České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra řídicí techniky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Miroslav Kubíček**

Studijní program: Kybernetika a robotika
Obor: Systémy a řízení

Název tématu: **Přenos dat po Ethernetu mezi vývojovými deskami FPGA**

Pokyny pro vypracování:

1. Navrhněte komunikaci po ethernetu mezi FPGA vývojovými deskami DE2 pomocí nízkoúrovňových protokolů.
2. Vypracujte samostatný modul ve VHDL nepoužívající procesor NIOS, který dovolí číst a zapisovat datový blok pevné délky do protilehlé desky DE2.
3. Komunikaci posléze rozšiřte i na propojení desky s programem v počítači PC.

Seznam odborné literatury:

[1] Pong P. Chu: RTL Hardware Design Using VHDL: Coding for Efficiency, Portability, and Scalability. Wiley-IEEE Press 2006
[2] W. Richard Stevens, TCP/IP Illustrated, Addison-Wesley Professional 1993
[3] TCP/IP Tutorial and Technical Overview, IBM 2006

Vedoucí: Ing. Richard Šusta, Ph.D.

Platnost zadání: do konce letního semestru 2015/2016

L.S.

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 29. 9. 2014

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra řídicí techniky

Bakalářská práce

Přenos dat po Ethernetu mezi vývojovými deskami FPGA

Kubíček Miroslav

Vedoucí práce: Ing. Richard Šusta, Ph.D.

4. ledna 2015

Poděkování

Rád bych poděkoval panu Ing. Richardu Šustovi, Ph.D., vedoucímu mého projektu, za vstřícnost, ochotu a cenné rady, které mi pomohly k vytvoření projektu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, avšak pouze k nevýdělečným účelům. Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 4. ledna 2015 .....

Odkaz na tuto práci

Kubíček, Miroslav. *Přenos dat po Ethernetu mezi vývojovými deskami FPGA*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta elektrotechnická, 2015.

Abstract

This thesis deals with 100Mbit/s ethernet communication at FPGA chip Altera DE2-115. Sending of data is handled via writing of them into SRAM memory whereas there's an ethernet frame sent with respective signal. While accepting packet, it is stored in an internal memory where it awaits signal which frees memory. Thesis describes MII interface via which we communicate with the chip which is afterwards responsible for the communication with outer world.

Keywords Ethernet,Ethernet communication, FPGA, Altera DE2-115,VHDL

Abstrakt

Tato bakalářská práce se zabývá 100Mbit/s ethernetovou komunikací na FPGA desce Altera DE2-115. Odesílání probíhá zapsáním dat do SRAM paměti, z které je s příslušným signálem odeslán ethernetový rámec. Při příjmu je paket uložen do vnitřní paměti, kde čeká na signál, který paměť opět uvolní. Práce popisuje rozhraní MII, za pomoci kterého komunikujeme s ethernetovým čipem, které komunikuje s okolním světem pomocí Ethernetu.

Klíčová slova Ethernet,Ethernetová komunikace, FPGA, Altera DE2-115,VHDL

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Bakalářská práce se zaměřuje na FPGA vývojovou desku DE2-115 Cyclone IV [4], kterou budeme dále označovat jenom jako DE2-115. Rozšiřuje její možností o komunikaci pomocí síťových protokolů Ethernetu, a to o přenos dat uložených ve vnitřní paměti DE2-115 na připojené druhé zařízení, například na druhou DE2-115 nebo počítač. Nalezená dosavadní řešení nepracují samostatně, ale používají soft-core procesor NIOS [5] jako svou řídící jednotku, takže do návrhu se přidává zbytečná složitost a také přenos dat je mnohem pomalejší. NIOS využívá jak komerční licencované řešení "Triple-Speed Ethernet MegaCore"[6], nabízené firmou Altera, tak i o open-source modul "Ethernet Communication Interface"for FPGA [7]] napsaný autory Michael Spanier a Alexander Gorenstein na Cornell University Unit Signature. Jejich projekt lze sice používat bezplatně, ale žel není dosud plně funkční.
Modul, který jsem navrhnul v rámci bakalářské práce, se liší od stávajících řešení tím, že pracuje zcela samostatně a dá se použít i u malých projektů. Stačí jen zapsat data do vnitřní paměti FPGA obvodu a poté inicializovat jejich přenos. Lze tak snadno propojit obvod navržený na DE2-115 s počítačem, či jiným zařízením.

Kapitola 1

Rešerše

1.1 Obecný popis funkcí Ethernetu

1.1.1 Vznik a historie

Myšlenka Ethernetu vznikla roku 1973, komerčně se poprvé využila v roce 1980. V roce 1983 vznikl i jeho standard IEEE 802.3. Díky jeho všudypřítomné a levné kabeláži (kroucené dvojlinky) se stal dominantní pro síťové spojení. První ethernetová verze zvládla komunikovat rychlostí 10 Mbit/s. V dnešní době bývá běžně dostupný i Gigabitový ethernet. Existují i jeho pokročilejší verze pro rychlosti až rychlostí 100Gbit/s a vyvíjejí se další rychlosti jako terabitový až 100 terabitový Ethernet [11]. Nejpoužívanější rychlostí pro většinu síťových zařízení je stále 100 Mbit/s Fast Ethernet, který lze propojit 4 bitovým 25 MHz synchronním paralelním rozhraním známým jako MII (anglicky Media Independent Interface) nebo 2bitovou 50 MHz variantou RMII (anglicky Reduced Media Independent Interface) [8]. Naše PHY rozhraní RMII nepodporuje [9], využijeme tedy rozhraní základní. Fast Ethernet používá 2 páry vodičů až do garantované vzdálenosti 100 metrů [8]. Nazývá se 100Base-TX a jím se budeme zabývat i my. Cílem této práce je vytvořit aplikaci pro výuku slovní zásoby, která by uživatelům umožnila učit se nová slova co nejefektivněji. Výsledná aplikace bude prvotně zaměřena na jednu z mobilních platformem.

1.2 Způsob komunikace

1.2.1 Polo duplexní režim - Přístupu ke sdílenému médiu CSMA/CD

Pokud používáme pro komunikaci koaxiální kabel nebo je v naší síti hub zařízení, pak ethernetová komunikace bude probíhat v polo-duplexním módu.

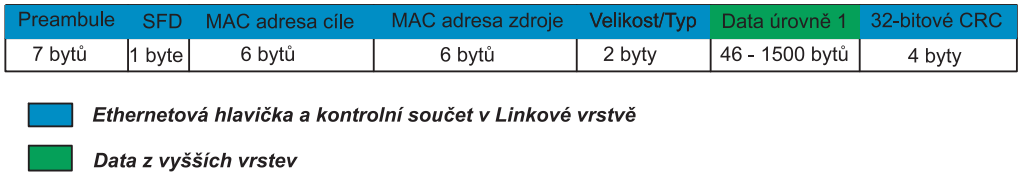
Pro správný chod sítě je nutné, abychom zavedli algoritmus CSMA/CD (Carrier Sense Multiple Access with Collision Detection). Chce-li stanice vyslat datový rámec nejdříve naslouchá sběrnici a čeká, dokud není sběrnice volná. Následně zahájí vysílání rámce a zároveň naslouchá, zda nepřichází signál z jiné stanice a nedošlo tím ke kolizi. Pokud ano, vyšle jam signál a vyčká náhodnou dobu, po které se pokusí vyslat rámec znovu.

1.2.2 Plně duplexní režim

V současné době se již nevyplatí stavět počítačové sítě s huby neboli rozbočovač, neboť jejich cena je srovnatelná s klasickými sítěmi se switch zařízeními neboli přepínače. Počítač je tedy připojen ke switchi, který odesílá pakety pouze cílovému počítači, pro které jsou data určena. Klasická dvojlinka obsahuje 4 páry vodičů, z nichž 2 páry jsou určené pro odesílání a 2 páry pro příjem. Je jasné, že páry vodičů pro odesílání u zdrojového zařízení jsou u cílového zařízení jako páry pro příjem. Propojíme-li počítač se switchem, nemusíme se o nic starat a použijeme přímý kabel, neboťi switch si automaticky prohodí páry vodičů pro příjem za páry na odesílání. Spojením zařízení na stejné úrovni, jako je například propojení dvou počítačů (peer-to-peer), je třeba páry prohodit použitím kříženého kabelu. Jedná se však opět o starší metodu, kde síťové karty nepodporovali tzv. Auto MDI-X. Auto MDI-X je procedura, která zajistí páru vodičů použitým zařízením pro vysílání, aby se na druhé straně kabelu použil pro příjem.

1.3 Struktura ethernetového rámce

Struktura ethernetového rámce je podrobně popsána v [8] sekce 5. Ethernetový rámec je posloupnost bytů neboli oktetů a každá datová část rámce obsahuje celočíselný počet bytů. Rámec začíná 7-bytovou preambulí, kde se střídají nuly a jedničky, poté následuje SFD(Start Frame Delimiter) byte, který odděluje preambuli od začátku rámec. Můžeme říci, že se jedná o Start byte. Po něm následují 6 bytové fyzické adresy cíle a zdroje, poté 2 bytová položka délka/typ, která je podle její hodnoty buď délka, pak její hodnota udává přímo velikost paketu nebo typ kde velikost paketu dáno podle nějakého protokolu obsahující uvnitř datového pole. Toto pole totiž následuje hned za touto položkou délka/typ a je maximálně 1500 bytů velké. Naopak pokud by bylo příliš malé, tak data musíme vyplnit tzv. výplní, která nám zajistí minimální velikost paketu a to 64 bytů. Nakonec je odeslán kontrolní 4 bytový kontrolní součet (CRC – Cyclic Redundancy Check), který potvrdí, že data jsou odeslána správně. Vše si podrobně popíšeme v další části.



Obrázek 1.1: Struktura ethernetového rámce

1.3.1 Preambule

Preambule slouží k synchronizaci obvodů a značí zahájení vysílání ethernetového rámce. Jeho velikost je 7 bytů a obsahuje následující sekvenci: 10101010 10101010 10101010 10101010 10101010 10101010 101010.

1.3.2 SFD

SFD je pole střídajících se jedniček a nul o velikosti jednoho bytu, avšak zakončený dvěma jedničkami. SFD je tedy následují sekvence bitů: 10101011. Toto pole nám tedy označuje, že po tomto bytu již budou vysílána vlastní data.

1.3.3 MAC adresa cíle

V tomto poli vysíláme 6 bytovou cílovou MAC adresu, unikátní adresa přiřazená zařízením.
Formát zápisu je většinou psán hexadecimálně jako MM : MM : MM : SS : SS : SS. Prvních 24 bitů MAC adresy obsahuje přidělené číslo výrobce síťových zařízení. Například, pokud má síťové zařízení MAC adresu s prefixem 00:A0:C9, tak toto zařízení indikuje, že se jedná o zařízení firmy Intel. Druhých 24 bitů MAC adresy, pak už reprezentuje unikátní sériové číslo zařízení dané společnosti. Pokud chceme vysílat rámec pro všechny zařízení (broadcast) nebo pro určitou skupinu, pak musí být první bit nastaven na jedničku. V případě broadcastu MAC adresu cíle tvoří samé jedničky.

1.3.4 MAC adresa zdroje

Pole obsahuje 6 bytovou MAC adresu zdroje a platí pro ni stejná pravidla jako pro MAC adresu cíle, až na výjimku, že ji nemůžeme nastavit pro skupinu případně broadcast. Musí být individuální a první bit adresy má vždy hodnotu 0.

1.3.5 Velikost/Typ

Dvou bytové pole, které může nabývat dvou různých významů. V jeho závislosti na jeho hodnotě toto pole signalizuje buď velikost nebo typ paketu.

- V případě, že se jedná o velikost menší nebo rovnou dekadické hodnotě 1500, tak tato hodnota signalizuje velikost dat, která budou poslána po odeslání tohoto pole. Minimální velikost dat můžeme být až i 0, avšak musí po datovém poli odvysílat, tzv. výplň, která splní minimální velikost odesílaného paketu a to 64 bytů. Data budou ignorována. Minimální velikost paketu je dána z důvodu správného fungování detekcí kolizí po ethernetové síti.

