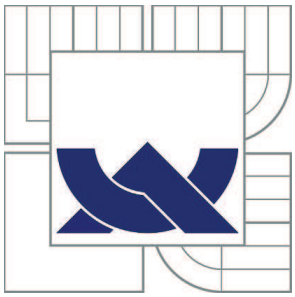
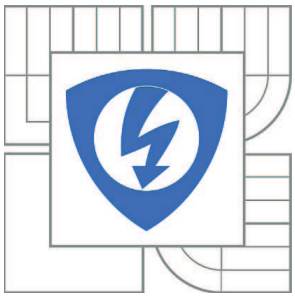


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV MIKROELEKTRONIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF MICROELECTRONICS

IMPLEMENTACE ETHERNETOVÉHO KOMUNIKAČNÍHO ROZHRANÍ DO OBVODU FPGA

IMPLEMENTATION OF ETHERNET COMMUNICATION INTEFACE INTO FPGA CHIP

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

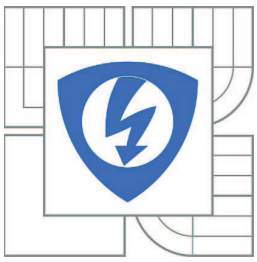
AUTOR PRÁCE — Bc. PETR SKIBIK

AUTHOR

VEDOUCÍ PRÁCE — Ing. MAREK BOHRN

SUPERVISOR

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav mikroelektroniky

Diplomová práce

magisterský navazující studijní obor
Mikroelektronika

Student:	Bc. Petr Skibik	***ID:***	50474
Ročník:	2	***Akademický rok:***	2010/2011

NÁZEV TÉMATU:

Implementace ethernetového komunikačního rozhraní do obvodu FPGA

POKYNY PRO VYPRACOVÁNÍ:

Do obvodu FPGA implementujte ethernetové komunikační rozhraní.
Rozhraní bude podporovat komunikaci po protokolu IP a nadstavbovém protokolu UDP. Výsledná implementace bude obsahovat všechny potřebné nižší síťové vrstvy.
Pro praktickou realizaci použijte cílový obvod FPGA řady Virtex-5 a zvažte možnosti přenositelnosti návrhu do jiných řad obvodů FPGA.
Dále vytvořte testovací aplikaci pro PC, která ověří správnost komunikace ethernetového rozhraní v obvodu FPGA. Testovací aplikace bude umožňovat přenášet data mezi PC a obvodem FPGA a bude podporovat funkci Ping.

DOPORUČENÁ LITERATURA:

Podle pokynů vedoucího práce.

Termín zadání:	7.2.2011	***Termín odevzdání:***	26.5.2011

Vedoucí práce: Ing. Marek Bohrn

prof. Ing. Vladislav Musil, CSc.
Předseda oborové rady

Abstrakt:

Tato práce se zabývá návrhem síťového komunikačního rozhraní na bázi Ethernetu a jeho implementací do obvodu FPGA. Pro popis hardwaru je použit programovací jazyk VHDL. Práce zahrnuje implementaci protokolu linkové vrstvy Ethernetu, dále síťové protokoly IPv4, ARP, ICMP a UDP. Výsledný návrh umožňuje obousměrný datový přenos na úrovni transportní vrstvy TCP/IP modelu. Pro implementaci rozhraní byla použita vývojová deska ML506 osazena FPGA obvodem Virtex5 od firmy Xilinx.

Abstract:

The thesis deals with the implementation of Ethernet-based network communication interface into FPGA chip. VHDL programming language is used for description of the hardware. The interface includes the implementation of link-layer Ethernet protocol and network protocols such as IPv4, ARP, ICMP and UDP. The final design allows bi-directional communication on the transport-layer level of TCP/IP model. The designed interface was implemented into Virtex5 FPGA chip on development board ML506 by Xilinx.

Klíčová slova:

Ethernet, IEEE803.2, TCP/IP model, síťová komunikace, linková vrstva, síťová vrstva, transportní vrstva, paket, IP, UDP, ARP, ICMP, kontrolní součet, CRC, VHDL, FPGA.

Keywords:

Ethernet, IEEE803.2, TCP/IP model, network communication, link layer, network layer, transport layer, packet, IP, UDP, ARP, ICMP, checksum, CRC, VHDL, FPGA.

Bibliografická citace díla:

SKIBIK, P. *Implementace ethernetového komunikačního rozhraní do obvodu FPGA.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2011. 69 s. Vedoucí diplomové práce Ing. Marek Bohrn.

Prohlášení autora o původnosti díla:

Prohlašuji, že jsem tuto vysokoškolskou kvalifikační práci vypracoval samostatně pod vedením vedoucího diplomové práce, s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury. Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne 26. 5. 2011

…….

Poděkování:

Děkuji vedoucímu diplomové práce Ing. Marku Bohrnovi za metodické a cíleně orientované vedení při plnění úkolů realizovaných v návaznosti na diplomovou práci.

OBSAH

1 Úvod

V moderních digitálních systémech je často kladen požadavek na přenos dat mezi zařízením a PC za účelem jejich dalšího zpracování, sdílení nebo archivace. Jenou z možností je použití síťové technologie Ethernet. Tato technologie dominuje na poli počítačových sítí a je pro ni vybudována rozsáhlá infrastruktura. Mezi některé výhody patří např. vysoká rychlost datového přenosu (současné síťové karty běžně podporují 1Gb/s) nebo možnost napájení přímo z rozhraní Ethernet (tzv. Power over Ethernet). Vzdálenost, na kterou je možné data přenášet, je omezena pouze rozsahem infrastruktury. Implementací vhodného internetového protokolu tato vzdálenost vzroste do globálních měřítek sítě Internet.

Cílem diplomové práce bylo navrhnout a implementovat modul, umožňující přenášet data mezi obvodem FPGA a PC na úrovni transportní vrstvy TCP/IP modelu za použití síťového rozhraní Ethernet. Práce je rozdělena na teoretickou a praktickou část.

Teoretická část je věnována stručnému popisu vrstev a protokolů síťového modelu TCP/IP, které jsou použity v praktické části. Tento popis zahrnuje především strukturu paketů jednotlivých komunikačních protokolů, princip jejich zpracování, směrování a analýzu zajištění integrity dat.

Prvních osm kapitol praktické části se zabývá samotným návrhem komunikačního modulu a jeho částí, nezbytných pro implementaci. Pro popis hardwaru byl zvolen programovací jazyk VHDL, pro syntézu a implementaci bylo použito vývojového prostředí ISE od firmy Xilinx. U každé vrstvy je uveden její popis, blokové schéma a případně simulace. Kapitola 3.9 zvažuje možnosti implementace navrženého modulu do různých řad obvodů FPGA. Kapitola 3.10 seznamuje čtenáře s navrženou aplikací pro PC, která je určena k ověření funkce datového přenosu. Poslední kapitola popisuje praktické testování modulu, obsahuje tabulku dosažených přenosových rychlostí při použití různých verzích síťového rozhraní Ethernet.

2 Teoretická část

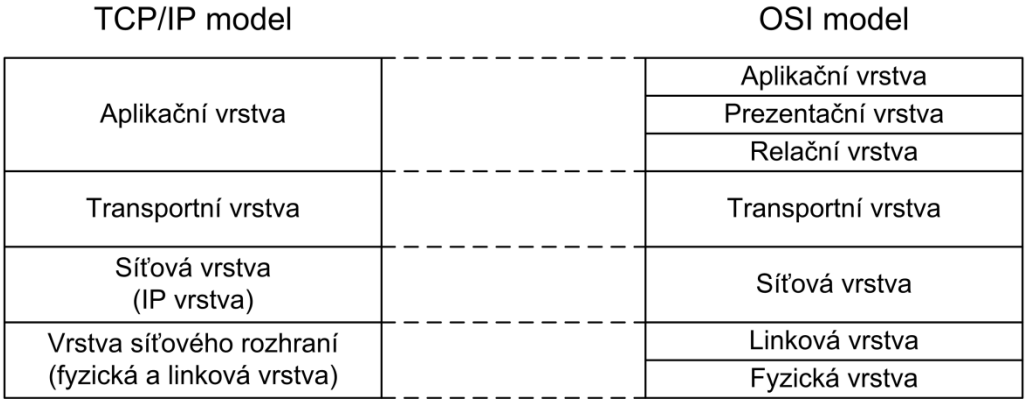
2.1 Model síťové architektury TCP/IP

Každá síťová architektura musí poskytovat svým uživatelům alespoň jednu ze dvou základních komunikačních služeb. *Spojovaná* služba před vlastní komunikací navazuje spojení s cílovým uživatelem. Cílový uživatel musí toto spojení akceptovat, potom je zahájena samotná komunikace, po níž je spojení ukončeno. Protikladem služby spojované je služba *nespojovaná*, která zahájí komunikaci okamžitě.

Každá z těchto služeb může mít navíc *spolehlivý* nebo *nespolehlivý* charakter. Spolehlivá služba je taková, která zajistí doručení všech odeslaných dat ve správném pořadí. Naproti tomu nespolehlivá služba doručení dat ani jejich správné pořadí nezajišťuje.

Mezi nejznámější koncepce síťové architektury patří model OSI (z ang. Open System Interconect) a model TCP/IP. OSI model není navržen přímo pro síťové komunikace, ale obecně pro spojování otevřených systémů. Nedefinuje přesné služby ani protokoly použité v každé vrstvě. Naproti tomu model TCP/IP byl navržen přímo pro síťovou architekturu TCP/IP [1].

Hlavním rozdílem je, že OSI model klade důraz na spojovaný a spolehlivý charakter každé vrstvy modelu. To v případě přenosu IP paketů v TCP/IP sítích neplatí. Naopak TCP/IP model předpokládá nespojovaný charakter přenosu a zajištění spolehlivosti služeb řeší až transportní vrstva. Srovnání modelů OSI a TCP/IP ukazuje obrázek 1 [1].



Obr. 1: Srovnání referenčních modelů sítové architektury [1]

Vrstva síťového rozhraní komunikuje přímo s přenosovým médiem. Jako jediná je závislá na technologii přenosu. Má na starosti vysílání a příjem jednotlivých bitů. Definuje rychlost přenosu, typ média, kódování, modulaci, obnovu taktu, formát přenášeného

linkového rámce apod. V TCP/IP sítích je nejčastěji implementována rozhraním Ethernet. V OSI modelu této vrstvě odpovídá linková a fyzická vrstva.

Síťová vrstva není závislá na technologii přenosu. Je implementována protokolem IP. Poskytuje nespojovanou a nespolehlivou službu. Zajišťuje doručování paketů mezi dvěma libovolnými zařízeními v síti. Zařízení jsou adresována unikátní síťovou adresou. Vrstva je totožná se síťovou vrstvou OSI modelu.

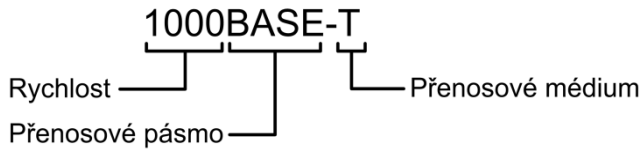
Transportní vrstva zajišťuje přenos paketů mezi dvěma koncovými účastníky, kterými jsou přímo PC aplikace. Podle jejich požadavků může transportní vrstva řídit tok dat, zajišťovat spolehlivost přenosu, nebo měnit nespojovanou službu síťové vrstvy na spojovanou. Vrstva je totožná s transportní vrstvou OSI modelu.

Aplikační vrstvu představují samotné aplikace, které komunikují přímo s transportní vrstvou. Mohou si sami zajistit další služby jako šifrování, kódování, potvrzování přijatých dat atd. V OSI modelu této vrstvě odpovídá relační, prezentační a aplikační vrstva [1], [2].

2.2 Síťové rozhraní Ethernet

Ethernet je komunikační rozhraní, původně vyvinuto firmami DEC, Intel a Xerox. Později jeho vývoj převzala organizace IEEE pod označením IEEE802.3. V současnosti existuje několik verzí Ethernetu, lišící se rychlostí a přenosovým médiem [2].

Skládá se z fyzické vrstvy, jejíž implementace se podle verze Ethernetu liší, a linkové vrstvy, která je na verzi prakticky nezávislá. Vrstvy mezi sebou komunikují nejčastěji přes rozhraní MII (z ang. Media Independent Interface), případně GMII (z ang. Gigabit Media Independent Interface). Značení Ethernetu podle IEEE802.3 ukazuje obrázek 2.



Obr. 2: Značení ethernetu podle IEEE802.3

Popis značení:

- *Rychlost:* udává teoretickou rychlost přenosu dat v megabitech za sekundu. Mezi nejpoužívanější verze patří 10/100/1000Mbit/s. Obvody podporující všechny tyto rychlosti jsou označovány jako *Tri-Mode Speed* [17].
- *Přenosové pásmo:* má spíše historický charakter, v současnosti všechny verze používají základní přenosové pásmo (BASE).
- *Přenosové médium:* udává typ média a s ním související provedení fyzické vrstvy. První písmeno *T* označuje metalické spojení přes kroucenou dvojlinku.

Ethernet rychlosti 100Mbit/s zahrnuje pro propojení kroucenou dvojlinkou až tři různé implementace, podle počtu použitých párů a typu spojení. Konkrétní implementace je v takovém případě definována dalším znakem. Mezi nejběžnější verze, které jsou vzájemně kompatibilní, patří 1000BASE-T, 100BASE-TX a 10BASE-T.

Použití optického vlákna jako přenosového média opět zahrnuje několik implementací, lišící se typem vlákna, kódováním, nebo vlnovou délkou použitého zdroje záření. Typická označení pro Ethernet přes optické vlákno jsou FX, SX, LX. Přesný popis jednotlivých implementací je dostupný ve standardech IEEE802.3.

2.2.1Přenosové médium

Přenosové médium tvoří fyzický spoj mezi jednotlivými prvky sítě. Je přímo spojeno s fyzickou vrstvou. První verze Ethernetu používaly jako médium koaxiální kabel. V současnosti se nejčastěji používá levnější UTP (z ang. Unshielded Twisted Pair) kabeláž. UTP kabel obsahuje čtyři kroucené diferenciální páry, je zakončen konektorem RJ-45. Mezi fyzickou vrstvou a konektorem jsou oddělovací transformátory.

Návrh a instalace kabeláže pro sítě Ethernet se řídí průmyslovou normou EIA-568 [4]. V tabulce 1 je uvedeno označení nejběžnějších UTP kabelů pro Ethernet. Všechny uvedené kategorie jsou definovány pro vzdálenost do 100 metrů a impedance 100Ω. Maximální použitelná frekvence je uvedena pro nestíněnou kabeláž [4].

Tab. 1: Rozdělení kabeláže podle standardu TIA/EIA-568 [4]

Třída	Označení	Maximální frekvence
C	Kategorie-3 (CAT-3)	16MHz
D	Kategorie-5e (CAT-5e)	100MHz
E	Kategorie-6 (CAT-6)	250MHz

V tabulce 2 je uvedeno připojení konektoru RJ-45 k UTP kabelu podle standardu EIA-568. Pro verze 10/100BASE-T(X) nejsou piny 3, 4, 6, 7 použity, ale obvykle se ke konektoru připojují.

Limitním faktorem pro přenos na velké vzdálenosti po metalickém vedení je útlum, proto při přenosu na vzdálenosti větší než 100m je nutné použít opakovače. Další možností je použít jako přenosové médium optické vlákno. Vícevidové optické vlákno je vhodné do vzdáleností několika kilometrů. Pro vzdálenosti řádově stovek kilometrů je určeno vlákno jednovidové.

Tab. 2: Připojení UTP kabelu ke konektoru RJ-45 podle standardu TIA/EIA-568 [4]

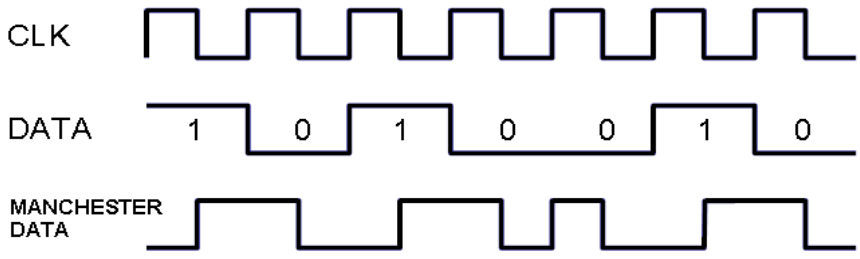
Pin na RJ-45	Barva kabelu	10/100BASE-T(X)	1000BASE-T
0	Bílo-oranžová	TX+	Bi0+
1	Oranžová	TX-	Bi0-
2	Bílo-zelená	RX+	Bi1+
3	Modrá	Nepoužito	Bi2+
4	Bílo-modrá	Nepoužito	Bi2-
5	Zelená	RX-	Bi1-
6	Bílo-hnědá	Nepoužito	Bi3+
7	Hnědá	Nepoužito	Bi3-

2.2.2Fyzická vrstva 10BASE-T

Přenos probíhá po jednom krouceném páru rychlostí 10Mbit/s. Data z linkové vrstvy jsou přes rozhraní MII vedena do fyzické vrstvy, kde jsou serializována, zakódována Manchester kódem a odeslána. Logické 1 je reprezentována napěťovou úrovní +2,5V, logická 0 napěťovou úrovní -2,5V.

KÓDOVÁNÍ MANCHESTER

Slučuje hodinový signál a data do jednoho bitového toku. Při každém přeneseném bitu dochází ke změně, to umožňuje snadnou obnovu hodinového taktu. Logická 0 je reprezentována sestupnou a logická 1 náběžnou hranou. Pro přenesení dat určitou rychlostí je vždy třeba dvojnásobná propustnost kanálu, tzn. pro přenos rychlostí 10Mbit/s je třeba propustnost kanálu 20Mbit/s. Příklad kódování Manchester ukazuje obrázek 3. Je možné jej realizovat hradlem XOR, kdy je na jeden vstup přiveden vzorkovací hodinový signál a na druhý bitový tok [5], [6].

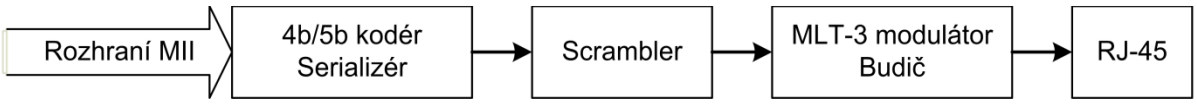


Obr. 3: Kódování Manchester

2.2.3Fyzická vrstva 100BASE-TX

Označuje se *Fast Ethernet.* Přenos probíhá rychlostí 100Mbit/s po dvou kroucených diferenciálních párech. Každý pár přenáší data jedním směrem. Vyžaduje kabel třídy CAT-5 nebo vyšší. Pro zakódování dat na symboly se používá kód 4B/5B. Bitový tok z kodéru

je logickou operací *xor* sečten s výstupem scrambleru a modulován tří úrovňovou modulací MLT-3. Blokové schéma vysílače je na obrázku 4.



Obr. 4: Vysílač 100BASE-TX [1]

KÓDOVÁNÍ 4B/5B

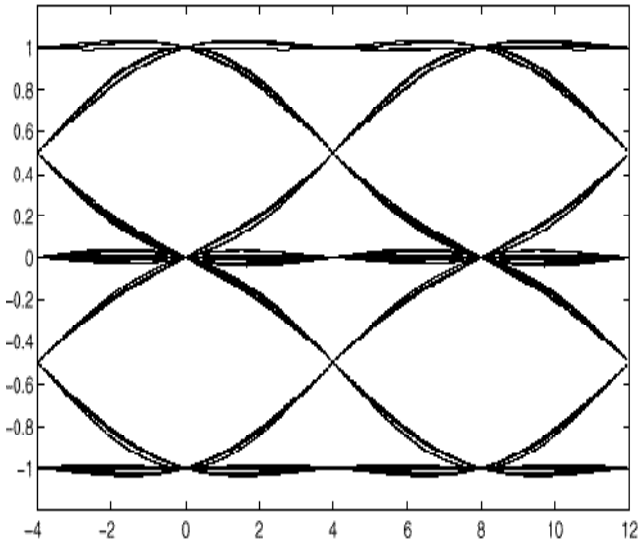
Každému 4bitovému slovu je přiřazeno slovo 5bitové (symbol). Dostupných je 32 symbolů, to umožňuje obsáhnout všech 16 vstupních znaků a vytvořit speciální symboly pro řízení přenosu. Vybírány jsou záměrně symboly, obsahující dostatečné množství přechodů pro spolehlivé obnovení hodinového signálu. Modulační rychlost je 125Mbaud. Přehled symbolů je uveden v tabulce 3 [5], [6].

Tab. 3: Symboly 4b/5b kódování [5]

Symbol	4b	5b	Funkce	Symbol	4b	5b	Funkce
0	0000	11110	Data 0_{HEX}	C	1100	11010	Data C_{HEX}
1	0001	01001	Data 1_{HEX}	D	1101	11011	Data D_{HEX}
2	0010	10100	Data 2_{HEX}	E	1110	11100	Data E_{HEX}
3	0011	10101	Data 3_{HEX}	F	1111	11101	Data F_{HEX}
4	0100	01010	Data 4_{HEX}	Q	-	00000	Quiet
5	0101	01011	Data 5_{HEX}	I	-	11111	Idle
6	0110	01110	Data 6_{HEX}	J	-	11000	Start #1
7	0111	01111	Data 7_{HEX}	K	-	10001	Start #2
8	1000	10010	Data 8_{HEX}	T	-	01101	End
9	1001	10011	Data 9_{HEX}	R	-	00111	Reset
A	1010	10110	Data A_{HEX}	S	-	11001	Set
B	1011	10111	Data B_{HEX}	H	-	00100	Halt

MODULACE MLT-3

Přenos jednotlivých bitů po diferenciálním páru používá modulaci MLT-3 (z ang. Multi Level Transmit). Jde o tříúrovňovou modulaci se symetrickými napěťovými úrovněmi 1V, 0V a -1V. Logická 0 na vstupu nemění úroveň signálu na výstupu. Logická 1 na vstupu mění výstupní signál o jednu úroveň podle posloupnosti 0V→1V→0V→–1V→0V. Průběh možných napěťových úrovní modulace ukazuje diagram oka na obrázku 5.



Obr. 5: Diagram oka MLT-3 modulace [7]

Výsledný signál lze aproximovat sinusovkou s frekvencí *f*. Z mechanismu modulace je jasné, že maximální frekvence f_{max} dosáhne sinusovka při logických 1 na vstupu modulátoru. V takovém případě dojde k dokončení cyklu sinusovky za 4 hodinové takty vzorkovacího signálu f_{vz}. Výpočet maximální frekvence pro modulační rychlost 125Mbaud udává rovnice (1). Proto je možné použít kabel CAT-5e pro rychlost přenosu 100Mb/s [5],[7].

$$f_{max} = \frac{f_{vz}}{4} = \frac{125}{4} = 31{,}25MHz \qquad (1)$$

SCRAMBLER

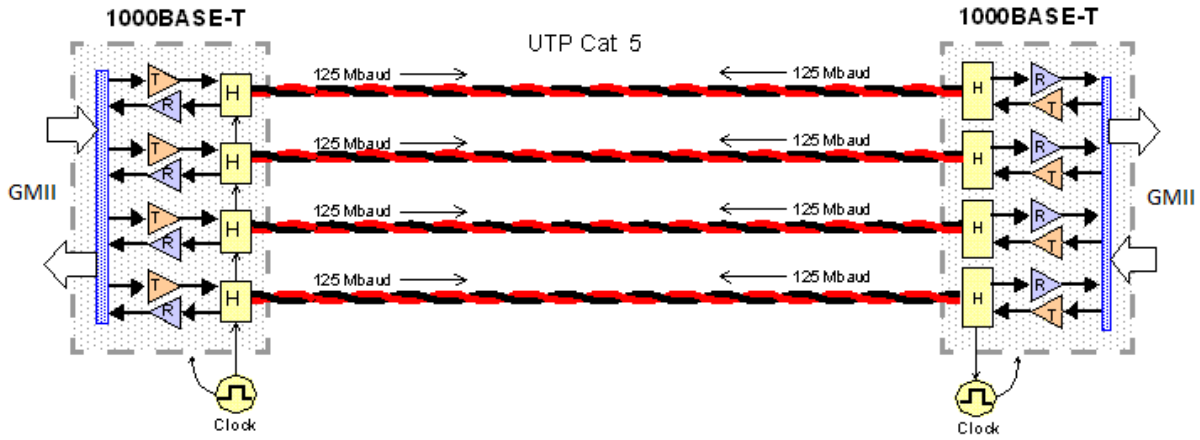
Kódování symbolů metodou 4b/5b v kombinaci s modulací MLT-3 způsobí rozložení spektra přenášeného signálu do těsné blízkosti frekvence f_{max} při symbolu *idle* $(11111)_b$. Tento symbol je vysílán při nečinnosti linky. To způsobí silné elektromagnetické vyzařování na frekvenci f_{max}. Pro potlačení vyzařování se používá scrambler, který rozprostře spektrum do širší oblasti a tím redukuje vyzařování na frekvenci f_{max}.

Scrambler je generátor pseudonáhodné sekvence bitů, realizovaný čítačem *LFSR* (z ang. Linear Feedback Shift Register). Výstupní posloupnost scrambleru je logickou operací xor sečtena s výstupní bitovou sekvencí kodéru 4B/5B. Takto upravený signál je modulován MLT-3 modulací. Při znaku *idle* $(11111)_b$ bude na vstupu modulátoru MLT-3 invertovaná pseudonáhodná sekvence výstupu scrambleru [6].

2.2.4Fyzická vrstva 1000BASE-T

Výhodou této implementace je to, že umožňuje jako přenosové médium použít UTP kabel typu CAT-5, určený pro verze 10BASE-T a 100BASE-TX. Využívá všechny čtyři páry. Každý pár umožňuje současný přenos v obou směrech. Symboly jsou modulovány čtyř

dimenzionální pětiúrovňovou amplitudovou modulací PAM-5 (z ang. Pulse Amplitude Modulation).

