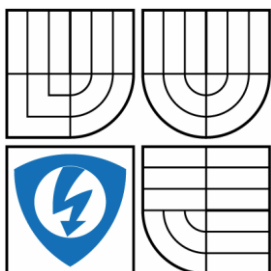


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH
TECHNOLOGIÍ

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF CONTROL AND INSTRUMENTATION

DATOVÝ KONCENTRÁTOR VE VHDL

SERIAL DATA CONCENTRATOR IN VHDL

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE

AUTHOR

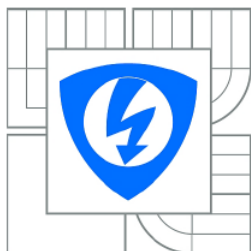
TOMÁŠ ŘEHÁČEK

VEDOUCÍ PRÁCE

SUPERVISOR

Ing. FRANTIŠEK BURIAN

BRNO 2013



VYSOKÉ UČENÍ
TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

Ústav automatizace a měřicí techniky

Bakalářská práce

bakalářský studijní obor
Automatizační a měřicí technika

Student: Tomáš Řeháček
Ročník: 3

ID: 136580
Akademický rok: 2012/2013

NÁZEV TÉMATU:

Datový koncentrátor ve VHDL

POKYNY PRO VYPRACOVÁNÍ:

1. Seznamte se se základními principy komunikace v digitální technice. Zaměřte se na sériové sběrnice (sériová sběrnice a Ethernet).
2. Seznamte se se základy VHDL a s hradlovými poli firmy Xilinx. Seznamte se s vývojovým kitem Xilinx Spartan-3A Evaluation Kit.
3. Navrhněte na vývojovém kitu jednoduchý datový koncentrátor jedné rychlé sériové linky na ethernetový port protokolem UDP/IP.
4. Při práci použijte ethernetový převodník PHY dodaný vedoucím práce.
5. Vytvořte program pro PC, který je schopen otestovat funkčnost navrženého zařízení.

DOPORUČENÁ LITERATURA:

ASHENDEN, Peter J. VHDL Cookbook. South Australia, 1990.

Termín zadání: 11.2.2013

Termín odevzdání: 27.5.2013

Vedoucí práce: Ing. František Burian
Konzultanti bakalářské práce:

doc. Ing. Václav Jirsík, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

Autor bakalářské práce nesmí při vytváření bakalářské práce porušit autorská práva třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č.40/2009 Sb.

Abstrakt

V textu bakalářské práce jsou shrnuty základy jazyka VHDL, popsáno vývojové prostředí ISE Webpack. Dále je popsán způsob komunikace a vlastnosti sériové linky. A také vlastnosti standardu Ethernet a způsob komunikace přes rozhraní MII. V další části je stručně představen hardware použitý pro návrh datového koncentrátoru. A na závěr je v práci rozebrán samotný návrh datového koncentrátoru pro přenos dat ze sériové linky na Ethernet pomocí protokolů UDP/IP.

Klíčová slova

VHDL, datový koncentrátor, sériová komunikace, Ethernet, IPv4, UDP, CRC, MII

Abstract

In the bachelor's thesis text there are summarized basics of VHDL language, ISE Webpack development environment is described. Also the way of communication and properties of serial unit are described as well as Ethernet standard properties and the way of communication via MII interface. Hardware used for the data concentrator design is briefly introduced in the next part. At the end of the thesis there is analysis of the data concentrator design for data transmission from serial unit to Ethernet using protocols UDP/IP.

Keywords

VHDL, data concentrator, serial communication, Ethernet, IPv4, UDP, CRC, MII

Bibliografická citace:

ŘEHÁČEK, T. *Datový koncentrátor ve VHDL*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2013. 56s. Vedoucí bakalářské práce byl Ing. František Burian.

Prohlášení

„Prohlašuji, že svou bakalářskou práci na téma Datový koncentrátor ve VHDL jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské 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 části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne: **24. května 2013**

.....
podpis autora

Poděkování

Děkuji vedoucímu bakalářské práce Ing. Františku Burianovi za odbornou pomoc, ochotu při konzultacích a další cenné rady při zpracování mé bakalářské práce. Děkuji také všem, kteří mě podporovali při vytváření práce.

V Brně dne: **24. května 2013**

.....
podpis autora

Obsah

1	Úvod	11
2	Jazyk vhd1	12
2.1	Představení jazyka	12
2.2	Popis základů jazyka VHDL	12
2.2.1	Zápis čísel, znaků a řetězců	13
2.2.2	Entita (entity)	13
2.2.3	Architektura (architecture)	13
2.2.4	Procesy	14
2.2.5	Podprogramy	14
2.2.6	Porty	15
2.2.7	Datové typy	15
2.2.8	Datové objekty	16
2.2.9	Operátory	16
2.2.10	Příkazy	17
2.2.11	Atributy	18
3	Xilinx software	18
3.1	ISE Webpack	18
3.1.1	XTS Synthesis	19
3.1.2	PlanAhead	19
3.1.3	ISE Simulator (ISim)	20
3.1.4	iMPACT	20
4	UART	20
4.1	Asynchronní sériová komunikace	20
5	Ethernet	22
5.1	Ethernetový rámec	22
5.1.1	IPv4 protokol	23
5.1.2	CRC	26
5.2	MII	29
6	Hardware použitý pro návrh	32
6.1	Spartan-3A Evaluation Kit	32
6.1.1	Xilinx Spartan-3A	33
6.1.2	Cypress PSoC	33
6.2	Programovací kabel JTAG HS2	34

6.3	Převodník USB na UART	34
6.4	Ethernetový modul	35
7	První programy.....	37
7.1	Vytvoření pomocí schématu.....	37
7.2	Vytvoření pomocí VHDL kódu.....	38
8	Návrh datového koncentrátoru	40
8.1	Sériová linka (Návrh sériové linky).....	42
8.1.1	Popis bloku (baud_rate_generator)	43
8.1.2	Popis bloku (seriova_linka_automat).....	43
8.1.3	Popis bloku (seriova_linka_prevod_dat).....	44
8.2	Ethernet (Návrh Ethernetu)	44
8.2.1	Popis bloku (Ethernet_test_blok).....	45
8.2.2	Popis bloku (Ethernet_detekce_COL)	46
8.2.3	Popis bloku (Ethernet_prenos_dat).....	46
8.3	Aplikace pro otestování funkčnosti	49
9	Výpisy z návrhu	50
10	Závěr.....	51

Seznam obrázků

Obrázek 3.1: XST syntéza kódu [3]	19
Obrázek 4.1: Asynchronní sériový přenos [10]	21
Obrázek 5.1: Propojení MII rozhraní	30
Obrázek 5.2: Způsob přenosu dat přes MII rozhraní [13]	31
Obrázek 6.1: Blokové schéma Spartan-3A Evaluation Kit [11]	32
Obrázek 6.2: Spartan-3A Evaluation Kit [11]	33
Obrázek 6.3: Programovací kabel JTAG HS-2 [5]	34
Obrázek 6.4: Převodník USB na UART	35
Obrázek 6.5: Blokové schéma Ethernetového modulu [14]	35
Obrázek 6.6: Ethernetový modul	35
Obrázek 6.7: Časový diagram pro Ethernetový modul [14]	36
Obrázek 7.1: Schéma hodinového generátoru 1Hz	38
Obrázek 7.2: Naměřený průběh z navržených generátorů	39
Obrázek 8.1: Schéma propojení hardwaru při návrhu	40
Obrázek 8.2: Softwarové schéma propojení bloků	42
Obrázek 8.3: Propojení bloků pro převod sériové informace na paralelní	42
Obrázek 8.4: Stavový automat pro sériovou linku	43
Obrázek 8.5: Simulace pro blok - seriova_linka_automat	44
Obrázek 8.6: Simulace pro blok - seriova_linka_prevod_dat	44
Obrázek 8.7: Zapojení bloků pro Ethernetový přenos	45
Obrázek 8.8: Simulace pro blok - Ethernet_prenos_dat	47
Obrázek 8.9: Testovací aplikace	50
Obrázek 8.10: Zobrazení přenosu dat s využitím aplikací	50

Seznam tabulek

Tabulka 2.1: Operátory ve VHDL	17
Tabulka 5.1: Ethernetový rámec	22
Tabulka 5.2: Formát protokolu IPv4	24
Tabulka 5.3: Formát protokolu UDP	25
Tabulka 5.4: První tabulka pro sestavení paralelního CRC	28
Tabulka 5.5: Druhá tabulka pro sestavení paralelního CRC	28
Tabulka 6.1: Časové parametry pro rychlost 10Mb/s (vztahují se k obrázku 6.7)	36
Tabulka 6.2: Časové parametry pro rychlost 100Mb/s (vztahují se k obrázku 6.7)	36
Tabulka 8.1: Tabulka propojení signálů	41
Tabulka 8.2: Konkrétní dosazené hodnoty v protokolu IPv4	48
Tabulka 8.3: Konkrétní dosazené hodnoty v protokolu UDP	48

1 ÚVOD

Úkolem této bakalářské práce je seznámit se s principy digitální komunikace a to se sériovou linkou, standardem Ethernet a s programovacím jazykem VHDL. A pomocí těchto znalostí navrhnu jednoduchý datový koncentrátor, který bude přijímat data po sériové lince a posílat je dále na Ethernetové rozhraní. Pro návrh bude využita vývojová deska s hradlovým polem od firmy Xilinx, konkrétně typ Spartan-3A Evaluation Kit. V prvních kapitolách práce budou shrnuty všechny informace získané při vypracovávání a potřebné pro úspěšný návrh datového koncentrátoru. V závěrečné části práce bude představen výsledný návrh, jakým je realizován datový koncentrátor pro přenos dat ze sériové linky na Ethernet pomocí protokolů UDP/IP a shrnuty jeho vlastnosti.

2 JAZYK VHDL

2.1 Představení jazyka

Jazyk VHDL (VHSIC Hardware Description Language) začal vznikat v roce 1981 v rámci výzkumného projektu VHSIC (Very High Speed Integrated Circuits) ministerstva obrany Spojených států amerických. Cílem projektu VHSIC bylo vytvořit efektivní popis velmi rozsáhlých integrovaných obvodů, který by nebyl závislý na cílové technologii, neboť popis obvodů byl do té doby řešen pomocí vzájemně nekompatibilních nástrojů a jazyků.

Základní definici jazyka vytvořili firmy IBM, Intermetrics a Texas Instruments, jazyk VHDL byl poprvé veřejně publikován v roce 1985. Další vývoj a standardizaci předalo ministerstvo obrany organizaci IEEE (Institute of Electrical and Electronics Engineers), která má každých 5 let jazyk revidovat. První standard vydala organizace v roce 1987 pod označením „IEEE Standard VHDL Language Reference Manual (IEEE Std 1076-1987)“, někdy také označován jako VHDL-87. Následovala revize „IEEE Std 1076-1993“ označována jako VHDL-93. Tyto dvě revize byly postupně implementovány do návrhových systémů všemi výrobci a zůstaly také nejvíce rozšířeny. Poslední revize vznikla roku 2009 publikována jako „IEEE Std 1076-2008“.

Jazyk VHDL je tedy jazyk vysoké úrovně a slouží k popisu a simulaci rozsáhlých číslicových obvodů (pro popis hardware). Má bohaté vyjadřovací schopnosti a je značně nezávislý na konečně použité technologii (použití pro hradlová pole PLD i FPGA). Díky tomu, že je standardizovaný, dá se stejný kód použít v různých prostředích od různých výrobců. Pro jakou cílovou technologii bude VHDL kód použit, záleží právě až na jeho kompilaci, kterou řeší jednotlivé překladače.

2.2 Popis základů jazyka VHDL

Hlavní části v jazyce VHDL tvoří deklarace Entity a popis Architektury, které dohromady vytvářejí základní moduly. Složité systémy jsou zpravidla popisovány hierarchickou strukturou jednotlivých modulů.

Jazyk VHDL popisuje hardware, a tak po syntéze kódu musí jít obvod, který jsme chtěli realizovat, sestavit pomocí tohoto hardwaru (s výjimkou testovacích programů a simulací), to znamená, že při psaní nelze využít všechny možnosti, které nám VHDL nabízí.

Jazyk VHDL nerozlišuje velká a malá písmena v klíčových slovech a názvech identifikátorů. Na velikosti písmen však záleží u řetězců nebo znaků ve výčtových typech.

Klíčová slova VHDL budou v textu vyznačena **tučně**.

2.2.1 Zápis čísel, znaků a řetězců

2.2.1.1 Čísla

Čísla mohou být zapsána jako celá nebo desetinná, desetinná část je oddělena tečkou. Exponent se zapisuje za znakem (**E** nebo **e**). Pro lepší čitelnost mohou být čísla od sebe oddělena podtržítkem, podtržítka nemají na hodnotu čísel žádný vliv. Čísla mohou být zapisována v různých soustavách od dvojkové až do šestnáctkové. Při zápisu v různých soustavách je nejdříve číslem uvedena soustava. Jednotlivé části zapisované v pořadí - soustava, reprezentace čísla a exponent jsou od sebe odděleny znakem #. Při zápisu v dekadické soustavě se soustava neuvádí a jednotlivé části se neoddělují #.

2.2.1.2 Znaky

Znaky se zapisují mezi apostrofy ('a')

2.2.1.3 Řetězce

Zapisují se mezi uvozovky, pokud má řetězec obsahovat uvozovku, musí být zapsána dvakrát ("text").

2.2.1.4 Bitové řetězce

Slouží pro zápis bitových polí (řetězců čísel) ve dvojkové, osmičkové a šestnáctkové soustavě. Před řetězcem v uvozovkách musí být znak soustavy **B** – dvojková, **O** – osmičková, **X** – šestnáctková.

2.2.2 Entita (entity)

Komponenta **entity** popisuje tělo navrhovaného modulu. Pojmenovává a definuje jeho porty, jakého budou typu a směr přenosu signálů (porty jsou popsány v kapitole 2.2.6 a datové typy v kapitole 2.2.7). Může představovat pouze jednoduché hradlo nebo rozsáhlý systém.

Deklarace entity může v hlavičce obsahovat položku **generic**, pomocí které se dají nastavovat parametry i z venku entity (například šířka sběrnice nebo zpoždění signálu). Zlepšuje se tak čitelnost a univerzálnost modulu při dalším použití.

2.2.3 Architektura (architecture)

Komponenta **architecture** popisuje vnitřní strukturu entity (modulu) a je vždy s touto entitou svázána.

Každá entita musí mít alespoň jednu architekturu, ale může být popsána i více architekturami s unikátním jménem. O tom, která architektura bude použita, se potom

rozhoduje pomocí konfigurace (s klíčovým slovem **configuration**), pokud konfigurace pro danou entitu neexistuje, automaticky se použije poslední architektura.

Architektura může být popsána třemi způsoby: behaviorálně, strukturálně a pomocí toku dat. Tyto způsoby popisu je možné kombinovat.

2.2.3.1 Behaviorální popis (popis chování)

Popisuje, jak se mění výstupy v závislosti na změnách vstupů, bez zjevné hardwarové realizace. Popis chování je realizován pomocí procesů (klíčové slovo **process**), procesy popsány v kapitole (2.2.4). Architektura může být popsána i více procesy, které mezi sebou potom komunikují pomocí signálů (signály popsány v kapitole 2.2.8.3).

2.2.3.2 Dataflow (popis toku dat)

Popisuje, jakým způsobem jsou data přenášena na výstupy, příkazy jsou však vykonávány paralelně.

