user);
sspec_setrealm(form, "Admin");
```

- tyto funkce vytvoří vlastní formulář

```
sspec_setformtitle(form, "Nastavení");
```

- definuje název (nadpis) formuláře

Tím je hotovo prvotní vytvoření formuláře, nyní je třeba pouze zadefinovat vlastní proměnné

```
var = sspec_addvariable("sendint", &sendint, INT16, "%d",
      SERVER_HTTP);
var = sspec_addfv(form, var);
sspec_setfvname(form, var, "Interval odesílání dat (s)");
sspec_setfvdesc(form, var, "Interval, ve kterém jsou data
                            odesílana na server (1-60s)");
sspec_setfvlen(form, var, 4);
sspec_setfvrange(form, var, 1, 60);
```

Nyní již je vše zadefinováno, takže spustíme vlastní server

```
http_init();
tcp_reserveport(80);
```

- provede inicializaci serveru a rezervuje pro něj port

```
http_handler();
```

- spustí vlastní server, voláno paralelně s ostatními funkcemi pomocí costate

6.5 WWW rozhraní

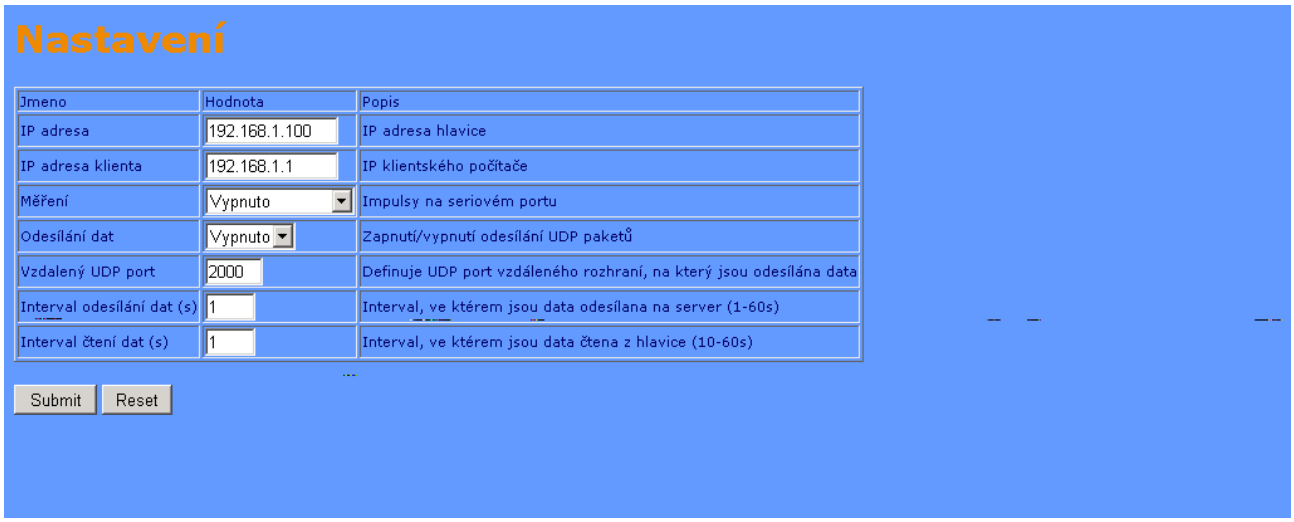
WWW rozhraní v modulu RMC2200 umožňuje sledování stavu a parametrů hlavice a také konfiguraci hlavice. Je dostupné na IP adrese 192.168.1.100

Obr. 6.1 – Zobrazení stavu hlavice

Na této základní obrazovce jsou zobrazovány aktuální informace o stavu měřící hlavice

Zobrazováno je:

- Základní nastavení hlavice
- Stav (zapnutí/vypnutí) služeb
- Jejich nastavení
- Počet odeslaných UDP paketů
- Aktuální naměřené hodnoty



Obr. 6.2 – Nastavení hlavice

Vstup do konfigurace hlavice odkazem Nastavení z hlavní stránky je z důvodu bezpečnosti chráněn jménem a heslem. Základní nastavení je

Jméno: admin

Heslo: abc234

Možné volby:

- Nastavení IP adresy
- Nastavení cílové IP adresy (klientského PC
- Cílový UDP port
- Zapnutí/Vypnutí měření
- Zapnutí/Vypnutí odesílání dat
- Interval odesílání dat
- Interval měření dat

7. Klientský program pro systém Windows

7.1 Používané funkce programu

Nejprve je třeba inicializovat sockety (knihovnu winsock.dll), což se provede následující funkcí:

```
WSADATA localWSA;
int status;
WSAStartup(MAKEWORD(1,1),&localWSA
```

která nám umožní používání komunikace pomocí protokolu TCP/IP.

Stálý příjem paketů z hlavice je zajištěn pomocí vlákna (thread):

```
void RecvThread(void * passedThread)
```

v němž jsou obsaženy potřebné funkce pro příjem paketu a zápis dat v něm obsažených.

Vlastní příjem paketu je zajištěn funkcí

```
recvfrom( GlobalSocket, tempBuff, 101, (int)NULL,  (SOCKADDR
*)&remote_sockaddr_in,(int *)  &remote_sockaddr_length);
```

Následují funkce pro získání dat z paketu. Prvně se získá aktuální datum a čas a vytvoří řetězec s datem a časem příjmu paketu a aktuálními daty (oddělené středníkem), vypíší se na obrazovku a zapíší do souboru

```
gettime(&t);
getdate(&d);

sprintf(tempBuff3,"%02d-%02d-%02d,%02d.%02d.%02d;%s;",d.da_year,
d.da_mon, d.da_day, t.ti_hour, t.ti_min, t.ti_sec, tempBuff);

Form1->Memo1->Lines->Add(tempBuff3);
writefile(tempBuff3);
```

Poté se oddělí jednotlivé hodnoty z přijatého řetězce pro účely jejich zobrazení na stránce Monitor v programu

```
int pomDelka = 0;
int strednik = 0;
int i = 0;
float rx_lev2;
   for(i; i< 256; i++)
   {
   char *pomocnaProm="";
   if(tempBuff3[i] != ';')
   {
           pomocnaProm[pomDelka] = tempBuff3[i];
           pomDelka++;
   }
   else
   {
```

```
      char pomocnaProm2[4];
      if (strednik == 1) rx_lev = RoundTo(atof(pomocnaProm), -2);
      if (strednik == 2) current = atof(pomocnaProm);
      if (strednik == 3) err_rate = StrToInt(pomocnaProm);
      strednik++;

      for (int j=0; j < pomDelka; j++)
      {
            pomocnaProm[j] = 0;
      }

      pomDelka=0;
      Form1->lbl_rxlev->Caption =  FloatToStrF(rx_lev,ffNumber,5,2);
      Form1->lbl_current->Caption =FloatToStrF(current,ffNumber,5,2);
   }
   }
```

Zápis do souboru je prováděn touto funkcí:

```
void writefile(char *tmpBuf)
{
   FILE * pFile;
   char sentence [256];
   char sentence2 [256];
   char filename [12];
   struct  date d;

   getdate(&d);
   sprintf(filename,"%02d%02d.log", d.da_mon, d.da_day);
   sprintf(sentence,tmpBuf);
   sprintf(sentence2,"%s\n", sentence);
   pFile = fopen (filename,"at");
   fputs (sentence2,pFile);
   fclose (pFile);
}
```

Opět se pomocí funkce `getdate(&d);` získá aktuální datum, pomocí kterého se utváří název souboru.

Přijatá data chybovosti se také zaznamenávají do grafu, který je tvořen komponentou TPerformanceGraph

Program dále umožňuje pro zkušební účely pracovat jako server, tj. Odesílat náhodná data na jiné PC s tímto programem v módu klient**.**

Generace dat je zajištěna funkcí

```
void genernum(void)   {
 float r,c,e;
 randomize();
 r = random(70);
 c = random(12);
 e = random(15);
 rx_lev = 50 - (r/10);
 current = 130 - (c/10);
 err_rate = 40 + e;
```

```
}
```

Tato data jsou poté periodicky odesílána protokolem UDP protistraně pomocí funkce

