České
vysoké
učení technické
v Praze

FEL

Fakulta elektrotechnická
Katedra měření

Monitoring komunikace v síti ethernet

Filip Valenta

Vedoucí: doc. Ing. Jiří Novák Ph.D.
Květen 2019

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Valenta** Jméno: **Filip** Osobní číslo: **405401**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra měření**

Studijní program: **Kybernetika a robotika**

Studijní obor: **Letecké a kosmické systémy**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Monitoring komunikace v síti Ethernet

Název diplomové práce anglicky:

Ethernet Communication Monitoring

Pokyny pro vypracování:

Navrhněte a implementujte funkční vzor zařízení pro monitoring komunikace v síti 100Base-T1. Postupujte v následujících krocích:
1. Seznamte se podrobně s technologií 100Base-T1, zejména s funkcionalitou fyzické vrstvy komunikace.
2. Navrhněte koncepci zařízení pro pasivní monitoring komunikace v jednom segmentu sítě prostřednictvím rozhraní Gb Ethernetu.
3. Realizujte hardware funkčního vzoru, preferujte variantu využívající FPGA kit.
4. V jazyce VHDL implementujte programové vybavení funkčního vzoru a otestujte jeho funkci.

Seznam doporučené literatury:

[1] Bučkovskij, D.: Využití sítí Ethernet v osobních automobilech, bakalářská práce ČVUT FEL, Praha 2016
[2] Correa, C.: Automotive Ethernet - The Definitive Guide, Interpids Control Systems 2014, ISBN: 978-0990538806
[3] Yanick, M.E.: Avionics full duplex switched ethernet (AFDX) data bus, Technická zpráva, 2007

Jméno a pracoviště vedoucí(ho) diplomové práce:

doc. Ing. Jiří Novák, Ph.D., K 13138 - katedra měření

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **09.10.2018** Termín odevzdání diplomové práce: _____

Platnost zadání diplomové práce:
do konce zimního semestru 2019/2020

doc. Ing. Jiří Novák, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

Datum převzetí zadání _____ Podpis studenta _____

Poděkování

Děkuji svému vedoucímu doc. Ing. Jiřímu Novákovi, Ph.D. za cenné rady, pomoc a čas, který mi věnoval při psaní této diplomové práce.

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně. Dále prohlašuji, že jsem všechny použité zdroje správně a úplně citoval a uvádím je v přiloženém seznamu použité literatury.

Nemám závažný důvod proti zpřístupňování této závěrečné práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) v platném znění.

V Praze, 24. května 2019

Abstrakt

V práci se zabývám technologií 100Base-T1 a s ní související Broad-R-Reach. Tyto technologie se začínají používat v automobilovém průmyslu místo technologie CAN, která již mnohým současným systémům nevyhovuje. 100Base-T1 se od klasického Ethernetu (např.1000Base-T) liší především na fyzické vrstvě. V práci proto popisuji rozdíly fyzické vrstvě oproti klasickému Ethernetu. V rámci teoretické části se okrajově věnuji také technologii AFDX, která je určena pro letecký průmysl.

V praktické části práce jsem osadil navrženou desku plošných spojů (PCB) a následně programoval kód, který zajišťuje monitorování komunikace mezi dvěma zřízeními. Tento kód jsem programoval v jazyce VHDL, který je určený pro hradlová pole. S vyrobenou deskou používám kit FPGA. Tato mnou vyrobená deska funguje jako opakovač, který propojuje dvě zařízení a přeposílá mezi nimi data. Zároveň tyto data posílá Gb rozhraním, které slouží k monitorování komunikace.

Klíčová slova: Automotive Ethernet, Broad-R-Reach, 802.3bw, 100Base-T1, MII, VHDL, AFDX

Vedoucí: doc. Ing. Jiří Novák Ph.D.

Abstract

In my thesis I am focused to 100Base-T1 and Broad-R-Reach technology which are connected to each other. These technologies are starting to be used in automotive industry instead of CAN technology, which often does not meet current needs. 100Base-T1 differs from classic Ethernet (eg 1000Base-T) mainly on the physical layer. Therefore, I describe differences of physical layer compared to classical Ethernet. In the theoretical part I also marginally deal with AFDX technology, which is intended for the aerospace industry.

In the practical part of the thesis I mounted a components to the designed PCB and then programmed the code which ensures the monitoring of communication between the two devices. I programmed this code in VHDL, which is designed for gate arrays. I use FPGA kit with the board. This board works as a repeater that connects two devices and forwards data between them. At the same time, it sends this data via the Gb interface, which is used to monitor communication.

Keywords: Automotive Ethernet, Broad-R-Reach, 802.3bw, 100Base-T1, MII, VHDL, AFDX

Title translation: Ethernet Communication Monitoring

Obsah

Obrázky

Tabulky

Kapitola 1

Úvod

Ethernet se v dnes používá na celém světě. Může se proto zdát, že téma Monitoring komunikace v síti Ethernet již není v dnešní době ničím zajímavé. Ethernet je standard, který můžeme najít v každé domácnosti, v každé kanceláři nebo kavárně. Již na začátku devadesátých let minulého století se stal Ethernet nejpoužívanější kabelovou technologií pro lokální sítě. Od té doby se k internetu přes kabel jinak než pomocí Ethernetu téměř nepřipojíme. Právě díky tomuto standardu je tak snadné a levné se připojit k internetu kdekoliv na světě a nepotřebujeme k tomu žádná speciální zařízení. Kde ale standard Ethernet ještě není samozřejmý, je letecký a automobilový průmysl.

V současnosti nejrozšířenější standard CAN je na hranici svých možností. Pro nynější vývoj technologií a datových přenosů je pomalý a příliš těžký. V době, kdy jsou automobily a letadla přeplněná elektronikou, senzorikou a hlavně kabeláží, je potřeba, aby se informace po kabelech sířily co nejrychleji a abychom toho dosáhli s co možná nejmenší hmotností kabeláže.

V mé práci se budu zabývat technologií 100Base-T1(a BroadR-Reach) a jejím použitím v automotive. V porovnání s klasickým Ethernetem (např.1000Base-T) se tento standard liší hlavně na fyzické vrstvě OSI/ISO modelu, které se budu z tohoto důvodu věnovat nejvíce. Protože se varianta 100Base-T1 používá v automobilech, v jedné z kapitol se věnuji také standardu AFDX [1] a staršímu standardu ARINC-429, které jsou vyvíjené a používáné v letectví.

V praktické části práce budu navrhovat a realizovat funkční vzor zařízení pro monitoring komunikace na základě technologie 100Base-T1, která se v současnosti začíná v automobilech používat. Návrh tohoto zařízení probíhal v programu KiCad. Následně probíhalo osazení a testování funkčního vzoru. Poté byla pro tento funkční vzor vytvořena programová realizace opakovače a monitoringu posílaných dat. Veškerá programovací část je vytvořena v jazyce VHDL s pomocí programu Quartus a simulačního prostředí ModelSim.

[1] Avionics Full-Duplex Switched Ethernet

Část I

Teoretická část

Kapitola 2

LAN

LAN označuje lokální počítačovou síť, která obsahuje pouze malý počet zařízení a rozkládá se na malém území (např. jedna budova firmy, automobil, letadlo). Tato síť funguje obvykle na základě standardu Ethernet (IEEE 802.3) nebo Wifi (IEEE 802.11) a přenosové rychlosti v síti mohou dosahovat až 100 Gb/s. [Dor18]

2.1 Síťová architektura

Návrh a sestavení počítačové sítě je rozsáhlá a složitá záležitost. Síť musí být funkční, dostatečně rychlá, efektivní a spolehlivá. Protože jde o komplexní problém, vzniklo v minulosti několik prostředků, které tento problém rozdělují na více menších celků a usnadňují tak jeho řešení.

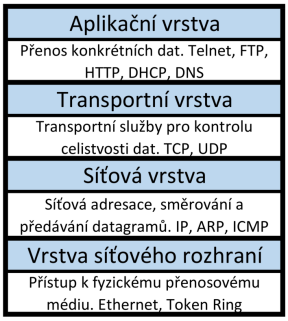
2.1.1 OSI/ISO model

OSI/ISO model je referenční model, který vytvořila organizace International Organization for Standardization (ISO). Jedná se o abstraktní model pro reálný síťový systém. Model funguje na bázi rozdělení problému do jednotlivých vrstev, kdy každá vrstva má vykonávat soubor předem daných funkcí. Je tvořen celkem sedmi vrstvami. Každá vrstva pro svou funkci využívá služeb nižší vrstvy a poskytuje své služby vrstvě vyšší. Nejnižší vrstva je vrstva fyzická. Přesto, že se tento standard považuje za základní strukturu každé síťové technologie, nebyl tento model využitý tak, jak se očekávalo. V některých aplikacích a sítích není potřeba implementovat všech sedm vrstev. Při praktickém použití je proto někdy vrstva, která není potřeba, prázdná a tím se celková architektura zjednoduší (příkladem je architektura TCP/IP).

Obrázek 2.1: Jednotlivé vrstvy OSI/ISO modelu i se základním popisem funkce každé vrstvy.

2.1.2 TCP/IP

Architektura TCP/IP byla vytvořena ministerstvem obrany USA původně pro vojenské účely. S rozšířením internetu byla uvolněna pro veřejnost. V dnešní době je součástí drtivé většiny zařízení, které připojujeme do sítě a všech operačních systémů. Architektura TCP/IP není závislá na přenosovém médiu a můžeme ji proto používat s jakoukoliv formou fyzického připojení. Původní myšlenka byla, že jako vzor pro síťovou komunikaci bude sloužit model OSI/ISO, ale v současnosti se z praktických důvodů používá v převážné většině aplikací architektura TCP/IP. Tato architektura obsahuje oproti modelu OSI/ISO pouze čtyři vrstvy. Rozpis jednotlivých vrstev je uveden v tab. 2.2 Jméno architektury vzniklo na základě dvou protokolů, které jsou pro tuto architekturu stěžejní. Protokol TCP je součástí transportní vrstvy a slouží k přenosu toku bajtů se spolehlivým doručováním dat. IP protokol je základním protokolem fungujícím na síťové vrstvě a poskytuje datagramovou službu. Tento protokol je nespolehlivý a rozlišuje jednotlivé síťové prvky v síti pomocí IP adres.[Dor18]



Obrázek 2.2: Jednotlivé vrstvy architektury TCP/IP i se základním popisem funkce každé vrstvy.

2.2 Ethernet

Původní varianta Ethernetu byla vyvinuta na základě sběrnicové technologie. Tuto technologii vytvořila a poprvé použila firma Xerox v roce 1975. [Wik] Ethernet definuje pouze dvě nejnižší vrstvy OSI/ISO modelu - fyzickou a linkovou. V minulosti se používal ve spojení s koaxiálním kabelem. Kvůli krátké maximální délce kabelu mezi uzly, a hlavně kvůli tomu, že na tomto kabelu mohla být provozována pouze half-duplex komunikace, se později přešlo na kroucenou dvoulinku, která se používá dodnes. Od roku 1985 je Ethernet součástí standardu IEEE 802.2 (Logical Link Control) a IEEE 802.3, jenž definuje přístupovou metodu k médiu CSMA/CD [1] a specifikuje fyzickou vrstvu Ethernetu. [Xer80]

Specifikace Ethernetu ve spolupráci Xerox, Intel a DEC byly: [Xer80]

- Rychlost: 10 Mb/s
- Základní pásmo - přenos bez použití modulace
- Maximální délka mezi segmenty je 500 m
- Maximálně 100 vysílačů na jeden segment

V současnosti je Ethernet základem drtivé většiny sítí LAN. IEEE institut standard postupně upravoval a přidával varianty a v současnosti je nejnovější norma ze srpna 2018, značená jako IEEE 802.3cj, která obsahuje například tyto specifikace: [Wik]

- 40 Gb/s přes UTP kabel kategorie 5 a 6
- 200 Gb/s přes jednoduché optické vlákno

[1] CSMA/CD (Carrier Sense Multiple Access with Collision Detection) je protokol pro přístup k přenosovému médiu v počítačových sítích.

Jsou to specifikace, které se v současnosti téměř nepoužívají, ale je odtud vidět, jakých možností lze pomocí technologie Ethernet dosáhnout. V současnosti se nejčastěji používá rychlost 1 Gb/s.

2.2.1 Varianty fyzické vrstvy

- 10BASE5: Tlustý koaxiální kabel s maximální přenosovou rychlostí 10 Mb/s. Maximální délka byla 500 m s použitím maximálně 4 opakovačů. Pouze half-duplex komunikace.
- 10BASE-F: Dvě optická mnohavidová vlákna s přenosovou rychlostí 10 Mb/s. Maximální délka 2 km. Full duplex komunikace.
- 10BASE2: Tenký koaxiální kabel s maximální přenosovou rychlostí 10 Mb/s. Maximální délka mezi segmenty je 185 m s použitím dvou opakovačů.
- 10BASE-T: První standard, který používá kabely s kroucenou dvoulinkou. Maximální přenosová rychlost je opět 10 Mb/s. Maximální délka mezi segmenty je 100 m. [Che99]
- 100BASE-TX: Kroucené páry kategorie 5 s maximální rychlostí 100 Mb/s. Maximální délka je 100 m.
- 100BASE-T2: Kroucená dvoulinka kategorie 3 s maximální rychlostí 100 Mb/s a maximální délkou 100 m.
- 100BASE-T4: Kroucená dvoulinka kategorie 3. Na rozdíl od předchozího standardu používá všechny čtyři páry. Funguje pouze na half-duplexu.
- 100BASE-FX: Dvojice optických vláken s maximální přenosovou rychlostí 100 Mb/s a délkou segmentu 412 m pro mnohavidové a 2 km pro jednovidové vlákno. Full-duplex komunikace.
- 1000BASE-T: Kroucená dvoulinka minimálně kategorie 5. Využívá všechny čtyři páry. Maximální délka segmentu je 100 m. Full-duplex komunikace. Maximální přenosová rychlost je 1 Gb/s.
- 1000BASE-CX: Stíněná kroucená dvoulinka. Maximální délka segmentu je 25 m. Používá se ve firmách. Full-duplex komunikace. Maximální přenosová rychlost je 1 Gb/s.
- 1000BASE-SX,LX: Mnohavidová a jednovidová optická vlákna. Maximální délka segmentu je 500 m (mnohavidová vlákna) a 2 km (jednovidová vlákna). Plně duplexní komunikace. Maximální přenosová rychlost je 1 Gb/s.
- 10GBASE-T: Kroucená dvoulinka minimálně kategorie 6. V případě kategorie kabelu 6a je maximální vzdálenost mezi segmenty 100 m. Maximální přenosová rychlost je 10 Gb/s.

- 10GBASE-SR: Mnohavidová optická vlákna s maximální přenosovou rychlostí 10 Gb/s. Maximální vzdálenost mezi segmenty je 400 m.
- 10GBASE-LR,ER: Jednovidová optická vlákna s maximální přenosovou rychlostí 10 Gb/s a dlouhou délkou segmentu. Maximální délka segmentu je 10 km, respektive 40 km. [Wik]

2.3 Opakovač

Opakovač je prvek, který pracuje na první (fyzické) vrstvě v rámci OSI/ISO modelu nebo na vrstvě síťového rozhraní v rámci protokolu TCP/IP. Je to zařízení, které spojuje dva segmenty Ethernetu. Signál, který přijme z jednoho segmentu, přeposílá do druhého a naopak. Opakovač je nejčastěji používaný na místě, kde je potřeba prodloužit dosah signálu pro pokrytí většího množství zařízení.

Některé opakovače jsou schopné přijímaný signál upravit a dále posílat signál změněný (změna frekvence nebo amplitudy). [Tec]

Obrázek 2.3: Funkce opakovače pouze na fyzické vrstvě.

Kapitola 3

Komunikační technologie v automobilovém průmyslu

3.1 CAN

V současnosti se ve většině případů pro komunikaci mezi senzory a řídicími jednotkami v automobilu používá sběrnice CAN. V automobilech byla tato sběrnice použita z důvodu redukce kabeláže, menší hmotnosti, jednoduššího připojení dalšího zařízení a také z důvodu jednoduché diagnostiky. Přenosová rychlost 1 Mb/s je pro spoustu současných aplikací nedostatečná a při použití technologie Ethernet dosáhneme ještě lepších parametrů než v případě požití CAN sběrnice. Technologie CAN je zde uvedena pouze okrajově, práce není tímto směrem zaměřena, protože se věnuje Automotive Ethernetu a monitoringu jeho komunikace. [Kal13]

3.2 Automotive Ethernet

Důvodem pro používání Ethernetu v automobilech je ten fakt, že se do nich instaluje stále více elektronických zařízení a senzorů. V minulosti to byly převážně informační prvky, ale postupně se rozšiřovaly i elektronicky ovládané a sledované kontrolní prvky auta jako je ABS, hlídání jízdy v pruzích nebo sledování dopravních značek a další senzoriky. Rozšiřování těchto prvků vyžaduje stále více kabeláže a také vyšší přenosové rychlosti. V automobilech se v dnešní době začíná používat upravený Ethernet standard podle IEEE 802.3 splňující požadavky, které jsou pro použití v automobilech nutností na rozdíl od klasického Ethernetu. Ten se používá ve standardních sítích LAN. [CMK14] Základní rozdíly mezi těmito standardy jsou:

- Snížení základní frekvence ze 125 MHz na 66 MHz kvůli zmenšení EMC rušení.

- Snížení maximální délky kabelu ze 100 m (10BASE-T1[1] s 100BASE-T1) na 15 m pro zachování přenosové rychlosti.
- Používání pouze jednoho páru kroucené dvoulinky místo čtyř z důvodu snížení hmotnosti.