2.2.3.3 Strukturální popis

Popisuje hierarchickou strukturu propojení jednotlivých modulů. Využívá již vytvořených modulů a pouze definuje, jakým způsobem mezi sebou budou propojeny jejich vstupy a výstupy. Pro použití již vytvořených entit v novém modulu, se používá klíčové slovo **component**. Propojování portů se popisuje pomocí **port map**, nastavení generických parametrů entity se provádí pomocí **generic map**.

2.2.4 Procesy

Klíčové slovo **process**. V těle procesu je zapsán algoritmus na základě příkazů, cyklů a podmínek. Příkazy v těle procesu se vykonávají sekvenčně. Pokud je ve stejném čase spuštěno více procesů, tak se tyto procesy vykonávají souběžně. Vykonávání procesu se spustí, pokud dojde ke změně některého signálu uvedeného v citlivostním seznamu (sensitivity list) v hlavičce procesu. Pokud není tento seznam uveden, musí být v těle procesu použity podmínky, které řídí jeho vykonávání – příkazy typu **wait** (kapitola 2.2.10.2). Každý proces je jednou spuštěn na začátku inicializační fáze.

2.2.5 Podprogramy

Podprogramy jsou ve VHDL realizovány pomocí procedur a funkcí. Lze je volat z procesů a jsou vykonávány sekvenčně.

2.2.5.1 Procedury

Klíčové slovo **procedure**. Při volání procedury jí mohou být předány vstupní i výstupní parametry a nevrací návratovou hodnotu.

2.2.5.2 Funkce

Klíčové slovo **function**. Při volání funkce jí mohou být předány pouze vstupní parametry a vrací návratovou hodnotu.

2.2.6 Porty

Porty definují vstupy a výstupy modulu (entity) pomocí klíčového slova **port**. Definují jejich jména, mód přenosu dat a datový typ, který může port přenášet.

2.2.6.1 Módy přenosu dat

- **IN** – vstupní port, data tímto portem do entity pouze vstupují
- **OUT** – výstupní port, data tímto portem mohou z entity pouze vystupovat
- **INOUT** – obousměrný port, data tímto portem mohou do entity vstupovat i z ní vystupovat
- **BUFFER** – výstupní port se zpětnou vazbou, data tímto portem z entity pouze vystupují, avšak mohou být zároveň čtena uvnitř entity (to v režimu **OUT** nelze); ovšem častěji se místo tohoto módu používá mód **OUT** v kombinaci se signálem deklarovaným v architektuře, který potom umožňuje zpětné čtení výstupních dat
- **LINKAGE** – pro neznámý směr toku dat

2.2.7 Datové typy

VHDL poskytuje skalární a kompozitní datové typy. Má i další typy jako je soubor nebo přístupové typy (podobné ukazatelům), ty zde ale nebudou popisovány.

2.2.7.1 Skalární

- výčtové – výčtový typ má definované stavy, kterých může nabývat; stav uveden v definici nejvíce vlevo má nejnižší hodnotu a ta se směrem doprava zvyšuje (předdefinované typy např. **bit**, **boolean**, **character**)
- celočíselné – definuje celočíselný rozsah hodnot (např. **integer**)
- fyzické – vyjadřuje množství vztažené k základní jednotce (např. **time**, kde ms = 1 000 us)
- s plovoucí desetinou čárkou – definuje rozsah desetinných čísel (např. **real**)

2.2.7.2 Kompozitní

- pole – struktura složená z více stejných elementů (např. **string**, **bit_vector**)
- záznam – struktura složená z více různých elementů

2.2.8 Datové objekty

Mezi objekty VHDL patří konstanty, signály, proměnné a soubory. Objekty lze deklarovat v architekturách, procesech atd. Ke každému objektu lze při deklaraci rovnou přiřadit hodnotu.

2.2.8.1 Konstanty

Klíčové slovo **constant**. Při deklaraci konstanty je jí přiřazena hodnota a ta už nemůže být nikde dále v kódu změněna.

2.2.8.2 Proměnné

Proměnné mohou být nesdílené (**variable**), potom jsou deklarované uvnitř procesu. Nebo sdílené (**shared variable**), ty jsou deklarovány v architektuře a jsou potom společné pro celou architekturu (více procesů). Hodnotu proměnných můžeme v kódu měnit. Přiřazení do proměnné se provádí pomocí symbolu ":=".

2.2.8.3 Signály

Klíčové slovo **signal**. Signály reprezentují skutečné vodiče v číslicových systémech. Slouží pro přenos dat mezi komponentami nebo mohou být použity i uvnitř architektur pro komunikaci mezi procesy. Pokud je signálu přiřazena počáteční hodnota, má význam pouze pro simulaci, při syntéze se ignoruje. Pro přiřazení do signálů se používá symbol "<=". Když je do signálu přiřazováno, dá se definovat doba zpoždění pomocí **after** a charakter zpoždění (příkazy **transport**, **reject**, **interval**), podrobnější popis zde nebude uveden.

2.2.8.4 Soubory (files)

Používá se jako přístup k souborům, pro otevření ke čtení a zápisu. Má význam pouze pro simulaci.

2.2.9 Operátory

Operátory v jazyce VHDL jsou vypsány v následující tabulce (tabulka 2.1), operátory s nejvyšší prioritou jsou v tabulce nahoře a směrem dolů se priorita snižuje.

sll, slr – logický posun (vlevo × vpravo) o daný počet pozic
sla, sra – aritmetický posun o daný počet pozic
rol, ror – rotace o daný počet pozic

Tabulka 2.1: Operátory ve VHDL

různé operátory	abs, not, ** (mocnina)
násobící operátory	*, /, mod (modulo), rem (zbytek po dělení)
znaménkové operátory	+, -
operátory sčítání	+, -, & (operátor spojení jednorozměrných polí)
operátory posuvu	sll, slr, sla, sra, rol, ror
relační operátory	=, /=, <, <=, >, >=
logické operátory	and, or, nand, nor, xor, xnor

2.2.10 Příkazy

Zde budou uvedeny typy příkazů a jejich základní rozdělení, zda se vykonávají souběžně nebo sekvenčně.

2.2.10.1 Souběžné (paralelní) příkazy

Všechny paralelní příkazy se vykonávají zároveň, nezáleží tedy na jejich pořadí.

- nepodmíněné přiřazení – přiřazení hodnoty do signálu
- podmíněné přiřazení – přiřazení do signálu, při splnění podmínky (konstrukce **when – else**)
- výběrové přiřazení – přiřazení do signálu na základě hodnoty jiného signálu (konstrukce - **with – select**, s rozhodovací podmínkou **when**)

2.2.10.2 Sekvenční příkazy

Sekvenčně jsou vykonávána těla procesů, procedur a funkcí.

- sekvenční přiřazení do proměnné – pomocí symbolu **:=**
- sekvenční přiřazení do signálu – provádí se stejně jako paralelní nepodmíněné přiřazení, ale s tím, že přiřazení se neprovede okamžitě, ale až v okamžiku, kdy dojde k ukončení nebo pozastavení procesu, pokud během procesu bylo do signálu zapsáno vícekrát, provede se pouze poslední přiřazení.
- příkaz **wait** – Slouží k pozastavení procesu na nějakou dobu.
- Má několik forem:
 - **wait on** – čeká, dokud nenastane změna signálu
 - **wait until** – čeká, dokud není splněna daná podmínka
 - **wait for** – čeká po určitou dobu (pouze pro simulaci)
 - **wait** – čeká pořád (ukončení procesu)

- příkaz **if** – pro vykonání sekvence příkazů pokud je splněna podmínka, při větvení příkazu pomocí **elsif** a **else**, lze vytvářet priority podmínek
- příkaz **case** – pro vykonání sekvence příkazů při splnění podmínky, jednotlivé podmínky nemají žádnou prioritu
- příkaz **loop** – pro cyklické vykonávání sekvence příkazů, dá se použít buďto s **for** (vykonání určitého počtu cyklů), nebo **while** (vykonávání cyklu, dokud je daná podmínka splněna); pro řízení běhu cyklu se dají ještě využít příkazy **next** (přeskočení právě běžící interace) a **exit** pro ukončení vykonávání příkazu **loop**.
- příkaz **null** – nic nevykonává, nejčastěji se používá u příkazu **case**, pro hodnoty, u kterých nechceme provádět žádnou akci
- příkaz **assert** – vyhodnotí podmínku a pokud není splněna, vypíše pomocí příkazu **report** hlášení na textový výstup
- příkaz **return** – pro ukončení vykonávání procedury nebo funkce

2.2.11 Atributy

Datové typy a objekty deklarované ve VHDL, mohou poskytovat další informace, které se dají zjistit pomocí atributů. Atributy se zapisují jako apostrof s názvem atributu za identifikátorem objektu. Různé typy objektů mohou mít různé atributy.

Příklady předdefinovaných atributů:

'**event** – vrátí hodnotu **true**, pokud právě došlo ke změně signálu

'**length** – vrátí počet prvků pole

'**last_value** – vrátí hodnotu před poslední změnou

3 XILINX SOFTWARE

Návrhový software od firmy Xilinx je distribuován jako jeden instalační soubor pod názvem "ISE Design Studio", zde bude popsána verze 14.2. Možnosti, pro jaký hardware se dá použít a jaké jeho části budou zpřístupněny, jsou řešeny jednotlivými licencemi.

3.1 ISE Webpack

Je to návrhové prostředí pro programování FPGA a PLD hradlových polí od výrobce Xilinx. Po zaregistrování na stránkách výrobce je možné získat volnou licenci a začít software využívat. Protože je volně dostupný, má omezené možnosti, které však pro potřeby tohoto projektu postačí.

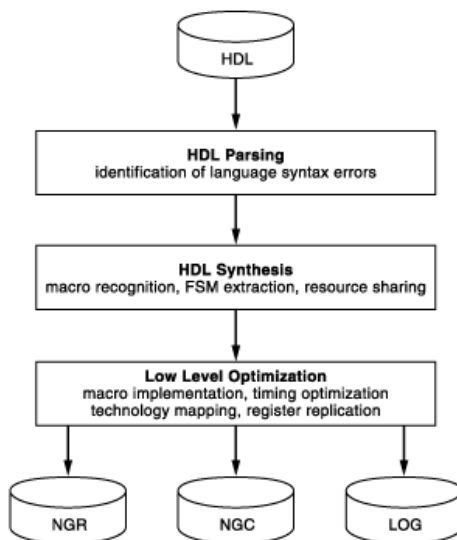
Návrh programu je možné provádět sestavením schématu z jednotlivých hradel nebo popsáním pomocí VHDL kódu. Celkový program se většinou vytváří jako hierarchická struktura a je tedy výhodné jednotlivé přístupy kombinovat. Například jednotlivé moduly v návrhu popsat VHDL kódem, vygenerovat z nich schematické symboly a ty v nejvyšší vrstvě návrhu mezi sebou propojit ve schématu, navržená struktura je potom přehlednější.

Pro syntézu kódu se používá nástroj popsáný níže v kapitole (3.1.1). Po syntéze následuje proces implementace, který se pro obvody FPGA skládá ze čtyř kroků a to z překladu (translate), mapování (map), umístění a trasování (place and route) a generování programovacího souboru (generate programming file).

V podkapitolách jsou stručně popsány nejdůležitější nástroje pro syntézu, simulaci a implementaci návrhu

3.1.1 XTS Synthesis

Nástroj určený pro syntézu kódu. Zkratka XTS znamená (Xilinx Synthesis Technology). Podporuje VHDL standardy *IEEE Std 1076-1987* a *IEEE Std 1076-1993*. Vstupem do syntetizátoru je VHDL kód, nejdůležitější výstup je soubor s příponou NGC, ve kterém je vytvořen speciální Xilinx-netlist, který je potřeba pro další zpracování při implementaci.



Obrázek 3.1: XST syntéza kódu [3]

3.1.2 PlanAhead

Dá se použít pouze při návrhu na hradlových polích typu FPGA. Slouží k plánování rozmístění signálů na jednotlivé piny pole v přehledném grafickém rozhraní. Poskytuje

nástroje pro analýzu, pomocí kterých se dají odhalit nedostatky v návrhu a zdokonalit tak vlastnosti obvodu.

3.1.3 ISE Simulator (ISim)

Je to nástroj umožňující simulovat chování a časové průběhy VHDL kódu. Může pracovat v grafickém režimu, kde zobrazuje časové průběhy signálů. Podporuje standard VHDL *IEEE Std 1076-1993*.

Testovací kódy se píšou ve vlastních souborech nazývaných „Test Bench“.

3.1.4 iMPACT

Nástroj umožňující konfiguraci zařízení a generování programovacích souborů (SVF, XSVF, ...). Umožňuje přímou konfiguraci a programování zařízení Xilinx pomocí kompatibilního kabelu. Také umožňuje zpětné čtení a ověření konfiguračních dat.

4 UART

UART je zkratka pro (Universal Asynchronous Receiver Transmitter) – univerzální asynchronní vysílač a přijímač. Jedná se tedy o zařízení pro sériovou asynchronní komunikaci, které v sobě má vysílač i přijímač. Každá UART jednotka má posuvný registr, který je základním prvkem pro převod paralelní informace na sériovou. Běžně využívá komunikační standardy RS-232, RS-422, RS-485. Pro komunikaci mezi mikroprocesory a integrovanými obvody se nejvíce využívá UART zařízení s TTL logikou, která má logické úrovně nula a jedna reprezentovány hodnotami napětí 0 V a 5 V. Pin jednotky UART vysílající data se značí jako TxD a pin přijímající data se značí jako RxD.

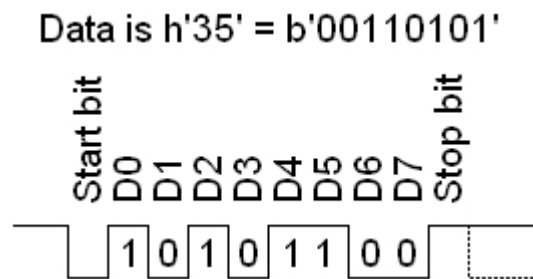
4.1 Asynchronní sériová komunikace

U asynchronního přenosu dat, nejsou přijímač a vysílač synchronizovány pomocí společného hodinového signálu. Ale synchronizace obou zařízení probíhá pokaždé, když je vyslána sekvence bitů, pomocí takzvaných start a stop bitů. Sekvence přenášených bitů se skládá právě ze start bitu následovaného sérií datových bitů (datovým slovem), za nimi může být přenesen paritní bit a celá sekvence je ukončena stop bitem.

- start bit – sériová linka je v klidovém stavu na hodnotě logické jedničky; příchodem start bitu, který je reprezentován logickou nulou, je zahájen přenos dat.

- série datových bitů (datové slovo) – může být tvořena pěti až osmi bity, protože bajt je tvořen právě osmi bity, používá se nejčastěji tento počet; jako první se odesílá nejméně významný bit
- paritní bit – používá se pro kontrolu správnosti přenesení datových bitů, paritní bit může a nemusí být v sekvenci přenášen
- stop bit – ukončuje sekvenci přenosu; je reprezentován logickou jedničkou; po jeho odeslání je možno vyslat další sekvenci; stop bit může mít šířku jednoho, dvou nebo jednoho a půl bitu

Aby bylo možné sériovou komunikaci uskutečnit, musí být na začátku nastaveny parametry pro přijímač a vysílač, jako je rychlost přenosu, šířka datového slova, šířka stop bitu a jestli bude nebo nebude přenášen paritní bit a jeho typ. Typické rychlosti přenosu sérových linek se odvíjí od násobku 300 bitů za sekundu, někdy se rychlost udává v jednotkách "baud" (při sériové asynchronní komunikaci 1 b/s = 1 baud).



