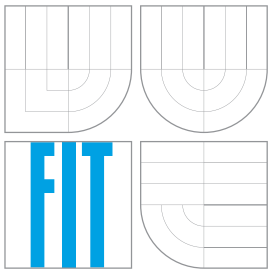
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

ANALYZÁTOR KOMUNIKACE NA SBĚRNICI USB

USB BUS COMMUNICATION ANALYZER

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. TOMÁŠ JANČO
AUTHOR

VEDOUCÍ PRÁCE — Ing. VÁCLAV ŠIMEK
SUPERVISOR

BRNO 2016

Zadání diplomové práce/10506/2015/xjanco04

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2015/2016

Zadání diplomové práce

Řešitel: **Jančo Tomáš, Bc.**
Obor: Počítačové a vestavěné systémy
Téma: **Analyzátor komunikace na sběrnici USB**
USB Bus Communication Analyzer
Kategorie: Počítačová architektura

Pokyny:

1. Podrobně nastudujte problematiku komunikace po sériové sběrnici USB. Zaměřte se na paketovou komunikaci a jednotlivé typy transakcí.
2. Zabývejte se oblastí programovatelných hradlových polí (FPGA). Seznamte s jazykem VHDL a vývojovými kity dostupnými na FIT.
3. Vytvořte detailní návrh koncepce analyzátoru komunikace na sběrnici USB, který bude využívat transceiver fyzické vrstvy této sběrnice a obvod FPGA.
4. S pomocí návrhového systému desek plošných spojů připravte realizaci hardware uvažovaného přípravku v podobě rozšiřujícího modulu pro některý z dostupných vývojových kitů.
5. V jazyce VHDL implementujte jednotky, které umožní zachytávat probíhající komunikaci a předávat tyto údaje dále do PC.
6. Naprogramujte obslužnou aplikaci s přehledným uživatelským rozhraním pro řízení přípravku a zobrazení naměřených dat.
7. Vytvořené řešení otestuje v praxi. Zhodnoťte dosažené výsledky a uvažujte možná vylepšení.

Literatura:

- dle pokynů vedoucího

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1-4 zadání.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Šimek Václav, Ing.**, UPSY FIT VUT
Datum zadání: 1. listopadu 2015
Datum odevzdání: 25. května 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
612 66 Brno, Božetěchova 2

L.S.

doc. Ing. Zdeněk Kotásek, CSc.
vedoucí ústavu

Abstrakt

Táto diplomová práca sa zaoberá návrhom analyzátoru komunikácie na zbernici USB. Prvá časť práce je venovaná zbernici USB a typom dátových prenosov na nej. Ďalej práca rozoberá existujúce riešenia analyzátorov zbernice USB. Následne je predstavená platforma FitKIT Minerva, na ktorej bude USB analyzátor implementovaný. Ďalšie časti práce sa zaoberajú návrhom architektúry analyzátoru a jeho častí - obvodu fyzickej vrstvy, komunikačného rozhrania a užívateľského rozhrania. Zvyšok práce sa venuje implementácií navrhnutého riešenia a vyhodnotením výsledkov. Výsledkom práce je USB analyzátor na platforme FitKIT Minerva s grafickým uživateľským rozhraním prístupným pomocou webového prehliadača.

Abstract

This diploma thesis focuses on designing a USB analyzer. First part of this work describes the USB bus and types of USB data transfers. Next part explores existing USB analyzers. After that, FitKIT Minerva is presented as target platform for USB analyzer implementation. Next part of this project is dedicated to design the analyzer architecture and its parts: physical layer circuit, communication interface and user interface. Rest of the work show how the analyzer was implemented and evaluates the results achieved. Final result of this thesis is a working USB analyzer on FitKIT Minerva platform with graphical user interface accesible via web browser.

Kľúčové slová

USB analyzátor, FitKIT Minerva, FPGA, Freescale Kinetis K60, USB3343, MQX

Keywords

USB analyzer, FitKIT Minerva, FPGA, Freescale Kinetis K60, USB3343, MQX

Citácia

JANČO, Tomáš. *Analyzátor komunikace na sběrnici USB*. Brno, 2016. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Šimek Václav.

Analyzátor komunikace na sběrnici USB

Prehlásenie

Prehlasujem, že túto prácu som vypracoval samostatne pod vedením Ing. Václava Šimka. Uviedol som všetky publikácie a literárne pramene, z ktorých som čerpal.

. . . . . .
Tomáš Jančo
22. mája 2016

Poďakovanie

Moja vďaka patrí môjmu vedúcemu Ing. Václavovi Šimkovi za cenné rady, ktoré mi poskytol pri písaní tejto práce a najmä za jeho ochotu a pomoc pri vyrábaní plošných spojov.

Obsah

Kapitola 1

Úvod

Zbernica USB sa stala všadeprítomným komunikačným rozhraním medzi osobnými počítačmi a periférnymi zariadeniami. Pri vývoji hardwaru obsahujúceho USB, alebo vývoji obslužného softwaru je pre programátora dôležité mať vhodné nástroje. Jedným z nástrojov uľahčujúcim porozumenie komunikácie na USB a hľadanie chýb je USB analyzátor.

V tejto diplomovej práci predkladám návrh analyzátora USB, ktorý môže byť použitý napríklad pri výuke v oblasti komunikačných rozhraní na FIT VUT.

V nasledujúcej kapitole sa práca zaoberá samotnou zbernicou USB, jej použitím, vlastnosťami, topológiou, spôsobom organizácie výmeny dát do paketov a fyzickou vrstvou signalizácie.

V kapitole 3 sú popísané existujúce riešenia, vrátane projektu OpenViszla, ktorý bol zvolený ako základ môjho riešenia. V kapitole 4 popisujem zvolenú cieľovú platformu, výukový kit Minerva, ktorá vznikla na FIT VUT ako nástupca vývojových platforiem FitKit 1 a FitKit 2. Zvolená platforma kombinuje moderné prostriedky z oblasti vstavaných zariadení: hradlové pole FPGA Xilinx Spartan 6 a mikrokontrolér Freescale Kinetis.

O spôsobe prispôsobenia projektu OpenViszla pre platformu výukového kitu Minerva hovorí kapitola 5, v ktorej je navrhnutý obvod fyzickej vrstvy na pripojenie USB zbernice k hradlovému poľu FitKit-u. Na ňu nadväzujú kapitoly 5.3 a 5.4, kde sa zaoberám výberom komunikačného rozhrania a návrhom užívateľského rozhrania analyzátoru. Na rozdiel od pôvodného projektu, v ktorom sa analyzátor pripája k počítaču lokálne prostredníctvom portu USB a na obsluhu sa používa konzolová aplikácia, v mojej práci navrhujem pripojenie analyzátoru k počítaču pomocou počítačovej siete Ethernet a obsluhu prostredníctvom grafického užívateľského rozhrania.

Kapitola 6 sa venuje implementácií navrhnutého riešenia. Najskôr je predstavený postup osadenia a oživenia obvodu fyzickej vrstvy a potom sú vysvetlené implementačné detaily jednotlivých modulov, ktoré dovedna tvoria USB analyzátor. Podrobne sú popísané komunikačné rozhrania medzi jednotlivými modulmi, aj medzi FPGA a MCU a medzi MCU a PC. Podkapitola 6.3 sa venuje použitiu nástroja Migen na generovanie HDL popisu z popisu s vyššou úrovňou abstrakcie založenom na jazyku Python.

Časť riešenia, ktorá je implementovaná v MCU využíva pre svoj beh operačný systém reálneho času Freescale MQX. Systému samotnému je venovaná časť 6.4.1 a následne je opísaná činnosť programu v MCU. Predposledná podkapitola 5.4 sa venuje implementácií užívateľského rozhrania zobrazovaného vo webovom prehliadači počítača a komunikácií tohto rozhrania s MCU pomocou protokolu HTTP.

Posledná kapitola 7 sa venuje dosiahnutým výsledkom práce a porovnáva ich s pôvodným projektom OpenViszla, ďalšími nástrojmi na analýzu USB ako aj s očakávaniami au-

tora. Navrhnuté sú postupy, ktoré by mohli potlačiť nedostatky alebo optimalizovať činnosť prístroja.

1.1 Ciele práce

V súlade so zadaním práce a s motiváciou uvedenou v úvode tejto správy stanovujem ciele tejto práce:

- Priblížiť čitateľovi spôsob fungovania zbernice USB.
- Preskúmať existujúce riešenia USB analyzátorov, ich architektúru a vlastnosti.
- Preskúmať cieľovú platformu Minerva so zameraním na použiteľnosť jej komponentov pre túto prácu.
- Na základe získaných znalostí navrhnúť architektúru analyzátora USB.
- Špecifikovať požiadavky na jednotlivé časti navrhnutej architektúry.
- Navrhnúť a zostrojiť jednotlivé časti tak, aby spĺňali stanovené požiadavky.
- Zostaviť a oživiť prípravok USB analyzátora ako celok.
- Otestovať jeho funkciu a porovnať výsledky s inými riešeniami.

Táto práca nadväzuje na rovnomenný semestrálny projekt, v ktorom boli riešené prvé 4 body z cieľov uvedených vyššie. Kapitoly 2 až 5 boli prevzaté z tohoto projektu. Výsledkom semestrálneho projektu bol návrh architektúry analyzátora a návrh obvodu fyzickej vrstvy, ktoré boli v diplomovej práci využité pri implementácií analyzátora v kapitole 6.

Kapitola 2

USB - Univerzálna sériová zbernica

Rozhranie USB je dostatočne všestranné na použitie v širokej škále periférií. Medzi bežné periférie vybavené USB patria myši, klávesnice, diskové jednotky, tlačiarne a audio/video zariadenia [2]. Z hľadiska užívateľa je USB jednoduché na použitie - nevyžaduje konfiguráciu a nastavenie parametrov ako napríklad sériový port. Na pripojenie periférií sa používajú štandardné prepojovacie vodiče. Porty, do ktorých sa USB zariadenia pripájajú sú spravidla rovnocenné. Periférie môžu byť pripájané a odpájané za behu a menšie periférie nepotrebujú osobitné napájanie. Medzi jeho ďalšie vlastnosti patrí vysoká prenosová rýchlosť a spoľahlivosť. O spôsoboch dosiahnutia spoľahlivosti pojednáva časť 2.1.1 o kontrole chýb.

2.1 Topológia

Topológia zbernice USB je viacstupňová hviezda. V hostiteľskom zariadení (*host*) sa nachádza koreňový rozbočovač (*root hub*), ktorý tvorí prvý stupeň topológie a je pripojený ku hostiteľskému radiču USB (*host controller*). Každý rozbočovač vrátane koreňového rozbočovača tvorí stred hviezdy. K rozbočovačom môžu byť pripojené periférne zariadenia, alebo ďalší rozbočovač, až do siedmej úrovne. Maximálny celkový počet zariadení pripojených k jednému radiču je 127, vrátane rozbočovačov.

Podľa funkcie jednotlivých prvkov rozlišujeme viacero pojmov opísaných v nasledujúcich odstavcoch prevzatých z príručky k rozhraniu USB [2].

2.1.1 Hostiteľ a hostiteľský radič (*host, host controller*)

Radič zbernice USB nachádzajúci sa v hostiteľskom zariadení spravuje celú komunikáciu. Medzi funkcie hostiteľského radiča patria:

Detekcia a enumerácia zariadení

Hostiteľ sleduje novo pripojené resp. odpojené zariadenia. V procese enumerácie je každému zariadeniu pridelená adresa a hostiteľ získava ďalšie informácie o zariadení.

Správa dátového toku

Hostiteľ riadi dátový tok na zbernici. Keďže viacero zariadení môže naraz požadovať prenos po zbernici, hostiteľ rozdeľuje dostupný čas prenosu a prideľuje ho jednotlivým zariadeniam. Počas enumerácie zariadenia deklarujú požiadavky na prenosové pásmo (pre prenosy

s garantovaným časovaním). Ak nie je možné splniť tieto požiadavky, komunikácia je zamietnutá a zariadenie musí počkať na uvoľnenie, alebo požiadať o menšiu šírku pásma. Prenosy bez garantovaného časovania nemajú vyhradené prenosové pásmo a musia čakať na uvoľnenie zbernice.

Kontrola chýb

K prenášaným dátam medzi hostiteľom a zariadením sú pridané kontrolné bity. Zariadenie, ktoré príjme dátový rámec, výpočtom overí kontrolné bity a v prípade úspešného overenia potvrdí príjem. Ak pri prenose dôjde k chybe, zariadenie túto chybu deteguje a príjem nepotvrdí. Tak sa hostiteľ dozvie o chybe prenosu. Podobne hostiteľ overuje kontrolné bity v dátach prijatých od zariadení.

Napájanie

Podľa špecifikácie USB hostiteľ poskytuje pripojeným zariadeniam napájanie pomocou napájacích vodičov +5V a GND (zem). Minimálny prúd dostupný pre pripojené zariadenie je 100 mA. Ak zariadenie požaduje vyšší prúd (až 500 mA), môže ho začať odoberať až po konfigurácií zo strany hostiteľa [21]. V prípade, že hostiteľ nemôže taký prúd poskytnúť, zariadenie nenakonfiguruje. Pre zariadenia pracujúce na zbernici USB 3.0 sú tieto hodnoty vyššie: 150 mA pre nenakonfigurované zariadenia a až 900 mA pre nakonfigurované [23].

Výmena dát

Hostiteľ zabezpečuje výmenu dát s periférami. Výmena dát môže prebiehať v určených časových intervaloch alebo na žiadosť nadradenej aplikácie. Komunikáciu zahajuje vždy hostiteľ vyslaním špeciálneho *token* paketu. Ten sa šíri od hostiteľa cez rozbočovače ku všetkým zariadeniam. Odpovedá iba zariadenie identifikované adresou v *token*-e.

2.1.2 Rozbočovač (*hub*)

Rozbočovač sa pripája k zbernici USB pomocou tzv. *upstream* portu a poskytuje niekoľko *downstream* portov na pripojenie ďalších zariadení. Rozbočovače poskytujú spätnú kompatibilitu medzi rôznymi verziami štandardu USB tak, že prispôsobujú rýchlosť komunikácie.

2.1.3 USB zariadenie (*USB device*)

Zariadenia sú podľa štandardu USB [21] rozdelené do tried, ako napríklad rozhranie medzi človekom a strojom (*human interface*), tlačiareň alebo hromadné dátové úložisko (*mass storage*). Špeciálnu triedu USB zariadení tvoria rozbočovače opísané vyššie.

2.2 Typy prenosov

USB bolo navrhnuté tak, aby umožňovalo pripojiť rôznorodé zariadenia, ktoré majú rôzne požiadavky na prenosovú rýchlosť, časovanie a opravu chýb. Preto USB definuje 4 rôzne typy prenosov, každý s inými vlastnosťami.

2.2.1 Riadiace prenosy (*control transfers*)

Riadiace prenosy slúžia na konfiguráciu novo pripojených zariadení a prípadne na ďalšie účely určené výrobcom zariadenia. Prenosy sú bezstratové (s opakovaním prenosu po chybe) a obojsmerné, forma prenosu je výmena správ. Všetky zariadenia USB musia podporovať tento typ prenosu.

2.2.2 Hromadné prenosy (*bulk transfers*)

Prenosy typu *bulk* slúžia na výmenu väčšieho objemu dát vysokou rýchlosťou. Oproti riadiacim prenosom nie sú založené na výmene správ, ale na jednosmerných prúdoch dát.

2.2.3 Prerušenia (*interrupt transfers*)

Prenos prerušení slúži na oznámenie o udalosti. Podľa štandardu USB 1.0 [20] môžu byť prerušenia signalizované len od zariadenia k hostiteľovi, vyššie verzie USB podporujú aj prenos prerušení opačným smerom. Oproti predchádzajúcim typom prenosov majú garantovanú latenciu.

2.2.4 Isochronné prenosy (*isochronous transfers*)

Posledný typ prenosov slúži na prenosy dát v reálnom čase, teda tam, kde je nutná garantovaná šírka pásma. V prípade chyby prenosu sa prenos neopakuje, dáta sa zahadzujú. Tento typ prenosu sa používa najmä na prenos multimédií v reálnom čase.

2.3 Fyzická vrstva

Špecifikácia USB definuje požiadavky na signalizáciu, napäťové úrovne, prepojovacie vodiče a konektory. Tieto požiadavky sú zhrnuté v nasledovných odstavcoch.

2.3.1 Konektory

Špecifikácia rozlišuje dva typy konektorov: A a B. Konektor typu A pasuje do zásuvky typu A, zásuvka typu A vždy smeruje k hostiteľovi. Naopak konektor typu B pasuje do zásuvky typu B a zásuvka B sa nachádza vždy na USB zariadení alebo rozbočovači. Týmto je zabezpečené správne rozlišovanie medzi hostiteľským a hosťovským zariadením. Pre mobilné zariadenia, kde je veľký rozmer zásuvky B prekážkou, bola špecifikácia doplnená o menšie konektory mini USB [19] a micro USB [22]. Pri použití USB 3.0 sú do konektorov doplnené ďalšie kontakty pre ďalšie dátové vodiče resp. konektor micro USB je doplnený o ďalší konektor.

2.3.2 Vodiče

Prepojovací kábel USB 2.0 a starších má 4 vodiče, z toho dva tvoria krútený pár na prenos dát a dva vodiče sú napájacie. Novší štandard USB 3.0 [23] používa na komunikáciu viac vodičov, ktoré tvoria diferenciálne páry troch sériových liniek. Pre obmedzenia vyplývajúce z útlmu a prenosového oneskorenia je obmedzená maximálna dĺžka vodičov v závislosti na maximálnej prenosovej rýchlosti.

2.3.3 Signalizácia

Dátová komunikácia prebieha sériovo v oboch smeroch po jednom (resp. troch pre USB 3.0) diferenciálnom páre. Pre potreby opisu fyzickej vrstvy USB sa zavádzajú pojmy diferenciálna 1 a diferenciálna 0.

Pre rýchlosti *low speed* a *full speed* sa považuje za diferenciálnu 1 keď na vodiči D+ je napätie aspoň 2V oproti zemi GND a rozdiel medzi vodičmi D+ a D- je viac ako 0,2V. Pre diferenciálnu 0 sa úloha vodičov D+ a D- vymení: $U_{D-} > 2V$ a $U_{D-} - U_{D+} > 0,2V$.

Pre rýchlosť *high speed* musí byť pre dif. 1 výstup D+ na napäťovej úrovni najmenej 0,36V a D- najviac 0,01 V. Pre dif. 0 sa napätia D+ a D- opäť vymenia.

Okrem týchto napäťových úrovní nastávajú na zbernici ďalšie stavy: *Single-Ended Zero (SE0)* nastáva keď D+ aj D- sú na nízkej úrovni. Tento stav sa používa na signalizáciu resetu, konca paketu a odpojenia zariadenia. Podobne *Single-Ended One (SE1)* nastáva keď D+ a D- sú na vysokej napäťovej úrovni a označuje neplatný stav zbernice, pri bežnej prevádzke sa tento stav nevyskytuje.

