Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Škvorc Urban

Rekonfiguracija strojne opreme v času izvajanja

magistrsko delo

magistrski program druge stopnje
računalništvo in informatika

Mentor: izr. prof. dr. Patricio Bulić
Somentor: doc. dr. Anton Biasizzo

Ljubljana, 2017

To delo je ponujeno pod licenco *Creative Commons Priznanje avtorstva-Deljenje pod enakimi pogoji 2.5 Slovenija* (ali novejšo različico). To pomeni, da se tako besedilo, slike, grafi in druge sestavine dela kot tudi rezultati diplomskega dela lahko prosto distribuirajo, reproducirajo, uporabljajo, priobčujejo javnosti in predelujejo, pod pogojem, da se jasno in vidno navede avtorja in naslov tega dela in da se v primeru spremembe, preoblikovanja ali uporabe tega dela v svojem delu, lahko distribuira predelava le pod licenco, ki je enaka tej. Podrobnosti licence so dostopne na spletni strani creativecommons.si ali na Inštitutu za intelektualno lastnino, Streliška 1, 1000 Ljubljana.

Izvorna koda diplomskega dela, njeni rezultati in v ta namen razvita programska oprema je ponujena pod licenco GNU General Public License, različica 3 (ali novejša). To pomeni, da se lahko prosto distribuira in/ali predeluje pod njenimi pogoji. Podrobnosti licence so dostopne na spletni strani `http://www.gnu.org/licenses/`.

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Zahvala

Zahvaljujem se mentorju, somentorju in moji družini, ki me je podpirala med študijem in izdelavo magistrske naloge.

Škvorc Urban, 2017

Contents

Povzetek

Abstract

1 Uvod **1**

1.1 Pregled sorodnih del 2

1.2 Prispevki magistrske naloge 2

1.3 Zgradba naloge 3

2 Rekonfiguracija v času izvajanja **5**

2.1 Skupne lastnosti rekonfigurabilnih primerov 6

3 Vodilo PCIe **9**

3.1 Naslovni prostori 10

3.2 PCIe paketi 13

3.3 Prekinitve 17

4 Izdelana rekonfigurabilna platforma **19**

4.1 Cilji in naloge platforme 19

4.2 Primer uporabe platforme 21

4.3 Struktura izdelane platforme 22

4.4 Uporabljena strojna in programska oprema 22

4.5 FPGA Vezje 24

4.6 Gostitelj 36

5 Rezultati **41**

5.1 Prostorske zahteve 41

5.2 Hitrost prenosa podatkov 44

5.3 Testne rekonfigurabilne aplikacije 45

5.4 Zahteve za uporabo platforme 47

5.5 Primerjava z obstoječimi rešitvami 48

5.6 Nadaljnje delo 50

6 Zaključek **51**

A Registri izdelanega FPGA sistema **53**

B Avtomat za nadzor nad rekonfiguracijo **57**

Seznam uporabljenih kratic

kratica	angleško	slovensko
FPGA	Field-programmable gate array	
ICAP	Internal Configuration Access Port	Notranji vmesnik za dostop do konfiguracije
PCIe	Peripheral Component Interconnect Express	Hitro povezovalno vodilo za dostop do notranjih naprav
MSI	Message Signaled Interrupts	Prekinitve, posredovane preko sporočil
LUT	look-up table	vpogledna tabela

Povzetek

Naslov: Rekonfiguracija strojne opreme v času izvajanja

Rekonfiguracija v času izvajanja omogoča spreminjanje dela FPGA vezja brez prekinitve delovanja ostalih delov sistema. Takšen način rekonfiguracije omogoča številne prednosti, kot so zmanjšanje prostorske porabe, pohitritev FPGA sistemov in odpravljanje napak med delovanjem sistema. V magistrski nalogi bomo najprej predstavili rekonfiguracijo v času izvajanja in njene prednosti. Sistemi, ki uporabljajo rekonfiguracijo v času izvajanja si delijo skupne lastnosti. Te vključujejo statični del, ki se med izvajanjem ne spreminja in skrbi za rekonfiguracijo in komunikacijo z ostalimi komponentami sistema. V glavnem delu magistrske naloge bomo predstavili razvito strojno platformo, ki razvijalcem nudi te skupne lastnosti. Razvita platforma poleg rekonfiguracije podpira tudi komunikacijsko vodilo PCIe. S tem je omogočena uporaba FPGA sistema kot koprocesor v večjem strojnem sistemu. Platforma vključuje tudi gonilnik in aplikacije, ki omogočajo enostavno uporabo FPGA vezja kot koprocesor v računalniškem sistemu, ki uporablja operacijski sistem Ubuntu.

Ključne besede

FPGA, Računalniška vezja, Rekonfiguracija

Abstract

Title: Runtime hardware reconfiguration

Runtime hardware reconfiguration allows developers to change a part of an FPGA system while it is running. This method of reconfiguration provides several advantages. These include reducing the space consumption of a FPGA system or increasing configuration speed. Another benefit is the ability of such a system to repair certain errors while it is running. In this thesis, we will first describe runtime reconfiguration and its advantages. FPGA systems that use runtime reconfiguration share several features. These include a non-changing static part that is in charge of the reconfiguration process, as well as of communicating with other system components. The main part of the thesis will be focused on the development of a hardware platform that aims to provide developers with some of those shared features. In addition to allowing runtime reconfiguration, the developed platform also provides support for the PCIe bus, which allows the FPGA system to be used as a coprocessor in a larger hardware system. Specifically, the developed platform includes a driver and software applications that allow the FPGA system to be used as a coprocessor in a computer system running the Ubuntu operating system.

Keywords

FPGA, Computer circuits, Reconfiguration

Poglavje 1

Uvod

Rekonfigurabilna vezja FPGA (Field-programmable gate array) za razliko od tradicionalnih vezij omogočajo spreminjanje vezja po njegovi izdelavi. Pri tradicionalnih vezjih je potrebno ob vsaki spremembi strojne logike izdelati novo vezje, kar je drago in zamudno opravilo. Z uporabo vezij FPGA to ni potrebno, saj je ob spremembi logike potrebno le ponovno programiranje vezja.

Rekonfiguracija v času izvajanja omogoča spreminjanje dela FPGA vezja med njegovim delovanjem. Takšen način rekonfiguracije ponuja razvijalcem strojne opreme možnost izdelave vezij, ki jih ne bi bilo mogoče implementirati s tradicionalnimi vezji. V tej nalogi bomo najprej na kratko opisali vezja FPGA. Nato bomo opisali prednosti rekonfiguracije v času izvajanja ter pregledali nekaj konkretnih primerov uporabe. Predstavili bomo skupne lastnosti, prisotne v večini rešitev, ki uporabljajo rekonfiguracijo v času izvajanja. Glavni praktični prispevek magistrske naloge je ustvariti rešitev oziroma platformo, ki bo razvijalcem rekonfigurabilnih sistemov ponujala storitve, ki si jih takšni sistemi delijo. V magistrski nalogi bomo opisali razvoj in lastnosti te rešitve.

1.1 Pregled sorodnih del

Rekonfiguracija v času izvajanja je uporabljena v mnogih različnih aplikacijah. Eden od primerov je odpravljanje napak v času izvajanja, predvsem v vesoljskih napravah, kjer sončno sevanje lahko poškoduje FPGA vezje [1, 2, 3, 4]. Takšen način odprave napak brez možnosti popravljanja konfiguracije vezja med izvajanjem ne bi bil možen.

Druga možnost je uporaba rekonfiguracije za izboljšanje že obstoječih sistemov. Tu lahko rekonfiguracija izboljša prostorsko učinkovitost mnogih FPGA algoritmov. Na FPGA vezju je lahko ob določenem trenutku naložen le tisti del algoritma, ki se trenutno izvaja. Takšen način delovanja je možno izrabiti v veliko različnih FPGA algoritmih. Primeri so grafično izrisovanje [5, 6], zaznavanje obrazov [7], strojno učenje z nevronskimi mrežami [8] in strojno definirani radijski sprejemniki in oddajniki [9].

Večina takšnih algoritmov si deli nekatere ključne lastnosti. FPGA vezje v rekonfigurabilnem sistemu se zelo pogosto deli na statični del, ki se med izvajanjem nikoli ne spreminja in skrbi na primer za komunikacijo z ostalimi komponentami sistema in za nadzor nad rekonfiguracijo. Poleg statičnega dela sistem vsebuje še več rekonfigurabilnih delov, ki se med izvajanjem lahko spreminjajo. [5, 10] podata skupne lastnosti, ki jih morajo podpirati takšni rekonfigurabilni sistemi.

Glavni cilj magistrske naloge je izdelava sistema, ki bo razvijalcem ponujal te skupne lastnosti. Nekaj takšnih sistemov je že bilo razvitih, vendar njihova implementacija velikokrat ni splošno dostopna ali pa ima določene pomanjkljivosti. Primeri dosedanjih implementacij so podani v delih [11, 12, 13, 14, 15].

1.2 Prispevki magistrske naloge

Algoritmi, ki uporabljajo rekonfiguracijo v času izvajanja, si delijo veliko skupnih gradnikov oziroma modulov [10]. Primeri so modul za nadzor nad rekonfiguracijo in modul za dostop do raznih dodatnih virov, na primer do pomnilnika. Poleg pregleda in opisa teh skupnih delov je glavni cilj magistrske naloge izdelava platforme, ki ponuja čim več od teh skupnih modulov. Razvijalcem algoritmov je z uporabo platforme potrebno razviti le dejanski algoritem, brez potrebe po razvoju celotne arhitekture, ki omogoča rekonfiguracijo med časom izvajanja. Platforma ponuja tudi možnost uporabe FPGA sistema kot koprocesorja v večjem strojnem sistemu (gostitelju). Prenos podatkov med gostiteljem in FPGA sistemom je omogočen z vodilom PCIe. Poleg tega platforma ponuja aplikacije in gonilnik za gostiteljski sistem, ki omogočajo krmiljenje platforme.

Platforma je sestavljena iz statičnega modula na vezju FPGA ter uporabniku prijaznega vmesnika na gostitelju. Modul, implementiran na vezju FPGA, skrbi za rekonfiguracijo in za dostop do raznih virov na vezju FPGA, na primer do pomnilniških enot. Ta modul skrbi tudi za komunikacijo z gostiteljem. Drugi modul uporabniku ponuja enostaven način za nadzor nad rekonfiguracijo iz gostiteljevega operacijskega sistema. Cilj izdelanega sistema je olajšanje razvoja rekonfigurabilnih sistemov, saj izdelana platforma razvijalcem ponuja že razvito ogrodje, ki ponuja storitve, skupne mnogim rekonfigurabilnim sistemom.

1.3 Zgradba naloge

V poglavju 2 bomo predstavili rekonfiguracijo v času izvajanja ter pregledali nekaj konkretnih primerov uporabe takšnih sistemov. Predstavili bomo lastnosti, ki so skupne pregledanim rešitvam.

V poglavju 3 bomo predstavili vmesnik PCIe, ki omogoča hiter prenos

podatkov med različnimi strojnimi napravami. Takšen prenos je pomemben, če želimo FPGA vezje vključiti v nek večji računalniški sistem, na primer namizni računalnik. S tem omogočimo uporabo FPGA vezja kot strojnega pospeševalnika, kar lahko pohitri delovanje mnogih algoritmov v primerjavi s tradicionalnimi računalniškimi procesorji.

V poglavju 4 bomo predstavili glavni prispevek naloge: izdelano platformo, ki nudi nekatere lastnosti skupne rešitvam, ki uporabljajo rekonfiguracijo v času izvajanja. Predstavili bomo tudi konkreten primer uporabe platforme.

V poglavju 5 bomo predstavili značilnosti in rezultate izdelane platforme. Platformo bomo tudi primerjali z nekaterimi obstoječimi rešitvami.

Poglavje 2

Rekonfiguracija v času izvajanja

Tradicionalen postopek rekonfiguracije FPGA vezja zahteva, da je vezje med procesom konfiguracije v nedelujočem stanju. Tako je potrebno ob kakršni koli spremembi vezja zaustaviti cel sistem, nanj prenesti novo konfiguracijo in sistem ponovno zagnati. Z rekonfiguracijo v času izvajanja je možno spremeniti del vezja, brez zaustavitve ostalih delov sistema. V tem poglavju bomo predstavili prednosti, ki jih prinaša rekonfiguracija v času izvajanja in lastnosti, katere si delijo sistemi, ki uporabljajo takšen način rekonfiguracije.

Rekonfiguracija v času izvajanja omogoča spreminjanje konfiguracije vezja med delovanjem sistema. Takšen način delovanja omogoča razvoj strojnih rešitev, ki ne bi bile možne z uporabo tradicionalnega postopka konfiguracije. Nekatere prednosti, ki jih prinaša rekonfiguracija v času izvajanja, so:

Zmanjšanje porabljenega prostora. Velikost FPGA vezja je eden izmed dejavnikov, ki zelo vplivajo na zmogljivost in ceno končnega sistema. Če je izdelano vezje preveliko za uporabljeno FPGA vezje, potem je potreben nakup večjega in s tem dražjega vezja.

Z uporabo rekonfiguracije v času izvajanja je možno sistem razdeliti na več delov, kjer je ob nekem trenutku na fizičnem vezju prisoten le eden

izmed teh delov. Primer je kodirnik/dekodirnik podatkov. V tradicionalnem sistemu bi morala biti na sistemu hkrati prisotna tako modul za kodiranje kot za dekodiranje podatkov. Z uporabo rekonfiguracije je možno na vezje naložiti le tisti modul, ki je trenutno potreben. Primeri takšnih sistemov so [16, 17, 9, 18].

Hitrejši čas začetne konfiguracije. Za nekatere sisteme je pomembno, da je FPGA vezje ob zagonu čim prej na voljo za uporabo. Primer je vodilo PCIe, ki zahteva, da je strojna oprema, ki uporablja to vodilo za priklop na gostitelja, pripravljena za uporabo v času do 100 ms po vklopu gostitelja [19], medtem ko lahko konfiguracija velikih vezij vzame tudi več sekund. Z uporabo rekonfiguracije je možno ob zagonu naložiti le majhen del vezja, ki skrbi za vmesnik PCIe, in šele potem naložiti preostali del vezja. Primera takšnih sistemov sta [20, 21].