Obrázek 4.1: Asynchronní sériový přenos [10]

Sériová linka může fungovat v simplexním, poloduplexním a plně duplexním režimu přenosu dat.

- simplexní (jednosměrný) režim – komunikace funguje pouze jednosměrně od vysílače k přijímači, na propojení stačí pouze dva vodiče, jeden propojuje TxD pin vysílače s RxD pinem přijímače a druhý propojuje zem.
- poloduplexní režim – obousměrná komunikace, avšak zařízení může buďto přijímat nebo vysílat, propojení je opět provedeno dvěma vodiči, to tak, že jsou nejprve mezi sebou spojeny TxD a RxD piny přijímače i vysílače a ty pak propojeny jedním vodičem mezi vysílačem a přijímačem, druhý vodič propojuje zem.
- Plně duplexní režim – obousměrná komunikace, kde zařízení může zároveň přijímat i vysílat, propojení je provedeno třemi vodiči, kde dva vodiče mezi sebou křížem propojují TxD a RxD piny vysílače a přijímače a třetí vodič propojuje zem.

Pro řízení toku dat se může použít mechanismus potvrzování (tzv. handshaking). Pokud je použit, může být buďto softwarový (XON/XOFF). Nebo hardwarový, při kterém se využívají další vodiče a jsou dva způsoby řízení (RTS/CTS, nebo DTR/DSR).

5 ETHERNET

Je to standard pro síťové technologie, popisuje požadavky na parametry propojovací kabeláže (přenosového média) a způsob přístupu k síti. Je definován normou IEEE 802.3 (literatura [13]).

Jsou různé verze Ethernetu, v této práci se budu zmiňovat pouze o dvou typech Ethernetu a to standard Ethernet (10 Mbit/s) a Fast Ethernet (100 Mbit/s). Verze se dále dělí podle způsobu propojení (podle použitého přenosového média). Nejběžněji využívané přenosové médium je kroucená dvojlinka. To odpovídá dvěma verzím, kterými se budu dále zabývat a to 10Base-T pro Ethernet a 100Base-TX pro Fast Ethernet. Další druhy přenosového média jsou koaxiální kabel (už se nepoužívá, byl pouze pro 10 Mbit/s) a optický kabel.

Kabel, kterým jsou propojena jednotlivá zařízení, je tvořen dvěma páry kroucených dvojlinek, kde jeden pár slouží pro odesílání a druhý pro příjem dat. Kroucená dvojlinka tvoří diferenciální pár, přes který jsou data přenášena. Data jsou při přenosu kódována do Manchester kódu.

Pro vysílání a přístup ke sdílenému přenosovému médiumu se využívá metoda CSMA/CD (Carrier Sense with Multiple Access and Collision Detection), je to metoda, která zajišťuje, že může vysílat pouze jedno zařízení. Při návrhu datového koncentrátoru však nebudu využívat sdílené přenosové médium.

Komunikace může probíhat v režimu half-duplex (poloviční duplex), kdy je aktivní komunikace pouze v jednom směru. Nebo full-duplex (plný duplex), kdy může zároveň probíhat vysílání i příjem dat.

5.1 Ethernetový rámec

Na Ethernetu jsou data přenášena pomocí rámců v pořadí, ve kterém je nejprve přenesen nejvíce významný oktet (oktet = skupina osmi bitů). Uvnitř oktetu je ale nejprve transportován nejméně významný bit. Uspořádání rámce je uvedeno v tabulce (tabulka 5.1). Přenášená data i s hlavičkami uvnitř rámce jsou označována jako pakety (pro spolehlivý přenos), nebo datagramy (pro nespolehlivý přenos).

V textu budou zapisovány hodnoty v hexadecimálním tvaru, kvůli rozpoznání budou zapisovány ve tvaru "0xhodnota".

Tabulka 5.1: Ethernetový rámec

preamble	SFD	MAC cíle	MAC zdroje	802.1Q tag (volitelný)	typ/délka	data	FCS	zpoždění mezi rámci
7 oktetů	1 oktet	6 oktetů	6 oktetů	4 oktety	2 oktety	(42) 46 - 1500 oktetů	4 oktety	12 oktetů
64 - 1522 oktetů								
84 - 1542 oktetů								

Popis jednotlivých polí:

- preamble - začátek rámce; slouží pro synchronizaci; přenášeny střídavě hodnoty log. 1 a log. 0 ($7 \times$ oktet – 10101010)
- SFD (Start of frame delimiter) – označení pro začátek rámce; definován formát oktetu - 10101011
- MAC cíle - MAC adresa cílového zařízení
- MAC zdroje – MAC adresa zařízení, ze kterého je rámec odeslán
- 802.1Q tag – volitelné pole; pokud je v rámci obsaženo, specifikuje informace pro virtuální LAN síť
- typ/délka – podle hodnoty v tomto poli se rozhoduje, jestli následující datové pole bude určeno pouze délkou dat, nebo typem vyššího protokolu
 - délka – pro hodnoty menší nebo rovny 1500 (0x05DC), tato hodnota udává délku následujícího datového pole (1500 je maximální délka datového pole)
 - typ – pro hodnoty větší nebo rovny 1536 (0x0600), hodnota v poli určuje typ vyššího protokolu, který bude použit v následujícím datovém poli, například hodnota 0x0800 znamená, že bude použit protokol IPv4
- data – datové pole; jeho minimální délka je 46 oktetů (42 oktetů, pokud rámec obsahuje pole 802.1Q tag); pokud chceme přenášet menší objem dat, musí být pole vyplněno do minimální délky
- FCS (Frame Check Sequence) – kontrolní pole, ve kterém se přenáší 32bitový CRC; CRC bude popsán v kapitole (5.1.2)
- zpoždění mezi rámci – před odesláním dalšího rámce musí být vložena mezera dlouhá minimálně 12 oktetů

5.1.1 IPv4 protokol

Při přenosu dat na ethernetový port budu používat protokol IPv4 (Internet Protocol version 4), takže zde uvedu jeho popis. Je to první masově rozšířená verze Internet Protokolu. Je popsán v RFC 791 (literatura [15]).

Formát IPv4 protokolu je uvedena níže v tabulce (tabulka 5.2), je tvořen hlavičkou, za kterou následují přenášená data. První řádek naznačuje rozdělení po oktetech a druhý po bitech, nejvýznamnější bit (MSB) je vlevo.

Tabulka 5.2: Formát protokolu IPv4

0								1								2								3							
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
verze				IHL				TOS								celková délka															
identifikace																příznaky		fragment offset													
TTL								protokol								kontrolní součet hlavičky															
zdrojová IP adresa																															
cílová IP adresa																															
volitelný blok (když IHL > 5)																															
data ...																															

Popis jednotlivých polí:

- verze – verze IP protokolu; pro IPv4 je to 4 (0x4)
- IHL (Internal Header Length) – délka hlavičky v bajtech, číslo je násobeno čtyřmi, minimální hodnota je 5 (0x5), to odpovídá délce hlavičky 20 bajtů ($5 \times 4 = 20$), a maximální délka je 60 bajtů (0xF)
- TOS (Type of Service) – původně mělo pole sloužit pro zvolení charakteru přepravy rámce. Nyní je nahrazeno poli s podobnou funkcí, která obsahuje informace pro službu QoS (Quality of Service), která řídí datový tok
- celková délka – délka datagramu v bajtech, to znamená délku hlavičky i s daty, která za ní následují
- identifikace – každému datagramu je přidělen identifikátor, identifikátor slouží k rozpoznání, které datagramy patří k sobě, pokud jsou data fragmentována (mají stejný identifikátor)
- příznaky – pro řízení fragmentace:
 - první bit – rezervován, musí být 0
 - druhý bit – DF (Don't Fragment), pokud je 1, data nejsou fragmentována
 - třetí bit – MF (More Fragments), pokud je 0, právě přenášený datagram je poslední fragment
- fragment offset – udává, na jakou pozici fragment patří, hodnota je udávána v počtu osmi bajtů
- TTL (Time To Live) – udává životnost datagramu, slouží jako ochrana proti zacyklení, hodnota je snižována při průchodu směrovačem, nebo každou sekundu, když hodnota dojde na nulu, přenášený datagram je zahozen
- protokol – udává, jaký vyšší protokol bude dále použit, například UDP protokol má číslo 17 (0x11).
- kontrolní součet hlavičky (checksum) – slouží pro detekci chyb, je vypočítán z dat v hlavičce, jeho výpočet bude popsán v kapitole (5.1.1.1)

- zdrojová IP adresa – IP adresa zařízení, ze kterého datagram přichází
- cílová IP adresa – IP adresa zařízení, na které datagram směřuje
- volitelný blok – obvykle není využit, nabízí další možnosti pro přenos datagramu
- data – v jakém formátu budou přenášena data v tomto poli, záleží na typu dalšího protokolu, který bude pro přenos použit (například protokoly UDP, TCP, ...)

5.1.1.1 Checksum (kontrolní součet)

Kontrolní součet se vypočítá vždy z určitých dat, slouží jako ochrana proti poškození dat při přenosu. Vždy se ověřuje jeho správnost, pokud je kontrolní součet chybný, přenášená data se zahodí.

Výpočet kontrolního součtu se provede jako jedničkový doplněk součtu 16-bitových hodnot. To znamená, že data, ze kterých je výpočet prováděn, se rozdělí do bloků po šestnácti bitech a provede se součet těchto bloků. Pokud při součtu vznikne přenos, přičte se k výsledku. Nad výsledek součtu se provede jedničkový doplněk (bitová inverze) a výsledkem této operace je požadovaný kontrolní součet.

Pokud se výpočet provede znovu se zahrnutím kontrolního součtu, měl by být výsledek roven 0. Data v takovém případě byla přenesena v pořádku.

Při výpočtu kontrolního součtu hlavičky IPv4 se výpočet provádí z celé hlavičky s tím, že pole kontrolního součtu se vyplní nulami. Příklad výpočtu bude proveden při návrhu datového koncentrátoru v kapitole (8.2.3.1).

5.1.1.2 UDP protokol

UDP (User Datagram Protocol) je jeden z rodiny protokolů TCP/IP. Je definován v RFC 768 (literatura [16]). UDP je nespojový protokol (někdy označován jako nespolehlivý), to znamená, že nenavazuje spojení mezi odesílatelem a příjemcem, a tak nezaručuje doručení jednotlivých datagramů nebo pořadí jejich doručení. Zato je jednoduchý a má nízké režijní nároky.

Formát UDP je uveden v tabulce (tabulka 5.3). První řádek naznačuje rozdělení po oktetech a druhý po bitech, nejvýznamnější bit (MSB) je vlevo.

Tabulka 5.3: Formát protokolu UDP

0								1								2								3							
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
zdrojový port (volitelný)																cílový port															
délka																kontrolní součet (volitelný)															
data ...																															

Popis jednotlivých polí:

- zdrojový port – číslo zdrojového portu; pokud není použit, měl by se nastavit na nulu
- cílový port – číslo cílového portu; jednotlivé aplikace mají přidělená svoje čísla portů, na kterých komunikují
- délka – délka UDP hlavičky a dat v bajtech
- kontrolní součet – kontrolní součet pro hlavičku a data, je volitelný, pokud ho nechceme využívat, stačí pole vyplnit nulami; počítá se z pseudo-hlavičky IPv4, kam patří zdrojová a cílová IP adresa, číslo protokolu (doplněné zleva o nuly), délka UDP a celá hlavička UDP i s daty
- data – toto pole už obsahuje uživatelská data, která chceme přenést pomocí UDP/IP protokolu

5.1.2 CRC

CRC (Cyclic Redundancy Check) je zkratka pro cylindrický redundantní součet. Je to také způsob výpočtu kontrolního součtu. Používá se pro kontrolu přenášených dat a jeho výsledek je připojen za těmito daty a přenášen společně s nimi. Vždy při přijetí dat je znovu přepočítán a v případě, že nesouhlasí, data jsou zahozena. V případě zahození dat kvůli chybnému CRC o tom není předávána žádná zpráva dále, takže není možné zjistit, že přenášený rámec byl zahozen.

Výsledný CRC se určí jako zbytek po dělení vstupních dat generačním polynomem, kde nad výslednými koeficienty zbytku je provedena operace modulo 2. Generační polynom se dá snadno převést na bitovou posloupnost. Při výpočtech CRC z bitových posloupností se využívá funkce XOR. Generační polynom může mít různou délku a různý tvar.

Výpočet se provádí ze všech polí rámce kromě preamble, SFD, FCS (do kterého je výsledek uložen) a samozřejmě zpoždění mezi rámci. Pro Ethernetové rámce se vypočítává 32-bitový cylindrický redundantní součet označován jako CRC32.

Jeho generační polynom je 32. řádu a má tvar:

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

To odpovídá bitové posloupnosti:

"1 0000 0100 1100 0001 0001 1101 1011 0111"

Příklad výpočtu pro 5-bitový CRC

Pro jednoduchost provedu příklad výpočtu pouze pro 5-bitový CRC, jako generační polynom zvolím polynom, který se používá pro CRC5-USB.

To je generační polynom: $x^5 + x^2 + 1$ (polynom 5. řádu)

Odpovídá bitové posloupnosti: 100101

Zvolíme datovou posloupnost: 10101010

Výpočet v bitové reprezentaci polynomů, využívá se operace XOR:

1 0 1 0 1 0 1 0	0 0 0 0 0	nejprve data doplníme zprava o tolik nul, jakého řádu je polynom
1 0 0 1 0 1		pod data na místě první jedničky zleva zapíšeme polynom
0 0 1 1 1 1 1 0	0 0 0 0 0	provedeme operaci XOR a zapíšeme výsledek
1 0 0 1 0 1		posuneme polynom na místo další jedničky
0 0 0 1 1 0 1 1	0 0 0 0 0	XOR
1 0 0 1 0 1	1	posun
0 0 0 0 1 0 0 1	1 0 0 0 0	XOR
1 0 0 1	0 1	posun
0 0 0 0 0 0 0 0	1 1 0 0 0	opakujeme, dokud na místě původních dat nebudou samé nuly, potom se na místě přidaných nul objeví výsledný CRC

Výsledný CRC je tedy "11000", to odpovídá polynomu $(x^4 + x^3)$. Stejný výsledek vyjde, pokud budeme počítat CRC přímo z polynomů. Kde je nejprve datový polynom vynásoben proměnnou $x^{(\text{řád polynomu})}$, v tomto případě x^5 (to je to stejné jako přidání nul na konec datové posloupnosti). Potom se tento polynom vydělí generačním polynomem a na koeficienty zbytku po dělení se aplikuje modulo 2.

5.1.2.1 Paralelní CRC

Na rozdíl od sériového způsobu výpočtu, kde se využívají posuvné registry a je možno vypočítat pouze jeden bit s každým hodinovým signálem. U paralelního výpočtu CRC je možné celý CRC vypočítat naráz. To se mi bude hodit právě při návrhu datového koncentrátoru.

Příklad výpočtu

Výpočet paralelního CRC je funkcí vstupních dat a inicializačního CRC polynomu. V první fázi je potřeba vytvořit předpis, podle kterého budou vypočítány jednotlivé bity CRC součtu. Tato fáze záleží pouze na bitové šířce dat a na šířce a typu zvoleného CRC. V druhé fázi je vytvořen tento předpis a už se do něj jenom dosazují konkrétní data, ze kterých chceme CRC vypočítat.

Uvedu zde příklad pro vypočítání paralelního CRC se stejnými parametry jako v předcházející kapitole. Tedy generační polynom CRC5-USB ($x^5 + x^2 + 1$) a datová posloupnost "10101010".