3.3 BroadR-Reach/IEEE 802.3

3.3.1 BroadR-Reach

BroadR-Reach [2] se může značit také jako 10BASE-T1 nebo 100BASE-T1[3] (podle standardu IEEE 802.3bw). Jedná se o standard, který splňuje přísné požadavky pro použití Ethernetu v automotive úpravou komunikačního protokolu fyzické vrstvy. [Ph.12]

Hlavní prvky BroadR-Reach:

- Maximální přenosová rychlost 100 Mb/s
- Pouze jedna kroucená nestíněná dvoulinka
- Plně duplexní přenos
- Nízká ceny a hmotnosti
- Splnění požadavků EMC u automobilů díky snížení šířky pásma na 33,3 MHz
- Maximální délka kabelu 15 m

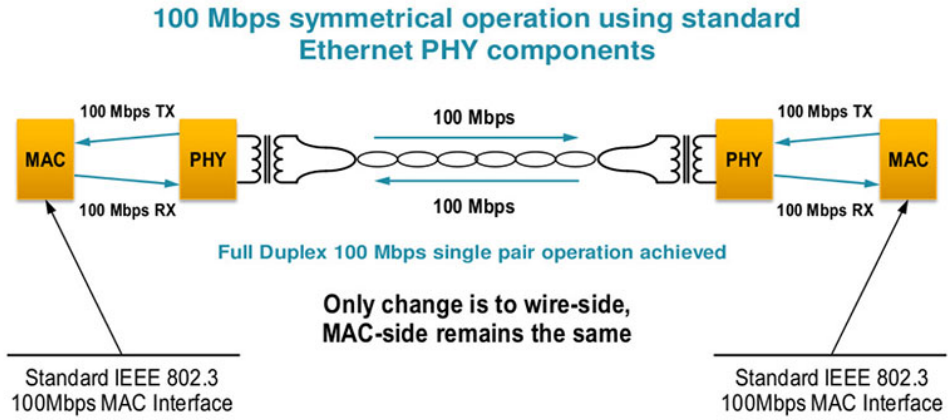
3.3.2 IEEE 802.3

Tento standard pod hlavičkou organizace IEEE vznikl na základě vývoje standardu BroadR-Reach pod hlavičkou organizace OPEN. IEEE 802.3 je ve svém základě naprosto stejný jako BroadR-Reach. IEEE tento standard dále rozvíjí a pracuje na něm. V současnosti již na základě tohoto standardu existuje kromě IEEE 802.3bw i IEEE 802.3bp (1000BASE-T1) s přenosovou rychlostí 1 Gb/s. [CMK14]

[1] Jedná se o standard IEEE 802.3cg-2019 [**?**]

[2] Standard BroadR-Reach je standardizován organizací OPEN Alliance SIG, která má v současnosti více než 300 členů a vznikla z důvodu použití Ethernetu v automotive.

[3] 10BASE-T1 a 100BASE-T1 je standardizován společností IEEE na základě BroadR-Reach, ale v podstatě se jde o stejnou technologii.



Obrázek 3.1: Ukázka fyzické vrstvy BroadR-Reach full-duplex komunikace.[Mal18]

Existuje už i pracovní skupina, která se zabývá možností přenosu 10 Gb/s po kroucené dvoulince. U těchto všech možností je maximální délka kabelu 15m při zachování dané přenosové rychlosti. Důležité je, že IEEE 802.3bw je kompatibilní se standardem BroadR-Reach, který používá mnoho firem, protože vznikl dříve a je mnohdy rozšířenější než IEEE 802.3bw.

3.3.3 Další požadavky vyžadované v automobilech

Pro některé prvky v automobilech potřebujeme splnit další požadavky, které tento standard nesplňuje. Nejčastějším je například nízké zpoždění nebo časová synchronizace. Dnes již v automobilech používáme i technologie, díky kterým se nám daří, tyto požadavky splnit. Tyto technologie je někdy důležité kombinovat, abychom dosáhli požadovaných výsledků. [IXI14]

Nízké zpoždění

U některých automobilových systémů a senzorů vyžadujeme, aby byly doručeny s nejmenším časovým zpožděním, záleží i na mikrosekundách. V klasickém Ethernetu nový paket čeká, než je odeslání již existujícího paketu dokončeno. Tento problém řeší standard IEEE 802.3br, který zajišťuje přednostní posílání prioritních paketů. Jednotlivé pakety mají různé priority a paket s vyšší prioritou může přerušit posílání paketu s nižší prioritou. V případě tohoto standardu mohou pakety s vysokou prioritou splňovat zpoždění s maximální hodnotou několik mikrosekund. [IXI14]

Časová synchronizace

Některé aplikace a algoritmy v automobilech vyžadují kontinuální příjem dat z více senzorů najednou a v tomto případě musíme mít informaci o tom, ve který čas nám daný paket ze všech potřebných senzorů přišel. V rámci těchto větví se senzory proto musíme mít synchronizovaný čas s přesností vyšší než jedna mikrosekunda. Těchto požadavků docílíme pomocí standardu IEEE 802.1AS. Tento standard používá také standard IEEE 1588, pomocí kterého získáme rychlejší metody pro výběr hlavního časovače. [IXI14]

3.4 Fyzická vrstva PHY

BroadR-Reach podporuje plně duplexní přenos pouze po jednom páru kroucených drátů UTP. Z těchto důvodů je maximální přenosová rychlost 100 Mb/s při maximální délce kabelu 15 m. Počet párů se snižuje z důvodu zmenšení celkové hmotnosti kabelu. [Ph.12]

BroadR-Reach musí být konfigurován bez Auto-Negotiation, protože potřebujeme docílit rychlého sestavení spojení mezi jednotlivými zařízeními. Proto musíme rozlišit, kdo má roli Master a kdo má roli Slave. Master potom používá lokální hodiny a Slave používá hodiny z přijatého signálu od Mastera.

3.4.1 Ochrana proti rušení

Jeden z nejdůležitějších rozdílů oproti standardnímu Ethernetu je silnější ochrana proti EMC rušení. Při nasazení Ethernetu v automobilu je v bezprostředním okolí kabelu spousta elektromagnetických impulzů, které mohou ovlivňovat signál vedený kabelem. Musí splňovat normu CIPSR25[4]. Kabel Ethernetu musí být také stíněn z důvodu vyzařování samotného kabelu a případného ovlivňování a rušení ostatních prvků v automobilu. [CMK14]

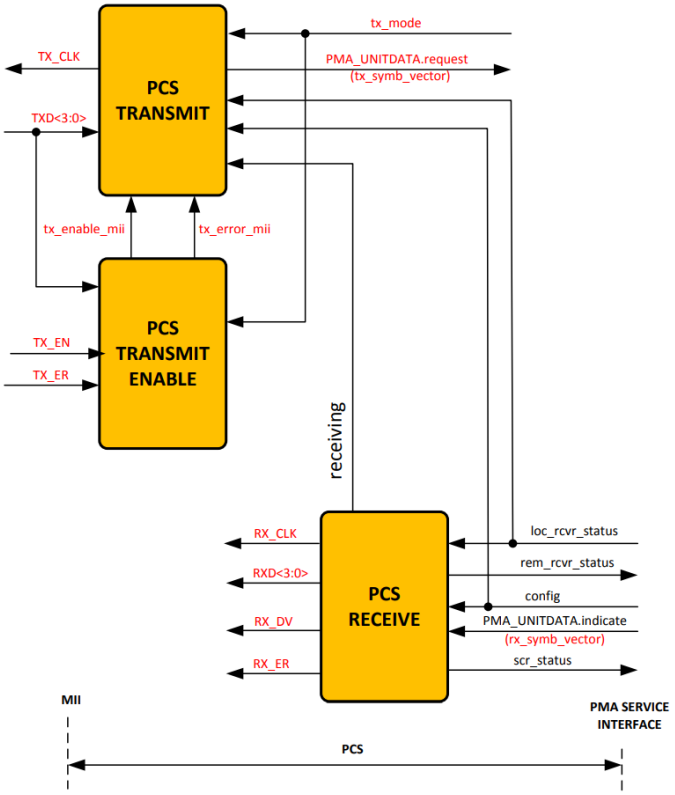
Standard používá 3-stupňovou pulzně-amplitudovou modulaci, která zajišťuje menší elektromagnetické rušení kvůli menší šířce pásma (33,3 MHz). Důležitou součástí ochrany proti rušení je také echo cancellation. [Cor14]

3.4.2 Kódování

Kódování zajišťuje u BroadR-Reach podvrstva fyzické vrstvy - BR-PCS (Physical Coding Sublayer). Tato podvrstva komunikuje se standardní MAC vrstvou pomocí 100 Mb/s rozhraní, které se nazývá MII (Media Independent

[4]Norma pro vyzařování komponent automotive standard v pásmu 150 kHz až 960 MHz

Interface) Obr. 3.5.



Obrázek 3.2: BR-PCS diagram

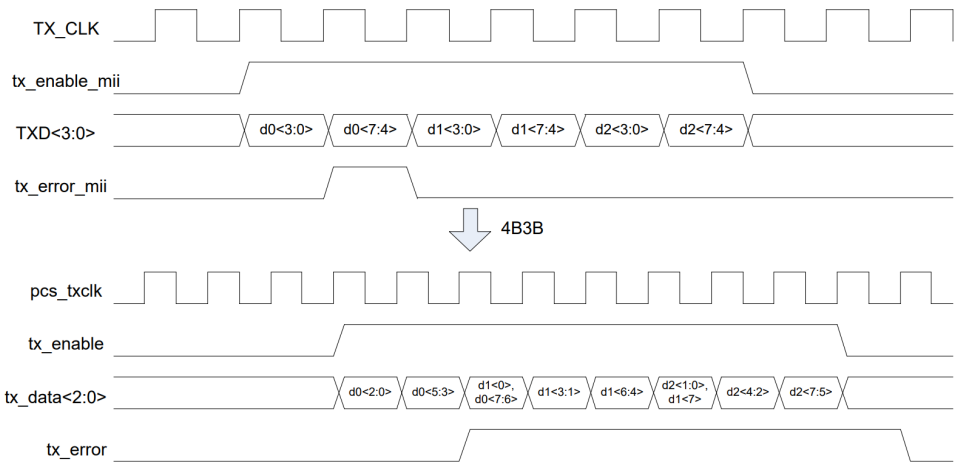
Standard MII posílá 4 bity dat s frekvencí 25 MHz podvrstvě BR-PCS. Podvrstva BR-PCS nejprve transformuje tyto bity pomocí konverze 4B3B. Jedná se o datovou konverzi, kdy jsou čtyřbitové bloky dat posílány rozhraním MII s hodinami o frekvenci 25 MHz a jsou převedeny na tříbitové bloky dat s hodinami o frekvenci 33,3 MHz. Pokud celkový počet bitů není násobek tří, potom konverze 4B3B doplní do posledního bloku dat jeden nebo dva bity. Tyto bity se nazývají stuff bity. [Buc16]

Jak probíhá přenos a konverze dat pro jednotlivé varianty je zobrazeno na obr. 3.3

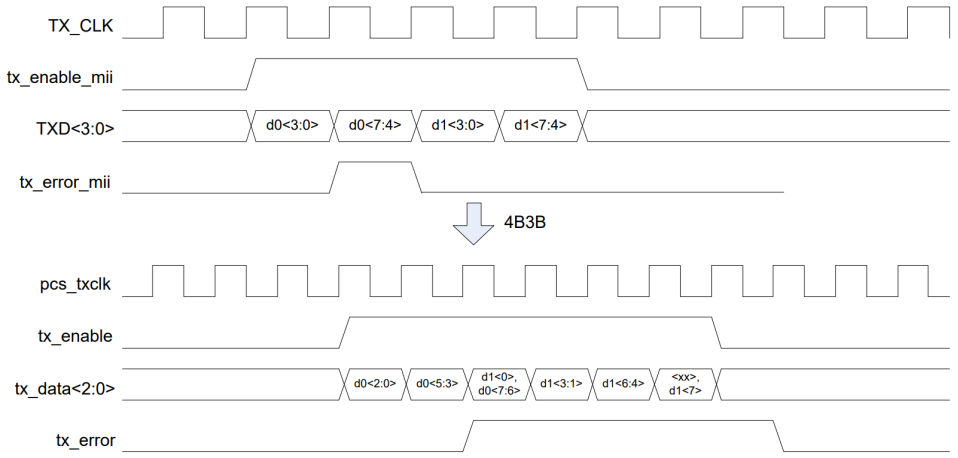
Tyto tříbitové bloky jsou následně náhodně promíchány pomocí Side-Stream Scrambleru a nakonec jsou zakódovány po párech jako ternární symboly s hodnotami -1, 0 nebo +1. Toto kódování je označováno jako 3B/T2. Výhodou tohoto kódování je, že jeden ternární pár není použit a je rezervován pro řídicí funkce. Ternární páry se následně serializují a přenášejí s frekvencí hodin 66,6 MHz. [CMK14]

Tato podvrstva kóduje a dekóduje data mezi MII a PMA (Physical Medium Attachment) vrstvami. Kódování BR-PCS obsahuje funkce PCS reset, PCS

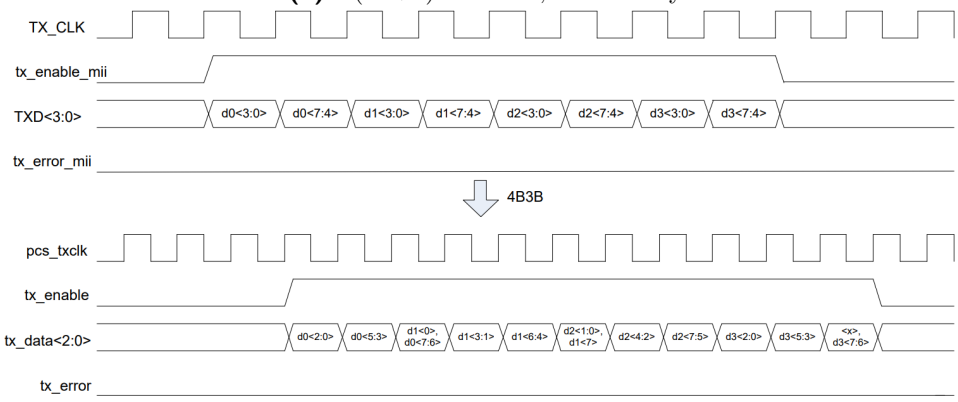
přenos dat a PCS příjem dat. Blokové schéma vrstvy PCS můžeme vidět na obr. 3.2 [Cor14]



(a) : 3n bitů dat, žádný stuff bit



(b) : (3n+1) bitů dat, 2 stuff bity



(c) : (3n+2) bitů dat, 1 stuff bit

Obrázek 3.3: 4B3B konverze datových signálů MII

3.5 Linková vrstva (MAC a LLC)

Vzhledem k tomu, že se v Ethernetu LLC podvrstva linkové vrstvy téměř nepoužívá, bude se tato podkapitola zabývat pouze podvrstvou MAC, kde jsou definovány a implementovány všechny operace. Na linkové vrstvě dochází také k zabalení dat přijatých z fyzické vrstvy do bloků neboli rámců, které přijímá od fyzické vrstvě. Tyto rámce jsou opatřeny fyzickou MAC adresou odesílatele i příjemce. Tato adresa slouží jako jednoznačný identifikátor síťového zařízení. Linková vrstva zajišťuje hlavně adresaci rámců. Dochází zde také ke kontrole dat a adresaci rámců k cílovému zařízení. V neposlední řadě zajišťuje řízení přístupu k médiu (CSMA/CD). Na MAC vrstvě pracují switche, huby, bridge a také repeatery. Vzhledem k tématu jsou pro tuto práci podstatné právě poslední zmíněné repeatery, protože ze zadání je potřeba takový repeater vytvořit. Fyzická vrstva PHY a linková vrstva MAC jsou propojeny pomocí rozhraní MII viz 3.7 [Con]

3.5.1 Formát rámce

Formátů rámce se používá v Ethernetu několik, ale liší se vždy jen v detailech. Například v tom, jestli daný rámec obsahuje VLAN tag nebo ne. V této práci bude popisován jeden z nejběžnějších rámců Ethernetu, který se používá i pro standard BroadR-Reach. ethernetový rámec má v tomto případě délku 1522 bajtů v případě plného datového pole. Součástí této podkapitoly je i Preambule a Start of Frame. Tyto dva bloky nejsou součástí rámce, ale patří do nadřazeného ethernetového paketu. [Con]

Preambule	SOF	Cílová adresa	Zdrojová adresa	VLAN Tag	Typ a délka	Datové pole	CRC kontrola
7 Bajtů	1 Bajt	6 Bajtů	6 Bajtů	4 Bajty	2 Bajty	46 - 1500 Bajtů	4 Bajty

Obrázek 3.4: Formát paketu Ethernetu.

Preambule

Preambule tvoří počátečních 7 bajtů ethernetového paketu. Skládá se z šesti pravidelně se střídajících jedniček a nul (10101010). Preambule slouží jako oznámení o začátku paketu a také k synchronizaci hodin mezi odesilatelem a příjemcem. [Wik]

SOF

Za preambulí následuje jednobajtový SOF, který signalizuje konec preambule a zároveň začátek ethernetového rámce a má tvar 10101011. Ihned za SOF následuje blok s cílovou adresou, což je první část ethernetového rámce.

Cílová a zdrojová adresa

Cílové a zdrojové adrese v ethernetovém rámci se někdy říká obecným označením "hlavička"rámce. Cílová i zdrojová MAC adresa má šest bajtů. Dohromady má tedy hlavička rámce velikost 12 bajtů.

VLAN tag

Tato část Ethernetového rámce je volitelná. V rámci tudíž být nemusí. Je to standard vytvořený IEEE a jedná se o standard 802.1Q. Pokud je tato část součástí ethernetového rámce, má velikost 4 bajty a definuje tak virtuální síť VLAN, která dokáže rozdělit fyzickou síť na několik virtuálních (logických) sítí. VLAN tag je obecně rozdělený na dvě stejně velké části. První část o velikosti 2 bajty obsahuje pouze identifikátor typu rámce (VLAN protocol ID). [Che99]

Druhá část o velikosti dva bajty obsahuje:

- Tříbitovou hodnotu uživatelské priority rámce (PCP)
- Identifikátor pořadí přenášeného rámce (CFI)
- Dvanáctibitový identifikátor čísla VLANu.

Typ a data

Tato část rámce je dvoubajtová a nese v sobě informaci o velikosti dat, případně o typu dat, které rámec obsahuje. Hodnoty o velikosti 1500 a menší znamenají indikaci velikosti datového bloku. Hodnoty o velikosti 1536 a vyšší indikují, jaký protokol uložená data obsahují. Hodnoty mezi 1500 a 1536 nejsou definovány. V případě, že tyto dva bajty nesou informaci o typu dat, velikost datového bloku je určena umístěním mezery mezi jednotlivými pakety a validní CRC kontrolou. [Wik]

CRC kontrola

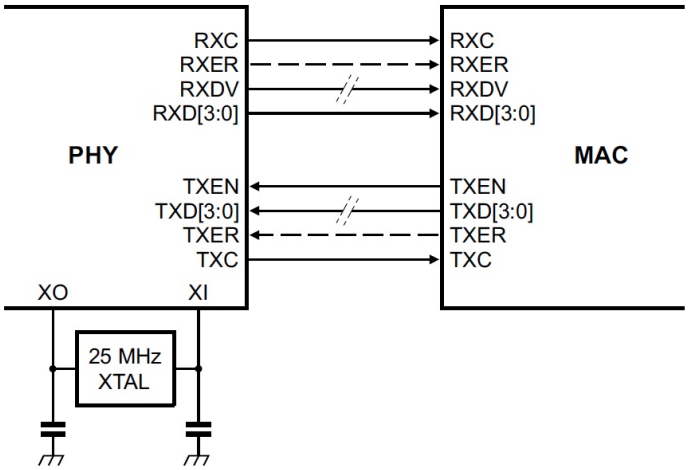
Anglicky se označuje FCS. Je kontrolní čtyřbajtový součet, který slouží ke kontrole dat v rámci a poskytuje základní ochranu dat. V případě, že data byla přijata bez chyby, výsledek CRC kódu je vždy nula.