2.3.4 Kódovanie

Dáta na zbernici sú kódované pomocou NRZI s bit stuffingom. Logická nula je kódovaná ako prechod medzi dvoma diferenciálnymi úrovňami, logická 1 ako zachovanie rovnakej úrovne. Dáta sú prenášané systémom LSB-first, teda najmenej významný bit sa prenáša ako prvý. Aby bolo možné úspešne zrekonštruovať synchronizačný signál aj v prenose obsahujúcom postupnosť logických 1 (t. j. dlhé obdobie nemenných napäťových úrovní), po šiestich prenesených log. 1 sa vkladá jedna log. 0 za poslednú (šiestu) log.1. Vďaka tomu sa za každých 7 odvysielaných bitov vyskytne aspoň jedna zmena napäťových úrovní.

2.3.5 Pakety

Všetky dátové prenosy USB sú organizované do paketov. Skupiny paketov tvoria transakcie resp. mikro-transakcie pri High-speed USB 2.0.

Každý paket začína hodnotou PID (*packet identifier*), ktorá určuje typ paketu. Pre možnosť detekcie chýb je táto hodnota na začiatku paketu zopakovaná v komplementárnej forme.

Token pakety

Token pakety odosiela hostiteľ a po hodnote PID nesú 11-bitovú adresu a CRC kontrolný súčet. Podľa PID rozlišujeme tieto typy:

- OUT - nesie adresu zariadenia, ktoré má prijať nasledujúci dátový paket
- IN - nesie adresu zariadenia, ktoré má odoslať dátový paket hostiteľovi
- SOF - slúži na synchronizáciu - označuje začiatok rámca
- SETUP - nesie adresu zariadenia, ktoré má prijať nasledujúci inicializačný dátový paket
- PING (USB 2.0) - slúži na zistenie stavu zariadenia

- SPLIT - slúži na vykonanie tzv. SPLIT transakcie, kde vysokorýchlostný hostiteľ komunikuje s pomalším zariadením. Rozbočovač medzi hostiteľom a zariadením obslúži komunikáciu so zariadením na nižšej rýchlosti a neblokuje pri tom vysokorýchlostnú časť zbernice.

Handshake pakety

Tento typ paketov slúži na potvrdzovanie prijatia iných typov paketov. Spravidla sú odosielané po dátovom pakete alebo po žiadosti o dáta token paketom IN:

- ACK - potvrdzuje prijatie dátového paketu
- NAK - oznamuje neprijatie dátového paketu a žiadosť o opakovanie prenosu
- STALL - oznamuje chybu prenosu, ktorú nie je možné opraviť opakovaním prenosu (napr. neinicializované zariadenie)
- NYET - označuje nepripravené zariadenie alebo nedokončenú SPLIT transakciu

Data pakety

Dátové pakety nesú až 1024 bytov dát a 16 bitov kontrolného súčtu CRC. Pri dátovom prenose sa striedajú pakety DATA0 a DATA1, čím sa zabezpečuje detekcia duplicitného prenosu - ak zariadenie prijme po sebe dva dátové pakety rovnakého typu, druhý z nich ignoruje ako duplikát. V špecifikácií USB 2.0 existujú ešte dátové pakety DATA2 a MDATA ktoré slúžia na vysokorýchlostné isochrónne prenosy.

Kapitola 3

Existujúce riešenia

V rámci tejto kapitoly som sa zaoberal existujúcimi riešeniami USB analyzátorov, ich rôznymi typmi, vlastnosťami a spôsobmi činnosti.

3.1 Softwarové analyzátory

Softwarový analyzátor beží na hostiteľskom systéme a sleduje komunikáciu s perifémymi zariadeniami na USB. Hlavnou výhodou je, že nie je potrebný žiaden dodatočný hardware. Z princípu ich činnosti však nie je možné sledovať komunikáciu medzi inými zariadeniami ako PC, na ktorom beží tento software a k nemu pripojenou perifériou. Tieto analyzátory sú dostupné ako aplikácie alebo programové knižnice, prípadne iné, užšie zamerané nástroje, sledujúce napríklad len enumeráciu pripojených zariadení.

3.1.1 USBlyzer

Ako príklad softwarového analyzátoru uvádzam nástroj USBlyzer [24]. Tento nástroj je softwarový USB analyzátor pre operačné systémy MS Windows. Podporuje USB až do verzie 3.0 na 32 aj 64 bitových systémoch. Okrem zachytávania komunikácie na USB zbernici počítača dokáže zobrazovať systémové informácie o pripojených zariadeniach, ich hierarchii a použitých ovládačoch. Zachytávanie USB komunikácie funguje v reálnom čase vrátane zobrazenia systémových volaní. Podporované je aj zachytávanie komunikácie s viacerými zariadeniami naraz. Zachytená komunikácia je dekódovaná a zobrazované sú zrozumiteľné informácie o význame jednotlivých prenosov. Údaje je možné filtrovať a vyhľadávať v nich, tiež je možné komunikáciu uložiť do proprietárneho formátu a opätovne ju načítať, prípadne exportovať do textového formátu či CSV alebo HTML.

3.2 Hardwarové analyzátory

Hardwarové usb analyzátory sa pripájajú medzi ľubovoľné hostiteľské zariadenie a USB periférii. Fungujú buď samostatne (disponujú ovládacími prvkami a displejom) alebo zozbierané údaje odosielajú do pripojeného počítača alebo logického analyzátora.

3.2.1 USB-2XT

Jedným z komerčných USB analyzátorov je zariadenie USB-2XT firmy Crescent Heart Software. Jedná sa o hardwarový modul pripojiteľný k logickému analyzátoru Tektronix. Podľa

produktovej špecifikácie [4] zariadenie analyzuje zbernicu USB 2.0/1.1/1.0 na rýchlostiach High Speed, Full Speed a Low speed. Výstupom je 34-kanálový signál, ktorý bude ďalej spracovaný v log. analyzátore. Zariadenie umožňuje nastavenie širokej škály filtrov a spúšťí. Zachytené nie sú len obsahy USB paketov, ale aj stavy zbernice na úrovni fyzickej vrstvy. Zariadenie sa ovláda prostredníctvom dodaného softwaru, ktorý komunikuje priamo s usb analyzátorom (pre riadenie) aj s logickým analyzátorom (pre čítanie nazbieraných dát).

Nevýhodou takéhoto zariadenia je jeho závislosť na samostatnom logickom analyzátore. Na druhej strane je to zaujímavý spôsob zjednodušenia, keďže analyzátor nemusí obsahovať obvody a pamäť na zber dát, ani komunikačné obvody na pripojenie k PC.

3.2.2 OpenViszla USB analyzátor

Projekt OpenViszla je USB analyzátor založený na FPGA s otvoreným zdrojovým kódom. Ako obvod fyzickej vrstvy používa transceiver USB3343 pripojený rozhraním ULPI k hradlovému poľu Xilinx Spartan 6LX FPGA. Na pripojenie k počítaču používa rozhranie USB pomocou obvodu FTDI FT2232H a upraveného firmwaru.

Vďaka otvorenej licencii a použitiu obvodu FPGA je projekt OpenViszla vhodný na portovanie na niektorú výukovú platformu dostupnú na FIT VUT, preto je jeho popisu venovaný najväčší priestor.

Architektúra

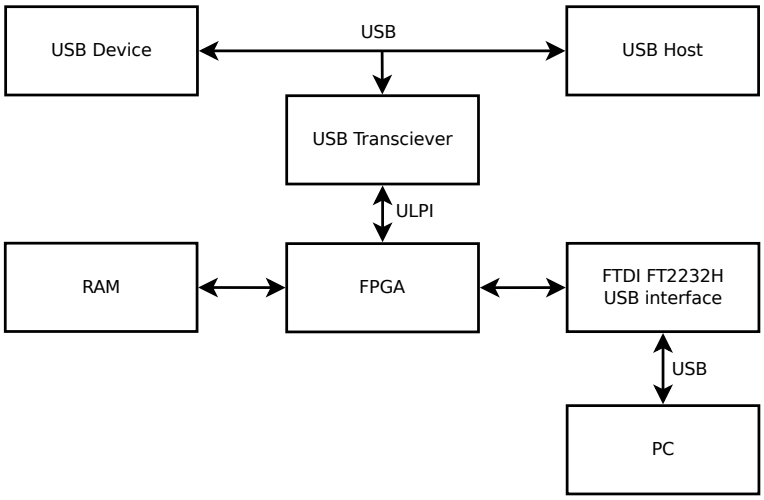
Architektúra projektu OpenViszla je zachytená na obr. 3.1. V srdci analyzátoru je obvod FPGA, v ktorom sú implementované komunikačné rozhrania ULPI na komunikáciu s USB transceiverom, radič pamäte SDRAM, paketový filter, radič synchrónnej zbernice FTDI pre komunikáciu s PC a obslužným softwarom a stavový automat orchestrujúci činnosť ostatných jednotiek. Činnosť analyzátora je priamočiara, po inicializácií sú pakety zachytené transceiverom preposielané na komunikačnú zbernicu smerom k PC. Okrem toho je spracovávaných niekoľko príkazov prijatých z komunikačnej zbernice umožňujúcich nastavenie parametrov filtra a prípadne je použité bufferovanie do pamäte RAM. (V poslednej vývojovej verzií bolo bufferovanie do RAM nefunkčné).

Komunikačné rozhranie

OpenViszla používa na komunikáciu cez USB obojsmerné FIFO rozhranie sprostredkované obvodom FTDI FT2232H na analyzátore a knižnicou libusb na strane počítača. Používajú sa *bulk transfers* vhodné na rýchly prenos veľkých objemov dát. Obvod FPGA je ku komunikačnému obvodu pripojený osembitovou zbernicou a obvod FTDI vykonáva (de)serializáciu komunikácie.

Obslužný software

Software bežiaci na PC zabezpečuje okrem prijímania analyzovaných dát aj nahrávanie konfigurácie do FPGA. Obsluha komunikácie je napísaná v jazyku C, konzolové užívateľské rozhranie a parser USB paketov sú napísané v jazyku Python.



Obr. 3.1: Bloková schéma analyzátoru OpenViszla

3.3 Vzťah existujúcich riešení k tejto práci

V tejto práci budú adresované nevýhody predstavených analyzátorov. V prípade SW analyzátorov je to spomínané obmedzenie na analýzu komunikácie výhradne medzi hostiteľským PC a jeho USB perifériou. Ďalej bude adresovaná nutnosť inštalácie špeciálneho obslužného softwaru pre prácu s analyzátorom. V tejto práci bude predstavený prístup, kde obslužný software beží priamo v MCU, ktoré je súčasťou analyzátora a užívateľ bude môcť analyzátor používať prostredníctvom webového rozhrania, ktoré nevyžaduje inštaláciu dodatočného SW. Nevýhodou (najmä komerčných) HW analyzátorov je ich vysoká cena. Tento problém bude riešený použitím cenovo dostupných komponentov, tak ako je to v prípade projektu OpenViszla.

Pre vytvorenie USB analyzátora na cieľovej platforme je možné použiť existujúci návrh a implementáciu niektorých jednotiek projektu OpenViszla. Ako vhodné sa ukázalo použitie transceiveru USB3343, preto ho využijem aj v tejto práci. Prenositeľné sú aj jednotky paketového filtra a radiča ULPI. Pre odlišné komunikačné rozhrania cieľovej platformy nebude použitá jednotka radiča FTDI, ale bude nahradená radičom príslušného typu. Zo schémy zapojenia a návrhu DPS je možné sa inšpirovať pri vytváraní obvodu fyzickej vrstvy: v mojej práci preberám časť zapojenia obsahujúcu obvod USB3343.

Kapitola 4

Cieľová platforma - kit Minerva

Vývojová doska Minerva je výučbová platforma vyvinutá na Fakulte informačných technológií VUT v Brne. Je to nástupca FITKit 2.0 oproti ktorému obsahuje modernejšie prvky. Medzi inými je to hradlové pole Xilinx Spartan 6, mikrokontrolér architektúry ARM Freescale Kinetis. Doska disponuje rozhraniami USB, Ethernet, HDMI, audio vstupom a výstupom, programovacími rozhraniami a sadou obecných vstupno-výstupných pinov. Na doske je integrovaný programátor, pamäť typu FLASH, v ktorej je možné uchovať konfiguráciu hradlového poľa a externá pamäť SDRAM. [27]

4.1 Freescale ARM KINETIS Cortex M4

Mikrokontrolér prítomný na doske Minerva (typové označenie MK60DN512ZVMD10) patrí do rodiny K60 mikrokontrolérov Kinetis firmy Freescale. Jeho jadrom je ARM Cortex-M4 taktovateľný až na frekvenciu 100 MHz. Priamo na čipe sa nachádzajú periférie ako sú časovače, analógovo-digitálne a digitálne-analógové prevodníky, DMA radiče, jednotka ochrany pamäte a jednotky pre hardwarovú podporu kryptografických funkcií (generátor náhodných čísel, akcelerátor algoritmov DES, 3DES, AES, MD5 a SHA) i hardwarový modul CRC. Mikrokontrolér je vhodný aj pre riešenia s nízkou spotrebou, jednak rozsahom napájacieho napätia od 1,71 V do 3,6 V, ale hlavne podporou viacerých nízkopríkonových režimov. [7]

4.2 FPGA Xilinx Spartan 6

Použité hradlové pole Xilinx Spartan 6 typu XC6SLX9 obsahuje 1430 slices obsahujúcich 4 šesťvstupové obvody LUT a osem klopných obvodov (*flip-flop*). Okrem toho toto hradlové pole obsahuje 16 DSP slices, slúžiacich na akceleráciu spracovania signálov, ktoré obsahujú násobičky, sčítačky a akumulačné registre. [3]

K hradlovému poľu je cez rozhranie SPI pripojená pamäť FLASH, ktorá slúži na uloženie konfigurácie FPGA. S mikrokontrolérom je FPGA spojené pomocou 32 bitovej FlexBus. K FPGA sú tiež pripojené niektoré periférne jednotky: pamäť SDRAM, a radič grafického rozhrania DVI. [27]

4.3 Periférne jednotky

Možnosti mikrokontroléru a hradlového poľa na doske Minerva sú rozšírené osadenými perifémymi jednotkami.[18]

4.3.1 Pamäť SDRAM

Pamäť typu DDR2 SDRAM výrobcu ISSI má veľkosť 512 Mbit a je pripojená priamo k FPGA. Vytvorením vhodného radiča v FPGA môže slúžiť ako externá pamäť pre logické obvody v hradlovom poli a nahradiť tak distribuovanú BRAM, ktorá zaberá prostriedky FPGA. Maximálna podporovaná frekvencia je 333 MHz, ale keďže sa jedná o pamäť typu DDR, dátový prenos prebieha na dvojnásobnej rýchlosti (dáta sú prenášané s nábežnou aj klesajúcou hranou hodinového signálu).

V tomto projekte môže byť pamäť RAM použitá ako vyrovnávacia pamäť pri zachytávaní USB komunikácie pred jej prenosom do počítača a zobrazením v obslužnom softwari.

4.3.2 Ethernet transceiver SMSC LAN8720

Transceiver fyzickej vrstvy Ethernet-u slúži na pripojenie Minerva kitu k lokálnej sieti LAN. Obvod je pripojený k mikrokontroléru Kinetis pomocou rozhrania RMII. V mikrokontroléri sú potom implementované vrstvy ISO/OSI od linkovej (MAC) vyššie. Podporované sú rýchlosti 10Mbps a 100Mbps.

4.3.3 Radič DVI

Radič DVI `TFP410` firmy Texas Instuments slúži na pripojenie zobrazovacieho zariadenia (displeja, projektoru). Na strane kitu je pripojený k FPGA pomocou 24 bitovej paralelnej zbernice (8 bitov na farebný kanál), riadiacich a hodinových signálov (vertikálna a horizontálna synchronizácia) a rozhrania I2C na konfiguráciu radiča. Rozhranie DVI je vyvedené na zodpovedajúci konektor.

4.3.4 Audio kodek

Na doske je prítomný aj audio kodek `SGTL5000` pripojený k FPGA rozhraním I2S. Jeho linkový a slúchadlový výstup ako aj mikrofónový vstup sú pripojené na audio konektory (jack) na kite. Medzi jeho vlastnosti patrí nízky príkon a vysoký odstup signálu od šumu.

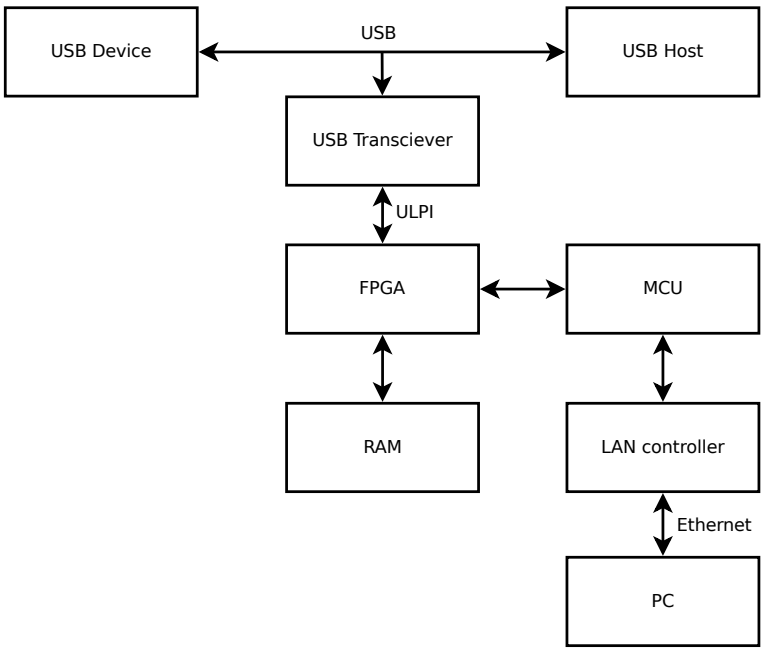
4.3.5 USB prevodník Vinculum-II

`FTDI VNC2-48L1B` je prevodník s procesorovým jadrom s 16-bitovou Harvardskou architektúrou. Rôznou konfiguráciou umožňuje prevádzať USB rozhranie na sériovú linku (UART), rozhranie SPI a iné. V kite Minerva je využitý na emuláciu sériového portu pre komunikáciu s MCU pomocou terminálu. Pôvodne bol určený na nahrávanie konfigurácie do FPGA, na tieto účely je však jeho prenosová rýchlosť príliš nízka [3].

Kapitola 5

Návrh architektúry analyzátora

Jadrom celého analyzátora bude hradlové pole FPGA. V ňom bude implementovaná väčšina logiky spracovania dátovej komunikácie. Prenos do počítača sa uskutoční pomocou sieťového rozhrania Ethernet. Komunikáciu s počítačom bude zabezpečovať MCU. Prenos dát medzi FPGA a MCU bude prebiehať po 32-bitovej zbernici FlexBus. Pripojenie k analyzovanej zbernici USB sa uskutoční pomocou externého obvodu fyzickej vrstvy s USB transceiverom. Tento obvod sa bude pripájať k FPGA pomocou rozširujúceho hrebeňového konektoru, komunikácia bude prebiehať podľa štandardu ULPI. Nazbierané údaje sa budú dočasne ukladať do SDRAM prítomnej na doske. Túto architektúru zachytáva schéma na obr. 5.1.



Obr. 5.1: Bloková schéma architektúry analyzátora

5.1 Obvody fyzickej vrstvy

Obvod fyzickej vrstvy obsahuje USB Transceiver USB3343, ktorý predstavuje rozhranie medzi sériovou zbernicou USB a paralelnou zbernicou ULPI. Transceiver je pripojený na dátové linky medzi komunikujúce USB zariadenia. Podporované sú rýchlosti až do USB 2.0 High-speed, t. j. 480 Mb/s.