První fáze:

- 1) Určíme si šířku dat a šířku CRC. Data si označíme jako D a jejich šířka je 8 (D = 8). Šířka CRC je 5, inicializační CRC polynom označíme jako M (M = 5).

- 2) Vypočítáme si sériový CRC pro D hodnot, když $M = 0$. Kdy každá hodnota D je zapsána v kódu nazývaném one-hot (to znamená, že je pouze jeden bit nastaven na hodnotu 1), pro D(x) to budou hodnoty 0000 0001, 0000 0010, ..., 1000 000.

příklady výpočtu pro některé hodnoty:

D (0)	D (1)	D (3)
0 0 0 0 0 0 0 1	0 0 0 0 0 0 1 0	0 0 0 0 1 0 0 0
1 0 0 1 0 1	1 0 0 1 0 1	1 0 0 1 0 1
0 0 0 1 0 1	0 0 1 0 1 0	0 0 0 1 0 1 0 0
		1 0 0 1 0 1
		0 0 0 0 0 1 1 0 1

- 3) Z jednotlivých výsledků v předchozím bodě sestavíme tabulku (tabulka 5.4), kde $CRC(x)$ je funkcí D. Řazení výsledků je jasné z tabulky.

Tabulka 5.4: První tabulka pro sestavení paralelního CRC

	CRC (4)	CRC (3)	CRC (2)	CRC (1)	CRC (0)
D (0)	0	0	1	0	1
D (1)	0	1	0	1	0
D (2)	1	0	1	0	0
D (3)	0	1	1	0	1
D (4)	1	1	0	1	0
D (5)	1	0	0	0	1
D (6)	0	0	1	1	1
D (7)	0	1	1	1	0

- 4) Vypočítáme si sériový CRC pro M hodnot, když $D = 0$. Kdy každá hodnota M je opět zapsána v one-hot kódu. Protože CRC polynom je kratší než data ($M < D$), budou se výsledky opakovat a je zbytečné je všechny počítat znovu. Pro tento příklad $D(3) \approx M(0)$, $D(4) \approx M(1)$, ...
- 5) Sestavení tabulky (tabulka 5.5) z výsledků v předchozím bodě, kde $CRC(x)$ je funkcí M.

Tabulka 5.5: Druhá tabulka pro sestavení paralelního CRC

	CRC (4)	CRC (3)	CRC (2)	CRC (1)	CRC (0)
M (0)	0	1	1	0	1
M (1)	1	1	0	1	0
M (2)	1	0	0	0	1
M (3)	0	0	1	1	1
M (4)	0	1	1	1	0

Druhá fáze:

Z tabulek vytvořených v první fázi sestavíme předpis pro výpočet paralelního CRC. CRC(x) je funkcí M a D, takže pro jednotlivé CRC(x) vybereme z tabulek (tabulka 5.4 a tabulka 5.5) řádky, ve kterých je hodnota 1 a mezi jednotlivými členy provádíme operaci XOR. Operace XOR je značena jako '^'.

$$\text{CRC}(0) = D(0) \wedge D(3) \wedge D(5) \wedge D(6) \wedge M(0) \wedge M(2) \wedge M(3)$$

$$\text{CRC}(1) = D(1) \wedge D(4) \wedge D(6) \wedge D(7) \wedge M(1) \wedge M(3) \wedge M(4)$$

$$\text{CRC}(2) = D(0) \wedge D(2) \wedge D(3) \wedge D(6) \wedge D(7) \wedge M(0) \wedge M(3) \wedge M(4)$$

$$\text{CRC}(3) = D(1) \wedge D(3) \wedge D(4) \wedge D(7) \wedge M(0) \wedge M(1) \wedge M(4)$$

$$\text{CRC}(4) = D(2) \wedge D(4) \wedge D(5) \wedge M(1) \wedge M(2)$$

Nyní je vytvořen předpis, do kterého stačí už jen dosazovat příslušné hodnoty, a tak získáme výsledný CRC.

Dosazení konkrétních dat ("10101010"), za podmínky $M = 0$. Data jsou reprezentována tak, že nejvýznamnější bit je vlevo (má index 7):

$$\text{CRC}(0) = 0 \wedge 1 \wedge 1 \wedge 0 = 0$$

$$\text{CRC}(1) = 1 \wedge 0 \wedge 0 \wedge 1 = 0$$

$$\text{CRC}(2) = 0 \wedge 0 \wedge 1 \wedge 0 \wedge 1 = 0$$

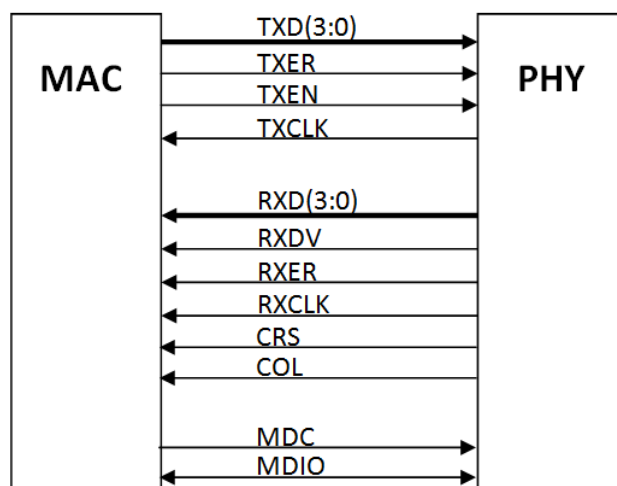
$$\text{CRC}(3) = 1 \wedge 1 \wedge 0 \wedge 1 = 1$$

$$\text{CRC}(4) = 0 \wedge 0 \wedge 1 = 1$$

Konečný výsledek: CRC = "11000". To je stejný výsledek jako v případě výpočtu sériového CRC v kapitole (5.1.2).

5.2 MII

MII (Media Independent Interface) je zkratka pro nezávislé rozhraní na typu použitého média a signálů. Poskytuje propojení mezi MAC (Media Access Control) a PHY (Physical Layer) a díky nezávislosti tohoto rozhraní je možné propojit různé typy zařízení. Rozhraní je definováno pro obě verze Ethernetu, jak pro 10 Mbit/s, tak pro 100 Mbit/s. Data jsou přes MII přenášena čtyřmi datovými vodiči v každém směru jako čtyřbitová slova. Čtyřbitové slovo se označuje jako „nibble“.



Obrázek 5.1: Propojení MII rozhraní

Vodiče rozhraní MII se dají rozdělit do tří základních skupin:

1) Signály pro odesílání dat (do PHY):

- TXD0 – odesílaný datový bit 0 (odesílán jako první)
- TXD1 – odesílaný datový bit 1
- TXD2 – odesílaný datový bit 2
- TXD3 – odesílaný datový bit 3
- TXEN (transmit enable) – signalizace, že data na vodičích TXD(0:3) mají být přenesena
- TXER (transmit error) – signalizuje chybu při přenosu a pokud je aktivní, data na vodičích TXD(0:3) nejsou přenesena; přítomnost tohoto signálu je volitelná a využívá se jenom zřídka
- TXCLK (transmit clock) – hodinový signál poskytující časování pro příjem dat; pokud je aktivní signál TXEN, tak s náběžnou hranou TXCLK jsou přenášena data z TXD(0:3); jedním hodinovým taktem jsou tedy přeneseny 4bity (nibble); zdrojem hodinového signálu je PHY; hodinový signál má frekvenci 25 MHz pro rychlost 100 Mb/s, nebo 2,5 MHz pro 10 Mb/s

2) Signály pro příjem dat (z PHY):

- RXD0 – přijímaný datový bit 0 (přijímaný jako první)
- RXD1 – přijímaný datový bit 1
- RXD2 – přijímaný datový bit 2
- RXD3 – přijímaný datový bit 3
- RXDV (receive data valid) – signalizuje, že na vodičích RXD(0:3) jsou připravena data pro příjem

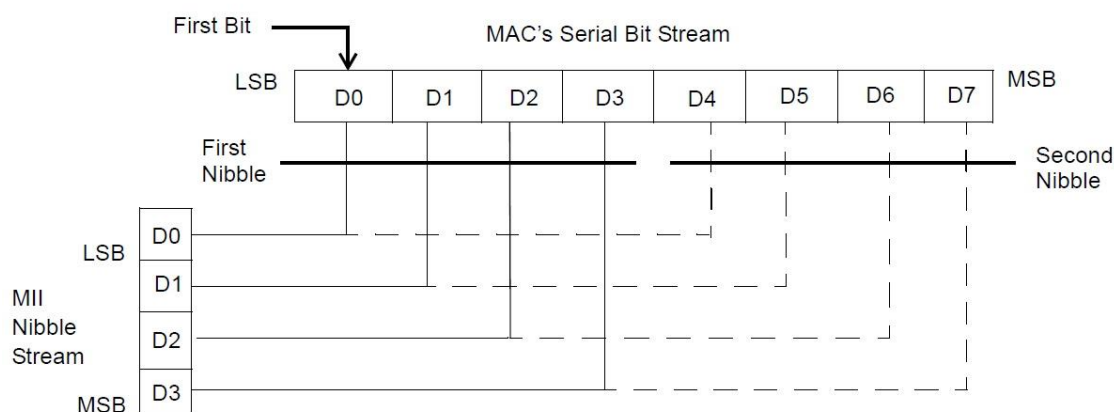
- RXER (receive error) – signalizuje chybu při odesílání dat
- CRS (carrier sense) – v half-duplex módu signalizuje, že probíhá odesílání nebo příjem dat; pokud je nastaven režim full-duplex, tak jeho chování není v normě specifikováno
- COL (collision detected) – signalizuje, jestli došlo ke kolizi (pouze half-duplex) a v případě rychlosti přenosu 10 Mb/s reaguje na SQE (signal quality error)
- RXCLK – hodinový signál poskytující časování pro odesílání dat; pokud je aktivní signál RXDV, tak s náběžnou hranou RXCLK jsou přenášena data na RXD(0:3); zdrojem hodinového signálu je PHY; hodinový signál má frekvenci 25 MHz pro rychlost 100 Mb/s, nebo 2,5 MHz pro 10 Mb/s

3) Signály pro konfiguraci PHY:

- MDC (management data clock) – externí hodinový signál (musí být přiveden do PHY) poskytuje časování pro MDIO
- MDIO (management data input/output) – obousměrný signál pro přenos dat sloužících k nastavení vlastností PHY

Způsob přenosu dat přes MII rozhraní

Každý oktet je přenesen za pomoci dvou čtyřbitových slov ($2 \times \text{nibble}$). Data jsou přenášena od nejméně významného bitu (LSB – least significant bit), jako poslední je přenesen nejvýznamnější bit (MSB – most significant bit). Způsob přenosu je znázorněn na obrázku (5.2).

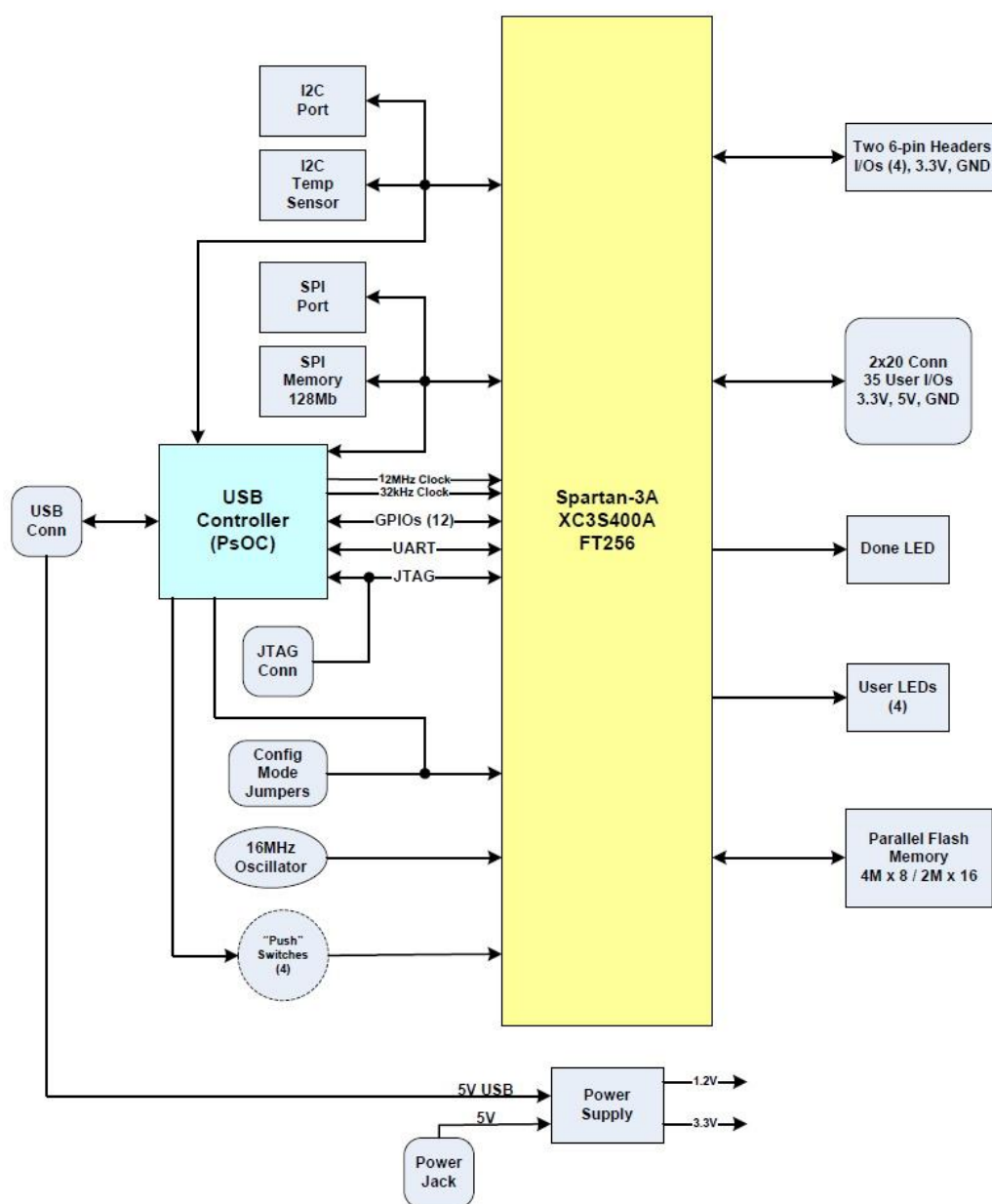


Obrázek 5.2: Způsob přenosu dat přes MII rozhraní [13]

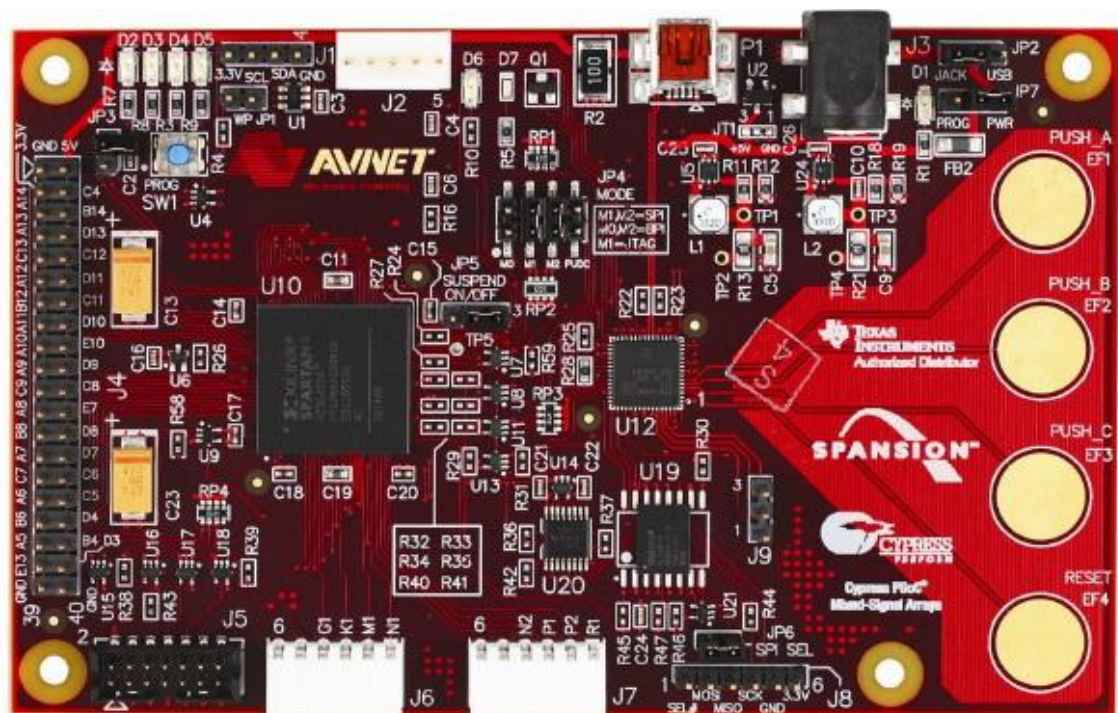
6 HARDWARE POUŽITÝ PRO NÁVRH

6.1 Spartan-3A Evaluation Kit

Vývojová deska, kde nejdůležitějšími součástmi jsou hradlové pole Xilinx Spartan-3A a obvod PSoC od firmy Cypress. Deska je napájena napětím 5 V buďto z USB, nebo z 2mm napájecího konektoru. Lze programovat pomocí několika rozhraní, mezi něž patří i konektor JTAG, který zároveň umožňuje zpětnou diagnostiku. Na desce je vyvedeno 35 vstupně/výstupních uživatelských pinů. Všechny součásti, které vývojová deska nabízí, jsou zobrazeny v blokovém schématu na obrázku (obrázek 6.1).