```
sendto(GlobalSocket,tempBuff,sizeof(tempBuff),(int)NULL,
(SOCKADDR *)&remote_sockaddr_in, sizeof(SOCKADDR_IN)
```

Jako adresáta ne možno zadat i DNS jméno, jehož překlad na IP adresu je zajištěn funkcí

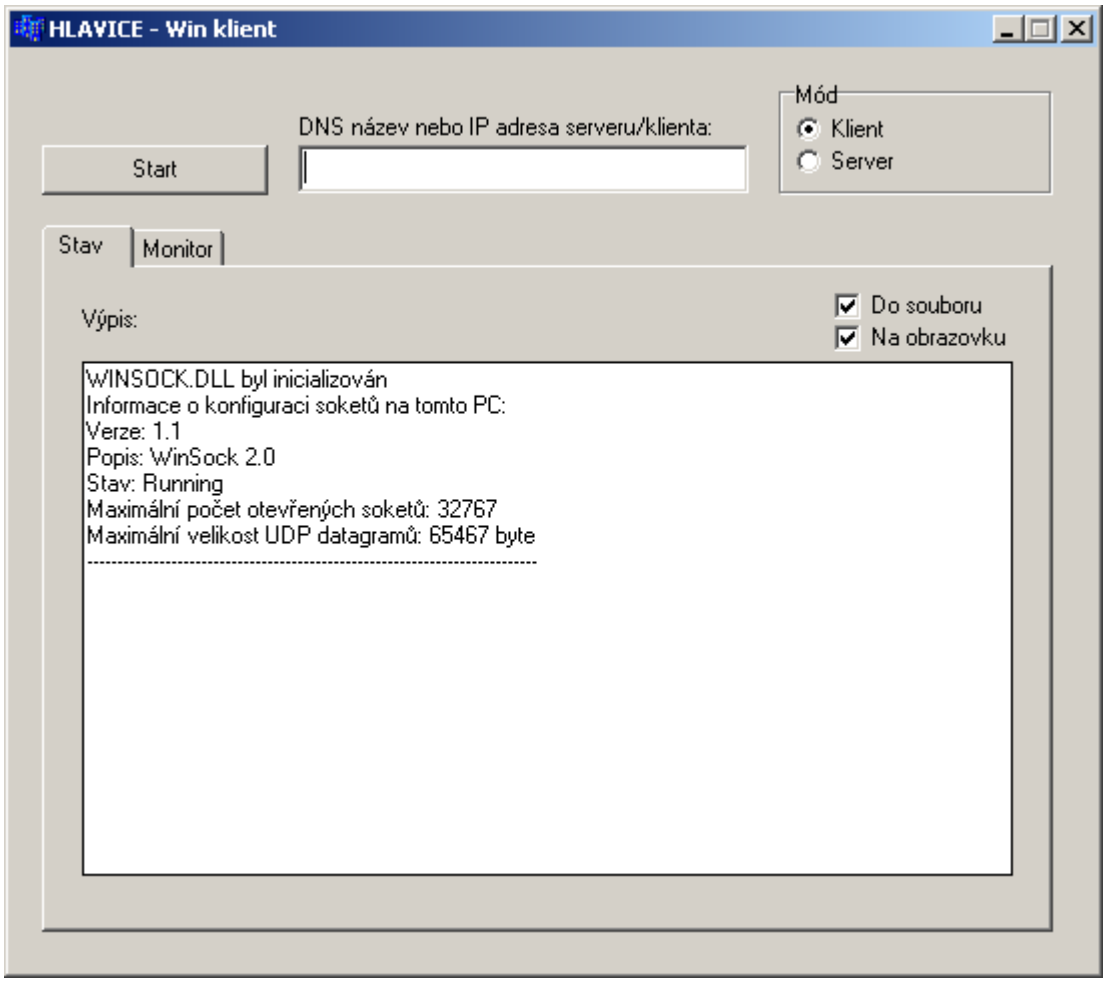
```
hostent *H = gethostbyname(Form1->Edit1->Text.c_str());
addr = *(long *)H->h_addr_list[0];
```

A opět je zjištěn aktuální čas a vypsán spolu s adresou adresáta na obrazovku