- Je-li hodnota větší nebo rovna dekadickému číslu 1536, což odpovídá 0x600 hexadeximálně, tak následující datové pole, které může být též označeno jako SDU (Service Data Unit) Linkové vrstvy nebo také PDU (Protocol Data Unit) vyšší vrstvy síťové viz obrázek 1.2, kde je zobrazeno zapouzdření TCP/IP paketu ve 4 vrstvách referenčního modelu OSI. Chceme-li odeslat paket pomocí IP protokolu verze 4, tak musí mít toto 2 bytové pole hodnotu 0x0800 hexadecimálně. Pro IP protokol verze 6 je hodnota pole 0x86DD hexadecimálně.

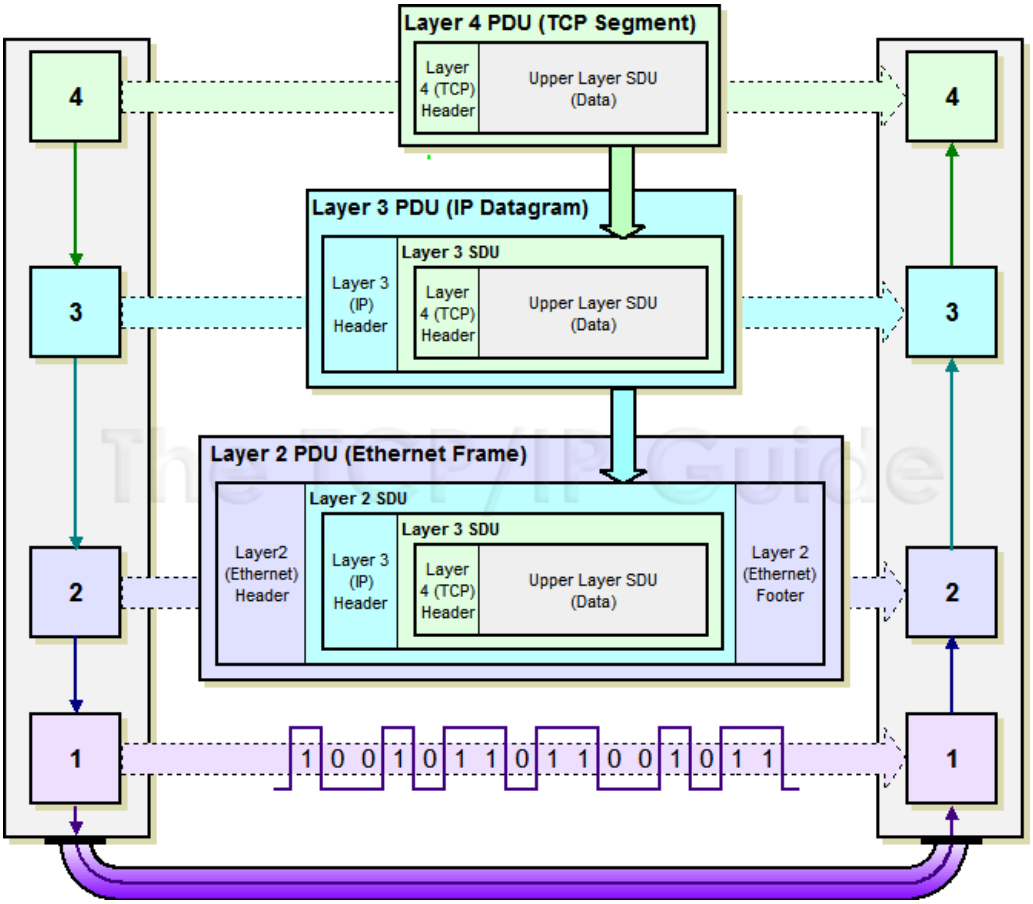
- Funkcionalita pro hodnoty v intervalu (1500,1536) není definována a proto je tento paket považován za chybný a měl by ho MAC v Linkové vrstvě vyřadit.

1.3.6 Data

Pole Data již obsahuje vlastní data, která chceme odeslat. Můžeme si sem nadefinovat vlastní protokol a zajistit přenos dat do druhého zařízení, který bude našemu protokolu rozumět. Pro komunikaci celosvětově dominuje protokol s názvem IP. Na obrázku 1.2 můžeme vidět, jak jsou data zapouzdřena z vyšších vrstev do našeho cílového pole, které odešleme v ethernetovém rámci. Transportní vrstva 4. v modelu OSI vezme data, které chce poslat a přiřadí jí TCP hlavičku, v které je mimo jiné specifikovaný port. Port určuje, pro jakou aplikaci jsou data určena. Data včetně hlavičky můžeme nazvat jako PDU 4.vrstvy. Data jsou přijata 3.vrstvou OSI modelu, které se nazývá Síťová vrstva. Síťová vrstva přijme data, které můžeme nazvat SDU 3.vrstvy, a připojí k ní hlavičku, v níž je především specifikována cílová a zdrojová IP adresa. Vznikne nám tak PDU 3.vrstvy. Síťová vrstva nám zajišťuje odeslání dat i do zařízení jejichž cílovou MAC adresu neznáme a nenalézá se v naší lokální síti. Síťová vrstva opět pošle data na nižší vrstvu, která se nazývá Linková vrstva, na které pracuje zařízení MAC. Ta v konečné fázi přiřadí k SDU 2. vrstvy cílovou a zdrojovou MAC adresu, typ/velikost a CRC kontrolním součtem.

1.3.7 CRC kontrolní součet

Pro ověření správnosti přijatých dat slouží právě 4 bytové pole. Správnost dat je ověřena ze všech odeslaných dat až na preambuli a SFD. Vstupní



Obrázek 1.2: Zapouzdření dat do ethernetového rámce, zdroj [10]

data pro CRC algoritmus jsou tedy počítána z cílové a zdrojové MAC adresy, typ/velikost a vlastních dat. Způsob výpočtu spočítává v postupném použití exkluzivního součtu XOR, což lze snadno provést na Galoisově typu posuvného registru s xor hradly umístěnými dle mocnin viz C. Stačí jen znát generující polynom, kterým budeme data a jeho zbytky dělit až do výsledného 32-bitového kontrolního součtu. Pro Ethernet má 32 bitový generující polynom hodnotu:

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

Binárně pak tento polynom zapíšeme:

$$G = 0000\ 0100\ 1100\ 0001\ 0001\ 1101\ 1011\ 0111_{(2)} \tag{1.1}$$

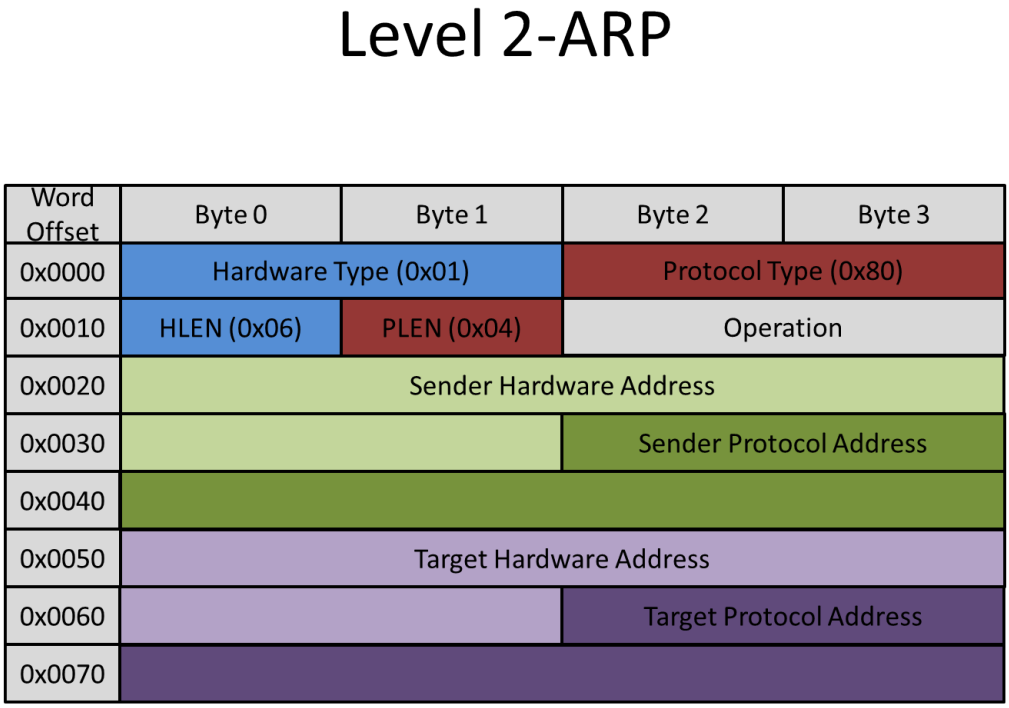
Pro výpočet kontrolního součtu MAC nepracuje s celými daty najednou. To by bylo příliš náročné, potřebovali bychom velké posuvné registry a výpočet

by trval několik hodinových taktů, než bychom se dopracovali k výslednému kontrolnímu součtu . Výpočet v MAC tedy probíhá tak, že postupně posíláme data do PHY čipu a během odesílání si je odchytí blok, který pomocí hradlové logiky aktualizuje 4 bytové CRC. MAC poté přiřadí výsledný vypočítaný kontrolní součet na konec rámce. To je také důvod, proč se odesílá jako poslední pole v ethernetovém rámci.

1.4 Základní internetové protokoly

Projekt je možné rozšířit pro komunikaci pomocí IP protokolů se zařízeními na internetu. Prakticky můžeme projekt rozšířit na komunikaci mezi dvě FPGA deskami připojených k Internetu z druhého konce světa. K tomu budeme určitě muset znát internetové protokoly. Popíšeme si dva základní IP a ARP, které jsou podrobně popsány v [1].

1.4.1 ARP



Obrázek 1.3: Hlavička protokolu ARP, zdroj [7]

Jeden ze základních IP protokolů, který slouží ke zjišťování MAC adresy z IP adresy se nazývá ARP (Address Resolution Protocol). Tento protokol slouží ke zjištění MAC adresy z již známe IP adresy. Každé síťové zařízení,

které chce komunikovat, musí MAC adresu znát. Tuto adresu získává z ARP tabulky. Je to jednoduchý seznam MAC adres a k nim přidělené IP adresy. Chceme-li získat MAC adresu cílového zařízení, odešleme ARP dotaz, který specifikujeme hodnotou 0x0806 v poli protokol a v její hlavičce typem operace na hodnotu 0x01. ARP hlavičku si můžeme prohlédnou na obrázku 1.3. V poli Hardware Type definujeme typ hardwaru a hodnota 0x0001 nám říká, že se jedná o Ethernet. Polem Protocol Type specifikujeme, o jaký typ protokolové adresy se jedná. Nejčastěji se tedy jedná o IPv4, kterou definujeme hodnotou 0x0800. Následují bytové položky pro délku hardwarové (MAC) adresy a délku protokolové adresy v bytech. Následuje již zmíněná položka s typem operace, která je buď 0x01 (dotaz) nebo 0x02 (odpověď). Následují pole pro MAC a IP adresu odesílatele a po ní MAC a IP adresu cíle. Doplníme hodnoty, které známe až na cílovou MAC adresu, kterou chceme zjistit k dané IP adrese. Místo cílové MAC adresy vložíme samé nuly. Paket odešleme v ethernetové hlavičce s MAC adresou pro broadcast (FF:FF:FF:FF:FF) a čekáme na odpověď od daného počítače v síti. Počítač na dané adrese odpoví ARP odpovědí. Téměř všechna pole zůstanou stejná až na typ operace, která se změní na 0x02 (odpověď) a přehozením cílových a zdrojových adres s již vyplněnou MAC adresou, kterou jsme chtěli najít. Po přijetí odpovědi na ARP dotaz si aktualizujeme tabulku s cílovou adresou. Zařízení nyní může komunikovat bez dalšího dotazování.