3.6 1000BASE-T1

V roce 2017 instituce IEEE standardizovala variantu Ethernetu o maximální přenosové rychlosti 1000 Mb/s přes jednu kroucenou dvoulinku. Jde tak o další posun v rychlosti pro použití například v automotive. Je zde uveden z toho důvodu, že varianta, která je používána zde v diplomové práci rozhodně není jediná a nejlepší. Určitě se bude tato technologie neustále zlepšovat. V případě tohoto standardu s přenosovou rychlostí 1 Gb/s se již ale nepoužívá standard MII, ale vylepšený standard GMII, který používá frekvenci hodin 125 MHz. [CMK14]

3.7 MII rozhraní

MII rozhraní je standard, který je schopný propojit různé typy MAC a PHY vrstvy. Znamená to, že k linkové vrstvě Ethernetu se můžeme připojit pomocí jakéhokoliv typu fyzického média. Přenos přes MII rozhraní probíhá pomocí konkrétních signálů mezi vrstvou PHY a vrstvou MAC. Data jsou přijímána a odesílána pomocí čtyřbitových signálů RXD a TXD. Příjem i odesílání dat je synchronizované pomocí hodinového signálu RXC a TXC. Oba hodinové signály jsou poskytovány fyzickou vrstvou a jsou typicky odvozeny od externího krystalu o frekvenci 25 MHz. Další dva důležité signály jsou RXDV, který indikuje bezproblémový příjem dat a signál TXEN, který iniciuje přenos dat. Tato varianta je kompatibilní pouze s variantou 10BASE-T1 a 100BASE-T1. [Cor14]



Obrázek 3.5: MII rozhraní

3.7.1 Signály a kódování

MII rozhraní obsahuje signály, které jsou směrovány v vrstvy PHY do vrstvy MAC (přijímací cesta) a také naopak signály z vrstvy MAC do vrstvy PHY (vysílací cesta). V každém z těchto dvou směrů je stejný počet signálů s podobnou funkcí. Data jsou přenášena pomocí čtyřbitových datových vodičů TXD a RXD. Přenos dat je synchronizován pomocí hodinových signálů TXC a RXC. Oba dva hodinové signály jsou poskytovány vrstvou PHY, jelikož jsou odvozeny od externího krystalu s frekvencí 25 MHz. Normální datový přenos je indikován signálem TXEN a signál RXDV indikuje standardní příjem dat.[Ph.12]

Vysílací cesta	Přijímací cesta
TXCLK	RXCLK
TXD <0:3>	RXD<0:3>
TXEN	RXDV
TXER	RXER

Tabulka 3.1: Signály pro vysílání a příjem

V následujících dvou tabulkách tab.3.2 a tab.3.3 vidíme popis kódování signálů TXEN, TXER, RXDV a RXER v závislosti na přijatých datových signálech. [NXP17]

TXEN	TXER	TXD[3:0]	Indication
0	0	0000 až 1111	normální rozhraní
0	1	0000 až 1111	rezervováno
1	0	0000 až 1111	normální datový přenos
1	1	0000 až 1111	šíření chyb přenosu

Tabulka 3.2: MII kódování TXD[3:0], TXEN a TXER

RXDV	RXER	RXD[3:0]	Indication
0	0	0000 až 1111	normální rozhraní
0	1	0000	normální rozhraní
0	1	0000 až 1111	rezervováno
0	1	1110	chybná indikace nosiče
0	1	1111	rezervováno
1	0	0000 až 1111	normální datový přenos
1	1	0000 až 1111	příjem dat s chybami

Tabulka 3.3: MII kódování RXD[3:0], RXDV a RXER

3.8 Budič TJA1100

TJA1100 je čip od firmy NXP, který funguje na základě standardu 100BASE-T1 a je optimalizován pro použití v automobilovém průmyslu. Zařízení poskytuje přenos dat s rychlostí 100 Mb/s pomocí jednoho nestíněného krouceného páru (UTP kabelu). Maximální délka kabelu při zachování této rychlosti je 15 metrů. Čip je přizpůsoben pro použití v oblastech IP kamer, asistenčních systémů pro řidiče a páteřní síť automobilu. TJA1100 byl navržen tak, aby jeho spotřeba a systémové nároky byly minimalizovány při zachování robustnosti systému, který je požadován pro použití v automobilovém průmyslu. [NXP17]

3.8.1 Vlastnosti

- Optimalizace kapacitního rušení pro UTP kabel.
- Vylepšená integrace PAM-3 modulace pro nízko-frekvenční vyzařování.
- Optimalizace příjmu dat po kabelu s délkou do minimálně 15 metrů.
- Snížená spotřeba elektrické energie díky konfigurovatelné amplitudě signálu přizpůsobené délce kabelu.
- Pin vyhrazený pro vypínání a zapínání PHY pro snížení spotřeby energie.
- Mód nízké spotřeby (Sleep) s vlastním buzením.

- Robustní vzdálené buzení přes sběrnici.
- Detekce podpětí s chováním tiché poruchy.
- Výkon ovladače výstupu optimalizovaný podle EMC pro MII a RMII.
- Diagnostika poruchy kabelu (zkrat nebo přerušení kabelu).
- Malé HVQFN-36 [5] pouzdro pro aplikace s omezeným prostorem na PCB.
- Ochrana MDI pinů proti ESD.
- Ochrana MDI pinů proti přechodům v automobilovém prostředí.
- Teplotní rozsah od -40 °C do +125 °C.
- Kvalifikace výrobku v souladu s AEC-Q100 [6].
- MII a RMII rozhraní.
- Reverzní MII pro propojení dvou PHY.
- 3V3 napájení s integrovanými 1V8 LDO regulátory.
- Integrované rezistory pro vyvážení UTP kabelu.
- Vnitřní i vnější a vzdálené zpětné vazby pro diagnostiku.
- Řídicí výstupní LED pro snadnější diagnostiku.

3.8.2 Blokové schéma

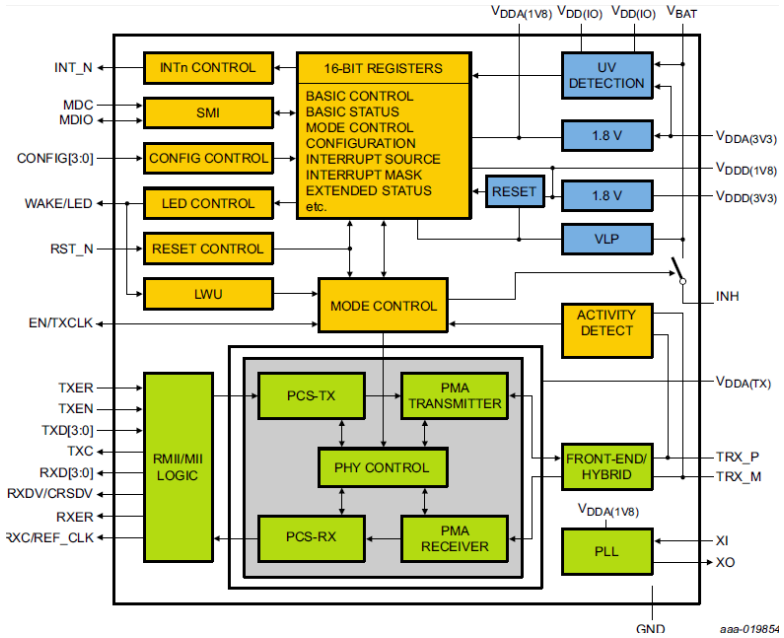
Blokové schéma na Obr. 3.6 ukazuje celou vnitřní strukturu čipu TJA1100. Vidíme zde 100BASE-T1 sekci, která je vyznačená zeleně a která obsahuje bloky založené na standardu IEEE 802.3. Máme zde PCS, PMA pro příjem i vysílání signálu a dále MII a RMII logiku pro následnou komunikaci s MAC vrstvou. [NXP17]

Dále jsou zde části, které zajišťuji například konfiguraci registrů nebo řídí LED diodu a reset čipu (žlutá barva). Jsou zde také bloky, které zajišťují napájení (modrá barva).

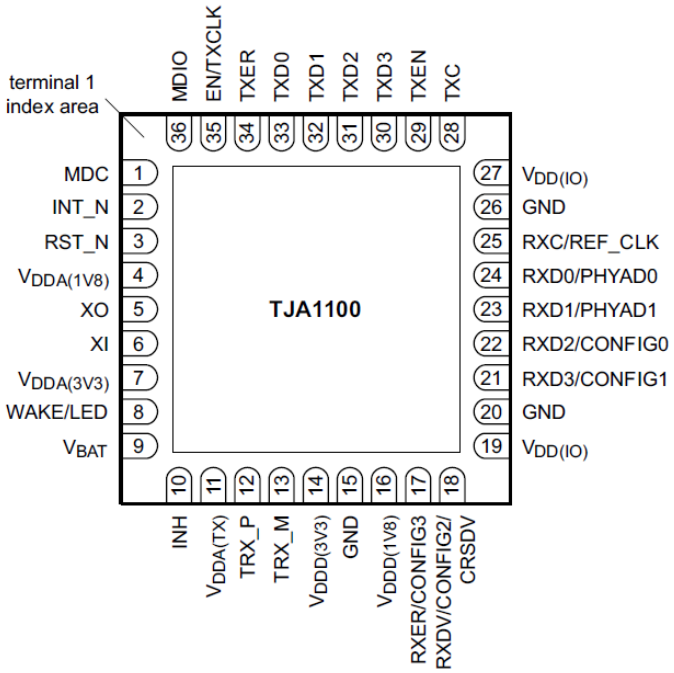
3.8.3 Rozmístění pinů

Čip TJA1100 má velikost 6x6 mm a má dohromady 36 pinů rozmístěných rovnoměrně po celém obvodu. Rozmístění jednotlivých pinů můžeme vidět na obr.3.7. Jak již bylo psáno výše, čip TJA1100 poskytuje podle standardu IEEE802.3 rozhraní MII, RMII včetně SMI. [NXP17]

[5] Jedná se o typ pouzdra, které firma NXP používá pro tento čip
[6] Kvalifikace zátěžového testu pro integrované obvody



Obrázek 3.6: Blokové schéma čipu TJA1100



Obrázek 3.7: Rozmístění pinů čipu TJA1100

Kapitola 4

Komunikační technologie v leteckém průmyslu

4.1 ARINC 429

ARINC 429 je v současnosti nejčastěji používaný standard pro přenos dat v komerčních letadlech. Jedná se o datovou sběrnici, která přenáší data jednosměrně ve velikosti 32bitových slov při bipolární modulaci Return to Zero. Pro obousměrnou komunikaci tedy potřebujeme dvě sběrnice. Jednotlivá zařízení jsou propojená pomocí stíněné kroucené dvoulinky, kterou vede diferenční signál. Tato technologie je však postupně nahrazována novější AFDX. Vlastnosti ARINC 429: [Fuc12]

- Napěťové úrovně mezi diferenčními vodiči: -10 V, 0 V, +10 V
- Bitové kódování: Bipolární Return to Zero
- Velikost slova: 32 bitů
- Přenosová rychlost: 100 Kb/s nebo 12,5 Kb/s

4.2 AFDX

4.2.1 Základní vlastnosti

AFDX je technologie, která v současných moderních letadlech nahrazuje starší technologii ARINC 429. AFDX poskytuje vysokorychlostní přenos dat, větší spolehlivost nebo také snížení váhy díky redukci kabeláže. Požadavky na elektronické systémy a kabeláž přišly v době, kdy se v letadlech začal

používat Fly-by-Wire systém. Dnes je většina řídicích prvků letadla napojena na FADEC, který dokáže podle zadaných pokynů ovládat letadlo.

V letadle jsou také další systémy, jako například palubní obrazovky nebo komunikační systémy, které vyžadují velkou míru spolehlivosti.

Technologie AFDX byla vyvinuta pro společnost Airbus a poprvé použita v modelu A380. V současnosti je připojena k motorům, navigačním prvkům, řídicím systémům, senzorům a dalším jednotkám, které vyžadují komunikaci a ovládání. [Yan07]

4.2.2 Technologie

AFDX je technologie, která je založena na protokolu Ethernetu (IEEE 802.3) a je specifikována jako ARINC 664 part 7. Hlavní přednosti AFDX oproti ARINC 429 jsou:

- Vyšší přenosová rychlost
- Menší hmotnost
- Plně duplexní přenos
- Větší spolehlivost doručení dat

4.3 Prvky AFDX

4.3.1 Avionics Subsystem

Jedná se o systémový prvek letadla jako například GPS nebo Flight Management System. Jedná se o zařízení, která přenášejí data přes síť. Každý Subsystem obsahuje koncový systém (End System), který připojuje Subsystem do sítě (AFDX Interconnect). [Yan07]

4.3.2 AFDX End System

Tento systém je rozhraní mezi Avionics Subsystem a AFDX Interconnect. Každý Subsystem je fyzicky spojen s prvkem End System. Zajišťuje přenos dat mezi jednotlivými subsystémy Avionics Subsystem.

Virtuální spojení

Koncové systémy si mezi sebou posílají rámce pomocí Virtuálních spojení (VL). Každý rámec má svou cílovou adresu, v rámci níž je 16bitová hodnota, která se jmenuje Virtual Link ID a účelem je směrování rámců v síti AFDX. Může tedy definovat více než 65 tisíc virtuálních spojení. Jeden koncový systém může poskytovat virtuální spojení pro jeden i více cílových systémů. To znamená, že jeden konkrétní odesílatel (koncový systém) má přiřazeno určité virtuální ID. Tento systém odesílá pakety pro konkrétní koncové systémy. Podobný princip funguje u starší technologie ARINC 429 a nazývá se Multidrop Bus. [Fuc12]

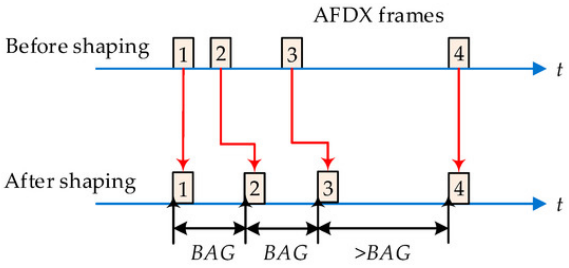
Tato virtuální (logická) propojení jsou jednosměrná (unidirectional). Regulace provozu na VL je typicky charakterizována dvěma parametry: Bandwidth Allocation Gap (BAG) a jitter. BAG definuje minimální časový interval mezi dvěma po sobě jdoucími rámci daného VL a jitter je zpoždění rámce nad míru BAG. Jitter nebude v síti přítomný, pokud je provoz zpracováván regulátorem, který pracuje na bázi jednoho kanálu (VL). Jitter se začne projevovat v síti s multiplexerem, kde dochází k určitému zpoždění. Dalším důležitým parametrem je maximální délka jednoho rámce S_{max}, díky kterému můžeme spočítat šířku pásma a také maximální jitter. Díky parametru BAG existuje v AFDX metoda, která zajišťuje, že jsou rámce od sebe vzdálené minimálně o čas BAG. Tato metoda se jmenuje Shaping a probíhá v koncových systémech sítě AFDX. Ukázku můžeme vidět na obr. 4.1. [HW12]

Parametry virtuálního spojení

- Přidělená šířka pásma
- Velikost rámce
- Maximální povolený jitter
- Počet virtuálních spojení
- Typ účtu
- Priorita
- Výběr sítě
- Maximální zkreslení

4.3.3 AFDX Interconnect

AFDX Interconnect představuje síť samotnou. Jedná se o plně duplexní síť se všemi pasivními i aktivními prvky. Její součástí jsou switche, které



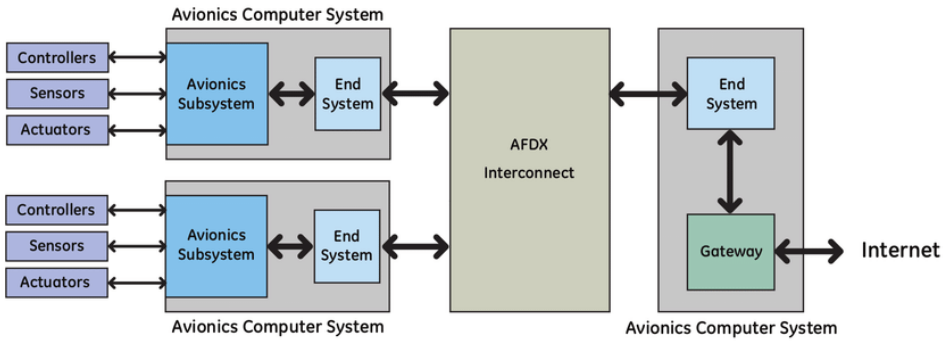
Obrázek 4.1: BAG Shaping

filtrují a přeposílají rámce do jejich cílové destinace (End System). Tato část propojuje všechny koncové systémy a kontroluje datový provoz podle zadaných konfigurací. V rámci bloku AfDX Interconnect je zde pět funkčních bloků, které jsou pojmenovány: Přepínací funkce (Switching), Filtrace a kontrola (Filtering and Policing), koncový systém přepínače, konfigurační tabulka a monitorovací funkce. [Fuc12]

Traffic Filtering a Policing

Filtrace znamená, že switchem dále projdou jen takové rámce, které switch uzná jako validní (velikost rámce, cílovou destinaci). Policing neboli kontrola je algoritmus zajišťující řízení provozu na každém VL. Každý rámec je zkontrolován na parametry, které má rámec přiřazen na základě VL (BAG, Jitter, S_{max}). AFDX definuje dva typy Traffic Policing:

- Byte-based provozní kontrola
- Frame-based provozní kontrola



Obrázek 4.2: Schéma AFDX včetně všech jeho součástí. [MVLC15]

4.3.4 Struktura zprávy a rámce

Rámec AFDX může mít velikost od 64 do 1518 bajtů podle velikosti dat. Strukturu jednoho rámce můžeme vidět na obr. 3.4. Nyní si podrobněji rozebereme zdrojovou a cílovou adresu. Část, ve které je cílová adresa, obsahuje identifikátor virtuálního spojení (VLID) a konstantní pole. Zdrojová část obsahuje větší množství informací. Jsou to tyto části: identifikátor sítě, identifikátor vybavení, identifikátor rozhraní a konstantní pole. [Yan07]

Kapitola 5

Jazyk VHDL

VHDL je jazyk, který je primárně určený pro návrh a programování integrovaných obvodů a nejčastěji programovatelných hradlových polí (FPGA). Jazyk byl vytvořen tak, aby byl programátor schopný popsat obvod na hradlové, RTL, ale i algoritmické úrovni. Jedná se tedy o jazyk, který popisuje chování hardware. Výhodou tohoto jazyka je jeho nezávislost na hardware, na kterém chceme, aby program fungoval. Nevýhodou je, že funguje naprosto jinak než běžné programovací jazyky typu C, a proto je nutné k němu i tak přistupovat. V případě přechodu z těchto jazyků může někomu dělat problém, že to, co se naučil, musí částečně zapomenout a nejlépe se učit programovat ve VHDL s čistým štítem. U tohoto jazyka dochází k silné kontrole, aby se odhalily návrhářské chyby již při překladu zdrojového kódu. V tomto programovacím jazyce bude v druhé části diplomové práce psán jednoduchý program pro ovládání desky, která byla navržena a vyrobena dříve v rámci této práce. [JP06]

5.1 Základní struktura

Jazyk VHDL popisuje jednotlivá číslicová zařízení pomocí dvou základních prvků:

- Entita
- Architektura

5.1.1 Entita

Entita je jednou z dvou hlavních návrhových jednotek jazyka VHDL. Tento prvek popisuje daný obvod jako "blackbox"se vstupy a výstupy. Popisuje tedy rozhraní objektu, které může představovat libovolný elektrický obvod

nebo i celý systém. Entita může obsahovat také několik volitelných sekcí, jež slouží k definování dalších funkcí. Sekce Generic umožňuje definovat generické konstanty, které lze využít pro parametrizaci entity.