```
struct  time t;
gettime(&t);
sprintf(tempBuff,"%02d:%02d:%02d-Odesláno na %s (%s)", t.ti_hour,
t.ti_min, t.ti_sec, name, inet_ntoa (remote_sockaddr_in.sin_addr));
Form1->Memo1->Lines->Add(tempBuff);
```

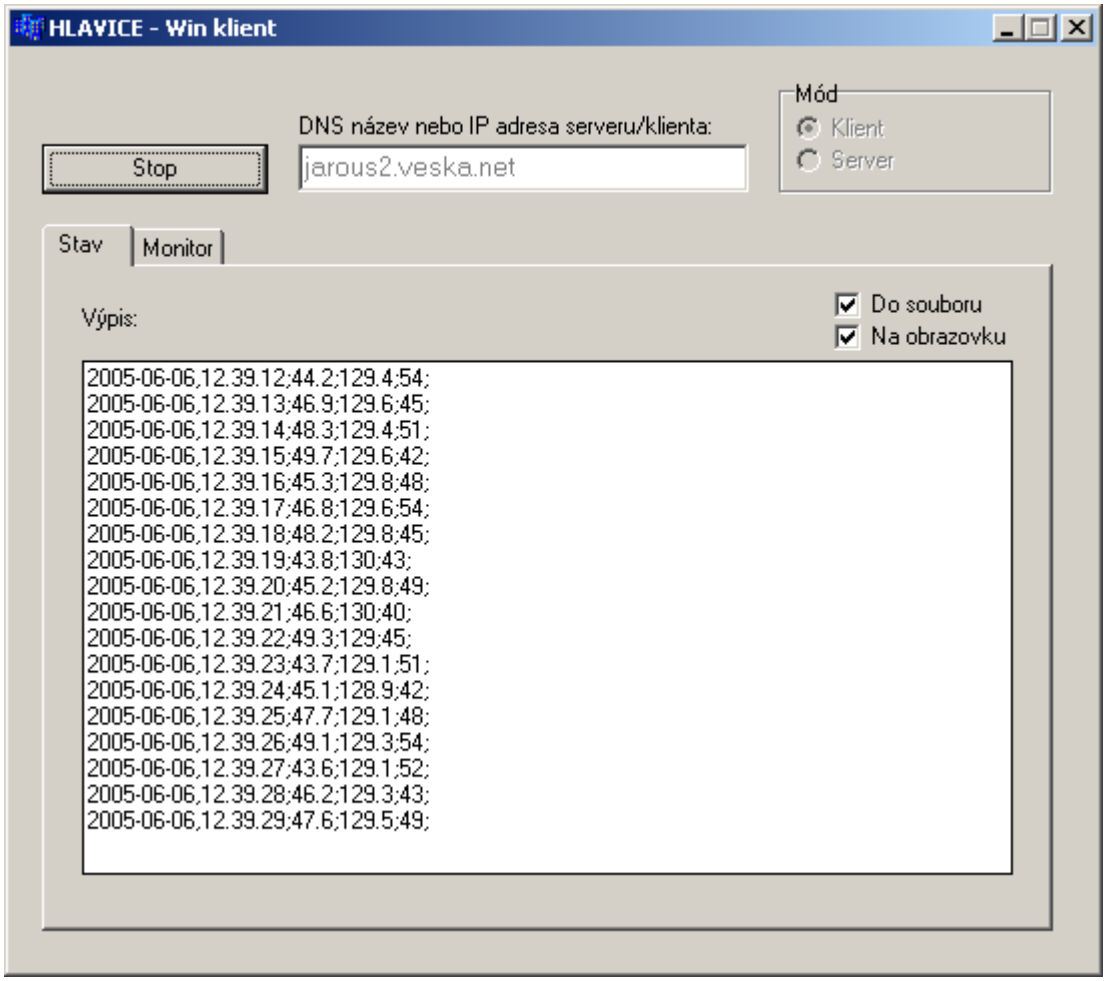
7.2 Uživatelské rozhraní

Po spuštění programu se objeví jeho základní obrazovka:



Obr. 7.1 – Hlavní okno programu

Ve výpisovém okně se po spuštění vypíše zpráva o aktualizaci soketů, je u něj také možnost volby zapnutí/vypnutí výpisu přijatých dat do souboru nebo na obrazovku.
Uživatel má možnost volby módu Klient nebo Server. V módu klient stačí spustit příjem tlačítkem Start, program zančne přijímat data na zvoleném UDP portu a dle zvolených možností je zobrazovat nebo ukládat do souboru.



Obr. 7.2 – Výpis přijatých dat na obrazovku

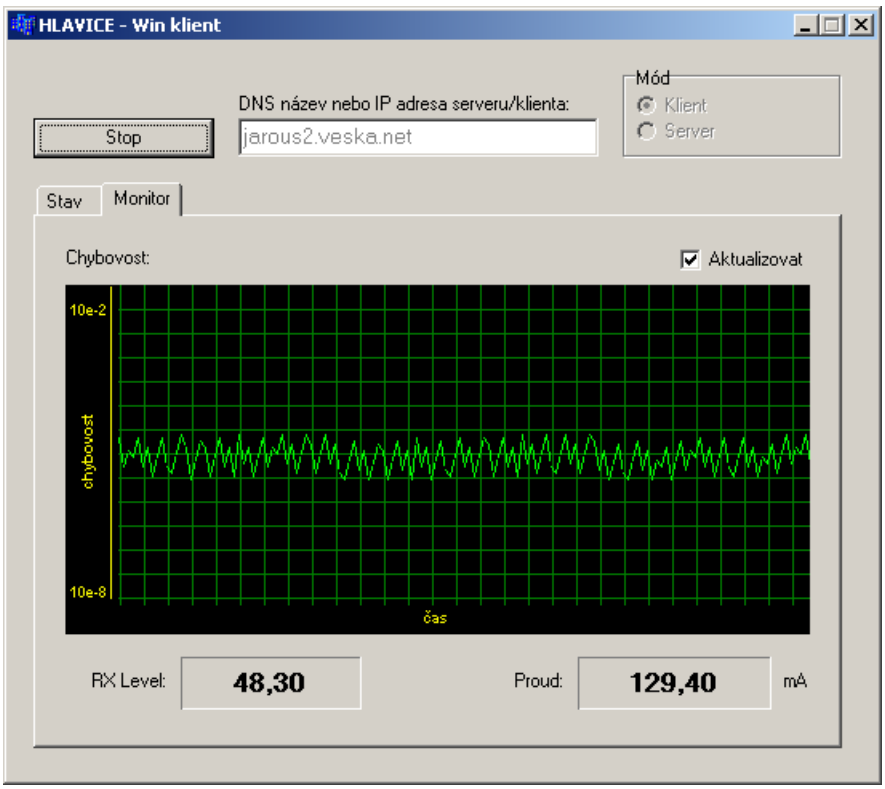
Výpis dat na obrazovku je uveden na obrázku 7.2, ve stejném formátu jsou data ukládána i do souboru, jehož název je tvořen aktuálním datem a příponou .log, tedy například „0406.log“.

Data jsou ukládána v následujícím formátu:

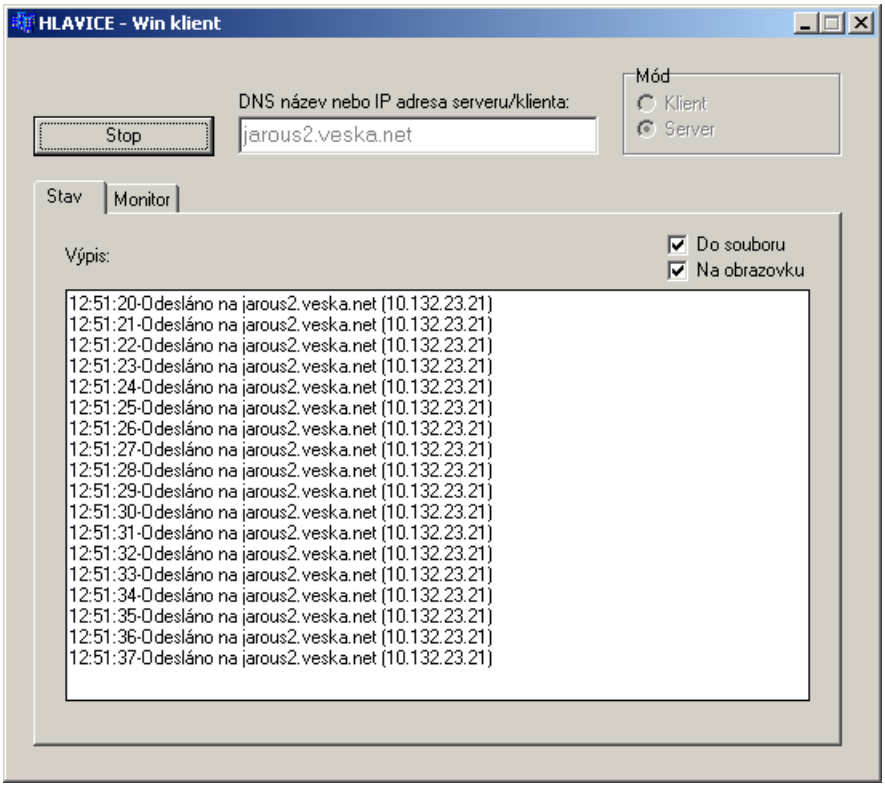
```
datum,čas;přijímaná_úroveň;proud_diodou;naměřená_chybovost;