5.1.1 Návrh zapojenia a DPS

Dekompozícia problému a požiadavky

Návrh zapojenia obvodu fyzickej vrstvy som rozdelil na tieto podproblémy:

- Výber vhodného obvodu fyzickej vrstvy
- Návrh prepojenia zbernice USB s obvodom fyzickej vrstvy
- Návrh prepojenia obvodu fyzickej vrstvy s kitom Minerva
- Návrh napájania obvodov
- Zakreslenie schémy zapojenia
- Návrh rozloženia a ciest dosky plošných spojov

Obvod fyzickej vrstvy bol zvolený na základe poznatkov z analýzy projektu OpenViszla. Použitie alternatívnych obvodov USB3320 alebo USB3300 by neprinieslo žiadne vylepšenie oproti spomínanému projektu - obvody majú porovnateľné vlastnosti. Hoci poskytujú možnosť práce s 16-bitovou zbernicou namiesto 8-bitovej a tým zníženie hodinového taktu, znamenalo by to zvýšenie počtu prepojovacích vodičov medzi obvodom a doskou Minerva a teda zložitejší návrh DPS.

Podľa špecifikácie USB, ako aj podľa existujúcich riešení (kap. 3), som zvolil spojenie so zbernicou USB pomocou konektoru typu B na strane hostiteľského zariadenia a konektoru typu A na strane analyzovaného zariadenia. Takto je možné k analyzátoru pripojiť vhodné USB káble na spojenie so zariadeniami rôznych typov.

Pri návrhu prepojenia obvodu s doskou Minerva som vychádzal z požiadavky na výstupné piny: Obvod musí byť pripojený k vstupno/výstupným pinom FPGA v napäťovej úrovni 3,3 V CMOS, hodinový signál musí byť vedený na pin FPGA schopný prepojenia s hodinovým subsystémom a dostupné musí byť aj napájanie a spoločná zem. Preto boli zvolené piny 5 až 16 konektoru P1 ako dátové, piny 1 až 4 ako napájacie a pin 35 ako hodinový signál. Priradenie funkcií jednotlivým pinom je uvedené v tabuľke 5.1.

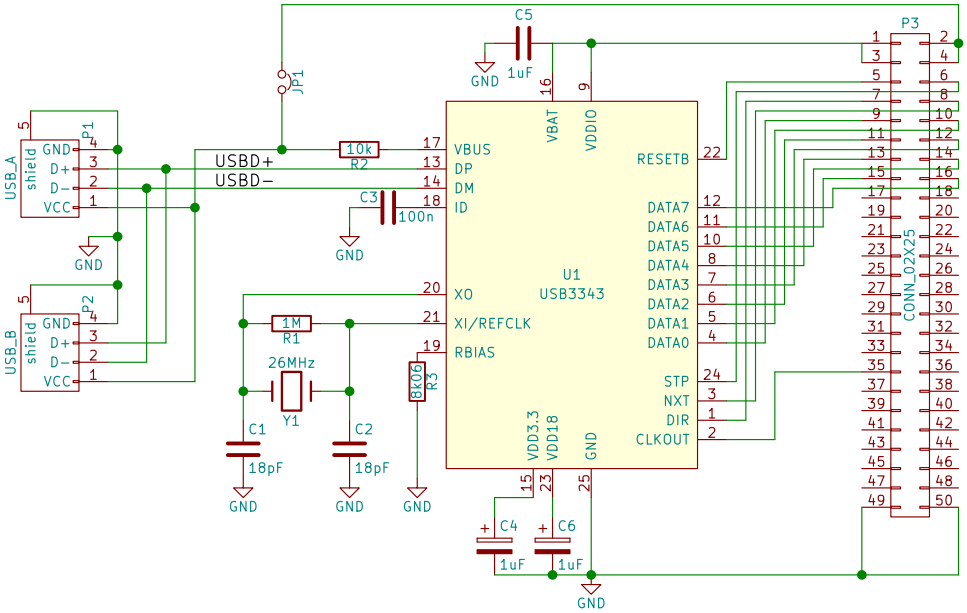
Podľa zložitosti pripravovanej DPS a dostupnosti puzdier uvažovaného transceiveru bola zvolená technológia povrchovej montáže súčiastok SMD na obojstrannej DPS.

Schéma zapojenia na obrázku 5.2 bola navrhnutá podľa požiadaviek uvedených vyššie, aplikačných príkladov z katalógového listu použitého transceiveru USB3343 [17] a existujúceho riešenia OpenViszla z kapitoly 3. Keďže FPGA a transceiver komunikujú na tej istej napäťovej úrovni signálov, sú prepojené priamo. Pridané boli nevyhnutné blokovacie kondenzátory na napájacie vývody obvodu USB3343 v hodnotách odporúčaných výrobcom. Tiež bol pripojený kryštál ako zdroj hodinového signálu o frekvencií 26 MHz. Prepojka JP1 umožňuje priviesť napájanie z hostiteľského zariadenia USB na hlavnú dosku kitu Minerva, takže analyzátor nebude vyžadovať ďalšie vodiče na napájanie.

Pin	Signál	Pin	Signál
1	+3.3V	2	+5V
3	+3.3V	4	+5V
5	RESET	6	STP
7	DIR	8	NXT
9	DATA0	10	DATA1
11	DATA2	12	DATA3
13	DATA4	14	DATA5
15	DATA6	16	DATA7
35	CLK		
49	GND	50	GND

Tabuľka 5.1: Priradenie pinov konektoru P1 dosky Minerva kit k signálom obvodu fyzickej vrstvy.

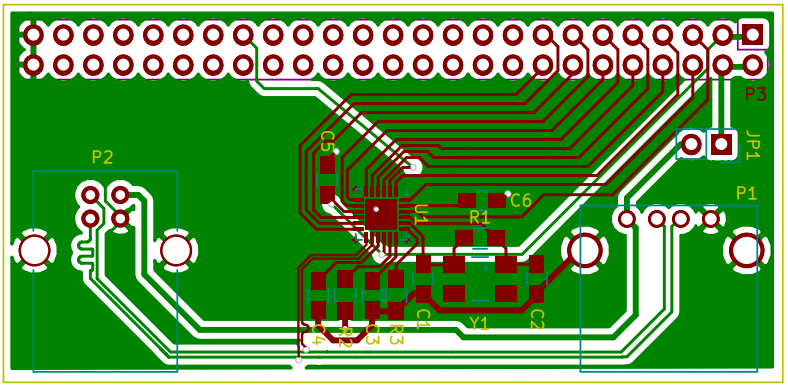
Špecifikácia USB vyžaduje zakončenie dátových vodičov terminačnými rezistormi, aby nedochádzalo k odrazom signálu [21]. S ohľadom na použitie USB analyzátora, tieto rezistory budú vždy prítomné v hostiteľskom USB radiči a pripojenej periférií - USB analyzátor sa pripája medzi tieto dve zariadenia a teda dátové linky sú správne zakončené. Nie je teda potrebné pridávať ďalšie zakončovacie rezistory k obvodu fyzickej vrstvy. (Naviac použitý obvod USB3343, určený na použitie v perifériách, tieto rezistory integruje v puzdre obvodu a umožňuje ich voliteľne zaradiť do dátovej linky.)



Obr. 5.2: Schéma zapojenia obvodu fyzickej vrstvy s transceiverom USB3343

Doska plošných spojov (obr. 5.3) bola navrhnutá v programe KiCAD. Najskôr bolo navrhnuté rozloženie pripojovacích konektorov tak, aby prípravok umožňoval jednoduchú manipuláciu: prípravok sa nasadí priamo na konektor P1 dosky Minerva a pomocou konek-

torov USB sa prepojí s hostiteľským a analyzovaným zariadením. Medzi tieto konektory bol vložený obvod USB3343 (U1) spolu s hodinovým kryštálom a ďalšími pasívnymi súčiastkami. Následne boli položené prepojovacie cesty podľa schémy zapojenia. Dátové signály USB (D+/D-) boli vedené ako diferenciálny pár s kompenzáciou dĺžky. Dĺžka cesty hodinového signálu bola kompenzovaná voči dĺžkam ciest dátových liniek. Na zadnej strane dosky bola použitá tzv. rozlievaná meď slúžiaca ako spoločná zem (GND).



Obr. 5.3: Návrh plošného spoja obvodu fyzickej vrstvy

5.2 Komunikačné rozhranie medzi FPGA a MCU

USB komunikácia spracovaná v FPGA bude ďalej preposielaná do MCU, preto je nutné navrhnúť vhodné komunikačné rozhranie medzi týmito dvoma prvkami. Z možností realizovateľných na kite Minerva bolo zvolené rozhranie FlexBus, pretože prítomný mikrokontrolér disponuje radičom zbernice FlexBus a potrebné spoje medzi FPGA a MCU sú vytvorené priamo na kite.

Integrovaná periféria FlexBus radiča umožňuje komunikovať s externými obvodmi pripojenými k MCU pomocou mapovania do pamäťového priestoru procesora. Adresa a dáta sú prenášané multiplexovane po paralelnej 32-bitovej zbernici. Komunikáciu iniciuje vždy MCU vystavením adresy na A/D časť zbernice a aktiváciou riadiacich signálov. Po vystavení adresy nasleduje prenos dát - podľa typu transakcie adresovaná periféria číta alebo zapisuje dátové slovo na zbernicu. Ukončenie transakcie potvrdzuje periféria. Jedná sa teda o typ master-slave komunikácie. Radič zbernice v MCU je konfigurovateľný, umožňuje nastavenie časovania zbernice, voľbu šírky prenášaného slova a voľbu dĺžky adresy [14].

Pre toto rozhranie bude potrebné v FPGA realizovať zodpovedajúci radič. Podľa princípu fungovania zbernice FlexBus opísaného vyššie, bude radič v FPGA plniť tieto funkcie:

- rozpoznanie začiatku transakcie,
- dekódovanie adresy vystavenej na zbernicu,
- prenos dátového slova zo zbernice (transakcia zápisu) a na zbernicu (transakcia čítania),
- potvrdenie ukončenia transakcie.

5.3 Komunikačné rozhranie medzi analyzátorom a PC

Vývojový kit Minerva disponuje viacerými komunikačnými rozhraniami: USB, HDMI, Ethernet, JTAG, analógový audio vstup a výstup, GPIO piny. Z týchto rozhraní boli vylúčené jednoúčelové alebo inak nevhodné rozhrania (HDMI, audio a JTAG). Z dvojice USB a Ethernet bolo zvolené rozhranie Ethernet. Vďaka tomu nebude potrebné programovať systémový ovládač (ako pri USB) a analyzátor bude možné použiť na diaľku (po lokálnej sieti LAN alebo cez internet). Hoci USB 2.0 má vyššiu maximálnu prenosovú rýchlosť ako je 100 Mbps rozhrania Ethernet (obmedzenie vyplýva z prítomného radiča, viď kapitolu 4.3), pri analýze prenosov USB sa nepredpokladá využitie celej šírky pásma. Analyzované pakety budú filtrované, vhodne skracované a ukladané do vyrovnávacej pamäte tak, aby bola prenosová rýchlosť dostatočná.

Podľa vrstvového modelu ISO/OSI [11] bude nad fyzickou vrstvou Ethernetu použitá sieťová a transportná vrstva TCP/IP. Nad ňou bude použitý aplikačný protokol HTTP. Protokoly boli zvolené s ohľadom na ich univerzálnosť. Použitie protokolu HTTP umožní použitie analyzátoru bez inštalácie špecifického ovládacieho softwaru, nakoľko bude možné vytvoriť rozhranie pracujúce v štandardnom webovom prehliadači.

5.4 Užívateľské rozhranie

Užívateľské rozhranie bude slúžiť na prezeranie a skúmanie zachytených paketov z USB zbernice, nastavovanie filtrov pre ich výber a celkové ovládanie analyzátora.

5.4.1 Požiadavky na užívateľské rozhranie

Pri návrhu užívateľského rozhrania boli najskôr stanovené požiadavky, ktoré by malo toto rozhranie spĺňať. O tom, do akej miery a akým spôsobom boli tieto požiadavky naplnené hovoria nasledujúce podkapitoly.

- Jednoduchosť použitia
- Kompatibilita s bežnými platformami (OS Linux a Windows)
- Práca so skúmanými údajmi v reálnom čase
- Použitie zaužívaných ovládacích prvkov

5.4.2 Analýza existujúcich riešení

V snahe priblížiť sa k existujúcim nástrojom na analýzu (nielen) USB komunikácie a potenciálne priniesť vylepšenie, v tejto časti práce rozoberám existujúce riešenia z pohľadu ich užívateľských rozhraní. Podľa typu je možné užívateľské rozhrania rozdeliť na rozhrania s príkazovým riadkom (CLI) a grafické (GUI) [12].

Rozhrania s príkazovým riadkom

Ako príklad rozhrania s príkazovým riadkom uvádzam pôvodný software k analyzátoru OpenViszla. Program sa spúšťa príkazom `ovctl.py` a zadaním požadovanej akcie (pre analýzu USB komunikácie je to príkaz `sniff`), prípadne spolu s ďalšími parametrami. Spustenie je teda jednoduché a nástroj nevyžaduje takmer žiadnu konfiguráciu.

```
[   F    ]   0.267814 d=  0.267814 [   .0 +267814.250] [  0]
[        ]   1.123487 d=  0.855673 [255.7 +  0.600] [  3] IN   : 2.1
[        ]   1.123488 d=  0.000001 [255.7 +  1.250] [  1] ACK
[        ]   1.224924 d=  0.101436 [101.2 + 54.550] [  3] SETUP: 2.0
[        ]   1.224924 d=  0.000000 [101.2 + 54.967] [ 11] DATA0: a3 00 00 00 02
00 04 00 f6 e1
[        ]   1.224927 d=  0.000003 [101.2 + 57.933] [  3] IN   : 2.0
[        ]   1.224928 d=  0.000001 [101.2 + 58.633] [  1] ACK
[        ]   1.224930 d=  0.000002 [101.2 + 60.800] [  3] OUT  : 2.0
[        ]   1.224930 d=  0.000000 [101.2 + 61.217] [  3] DATA1: 00 00
[        ]   1.225175 d=  0.000245 [101.4 + 55.900] [  3] SETUP: 2.0
[        ]   1.225175 d=  0.000000 [101.4 + 56.317] [ 11] DATA0: 23 01 14 00 02
00 00 00 ef 95
```

Výpis 5.1: Príklad textového výstupu nástroja OpenViszla. Zdroj: [15]

Výstup nástroja je čisto textový, ako je ukázané na výpise 5.1. Jednotlivé riadky výstupu poskytujú informácie o type paketu, čase prijatia, príznakoch a dĺžke paketu. Výstup je možné zachytiť do súboru pomocou presmerovania. Chýba však akákoľvek interaktivita - po spustení nástroja je možné len sledovať výstup, nie je možné filtrovanie paketov, zobrazovanie detailov ani zmena nastavení.

Ako bolo spomenuté v kapitole 3, nástroj je napísaný v jazyku Python.

Grafické rozhrania

Ako príklad grafického rozhrania uvádzam analyzátor Wireshark [25]. Hoci sa nejedná priamo o analyzátor USB komunikácie, ale najmä sieťový analyzátor, spôsob použitia je rovnaký. Po spustení program umožňuje načítať zachytenú komunikáciu zo súboru, alebo spustiť nové zachytávanie na zvolenom rozhraní. Pakety zachytenej komunikácie sa zobrazujú v hornej časti okna ako riadky tabuľky. Kliknutím myšou na konkrétny riadok je možné tento paket označiť a jeho detaily sa zobrazia v dolnej časti okna. Detaily paketu sú organizované hierarchicky - po sieťových vrstvách. Detaily z každej vrstvy je možné zobraziť podrobnejšie pomocou kliknutia na tlačidlo `[+]` (rozbaliť). V prípade binárnych protokolov sa zobrazuje hexadecimálna hodnota jednotlivých častí a ich význam. Ak sú tieto údaje vo význame príznakov organizované ako bitové pole, príznaky sú priradené k jednotlivým bitom. Textové protokoly sú zobrazené v pôvodnej forme, často rozdelené na hlavičku a telo.

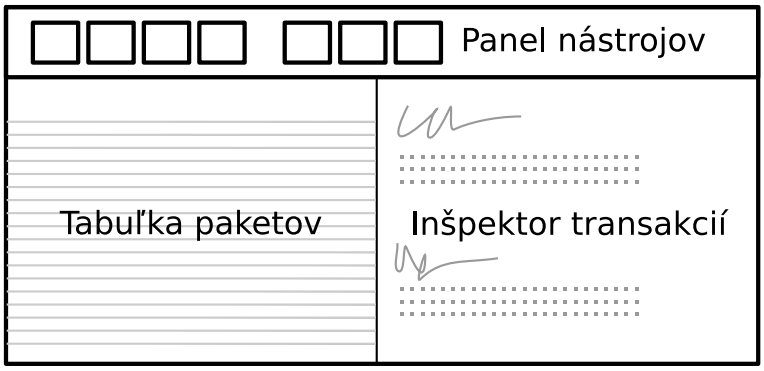
V hornej časti obrazovky je možné zadať logický výraz, ktorý slúži ako filter - zobrazované budú len pakety, pre ktoré je výraz pravdivý. Výraz je možné zadať textovo, alebo skladať výberom z možností pomocou editora.

Okrem základného zobrazenia paketov program umožňuje hlbšiu analýzu. Analýza sa spúšťa z kontextového menu výberom príslušnej položky. Takto je možné sledovať skupiny paketov, extrahovať z nich telo správy, vyhľadávať pakety so zhodou v niektorom z atribútov a podobne.

Tento nástroj je napísaný v jazyku C resp. C++ a užívateľské rozhranie je vytvorené pomocou frameworku Qt. V starších verziách bolo grafické rozhranie vytvorené pomocou knižnice GTK.

5.5 Návrh užívateľského rozhrania

V prvej fáze bola navrhnutá štruktúra užívateľského rozhrania ako na obr. 5.4. Panel nástrojov bude slúžiť na ovládanie analyzátora. Tabuľka paketov bude zobrazovať zachytené pakety. V pravej časti okna sa bude nachádzať inšpektor transakcií, ktorý bude po zvolení transakcie z tabuľky paketov zobrazovať detailnejšie informácie o celej transakcií aj o jednotlivých paketoch.



Obr. 5.4: Diagram zobrazujúci návrh užívateľského rozhrania.

5.6 Vybrané technológie

Ako veľmi praktické na analýzu komunikácie hodnotím interaktívne grafické užívateľské rozhranie. Na rozdiel od textového rozhrania umožňuje voliť mieru detailov zobrazených informácií. Užívateľ sa pri menej detailnom zobrazení lepšie orientuje vo veľkom množstve zachytených paketov. Potom je možné sa sústrediť na vybrané detaily v podrobnejšom zobrazení. Preto ako užívateľské rozhranie volím pre túto prácu cestu grafického rozhrania.

Na rozdiel od uvedených príkladov, ktoré vyžadujú prítomnosť ovládacieho programu na PC užívateľa, navrhujem riešenie, ktoré umožní používať analyzátor bez špecializovaného softwaru: obslužný software pobeží priamo v analyzátore a užívateľské rozhranie bude zobrazované pomocou bežného internetového prehliadača. Tomuto zodpovedá použitie komunikačného rozhrania Ethernet dostupného na zvolenej platforme a použitie protokolu a HTTP z rodiny TCP/IP.