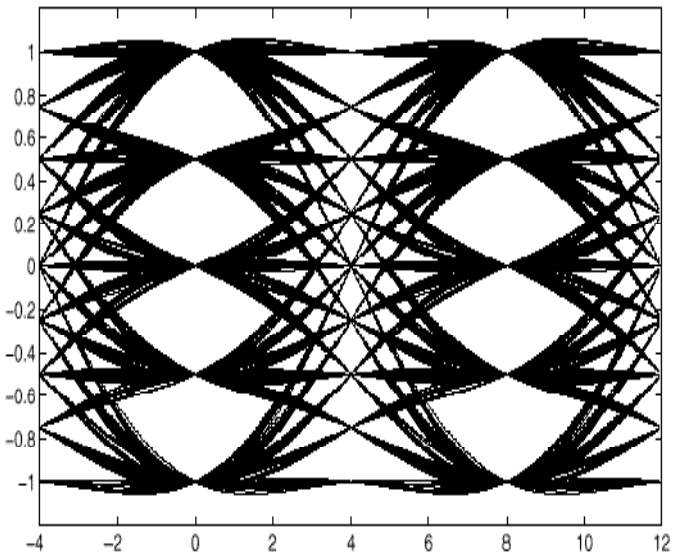


Obr. 6: Schéma obousměrného přenosu 1000BASE-T přes kabel CAT-5e [8]

Na obrázku 6 je zobrazen obousměrný přenos přes čtyři diferenciální kroucené páry. Hybridní obvody H zajišťují obnovení hodinového signálu, umožňují současně přijímat a odesílat po jedné lince. Musí podporovat potlačení vlastního vysílaného signálu tzv. echo. Bloky T a R implementují modulaci/demodulaci PAM-5, konvoluční kodér/dekodér a scrambler/descramler [8].

PAM-5 MODULACE

Pěti úrovňová amplitudová modulace. Při využití všech čtyř oddělených párů je k dispozici 5^4 (625) symbolů. Pro data se využívá pouze 256 symbolů, ostatní symboly jsou použity pro speciální znaky řídící přenos a pro korekci chyb.



Obr. 7: Diagram oka PAM-5 modulace [7]

Modulace umožňuje přenést v jednom hodinovém taktu 8bitovou informaci. K dosažení rychlosti 1Gb/s tedy stačí modulační rychlost 125Mbaud. Z diagramu oka na obrázku 7 je vidět, že při modulační rychlosti 125Mbaud dojde k dokončení cyklu sinusovky po 16ns. Požadavek na maximální frekvenci kabelu je tedy 62.5MHz. Kódování dat na symboly zajišťuje scrambler a konvoluční kodér, nejčastěji implementovaný signálovým procesorem [5], [6].

GMII ROZHRANÍ

Rozhraní GMII zajišťuje přenos dat mezi fyzickou a linkovou vrstvou. Přenášen je celý linkový rámec včetně preambule a CRC32. Pracovní frekvence sběrnice je 125MHz pro rychlost 1Gb/s, data jsou přenášeny po 8 bitech. Rozhraní GMII je zpětně kompatibilní s rozhraním MII (použito u 10/100Mbit/s), které používá pouze dolní 4 bity. Taktovací frekvence pro MII je 25 MHz při rychlosti 100Mbit/s a 2,5MHz při rychlosti 10Mbit/s.

Signály vysílače:

- *GTXCLK:* hodinový signál pro vysílání dat (125 MHz), zdrojem je MAC vrstva (vyhrazeno pouze pro rychlost 1 Gbit/s)
- *TXCLK:* hodinový signál pro 10/100 Mbit/s (2,5/25 MHz), zdrojem je fyzická vrstva
- *TXD[7:0]:* odesílaná data
- *TXEN*: odesílání povoleno
- *TXER:* chyba v datech, paket je neplatný

Signály přijímače:

- *RXCLK:* hodinový signál pro taktování přijatých dat (získán z přijímaného bitového toku)
- *RXD[7:0]:* přijatá data
- *RXDV:* přijatá data jsou platná
- *RXER:* přijatá data obsahují chybu
- *COL:* nastala kolize (pouze pro poloviční duplex)
- *CS:* sdílené médium je obsazeno (pouze pro poloviční duplex)

Signály pro nastavení fyzické vrstvy:

- *MDC:* hodinový signál
- *MDIO:* datový signál pro obousměrný přenos [3].

2.2.5Linková vrstva

Linková vrstva, někdy nazývaná vrstva datového spoje, dělí vstupní bity do datových bloků (paketů), přesně dané struktury. Každý blok se skládá z hlavičky, datové části a kontrolní sekvence. Komunikuje přímo s fyzickou vrstvou přes rozhraní nezávislé

na přenosovém médiu (MII/GMII). Zajišťuje kontrolu toku dat speciálním typem rámce. Kontrola toku je často implementována i vyššími protokoly. V případě polovičního duplexu řeší přístup ke sdílenému médiu a detekci kolizí algoritmem CSMA/CD (z ang. Carrier Sense Multiple Access / Collision Detection) [1].

POLO-DUPLEXNÍ PROVOZ

Sdílené médium se dnes používá především u rádiových sítí. Jedno přenosové médium sdílí více stanic. Vždy může vysílat pouze jedna stanice a ostatní naslouchají. Pokud stanice nevysílá, monitoruje provoz na médiu. V případě, že některá stanice vysílá, nastaví fyzická vrstva příznak *Carrier Sense* do logické 1. Pokud začne vysílat více stanic najednou, dojde k interferenci paketů a výsledný bitový tok je detekován jako kolize.

Zařízení, které chce vysílat, zjistí, zda je komunikační kanál volný. Pokud ano, začne vysílat, v opačném případě čeká na jeho uvolnění. Odesílatel naslouchá, zda nedošlo ke kolizi. Pokud ano odešle signál *JAM* a odmlčí se na náhodnou dobu, po které se pokusí rámec odeslat znovu [3].

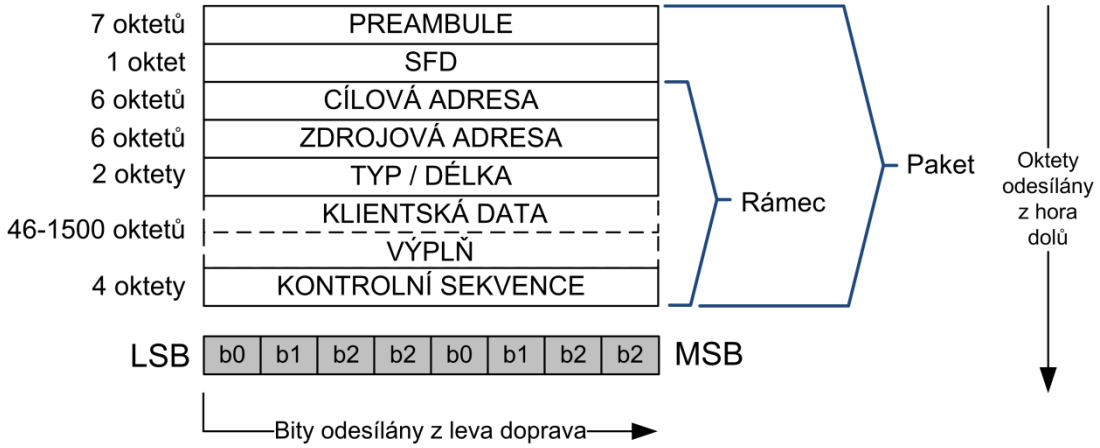
PLNĚ-DUPLEXNÍ PROVOZ

V současnosti nejpoužívanější mód Ethernetových sítí typu hvězda. Plně duplexní provoz dovoluje stanicím současně vysílat i přijímat data. Např. Ethernet rychlosti 100Mbit/s umožňuje odesílat data rychlostí 100Mbit/s a zároveň je touto rychlostí přijímat. Datová propustnost tedy vzroste oproti polovičnímu duplexu dvojnásobně. Nedochází k potenciálním kolizím, což se promítne ve zvýšení efektivity přenosu. Plně duplexní provoz musí umožňovat přenosové médium bez vzájemné interference signálů [3].

PAKET LINKOVÉ VRSTVY

Na obrázku 8 je struktura paketu linkové vrstvy standardu IEEE802.3 a Ethernet II. Liší se od sebe polem, udávající délku/typ. Pokud je v tomto poli hodnota do čísla 1500_{DEC}, definuje pole délku a jde o rámec IEEE802.3. V rámci typu Ethernet II jsou hodnoty pole typ větší než 1500_{DEC} (je to kvůli maximální délce rámce) a definují typ vyššího protokolu. Hodnoty pole *typ* předepisuje organizace IANA (z ang. Internet Assigned Numbers Authority), seznam je dostupný online [10].

Největší dovolená délka datové oblasti linkového rámce se označuje MTU (z ang. Maximum Transmission Unit). Mezi odesílané pakety se vkládá minimálně 12bajtová mezi-rámcová mezera označována IFG (z ang. InterFrame Gap) [3].



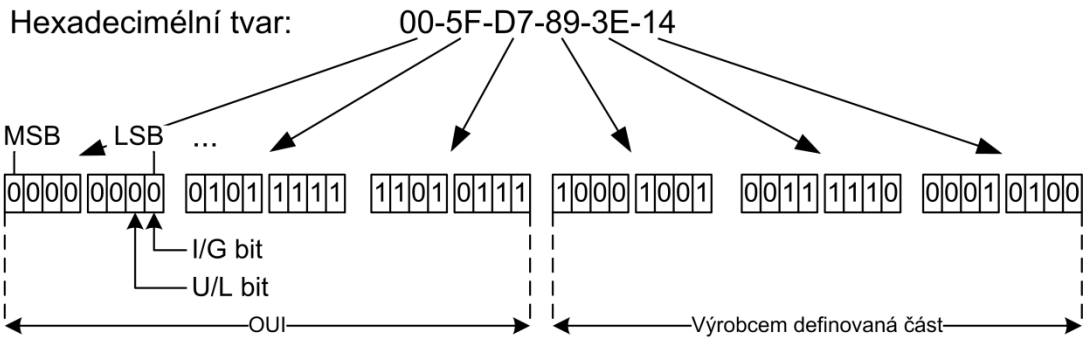
Obr. 8: Paket linkové vrstvy podle standardu IEEE802.3 [3]

Popis datových polí linkového paketu:

- *Preambule:* Synchronizuje hodinový signál a data. U nových verzí je pole obsaženo v rámci pouze z důvodu zpětné kompatibility. Obsahuje 7 oktetů hodnoty 10101010_{BIN}.
- *SFD* (z ang. Start Frame Delimite)*:* Určuje začátek rámce. Hodnota pole je 10101011_{BIN}.
- *Cílová adresa:* 48bitová MAC adresa cílového rozhraní.
- *Zdrojová adresa:* 48bitová MAC adresa odesílatele.
- *Typ / délka:* Rámec Ethernet II vyplní typ vyššího protokolu. Rámec IEEE802.3 vyplní délku datové části.
- *Klientská data:* Data délky 46 – 1500 oktetů. Někteří výrobci podporují delší rámec, tzv. Jumbo frame, za účelem zvýšení datové propustnosti. Tyto rámce však nejsou standardizované.
- *Výplň:* Pokud jsou odesílaná data menší než 46B, doplní se pole nulami.
- *Kontrolní sekvence:* 32bitová kontrolní sekvence počítaná algoritmem CRC32.

MAC ADRESY

Linková vrstva používá k identifikaci stanic 48bitové MAC adresy. Teoretický počet adresátů je 2^{48}. Adresa se zapisuje v hexadecimálním tvaru, jednotlivé oktety se pro přehlednost oddělují separačním znakem. Na obrázku 9 je uveden význam jednotlivých bitů MAC adresy.



Obr. 9: Struktura MAC adresy [3]

Dva nejnižší bity adresy mají speciální význam:

I/G bit se používá pro identifikaci cílové adresy jako individuální nebo skupinové. Logická 0 indikuje cílovou adresu jako individuální. Ta může být v jedné síti pouze jedna (unicastová adresa). Logická 1 značí adresu pro více cílů (jsou to multicastové a broadcastové adresy).

U/L bit definuje globálně nebo lokálně spravovanou adresu. Logická 1 označuje lokálně spravovanou adresu. Ta musí být unikátní v rámci lokální sítě, nikoli však celosvětově. Naproti tomu globálně spravované adresy jsou celosvětově unikátní. V takovém případě je výrobcům přidělen jednoznačný identifikátor OUI (z ang. Organizationally Unique Identifier), zbylý adresní prostor (horní 3 bajty) mohou výrobci použít pro svá zařízení. OUI spravuje a přiděluje organizace IEEE. Seznam výrobců je dostupný online [3], [9].

2.3 Síťová vrstva

Síťová vrstva je implementována protokolem IP, který zajišťuje nespojovanou a nespolehlivou službu. Pokud spolu chtějí komunikovat dvě zařízení na úrovni síťové vrstvy, musí své pakety balit do datového pole linkových rámců. Problém nastává, pokud je IP paket větší než maximální datová část Ethernetového rámce.

Přidělování a spravování IP adres zajišťuje organizace IANA. V současnosti existují dvě verze IP protokolu – IPv4 a IPv6.

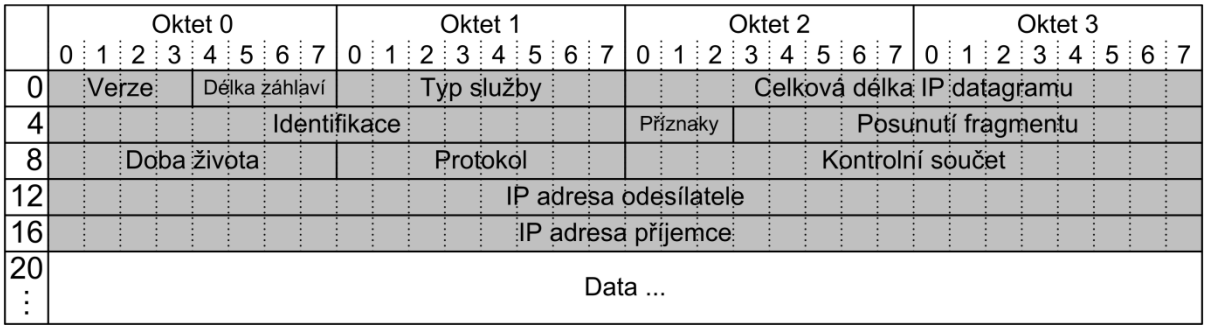
Verze 4 adresuje účastníky na základě 32 bitových adres. Teoretické množství adresátů je 2^{32} (4.294.967.296 adresátů). Praktické množství je menší, protože některé adresy jsou vyhrazeny pro speciální účely. V současnosti je adresní prostor IP protokolu verze 4 vyčerpán, dne 1.2.2011 byly organizací IANA předděleny poslední adresy [10].

Z důvodu nedostatečného množství adres byl vyvinut IPv6, který adresuje účastníky 128bitovou adresou. Teoretický adresní prostor je 2^{128} a v současnosti se zdá jako

nevyčerpatelný. Dalším rozdílem je, že hlavička IPv6 nepoužívá kontrolní součet a neimplementuje fragmentaci paketů, která je nežádoucí [1].

2.3.1 Protokol IPv4

Protokol je popsán v RFC791 [11]. Skládá se z 20 bajtové hlavičky, po níž následuje datová část. Hlavička může mít i více bajtů pro některé rozšířené funkce, ale některé směrovače nemusí rozšíření podporovat a takto modifikované pakety budou zahazovat. Strukturu paketu ukazuje obrázek 10.



Obr. 10: Formát paketu IPv4

Popis datových polí IPv4 paketu:

- *Verze:* verze IP protokolu = 4.
- *Délka záhlaví*: délka hlavičky v 32 bitových slovech = 5.
- *Typ služby*: pole je někdy využíváno k zajištění kvality služeb QoS.
- *Délka IP datagramu:* Celková délka včetně hlavičky v bajtech.
- *Identifikace:* pokud dojde k fragmentaci, slouží pole k identifikaci paketů patřících do jednoho bloku.
- *Příznaky:* řídí fragmentaci nebo ji zakazují.
- *Posunutí fragmentu:* při fragmentaci udává pozici fragmentu v původním paketu.
- *Doba života*: slouží k ochraně sítě před cyklujícími pakety. Při každém průchodu směrovačem je hodnota pole snížena o jedničku. V případě že je doba života nulová, je paket zahozen. Doba života paketu se může lišit pro různé operační systémy.
- *Protokol:* určuje protokol vyšší vrstvy. Jednotlivá čísla protokolů spravuje organizace IANA.
- *Kontrolní součet:* kontrolní sekvence, počítá se pouze z hlavičky.
- *Adresa odesílatele:* 32bitová IPv4 adresa.
- *Adresa příjemce:* 32bitová IPv4 adresa.
- *Data*: datová oblast, obvykle nese paket vyšší (transportní) vrstvy. [11]

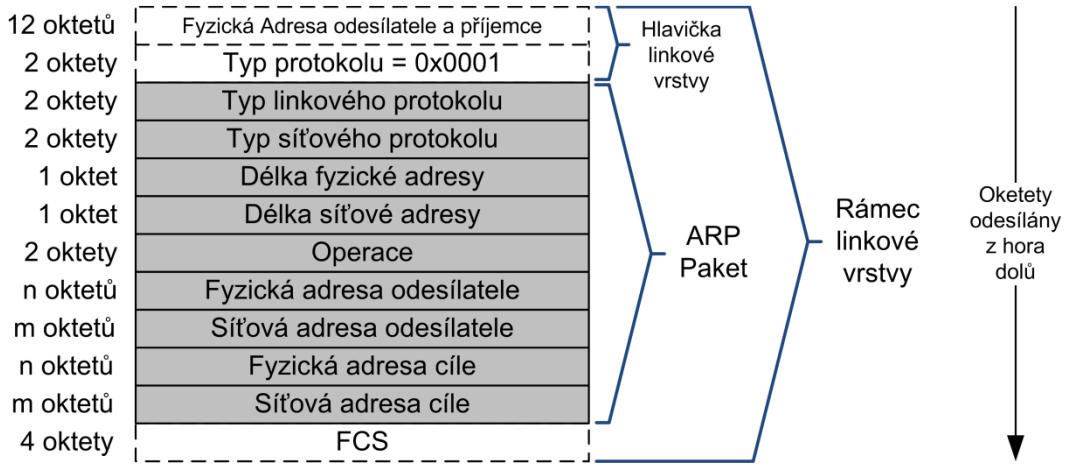
Pro jednoznačnou identifikaci cílové stanice je nutné znát nejen adresu síťového rozhraní (IP adresu), ale i adresu linkového rozhraní (MAC adresu). Mapování síťových adres na linkové adresy zajišťuje konkrétní implementace protokolu.

Jednou z možností je statické přiřazení adres, kdy se vytvoří pevně definovaná tabulka, ve které je ke každé síťové adrese, přiřazena příslušná linková adresa. Odesílatel vyhledá v tabulce podle cílové IP adresy MAC adresu a doplní ji do hlavičky linkového paketu.

Další z možností je použít protokol ARP (z ang. Address Resolution Protocol), který umožňuje aktualizaci této tabulky dynamicky formou dotazů. Výhodou tohoto řešení je snadná přenositelnost stanice do jiné sítě bez nutnosti ručně konfigurovat ARP tabulku. Nevýhodou je nízká bezpečnost a náchylnost na útoky typu ARP spoofing [2], [11].

2.3.2 Protokol ARP

Protokol je specifikován v RFC826 [12]. Řeší překlad síťové adresy na linkové adresy formou dotazů. ARP paket je balen přímo do rámce linkové vrstvy, proto je nezávislý na IP protokolu a je možné ho použit i v jiných typech sítí. V praxi je však spojován výhradně s protokolem IPv4. Může být provozován pouze v lokální síti [1]. Formát paketu ARP a způsob jeho balení do linkového rámce ukazuje obrázek 11.



Obr. 11: Formát ARP paketu

Popis datových polí ARP paketu:

- *Typ linkového protokolu:* specifikuje linkový protokol. Pro EthernetII je vyhrazeno číslo 0001_{HEX}. Seznam čísel je dostupný na stránkách organizace IANA [9].
- *Typ síťového protokolu:* používá stejná značení jako linkový protokol Ethernet II. Pro IP protokol je vyhrazeno číslo 0800_{HEX}.

- *Délka fyzické a síťové adresy:* určuje počet oktetů fyzické adresy (pro 48 bitovou MAC adresu n = 6). Délka síťové adresy určuje počet oktetů síťové adresy (pro IPv4 m = 4).
- *Operace:* specifikuje typ paketu. V případě ARP dotazu je hodnota pole 01_{HEX}, pro odpověď je hodnota nastavena na 02_{HEX}.
- *Fyzická a síťová adresa odesílatele a cíle:* MAC a IP adresy [12].

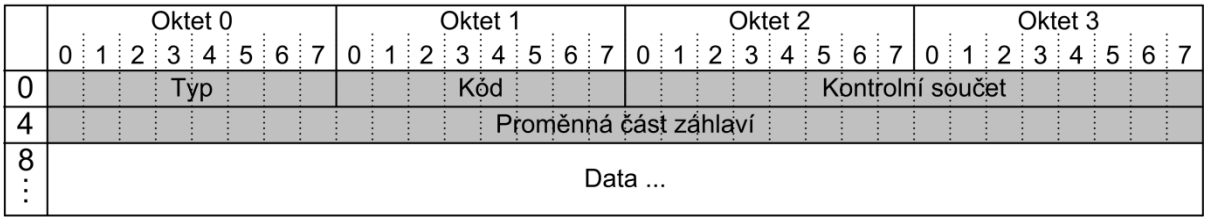
POPIS PŘEKLADU ADRES

Pro zjištění cílové MAC adresy se do lokální sítě se odešle paket typu *ARP request*, zabalený do linkového rámce, jako oběžník (adresa FF:FF:FF:FF:FF:FF). Tím je zajištěno, že dotaz obdrží všechny stanice v síti. V dotazu je vyplněno pole fyzické i síťové adresy odesílatele a síťové adresy cílové stanice. Fyzická adresa cílové stanice je vyplněna nulami. Všechny stanice v síti přijmou tento paket a porovnají pole cílové síťové adresy se svou IP adresou. Pokud se shodují, doplní cílová stanice pole pro fyzickou adresu, změní pole *operace* z dotazu na odpověď a odešle paket zpět na fyzickou adresu, ze které dotaz obdržela.

Obě stanice si zapíší (příjemce z dotazu, odesílatel z odpovědi) do své ARP tabulky síťovou a k ní příslušnou fyzickou adresu protější stanice. Životnost položek v tabulce závisí na konkrétní implementaci, řádově se pohybuje v jednotkách až desítkách minut [1], [12].

2.3.3Protokol ICMP

ICMP (z ang. Internet Control Message Protocol) je součástí protokolu IP, slouží k informování uživatele o událostech v síti. Na rozdíl od protokolu ARP je balen až do IP datagramu, proto se chová jako protokol vyšší vrstvy. Pomocí ICMP je možné signalizovat nejrůznější situace v síti, skutečnost je však taková, že konkrétní implementace podporují vždy jen jistou část těchto signalizací. Z bezpečnostních důvodů mohou být na směrovačích mnohé ICMP zahazovány. Struktura ICMP paketu ukazuje obrázek 12. Hlavička je vždy 8 bajtů dlouhá, proměnná část závisí na typu paketu [1].



Obr. 12: Formát ICMP paketu

Popis datových polí ICMP paketu:

- *Typ:* je hrubým dělením ICMP paketů, závisí na něm proměnná část záhlaví.
- *Kód:* definuje konkrétní zprávu (jemné dělení).
- *Kontrolní součet:* je 16bitová zabezpečovací sekvence, počítá se ze všech polí paketu.
- *Proměnná část záhlaví:* Obsah horních 4 bajtů závisí na typu paketu. Paket typu Echo obsahuje v tomto poli 16bitový identifikátor a 16bitové sekvenční číslo [13].

2.4 Transportní vrstva

Transportní vrstva zajišťuje komunikaci mezi aplikacemi nebo procesy. Vrstva poskytuje dva protokoly – TCP (z ang. Transmission Control Protocol) a UDP (z ang. User Datagram Protocol).

Protokol TCP poskytuje *spojovanou a spolehlivou službu* – před odesláním dat je nejprve navázáno spojení s cílovým počítačem a veškerá odeslaná data musí být potvrzena cílovým PC, takže je zajištěno jejich doručení.

Nespojovaná a nespolehlivá služba je realizována protokolem UDP. Data jsou odeslána na cílovou adresu a port, ale není zajištěno jejich doručení. V multimediálních aplikacích je to výhodné, protože data jsou požadována v reálném čase i za cenu určité ztrátovosti paketů. Pokud je vyžadován spolehlivý charakter přenosu, je nutné spolehlivost zajistit aplikací.

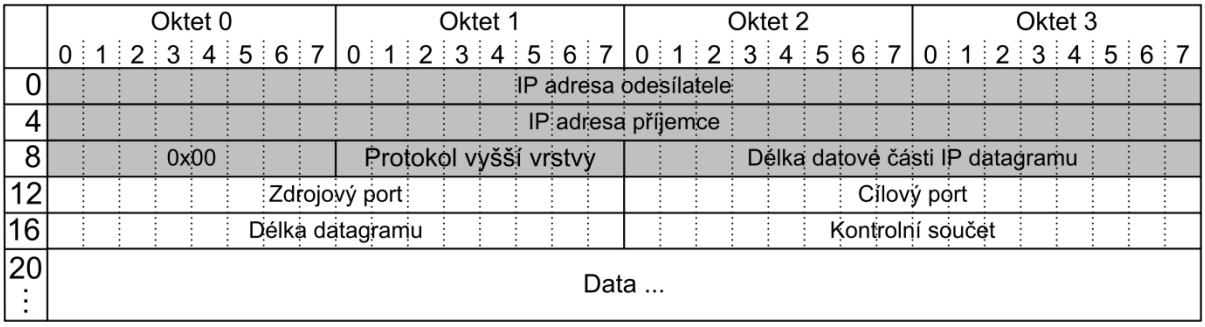
Číslem portu je jednoznačně identifikována aplikace, pro kterou jsou data určena. Porty protokolů UDP a TCP jsou na sobě nezávislé, obvykle však pro stejnou službu nesou stejné číslo. Port je 16bitové číslo obsažené v hlavičce paketu transportní vrstvy [1], [10].