Odprava napak v času izvajanja. Razni fizikalni pojavi lahko spremenijo dele FPGA konfiguracije. Primer je sončno sevanje, ki predstavlja probleme predvsem za vesoljske aplikacije FPGA vezij. Z uporabo rekonfiguracije v času izvajanja je možno zaznati in odpraviti te napake brez zaustavitve celotnega sistema. Primeri takšnih sistemov so [2, 3, 22]. Nidhin v [1] ponuja pregled tega področja.

2.1 Skupne lastnosti rekonfigurabilnih primerov

Primeri rešitev, opisanih v tem poglavju, si delijo veliko skupnih lastnosti. Za vse velja, da so razdeljeni na več delov. Eden od teh delov se med izvajanjem sistema ne spreminja. Ta del skrbi za naloge, ki so vedno potrebne za delovanje sistema. Primeri teh nalog so povezljivost z zunanjimi komponentami in nadzor nad procesom rekonfiguracije. Poleg statičnega dela vsak

sistem vsebuje tudi enega ali več dinamičnih delov. Ti deli se med izvajanjem sistema lahko spreminjajo in predstavljajo glavni del celotne rešitve.

Primeri se razlikujejo v tem, kje hranijo možne konfiguracije. V prvem načinu primer že ob zagonu vsebuje vse konfiguracije, ki jih uporablja. Te konfiguracije so lahko shranjene ali na FPGA vezju ali na zunanjih pomnilniških enotah in se med izvajanjem ne spreminjajo. Rešitev lahko torej le preklaplja med že določenimi konfiguracijami. V drugem načinu je rešitev povezana z neko zunanjo napravo (na primer z namiznim računalnikom), ki ji pošilja konfiguracije. V tem primeru lahko zunanja naprava sama ustvarja in spreminja možne konfiguracije. S tem načinom je možna večja fleksibilnost, saj konfiguracij ni potrebno poznati ob zagonu sistema. Takšen način hranjenja konfiguracij prinaša dodatno kompleksnost, saj mora FPGA vezje znati komunicirati z zunanjo napravo. Takšen način je smiseln, če želimo ustvariti bolj splošno rešitev. Uporaben je tudi, če je FPGA naprava uporabljena kot koprocesor v večjem strojnem sistemu (gostitelju), saj je v tem primeru komunikacija med FPGA vezjem in gostiteljem že prisotna.

Poglavje 3

Vodilo PCIe

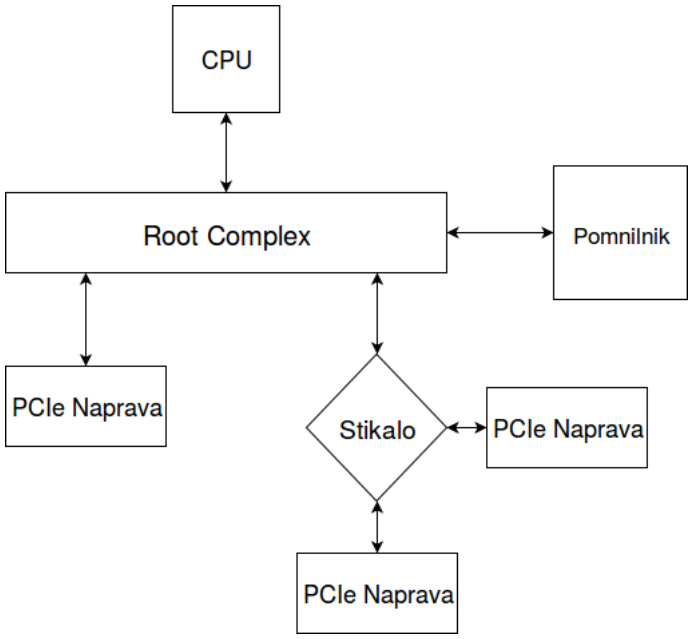
Eden izmed načinov uporabe FPGA vezja je kot koprocesor v nekem večjem računalniškem sistemu (gostitelju) [23]. V tem primeru je potrebna implementacija prenosa podatkov med FPGA vezjem in gostiteljem. Ker izdelana platforma, opisana v poglavju 4, podpira takšen način delovanja, bomo v tem poglavju opisali enega izmed načinov takšnega prenosa podatkov, vodilo PCIe (Peripheral Component Interconnect Express) [24].

Vodilo PCIe je nastalo leta 2003 kot nadgradnja starejših vodil PCI in PCI-X. Danes je vodilo PCIe daleč najbolj priljubljen način povezave naprav, kot so grafične kartice, zvočne kartice in ostale naprave, ki potrebujejo način za izjemno hiter prenos podatkov na in iz gostiteljevega pomnilnika. PCIe je serijsko vodilo, v katerem so naprave povezane z gostiteljem preko komponente, imenovane *root complex*. Ta komponenta, ki je lahko ločena naprava ali danes bolj pogosto del gostiteljevega procesorja, skrbi za dostop do gostiteljevega pomnilnika in za komunikacijo z gostiteljevim procesorjem.

Vodilo PCIe si deli lastnosti s sodobnimi omrežnimi protokoli. Naprave, ki se povezujejo z gostiteljem, so organizirane v omrežno topologijo, prikazano v sliki 3.1, prenos podatkov pa je organiziran v pakete.

PCIe vmesnik za prenos podatkov uporablja podatkovne pasove. PCIe naprave lahko uporabljajo od enega do šestnajst vzporednih pasov. Uporaba

več podatkovnih pasov pomeni večjo hitrost prenosa podatkov.



Slika 3.1: Topologija PCIe vodila. PCIe naprave se neposredno ali preko stikal povežejo na gostitelja preko komponenta root complex. Ta komponenta je lahko ali ločeno vezje ali pogosto del gostiteljevega procesorja.

3.1 Naslovni prostori

Naprave, ki uporabljajo vodilo PCIe, hranijo svoje podatke v treh različnih naslovnih prostorih. Vsak od njih ima svoj specifičen namen, in platforma, izdelana v sklopu te magistrske naloge, uporablja vse tri. Spodaj so na kratko predstavljeni vsi 3 naslovni prostori:

I/O prostor. Podpira le 32-bitne naslove in je namenjen predvsem združljivosti s starejšimi PCI standardi [25]. Prednost tega prostora je njegova enostavna uporaba. I/O prostor uporablja način prenosa podatkov, imenovan *programmed input/output*. V tem načinu je procesor gostitelja

zadolžen za celoten postopek prenosa podatkov. Ker procesor med prenosom podatkov ne more izvajati drugih opravil, je takšen prenos počasen.

V izdelani platformi je ta naslovni prostor namenjen za prenos raznih kontrolnih sporočil med gostiteljem in FPGA sistemom. Takšna sporočila so zelo kratka, zato hitrost prenosa podatkov pri njih ni pomembna. Ta sporočila vključujejo prenos podatkov, ki so potrebni za sprožitev hitrejših načinov prenosa podatkov, opisanih spodaj. Ta način se uporablja tudi za nastavitev začetnih parametrov FPGA sistema.

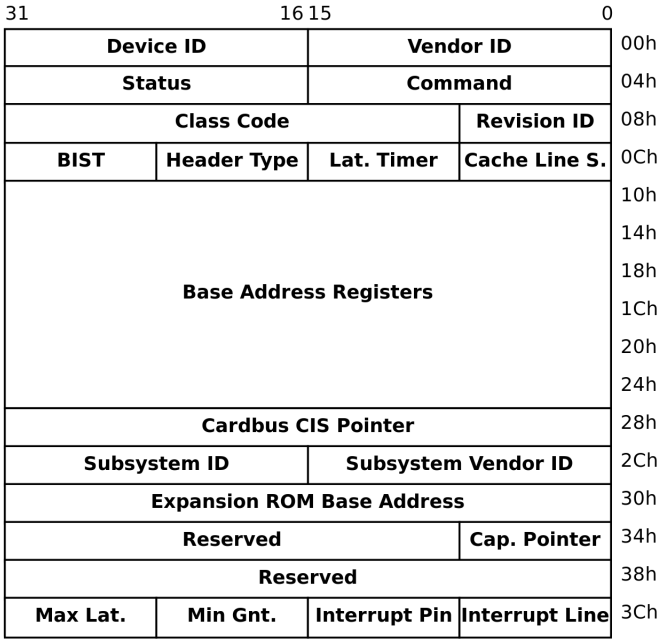
Pomnilniški prostor. Podpira tako 32- kot 64-bitne naslove in prenos podatkov z *neposrednim dostopom do pomnilnika* (ang. Direct Memory Access oziroma DMA). Z uporabo neposrednega dostopa do pomnilnika PCIe naprava prevzame nadzor nad prenosom podatkov. PCIe naprava dostopa do gostiteljevega pomnilnika preko komponente root complex. PCIe naprava tej komponenti pošlje ustrezno sporočilo in od komponente dobi odgovor, ki vsebuje želene podatke. Med prenosom lahko procesor gostitelja izvaja poljubne operacije. Takšen način prenosa podatkov je znatno hitrejši od načina *programmed input/output*, ki ga uporablja I/O prostor, zato je ta naslovni prostor uporabljen za hiter splošen prenos podatkov. Ta naslovni prostor predstavlja glavni podatkovni prostor PCIe naprave. V izdelani platformi je ta prostor uporabljen za hiter prenos podatkov, ki jih obdeluje PCIe naprava.

Konfiguracijski prostor. Vsebuje razne nadzorne podatke PCIe naprave. To so na primer identifikacijska številka naprave ter proizvajalca in podatki o zmogljivostih, ki jih naprava podpira. Konfiguracijski prostor je velik od 256 bajtov do 4 kilobajte. Slika 3.2 prikazuje standardizirane registre za tip PCIe naprav, ki je uporabljen v naši izdelani platformi. Nekateri pomembni registri so:

Device ID. Identifikacijska številka PCIe naprave. Ta številka je določena s strani PCIe naprave. Nanjo se sklicujejo gonilniki, napisani za to napravo.

Status. Statusni register PCIe naprave, ki ima dve glavni nalogi. Prva sporoča gostitelju zmogljivosti PCIe naprave. Druga sporoča probleme in napake na PCIe napravi.

Command. Nadzorni register, ki omogoča gostitelju nadzor nad določenimi zmogljivostmi PCIe naprave. S pisanjem v ta register je mogoče na primer omogočiti oziroma onemogočiti dostop do I/O in pomnilniškega prostora, kot tudi PCIe napravi omogočiti oziroma onemogočiti prenos podatkov v načinu DMA.



Slika 3.2: Standardizirani registri v PCIe naslovnem prostoru za tip naprave 0 (non-bridged). Velikost naslovnega prostora je 256 bajtov. Vir: [26]

3.2 PCIe paketi

Prenos podatkov preko PCIe poteka podobno kot v sodobnih omrežnih protokolih, kot je na primer protokol TCP (Transmission Control Protocol). Podatki so tako pri prenosu preko vodila PCIe organizirani v pakete. V tem poglavju bomo okvirno predstavili PCIe pakete. Osredotočili se bomo na lastnosti, ki smo jih potrebovali za razvoj platforme.

PCIe paketi se delijo v dve vrsti: zahteve (ang. requests) in odgovore (ang. completions). Paketi za zahteve se nadaljnje delijo na pakete za pisanje in pakete za branje. Tako PCIe uporablja vsaj tri različne pakete: zahtevo za branje, zahtevo za pisanje in odgovor na branje (ki vrne podatke, po katerih je poizvedoval zahtevek za branje). Pri pisanju z uporabo neposrednega dostopa do pomnilnika se odgovor na pisanje ne uporablja. Tako napravi, ki piše podatke, ni potrebno čakati na odgovor od prejemnika, kar omogoča hitrejši prenos. Pri pisanju v naslovni prostor I/O se uporablja odgovor na pisanje, torej mora naprava, ki piše, čakati na odgovor prejemnika.

Vsak PCIe paket se začne z glavo, ki je lahko dolga 3 ali 4 32-bitnih besed. Glave dolžine treh besed se uporabljajo za operacije v 32-bitnem naslovnem prostoru, medtem ko so 4 besedne glave uporabljene za operacije v 64-bitnem naslovnem prostoru. Prva beseda glave je enaka za vse pakete in vsebuje sledeča glavna polja:

Format (Fmt), 2 bita pove, za kakšno obliko paketa gre. Bit 0 pove, ali paket po glavi vsebuje še dodatne podatke (kot jih pri paketih za pisanje in pri odgovorih). Bit 1 pove, ali ima glava 3 ali 4 besede.

Tip (Type), 5 bitov bolj podrobno opiše tip paketa. Tu je na primer določeno, v kateri naslovni prostor se piše/bere (I/O ali pomnilniški prostor).

Dolžina (Length), 10 bitov pove, koliko besed vsebuje podatkovni del paketa pri pisanju oziroma koliko besed želimo prebrati.

Naslednje besede se razlikujejo glede na tip paketa. Paketi, ki zahtevajo branje/pisanje podatkov, vsebujejo v teh besedah sledeče podatke:

ID Pošiljatelja (Requester ID), 16 bitov. To polje služi za identifikacijo pošiljatelja. Uporablja se za to, da lahko prejemnik odgovore pošlje pravilni napravi. Prisotna mora biti tudi pri DMA zahtevkih za pisanje, čeprav tu nima ključne vloge. ID prejemnika ni potreben, ker je sprejemna naprava pri DMA prenosu že enolično določena z naslovom.

Značka (Tag), 8 Bitov. V značko lahko pošiljatelj vključi poljubne podatke. Odgovori na ta paket bodo vsebovali enako značko. To lahko prejemnik uporabi, da poveže poslane pakete s pravilnimi odgovori.