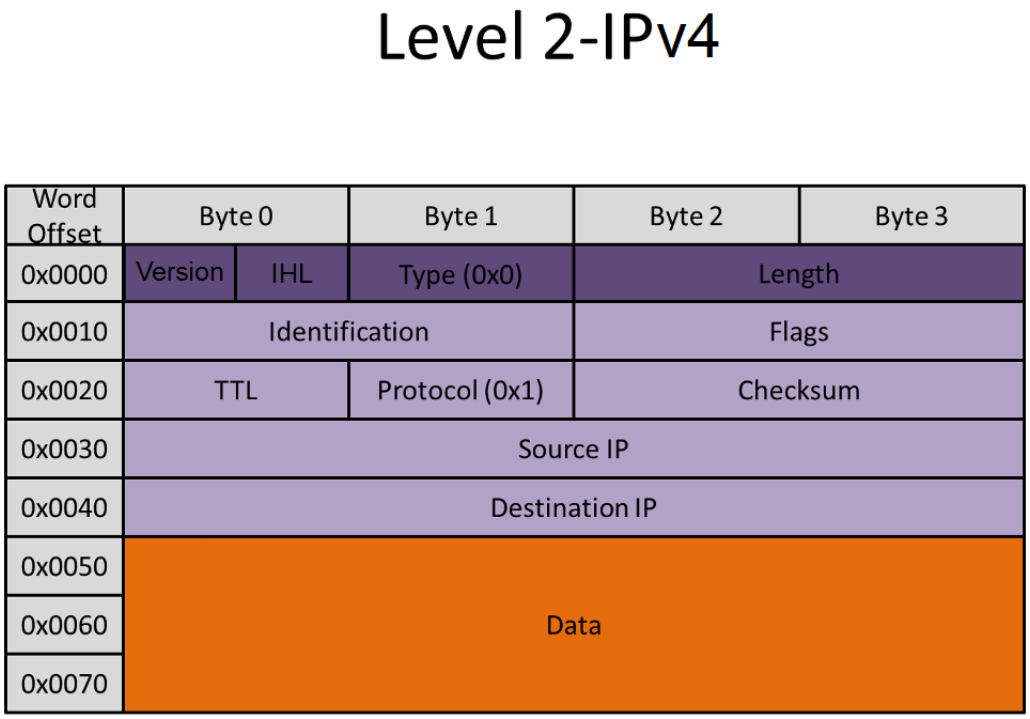
1.4.2 IPv4

Internet Protocol verze 4 patří k nejpoužívanějším paketům pro ethernetovou komunikaci. Tento protokol specifikujeme v typu protokolu hodnotou 0x0800. Tento protokol nezaručuje přenos dat se zárukou správného doručení paketu. Pro záruku doručení se tento paket kombinuje s protokolem vyšší vrstvy TCP, který tento nespolehlivý paket odesílá tak dlouho, dokud není správně přijat. Adresování síťových zařízení je pomocí 32-bitové adresy, která dokáže adresovat 2^{32} ($\approx$ 4 miliardy) adres. Nyní jsou již všechny IPv4 adresy vyčerpány a komunikace se převádí na nový protokol IPv6, kterým se v naší práci zabývat nebudeme. Na obrázku 1.4 můžeme vidět strukturu protokolu IPv4. Opět jako pro ARP protokol si jí popíšeme.

- První 4 bitové pole nám definuje verzi protokolu. Existuje více verzí, ale v tomto protokolu se uchytila pouze verze 4. Pole bude mít hodnotu 0x4.

- Pole IHL (Internet Header Lenghth) nám definuje velikost IP hlavičky ve 4 bytech. Nejčastěji je uváděna standardní hodnota a zároveň minimální hodnota velikosti hlavičky, a to je 0x5, tedy 20 bytová hlavička.

- Pole Type mělo podle původních představ signalizovat charakter paketu a směrovače podle tohoto pole měli rozhodovat, jakou cestou budou pakety odeslány. V našem případě do pole Type zapíšeme samé nuly.



Obrázek 1.4: Hlavička protokolu IPv4, [7]

- V poli Length je zapsána celková délka paketu včetně hlavičky v bytech. Minimální hodnota je tedy 20.

- Hodnota v poli Identification nám říká, o jaký paket se jedná, dojde-li k fragmentaci paketu. Pakety se stejnou hodnotou pole Identification patří k sobě a jsou v cílovém pořadí opět složeny dohromady.

- V poli Flags nastavujeme příznaky, může-li být paket fragmentován nebo zda-li je poslední. Obsahuje též fragment offset, kterým určíme, v jaké části v originálním nerozděleným paketu data začínají.

- Pole TTL (Time To Live) nám říká, kolika routery může datový paket projít. Toto pole je velmi důležité a chrání routery, aby nedocházelo k zacyklení. Nebýt tohoto pole, mohlo by se stát, že by si dva routery přehazovaly ten samý paket stále dokola. Tato položka je s každým průchodem routeru dekrementovaná o jedničku. V případě, že je tato položka nulová, paket je nenávratně zahozen.

- Další důležitým polem je bytové pole Protocol. Všechna tato správa nám probíhá v 3.vrstvě OSI a to Síťové vrstvě a hodnota této položky nám

určuje, jaký další protokol z vyšší Transportní vrstvy je vnořen do položky Data. Nejpoužívanější protokoly jsou rozhodně TCP (Transmission Control Protocol) pro hodnotu 0x06 a UDP (User Datagram Protocol) pro hodnotu 0x11. Následně položka Checksum, neboli kontrolní součet, která nám ověří správnost dat. Konec hlavičky obsahuje zdrojovou a cílovou 4 bytovou IP adresu.

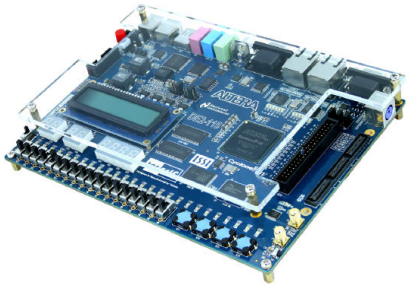
– Datové pole v Síťové vrstvě obsahuje data z vyšší vrstvy. Pro spolehlivý přenos dat používáme protokol TCP/IP. Pro nespolehlivý přenos, avšak rychlejší, používáme přenos pomocí protokolu UDP. TCP/IP nám zajišťuje spolehlivost pomocí opakovaného odesílání nespolehlivých IP paketů. Pakety jsou rozděleny do maximální velikosti ethernetového rámce, která je dána parametrem MTU (Maximum Transmission Unit) na MAC, což odpovídá hodnotě 1500 bytů. Segmenty jsou očíslovány, neboť ne každý segment nemusí přijít ve stejném pořadí, v jaké byl odeslán. Díky očíslování si příjemce může data setřídit ve správném pořadí. Jak TCP, tak UDP pakety obsahují pole pro zdrojový a cílový port, specifikující pro jakou vyšší Aplikační vrstvu jsou data určena. Tím rozlišíme data pro internet (HTTP), email (SMTP,IMAP) aj.

Kapitola 2

Analýza

2.1 FPGA Altera deska DE2-115

Nyní bych vás seznámil s FPGA deskou DE2-115 firmy Altera, pro kterou jsme v této bakalářské práci navrhli MAC (Media Access Control) a další potřebné komponenty ve vyšších vrstvách pro správnou komunikaci po ethernetové síti.



Obrázek 2.1: Vývojová deska Altera DE2-115 deska , [4]

Deska obsahuje:

- Hradlové pole Cyclone EP4CE115
  - 114 480 logických buněk
  - 528 uživatelských I/O pinů
  - 266 zabudovaných 18 bit x 18 bit násobiček
- Paměť
  - 128 MB (32 milion x 32bit) SDRAM
  - 2 MB (1 milion x 16 bit) SRAM
- 18 přepínačů
- 4 tlačítka
- 8 sedmi- segmentových displejů
- PS/2 port pro připojení klávesnice a myši
- SD slot

- 2 10/100/1000 integrovaný ethernet
- 2 USB porty typu B pro konfiguraci FPGA
- 1 USB port typu A 2.0
- RS 232 port
- VGA port pro připojení monitoru
- Infračerven
- a další

Budeme potřebovat znát funkci některých periferií na desce, které jsou v projektu použity. Popíšeme je v následujících odstavcích.

2.1.1 10/100/1000 ethernetový čip

DE2-115 deska poskytuje pro Ethernetovou komunikaci dva PHY čipy Marvell 88E1111, které poskytují rychlosti 10/100/1000 Mbits/s pro rozhraní

- MII (Media Independent Interface)
- GMII (Gigabit MII)
- RGMII (Reduced GMII)
- TBI (Ten-Bit Interface)

Komunikace FPGA desky pouze poskytuje rozhraní MII a RGMII. Ostatní rozhraní nejsou na této desce možné.
Výběr, jakým rozhraním chceme komunikovat, nastavíme pomocí propojovacích přepínačů (jumperů) na desce. FPGA deska rovněž nastavuje ostatní parametry čipu. Jelikož MAC může komunikovat až s 32 PHY čipy, tak každý musí mít svoji vlastní adresu, která je uložena v registru PHYADDR[4..0]. Defaultní hodnota adresy pro ETHERNET0 je nastavena na 1000, pro ETHERNET1 je 10001. Pro nás další důležitá hodnota, kterou bychom v tomto projektu měli znát, je hodnota registru ANEG[3..0], která nám zajišťuje autonegatiation po kroucené dvojlince.
Metoda autonegatiation nám zajišťuje, že se bude komunikovat maximální možnou rychlostí, kterou oboje zařízení podporují. Kompletní nastavení můžeme nalézt v datasheetu FPGA desky. V tabulce 2.1 můžeme vidět nastavení pro desku DE2-115. Data jsem použil z datasheetu desky [3].

Konfigurace desky nastavujeme pomocí 7-bitové sběrnice CONFIG[0..6], která je vyvedena na ethernetovém čipu. Tyto piny však nejsou připojené k obvodu

Configuration	Description	Default Value
Default Value	PHY Address in MDIO/MDC Mode	10000 for Enet0;10001 for Enet1
ENA_PAUSE	Enable Pause	1-Default Register 4.11:10 to 11
ANEG[3:0]	Auto negotiation configuration for copper mode	1110-Auto-neg, advertise all capabilities, prefer master
ENA_XC	Enable Crossover	0-Disable
DIS_125	Disable 125MHz clock	1-Disable 125CLK
HWCFG[3:0]	Hardware Configuration Mode	1011/1111 RGMII to copper/GMII to copper
DIS_FC	Disable fiber/copper interface	1-Disable
DIS_SLEEP	Energy detect	1-Disable energy detect
SEL_TWSI	Interface select	0-Select MDC/MDIO interface
INT_POL	Interface select	1-INTn signal is active LOW
75/50OHM	Termination resistance	0-50 ohm termination for fiber

Tabulka 2.1: Defaulní Konfigurace pro 88E1111, zdroj [3]

FPGA desky DE2-115, a proto není možné s nimi pracovat, výjimkou režimu, v kterém budeme data posílat (RGMII/MII). V mém projektu budeme pracovat v režimu MII, a proto patřičně nastavím jumpery pro oba ethernetové čipy.

2.1.2 Tlačítka a přepínače

Vstupními periferiemi jsou především dvoupolohová tlačítka a přepínače. Přepínač ve své horní poloze přivede napětí 2.5V na FPGA pin, tedy logické 1. Při poloze dolní je vstupní hodnota zem, to odpovídá logické 0. U tlačítek je třeba dát si pozor, že mají opačnou logiku. Při nesepnutém stavu generují logickou hodnotu 1. Při stisknutí tlačítka pak logickou 0.
Tlačítka i přepínače jsou asynchronní. Navrhujeme-li synchronní obvod, je vhodné ošetřit synchronizací, aby nemohlo dojít k případné metastabilitě. To můžeme například ošetřit zapojením dvou D-klopných obvodů v sérii za tlačítkem.

2.1.3 LED

LED (anglicky Light-Emitting Diode) patří k nejjednodušším výstupním periferiím. Jsou připojeny anodou na výstupní port, který generuje 2.5 V pro logickou 1 a 0 V pro logickou 2. Katoda je připojena na ochranný odpor, který je připojen na zem. Připojený odpor omezuje procházející proud.

2.1.4 7-Segmentový displej

Další možností zobrazení výstupních dat nám dává 7-segmentový displej. Displeji je přirazen 7 bitový vektor určité hodnoty, kde každý bit vektoru odpovídá jednomu segmentu displeje. Jednotlivý segment svítí při přiřazení logické 0 na jeho vstupní pin.

2.2 Media Independent Interface

MII je standardní ethernetové rozhraní [8] mezi PHY a MAC pro Fast Ethernet (100 Mbit/s). Rozhraní je definováno, protože můžeme chtít z nějakého důvodu mít PHY čip mimo MAC, proto byl zaveden standard IEEE 802.3u, který definuje MII sběrnici. Všechny tyto piny sloužící pro komunikaci přes MII s čipem jsou vyvedeny na FPGA desku. S tímto rozhraním právě budeme pracovat a snažit se komunikovat s PHY čipem. Vysvětlení propojení si můžeme prohlédnout na obrázku 2.2, kde mimo jiné je zobrazeno i komunikační rozhraní MIIM, které slouží ke komunikaci s vnitřními registry PHY. Registry budou popsány později.