```

Na záložce Monitor je potom možno naměřená data online sledovat – úroveň signálu a proud diodou číselně, hodnotu chybovosti spoje i v grafu, jak lze vidět na obrázku 7.3

Ve zkušebním serverovém módu je třeba zadat do textového pole IP adresu nebo jméno klientského počítače. Po spuštění jsou náhodná data periodicky odesílána příjemci, záznam o odeslání je opět vypisován na obrazovku podle obrázku 7.4



Obr. 7.3 – Zobrazení naměřených dat



Obr. 7.4 – Výpis o odeslání paketu

8. Klientský program pro systém Unix (Linux)

Klientský program pro operační systém Unix má v podstatě stejnou funkci, jako program pro systém Windows, pouze, jelikož se jedná o systémového daemona, neumožňuje přímé pozorování naměřených dat a zajišťuje tak pouze jejich ukládání do souboru.

Programy (procesy) v Unixových systémech mohou běžet v popředí anebo na pozadí. Proces běžíci v popředí komunikuje s uživatelem pomocí terminálu, zatímco proces na pozadí běží samostatně, neblokuje terminál a většinou nedává najevo, že je spuštěn. Pro náš účel je tedy výhodný program na pozadí – tzv. „Daemon“

Nejprve je třeba program „démonizovat“, aby pracoval na pozadí