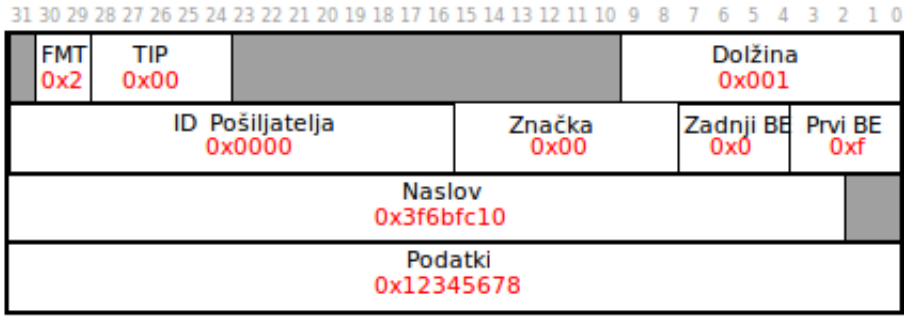
Prvi/Zadnji Omogočeni Bajti (First/Last Byte Enable), 4 bite vsak. V teh dveh poljih pošiljatelj pove, katere bajte prve in zadnje besede želi prebrati ali zapisati. To je pomembno, če želi pošiljatelj pisati/brati polje dolžine, ki ni večkratnik 4 bajtov. Če želi na primer pisati 2 bajta, bo poslal zahtevek za branje dolžine 1 besede, vrednost polja First Byte Enable pa bo 0x0F. Zapisana bosta le nižja 2 bajta.

Naslov (Address), 30 ali 62 bitov. Pove naslov prve besede, ki jo pošiljatelj želi brati ali pisati. Tu je pomembno, da so 32-bitni naslovi označeni le s 30 biti (64-bitni naslovi pa le z 62 biti). To polje ne vsebuje dveh najnižjih bitov. Za dejanski naslov je to polje potrebno pomnožiti s 4.

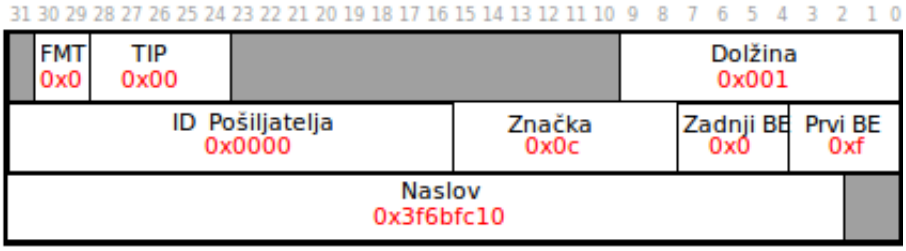
Po glavi zahtevki za pisanje vsebujejo polja s podatki. Podatki uporabljajo urejenost bajtov po *pravilu debelega konca* (ang. *big endian*), medtem ko sodobni procesorji arhitekture x86 uporabljajo urejenost po *pravilu tankega konca* (ang. *little endian*). Podatki so tako v PCIe paketih obrnjeni. 32-bitna beseda, ki je v urejenosti po pravilu tankega konca predstavljena kot 0x12345678, bo v urejenosti po pravilu debelega konca predstavljena kot 0x78563412. Sam paket ne vsebuje nobenih informacij o tem, kako so bajti

urejeni v pošiljateljevi arhitekturi. Zato je potrebno, da ali pošiljatelj ali prejemnik pravilno uredita bajte.

Na sliki 3.3 je predstavljen paket, ki v naslov 0xfdaff040 zapiše vrednost 0x12345678. Opazimo, da je polje za naslov enako 0x3f6bfc10. Ko to pomnožimo s štiri, dobimo dejanski naslov 0xfdaff040. Na sliki 3.4 je prikazan paket za branje podatkov dolžine enega bajta iz naslova 0xfdaff040.



Slika 3.3: Shema PCIe DMA paketa za branje podatkov. Prve tri besede predstavljajo glavo paketa. Zadnja beseda so podatki, ki jih paket želi zapisati. Polja, označena s sivo barvo, predstavljajo polja, ki imajo ali vedno vrednost 0, ali pa jih večina PCIe naprav ignorira. Vir: [27]



Slika 3.4: Shema PCIe DMA paketa za pisanje podatkov. Paket je zelo podoben paketu za branje. Razlika je le polje Fmt (Format), ki pove, da je to bralni in ne pisalni paket. Manjka tudi zadnja beseda, saj za branje ni potrebno podati podatkov. Vir: [27]

Paketi odgovorov so podobni kot paketi zahtev. Polja v njihovi glavi se razlikujejo v sledečih lastnostih:

Dolžina (Length), 10 bitov. Pomembno se je zavedati, da lahko en zahtevek potrebuje več kot en paket odgovora. To je zato, ker imajo PCIe paketi omejeno dolžino, odvisno od posamezne PCIe naprave. Če zahtevek želi prebrati 50 besed, največja dolžina podatkov v paketu pa je 10 besed, bo ta zahtevek potreboval 5 paketov odgovorov. Polje dolžina vsebuje število besed v posameznem paketu odgovora.

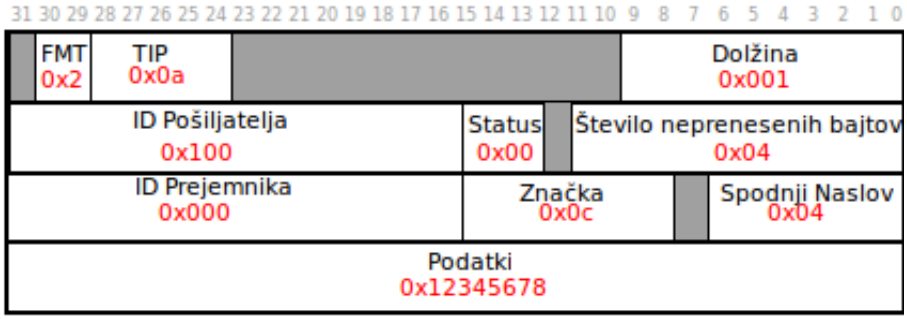
Število neprenesenih bajtov (Byte count), 12 bitov . To polje zamenja polji Byte Enable pri zahtevkih. Vsebuje število bajtov, ki jih je še potrebno prenesti, vključno s trenutnim paketom. Če za odgovor zadostuje en paket, je to število enako številu vseh bajtov, ki jih je zahteval zahtevek. Če je za odgovor potrebno več paketov, to polje vsebuje število besed v prejetem paketu plus število besed v preostalih paketih.

Spodnji Naslov (Lower Address), 7 bitov. Namesto polnega naslova odgovori vsebujejo le sedem najmanj pomembnih bitov naslova. Ti stsoa uporabljeni za ureditev prejetih podatkov, ko je potrebno več paketov odgovora za en zahtevek.

Status, 2 bita. To polje ima vrednost 0, če je bilo branje uspešno. Neničelne vrednosti predstavljajo napake pri branju

ID Pošiljatelja in ID prejemnika, 16 bitov vsak. Poleg ID-ja naprave, ki paket pošilja (Completer ID), je pri odgovorih potreben tudi ID prejemnika. Pri zahtevkih je ciljna naprava določena s pomnilniškim naslovom, pri odgovorih pa to ni možno, zato je uporabljena ID številka, ki jo je naprava prejela v paketu zahtevka.

Tudi pri odgovorih glavi sledijo podatki, za katere veljajo enaka pravila kot pri zahtevkih. Slika 3.5 prikazuje primer odgovora na paket iz slike 3.4.



Slika 3.5: Shema PCIe DMA paketa odgovora na zahtevo po branju podatkov. Prve tri besede predstavljajo glavo paketa, zadnja pa podatke, ki jih je zahteval predhodni paket za branje podatkov. Vir: [27]

3.3 Prekinitve

Poleg zgoraj opisanih sporočil vodilo PCIe za izmenjavo informacij uporablja tudi prekinitve. Prekinitve lahko naprave uporabljajo za hitro sporočanje informacij izven podatkovnih prenosov. Izdelana FPGA platforma s prekinitvami sporoča gostitelju, da je prenos podatkov končan.

PCIe protokol za komunikacijo med napravami ponuja dve vrsti prekinitev: tradicionalne prekinitve (legacy interrrupts) in MSI prekinitve (Message Signaled Interrupts). Tradicionalne prekinitve so ponujene za kompatibilnost s starejšimi PCI napravami.

PCI naprave so vsebovale štiri enobitne fizične linije za sporočanje prekinitev. Vsaka naprava je poslala prekinitev tako, da je določeno linijo nastavila na logično vrednost 0. PCIe teh fizičnih linij ne vsebuje. Za kompatibilnost s starejšimi napravami PCIe ponuja prekinitve, ki se obnašajo kot tradicionalne PCI prekinitve. Te prekinitve uporabljajo posebna sporočila, da oponašajo fizične prekinitvene linije. Ta sporočila povedo, katera navidezna linija naj se nastavi na katero logično vrednost. Ko procesor gostitelja (oziroma komponenta procesorja root complex) prejme ta posebna sporočila, jih

obravnava kot proženje prekinitev in prekinitev sporoči prekinitvenemu krmilniku. Te prekinitve imajo število slabosti. Vsaka naprava je omejena le na štiri različne tipe prekinitev, saj so tradicionalne PCI naprave uporabljaje le štiri različne prekinitvene linije. Poleg tega mora gostitelj po obdelavi prekinitve sporočiti napravi, da je prekinitev obdelal, da naprava prekinitveno linijo nastavi nazaj na neaktivno vrednost.

MSI Prekinitve so nadgrajena implementacija prekinitev, uvedene v PCIe standardu. Te prekinitve naprava sproži s pisanjem v vnaprej določeno lokacijo. Pri sodobnih procesorjih naprava prekinitve sproži kar z zahtevkom za pisanje v določeni register gostiteljevega prekinitvenega krmilnika [28]. Takšen način prekinitev prinaša nekatere prednosti. Gostitelju ni več treba sporočiti PCIe napravi, da je obdelal njeno prekinitev. Poleg tega te prekinitve ponujajo napravi uporabo večjega števila tipov prekinitev, saj lahko podatki, ki jih PCIe naprava piše, sporočajo tip prekinitve.

Poglavje 4

Izdelana rekonfigurabilna platforma

V tem poglavju bomo podrobno predstavili glavni praktični prispevek magistrske naloge: FPGA platformo za olajšanje razvoja rešitev, ki uporabljajo rekonfiguracijo v času izvajanja. Glavni cilj izdelane platforme je razvijalcem ponuditi ogrodje, ki že vsebuje nekatere komponente, skupne sistemom, ki uporabljajo rekonfiguracijo v času izvajanja. Tako se lahko razvijalci posvetijo zgolj konkretni rešitvi specifičnega problema, brez zamudnega razvoja celotne arhitekture, ki omogoča rekonfiguracijo v času izvajanja.

4.1 Cilji in naloge platforme

Cilj izdelane platforme je razvijalcem ponuditi rešitve za probleme, skupne sistemom, ki uporabljajo rekonfiguracijo v času izvajanja. Izdelana platforma je namenjena predvsem algoritmom, ki uporabljajo rekonfigurabilna FPGA vezja kot koprocesor. V tem primeru je FPGA sistem povezan z gostiteljem (na primer z namiznim računalnikom). FPGA sistem je lahko uporabljen za pospešitev določenih delov algoritma, kjer je FPGA sistem hitrejši ali bolj učinkovit kot procesor gostitelja. Izdelana platforma je lahko uporabljena

tudi za katero koli rešitev, ki želi prenašati podatke iz gostitelja na FPGA sistem in obratno.

Konkretne lastnosti, ki jih ponuja izdelana platforma, so:

Prenos podatkov med FPGA sistemom in gostiteljem. Ker je platforma zasnovana tako, da FPGA sistem deluje znotraj gostitelja, je potreben način za prenos podatkov med gostiteljem in FPGA sistemom. Platforma uporablja dve različni kategoriji podatkov. Prvi so kontrolni podatki, s katerimi gostitelj nadzira lastnosti FPGA sistema ali prebere podatke o FPGA sistemu. Drugi del so podatki, ki jih FPGA sistem dejansko obdeluje. Glavna razlika med dvema tipoma podatkov je njihova količina. Kontrolnih podatkov je načeloma malo, medtem ko je lahko podatkov, katere mora FPGA sistem obdelati, zelo veliko. Zato mora biti izbrani način prenosa zmožen prenašati tudi velike količine podatkov.

Nadzor nad rekonfiguracijo FPGA vezja. Platforma mora razvijalcem ponujati način za nadzor nad rekonfiguracijo FPGA vezja. Sem spada prenos datotek, ki opisujejo želene spremembe konfiguracije vezja iz gostitelja na FPGA sistem in proženje procesa rekonfiguracije. Proces rekonfiguracije se mora izvršiti brez zaustavitve FPGA sistema in gostitelja.

Aplikacije in gonilniki za nadzor s strani gostitelja. Želimo, da lahko zgoraj opisani storitvi uporabniki krmilijo preko operacijskega sistema, ki je prisoten na gostitelju. Zato je potrebno implementirati tako nizkonivojske gonilnike kot uporabniku prijazne aplikacije za nadzor nad platformo. Glavni cilj je, da lahko uporabnik preko ukazne vrstice krmili celoten proces prenosa podatkov in rekonfiguracije.

Ključnega pomena za izdelano platformo je komunikacija med vezjem FPGA in gostiteljem. Zato platforma uporablja vodilo PCIe. Vodilo PCIe

je izbrano zaradi dveh razlogov. Prvi razlog je njegova razširjenost. Vodilo je prisotno na vseh sodobnih namiznih računalnikih, kot tudi na veliki večini naprednih FPGA sistemih. Drugi razlog je hitrost prenosa podatkov. Standard PCI Express 1.0 ponuja teoretično enosmerno hitrost prenosa podatkov do okoli 250MB/s na podatkovni pas v vsako smer. V praksi so hitrosti prenosa podatkov manjše zaradi dejavnikov, kot so overhead pri prenosu podatkov [29]. Primer so PCIe paketi, opisani v poglavju 3.2, kjer poleg samih podatkov vsak paket vsebuje še dodatne informacije, kot so naslov in dolžina podatkov.

Platforma je zasnovana za uporabo na sledeči način. Uporabniki najprej v rekonfigurabilni del naložijo želeno FPGA aplikacijo. Nato tej aplikaciji iz gostitelja pošiljajo podatke. Ko FPGA aplikacija te podatke obdela, jih uporabnik lahko prebere iz FPGA vezja nazaj v gostitelja. Z rekonfiguracijo v času izvajanja je možno kadar koli, brez zaustavitve FPGA sistema, zamenjati naloženo FPGA aplikacijo. To omogoča neprekinjeno delovanje PCIe povezave med FPGA sistemom in gostiteljem.

4.2 Primer uporabe platforme

Uporabniki izdelano platformo uporabljajo na sledeči način.