Vysvětlení jednotlivých pinů v rozhraní si můžeme prohlédnout v tabulce 2 nebo přímo ve zdrojovém dokumentu IEEE 802.3 v sekci 2 [8]:

Na FPGA desce máme vyvedené i další piny, které nejsou součástí rozhraní MII a MIIM. Jsou to signály RST_N, INT_N, LINK100.

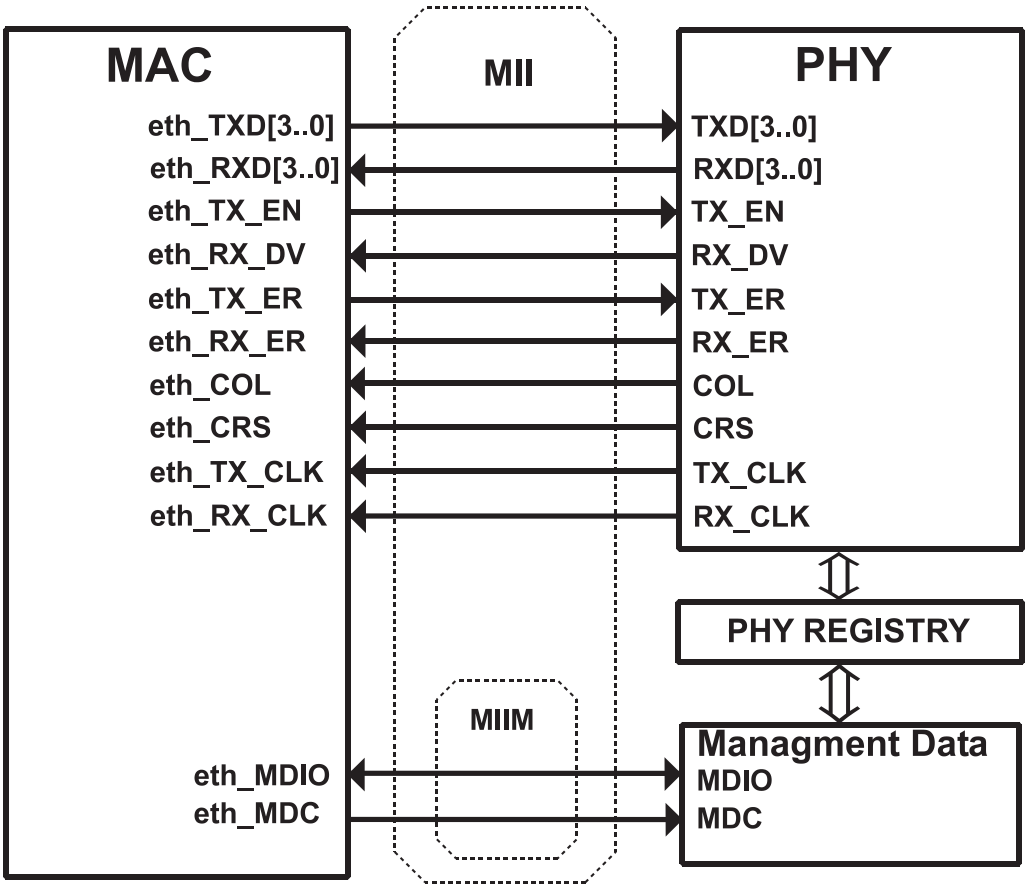
- Vstupní pin RST_N je hardwarový reset PHY čipu, který musí být aktivovaný minimálně na 10 hodinových cyklů krystalu připojeného na XTAL1. Tento pin je aktivní v 0.
- Výstupní INT_N je příznak, který je nastaven, pokud je v PHY vyvolána chyba, která se zapíše do registru 19. Zakázání vyvolávání chyb můžeme zajistit pomocí zakázání bitu interrupt enable v registru 18.
- Výstupní LINK100 nám signalizuje, zda je v danou chvíli povolen 100 Mbit/s ethernetový přenos.

2.3 MDIO komunikace a PHY registry

Pro čtení a zápis do registrů na PHY využíváme jednobitovou obousměrnou sběrnici MDIO, která je synchronně řízena pomocí hodin MDC. V operaci potřebujeme znát PHY adresu a adresu registru, z kterého chceme číst/zapisovat. PHY výrobce uvádí ve svém datasheetu popis jednotlivých registrů a jejich význam. Ne každý bit v registru musí něco znamenat a ne vždy je možné do něj zapisovat, avšak podle normy IEEE 802.3 musí obsahovat standardní registry, které spravují ethernetovou komunikaci. Data jsou synchronně odesílána s hodinami MDC s tím, že data z MAC do PHY jsou čtena náběžnou

Signál	Směr	Význam
TXD [3..0]	Input	4 bitová sběrnice, po které jsou s každou náběžnou hranou hodin CLK_TX načtena data k odeslání čipem. Bit TXD0 je odeslán jako první TXD3 jako poslední. Synchronní s CLK_TX.
RXD[3..0]	Output	4 bitová sběrnice, na kterou jsou s každou náběžnou hranou hodin CLK_RX načtena data z čipu. Bit RXD0 je přijat jako první RXD3 jako poslední. Synchronní s CLK_RX.
TX_EN	Input	Výstupní signál, který je nastaven do logické 1 po celou dobu kdy probíhá odesílání paketu. Tedy od preambule až po CRC. Synchronní s CLK_TX.
RX_DV	Output	Vstupní signál, který nám signalizuje v logické 1,že přichází ethernetový rámec. RX_DV je nastaven po celý čas ethernetového rámce. Synchronní s CLK_RX.
TX_ER	Input	Signál je nastaven do logické 1, pokud nastane chyba při vysílání. Je nastavováno velmi zřídka. Synchronní s CLK_TX.
RX_ER	Output	Signál je nastaven do logické 1, pokud nastane chyba při příjmu. Je nastavováno velmi zřídka. Synchronní s CLK_RX.
COL	DOutput	Signál, který nám signalizuje, že došlo ke kolizi s jiným paketem na sdíleném médiu. Tento signál bude vždy nulový, komunikujeme-li v duplexním režimu.
CRS	Output	Signál je nastaven, pracujeme-li v poloduplexním režimu a PHY zrovna přijímá nebo vysílá ethernetový rámec.
TX_CLK	Output	Hodiny vysílače, které poskytují frekvenci 25MHz v 100Base-TX a 2.5 MHz v 10Base-T.
RX_CLK	Output	Hodiny přijímače, které poskytují frekvenci 25MHz v 100Base-TX a 2.5 MHz v 10Base-T.
MDIO	Bidirectional	Obousměrná jednobitová sběrnice, která složí pro správu PHY. Pro správné fungování potřebuje tento pin pull-up rezistor v rozsahu 1.5 kohm až 10 kohm. Synchronní s MDC.
MDC	Input	Hodiny pro komunikaci se správou řídících dat v PHY. Maximální frekvence je udávána výrobcem. Pro PHY 88e1111 je maximální frekvence 8.3 Mhz.

Tabulka 2.2: Význam jednotlivých pinů v MII rozhraní



Obrázek 2.2: MII rozhraní mezi PHY a MAC

32-bitová preambule	Začátek rámce	Čtení=10 Zápis=01	5-bit PHY adresa	5-bit adresa registru	TA Čtení=z0 Zápis=10	16-bit datové pole	Nečinnost zařízení
111..1111	01	10	10001	00000	10	11000..0	11..111

Tabulka 2.3: Příklad MDIO komunikace s PHY čipem

hranou hodin MDC a data z PHY do MAC jsou čtena sestupnou hranou MDC. Master je v této komunikaci vždy MAC a Slave je PHY.

2.3.1 Registry

V PHY se nachází 32 šestnáctibitových registrů, s kterými můžeme komunikovat pouze přes MDIO. Prvních 16 registrů jsou však přesně definovaná normou IEEE 802.3, zbylých 16 registrů si již výrobce definuje sám. V ether-

netové komunikaci jsou nejdůležitější první dva registry na adrese 00000 a 00001. Registr na adrese 0 nazýváme Control registr, tím můžeme nastavit software reset čipu, ethernetovou rychlost přenosu nebo plně duplexní režim. Schéma dat je zobrazené v tabulce 4. Druhý registr na adrese 1 je nazývá Status registr a zobrazuje nám stavy, v jakých se nachází PHY. Zjistíme z něj, zda zapsaná data do Control registru nabyla platnosti, které se zobrazí v Status registru.

Pro komunikaci v 100 Mbit/s plně duplexním režimu je potřeba do Control registru zapsat hodnotu 1010 0001 0000 0000 = 0xA100.

Bit(s)	Name	Description	R/W [a]
0.15	Reset	1=PHY reset 0=normal operation	R/W SC
0.14	Loopback	1 = enable loopback mode 0 = disable loopback mode	R/W
0.13	Speed Selection (LSB)	0.6 0.13 11 = Reserved 10 = 1000 Mb/s 01 = 100 Mb/s 00 = 10 Mb/s	R/W
0.12	Auto-Negotiation Enable	1 = enable Auto-Negotiation process 0 = disable Auto-Negotiation process	R/W
0.11	Power Down	1 = power down 0 = normal operation [b]	R/W
0.10	Isolate	1 = electrically Isolate PHY from MII or GMII 0 = normal operation [b]	R/W
0.9	Restart Auto-Negotiation	1 = restart Auto-Negotiation process 0 = normal operation	R/W SC
0.8	Duplex Mode	1 = full duplex 0 = half duplex	R/W
0.7	Collision Test	1 = enable COL signal test 0 = disable COL signal test	R/W
0.6	Speed Selection (MSB)	11 = Reserved 10 = 1000 Mb/s 01 = 100 Mb/s 00 = 10 Mb/s	R/W
0.5	Unidirectional enable	When bit 0.12 is one or bit 0.8 is zero, this bit is ignored.When bit 0.12 is zero and bit 0.8 is one: 1 = Enable transmit from media independent interface regardless of whether the PHY has determined that a valid link has been established 0 = Enable transmit from media independent interface only when the PHY has determined that a valid link has been established	R/W
0.4:0	Reserved	Write as 0, ignore on read	R/W

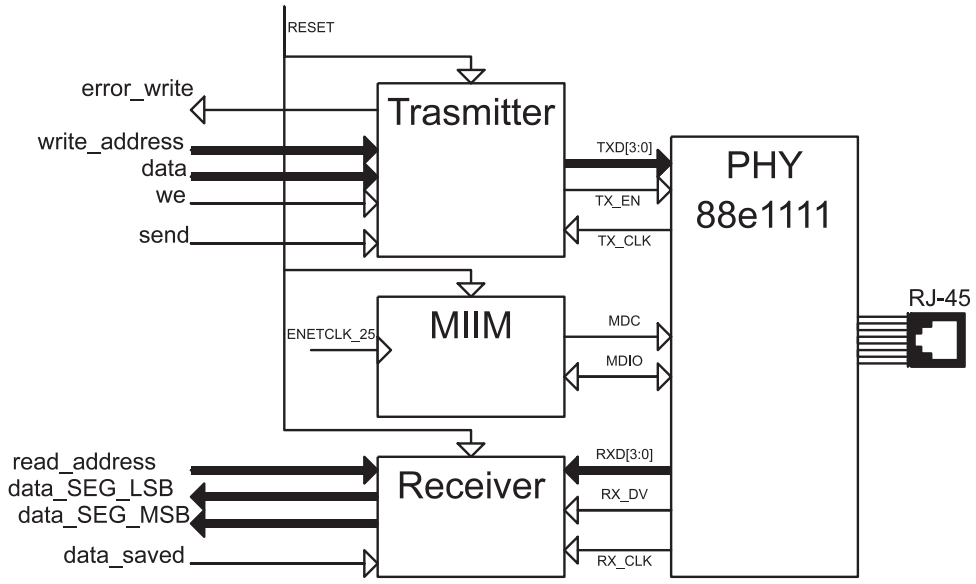
Tabulka 2.4: Control register bit definitions, zdroj [8] sekce 2

[a] R/W = Read/Write, SC =Self-clearing
[b] For normal operation, both 0.10 and 0.11 must be cleared to zero.

Kapitola 3

Řešení projektu

Projekt je rozdělen na 3 hlavní na sebe nezávislé bloky. Jednotlivé bloky jsou Ethernetový vysílač (anglicky Trasmitter), Ethernetový přijímač (anglicky Receiver) a Sériové rozhraní pro správu PHY (MIIM) . Propojení navržených bloků je definováno v top modulu Eth 3.1.