Obrázek 6.1: Blokové schéma Spartan-3A Evaluation Kit [11]



Obrázek 6.2: Spartan-3A Evaluation Kit [11]

6.1.1 Xilinx Spartan-3A

Přesné označení je XC3S400A-4FTG256C. Kde XC3S400A je typ zařízení, -4 je rychlost obvodu (standardní), FTG256 značí typ pouzdra a počet pinů (pouzdro FTBGA) a C označuje rozsah teplot (0 °C – 80 °C).

Je to hradlové pole typu FPGA (Field Programmable Gate Array). Obsahuje 400 000 logických hradel sdružených do 896 programovatelných logických bloků (CLB - Configurable Logic Block). Každý CLB je tvořen čtyřmi logickými řezy ("slice"). Piny slouží jako rozhraní s obvodem, rozhraní podporuje nejpoužívanější logické standardy jako LVCMOC, LVTTL, HSTL, SSTL, a proto také využívá i různé úrovně napětí, a to 3.3 V, 2.5 V, 1.8 V, 1.5 V a 1.2 V. Nejvyšší dovolený odběr proudu na pinu je 24 mA. Detailnější popis je možno najít v odkazu (literatura [4]).

6.1.2 Cypress PSoC

PSoC (programmable system on chip), na desce je umístěn typ CY8C24894. Jedná se o čip, ve kterém je kombinace osmibitového procesoru se systémem konfigurovatelných analogových a digitálních bloků. Detailní popis je možno najít v odkazu (literatura [7]).

6.2 Programovací kabel JTAG HS2

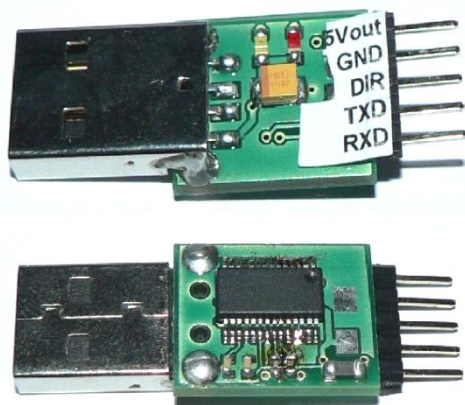
Tento kabel je plně kompatibilní s nástroji od firmy Xilinx i s vývojovou deskou popsanou v kapitole (5.1). K počítači se připojuje přes USB port a na desku přes konektor JTAG. Pomocí tohoto kabelu je bez problému možno naprogramovat vývojovou desku prostřednictvím nástroje iMPACT popsaného v kapitole (3.1.4). Díky plné kompatibilitě je možné z prostředí iMPACT kapitola (3.1.4) snadno nastavit parametry a nahrát program do vývojové desky.



Obrázek 6.3: Programovací kabel JTAG HS-2 [5]

6.3 Převodník USB na UART

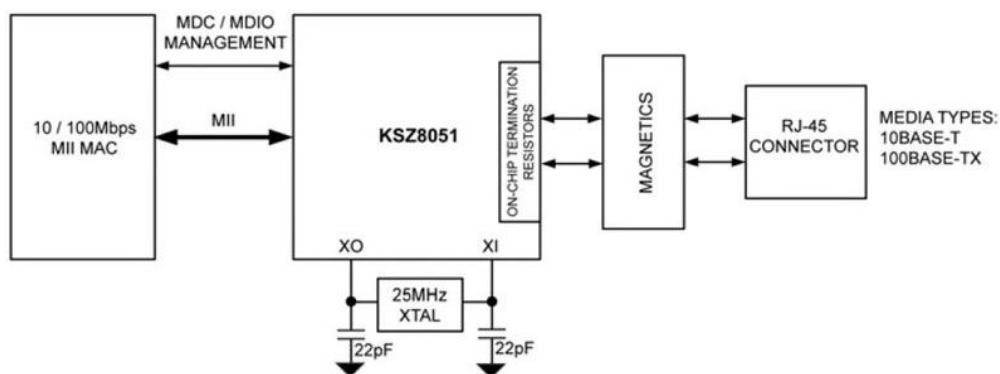
Pro realizaci datového koncentrátoru na vývojové desce potřebuji přijímat data ze sériové linky. Protože klasický RS232 sériový port z počítačů mizí a stejně tak chybí i u mého notebooku, potřebuji tento převodník na vysílání dat. Převodník pracuje v logice TTL s úrovní napětí 5 V, zatímco deska má úroveň napětí v TTL logice 3.3 V, z tohoto důvodu je na vedení od pinu TxD převodníku umístěn odpor, který hodnotu napětí snižuje (převodník je prozatím využíván pouze v simplexním módu). Převodník je realizován na základě čipu FT232LR od firmy FTDI, má rychlost přenosu od 300 baud do 3 Mbaud, podporuje 7, nebo 8 bitový přenos s šířkou stop bitu 1 nebo 2 bity a možnost přenosu paritního bitu.



Obrázek 6.4: Převodník USB na UART

6.4 Ethernetový modul

Ethernetový modul je postaven na PHY čipu KSZ8051MLL a podporuje standardy 10BASE-T a 100BASE-TX. Pro komunikaci nabízí rozhraní MII. Má možnost automatické konfigurace (Auto-negotiation) pro nastavení duplexu a nejvyšší rychlosti. A má funkci HP Auto MDI/MDI-X, která slouží pro rozpoznání, zda je připojen přímým nebo kříženým kabelem a v případě potřeby automaticky překříží přijímací a odesílající diferenciální páry.

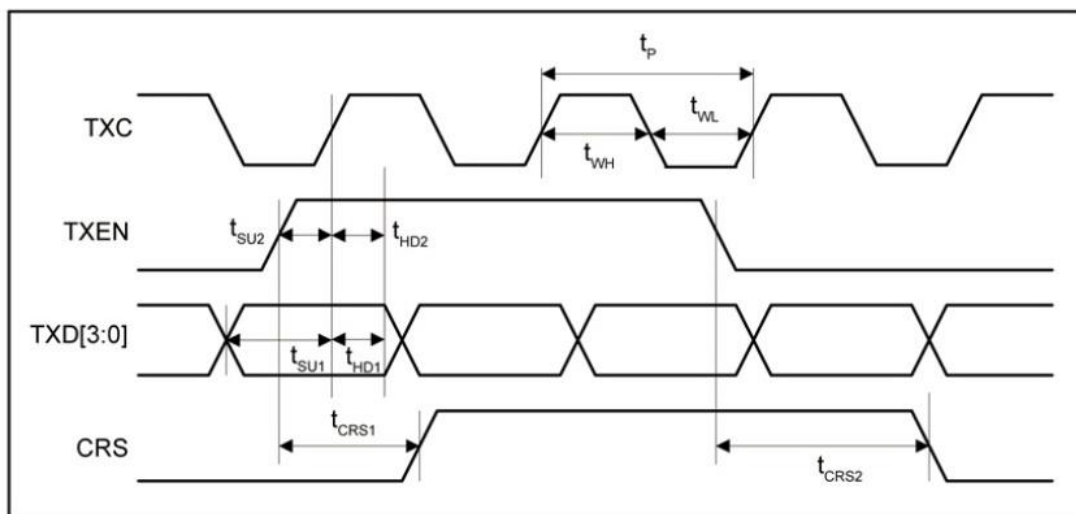


Obrázek 6.5: Blokové schéma Ethernetového modulu [14]



Obrázek 6.6: Ethernetový modul

Časování signálů pro odesílání dat modulem je uvedeno na následujícím obrázku a v tabulkách pod ním. Sloupce "min." v tabulkách znamenají minimální potřebný čas a sloupce "typ." vyjadřují typické časy při přenosu.



Obrázek 6.7: Časový diagram pro Ethernetový modul [14]

Tabulka 6.1: Časové parametry pro rychlost 10Mb/s (vztahují se k obrázku 6.7)

časový parametr	popis	min. [ns]	typ. [ns]
t_P	perioda TxC		400
t_{WL}	šířka pulsu TxC v 0		200
t_{WH}	šířka pulsu TxC v 1		200
t_{SU1}	setup time TXD(3:0) před náběžnou hranou TxC	120	
t_{SU2}	setup time TXEN před náběžnou hranou TxC	120	
t_{HD1}	hold time TXD(3:0) po náběžné hraně TxC	0	
t_{HD2}	hold time TXEN po náběžné hraně TxC	0	
t_{CRS1}	náběžná hrana CRS po náběžné hraně TXEN		200
t_{CRS2}	sestupná hrana CRS po sestupné hraně TXEN		550

Tabulka 6.2: Časové parametry pro rychlost 100Mb/s (vztahují se k obrázku 6.7)

časový parametr	popis	min. [ns]	typ. [ns]
t_P	perioda TxC		40
t_{WL}	šířka pulsu TxC v 0		20
t_{WH}	šířka pulsu TxC v 1		20
t_{SU1}	setup time TXD(3:0) před náběžnou hranou TxC	10	
t_{SU2}	setup time TXEN před náběžnou hranou TxC	10	
t_{HD1}	hold time TXD(3:0) po náběžné hraně TxC	0	
t_{HD2}	hold time TXEN po náběžné hraně TxC	0	
t_{CRS1}	náběžná hrana CRS po náběžné hraně TXEN		35
t_{CRS2}	sestupná hrana CRS po sestupné hraně TXEN		36

7 PRVNÍ PROGRAMY

Pro seznámení s programovacím prostředím, způsobem sestavování programu a jeho přenesení na vývojovou desku, jsem začal s primitivními programy ve schematickém režimu, kde jsem si vyzkoušel práci s kapacitními tlačítky, diodami a uživatelskými piny. Jako první zde představím program pro generování signálu o frekvenci 1 Hz. Generátor jsem vytvořil jak sestavením z hradel ve schematickém režimu, taky i jeho popsáním VHDL kódem. Oba návrhy zde představím pro znázornění představy o tom, jaké řešení je výhodnější.

Sestavování programů ve schématu je výhodnější pouze pro jednoduché programy, při realizaci složitějších programů se schémata stávají nepřehlednými a i menší úprava zabere hodně času. Zatímco pokud použijeme jazyk VHDL je úprava jednoduchá a lépe se popisují složitější řešení. Nejlepší způsob je vytvářet programy kombinací obou přístupů. Kombinaci obou přístupů budu používat k vytvoření datového koncentrátoru v následující kapitole (kapitola 8).

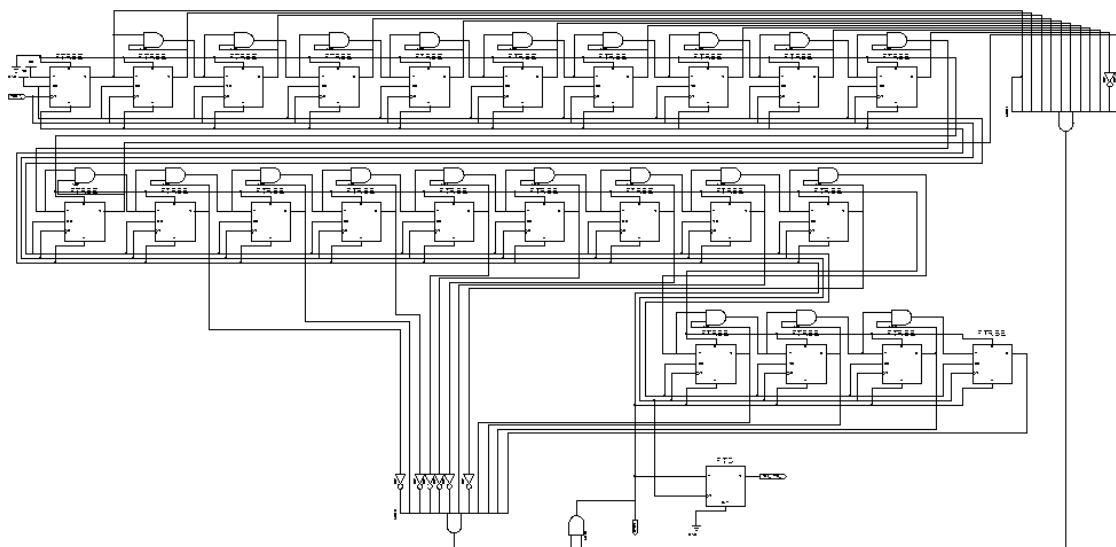
7.1 Vytvoření pomocí schématu

Frekvenci 1 Hz získávám podělením hodinového signálu s frekvencí 16 MHz. Pro dělení signálu využívám synchronní čítače, sestavené z T klopných obvodů, kde každý čítač vydělí frekvenci dvěma, a tak vývody z čítačů reprezentují zápis čísla v binárním kódu (vývod nejvzdálenějšího čítače představuje bit s nevyšším významem). Pokud chci dosáhnout frekvence 1 Hz, musím napočítat přesný počet impulsů a překlomit signál.

Pro získání generátoru signálu o frekvenci 1 Hz musím během jedné vteřiny vygenerovat dvakrát signál reset, který vynuluje čítače a zároveň překlomí T klopný obvod, na jehož výstupu je generován právě požadovaný hodinový signál. To znamená, že při využití hodinového signálu s frekvencí 16 MHz, musím napočítat 8 000 000 impulsů a vyvolat reset.