```
i=fork();
if (i<0) exit(1); /* fork error */
if (i>0) exit(0); /* parent exits */
/* child (daemon) continues */
```

Systémová funkce fork() je použita pro vytvoření kopie našeho procesu, poté ukončí rodičovský proces. Proces je tedy uvolněn od svého původního procesu a pracuje na pozadí.

pak je potřeba nastavit identitu procesu. Každý proces dostává signály z terminálu, ke kterému je připojen a námi nově vytvořený proces „dědí“ tyto signály od svého rodčovského procesu. Tato vlastnost je nežádoucí a proto proces oddělíme pomocí následující funkce

```
setsid() /* získá novou skupinu procesu */
```

Proces tak získá svou novou skupinu a pracuje potom zcela nezávisle.

Podobně se na nově vytvořený proces přesouvají i otevřené popisovače (deskriptory). Aby k tomu nedocházelo, měly by být uzavřeny ještě před voláním funkce fork() anebo ihned po spuštění nového procesu.

```
for (i=getdtablesize();i>=0;--i) close(i);
```

Jelikož rutiny standardní knihovny mohou číst nebo zapisovat do standardních I/O deskriptorů (standart input 'stdin' (0), standart output 'stdout' (1), standart error 'stderr' (2)), je dobré je mít z důvodu bezpečnosti otevřené a pokud nejsou využívány, spojeny s nulovým výstupem (/dev/null)

```
i=open("/dev/null",O_RDWR); /* stdin */
dup(i); /* stdout */
dup(i); /* stderr */
```

Pro účely monitorování potřebujeme spustit pouze jednu kopii programu. Toho se dá jednoduše docílit metodou uzabykání souboru, první instance programu vytvoří a uzamkne soubor, podle čehož další spouštěné instance poznají, že program již běží. Při ukončení programu je soubor automaticky uvolněn, aby jej bylo možné znovu spustit. Do souboru se také zaznamená aktuální PID procesu.