Obrázek 3.1: Blokové schéma projektu

3.1 Top modul

Top modul nebo top-level entita propojí námi navržené bloky do jednoho na vývody obvodu FPGA na desce DE2-115. Top level entita nám přiřadí následující piny:

- SW(17) – reset obvodu, po kterém mimo jiné proběhne zápis dat do ethernetových čipů.
- SW(16) – hardware reset ethernetových čipů ENET0 a ENET1.
- SW(15) – signál data_saved, který v případě, že je roven 0, nedovolí uložení dalšího ethernetového rámce. Nastavíme-li tento signál do logické 1, přijímané rámce se budou ukládat a vzájemně přepisovat.
- KEY(0) – tlačítko je připojené na negovaný signál send. Signál je aktivován v případě stisknutí tlačítka. Tlačítkem vyšleme signál pro odeslání rámce. Rámce jsou dokola generovány, dokud signál send není v logické 0.
- ENET0/1_TX_CLK, ENET0/1_TX_CLK, ENETCLK_25 – jedná se o hodiny pro příslušné bloky. I když je velice pravděpodobné, že se jedná o jedny a ty samé hodiny, může být mezi nimi časový posun. Pro komunikaci mezi jednotlivými bloky je nutné ověřit synchronizaci, aby případně nedošlo k metastabilitě obvodu

3.2 Ethernetový vysílač - Trasmitter

Vysílač je synchronně ovládán s náběžnou hranou hodin TX_CLK. Jeho funkce je odesílat pevně definovaný paket z SRAM pamětí TX_SRAM_HEADER a TX_SRAM_DATA, ke které nám TX_CRC32 počítá kontrolní 4 bytový součet, který přiřadí na konec celého rámce. Obrázek 3.2.

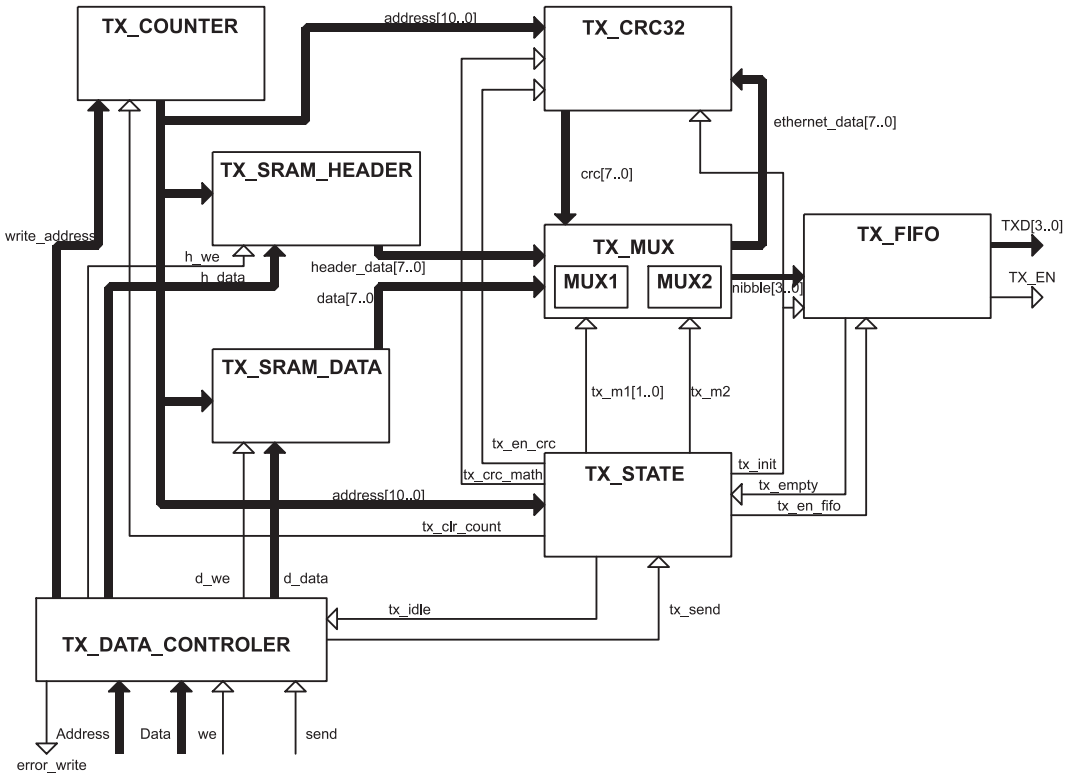
3.2.1 TX_COUNTER

11-bitový čítač s každým druhým taktem inkrementuje svoji hodnotu o 1. Důvod je ten, že s každým taktem jsou odeslána 4 bitová data, avšak paměti a crc pracují s 8 bitovou hodnotou. Vstupní signál tx_clr_count nastaví výstupní adresu čítače do 0, kde přetrvá dva hodinové takty.

Vstupní adresa pro nový zápis dat je nastavena na výstupní adresu pro paměti v případě, že tato vstupní adresa není nulová, pak značí adresu, kam chceme zapsat data. Chceme-li zapsat data na nulovou adresu, pak při zápise se musí stavový automat nacházet ve stavu idle (nutná podmínka pro zápis) a v tomto stavu je v každém taktu counter resetován, tím docílíme adresu pro zápis rovný nule. TX_DATA_CONTROLER již nastaví potřebná data i signál povolení zápisu pro daný blok. viz 3.2.3.

3.2.2 TX_SRAM_HEADER a TX_SRAM_DATA

Jedná se o registrové SRAM paměti, z kterých jsou posílána data. TX_SRAM_HEADER je 16 bytová paměť se vstupní 4 bitovou adresou (vyšší bity adresy jsou ignorovány). Obsahuje postupně od adresy 0, tyto položky:



Obrázek 3.2: Blokové schéma ethernetového vysílače TX_ETH

- Cílová MAC adresa (6 bytů)
- Zdrojová MAC adresa (6 bytů)
- Typ/Délka (2 byty)
- Zbylá nepoužívaná data (2 byty)

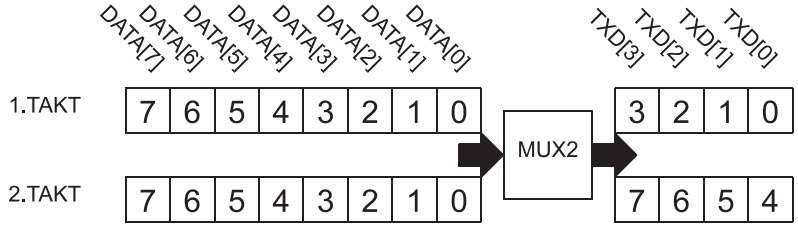
TX_SRAM_DATA je 2048 bytová paměť, v které jsou uložena data z vyšší vrstvy. Adresa 0 v datové paměti obsahuje byte, který je odeslán hned po odeslání pole Délka/Typ z hlavičkové paměti.

3.2.3 TX_DATA_CONTROLER

Tento blok řídí zápis dat do SRAM_HEADER a SRAM_DATA. Data jsou zapisována až po přijetí signálu tx_idle, že data nejsou odesílána a může být proveden zápis. Pokud se stavový automat nenachází v nečinném stavu, signál error_write se nastaví do jedničky a značí, že zrovna nemůže být proveden zápis. Je zde možnost rozšíření, když k němu přidáme FIFO paměť, která bude obsahovat pakety, které mají být odeslány.

3.2.4 TX_MUX

Obsahuje dva multiplexery typu data flow MUX1 a MUX2. MUX1 podle 2 bitového vstupního signálu tx_m1 rozhodne o výstupu ze tří různých vstupů. Jedná se o vstupy z TX_SRAM_HEADER, TX_SRAM_DATA a TX_CRC32. 8 bitový výstup přichází do MUX2 na dva hodinové cykly, kde je v prvním hodinovém cyklu odeslán nejprve dolní půl byte a poté horní půl byte. Toto přiřazení však neplatí pro ethernetovou preambuli a crc, kde jsou data do TXD přiřazena reverzně.



Obrázek 3.3: Správné formátování bytu pro MII

3.2.5 TX_CRC32

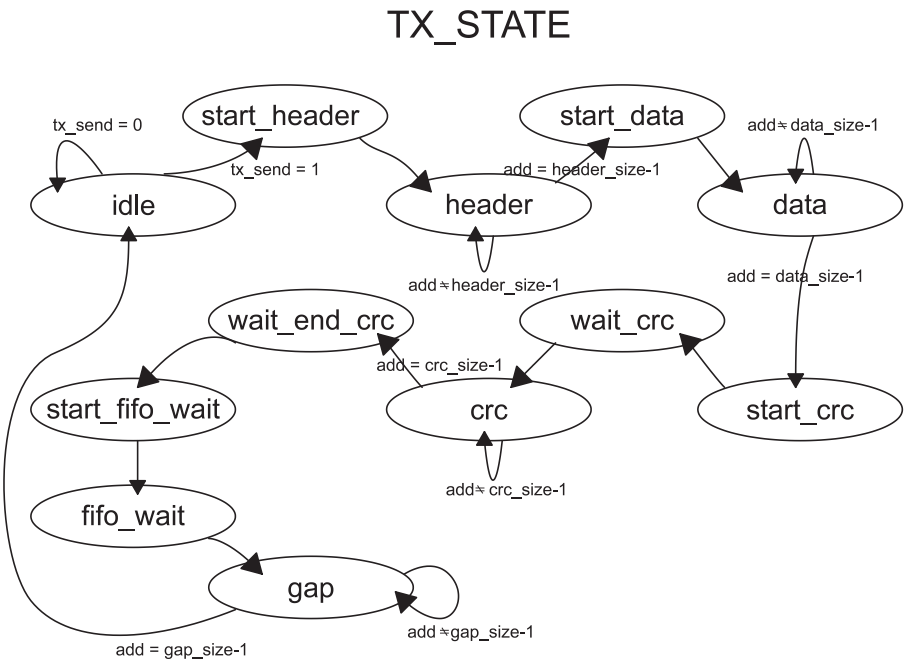
Je Galoisův posuvný registr (viz příloha A), který nám s přicházejícími daty z MUX1 počítá výsledný kontrolní součet 4 bytový součet.

3.2.6 TX_FIFO

Jedná se o 32 x 4 bitovou FIFO paměť (anglicky First In First Out) s. Vstupem jsou 4 bitová data z MUX2. Před samotným odesláním dat je vyslán signál init, který uloží do paměti 8 bytovou preambuli v reverzním tvaru, tedy: 55 55 55 55 55 55 55 5D hexadeximálně, která je později přiřazena výstupní sběrnici TXD(3..0). Po odeslání všech dat včetně crc, vyšle signál tx_empty, který nám říká, že všechna data byla odeslána.

3.2.7 TX_STATE

Je konečný automat typu Moore. Jedná se tedy o automat, kde změnou vstupního signálů vygeneruje výstupní signály až v dalším taktu. Pro každý stav automatu jsou dány výstupy, které jsou v daném stavu neměnné. Změna výstupních signálu nastane, změní-li se stav automatu vlivem vstupních signálů. Stavový diagram je definován na obrázku 3.4. Stavový automat se po resetu nachází ve stavu idle. Signál tx_send signalizuje stisknutí tlačítka pro odesílání dat. Hodnoty header_size, data_size,crc_size, gap_size jsou pevně nastavené ve zdrojovém kódu a udávají velikost jednotlivých částí paketu. Pro



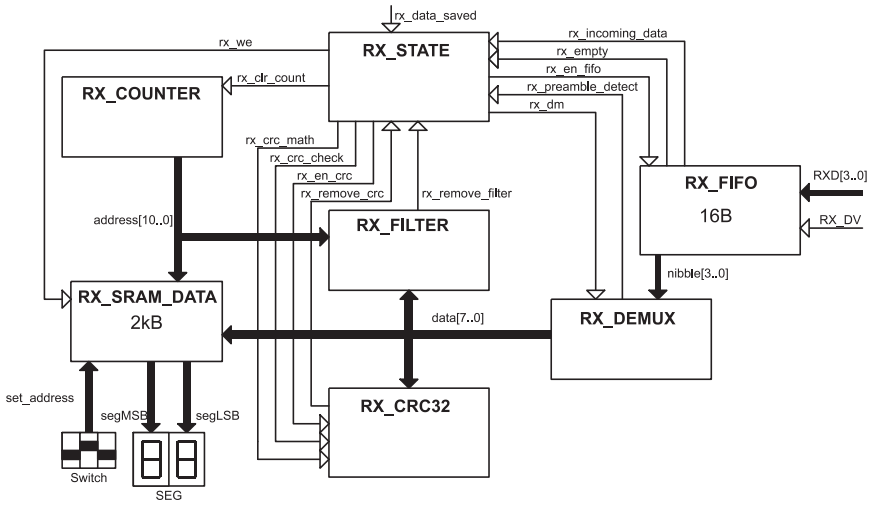
Obrázek 3.4: Stavový diagram konečného automatu TX_STATE

správně fungování je třeba nechat všechny tak, jak jsou nastavené. Výjimku tvoří data_size, které můžeme libovolně nastavit na velikost odesílaných dat.