Porty můžeme rozdělit na tři typy:

- *Známé (rozmezí 0-1023):* jsou vyhrazeny pro systémové procesy. Hodnoty definuje IANA a je možné je vyhledat online [10].
- *Registrované (1024-49151)*: jsou určeny pro běžné uživatelské aplikace. Porty registruje IANA, jsou dostupné online [10].
- *Dynamické (49152-65535)*: nejsou nikde registrovány, aplikace si je volí dynamicky. Používají se pro komunikaci klienta se serverem, kdy více klientů na jedné IP adrese může používat stejnou aplikaci [1], [10].

2.4.1 Protokol UDP

UDP je protokol transportní vrstvy sloužící pro přenos uživatelských dat. Je zabalen do datové části IP paketu. Na obrázku 13 je pseudohlavička pro výpočet kontrolního součtu UDP paketu. Vlastní UDP paket je znázorněn bílou barvou, začíná oktetem 12.



Obr. 13: Pseudo-hlavička pro výpočet kontrolního součtu UDP paketu

Popis polí UDP paketu:

- *Zdrojový port:* je nepovinný, identifikuje aplikaci, která paket odeslala. V případě, že není použit zdrojový port, je tato hodnota nastavena na 0.
- *Cílový port:* je povinný, definuje cílovou aplikaci, pro kterou je paket určen.
- *Délka:* určuje celkovou délku UDP paketu včetně hlavičky.
- *Kontrolní součet:* je nepovinný, ale z důvodu nižší spolehlivosti protokolu, se ve většině případů používá [14].

KONTROLNÍ SOUČET

Opět se používá 16bitová kontrolní sekvence. Pokud není použit kontrolní součet, nastaví se jeho hodnota v hlavičce na 0000_{HEX}. V opačném případě se sekvence počítá z upravené pseudo hlavičky a datové části paketu. Pokud datová část končí lichým oktetem, doplní se zbývající pole fiktivní hodnotou 00_{HEX}. Na obrázku 13 jsou zobrazena pole paketu, ze kterých se kontrolní součet počítá. Zvýrazněna jsou pole z IP datagramu. Pro výpočet se za pole kontrolního součtu dosazuje hodnota 0000_{HEX} [14].

2.5 Zajištění integrity dat

Každý moderní komunikační protokol používá jeden nebo více algoritmů pro detekci chyb v přenosu. Základní rámec Ethernetu používá 32bitovou cyklickou redundantní kontrolu – CRC (z ang. Cyclic Redundancy Check). Vyšší vrstvy síťového modelu používají nejčastěji klasický kontrolní součet (checksum). Další možností kontroly dat jsou paritní bity, hash funkce apod.

2.5.1Cyklická redundantní kontrola

Cyklická redundantní kontrola (CRC) je velmi často používaná metoda pro zabezpečení přenosu dat. Princip metody vychází z dělení polynomu polynomem, resp. dělení zprávy charakteristickým polynomem GP(x). Výhodou CRC je velká rychlost algoritmu, snadná implementace a schopnost detekovat určitý typ chyb vhodnou volbou dělícího polynomu.

Vlastnosti CRC zabezpečení jsou závislé na *(n+1)bitovém* dělícím polynomu GP(x). Bit *GP(n+1)* je vždy jedničkový a slouží k určení délky polynomu. Polynom GP(x) předepisuje komunikační protokol. Norma IEEE802.3 předepisuje pro zabezpečení přenosu polynom (2), nejčastěji se zapisuje v hexadecimálním tvaru - $04C11DB7_{HEX}$.

$$GP(x)=x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+x+1 \qquad (2)$$

CRC detekuje:

- Všechny jednobitové chyby
- Všechny dvoubitové chyby, pokud GP má aspoň 2 členy
- Všechny liché počty chyb, pokud GP obsahuje *x+1*
- Všechny shluky chyb do délky *n* (délka FCS)
- Shluky chyb délky větší než *n* neodhalí s pravděpodobností $1/2^n$ [1].

M(X) k-bitová zpráva	FCS(X) n-bitová kontrola

Obr. 14: Ukázka připojení kontrolní sekvence ke zprávě [1]

Odesílaný rámec se skládá ze zprávy M(x) a kontrolní sekvence FCS(x). Jeho celková délka je *k* + *n* bitů. Princip výpočtu znázorňuje rovnice (3), [1].

$$\frac{X^n \cdot M(x)}{GP(x)} = Q(x) + \frac{FCS(x)}{GP(x)} \qquad (3)$$

Vysílač nejprve vynásobí zprávu k odeslání hodnotou x^n (z bitového hlediska jde o posun zprávy vlevo o n-bitů). Vzniklý polynom je následně vydělen polynomem GP(x). Zbytek po dělení je přiložen na konec zprávy jako kontrolní sekvence FCS (z ang. Frame Check Sequence). Linkový rámec T(x) je odeslán ve tvaru:

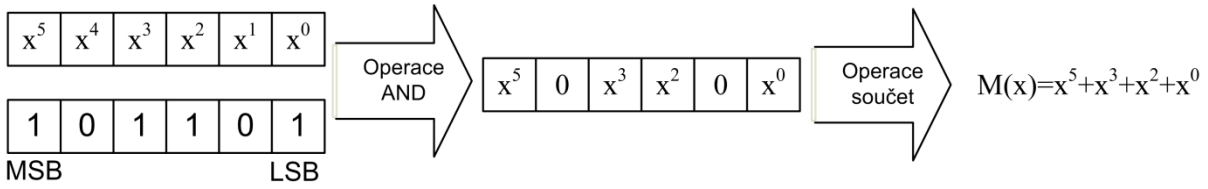
$$T(x) = x^n \cdot M(x) + FCS(x) \qquad (4)$$

Přijímač po přijetí rámce T(x) oddělí zprávu M(x) od kontrolní sekvence FCS(x) a provede pro ni výpočet CRC. Zbytek po dělení porovná s přijatým FCS(x). Další možností

přijímače ověřit správnost dat, je vydělit celý přijatý rámec polynomem GP(x). V případě, že přenesená data neobsahují chybu, je zbytek po dělení nulový [1].

BINÁRNÍ INTERPRETACE POLYNOMU

Při přenosu dat se samozřejmě nepřenášejí polynomy, ale sekvence bitů. Bitový tok lze zapsat jako polynom. Princip ukazuje obrázek 15. Vynásobením každého bitu sekvence hodnotou x^i, kde *i* je pozice bitu v posloupnosti, získáme jednotlivé koeficienty polynomu. Pro nejnižší bit (LSB) je $i = 0$. Sečtením koeficientů dostaneme ekvivalentní polynom.



Obr. 15: Převod bitové sekvence na polynom

Proces výpočtu dělení polynomů lze efektivně implementovat pomocí posuvného registru a hradel XOR. Tato implementace vyžaduje sériová vstupní data. Zvyšováním datové propustnosti rostou i požadavky na frekvenci taktovacího signálu a rychlost obvodů. Při datové propustnosti 1Gbit/s je požadavek na taktování posuvného registru 1GHz, to je problémové i pro moderní obvody FPGA. Zvýšení efektivity výpočtu CRC lze dosáhnou paralelizací algoritmu.

TABULKOVÁ METODA VÝPOČTU CRC

U paralelního výpočtu je nutné uvažovat kromě parametrů vlastního CRC i šířku datové sběrnice. Mezi jednu z nejrychlejších implementací patří tabulkový výpočet. Využívá lineární vlastnosti CRC mechanismu, který ukazuje rovnice (5) [18].

$$CRC(A+B) = CRC(A)+CRC(B) \quad (5)$$

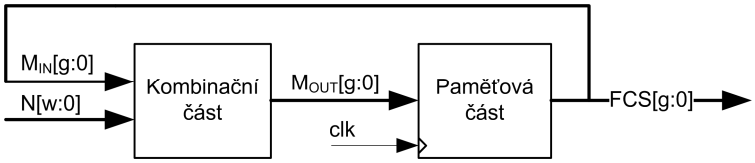
Vstupní data jsou zpracovávána po *N* bitech. Pro všechny možné kombinace vstupních dat o šířce *N* je v tabulce (např. v ROM paměti) uložena hodnota zbytku po dělení. Zbytek po dělení se přečte z tabulky pro každá vstupní paralelní data a sečte se s hodnotou zbytku po dělení předchozího stavu. Velikost ROM paměti závisí na šířce vstupních dat *M* a délce kontrolní sekvence *N*. Například pokud je šířka datové sběrnice 8 bitů ($M = 8$) a protokol Ethernetu definuje 32bitové CRC ($N = 32$), je velikost potřebné ROM paměti dána vztahem (6).

$$ROM = N \cdot 2^M = 32 \cdot 2^8 = 8192 bitů = 1kB \quad (6)$$

Pro šířku sběrnice M = 16 vzroste velikost paměti na 262kB. Pro takto široké sběrnice je implementace pomocí tabulky nevhodná.

METODA XOR VÝPOČTU CRC

Tato metoda má oproti tabulkovému výpočtu tu výhodu, že k její implementaci není nutná paměť pro předpočítané hodnoty. Hodnota každého bitu výstupní sekvence je dána kombinační logickou funkcí vstupních dat *N* a hodnotě kontrolní sekvence v předchozím stavu M_{IN}. Blokové schéma paralelního CRC generátoru je na obrázku 16.



Obr. 16: Blokové schéma paralelního CRC generátoru [18]

Metoda opět využívá lineárních vlastností CRC. Generátor se skládá z kombinační a paměťové části. *N* je vstupní datová sběrnice o šířce *w*. M_{OUT} je výstupní kontrolní sekvence a M_{IN} je kontrolní sekvence předchozího stavu o bitové šířce dělícího polynomu GP.

Pro určení bitových rovnic výstupu M_{OUT} je vhodné, v libovolném programovacím jazyce, implementovat sériový výpočet CRC s možností nastavit počáteční hodnotu posuvného registru. Následující funkce v jazyce C ukazuje implementaci sériového CRC32 Ethernetu s 8bitovým vstupem pro data. Funkce slouží jako CRC kalkulátor.

Zdrojový kód sériového výpočtu CRC32 z 8 bitové datové sekvence v jazyce C:

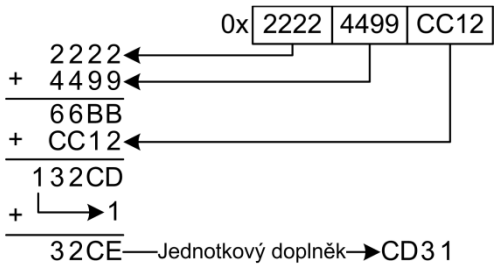
```
// Implementace sériového CRC32 s nastavitelnou hodnotou posuvného registru
unsigned int crc32_calculate(unsigned char N, unsigned int M)
{
   unsigned int crc = M ^ (N << 24);     // Nastaveni posuvneho registru
   for (int i = 0; i < 8; i++) {
      if (crc & (1 << 31)) {             // Pokud je MSB v registru 0
         crc = (crc << 1) ^ 0x04c11db7; // posun a xor s polynomem
      } else {                           // jinak
         crc = crc << 1;                 // jenom posun
      }
   }
   return crc;
}
```

Vypočítáme CRC pro všechny kombinace one-hot kódu na vstupu *N* s nulovou hodnotou registru M_{IN}. Tomu odpovídá volání funkce M_{OUT}*=crc32_calculate(one-hot,0).* Výsledné hodnoty převedeme do binárního kódu a zapíšeme do matice H_1, každý řádek reprezentuje jednu hodnotu one-hot kódu. Stejným způsobem sestavíme matici H_2 pro nulovou hodnotu vstupních dat *N* a všechny kombinace one-hot kódu na vstupu M_{IN}. Odpovídá tomu volání funkce M_{OUT}*=crc32_calculate(0,one-hot).* Řádky matic H_1 a H_2 jsou lineárně nezávislé, proto můžeme určit rovnici M_{OUT}. Každý sloupec tabulky reprezentuje

příslušný bit sběrnice M_{OUT}, řádky reprezentují vstupní proměnné. Vstupní bity ovlivňující příslušný výstup obsahují logickou 1 a je mezi nimi operace exkluzivní logický součet [18]. V příloze 1 je uveden příklad určení rovnice z matic H1 a H2 pro 8bitovou vstupní datovou sběrnici a 32bitové CRC.

2.5.2 Kontrolní součet

Kontrolní součet funguje na principu sčítání sousedních *n*-bitových slov ve zprávě. Parametr *n* udává bitovou šířku kontrolního součtu. Pro základní komunikační protokoly (tzn. IP, UDP a TCP) se v IP sítích se používá 16bitový kontrolní součet. Pro různé protokoly se jednotlivé implementace liší pouze ve slovech, které jsou do součtu zahrnuty. Na obrázku 17 je uveden postup výpočtu kontrolního součtu.



Obr. 17: Příklad výpočtu kontrolního součtu

Odesílaná zpráva se rozdělí na 16bitová slova, která jsou následně sečtena. Pokud součet překročí 16bitový rozsah, je překročená hodnota oddělena a připočítána ke zbytku. Z konečného součtu se inverzí bitů vypočítá jednotkový doplněk, který se použije jako kontrolní sekvence k odesílané zprávě.

Přijímač vypočítá obdobným způsobem součet. Do součtu zahrne i kontrolní sekvenci. Pokud je výsledek roven $FFFF_{HEX}$ byl přenos bezchybný. Kompletní popis výpočtu a jeho optimalizace je uveden v RFC1071 [15].

Mechanismus součtu neodhalí některé, vzájemně kompenzující se, periodické chyby. Výskyt neodhalitelné chyby zobrazuje obrázek 18. Tento typ chyby spolehlivě detekuje CRC algoritmus v linkové vrstvě.

$$\begin{array}{r|l} 1.oktet & 0101\ 0101 \\ 2.oktet & 0010\ 0110 \\ \hline Checksum & 0111\ 1011 \end{array} \xrightarrow{0x26,0x55} bitová\ chyba\ přenosu \xrightarrow{0x27,0x54} \begin{array}{l} 0101\ 0100 \\ 0010\ 0111 \\ \hline 0111\ 1011 \end{array}$$

Obr. 18: Příklad selhání detekce chyby kontrolním součtem

3 Praktická část

Komunikační rozhraní je vytvořeno ve formě modulu v jazyce VHDL. Pro syntézu a implementaci bylo použito vývojové prostředí ISE 12.1 od firmy Xilinx. Navržený modul byl testován na vývojové desce ML506 od firmy Xilinx, kterou ukazuje obrázek 19.

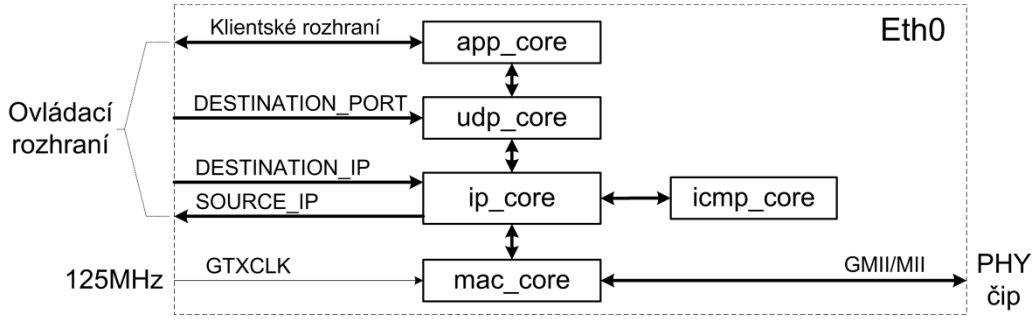
Vývojová deska ML506 obsahuje FPGA obvod Virtex5 s označením XC5VSX50T, ke kterému jsou připojeny ostatní periferie, mezi které patří např. LCD displej, USB rozhraní, konektor pro DDR2 paměti a jiné běžně používané periferie. Přehled všech periferií je uveden v technické dokumentaci [16].

Připojení k Ethernetu zajišťuje konektor RJ-45, který je přes oddělující transformátory připojen k čipu fyzické vrstvy 88E1111 od firmy Alaska. Obvod 88E1111 podporuje rychlosti 10Mbit/s, 100Mbit/s a 1Gbit/s. Umožňuje použít jako přenosové médium buď UTP kabel nebo optické vlákno (tato možnost není implementována na vývojové desce ML506). K cílovému obvodu je čip fyzické vrstvy připojen přes rozhraní GMII.

Obr. 19: Vývojová deska ML506

3.1 Hlavní modul eth0

Eth0 je nejvyšší vrstva, která spojuje všechny navržené bloky do jednoho celku. Jednotlivé nižší vrstvy jsou popsány v následujících kapitolách. Blokové schéma hlavního modulu ukazuje obrázek 20. Všechny protokolové vrstvy mají kvůli synchronizaci registrové výstupy a jsou řízeny náběžnou hranou hodinového signálu. Modul *eth0* vyžaduje pro správnou činnost vnější zdroj hodinového signálu o frekvenci 125MHz.



Obr. 20: Blokové schéma hlavního modulu *eth0*

Protokolové vrstvy mezi sebou komunikují přes 8bitovou datovou sběrnici. Pokud chce vyšší vrstva odeslat data do nižší vrstvy, přivede na výstupní sběrnici první oktet a jeho platnost určí signálem **_VLD* v logické 1. Vyšší vrstva musí vždy čekat na potvrzení načtení prvního oktetu nižší vrstvou (signál **_ACK* v logické 1), poté odešle zbývající data.

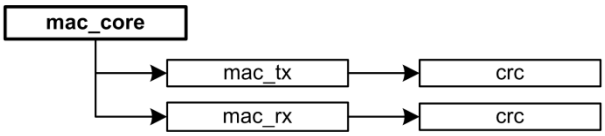
Modul *mac_core* implementuje linkovou vrstvu Ethernetu. Ta je připojena k externímu čipu fyzické vrstvy podle tabulky 4 a k modulu *ip_core,* který implementuje síťovou vrstvu. Na síťovou vrstvu navazuje vrstva transportní, realizovaná modulem *udp_core*. Modul *icmp_core* zajišťuje funkci ICMP Echo (Ping). Modul *app_core* implementuje jednoduchý potvrzovací protokol pro zabezpečení přenosu dat, který pro vlastní funkci není nezbytný, ale je vhodný při přenosu dat po síti Internet.

Tab. 4: Mapování vývodů pro připojení čipu fyzické vrstvy na vývojové desce ML506

Signál	Označení v GMII specifikaci	Pin obvodu Virtex-5	Napěťová úroveň
PHY_TXD[7..0]	TXD[7..0]	AG11, AG10, AH8, AG8, AH10, AH9, AE11, AF11	3,3V
PHY_TX_EN	TXEN	AJ10	3,3V
PHY_TX_ER	TXER	AJ9	3,3V
PHY_CS	CS	E34	3,3V
PHY_COL	COL	B32	3,3V
PHY_TX_CLK	GTXCLK	K17	2,5V
PHY_RXD[7..0]	RXD[7..0]	F33, D34, C34, D32, C32, C33, B33, A33	2,5V
PHY_RX_DV	RXDV	E32	2,5V
PHY_RX_ER	RXER	E33	2,5V
PHY_RX_CLK	RXCLK	H17	2,5V
PHY_RESET	RESET	J14	2,5V

3.2 Modul mac_core

Obvod Virtex5 obsahuje integrovanou hardwarovou komponentu TEMAC (tzv. Embedded Tri-Mode Ethernet MAC), která slouží pro implementaci linkové vrstvy Ethernetu [17]. Použití této komponenty zabraňuje přenositelnosti kódu na jiné cílové obvody, proto bylo navrženo vlastní řešení.



Obr. 21: Struktura modulu *mac_core*

Strukturu modulu *mac_core* ukazuje obrázek 21. Skládá se z bloku pro zpracování přijatých dat *mac_rx* a bloku pro zpracování odesílaných dat *mac_tx*, které pracují nezávisle na sobě. Logika v bloku *mac_core* detekuje rychlost sítě, generuje hodinové signály a zajišťuje převod mezi rozhraními GMII a MII. Blok *crc* zajišťuje výpočet cyklické redundantní kontroly. Podporovány jsou tři rychlosti (tzv. Tri-Mode Speed) 10/100/1000Mbit/s. Popis vstupních a výstupních signálů modulu *mac_core* je v následujících tabulkách.

Tab. 5: Hodinové signály bloku *mac_core*

Signál	Směr	Popis
GTXCLK	IN	Hlavní hodinový signál, požadovaná frekvence je 125 MHz.
PHY_TXCLK	IN	Hodinový signál pro řízení MII dat do PHY čipu (10 a 100Mb).
PHY_GTXCLK	OUT	Hodinový signál pro řízení GMII dat do PHY čipu (1Gb).
PHY_RXCLK	IN	Hodinový signál pro řízení MII a GMII dat z PHY čipu.
TX_CLK	OUT	Hodinový signál řídící data k odeslání z vyšších vrstev.
RX_CLK	OUT	Hodinový signál řídící přijatá data a zpracovaná data.

Tab. 6: Signály pro přenos dat mezi *mac_core* blokem a vyšší vrstvou

Signál	Směr	Popis
TX_DATA[7:0]	IN	Sběrnice odesílaných dat řízena hodinovým signálem TX_CLK.
TX_DATA_VLD	IN	Logická 1 určuje platnost dat na sběrnici TX_DATA.
TX_DATA_ACK	OUT	Potvrzení přečtení prvního oktetu (další oktety se už nepotvrzují).
RX_DATA [7:0]	OUT	Sběrnice přijatých dat řízena hodinovým signálem RX_CLK.
RX_DATA_VLD	OUT	Logická 1 určuje platnost dat na sběrnici RX_DATA.
RX_DATA_GOOD	OUT	Potvrzuje, že přijatý paket neobsahuje chybu (po ověření crc32).
RX_DATA_BAD	OUT	Potvrzuje, že přijatý paket obsahuje chybu a je neplatný.

Tab. 7: Signály pro přenos dat mezi *mac_core* blokem a PHY čipem

Signál	Směr	Popis
PHY_TXD [7:0]	OUT	GMII sběrnice pro odesílaná data.
PHY_TXEN	OUT	Logická 1 určuje platná data na sběrnici PHY_TXD.
PHY_TXER	OUT	Logická 1 označuje odesílaná data jako neplatná.

PHY_RXD [7:0]	IN	GMII sběrnice pro přijatá data.
PHY_RXDV	IN	Logická 1 určuje platná data na sběrnici PHY_RXD.
PHY_RXER	IN	Logická 1 určuje chybu v přijatých datech.
PHY_COL	IN	GMII signál pro detekci kolize (pouze pro half-duplex).
PHY_CS	IN	GMII signál pro přístup k médiu (pouze pro half-duplex).

Tab. 8: Konfigurační a informační signály

Signál	Směr	Popis
SPEED1G	OUT	Detekovaná rychlost Ethernetu (1 = 1Gb, 0 = 100/10Mb).
COLLISION	OUT	Detekce kolize. Aktivní úroveň logická 1 (pouze pro half-duplex).
IFG [7:0]	IN	Definice mezi-rámcové mezery (násobky 12).
FULL_DUPLEX	IN	Nastavuje režim plného duplexu v logické 1.
RESET	IN	Globální synchronní reset aktivní v logické 1.

Tabulka 8 obsahuje signály pro konfiguraci. MAC adresu modulu není možné po implementaci měnit, ale je definovatelná před vlastní syntézou v souboru *my_constants.vhd* konstantou:

```
constant FPGA_MAC: std_logic_vector(47 downto 0);          -- mac adresa jádra
```

3.2.1 Taktování jádra

Obvody FPGA obvykle neobsahují generátor hodinového signálu, proto je nutné celé jádro řídit hodinovým signálem z vnějšího zdroje. Vrstva *mac_core* generuje hodinové signály pro všechny vyšší vrstvy z referenčního hodinového signálu *GTXCLK*.

GTXCLK: Vstupní referenční hodinový signál, je důležitý pro určení rychlosti Ethernetu. Při rychlosti 1Gb/s je zdrojem řídícího signálu pro vysílací část fyzické vrstvy. Požadovaná frekvence je 125MHz. Vývojová deska ML506 neobsahuje generátor hodinového signálu této frekvence, proto je použit generátor s frekvencí 100MHz v kombinaci s blokem DCM (z ang. Digital Clock Manager) s dělícím poměrem 5:4.

TX_CLK: Výstupní hodinový signál, řídící data z vyšších vrstev. Modul *mac_core* ho nastavuje multiplexorem podle detekované rychlosti Ethernetu.

RX_CLK: Výstupní hodinový signál, řídící přijatá data určená pro vyšší vrstvy. Modul *mac_core* ho nastavuje multiplexorem podle detekované rychlosti Ethernetu.

PHY_RXCLK: Hodinový signál řídící data přijatá z čipu fyzické vrstvy. Je generován externím čipem fyzické vrstvy.

PHY_TXCLK: Hodinový signál řídící data vstupující do čipu fyzické vrstvy v režimu 10/100Mb/s. Je generován externím čipem fyzické vrstvy.

PHY_GTXCLK: Hodinový signál řídící data vstupující do čipu fyzické vrstvy v režimu 1Gb/s. Je generován modulem *mac_core* ze signálu *GTXCLK*.

3.2.2 Detekce rychlosti ethernetu

Režim *Tri-Mode Speed* zahrnuje rychlosti 10/100/1000Mbit/s. Ethernet používá pro rychlosti 10/100Mbit/s rozhraní MII (4 bity) a pro rychlost 1000Mbit/s rozhraní GMII (8 bitů). K zajištění kompatibility jádra pro všechny rychlosti je nezbytné detekovat rychlost 1000Mb/s a podle toho přizpůsobit rozhraní *mac_core – čip fyzické vrstvy*. K určení rychlosti je využit hodinový signál *PHY_RXCLK*, protože je jako jediný generován fyzickou vrstvou při všech rychlostech a podle jeho frekvence je možné určit aktuálně používanou rychlost.