```
lfp=open("exampled.lock",O_RDWR|O_CREAT,0640);
if (lfp<0) exit(1); /* can not open */
if (lockf(lfp,F_TLOCK,0)<0) exit(0); /* nemuze uzamknout */
/* jen první instance pokracuje */

sprintf(str,"%d\n",getpid());
write(lfp,str,strlen(str)); /* ulozi PID */
```

Daemon by měl také reagovat na signály přijaté od uživatele nebo jiného programu, proto je třeba tyto signály zachytávt, což obstarává tento kód

```
signal(SIG_IGN,SIGCHLD); /* child terminate signal */

void Signal_Handler(sig) /* funkce obsluhy signálů */
int sig;
{
switch(sig){
case SIGHUP:
/* restart programu */
break;
case SIGTERM:
/* ukončení programu */
exit(0)
break;
}
}

signal(SIGHUP,Signal_Handler); /* hangup signal */
signal(SIGTERM,Signal_Handler); /* signal softwarového
ukončení příkazu kill */
```

Funkce pro příjem dat přes UDP protokol a pro ukládání přijatých hodnot do souboru jsou v podstatě stejné s programem pro Windows a proto nebudou blíže rozebrány.

Klientský program pro systém Unix tedy pracuje na pozadí jako daemon, poslouchá na zvoleném UDP portu a přijímá naměřená data odeslaná z měřící hlavice. Ta jsou ukládána do souboru pojmenenovaného podle aktuálního data a to ve stejném formátu, jako u Windows programu.

9. Závěr

Tato bakalářská práce se zabývala dálkovým monitorováním bezkabelového spoje. Byla navržena koncepce měřícího systému, možnost realizace měřící hlavice a samotného testeru. Po seznámení se s různými komunikačními protokoly byl pro přenos naměřených dat mezi měřící hlavicí a klientským počítačem vybrán pro svou nenáročnost protokol UDP. Po prostudování parametrů mikropočítače Rabbit RCM2200 a zjištění, že signál generovaný časovačem je značně nestabilní a tudíž pro přímé měření dat nevhodný, byla navržena realizace měření dat externím FPGA obvodem Xilinx Spartan 3. Mikropočítač Rabbit periodicky čte naměřená data z tohoto obvodu a odesílá je po síti Ethernet protokolem UDP do klientského počítače. Také v něm byl implementován webserver pro jednoduchou konfiguraci systému pomocí www rozhraní. Pro klientské počítače byly vytvořeny programy pro operační systémy Windows a Unix (Linux), které přijímají naměřená data odeslaná z hlavice a ukládají je do souboru podle aktuálního data. Program pro Windows umožňuje navíc přímé sledování naměřených dat v podobě textu a grafu.
...

Použitá literatura a zdroje:

[1] Kabelová, A., Dostálek, L.: V*elký průvodce protokoly TCP/IP a systémem DNS,* 2.vyd. Brno: Computer Press, 2002, ISBN 80-7226-323-4

[2] Prokeš, A.: Koncepce současných optických směrových spojů, Elektrorevue [online], 2000, [cit. 2005]. Dostupné na WWW <http://www.elektrorevue.cz>

[3] RFC-0760. *Internet Protocol Standard*. 1980

[4] RFC-0761. *Transmission Protocol Standard*. 1980

[5] RFC-0768. *User Datagram Protocol.* 1980

[6] RFC-1157. *A Simple Network Management Protocol*. 1990

[7] RFC-1945. *Hypertext Transfer Protocol – HTTP/1.0.* 1996

[8] Dostál, R.: Sokety a C++, Builder [online], 2002, [cit. 2005]. Dostupné na WWW <http://www.builder.cz/art/cpp/sokety_a_cpp.html>

[9] Peterka, J.: Rodina protokolů TCP/IP, E-archiv [online], 2000, [cit. 2005]. Dostupné na WWW <http://www.earchiv.cz/l207/index.php3>

[10] Zapletal, L.: Protokol HTTP 1.1 pod lupou, Root [online], 2001, [cit. 2004]. Dostupn0 na WWW <http://www.root.cz/clanek/629>

[11] Bharghava, N.: UDP Sockets-based Client Server Programs, Linux Gayette [online], 2004, [cit. 2004]. Dosupné na WWW <http://www.linuxgazette.com/node/view/8758>

[12] Pech, J.: Nebojte se FPGA, HW Server [online], [cit. 2005]. Dostupné na WWW <http://www.hw.cz/docs/fpga/fpga.html>