Výstupní signály jednotlivých stavů můžeme dohledat ve zdrojovém kódu projektu.

3.3 Ethernetový přijímač – Receiver

Ethernetový přijímač je synchronně ovládán na náběžnou hranu hodinami RX_CLK. Ukládá datový rámec, při kterém vyhodnocuje, zda má být rámec uložen nebo být zahozen. Existují dva důvody zahození paketu. Buď neodpovídá kontrolní součet přijímaným datům nebo blok RX_FILTER vyhodnotí nevyhovující data (například pokud MAC adresa se neshoduje). V případě uložení dat, čeká na signál, že paměť může být opět přepisována. Obrázek 3.5.



Obrázek 3.5: Blokové schéma ethernetového přijímače RX_ETH

3.3.1 RX_COUNTER

11-bitový čítač, který drží výstupní adresu na dobu 2 taktů, poté je adresa o jedničku inkrementována. viz TX_COUNTER 3.2.1.

3.3.2 RX_SRAM_DATA

2kb x 8 registrová SRAM paměť, která ukládá přijatá data do své paměti, je-li signál z tx_state we = 1 od adresy 0. Ethernetový paket je ukládán od pole obsahující cílovou MAC adresu. Data z paměti jsou pak zobrazována na 2 segmentové displeje, které jsou na adrese definované binární hodnotou přepínačů SW(10..0).

3.3.3 RX_FILTER

RX_FILTER kontroluje data a porovnává s daty, které jsou v něm obsažené. Tímto blokem například můžeme zajistit, abychom přijali rámec, který obsahuje pouze naší MAC adresu.

3.3.4 RX_DEMUX

Vstupní hodnotou bloku je 4-bitová sběrnice, která odpovídá vstupním přijatým datům. První data obsahují ethernetovou preambuli, proto blok vyšle signál, že preambuli detekoval (rx_preamble_detect = 1). Následně čeká v nečinnosti do doby, než přijatá data odpovídají hodnotě SFD, tedy 0xD hexadecimálně. Po detekování začátku rámce zruší signál rx_preamble_detect a začne formátovat data do velikosti jednoho bytu, které jsou odesílány s každou druhou náběžnou hranou hodin.

3.3.5 RX_FIFO

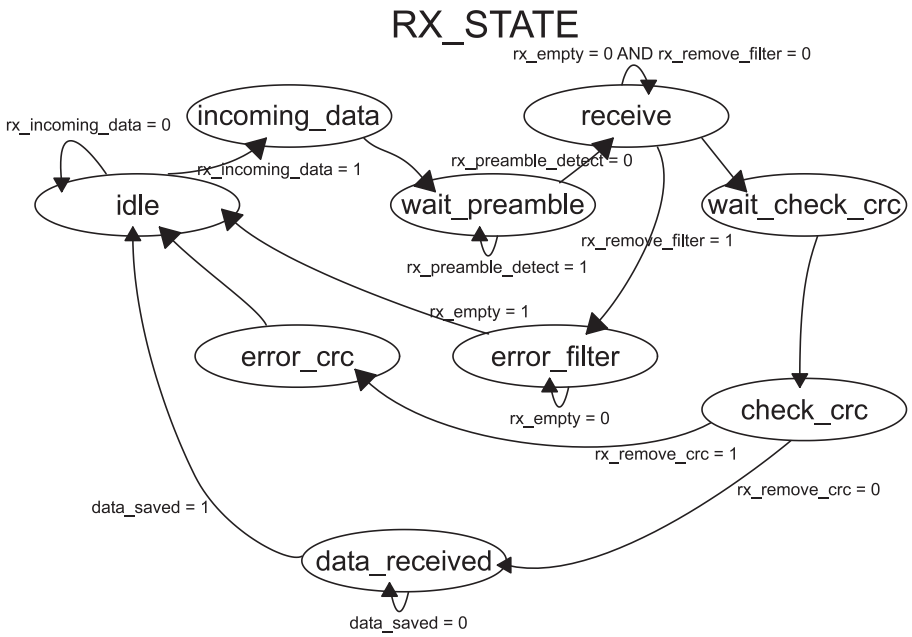
FIFO paměť velikost 32x4 bitu, která zachycuje přijatá data a odesílá signály o svém stavu pro rx_state. Signál rx_incoming_data se nastaví k okamžiku, kdy přicházejí validní data ze sběrnice RXD. Když je nastaven signál rx_en_fifo, tak data v paměti jsou s další náběžnou hranou hodin přiřazena na výstup paměti a následně odebrána z paměti. Signál rx_empty se nastaví, je-li paměť zcela prázdná, tedy už neprobíhá žádný příjem dat.

3.3.6 RX_CRC32

Existují dva způsoby jak ověřit přijatý kontrolní součet. Můžeme spočítat kontrolní součet pro celá přijatá data a poté ověřit, zda kontrolní součet odpovídá 4 bytové hodnotě na konci rámce. Druhá možnost je o něco jednodušší. Spočívá v tom, že počítáme kontrolní součet pro celá data včetně kontrolního rámce na konci. Pokud jsou data správná, vyjde nám tzv. magické číslo 0xC704DD7B, které je pro všechny pakety s CRC32 stejné. Blok spočítaný kontrolní součet porovná a neshodují-li se, pak odešle chybný signál rx_remove_crc, který signalizuje paket s chybným kontrolním součtem

3.3.7 RX_STATE

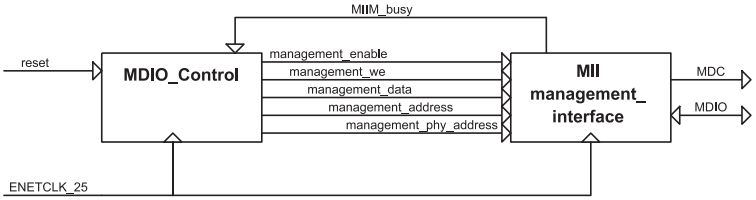
Jak u TX_STATE se jedná o Moore konečný automat. Po resetu se nachází ve stavu idle a čeká na signál rx_incoming_data přicházející z RX_FIFO, který značí příchod nových dat. Povolí odesílání dat do RX_DEMUX a čeká na zrušení signálu rx_preamble_detect. Po něm totiž nastává příjem platných dat rámce začínající cílovou MAC adresou. Při přijímání dat poslouchá signál rx_remove_filter, po kterém by vyhodnotil paket jako neplatný. Po přijetí poté vyšle signál pro RX_CRC pro ověření správnosti dat. Jsou-li v pořádku, paket je uložen do paměti a čeká na signál data_saved, který je v projektu přiřazen na SW(15), ten paměť uvolní a je možný opět nový zápis do paměti.



Obrázek 3.6: Stavový diagram pro RX_STATE

3.4 Sériové rozhraní pro správu PHY (MIIM)

Bez komunikace pro správu PHY by oba bloky pro vysílání a přijímání paketu nemohly fungovat. MIIM po resetu zapíše do PHY registrů data, která jsou definována ve zdrojovém kódu. Zápisem zajistíme ethernetový přenos 100Mbit/s v duplexním režimu neboli Fast Ethernet.

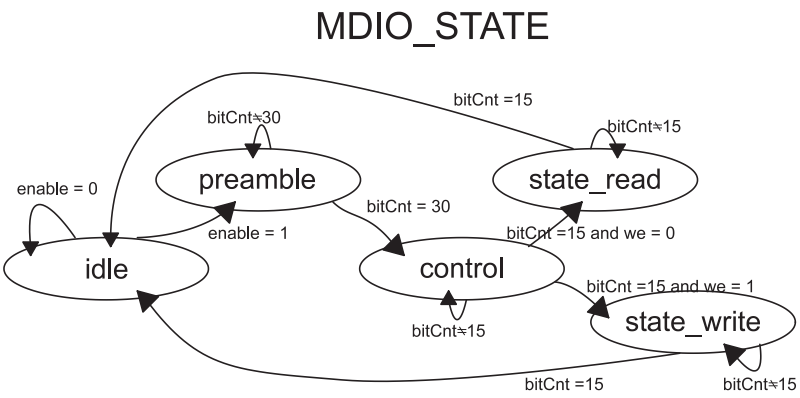


Obrázek 3.7: Blokové schéma pro MIIM

3.4.1 MII_Management_Interface

MII_Management_Interface obsahuje konečný automat který se stará o správnou sériovou komunikaci pomocí sběrnice MDIO. Hodiny MDC jsou tvořeny děličkou 16 hodinového signálu ENETCLK_25. Vnitřní logika obsahuje stavový automat, který v závislosti na vstupních signálech generuje výstupní sériový signál po obousměrné jednobitové sběrnici MDIO. Po resetu se auto-

mat nachází ve stavu idle a čeká na signál enable, který spustí komunikaci. Poté nastane předdefinovaná komunikace daná normou IEEE 802.3 po které, se automat vrátí do svého nečinného stavu idle.



Obrázek 3.8: Konečný automat pro MII Management Interface

3.4.2 MDIO_Control

MDIO_Control čeká na signál MIIM_busy = 0, kterým odešle MII_management_interface potřebná data pro čtení/zápis registru na dané adrese. Už se nestará o průběh komunikace a čeká na signál MIIM_busy = 0. Po odeslání všech dat už tento blok je nečinný a opět se zaktivuje až při novém resetu.

Kapitola 4

Testování

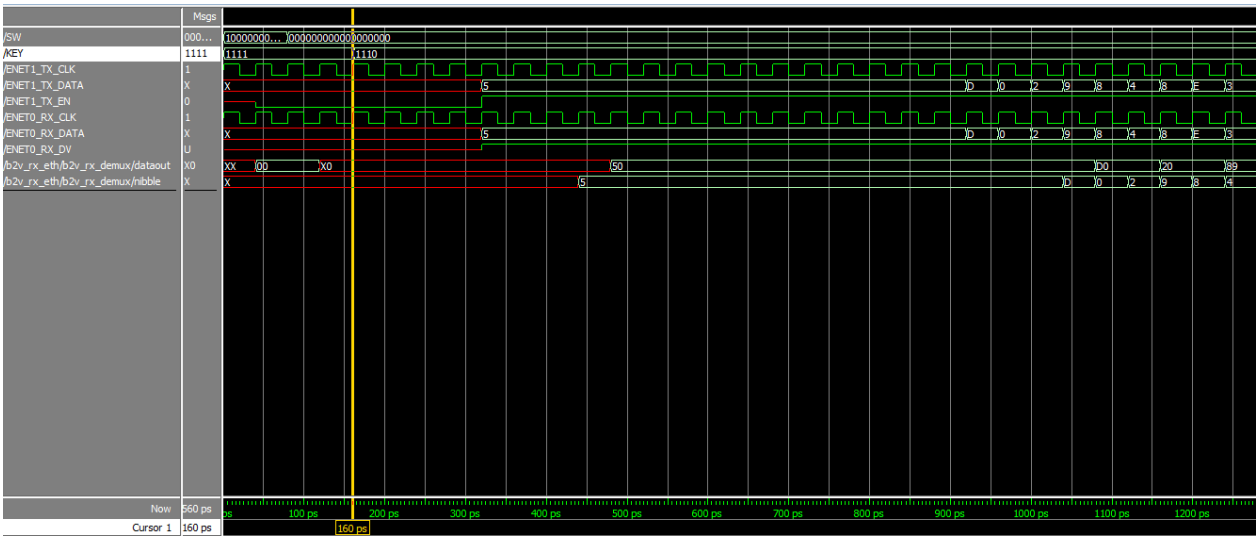
Nyní přichází na řadu testování naprogramovaného projektu. Projekt nabývá velikost přibližně 3000 řádků kódu a je rozdělen celkem na 21 VHDL souborů. Dle zadání projekt musí být schopen odesílat data do protilehlé FPGA desky, a ta je musí uložit do své paměti. V poslední řadě si také otestujeme, zda jsou pakety korektní a je-li možné je detekovat počítačem připojeným k desce.