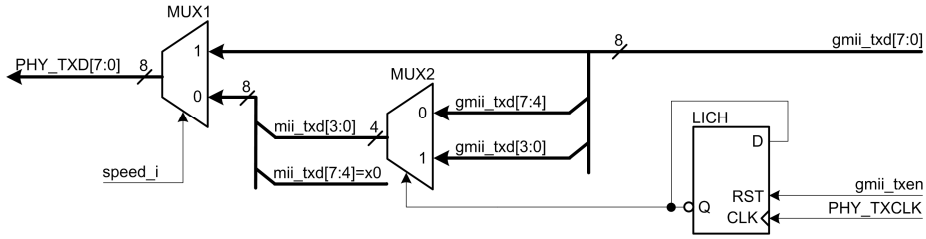
Základem detektoru rychlosti je 2bitový čítač, který je řízen náběžnou hranou hodinového signálu *GTXCLK*. Count enable a asynchronní reset čítače je řízen signálem *PHY_RXCLK*. Pokud je frekvence hodinového signálu *PHY_RXCLK* srovnatelná s frekvencí 125MHz, čítač nepřeteče dřív, než je vynulován a rychlost je vyhodnocena jako 1Gb/s (*speed_i = 1*). Přetečením čítače je rychlost vyhodnocena jako 10/100Mbit/s (*speed_i = 0).* K rozlišení rychlostí 10/100Mbit/s by bylo třeba použít více bitový čítač, ale pro funkci modulu není nutné tyto rychlosti rozlišovat.

3.2.3 Zajištění kompatibility MII a GMII

Data ve všech navržených modulech se zpracovávají po 8 bitech, proto je nezbytné, při rychlostech 10/100Mbit/s zprostředkovat převod mezi MII a 8bitovým rozhraním. Obvody řady Virtex4 a vyšší umožňují pro tento účel použít registr *ODDR*, podporující DDR přenos. Takto vytvořený modul by nebylo možné implementovat do obvodů nižších řad, proto byla navržena univerzální varianta.

VYSÍLACÍ ČÁST PŘEVODNÍKU ROZHRANÍ

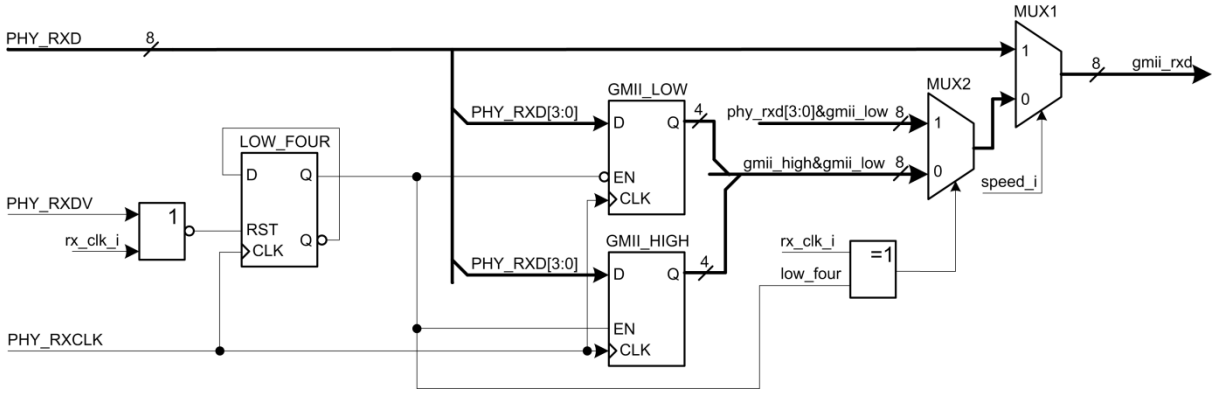
Princip přizpůsobení rozhraní ukazuje obrázek 22. Detekováním rychlosti 1Gb/s je na výstupní sběrnici *PHY_TXD* přivedena přes multiplexor *MUX1* celá šířka 8bitové sběrnice *gmii_txd* z modulu *mac_tx*. Detekováním rychlosti 10/100Mbit/s je použit multiplexor MUX2, který při každém cyklu hodinového signálu *PHY_TXCLK*, přivede na výstup polovinu 8bitového slova. Správné pořadí jednotlivých slov je zajištěno signálem reset klopného obvodu *LICH*. Signály vedené do čipu fyzické vrstvy jsou registrové.



Obr. 22: Schéma zapojení převodníku rozhraní pro vysílací část modulu *mac_core*

PŘIJÍMACÍ ČÁST PŘEVODNÍKU ROZHRANÍ

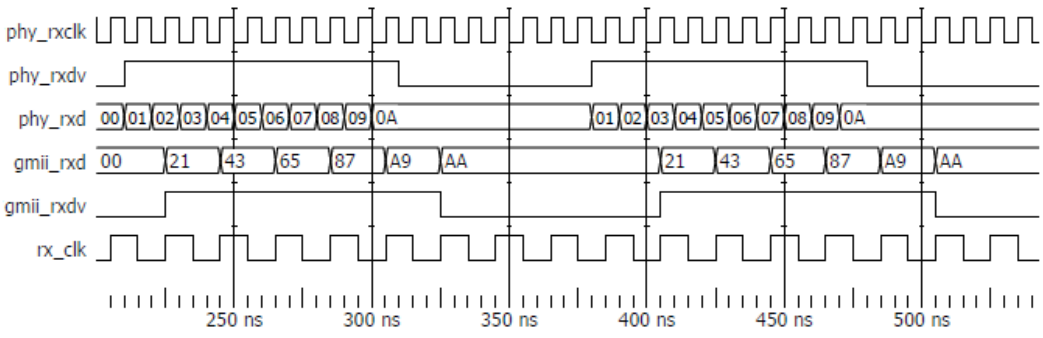
Princip přizpůsobení rozhraní ukazuje obrázek 23. Detekováním rychlosti 1Gbit/s je pres multiplexor *MUX1* přivedena na sběrnici *gmii_rxdv* celá šířka vstupní 8bitové sběrnice *PHY_RXD*, která je řízena hodinovým signálem *PHY_RXCLK*. Detekcí nižší rychlosti je signál *gmii_rxdv* řízen hodinovým signálem *rx_clk_i*, jehož frekvence je *PHY_RXCLK/2*. Do registrů *GMII_LOW a GMII_HIGH* jsou ukládány jednotlivá 4bitová slova z rozhraní *MII* (dolní 4 bity sběrnice *PHY_RXD*). Hradlo *XOR* tvoří fázový komparátor, který zajišťuje správné pořadí 4bitových slov při jejich sestavení do 8bitového slova. Sběrnice *gmii_rxd* je zakončena registrovým výstupem a připojena k modulu *mac_rx*.



Obr. 23: Schéma zapojení převodníku rozhraní pro přijímací část modulu *mac_core*

SIMULACE PŘEVODU ROZHRANÍ Z MII NA GMII

Simulace příchodu dat z MII rozhraní ukazuje obrázek 24. Levá část zobrazuje příchod prvního 4bitového slova bezprostředně po náběžné hraně signálu *rx_clk*, který řídí data na sběrnici *gmii_rxdv*. V pravé části je simulován příchod dat těsně před náběžnou hranou hodinového signálu *rx_clk*.



Obr. 24: Simulace převodu rozhraní z GMII na MII

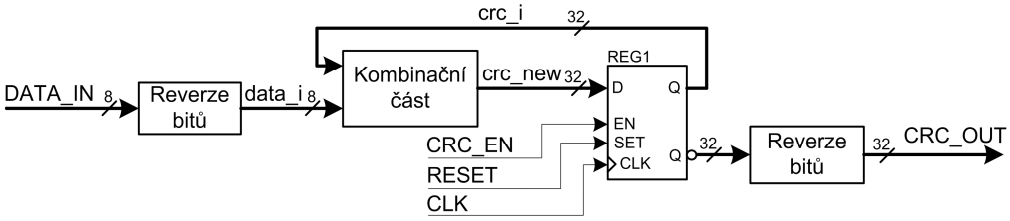
3.2.4 Výpočet CRC – modul crc

Modul *crc* implementuje výpočet 32bitové kontrolní sekvence FCS pro linkový rámec Ethernetu. Používá paralelní metodu výpočtu pomocí hradel XOR, popsanou v teoretické části. Popis signálů je uveden v tabulce 9. Modul je použit v přijímací části *mac_rx* pro ověření integrity paketu a ve vysílací části *mac_tx* pro generaci FCS sekvence.

Tab. 9: Vstupní a výstupní signály bloku *crc*

Signál	Směr	Popis
CLK	IN	Řídící hodinový signál, aktivní hrana je náběžná.
CRC_OUT[32:0]	OUT	32bitová kontrolní sekvence (FCS).
DATA_IN[7:0]	IN	Vstupní 8bitová datová sběrnice řízená signálem CLK.
CRC_EN	IN	V logické 1 určuje platnost dat na sběrnici DATA_IN.
RESET	IN	Synchronní reset.

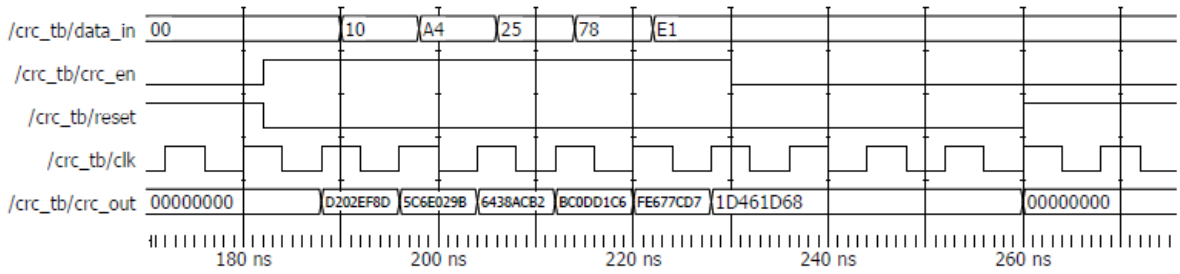
Blokové schéma modulu *crc* je na obrázku 25. Nejprve musí být provedena reverze bitů na datové sběrnici, protože v Ethernetu se posílá nejprve LSB bit každého oktetu. Počáteční hodnota CRC registru *REG1* je nastavena na $FFFFFFFF_{HEX}$. Je to z důvodu detekce nul na počátku datové sekvence. Tabulky pro určení bitových rovnic kombinační části jsou uvedeny v příloze 1. Reverze bitů je provedena i pro výstupní kontrolní sekvenci, aby tato hodnota byla připravena pro vložení do linkového rámce.



Obr. 25: Blokové schéma modulu *crc*

SIMULACE VÝPOČTU CRC32

Simulace na obrázku 26 znázorňuje funkci modulu. Data řízena hodinovým signálem *CLK* jsou přivedena na sběrnici *DATA_IN* a jejich platnost je potvrzena logickou 1 na signálu *CRC_EN*. Reset je nutné provést před každým výpočtem.



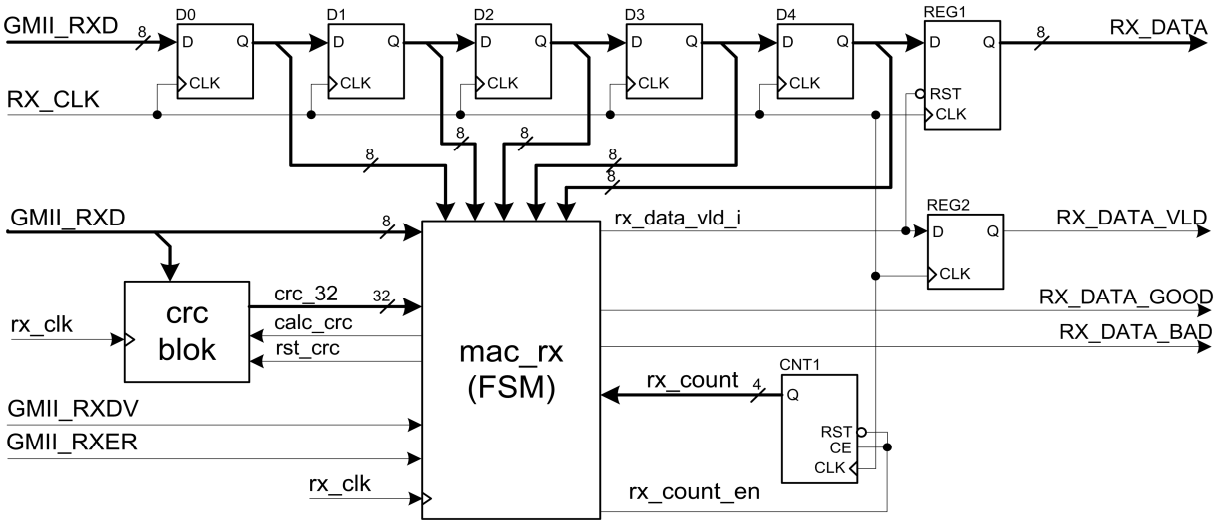
Obr. 26: Simulace modulu *crc*

3.2.5Přijímací část – modul mac_rx

Přijatá data jsou zpracovávána modulem *mac_rx*. Veškerý datový tok je zpožděn o 6 hodinových taktů z důvodu filtrace rámců, směřovaných na jiné MAC adresy. Zpracování přijatých rámců zahrnuje několik operací:

- Kontrola integrity dat CRC výpočtem.
- Odstranění synchronizačního pole preambule a SFD.
- Odstranění kontrolního pole FCS.
- Filtrování rámců, které jsou směrovány na jinou adresu, než je adresa modulu.
- Předání upraveného linkového rámce vyšší vrstvě.

Blokové schéma modulu je na obrázku 27. V rámci podpory paketů typu Ethernet II i typu IEEE802.3 není možné dopředu stanovit délku paketu a tedy ani pozici kontrolní sekvence FCS. Pole FCS se nesmí promítnout do výstupních dat, proto je důležité, aby data byla zpožděna posuvným registrem z klopných obvodů *D0 – D4* o 6 hodinových taktů. Blok *crc* počítá kontrolní sekvenci FCS z přijatých dat. Čítač *CNT1* slouží k identifikaci jednotlivých oktetů hlavičky linkového rámce. Stavový automat je popsán diagramem na obrázku 28.



Obr. 27: Blokové schéma modulu *mac_rx*

STAVOVÝ AUTOMAT MAC_RX

St_idle: Počáteční stav, ve kterém automat čeká do příchodu platných dat. Platnost dat je určena signálem *gmii_rxdv* v logické 1. Je zde resetován čítač *rx_count* a modul *crc*. Výstupy jsou nastaveny do počátečních hodnot.

St_preamble: Čeká na dokončení synchronizační sekvence preambule. Příchodem start oktetu (hodnota $D5_{HEX}$) se přechází do následujícího stavu.

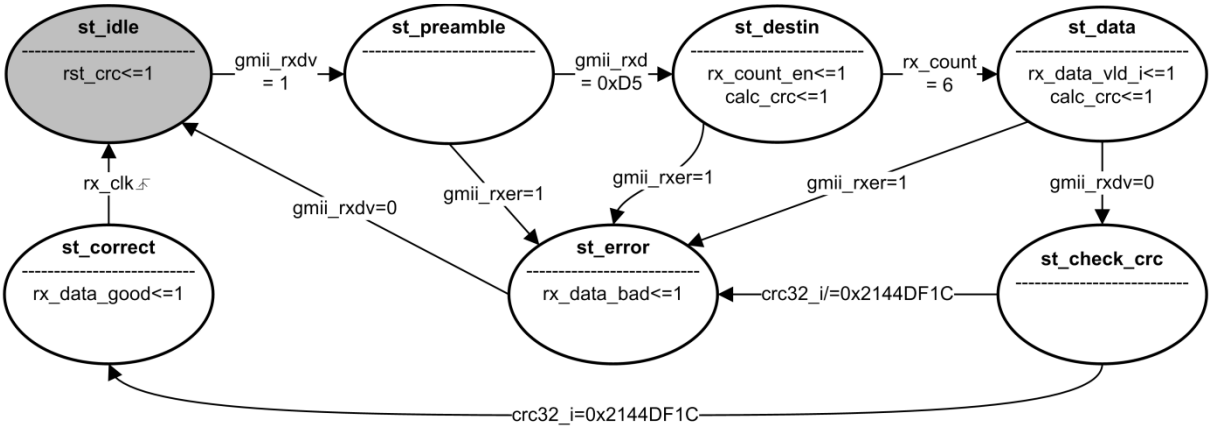
St_destin: Začíná se počítat CRC. Stav trvá 6 hodinových taktů, během kterých je načtena cílová MAC adresa do registru *D0-D4*. Adresa je následně porovnána s adresou modulu (konstanta *FPGA_MAC*) a broadcastovou adresou FF-FF-FF-FF-FF-FF. Pokud je detekována shoda, přejde automat do stavu *st_data*. V opačném případě přechází do stavu *st_error*.

St_data: Linkový rámec je odeslán na výstupní sběrnici *rx_data* a jeho platnost je potvrzena signálem *rx_data_vld*. Logickou 0 na signálu *gmii_rxdv* je detekován konec rámce a přechází se do následujícího stavu.

St_check_crc: Porovná výstup CRC modulu s hodnotou $2144DF1C_{HEX}$, která reprezentuje platný paket. Trvá jeden hodinový cyklus.

St_error: Chybový stav. Při výskytu chyby přechází automat do tohoto stavu, ve kterém se čeká na konec přenosu. Poté je přijatý paket označen jako neplatný logickou 1 na signálu *rx_data_bad* a přechází se do počátečního stavu.

St_correct: Trvá 1 hodinový cyklus. Generuje logickou 1 na signálu *rx_data_good*, která určuje, že je přijatý rámec pořádku.



Obr. 28: Diagram stavového automatu *mac_rx*

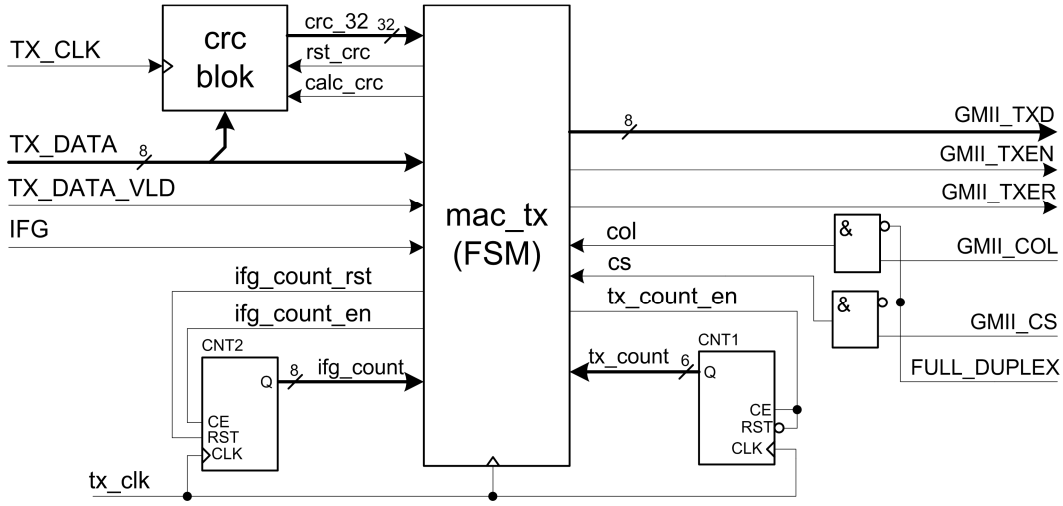
3.2.6 Vysílací část – modul mac_tx

Vlastní linkový rámec sestavuje vrstva *ip_core*, která v sobě implementuje ARP protokol. Vrstva *mac_tx* upravuje linkový rámec pro fyzickou vrstvu. To zahrnuje následující operace:

- Přidání synchronizačního pole preambule a start bajtu SFD.
- Kontrola minimální délky rámce, v případě potřeby doplnění rámce o tzv. patku.
- CRC výpočet.
- Připojení pole kontrolní sekvence FCS pro zajištění integrity dat.

- Přístup k médiu a řešení kolizí algoritmem CDMA/CS při poloviční duplexu.
- Předání paketu fyzické vrstvě.
- Implementace volitelné mezi-rámcové mezery IFG.

Obrázek 29 zobrazuje blokové schéma vrstvy *mac_tx*. Vrstva se skládá z *crc* modulu, který je ovládán stavovým automatem. *CNT1* je čítač do hodnoty 60_{DEC}, identifikuje jednotlivá pole hlavičky a kontroluje minimální délku rámce. Zároveň slouží ke stanovení základní mezi-rámcové mezery. Čítač *CNT2* implementuje uživatelem definovanou mezi-rámcovou mezeru, která je vždy násobkem základní, 12 bajtové, mezery. Nastavení plného duplexu logickou 1 na signálu *FULL_DUPLEX* jsou blokovány vstupy pro detekci kolizí a přístup k médiu.



Obr. 29: Blokové schéma modulu *mac_tx*

Funkci stavového automatu znázorňuje diagram na obrázku 30. Žlutou barvou jsou označeny stavy, které jsou přístupné pouze při poloviční duplexu.

STAVOVÝ AUTOMAT MAC_TX

St_idle: Počáteční stav, ve kterém automat čeká do příchodu platných uživatelských dat. Platnost dat je určena logickou 1 na signálu *tx_data_vld*. Je zde resetován čítač *CNT1*, *CNT2* a modul *crc*. Výstupy jsou nastaveny do počátečních hodnot.

St_preamble: Odešle do fyzické vrstvy prvních 7 oktetů preambule a start bajt SFD, nakonec vygeneruje logickou 1 na signálu *tx_data_ack* a přechází do dalšího stavu.

St_data: Odesílá uživatelská data a počítá z nich CRC. Minimální délka rámce je ověřována čítačem s výstupem *tx_count*. Není-li délka rámce dostatečná, následuje stav *st_padding*.

St_padding: Doplní rámec o patku, složenou z nulových oktetů, aby vyhovoval minimální délce linkového rámce.

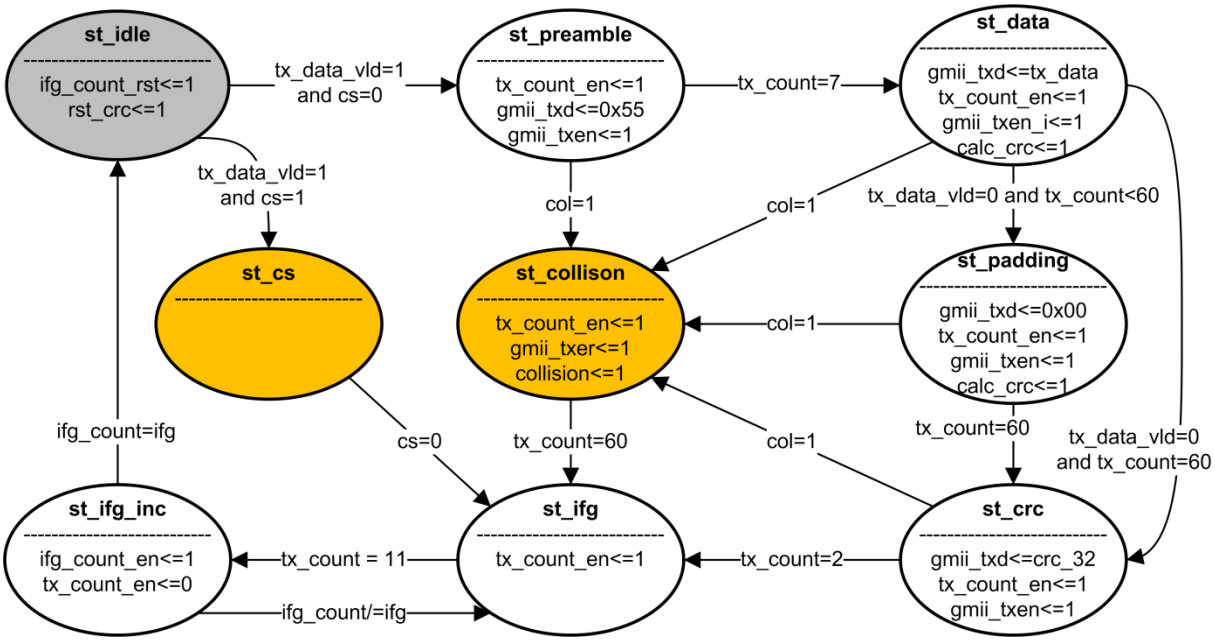
St_crc: Doplní rámec o kontrolní sekvenci FCS. Na konci resetuje čítač s výstupem *tx_count*, který bude následně použit ve stavu *st_ifg* pro určení mezi-rámcové mezery.

St_ifg: Ukončí rámec a vloží za něj mezi-rámcovou mezeru délky 11 bajtů (s následujícím stavem *st_ifg_inc* je to 12B podle IEEE802.3).

St_ifg_inc: Stav implementuje uživatelem definovanou mezi-rámcovou mezeru. Hodnota na sběrnici *IFG* udává počet 12 bajtových mezer, které budou vloženy za odeslaný rámec. Zároveň resetuje čítač s výstupem *tx_count*.

St_cs: Pokud vyšší vrstva požaduje odeslání dat a jiná stanice v síti vysílá, přechází automat do tohoto stavu a čeká, dokud není médium uvolněno.

St_collision: Automat přechází do tohoto stavu, pokud došlo ke kolizi odesílaných dat. Vygeneruje chybový signál pro fyzickou vrstvu (logická 1 na signálu *gmii_txer*) a signál detekující kolizi pro vyšší vrstvu (logická 1 na signálu *collision*). Poté čeká na naplnění čítače s výstupem *tx_count* a po vložení mezi-rámcové mezery přechází do počátečního stavu. Následně je možné rámec odeslat znovu v případě, že jiná stanice nevysílá.