Další volitelná sekce, port, umožňuje definovat porty entity, které si můžeme představit jako vstupy a výstupy našeho obvodu. Každý port musí mít své jméno, režim přenosu dat a typ dat, které bude přenášet. Pro nás je podstatný režim přenosu dat, který určuje, jestli půjde o vstup, výstup nebo půjde o jiný režim přenosu dat. Režimy přenosu dat jsou tyto:

- IN - Port tohoto typu je vstup do našeho obvodu. Znamená to, že data mohou pouze vstupovat do tohoto portu.
- OUT - Tento port je výstupem z našeho obvodu a data tedy vystupují z našeho obvodu ven. Vlastní entita tedy nemůže číst port tohoto typu.
- BUFFER - Tento port může být buzen pouze z vnitřku entity stejně jako u portu OUT. Na rozdíl od portu OUT může entita data zpětně i číst.
- INOUT - Port tohoto typu je kombinací portu IN a portu OUT. Data tímto portem mohou vstupovat i vystupovat.
- LINKAGE - Tento port se v praxi téměř nepoužívá.

5.1.2 Architektura

Architektura popisuje vlastní chování obvodu a funkci entity a je nazývána takzvaně sekundární návrhovou jednotkou jazyka VHDL. Každá entita může mít více architektur. Minimálně musí mít entita jednu architekturu. Jména více architektur v rámci jedné entity se musí vzájemně lišit. Chování obvodu může být v jazyce VHDL napsáno třemi způsoby neboli také třemi programovacími popisy.

Listing 5.1: Celek

```
entity NAZEV_BLOKU is
  Port (
    -- Definice vstupu a vystupu
    --
  );
end NAZEV_BLOKU;

architecture Behavioral of NAZEV_BLOKU is
  -- Definice promennych a signalu
  --
begin
  -- Vlastni kod
  --
```

```
end Behavioral;
```

5.2 Programovací popisy

Jazyk VHDL je složitý v tom, že můžeme přistupovat k programování obvodů několika způsoby a jeden problém můžeme naprogramovat různými styly. Tímto se poměrně zásadně liší od programování například v jazyku C nebo Pascal. Každý programovací popis je vhodný na jiné aplikace a zároveň je možné každou aplikaci napsat mnoha rozdílnými způsoby. V této práci jsou použity všechny tři programovací popisy podle toho, co je pro daný problém a konkrétní funkci programu nejjednodušší. [JP06] Tyto popisy jsou tři:

- Strukturální popis
- Behaviorální popis
- Popis datových toků (DataFlow)

5.2.1 Strukturální popis

Kód, jenž je psaný jako strukturální popis, je návrh programu, který je rozdělený do několika vzájemně propojených bloků a tvořících kompletní návrh. V případě, že je náš program jednoduchý, můžeme program popsat pouze v jednom bloku. Pokud je ale program komplexnější, je přehlednější rozložit jej do několika menších bloků, z nichž každý bude mít svou funkci. Strukturální popis tedy ukazuje, jak je systém uspořádaný a propojený. [JP06]

5.2.2 Behaviorální popis

Behaviorální popis vyjadřuje chování jednotlivých součástek v modelu. Také přesně popisuje, co se stane na vstupech a následně na výstupech. Soustředíme se zde na operaci vykonanou systémem a příliš nás nezajímá konkrétní vnitřní uspořádání a funkce. [JP06]

5.2.3 Popis datových toků

Datové toky popisují, kterou cestou vedou informace přes model VHDL. Tento popis je nejvíce spojen s fyzickým uspořádáním celého problému. [JP06]

Část II

Praktická část

Kapitola 6

Hardware

6.1 Návrh schématu desky

Při tvorbě schématu byl použit volně dostupný program pro tvorbu plošných spojů - KiCad. Jako základ a hlavní prvek desky byl vybrán čip TJA1100 od firmy NXP viz kapitola 3.8. Tyto čipy musí být na desce dva, protože navrhujeme opakovač, který přijímá data ze dvou zařízení (kanálů). Plošný spoj přijímá data z kanálu 1 a přes první čip TJA1100 je posílá do druhého čipu TJA1100, který posílá tato data do kanálu 2. Pro data z kanálu 2 je princip pouze obrácený.

K čipům jsem následně s pomocí datasheetu připojil veškeré periferie, které TJA1100 potřebuje. Jsou zde umístěny ochranné kondenzátory z důvodu přepětí, krystal pro hodinový signál a dále také LED pro kontrolu správné funkce čipu. Následně jsem připojil napájení a uzemnění čipů. Dále bylo potřeba vyřešit, jak bude tato deska připojena k FPGA kitu. Z důvodu tvorby první vývojové desky jsem zvolil 36 pinový GPIO konektor, na který mám vyvedeny všechny důležité signály. Pro testovací účely jsem na desku přidal také další čtyři šestnáctipinové konektory pro snadný debuging. Nakonec jsem přidal konektory pro datové signály včetně ochranných kondenzátorů a ochranných cívek a dalších prvků kvůli rušení signálu.

6.1.1 Výpočet velikosti součástek

Při návrhu schématu desky plošných spojů bylo potřeba zjistit nebo vypočítat hodnoty velikostí jednotlivých součástek. Jsou to filtrační kondenzátory a také rezistory. Většinu součástek jsem nemusel počítat, protože jsem čipy zapojoval podle příkladů v datasheetu [NXP17] nebo jsou pro dané velikosti napětí tyto hodnoty součástek známé.

6.1.2 Výběr součástek

Po dokončení návrhu desky plošných spojů jsem vybral konkrétní součástky, které budou odpovídat pouzdrům, vytvořeným při návrhu desky. Kromě čipů TJA1100, které nemají více variant, jsem vybíral všechny součástky podle požadované hodnoty, typu a ceny součástky. Ochranné prvky proti rušení signálu v automobilu (EMI filtr, cívky a ESD ochranu) jsem zatím pro testovací účely neosazoval, protože tato deska nebude používána v automobilu, kde vznikají tyto rušivé signály. Deska byla osazena keramickými kapacitory typu SMD 0805 a uhlíkovými rezistory také typu SMD 0805.

6.2 Ochranné prvky

Na desce je umístěno několik ochranných prvků, aby nedošlo k poškození desky z důvodu přepětí napájení nebo přepětí na datových vodičích.

- Filtrační kondenzátory
- ESD
- Filtr souhlasného rušení

Kapitola 7

Testování desky

Osazení desky probíhalo ručně v laboratoři pomocí hrotové páječky, cínu a případně lupy. Při pájení jsem si dával pozor na případné studené spoje, u kterých se hledá hůře chyba. Po osazení desky všemi součástkami jsem byl připraven na první připojení desky.

Nejdříve jsem vyzkoušel, co se stane, když připojím napájení 3V3. Očekával jsem, že budu mít napájené čipy TJA1100 a budou fungovat hodinové signály z krystalů. Protože jsem si nebyl jistý, co se stane, když desku připojím k laboratornímu napěťovému, relativně tvrdému zdroji, používal jsem laboratorní zdroj s proudovým omezením. Díky tomuto zdroji jsem si byl jistý, že v případě nějakého problému vyrobenou desku nezničím hned po prvním připojení.

7.1 Oživení desky

Po počátečních přípravách na připojení desky opakovače ke zdroji napětí jsem byl připraven provést první test. Desku jsem úspěšně připojil ke zdroji napětí a multimetrem jsem změřil, jestli jsou na daných místech požadované hodnoty. Následně jsem pomocí osciloskopu změřil, jestli fungují krystaly u obou čipů.

Naměřil jsem tyto hodnoty pro oba dva čipy TJA1100:

- Pin 7 - hodnota 3.3V
- Pin 9 - hodnota 3.3V
- Pin 4 - hodnota 1.8V
- Pin 27 - hodnota 3.3V
- Pin 26 - hodnota GND

- Pin 20 - hodnota GND
- Pin 19 - hodnota 3.3V
- Pin 14 - hodnota 3.3V
- Pin 11 - hodnota 3.3V
- Pin 16 - hodnota 1.8V
- Piny 5 a 6 - hodinový signál o frekvenci 25 MHz

Po tomto testu bylo potřeba ověřit, že konektory na desce jsou správně připojeny. Myslím tím, jestli se moje schéma vytvořené v KiCadu shoduje s realitou. Tento test jsem dělal kvůli vstupům a výstupům vyvedeným na konektory, jejichž připojení k FPGA kitu a k logickému analyzátoru popíši v následující kapitole. Pokud bych měl v programu uvedené něco, co by neodpovídalo realitě, mohlo by se stát, že zničím FPGA kit nebo svou desku, protože by se mohly potkat dva vstupy nebo v horším případě dva výstupy na obou koncích.

Pomocí multimetru jsem proměřil všechny piny všech pěti konektorů, co se týče vstupů a výstupů. Rozdíl mezi vstupy a výstupy se pozná podle odlišného napětí. Na výstupech je nenulové napětí, kdežto na vstupech je napětí nulové. I tento test dopadl dobře.

Všechno fungovalo, jak mělo, a proto jsem přešel k další fázi testování desky, a to sice testování datových signálů. Pro testování této funkce již budu potřebovat jednoduchý program, který zajistí, že si budu moci do jednotlivých vstupů posílat různé signály.

7.2 Chyby

V průběhu testování desky jsem zjistil, že by po zapojení desky k napájení měla začít svítit signalizační LED dioda. LEDka se ale nerozsvítila i přes to, že veškeré hodnoty na desce byly správné a deska fungovala tak, jak má.

Abych zjistil, kde nastala chyba, změřil jsem napětí na pinu TJA1100, který je vyhrazen pro LEDku. Následně jsem změřil napětí na diodě a zjistil, že napětí na diodě bylo téměř nulové, protože jsem při návrhu špatně vypočítal velikost předřadného rezistoru. Původně jsem měl na desce rezistor o velikosti $R = 1000\Omega$. Novou hodnotu rezistoru jsem tedy spočítal znovu. Znám parametry umístěné LED:

$$I_D = 20\,mA,\ U_D = 1.8\,V$$

Velikost napájecího napětí je: $3.3V$.

Nejdříve jsem spočítal velikost napětí na rezistoru rozdílem napětí na diodě a napětí zdroje:

$$U_R = U_Z - U_D$$

Následně jsem pomocí Ohmova zákona dopočítal potřebnou velikost rezistoru:

$$R = \frac{U_R}{I_D} = \frac{1.5\,V}{0.02\,A} = 75\,\Omega$$

Kapitola 8

Základní funkcionalita

Deska je napájena pomocí 3V3 napětí a bylo potřeba otestovat, jestli datové a řídicí propojení fungují tak, jak by měly. Základní funkcionalita mé desky znamená, že data, která přicházejí do prvního čipu TJA1100 přes rozhraní MII, projdou přes druhý čip TJA1100 a následně odcházejí ze zařízení. Také bylo potřeba ověřit, jestli hodinový signál z krystalu vystupuje z čipu na pinech RXC a TXC. Dále jsem zkontroloval další signály jako např. DataValid nebo DataError.

8.1 FPGA kit

V rámci práce jsem pracoval s kitem FPGA. Konkrétně s vývojovou deskou Altera DE-115, na které je umístěn procesor Cyclone IV. Vzhledem k tomu, že moje deska je vývojový kus, bylo propojení s procesorovým kitem pouze dočasné řešení, které je již v současnosti nahrazeno kompaktní deskou s integrovaným procesorem. FPGA kit umožňuje velký rozsah použití. Má spoustu vstupů, výstupů i vstupně výstupních portů. Vstupy:

- Přepínače
- Tlačítka
- Zvukové vstupy

Výstupy:

- LED diody
- Sedmi-segmentové displeje
- Audio výstup

- Displej
- VGA výstup

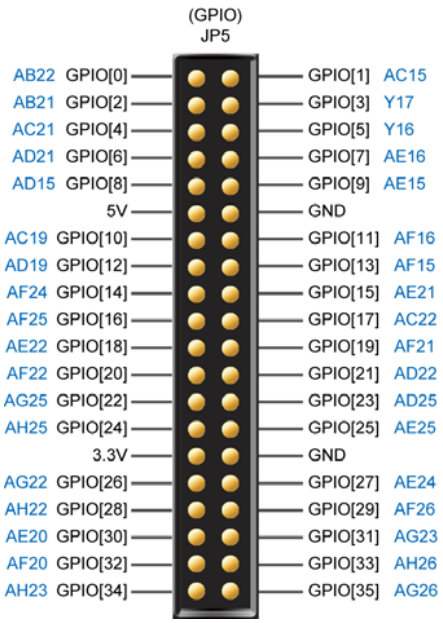
Vstupně-výstupní porty:

- USB port
- RJ45 port
- RS-232
- SDRAM
- Slot pro SD kartu
- GPIO konektor

Z těchto dostupných portů jsem pro své potřeby používal pouze část přepínačů (SW0 - SW2) a GPIO konektor. Při debugování jsem používal i sedmisegmentové displeje.

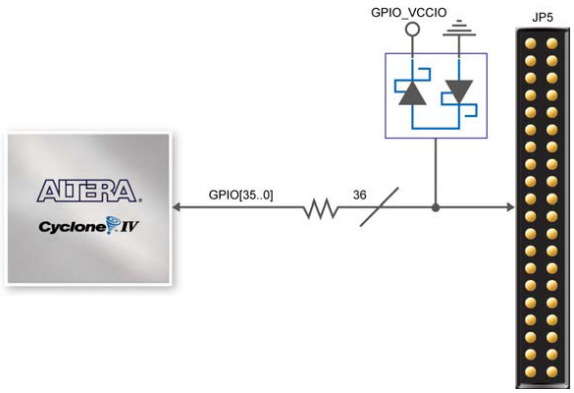
8.1.1 GPIO konektor

GPIO konektor poskytuje 36 vstupně-výstupních datových pinů. Dále poskytuje dva napájecí piny 3,3 V a 5 V, a také dva zemnící GND piny. V případě této práce jsem používal napájecí pin 3,3 V, který má omezený maximální proud na hodnotu 1,5 A. Celkové zapojení a popis jednotlivých pinů je vidět na obr. 8.1.



Obrázek 8.1: GPIO konektor s labely jednotlivých pinů

Všechny datové vstupně-výstupní piny jsou připojeny k procesoru přes dvě diody a rezistor. Toto zapojení poskytuje ochranu proti případnému přepětí nebo podpětí. Ukázka zapojení na jednom pinu je na obr. 8.2.



Obrázek 8.2: GPIO konektor s ukázkou ochrany jednoho pinu

8.2 Zapojení opakovače

Svou desku opakovače jsem připojil k FPGA kitu pomocí 40ti pinového GPIO konektoru, na kterém mám vyvedeny všechny důležité signály pro ovládání a sledování funkce opakovače. Jsou zde datové signály, ale také konfigurační signály pro nastavení správné funkce čipu TJA1100. Vzhledem k tomu, že čip TJA1100 může fungovat i samostatně, není zapojení dvou čipů za sebou tak jednoduché. Výrobce čipu ale s možností tohoto zapojení počítal a pomocí konfiguračních pinů jsem mohl čipy nastavit tak, aby splňovaly mé požadavky a plnily funkci opakovače. Nejdůležitější signály s názvy, piny na procesoru a piny na GPIO konektoru a jejich vlastnosti jsou uvedeny v tab. 8.1.

8.3 Konfigurace PHY

Při spuštění programu a připojení opakovače k napájení bylo v první fázi potřeba nakonfigurovat čipy TJA1100 tak, aby deska jako opakovač fungovala. Toto nastavení se může dělat pomocí hardware i pomocí software. Nastavení čipu jsem prováděl na jeho pinech 17, 18, 21 a 22.

Konkrétně jsem nastavil defaultní konfiguraci PHY během startu nebo po hardwarovém resetu. Tomuto nastavení se říká pin strapping. Pin strapping nastavuje na čipu na prvním pinu, jestli je typu Master nebo Slave Na druhém pinu, jestli bude pracovat autonomně nebo bude jeho aktivita spravována v průběhu aktivního režimu. Nakonec se na dalších dvou pinech konfiguruje různé nastavení MII. Nastavení pinů i s významem můžeme vidět v tab. 8.2

Symbol	Pin GPIO	Funkce
INT_N	GPIO[16,21]	Interrupt výstup
RST_N	GPIO[14,23]	Reset vstup
RXER(CONFIG3)	GPIO[10,27]	MII chybový výstup
RXDV(CONFIG2)	GPIO[12,25]	MII validní data výstup
RXD3(CONFIG1)	GPIO[8,29]	Přijímaná data bit 3
RXD2(CONFIG0)	GPIO[6,31]	Přijímaná data bit 2
RXD1(PHYAD1)	GPIO[4,33]	Přijímaná data bit 1
RXD0(PHYAD0)	GPIO[2,35]	Přijímaná data bit 0
RXC	GPIO[0,34]	Přijímaný hodinový signál
TXC	GPIO[1,32]	Přenášený hodinový vstup
TXEN	GPIO[5,28]	Povolení přenosu
TXD3	GPIO[13,20]	Přenesená data bit 3
TXD2	GPIO[11,22]	Přenesená data bit 2
TXD1	GPIO[9,24]	Přenesená data bit 1
TXD0	GPIO[7,26]	Přenesená data bit 0
TXER	GPIO[3,30]	Chyba přenosu
EN	GPIO[15,18]	Zapnutí fyzické vrstvy čipu TJA1100

Tabulka 8.1: Popis jednotlivých signálů z hlediska TJA1100, z hlediska FPGA kitu i s jejich specifikací.