Protože používám synchronní čítače se synchronním resetem, musím napočítat o jeden impuls méně, než je přesný počet pro dosažení požadované frekvence, to znamená, že musím generovat signál pro reset při 7 999 999 impulsích, aby právě s příchodem příštího impulsu byly čítače vynulovány a výstupní hodinový signál překlomen. Jelikož na výstupech z čítačů je číslo přímo reprezentováno v binárním kódu, mohu snadno pomocí hradel AND a invertorů přivést správnou kombinaci výstupů jako signál reset. Pro reprezentaci čísla 7 999 999 v binárním kódu musím použít 23 článků čítače.

$$7\,999\,999_D = 111\,1010\,0001\,0001\,1111\,1111_B$$



Obrázek 7.1: Schéma hodinového generátoru 1Hz

Výsledný signál jsem pro ověření změřil pomocí osciloskopu, průběh je zobrazen na obrázku (obrázek 7.2).

7.2 Vytvoření pomocí VHDL kódu

Pro naprogramování stejné funkce už byla práce (po seznámení se základy programování ve VHDL) jednodušší. Opět jsem potřeboval překlopit signál dvakrát v průběhu jedné sekundy, abych vytvořil hodinový generátor s frekvencí 1Hz. Pro tento účel jsem si v procesu vytvořil dvě proměnné, jednu pro počítání impulsů (proměnná i) a druhou pro uložení dočasné hodnoty signálu (Q_t), která je na konci procesu vždy přiřazena do výstupního signálu. Obě proměnné jsou na začátku inicializovány na hodnotu nula. Při každé náběžné hraně vstupního hodinového signálu o frekvenci 16 MHz je proměnná i inkrementována a při dosažení hodnoty 8 000 000 (odpovídá času 500 ms) je výstupní signál nastaven do logické jedničky, inkrementace pokračuje dále a při hodnotě 16 000 000 je výstupní signál nastaven do logické nuly, proměnná i je vynulována a s dalším impulsem se proměnná opět inkrementuje.

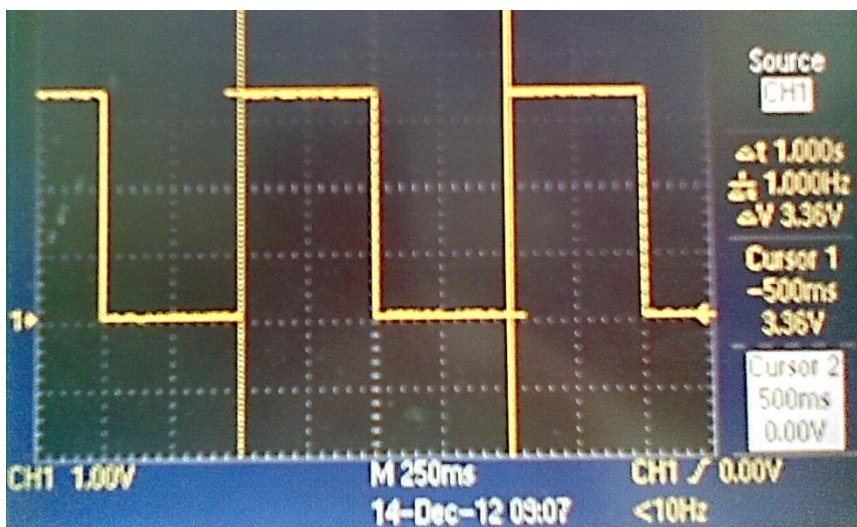
VHDL kód:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity delicka16M is
    Port ( C : in  STD_LOGIC;
          Q : out STD_LOGIC);
end delicka16M;