1. Ob prižigu gostitelja (namiznega računalnika) se FPGA platforma preko PCIe vmesnika poveže z gostiteljem.

2. Ob zagonu na rekonfigurabilnem delu FPGA vezja ni naložena nobena aplikacija. Uporabnik mora zato najprej naložiti želeno aplikacijo v rekonfigurabilni del preko ukazne vrstice z ukazom *load_config ime_konfiguracijske_datoteke*.

3. Ko je v rekonfigurabilni del naložena aplikacija, lahko uporabnik preko PCIe vodila na FPGA vezje prenese podatke, za katere želi, da jih

aplikacija obdela. Tudi to lahko stori preko ukazne vrstice z ukazom *load_data ime_podatkovne_datoteke*.

4. Ko so podatki obdelani, uporabnik preko vodila PCIe prenese rezultate z ukazom *read_data ime_podatkovne_datoteke*. Rezultate je možno brati takoj, ko so na voljo, tudi če istočasno pišemo vhodne podatke.

5. Kadar koli med izvajanjem lahko uporabnik zamenja trenutno naloženo aplikacijo tako, da spet uporabi ukaz *load_config*.

4.3 Struktura izdelane platforme

V grobem se izdelana platforma deli na dva dela. Prvi del je prisoten na FPGA sistemu, drugi pa na gostitelju (namiznem računalniku, ki uporablja operacijski sistem Linux).

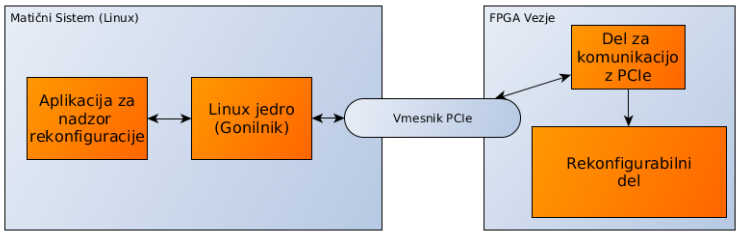
Del na FPGA vezju ima dve glavni nalogi. Prva je nadzor nad prenosom podatkov iz gostitelja na FPGA sistem in obratno. Prenos podatkov je implementiran preko vodila PCIe. Druga naloga je nadzor nad rekonfiguracijo FPGA vezja.

Drugi del, ki deluje v operacijskem sistemu Linux, omogoča uporabniku enostaven nadzor nad procesom rekonfiguracije. Ta del lahko razdelimo v dve glavni komponenti. Prva komponenta je gonilnik, ki omogoča, da operacijski sistem zazna priključeno FPGA kartico in z njo izmenjuje podatke. Drugi del so uporabniške aplikacije, ki omogočajo enostaven nadzor nad prenosom podatkov na in iz FPGA vezja ter nadzor nad procesom rekonfiguracije.

Slika 4.1 prikazuje grobo shemo izdelane platforme.

4.4 Uporabljena strojna in programska oprema

Del platforme, izdelan na gostitelju (namiznem računalniku), deluje na operacijskem sistemu Linux. Pri testiranju platforme smo uporabili različico



Slika 4.1: Shema izdelane platforme za nadzor nad rekonfiguracijo. Platforma je razdeljen na dva dela. Prvi je del na FPGA sistemu, drugi del na gostitelju. Dela med seboj komunicirata preko vmesnika PCIe

.

Ubuntu 14.10. Gonilnike in uporabniške aplikacije smo implementirali v programskem jeziku C. V nekaterih uporabniških aplikacijah uporabljamo tudi programski jezik C++. Za razvoj gonilnikov in uporabniških aplikacij smo uporabili zgolj sistemske knjižnice jezika C/C++ in operacijskega sistema Linux.

Za FPGA sistem smo uporabili sistem Xilinx XUPV5-LX110T [30]. Izbrani FPGA sistem ponuja nujni komponenti, ki sta povezljivost z vodilom PCIe in FPGA vezje z podporo za rekonfiguracijo v času izvajanja.

Za opis vezja smo uporabili strojno opisni jezik Verilog. Za razvoj platforme smo uporabili razvojna orodja, ki jih ponuja proizvajalec uporabljenega FPGA sistema, podjetje Xilinx. Za implementacijo osnovnih storitev platforme, ki ne potrebujejo rekonfiguracije, smo uporabili razvojno orodje ISE. To orodje ponuja pretvorbo Verilog kode v konfiguracijske datoteke, združljive s ciljno FPGA platformo. Za razvoj delne rekonfiguracije smo dodatno uporabili orodje PlanAhead. PlanAhead je uradno podprto orodje za razvoj Xilinoxivh sistemov, ki uporabljajo rekonfiguracijo v času izvajanja.

4.5 FPGA Vezje

V tem delu bomo predstavili del izdelane platforme, ki je prisoten na FPGA vezju.

4.5.1 Komunikacija z gostiteljem

Ker je izdelana platforma namenjena sistemom, ki FPGA vezje uporabljajo kot koprocesor, je komunikacija med FPGA vezjem in gostiteljem ključnega pomena. Izdelana platforma za prenos podatkov med FPGA vezjem in gostiteljem uporablja vodilo PCIe.

Za implementacijo vmesnika vodila PCIe na FPGA vezju smo izhajali iz Xilinxovega jedra *LogiCORE IP Endpoint Block Plus v1.15* [31]. To jedro ponuja osnovno implementacijo PCIe vmesnika, kompatibilnega s standardom PCIe 1.1. Novejših verzij PCIe standarda uporabljeni razvojni sistem (XUPV5-LX110T) ne podpira.

Pri razvoju logike za prenos podatkov smo kot osnovo uporabili Xilinxov referenčni primer *Bus Master Performance Demonstration Reference Design for the Xilinx Endpoint PCI Express Solutions (XAPP1052)* [32]. Ta referenčni sistem ponuja osnovno FPGA aplikacijo, namenjeno testiranju hitrosti prenosa podatkov preko FPGA vodila. Primer ponuja le osnovno logiko za prenos podatkov med FPGA vezjem in gostiteljem ter vsebuje sledeče glavne napake in pomanjkljivosti, ki smo jih morali odpraviti.

Obdelava in shranjevanje prejetih podatkov. Referenčni primer podatke, ki jih FPGA vezje prejme od gostitelja, takoj zavrže. Razlog za to je, da referenčni primer zanima le hitrost prenosa, ne vsebina podatkov. Izdelana platforma prejete podatke shrani v vrsto FIFO (First In First Out) [33], v kateri so podatki shranjeni, dokler jih ne zahtevajo ostali deli platforme. Za pravilno branje in shranjevanje podatkov smo morali v referenčno aplikacijo vključiti sistem za pravilno dekodiranje PCIe paketov.

Podatki, ki jih izdelana platforma sprejema, spadajo v tri različne kategorije. Prva kategorija so ukazi za branje in pisanje registrov, ki se obdelajo takoj in niso shranjeni v vrsto FIFO. Druga kategorija so podatki o konfiguraciji, ki so shranjeni v vrsti FIFO, dokler uporabnik ne sproži ukaza za začetek rekonfiguracije. Tretja kategorija so podatki, namenjeni obdelavi v rekonfigurabilnem delu sistema. Ti so lahko poljubne oblike in so shranjeni v vrsti FIFO, dokler jih ne zahteva rekonfigurabilni del.

Izdelana platforma za hranjenje podatkov o konfiguraciji in podatkov, ki jih želimo obdelati, uporablja le eno vrsto FIFO. Zato je potrebno pred začetkom rekonfiguracije vrsto povsem izprazniti, saj del za nadzor nad postopkom rekonfiguracije ne more razločiti vrste prejetih podatkov (ali gre za opis konfiguracije ali za podatke, ki jih mora obdelati rekonfigurabilni del). Tip podatkov, ki se nahajajo v vhodni FIFO vrsti, uporabnik določi s pisanjem v poseben register.

Vrsta FIFO, v kateri so shranjeni podatki, ima omejen prostor. Vsebuje lahko največ 1024 32-bitnih besed. Zato je pomembno, da FPGA vezje beleži preostali prostor v vrsti in gostitelju sporoči, ko je vrsta polna. V ta namen je uporabljen statusni register, v katerem določeni bit gostitelju pove, da je vrsta polna in da gostitelj FPGA vezju ne sme pošiljati novih podatkov. Gostitelj mora vsakič, ko pošlje podatke, najprej preveriti ta bit.

Pošiljanje podatkov iz FPGA vezja. Podobno kot pri prejemanju podatkov referenčni sistem pošilja le vnaprej določene podatke. Vsak poslani paket referenčnega sistema vsebuje vnaprej določeno zaporedje bajtov. Izdelana platforma mora namesto tega gostitelju pošiljati rezultate algoritma, ki se nahaja na rekonfigurabilnem delu. Podobno kot pri pisanju se izhodni podatki shranjujejo v vrsto FIFO, dokler jih ne zahteva gostitelj. Pri pošiljanju določeni bit statusnega registra go-

stitelju pove, če je izhodna vrsta FIFO prazna. V tem primeru gostitelj ne sme zahtevati izhodnih podatkov.

Izhodni podatki so le enega tipa. To so rezultati rekonfigurabilnega dela sistema. Pri izhodnih podatkih za razliko od vhodnih podatkov ni potrebno skrbeti, da so različne vrste podatkov ločene. Podobno kot pri prejemanju podatkov sistem tu beleži, če je izhodna vrsta FIFO prazna. V tem primeru gostitelj ne sme brati podatkov iz FPGA sistema.

Prekinitve. Referenčni sistem vsebuje osnovno implementacijo PCIe prekinitev. Ta implementacija ima napako, da ob koncu prenosa podatkov neprekinjeno pošilja prekinitve. Pravilna implementacija prekinitev, ki smo jo vključili v izdelano platformo, proži prekinitev le ob koncu prenosa in nato čaka na potrditev gostitelja, da je prekinitev sprejel in obdelal. Prekinitve platforma uporablja za sporočanje gostitelju, da je prenos podatkov končan.

Registri. Referenčni sistem ponuja veliko število registrov za nadzor nad prenosom podatkov in merjenjem hitrosti. Večina od njih je za našo platformo nepomembnih, zato smo za preprostost uporabe platforme skoraj povsem spremenili podane registre. Registri, ki jih uporablja platforma, so opisani v Dodatku A. Vsi registri so 32-bitni.

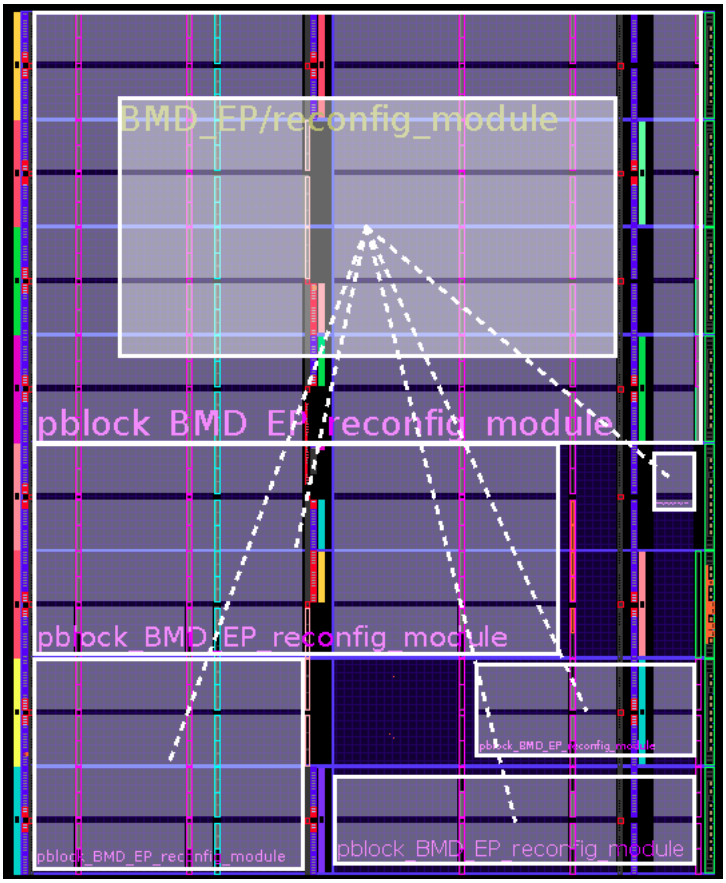
Glavni namen registrov izdelane platforme je nadzor nad procesom prenosa podatkov med gostiteljem in FPGA sistemom. Registri vsebujejo podatke o pomnilniškem naslovu in dolžini podatkov, ki jih uporabnik želi prenesti. Prav tako se prenos podatkov sproži s pisanjem v določeni register. Poleg registrov za nadzor prenosa podatkov izdelana platforma vsebuje tudi dva splošno namenska registra. Ta registra lahko uporablja rekonfigurabilni del. Eden od registrov je namenjen vhodnim nadzornim ukazom, drugi izhodnim statusnim ukazom. Pomen vsebine teh registrov določi rekonfigurabilna aplikacija. Primer je kodirnik/dekodirnik podatkov, kjer lahko bit 0 v vhodnem registru

pomeni, da želi uporabnik kodirati vhodne podatke, bit 1 pa, da želi vhodne podatke dekodirati.

4.5.2 Nadzor nad rekonfiguracijo

Nadzor nad rekonfiguracijo FPGA vezja je druga glavna naloga izdelane platforme. Modul za nadzor nad rekonfiguracijo omogoča uporabniku spreminjanje določenega dela FPGA vezja brez potrebe po zaustavitvi delovanja celotnega FPGA sistema.

Za pravilno delovanje rekonfiguracije je potrebno najprej določiti območja FPGA vezja, za katera želimo, da jih je možno spreminjati med izvajanjem. Te dele imenujemo rekonfigurabilne particije. Velikost in obliko rekonfigurabilnih particij je potrebno določiti vnaprej in jih ni možno spreminjati. Primer določitve particij z uporabo Xilinxovih programskih orodij je prikazan na sliki 4.2. Ker velikosti rekonfigurabilnega dela ni mogoče spreminjati v izdelanem sistemu, to območje zaseda čim večji prostor. Rekonfigurabilna particija zaseda ves prostor, ki ga ne uporablja statični deli platforme. Rekonfigurabilni del tako zaseda okoli 90 % celotne velikosti FPGA vezja, medtem ko je ostalih 10 % namenjenih statičnemu delu.