Před zapnutím opakovače jsem nastavil nastavit tyto čtyři piny tak, aby moje deska fungovala jako opakovač. Stejně tak jsem musel zajistit, aby se provedlo opětovné nastavení při zapnutí resetovacího tlačítka.

Nejprve jsem potřeboval ověřit funkčnost desky, a proto jsem neřešil nastavení pomocí programu. Na desku jsem dále nechal připájet na dané piny pull-up nebo pull-down rezistory. Tyto rezistory při zapnutí napájení desky nastaví požadované hodnoty na daných pinech a nadále budou fungovat jako výstupy pro posílaná data. Pro požadovanou funkčnost desky jsem nastavil konfiguraci pinů na obou čipech podle tabulky 8.3

Z tabulky se dá vyčíst, že na pinu 22 je jeden čip nastaven jako Master a druhý jako Slave. Na pinu 21 jsou oba čipy nastaveny tak, aby pracovaly autonomně. Na posledních dvou pinech jsem nastavil mód MII. Toto nastavení je klíčové pro správnou funkci desky jako opakovače. Abych nastavil desku jako opakovač, musí být jeden čip nastaven jako normální MII mód a druhý čip jako reverzní MII mód. Přesně tak jsem čipy na desce nastavil.

Při rozšíření programu jsem již nepotřeboval pro nastavení pull-up a pull-down rezistory, ale nastavení jsem prováděl pomocí úpravy programu ve VHDL. Konfigurace desky pro požadovanou funkci pomocí kódu VHDL je vidět v kódu 8.1

Pin	Hodnota	Popis funkce
CONFIG0 (pin22)	1	PHY je nastaven jako Master
	0	PHY je nastaven jako Slave
CONFIG1 (pin21)	1	Autonomní činnost
	0	Spravovaná činnost
CONFIF3/2 (pin17/pin18)	0 0	Normální MII mód
	0 1	RMII mód (ext. oscilátor 50MHz)
	1 0	RMII mód (25MHz krystal)
	1 1	Reverzní MII mód

Tabulka 8.2: Pin strapping

Pin	Hodnota čipu 1	Hodnota čipu 2
CONFIG0 (pin22)	1	0
CONFIG1 (pin21)	1	1
CONFIG2 (pin18)	0	1
CONFIG3 (pin17)	0	1

Tabulka 8.3: Nastavení čipů TJA1100 při spouštěcí konfiguraci

Listing 8.1: Počáteční konfigurace pinů čipu pro nastavení jeho funkce.

```
if (SW(2) = '0') then
    RXD1(2)     <=  '1';
    RXD1(3)     <=  '1';
    RXD2(2)     <=  '0';
    RXD2(3)     <=  '1';
    RXDV1_C21   <=  '0';
    RXDV2_C22   <=  '1';
    RXER1_C31   <=  '0';
    RXER2_C32   <=  '1';
```

8.4 Struktura programu

Základní program pro jednoduchou funkci opakovače není složitý. I přesto bych rád rozebral jednotlivé části programu. Program v první fázi definuje jednotlivé vstupy a výstupy opakovače a případně jim nastavujeme defaultní hodnotu. Tato definice je uvedena v Entitě. Následuje architektura, která již zajišťuje posílání dat přes oba čipy TJA1100. Z ukázky kódu v Listings 9.1 můžeme vidět, že data jsou posílána jen v případě, že přepínač SW(2) je přepnutý na hodnotu SW(2) = 0. Toto je základní funkcionalita mého prvního programu pro ovládání funkce opakovače.

Listing 8.2: Základní kód programu pro funkci Repeateru

```
ARCHITECTURE Behavioral OF Repeater IS
begin
process (SW(2)) is
    begin
        -- Prepinac2 vypina a zapina piny Enable
        EN1 <= SW(2);
        EN2 <= SW(2);
        if (SW(2) = '0') then
            RXC2    <= TXC1;
            TXC2    <=  RXC1;
            TXD1(0) <= RXD2(0);
            TXD1(1) <= RXD2(1);
            TXD2(0) <= RXD1(0);
            TXD2(1) <= RXD1(1);
            TXD2(2) <= RXD1(2);
            TXD2(3) <= RXD1(3);
            TXD1(2) <= RXD2(2);
            TXD1(3) <= RXD2(3);
            TXEN2   <= RXDV1_C21;
            TXEN1   <= RXDV2_C22;
            TXER2   <= RXER1_C31;
            TXER1   <= RXER2_C32;
        end if;
end process;
```

8.5 Ověření funkčnosti jednotlivých částí

Ověření funkčnosti ostatních částí desky jsem dělal pomocí jednoduchého základního programu, který popisuji v kapitole 8.4. Abych věděl, jakou mají jednotlivé piny hodnotu, zobrazoval jsem je pomocí logického analyzátoru Tektronix. Právě z důvodu ověřování správné funkce a posílání jednotlivých signálu jsem na desku přidal čtyři 16pinové konektory pro ploché kabely, kterými jsem tyto signály pomocí logického analyzátoru sledoval a analyzoval.

8.5.1 Logický analyzátor

Logický analyzátor ve své práci používám pro vyhodnocení výstupů z desky opakovače. Je to zařízení, ke kterému pomocí plochých kabelů připojil všechny mnou definované signály z desky opakovače. Tyto signály jsem dokázal všechny najednou zobrazit na obrazovce logického analyzátoru v závislosti na čase.

8.6 Programovací prostředí

Pro programování svého projektu jsem zvolil IDE Quartus Prime z důvodu velkého rozšíření a možností tohoto programu. V tomto prostředí jsem již programoval dříve, a proto bylo nejjednodušší se opět vrátit k tomu, co jsem už používal.

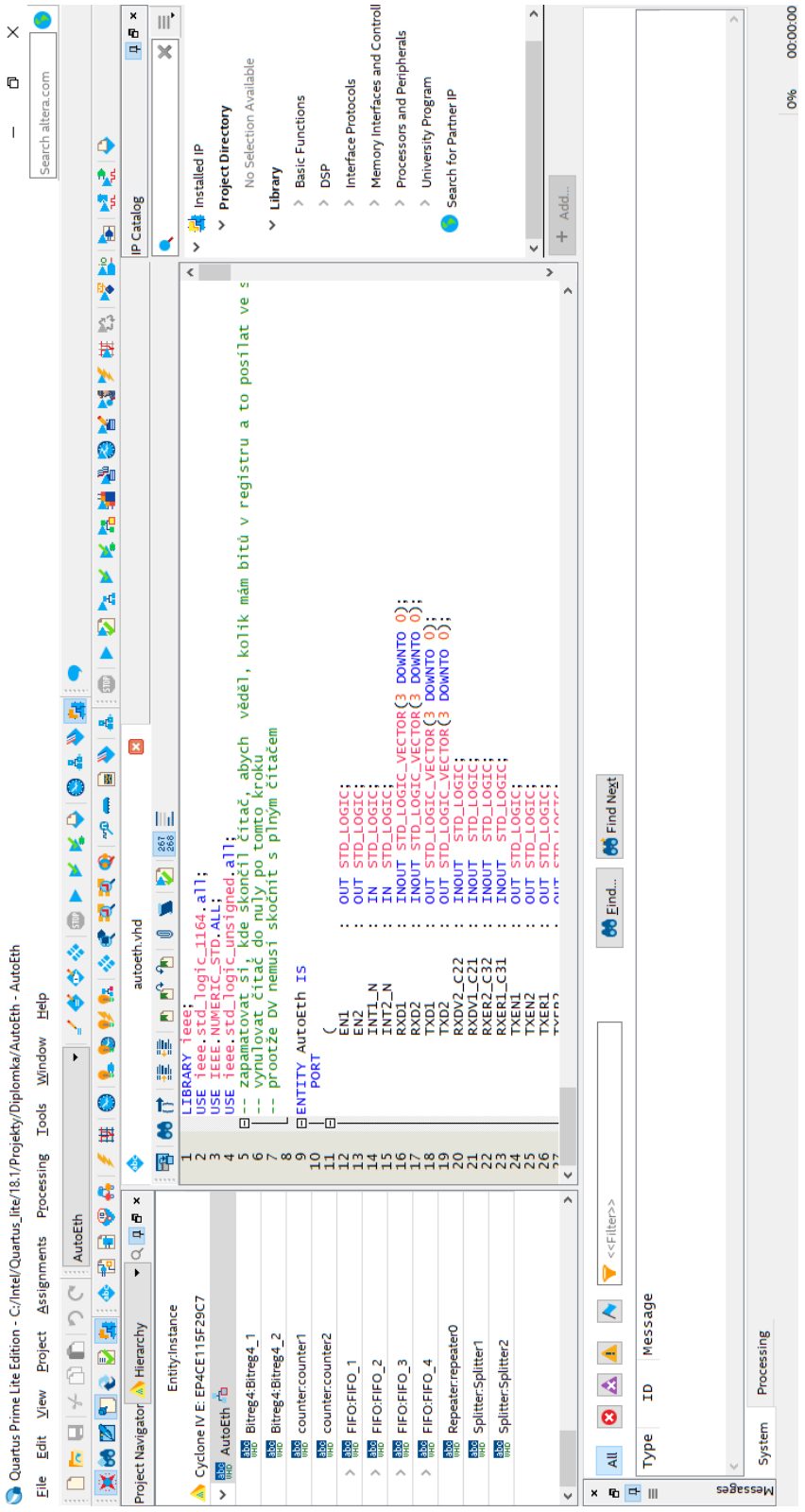
8.6.1 Quartus

Quartus Prime je IDE prostředí od Intelu, který je přímo určen k programování hradlových polí FPGA. Hradlová pole můžeme programovat ve všech nejrozšířenějších jazycích určených k jejich programování (VHDL, Verilog, SystemC, AHDL a další). V rámci Quartusu je k dispozici kompilace designu, časová analýza, RTL diagramy, debuging. Jedna z důležitých součástí kromě časové analýzy je Pin assignment. Je to okno, ve kterém se přiřazují k jednotlivým softwarově vytvořeným portům hardwarové porty na desce s FPGA čipem.

8.6.2 ModelSim

Toto prostředí je součástí Quartusu. Jedná se o simulační prostředí, které používám předtím, než svůj kód nahraji do FPGA kitu. Slouží tedy k odladění chyb v kódu, případně ke kontrole, zdali jednotlivé bloky, entity a architektury programu fungují tak, jak jsem si představoval.

ModelSim šetří spoustu času, protože v případě, že člověk neví, jestli je jeho myšlenka správná, může si implementaci během chvilky vyzkoušet v praxi. Ukázky simulací v programu ModelSim budou k dispozici v praktické části pro konkrétní řešené problémy (například obr. 10.2).



Obrázek 8.3: Základní okno Quartusu

Kapitola 9

Tvorba hlavního programu a simulačního programu

Při psaní kódu jsem celý projekt rozdělil do několika funkčních bloků, které jsou vzájemně propojeny a mají spolu určitou vazbu. Každý tento blok má své vstupy a výstupy, a tudíž je každý blok popsán v samostatném VHDL souboru jako entita. Všechny tyto jednotlivé VHDL soubory jsou propojeny v nadřazeném souboru, který tyto entity spojuje pomocí komponent v jeden projekt.

Z důvodu změny dalšího projektu, který na tuto práci navazuje, jsem v průběhu musel změnit část svého programu, a nakonec jsem pro přehlednost vytvořil dvě verze hlavního programu pro ovládání opakovače. Protože nebylo jasné, jak bude projekt, který zpracovává data z opakovače vypadat. Nejdříve jsem vytvořil jsem projekt, jehož výstupem jsou dvě FIFO paměti. V průběhu tvorby programu se ukázalo, že lepší varianta bude použití dvoubránové RAM a z toho důvodu jsem přestal pracovat na variantě s FIFO pamětí, a dále jsem se věnoval pouze kódu, jehož výstupem budou data, která budu posílat na vstup dvoubránové RAM paměti.

9.1 Funkce programu

Celý program funguje jako standardní opakovač. Program přijme data a vzápětí je posílá druhým portem ven. Data kromě této funkce ještě program zpracuje pomocí registru a z těchto registrů je posílá do dvoubránové paměti RAM. Dvou-branná paměť již není součástí mého programu. Mým úkolem je data pouze připravit pro odesílání. Tato data jsou následně odesílána z opakovače do třetího zařízení, které se stará o monitoring a případnou úpravu těchto dat.

9.2 Části programu

Jak jsem již zmiňoval v úvodu, svůj řídicí program jsem si rozdělil do několika bloků. Každý blok plní jinou funkci ve výsledném programu. V následujících podkapitolách každý blok programu rozeberu a popíšu, aby bylo jasné, k čemu slouží.

Rozdělení na jednotlivé bloky proběhlo i z důvodu větší přehlednosti. Každý jeden blok, který je zároveň samostatným souborem, představuje funkčně odlišnou strukturu. Celkové blokové schéma hlavního programu je pro lepší představu zobrazeno na Obr.A.4 v příloze.

9.2.1 Opakovač

Tento blok vychází z původního základního programu, který pouze posílal data z jednoho čipu do druhého. V této verzi je ale rozšířen o funkci konfigurace pin strapping. V tomto případě je nutné přepínat porty mezi vstupy a výstupy, protože při konfiguraci pinů jsou porty z hlediska programu jako výstupy a následně se musí přepnout na vstupy, aby se mohl přijímat signál z těchto portů. Nakonec je k tomuto bloku připojena funkce filtrace zákmitů na přepínačích. Tato funkce má svůj samostatný blok 9.2.4

Listing 9.1: Základní kód programu pro funkci Repeateru

```
ARCHITECTURE Behavioral OF Repeater IS

component filter
  port(
    clk         : in  std_logic;
    inswitch    : in  std_logic;
    outswitch   : out std_logic);
end component;

    signal sw0f : STD_LOGIC;
    signal sw1f : STD_LOGIC;
    signal sw2f : STD_LOGIC;
begin

rRXC2   <= rTXC1;
rTXC2   <= rRXC1;
rRSTN_1 <= sw0f;
rRSTN_2 <= sw1f;

-- prirazeni kvuli ovladani counteru

rRstCount1 <= sw0f;
```

```
rRstCount2 <= sw1f;

process (rRXC1, sw2f) is -- EN je nastaveno na jednicku
    begin
        rEN1 <= sw2f;
        rEN2 <= sw2f;
-- RST je v nule po celou dobu toho,kdyz probiha pin strapping
--      if (rising_edge(rRXC1) and sw2f = '0') then
--RXD2 a 3 a RXDV a RXER jsou vystupy
            rRXD1(0)        <= '1';
            .
            .
            .
        end if;
        if (rising_edge(rRXC1) and sw2f = '1') then
            rRXD1(0)        <= 'Z';
            .
            .
            .
            rTXD1(0) <= rRXD2(0);
            rTXD1(1) <= rRXD2(1);
            rTXD2(0) <= rRXD1(0);
            rTXD2(1) <= rRXD1(1);
            rTXD2(2) <= rRXD1(2);
            rTXD2(3) <= rRXD1(3);
            rTXD1(2) <= rRXD2(2);
            rTXD1(3) <= rRXD2(3);
            rTXEN2  <= rRXDV1_C21;
            rTXEN1  <= rRXDV2_C22;
            rTXER2  <= rRXER1_C31;
            rTXER1  <= rRXER2_C32;
        end if;
end process;


END architecture;
```

9.2.2 Čítač

Programový blok čítače je v projektu z důvodu čítání počtu přijatých bloků čtyřbitových dat, které jsem dále zpracovával. Jedná se o tříbitový čítač, který se inkrementuje s náběžnou hranou hodinového signálu a v případě splnění podmínky, že přijímám validní data. Tuto informaci čítač posílá dál, abych věděl, kolik bloků dat jsem přijal a abych o velikosti dat mohl informovat registr a paměť RAM. Čítače mám v projektu dva. Pro každý tok dat jeden.

Listing 9.2: Hlavní část kódu čítače

```
process (clock, DValid, rst)
begin
    if (DValid = '0' or rst = '0') then
        tmp <= "000";
    elsif (rising_edge(clock) and DValid = '1') then
        tmp <= tmp + 1;
    end if;
end process;
Numb <= tmp;
```

9.2.3 Datový registr

Datové registry mám ve svém kódu také dva, stejně jako v případě čítače a ze stejného důvodu. Z jednoho čipu ukládám data do prvního registru a z druhého čipu do druhého registru. Registr používám k ukládání dat. Slouží jako mezistupeň předtím, než data začnu posílat do paměti RAM. Jako všechny bloky je i tento synchronizovaný hodinami z krystalu opakovače o frekvenci 25 MHz. Do registru ukládám postupně data podle hodnoty, která je uložená v čítači. Při každém zvýšení hodnoty čítače se uloží hodnoty v datech do signálu Q. Mám tedy k dispozici 32bitový registr, protože při každém pulzu hodin přijímám 4 bity dat, a to mohu opakovat osmkrát.

Osm bloků dat, které ukládám do signálu Q následně spojuji do jednoho 32bitového vektoru, pomocí kterého posílám data do dual ported RAM.

Kromě ukládání dat slouží registr k zapamatování si toho, kolik bajtů validních dat jsem přijal. Informace o počtu bajtů je v registru také nulována. Nulování probíhá při pulzu, který přijde z dual-ported RAM. Tato informace je pro mě důležitá z důvodu informování paměti RAM o tom, kolik dat se má zpracovat.

Součástí tohoto bloku je také čítač, který počítá, kolik 32bitových bloků dat jsem poslal. Tento čítač je zde z důvodu adresace paměti RAM a také, abych věděl jak je paměť zaplněná. V ukázkách bloku registru mohu vidět přiřazení dat na výstup podle toho, v jakém stavu je čítač.