4.1 Čtení a zápis pevného bloku dat po Ethernetu

4.1.1 Simulace

Testování projektu, pokud nevlastníme desku nebo se chceme podívat na jednotlivý drát či registr na desce, provádíme pomocí simulace v počítači. Pro naši Alteru DE2-115 máme k dispozici dva simulátory. Jsou to Quartus II Simulátor, zabudovaný simulátor do vývojového prostředí Quartus a ModelSim Altera, fungující jako samostatný program poskytující profesionální prostředí pro simulaci, v něm také budeme výsledný projekt simulovat.
V projektu komunikujeme s PHY čipem, který nemůžeme nijak odsimulovat, neboť k němu neexistují zdrojové kódy. Vstupní data z PHY čipu do našeho projektu musíme v simulaci předdefinovat, stejně jako vstupní hodnoty z přepínačů a tlačítek. Na obrázku 4.1 je znázorněna simulace v ModelSimu.

- Vysílač
  - v čase 0 – 80 je provedeno resetování pomocí SW(17)
  - v čase 160 je stisknuto odeslán signál send pomocí KEY(0)
  - v čase 320 začíná být odesílána preambule pro PHY, končící SFD (0x5D)
  - od času 960 jsou odesílána vlastní data, LSB poté MSB



Obrázek 4.1: ModelSim simulace obvodu

- Přijímač
  - v čase 0 – 80 je provedeno resetování pomocí SW(17)
  - v čase 320 jsou nastavena data, které očekáváme z PHY čipu
  - v čase 480 je detekována preambule a blok DEMUX čeká na SFD.
  - v čase 1160 jsou již data přiřazována do vnitřní paměti přijímače

RTL simulace obvodu probíhá správně. Funkčnost ověříme i na desce DE2-115.

4.1.2 Realizace na DE2-115

Pro správné fungování komunikace je třeba propojit jednotlivé ethernetové porty kříženým kabelem. Následně provedeme hardware reset obou čipů SW(16) (necháme v poloze generující logickou 1). Tento reset vede k defaultnímu nastavení PHY zejména jeho registrů. Po několika sekundách je proces dokončen. Resetujeme náš obvod čímž, nastavíme náš čip na Fast Ethernet. Nyní je obvod funkční.

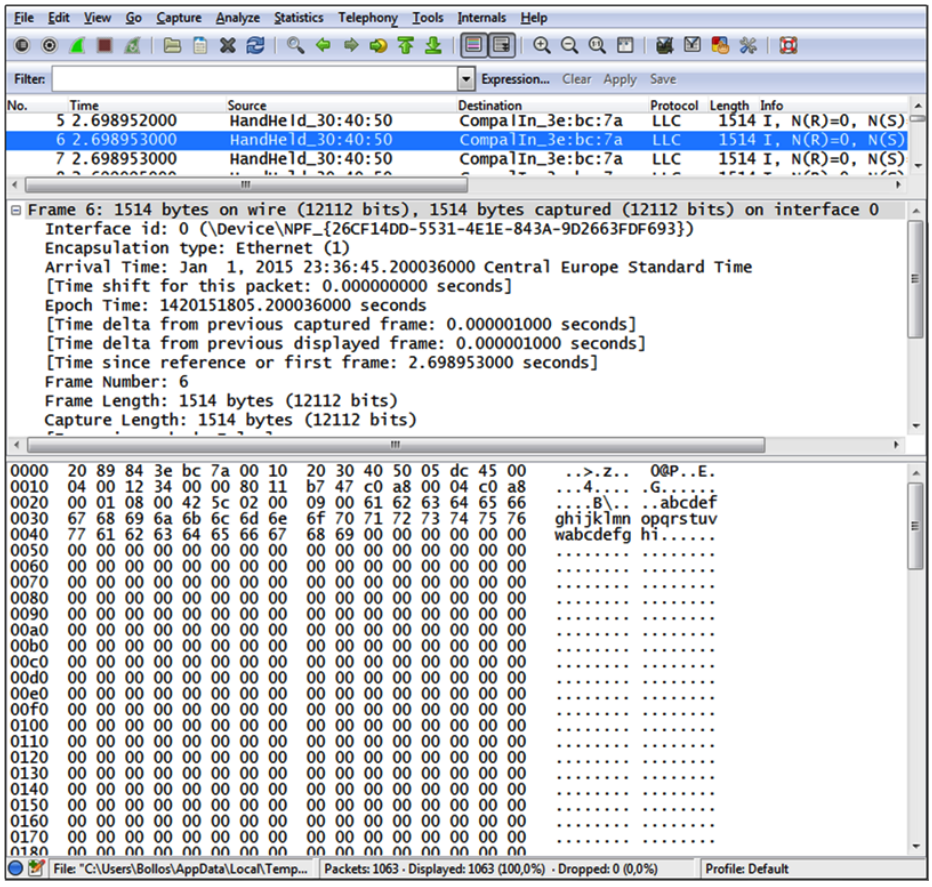
- Data z paměti odesíláme stisknutím KEY(0) (send = 1).
- Data jsou přijata pokaždé, je-li SW(15) v horní poloze (data_saved = 1).

Signál data_saved má přijímači oznamovat, že přijatý paket je uložen. Přijímač bude poté svoji paměť opět využívat pro uložení nově příchozích

paketů. Pro kontrolu správnosti paketu stačí vyvést signál rx_data_valid z RX_STATE. Signál nastavíme do jedničky, nachází-li se RX_STATE ve stavu data_received, pak signál značí správně přijatý paket.

4.2 Detekování odeslaných a přijatých paketů počítačem

Detekce paketu na počítači provedeme pomocí freeware softwaru Wireshark [2]. Program detekuje správně přijaté pakety a zobrazí jejich data. Paket, který obsahuje špatný kontrolní součet, je zahozen MAC na síťové kartě, proto ho tento program nedokáže detekovat. Naše pakety však mají formát správný a jsou programem detekovány. Můžeme si zobrazit jednotlivá data na dané adrese.
Důležitou položkou je zde určitě čas, kterým můžeme vypočítat, kolik paketů je za daný časový úsek generováno. Tato hodnota pak odpovídá přenosové rychlosti. Provedl jsem dva testy a vypočítal rychlost přenosu.



Obrázek 4.2: Detekování paketu programem Wireshark

- Odesláno 2355 paketů o datové velikosti 1500 bytů s časem 0.2899 sekund. Celková přenosová rychlost je 97.48 Mbit/s
- Odesláno 103145 paketů o datové velikosti 1500 bytů s časem 15.1146 sekund. Celková přenosová rychlost je 81.89 Mbit/s

Přenos dat není 100 Mbit/s, neboť ethernetový rámec obsahuje preambuli, kontrolní součet a minimální 32 bitovou mezeru mezi rámci. Také nesmíme zapomenout, že nějaké pakety mohly být zahozeny, neboť nesouhlasil kontrolní součet. Zpomalení rychlosti při odesílání většího počtu paketů nám nejspíše říká, že sám počítač nestíhá tolik paketů najednou přijmout.

Závěr

V mojí bakalářské práci jsem navrhl obvod pro ethernetovou komunikaci, který jednoduše odešle data na druhé zařízení. Vytvořil jsem v jazyce VHDL 3 nezávislé pracující bloky: Vysílač, Přijímač a Sériovou komunikaci s PHY. Jejich hlavní přínosy spočívají v rychlosti a jednoduchosti, protože pracující samostatně, a ne jako periferie procesoru NIOS, což používají dosavadní známá řešení problému, viz [7] a [6]. Přenos je navíc rychlý, protože probíhá na úrovni hardwaru. Navíc zde nejsou problémy s licencí. Projekt jsem vytvořil jako open source a může ho tedy využít každý ve svém vlastním projektu podle podmínek licence GNU (GNU General Public License). Ethernetová komunikace může probíhat jako s jinou deskou DE2-115, tak i s počítačem. Zde existuje značný prostor pro pokračování či rozšíření projektu.
Výsledky testů ukazují, že všechna data nemusí nutně dorazit na cílové zařízení, což je charakter internetového protokolu UDP. Pokud však data přijdou, díky kontrolnímu součtu máme jistotu jejich správnosti. Zde se samozřejmě nabízí možné rozšíření projektu o potvrzování přijatých dat pomocí implementace TCP/IP protokolu. Výsledný projekt by pak mohl komunikovat v síti Internet jako plnohodnotné zařízení.

Literatura

[1] Internet Protocols. [online], 1999. Dostupné z: http://fab.cba.mit.edu/classes/MIT/961.04/people/neil/ip.pdf

[2] Wireshark Freeware. [online], 2014. Dostupné z: http://www.wireshark.org

[3] Altera: *Altera Datasheet DE2-115*. Altera, 2010, [cit. 2014-12-28]. Dostupné z: ftp://ftp.altera.com/up/pub/Altera_Material/13.0/Boards/DE2-115/DE2_115_User_Manual.pdf

[4] Altera: DE2-115 Development and Education Board. [online], 2014, [cit. 2014-12-28]. Dostupné z: http://www.altera.com/education/univ/materials/boards/de2-115/unv-de2-115-board.html

[5] Altera: NIOS Embedded Processor. [online], 2014, [cit. 2014-12-28]. Dostupné z: http://www.altera.com/devices/processor/nios2/ni2-index.html

[6] Altera: Triple-Speed Ethernet MegaCore. [online], 2014, [cit. 2014-12-28]. Dostupné z: http://www.altera.com/products/ip/iup/ethernet/m-alt-ethernet-mac.html

[7] Cornell University, G. A., Spanier Michael: Ethernet Communication Interface for the FPGA. [online], 2011, [cit. 2014-12-28]. Dostupné z: http://people.ece.cornell.edu/land/courses/ece5760/FinalProjects/f2011/mis47_ayg6/mis47_ayg6/

[8] Institute of Electrical and Electronics Engineers: *IEEE 802.3*. IEEE, 2012, [cit. 2014-12-28]. Dostupné z: http://standards.ieee.org/about/get/802/802.3.html

[9] Marvell: *Marvell-Alaska-Ultra-88E1111 Datasheet*. Marvell, 2013, [cit. 2014-12-28]. Dostupné z: http://www.marvell.com/transceivers/assets/Marvell-Alaska-Ultra-88E1111-GbE.pdf

[10] Petr, B.: TCP/IP model. [online], 2007, [cit. 2014-12-28]. Dostupné z: http://www.samuraj-cz.com/clanek/tcpip-model-encapsulace-paketu-vs-ramec/

[11] Wikipedia.org: Terabit Ethernet. [online], 2014, [cit. 2014-12-28]. Dostupné z: http://en.wikipedia.org/wiki/Terabit_Ethernet

Příloha **A**

Seznam použitých zkratek

100Base-TX 100MBits Twisted pair (Fast Ethernet)

ARP Address Resolution Protocol

Auto MDI-X Auto Medium Dependent Interface Crossover

CSMA/CD Carrier Sense Multiple Access with Collision Detection

CRC Cyclic Redundancy Check

DE2-115 Development and Education board

FIFO First In First Out

FPGA Field Programmable Gate Array

GMII Gigabit Media Independent Interface

IEEE Institute of Electrical and Electronics Engineers

IP Internet Protocol

LED Light-Emitting Diode

MAC Media Access Control

MDC Management Data Clock

MDIO Management Data Input/Output

MII Media Independent Interface

MIIM Media Independent Interface Management

MTU Maximum transmission unit

MUX Multiplexer

OSI Open Systems Interconnection

PDU Protocol data unit

RMII Reduced Media Independent Interface

RXD Receive Data

SDU Service data unit

SRAM Static Random Access Memory

TCP Transmission Control Protocol

UDP User Datagram Protocol

TBI Ten-Bit Interface

TXD Transmit Data

VHDL Very High Speed Integrated Circuit Hardware Description Language

Příloha **B**

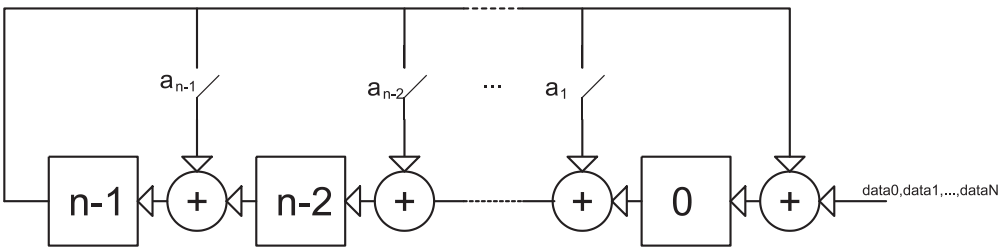
Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `GNU General Public License v3.0.html` ..... GNU GPL licence
- `src`
  - `eth` ..... zdrojové kódy ethernetové komunikace ve VHDL
    - `eth.vhd` ..... finální VHDL projekt
    - `rx_eth.vhd` ..... Struktura propojení bloky přijímače
    - `tx_eth.vhd` ..... Struktura propojení bloky vysílače
    - `miim.vhd` ..... Struktura propojení MIIM rozhraní
    - `*.vhd` ..... vnitřní stkruktury, celkem 17 VHDL souborů
    - `wave.do` ..... předdefinování vstupních parametrů v modelSimu
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
    - `pic` ..... zdrojové obrázky pro LaTeX pdf
- `text`
  - `thesis.pdf` ..... text práce ve formátu PDF