Ako technológie na vytvorenie užívateľského rozhrania volím štandard HTML5 a jazyk kaskádových štýlov CSS spolu s programovacím jazykom javascript. Tieto technológie sú podporované všetkými bežnými internetovými prehliadačmi.

Kapitola 6

Implementácia USB analyzátora

6.1 Realizácia obvodu fyzickej vrstvy

Obvod navrhnutý v časti 5.1 bol realizovaný na obojstrannú DPS. Najskôr bol osadený transceiver USB: po nanesení spájkovacej pasty bolo puzdro súčiastky uložené na miesto a spájkovanie bolo vykonané pomocou teplovzdušnej pištole. Ostatné súčiastky boli zaspájkované perom s mikrohrotom. Konektory USB a hrebeňový konektor boli osadené ako posledné.

6.1.1 Oživenie obvodu

Počas písania tejto práce bol obvod fyzickej vrstvy zhotovený skôr, ako zodpovedajúci program umožňujúci jeho otestovanie. Preto boli navrhnuté a vykonané čiastočné testy funkčnosti pri oživení obvodu.

Ako prvé je možné otestovať transparentnosť obvodu pre analyzovanú zbernicu. Ak nie je obvod pripojený k Minerva kitu, transceiver USB nie je napájaný. Pripojením prípravku k počítaču pomocou kábla s USB B konektorom a pripojením USB zariadenia na USB A konektor je možné skontrolovať, či systém pripojené zariadenie rozpozná, čím sa overí celistvosť USB zbernice. Ďalej privedením napätia +3.3V na piny 1,3 a 5 hrebeňového konektoru a napätia 5V na piny 2, 4 (prípadne zapnutím prepojky P1) sa obvod uvedie do činnosti. Obvod po resete nemá aktivované výstupné budiče zbernice, preto nesmie narušiť komunikáciu na zbernici. Môžeme však pozorovať prítomnosť hodinového signálu na pine 35.

6.2 Prvky implementované v FPGA

Táto podkapitola ukazuje implementačné detaily jadra USB analyzátora implementovaného v FPGA. Základ bol prevzatý z projektu OpenViszla opísaného v kapitole 3.2.2. Jeho fungovanie a použité úpravy sú opísané ďalej.

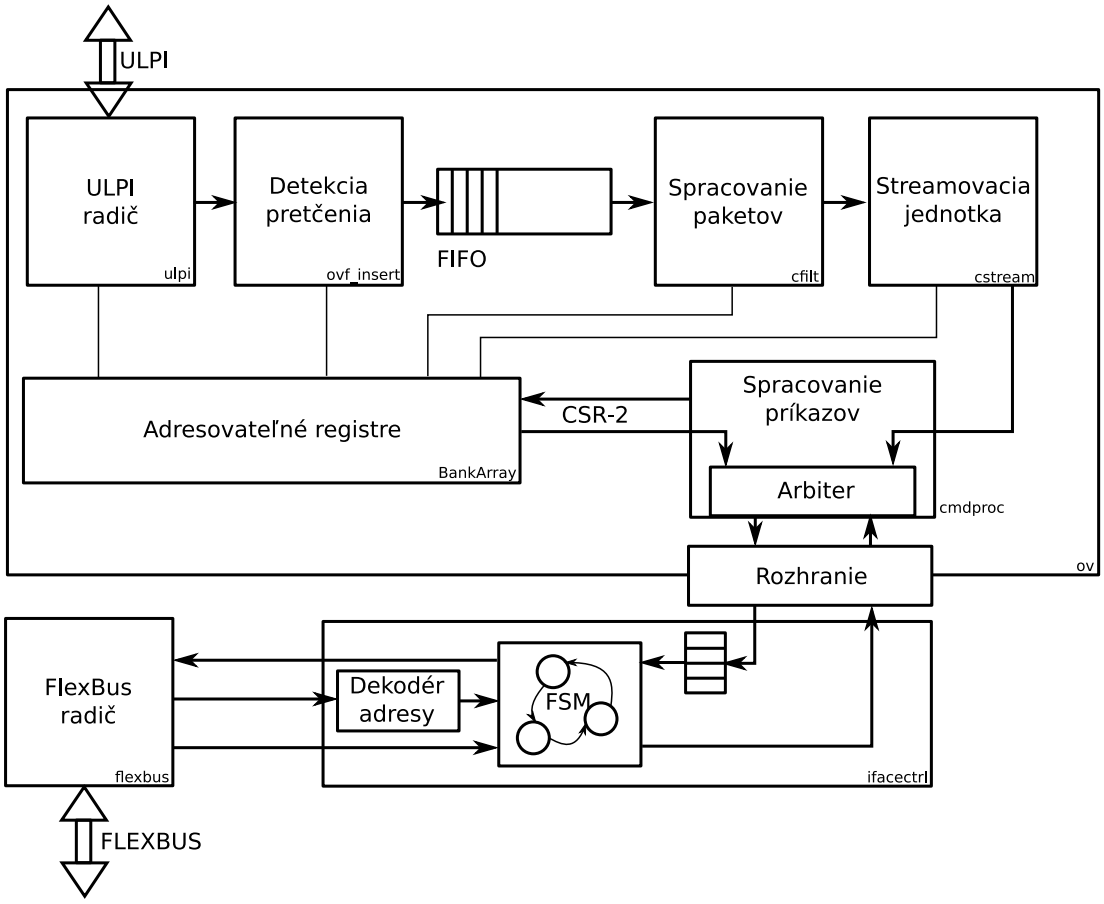
6.2.1 Architektúra a prepojenie modulov

Architektúra jednotky `minerva_top` na najvyššej úrovni reprezentuje samotný obvod FPGA a jeho vstupy a výstupy. Tie sú rozdelené podľa funkcie nasledovne:

- zozhranie ULPI - signály `ulpi_data`, `ulpi_dir`, `ulpi_nxt`, `ulpi_stp`, `ulpi_rst` a `ulpi_clk`,

- zbernica FlexBus - signály `fxb_ad`, `fxb_clk`, `fxb_cs`, `fxb_rw`, `fxb_ta`,
- systémový hodinový signál `refclk`,
- pomocné výstupy - indikačné LED diódy.

Jednotlivé skupiny signálov sú vedené do zodpovedajúcich jednotiek, ktorých činnosť je opísaná ďalej. Ich prepojenie zobrazuje obrázok 6.1.



Obr. 6.1: Vzájomné prepojenie jednotiek implementovaných v FPGA.

Radič FlexBus

Radič, ktorý vo svojej diplomovej práci implementoval Petr Buchta[3] bol upravený, aby poskytoval vyššiu priepustnosť a boli vylepšené niektoré jeho nedostatky. Použitá šírka zbernice bola zvýšená z 16 na 32 bitov.

Pôvodný radič spôsoboval zatuhnutie komunikácie v prípade, že MCU bolo inicializované skôr, ako FPGA. V praxi nie je možné vždy dodržať poradie inicializácie týchto dvoch prvkov, napríklad môže dôjsť k resetu MCU bez vynútenia resetu FPGA. Potom nastala situácia, kde MCU zahájilo transakciu, ale radič v FPGA začiatok transakcie zmeškal a nemohol na ňu zareagovať. Preto transakcia nebola zo strany FPGA potvrdená, takže nemohla byť ani dokončená. V prípade použitia blokujúceho režimu na strane MCU došlo k úplnému

zastaveniu činnosti MCU. Preto bol do radiča pridaný inicializačný stav, ktorý po resete radiča ruší aktuálne prebiehajúcu transakciu na zbernici.

Radič je implementovaný pomocou stavového automatu s 6 stavmi. Prvý je spomínaný inicializačný stav, v ďalších stavoch sa čaká na začiatok transakcie, dáta zo zbernice sa ukladajú do registra (resp. obsah registra sa vystavuje na zbernicu) a čaká sa na potvrdenie transakcie.

Po zahájení transakcie sa s nasledujúcim hodinovým taktom vystaví adresa zo zbernice na výstup radiča. Typ transakcie (čítacia alebo zápisová) je určený kombinačne. Túto adresu dekódujú ďalšie moduly pripojené k radiču. V ďalšom hodinovom takte je v prípade zápisovej transakcie vystavený obsah zbernice na dátový výstup radiča `WDATA`. Modul, ktorému náleží daný adresový priestor je ďalej zodpovedný za ukončenie transakcie aktiváciou signálu `ACK`. V prípade čítacej transakcie sú dáta vystavené na zbernicu v takte nasledujúcom po aktivácií `ACK`. Transakcia je ukončená s ďalším taktom, aktiváciou signálu `FXB_TA` zbernice FlexBus.

6.2.2 Radič ULPI

Radič ULPI slúži na komunikáciu s obvodom USB3343 prítomným na doske fyzickej vrstvy. Tento obvod zabezpečuje ovládanie transceiverov USB a (de-)serializáciu USB komunikácie. Zbernica ULPI je synchrónna, dátové a riadiace signály sú vzorkované pri nábežnej hrane hodinového signálu. Definovaných je 5 druhov správ: zápis do registra, čítanie registra, vysielanie USB paketu a príjem USB paketu a stavová informácia RXCMD.

Správu o príjme USB paketu zahajuje transceiver kedykoľvek, keď je zbernica voľná (označujeme stavom *Idle*). Zápis do registra a čítanie z registra zahajuje FPGA. Pri čítaní z registra dôjde k prebratiu zbernice transceiverom a odoslaniu hodnoty uloženej v registri. Túto zmenu označujeme ako *turn around*) [1]. Príkaz na odoslanie USB paketu nebol v projekte USB analyzátora použitý.

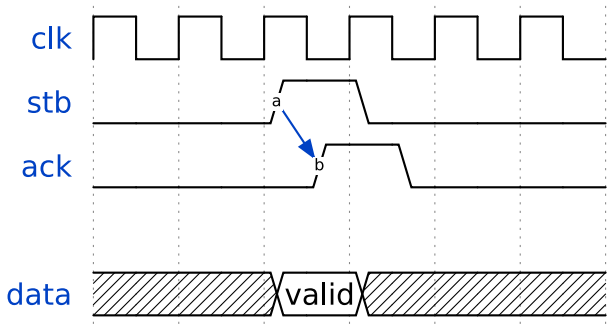
V prípade, že sa zmení stav USB zbernice (napr. pri pripojení alebo odpojení zariadenia alebo zahájení vysielania paketu), USB PHY o tom informuje správou RXCMD. Tá nahradzuje out-of-band signálne vodiče rozhrania UTMI, z ktorého špecifikácia ULPI vychádza.

Radič ULPI je implementovaný ako stavový automat, kde jednotlivé stavy zodpovedajú stavom zbernice podľa špecifikácie ULPI. Riadiace signály zbernice sú určené kombinačne od stavu automatu. Prechody automatu sú synchrónne s hodinovým signálom `ULPI_CLK`.

Činnosť radiča je riadená hodnotami konfiguračných registrov. Prehľad registrov a ich význam je uvedený v prílohe B.

Tieto registre sú súčasťou súboru registrov adresovateľného zo strany MCU. Transakcie zápisu a čítania registrov transceiveru sa odohrávajú nastavením požadovanej činnosti v konfiguračných registroch radiča ULPI a spustením transakcie zápisom príslušného príznaku.

Pakety prijaté zo zbernice USB, predané po rozhraní ULPI do radiča ULPI sú ďalej predávané inou, efektívnejšou cestou: dáta sú vystavené na rozhranie s obojstranným potvrdením. Radič ULPI po vystavení dát aktivuje signál označujúci túto udalosť(`stb` - strobe) počas jedného taktu. Komponent pripojený k radiču tieto dáta prevezme a po ich spracovaní aktivuje signál potvrdenia (`ack`). To umožňuje rýchle predávanie dát bez nutnosti dlhých transakcií zahŕňajúcich adresovanie registra. Postupnosť signálov je zobrazená na obr. 6.2, signál ACK je určený kombinačne, takže prenos jedného slova trvá iba jeden takt.



Obr. 6.2: Postupnosť signálov výstupného rozhrania ULPI radiča.

6.2.3 Detektor pretečenia

Pakety USB sú spracovávané zreťazeným spôsobom a na konci zreťazenej linky sú predávané do modulu zabezpečujúceho komunikáciu s MCU. Keďže komunikácia je iniciovaná MCU a neprebieha nepretržite, na vyrovnanie rýchlostí sú do tejto cesty vložené fronty FIFO. Ak dôjde k zastaveniu alebo spomaleniu komunikácie a fronty FIFO sa naplnia, novo prijaté pakety nie je možné spracovať a budú zahodené. Aby bolo možné tento stav odhaliť, do dátovej cesty je vložený detektor pretečenia. V prípade, že na vstupe detektora sú platné dáta (aktívny signál `stb`), ale modul zaradený za ním nemôže ďalšie dáta prijať (nepotvrdil prijatie posledného paketu signálom `ack`), detektor tento stav zaznamená, inkrementuje počítadlo zahodených paketov a do prúdu dát vloží špeciálny (tzv. *magic*) paket označujúci udalosť pretečenia.

6.2.4 Spracovanie paketov a detektor stavu linky

Výnimočné udalosti (ako napr. stav pretečenia) sú signalizované špeciálnou správou (tzv. *magic packet*). Táto správa je odchytená v jednotke spracovania paketov. Podľa typu správy sú nastavené príznaky, ktoré sa ďalej prenášajú mimo samotného obsahu dátových paketov (mimo hlavného pásma, *out-of-band*). Takto sú signalizované nasledovné stavy:

- Začiatok USB paketu - `SOP` - príznak je nastavený pri prvom prenesenom byte USB paketu
- Koniec USB paketu - `EOP` - príznak je nastavený na poslednom prenesenom byte USB paketu
- Chyba - `ERR` - príznak je nastavený v prípade nešpecifikovanej chyby prenosu
- Pretečenie - `OVF` - príznak je nastavený v prípade prijatia správy o udalosti pretečenia z detektoru pretečenia

Okrem príznakov je k paketu pridaná časová značka.

6.2.5 Fronta FIFO

Ďalej je v dátovej ceste zaradená fronta FIFO, ktorá slúži na oddelenie dvoch častí pracujúcich na rozdielnych taktovacích frekvenciách. Časť pred frontou FIFO je taktovaná z externého obvodu s USB transceiverom, pretože prenos paketov musí byť synchronizovaný so zbernicou USB. Časť za touto frontou zasa musí byť synchronizovaná s prenosom

dát do MCU a teda je taktovaná z hodinového signálu zbernice FlexBus. Fronta má nastaviteľnú dĺžku, konkrétna použitá dĺžka bola 32768 správ o šírke 9 bitov. 8 bitov tvoria samotné dáta paketu a 1 bit je príznak.

6.2.6 Streamovacia logika

Generovanie prúdu dát so zachytenými USB paketmi je zabezpečené jednotkou `Whacker`, ktorá funguje na princípe producent-konzument. Producent prijíma dáta z predchádzajúcich jednotiek, identifikuje jednotlivé pakety a každému paketu pridá hlavičku. Tieto dáta sú uložené do medzipamäte BRAM tvorenej prostriedkami FPGA.

Hranica po sebe idúcich paketov je určená na základe príznakov `SOP` a `EOP`. Hlavička nesie informácie o dĺžke paketu, časovej značke a prípadných chybových stavoch. Chybové stavy sú zistené z prítomnosti príznakov `ERR` a `OVF` a sú akumulované do hlavičky. Ďalší chybový stav môže nastať, ak je paket dlhší ako je veľkosť bufferu. Vtedy je dátová časť paketu orezaná a táto skutočnosť je označená príznakom `CLIP` v hlavičke paketu.

Konzument potom vyberá 8-bitové bloky dát z medzipamäte a posiela ich smerom k výstupnému rozhraniu.

6.2.7 Komunikačné rozhranie

Pôvodný projekt OpenViszla predpokladal použitie USB prevodníka FT232H na komunikáciu analyzátoru s počítačom. V mojej práci však FPGA jadro analyzátoru komunikuje s MCU pomocou radiča FlexBus opísaného v časti 6.2.1. Preto bola logika spracúvajúca sériovú komunikáciu nahradená jednotkou, ktoré sa dá použiť s týmto radičom.

Táto jednotka je definovaná v súbore `interfacectrl.vhd` ako VHDL entita a jej rozhranie pozostáva z dvoch častí: prvá časť komunikuje s radičom FlexBus a druhá s modulom reprezentujúcim projekt OpenViszla (ďalej len `OV`). Komunikácia s `OV` prebieha po dvoch rozhraniach s dátovou šírkou 8 bitov a signálmi označujúcimi, či sú dáta pripravené na čítanie `OV_READABLE` a či je možné zapísať nové dáta `OV_WRITABLE`.

Jednotka komunikačného rozhrania dekóduje adresu zo zbernice FlexBus a v prípade zhody prechádza do režimu komunikácie. Pre čítaciu transakciu sú dáta zapísané na FlexBus a transakcia je ukončená. Pre zápisovú transakciu sa dáta vystavia na zápisové rozhranie `OV`, ale v prípade, že signál `OV_WRITABLE` je neaktívny, čaká sa na uvoľnenie miesta pre zápis a transakcia sa ukončí až po aktivácií tohoto signálu zo strany `OV`. S ukončením transakcie je na jeden takt aktivovaný signál `OV_WE` (*write enable*) čím sa dáta presunú do `OV`.

Pre lepšie využitie zbernice FlexBus nie je pri čítaní prenášaný iba jeden byte, ale najviac 3 byty dát. Keďže z `OV` môžu byť dáta vyčítavané len po jednotlivých bytoch, nachádza sa v tejto jednotke zostavovací buffer. V bufferi môže byť prítomných 0 až 3 platných dátových bytov. So začiatkom čítacej transakcie je buffer zmrazený a z jeho obsahu je zostavená správa, ktorá bude prenesená. Z 32 bitovej šírky FlexBus-u je jeden byte využitý na prenos príznakov a informácie o platnosti jednotlivých dátových bytov.

Keďže zápisy smerom do analyzátora zďaleka nedosahujú takú početnosť ako opačný smer komunikácie, táto optimalizácia nebola pre zápisové transakcie použitá.

6.2.8 Obmedzenia syntézy

Pri syntéze HW z popisu v HDL jazyku je potrebné špecifikovať obmedzenia (*constraints*), ktoré vplývajú na výsledok syntézy. V tomto projekte sú obmedzenia popísané v súbore `net.ucf` a sú to:

1. obmedzenia upravujúce pripojenia signálov top-level VHDL entity na fyzické piny puzdra obvodu FPGA
2. obmedzenia špecifikujúce frekvenciu taktovacieho signálu
3. obmedzenia špecifikujúce hranice časových domén

Prvý typ obmedzení je potrebný na správne priradenie výstupných signálov na zodpovedajúce cesty DPS pripájajúce FPGA k iným obvodom.

Druhý typ slúži nástrojom syntézy na správne navrhnutie logických ciest a kombinačnej logiky tak, aby boli dodržané *setup* a *hold* časy registrov. V prípade, že by sa syntetizačným nástrojom nepodarilo zostaviť obvod tak, aby časovanie bolo dodržané, syntéza skončí s chybou a je o tom informovaný programátor.