Listing 9.3: Hlavní část kódu registru

```
if rising_edge(clock) and ValidD = '1' and rst = '1'
                      and data_width = 32 then
-- sample on rising edge of CLK
    case (Count) is
        when "000" => tmpQ1 <= D;
        when "001" => tmpQ2 <= D;
        when "010" => tmpQ3 <= D;
        when "011" => tmpQ4 <= D;
        when "100" => tmpQ5 <= D;
        when "101" => tmpQ6 <= D;
        when "110" => tmpQ7 <= D;
        when "111" => tmpQ8 <= D;
        when others => tmpQ1<= D;
    end case;
    tmpadr <= tmpadr + 1;
end if;
```

V bloku registru je implementovaná ještě funkce přepínání posílání dat na první buffer paměti nebo na druhý buffer paměti. K přepnutí aktivního bufferu dochází v případě, že v průběhu posílání dat spadne signál DataValid do nuly. Při opětovném přepnutí signálu DataValid do jedničky se přepne aktivní buffer a vynuluje se adresní signál.

Listing 9.4: Signály pro přepínání bufferů RAM

```
if rising_edge(clock) and tmpadr = "100000000000" then
-- kdyz se zapisuje do prvniho portu RAM-flag = 0
            flag <= not flag;
        end if;
        -- Zapinani a vypinani portu
        if flag = '0' then
            Port0_en <= '1';
        else
            Port0_en <= '0';
        end if;

        if flag = '1' then
            Port1_en <= '1';
        else
            Port1_en <= '0';
        end if;
```

Součástí tohoto bloku programu jsou ještě další procesy, které zajišťují, aby k začátku a konci čtení dat docházelo ve správný okamžik. S tím souvisí i správné nastavení a ovládání signálu write enable, který ve stavu 1 signalizuje data, která jsou připravena k odeslání. Jsou zde také další signalizační porty

a signály, které jsou důležité pro správnou funkci celého programu. V ukázce 9.5 vidíme přepínání signálu write enable a dále inkrementaci adresy, kterou posílám do dalšího bloku programu.

Listing 9.5: Write enable

```
    if ackw = '0' and (write_flag = '1') then
        write_en <= '1';
        write_enHelp <= '1';
        if ackdelay = '1' then
            numWords <= tmpadr;
            tmpadr <= tmpadr + 1;
            dataAdr <= tmpadr;
        end if;
    elsif ackw = '1' or writeReg = '0'  then
        write_en <= '0';
        write_enHelp <= '0';
    end if;

    if falling_edge(clock) and ValidD_d = '1' then
        tmpadr <= "000000000000";
        numWords <= "000000000000";
    end if;
```

9.2.4 Filtrace

Blok filtrace je malý blok, který má na starost filtraci zákmitů na vstupech přepínačů v případě jejich přepínání z jednoho stavu do druhého. Tato funkcionalita zajišťuje to, že se přepínač přepne do daného stavu až ve chvíli, kdy nejsou na vstupu žádné zákmity a tím pádem může opakovač začít fungovat tak, jak bylo definováno.

Listing 9.6: Filtr

```
outswitch <= state;      -- Prirazeni hodnoty na vystup

 process(clk)
    begin
   if(rising_edge(clk)) then
    If(inswitch /= state and counter > 0) then
       counter <= counter- 1;
     elsif counter = 0 then
     -- Nacteni hodnoty prepinace do "registru"
         state   <= inswitch;
         counter <= limitTime;
       -- Reset counteru if se switch nezmenil
       else
```

```
          counter <= limitTime;
      end if;
    end if;
  end process;
```

9.2.5 Registr velikosti dat

Tento blok programu slouží k informaci o velikosti jednotlivých datových slov. Do výstupního vektoru jsem uložil, kolik bajtů jsem do daného slova uložil. Tuto informaci jsem bral z horních dvou bitů čítače, který čítá počet čtyřbitových bloků dat a dále s informací o počtu 32bitových slov. Vzhledem k velikosti dat v ethernetovém rámci je tato informace velikosti 11 bitů.

Listing 9.7: Adresní registr

```
process (clock)
 begin
     if falling_edge(clock) and writeBytes = '1' then
         tmpcount <= Count(2 downto 1);-- + 1;
     end if;
     if wrPulse = '0' and byteRead = '1' then
         tmpsize(1 downto 0) <= tmpcount;
         tmpsize(10 downto 2) <= tmpadr(8 downto 0);
         tmpsize(30 downto 11) <= zeros;
         tmpsize(31) <= '1';
     elsif falling_edge(clock) and ack_del = '1' then
         tmpsize <= "00000000000000000000000000000000";
     end if;
     if tmpsize = "00000000000000000000000000000000" then
         datasize <= tmpsize;
     end if;
     if falling_edge(clock) and ackw = '1'
                 and byteRead = '0' then
         datasize <= tmpsize;
     end if;
 end process;
 write_test <= write_en;
```

9.2.6 Výstup dat do RAM

Jak již bylo zmíněno, v programu, který je součástí této práce, neukládám data do dvoubránové RAM. Tato data pouze připravuji na uložení do paměti. Pro ověření, že data a veškeré další informace budou přijaty do RAM, jak jsem zamýšlel, vytvořil jsem si pomocné bloky, které zpracovávají data a adresy pro RAM.

9.2.7 Nadřazený program

Nadřazený program slouží k propojení všech bloků do jednoho celku. Jedná se tedy o entitu, která definuje jednotlivé komponenty, se kterými bude pracovat. Následně je v architektuře nadřazeného programu zajištěno namapování jednotlivých portů. Ukázka mapování portů je zde:9.8.

Listing 9.8: Mapování portů u prvního registru

```
Bitreg4_1:Bitreg4
Port map   (
            clock => TXC1,
            Count => iCount1,
            D => RXD1,
            Q => DataSplit1,
            Err => RXER1_C31,
            ValidD => RXDV1_C21
            );
```

9.3 Simulační program

Posledním článkem celého programu bylo vytvoření simulačního programu pro simulaci programu se všemi bloky. Nejdříve byl můj testovací kód velice obsáhlý, protože jsem se simulacemi neměl žádné zkušenosti a všechno jsem se učil v průběhu práce. Nakonec jsem ale zjistil, že pro potřeby tohoto projektu postačí jednoduchý testovací kód, který jsem vyladil do podoby, jehož hlavní část je zobrazena v ukázce kódu 9.9. Simulaci programu rozebírám do detailu v následující kapitole 10.

Listing 9.9: Simulační program

```
Bitreg4_1:Bitreg4
    signals     EN1, tINT1_N, tRXDV2_C22, ... : std_logic;
    signal  tRQ1, ..., tRQ4 : std_logic_vector (31 downto 0);

    begin
    AutoEth_test:AutoEth
```

```
        port map (
            EN1     =>  EN1,
            EN2     =>  EN2,
            INT1_N  =>  tINT1_N,
            INT2_N  =>  tINT2_N,
            RXD1    =>  tRXD1,
            .   -- Continue the same as before
            .
            .
            );
-- CLOCK
    tRXC1 <= not tRXC1 after 25 NS;
    tTXC1 <= not tTXC1 after 25 NS;

process
   begin
         wait for 40 ns;
         tSW(0) <= '0';
         tSW(1) <= '0';
         tSW(2) <= '1';

         wait for 120 ns;
         tSW(0) <= '1';
         tSW(1) <= '1';
        wait;
end process;

process
begin
            wait for 100 ns;
            tRXDV1_C21  <=  '0';
            tRXDV2_C22  <=  '0';
            tRXER1_C31  <=  '0';
            tRXER2_C32  <=  '0';
            wait for 1760 ns;
            tRXDV1_C21  <=  '1';
            tRXDV2_C22  <=  '1';
            wait for 2300 ns;
            tRXDV2_C22  <=  '0';
end process;

    tRXD1(0)        <= not tRXD1(0) after 100 ns;
    tRXD1(1)        <=  not tRXD1(1) after 150 ns;
    tRXD2(0)        <=  not tRXD2(0) after 50 ns;
    tRXD2(1)        <=  not tRXD2(1) after 300 ns;
    tRXD1(2)        <= not tRXD1(2) after 100 ns;
```

```
    tRXD1(3)        <=  not tRXD1(3) after 200 ns;
    tRXD2(2)        <=  not tRXD2(2) after 100 ns;
    tRXD2(3)        <=  not tRXD2(3) after 250 ns;
end bench;
```

9.4 Původní verze programu s FIFO pamětí

Původní zadání projektu bylo s přítomností FIFO paměti, která byla přímo implementovaná v mé části práce. Z tohoto důvodu mám naimplementovanou i variantu, kde používám FIFO paměť pro ukládání dat z opakovače. Protože se zadání změnilo, uvádím zde tuto variantu spíše pro zajímavost, jak zpracování dat vypadá v případě ukládání do FIFO paměti.

9.4.1 Bloky programu s FIFO pamětí

Splitter

Splitter je blok programu, který zajišťuje ukládání dat do jednotlivých FIFO pamětí. Protože FIFO paměť již posílá data ven a nějakou dobu trvá, než se naplní a následně vyprázdní, musím mít pro každý tok dat dvě FIFO paměti mezi kterými budu přepínat. Toto přepínání posílání dat mezi FIFO paměťmi zajišťuje právě splitter. Funguje tak, že při naplnění první paměti nebo při skončení přenosu dat v případě, že splitter přepne výstup, přes který se posílají data, na vstup druhé paměti.

Listing 9.10: Splitter

```
process (F1)
begin
--Full(F1) and Empty(E1)
    if F1 = '1' and E1 = '0' then
        Q2 <= D;
    elsif F2 = '1' and E2 = '0' then
        Q1 <= D;
    end if;
end process;
```

FIFO

Poslední blok kódu je samotná FIFO paměť. Vzhledem k větší komplexnosti celého tohoto bloku a doporučení Intelu jsem většinu kódu pro FIFO paměť

použil template od Intelu ve Quartusu, který jsem následně upravil pro své potřeby. FIFO paměti mám v programu celkem čtyři. Jak již bylo zmíněno, pro každý tok dat jsem použil dvě FIFO paměti a mezi nimi podle potřeby a naplněnosti paměti přepínám.

V paměti mám port pro příjem dat. Dále také příznaky, které indikují plnou, respektive prázdnou paměť. Vzhledem k povaze FIFO paměti, která funguje tak, že první prvek, který do paměti uložím z něj i poprvé odchází, nepotřebuji jednotlivé bloky (paměťová slova) adresovat. Oproti druhé variantě programu mám tedy tuto fázi programu jednodušší na implementaci. V případě ukládání dat budu mít ale jednodušší další části programu. Každá moje FIFO paměť má velikost pro 32 32bitových slov.

Kapitola 10

Testování funkčnosti programu v praxi

10.1 Testování programu v ModelSimu

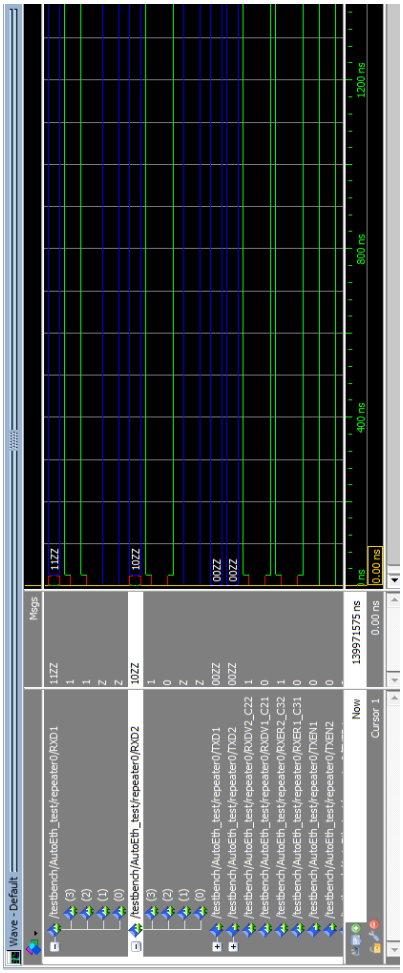
Celý program jsem testoval v simulačním programu ModelSim z důvodu snadnějšího a rychlejšího debugování než v případě, že bych musel pokaždé nahrát program do kitu s FPGA. Simulace byla prováděna také z důvodu bezpečnosti pro kit FPGA a pro desku opakovače. V případě špatného naprogramování nic nezničím a stačí program opravit a znovu odsimulovat.

Testování celého programu jsem rozdělil do celkem sedmi částí. O každé části budu psát v následujících podkapitolách. Každá se zabývá konkrétním problémem a oblastí v rámci programu. Většinou jsou jednotlivé části testování spojené i s jednotlivými bloky programu, o kterých jsem psal v kapitole 9.2. V této kapitole budu psát hodně o jednotlivých signálech, které ve svém programu používám. Pro přehlednost jsou významy všech signálů uvedeny v příloze.

10.1.1 Simulace programu Opakovače

První simulace se týkala pin-strappingu, funkčnosti hodinového signálu, přepínačů a propojení datových signálů mezi TJA budiči. Simulace probíhala tak, že jsem pomocí simulačního kódu v programu Quartus vygeneroval hodinový signál a nastavil na jednotlivých datových portech hodnoty, jako při prvotním nastavení opakovače. Následně jsem manuálně měnil jednotlivé hodnoty přepínačů a sledoval jsem, jestli se v souvislosti s těmito změnami mění i další hodnoty, které by se v souvislosti se změnou přepínačů měly měnit.

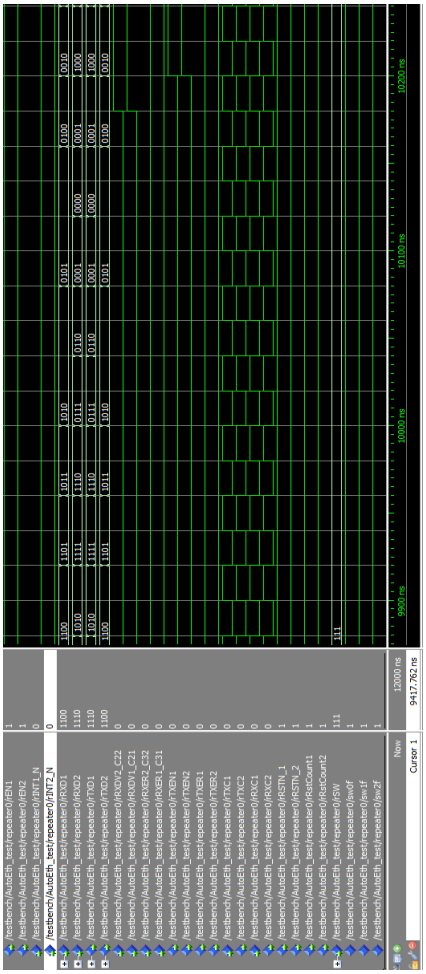
Nejdůležitější částí bylo ověření funkčnosti autokonfigurace při startu (pin-strapping) a následné ověření funkčnosti posílání dat. Autokonfigurace je blíže popsaná v kap. 8.3. Konfiguraci provádím při aktivním resetu čipu (tj. RSTN_1 a RSTN_2 jsou nastavené na nulu). Jak toto nastavení vypadá v



Obrázek 10.1: Pin strapping

ModelSimu, je vidět na obr. 10.1. Při posílání dat jsem narazil na problém. Protože první test odhalil, že se data neposílala mezi jednotlivými bloky a tím pádem jsem data nemohl posílat na výstup pro čtení RAM. Problém byl ve špatné inkrementaci čítače a špatně nastaveným příznakem DataValid.

Ukázka simulace posílání dat je zobrazena na obrázku 10.2. Můžeme vidět, že data z portu RXD1 jsou posílaná do portu TXD2 a data z portu RXD2 jsou posílaná do portu TXD1. Data tedy posíláme přes budiče TJA1100 na opačné strany, tak jak od opakovače očekáváme. Můžeme zde vidět také správné posílání signálů RXDV a RXER, které se také posílají z jednoho směru na druhý a obráceně.



Obrázek 10.2: Ukázka posílání dat v opakovači

10.1.2 Datový registr

V bloku programu datového registru potřebuji zajistit správné ukládání dat do jednotlivých čtyřbitových signálů, které následně ukládám do 32bitového výstupního portu. Současně s tím při každém načtení dat do 32bitového portu inkrementuji dva čítače. Jeden adresní a druhý bajtový, které následně posílám do dalšího bloku programu. O čítačích se budu více zmiňovat v podkapitole 10.1.3.

V rámci této části simulace jsem kontroloval, jestli jsou vstupní data správně ukládaná do dočasných čtyřbitových signálů a následně posílaná do výstupního portu Q v případě, že jsou data připravena (signál $write_en$ je 1). Simuloval jsem zde také potvrzovací signál, který posílá zpět do mého programu operační paměť RAM. V případě, že potvrzovací signál přijde, posílá se další blok dat na další adresu v paměti.

Probíhá tedy ještě kontrola zápisu na správnou adresu v paměti. V rámci simulace posílám rámec o velikosti 64 bajtů (obr. 10.3a).

Následně jsem zkoušel, jak bude program fungovat v případě, že nebudu posílat rámec o velikosti, která je dělitelná čtyřmi. Tuto funkcionalitu zkouším z toho důvodu, že registr, do kterého ukládám data je velký čtyři bajty. Chci tedy otestovat, jak se bude program chovat v případě, že budu posílat jen část validních dat z registru. Funkcionalitu zkouším pro 65 (obr. 10.3b), 66 (obr. 10.4a) a 67 bajtů (obr. 10.4b). Ve všech případech můžeme vidět signál datasize, který po zapsání všech bajtů a spadnutí signálu validD zapíše na výstup celkový počet bajtů, který byl poslaný do operační paměti. Program tedy funguje správně i pro velikosti dat, kdy bajty nejsou dělitelné čtyřmi.