Slika 4.2: Izbira območja rekonfigurabilne particije v orodju PlanAhead. Beli deli predstavljajo območje rekonfigurabilne particije. V izdelani platformi rekonfigurabilne particija zavzema okoli 90 % celotne površine FPGA vezja. Neoznačeni modri deli predstavljajo statični del FPGA vezja. Lokacija statičnega dela je deloma določena že v Xilinxovem primeru [32], zaradi česar je statični del nekoliko razdrobljen.

Proces rekonfiguracije v času izvajanja poteka v več korakih. FPGA vezje najprej preko PCIe vodila prenese konfiguracijske podatke iz gostitelja. Ti konfiguracijski podatki so vsebovani v standardni konfiguracijski datoteki, ki se uporablja tudi za običajen proces programiranja FPGA vezja. Konfi-

guracijsko datoteko je možno ustvariti z Xilinxovim razvojnim orodjem PlanAhead, kot je opisano v viru [34]. Sama zgradba konfiguracijske datoteke je opisana v viru [35]. Po prenosu konfiguracijske datoteke modul za nadzor nad rekonfiguracijo naloži to konfiguracijo na vezje. Celoten postopek se izvede brez prekinitve izvajanja FPGA sistema.

Za nadzor nad rekonfiguracijo uporabljamo Xilinxov vmesnik ICAP (Internal Configuration Access Port) [35]. Xilinx ponuja dostop do vmesnika ICAP preko Verilog in VHDL kode. Verilog koda vmesnika, uporabljena v izdelani platformi, je prikazana na sliki 4.3.

```
ICAP_VIRTEX5 #(
    .ICAP_WIDTH()          // Širina vodila (X8,X16,X32)
) ICAP_VIRTEX5_inst (
    .CE(),                 // Clock Enable. 0 Vkljuci ICAP vmesnik
    .CLK(),                // Ura. Do 100Hz
    .I(),                  // Vhodni podatki
    .O(),                  // Izhodni podatki
    .WRITE()               // 0, ko pišemo podatke
);
```

Slika 4.3: Verilog koda za dostop do vmesnika ICAP

Kot je razvidno iz slike, je dostop do vmesnika ICAP precej enostaven. Najprej je potrebno določiti širino izhodnega in vhodnega vodila (ICAP podpira tri možne širine: 8, 16 ali 32 bitov). Širina vodila določa, koliko bitov se bo zapisalo v ICAP vmesnik vsako urino periodo. Večja širina zato pomeni hitrejši prenos podatkov in s tem krajši čas rekonfiguracije. V izdelani platformi smo uporabili največjo možno širino 32 bitov, ki je tudi enaka kot širina podatkov na PCIe vodilu. Po določitvi širine vodil je potrebno vmesniku ICAP samo še podati ustrezno konfiguracijske podatke za sprožitev rekonfiguracije. Podani konfiguracijski podatki, ki jih ustvari Xilinxovo orodje

PlanAhead, tako vsebujejo vse potrebne informacije za uspešno izvedbo rekonfiguracije. Algoritem 1 prikazuje osnoven postopek rekonfiguracije preko vmesnika ICAP.

Algorithm 1 Osnoven postopek pisanja konfiguracije z vmesnikom ICAP

1: $ICAP\_WRITE \leftarrow 0$ (Način pisanja)
2: $CE \leftarrow 0$(vključi ICAP)
3: Vsako periodo ure vpiši 32 bitov podatkov na vhod I
4: Ponovi korak 3, dokler ni zapisana celotna konfiguracijska datoteka.
5: $CE \leftarrow 1$ (onemogoči ICAP).
6: $ICAP\_Write \leftarrow 1$ (prenehaj pisati).

Osnovni algoritem 1 smo morali za pravilno delovanje platforme nadgraditi tako, da je rešil sledeče probleme.

Možnost začasne prekinitve pisanja. Modul za nadzor nad procesom rekonfiguracije služi za to, da prenese konfiguracijske podatke iz PCIe modula v vmesnik ICAP. PCIe modul konfiguracijske podatke hrani v vrsti FIFO, ki ima omejeno kapaciteto podatkov. Kapaciteta FIFO vrste je precej manjša od celotne konfiguracijske datoteke (FIFO vrsta ima kapaciteto 32768 bitov, medtem ko so konfiguracijske datoteke lahko velike tudi okoli 4 megabajte, odvisno od velikosti rekonfigurabilne particije). Zaradi tega ni možno na FPGA vezje prenesti celotne konfiguracije in šele potem začeti s postopkom rekonfiguracije. Namesto tega mora gostitelj med procesom rekonfiguracije konfiguracijsko datoteko pošiljati v delih.

Med tem prenosom se zaradi razlik v hitrosti prenosa podatkov med PCIe vodilom in internim prenosom v ICAP vmesnik pogosto zgodi, da bo vhodna FIFO vrsta prazna (torej je prenos v vmesnik ICAP hitrejši kot prenos preko PCIe vodila). V takem primeru je nujno potrebno začasno zaustaviti proces rekonfiguracije, saj bi vmesnik ICAP

drugače vsako urino periodo pričakoval nove podatke. Zaustavitev procesa rekonfiguracije se izvede z zapisom vrednosti 1 v ICAP vhod CE (clock enable).

Pravilno urejanje vhodnih podatkov. ICAP pričakuje bite podatkov, urejene v nekoliko drugačnem zaporedju, kot so urejeni v konfiguracijski datoteki. Pred pisanjem podatkov v vmesnik ICAP jih je potrebno pravilno preurediti. V vsakem bajtu posebej je potrebno obrniti vrstni red bitov. V bajtu 0 so tako biti obrnjeni na sledeči način: $0 <-> 7, 1 <-> 6, 2 <-> 5, 4 <-> 3$. Enak postopek je potreben tudi v preostalih 3 bajtih.

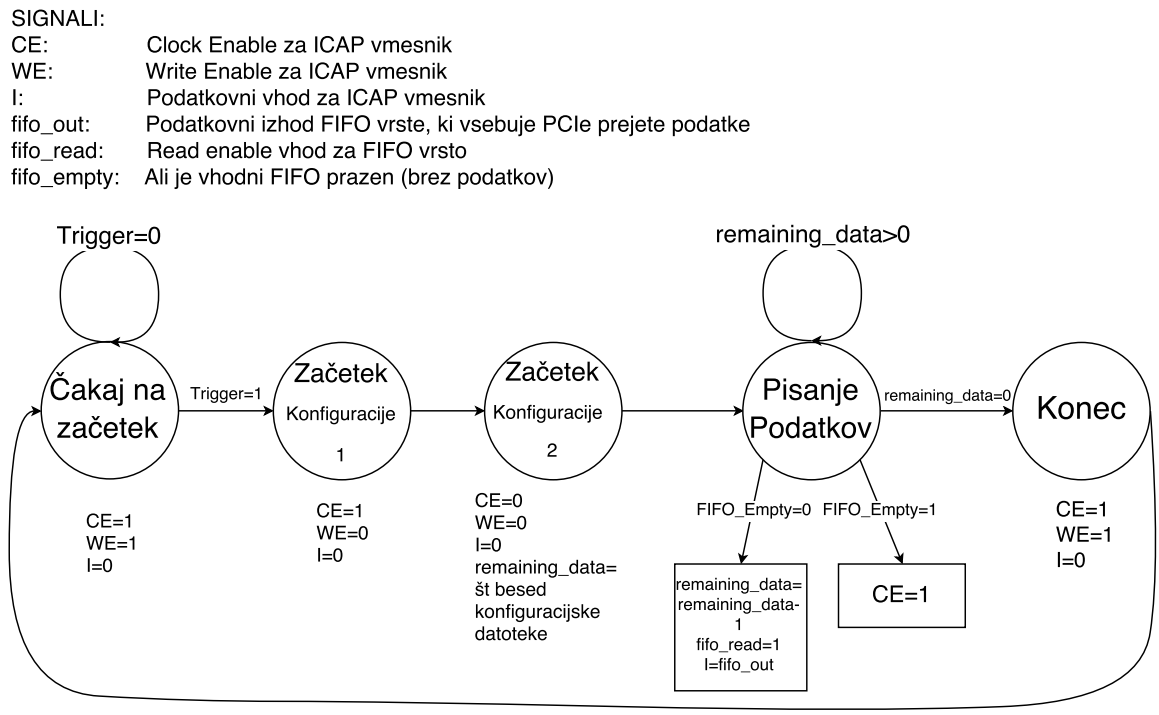
Beleženje dolžine preostalih podatkov. Vmesnik ICAP ne ponuja nobenih informacij o tem, kdaj je proces rekonfiguracije končan. Zato je potrebno ročno beleženje števila preostalih podatkov. Da FPGA vezje ve dolžino celotne konfiguracijske datoteke, uporabnik pred začetkom rekonfiguracije v določen register ročno vpiše dolžino konfiguracijske datoteke.

Celoten algoritem za nadzor nad rekonfiguracijo, ki upošteva zgoraj opisane dodatke, je predstavljen v algoritmu 2.

Algorithm 2 Celoten postopek pisanja konfiguracije z vmesnikom ICAP

1: len ← Dolžina celotne konfiguracijske datoteke
2: $ICAP\_WRITE \leftarrow 0$
3: $CE \leftarrow 0$ (vključi ICAP)
4: **while** $len \geq 0$ **do**
5: **if** fifo_empty **then**
6: $CE \leftarrow 1$
7: **else**
8: $CE \leftarrow 0$
9: $word\_to\_write \leftarrow flip\_bits(FIFO[0])$ (Obrni bite v FIFO)
10: $I \leftarrow word\_to\_write$ (Zapiši obrnjeno prvo besedo iz FIFO v ICAP)
11: $len \leftarrow len - 1$
12: **end if**
13: **end while**
14: $CE \leftarrow 1$
15: $ICAP \leftarrow 1$

Slika 4.4 prikazuje diagram prehajanja stanj avtomata, ki implementira ta algoritem. Verilog koda avtomata je predstavljena v Dodatku B



Slika 4.4: Diagram prehajanja stanj avtomata, ki nadzira rekonfiguracijo z uporabo vmesnika ICAP.

Med procesom rekonfiguracije FPGA logika, zajeta v rekonfigurabilni particiji, ne bo delovalo pravilno. Zato uporabnik med potekom rekonfiguracije z branjem podatkov iz rekonfigurabilnega dela ne bo dobil pravilnih rezultatov. Po končanem postopku rekonfiguracije je zelo priporočeno, da se pobriše vsebina izhodne podatkovne vrste, saj podatki v njej ne bodo pravilni. Ker platforma uporablja isto vhodno FIFO vrsto za vnos konfiguracijske datoteke

in podatkov, uporabnik med procesom rekonfiguracije v podatkovno vrsto ne sme pisati podatkov, ki niso namenjeni postopku rekonfiguracije.

4.5.3 Komunikacija med statičnim in rekonfigurabilnim delom FPGA vezja

Za možnost pravilne komunikacije med rekonfigurabilnim delom in preostankom FPGA vezja je potrebno, da ima vsaka konfiguracija, ki jo naložimo v rekonfigurabilno particijo, enak vmesnik, torej enake vhodne in izhodne signale. Seznam in kratek opis signalov rekonfigurabilne particije v izdelani platformi je prikazan na sliki 4.5.

```
reconfigurable_module(
.clk(),                  //vhod, ura
.rst_i(),                //vhod, reset
.data_in(),              //vhod, vhodni podatki (32 bitov)
.data_in_available(),    //vhod, statični del ima vhodne podatke
.data_in_rdy(),          //izhod, rekonfigurabilni del
                         //želi brati vhodne podatke
.data_out(),             //izhod, izhodni podatki (32 bitov)
.data_out_available(),   //izhod, rekonfigurabilni
                         //del ima izhodne podatke
.data_out_rdy(),         //vhod, statični del lahko
                         //bere izhodne podatke
.cmd_in(),               //vhod, razni kontrolni podatki
                         //(32 bitov)
.cmd_write_in(),         //vhod, statični del želi
                         //pisati kontrolne podatke
.status_o()              //izhod, razni statusni podatki
                         //(32 bitov)
);
```

Slika 4.5: Vhodni in izhodni signali rekonfigurabilnega dela. Vsako vezje, ki ga želimo uporabljati v rekongifurabilnem delu, mora uporabljati te signale

Glavna naloga takšnega vmesnika je sinhronizacija pretoka podatkov med statičnim in rekonfigurabilnim delom. Statični del vsebuje dve vrsti FIFO, v katerih hrani podatke. V eni vrsti hrani vhodne podatke, ki jih je prejel od gostitelja in jih mora prenesti v rekonfigurabilni del. V drugi vrsti hrani izhodne podatke. Ti predstavljajo rezultate obdelanih vhodnih podatkov. Statični del jih prejme od rekonfigurabilnega dela in jih mora prenesti gostitelju. Prav tako rekonfigurabilni del verjetno vsebuje dve vrsti, v katerih hrani vhodne in izhodne podatke.

Za vse vrste velja, da imajo omejeno kapaciteto vsebovanih podatkov in da so lahko prazne. Statični del ne sme pisati v rekonfigurabilni del, če je njegova vhodna vrsta polna, in iz njega ne sme brati, če je njegova izhodna vrsta prazna. Enako velja tudi za rekonfigurabilni del. Signali rekonfigurabilnega dela zato poleg podatkovnih signalov *data_in* in *data_out* vsebujejo tudi signale o statusu vhodnih in izhodnih vrst *data_in_rdy*, *data_out_rdy*, *data_in_available* in *data_out_available*.

Zadnja dva signala *cmd_in* in *status_o* FPGA sistem uporabi za izmenjavo raznih statusnih in kontrolnih sporočil, ločeno od glavnih podatkov. *cmd_in* predstavlja nadzorne ukaze v rekonfigurabilni del, medtem ko *status_o* predstavlja statusne informacije iz rekonfigurabilnega dela. Signala sta namenjena splošnim dodatnim informacijam, ki jih bo rekonfigurabilni del morda potreboval. Dejanski pomen teh dveh signalov je poljubno določen v rekonfigurabilnem delu. Signala se preslikata v PCIe registra. Tako ju je možno enostavno spreminjati in brati iz gostitelja.