Tretí typ obmedzení spôsobí ignorovanie pravidiel časovania na hranici medzi dvoma nezávislými časovými doménami. Tu syntetizačné nástroje nedokážu určiť, akým spôsobom je zabezpečené dodržanie časovania (napr. použitým komunikačným protokolom) a táto zodpovednosť ostáva na programátorovi [26].

6.2.9 Časovací subsystém

V predchádzajúcich častiach boli popísané jednotlivé moduly a synchrónne udalosti boli popisované vzhľadom na lokálny hodinový signál. V tejto časti je popísaný globálny pohľad na zdroje taktovacieho signálu a ich použitie v jednotlivých moduloch. Hodinové signály vstupujúce do FPGA sú:

- `ULPI_CLK` generovaný USB transceiverom USB3343, s ktorým sú synchronizované prenosy po rozhraní ULPI. Taktovacia frekvencia tohoto signálu je 60 MHz [17].
- Hodinový signál `REFCLK` o frekvencii 50 MHz generovaný oscilátorom, ktorý je štvornásobným hodinovým bufferom NB3N551 [16] rozvetvený do MCU, FPGA a ďalších obvodov na doske Minerva [27].
- Hodinový signál `FLEXBUS_CLK` zbernice FlexBus generovaný z MCU. Jeho frekvencia je odvodená od hlavného hodinového signálu MCU. V tejto práci bola použitá maximálna podporovaná frekvencia FlexBus a to 50 MHz [7].

Radiče ULPI a FlexBus pracujú synchrónne s hodinovým signálom prislúchajúcim k danej zbernici. Ich hodinové domény sa stretajú s doménou jadra analyzátoru v jednotkách FIFO fronty 6.2.5 a streamovacej logiky 6.2.6. Jadro je taktované hodinovým signálom odvodeným z `REFCLK` pomocou obvodu fázového závesu PLL v FPGA. Táto frekvencia je konfigurovateľná pomocou parametrov PLL. Použité bolo zdvojnásobenie frekvencie na 100 MHz.

6.3 Zostavenie a syntéza pomocou nástroja Migen

Projekt OpenViszla využíva nástroj Migen na syntézu logických obvodov z popisu s vysokou úrovňou abstrakcie (tzv. *high level synthesis*). Jedná sa o knižnice programovacieho jazyka Python, ktoré definujú nové funkcie a triedy, pomocou ktorých je možné vytvoriť syntetizovateľný HDL popis hardware s použitím konštruktov podobných objektovému programovaniu. Táto podkapitola opisuje konštrukty použité v projekte OpenViszla ako aj pri portovaní tohoto projektu na výukový kit Minerva.

6.3.1 Syntéza systémovej zbernice

Zbernica označovaná ako CSR-2 je určená na prístup ku konfiguračným a stavovým registrom, je šetrná k prostriedkom a ploche obvodu v FPGA, má však nízku priepustnosť. Tento typ zbernice je vytvorený automaticky inštancovaním triedy `Interconnect` z knižnice `migen.bus.csr`. Je k nej možné pripojiť adresovateľné registre.

V projekte boli použité dva typy registrov. Prvý z nich, `CSRStatus` je určený len na čítanie (zo strany zbernice) a spravidla sa používa ako stavový register. Druhý typ, `CSRStorage` je určený na čítanie aj zápis a teda býva použitý ako konfiguračný register. Registre majú voliteľnú šírku. V prípade, že šírka registra je väčšia ako šírka zbernice, register je automaticky rozdelený na časti zodpovedajúce šírke zbernice. Zápis do takéhoto registra môže byť atomický, vyžaduje to však dvojnásobné množstvo prostriedkov FPGA (na vytvorenie skladacieho bufferu) ako register bez atomického zápisu. Registre môžu tvoriť adresovateľný súbor registrov pomocou ich zoskupenia do `migen.bank.csrgen.BankArray`. Trieda `BankArray` potom zabezpečí aj automatické priradenie adries jednotlivým registrom.

6.3.2 Syntéza komunikačných rozhraní

Na komunikáciu medzi jednotlivými modulmi Migen generuje jednosmerné rozhrania. Rozhranie typu `Source` sa používa na výstup dát z modulu a rozhranie typu `Sink` na vstup dát. Rozhrania sú synchrónne, so signálom *strobe* a s potvrdzovaním pomocou `ack`. Dáta sú dostupné na signáloch *payload*.

Šírka rozhrania je určená použitým dátovým typom. Ten môže byť štruktúrovaný, čo pomáha čitateľnosti a zrozumiteľnosti zdrojového kódu.

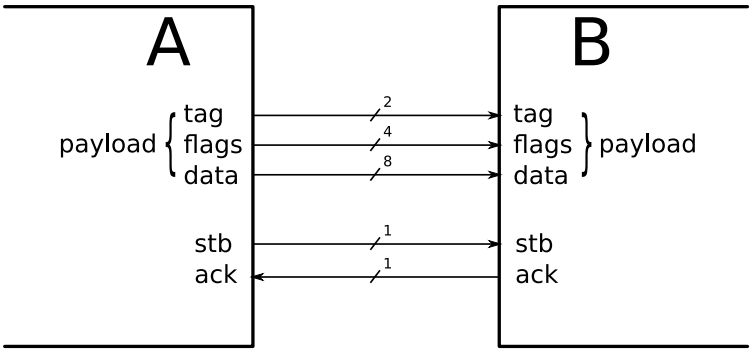
Prepojená môže byť vždy len dvojica rozhraní, kde jedno rozhranie je typu `Source` a druhé `Sink`, pričom obe rozhrania musia byť rovnakého dátového typu. Príklad takéhoto prepojenia je na obr. 6.3 a výpise 6.1.

```
from migen.flow.actor import Source, Sink
# deklarácia štrukturovaného dátového typu
# (2 bity pre Tag, 4 bity pre príznaky, 8 bitov dát)
PACKET_TYPE = [('tag', 2), ('flags', 4), ('data', 8)]

# vytvorenie rozhraní
A = Source(PACKET_TYPE)
B = Sink(PACKET_TYPE)

# prepojenie A -> B
B.connect(A)
```

Výpis 6.1: Príklad vytvorenia komunikačných rozhraní pre štrukturovaný dátový typ v jazyku Python s použitím knižnice Migen.

6.3.3 Syntéza FIFO front a pamäte BRAM

Fronty FIFO majú podobné rozhranie ako sú `Source` a `Sink` uvedené vyššie. Pre zápis do fronty sa použije dátový vstup `din` (*data input*) a zápis sa vykoná v nasledovnom takte po aktivácií signálu na povolenie zápisu `we` (*write enable*). Údaj z čela fronty je vždy vystavený



Obr. 6.3: Diagram zobrazujúci vytvorené rozhrania z výpisu 6.1

na výstup `dout` (*data output*), posun na ďalšiu položku nastane opäť v takte nasledujúcom po aktivácií signálu `re` (*read enable*). Informácia o stave fronty je dostupná z dvoch signálov - `writable` je v log. 1 ak fronta nie je plná, tzn. je možný zápis ďalšej položky. Signál `readable` je aktívny práve vtedy, ak je fronta neprázdna, tzn. na výstupe `dout` sú platné dáta.

Fronty v knižnici Migen sú dvojakého typu. Synchrónna fronta `migen.genlib.fifo.SyncFIFO` má obe rozhrania (pre čítanie aj zápis) synchrónne k jednej časovej doméne. Príklad takejto fronty sa nachádza na výpise 6.2. Oproti tomu fronta `migen.genlib.fifo.AsyncFIFO` pracuje s dvoma časovými doménami, jednou pre čítanie a druhou pre zápis.

Aby bolo možné FIFO fronty priamo pripojiť k iným modulom s rozhraniami `Source` a `Sink`, v knižnici sú definované aj obaľovacie triedy `migen.actorlib.fifo.SyncFIFO` a `AsyncFIFO`, ktoré signály uvedené vyššie obaľujú do `Source` a `Sink` rozhraní.

Pamäťový priestor fronty FIFO je tvorený pamäťou BRAM vytvorenou z prostriedkov FPGA. Tá je reprezentovaná triedou `migen.fhdl.specials.Memory`, jej inštancie majú konfigurovateľnú veľkosť (kapacitu pamäte aj dĺžku slova). Pripojenie k iným modulom sa deje pomocou portu vytvoreného volaním metódy `get_port`. Port pamäte má dátovú a adresovú časť, riadiace signály `we` a `re` a hodinový signál. Z každej inštancie je vygenerovaný príslušný HDL popis pamäte a jej rozhraní. Za jej implementáciu v FPGA je potom zodpovedný použitý syntetizačný nástroj.

```
from migen.genlib.fifo import SyncFIFO
myFIFO = SyncFIFO(8, 256)
```

Výpis 6.2: Príklad vytvorenia FIFO fronty s kapacitou 256 položiek a šírkou dát 8 bitov.

6.3.4 Syntéza konečných stavových automatov

Konečný stavový automat sa vytvorí ako inštancia triedy `migen.genlib.fsm.FSM`. Jednotlivé stavy sa definujú volaním metódy `act` s názvom nového stavu a zoznamom príkazov, ktoré sa majú v danom stave vykonať. Príklad vytvorenia stavového automatu sa nachádza na výpise 6.3.

```
from migen.genlib.fsm import FSM, NextState
from migen.fhdl.std import *

fsm = FSM()
input = Signal()
output = Signal()

# stav INIT: ak je vstup aktívny, prejdi do STATE1
fsm.act("INIT", If(input, NextState("STATE1")))
# stav STATE1: aktivácia výstupu, prechod do STATE2
fsm.act("STATE1", output.eq(1), NextState("STATE2"))
# stav STATE2: deaktivácia výstupu, návrat do stavu INIT
fsm.act("STATE2", output.eq(0), NextState("INIT"))
```

Výpis 6.3: Príklad vytvorenia stavového automatu s 3 stavmi.

6.3.5 Preklad do iných jazykov HDL

Cieľový HDL popis je vytvorený na základe inštancií objektov popísaných vyššie, ktoré tvoria nezávislé moduly. Tie sú nakoniec združené v jednom top-level objekte (podobne ako top-level entita VHDL). Pre vygenerovanie popisu HDL sa použije príslušný modul. V dobe písania práce bol dostupný len modul `migen.fhdl.verilog` pre prevod do jazyku Verilog. Preklad sa vykoná volaním funkcie `convert(top)` kde `top` je spomínaný top-level objekt.

Pre kompletnú syntézu z HDL popisu je ešte nutné špecifikovať cieľovú platformu. To sa vykoná inštanciovaním triedy `Platform` z príslušného balíka, napríklad `migen.build.xilinx` pre FPGA výrobcu Xilinx. Pre definovanie obmedzení (*constraints*) je možné vytvoriť podtriedu, kde sa tieto obmedzenia zahrnú. Platforma môže definovať aj spôsob syntézy z HDL popisu do konfigurácie pre dané FPGA. Tak je to aj v prípade `migen.mibuild.xilinx_ise` kde je implementované volanie nástrojov Xilinx ISE pre vykonanie syntézy.

V projekte OpenViszla je proces zostavenia implementovaný v súbore `build.py`, kde sa okrem vygenerovania HDL popisu a jeho syntézy vytvorí aj mapa všetkých adresovateľných registrov. Výstupom sú súbory v zložke `build`, najmä `top.v` obsahujúci Verilog HDL popis obvodu, `top.ucf` obsahujúci *constraints*, `map.txt` s mapou registrov a top.bin s výslednou konfiguráciou FPGA.

V tejto práci boli použité výstupy HDL popisu a mapa registrov. Top-level design z OpenViszla bol použitý ako jeden z modulov tejto práce. Ostatné moduly boli naprogramované v jazyku VHDL a syntéza sa vykonala z prostredia Xilinx ISE.

6.3.6 Spolupráca s komponentmi popísanými VHDL

Aby bolo možné zakomponovať design vygenerovaný pomocou Migen v tomto projekte, z Verilog popisu bolo vytvorené zodpovedajúce rozhranie VHDL entity. Túto entitu je potom možné štandardne inštancovať rovnako ako ostatné VHDL entity. Pre komunikáciu medzi modulom z OpenViszla a zvyškom projektu bol do OpenViszla pridaný komunikačný modul. Tento modul je implementovaný v súbore `interface.py` a slúži na namapovanie signálov vstupného rozhrania `Sink` a výstupného rozhrania `Source` na vstupné a výstupné signály top-level entity. Tieto signály sú potom pripojené k rozhraniu z kapitoly 6.2.7.

6.4 Obsluha analyzátoru mikrokontrolérom

V pôvodnom projekte OpenViszla bolo FPGA pripojené k PC pomocou USB a prevodníka na rýchlu sériovú linku. Obsluha analyzátoru prebiehala prostredníctvom užívateľskej aplikácie v jazyku Python. V tomto projekte je však FPGA pripojené k MCU a nízkoúrovňová obsluha bude prebiehať priamo z MCU. Užívateľské príkazy sú do MCU zadávané vhodným rozhraním cez PC, tie sú v MCU spracované a preložené do príslušných nastavení registrov obvodu v FPGA. Jednotlivé časti tejto podkapitoly sa zaoberajú implementáciou tejto obsluhy v MCU.

6.4.1 Freescale MQX RTOS

Pre platformu Minerva existuje implementácia operačného systému Freescale MXQ RTOS vytvorená na FIT VUT v spolupráci s firmou Freescale resp. NXP Semiconductor (Firma Freescale Semiconductor, Ltd. v roku 2015 vykonala fúziu s NXP Semiconductors N.V.).

Tento systém implementuje základné funkcie ako sú plánovač a prepínanie úloh, semafory, fronty, zasielanie správ. Modul `RTCS` implementuje obsluhu Ethernet radiča a TCP/IP stack vrátane jednoduchého HTTP serveru.

Tieto prvky výrazne uľahčia implementáciu úloh kladených na MCU v tomto projekte, preto bol ako základ zvolený tento operačný systém. Popis vlastností MQX RTOS bol voľne prevzatý z príručky k tomuto systému [10].

Prehľad

Freescale MQX RTOS (*Message Queue Executive Real Time Operating System*) je operačný systém reálneho času navrhnutý pre jednoprocesorové, viacprocesorové a distribuované vstavané systémy pracujúce v reálnom čase. Vychádza zo systému MQX, ktorý bol firmou Freescale upravený pre ich mikroprocesory.

MQX RTOS je rozdelený na jadro (obsahujúce plánovač, správu úloh a základné synchronizačné prostriedky) a voliteľné súčasti implementujúce obsluhu časovačov, systém predávania správ a udalostí, semafory a mutexy, fronty, logovanie, I/O a ďalšie funkcie.

Úlohy

Aplikácie bežiace na MQX sú organizované do úloh. V jednom čase beží (má priradený procesor) práve jedna úloha a úlohy sa v behu striedajú. Úlohu, ktorá má bežať, vyberá plánovač na základe priorít a zvoleného plánovacieho algoritmu. Tie sú k dispozícií dva: FIFO a round-robin. Plánovací algoritmus sa nevolí globálne, ale pre každú úlohu zvlášť. Tak je možné kombinovať oba.

Pri plánovaní FIFO sa vyberie tá úloha, ktorá má najvyššiu prioritu. Ak je pripravených viac úloh s rovnakou prioritou, vyberie sa tá, ktorá na beh čakala najdlhší čas. Úloha beží dovtedy, kým sa nevzdá procesoru (dobrovoľne, alebo volaním blokujúcej funkcie), nie je zastavená prerušením s vyššou prioritou alebo iná úloha s vyššou prioritou prejde do stavu `ready` (pripravená).

Pri plánovaní round-robin má úloha pridelený procesor počas nastaveného časového okna (*time slice*) a po jeho uplynutí je zastavená a namiesto nej pokračuje ďalšia úloha z čela fronty úloh pripravených na beh. Zastavená úloha sa vloží na koniec fronty pripravených úloh. Takto môže procesor spravodlivo zdieľať viaceré úlohy s rovnakou prioritou.

Synchronizačné prostriedky

V prípade potreby beh úloh synchronizovať, MQX RTOS poskytuje rozmanité prostriedky: udalosti, semafory, mutexy, zasielanie správ a fronty úloh.

Udalosti fungujú ako polia príznakov, označujúce výskyt udalostí, pričom úloha môže čakať na splnenie určitej kombinácie. Úloha čakajúca na udalosť je vyradená z fronty pripravených úloh. Keď daná udalosť nastane, t. j. iná úloha označí výskyt danej udalosti, dispatcher zaradí čakajúcu úlohu naspäť medzi pripravené úlohy.

Mutexy sú špeciálnym prípadom semaforu (semafor inicializovaný na hodnotu 1). Zabraňujú viacnásobnému vstupu do kritickej sekcie.

Zasielanie správ funguje pomocou front správ (*message queues*), kde jedna úloha správu zaradí na koniec fronty a iná úloha ju môže z čela fronty vybrať. Zápis aj čítanie správy majú blokujúce aj neblokujúce varianty. V prípade blokujúcej varianty je vykonávanie úlohy pozastavené, ak je daná fronta plná (resp. prázdna pri čítaní).

Udalosti, semafory a systém zasielania správ majú svoje odľahčené verzie, označené ako *lightweight*, zaberajúce menej systémových prostriedkov. Neposkytujú však toľko možností, ako ich plnohodnotné verzie. Je preto na programátorovi, aby určil, ktoré vlastnosti sú nevyhnutné pre jeho program a vybral vhodnú verziu týchto prostriedkov.

Konfigurácia a práca s perifériami MCU

O inicializáciu periférií sa stará MQX pred tým, než spustí užívateľské úlohy. Toto chovanie je napevno zabudované do programu operačného systému. Pre každý mikroprocesor, resp. pre každú variantu systému mikroprocesora s externými perifériami sa musí vytvoriť tzv. *Board support package* (BSP), ktorý obsahuje nízkoúrovňový kód pre inicializáciu a obsluhu HW daného systému [8].

Zostavenie

Pred použitím MQX RTOS v projekte je potrebné zostaviť samotný operačný systém. V distribúcií zdrojového kódu sú pribalené skripty na zostavenie pomocou viacerých sád nástrojov, ako sú GCC, IAR, UV4. Po zostavení OS sa vytvoria súbory knižníc a k nim zodpovedajúce hlavičkové súbory (súbory `.a` a `.h` v adresári `lib`). Tieto je možné zaradiť do projektu využívajúceho MQX.

6.4.2 Inicializácia aplikácie

Po zavedení MQX sa automaticky spustia úlohy označené príznakom `autostart` v tabuľke úloh. Pred použitím niektorých softwarových komponent je potrebná ich inicializácia. Tá prebehne na začiatku jednotlivých úloh:

- inicializácia rozhrania FlexBus,
- inicializácia TCP/IP stacku,
- vytvorenie synchronizačných semaforov a front.

Ako bolo spomenuté v kapitole 6.4.1, inicializáciu HW zabezpečuje priamo operačný systém. Predpokladané využitie zbernice FlexBus je pripojenie rozširujúcej pamäte alebo inej periférie založenej na pamäti (napr. display s videopamäťou). V tomto projekte sa však

FlexBus používa inak ako bolo predpokladané, preto bol kód inicializácie zbernice z BSP vyňatý a bol presunutý do aplikačnej časti, do funkcie `flexbus_init` volanej z úlohy `Ov_Task`. Inicializácia FlexBus-u pred naštartovaním aplikácie neumožňovala upravovať konfiguračné parametre (boli natvrdo zabudované do BSP).