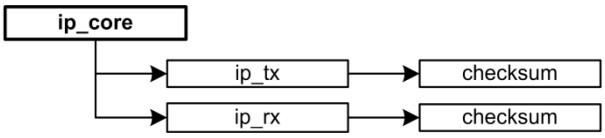


Obr. 30: Diagram stavového automatu modulu *mac_tx*

3.3 Modul ip_core

Modul *ip_core* implementuje protokol IPv4 a protokol ARP, zajišťující dynamickou aktualizaci ARP tabulky. Strukturu modulu ukazuje obrázek 31. Pro odeslání paketu

požadované délky je nutné zadat i délku datové části paketu. Je to z důvodu minimalizace použitých paměťových zdrojů obvodu FPGA. Data je možné odeslat na adresu v lokální síti i na adresu internetovou prostřednictvím brány a síťové masky. Parametry pro nastavení sítě se dají měnit pouze před syntézou modifikací konstant v souboru *my_constants.vhd*. Simulace vrstvy je z důvodu rozsahu uvedena na přiloženém cd.



Obr. 31: Struktura bloku *ip_core*

Vrstva neumožňuje zpracovávat fragmentované pakety, proto je nezbytné, aby při komunikaci s PC byla fragmentace buď zakázána, nebo volena vhodná velikost paketů vyšších vrstev. Modul je sestaven z vysílací a přijímací části, které pracují nezávisle na sobě. Přijímací vrstva aktualizuje ARP tabulku, do které se vejde maximálně jeden záznam. Vstupní a výstupní signály modulu *ip_core* jsou uvedeny v následujících tabulkách.

Tab. 10: Hodinové signály bloku *ip_core*

Signál	Směr	Popis
RX_CLK	IN	Hodinový signál pro řízení přijímaných dat.
TX_CLK	IN	Hodinový signál pro řízení odesílaných dat.

Tab. 11: Signály pro přenos dat mezi *ip_core* a vyšší vrstvou

Signál	Směr	Popis
RX_DATA[7:0]	OUT	Výstupní 8bitová sběrnice pro data do vyšší vrstvy.
RX_DATA_VLD	OUT	Logická 1 určuje platnost dat na sběrnici RX_DATA.
RX_DATA_GOOD	OUT	Potvrzuje, že přijatý paket neobsahuje chybu.
RX_DATA_BAD	OUT	Potvrzuje, že přijatý paket obsahuje chybu.
TX_DATA	IN	Vstupní 8bitová sběrnice pro data z vyšší vrstvy.
TX_DATA_VLD	IN	Logická 1 určuje platnost dat na sběrnici RX_DATA.
TX_DATA_LEN[param:0]	IN	Délka datového bloku k odeslání. *Param* závisí na MTU.
TX_DATA_ACK	OUT	Potvrzuje načtení prvního bajtu ze sběrnice TX_DATA.

Tab. 12: Signály pro přenos dat mezi *ip_core* a nižší vrstvou

Signál	Směr	Popis
DATA_FROM_MAC[7:0]	IN	Vstupní 8bitová sběrnice pro data vrstvy *mac_core*.
DATA_FROM_MAC_VLD	IN	Logická 1 určuje platnost dat na DATA_FROM_MAC.
DATA_FROM_MAC_GOOD	IN	Potvrzuje, že přijatý paket neobsahuje chybu.
DATA_FROM_MAC_BAD	IN	Potvrzuje, že přijatý paket obsahuje chybu.
DATA_TO_MAC[7:0]	OUT	Výstupní 8bitová sběrnice pro data do vrstvy *mac_core*.
DATA_TO_MAC_VLD	OUT	Logická 1 určuje platnost dat na DATA_TO_MAC.
DATA_TO_MAC_ACK	IN	Potvrzuje načtení prvního bajtu vrstvou *mac_core*.

Tab. 13: *Řídící, směrovací a informační signály*

Signál	Směr	Popis
DESTINATION_IP[31:0]	IN	IP adresa, na kterou bude paket odeslán.
DESTINATION_PROTOCOL[7:0]	IN	Cílový protokol vyšší vrstvy (UDP = 17. ICMP = 1).
SOURCE_IP[31:0]	OUT	IP adresa, ze které přišel aktuální paket.
SOURCE_PROTOCOL[7:0]	OUT	Protokol, pro který jsou určena přijatá data (UDP = 17).
RESET	IN	Synchronní reset.

Konstanty pro nastavení sítě:

```
constant FPGA_IP      : std_logic_vector(31 downto 0);   -- ip adresa obvodu fpga
constant IP_MASK      : std_logic_vector(31 downto 0);   -- maska sítě
constant GATEWAY      : std_logic_vector(31 downto 0);   -- defaultní brána
```

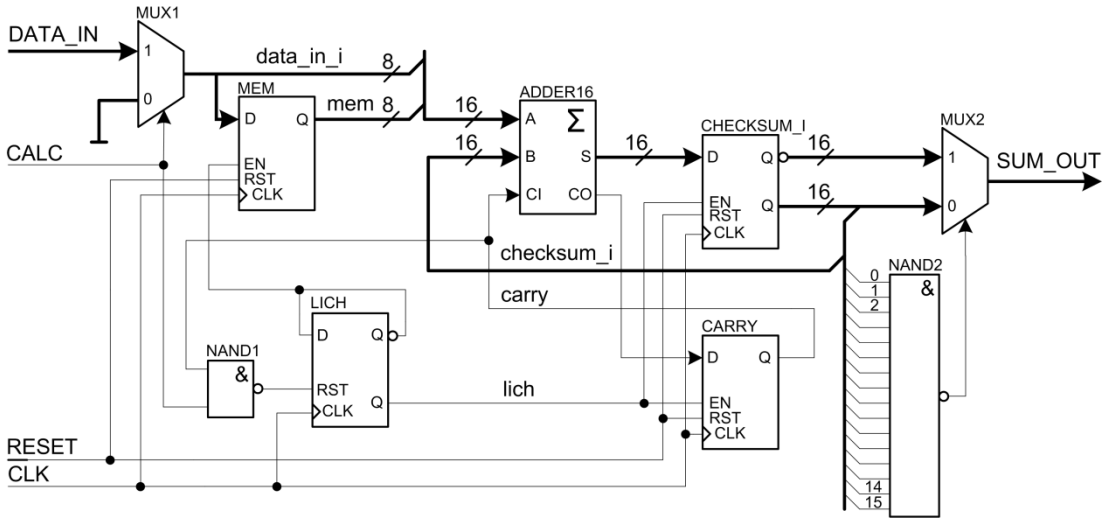
3.3.1 Výpočet kontrolního součtu – modul checksum

Modul zajišťuje výpočet 16bitové kontrolní sekvence u vrstev *ip_core*, *udp_core* a *icmp_core*. Ovládací signály jsou uvedeny v tabulce 14. V přijímací části je modul určen pro potvrzení platného paketu, ve vysílací části pro generaci kontrolní sekvence checksum.

Tab. 14: Vstupní a výstupní signály vrstvy *checksum*

Signál	Směr	Popis
CLK	IN	Řídící hodinový signál, aktivní hrana je náběžná.
SUM_OUT[15:0]	OUT	Výstupní 16bitová kontrolní sekvence checksum.
DATA_IN[7:0]	IN	Vstupní 8bitová datová sběrnice.
CALC	IN	Logická 1 určuje platnost dat na sběrnici DATA_IN.
RESET	IN	Synchronní reset.

Schéma zapojení modulu *checksum* ukazuje obrázek 32. Nejprve jsou vstupní data složena do 16bitového slova – dolní oktet je uložen do registru *MEM*, horní oktet je přiveden přímo na vstup sčítačky *ADDER16*.



Obr. 32: Schéma zapojení modulu *checksum*

Každé složené slovo je přičteno úplnou sčítačkou k předchozí hodnotě kontrolního součtu, která je uložena v registru *CHECKSUM_I*. Výsledek je převeden na jednotkový doplněk. Pokud je výsledný jednotkový doplněk nulový, nastaví se hodnota výstupu na $FFFF_{HEX}$. Výpočet je ukončen logickou 0 na signálu *CALC*.

Při neúplném vstupním slově je dolní oktet doplněn fiktivními nulami pro výpočet. Pokud výsledek překročí 16bitový rozsah, přičte se bit *CO* (přetečení) k výsledku následující hodinový takt jako bit *CI*.

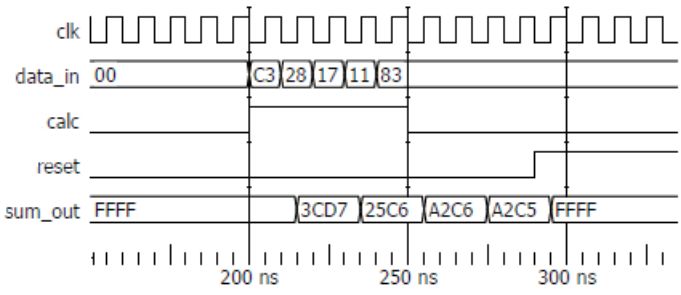
Existují čtyři možnosti korektního ukončení výpočtu:

1. Ukončení úplným 16bitovým slovem bez překročení bitového rozsahu sčítačky *(výsledek je na výstupu okamžitě).*
2. Ukončení úplným 16bitovým slovem s překročením bitového rozsahu sčítačky *(výsledek je na výstupu za 1 hodinový cyklus)*
3. Ukončení neúplným slovem bez překročení bitového rozsahu sčítačky *(výsledek je na výstupu za 1 hodinový cyklus).*
4. Ukončení neúplným slovem s překročením bitového rozsahu sčítačky *(výsledek je na vstupu za 3 hodinové cykly).*

Pro správnou funkci ve všech uvedených případech, je nutné čekat na výsledek minimálně 3 hodinové cykly. Před každým výpočtem je potřeba modul resetovat signálem *RESET*, aktivním v logické 1.

SIMULACE VÝPOČTU KONTROLNÍHO SOUČTU

Simulace vrstvy *checksum* pro vstupních 5 oktetů ukazuje obrázek 33. Data jsou volena tak, aby výpočet trval maximální možnou dobu, tzn. ukončení neúplným vstupním slovem s překročením bitového rozsahu. Platný jednotkový doplněk je na výstupu po třech hodinových taktech od ukončení výpočtu signálem *CALC* v logické 0.

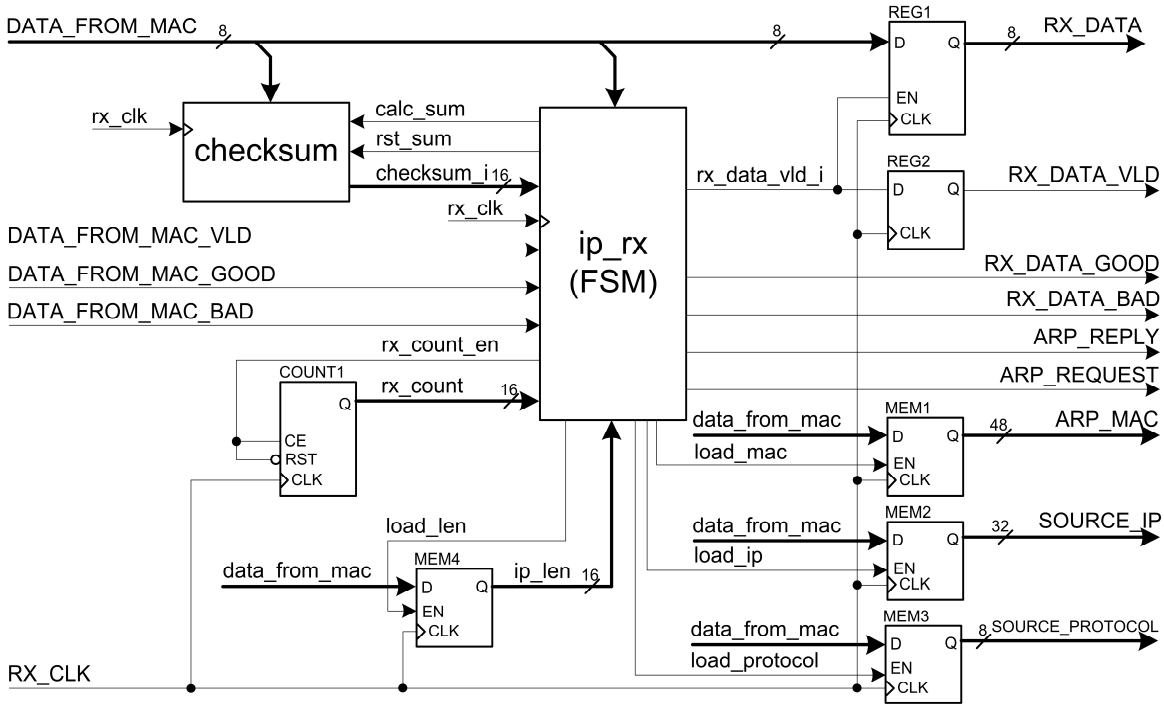


Obr. 33: Simulace vrstvy *checksum*

3.3.2 Přijímací část – modul ip_rx

Vrstva zpracovává přijaté linkové rámce z modulu *mac_core* na úrovni síťové vrstvy. Zpracování zahrnuje několik operací:

- Aktualizuje ARP tabulku podle příchozích ARP paketů.
- Generuje požadavek na odeslání ARP odpovědi pro vysílací vrstvu *ip_tx*.
- Ověří integritu IP paketu výpočtem kontrolního součtu.
- Filtruje pakety směrované na jinou IP adresu.
- Předá vyšší vrstvě datovou část, zdrojovou IP adresu a číslo protokolu vyšší vrstvy přijatého IP paketu.



Obr. 34: Blokové schéma modulu *ip_rx*

Blokové schéma modulu je na obrázku 34. Modul *checksum* počítá kontrolní součet hlavičky IP paketu. Čítač *COUNT1* slouží k identifikaci jednotlivých oktetů přijatého paketu a zajišťuje správnou délku datové části IP paketu. Všechny potřebné informace z hlavičky uchovávají registry *MEM1*, *MEM2*, *MEM3* a *MEM4*. Výstupy stavového automatu *ip_rx* jsou registrové, jeho funkci znázorňuje diagram na obrázku 35.

STAVOVÝ AUTOMAT *IP_RX*

St_idle: Počáteční stav, ve kterém automat čeká do příchodu platných dat z vrstvy *mac_core*. Je zde resetován modul *checksum* a čítač s výstupem *rx_count*. Výstupy jsou nastaveny do počátečních hodnot.

St_detect_type: Trvá 14 hodinových cyklů, v posledním cyklu se analyzuje první oktet datové části linkového paketu. Hodnota 45_{HEX} označuje IP paket, hodnota 00_{HEX} ARP paket. V případě že není identifikován typ, přechází se do čekacího stavu *st_wait_end*.

St_arp_type: Paket je identifikován jako ARP, ověří se, zda jsou všechna pole hlavičky správná a určí se, zda jde o dotaz nebo o odpověď.

St_request: Uloží do paměťového registru IP a MAC adresu odesílatele z přijatého ARP dotazu.

St_request_ack: Čeká na potvrzení platnosti ARP dotazu vrstvou *mac_core*, odešle do vrstvy *ip_tx* obsah paměťových registrů a adresami odesílatele, následně vygeneruje požadavek na odeslání odpovědi ARP reply.

St_reply: Uloží do paměťového registru IP a MAC adresu odesílatele z přijaté ARP odpovědi.

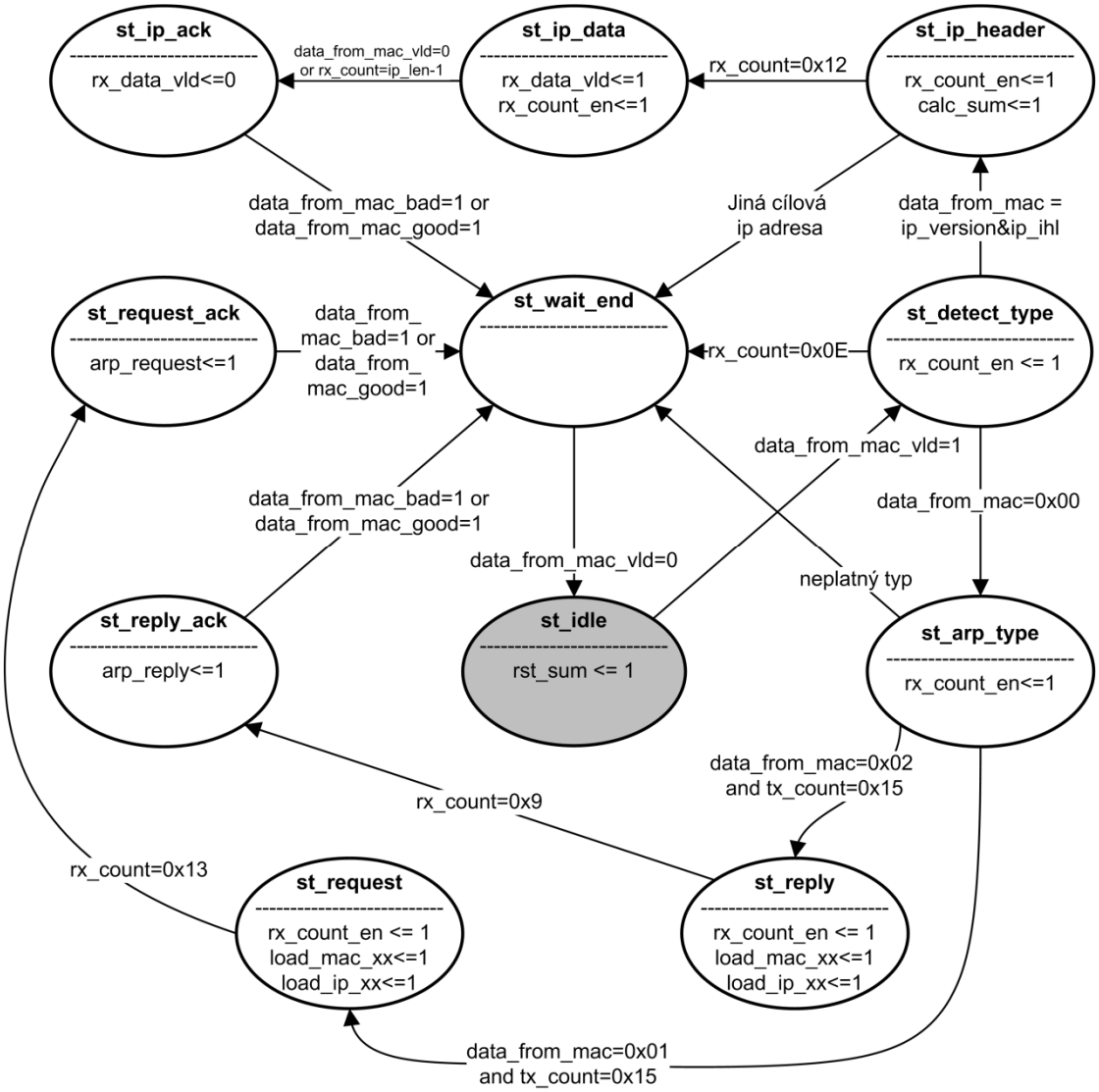
St_reply_ack: Čeká na potvrzení platnosti paketu vrstvou *mac_core*, odešle do vrstvy *ip_tx* obsah paměťových registru a adresami odesílatele.

St_ip_header: Analyzuje hlavičku přijatého IP paketu, uloží do paměťových registrů celkovou délku paketu, zdrojovou IP adresu a protokol vyšší vrstvy. Ověří, zda je paket směrován na IP adresu modulu. Zároveň je počítán kontrolní součet hlavičky, který je na konci paketu otestován.

St_ip_data: Odešle datovou část IP paketu do vyšší vrstvy prostřednictvím sběrnice *rx_data* a signálu *rx_data_vld*.

St_ip_ack: Vyčkává na příchod potvrzení platného nebo neplatného paketu z vrstvy *mac_core* (signály *data_from_mac_bad* a *data_from_mac_good*). Podle výsledku kontrolního součtu generuje potvrzení platnosti přijatých dat pro vyšší vrstvu signálem *rx_data_good* nebo *rx_data_bad*.

St_wait_end: Čeká na dokončení přenosu neplatného rámce z vrstvy *mac_core*. Data, přijata v tomto stavu, jsou zahozena. Dokončení přenosu je signalizováno signálem *data_from_mac_vld* v logické 1.



Obr. 35: Stavový automat *ip_rx*

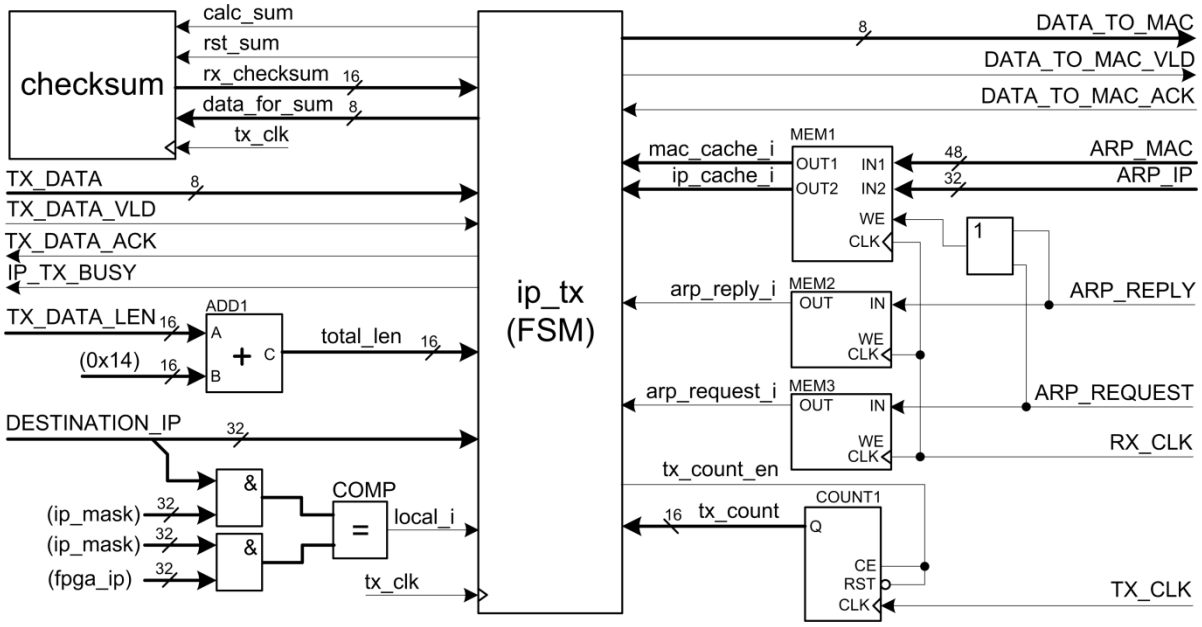
3.3.3Vysílací část – modul ip_tx

Vrstva implementuje odesílání uživatelských dat, která jsou balena do datového pole IP paketu. Připravuje linkové pakety pro vrstvu *mac_core*. To zahrnuje následující operace:

- Přidá hlavičku linkového rámce.
- Generuje podle potřeby pakety typu ARP request a ARP reply.
- Počítá kontrolní součet hlavičky IP paketu a doplní jej do pole checksum.
- Generuje hlavičku IP protokolu a připojí k ní uživatelská data.
- Detekuje, zda se cílová adresa nachází v lokální síti, pokud ne, směruje paket na defaultní bránu.

Blokové schéma vrstvy ukazuje obrázek 36. Modul *checksum* počítá kontrolní součet hlavičky IP paketu. Paměťový registr *MEM1* slouží jako ARP tabulka, zápis do ní je řízen

vrstvou *ip_rx*. Registry *MEM2* a *MEM3* uchovávají požadavky z přijímací části do doby, než budou zpracovány. Komparátor *COMP* porovnává, zda cílová IP adresa leží v lokální síti. Sčítačka *ADD1* rozšiřuje délku vstupních dat o délku IP hlavičky. Stavový automat má registrové výstupy, jeho funkci znázorňuje diagram na obrázku 37.



Obr. 36: Blokové schéma modulu *ip_tx*

STAVOVÝ AUTOMAT IP_TX

St_idle: Počáteční stav, ve kterém automat čeká do příchodu uživatelských dat z vyšší vrstvy. Je zde resetován modul *checksum* a čítač s výstupem *tx_count*. Výstupy jsou nastaveny do počátečních hodnot.