10.1.3 Čítače

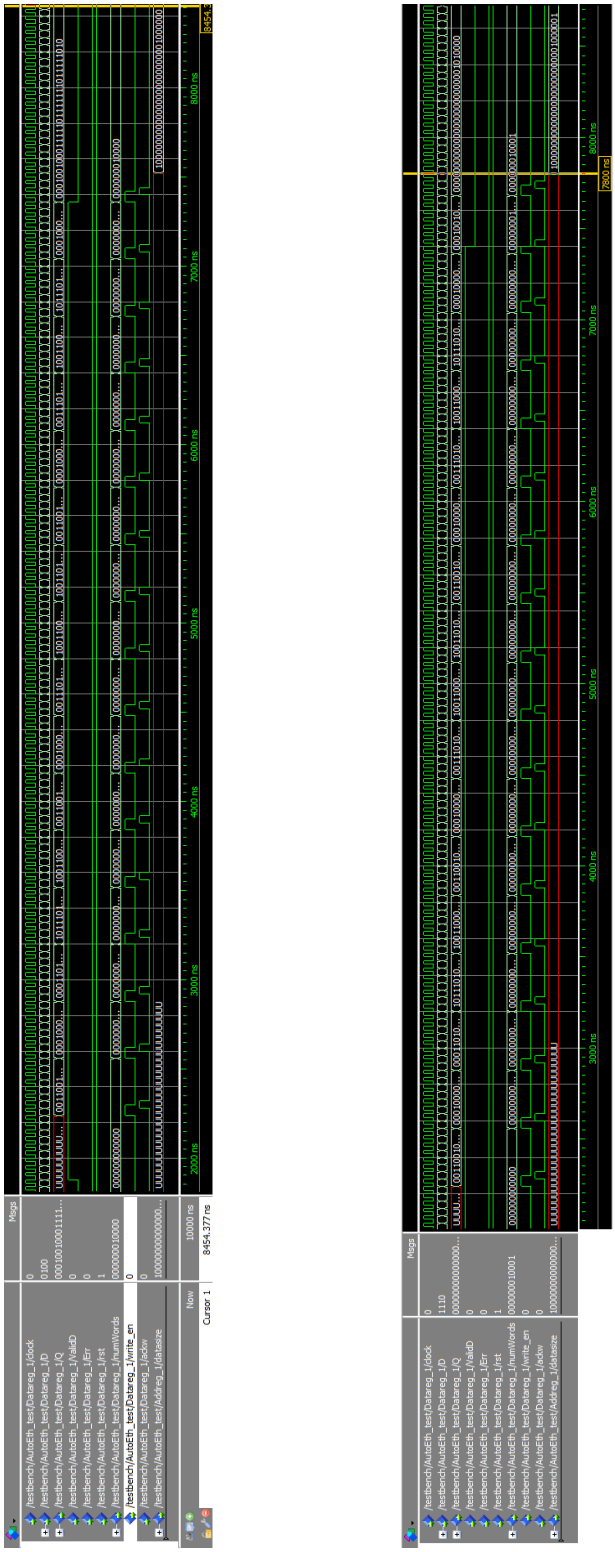
Tato část simulace je součástí několika bloků. Jeden čítač má přímo svůj blok a další čítače jsou součástí datového registru, protože potřebuji počítat současně se zápisem dat jednotlivé bloky a byla pro mě tedy tato varianta přehlednější.

Čítač přijatých bloků dat

Tento čítač má samostatný blok v programu. Jedná se o tříbitový čítač a čítá vždy, když program přijímá validní data a zároveň, pokud není aktivní signál Err. Počítá do osmi, protože do registru ukládám celkem osm čtyřbitových bloků dat. V obrázku 10.5a ze simulace můžeme vidět, že pokud je signál ValidD = ’1’ a signál Err = ’0’ potom se čítač inkrementuje při každém příjmu čtyř bitů dat.

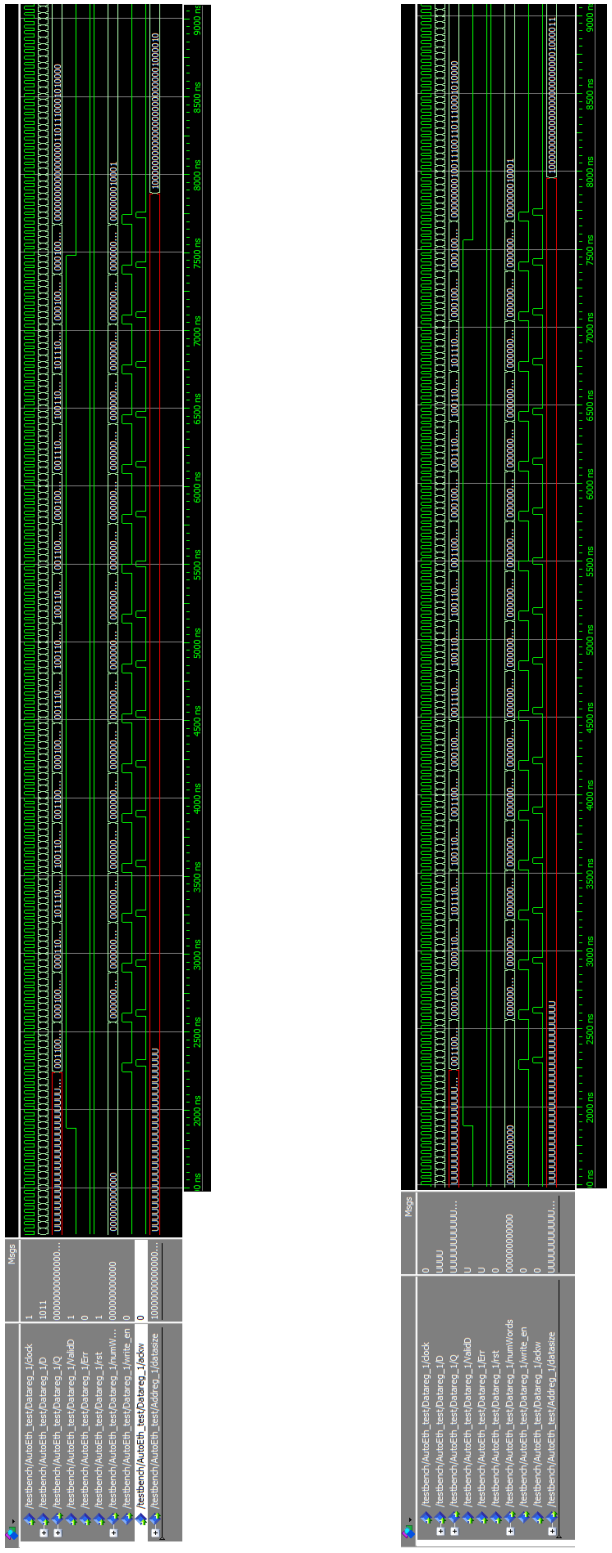
Čítače v rámci datového registru

V bloku datového registru jsou další čítače. První, dvanáctibitový čítač, slouží pro načítání správné adresy pro ukládání dat do operační paměti. Po každém zápisu do operační paměti a následném příchozím signálu ACK zvýším čítač o jedničku. Další čítače jsou zde pouze interně pro nastavování a zpožďování různých pomocných signálů.



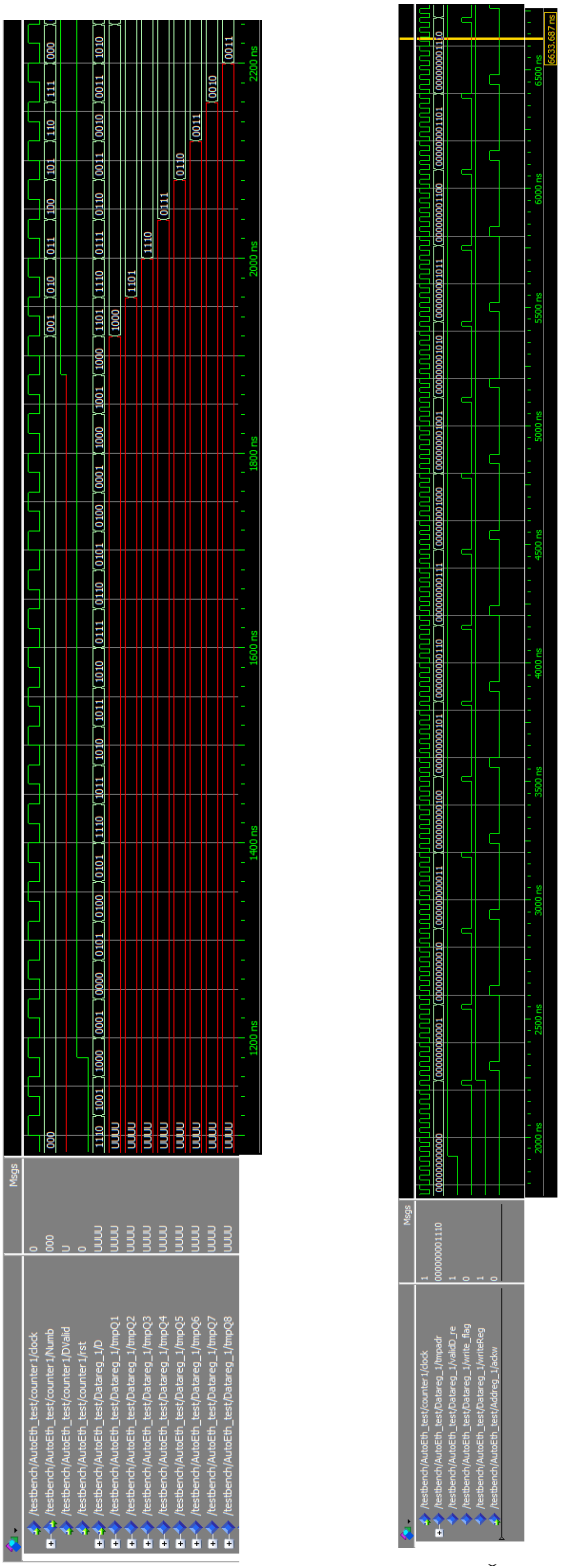
(a) : 64 bajtů dat **(b) :** 65 bajtů dat

Obrázek 10.3: Posílání dat v simulaci ModelSimu



(a) : 66 bajtů dat **(b) :** 67 bajtů dat

Obrázek 10.4: Posílání dat v simulaci ModelSimu



(a) : Čítač přijatých bloků dat

(b) : Čítač adresy do operační paměti

Obrázek 10.5: Čítače v simulaci

10.1.4 Přepínání bufferů RAM

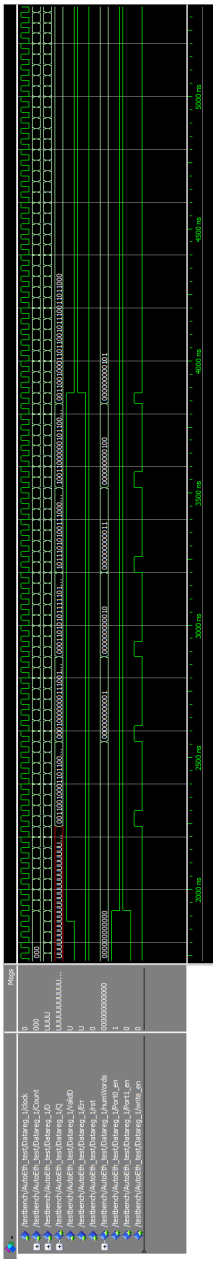
Přepínání bufferů je součástí bloku registru dat. Testování tohoto bloku se týká přepínání aktivního bufferu v operační paměti. Můj program tedy poskytuje pouze dva příznaky pro přepínání bufferů dvouportové paměti. Samotné přepínání není součástí tohoto programu. Jak již bylo řečeno v předchozí kapitole, buffer se přepíná v případě přepnutí signálu DataValid z nuly na jedničku. Současně s přepnutím aktivního bufferu by se správně měla nastavit adresa, na kterou se bude ukládat první blok dat, do nuly. Na obrázku 10.7a vidíme, že při přepnutí ValiD z nuly do jedničky dochází k přepnutí aktivního bufferu a zároveň dochází i k vynulování adresy.

10.1.5 Testování filtrů

Testbench filtrů nám ukazuje, že při přepnutí přepínače z jednoho stavu do druhého se výsledný filtrovaný signál přepne na výstup, který ovlivňuje další části programu, až po jedné mikrosekundě. Filtrace funguje stejně i v případě přepnutí přepínačů z jedničky do nuly.

10.1.6 Chování programu při změně RXDV a RXER

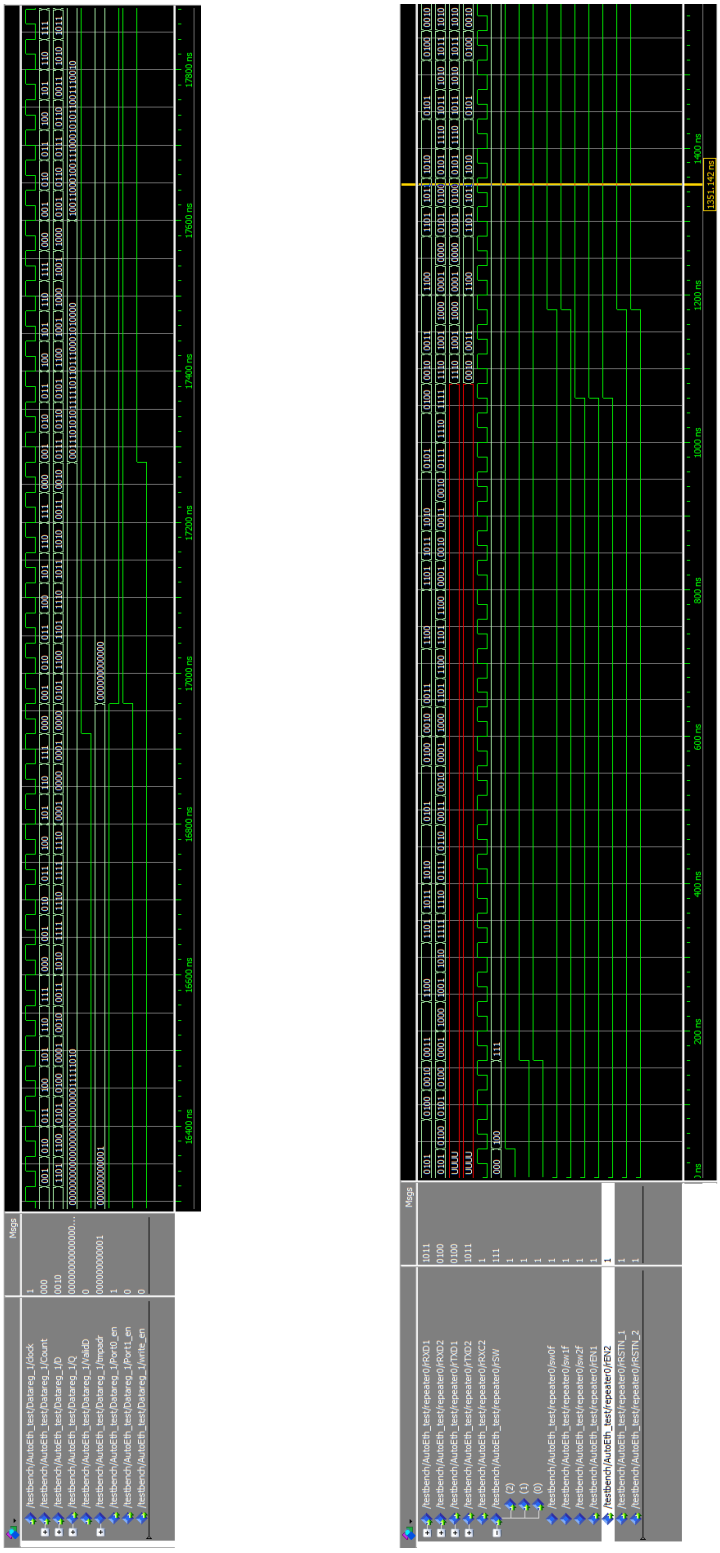
V této podkapitole testuji, jak se program chová v případě přepnutí signálu Err do jedničky. Jak můžeme vidět na obrázku 10.6, v případě přepnutí signálu Err do jedničky se přestane inkrementovat adresa na kterou se zapisují data do paměti. Také se přestanou posílat data přes výstupní port Q a signál $write_en$, spadne do nuly.



Obrázek 10.6: Chování při aktivním chybovém vstupu

10.2 Testování na FPGA kitu

Testování programu na FPGA kitu se mi z časových důvodů nepodařilo stihnout a proto není součástí této práce. Rád bych ale na této práci pokračoval dál a reálné testování na kitu FPGA dokončil.



(a) : Přepínání bufferů pro operační paměť

(b) : Filtrování zákmitů na přepínačích

Obrázek 10.7: Filtrace a přepínání bufferů

Kapitola 11

Závěr

Hlavní úkol této diplomové práce je obsažen její praktické části. Tuto praktickou část práce jsem podpořil teoretickým úvodem, ve kterém jsem se zabýval převážně fyzickou vrstvou standardu 100Base-T1 a BroadR-Reach z důvodu, že na této vrstvě je změna oproti klasickém Ethernetu největší. Zmínil jsem se také o využití AFDX v leteckém průmyslu, jak tento standard funguje a jak se liší od klasického Ethernetu. Dále jsem popsal funkci čipu TJA1100 od firmy NXP, se kterým pracuji v praktické části a který je součástí funkčního vzoru opakovače.

V rámci praktické části diplomové práce jsem se nejdříve věnoval hardwaru. Začal jsem osazením desky součástkami, pokračoval volbou velikosti jednotlivých součástek a následně jsem oživoval a testoval funkční vzor opakovače. Dále jsem se věnoval popisu vytváření programu pro monitorování komunikace, na který zabral nejvíce času při tvorbě práce. Postup při tvorbě programu byl následující. Nejdříve jsem si vytvořil jednoduchý program, který pouze zprostředkovával komunikaci mezi dvěma zařízeními a následně jsem tento kód rozšířil o další funkce, které zajišťují ukládání dat do paměti a posílaní do dvoubránové RAM. Tuto část jsem popsal detailně včetně ukázek kódů a ukázek chování programu v simulačním prostředí. Na tuto práci navazují další lidé, kteří ji budou doplňovat dalšími prvky.