Inicializácia sieťového TCP/IP stacku spočíva v nastavení IP adresy sieťového rozhrania a spustenia úloh zodpovedajúcich za chod HTTP servera.

Pamäťový priestor pre použité semafory a frontu správ je alokovaný staticky, ale tieto prostriedky treba najskôr inicializovať. Semafory `command_sem` a `response_sem`, ktorých použitie je opísané nižšie, sú inicializované na hodnoty 1 a 0 (v danom poradí), čo značí, že je možné zadať jeden príkaz do FPGA a ešte nie je dostupná žiadna odpoveď z FPGA.

Fronta správ `packet_queue` je po inicializácií prázdna.

6.4.3 Komunikácia s FPGA

Komunikáciu s FPGA zabezpečujú funkcie `fpga_write_byte` a `fpga_read_buffer`. Prvá z nich zapíše jeden byte do FPGA pomocou rozhrania FlexBus. V MCU je to implementované ako zápis do pamäťovej oblasti, kde je mapovaný pamäťový priestor FlexBus-u. Druhá prečíta až 3 byty dát a jeden byte príznakov z FPGA. Takéto čítanie po blokoch bolo implementované pre zvýšenie priepustnosti dát pri čítaní z FPGA ako bolo opísané v kapitole 6.2.7.

Po inicializácií FlexBus-u do režimu 32-bitovej adresy a 32-bitových dát sa v úlohe `Ov_Task` začne v nekonečnej slučke vyčítavať obsah bufferu z FPGA. Podľa príznakov sa určí prítomnosť (resp. platnosť) prvého, druhého a tretieho dátového bytu v bufferi a každý byte sa spracuje volaním `ov_consume`, kde sú dáta dekódované.

Oproti použitému riešeniu využívajúcemu tzv. *polling* by bolo možné navrhnúť systém s prerušením, kde FPGA vyvolá prerušenie v prípade, že sú dostupné nové dáta. Taký prístup by jednak obmedzil zbytočnú činnosť CPU v čase, keď dáta nie sú dostupné a zároveň umožnil nové dáta vyčítať z FPGA skôr. V mojej implementácií však ani jedna z chýbajúcich vlastností nie je na závadu, pretože sa predpokladá prítomnosť dostatočne veľkej vyrovnávacej pamäte v FPGA a tiež sa predpokladá, že počas zachytávania USB paketov bude komunikácia medzi FPGA a MCU taká intenzívna, že úspora procesorového času v prípade použitia prerušenia by bola malá. Bolo teda uprednostnené jednoduchšie riešenie s *polling*-om.

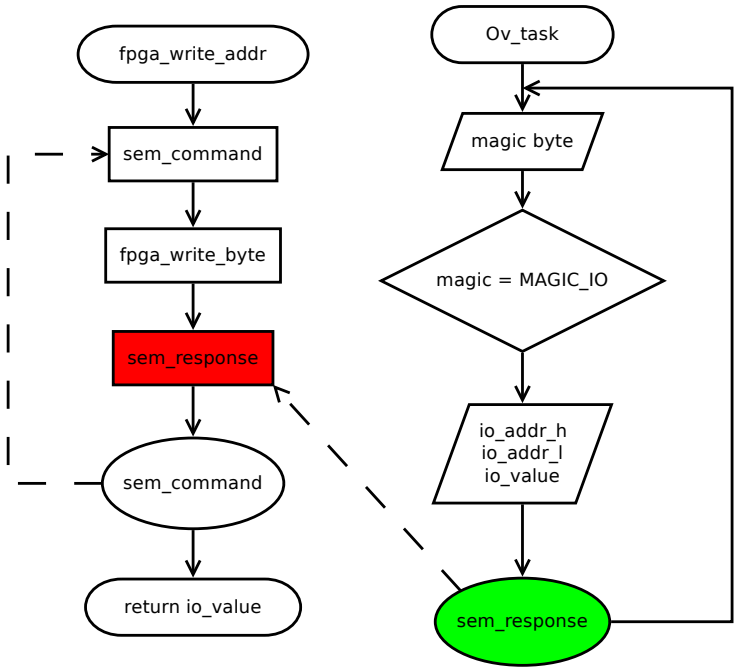
6.4.4 Dekódovanie komunikácie

Každý byte dát prijatý z FPGA je spracovaný funkciou `ov_consume`, v ktorej je implementovaný stavový automat dekódujúci danú komunikáciu. Použitý komunikačný protokol vychádza z protokolu použitého v OpenViszla.

Typ správy je identifikovaný hodnotou prvého bytu správy. Rozlišované sú dva typy správ: správa o hodnote registra a prijatie usb paketu. Správa o zmene hodnoty registra začína hodnotou `0x55` a po nej nasledujú dva byty adresy registra, jeden byte s hodnotou a jeden byte kontrolného súčtu.

Správa obsahujúca USB paket začína hodnotou `0xA0`, po nej nasledujú dva byty príznakov, dva byty označujúce dĺžku dát N, tri byty nesúce časovú značku a N dátových bytov.

S každým prijatým bytom sa automat posunie do ďalšieho stavu. Keď je prijatá kompletná správa, oznámi sa to ostatným úlohám pomocou synchronizačných prostriedkov. Pre jednotlivé typy správ je to popísané nižšie.



Obr. 6.4: Vývojový diagram zobrazujúci synchronizáciu funkcie `fpga_write_addr` s načítavaním dát z FPGA v úlohe `Ov_Task`.

6.4.5 Čítanie a zápis registrov FPGA

Zápis aj čítanie registrov v FPGA funguje podobne. Najskôr je do FPGA prenesená požiadavka na I/O operáciu na zadanej adrese. Potom FPGA odpovie hodnotou, ktorá je (novo-) uložená na danej adrese odoslaním správy ako bolo ukázané vyššie. Aby bolo možné z jednej úlohy vyvolať akciu čítania alebo zápisu a zároveň v nej zistiť výsledok tejto operácie (ktorý MCU spracuje v úplne inej úlohe), bola vytvorená funkcia `fpga_write_addr` a komunikačný kanál s globálnymi premennými a dvojicou semaforov na zabezpečenie synchronizácie medzi komunikujúcimi úlohami. Táto funkcia funguje nasledovne:

1. Zamkni semafor `command_sem`, označujúci kritickú sekciu vykonávania príkazu v FPGA.
2. Odošli správu s danou operáciou do FPGA
3. Zamkni semafor `response_sem`. Tu úloha čaká kým nie je dostupná odpoveď z FPGA.
4. Načítaj hodnotu výsledku z globálnej premennej `io_val`.
5. Odomkni semafor `command_sem`, t. j. opusti kritickú sekciu.
6. Vráť výsledok.

V rovnakom čase beží úloha `Ov_Task` prijímajúca dáta z FPGA. Tie sú dekódované a v momente, keď je prijatá správa o hodnote registra, je táto hodnota uložená do `io_val` a adresa príslušného registra do `io_addr_h` a `io_addr_l`. Vtedy sa v `ov_consume` odomkne semafor `response_sem`, čím sa uvoľní funkcia `fpga_write_addr` z čakania ako na obrázku 6.4.

Čítanie z registra s adresou A je implementované ako zápis ľubovoľnej hodnoty na adresu A. Hodnota je v tomto prípade ignorovaná. Zápis do tohoto registra je implementovaný ako zápis danej hodnoty na adresu A | `0x8000`, kde | je operácia bitového súčtu (*bitwise or*), t. j. najvyšší bit adresy slúži ako *write enable* signál príslušného registra.

6.4.6 Čítanie a zápis registrov ULPI

Okrem konfiguračných registrov FPGA je nutné pristupovať aj do registrov transceiveru fyzickej vrstvy cez rozhranie ULPI. Na to slúžia špeciálne registre v FPGA. Zapísaním adresy registra ULPI do registra `REG_UCFG_RCMD` spolu s príznakom `UCFG_REG_GO` označujúcim zahájenie operácie sa do registra `REG_UCFG_RDATA` prenesie hodnota z daného registra ULPI. Zápis funguje podobne. Do registra FPGA `REG_UCFG_WCMD` sa zapíše adresa a príznak `UCFG_REG_GO`, čím sa do daného registra ULPI prenesie hodnota z dátového registra `REG_UCFG_WDATA`.

Pre prehľadnosť sú tieto operácie implementované vo funkciách `ulpi_read` a `ulpi_write`.

6.4.7 Spracovanie USB paketov

Pakety USB sú prijaté mechanizmom opísaným vyššie. Informácie o spracovávanom pakete sú uložené do dátovej štruktúry `USB_PACKET`. Paket je potom zaradený do fronty `packet_queue`. Operácia vloženia do fronty je blokujúca, t. j. ak je fronta plná, spracovávanie ďalšej komunikácie z FPGA sa zastaví. Pakety sú z fronty vyberané v rutine komunikácie s PC.

Fronta je implementovaná ako `lwmsgq`, teda odľahčená fronta správ z operačného systému MQX. Pamäťový priestor pre túto frontu je alokovaný staticky. Keďže veľkosť USB paketov nie je dopredu známa, priestor pre samotné pakety je alokovaný dynamicky a do fronty sú zaraďované len ukazovatele na ne. Za uvoľnenie dynamicky alokovanej pamäte teda musí byť zodpovedná tá časť programu, ktorá daný ukazovateľ z fronty vyberie.

6.4.8 Komunikácia s PC

Keďže ako užívateľské rozhranie bolo navrhnuté použitie HTML stránky s JavaScript aplikáciou bežiacou v prostredí webového prehliadača, je ako komunikačný protokol použitý protokol HTTP. Ten slúži jednak na prenos samotnej aplikácie ku klientovi (do prehliadača), tak aj na prenos príkazov z PC do MCU a tiež na prenos dát a informácií o stave.

Statické časti (t. j. kód HTML, JavaScript, CSS a ďalšie súčasti užívateľského rozhrania) sú klientovi poskytnuté na definovaných URL pomocou základnej implementácie HTTP servera z knižnice RTCS. Dynamický obsah je generovaný v užívateľom definovaných funkciách označených ako CGI. V prípade, že klient požiada o URL, ktorá smeruje na CGI rozhranie, server zavolá funkciu asociovanú s danou URL. Funkcia potom vygeneruje HTTP odpoveď, ktorá sa odošle klientovi.

CGI rozhranie

Funkcie pre CGI rozhranie majú dátový typ `_mqx_int * (HTTPSRV_CGI_REQ_STRUCT*)` kde jediný parameter je ukazovateľ na štruktúru nesúcu informácie o spracovávanej HTTP požiadavke a návratová hodnota zodpovedá dĺžke dát, ktoré sa majú odoslať klientovi.

Zostavenie odpovede a jej odosielanie sa deje pomocou vytvorenia štruktúry `HTTPSRV_CGI_RES_STRUCT` a jej odoslaní volaním `HTTPSRV_cgi_write(HTTPSRV_CGI_RES_STRUCT *)`.

V tejto štruktúre je potrebné vyplniť polia označujúce aktuálne sedenie `ses_handle`, stavový kód HTTP `status_code` a MIME typ odpovede `content_type` [9].

Ukazovateľ na dáta odpovede (reťazcového typu `char *`) sa uloží do položky `data` a dĺžka týchto dát do `data_length`. Ak nie je dopredu známa dĺžka celej odpovede, tá sa uvedie do položky `content_length` a celá odpoveď sa odošle naraz. V prípade, že nie je možné dopredu určiť dĺžku odpovede, alebo by to bolo neefektívne, je možné odpoveď odosielať po častiach. Do `content_length` sa uvedie 0 a každá časť sa odošle vlastným volaním `HTTPSRV_cgi_write`. Tento typ prenosu využíva režim *chunked* protokolu HTTP definovaný v RFC2616 [6].

Prenos riadiacich príkazov

Príkazy vyvolané užívateľom sú do MCU prenesené pomocou HTTP a sú spracované CGI rozhraním `ov.cgi`. Príkazy používajú HTTP metódu POST. Základným príkazom je zmena režimu činnosti analyzátora. Pre nastavenie režimu LowSpeed, FullSpeed, HighSpeed alebo zastavenie činnosti sa odošle požiadavka typu POST, kde sa ako parameter uvedie požadovaný režim: `MODE_ls`, `MODE_fs`, `MODE_hs` alebo `MODE_stop`. Táto požiadavka je spracovaná v obslužnej funkcií CGI rozhrania a režim je prepnutý pomocou `ov_usb_mode( OV_MODE_...)`. Odpoveď nesie stavový kód HTTP 200 a text `OK`.

Prenos stavových informácií

Stavové informácie z FGPA a MCU sú do PC prenesené na základe HTTP dopytu GET s parametrom `status` na spomínané CGI rozhranie. Odpoveď obsahuje hodnoty stavových registrov a počítadiel paketov.

Prenos dát

Požiadavka na prenos dát z MCU do PC začína PC zaslaním HTTP GET požiadavky na rovnaké rozhranie `ov.cgi`. Pri spracovávaní sa postupne zostavuje a odosiela odpoveď vo formáte JSON ako pole objektov, kde každý objekt reprezentuje jeden USB paket. Odpoveď začína otvorením poľa znakom `[`. USB pakety sú vyberané z fronty `packet_queue`, pre každý paket je vytvorená JSON reprezentácia a tá je odoslaná klientovi. Následne je uvoľnená pamäť, v ktorej bol uložený obsah paketu.

Výber prvku z fronty je v tomto prípade neblokujúci. V prípade, že je fronta prázdna, cyklus sa ukončí a odošle sa znak `]` na uzatvorenie JSON poľa. Takto je vždy vytvorená validná JSON odpoveď, ako je napríklad uvedené vo výpise 6.4.

Pri ladení bolo zistené, že v niektorých prípadoch bola fronta plnená tak rýchlo, že ju vo funkcií obsluhy prenosu do PC nebolo možné úplne vyprázdniť, alebo sa vyprázdnila až po dlhej dobe. To viedlo k časovým oneskoreniam pri zostavovaní odpovede a užívateľské rozhranie reagovalo s dlhou odozvou. Preto bola stanovený maximálny počet USB paketov odoslaných v jednej odpovedi na 500 paketov.

6.5 Užívateľské rozhranie

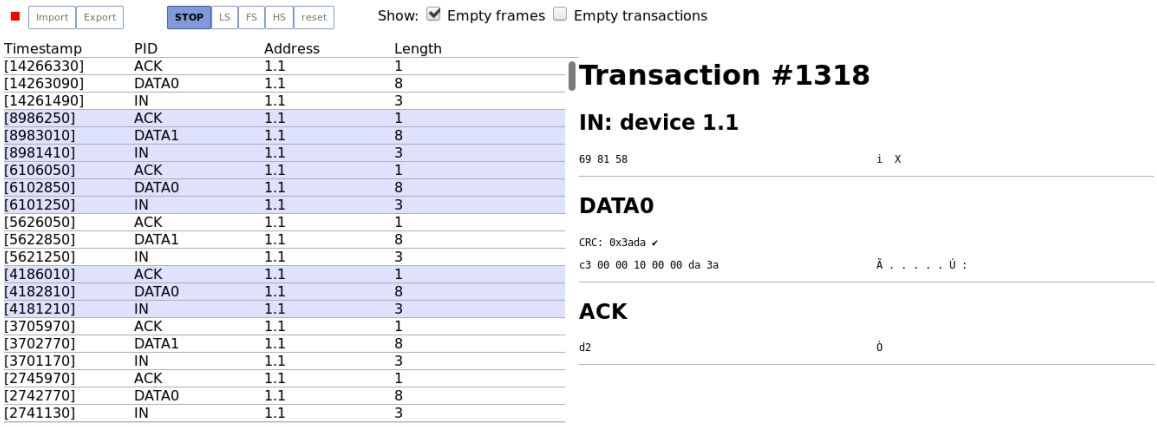
Užívateľské rozhranie bolo vytvorené pomocou značkovacieho jazyka HTML, jazyka kaskádových štýlov CSS a programovacieho jazyka JavaScript.

```
[
  {"pid": 105, "len": 3, "timestamp": 5405628, "flags": 0, "free": 59456,
  "data":[105,132,152]},
  {"pid": 90, "len": 1, "timestamp": 5407228, "flags": 0, "free": 59488,
  "data":[90]},
  {"pid": 105, "len": 3, "timestamp": 5885668, "flags": 0, "free": 59520,
  "data":[105,132,152]}
]
```


Výpis 6.4: Príklad JSON štruktúry so zachytenými paketmi. (Zlomy riadkov doplnené pre čitateľnosť, pozn. autora.)

6.5.1 Časti rozhrania

Štruktúra rozhrania bola navrhnutá v časti 5.5. V tejto časti sú rozobrané implementačné detaily jednotlivých častí užívateľského rozhrania. Konkrétna podoba užívateľského rozhrania je viditeľná na obr. 6.5.



Obr. 6.5: Snímka obrazovky užívateľského rozhrania analyzátoru.

Panel nástrojov

Na paneli nástrojov sa nachádzajú stavové informácie, tlačidlá na prepínanie režimu analyzátora a ovládanie filtrov. Podfarbením tlačidla je indikovaný aktívny režim. Tlačidlá sú vytvorené HTML značkou `<button>`. Spôsob prepojenia akcie tlačidla s činnosťou analyzátora je podrobnejšie opísaný v časti 6.5.2.

Farebný terčík zobrazuje stav analyzátora. Červenou farbou je indikovaná chyba v prípade, že sa nepodarí nadviazať spojenie s analyzátorom, alebo spojenie bolo prerušené. Oranžovou farbou je zobrazený stav, keď analyzátor komunikuje s užívateľským rozhraním, ale neodosiela žiadne pakety (stav STOP alebo nepripojené USB zariadenie). Zelenou farbou je označená normálna prevádzka, kedy analyzátor odosiela zachytené pakety a tie sú zobrazené v užívateľskom rozhraní.

Zaškrtávacie políčka umožňujú nastavenie filtrov paketov. Prvé z nich, „Show empty frames“ umožňuje zobraziť alebo skryť pakety `SOF`, ktoré označujú začiatok rámca, ak v tomto

rámci neboli prenesené žiadne ďalšie pakety. Zabraňuje sa tým zahlteniu užívateľa veľkým množstvom bezvýznamných dát.

Druhé, „Show transactions without data“ umožňuje vyfiltrovať transakcie typu IN, v ktorých adresované zariadenie neodoslalo žiadne dáta (ukončilo transakciu pomocou `NAK`). Tento filter je vhodný na sledovanie prenosov typu *interrupt*, pri ktorých je zariadenie často vyzývané hostiteľom, ale prenos dát sa uskutočňuje len zriedkavo.

Tabuľka zachytených paketov

Tabuľka zobrazuje zachytené pakety a základné informácie o nich. Pakety z jednej transakcie sú vizuálne zoskupené rovnakým podfarbením riadkov tabuľky. To je dosiahnuté priradením CSS štýlov `transaction-odd` a `transaction-even` pre riadky s paketmi patriacimi do transakcií s párnym resp. nepárnym poradovým číslom. Tabuľka umožňuje scrollovanie nezávisle na zvyšku obrazovky. Kliknutím na riadok s paketom sa transakcia, do ktorej daný paket patrí, zobrazí v inšpektore transakcií.