architecture Behavioral of delicka16M is
begin
    delicka: process (C)
        variable i: integer :=0;
        variable Qt: std_logic := '0';
    begin
        if C = '1' and C'event then
            i := i+1;
            if i = 8E6 then
                Qt := '1';
            elsif i = 16E6 then
                Qt := '0';
                i := 0;
            end if;
        end if;
        Q <= Qt;
    end process delicka;
end Behavioral;
```

Výsledný průběh signálu byl naprosto stejný jako v předešlém případě, kdy jsem návrh prováděl sestavením schématu. Výsledný průběh naměřený osciloskopem je zobrazen na obrázku dole (obrázek 7.2).



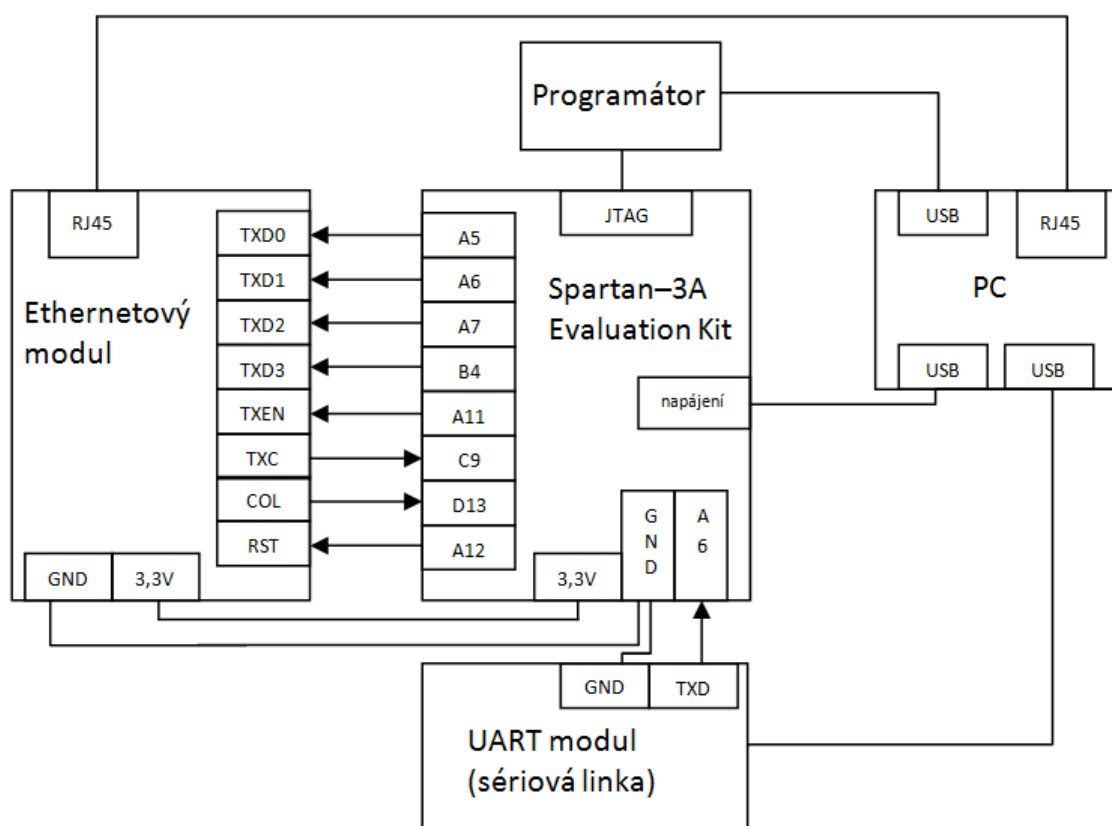
Obrázek 7.2: Naměřený průběh z navržených generátorů

8 NÁVRH DATOVÉHO KONCENTRÁTORU

V této kapitole představím svůj návrh datového koncentrátoru a aplikaci pro otestování jeho funkčnosti. Návrh jednotlivých bloků koncentrátoru je proveden v jazyce VHDL. Bloky jsou mezi sebou propojeny ve schématu. Pro návrh využiji veškeré znalosti získané a popsané v předešlých kapitolách.

Návrh má tři hlavní části, kde první část zajišťuje zachycení dat ze sériové linky a jejich paralelní reprezentaci pro přenos do druhé části, kde druhá část zajišťuje odeslání dat na Ethernetový port protokolem UDP/IP. Ve třetí části je vytvořen program v jazyce C# pro otestování funkčnosti celého návrhu.

Na obrázku (obrázek 8.1) je schéma propojení hardwaru při návrhu koncentrátoru i se znázorněním připojení jednotlivých signálů na konkrétní piny vývojové desky Spartan-3A Evaluation Kit. Propojení signálů je pro přehlednost uvedeno i v tabulce (tabulka 8.1). Všechny vstupní/výstupní piny desky jsou nastaveny na standard LVTTTL.



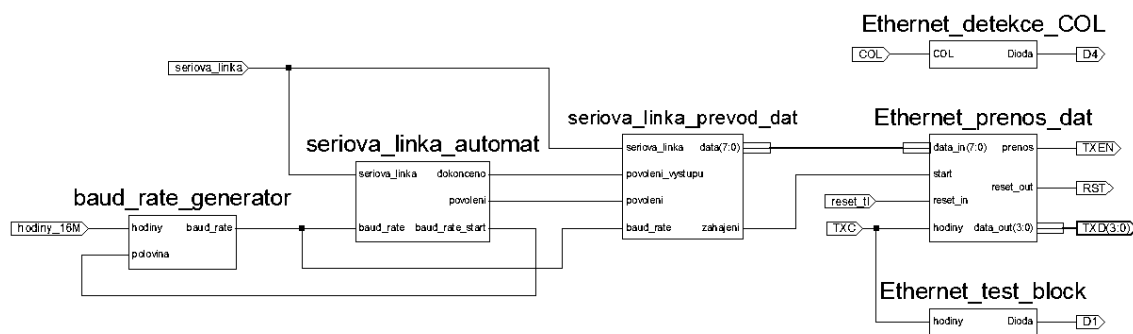
Obrázek 8.1: Schéma propojení hardwaru při návrhu

Tabulka 8.1: Tabulka propojení signálů

označení signálů		označení pinů
Ethernetový modul	sériová linka	vývojová deska
TXD0		A5
TXD1		A6
TXD2		A7
TXD3		B4
TXEN		A11
TXC		C9
COL		D13
RST		A12
+3,3V		3,3V
GND		GND
	TXD	A6
	GND	GND

Na obrázku (obrázek 8.2) je softwarové schéma, které zobrazuje propojení jednotlivých bloků vytvořených v jazyce VHDL. Toto schéma zajišťuje přenos dat ze sériové linky na Ethernet. Uvedu zde stručný popis funkce schématu, podrobněji budou jednotlivé bloky rozebrány v následujících kapitolách. První blok (v pořadí zleva) se stará o generování vzorkovací frekvence pro sériovou linku. Druhý blok zajišťuje řízení prvního a třetího bloku na základě signálu ze sériové linky, kde při příchodu start bitu odesílá signál do prvního bloku a zajišťuje tím synchronizaci. Třetí blok je řízen na základě signálů z druhého bloku a s náběžnými hranami signálu "baud_rate" vzorkuje datové bity ze sériové linky a ukládá si je do vnitřních signálů v bloku (8-bitový posuvný registr), aby je na výstupu mohl reprezentovat v paralelním uspořádání. Signálem "zahajeni" informuje o dostupnosti datového bajtu na výstupech "data(7:0)". Výstupy z třetího bloku vstupují do bloku, který se stará o přenos dat na Ethernet přes rozhraní MII. Bloky "Ethernet_detekce_COL" a "Ethernet_test_blok" mají doplňkové funkce hlavně pro vizuální kontrolu a výstupy zobrazují na LED diodách umístěných přímo na vývojové desce.

Pokud by bylo pro komunikaci použito více sériových linek, tak by se první tři bloky daly opětovně využít pro převod informací (sériové na paralelní) z více sériových linek současně.



Obrázek 8.2: Softwarové schéma propojení bloků

8.1 Sériová linka (Návrh sériové linky)

Návrh bloku pro dekódování dat ze sériové linky je proveden pro následující nastavení sériové linky:

Rychlost: 115 200 baud (maximální standardní rychlost sériové linky)

Počet datových bitů: 8

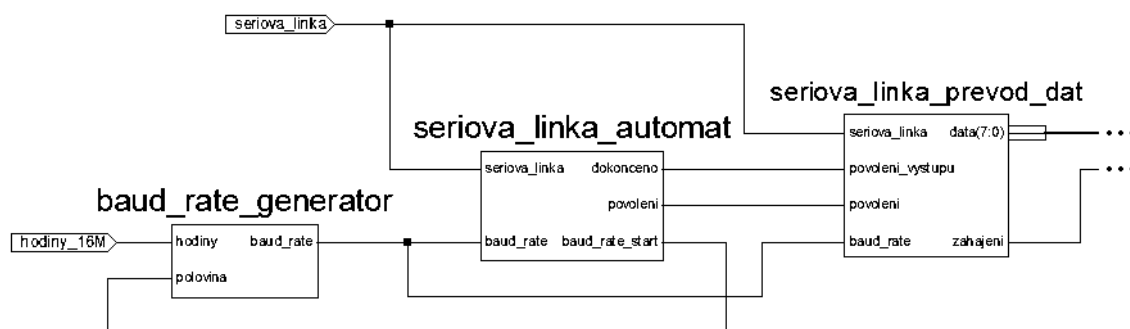
Parita: žádná

Stop bit: jeden (vzhledem ke způsobu návrhu mohou být i dva)

Potvrzování (handshaking): žádné

Pro převod dat ze sériové linky na paralelní informaci slouží tři bloky popsané ve VHDL kódu. Jsou vzájemně propojeny ve schématu způsobem, jaký zobrazuje následující obrázek (obrázek 8.3). Detailnější popisy jednotlivých bloků budou uvedeny v samostatných kapitolách.

Vstupy do této části schématu jsou 16 MHz hodinový signál a vodič ze sériové linky vysílající data (TxD). Výstupy jsou paralelně reprezentované bity získané z přenosu bajtu ze sériové linky a signál informující o tom, že bity už jsou dostupné na výstupech "data(7:0)". Tyto výstupy jsou propojeny s dalším blokem, který se stará o přenos dat na rozhraní Ethernet, tento blok bude popsán v kapitole (8.2).



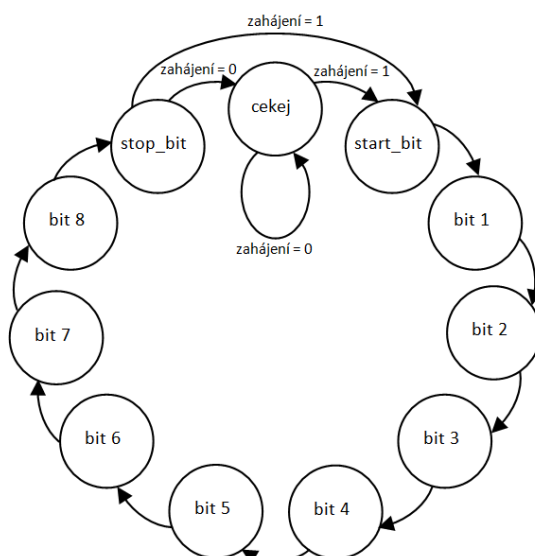
Obrázek 8.3: Propojení bloků pro převod sériové informace na paralelní

8.1.1 Popis bloku (baud_rate_generator)

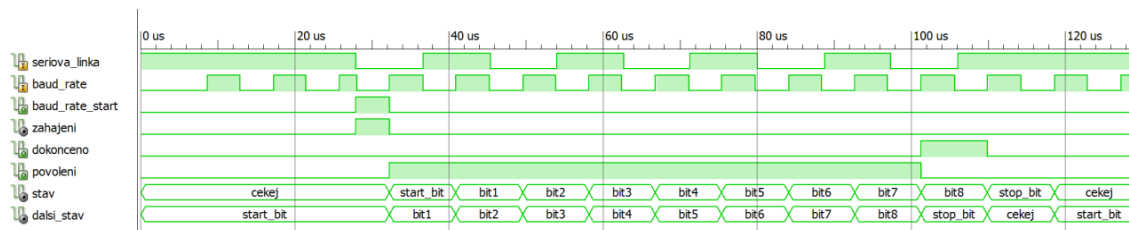
Tento blok vytváří signál, jehož nástupnými hranami budou vzorkována data přicházející po sériové lince a zajišťuje, aby vzorkování probíhalo blízko středu právě přenášeného bitu. Vstupem do bloku je hodinový signál o frekvenci 16 MHz. Zdrojem tohoto signálu je přímo hradlové pole. Signál je podělen hodnotou 138, což odpovídá vzorkovací frekvenci přibližně 115 942 Hz (to je přibližně perioda 8,625 μ s). Frekvence sériové linky je 115 200 Hz (perioda přibližně 8,681 μ s). Rozdíl mezi periodami je 56 ns, to znamená, že při sekvenci 10 bitů se hrana pro vzorkování posune o 0,56 μ s od středu bitu. Protože při příchodu každého start bitu je vzorkovací hrana synchronizována na střed bitu přicházejícího po sériové lince, tak vzorkování proběhne vždy na úrovni správného bitu. Synchronizace na střed je řízena vstupem "polovina", kdy při příchodu log. 1 je nastavena proměnná počítající impulsy na hodnotu odpovídající polovině její maximální hodnoty.

8.1.2 Popis bloku (seriova_linka_automat)

Do tohoto bloku vede signál ze sériové linky a s příchodem start bitu (sestupné hrany) je vygenerován signál "baud_rate_start", který vede do předchozího bloku a zajišťuje synchronizaci vzorkovacího signálu. Průběh signálu "baud_rate_start" je kopií proměnné "zahájení", která slouží uvnitř bloku jako signalizace, že je na sériové lince přenášena nová sekvence bitů. V tomto bloku je vytvořen stavový automat, který má jedenáct stavů (cekej, start_bit, bit1, bit2, bit3, bit4, bit5, bit6, bit7, bit8, stop_bit), které slouží pro rozpoznání, jaký bit je na sériové lince zrovna přenášen. Pokud neprobíhá žádná komunikace, automat se nachází ve stavu "cekej". Grafické znázornění stavového automatu je na obrázku (obrázek 8.4).



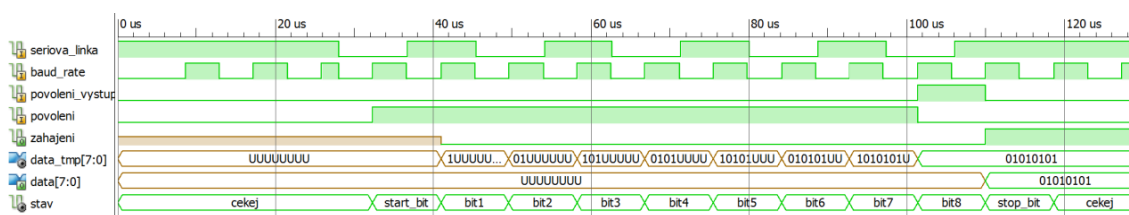
Obrázek 8.4: Stavový automat pro sériovou linku



Obrázek 8.5: Simulace pro blok - seriova_linka_automat

8.1.3 Popis bloku (seriova_linka_prevod_dat)

Tento blok slouží pro převod bitů ze sériové reprezentace na paralelní. Je řízen signály ze stavového automatu (předchozí blok). Pokud je aktivní signál "povoleni", jsou data na sériové lince s náběžnou hranou vzorkována a postupně ukládána do vnitřních signálů "data_tmp(7:0)". Při přijmutí všech datových bitů přejde signál "povoleni" do neaktivního stavu, signál "povoleni_vystup" do aktivního, s příští náběžnou hranou signálu "baud_rate" jsou data z vnitřních signálů "data_tmp(7:0)" přepsána na výstupy "data(7:0)" a je aktivován výstupní signál "zahajeni", který signalizuje, že právě na výstupech "data(7:0)" je připravena paralelní sekvence bitů.



Obrázek 8.6: Simulace pro blok - seriova_linka_prevod_dat

8.2 Ethernet (Návrh Ethernetu)

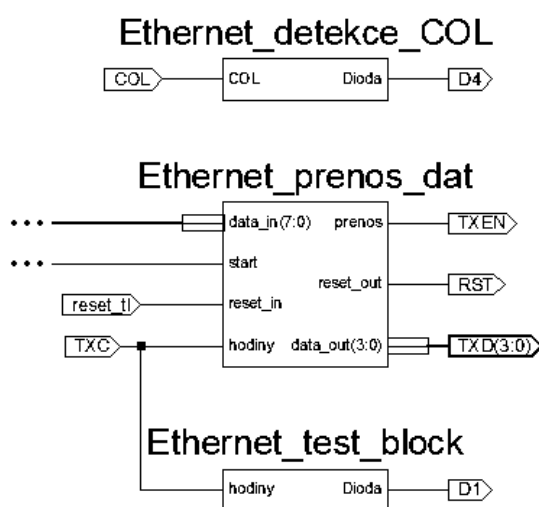
Tato část zajišťuje odeslání datového bajtu protokolem UDP/IP na Ethernetový port přes rozhraní MII. Odesílaný bajt vstupuje do bloku jako paralelní sekvence bitů.

Vzhledem ke stejnému způsobu odesílání dat při rychlosti 10 Mb/s i 100 Mb/s přes rozhraní MII funguje návrh pro oba standardy. Návrh je proveden pro full-duplex režim.

V každém Ethernetovém rámci je přenášen jeden bajt přijatý ze sériové linky. Každý bajt je okamžitě po přijmutí ze sériové linky přenesen na Ethernetový port. Sériová linka má rychlost 115 200 bit/s. Při nastavení z kapitoly (8.1) musí pro přenos jednoho bajtu přenést minimálně 10 bitů, to znamená maximální přenosovou rychlost 11 520 bajtů/s. Pro odeslání jednoho bajtu na Ethernet způsobem popsáným výše je zapotřebí odeslat jeden rámec, který přenáší s veškerou rezií 672 bitů, to pro rychlost 10Mbit/s znamená přenosovou rychlost 14 880 bajtů/s. Takže způsob přímého přenosu jednoho bajtu ze sériové linky jedním Ethernetovým rámcem je možný ($14\,880 > 11\,520$).

Pro nastavení Ethernetového modulu využívám možnost automatického nastavení (auto-negotiation proces). Takže rychlost a typ přenosu nastavím pouze u síťové karty v PC a modul se automaticky nakonfiguruje na stejné parametry.

Ethernetový rámec je na úrovni MAC adres odesílán jako broadcast z důvodu, aby tento rámec mohla přijmout všechna zařízení. Ale na úrovni IP adres už je odesílán na konkrétní adresu a to 192.168.0.4. Protože nepotřebuji, aby cílové zařízení odesílalo nějaké rámce zpět na Ethernetový modul, tak jsem mohl zvolit zdrojovou MAC adresu téměř jakoukoliv. Podmínkou je pouze dodržení, že první bit ve zdrojové MAC adrese musí být nastaven jako nula. Proto jsem zvolil zdrojovou MAC adresu reprezentovanou samými nulami. Jako zdrojová IP adresa bude odesílána adresa 192.168.0.44.



Obrázek 8.7: Zapojení bloků pro Ethernetový přenos

8.2.1 Popis bloku (Ethernet_test_blok)

V tomto bloku je pouze frekvence vydělena 2,5 milióny a vyvedena na diodu označenou na vývojové desce jako D5. Dioda slouží pouze pro vizuální kontrolu hodinového signálu TXD a podle frekvence blikání se dá rozpoznat, v jakém režimu pracuje Ethernetový modul. Frekvence je vydělena hodnotou právě 2,5 miliónů, protože jsem návrh nejprve prováděl při rychlosti 10 Mb/s (frekvence hodinového signálu 2,5 MHz). To znamená, že dioda při této rychlosti bliká s frekvencí 1 Hz, při rychlosti 100 Mb/s (hodinový signál 25 MHz) bliká s frekvencí 10 Hz. Tyto dvě frekvence se od sebe dají snadno rozpoznat okem.

8.2.2 Popis bloku (Ethernet_detekce_COL)

Tento blok zobrazuje stav signálu COL na diodě označené na vývojové desce jako D2 (hodnota $\log.1 \approx$ dioda svítí). Protože modul bude nastaven do full-duplex režimu, tak signalizace na diodě má význam pouze pro rychlost 10 Mbit/s. Pokud je hodnota na vodiči TXEN ve stavu $\log.1$ déle než 20 ms, nastal tzv. stav "jabber", COL přejde do úrovně $\log.1$ a přenos je zablokován. Stav jabber znamená, že nastal abnormálně dlouhý přenos a došlo k porušení běžných podmínek při přenosu. Tento stav je detekován pomocí jabber funkce. Pokud svítí dioda D2, přenos na modulu je blokován.

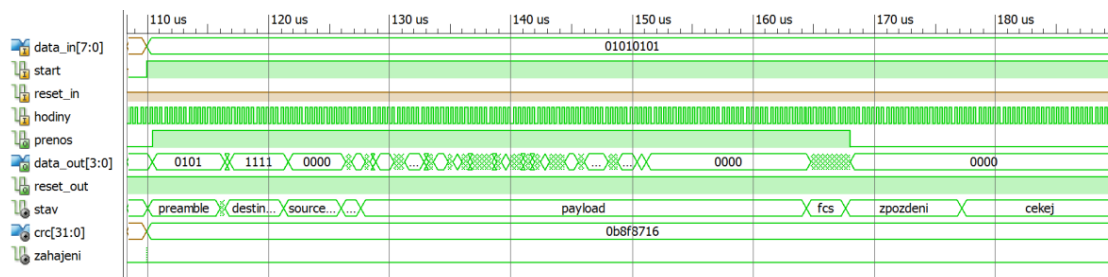
8.2.3 Popis bloku (Ethernet_prenos_dat)

Jak už bylo několikrát zmíněno, data jsou po Ethernetu přenášena protokolem UDP/IP. Tento blok se stará o takovýto přenos s využitím MII rozhraní, přes které je připojen na Ethernetový modul.

Protože mi stačí data přenášet pouze jedním směrem, a to z vývojové desky do Ethernetového modulu (z MAC do PHY), vystačím si s vodiči TXC, TXD0, TXD1, TXD2, TXD3 a TXEN. Význam jednotlivých vodičů je vysvětlen v kapitole (5.2).

Vstupy do bloku jsou paralelně reprezentovány bity ze sériové linky pomocí osmi signálů "data_in(7:0)". Další vstup je signál "start", který informuje o tom, že na vstupu jsou připravena data pro přenos a s náběžnou hranou signálu "start" je zahájen přenos Ethernetového rámce, ve kterém budou tyto data obsažena. Mezi vstupy patří i tlačítko pro reset "reset_tl", které nastaví stavový automat uvnitř bloku do výchozího stavu "cekej" a na výstupu "RST" vygeneruje úroveň $\log.0$, která resetuje Ethernetový modul. Poslední vstup je hodinový signál "TXC".

Uvnitř bloku je vytvořen stavový automat se stavy (cekej, preamble, SFD, destination_MAC, source_MAC, Ethernet_type, payload, FCS a zpoždění), které zajišťují odeslání dat jednotlivých polí Ethernetového rámce. Přesun mezi stavy závisí na předchozím stavu a na hodnotě proměnné "citac", ta je inkrementována s každou sestupnou hranou hodinového signálu TXC a při přechodu do dalšího stavu je nulována. Při přechodu do nového stavu jsou nejprve první sestupnou hranou generována poslední data ze stavu, ve kterém se právě nachází a zároveň nastaven nový stav. Teprve až následující sestupnou hranou jsou na výstupy generována data z nového stavu. Proto je při simulaci vidět, že je generování dat zpožděno za přechodem do nového stavu. Podle hodnoty čítače jsou v každém stavu na výstupy "TXD(3:0)" a "TXEN" generovány příslušné hodnoty signálů zajišťující odesílání dat Ethernetového rámce. Data jsou na výstupy z bloku generována vždy se sestupnou hranou hodinového signálu TXC, aby s jeho náběžnou hranou mohla být vzorkována a přenášena Ethernetovým modulem. Nastavením dat na výstupy při sestupné hraně hodinového signálu jsou splněny požadavky modulu na setup a hold time signálů při vzorkování.



Obrázek 8.8: Simulace pro blok - Ethernet_prenos_dat

Odesílání Ethernetového rámce je zahájeno nastavením signálu TXEN do hodnoty log. 1 a synchronizací hodin ve stavu "preamble" (pole preamble v Ethernetovém rámci). Po synchronizaci se přejde do stavu "SFD", kde po odeslání bitové posloupnosti "1011" začne odesílání MAC adresy cílového zařízení.

Data jsou přes MII rozhraní odesílána v pořadí, kdy je nejdříve odeslán nejméně významný bit, proto se nejprve musí přenést spodní nibble (4 bity) oktetu.

- MAC adresa cíle (destination_MAC) – je nastavena jako broadcast (0xFF:FF:FF:FF:FF:FF)
- MAC adresa zdroje (source_MAC) – nastavena na hodnotu samých nul