Příloha C

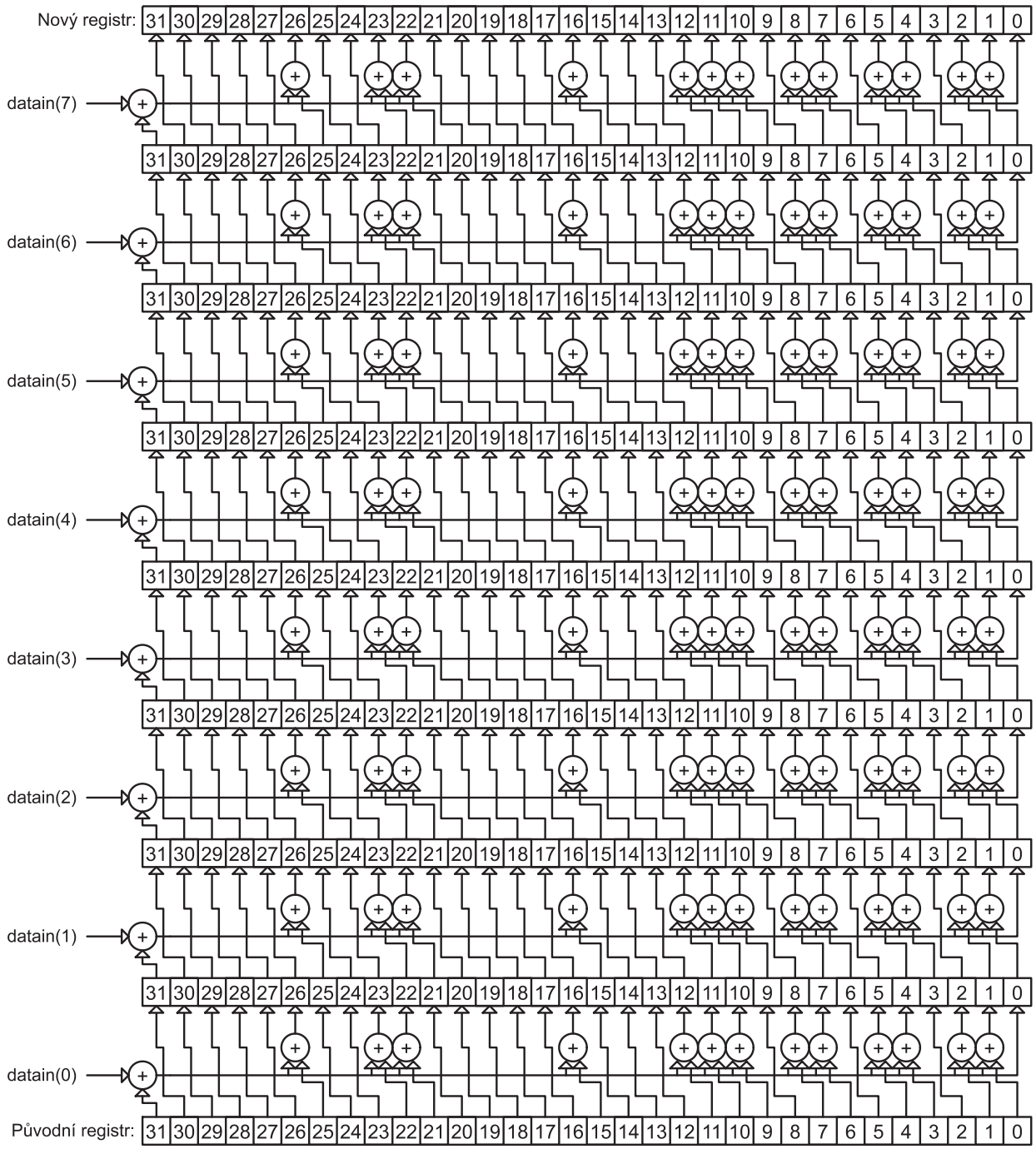
Galoisův posuvný registr

Výpočet kontrolního součtu probíhá za pomocí Galoisova lineárního zpětnovazebního posuvného registru (anglicky Galois Linear Feedback Shift Register). Koeficienty generujícího polynomu nám označují XOR mezi jednotlivými bity posuvného registru. Vstupní hodnotou je jednobitová hodnota, která je exkluzivně sečtena s poslední hodnotou Galoisova registru a zapsána na první pozici. Obecný Galoisův posuvný registr je zobrazen na obrázku C.1.



Obrázek C.1: Obecná struktura Galoisova posuvného registru

V projektu aktualizujeme Galoisův posuvný registr po vstupních datech velikosti 8 bitů. Registr se tedy aktualizuje osmi kroky. Výsledný posuvný registr si můžeme představit jako strukturu po sobě jdoucích posuvných registrů, které je zobrazeno na obrázku C.2. Operace XOR s příslušným signálem je tedy použita na každý původní bit posuvného registru tolikrát, kolik má v cestě k výsledné hodnotě XORů.



Obrázek C.2: Obecná struktura Galoisova posuvného registru

Příloha D

Zdrojový kód pro top-level-entity

Top-level entita, obsahuje propojení (port map) ostatních komponent vnořených do této entity. Jedná se tedy o vrchní část obvodu, kde jsou popsány spoje mezi vstupní/výstupními piny desky s vnitřními bloky (TX,RX,MIIM) obvodu.

```
-- PROGRAM    "Quartus II 64-Bit"
-- CREATED    "Tue Nov 18 21:25:06 2014"

LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY work;

ENTITY Eth IS
  port
  (
    --ENET0_TX_CLK   : in  std_logic;
    ENET1_TX_CLK    : in  std_logic;
    ENET0_RX_CLK    : in  std_logic;
    --ENET1_RX_CLK   : in  std_logic;
    ENET0_RX_DATA   : in  std_logic_vector(3 DOWNTO 0);
    --ENET1_RX_DATA   : in  std_logic_vector(3 DOWNTO 0);
    --ENET0_TX_DATA   : out  std_logic_vector(3 DOWNTO 0);
    ENET1_TX_DATA   : out  std_logic_vector(3 DOWNTO 0);
    --ENET0_TX_EN    : out  std_logic;
    ENET1_TX_EN     : out  std_logic;
    ENET0_RX_DV     : in  std_logic;
    --ENET1_RX_DV    : in  std_logic;
```

```
    ENET0_RST_N    : out  std_logic;
    ENET1_RST_N    : out  std_logic;
    ENETCLK_25     : in   std_logic;
    ENET0_MDC      : out  std_logic;
    ENET1_MDC      : out  std_logic;
    ENET0_MDIO     : inout  std_logic;
    ENET1_MDIO     : inout  std_logic;
    ENET1_GTX_CLK   : out std_logic;
    ENET1_TX_ER    : out std_logic;
    KEY        : in  std_logic_vector(3 DOWNTO 0);
    SW       : in  std_logic_vector (17 DOWNTO 0);
    --LEDG     : out  std_logic_vector(8 DOWNTO 0);
    LEDR       : out  std_logic_vector(17 DOWNTO 0);
    HEX0       : out  std_logic_vector(6 downto 0);
    HEX1       : out  std_logic_vector(6 downto 0);

    -- pro odesilani dat
    we       : in std_logic;
    datain       : in std_logic_vector(7 downto 0);
    address    : in std_logic_vector(10 downto 0)

  );
end Eth;

ARCHITECTURE top_type OF Eth IS

  COMPONENT MIIM
    port(
        eth_CLK      : in  std_logic;
        eth_reset      : in std_logic;
        eth_MDC      : out  std_logic;
        eth_MDIO     : inout  std_logic

    );
  end COMPONENT;

  COMPONENT rx_eth
    port(
        eth_RX_CLK     : in  std_logic;
        eth_reset      : in std_logic;
        eth_RX_DATA    : in  std_logic_vector(3 DOWNTO 0);
        eth_RX_DV      : in  std_logic;
        eth_sram_data  : out std_logic_vector(7 downto 0);
        eth_address      : in std_logic_vector(10 downto 0);
```

```
        data_saved     : in std_logic;
        eth_SEGout_LSB  : out std_logic_vector(6 downto 0);
        eth_SEGout_MSB  : out std_logic_vector(6 downto 0)
    );
end COMPONENT;

COMPONENT tx_eth
    port(
        eth_TX_CLK     : in  std_logic;
        eth_reset      : in std_logic;
        eth_TX_DATA    : out  std_logic_vector(3 DOWNTO 0);
        eth_TX_EN      : out  std_logic;
        eth_counter    : out std_logic_vector(1 downto 0);
        eth_send     :  in std_logic;
        eth_we     : in std_logic;
        eth_data     : in std_logic_vector(7 downto 0);
        eth_address    : in std_logic_vector(10 downto 0);
        eth_error_write : out std_logic

    );
end COMPONENT;

    signal  mdc_wire       : std_logic;
    signal  mdio_wire      : std_logic;
    signal   reset_wire      : std_logic;
    signal   data_saved_wire     : std_logic;
    signal   eth_send_wire     : std_logic;
    signal  hardware_reset_N_wire : std_logic;
    signal  eth_we_wire      : std_logic;
    signal  eth_data_wire    : std_logic_vector(7 downto 0);
    signal  eth_address_write_wire  : std_logic_vector(10 downto 0);

begin
    reset_wire <= SW(17);
    hardware_reset_N_wire <= SW(16);
    data_saved_wire <= SW(15);
    eth_send_wire <= not (KEY(0));
    eth_we_wire <= we;
    eth_data_wire <= datain;
    eth_address_write_wire <= address;
    ENET0_RST_N <= hardware_reset_N_wire;
    ENET1_RST_N <= hardware_reset_N_wire;
    ENET1_GTX_CLK <= '0';
    ENET1_TX_ER <= '0';
```

```
    ENET0_MDC <= mdc_wire;
    ENET1_MDC <= mdc_wire;
    ENET0_MDIO <= mdio_wire;
    ENET1_MDIO <= mdio_wire;

    b2v_MIIM : MIIM
    port map(

        eth_CLK    => ENETCLK_25,
        eth_reset    => reset_wire,
        eth_MDC      => mdc_wire,
        eth_MDIO     => mdio_wire
     );

    b2v_rx_eth : rx_eth
    port map(
        eth_RX_CLK     => ENET0_RX_CLK,
        eth_reset      => reset_wire,
        eth_RX_DATA    => ENET0_RX_DATA,
        eth_RX_DV      => ENET0_RX_DV,
        eth_sram_data => LEDR(7 downto 0),
        eth_address      => SW(10 downto 0),
        data_saved     => data_saved_wire,
        eth_SEGout_LSB  => HEX0,
        eth_SEGout_MSB  => HEX1
    );

    b2v_tx_eth : tx_eth
    port map(

        eth_TX_CLK     => ENET1_TX_CLK,
        eth_reset      => reset_wire,
        eth_TX_DATA    => ENET1_TX_DATA,
        eth_TX_EN      => ENET1_TX_EN,
        eth_counter    => LEDR(17 downto 16),
        eth_send     => eth_send_wire,
        eth_we     => eth_we_wire,
        eth_data     => eth_data_wire,
        eth_address    => eth_address_write_wire,
        eth_error_write    => LEDR(15)
     );
end top_type;
```