4.6 Gostitelj

Gostiteljev del platforme se deli na dva dela. Prvi del predstavlja PCIe gonilnik, ki omogoča prenos podatkov med gostiteljem in FPGA sistemom. Drugi del predstavljajo aplikacije za nadzor platforme. S temi aplikacijami lahko uporabnik upravlja z rekonfiguracijo na FPGA sistemu, ter prenaša

podatke iz gostitelja na vezje in obratno.

4.6.1 Gonilnik

Gonilnik izdelane platforme omogoča komunikacijo med FPGA napravo in jedrom operacijskega sistema gostitelja. V izdelani platformi smo za operacijski sistem izbrali Ubuntu 14.10. Izdelani gonilnik omogoča upravljanje s FPGA sistemom preko programskega jezika C. Glavni cilj gonilnika je uporabniku omogočiti nadzor nad FPGA sistemom. Tako mora gonilnik podpirati prenos podatkov iz in na FPGA vezje ter branje in pisanje v registre FPGA naprave. Poleg tega mora gonilnik skrbeti za spremljanje in procesiranje prekinitev FPGA naprave. Uporabnik bo sicer sistem dejansko nadzoroval preko uporabniških aplikacij, ki so opisane v poglavju 4.6.2, vendar bodo te aplikacije komunikacijo s FPGA napravo izvajale preko gonilnika.

Tako kot PCIe modul na vezju FPGA je tudi gonilnik zasnovan po kodi iz Xilinxovega referenčnega primera XAPP1052 [32]. Gonilnik iz referenčnega primera je bil razvit za operacijski sistem Fedora 10, ne za Ubuntu. Za prilagoditev na ciljni operacijski sistem in za dodatne lastnosti so bile potrebne precejšnje spremembe gonilnika iz primera. Te spremembe in dodane lastnosti so:

Kompatibilnost z operacijskim sistemom Ubuntu 14.10. Referenčni gonilnik je bil razvit za operacijski sistem Fedora 10. Gonilnik je bilo potrebno posodobiti na sodobno verzijo Linux jedra, saj je operacijski sistem Fedora 10 izšel leta 2008. Zaradi tega nekatere funkcije, uporabljene v gonilniku, niso delovale pravilno ali pa so bile odstranjene iz novejših verzij jedra.

Obravnavanje prekinitev. Kot opisano v implementacij PCIe vodila referenčni FPGA sistem ni pravilno prožil prekinitev. Prav tako gonilnik referenčnega sistema nikoli ni sporočil FPGA sistemu, da je prekinitev

obdelal. Gonilnik smo popravili tako, da je obdelavo prekinitve sporočil FPGA sistemu s pisanjem v določen bit kontrolnega registra.

Posodobitev registrov. Podobno kot pri referenčni FPGA napravi je tudi gonilnik referenčnega sistema vseboval veliko število nepotrebnih registrov. Gonilnik smo spremenili tako, da podpira spremenjene registre FPGA naprave.

4.6.2 Aplikacije za nadzor platforme

Za lažjo uporabo platforme smo razvili uporabniške aplikacije, s katerimi lahko uporabnik preko ukazne vrstice enostavno krmili celotno platformo. Aplikacije omogočajo prenos podatkov na in iz FPGA vezja ter nadzor nad rekonfiguracijo platforme. Aplikacijo za pisanje podatkov uporabnik uporabi tako, da le poda ime datoteke, ki jo želi prenesti na FPGA vezje. Aplikacija za branje podatkov vse podatke v FPGA vezju izpiše na standardni izhod ukaznega vmesnika. Za sprožitev rekonfiguracije uporabnik najprej na FPGA vezje prenese konfiguracijsko datoteko. Nato uporabnik kliče aplikacijo za začetek rekonfiguracije.

Prenos podatkov aplikaciji za branje in pisanje izvedeta z ustreznim pisanjem v PCIe registre FPGA naprave. V registre je potrebno zapisati začetni naslov podatkov, velikost posameznega PCIe paketa in dolžino podatkov, izraženo v številu PCIe paketov. Po vnosu podatkov aplikacija nastavi bit v nadzornem registru FPGA naprave, ki napravi pove, da naj začne s prenosom podatkov. Dejanski prenos podatkov v celoti izvede FPGA naprava s pošiljanjem ustreznih PCIe paketov. Pri prenosu podatkov aplikaciji uporabljata pakete dolžine 4 bajtov in v posameznem prenosu pošljeta po 32 paketov. V vsakem prenosu je torej prenesenih 128 bajtov podatkov. Po vsakem prenosu 32 paketov aplikacija za pisanje preveri preko PCIe registra, ali je vhodna FIFO vrsta FPGA naprave polna. Če je, aplikacija čaka, dokler se ne izprazni. Psevdokoda aplikacije za pisanje podatkov je predstavljena v

Algoritmu 3.

Algorithm 3 Psevdokoda algoritma za pisanje podatkov v FPGA naprave

```
Remaining_Data ← Velikost podatkov, v bajtih
while Remaining_Data > 0 do
    Kopiraj podatke največ dolžine 128 bajtov iz podatkovne datoteke v DMA medpomnilnik
    Remaining_Data ← Remaining_Data - 128 {Vsak prenos prenese 128 bajtov}
    V PCIe Register 9 (Read Packet Count) zapiši število kopiranih besed

    Preveri, ali je vhodni FIFO poln: FIFO_Full ← Bit 3 PCIe registra 0
    while FIFO_Full do
        Čakaj, dokler FIFO ni več poln: FIFO_Full ← Bit 3 PCIe registra 0
    end while
    Začni Prenos. V Bit 15 registra 1 zapiši vrednost 1
end while
```

Aplikacija za branje podatkov deluje podobno kot aplikacija za pisanje podatkov. Aplikacija mora pred branjem podatkov preveriti, ali izhodna FIFO vrsta na FPGA sistemu vsebuje podatke. Če jih ne, potem aplikacija ne sme poslati zahtev po branju podatkov, ampak mora čakati, dokler podatki niso na voljo. Kot pri pisanju aplikacija bere po 128 bajtov na enkrat, zato mora izhodna FIFO vrsta vsebovati vsaj 128 bajtov podatkov, da je branje možno. Psevdokoda aplikacije za branje podatkov je predstavljena v algoritmu 4.

Algorithm 4 Psevdokoda algoritma za branje podatkov iz FPGA naprave

```
1: Remaining_Data ← Velikost podatkov, ki jih želimo prebrati, v bajtih
2: while Remaining_Data > 0 do
3:    Remaining_Data ← Remaining_Data - 128 {Vsak prenos prenese 128
      bajtov}
4:    V PCIe Register 4 (Write Packet Count) zapiši število kopiranih besed

5:    Preveri, ali ima izhodni FIFO vsaj 128 bajtov podatkov: FIFO_Empty
      ← Bit 5 PCIe registra 0
6:    while FIFO_Full do
7:       Čakaj, dokler FIFO ne dobi dovolj podatkov: FIFO_Empty ← Bit
         5 PCIe registra 0
8:    end while
9:    Začni Prenos. V Bit 0 registra 1 zapiši vrednost 1
10: end while
```

Poglavje 5

Rezultati

V tem poglavju bomo predstavili zmogljivosti in rezultate izdelane platforme. Najprej bomo predstavili velikost, ki jo platforma zasede na FPGA vezju in hitrost prenosa podatkov. Nato bomo predstavili testne rekonfigurabilne aplikacije, s katerimi smo preverili pravilno delovanje sistema. Predstavili bomo tudi, kakšne so strojne in programske zahteve izdelanega sistema in kaj je potrebno, če želimo izdelano platformo prilagoditi za FPGA vezja, ki se razlikujejo od tistega, na katerem smo platformo razvili. Na koncu bomo razvito platformo primerjali z obstoječimi rešitvami za rekonfiguracijo v času izvajanja.

5.1 Prostorske zahteve

Zaželeno je, da statični del platforme potrebuje čim manj prostora na FPGA vezju. Čim manjši je statični del, tem več prostora je na voljo za rekonfigurabilni del, kar pomeni podporo za večje uporabniške aplikacije na rekonfigurabilnem delu.

Statični del za delovanje potrebuje okoli 4000 vpoglednih tabel (ang. look-up table oziroma LUT) in flip-flopov. Testno vezje Virtex-5 XUPV5-LX110T vsebuje skupno 69120 vpoglednih tabel in flip flopov. Za implemen-

tacijo FIFO vrst statični del potrebuje 6 blokovnih pomnilniških elementov (BRAM) od skupno 148. Tako statični del zasede okoli 5 % vezja, rekonfigurabile aplikacije pa razpolagajo s preostalimi 95 %.

Xilinxova vezja serije Virtex-5, ki podpirajo povezljivost z vodilom PCIe, vsebujejo od okoli 12000 do 200000 vpoglednih tabel ter flip-flopov in od 26 do 324 blokovnih pomnilniških enot [36]. Tako bi na najmanjšem Virtex-5 vezju statični del zasedel približno 25 % celotnega vezja, na največjem pa 2 %.

Izdelana platforma za prenos podatkov uporablja en pas PCIe vodila, saj uporabljeni FPGA sistem fizično ne podpira več pasov.

Podroben opis porabljenih komponent je prikazan v tabeli 5.1. V tabeli 5.2 je prikazana poraba referenčnega primera XAPP1052 [32], na katerem je bil zasnovan modul za prenos podatkov preko PCIe vodila.

Komponenta	Poraba	Poraba (%)	Število Vseh Komponent
LUT	3910	6%	69120
Flip Flop	4175	6%	69120
Block RAM (36kb)	8	5%	148
PCIe pasovi	1	100%	

Tabela 5.1: Prostorska poraba statičnega dela izdelanega sistema. Statični del porabi okoli 5 % vseh komponent FPGA vezja. Tako je preostalih 95 % vezja na voljo rekonfigurabilnemu delu

Komponenta	Poraba	Poraba (%)	Število Vseh Komponent
LUT	3557	5%	69120
Flip Flop	3706	5%	69120
Block RAM (36kb)	6	5%	148
PCIe pasovi	1	100%	

Tabela 5.2: Prostorska poraba le referenčnega primera, na katerem je zasnovan modul za prenos podatkov preko PCIe vodila [32]. V primerjavi s celotno porabo v tabeli 5.1 je razvidno, da ta del zavzame veliko večino celotne platforme.

Tabela 5.3 predstavlja porabo zgolj tistih delov sistema, ki niso bili vključeni v referenčnem primeru. Sem spada predvsem modul za nadzor nad rekonfiguracijo. Rekonfigurabilni del potrebuje 353 komponent LUT in 469 flip-flopov. Primerljiv Xilinxov modul za nadzor nad rekonfiguracijo [37] potrebuje okoli 1000 LUT in 1100 flip-flop komponent. Torej je razviti modul za nadzor nad rekonfiguracijo prostorsko učinkovit.

Komponenta	Poraba	Poraba (%)	Število Vseh Komponent
LUT	353	0.5%	69120
Flip Flop	469	0.6%	69120
Block RAM (36kb)	2	1.3%	148

Tabela 5.3: Poraba le delov sistema, ki niso bili vključeni v referenčnem primeru [32]. To predstavlja modul za nadzor nad rekonfiguracijo in popravke ter nadgradnje referenčnega primera. Modul za nadzor nad rekonfiguracijo je torej izjemno majhen.

5.2 Hitrost prenosa podatkov

Preverili smo tudi hitrost prenosa podatkov preko vodila PCIe. Hitrost smo preverili tako, da smo iz gostitelja na FPGA sistem prenesli določeno število podatkov in izmerili pretečeni čas od začetka pisanja do prejema prekinitve, ki označuje konec prenosa. Pri prenosu smo uporabljali pakete dolžine 4 bajte in v vsakem prenosu prenesli 32 paketov. Takšen prenos smo ponovili 100000-krat in izračunali povprečno hitrost prenosa podatkov. Izmerjena hitrost prenosa podatkov je bila 12,19 MB/s.

Dosežena hitrost je precej nižja od najvišje možne hitrosti, ki jo ponuja uporabljeni PCIe standard 1.1 pri enopasovni povezavi (250 MB/s). Razlog za nižjo hitrost je najverjetneje majhna velikost paketov, saj sistem trenutno ne podpira dekodiranja daljših paketov. Manjši paketi namreč povečajo dodatno količino dejansko prenesenih podatkov, saj vsak paket poleg samih podatkov vsebuje še dodatne informacije, kot sta naslov in dolžina paketa. Dosežena hitrost je primerljiva s podobnimi sistemi, kot je [38], ko pošiljajo majhno število podatkov.

Hitrost procesa rekonfiguracije je vezana na vmesnik ICAP. Vmesnik vsako urino periodo prebere 32 bitov podatkov. Z urino frekvenco 62.5 MHz to pomeni hitrost prenosa podatkov 250 MB/s. Ker so delne rekonfiguracijske datoteke velike okoli 400 kB to pomeni, da se rekonfiguracija izvede v 1,6 milisekundah. Ker je hitrost prenosa podatkov preko vodila PCIe počasnejša kot hitrost prenosa v vmesnik ICAP, je v praksi hitrost rekonfiguracije omejena s hitrostjo prenosa konfiguracijske datoteke iz gostitelja na FPGA sistem. Pri prenosu s hitrostjo 12,19 MB/s se rekonfiguracija izvede v približno 32,8 milisekundah.

5.3 Testne rekonfigurabilne aplikacije

Delovanje razvite platforme smo preverili na različnih testnih FPGA aplikacijah. Testiranje procesa rekonfiguracije je potekalo tako, da je rekonfigurabilni FPGA sistem začel brez naložene rekonfigurabilne aplikacije. Nato je uporabnik preko ukazne vrstice na sistem zaporedoma naložil različne testne rekonfigurabilne aplikacije in s prenosom podatkov preveril, da te delujejo pravilno.