- typ vyššího protokolu (Ethernet_type) – IPv4 (hodnota 0x0800)
- pole data (payload) – bude popsáno v kapitole (8.2.3.1)
- FCS – bude popsáno v kapitole (8.2.3.2)
- zpoždění mezi rámci (zpozdeni) – hodnota signálu TXEN je nastavena do log.0 a je vytvořena mezera pro odeslání 12 oktetů (24 period hodinového signálu TXC)

8.2.3.1 Hodnoty přenášené v datové části rámce (poli data)

Zde uvedu konkrétní hodnoty přenášené v datové části rámce. V předcházejícím poli je uvedena hodnota, která říká, že bude použit protokol IPv4. V hlavičce IPv4 bude jako další protokol zvolen UDP. Konkrétní dosazené hodnoty do protokolů jsou uvedeny v tabulkách níže (tabulka 8.1 a tabulka 8.2).

Je nastavena minimální velikost pro toto pole, a to je 46 bajtů (20 bajtů má hlavička IPv4 a 26 bajtů má UDP protokol i s přenášenými uživatelskými daty).

Tabulka 8.2: Konkrétní dosazené hodnoty v protokolu IPv4

0								1								2								3								
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
verze (0x4)				IHL (0x5)				TOS (0x00)								celková délka (0x002E ≈ 46 bajtů)																
identifikace (0x9999)																příznaky			fragment offset													
TTL (0x80)				protokol (0x11) ≈ 17 (UDP)								kontrolní součet hlavičky (0xDFA4)																				
zdrojová IP adresa (C0.A8.00.2C ≈ 192.168.0.44)																																
cílová IP adresa (C0.A8.00.04 ≈ 192.168.0.4)																																
zde začíná UDP protokol																																

Kontrolní součet pro hlavičku IPv4 bude v každém přenášeném rámci stejný. Příklad výpočtu v hexadecimálním tvaru pro konkrétní data v hlavičce:

- 1) $4500 + 002E + 9999 + 4000 + 8011 + 0000 + C0A8 + 002C + C0A8 + 0004 = 32058$
- 2) $3 + 2058 = 205B$
- 3) $\sim 205B = DFA4$

Kontrola:

- 1) $4500 + 002E + 9999 + 4000 + 8011 + DFA4 + C0A8 + 002C + C0A8 + 0004 = 3FFFC$
- 2) $3 + FFFC = FFFF$
- 3) $\sim FFFF = 0000$

Tabulka 8.3: Konkrétní dosazené hodnoty v protokolu UDP

0								1								2								3							
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
zdrojový port (0x0400) ≈ 1024																cílový port (0x0400) ≈ 1024															
délka (0x001A) ≈ 26 bajtů																kontrolní součet (0x0000)															
<i>bajt ze sériové linky</i>								0x00								0x00								0x00							
0x00								0x00								0x00								0x00							
0x00								0x00								0x00								0x00							
0x00								0x00								0x00								0x00							
0x00								0x00								0x00								0x00							

Kontrolní součet pro UDP protokol není povinný, takže toto pole je vyplněno nulami. V prvním oktetu sloužícím pro přenos uživatelských dat je přenášen bajt, který byl přijat ze sériové linky. Zbývající oktety jsou vyplněny hodnotami 0, aby byla splněna minimální délka datagramu.

8.2.3.2 Způsob výpočtu CRC přenášeného v poli FCS

Pro výpočet CRC v hradlovém poli používám paralelní způsob výpočtu, který je vysvětlen a uveden na příkladu v kapitole (5.1.2.1). Vzhledem k tomu, že data, ze kterých se musí 32-bitový CRC vypočítat, mají délku 480 bitů, nechal jsem si předpis pro paralelní výpočet CRC vygenerovat programem staženým z webových stránek uvedených v literatuře (literatura [12]). Program je přímo dostupný ke stažení na stránkách (literatura [24]).

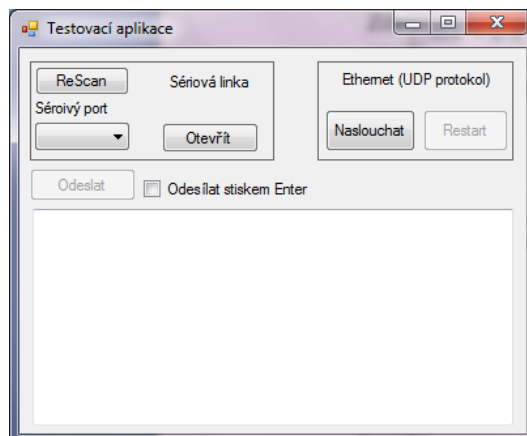
Protože ve všech rámcích se při přenosu mění pouze jeden oktet (bajt přijatý ze sériové linky), tak se výpočet CRC provádí vždy při změně dat na vstupu do bloku (vstupy "data_in(7:0)").

Aby výpočet CRC odpovídal normě IEEE 802.3, musí být pořadí bitů v každém oktetu invertováno a inicializační CRC polynom je třeba nastavit na hodnotu 0xFFFFFFFF. Potom takto připravená data vstupují do předpisu pro paralelní výpočet CRC a po jeho provedení musejí být jednotlivé oktety opět invertovány. Na závěr musí být provedena bitová inverze (vypočítán první doplněk) těchto dat a výsledek je požadovaný CRC, který se přenáší v rámci.

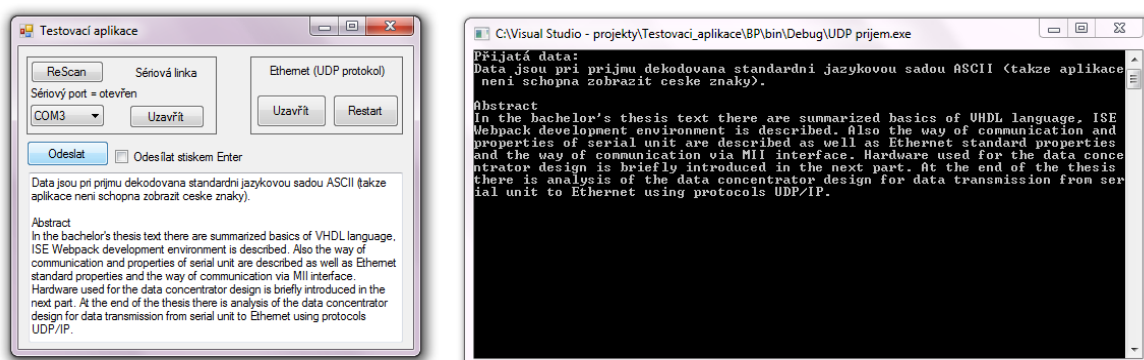
8.3 Aplikace pro otestování funkčnosti

Pro otestování funkčnosti navrženého datového koncentrátoru jsem vytvořil dvě aplikace v jazyce C#. Jedna je konzolová ("UDP prijem.exe") a slouží pro příjem UDP datagramů z Ethernetu na portu 1024. Data jsou při příjmu dekodována standardní jazykovou sadou ASCII (aplikace není schopna zobrazit české znaky). Druhá aplikace je formulářová ("Testovací_aplikace.exe") a slouží pro otevření sériového portu a odesílání dat přes tento sériový port, zároveň se z ní dá spustit první aplikace pro naslouchání na Ethernetu (pokud je ve stejném adresáři).

Formulářová aplikace je na obrázku (8.9). Při startu aplikace jsou skenovány aktivní sériové porty a jejich seznam je vypsán v rolovací nabídce. Tlačítko "ReScan" slouží pro nové skenování portů a jejich výpis do rolovací nabídky pod tímto tlačítkem. Tlačítko "Odeslat" se stane aktivní při otevření sériového portu a jeho stisknutím se na sériový port odešle text napsaný v textovém poli. Spuštění konzolové aplikace se provede stiskem tlačítka "Naslouchat", vedle něj je tlačítko "Restart", které konzolovou aplikaci zavře a znovu spustí. Při otevření sériového portu nebo konzolové aplikace tlačítka změní svoji funkci a název a slouží k uzavření.



Obrázek 8.9: Testovací aplikace



Obrázek 8.10: Zobrazení přenosu dat s využitím aplikací

9 VÝPISY Z NÁVRHU

Rozpis obsazení částí hradlového pole:

Počet zabraných logických řezů ("slice"):	251 z 3584	7%
Počet zabraných klopných obvodů:	138 z 7168	1%
Počet zabraných 4-vstupých LUT (lookup table):	468 z 7168	6%
Počet zabraných vstupů/výstupů:	13	
Počet zabraných vstupních/výstupních bloků :	13 z 195	6%
Počet zabraných GCLK (globální hodinový blok):	4 z 24	16%

Výpis časování v hradlovém poli:

Minimální perioda:	10,846 ns (to odpovídá maximální frekvenci 92,204 MHz)
Minimální čas příchodu signálu pře hodinovým signálem:	5,988 ns
Maximální čas nastavení výstupního signálu po hodinovém signálu:	5,271 ns
Maximální čas zpoždění pro kombinační logiku:	7,168 ns

10 ZÁVĚR

V práci jsou shrnuty informace získané při vytváření datového koncentrátoru, které jsou nutné pro jeho úspěšný návrh, jako je popis jazyka VHDL, ve kterém jsou popsány jednotlivé bloky vytvořeného datového koncentrátoru. Popis vývojového prostředí ISE Design Suite 14.2, ve kterém je zajištěna syntéza kódu a jeho implementace ve vývojové desce pomocí programátoru připojeného na desku přes JTAG rozhraní. Jako další je v práci popsán způsob asynchronního přenosu dat po sériové lince, standard Ethernet, jednotlivé protokoly (IPv4 a UDP) a způsob přenosu dat přes rozhraní MII. Dalším tématem je hardware, se kterým bude návrh datového koncentrátoru realizován. V závěrečné části jsou popsány parametry a samotný návrh datového koncentrátoru.

Datový koncentrátor je jednosměrný a přenáší data ze sériové linky na Ethernet. Vstupující sériová linka musí mít nastavenou rychlost 115 200 baud (bit/s), přenos osmi datových bitů, žádnou paritu a na délce stop bitu nezáleží. Hodnota 115 200 baud je nejvyšší standardní rychlost pro sériové linky. Ethernetový modul se dá nastavit pomocí automatické konfigurace (auto-negotiation proces). Musí být nastaven do režimu full-duplex, ale může pracovat při obou rychlostech přenosu, jak při rychlosti 10 Mb/s, tak při rychlosti 100 Mb/s. V jakém režimu modul pracuje, se dá rozeznat přímo na vývojové desce, a to na frekvenci blikání diody označené D5. Pro režim 10 Mb/s bliká s frekvencí 1 Hz a pro režim 100 Mb/s bliká s frekvencí 10 Hz. Pokud je modul v módu 10 Mb/s, detekuje stav jabber (zaseknutí modulu) a tento stav je zobrazen na diodě D2 (pokud svítí, přenos modulu je blokován).

Pro otestování přenosu dat jsou vytvořeny dvě aplikace v jazyce C#, kde jedna je konzolová a naslouchá na portu 1024 pro UDP. Kvůli univerzálnosti dekoduje data podle jazykové sady ASCII, takže správně zobrazuje jenom znaky patřící do této sady. Druhá aplikace je formulářová, slouží pro otevření sériového portu a odeslání dat, zároveň se z ní dá spustit i první konzolová aplikace.

Způsob přenosu dat probíhá tak, že každý bajt je přenášen v samostatném Ethernetovém rámci. Okamžitě po přijmutí bajtu ze sériové linky je na Ethernet odeslán rámec s tímto bajtem. Odesílání rámců na úrovni MAC adresy proběhne na všechna zařízení (MAC adresa je broadcast). Na úrovni IP adresy je už v hlavičce protokolu nastavena konkrétní adresa, takže připojené zařízení musí mít nastavenou IP adresu 192.168.0.4.

Vytvořený datový koncentrátor zvládne bez chyby přenést 700 těsně za sebou navazujících bajtů ze sériové linky. Vytvořený návrh zabírá asi 7% plochy hradlového pole Spartan-3A. Pole dokáže pracovat s maximální frekvencí asi 92 MHz.

Literatura

- [1] ASHENDEN, Peter J. *The VHDL Cookbook*. South Australia: University of Adelaide, 1990.
- [2] PINKER, Jiří a Martin POUPA. *Číslicové systémy a jazyk VHDL*. Praha: BEN - technická literatura, 2006, 349 s. ISBN 80-730-0198-5.
- [3] ISE Help. XILINX. *All Programmable Technologies from Xilinx Inc.* [online]. © 2012 [cit. 2013-01-03]. Dostupné z: http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_2/isehelp_start.htm
- [4] XILINX. *Spartan-3A FPGA Family: Data Sheet*. [online]. 19 August 2010. [cit. 2013-01-03]. Dostupné z: http://www.xilinx.com/support/documentation/data_sheets/ds529.pdf
- [5] DIGILENT. *JTAG-HS2: Programming Cable for Xilinx FPGAs*. [online]. July 24, 2012. [cit. 2013-01-03]. Dostupné z: http://www.digilentinc.com/Data/Products/JTAG-HS2/JTAG-HS2_rm.pdf
- [6] FTDI. *FT232R USB UART IC Datasheet*. [online]. © 2010. [cit. 2013-01-03] Dostupné z: http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT232R.pdf
- [7] CYPRESS. *CY8C24894*. [online]. June 2, 2011. [cit. 2013-01-03]. Dostupné z: <http://www.cypress.com/?docID=29949>
- [8] Asynchronous serial communication. *Wikipedia, the free encyclopedia* [online]. 26 June 2012 [cit. 2013-01-03]. Dostupné z: http://en.wikipedia.org/wiki/Asynchronous_serial_communication
- [9] Universal asynchronous receiver/transmitter. *Wikipedia, the free encyclopedia* [online]. 3 January 2013 [cit. 2013-01-03]. Dostupné z: http://en.wikipedia.org/wiki/Universal_asynchronous_receiver/transmitter
- [10] USART. *Wikipedie, otevřená encyklopedie* [online]. 30. 5. 2012 [cit. 2013-01-03]. Dostupné z: <http://cs.wikipedia.org/wiki/USART>
- [11] AVNET. *Xilinx Spartan-3A Evaluation Kit: User Guide*. Rev. 2.1. 11.20.2008. Dostupné z: http://www.files.em.avnet.com/files/177/xlx_s3a_evl_ug_rev2_112008.pdf
- [12] OUTPUTLOGIC.COM. *OutputLogic.com* [online]. © 2009-2011 [cit. 2013-05-20]. Dostupné z: <http://outputlogic.com/>
- [13] IEEE Std 802.3. *IEEE 802.3™: Ethernet*. New York: IEEE, 26. December 2008.
- [14] MICREL. *KSZ8051MLL: 10BASE-T/100BASE-TX Physical Layer Transceiver* [online]. July 2010 [cit. 2013-05-20]. Dostupné z: <http://www.micrel.com/PDF/Ethernet/datasheets/ksz8051mll.pdf>
- [15] *RFC 791: INTERNET PROTOCOL* [online]. September 1981 [cit. 2013-05-20]. Dostupné z: <http://tools.ietf.org/html/rfc791#section-2.1>
- [16] *RFC 768: User Datagram Protocol* [online]. 28 August 1980 [cit. 2013-05-20]. Dostupné z: <https://tools.ietf.org/html/rfc768>
- [17] Ethernet frame. In: *Wikipedia: the free encyclopedia* [online]. 13 May 2013 [cit. 2013-05-20]. Dostupné z: https://en.wikipedia.org/wiki/Ethernet_frame#cite_note-1

- [18] Cyclic redundancy check. In: *Wikipedia: the free encyclopedia* [online]. 17 May 2013 [cit. 2013-05-20]. Dostupné z: http://en.wikipedia.org/wiki/Cyclic_redundancy_check
- [19] Cyklický redundantní součet. In: *Wikipedie: otevřené encyklopedie* [online]. 19. 5. 2013 [cit. 2013-05-20]. Dostupné z: http://cs.wikipedia.org/wiki/Cyklick%C3%BD_redundantn%C3%AD_sou%C4%8Det
- [20] IPv4. In: *Wikipedia: the free encyclopedia* [online]. 19 May 2013 [cit. 2013-05-20]. Dostupné z: <https://en.wikipedia.org/wiki/IPv4>
- [21] User Datagram Protocol. In: *Wikipedia: the free encyclopedia* [online]. 10 May 2013 [cit. 2013-05-20]. Dostupné z: https://en.wikipedia.org/wiki/User_Datagram_Protocol
- [22] IPv4 header checksum. In: *Wikipedia: the free encyclopedia* [online]. 7 February 2013 [cit. 2013-05-20]. Dostupné z: https://en.wikipedia.org/wiki/IPv4_header_checksum
- [23] Media Independent Interface. In: *Wikipedia: the free encyclopedia* [online]. 17 April 2013 [cit. 2013-05-20]. Dostupné z: http://en.wikipedia.org/wiki/Media_Independent_Interface
- [24] Parallel CRC Generator :: Overview :: OpenCores. *OpenCores* [online]. © copyright 1999-2013 [cit. 2013-05-21]. Dostupné z: <http://opencores.org/project,parallelcrcgen>

Seznam zkratek

VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuits
IEEE	Institute of Electrical and Electronics Engineers
FPGA	Field-Programmable Gate Array
PLD	Programmable Logic Device
CLB	Configurable Logic Block
XTS	Xilinx Synthesis Technology
SFV	Serial Vector Format
XSFV	Xilinx Serial Vector Format
UART	Universal Asynchronous Receiver Transmitter
CSMA/CD	Carrier Sense with Multiple Access and Collision Detection
SFD	Start of frame delimiter
FCS	Frame Check Sequence
MAC	Media Access Control
PHY	Physical Layer
RFC	Request For Comments
MSB	Most Significant Bit
LSB	Least Significant Bit
IPv4	Internet Protocol version 4
UDP	User Datagram Protocol
CRC	Cyclic Redundancy Check
MII	Media Independent Interface
JTAG	Joint Test Action Group

Seznam příloh

Příloha 1. CD s elektronickou verzí bakalářské práce

Příloha 1. CD s elektronickou verzí bakalářské práce

||elektronická verze práce

||projekt (návrh koncentrátoru)

||projekty (testovací aplikace)

||spustitelné aplikace