Inšpektor transakcií

Inšpektor transakcií zobrazuje na jednom mieste celú transakciu, napríklad postupnosť paketov `IN - DATA0 - ACK`. Pri každom pakete je zobrazený typ paketu (PID), zariadenie, ktoré daný paket odoslalo (vo formáte device.endpoint) a dátová časť paketu. Tá je zobrazená v hexadecimálnom aj textovom tvare. Užívateľ tak môže ľahšie identifikovať hodnoty prenášané v paketoch.

6.5.2 Programová obsluha užívateľského rozhrania

Užívateľské rozhranie zobrazí webový prehliadač na základe HTML popisu a CSS štýlov. Aby rozhranie mohlo byť interaktívne, je obsluhované programom v jazyku JavaScript, ktorý so zobrazenou podobou rozhrania komunikuje pomocou modelu `DOM`. Takto je možné pridávať a odoberať prvky (ako napr. riadky tabuľky), meniť ich vlastnosti (farbu tlačidla) alebo ich obsah (text) a reagovať na udalosti (kliknutie myši, stlačenie tlačidla).

Na získanie referencie na daný objekt v strome modelu DOM slúžia funkcie ako `getElementById`, `getElementsByClassName`, `getElementsByTagName` a podobne [13]. Funkcia `getElementById` pracuje na globálnej úrovni celého dokumentu, čo porušuje princíp zapuzdrenia a modularity. Jedným riešením by bolo prvky vytvárať po jednom programovo a referencie na ne ukladať v momente ich vytvorenia. To však zvyšuje komplexnosť celého riešenia. Preto bol zvolený iný postup.

V prípade, že je požadovaná referencia na nejaký prvok, je tento prvok označený triedou `bind` a jednoznačný identifikátor je uložený do atribútu `data-ref`. V HTML zápise to vyzerá napríklad takto: `<div class="bind" data-ref="myDiv">`. Potom vo funkcii `init` sa vyhľadajú takéto elementy pomocou `getElemetsByClassName` a na každý element sa vytvorí príslušná referencia. Na element z príkladu by bolo možné odkazovať pomocou `elements.myDiv`.

Spracovanie udalostí

Funkcie obsluhy udalostí sú k elementom registrované volaním `element.addEventListener(event, handler)`. Parameter `event` označuje typ udalosti, napríklad `"click"`, `handler` je

funkcia, ktorá sa vykoná, keď udalosť nastane. Využitím uzáverov (*closure*) je možné vytvoriť generátor obslužných funkcií, ako je napríklad `setModeHandler`, ktorý vracia obslužnú funkciu pre prepnutie režimu analyzátoru podľa zadaného parametra.

Podobne sú spracované aj udalosti kliknutia na paket v tabuľke paketov. Tu je obslužná funkcia priradená k udalosti kliknutia v kóde generujúcom riadky tabuľky pre nové pakety. Kliknutie na paket zavolá funkciu `showTransaction(packet)`, ktorá zobrazí zodpovedajúcu transakciu v okne inšpektora transakcií.

Spracovanie a zobrazovanie USB paketov

Pakety sú prijaté mechanizmom HTTP GET dopytov podrobnejšie opísaných v časti 6.4.8. Prijaté pole objektov v JSON reprezentácií je prevedené do JavaScript poľa Array a objektov Object volaním parsera `JSON.parse(data)`. Parser JSON je štandardnou súčasťou webových prehliadačov [5].

Nad každým paketom je zavolaná funkcia `pushPacket`, ktorá zabezpečí parsovanie paketu a jeho zobrazenie v užívateľskom rozhraní. Najskôr sa paket parsuje vo funkcii `parsePacket`, podľa PID a hodnôt v dátovej časti paketov sa určia jeho vlastnosti, skontroluje sa CRC kontrolný súčet, dekódujú sa príznaky a z týchto informácií sa vytvorí zrozumiteľnejšia reprezentácia paketu. Tá je potom uložená do poľa obsahujúceho všetky pakety. Z paketu je potom vytvorená jeho DOM reprezentácia pomocou funkcie `renderPacket` a tá je vložená na začiatok tabuľky paketov. Tým je paket zobrazený.

Inšpekcia transakcií

Kliknutím na paket sa zobrazí celá transakcia, do ktorej paket patrí. Preto sa pri parsovaní paketov zostavuje pole transakcií a pre každú transakciu pole paketov v nej. Nová transakcia sa rozpozná prijatím paketu typu `IN`, `OUT`, `SETUP` alebo `PING`. Transakcia potom zvyčajne obsahuje dátový paket `DATA0`, `DATA1` alebo `DATA2` a je ukončená potvrdením `ACK` alebo odmietnutím `NAK`, prípadne výskytom chyby. Transakcia sa zobrazí vytvorením DOM reprezentácie pre hlavičku transakcie a jej vložením do elementu `packetInspector`. Za ňu sa pre každý paket z transakcie vloží jeho DOM reprezentácia vygenerovaná pomocou `renderPacketInspector`.

Hexadecimálna a textová reprezentácia dátovej časti paketu sa vytvorí vo funkcii `hexdump`. Tu sa dáta paketu rozdelia na 16 bytov dlhé bloky a každý byte z bloku sa prevedie do hexadecimálnej formy. Zároveň sa byte prevedie na znak z ASCII tabuľky. V prípade, že daný byte nemá znakovú reprezentáciu (riadiace znaky ASCII) nahradí sa znakom bodka „.“. Tieto dve reprezentácie sú potom pre každý blok dát konkatenované, vytvárajúce dva stĺpce. Takéto zobrazenie je známe z HEX editorov.

6.5.3 Obsluha komunikácie s analyzátorom

V kapitole 6.4.8 bola popísaná obsluha komunikácie zo strany MCU. Po otvorení adresy analyzátora vo webovom prehliadači sa stiahne HTML dokument s kostrou užívateľského rozhrania. Ten obsahuje odkazy na kaskádové štýly CSS (pomocou značky `<link>`) a na program (skript) v jazyku JavaScript, ktorý obsluhuje užívateľské rozhranie (pomocou značky `<script>`).

Po stiahnutí všetkých súčastí webový prehliadač začne vykonávať daný skript. Trieda `Comm` obsahuje metódy na komunikáciu s analyzátorom. Tie využívajú `XMLHttpRequest` API webového prehliadača na vytvorenie a odoslanie HTTP dopytov na server v analyzátore.

Dopyt sa vytvorí v metóde `request`, prípadne `get` alebo `post`. Keďže HTTP komunikácia je asynchrónna voči vykonávaniu skriptu, výsledok dopytu nemôže byť vrátený ako návratová hodnota pomocou `return`, ale v prípade dokončenia komunikácie sa zavolá funkcia (*handler*) predaná ako parameter `callback` - podobne ako pri obsluhe udalostí.

Trieda Comm ďalej implementuje metódy vyššej úrovne: `setMode` slúži na prepnutie režimu analyzátora a `reset` na reštartovanie mikrokontroléru. Funkcia `getPackets` načíta pakety z analyzátora a vráti ich (mechanizmom spomínaným vyššie) v podobe JS objektu.

Podobne ako výsledok sú predávané prípadné chyby komunikácie - zavolá sa funkcia predaná ako parameter `errCallback`.

6.5.4 Import a export

Aby bolo možné namerané údaje uchovať a neskôr znova preskúmať, prípadne použiť na ich analýzu ďalšie nástroje, užívateľské rozhranie obsahuje funkcie importu a exportu. Pri exporte je zoznam všetkých paketov serializovaný do formátu JSON a tento výstup je predaný prehliadaču ako súbor na stiahnutie. Takto môže užívateľ uchovať namerané hodnoty.

Pre ich opätovné načítanie slúži funkcia importu. Tá je realizovaná ako *drag&drop*, t. j. pre import údajov užívateľ myšou pretiahne súbor s údajmi do okna prehliadača. Tú istú akciu možno vykonať aj kliknutím na tlačidlo importu. V tom prípade užívateľ vyberie požadovaný súbor v systémovom dialógu. Importované údaje nahradia aktuálny obsah tabuľky paketov.

Načítanie súboru je realizované pomocou `File` API prehliadača. Vstup z dialógu alebo z udalosti *drag&drop* je spracovaný vo funkcií `handleImport`, kde je obsah súboru načítaný pomocou `FileReader`. Obsah je prevedený z formátu JSON na JS objekty pomocou `JSON.parse` a jednotlivé pakety sú spracované podobne, ako pri ich príjme z analyzátoru.

6.5.5 Zásuvné moduly

Aby bolo možné funkcie analyzátora ďalej rozširovať, bol vytvorený systém zásuvných modulov. Zásuvný modul je v tomto prípade program v jazyku JavaScript, ktorý je možné do užívateľského rozhrania pridať za behu. Načítanie modulu je realizované ako dynamické pridanie značky `<script>` so zdrojovým kódom modulu do HTML dokumentu. O vykonanie programu, ktorý nový modul obsahuje, sa postará webový prehliadač.

Aby mohol byť nový modul zaregistrovaný a aby jeho činnosť neovplyvnila ostatné moduly, celé chovanie modulu musí byť uzavreté do jednej funkcie. Túto funkciu modul predá ako argument pri volaní `registerModule`. Kostra modulu je zachytená na výpise 6.5.

```
registerModule(function(app){
  //vlastné telo modulu
  app.$app // hlavné okno aplikácie
  app.elements // prvky aplikácie
});
```

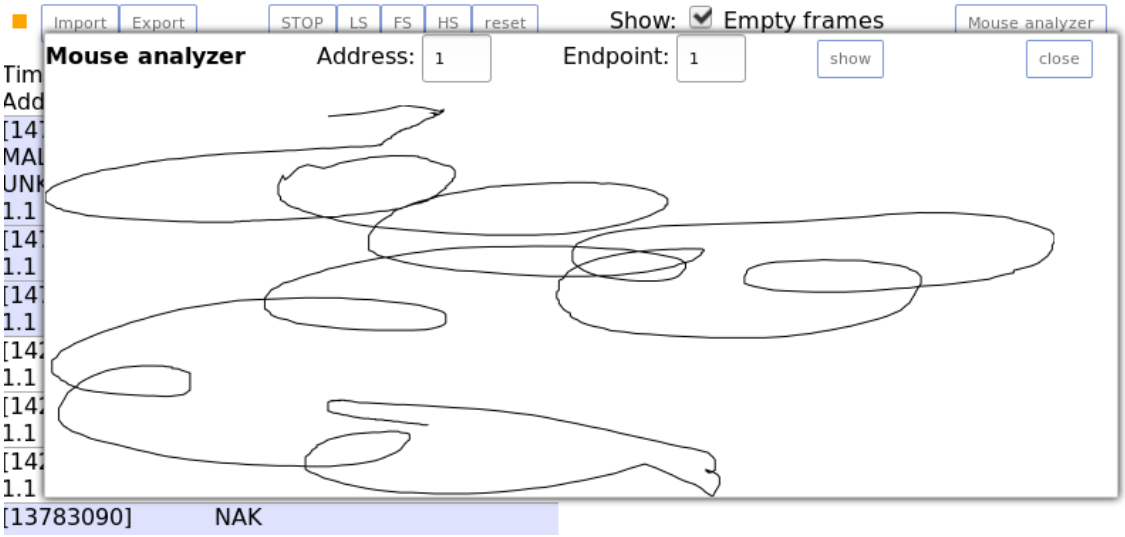
Výpis 6.5: Štruktúra zásuvného modulu

Moduly môže užívateľ načítať pomocou *drag&drop* rovnako ako pri importe dát.

Interpretácia HID polohovacieho zariadenia

Ako príklad zásuvného modulu bol vytvorený modul na interpretovanie dát zachytených od polohovacieho zariadenia - myši. Formát dátových paketov je podrobnejšie opísaný v časti 7.1.1 kapitoly Výsledky.

Modul pridá do panelu nástrojov tlačidlo na spustenie analýzy pohybu myši. Po jeho stlačení sa vytvorí nové dialógové okno (`<div class="window">`), kde užívateľ zadá adresu a číslo *endpoint*-u zariadenia. Po spustení analýzy tlačidlom `Show` modul prejde zachytené transakcie, z nich vyberie tie, ktoré zodpovedajú zadanému zariadeniu a nesú údaje o zmene polohy. Pohyb je vykreslený graficky do kresliaceho plátna `<canvas>`, ako je ukázané na obrázku 6.6.



Obr. 6.6: Snímka obrazovky s modulom interpretácie pohybu polohovacieho zariadenia.

Kapitola 7

Výsledky

V rámci diplomovej práce bol realizovaný kompletný a funkčný analyzátor USB zbernice. Jeho vlastnosti, obmedzenia a porovnania s inými riešeniami sú zhrnuté v tejto kapitole.

Projekt OpenViszla zvolený ako základ analyzátoru bol úspešne prenesený na cieľovú platformu Minerva kit. Prenos si vyžadoval vytvorenie vlastného návrhu obvodu a dosky plošných spojov pre obvod s USB transceiverom. Obvod bol navrhnutý a zostavený bez väčších problémov s použitím katalógových zapojení použitých obvodov a odporúčaní výrobcu. Oživenie obvodu nepredstavovalo žiaden problém, obvod fungoval na prvé zapojenie.

Mechanické prevedenie dosky s týmto obvodom sa osvedčilo. Doska s obvodom sedí na hrebeňovom konektore dostatočne pevne, pripojené vodiče USB svojou váhou dosku nedeformujú ani nevytvárajú príliš veľký tlak na konektor. Odsadenie konektorov USB od seba a od kraja dosky je dostatočné, poskytuje dostatočný manipulačný priestor pre pripájanie USB káblov alebo menších periférií (USB kľúč a pod.).

Oproti pôvodnému plánu bol na DPS pridaný druhý hrebeňový konektor zo strany súčiastok. To umožní využiť nepoužité piny FPGA pre ďalšie rozširujúce obvody.

Celkový návrh architektúry analyzátora spĺňa požiadavky, ktoré boli stanovené v úvode práce. Činnosť je rozdelená medzi FPGA a MCU, kde FPGA pracuje na nižšej úrovni jednotlivých bytov prenášaných z USB, MCU zostavuje informácie o paketoch a rozbor transakcií zložených s viacerých paketov je ponechaný na aplikáciu s užívateľským rozhraním na PC. Z toho vyplývajú obmedzenia na možnosti predspracovania nameraných údajov. Napríklad filtrovanie na úrovni transakcií by bolo v FPGA ťažko implementovateľné, pretože v jednom momente je v FPGA spracovávaný len jeden paket, alebo dokonca len jeho časť.

Komunikačné rozhranie medzi FPGA a MCU je dostatočne flexibilné. Použitý protokol a registrová architektúra v FPGA umožňuje jednoduché pridanie ďalších príkazov alebo registrov pre ovládanie funkcie FPGA.

Použitý operačný systém MQX výrazne zrýchlil vývoj a umožnil pohodlné ladenie programu v MCU. Z hľadiska softwarovej architektúry je program v MCU rozdelený na samostatné funkčné celky v podobe úloh. Úlohy medzi sebou komunikujú vysokoúrovňovými prostriedkami v podobe front správ a semaforov. Istou nevýhodou môže byť použitie globálnych premenných ako odkazov na tieto komunikačné prostriedky. Znižuje to znovupoužiteľnosť porušením princípov zapuzdrenia.

Pre jednoúčelový program z oblasti vstavaného hardwaru to však nehodnotím ako závažný nedostatok návrhu SW. Možným riešením by bolo vytvorenie jednej úlohy, slúžiacej na inicializáciu týchto prostriedkov, ktorá by dané prostriedky vytvorila lokálne a nie globálne a následne spustila zvyšné úlohy, ktorým by referencie na tieto prostriedky predala

parametrom. Prostriedky na dynamické vytváranie úloh sú obsiahnuté v operačnom systéme MQX, ich použitie je však zložitejšie ako statická definícia úlohy s automatickým spustením.

Vytvorené užívateľské rozhranie je rýchle a prehľadné. Všetky funkcie sú dostupné z hlavnej obrazovky. Ich pomenovanie by malo byť užívateľovi orientujúcom sa v problematike zbernice USB zrozumiteľné. Import a export do formátu JSON poskytuje možnosť interoperability s inými nástrojmi. Možnosť rozšírenia pomocou užívateľských zásuvných modulov je prostriedkom pre pridanie chýbajúcej funkcionality.

Použitý komunikačný protokol s HTTP GET dopytmi pre získanie nameraných údajov sa javí ako dostatočné, má však zbytočne veľkú réžiu. Zlepšením by mohlo byť použitie protokolu WebSockets, ktorý umožňuje vytvorenie perzistentného kanálu nad protokolom HTTP.

Z pôvodného plánu sa nepodarilo implementovať použitie pamäte DRAM prítomnej na doske Minerva. Radič pamäte DDR2 vygenerovaný nástrojmi Xilinx ISE sa nepodarilo sprevádzkovať s danou pamäťou. Ako náhradné riešenie bolo zvolené použitie pamäte typu SRAM, ktorej obsluha je značne jednoduchšia. Táto pamäť sa pripojila k rozširujúcemu konektoru dosky s obvodom fyzickej vrstvy. Bol vytvorený radič pre obsluhu tejto pamäte, v konečnom riešení však nebola použitá.

Optimalizáciou prenosu medzi FPGA a MCU sa podarilo dosiahnuť dostatočnú priepustnosť, dovoľujúcu analýzu USB aj na úrovni rýchlosti HighSpeed.

V prípade plného vyťaženia zbernice USB (napríklad čítaním z rýchleho disku USB, pripojením viacerých zariadení cez USB hub a pod.) dochádza k preplneniu vyrovnávacích pamätí v FPGA. Tieto stavy sú však korektne ošetrené a užívateľ je na to upozornený príznakom OFL (*overflow*).

7.1 Vykonané analýzy

Na overenie funkčnosti analyzátora bola jeho činnosť otestovaná s rôznymi zariadeniami. Ako USB host bol použitý počítač autora. Výber zariadení bol podmienený otestovaním všetkých troch rýchlostných úrovní zbernice USB 2.0.

7.1.1 Polohovacie zariadenie - myš

USB myš Trust 17026 sa systému prezentuje ako zariadenie HID (*human interface device*). Zariadenie pracuje na prenosovej rýchlosti Low Speed. V prvej fáze komunikácie je zariadenie nakonfigurované riadiacimi prenosmi SETUP a komunikáciou s endpoint-om 0. Po jeho nakonfigurovaní systém pravidelne vyzýva zariadenie na prenos dát tokenom IN (jedná sa o prenos typu interrupt).

V čase, keď je myš bez pohybu, každá takáto transakcia je ukončená bez odoslania dát paketom NAK. V prípade, že užívateľ myšou pohne po podložke (prípadne stlačí tlačidlo myši), v najbližšej transakcií sú odoslané údaje o tejto udalosti.