Delovanje platforme smo preverili s tremi testnimi aplikacijami. Prvi dve sta bili preprosti aritmetični aplikaciji, ki sta prejetim podatkom le prišteli ali odšteli 1. Verilog koda testne aritmetične aplikacije je prikazana v sliki 5.1.

```
module reconfig_module(clk,
                       data_in,
                       data_in_available,
                       data_in_rdy,
                       data_out,
                       data_out_available,
                       data_out_rdy);
    input clk;
    input [31:0]data_in;
    input data_in_available;
    output data_in_rdy;
    output [31:0] data_out;
    output data_out_available;
    input data_out_rdy;

    wire data_in_rdy;
    wire data_out_available;
    wire [31:0] data_out;
    assign data_in_rdy = data_in_available;
    assign data_out_available =
        data_out_rdy & data_in_available;
    assign data_out=data_in + 1;
endmodule
```

Slika 5.1: Verilog koda enostavne aritmetične testne aplikacije. Aplikacija vhodnim podatkom prišteje 1 in jih poda na izhod. Aplikacija nenehno bere podatke in jih takoj zapiše na izhod

Po osnovnem preizkusu smo platformo preverili tudi na bolj kompleksnem primeru. Preizkusili smo FPGA implementacijo kodirnega algoritma AES [39]. Uporabljeno jedro je zasnovano po [40, 41]. Uporabljeno jedro je poleg

samih podatkov potrebovalo tudi dodatne informacije, na primer, ali vhodni podatki predstavljajo kodirni ključ ali podatke za kodiranje. Dodatne informacije, ki jih algoritem potrebuje, lahko uporabnik poda s pisanjem v namenski PCIe register. Tudi AES algoritem je na izdelani platformi deloval pravilno.

5.4 Zahteve za uporabo platforme

Rekonfigurabilna platforma, opisana v magistrski nalogi, je bila izdelana in preizkušena na FPGA sistemu Xilinx XUPV5-LX110T [30]. V tem poglavju bomo opisali zahteve, ki jih mora podpirati izbrani FPGA sistem, da na njem lahko deluje izdelana platforma.

Pri izdelavi sistema smo uporabili sledeče obstoječe komponente in orodja:

ICAP. Xilinxov vmesnik za nadzor nad konfiguracijo. Vmesnik ICAP je prisoten na vseh sodobnih Xilinxovih FPGA vezjih. Dostop do vmesnika ICAP je pri večini Xilinxovih vezjih zelo podoben. ICAP se na različnih vezjih razlikuje le po imenih vhodnih in izhodnih signalov in po širini podatkovnih signalov. Zato modul za rekonfiguracijo brez velikih sprememb lahko deluje na vseh sodobnih Xilinxovih FPGA vezjih.

PCIe vmesnik. FPGA sistem, ki želi uporabljati izdelano platformo nujno potrebuje povezljivost s PCIe vmesnikom. Izdelana platforma serije Virtex-5 za implementacijo PCIe vmesnika uporablja Xilinxov referenčni primer XAPP 1052. Referenčni primer poleg serije Virtex-5 podpira serije Virtex-6, Kintex-7, Spartan-6 in Spartan-3. Tako bi PCIe vmesnik brez velikih sprememb lahko deloval na vseh napravah teh serij.

Razvojno orodje PlanAhead. Orodje PlanAhead je uporabljeno za definicijo rekonfigurabilnih območij in izdelavo ustreznih konfiguracijskih

datotek. Uporaba platforme je torej vezana na Xilinxova razvojna orodja in njihove licence.

Gonilnik in aplikacije. Gonilnik za FPGA napravo je bil razvit za operacijski sistem Ubuntu 14.10. Predvidevamo, da bi deloval tudi na ostalih sodobnih Linux distribucijah. Same aplikacije za nadzor nad platformo ne uporabljajo nobenih zunanjih knjižnic ali specifičnih sistemskih klicev in bi delovale na vseh platformah, na katerih bi deloval gonilnik.

Izdelana platforma je bila preizkušena na ciljnem FPGA sistemu XUPV5-LX110T iz serije vezij Virtex-5. Težave za uporabo platforme na ostalih FPGA sistemih predstavlja predvsem modul za nadzor nad prenosom podatkov preko PCIe vodila. Ta modul namreč uporablja Xilinxovo jedro, ki je na voljo le na FPGA vezjih serije Virtex-5, Virtex-6, Kintex-7, Spartan-6 in Spartan-3. Modul za nadzor nad rekonfiguracijo je lažje prenosljiv na druge FPGA sisteme. Modul je vezan le na Xilinxov vmesnik ICAP, ki je z majhnimi razlikami, kot so različna širina vhodnih podatkov ali imena vhodov, prisoten v vseh Xilinxov sodobnih FPGA vezjih.

Kot je opisano v poglavju 5.1, prostorska poraba izdelane platforme na FPGA vezju ni velik problem. Vsa sodobna Xilinxova vezja, ki podpirajo vodilo PCIe, so dovolj velika za izdelano platformo. Izdelana platforma potrebuje za delovanje 2 W električne moči, kar ne predstavlja velikih omejitev. Na primer, napajalnik, priložen testni platformi XUPV5-LX110t, ponuja 30 W električne moči.

5.5 Primerjava z obstoječimi rešitvami

V tem poglavju bomo predstavili tri obstoječe rešitve, ki ponujajo podobno funkcionalnost kot izdelana platforma in predstavili njihove prednosti in slabosti v primerjavi z našo platformo.

Prva podobna rešitev je RIFFA [42]. RIFFA podobno kot naša platforma ponuja povezljivost preko vodila PCIe s priloženimi gonilniki in knjižnicami za branje in pisanje podatkov iz FPGA sistema. Prednost te rešitve je njena splošnost, saj je rešitev na voljo za operacijska sistema Linux in Windows, kot tudi za veliko različnih FPGA vezij dveh različnih proizvajalcev (Altera in Xilinx). Naša rešitev je omejena zgolj na Linux in Xilinxova FPGA vezja. Glavna slabost te rešitve je, da ne ponuja možnosti za rekonfiguracijo.

Druga podobna rešitev je predstavljena v članku [14]. Ta rešitev ponuja FPGA modul za nadzor nad rekonfiguracijo, ki ga je možno krmiliti preko namiznega računalnika, vendar povezavo med računalnikom in FPGA sistemom implementira preko vmesnika UART. UART je znatno počasnejši od vodila PCIe in je v sistemu uporabljen zgolj za prenos ukazov in konfiguracijskih datotek. Dejanskih podatkov iz računalnika ni možno prenašati na FPGA vezje.

Tretja rešitev DyRACT [38] je najbolj primerljiva z našo platformo. DyRACT ponuja tako prenos podatkov preko vodila PCIe kot nadzor nad rekonfiguracijo FPGA vezja. Od naše platforme se razlikuje v sledečih lastnostih. DyRACT podpira zgolj Xilinxova vezja od serije Virtex-6 naprej, medtem ko je naša platforma osredotočena na serijo Virtex-5. Prednost naše platforme je njena preprostost in s tem manjša prostorska poraba. DyRACT za implementacijo na FPGA vezju potrebuje okoli 6600 komponent LUT in 26 komponent BRAM, medtem ko naša platforma potrebuje le 4000 komponent LUT in le 8 komponent BRAM. Prav manjša poraba komponent BRAM je pomembna, saj manjša Xilinxova vezja vsebujejo le okoli 25 teh komponent [36, 43], kar je premalo za uporabo rešitve DyRACT. Prednost rešitve DyRACT je boljša implementacija PCIe prenosa, ki omogoča prenose podatkov s hitrostjo 1542 MB/s pri štiripasovni PCIe povezavi.

5.6 Nadaljnje delo

Trenutna izdelana platforma ponuja povsem delujoče osnovne funkcionalnosti, opisane v poglavju 4.1. Glavna trenutna slabost platforme je hitrost prenosa podatkov preko vodila PCIe, kot je opisano v poglavju 5.

Hitrost prenosa bi bilo možno izboljšati na dva načina. Platforma trenutno podpira le najmanjše PCIe pakete, ki prenašajo po eno besedo podatkov na paket. To omogoča enostavno obdelavo in dekodiranje paketov, saj so podatki vedno enake dolžine. Paketi z večjim številom podatkov bi bili bolj kompleksni za obdelavo, vendar bi povečali hitrost prenosa podatkov, saj bi bilo za enako število podatkov potrebno manjše število paketov.

Drugi način povečanja prenosa podatkov je uporaba večpasovnega prenosa preko PCIe vodila. Trenutno platforma uporablja le en PCIe pas. Uporaba več pasov bi močno povečala hitrost prenosa. Z uporabo osmih pasov bi bilo možno hkrati pošiljati in brati osem podatkovnih paketov. Večpasovne povezave zaradi omejitev FPGA sistema, na katerem smo implementirali platformo, nismo mogli uporabiti. Izbrani FPGA sistem namreč podpira le enopasovno povezavo.

Poglavje 6

Zaključek

Rekonfiguracija v času izvajanja omogoča spreminjanje programske logike na FPGA vezju med brez zaustavitve celotnega sistema. V magistrski nalogi smo opisali izdelano platformo za nadzor nad rekonfiguracijo v času izvajanja. Cilj platforme je razvijalcem ponuditi splošno ogrodje, ki ponuja rešitve za probleme, skupne velikemu številu aplikacij, ki uporabljajo rekonfiguracijo v času izvajanja.

Platforma ponuja dve glavni lastnosti. Prva lastnost je prenos podatkov med FPGA vezjem in gostiteljem, druga lastnost pa je nadzor nad izvajanjem rekonfiguracije. Platforma ponuja tudi aplikacije za ukazno vrstico, preko katerih uporabniki krmilijo prenos podatkov in rekonfiguracijo. Za povezavo FPGA vezja z gostiteljem smo razvili tudi gonilnik. Tako gonilnik kot uporabniške aplikacije delujejo na operacijskem sistemu Ubuntu 14.10.

Razvito platformo smo preizkusili na dveh enostavnih testnih FPGA aplikacijah (seštevalniku in odštevalniku) in na bolj zapletenem primeru AES kodiranja. V vseh treh primerih je platforma delovala pravilno. V primerjavi z obstoječimi rešitvami platforma ponuja primerljive zmogljivosti ob manjši prostorski porabi, vendar z nižjo hitrostjo prenosa podatkov preko PCIe vodila.

Dodatek A

Registri izdelanega FPGA sistema

V tem dodatku so predstavljeni glavni registri izdelanega FPGA sistema. Vsi registri so 32-bitni. Besedi Write za pisanje in Read za branje sta pri imenih registrov uporabljeni iz stališča FPGA sistema. Registri Read se nanašajo na prenos podatkov iz gostitelja na FPGA sistem, medtem ko se registri Write nanašajo na prenos iz FPGA sistema na gostitelja.

Register 0: Statusni/Kontrolni Register. Statusni del registra vsebuje razne podatke o sistemu, medtem ko lahko s pisanjem v bite kontrolnega dela registra nadzorujemo delovanje sistema. Uporabljeni biti registra so:

Bit 0: Reset. S pisanjem vrednosti 1 v ta register ponastavimo sistem na njegove privzete vrednosti.

Bit 1: Interrupt Done. S pisanjem vrednosti 1 v ta register gostitelj FPGA vezju sporoči, da je prejelo in obdelalo njegovo prekinitev.

Bit 2: Začni rekonfiguracijo. S pisanjem v ta bit gostitelj FPGA vezju sporoči, da naj začne postopek rekonfiguracije. FPGA vezje

bo naložilo prejeto konfiguracijsko datoteko v rekonfigurabilni del vezja.

Bit 3: FIFO Full. Ta bit gostitelju sporoči, da je vhodna FIFO vrsta polna in da gostitelj vanjo ne more pisati podatkov. Ker zaradi večje hitrosti gostitelj pošilja po 32 32-bitnih besed na enkrat, ima ta bit vrednost 1, ko je v vhodni vrsti FIFO manj kot 32 prostih besed.

Bit 4: FIFO reset. S pisanjem vrednosti 1 v ta bit gostitelj zbriše podatke v vhodni in izhodni FIFO vrsti.

Bit 5: Out FIFO Empty. Ko je vrednost tega bita 1, je izhodna FIFO vrsta prazna. Ta vrsta vsebuje rezultate, dobljene iz rekonfigurabilnega dela. Ko je vrsta prazna, gostitelj z branjem iz nje ne bo dobil smiselnih rezultatov.

Register 1: Statusni/Kontrolni Register za DMA prenos. Statusni del registra poda informacije o tem, ali se je DMA prenos pravilno zaključil. To je namenjeno le preverjanju pravilnega delovanja sistema, saj glavni način sporočanja končanja prenosa uporablja prekinitve. Kontrolni del registra omogoča proženje prenosa podatkov in onemogočanja proženja prekinitev. Uporabljeni biti tega registra so:

Bit 0: Start Write. S pisanjem vrednosti 1 v ta bit se sproži prenos podatkov iz FPGA vezja na gostitelja.

Bit 7: Write Interrupt Disable. S pisanjem vrednosti 1 v ta register so onemogočene prekinitve pri prenosu iz FPGA vezja na gostitelja.

Bit 8: Write Done. Ko je vrednost tega bita 1, je sistem uspešno končal prenos podatkov iz FPGA vezja na gostitelja.

Bit 15: Start Read. S pisanjem vrednosti 1 v ta bit se sproži prenos podatkov iz gostitelja na FPGA vezje.

Bit 22: Read Interrupt Disable. S pisanjem vrednosti 1 v ta register so onemogočene prekinitve pri prenosu iz gostitelja na FPGA vezje.

Bit 23: Read Done. Ko je vrednost tega bita 1, je sistem uspešno končal prenos podatkov iz gostitelja na FPGA vezje.

Register 2: Write Address. V ta register gostitelj vnese začetni naslov prejetih podatkov. Ta naslov predstavlja lokacijo prve besede, ki jo bo sistem prejel pri prenosom podatkov v načinu DMA. Naslov je 32 biten.

Register 3: Write Length. V ta register gostitelj vnese velikost paketa (v številu besed) pri prenosu iz FPGA vezja na gostitelja. V izdelanem sistemu so podprti le paketi dolžine 1 besede.

Register 4: Write packet count. V ta register gostitelj vpiše dolžino podatkov pri prenosu iz FPGA vezja na gostitelja. Dolžina je izražena v številu paketov, kjer je dolžina paketa določena v registru 3.

Register 5: Configuration Length. V ta register gostitelj vpiše dolžine konfiguracije, ki jo želi naložiti v rekonfigurabilni del. Ta register tudi določa, ali so podatki prejeti iz gostitelja konfiguracijski podatki. Če je vrednost registra večja od 0, potem platforma vhodne podatke obdeluje kot konfiguracijske podatke.