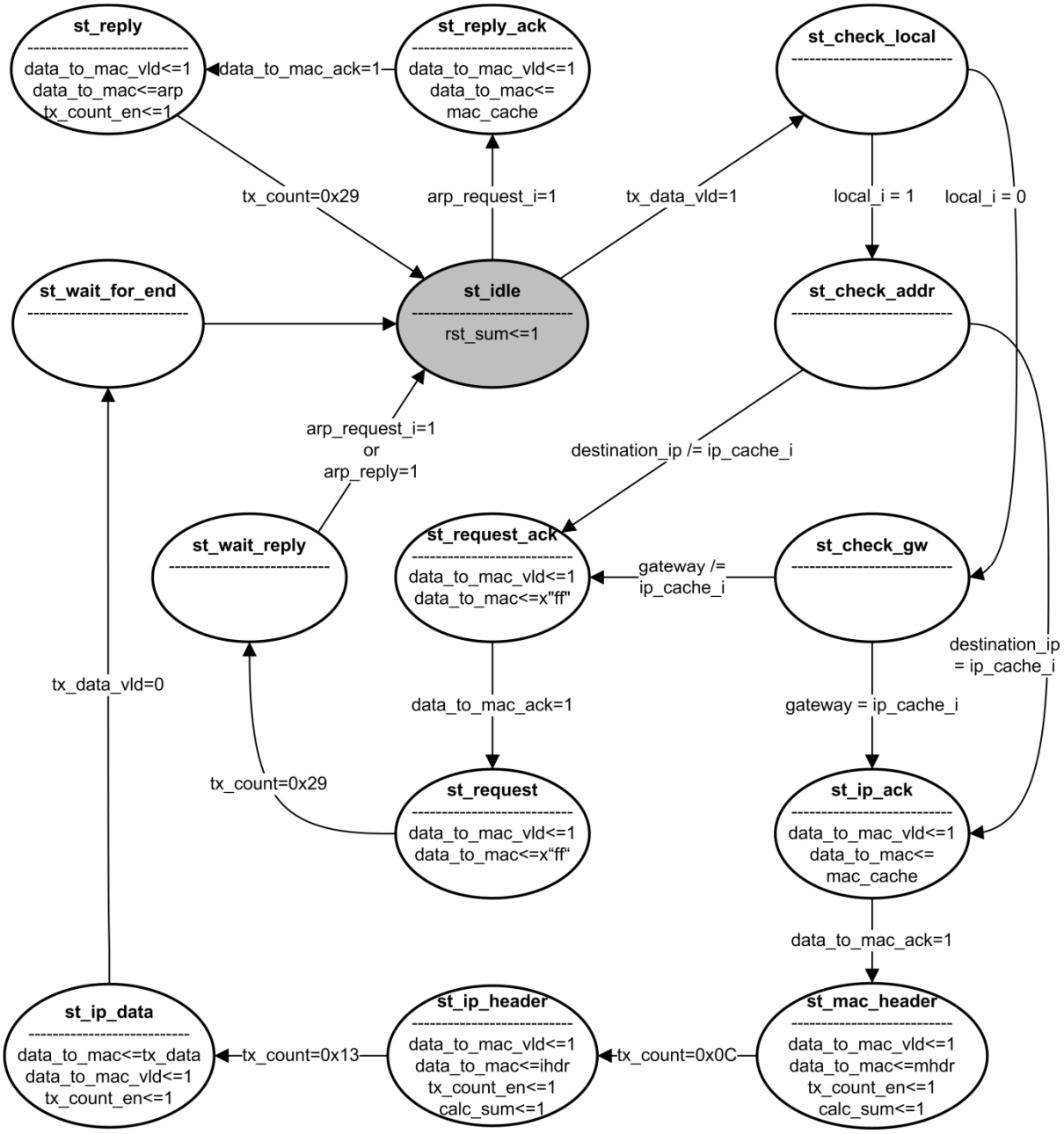
St_check_local: Trvá 1 hodinový cyklus. Je zde ověřeno, zda cílová IP adresa leží v lokální síti.

St_check_addr: Trvá 1 hodinový cyklus. Cílová adresa leží v lokální síti. Zjistí, se zda ARP tabulka obsahuje cílovou IP adresu. Pokud ne, je nutné vyslat ARP dotaz.

St_check_gw: Trvá 1 hodinový cyklus. Cílová adresa není v lokální síti. Zjistí se, zda ARP tabulka obsahuje IP adresu defaultní brány. Pokud ne, je nutné vyslat ARP dotaz.

St_request_ack: Zahájí odesílání paketu ARP request. Na sběrnici *data_to_mac* je přiveden první oktet linkového rámce, jeho platnost se určí logickou 1 na signálu *data_to_mac_vld* a čeká se na potvrzení jeho přijetí vrstvou *mac_core*.

St_request: Odešle ostatní oktety paketu ARP request do vrstvy *mac_core*. Poté přechází do čekacího stavu *st_wait_reply*.



Obr. 37: Stavový automat *ip_tx*

St_wait_reply: Čeká do příchodu odpovědi na odeslaný ARP dotaz, nebo do příchodu nového dotazu.

St_reply_ack: Začíná odesílat paket typu ARP reply. Na sběrnici *data_to_mac* je přiveden první oktet paketu linkového rámce, jeho platnost je určena signálem *data_to_mac_vld* v logické 1. Čeká se na potvrzení jeho přijetí vrstvou *mac_core*.

St_reply: Odešle se zbytek paketu ARP reply do vrstvy *mac_core* a přechází se do počátečního stavu *st_idle*.

St_ip_ack: Na sběrnici *data_to_mac* je přiveden první oktet linkového rámce (první oktet cílové MAC adresy), jeho platnost se určí signálem *data_to_mac_vld* a čeká se na potvrzení jeho přijetí vrstvou *mac_core*.

St_mac_header: Je zde odeslán zbytek hlavičky linkového rámce a počítán kontrolní součet pro IP paket. To umožňuje získat kompletní součet IP hlavičky před polem checksum.

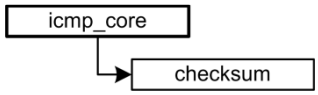
St_ip_header: Odešle do vrstvy *mac_core* hlavičku IP paketu. Výpočet kontrolního součtu je dokončen před odesláním pole checksum, proto je možné toto pole korektně vyplnit.

St_ip_data: Odešle se datová část IP paketu, která obsahuje uživatelská data z vyšší vrstvy. Počet oktetů je určen signálem *tx_data_len*, který generuje vyšší vrstva.

St_wait_end: Trvá jeden hodinový cyklus, ve kterém jsou výstupy nastaveny do počátečních hodnot.

3.4 Modul icmp_core

Modul implementuje funkci ICMP Echo (Ping). Strukturu modulu ukazuje obrázek 38. Po přijetí paketu Echo request je datová část odeslána zpět k odesílateli jako paket Echo reply. Odesílatel měří dobu mezi odesláním dotazu a přijetím odpovědi. Délka odeslané datové části není pevně určena a může se pro různé implementace lišit. Vstupní a výstupní signály vrstvy *icmp_core* jsou uvedeny v tabulce 15.



Obr. 38: Struktura bloku *icmp_core*

Tab. 15: Vstupní a výstupní signály bloku *icmp*

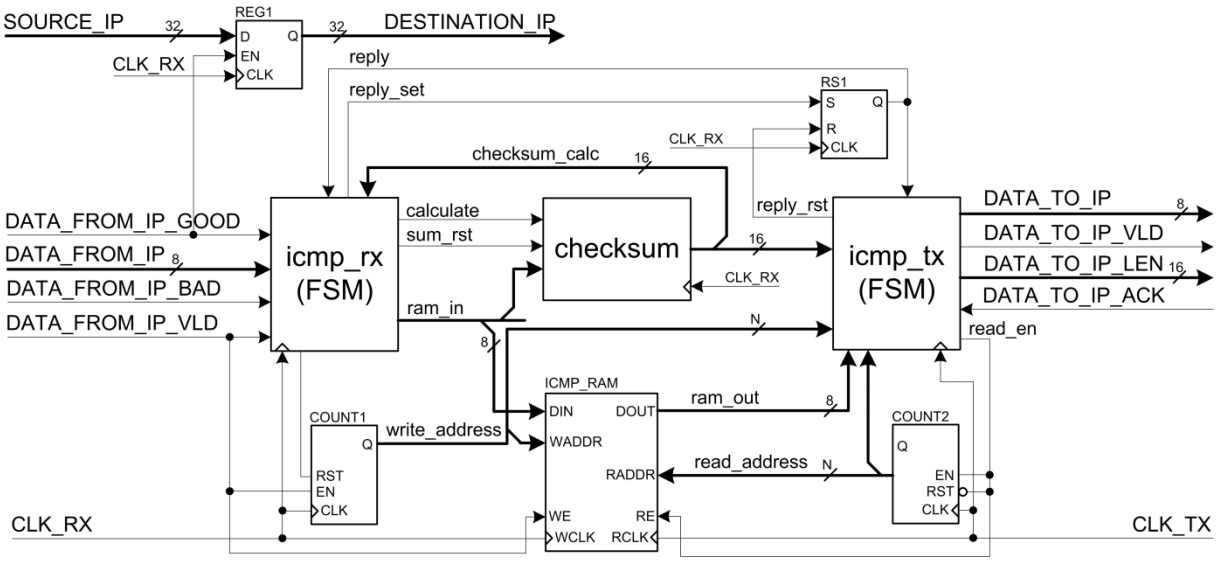
Signál	Směr	Popis
RX_CLK	IN	Hodinový signál pro řízení přijímaných dat.
TX_CLK	IN	Hodinový signál pro řízení odesílaných dat.
DATA_FROM_IP [7:0]	IN	Vstupní 8bitová sběrnice pro data z *ip_core*.
DATA_FROM_IP _VLD	IN	Určuje platnost dat na sběrnici DATA_FROM_IP.
DATA_FROM_IP _GOOD	IN	Potvrzuje, že přijatý paket neobsahuje chybu.
DATA_FROM_IP _BAD	IN	Potvrzuje, že přijatý paket obsahuje chybu.
DATA_TO_IP	OUT	Výstupní 8bitová sběrnice pro data do *ip_core*.
DATA_TO_IP _VLD	OUT	Určuje platnost dat na sběrnici DATA_TO_IP.
DATA_TO_IP_LEN[param:0]	OUT	Délka odesílané odpovědi. Param závisí na MTU.
DATA_TO_IP _ACK	IN	Potvrzení přijetí prvního bajtu nižší vrstvou.
SOURCE_ADDRESS [31:0]	IN	IP adresa, ze které paket přišel.
DESTINATION_ADDRESS [31:0]	OUT	IP adresa, na kterou bude odeslána odpověď.
RESET	IN	Synchronní reset.

Standardně se délka paketů ICMP Echo pohybuje v desítkách bajtů, proto by pro většinu případů stačila menší distribuovaná paměť. Službu ICMP Echo je však možné použít i pro měření MTU mezi dvěma body. V takovém případě by paměť, menší než je velikost MTU, způsobila problém. Proto je použita parametrická bloková paměť, jejíž velikost je nastavena konstantou pro maximální velikost UDP paketu:

```
constant UDP_ADDR_WIDTH: integer; -- velikost UDP paketu
```

Blokové schéma modulu je na obrázku 39. Skládá se z paměťového registru *REG1*, do kterého se uloží IP adresa odesílatele dotazu. Na tuto adresu se odešle odpověď. Čítač *COUNT1* adresuje zápis vstupních dat do dvou portové blokové paměti *ICMP_RAM*. Přijatá data zpracovává stavový automat *icmp_rx*, který zároveň ovládá modul pro výpočet kontrolního součtu *checksum*.

Stavový automat *icmp_tx* sestavuje paket Echo reply z dat v paměti *ICMP_RAM*. Čítač *COUNT2* adresuje čtení z paměti. Klopný obvod *RS1* synchronizuje požadavek na odeslání odpovědi mezi stavovými automaty, které jsou řízeny odlišnými hodinovými signály.



Obr. 39: Blokové schéma modulu *icmp_core*

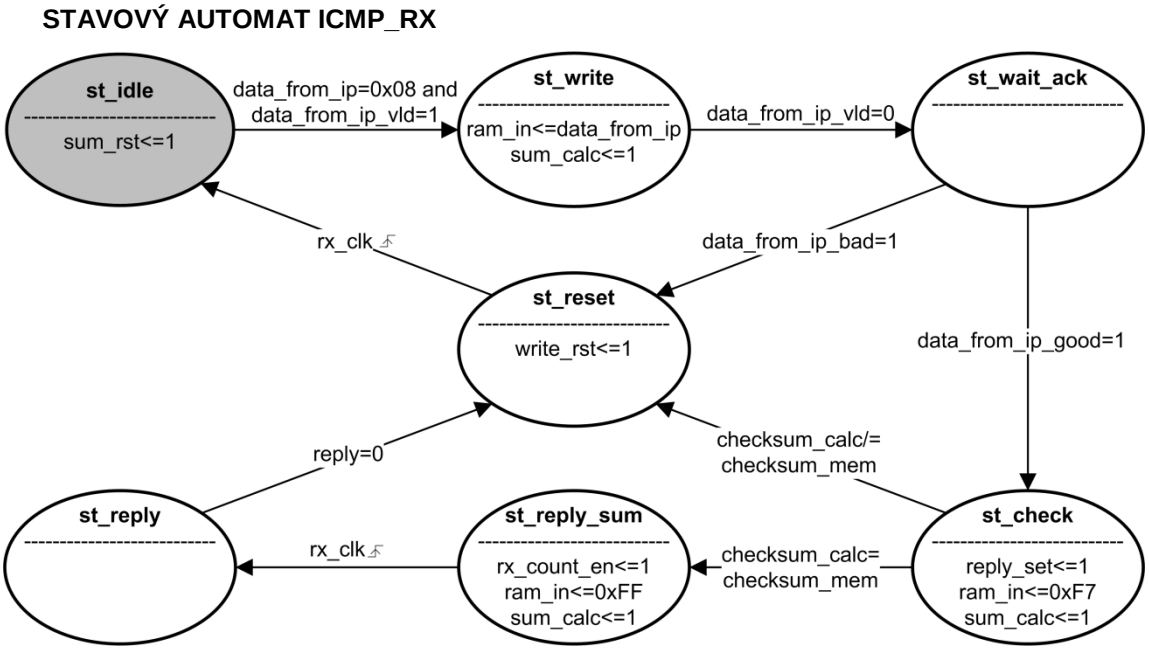
3.4.1 Přijetí dotazu Echo request

Přijetí dotazu zpracovává stavový automat *icmp_rx*, jehož diagram je na obrázku 40. Celý dotaz je uložen do blokové paměti. Kontrolní součet ICMP paketu se počítá z hlavičky i datové části. Výpočet provádí modul *checksum*. Paket typu Echo request je stejný jako paket typu Echo reply, kromě prvního pole hlavičky určující typ. Kontrolní součet odpovědi je výhodné vypočítat z kontrolního součtu přijatého dotazu podle rovnice (7), kde hodnota 0008_{HEX} je rozdíl hodnot polí typ dotazu a odpovědi.

$$CHECKSUM_{reply} = CHECKSUM_{request} - 0008_{HEX} \quad (7)$$

Modul *checksum* nepodporuje přímo operaci odčítání, ale je možné použít jednotkový doplněk pro implementaci této operace, jak ukazuje rovnice (8).

$$CHECKSUM_{reply} = CHECKSUM_{request} + FFF7_{HEX} \quad (8)$$



Obr. 40: Diagram stavového automatu *icmp_rx*

St_idle: Počáteční stav. Resetuje se modul *checksum*. Dotaz Echo request je detekován platnou hodnotou 08_{HEX} na sběrnici *data_from_ip*. Tím začne zápis příchozích dat do paměti *ICMP_RAM* a výpočet kontrolního součtu.

St_write: Zapíše zbytek paketu Echo request do paměti a vypočítá jeho kontrolní součet. Konec paketu je detekován logickou 0 na signálu *data_from_ip_vld*.

St_wait_ack: Stav čeká na potvrzení přijatého paketu (logickou 1 na signálech *data_from_ip_good* nebo *data_from_ip_bad*) vrstvou *ip_core*.

St_check: Trvá jeden hodinový cyklus. Porovná vypočítaný kontrolní součet se součtem v hlavičce. Pokud jsou totožné, je přijatý dotaz platný. Odečtením hodnoty 0008_{HEX} od výsledného kontrolního součtu se vypočítá součet pro odpověď.

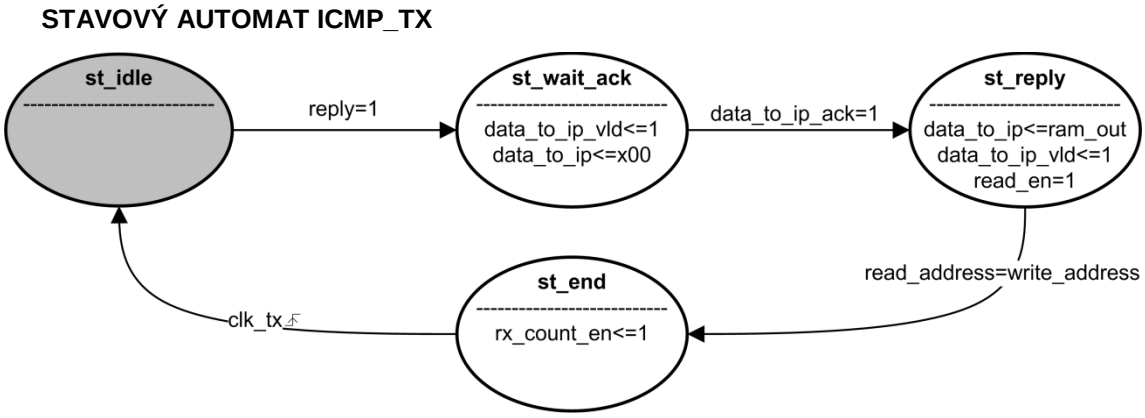
St_reply_sum: Započítá poslední oktet jednotkového doplňku do kontrolního součtu. Trvá jeden hodinový cyklus.

St_reply: Automat setrvává v tomto stavu, dokud není odeslána odpověď na přijatý dotaz, aby nedošlo k přepisu dat v paměti.

St_reset: Resetuje obsah paměti *ICMP_RAM*. Trvá jeden hodinový cyklus, po kterém se přechází do počátečního stavu.

3.4.2 Odeslání odpovědi Echo reply

Po přijetí platného paketu Echo request je aktivován stavový automat *icmp_tx*, který zajišťuje sestavení a odeslání odpovědi Echo reply do vrstvy *ip_core*. Jako odpověď jsou použita data uložená v blokové paměti. V hlavičce se změní pole *typ* na hodnotu 00_{HEX} a pole *checksum* na hodnotu signálu *checksum_calc*, který obsahuje kontrolní součet určený pro odpověď. Diagram stavového automatu *icmp_tx* je na obrázku 41.



Obr. 41: Diagram stavového automatu *icmp_tx*

St_idle: Počáteční stav. Výstupy jsou nastaveny do počátečních hodnot. Po příchodu požadavku na odeslání odpovědi se přechází do dalšího stavu.

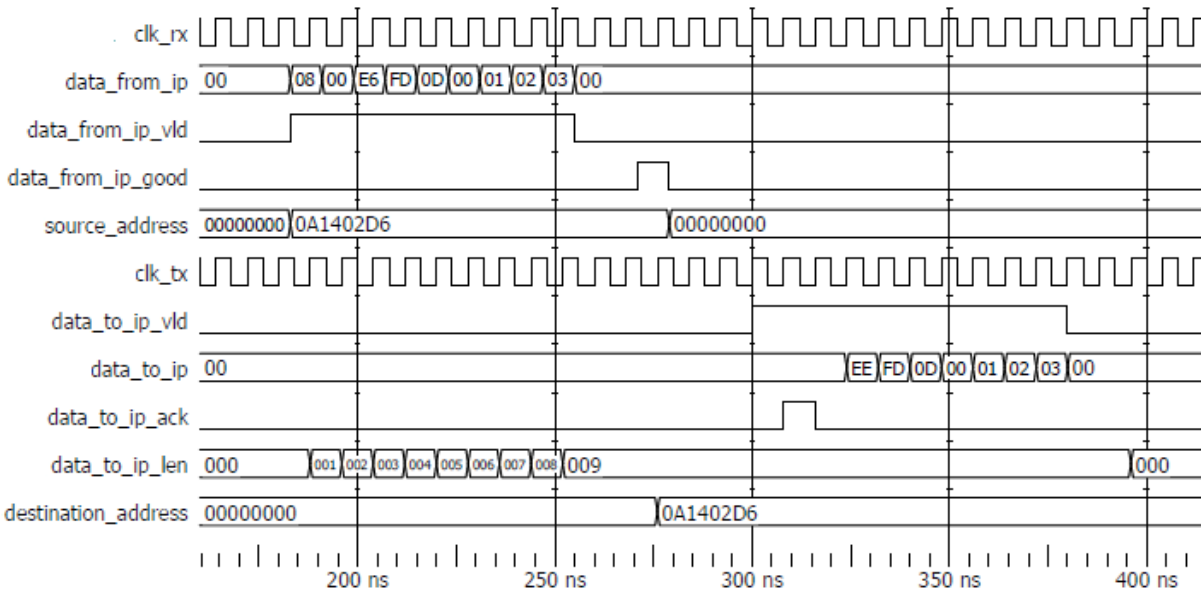
St_wait_ack: V prvním hodinovém cyklu je na sběrnici *data_to_ip* přiveden první oktet odpovědi Echo reply (hodnota 00_{HEX}), jeho platnost určuje logická 1 na signálu *data_to_ip_vld*. Po načtení prvního oktetu vrstvou *ip_core* přechází automat do dalšího stavu.

St_reply: Do nižší vrstvy jsou odeslána data z blokové paměti počínaje adresou 1 (adresa 0 reprezentuje *typ*, odeslaný v předchozím stavu). Na adrese 3 a 4 je kontrolní součet nahrazen přepočítanou hodnotou.

St_reply: Resetuje hodnotu registru *reply*, kterým uvolňuje paměť pro zápis. Po dalším hodinovém taktu přechází do počátečního stavu.

3.4.3 Simulace modulu icmp_core

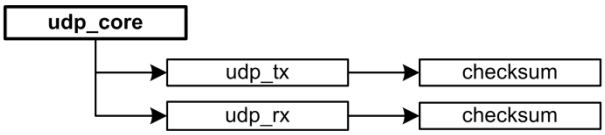
Na obrázku 42 je průběh simulace vrstvy *icmp_core* při dotazu Echo request. Dotaz má pro názornost jen 9B (6B hlavička + 3B data). Data jsou číslována vzestupně. Po přijetí dotazu je vrstvou odeslána odpověď.



Obr. 42: Simulace vrstvy *icmp_core* – reakce na paket Echo request

3.5 Modul udp_core

Modul implementuje UDP protokol. Skládá se z vrstvy pro vysílání *udp_tx* a vrstvy pro příjem *udp_rx*, které pracují nezávisle na sobě. Strukturu modulu zobrazuje obrázek 43.



Obr. 43: Struktura modulu *udp_core*

Výpočet kontrolních součtu zajišťuje modul *checksum*, popsaný v kapitole 3.3.1. Následující tabulky obsahují přehled ovládacích signálů modulu *udp_core*.

Tab. 16: Řídící hodinové signály bloku *udp_core*

Signál	Směr	Popis
RX_CLK	IN	Hodinový signál řídící přijatá data.
TX_CLK	IN	Hodinový signál řídící odesílaná dat.

Tab. 17: Signály pro přenos dat mezi *udp_core* a vyšší vrstvou

Signál	Směr	Popis
RX_DATA[7:0]	OUT	Výstupní 8bitová sběrnice pro přijatá data.
RX_DATA_VLD	OUT	Logická 1 určuje platnost dat na sběrnici RX_DATA.
RX_DATA_GOOD	OUT	Potvrzuje platnosti přijatého paketu UDP paketu.
RX_DATA_BAD	OUT	Potvrzuje neplatnost přijatého paketu UDP paketu.
TX_DATA[7:0]	IN	Vstupní 8bitová sběrnice pro odesílaná UDP data.
TX_DATA_VLD	IN	Logická 1 určuje platnost dat na sběrnici TX_DATA.
TX_DATA_EN	OUT	Logická 1 umožňuje zápis dat k odeslání do blokové paměti.

Tab. 18: Signály pro přenos dat mezi *udp_core* a vrstvou *ip_core*

Signál	Směr	Popis
DATA_FROM_IP[7:0]	IN	Vstupní 8bitová sběrnice pro přijatá IP data.
DATA_FROM_IP_VLD	IN	Logická 1 určuje platnost dat na sběrnici DATA_FROM_IP.
DATA_FROM_IP_GOOD	IN	Potvrzení platnosti přijatého paketu.
DATA_FROM_IP_BAD	IN	Přijatý paket je neplatný.
DATA_TO_IP[7:0]	OUT	Výstupní 8bitová sběrnice pro data odesílaná do *ip_core*.
DATA_TO_IP_VLD	OUT	Logická 1 určuje platnost dat na sběrnici DATA_TO_IP.
DATA_TO_IP_LEN[param:0]	OUT	Počet bajtů UDP paketu. Param závisí na velikosti MTU.
DATA_TO_IP_ACK	IN	Potvrzení přijetí prvního oktetu vrstvou *ip_core*.

Tab. 19: *Řídící a směrovací signály*

Signál	Směr	Popis
DESTINATION_PORT[15:0]	IN	UDP port cílové aplikace.
DESTINATION_IP[31:0]	IN	Cílová IP adresa. Slouží pro výpočet kontrolního součtu.
SOURCE_IP[31:0]	IN	Zdrojová IP adresa. Slouží pro výpočet kontrolního součtu.
RESET	IN	Synchronní reset.

UDP port obvodu FPGA je nastaven konstantou v souboru *my_constants.vhd* a není možné jej po implementaci měnit. Data, která nejsou směrována na tento port, se vyhodnotí jako neplatná.