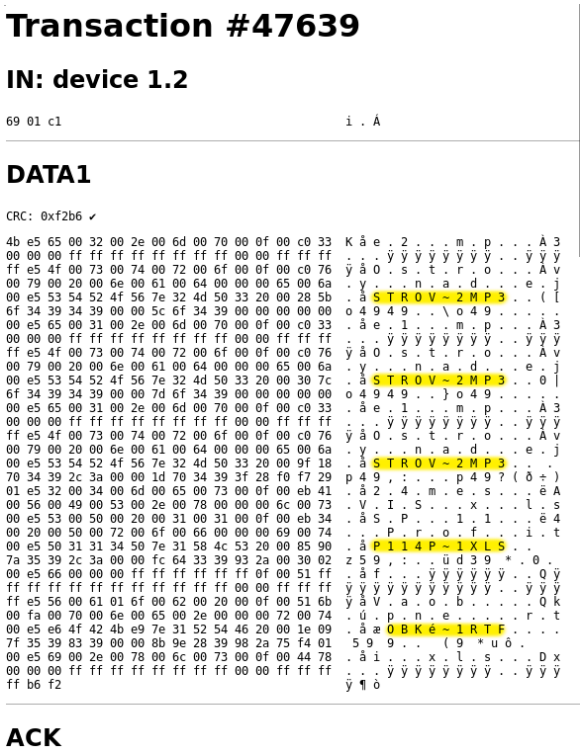
Informácia o pohybe je relatívna - uvedený je rozdiel oproti predchádzajúcej polohe. Stlačené tlačidlá sú indikované bitovým poľom, kde každý bit zodpovedá stavu jedného tlačidla.

Ďalej bola otestovaná funkcia na rýchlostnej úrovni Full Speed. Ako USB zariadenie bola použitá bezdrôtová myš Logitech M235. Štruktúra zachytených údajov je podobná ako v predchádzajúcom prípade.

7.1.2 USB kľúč

Ako príklad zariadenia pracujúceho na rýchlosti High Speed bol použitý USB kľúč PQI Travelling Disk U172P. Zariadenie sa prezentuje ako Mass Storage a použitý súborový systém bol FAT16.

Po pripojení zariadenia je možné opäť pozorovať transakcie typu SETUP a následne komunikáciu s dvoma endpoint-mi: 1 a 2. Na druhom endpoint-e je možné pozorovať prenosy typu BULK. Na obrázku 7.1 je zachytená transakcia typu IN, kde sú z USB kľúča čítané dáta. Viditeľná je časť FAT tabuľky a vyznačené boli identifikovateľné názvy súborov.



Obr. 7.1: Transakcia čítania dát z USB kľúča.

Ďalej je možné pozorovať sekvencie čakania na dokončenie zápisu do zariadenia: ak hostiteľ odošle dátový paket pomocou OUT, zariadenie ho spracuje, ale ďalší paket by nebolo schopné prijať a spracovať, odpovedá teda pomocou paketu NYET namiesto ACK. Hostiteľ potom vyčkáva na zariadenie a periodicky sa dotazuje na pripravenosť prijať ďalší dátový paket pomocou PING. Zariadenie odpovedá ACK, ak je pripravené, inak odpovedá NACK.

7.2 Porovnanie s inými nástrojmi

Oproti projektu OpenViszla, z ktorého práca vychádza, prináša táto práca viaceré vylepšenia. Časť spracovania nameraných údajov sa presunula z obslužnej aplikácie do MCU, čo zjednodušuje samotnú obslužnú aplikáciu. Použité rozhranie Ethernet umožňuje použitie prostredníctvom počítačovej siete. Analyzátor je ovládaný z pohodlnejšieho grafického užívateľského rozhrania namiesto príkazového riadku. Architektúra klient-server, kde klientom je bežný internetový prehliadač, umožňuje použitie analyzátoru bez potreby inštalácie

ďalšieho softwaru.

V prostredí FIT VUT je tento analyzátor dostupnejší pre študentov ako iné HW analyzátory vďaka možnosti zapožičania vývojovej dosky FitKit, na ktorej bol analyzátor implementovaný.

Oproti SW analyzátorom ako Wireshark má tento analyzátor obecnú výhodu HW analyzátorov: skúmaná časť zbernice nemusí byť vôbec pripojená k užívateľskému PC, je tak možné študovať komunikáciu medzi dvoma inými, nezávislými zariadeniami (napr. prehrávač a kľúč USB).

Užívateľské rozhranie je však omnoho jednoduchšie a neposkytuje toľko funkcií ako analyzátory predstavené v kapitole 3. Rozšírenie funkcií analyzátora a užívateľského rozhrania je tak námetom na ďalšiu prácu.

Kapitola 8

Záver

V tejto práci bol čitateľ zoznámený so zbernicou USB a nástrojmi na analýzu komunikácie na nej. Spomedzi týchto nástrojov bol zvolený projekt OpenViszla ako referenčná implementácia hardwarového USB analyzátora s cieľom preniesť túto implementáciu na vývojovú platformu Minerva.

Po preskúmaní existujúcich riešení bola navrhnutá architektúra USB analyzátora, ktorá využíva USB transceiver USB3343, hradlové pole Xilinx Spartan 6, mikrokontrolér Freescale Kinetis a Ethernet radič. Pre pripojenie transceiveru k hradlovému poľu sa navrhla schéma zapojenia a doska plošných spojov. Následne bolo navrhnuté použitie siete Ethernet ako komunikačného rozhrania medzi analyzátorom a počítačom, ako aj využitie technológií HTML, CSS a JavaScript na vytvorenie užívateľského rozhrania dostupného cez webový prehliadač. Použitie zariadenia tak nevyžaduje inštaláciu žiadneho dodatočného softwaru.

Existujúci projekt OpenViszla bol prenesený na FPGA kitu Minerva úpravou existujúcich jednotiek a doprogramovaním vlastných, najmä tých, ktoré sa zaoberajú komunikáciou s ďalšími obvodmi. Pre MCU sa implementovala obsluha hardwarovej časti analyzátoru nachádzajúcej sa v FPGA. S použitím prostriedkov operačného systému Freescale MQX RTOS sa podarilo vytvoriť komunikačné rozhranie na báze HTTP serveru pre odosielanie nameraných údajov do PC.

Pre PC bolo vytvorené užívateľské rozhranie vo forme HTML/JavaScript aplikácie, ktorá sa zobrazuje vo webovom prehliadači a načítava sa priamo z HTTP serveru USB analyzátora.

Testovaním implementácie pri analýze komunikácie osobného počítača s rôznymi USB zariadeniami sa overila funkčnosť aj možnosti analyzátora. Analyzátor umožňuje základné zachytávanie USB paketov a ich študovanie v súvislosti s inými paketmi tvoriacimi transakcie. Dostupné sú informácie o časovej postupnosti a obsahu paketov. Hlbšia inšpekcia paketov bola demonštrovaná na jednom príklade a to na zobrazení pohybu polohovacieho zariadenia (myši).

Analyzátor je limitovaný v šírke pásma, ktorú dokáže spracovať. Zvýšenie tejto úrovne by bolo možné vytvorením vhodného radiča pre rýchlu pamäť DRAM prítomnú na doske kitu Minerva a jeho použitím ako buffer pre zachytené pakety. Ďalšie vylepšenia by boli možné vytvorením vhodných modulov pre inšpekciu obsahu paketov.

Prínos tejto práce je v rozšírení možností platformy Minerva, ako výukovej platformy na fakulte FIT VUT, o demonštráciu činnosti zbernice USB a sprístupnenie nástrojov na analýzu komunikácie USB zariadení napr. pri ich ladení.

Literatúra

[1] *UTMI+ Low Pin Interface (ULPI) Specification*. October 20, 2004, rev. 1.1.

[2] Axelson, J.: *USB Complete: Everything You Need to Develop Custom USB Peripherals*. Complete Guides series, Lakeview Research, 2005, ISBN 9781931448024.

[3] Buchta, P.: *Framework pro vývoj aplikací na platformě ARM*. Diplomová práce, FIT VUT v Brně, 2015.

[4] Crescent Heart Software: *USB-2XT – USB PROBING SUPPORT PRODUCT*. 2008 [cit. 2016-05-18].
URL http://www.c-h-s.com/USB-2XT.DataSheet.pdf

[5] Deveria, A.: *Can I use... Support tables for HTML5, CSS3, etc*. 2016 [cit. 2016-06-05].
URL http://caniuse.com

[6] Fielding, R.; Gettys, J.; Mogul, J.; aj.: Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, RFC Editor, June 1999.
URL http://www.rfc-editor.org/info/rfc2616

[7] Freescale Semiconductor, Inc.: *K60 Sub-Family Data Sheet*. 2013, document number: K60P144M100SF2.

[8] Freescale Semiconductor, Inc.: *Freescale MQX^{TM} RTOS BSP Porting Guide*. 2014, rev. 0.

[9] Freescale Semiconductor, Inc.: *Freescale MQX^{TM} RTOS RTCS User's Guide (IPv4 and IPv6)*. 2015, rev. 2.

[10] Freescale Semiconductor, Inc.: *Freescale MQX^{TM} RTOS User's Guide*. 2015, rev. 14.

[11] ISO/IEC 7498-1: *Information Technology - Open Systems Interconnection - Basic Reference Model: The Basic Model*. 1996.

[12] Kumar, R. R.: *Human Computer Interaction*. Laxmi Publications, 2005, ISBN 9788170087953.

[13] Mozilla Developer Network: *Document - Web APIs*. 2016.
URL https://developer.mozilla.org/en-US/docs/Web/API/Document

[14] Musich, C.; Lozano, A.: *Using FlexBus Interface for Kinetis Microcontrollers*. Freescale Semiconductor, 2012, document number: AN4393.

[15] pizthewiz: *example: HS - openvizslaov_ftdi Wiki - GitHub*. 2015-01-02.
URL https://github.com/openvizsla/ov_ftdi/wiki/example%3A-HS

[16] Semiconductor Components Industries, LLC: *NB3N551 3.3 V / 5.0 V Ultra-Low Skew 1:4 Clock Fanout Buffer*. 2012, rev. 4.

[17] Standard Microsystems Corporation: *USB334x Enhanced Single Supply Hi-Speed USB ULPI Transceiver*. 2013-08-02.

[18] Straka, M.: *MINERVA KIT | Výuková a experimentální platforma s FPGA a MCU*. Fakulta informačních technologií, VUT v Brně, 2016 [cit. 2016-01-02].

[19] USB Implementers Forum, Inc: *USB 2.0 Specification Engineering Change Notice (ECN) #1: Mini-B connector*. 2000-10-20.

[20] USB Implementers Forum, Inc: *Universal Serial Bus Specification Revision 1.1*. 2000-4-27.

[21] USB Implementers Forum, Inc: *Universal Serial Bus Specification Revision 2.0*. 2000-4-27.

[22] USB Implementers Forum, Inc: *MicroUSB Specification to the USB 2.0 Specification, Revision 1.01*. 2007-4-4.

[23] USB Implementers Forum, Inc: *Universal Serial Bus Specification Revision 3.0*. 2008-11-12.

[24] usblyzer.com: USBlyzer - USB Protocol Analyzer and USB Traffic Sniffer. 2014 [cit. 2015-12-21].
URL http://www.usblyzer.com/

[25] Wireshark Foundation: *Wireshark - About Wireshark*. 2015 [cit. 2016-01-01].
URL https://www.wireshark.org/

[26] Zwoliński, M.: *Digital system design with VHDL*. Prentice Hall, 2004.

[27] Šimek, V.: *Schéma obvodového zapojení výukového kitu Minerva*. Fakulta informačních technologií VUT Brno, 2013.

Zoznam skratiek

- ACK - *acknowledgement* - potvrdenie
- CLI - *command line interface* - rozhranie s príkazovým riadkom
- CLK - *clock* - hodinový signál
- CSS - *cascading style sheets* - kaskádové štýly
- DPS - doska plošných spojov
- EOP - *end of packet* - koniec paketu
- FIFO - *first in, first out* - princíp „prvý dnu, prvý von“
- FPGA - *field-programmable gate array* - hradlové pole
- GND - *ground* - zem, uzemnenie
- GUI - *graphic user interface* - grafické užívateľské rozhranie
- HTML - *hypertext markup language* - hypertextový značkovací jazyk
- JS - JavaScript
- LAN - *local area network* - lokálna sieť
- MCU - mikrokontrolér
- PC - *personal computer* - osobný počítač
- PID - *packet identifier* - identifikátor paketu
- PLL - *phase locked loop* - sľučka s fázovým závesom
- RAM - *random access memory* - pamäť s náhodným prístupom
- SMD - *surface mount devices* - technológia povrchovej montáže súčiastok na DPS
- SOF - *start of frame* - začiatok rámca
- SOP - *start of packet* - začiatok paketu
- USB - *universal serial bus* - univerzálna sériová zbernica

Prílohy

Zoznam príloh

Príloha A

Obsah CD

Priložený nosič CD obsahuje nasledovnú adresárovú štruktúru:

- /doc – elektronická verzia technickej správy
- /fpga – program pre FPGA
  - /bin – binárna konfigurácia FPGA
  - /ise – projekt Xilinx ISE a zdrojové kódy VHDL
  - /ov – zdrojové kódy OpenViszla
- /mcu – program pre MCU
  - /bin – binárny súbor s programom pre MCU
  - /gui – zdrojový kód užívateľského rozhrania
  - /kds – projekt Kinetis Design Studio a zdrojové kódy pre MCU
  - Freescale_MQX_4_2_FITKIT_KDS300.zip – systém Freescale MQX pre kit Minerva
- /pcb – projekt dosky plošných spojov pre KiCad
- /tex – zdrojové texty technickej správy vo formáte LaTeX
- README.txt – textový súbor popisujúci obsah disku CD

Príloha B

Mapa registrov FPGA

názov	adresa	bit	význam
CSTREAM_CFG	0x800		riadiaci register streamovacej jednotky
		0	aktivácia streamu
		1:7	N/A
OVF_INSERT_CTL	0x1000		riadiaci register jednotky detekcie pretečenia
		0	reset počítadiel
		1:7	N/A
OVF_INSERT_NUM_OVF	0x1001:0x1004		počítadlo pretečení
OVF_INSERT_NUM_TOTAL	0x1005:0x1008		počítadlo paketov
UCFG_RST	0x400		ovládanie resetu ULPI
		0	reset obvodu USB3343
		1	reset ULPI radiča
		2	zneplatnenie obsahu zbernice pri resete
		3:7	N/A
UCFG_STAT	0x401		stavový register ULPI radiča
		0	hodinový signál prítomný
		1:7	N/A
UCFG_WDATA	0x402		dátový register zápisu do ULPI
UCFG_WCMD	0x403		riadiaci register zápisu do ULPI
		0:5	adresa registra
		6	N/A
		7	príznak vykonávaného zápisu
UCFG_RDATA	0x404		dátový register čítania z ULPI
UCFG_RCMD	0x405		riadiaci register čítania z ULPI
		0:5	adresa registra
		6	N/A
		7	príznak vykonávaného čítania

Príloha C

Návod na zostavenie a oživenie analyzátoru

C.1 Preklad OpenViszla nástrojom Migen

Pred spustením samotného prekladu je nutné mať nainštalované behové prostredie jazyka Python 3. To je možné stiahnuť na adrese `https://www.python.org/downloads/`.

1. Nainštalujte balík `python3`.

Po inštalácií Python-u je potrebné nainštalovať závislosti knižnice Migen.

2. `cd <projekt>/fpga/migen`
3. `python setup.py`

Preklad sa spustí skriptom `build.py`.

4. `cd <projekt>/fpga/ov3`
5. `python build.py`

Vygeneruje sa súbor `top.v` v zložke `<projekt>/fpga/ov3/build`.

C.2 Preklad VHDL nástrojmi Xilinx ISE

Pred prekladom VHDL do bitstreamu pre FPGA je nutné nainštalovať nástroje Xilinx ISE. Bezplatná verzia ISE WebPack je dosupná na `http://www.xilinx.com/products/design-tools/ise-design-suite/ise-webpack.html`.

1. Nainštalujte balík Xilinx ISE alebo Xilinx ISE WebPack

Potom je možné otvoriť súbor projektu v nástroji ISE Project Navigator.

2. Otvorte súbor projektu `<projekt>/fpga/ise/projekt.xise`

Vykonajte preklad a generovanie výsledného bitstreamu.

3. Vyberte jednotku `minerva_top` a dva krát kliknite na *Generate programming file* v časti *Processes*

C.3 Naprogramovanie konfigurácie FPGA

Pomocou nástroja iMPACT je možné nahrať konfiguráciu do FPGA aj do nevolatilnej pamäte Flash prítomnej na kite Minerva.

1. Pripojte JTAG programátor k doske Minerva na programovací konektor P7 podľa tabuľky C.1.

Pin P7	Signál	Farba vodiča
1	GND	čierna
2	TDO	fialová
3	TCK	žltá
4	TDI	biela
5	TMS	zelená
6	VREF	červená
-	HALT	šedá

Tabuľka C.1: Zapojenie programátora na konektor P7 kitu Minerva. Farby vodičov zodpovedajú programovaciemu adaptéru dodávanému k Xilinx Platform Cable.

2. Spustite nástroj iMPACT dvojitým kliknutím na *Manage Configuration Project*
3. Kliknite pravým tlačidlom na symbol FPGA a vyberte možnosť *Program*

Aby bolo možné program do FPGA nahrať, môže byť potrebné vyradiť z činnosti obvod VNC na kite Minerva, ktorý bol určený na programovanie FPGA bez JTAG programátora. Ak sa programovanie nepodarilo, uveďte obvod VNC do stavu reset.

4. Prepojte piny 3 a 4 konektoru P4 na doske Minerva

C.4 Zostavenie programu pre MCU

Na vývoj programu pre MCU bolo použité prostredie Kinetis Design Studio 3.0. Program je dostupný na stiahnutie na stránkach firmy NXP: `http://www.nxp.com/products/software-and-tools/run-time-software/kinetis-software-and-tools/ides-for-kinetis-mcus/kinetis-design-studio-integrated-development-environment-ide:KDS_IDE`

1. Nainštalujte Kinetis Design Studio (KDS)

Po prvom spustení bude užívateľ vyzvaný na vytvorenie tzv. workspace.

2. Spustite KDS a vytvorte pracovný priečinok (workspace).

Importujte projekt s programom.

3. Z ponuky File zvoľte Import
4. Vyberte General, Existing Projects into Workspace a pokračujte stlačením Next
5. Zvoľte možnosť Select root directory
6. Stlačením Browse vyberte priečinok `<projekt>/mcu/kds/DP_xjanco04_MQX`
7. Dokončite import stlačením Finish

Zostavenie programu sa vykoná stlačením tlačidla Build s ikonou kladivka.
Voliteľne môžete upraviť IP adresu zariadenia v súbore `Sources/ip.h`

C.5 Zavedenie a ladenie programu v MCU

Ladenie programu a jeho zavedenie do MCU je možné priamo z prostredia KDS.

1. Pripojte kit Minerva pomocou USB portu k počítaču
2. Z ponuky Debug vyberte Debug configurations
3. Vyberte alebo vytvorte novú konfiguráciu GDB PEMicro Interface Debugging
4. V záložke Debugger, v časti Interface zvoľte USB Multilink
5. V časti Port zvoľte pripojené zariadenie
6. V časti Device zvoľte K60DN512ZM10
7. Ladenie spustite kliknutím na Debug

Napálenie programu do MCU je možné pomocou tlačidla Flash from file s ikonou blesku.

C.6 Pripojenie obvodu s transceiverom USB

1. Pripojte prípravok na port P5 kitu Minerva ako je uvedené na obrázku v prílohe D

C.7 Prístup k užívateľskému rozhraniu

Užívateľské rozhranie je dostupné cez počítačovú sieť pomocou webového prehliadača.

1. Pripojte kit Minerva k sieti Ethernet pomocou konektoru RJ45
2. Otvorte vo webovom prehliadači adresu `http://<ip>`

`<ip>` je IP adresa z `Sources/ip.h`

Príloha D

Fotografia USB analyzátora s kitom Minerva