Register 6: Neuporabljen.

Register 7: Read Address. V ta register gostitelj vnese začetni naslov poslanih podatkov. Ta naslov predstavlja lokacijo prve besede, ki jo gostitelj želi prenesti na FPGA vezje.

Register 8: Read Length. V ta register gostitelj vnese velikost paketa (v številu besed) pri prenosu iz gostitelja na FPGA vezje. V izdelanem sistemu so podprti le paketi dolžine 1 besede.

Register 9: Read packet count. V ta register gostitelj vpiše dolžino podatkov pri prenosu iz gostitelja na FPGA vezje. Dolžina je izražena v številu paketov, kjer je dolžina paketa določena v registru 8.

Splošno namenski registri za rekonfigurabilni del. Registra številka 50 in 51 sta splošno namenska registra, ki ju uporablja rekonfigurabilni del sistema. Register 50 je vhodni nadzorni register. Vanj lahko uporabnik sistema vnese poljubne konfiguracijske podatke, ki jih potrebuje rekonfigurabilni del. Register 51 je izhodni statusni register. Vanj lahko rekonfigurabilni del zapiše poljubne statusne podatke, ki jih lahko uporabnik razbere z branjem tega registra.

Dodatek B

Avtomat za nadzor nad rekonfiguracijo

V tem dodatku se nahaja verilog koda avtomata za nadzor nad rekonfiguracijo preko vmesnika ICAP.

```
ICAP_VIRTEX5 #(   //ICAP vmesnik
    .ICAP_WIDTH("X32")
) ICAP_VIRTEX5_inst (
    .CE(CE),
    .CLK(trn_clk_c),
    .I(I),
    .O(O),
    .WRITE(ICAP_WRITE)
);
```

```
always@(negedge trn_clk_c)
begin
  // prožilec za začetek rekonfiguracije
  if(TRIGGER == 1'b1)
  begin
```

```
case (NEXT_STATE)

  //Začetno stanje
  STATE_00:
  begin
    TRIGGER<=1'b1;
    ICAP_WRITE <= 1'b1;
    CE <= 1'b1;
    I <= 32'h00000000;
    NEXT_STATE <= STATE_01;
    //Dolžina se določi v namenskem registru
    REMAINING_WRITE<=ICAP_length;
    address<=13'd0;
    fifo_read<=1'b0;
  end

  //ICAP_WRITE na 0
  STATE_01:
  begin
    TRIGGER<=1'b1;
    ICAP_WRITE <= 1'b0;
    CE <= 1'b1;
    I <= 32'h00000000;
    fifo_read<=1'b0;
    NEXT_STATE <= STATE_02;
  end

  STATE_02:    //CE na 0
  begin
    TRIGGER<=1'b1;
    ICAP_WRITE <= 1'b0;
    CE <= 1'b0;
```

```
    I <= 32'h00000000;
    fifo_read <=1'b0;
    NEXT_STATE <= STATE_03;
  end

  //Začetek pisanja podatkov
  STATE_03:
  begin
    TRIGGER<=1'b1;
    ICAP_WRITE <= 1'b0;
    if (REMAINING_WRITE > 32'h0)
    begin
      if (fifo_empty)
      begin
        CE<=1'b1;
        fifo_read <=1'b0;
        NEXT_STATE<=STATE_03;
      end else
      begin
        CE <= 1'b0;
        NEXT_STATE <= STATE_03;
        fifo_read <=1'b1;
        //I<=obrnjeni podatki iz FIFO
        I <= mem_data_out_swapped;
        REMAINING_WRITE<=REMAINING_WRITE-1;
        address<=address+1;
      end
    end else
    begin
      //Ko je konec v tem ciklu, zapiši ukaz
      //NO_OP (brez operacije)
      //V naslednjem ciklu nato v STATE_04 (Končni state)
```

```
      CE <= 1'b0;
      I<=32'h04000000;
      NEXT_STATE <= STATE_04;
      fifo_read <=1'b0;
    end
  end

  STATE_04:
  begin
    ICAP_WRITE <= 1'b0;
    CE <= 1'b0;
    I <= 32'h04000000;
    NEXT_STATE <= STATE_05;
    TRIGGER<=1'b1;
    fifo_read <=1'b0;
  end

  //Konec, CE na 1, TRIGGER nazaj na 0
  STATE_05:
  begin
    ICAP_WRITE <= 1'b0;
    CE <= 1'b1;
    I <= 32'h04000000;
    NEXT_STATE <= STATE_05;
    fifo_read <=1'b0;
    TRIGGER<=1'b0;
  end

  default:
  begin
    ICAP_WRITE <= 1'b1;
    CE <= 1'b1;
```

```
            I <= 32'hAAAAAAAA;
            NEXT_STATE <= STATE_00;
            fifo_read <=1'b0;
          end
        endcase


    end
    else
    begin
      //Če sprožilec za rekonfiguracijo ni sprožen
      //Potem samo čakaj
      address<=0;
      ICAP_WRITE <= 1'b1;
      CE <= 1'b1;
      //Ker ne pišemo, je I lahko kar koli
      I <= 32'hAAAABBBB;
      NEXT_STATE <= STATE_00;
      TRIGGER<=ICAP_reconfig_trigger;
      REMAINING_WRITE<=ICAP_length;
      address<=0;
      fifo_read <=1'b0;
    end
end
```

Literatura

[1] T. Nidhin, A. Bhattacharyya, R. Behera, T. Jayanthi, A review on seu mitigation techniques for fpga configuration memory, IETE Technical Review (2017) 1–12.

[2] C. Bolchini, A. Miele, M. D. Santambrogio, Tmr and partial dynamic reconfiguration to mitigate seu faults in fpgas, in: Defect and Fault-Tolerance in VLSI Systems, 2007. DFT'07. 22nd IEEE International Symposium on, IEEE, 2007, pp. 87–95.

[3] E. Stott, P. Sedcole, P. Y. Cheung, Fault tolerant methods for reliability in fpgas, in: Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on, IEEE, 2008, pp. 415–420.

[4] A. M. Jacobs, Reconfigurable fault tolerance for space systems, University of Florida, 2013.

[5] J. Burns, A. Donlin, J. Hogg, S. Singh, M. De Wit, A dynamic reconfiguration run-time system, in: Field-Programmable Custom Computing Machines, 1997. Proceedings., The 5th Annual IEEE Symposium on, IEEE, 1997, pp. 66–75.

[6] S. Singh, P. Bellec, Virtual hardware for graphics applications using fpgas, in: FPGAs for Custom Computing Machines, 1994. Proceedings. IEEE Workshop on, IEEE, 1994, pp. 49–58.

[7] V. Suse, D. Ionescu, A real-time reconfigurable architecture for face detection, in: ReConFigurable Computing and FPGAs (ReConFig), 2015 International Conference on, IEEE, 2015, pp. 1–6.

[8] P. Lysaght, J. Stockwood, J. Law, D. Girma, Artificial neural network implementation on a fine-grained fpga, in: International Workshop on Field Programmable Logic and Applications, Springer, 1994, pp. 421–431.

[9] J. Delorme, J. Martin, A. Nafkha, C. Moy, F. Clermidy, P. Leray, J. Palicot, A fpga partial reconfiguration design approach for cognitive radio based on noc architecture, in: Circuits and Systems and TAISA Conference, 2008. NEWCAS-TAISA 2008. 2008 Joint 6th International IEEE Northeast Workshop on, IEEE, 2008, pp. 355–358.

[10] D. Mesquita, F. Moraes, J. Palma, L. Moller, N. Calazans, Remote and partial reconfiguration of fpgas: tools and trends, in: Parallel and Distributed Processing Symposium, 2003. Proceedings. International, IEEE, 2003, pp. 8–pp.

[11] Z. Alassaad, M. Saghir, Cerpid: a reconfigurable platform interface driver for windows ce. net, in: Information and Communications Technology, 2005. Enabling Technologies for the New Knowledge Society: ITI 3rd International Conference on, IEEE, 2005, pp. 839–850.

[12] M. Al Kadi, P. Rudolph, D. Gohringer, M. Hubner, Dynamic and partial reconfiguration of zynq 7000 under linux, in: Reconfigurable Computing and FPGAs (ReConFig), 2013 International Conference on, IEEE, 2013, pp. 1–5.

[13] E. L. Horta, J. W. Lockwood, Parbit: a tool to transform bitfiles to implement partial reconfiguration of field programmable gate arrays (fpgas).

[14] K. Vipin, S. A. Fahmy, A high speed open source controller for fpga partial reconfiguration, in: Field-Programmable Technology (FPT), 2012 International Conference on, IEEE, 2012, pp. 61–66.

[15] G. Kojek, Vmesnik pci express na fpga napravi z uporabo delne rekonfiguracije vezja: diplomsko delo visokošolskega strokovnega študija, univerza v Ljubljani, Fakulteta za Elektrotehniko (2015).

[16] J. G. Eldredge, B. L. Hutchings, Density enhancement of a neural network using fpgas and run-time reconfiguration, in: FPGAs for Custom Computing Machines, 1994. Proceedings. IEEE Workshop on, IEEE, 1994, pp. 180–188.

[17] D. Koch, Partial Reconfiguration on FPGAs: Architectures, Tools and Applications, Vol. 153, Springer Science & Business Media, 2012.

[18] A. Sadek, H. Mostafa, A. Nassar, Y. Ismail, Towards the implementation of multi-band multi-standard software-defined radio using dynamic partial reconfiguration, International Journal of Communication Systems.

[19] PCI-SIG, Pci express base specification revision 1.1, `https://pcisig.com/specifications/pciexpress/`, citirano: 2017-07-27 (2005).

[20] J. Meyer, J. Noguera, M. Hübner, L. Braun, O. Sander, R. M. Gil, R. Stewart, J. Becker, Fast start-up for spartan-6 fpgas using dynamic partial reconfiguration, in: Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011, IEEE, 2011, pp. 1–6.

[21] M. Hübner, J. Meyer, O. Sander, L. Braun, J. Becker, J. Noguera, R. Stewart, Fast sequential fpga startup based on partial and dynamic reconfiguration, in: VLSI (ISVLSI), 2010 IEEE Computer Society Annual Symposium on, IEEE, 2010, pp. 190–194.

[22] U. Legat, A. Biasizzo, F. Novak, Seu recovery mechanism for sram-based fpgas, IEEE Transactions on Nuclear Science 59 (5) (2012) 2562–2571.

[23] P. Garcia, K. Compton, M. Schulte, E. Blem, W. Fu, An overview of reconfigurable hardware in embedded systems, EURASIP Journal on Embedded Systems 2006 (1) (2006) 13–13.

[24] D. Mayhew, V. Krishnan, Pci express and advanced switching: evolutionary path to building next generation interconnects, in: High Performance Interconnects, 2003. Proceedings. 11th Symposium on, IEEE, 2003, pp. 21–29.

[25] P. Sig, Pci local bus specification revision 2.2, PCI SIG 12.

[26] V. Kumar, Standard registers of pci type 0 (non-bridge) configuration space header, https://en.wikipedia.org/wiki/PCI_configuration_space#/media/File:Pci-config-space.svg.

[27] E. Billauer, Down to the TLP: How PCI express devices talk (Part 1), http://xillybus.com/tutorials/pci-express-tlp-pcie-primer-tutorial-guide-1, citirano: 2017-07-27.

[28] J. Coleman, Reducing interrupt latency through the use of message signaled interrupts, Intel, Tech. Rep.

[29] A. Goldhammer, J. Ayer Jr, Understanding performance of pci express systems, Xilinx WP350, Sept 4.

[30] Xilinx, XUPV5-LX110t, https://www.xilinx.com/univ/xupv5-lx110t.htm, citirano: 2017-07-27.

[31] Xilinx, LogiCORE IP Endpoint Block Plus v1.15 for PCI Express (UG341), https://www.xilinx.com/support/documentation/ip_documentation/pcie_blk_plus/v1_15/pcie_blk_plus_ug341.pdf, citirano: 2017-07-27.

[32] J. Wiltgen, J. Ayer, Bus master dma performance demonstrationreference design for the xilinx endpoint pci express solutions (XAPP1052), http://www.xilinx.com/support/documentation/application_notes/xapp1052.pdf, citirano: 2017-07-27.

[33] Xilinx, LogiCORE IP FIFO Generator v8.1 (UG175), https://www.xilinx.com/support/documentation/ip_documentation/fifo_generator_ug175.pdf, citirano: 2017-07-27.

[34] Xilinx, PlanAhead User Guide (UG-632), https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_4/PlanAhead_UserGuide.pdf, citirano: 2017-07-27.

[35] Xilinx, Virtex-5 FPGA Configuration User Guide (UG191), https://www.xilinx.com/support/documentation/user_guides/ug191.pdf, citirano: 2017-07-27.

[36] Xilinx, Virtex-5 Family Overview (DS-100), https://www.xilinx.com/support/documentation/data_sheets/ds100.pdf, citirano: 2017-07-27.

[37] Xilinx, Partial Reconfiguraction Controller v1.1, https://www.xilinx.com/support/documentation/ip_documentation/prc/v1_1/pg193-partial-reconfiguration-controller.pdf, citirano: 2017-07-27.

[38] K. Vipin, S. A. Fahmy, Dyract: A partial reconfiguration enabled accelerator and test platform, in: Field Programmable Logic and Applications (FPL), 2014 24th International Conference on, IEEE, 2014, pp. 1–7.

[39] N. F. Pub, 197: Advanced encryption standard (aes), Federal information processing standards publication 197 (441) (2001) 0311.

[40] P. Chodowiec, K. Gaj, Very compact fpga implementation of the aes algorithm, in: Ches, Vol. 2779, Springer, 2003, pp. 319–333.

[41] U. Legat, A. Biasizzo, F. Novak, A compact aes core with on-line error-detection for fpga applications with modest hardware resources, Microprocessors and Microsystems 35 (4) (2011) 405–416.

[42] M. Jacobsen, D. Richmond, M. Hogains, R. Kastner, Riffa 2.1: A reusable integration framework for fpga accelerators, ACM Transactions on Reconfigurable Technology and Systems (TRETS) 8 (4) (2015) 22.

[43] Xilinx, 7 series FPGAs Data Sheet: Overview (DS-180), `https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf`, citirano: 2017-07-27.