Konstanty pro nastavení vrstvy udp_core:

```
constant UDP_ADDR_WIDTH   : integer;                    -- maximální velikost UDP paketu
constant FPGA_PORT        : std_logic_vector(15 downto 0); -- UDP port modulu udp_core
```

Kontrolní součet UDP paketu se počítá i z datové části. To vyžaduje, aby data určená k odeslání, byla nejprve uložena do paměti. Vysílací část proto obsahuje parametricky nastavitelnou blokovou paměť, zapojenou jako FIFO. Velikost paměti je nastavitelná parametricky šířkou adresové sběrnice *UDP_ADDR_WIDTH* a je dána rovnicí (9).

$$velikost\ RAM = 8 \cdot 2^{UDP\_ADDR\_WIDTH}\ [B] \qquad (9)$$

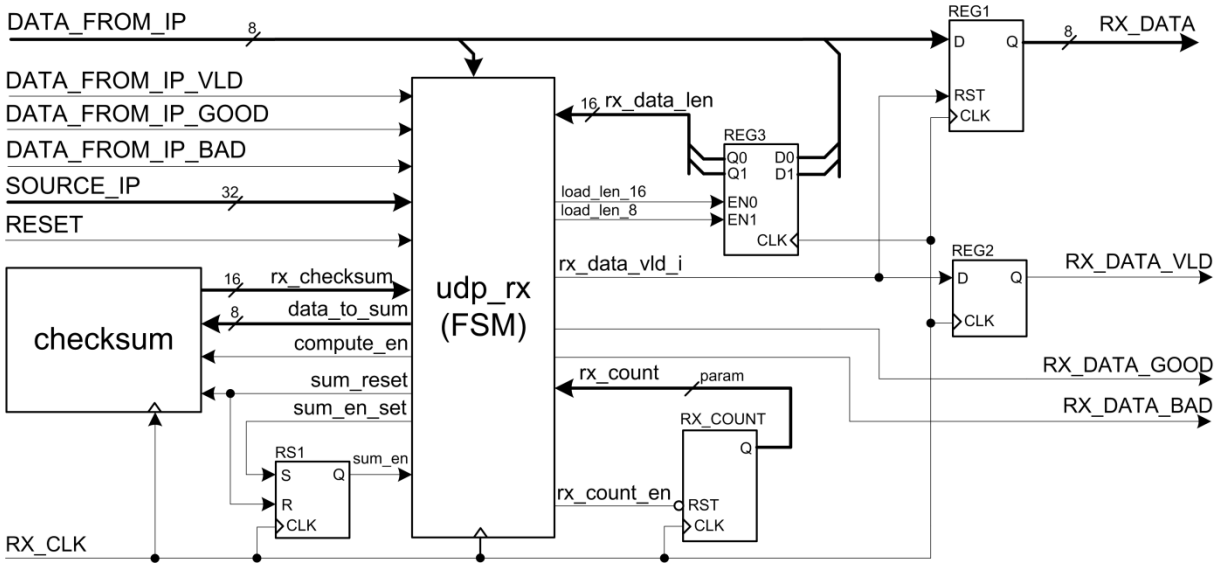
Uživatel si musí při odesílání datových bloků ohlídat zaplnění této paměti buď vlastním čítačem, nebo signálem *TX_DATA_EN*, který vrstva *udp_core* nastaví do logické 0 v případě zaplněné paměti. Celková délka MTU je rovna *velikosti RAM + 8+20.*

3.5.1Přijímací část – modul udp_rx

Modul *udp_rx* zpracovává UDP pakety přijaté z vrstvy *ip_core*. Zpracování paketu zahrnuje následující operace:

- Oddělení datové části od hlavičky a odeslání dat do vyšší vrstvy.
- Filtrace paketů směrované na jiný cílový port než *FPGA_PORT*.
- Kontrola platnosti přijatého paketu výpočtem kontrolního součtu.

Blokové schéma modulu *udp_rx* je na obrázku 44. Skládá se z čítače *RX_COUNT* s parametrickou šířkou výstupu (*UDP_ADDR_WIDTH-1*), který slouží k identifikaci jednotlivých oktetů v paketu. Modul *checksum* počítá kontrolní součty, je ovládán stavovým automatem *udp_rx*. Synchronní klopný obvod RS1 detekuje nulové pole kontrolního součtu v hlavičce (tzn. výpočet je zakázán). Do registru *REG3* je z hlavičky uložena informace o délce přijatého paketu. Stavový automat *udp_rx* má registrové výstupy, jeho funkci znázorňuje diagram na obrázku 45.



Obr. 44: Blokové schéma vrstvy *udp_rx*

STAVOVÝ AUTOMAT UDP_RX

St_idle: Počáteční stav. Resetuje modul *checksum*. Všechny výstupy jsou nastaveny do počátečních hodnot. Automat zůstává v tomto stavu do příchodu platných dat z vrstvy *ip_core*.

St_header: Analyzuje UDP hlavičku a počítá kontrolní součet. Celková délka paketu se uloží do registru *rx_data_len*. Ověří se, zda je v daném paketu povolen výpočet kontrolního součtu a podle toho je nastavena výstupní hodnota klopného obvodu RS1. Porovná se cílový port s konstantou *FPGA_PORT*, pokud jsou data směrována na jiný port, automat přejde do stavu *st_error*.

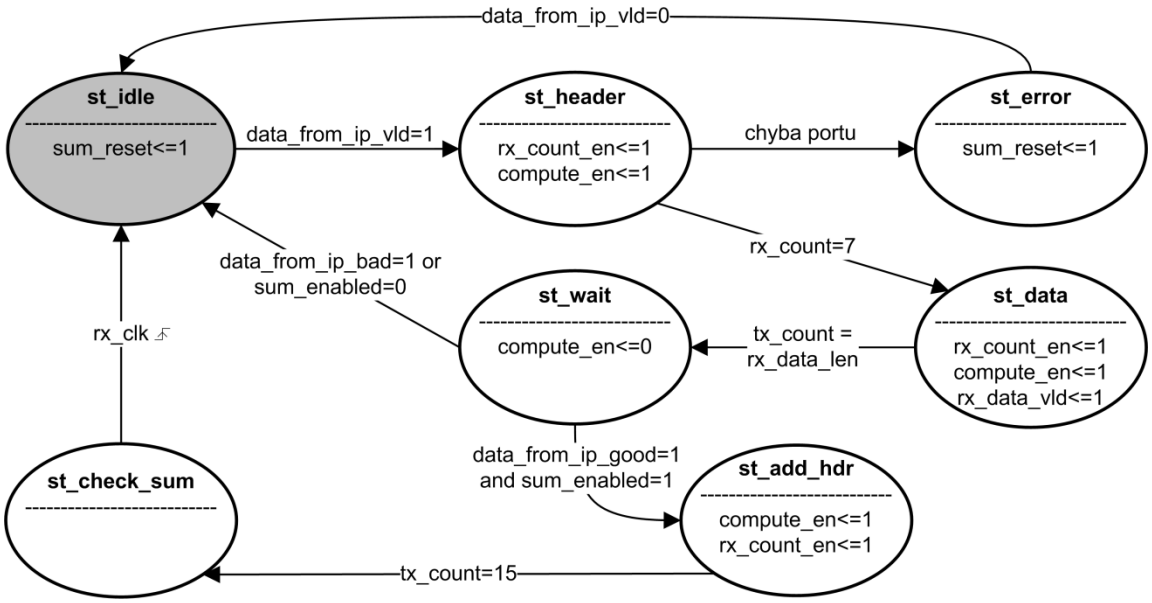
St_error: Čeká v tomto stavu v případě, že je přijatý paket směrován na jiný UDP port. Do počátečního stavu se přechází po ukončení přenosu paketu.

St_data: Přijatá data se odesílají do vyšší vrstvy po sběrnici *rx_data*, jejich platnost je určena logickou 1 na signálu *rx_data_vld*. Počítá se kontrolní součet. Do následujícího stavu přechází automat po odeslání všech oktetů datové části UDP paketu.

St_wait: Stav čeká na potvrzení platnosti přijatého paketu. Signál *data_from_ip_bad* značí, že přijatá data jsou neplatná. V tomto případě přechází automat do počátečního stavu a generuje pro vyšší vrstvu jedničkový impulz na výstupním signálu *rx_data_bad*. Signál *data_from_ip_good* značí, že přijatá data jsou pořádku. Ověří se, zda je povolen výpočet kontrolních součtů. Pokud ne, přechází se do počátečního stavu a data se označí jako platná jedničkovým impulzem na signálu *rx_data_good*. Pokud je povolen výpočet kontrolního součtu, musí se dopočítat pseudohlavička k aktuální hodnotě součtu.

St_add_hdr: Připočítá pseudohlavičku k aktuálnímu kontrolnímu součtu. Celý výpočet trvá 15 hodinových cyklů. Během této doby není možné zahltit modul dalším UDP paketem, protože hlavička IP paketu je zpracovávána 20 hodinových cyklů vrstvou *ip_core*.

St_check_sum: Konečný stav. Trvá 1 hodinový cyklus, po kterém automat přechází do počátečního stavu. Pokud je výsledný kontrolní součtu správný, generuje se jedničkový impulz na signálu *rx_data_good*, v opačném případě na signálu *rx_data_bad*.



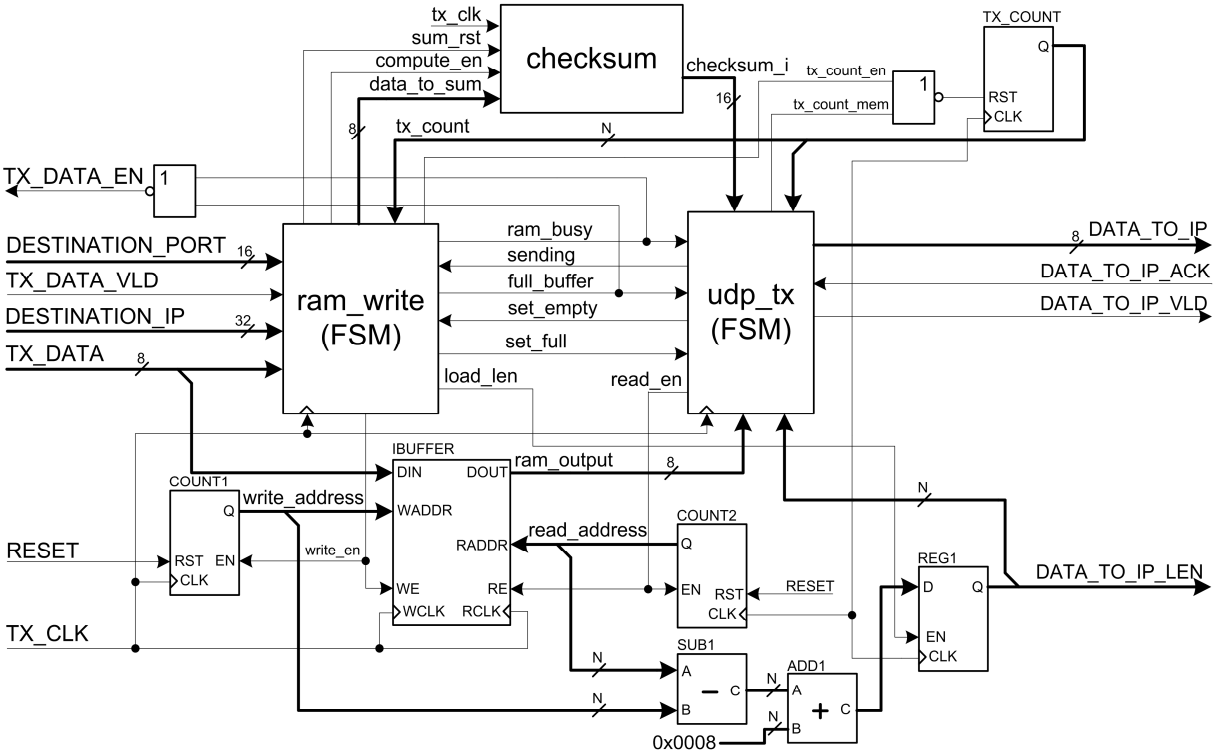
Obr. 45: Diagram stavového automatu *udp_rx*

3.5.2 Vysílací část – modul udp_tx

Vrstva *udp_tx* odesílá do modulu *ip_core* uživatelská data zabalená do UDP paketu. To zahrnuje následující operace:

- Uložení uživatelských dat do blokové paměti.
- Výpočet kontrolního součtu a jeho dosazení do UDP hlavičky.
- Připojení UDP hlavičky k uživatelským datům.
- Odeslání UDP paketu do vrstvy *ip_core*.

Schéma zapojení vysílací části je na obrázku 46. Data k odeslání se nejprve uloží do blokové paměti *IBUFFER*. Zápis do paměti adresuje čítač *COUNT1*. Paměť je zapojena jako FIFO, vejde se do ní vždy jeden paket. Další data je možné zapsat až po odeslání hlavičky aktuálního paketu, aby nedošlo k přepisu aktuálních dat v paměti. Signál *TX_DATA_EN* v logické 1 informuje vyšší vrstvu, že je možné zapsat nová data. Signál *full_buffer* indikuje data k odeslání v paměti *IBUFFER*. Čtení z paměti adresuje čítač *COUNT2*. Identifikace jednotlivých polí hlavičky a kontrola délky paketu je zajištěna čítačem *TX_COUNT*. Paměťový registr *REG1* uchovává celkovou délku UDP paketu, kterou počítají sčítačky *SUB1* a *ADD1* z adres paměti *write_address* a *read_address*. Parametr *N* udává uživatelsky definovanou šířku sběrnice, která závisí na konstantě *UDP_ADDR_WIDTH*.



Obr. 46: Schéma zapojení modulu *udp_tx*

STAVOVÝ AUTOMAT WRITE_RAM

Stavový automat *ram_write* ukládá vstupní data k odeslání do blokové paměti a ovládá modul *checksum*. Popis funkce stavového automatu ukazuje diagram na obrázku 47.

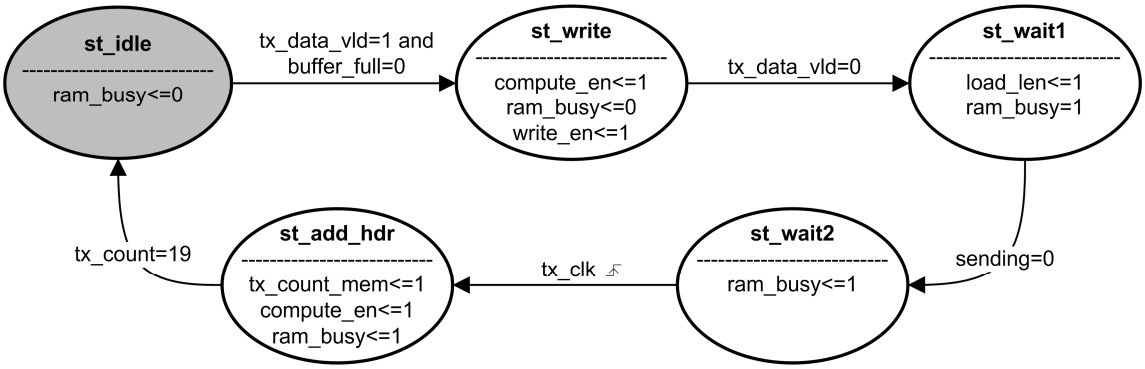
St_idle: Počáteční stav. Automat v něm zůstává, dokud není oznámena platnost uživatelských dat z vyšší vrstvy na sběrnici *tx_data* signálem *tx_data_vld* v logické 1.

St_write: Zapisuje příchozí data ze sběrnice *tx_data* do paměti RAM, a počítá z nich kontrolní součet v reálném čase.

St_wait1: Stav čeká na dokončení odeslání předchozího paketu, aby se mohl použít registr pro načtení celkové délky. Signál *ram_busy* zablokuje další přístup vyšší vrstvy do paměti. Celková délka nového paketu se načte do registru *REG1* až po dokončení přenosu.

St_wait2: Stav má za úkol vyčkat 1 hodinový cyklus z důvodu dokončení výpočtu aktuálního kontrolního součtu.

St_add_hdr: Ke kontrolnímu součtu připočítá pseudohlavičku UDP paketu. Po výpočtu, zůstává paměť stále zablokována, uvolnit ji může až stavový automat *udp_tx* při zahájení odesílání datové části UDP paketu.



Obr. 47: Diagram stavového automatu *ram_write*

STAVOVÝ AUTOMAT UDP_TX

Zajišťuje sestavení UDP paketu z hlavičky a datové části, uložené v paměti RAM a jeho odeslání do vrstvy *ip_core*. Funkci automatu znázorňuje diagram na obrázku 48.

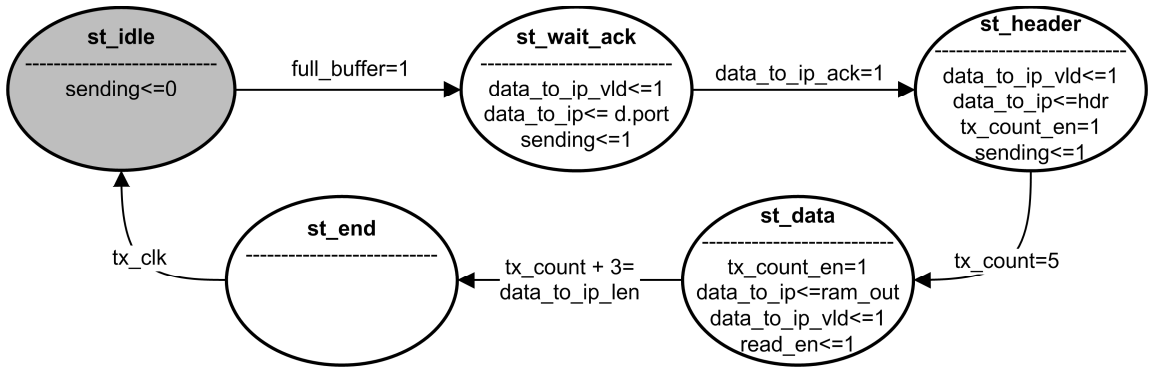
St_idle: Počáteční stav. Výstupy jsou nastaveny do počátečních hodnot. Automat v něm zůstává do doby, než je do paměti *IBUFFER* uložen blok dat z vyšší vrstvy, to detekuje signál *full_buffer* v logické 1.

St_wait_ack: Přivede na výstupní datovou sběrnici *data_to_ip* první oktet UDP hlavičky (horních 8 bitů 16bitového cílového portu *DESTINATION_PORT*) a potvrdí jeho platnost signálem *data_to_ip_vld*. Následně čeká na potvrzení načtení prvního bajtu vrstvou *ip_core*.

St_header: Čítač *tx_count* je spuštěn signálem *tx_count_en*. Během následujících šesti hodinových taktů se odešle do vrstvy *ip_core* zbytek UDP hlavičky. Na konci je signálem *set_empty* uvolněna paměť *IBUFFER* pro zápis a následuje přechod do stavu *st_data*.

St_data: Obsah paměti *IBUFFER* o délce *data_to_ip_len* je odeslán do vrstvy *ip_core*. V tuto chvíli je už možné spustit paralelně proces *ram_write*, řídící zápis dalšího paketu do paměti.

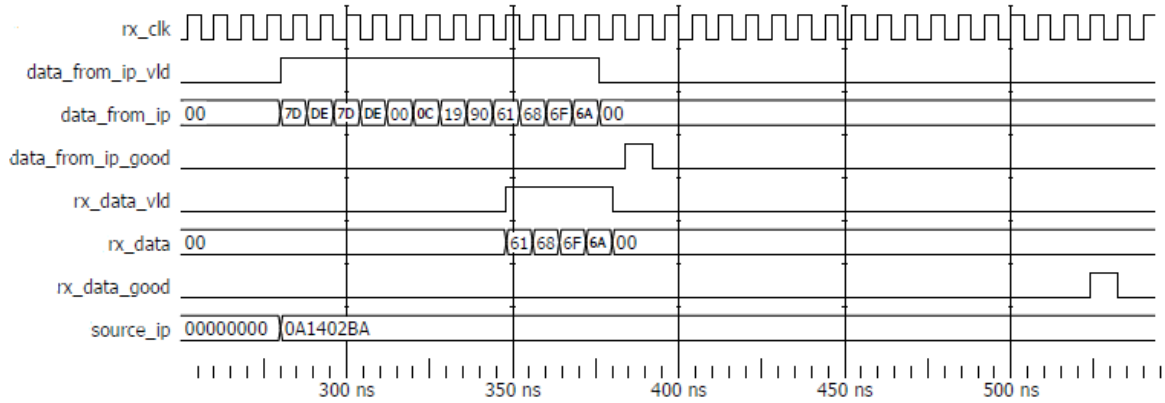
St_end: Konečný stav, trvá 1 hodinový cyklus. Slouží k nastavení vnitřních signálů automatu do počátečních hodnot.



Obr. 48: Diagram stavového automatu *udp_tx*

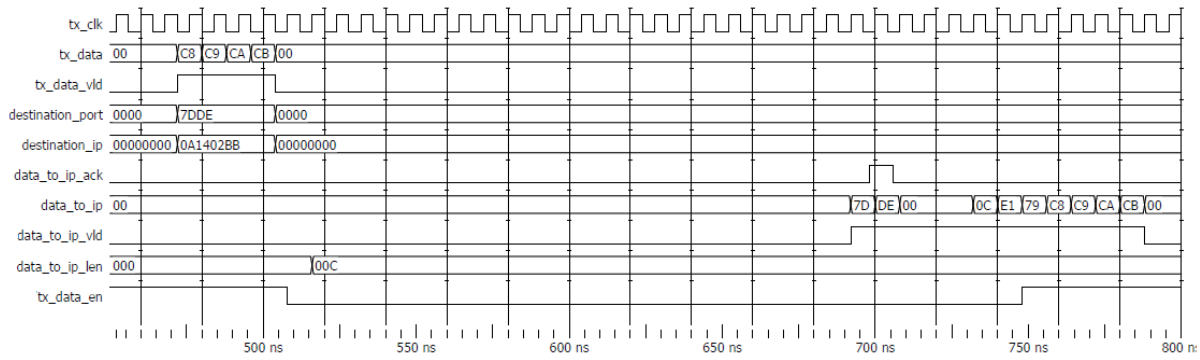
3.5.3 Simulace modulu udp_core

Simulace na obrázku 49 ukazuje přijetí paketu vrstvou *udp_core*. Pro názornost byl vytvořen paket délky pouze 12B (8B hlavička + 4B data). Simulace ukazuje, že data odeslaná do vyšší vrstvy, jsou potvrzena až po 17 hodinových taktech. Důvodem je započítání pseudohlavičky do kontrolního součtu.



Obr. 49: Simulace přijetí platného UDP paketu

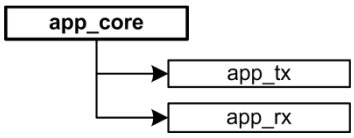
Simulace na obrázku 50 ukazuje odeslání uživatelských dat do vrstvy *ip_core*. V první části jsou data uložena do blokové paměti a následných 17 hodinových taktu je počítán kontrolní součet z pseudohlavicky. Po dokončení výpočtu je sestavený UDP paket odeslán do vrstvy *ip_core*.



Obr. 50: Simulace odeslání UDP paketu

3.6 Modul app_core

Modul implementuje potvrzovací protokol pro zajištění doručení dat. Struktura modulu je na obrázku 51. Skládá se z vysílací a přijímací části. Přijímací část *app_rx* generuje na základě přijatých paketů řídící signály pro vysílací vrstvu *app_tx*.



Obr. 51: Struktura vrstvy *app_core*

Modul obsahuje parametrickou blokovou paměť *OBUFFER*, která uchovává odeslaná data do doby, než je potvrzeno jejich doručení. Pokud je výstupní paměť plná, není možné odesílat další data. Data z výstupní paměti jsou odeslána znovu, pokud do určité doby (timeout) nepřijde potvrzovací nebo chybový paket.

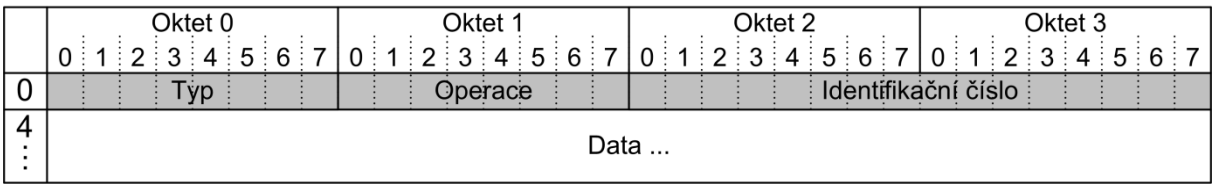
V rámci optimalizace přenosové rychlosti se nemusí potvrzovat každý odeslaný paket, ale celý datový blok, složený z několika paketů. Velikost bloku je možné nastavit konstantou v souboru *my_constants.vhd*.

Nastavení parametrů modulu app_core:

```
constant APP_ADDR_WIDTH : integer;                      -- Šířka adresové sběrnice výstupní paměti
constant WINDOW_SIZE    : std_logic_vector(3 downto 0); -- Velikost datového bloku (počet paketů)
constant TIMEOUT_WIDTH  : integer;                      -- Bitová šířka čítače pro timeout
```

3.6.1 Návrh komunikačního protokolu

Paket komunikačního protokolu ukazuje obrázek 52. Hlavička se skládá ze 4 oktetů. První oktet určuje typ paketu, druhý nese informaci o tom, zda má být daný paket potvrzen. Následuje 16bitové identifikační číslo, které zároveň slouží ke kontrole pořadí paketů. Přehled jednotlivých typů je uveden v tabulce 20.

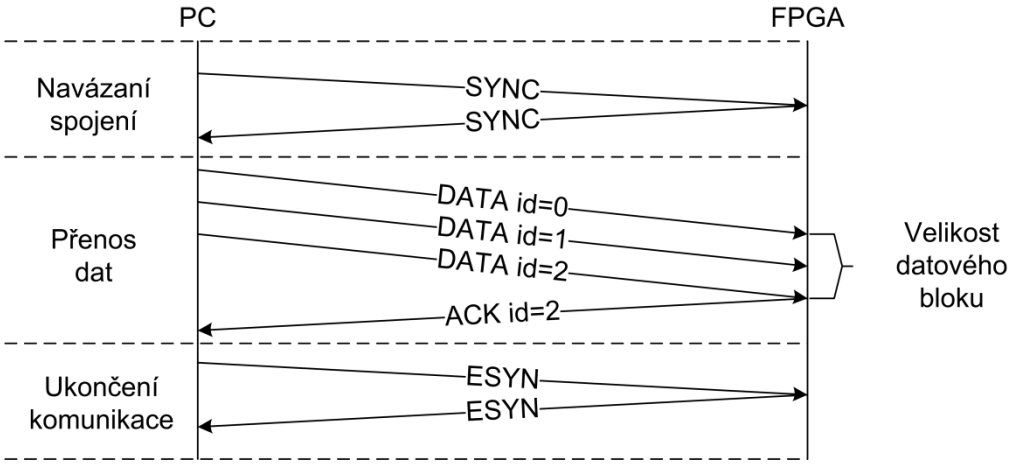


Obr. 52: Paket aplikačního protokolu pro zabezpečení přenosu.