Cíle a motivace stanovené před zahájením diplomové práce:

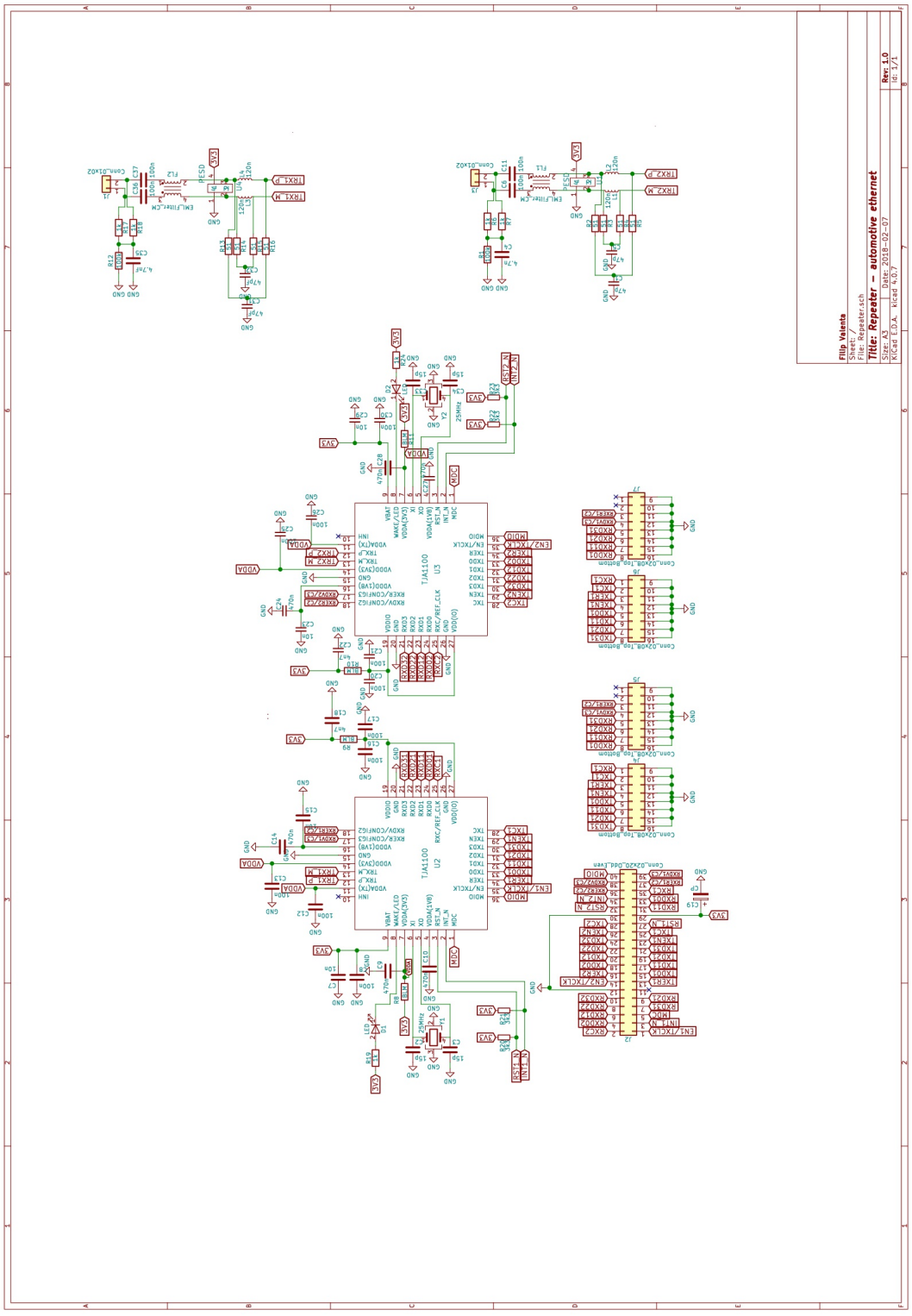
- Seznámit se s technologií 100Base-T1.
- Navrhnout koncepci zařízení pro pasivní monitoring komunikace.
- Realizovat HW funkčního vzoru.
- Implementovat programové vybavení funkčního vzoru v jazyce VHDL.

V rámci diplomové práce se mi podařilo splnit všechny cíle, které jsem si před zahájením stanovil.

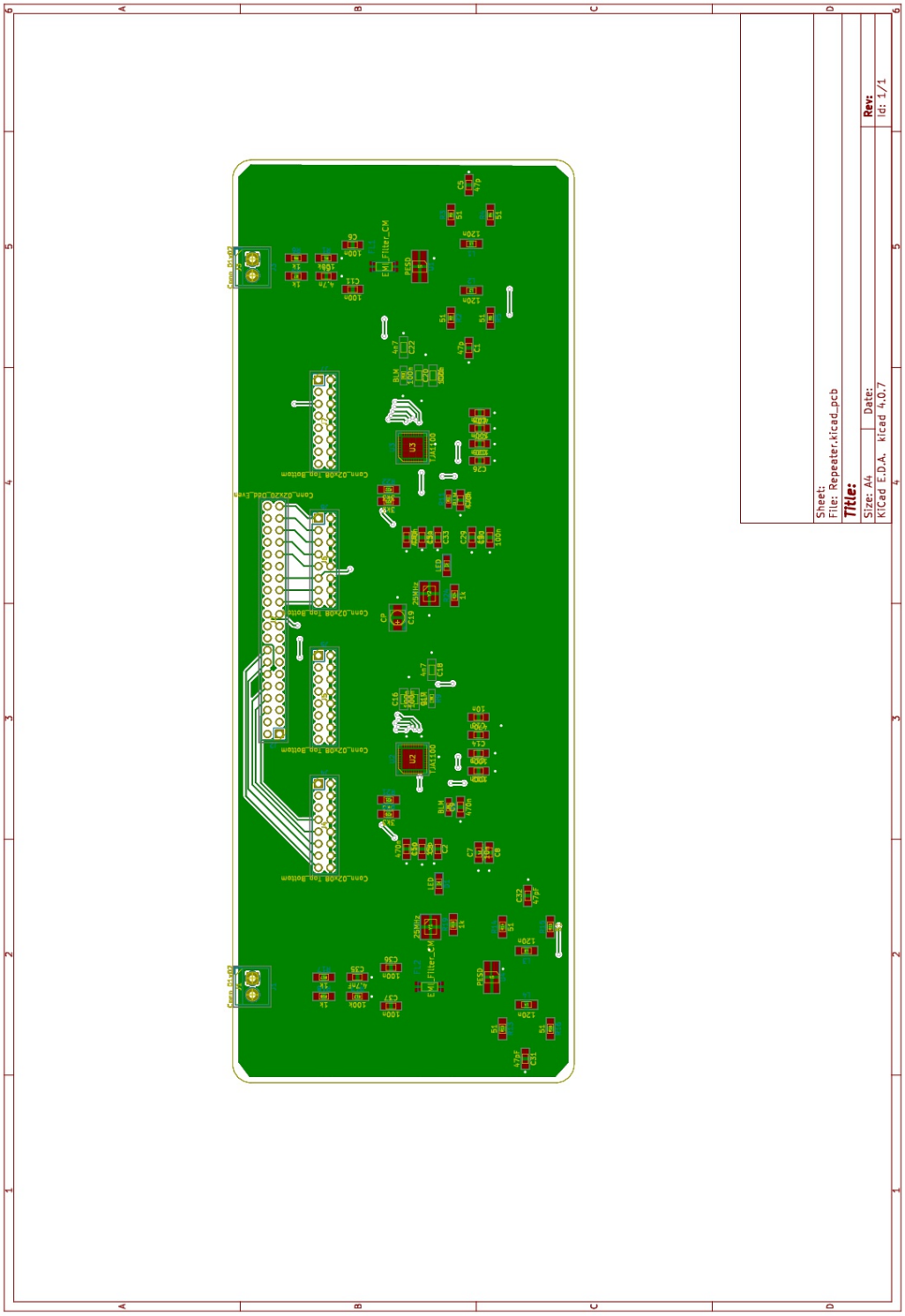
Přílohy

Příloha A

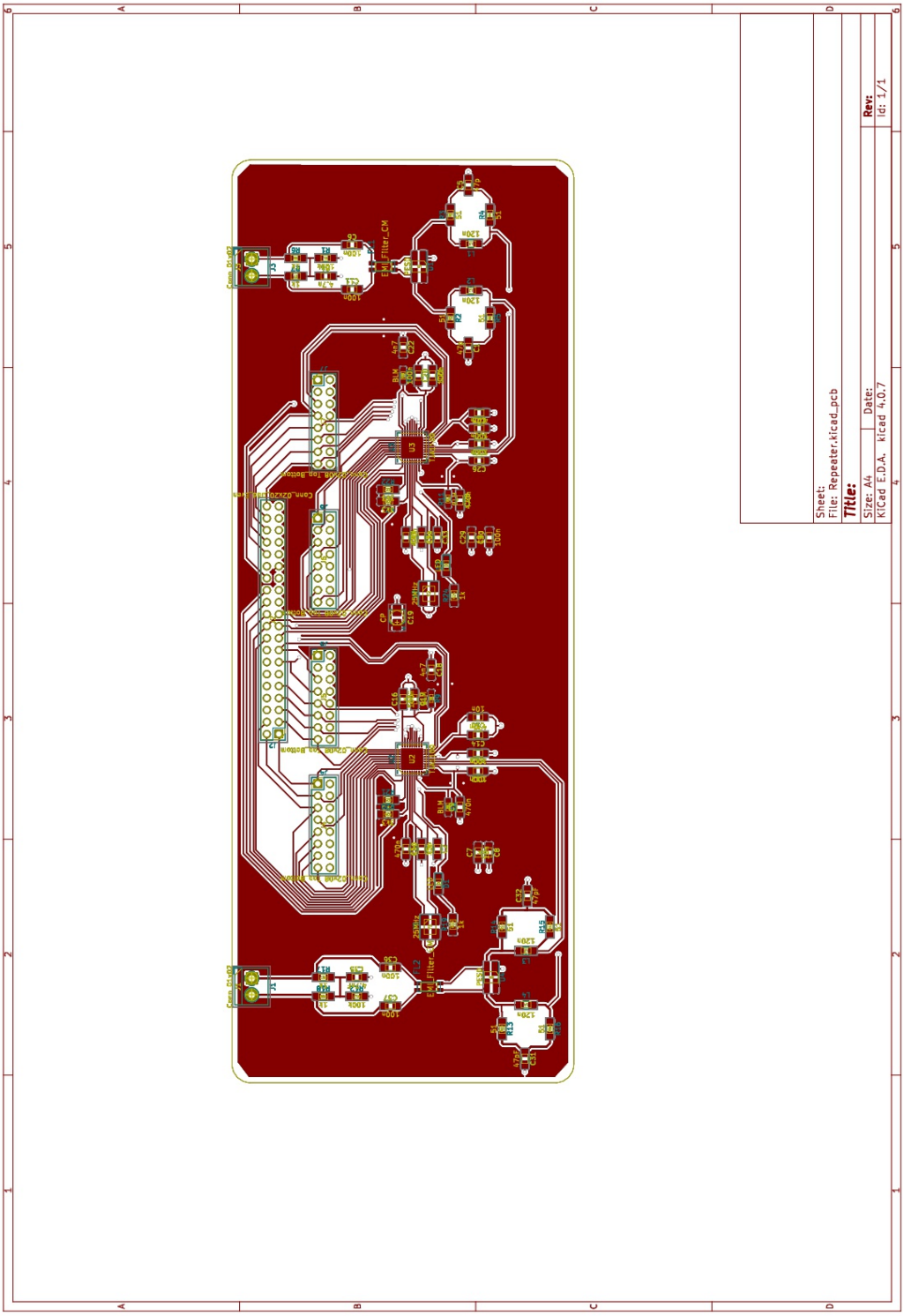
Schémata



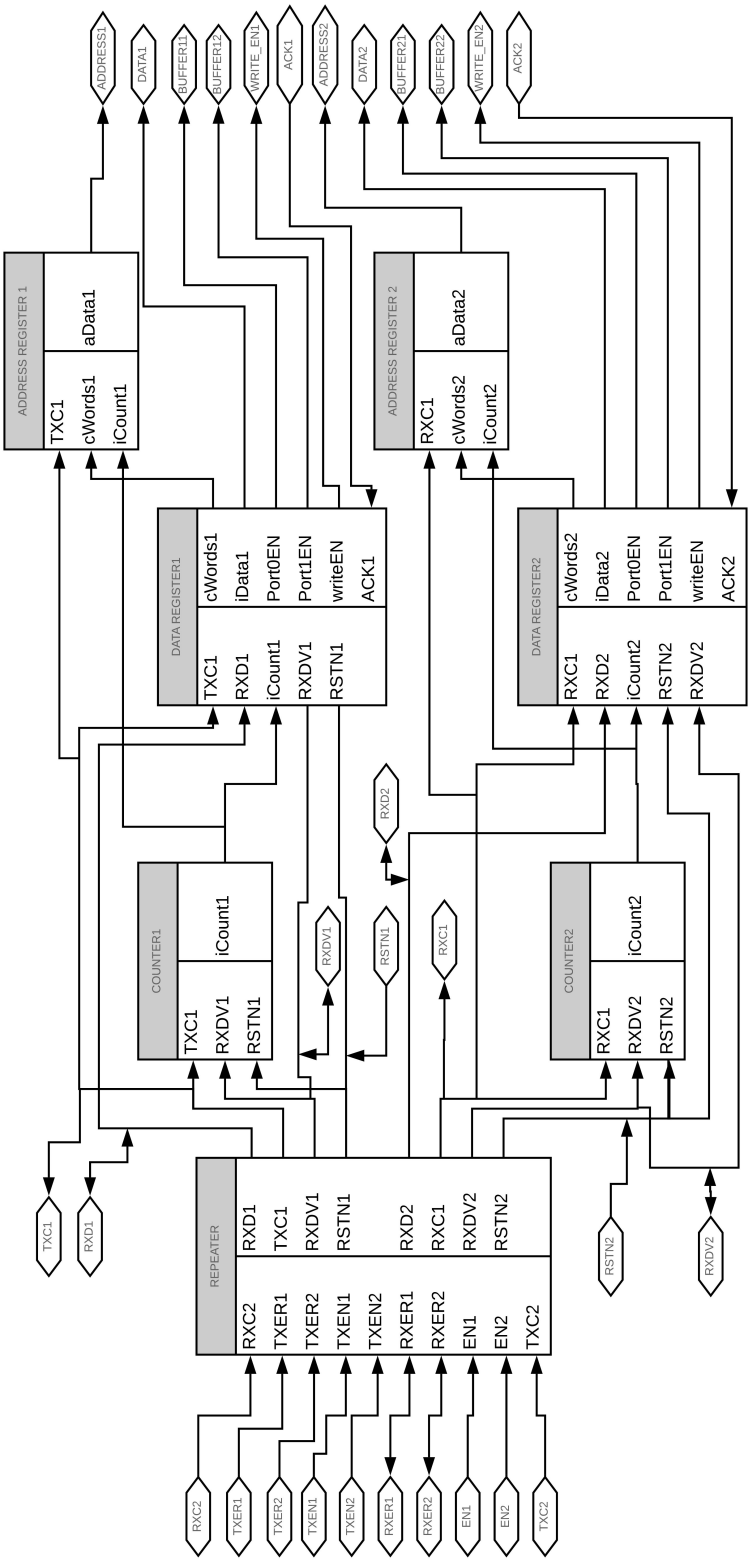
Obrázek A.1: Schéma desky opakovače



Obrázek A.2: Zadní strana desky opakovače



Obrázek A.3: Přední strana desky opakovače



Obrázek A.4: Blokové schéma celého hlavního programu včetně jednotlivých bloků

Příloha B

Popisy signálů programu VHDL

RXD	Čtyřbitový datový výstup
TXD	Čtyřbitový datový vstup
RXC1, TXC1	Hodinový výstup (také jako clock1)
RXC2, TXC2	Hodinový vstup (také jako clock2)
RXDV	Signalizuje validní data - výstup (také jako ValidD)
RXER	signalizuje chybová data - výstup (také jako Err)
RST	Signál reset - aktivní v nule
SW	přepínače pro ovládání resetů a enable
sw0f	Filtrovaný signál z přepínače (také sw1f a sw2f)
ackw	Potvrzovací signál o přečtení dat a jejich uložení do RAM
$write_en$	Data jsou připravena k poslání na RAM
Q	Datový výstup
NumWords	Adresní výstup
$Port0_en$	Ovládání aktivního bufferu RAM
ackdel	Signál pro nulování počtu bajtů uložených v RAM
tmpQ	čtyřbitové registry pro data
DataSize	Výstup posílající počet bajtů v RAM
EN	Enable - zapnutí budičů pro příjem a posílání dat

Příloha C

Zkratky

LAN	Local Area Network
OSI	Open Systems Interconnection
CSMA/CD	Carrier Sense Mulitple Access/Collision Detection
UTP	Unshielded Twisted Pair
EMI	Electromagnetic Interference
EMC	Electromagnetic Compatibility
AFDX	Avionics Full-Duplex Switched Ethernet
IEEE	Institute of Electrical and Electronics Engineers
SOF	Start of Frame
CRC	Cyclic Redundancy Check
FCS	Frame Control Sequence
PCB	Printed Circuit Board
MDI	Medium Dependent Interface
ESD	Electrostatic Discharge
PCS	Physical Coding Sublayer
PMA	Physical Medium Attachment
IP	Internet protokol
UDP	User Datagram Protocol
ARINC	Aeronautical Radio Inc.
MAC	Podvrstva linkové vrstvy v rámci modelu OSI/ISO
PHY	Fyzická vrstva v rámci modelu OSI/ISO
MII	Media Independent Interface
VHDL	VHSIC Hardware Description Language
BLM	Feritové perličky (BLM18AG601SN1)

Příloha D

Literatura

[Buc16] D. Buckovsky, *Využití sítí ethernet v osobních automobilech*, bachelor thesis, CTU in Prague, may 2016.

[Che99] Adrian Chesney, *Introduction to ethernet*, The Extension **3** (1999).

[CMK14] Robert B. Boatright Jeffrey Quesnelle Charles M. Kozierok, Colt Correa, *Automotive ethernet: Definitive guide*, Intrepid Control Systems, October 2014.

[Con] Phoenix Contact, *Ethernet basics*.

[Cor14] Broadcom Corporation, *Broadr-reach physical layer trasciever specification for automotive applications*.

[Dor18] Peter L Dordal, *An introduction to computer networks*, December 2018.

[Fuc12] Christian M. Fuchs, *The evolution of avionics networks from arinc 429 to afdx*.

[HW12] Wensheng Niu Hongchun Wang, *Design and analysis of afdx network based high-speed avionics system of civil aircraft*.

[IXI14] IXIA, *Automotive ethernet: An overview*.

[JP06] Martin Kroupa Jiří Pinker, *Číslicové systémy a jazyk vhdl*, BEN - Technická literatura, 2006.

[Kal13] Martin Kalčík, *Implementace sběrnice can s ohledem na emc*, june 2013.

[Mal18] David Maliniak, *What's the difference between broadr-reach and 100base-t1?*

[MVLC15] Renato Mancuso, Andrew V. Louis, and Marco Caccamo, *Using traffic phase shifting to improve afdx link utilization*, 10 2015, pp. 256–265.

[NXP17] NXP Semiconductors, *100base-t1 phy for automotive ethernet*, 5 2017, Rev. 3.

[Ph.12] Dr. Ali Abaye Ph.D., *Broadr-reach technology: Enabling one pair ethernet*, 2012.

[Tec] IServe Technologies, *Ethernet repeaters rules*.

[Wik] Wikipedia, *Ethernet*, Wikipedia.

[Xer80] Digital Equipment Xerox, Intel, *The ethernet*.

[Yan07] Muhammet Yanik, *Avionics full duplex switched ethernet (afdx) data bus*.