Tab. 20: Popis pole typ aplikačního protokolu

Název	Hodnota	Popis
SYNC	0xAA	Synchronizační paket, slouží k navázání spojení a reset globálních čítačů identifikačních čísel.
ACK	0xBB	Potvrzovací paket, v poli identifikačního čísla je ID paketu, ke kterému potvrzení náleží. Všechny pakety s menším ID jsou v pořádku.
DATA	0xCC	Datový paket. Pokud je v poli operace hodnota 0x01, vyžaduje daný paket potvrzení. Pro menší zatížení sítě a zvýšení rychlosti přenosu není nutné potvrzovat každý přijatý paket, ale celé datové bloky. Velikost datového bloku je nastavitelná konstantou WINDOW_SIZE.
ESYN	0xDD	Ukončení komunikace.
ERROR	0xEE	Informuje odesílatele, že přijatý paket je chybný, nebo jeho identifikační číslo je jiné, než se očekávalo. V poli identifikačního čísla je uvedeno ID paketu, který je očekáván.

Princip komunikace ukazuje obrázek 53. Velikost výstupní paměti musí být větší nebo rovna velikosti datového bloku. Pokud je paměť dostatečně velká, umožňuje odeslat i několik datových bloků bez čekání na potvrzení, to značně zvýší rychlost přenosu.



Obr. 53: Model bezchybné komunikace aplikačního protokolu

3.7 Implementace modulu Eth0

Tabulka 21 ukazuje spotřebu registrů a LUT pro různé typy cílových obvodů. Je vidět, že nejmenší verze řady Spartan3 s 50k hradly nedokáže pojmout celý modul, proto při implementaci do této řady je nutné použít obvod s vyšším počtem hradel. Dostupné jsou

obvody XC3S200 nebo vyšší. Je možné také použít rozšířenou řadu Spartan3A nebo Spartan3E.

Obvod Virtex5 obsahuje 6vstupové LUT, proto je jejich spotřeba při implementaci podstatně menší než u obvodů nižších řad, které disponují pouze 4vstupovými LUT. Počet registrů zůstává stejný pro všechny typy obvodů. Spotřebu použitých zdrojů je možné do jisté míry (řádově několik %) ovlivnit nastavením syntetizátoru. V tabulce uvedený parametr *N* reprezentuje počet blokových pamětí RAM a závisí na volbě uživatele.

Tab. 21: Implementace jádra do různých řad FPGA obvodů

Blok	Virtex5 (XC5VSX50T) Registr/LUT/BRAM	Virtex4 (XC4VLX15) Registr/LUT/BRAM	Spartan3 (XC3S50) Registr/LUT/BRAM	Spartan3E (XC3S100E) Registr/LUT/BRAM
mac_core	202/240/0 (>1%)	202/313/0 (2%)	202/380/0 (26%)	202/380/0 (21%)
ip_core	273/490/0 (1%)	273/670/0 (5%)	273/676/0 (50%)	273/676/0 (39%)
icmp_core	74/132/1 (>1%)	74/153/1 (1%)	74/153/1 (11%)	74/153/1 (8%)
udp_core	135/318/1 (1%)	135/419/1 (3%)	135/426/1 (30%)	135/423/1 (24%)
app_core	184/244/N (>1%)	184/314/N (2%)	184/296/N (20%)	184/296/N (16%)
Eth0 (udp)	775/1242/N (3%)	775/1586/N (14%)	775/1600/2 (116%)	775/1600/2 (94%)
Eth0 (app)	949/1501/N (4%)	949/1871/N (16%)	949/1892/N (133%)	949/1890/N (110%)

3.8 Software pro komunikaci s modulem Eth0

Aplikace pro PC je naprogramována ve vývojovém prostředí C++ Builder 2010. Aplikace zajišťuje komunikaci s obvodem FPGA. Umožňuje odesílat celé soubory nebo ascii řetězce do cílového obvodu prostřednictvím UDP paketů. Přijatá data je možné uložit do souboru. Implementuje funkci Ping a obsahuje proceduru umožňující měřit rychlost přenosu.

Aplikace je více vláknová. Práci s vlákny umožňuje standardní knihovna C++ *TThread*. Data jsou přijímána v samostatném vláknu, které je zároveň ukládá do paměti. Samotný zápis do souboru provádí druhé vlákno s menší prioritou. Vlastní odeslání dat je potom implementováno třetím vláknem z důvodu maximalizace přenosové rychlosti.

3.8.1 Knihovna winsock

Komunikaci se sítí zajišťuje standardní programátorská knihovna Windows Socket (WinSock). Přistupuje ke komunikaci pomocí tzv. socketů. Socket definuje jednotlivá připojení, může navazovat TCP spojení, přijímat a odesílat pakety apod. Socketů může být najednou otevřeno více (např. pro komunikaci na různých UDP portech apod.). Aplikace se na ně odkazuje prostřednictvím pointerů.

Pokud chceme použít socket, musí nejprve proběhnout inicializace knihovny WinSock a nastavení socketu pro příslušný typ komunikace. Po úspěšné inicializaci je možné používat

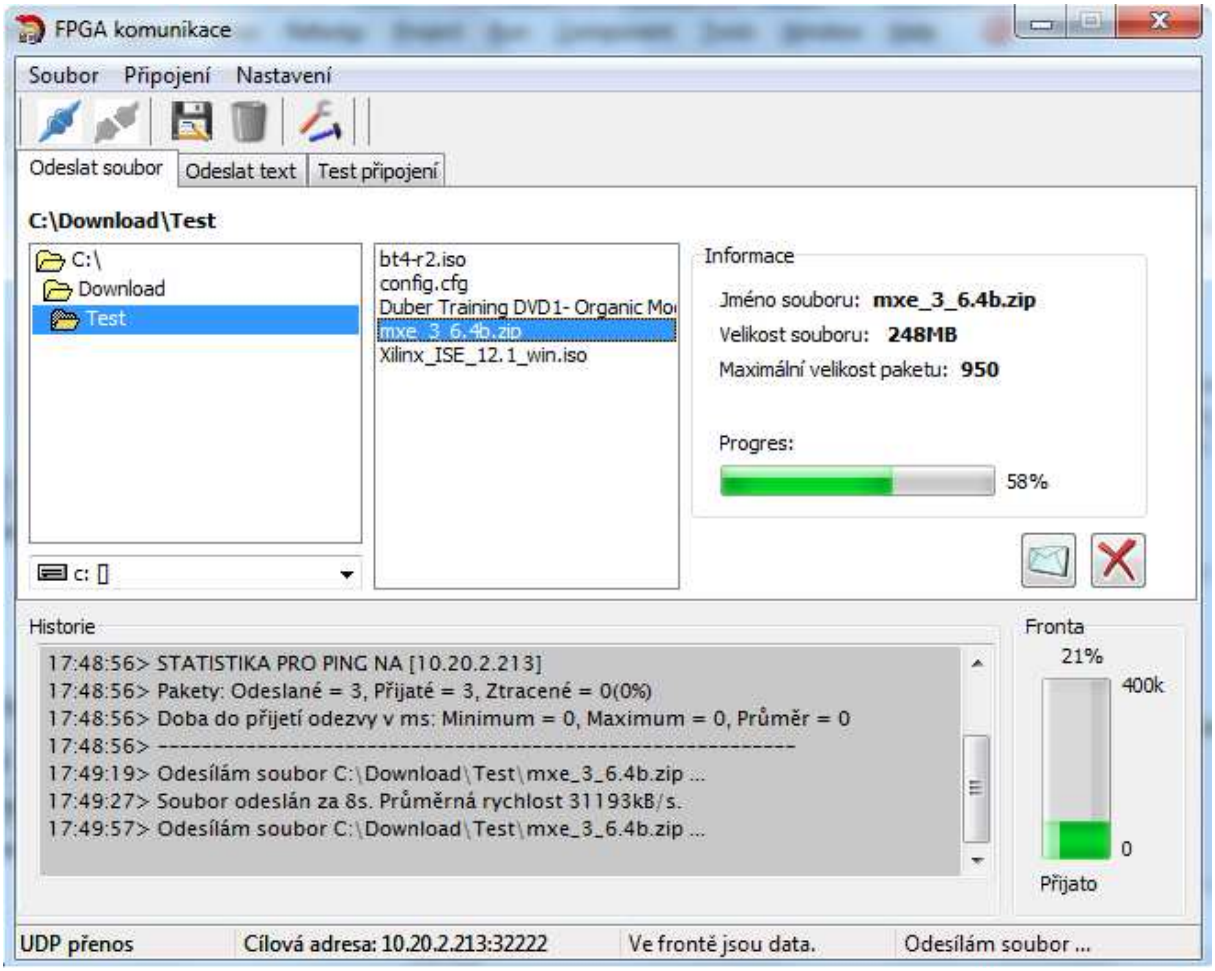
základní funkce přenosu dat *recvfrom()* a *sendto()*. Syntaxe pro jednotlivé funkce a popis parametrů je dostupný v dokumentaci Winsock MSDN [19].

3.8.2 Popis aplikace

Aplikace podporuje dva režimy komunikace. První režim je určen pro modul *Eth0* s koncovou vrstvu *udp_core*. Program pouze vysílá nebo přijímá UDP pakety.

Druhý režim je určen pro modul *Eth0* s koncovou vrstvou *app_core*. Přenos probíhá také prostřednictvím UDP paketů, ale je opatřen potvrzovacím protokolem v aplikační vrstvě.

Ukázka aplikace je na obrázku 54. Obsahuje tři záložky. Záložka *Odeslat soubor* umožňuje přes jednoduchý souborový průzkumník odeslat libovolný soubor z PC do FPGA. Záložka *Odeslat text* slouží k odeslání ASCII řetězce na cílovou IP adresu prostřednictvím jednoho paketu. Záložka *Test připojení* implementuje program *Ping* pro testování odezvy.



Obr. 54: Aplikace pro přenos dat

Textové pole *Historie* složí jako zpětná vazba a informuje uživatele o všech důležitých událostech. Progres bar *Fronta* zobrazuje zaplnění paměti přijatými daty. Všechny důležité parametry je možné konfigurovat v nabídce nastavení.

3.9 Praktické testování modulu

Modul byl v praxi testován na několika síťových rozhraních v plně duplexním provozu. Při komunikaci mezi FPGA a PC rychlostí 1Gb/s jsou spíše než limitní rychlosti samotného modulu testovány limitní faktory PC.

Některé síťové karty podporují hardwarový výpočet kontrolních součtů síťových protokolů. Problém nastává ve chvíli, kdy je výsledný kontrolní součet roven 0000_{HEX}. V takovém případě by se měla kontrolní sekvence invertovat, to však síťová karta neprovádí a doplní součet nulový. Problém jde vyřešit zakázáním hardwarového výpočtu kontrolních součtů v nastavení síťové karty (Checksum Offload = off). V takovém případě je kontrolní součet počítán softwarově a je korektní i v uvedeném případě.

3.9.1 Test rychlosti přímého připojení

Použité PC obsahovalo integrovanou síťovou kartu *Realtek PCIe GBE* s podporou gigabitového Ethernetu. Ukázalo se, že síťová karta nezvládá minimální mezi-rámcovou mezeru, proto muselo být v modulu nastaveno IFG na hodnotu 03_{HEX}, tím se mezi-rámcová mezera zvýšila 3 krát. Velikost datové části UDP paketu byla nastavena na 950B. Velikost datového bloku pro zabezpečený přenos byla nastavena na 4 pakety s výstupní pamětí velikosti 8 paketů. U přímého propojení je ztrátovost paketů na přenosové trase prakticky nulová. V tabulce 22 jsou uvedeny naměřené přenosové rychlosti.

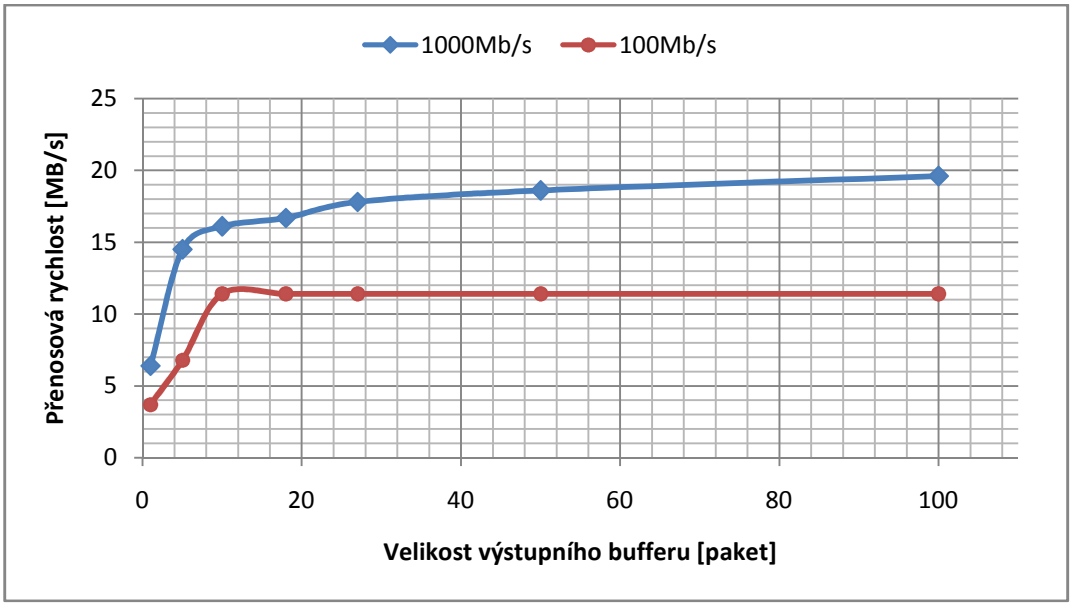
Ve směru PC→FPGA byl odeslán souboru velikosti 250MB. Nižší rychlost přenosu u 1Gb/s Ethernetu je způsobena především nedostatečným výkonem PC a pomalým harddiskem.

Tab. 22: Výsledky testu přenosu přímého propojení kabelem

Směr	UDP přenos	Zabezpečený přenos
Rozhraní 1000BASE-T		
FPGA → PC	57,6MB/s	18,7MB/s
FPGA ← PC	31,2MB/s	16,1MB/s
Rozhraní 100BASE-TX		
FPGA → PC	10,5MB/s	10,1MB/s
FPGA ← PC	11,5MB/s	10,7MB/s
Rozhraní 10BASE-TX		
FPGA → PC	1,1MB/s	1,0MB/s
FPGA ← PC	1,2MB/s	1,0MB/s

V opačném směru byla měřena doba, potřebná k zaplnění fronty přijatými daty, které generovalo FPGA. Rychlost přenosu v tomto směru je u 10/100Mb/s Ethernetu nižší. Je to způsobeno především nastavením IFG na nenulovou hodnotu a výpočtem kontrolní sekvence UDP paketu.

Dále byla měřena závislost přenosové rychlosti na velikosti výstupní paměti ve směru PC→FPGA při zabezpečeném přenosu. Velikost datového bloku byla nastavena na 5 paketů. Tato závislost je zakreslena v grafu na obrázku 55. Rychlosti jsou orientační a mohou se pro různá PC lišit. Z grafu je vidět, že efektivní velikost výstupní paměti je asi 10 paketů. Při dalším zvětšování paměti se již nárůst přenosové rychlosti tolik neprojevuje.



Obr. 55: Závislost přenosové rychlosti na velikosti výstupní paměti

4 Závěr

Cílem práce bylo navrhnout v jazyce VHDL a implementovat do obvodu FPGA modul umožňující obousměrnou komunikaci mezi obvodem FPGA a PC přes síťové rozhraní Ethernet. Komunikace měla probíhat přes protokol UDP a implementovány měly být všechny potřebné síťové protokoly. K ověření funkčnosti návrhu bylo potřeba naprogramovat aplikaci pro PC, umožňující přenos dat a funkci Ping.

Modul byl prakticky otestován na vývojové desce ML506, která je osazena obvodem FPGA Virtex5-XC5VSX50T, čipem fyzické vrstvy 88E1111 a konektorem RJ-45 pro připojení k Ethernetu. K testování datového přenosu bylo použito PC obsahující integrovanou síťovou kartu Realtek PCIe GBE podporující 1Gb/s Ethernet.

Modul umožňuje přenášet data po síti Internet. Podmínkou pro provozování v této síti je nutnost přiřazení veřejných IP adres pro modul i pro PC. Další podmínkou je použití vhodného UDP portu pro komunikaci, ten nesmí být blokován případným firewallem v datové cestě. UDP protokol nepodporuje řízení toku dat, proto takto odeslané pakety měly vysokou ztrátovost v důsledku přenosových limitů, které různí provozovatelé internetu poskytují. Řešení tohoto problému poskytuje vrstva *app_core*, která implementuje potvrzovací protokol.

Navržené rozhraní může najít uplatnění jako doplněk vývojové desky ML506, v podobě vysokorychlostního komunikačního rozhraní, při návrhu různých systémů a algoritmů, zpracovávající velký objem dat z PC (např. komprimační, hashovací nebo šifrovací algoritmy apod.). Pro použití stačí pouze implementovat blok do návrhu vyžadujícího vysokorychlostní datové rozhraní. Obvody Virtex5 jsou cenově značně nákladné a pro realizaci samostatného komunikačního rozhraní nevhodné.

Zajímavou alternativou může být využití modulu v cenově příznivých obvodech řady Spartan3, např. Spartan3A s 200k hradly, jehož cena se pohybuje kolem 300kč. Připojení k Ethernetu je možné zajistit čipem fyzické vrstvy např. snadno dostupným KSZ9021GQ od firmy Micrel Semiconductor. Tento obvod podporuje rozhraní GMII/MII a po připojení vnějšího oscilátoru generuje hodinový signál frekvence 125MHz, který je možné využít pro taktování modulu. Cena takto implementovaného rozhraní nepřesáhne 500kč. Takový návrh bude omezen pouze na rychlosti 10/100Mbit/s, protože obvody typu Spartan3 nejsou dostatečně rychlé pro zpracování paketů rychlosti 1Gbit/s.

Cíl práce byl splněn, výsledný návrh je funkční a umožňuje obousměrnou síťovou komunikaci na úrovni transportní vrstvy mezi obvodem FPGA a PC. Komunikace na úrovni síťové vrstvy je zajištěna implementovaným protokolem IPv4. Adresní prostor tohoto

protokolu je v současnosti vyčerpán, ale jeho kompletní nahrazení protokolem IPv6 bude ještě nějakou dobu trvat. Požadavek na možnost implementace modulu do cílových obvodů jiných řad byl zohledněn při návrhu, proto jsou komponenty omezující implementaci na jeden typ obvodu nahrazeny vlastním řešením.

5 Seznam použité literatury

[1] KOLKA, Zdeněk. *Počítačové a komunikační sítě*. Elektronická skripta FEKT VUT. Brno: 31.11.2007 [cit. 2011-04-04]

[2] DOSTÁLEK, Libor; KABELOVÁ, Alena. *Velký průvodce protokoly TCP/IP a systémem DNS*. 2. vydání. Praha: Computer Press, 2000. 426s. ISBN 80-7226-323-4.

[3] *IEEE 802.3 LAN/MAN CSMA/CD (Ethernet) Access Method*, (Revision of IEEE Std 802.3, 2008. Dostupný z WWW: <http://standards.ieee.org/getieee802/802.3.html>

[4] TIA/EIA Standard. *TIA/EIA-568-B.1*. Arlington: TIA, 2001. 94 s.

[5] CHOWDHURY, Dhiman. *High speed LAN technology handbook*. Berlin: Springer-Verlag, 2000. 473 s. ISBN 3-540-66597-8.

[6] KAPLAN, Hadriel; NOSEWOTHY, Robert. *The Ethernet Evolution From 10 Meg to 10 Gig: How it all Works!*. Atlanta: NetWorld Interop, 2001. 367 s. Dostupné z WWW: <http://www.scribd.com/doc/38770311/The-Ethernets-New-Handout>.

[7] *IEEE802.3ab: A tutorial presentation*. Irvine, CA: IEEE, 1997. 63 s. Dostupné z WWW: <http://grouper.ieee.org/groups/802/3/tutorial/march98/mick_170398.pdf>.

[8] *Trend Communications* [online]. 2011 [cit. 2011-12-04]. 1000BASE-T Architecture. Dostupné z WWW: <www.trendcomms.com>.

[9] *Standards.ieee.org* [online]. 2010 [cit. 2011-04-03]. IEEE-SA-Registration Authority. Dostupné z WWW: <http://standards.ieee.org/develop/regauth/oui/index.html>.

[10] *IANA-Internet Assigned Numbers Authority* [online]. 2010-07-15 [cit. 2011-04-04]. IANA. Dostupné z WWW: <http://www.iana.org/>.

[11] RFC791. *Internet protokol* [online]. IEFT, 1981. 45 s. Dostupné z WWW: <www.ietf.org/rfc/rfc791.txt>.

[12] RFC826. *An Ethernet Address Resolution Protocol* [online]. IETF, 1982. Dostupné z WWW: <www.ietf.org/rfc/rfc826.txt>

[13] RFC792, *Internet control message protocol* [online], September 1981. Dostupný z WWW: <www.ietf.org/rfc/rfc792.txt>

[14] RFC768. *User Datagram Protocol* [online]. IEFT, 28 August, 1980. Dostupný z WWW: <http://www.ietf.org/rfc/rfc768.txt>

[15] RFC1071. *Computing the Internet Checksum*. IEFT, September 1988. Dostupné z WWW: <www.faqs.org/rfcs/rfc1071.html>.

[16] *ML505/506/507 Reference design: User Guide*. Xilinx, 2009. 24 s. Dostupné z WWW: <www.xilinx.com/support/documentation/boards_and_kits/ug349.pdf>.

[17] *Virtex-5 FPGA Embedded Tri-Mode MAC: User Guide*. Xilinx, 2011. 224s. Dostupné z WWW: <www.xilinx.com/support/documentation/user_guides/ug194.pdf>.

[18] STAVINOV, Evgeni. *A Practical Parallel CRC Generation Method* [online]. 2010, Dostupný z WWW: <http://outputlogic.com/my-stuff/circuit-cellar-january-2010-crc.pdf>.

[19] *MSDN* [online]. Socket Reference (Windows). 2011. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/ms741416%28v=VS.85%29.aspx>

Příloha 1: Tabulky výpočtu CRC32

Tab. 23: Výsledky CRC výpočtu pro M_{IN} = 0 a N = one-hot kód

		M_{OUT}																													
	i	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
N_{IN}	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	1	0	0	0	1	1	1	0	1	1	0	1	1	0	1	1	1
	1	0	0	0	0	1	0	0	1	1	0	0	0	0	0	1	0	0	0	1	1	1	0	1	1	0	1	1	0	1	1	1	0
	2	0	0	0	1	0	0	1	1	0	0	0	0	0	1	0	0	0	1	1	1	0	1	1	0	1	1	0	1	1	1	0	0
	3	0	0	1	0	0	1	1	0	0	0	0	0	1	0	0	0	1	1	1	0	1	1	0	1	1	0	1	1	1	0	0	0
	4	0	1	0	0	1	1	0	0	0	0	0	1	0	0	0	1	1	1	0	1	1	0	1	1	0	1	1	1	0	0	0	0
	5	1	0	0	1	1	0	0	0	0	0	1	0	0	0	1	1	1	0	1	1	0	1	1	0	1	1	1	0	0	0	0	0
	6	0	0	1	1	0	1	0	0	1	0	0	0	0	1	1	0	0	1	1	1	0	0	0	0	0	1	1	1	0	1	1	0
	7	0	1	1	0	1	0	0	1	0	0	0	0	1	1	0	0	1	1	1	0	0	0	0	0	1	1	1	0	1	1	1	0

Tab. 24: Výsledky CRC výpočtu pro M_{IN} = one-hot kód a N = 0

		M_{OUT}																													
	i	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
M_{IN}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	10	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	11	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	12	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	13	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	14	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	15	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	17	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	18	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	20	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	21	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	22	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	23	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	24	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	1	0	0	0	1	1	1	0	1	1	0	1	1	0	1	1	1
	25	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0	0	1	1	1	0	1	1	0	1	1	0	1	1	1	0
	26	0	0	0	0	1	0	1	1	0	0	0	0	0	1	0	0	0	1	1	1	0	1	1	0	1	1	0	1	1	1	0	0
	27	0	0	0	1	0	1	1	0	0	0	0	0	1	0	0	0	1	1	1	0	1	1	0	1	1	0	1	1	1	0	0	0
	28	0	1	0	0	0	1	0	0	0	0	0	1	0	0	0	1	1	1	0	1	1	0	1	1	0	1	1	1	0	0	0	0
	29	1	0	0	0	1	0	0	0	0	0	1	0	0	0	1	1	1	0	1	1	0	1	1	0	1	1	1	0	0	0	0	0
	30	0	0	0	1	1	1	0	0	1	0	0	0	0	1	1	0	0	1	1	1	0	0	0	0	0	1	1	1	0	1	1	1
	31	0	1	0	1	0	0	0	1	0	0	0	0	1	1	0	0	1	1	1	0	0	0	0	0	1	1	1	0	1	1	1	0

Příklad určení rovnice pro 26. bit CRC32:

$M_{OUT}(26)$ <= N(6) xor N(4) xor N(3) xor N(0) xor $M_{IN}(18)$ xor M_{IN} (24) xor M_{IN} (27) xor M_{IN} (28) xor $M_{IN}(30)$;

Příloha 2: Obsah přiloženého CD

```
.
| – DP
|     \ – dp.pdf                    % Textová část diplomové práce
| – SRC                             % Adresář zdrojových kódů
|     | – Eth0udp                   % Adresář s modulem pro UDP přenos
|     |       | – Implementation    % Adresář se soubory k implementaci
|     |       \ – Simulation        % Test bench soubory pro simulaci
|     | – Eth0app                   % Adresář s modulem pro zabezpečený přenos
|     |       | – Implementation    % Adresář se soubory k implementaci
|     |       \ – Simulation        % Test bench soubory pro simulaci
|     | – Ethernet                  % Demonstrační ukázka pouzití modulu v projektu
|     \ – Aplikace                  % Zdrojové texty s testovací aplikací pro PC
| – Report                          % Obsahuje výsledky simulací
\ – obsah.txt                       % Obsah CD
```